Adversarial Search
&
Logic and Reasoning

Recall from Last Time: Adversarial Games as Search

Convention: first player is called MAX,

2nd player is called MIN

MAX moves first and they take turns until game is over

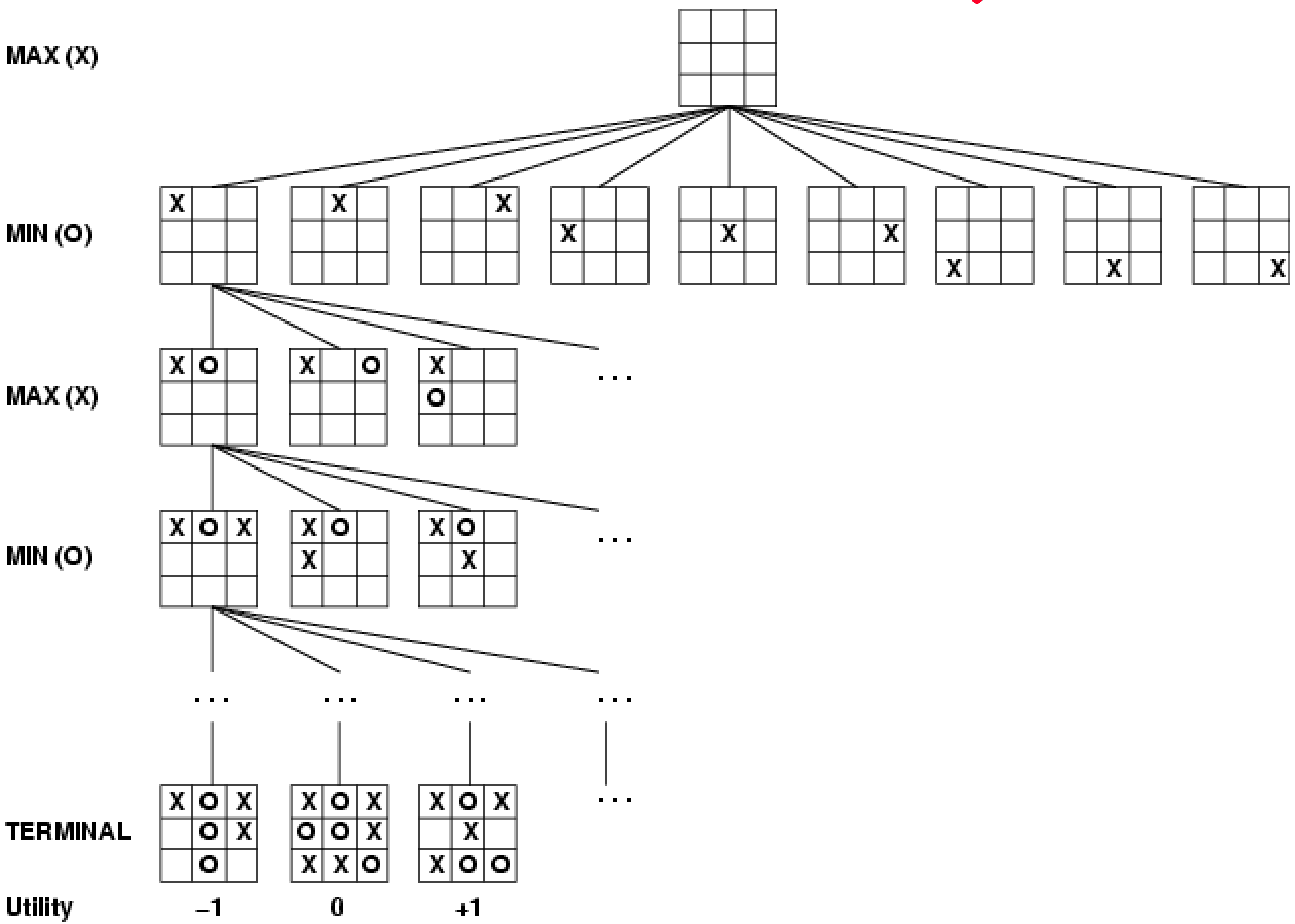
Winner gets reward, loser gets penalty

Utility values stated from MAX's perspective

Initial state and legal moves define the *game tree*

MAX uses game tree to determine next move

Tic-Tac-Toe Example



Optimal Strategy: Minimax Search

Find the contingent *strategy* for MAX assuming an infallible MIN opponent

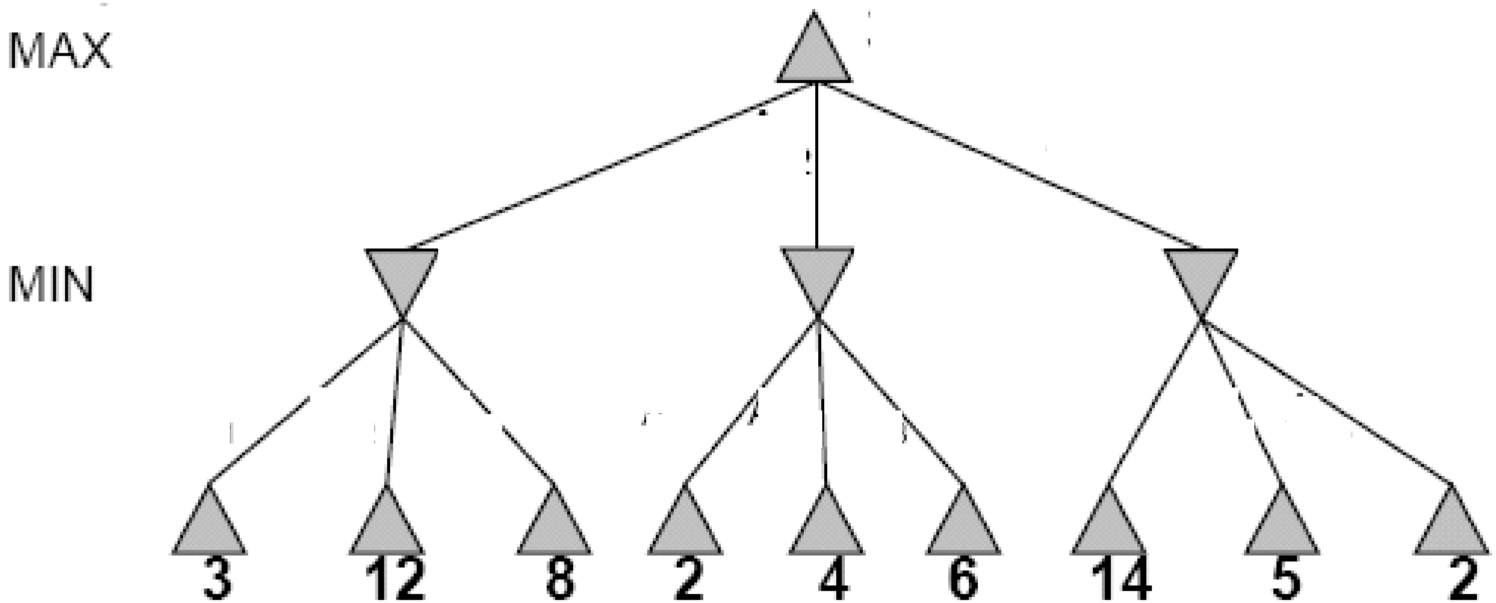
Assumption: Both players play optimally!

Given a game tree, the optimal strategy can be determined by using the *minimax* value of each node (defined recursively):

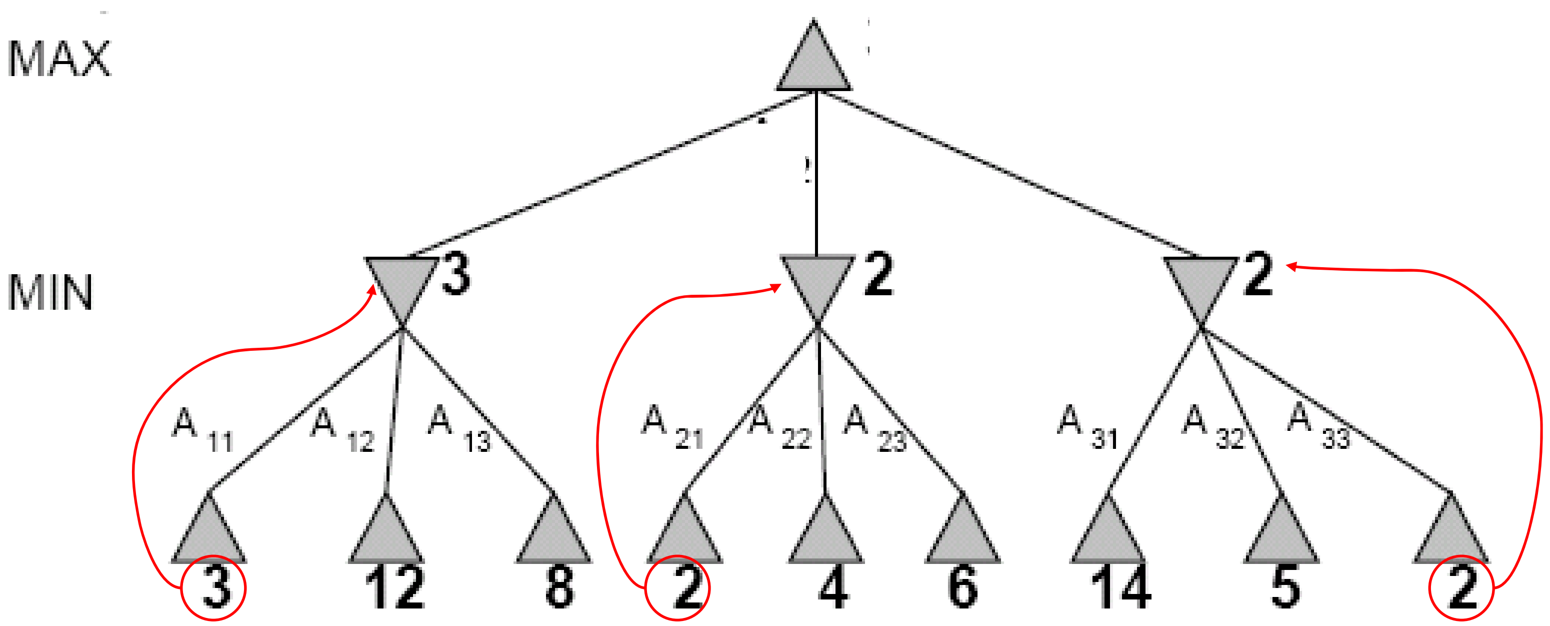
MINIMAX-VALUE(n)=

UTILITY(n)	If n is a terminal
$\max_{s \in succ(n)}$ MINIMAX-VALUE(s)	If n is a MAX node
$\min_{s \in succ(n)}$ MINIMAX-VALUE(s)	If n is a MIN node

Two-Ply Game Tree

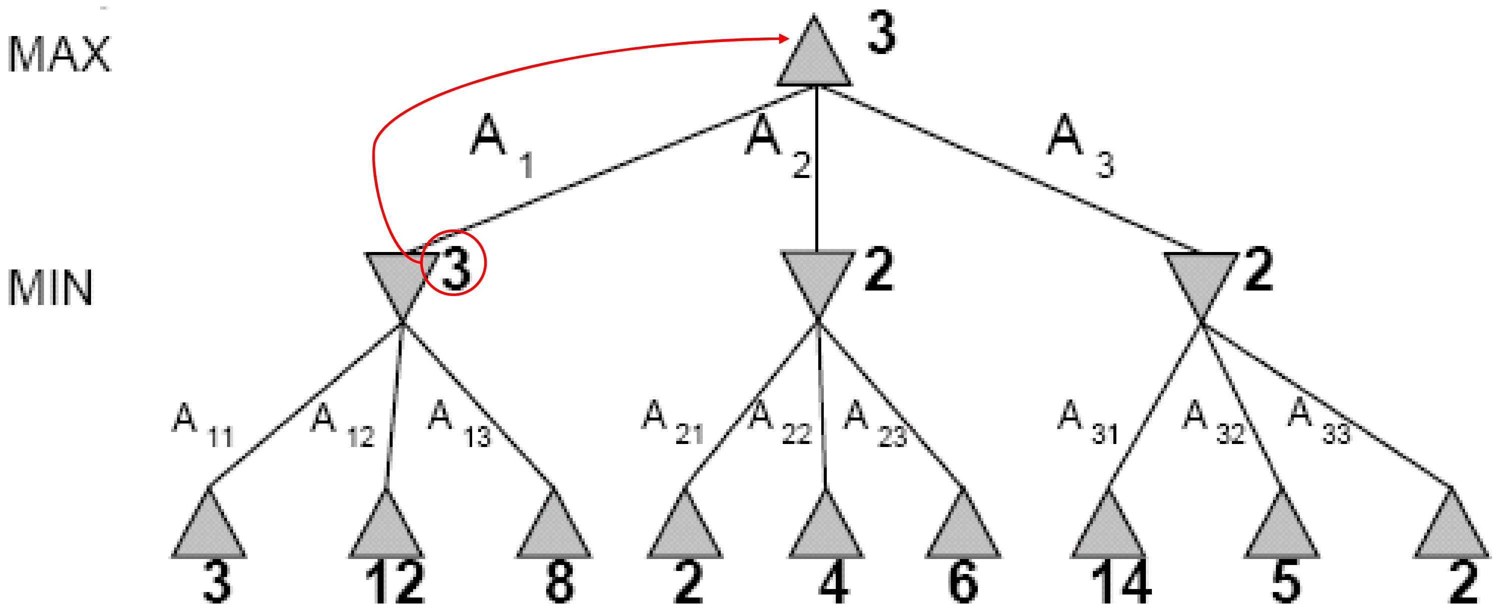


Two-Ply Game Tree



Two-Ply Game Tree

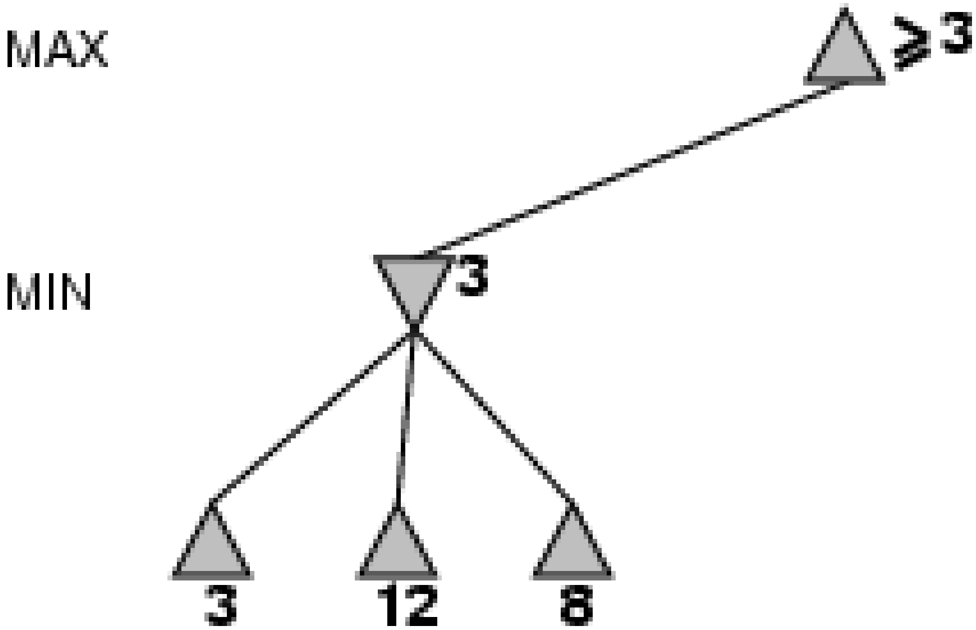
Minimax decision = A_1



Minimax maximizes the worst-case outcome for max

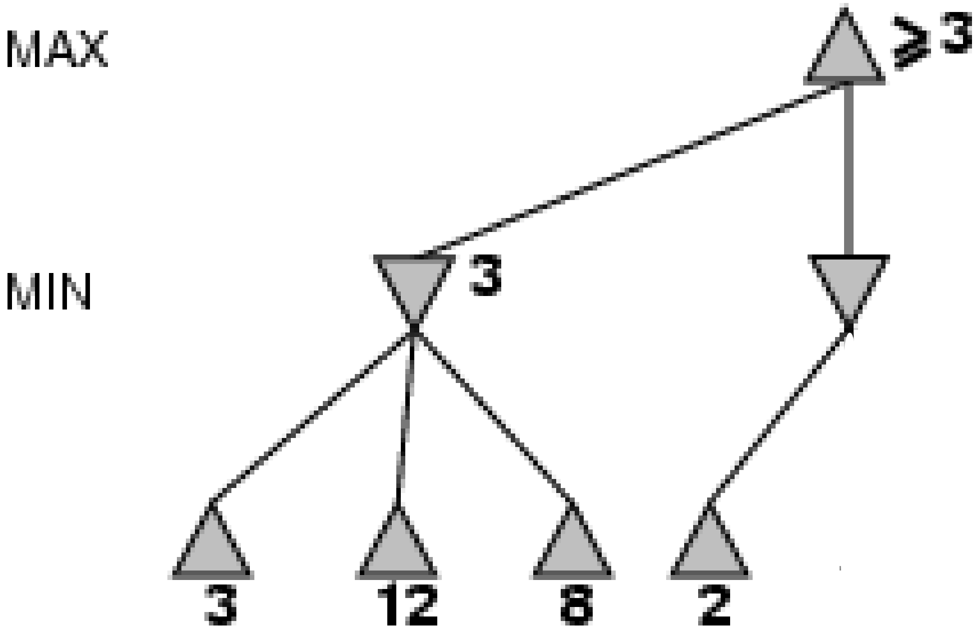
Is there anyway I could speed up this search?

Pruning trees

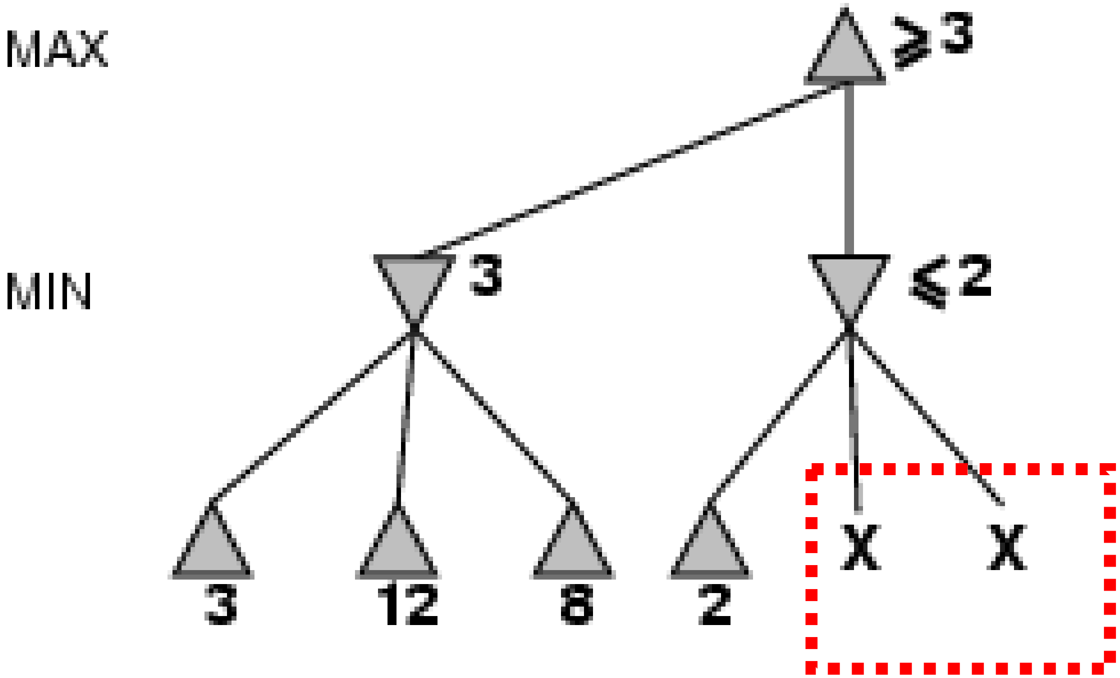


Minimax algorithm explores depth-first

Pruning trees

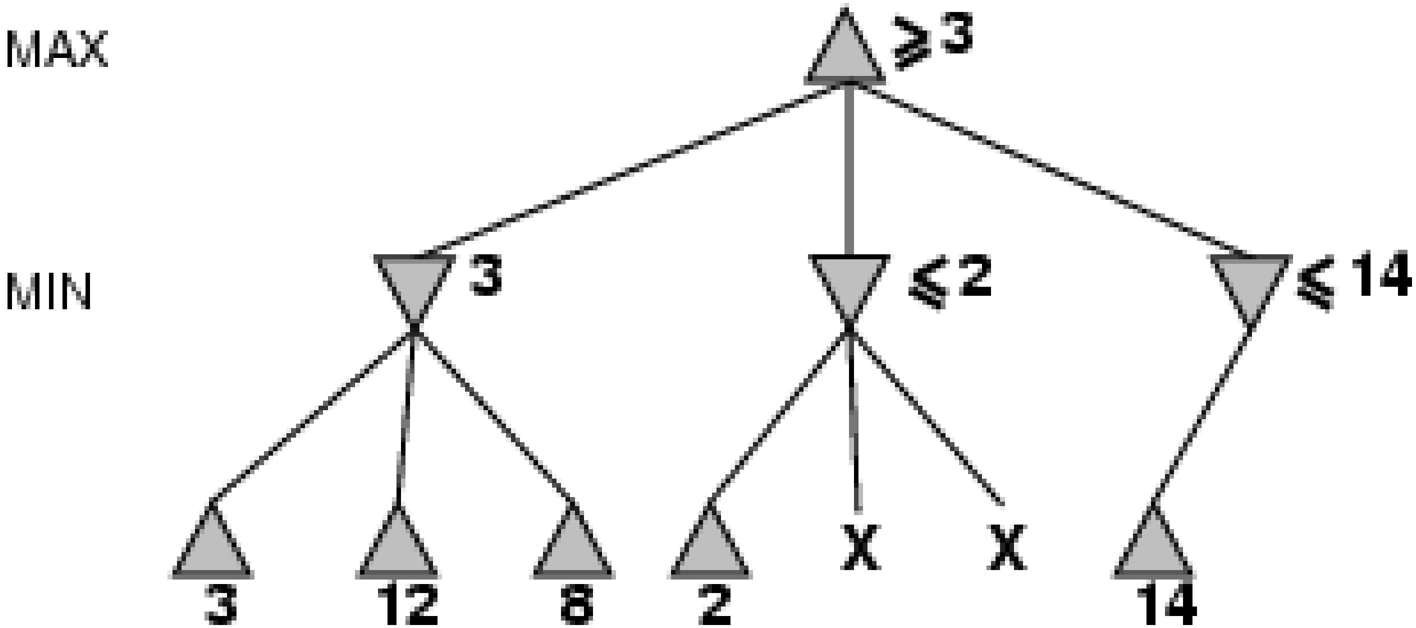


Pruning trees

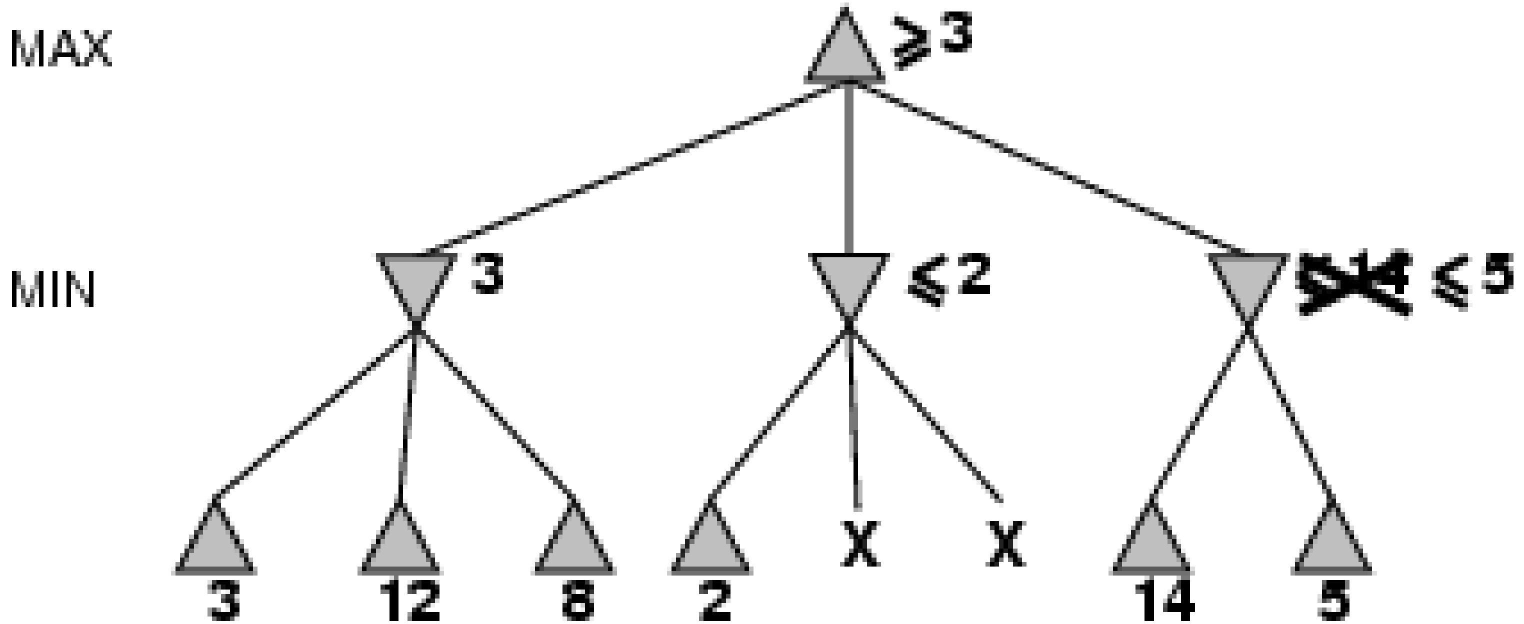


No need to look at or expand these nodes!!

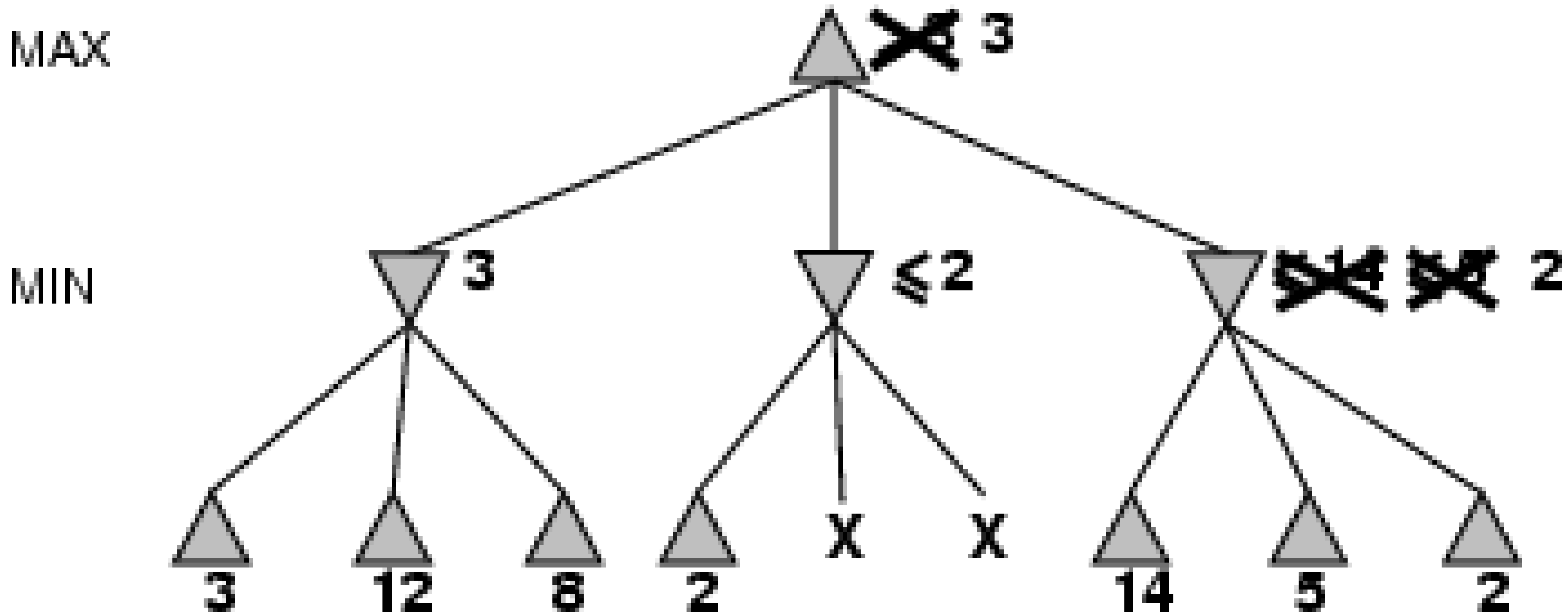
Pruning trees



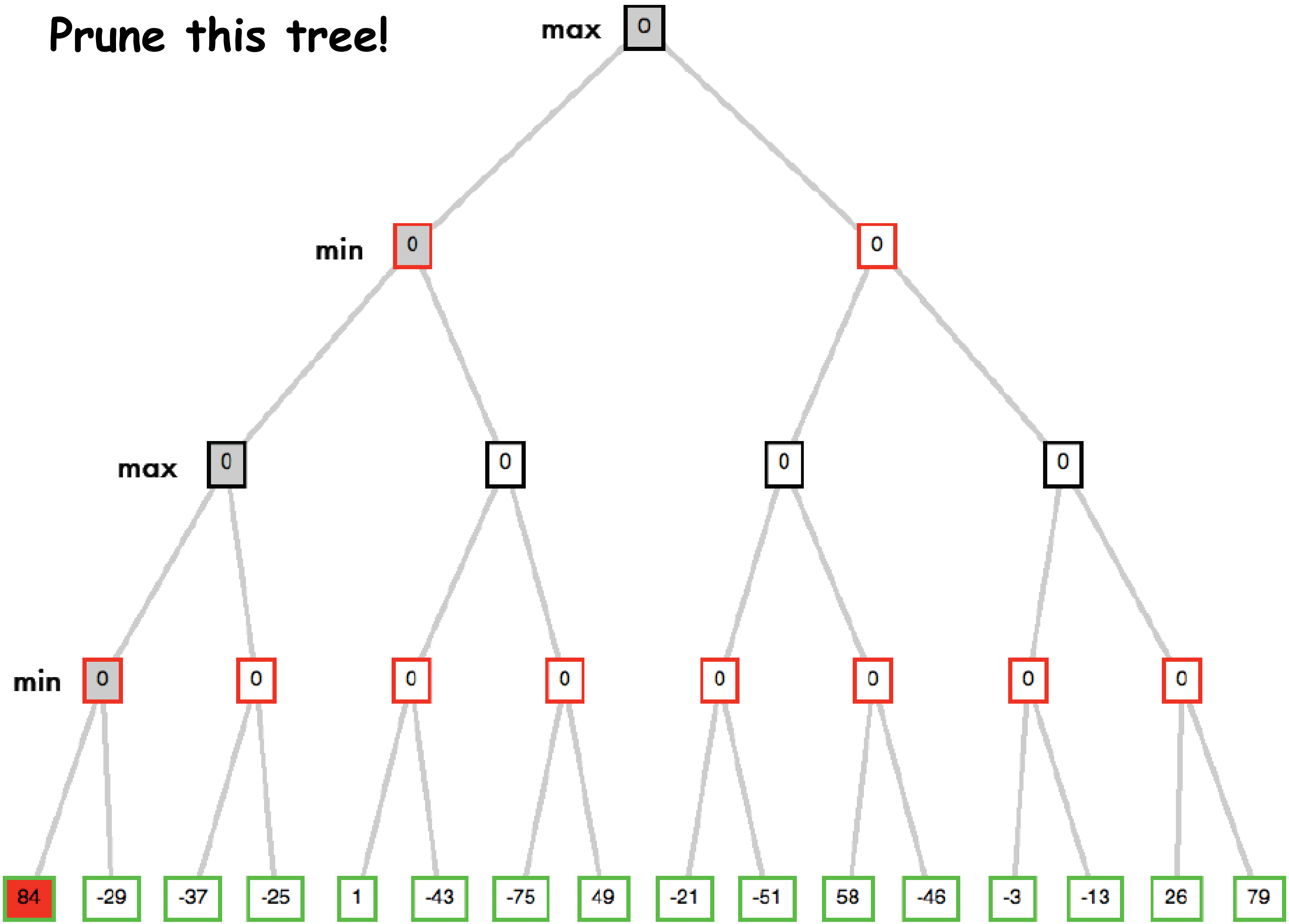
Pruning trees

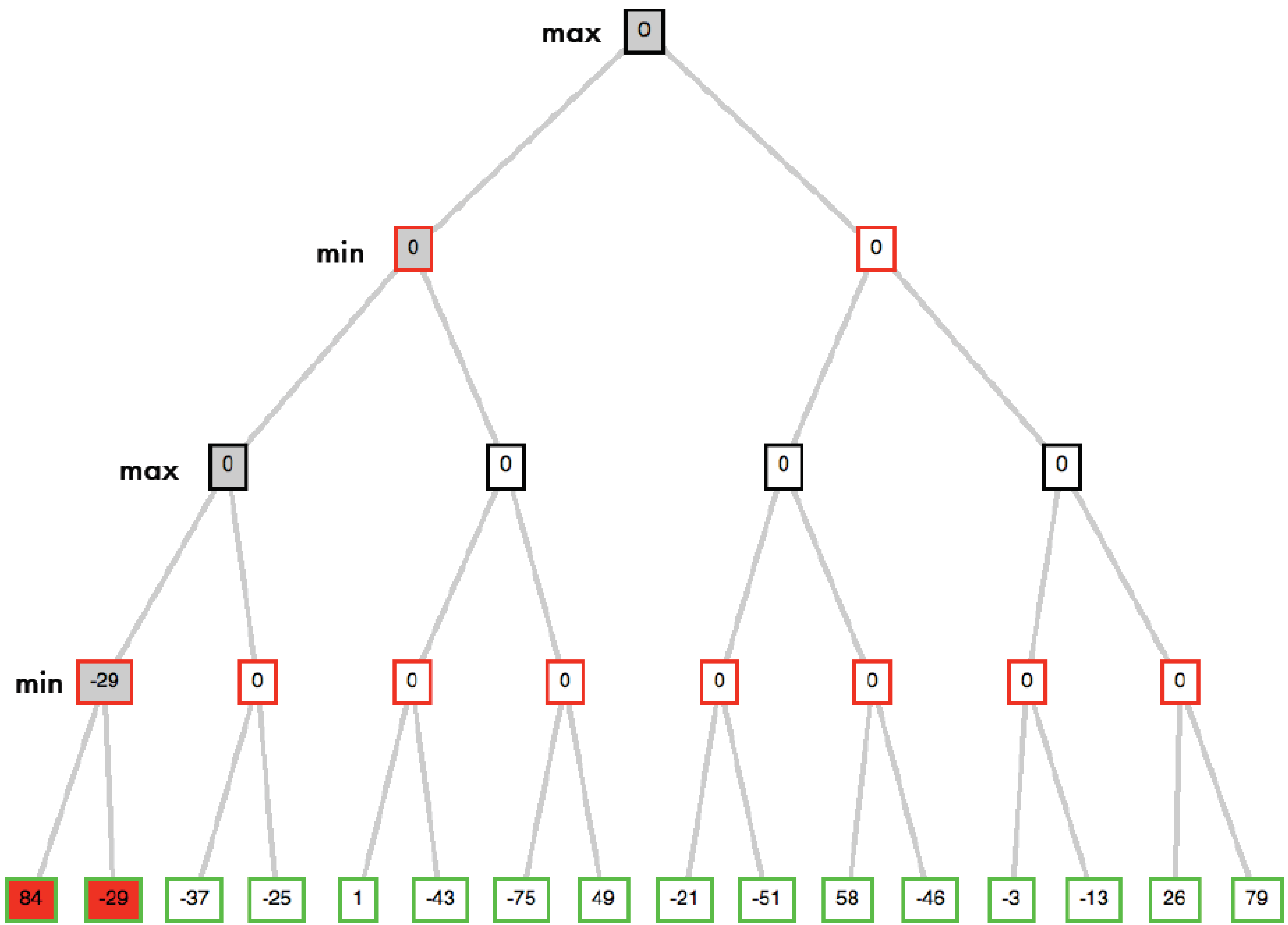


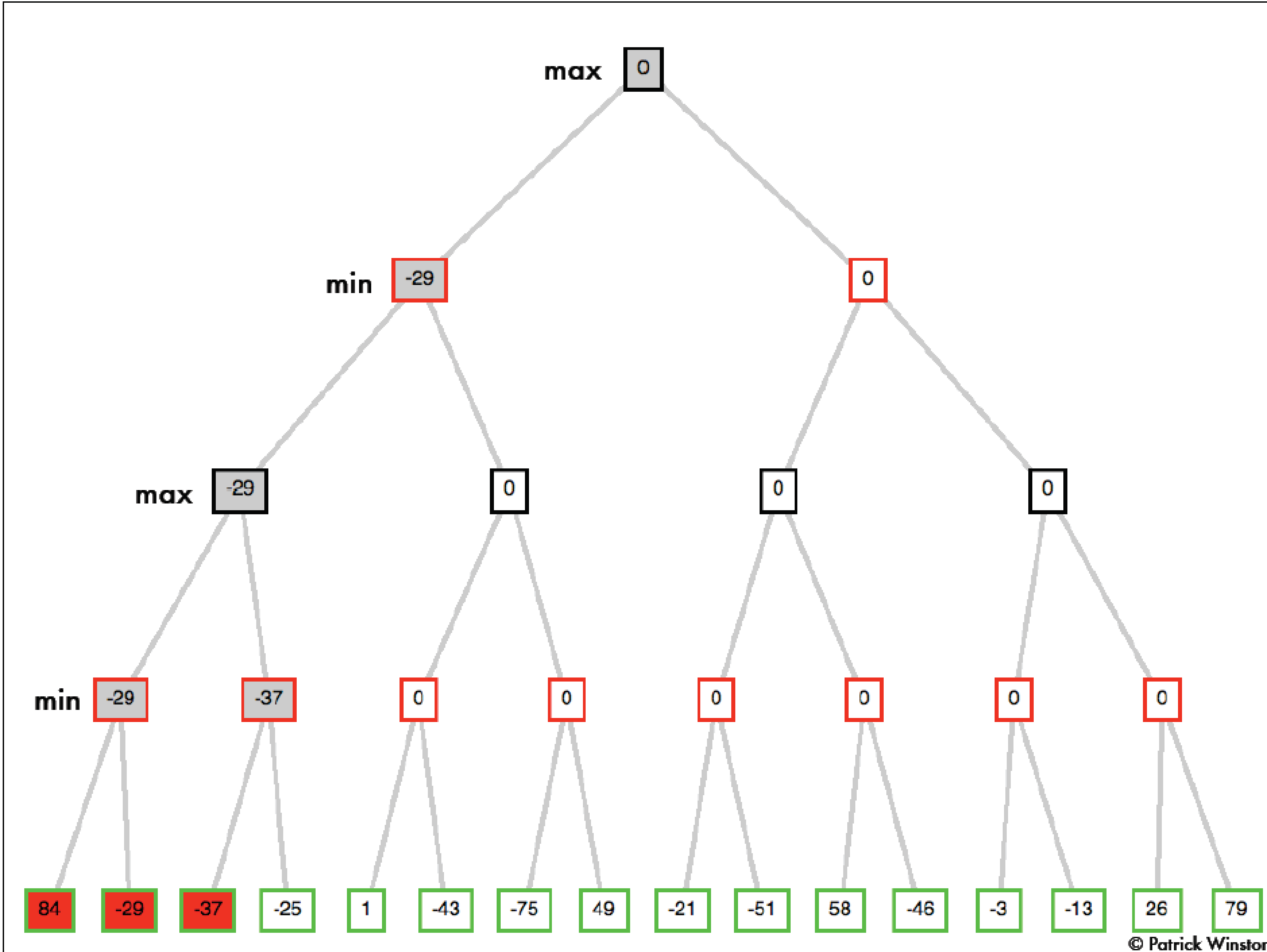
Pruning trees



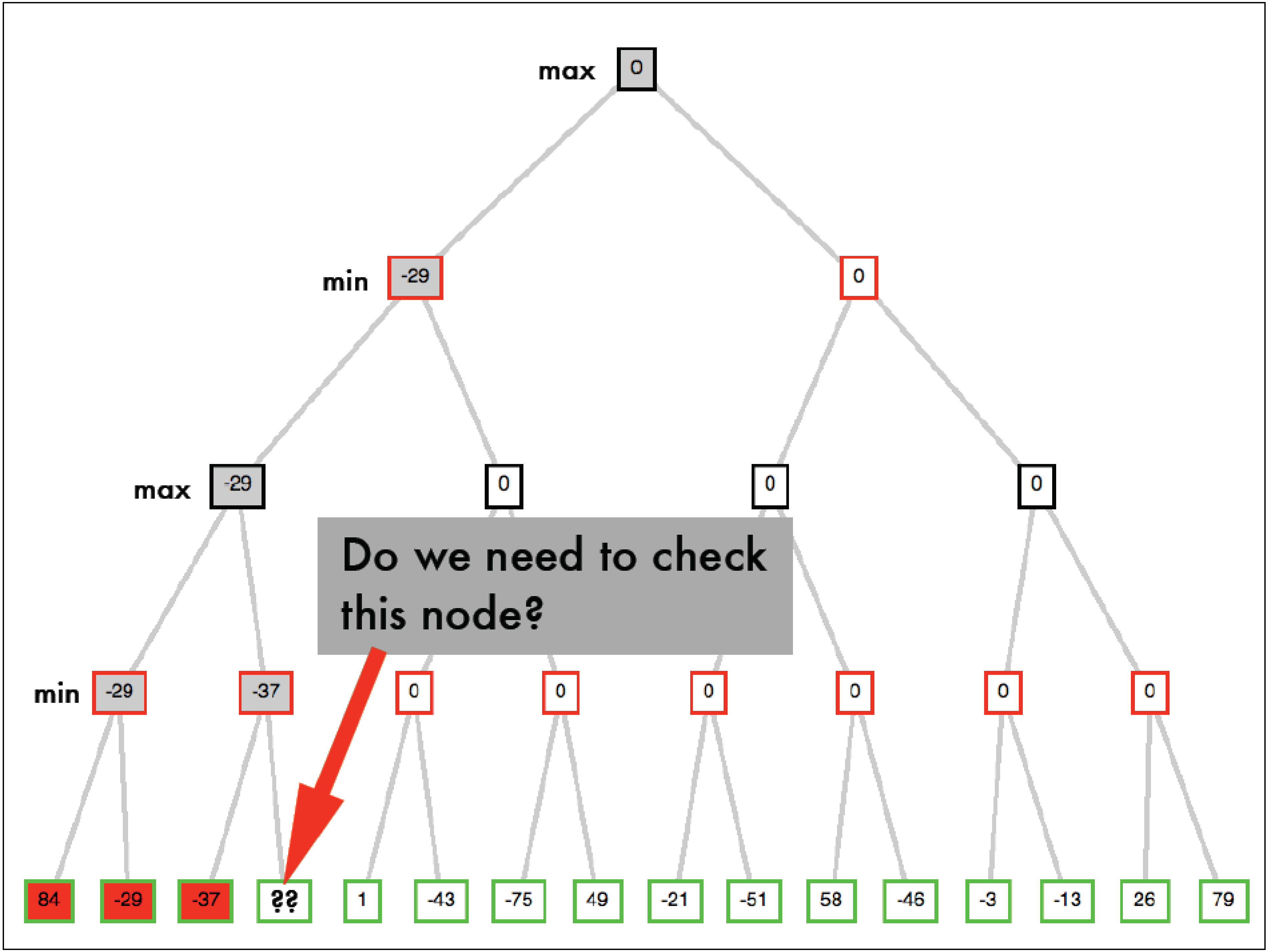
Prune this tree!

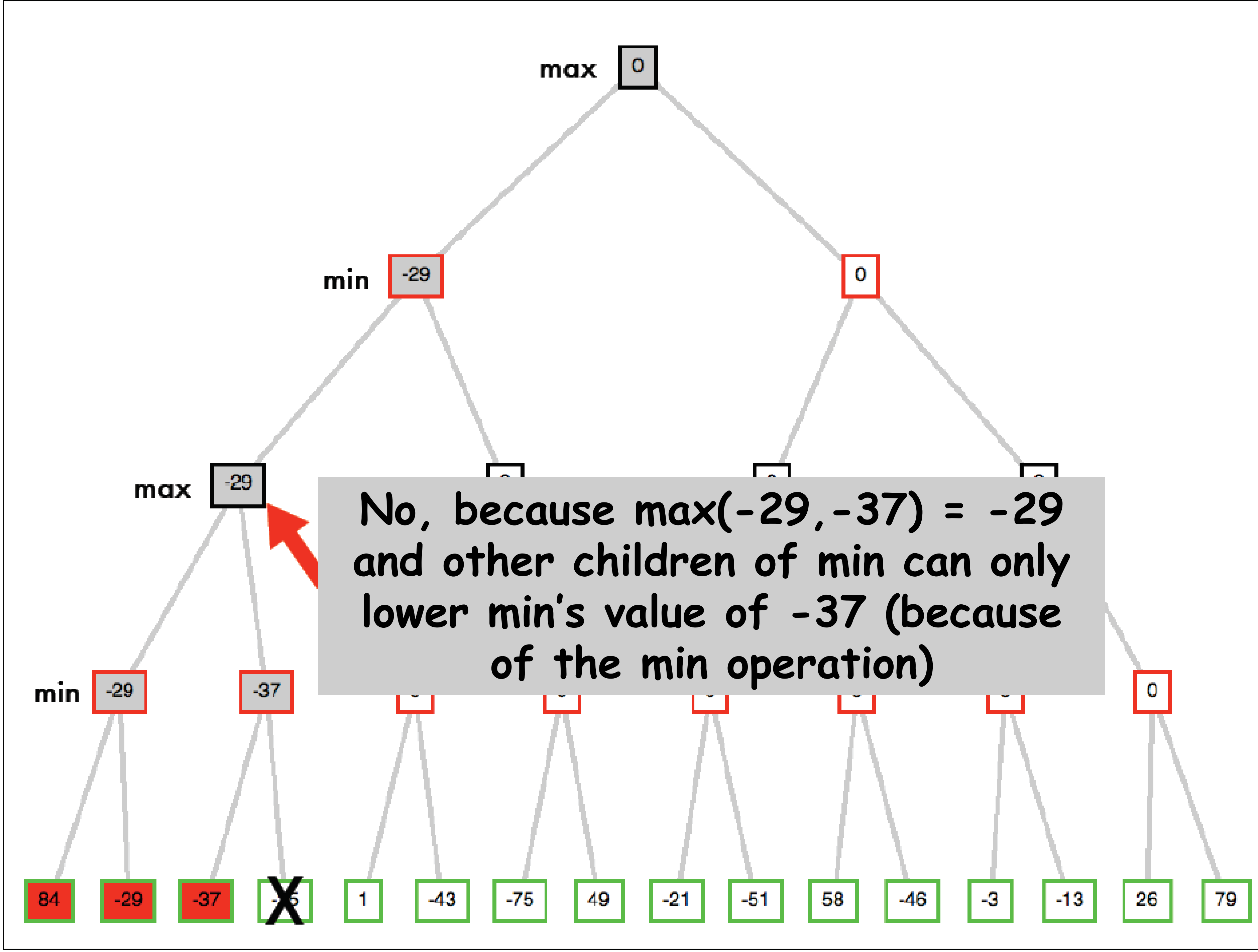



max
0
min
0
0
max
0
0
0
0
min
-29
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

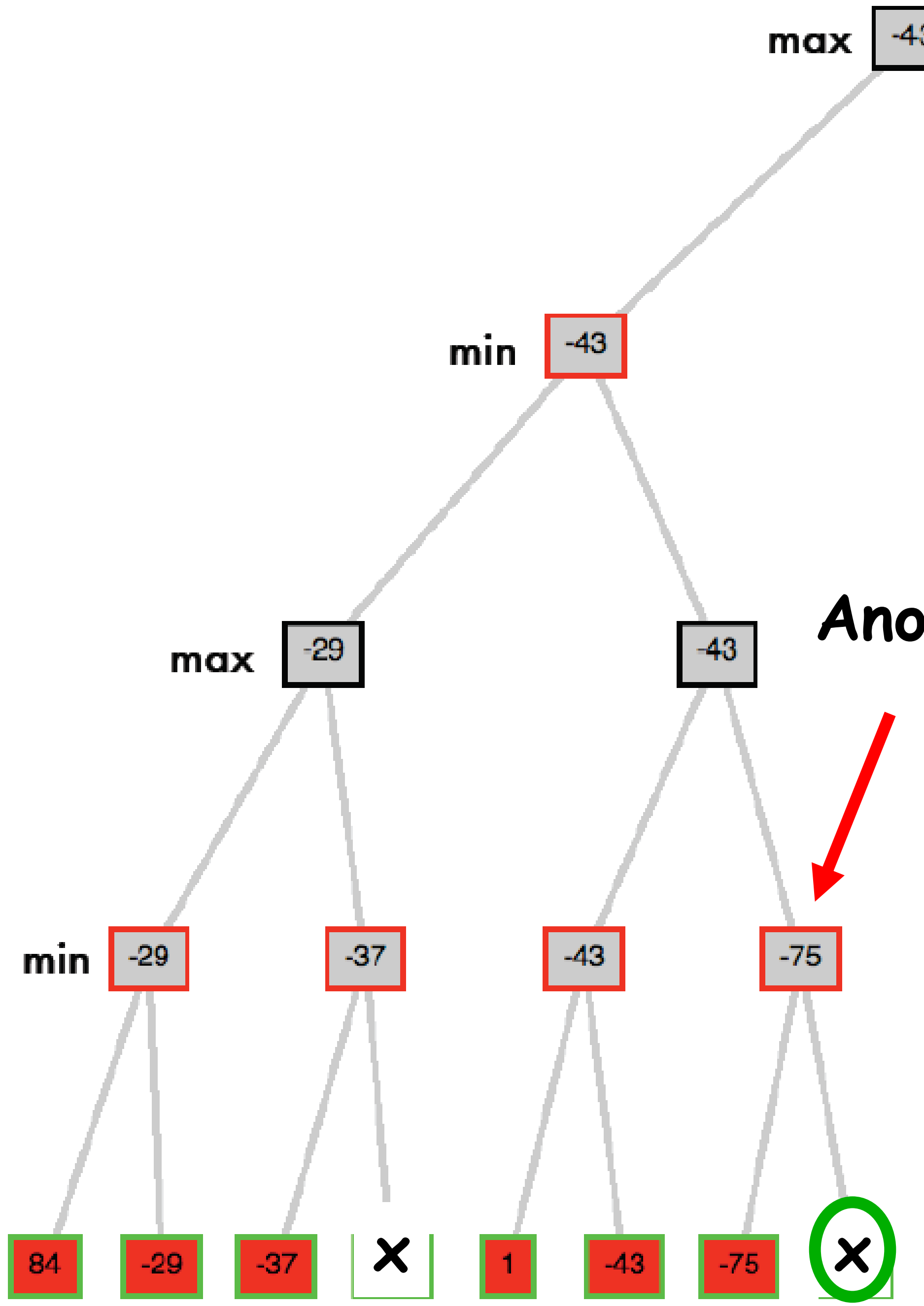


max
0
min
-29
0
max
-29
0
0
0
min
-29
-37
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79
© Patrick Winston

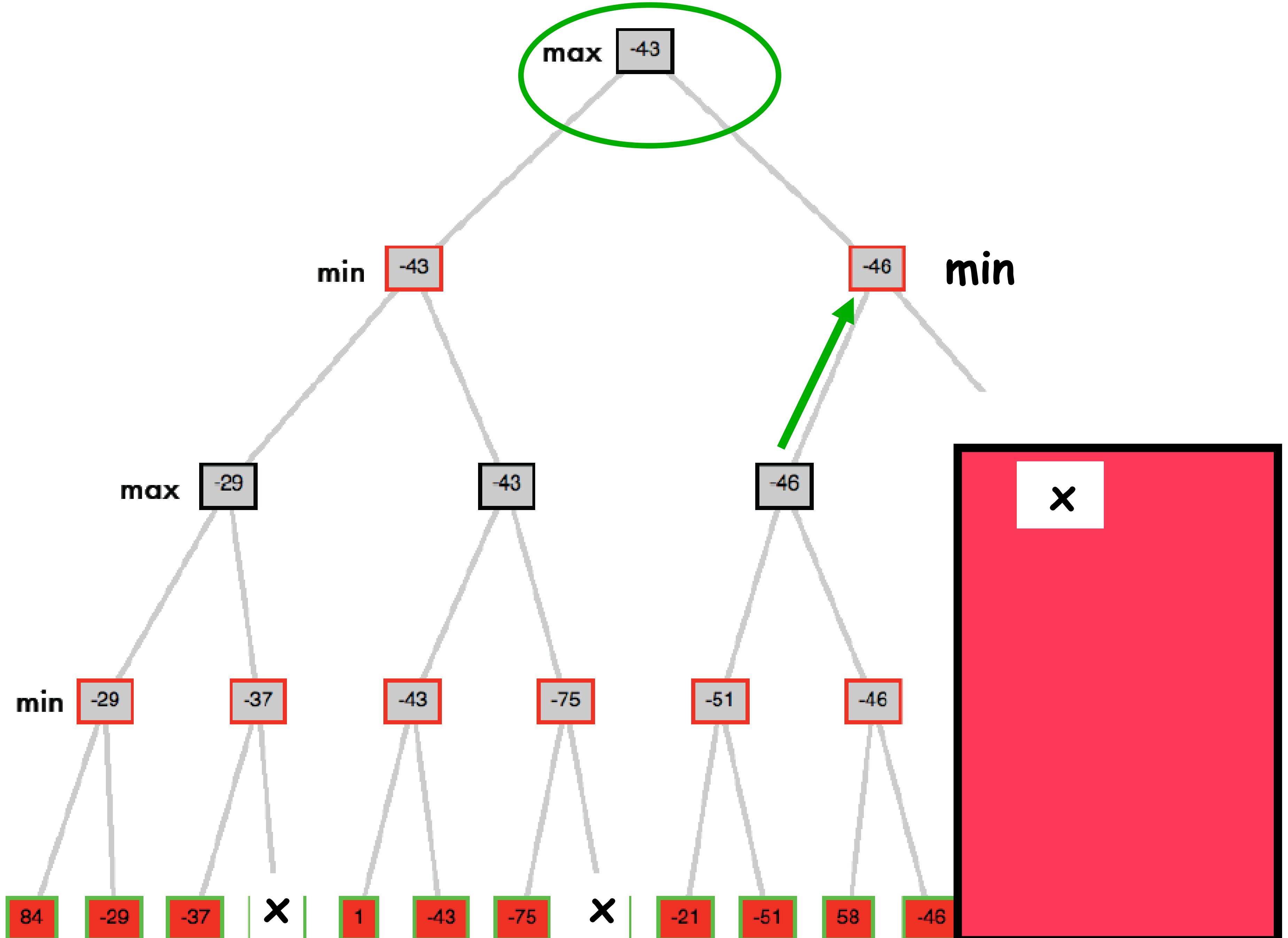

max
0
min
-29
0
max
-29
0
0
0
Do we need to check this node?
min
-29
-37
0
0
0
0
0
0
84
-29
-37
??
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79


max
0
min
-29
0
max
-29
min
-29
-37
0
84
-29
-37
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79
No, because max(-29,-37) = -29 and other children of min can only lower min's value of -37 (because of the min operation)

Another pruning opportunity!



Pruning can eliminate entire subtrees!



max
-43
min
-43
-46
min
max
-29
-43
-46
x
min
-29
-37
-43
-75
-51
-46
84
-29
-37
x
1
-43
-75
x
-21
-51
58
-46

This form of tree pruning is known as alpha-beta pruning

alpha = the highest (best) value for MAX along path
beta = the lowest (best) value for MIN along path

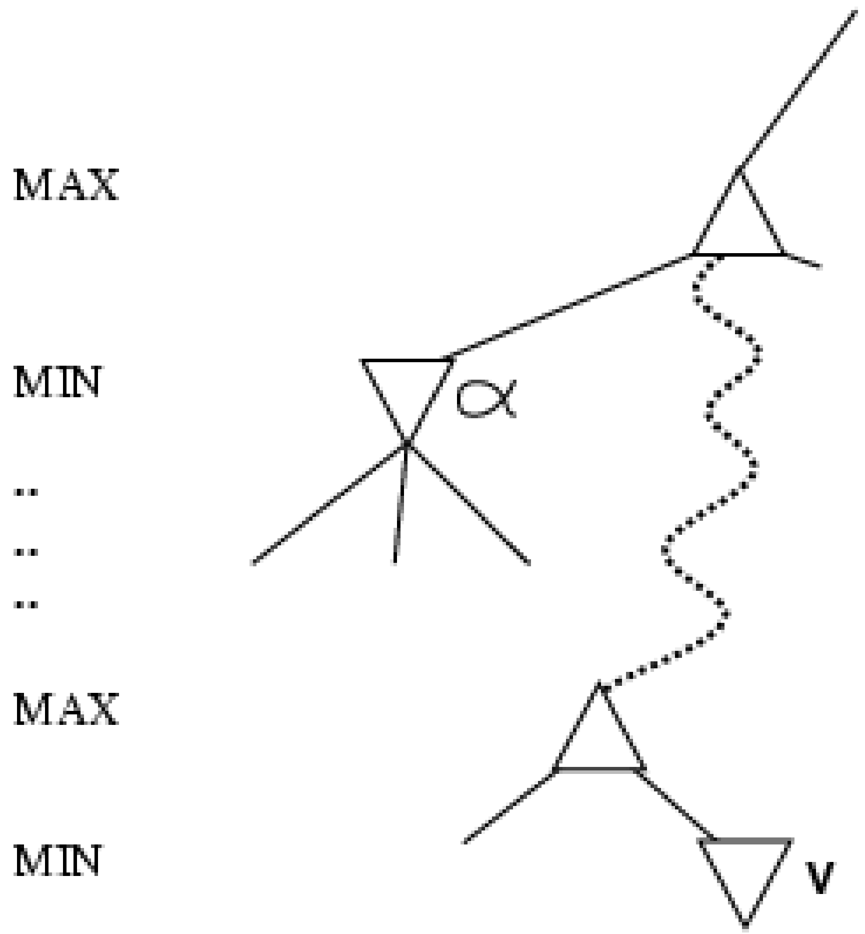
Why is it called α-β?

α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*

If *v* is worse than α, *max* will avoid it

→prune that branch

Define β similarly for *min*



The α-β algorithm

(minimax with four lines of added code)

function ALPHA-BETA-SEARCH($state$) **returns** *an action*
  inputs: $state$, current state in game

  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in SUCCESSORS($state$) with value v

---

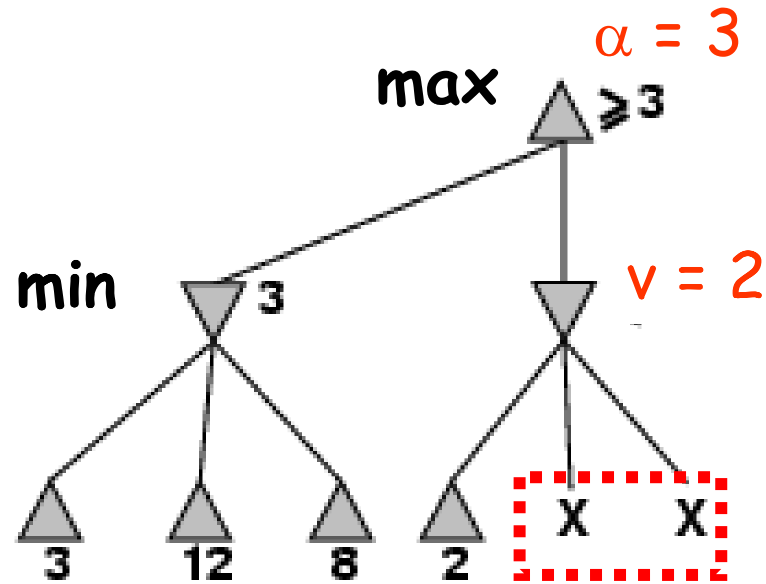
function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  inputs: $state$, current state in game
    α, the value of the best alternative for MAX along the path to $state$
    β, the value of the best alternative for MIN along the path to $state$

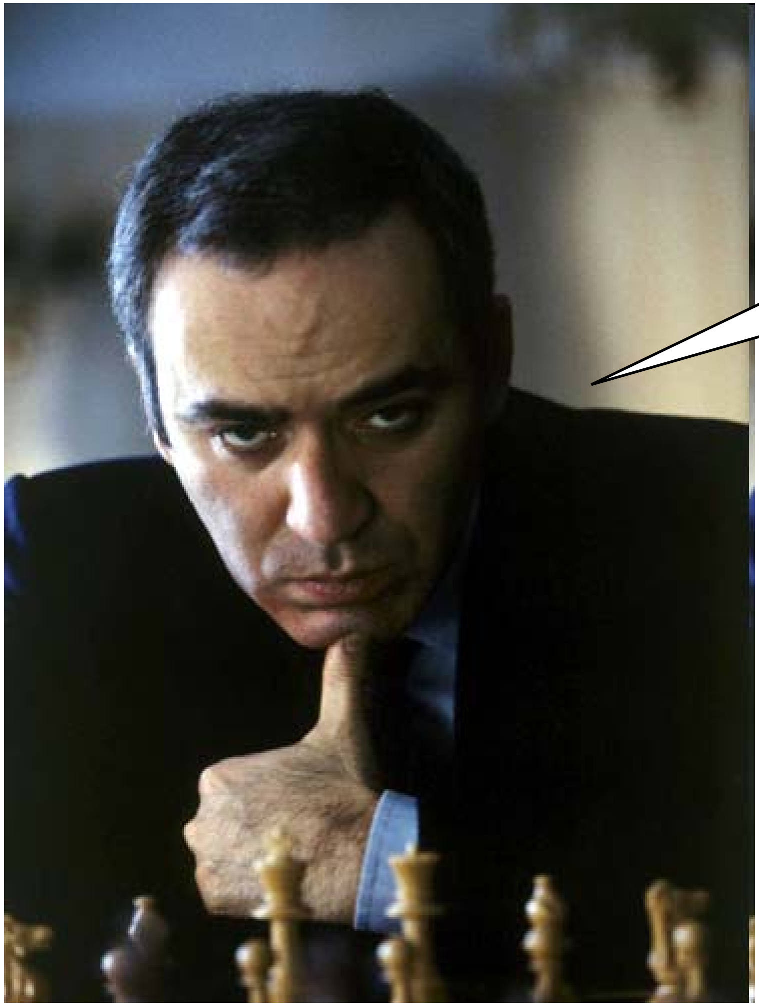
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
New   **if** $v \geq \beta$ **then return** v → Pruning
    $\alpha \leftarrow$ MAX(α, v)
  return v

The α-β algorithm (cont.)

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(s, α, β))
    if $v \leq \alpha$ **then return** v → Pruning
    $\beta \leftarrow$ MIN(β, v)
  return v





Does alpha-beta pruning change the final result? Is it an approximation?

Properties of α-β

Pruning does not affect final result

Effectiveness of pruning can be improved through good move ordering

(e.g., in chess, captures > threats > forward moves > backward moves)

With "perfect ordering," time complexity = $O(b^{m/2})$

→allows us to search deeper - doubles depth of search

A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

Good enough?

Chess:

- branching factor $b \approx 35$
- game length $m \approx 100$
- α-β search space $b^{m/2} \approx 35^{50} \approx 10^{77}$

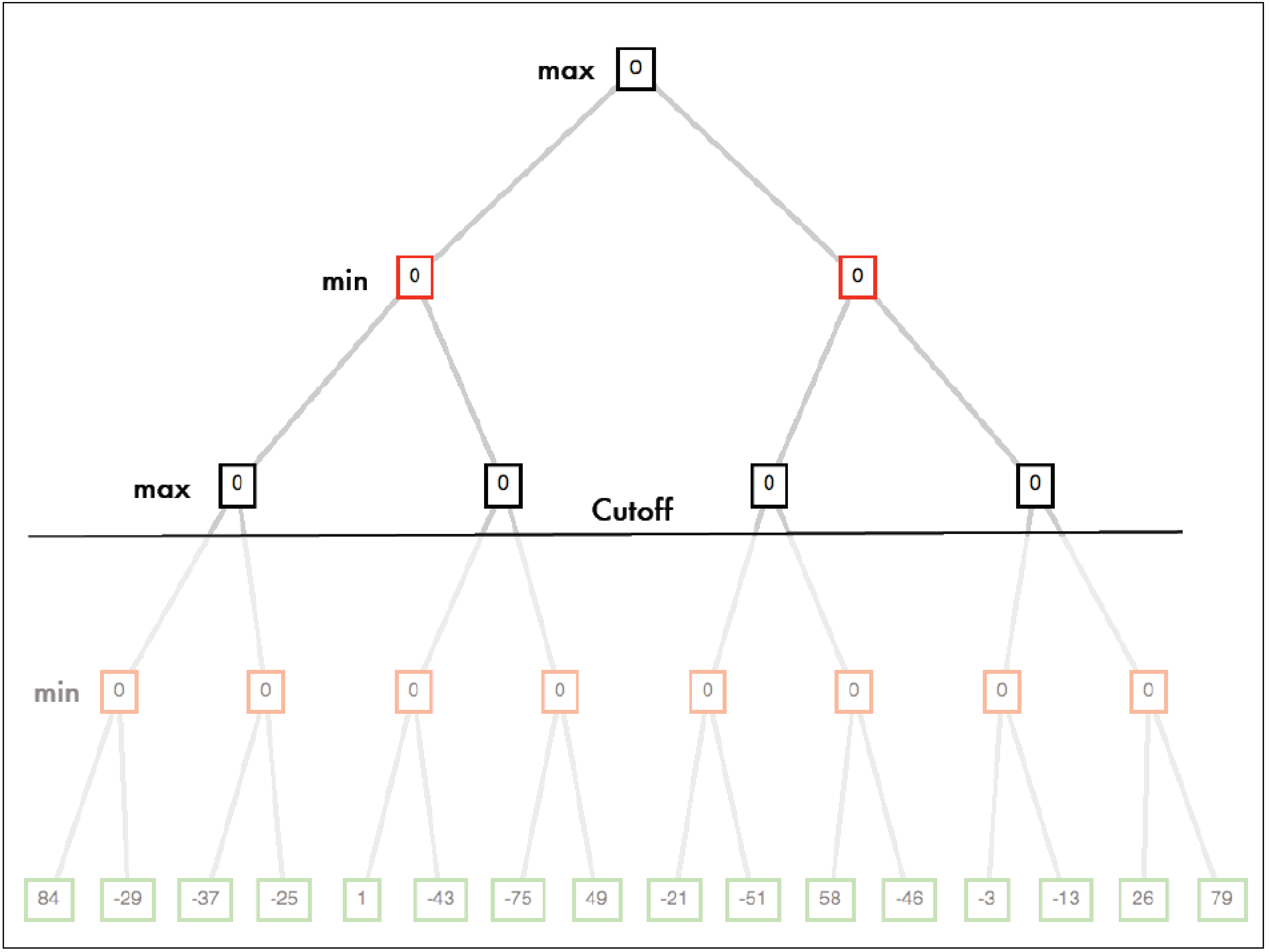
The Universe:

- number of atoms $\approx 10^{78}$
- age $\approx 10^{21}$ milliseconds

Can we do better?

Strategies:

- **search to a fixed depth (cut off search)**
- **iterative deepening search**


max
0
min
0
0
max
0
0
0
0
Cutoff
min
0
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

Evaluation Function

- When search space is too large, create game tree up to a certain depth only.
- Art is to estimate utilities of positions that are not terminal states.
- Example of simple evaluation criteria in chess:
  - Material worth: pawn=1, knight =3, rook=5, queen=9.
  - Other: king safety, good pawn structure
  - Rule of thumb: 3-point advantage = certain victory

eval(s) =

w1 * material(s) +
w2 * mobility(s) +
w3 * king safety(s) +
w4 * center control(s) + ...

Cutting off search

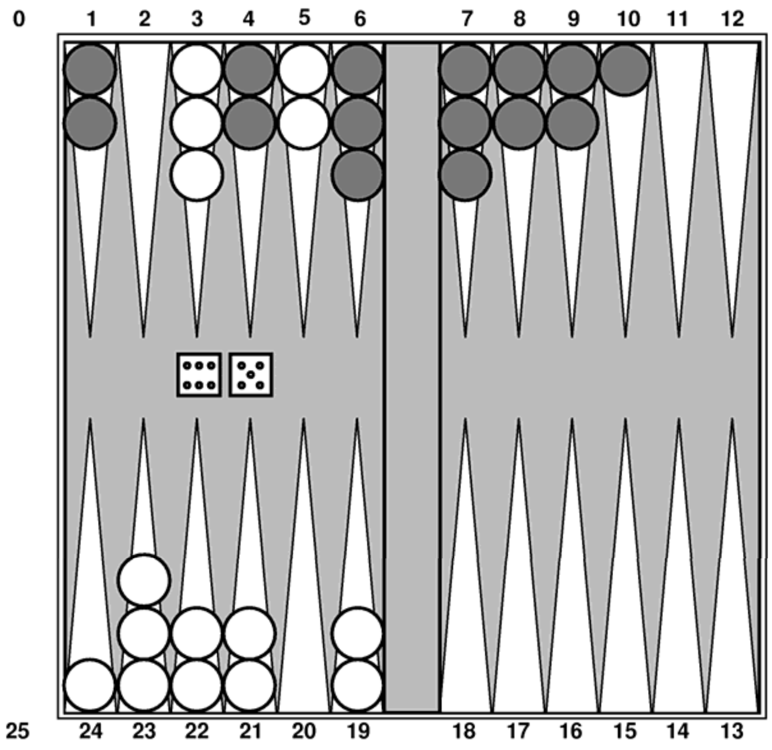
Does it work in practice?

If $b^m = 10^6$ and $b=35 \Rightarrow m=4$

4-ply lookahead is a hopeless chess player!

- 4-ply ≈ human novice
- 8-ply ≈ typical PC, human master
- 14-ply ≈ Deep Blue, Kasparov
- 18-ply ≈ Hydra (64-node cluster with FPGAs)

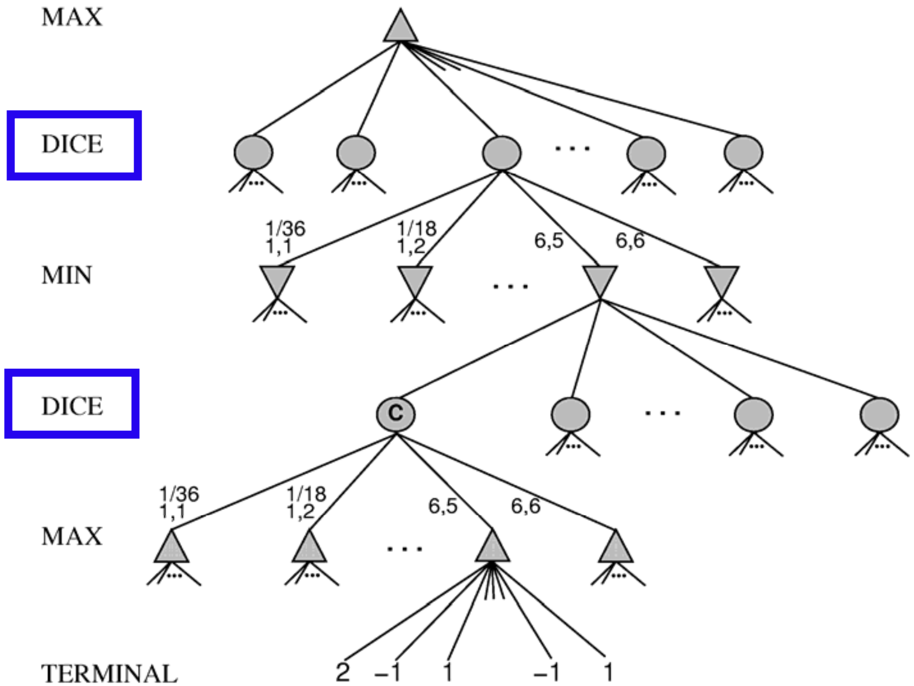
What about Games that Include an Element of Chance?



White has just rolled 6-5 and has 4 legal moves.

Game Tree for Games with an Element of Chance

- In addition to MIN- and MAX nodes, we include chance nodes (e.g., for rolling dice).



Expectiminimax Algorithm:

For chance nodes, compute expected value over successors

- Search costs increase: Instead of $O(b^d)$, we get $O((bn)^d)$, where n is the number of chance outcomes.

Imperfect Information

E.g. card games, where opponents' initial cards are unknown or Scrabble where letters are unknown

Idea: For all deals consistent with what you can see

- compute the minimax value of available actions for each of possible deals
- compute the expected value over all deals

Game Playing in Practice

- Chess: Deep Blue defeated human world champion Gary Kasparov in a 6 game match in 1997. Deep Blue searched 200 million positions per second, used very sophisticated evaluation functions, and undisclosed methods for extending some lines of search up to 40 ply
- Checkers: Chinook ended 40 year reign of human world champion Marion Tinsley in 1994; used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions (!)
- Othello: human champions refuse to play against computers because software is too good
- Go: human champions refuse to play against computers because software is too bad

Summary of Game Playing using Search

Basic idea: Minimax search (but can be slow)

Alpha-Beta pruning can increase max depth by factor up to 2

Limited depth search may be necessary

Static evaluation functions necessary for limited depth search

Opening and End game databases can help

Computers can beat humans in some games (checkers, chess, othello) but not in others (Go)

Next: Logic and Reasoning

“Thinking Rationally”

Computational models of human “thought” processes

Computational models of human behavior

Computational systems that “think” rationally

Computational systems that behave rationally

Logical Agents

Chess program calculates legal moves, but doesn't know that no piece can be on 2 different squares at the same time

Logic (Knowledge-Based) agents combine general knowledge about the world with current percepts to infer hidden aspects of current state prior to selecting actions

- Crucial in partially observable environments

Outline

Knowledge-based agents

Wumpus world

Logic in general

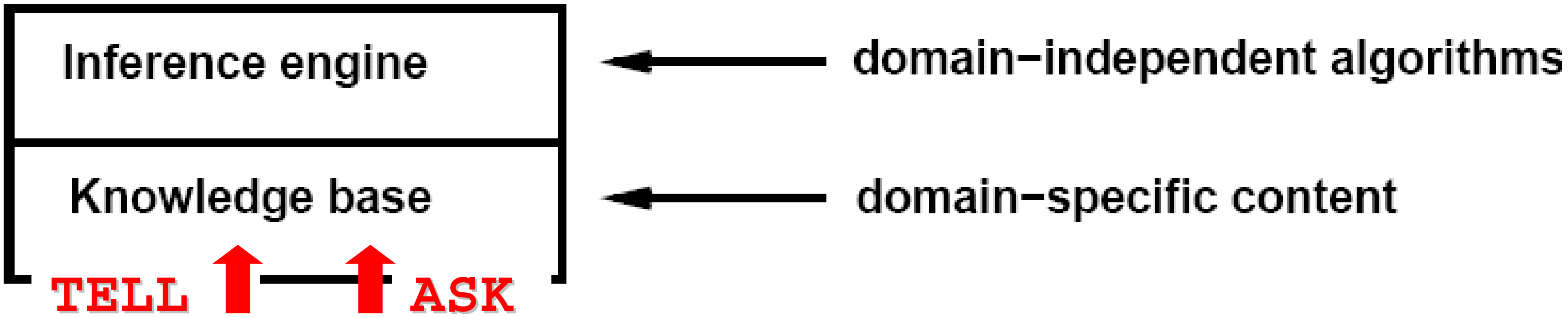
Propositional logic

- Inference, validity, equivalence and satisfiability
- Reasoning
  - Resolution
  - Forward/backward chaining

Knowledge Base

Knowledge Base : set of sentences represented in a knowledge representation language

- stores assertions about the world



Inference rule: when one ASKs questions of the KB, the answer should *follow* from what has been TELLed to the KB previously

Generic KB-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*))
  $t \leftarrow t + 1$
  return *action*

Abilities of a KB agent

Agent must be able to:

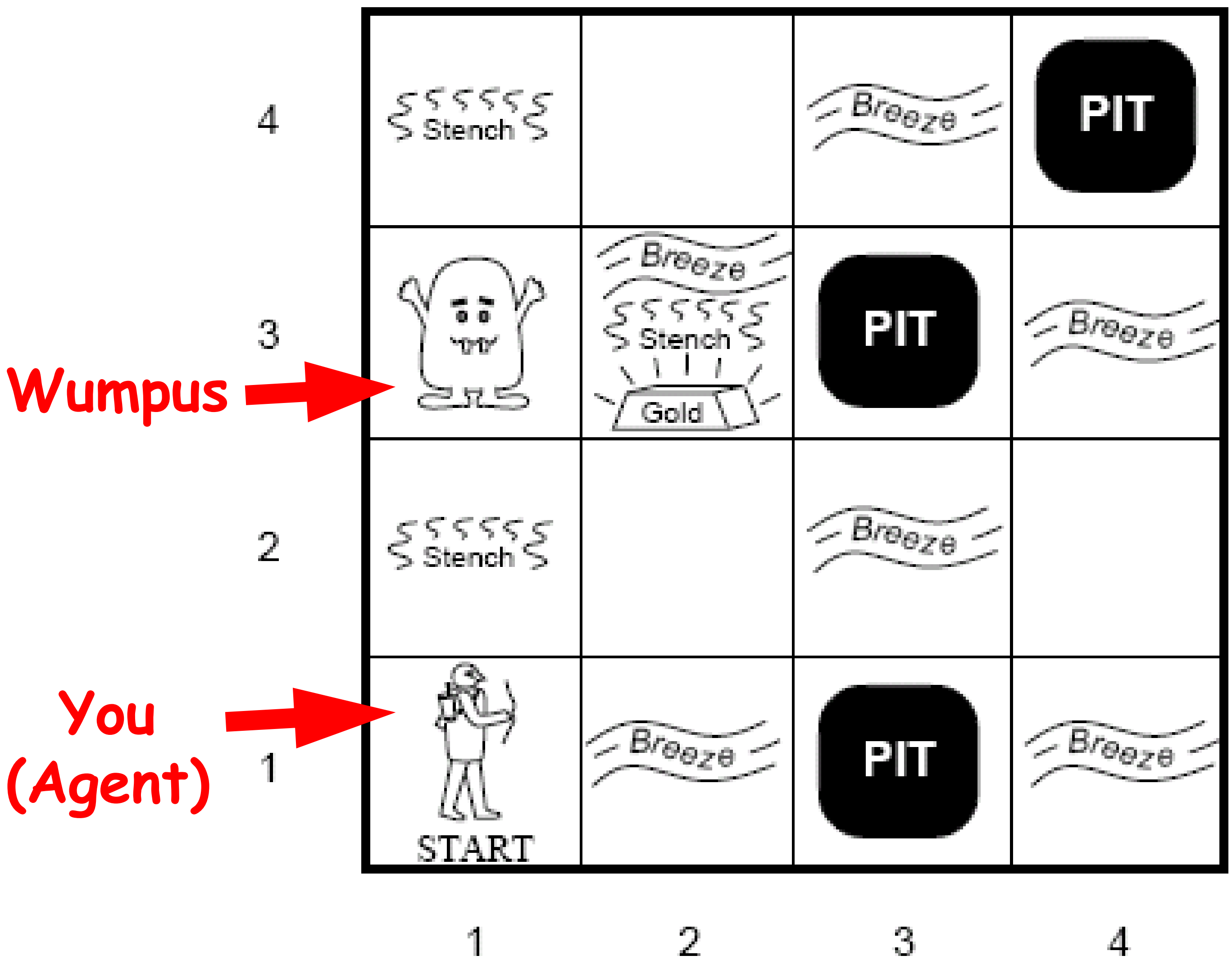
- Represent states and actions
- Incorporate new percepts
- Update internal representation of the world
- Deduce hidden properties of the world
- Deduce appropriate actions

Description level

Agents can be described at different levels

- Knowledge level
  - What they know, regardless of the actual implementation (Declarative description)
- Implementation level
  - Data structures in KB and algorithms that manipulate them, e.g., propositional logic and resolution

A Typical Wumpus World



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
- Climbing in [1,1] gets agent out of the cave

Sensors Stench, Breeze, Glitter, Bump, Scream

Actuators TurnLeft, TurnRight, Forward, Grab, Shoot, Climb

Wumpus World Characterization

Observable?

Deterministic?

Episodic?

Static?

Discrete?

Single-agent?

Wumpus World Characterization

Observable? No, only local perception

Deterministic?

Episodic?

Static?

Discrete?

Single-agent?

Wumpus World Characterization

Observable? No, only local perception

Deterministic? Yes, outcome exactly specified

Episodic?

Static?

Discrete?

Single-agent?

Wumpus World Characterization

Observable? No, only local perception

Deterministic? Yes, outcome exactly specified

Episodic? No, sequential at the level of actions

Static?

Discrete?

Single-agent?

Wumpus World Characterization

Observable? No, only local perception

Deterministic? Yes, outcome exactly specified

Episodic? No, sequential at the level of actions

Static? Yes, Wumpus and pits do not move

Discrete?

Single-agent?

Wumpus World Characterization

Observable? No, only local perception
Deterministic? Yes, outcome exactly specified
Episodic? No, sequential at the level of actions
Static? Yes, Wumpus and pits do not move
Discrete? Yes
Single-agent?

Wumpus World Characterization

Observable? No, only local perception

Deterministic? Yes, outcome exactly specified

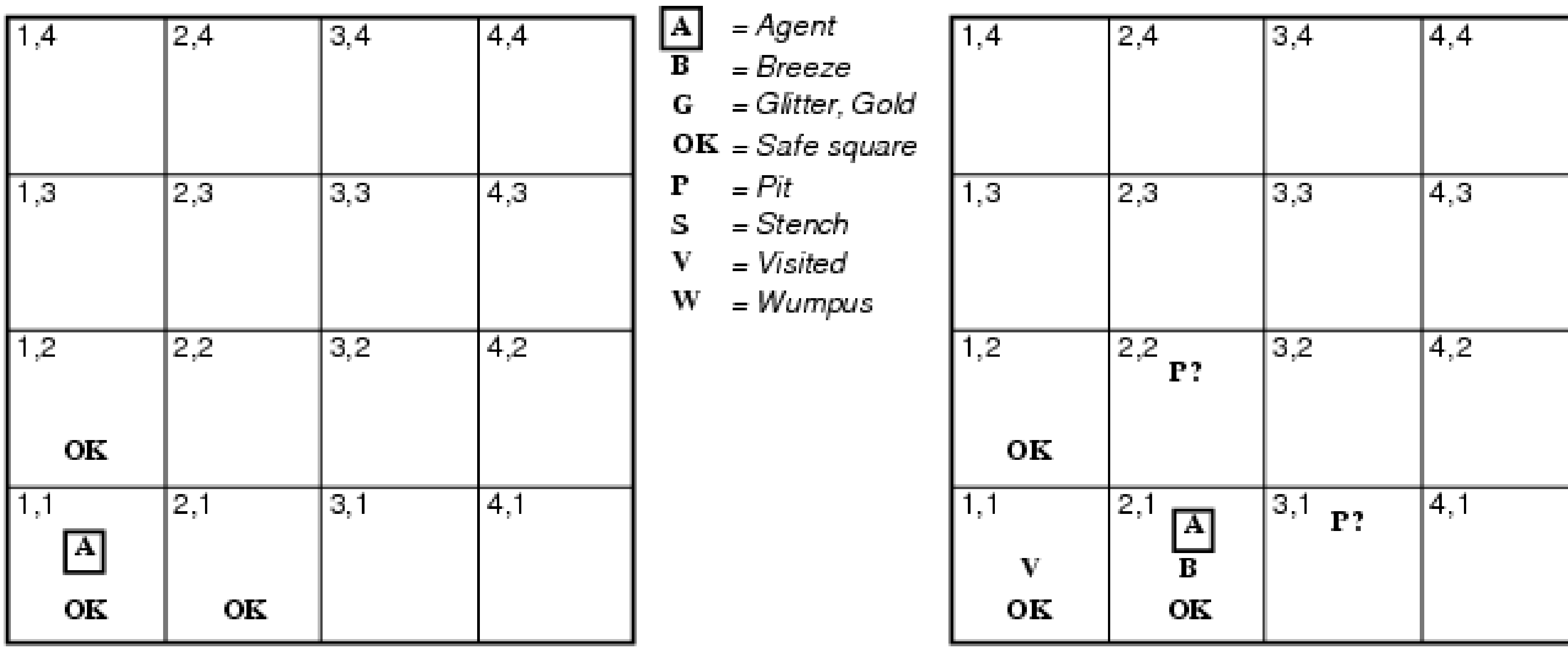
Episodic? No, sequential at the level of actions

Static? Yes, Wumpus and pits do not move

Discrete? Yes

Single-agent? Yes, Wumpus is essentially a "natural" feature of the environment

Exploring the Wumpus World

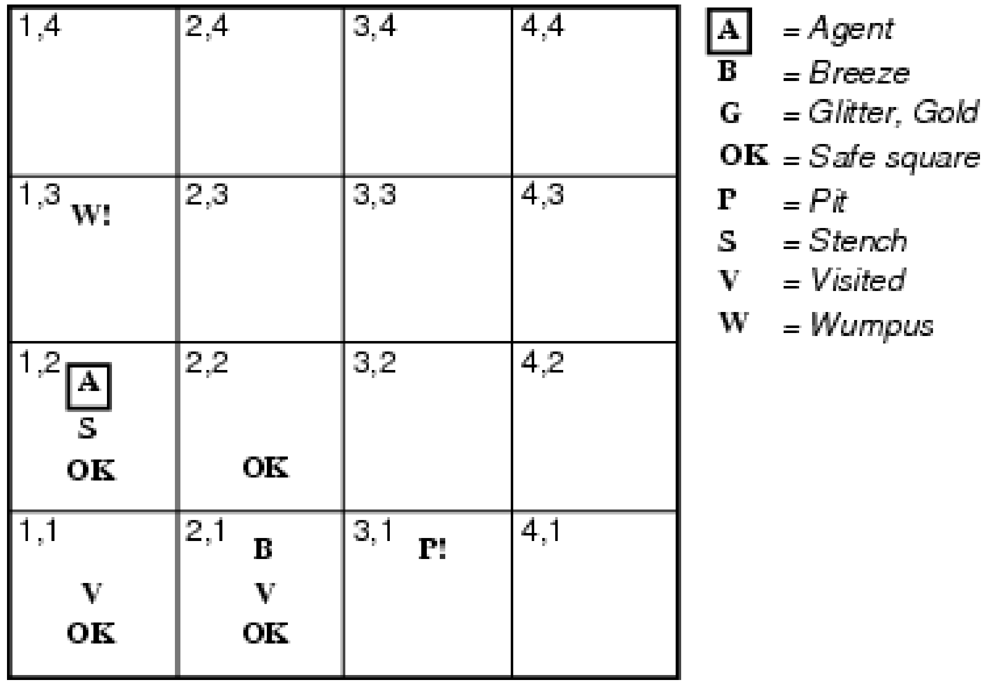


(a) (b)

[1,1] KB initially contains the rules of the environment. First percept is [*none,none,none,none,none*], move to safe cell e.g. 2,1

[2,1] Breeze which indicates that there is a pit in [2,2] or [3,1], return to [1,1] to try next safe cell

Exploring the Wumpus World



[1,2] Stench in cell which means that wumpus is in [1,3] or [2,2] but not in [1,1]

YET … wumpus not in [2,2] or stench would have been detected in [2,1]

THUS … wumpus must be in [1,3]

THUS [2,2] is safe because of lack of breeze in [1,2]

THUS pit in [3,1]

move to next safe cell [2,2]

Exploring the Wumpus World

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus

1,4	2,4 P?	3,4	4,4
1,3 W!	2,3 A S G B	3,3 P?	4,3
1,2 S V OK	2,2 V OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

[2,2] Move to [2,3]
[2,3] Detect glitter, smell, breeze
Grab gold
THUS pit in [3,3] or [2,4]

How do we represent rules of the world and percepts encountered so far?

Why not use logic?

What is a logic?

A formal language

- Syntax – what expressions are legal (well-formed)
- Semantics – what legal expressions mean
  - In logic the truth of each sentence evaluated with respect to *each possible world*

E.g the language of arithmetic

- x+2 >= y is a sentence, x2y+= is not a sentence
- x+2 >= y is true in a world where x=7 and y=1
- x+2 >= y is false in a world where x=0 and y=6

How do we draw conclusions and deduce new facts about the world using logic?

Entailment

Knowledge Base = KB

Sentence α

KB $\models$ α (KB "entails" sentence α)
if and only if α is true in all worlds (models) where KB is true.

E.g. x+y=4 entails 4=x+y
(because 4=x+y is true for all values of x, y for which x+y=4 is true)

Models and Entailment

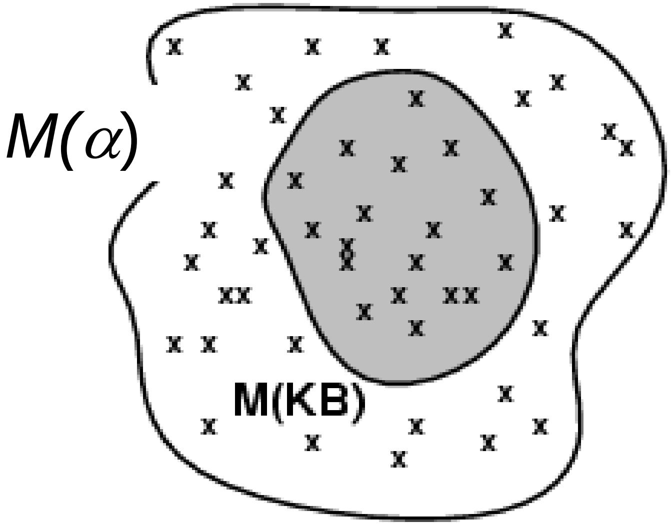
m is a model of a sentence α if α is true in *m*

e.g. α is "4=x+y" and $m = \{x=2, y=2\}$

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

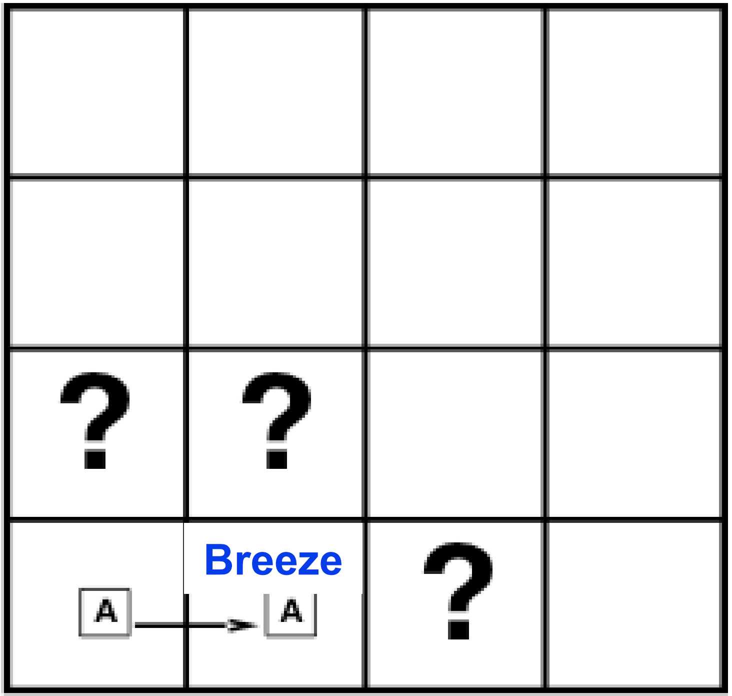
Then KB $\models \alpha$ iff $M(KB) \subseteq M(\alpha)$

E.g. *KB* = CSEP 573 students are bored and
CSEP 573 students are sleepy;
α = CSEP 573 students are bored



Wumpus world model

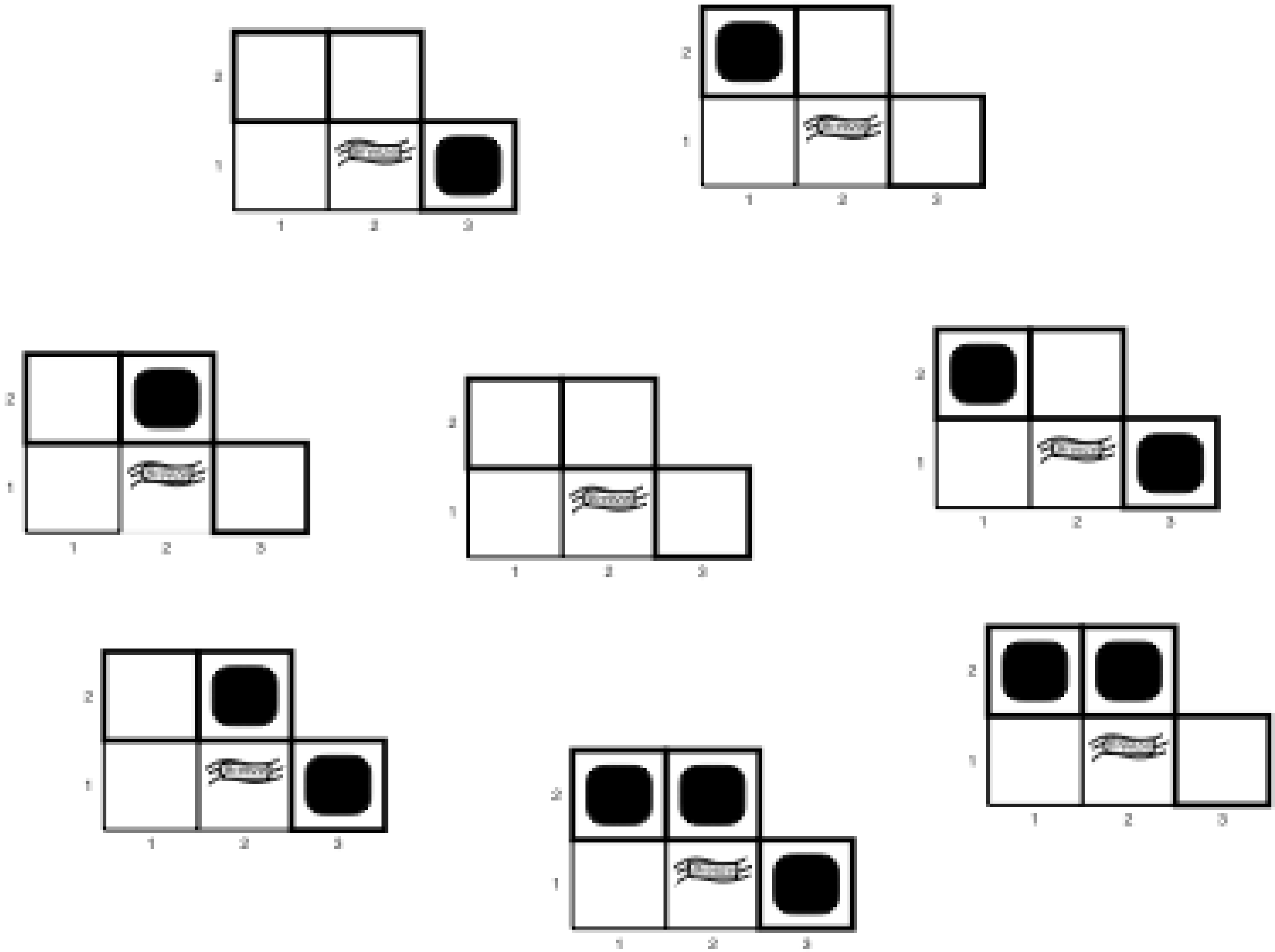
Situation after detecting nothing in [1,1], moving right, breeze in [2,1]



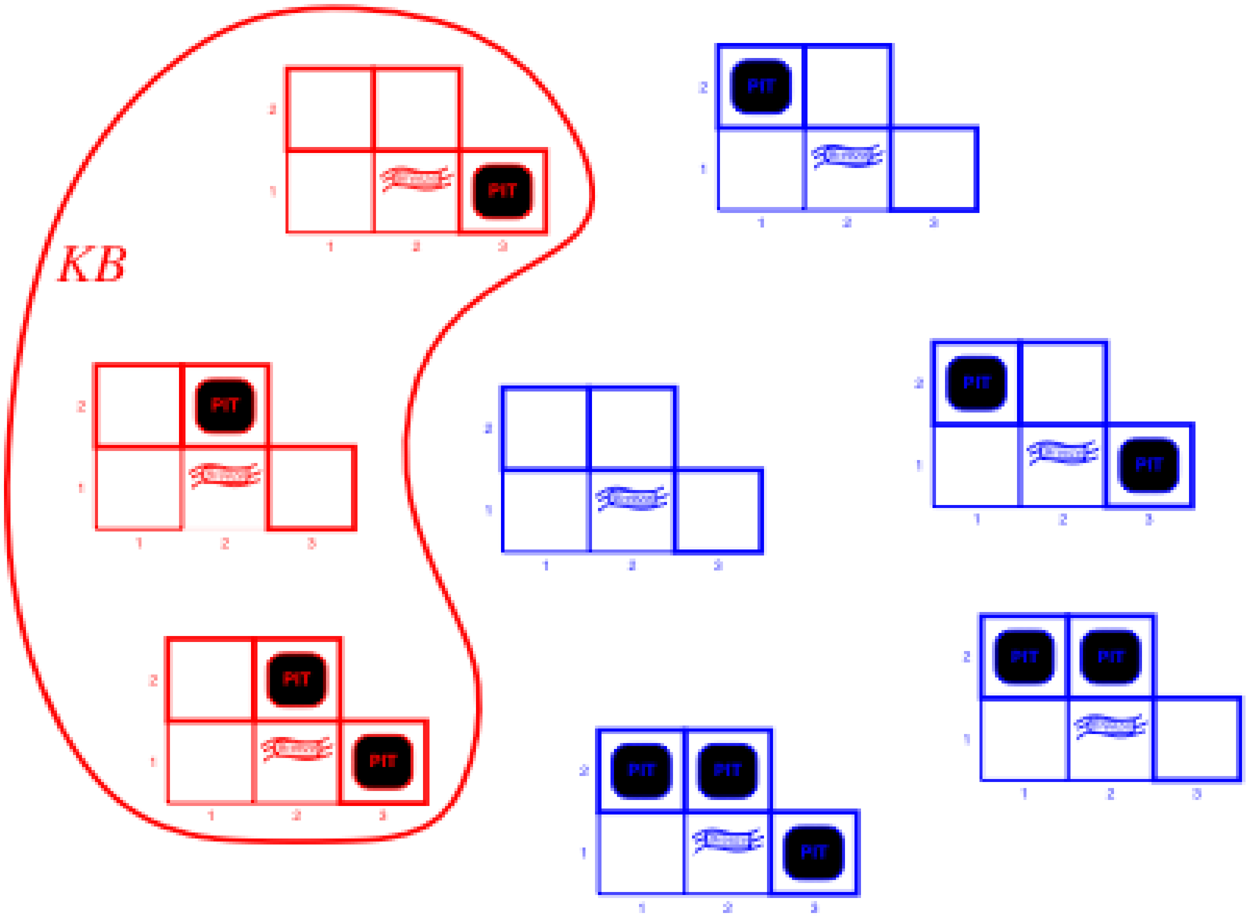
Consider possible models for ?s assuming only pits

3 Boolean choices $\Rightarrow$ 8 possible models

Wumpus possible world models



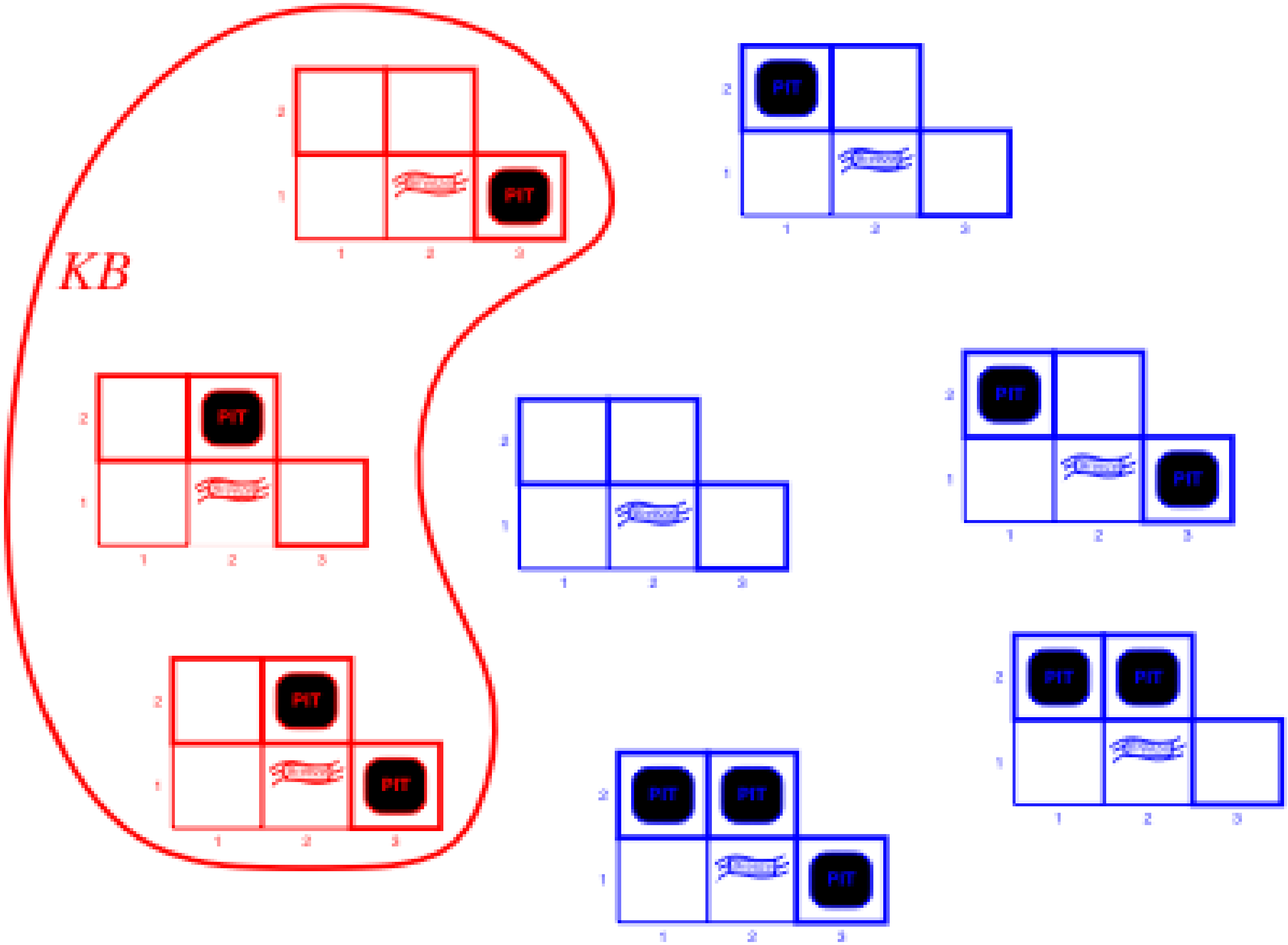
Wumpus world models consistent with observations



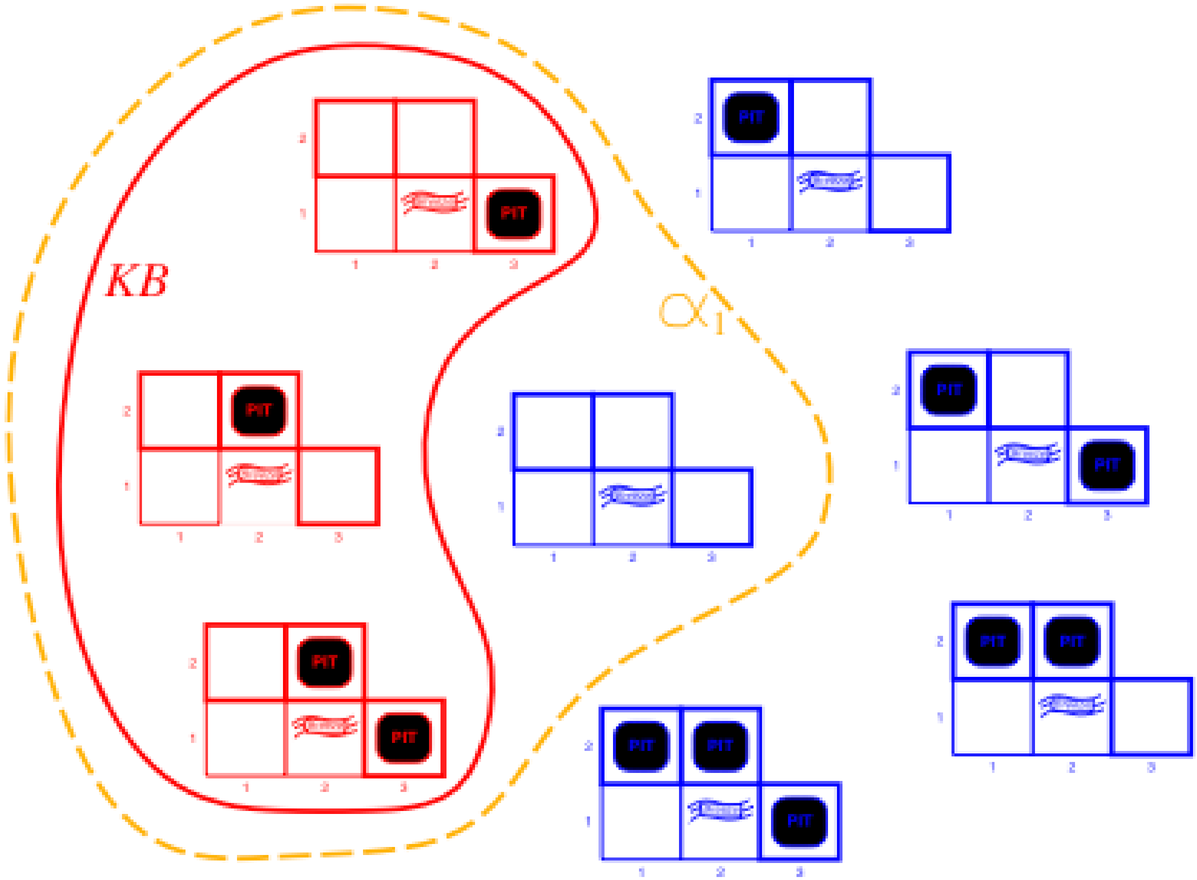
KB = wumpus-world rules + observations

Example of Entailment

Is [1,2] safe?



Example of Entailment



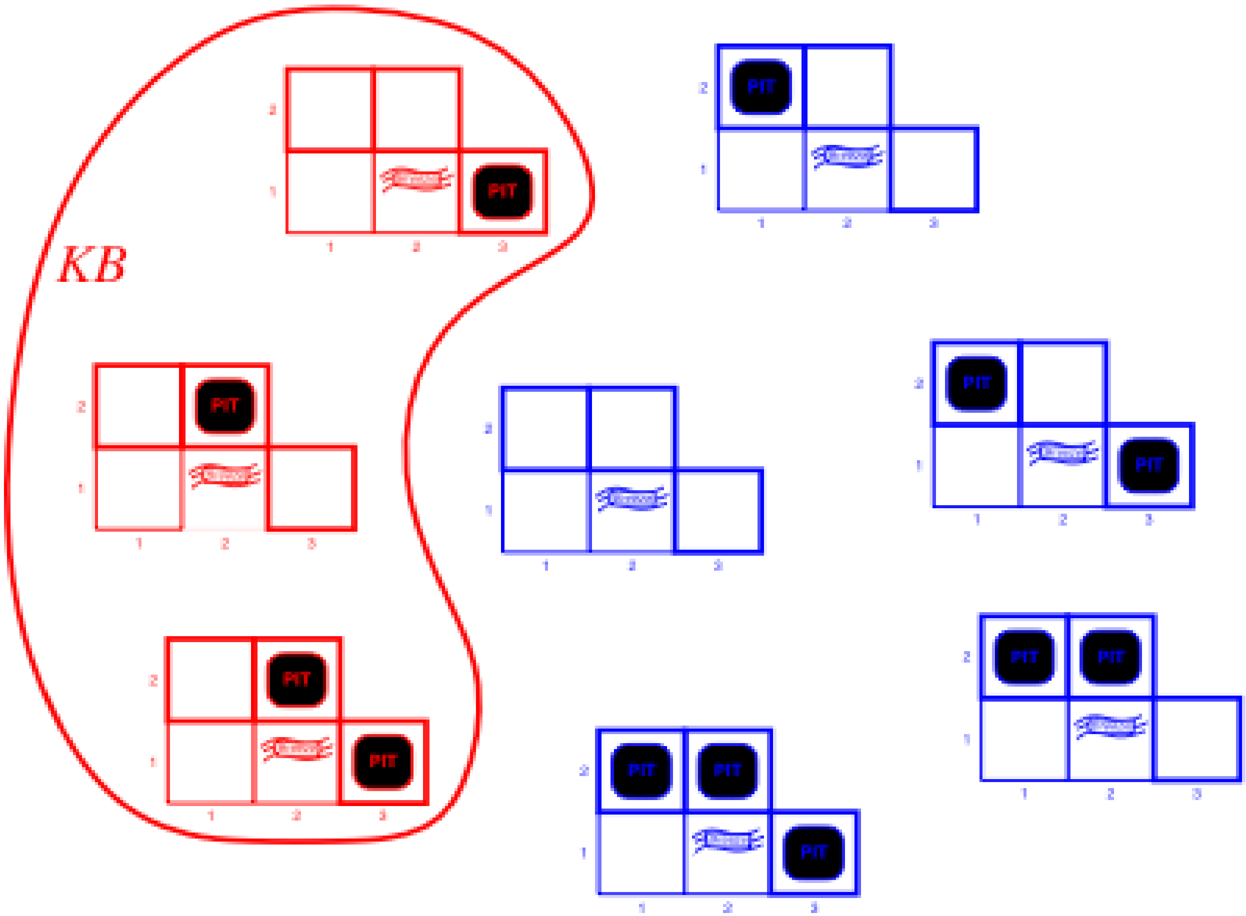
KB = wumpus-world rules + observations

$M(KB) \subseteq M(\alpha_1)$

α_1 = "[1,2] is safe", $KB \models \alpha_1$, proved by model checking

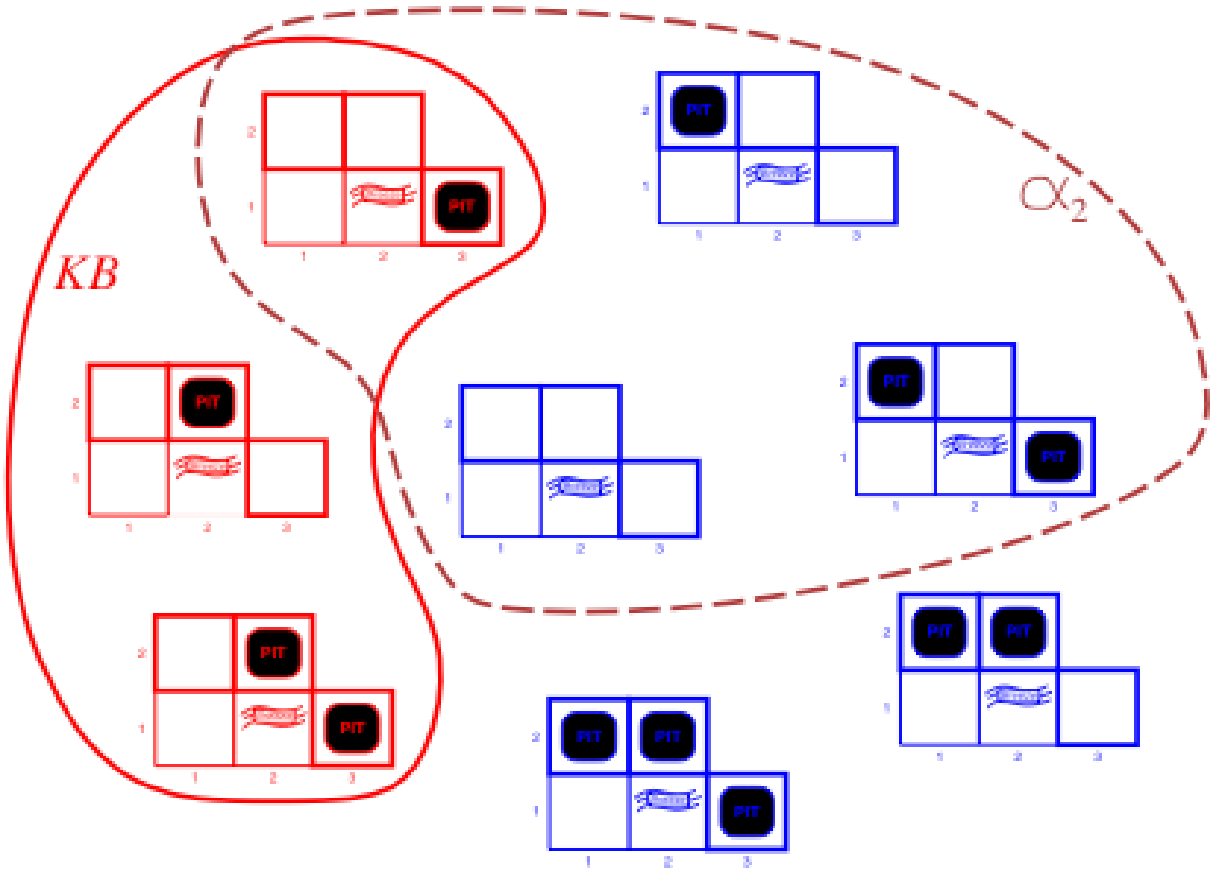
Another Example

Is [2,2] safe?



KB = wumpus-world rules + observations

Another Example



KB = wumpus-world rules + observations

α_2 = "[2,2] is safe", $\boxed{KB \not\models \alpha_2}$ **$M(KB) \not\subseteq M(\alpha_2)$**

Soundness and Completeness

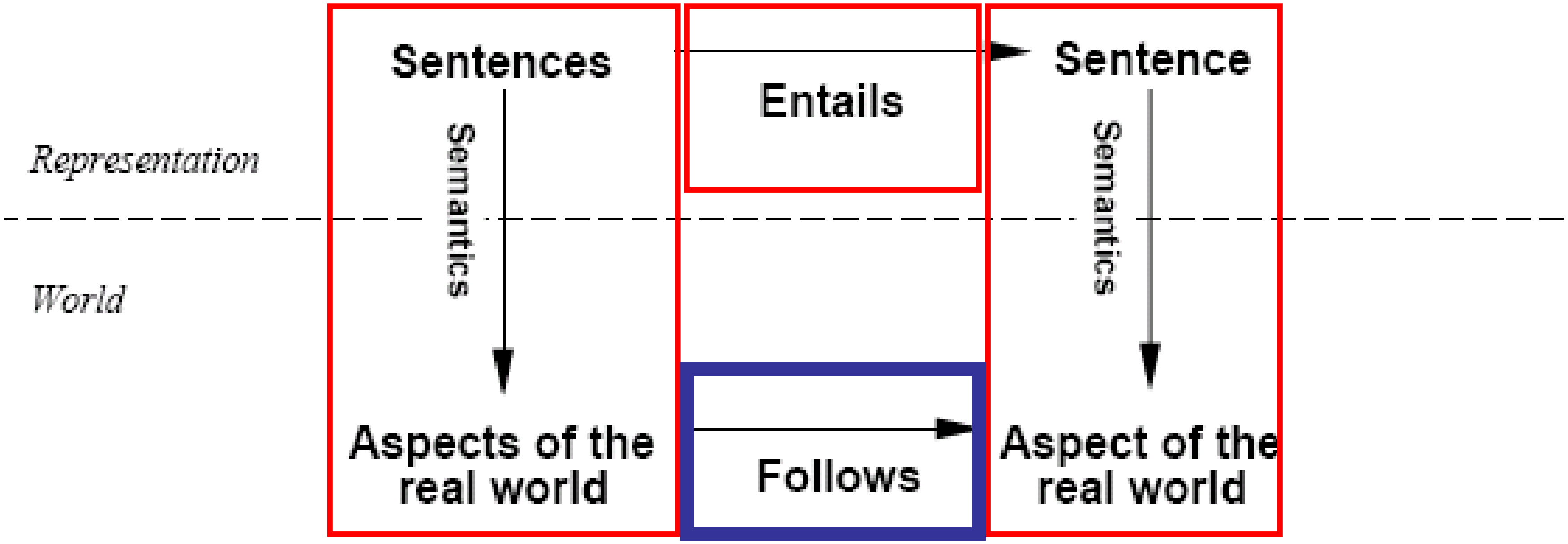
If an inference algorithm only derives entailed sentences, it is called *sound* (or *truth preserving*).

- Otherwise it just makes things up
- *Algorithm i is sound if whenever KB $\vdash_i \alpha$ (i.e. α is derived by i from KB) it is also true that KB $\models \alpha$*

Completeness: An algorithm is complete if it can derive any sentence that is entailed.

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

Relating to the Real World



If KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world

Propositional Logic: Syntax

Propositional logic is the simplest logic – illustrates basic ideas

Atomic sentences = proposition symbols = A, B, $P_{1,2}$, $P_{2,2}$ etc. used to denote properties of the world

- *Can be either True or False*

E.g. $P_{1,2}$ = "There's a pit in location [1,2]" is either true or false in the wumpus world

Propositional Logic: Syntax

Complex sentences constructed from simpler ones <u>recursively</u>

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, $S_1 \Rightarrow S_2$ is a sentence (implication)

If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Propositional Logic: Semantics

A <u>model</u> specifies true/false for each proposition symbol

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

Rules for evaluating truth <u>w.r.t. a model *m*</u>:

$\neg S$	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
$S_1 \Rightarrow S_2$	is true	iff	S_1 is false or S_2 is true
$S_1 \Leftrightarrow S_2$	is true	iff	both $S_1 \Rightarrow S_2$ and $S_2 \Rightarrow S_1$ are 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*

Propositional Logic: Semantics

Simple recursive process can be used to evaluate an arbitrary sentence

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

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

$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1})$

$= \textit{true} \wedge (\textit{true} \vee \textit{false})$

$= \textit{true} \wedge \textit{true}$

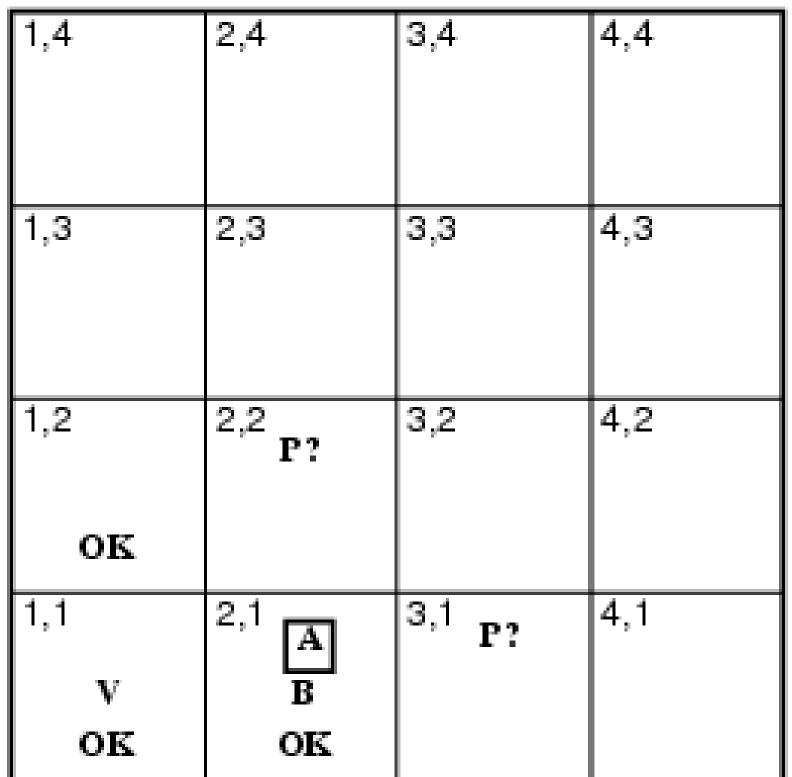
$= \textit{true}$

Example: Wumpus World

Proposition Symbols and Semantics:

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].



Wumpus KB

Knowledge Base (KB) includes the following sentences:

Statements currently known to be true:

$\neg P_{1,1}$

$\neg B_{1,1}$

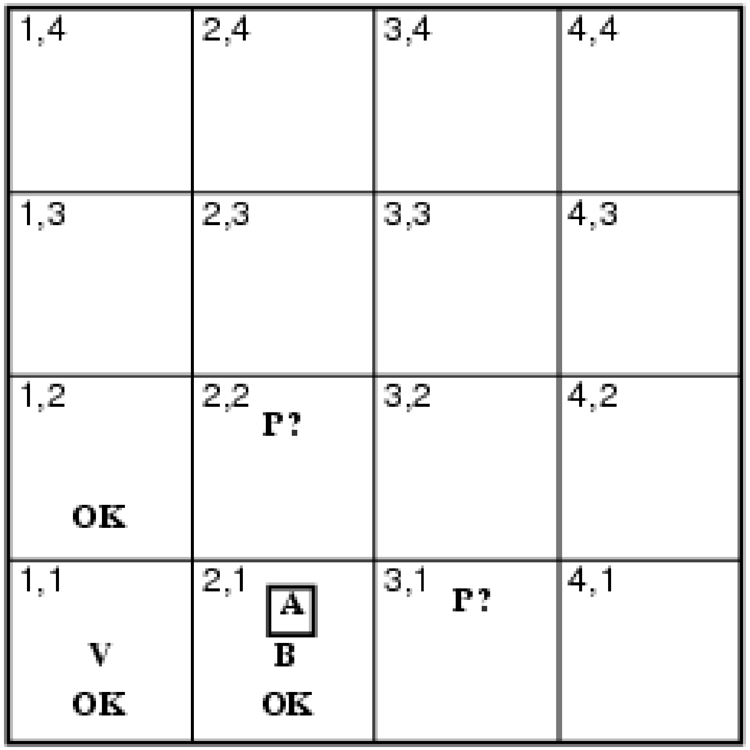
$B_{2,1}$

Properties of the world: E.g., "Pits cause breezes in adjacent squares"

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

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

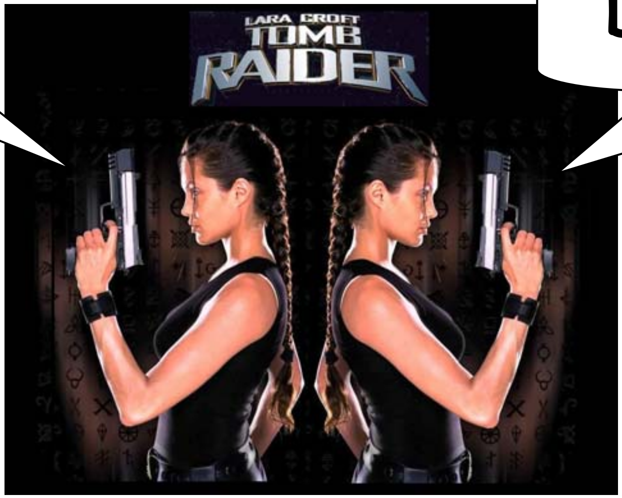
(and so on for all squares)



Can a Wumpus-Agent use this logical representation and KB to avoid pits and the wumpus, and find the gold?

Is there no pit in [1,2]?

Does KB $\models \neg P_{1,2}$?



Inference by Truth Table Enumeration

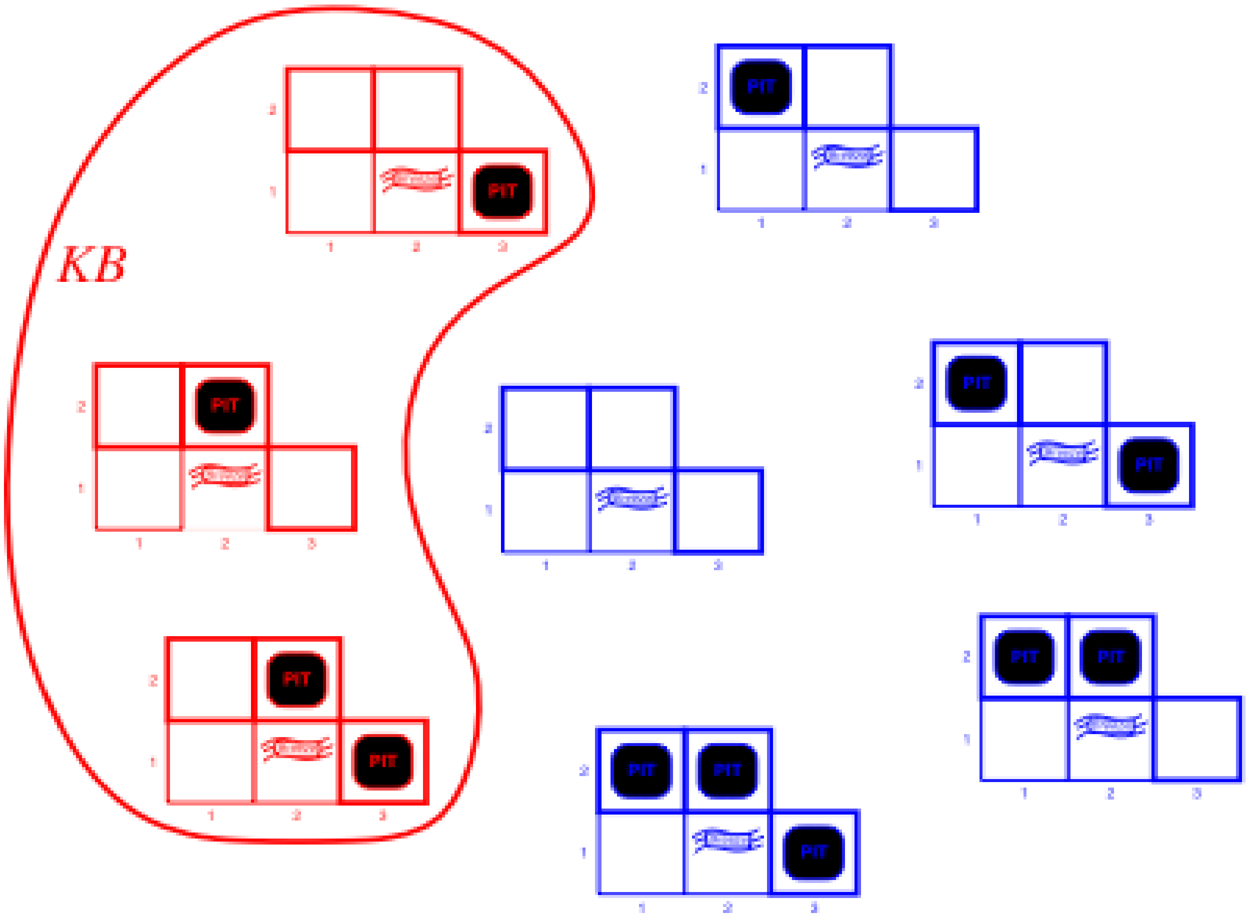
$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	$\neg P_{1,2}$
false	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*false*	*false*	*false*	*false*	*false*	*true*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*true*	*false*	*false*	*false*	*false*	*true*	<u>*true*</u>	<u>*true*</u>
false	*true*	*false*	*false*	*false*	*true*	*false*	<u>*true*</u>	<u>*true*</u>
false	*true*	*false*	*false*	*false*	*true*	*true*	<u>*true*</u>	<u>*true*</u>
false	*true*	*false*	*false*	*true*	*false*	*false*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*false*

$\neg P_{1,2}$ true in all models in which KB is true

Therefore, KB $\models \neg P_{1,2}$

Another Example

Is there a pit in [2,2]?



Inference by Truth Table Enumeration

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB
$false$	$false$	$false$	$false$	$false$	$false$	$false$	$false$
$false$	$false$	$false$	$false$	$false$	$false$	$true$	$false$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$false$	$true$	$false$	$false$	$false$	$false$	$false$	$false$
$false$	$true$	$false$	$false$	$false$	$false$	$true$	$\underline{true}$
$false$	$true$	$false$	$false$	$false$	$true$	$false$	$\underline{true}$
$false$	$true$	$false$	$false$	$false$	$true$	$true$	$\underline{true}$
$false$	$true$	$false$	$false$	$true$	$false$	$false$	$false$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$true$	$true$	$true$	$true$	$true$	$true$	$true$	$false$

$P_{2,2}$ is false in a model in which KB is true

Therefore, KB $\not\models P_{2,2}$

Inference by TT Enumeration

Algorithm: Depth-first enumeration of all models (see Fig. 7.10 in text for pseudocode)

- Algorithm is sound & complete

For *n* symbols:

time complexity = $O(2^n)$, space = $O(n)$

Concepts for Other Techniques: Logical Equivalence

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

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

Concepts for Other Techniques: Validity and Satisfiability

A sentence is *valid* if it is true in *all* models (a tautology)

e.g., *True*, $A \lor \neg A$, $A \Rightarrow A$, $(A \land (A \Rightarrow B)) \Rightarrow B$

Validity is connected to inference via the Deduction Theorem:

$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

A sentence is *satisfiable* if it is true in *some* model

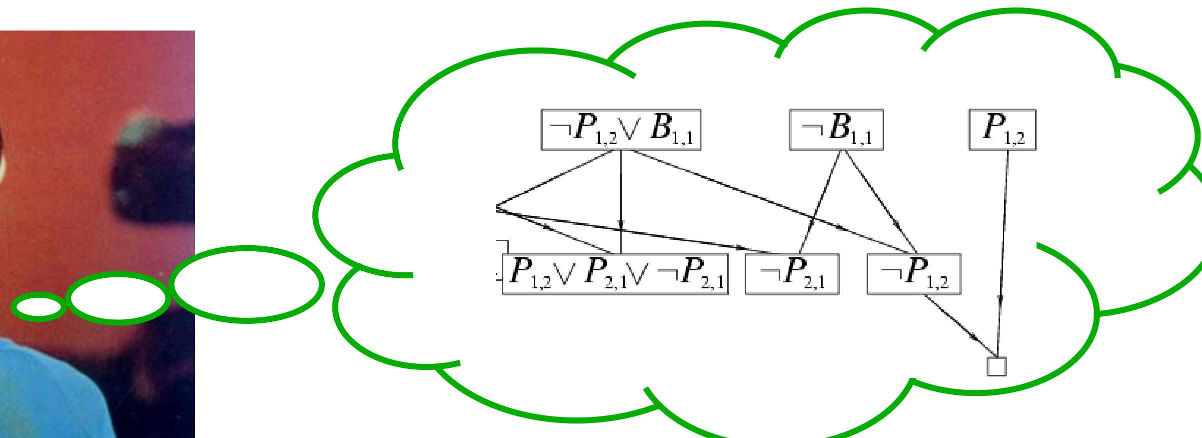
e.g., $A \lor B$, C

A sentence is *unsatisfiable* if it is true in no models

e.g., $A \land \neg A$

Satisfiability is connected to inference via the following: $KB \models \alpha$ if and only if $(KB \land \neg\alpha)$ is unsatisfiable (proof by contradiction)

Inference Techniques for Logical Reasoning



Inference/Proof Techniques

Two kinds (roughly):

Model checking

- Truth table enumeration (always exponential in *n*)
- Efficient backtracking algorithms
  e.g., Davis-Putnam-Logemann-Loveland (DPLL)
- Local search algorithms (sound but incomplete)
  e.g., randomized hill-climbing (WalkSAT)

Successive application of inference rules

- Generate new sentences from old in a sound way
- Proof = a sequence of inference rule applications
- Use inference rules as *successor function* in a standard search algorithm

Inference Technique I: Resolution

Terminology:

Literal = proposition symbol or its negation

E.g., A, $\neg A$, B, $\neg B$, etc.

Clause = disjunction of literals

E.g., $(B \vee \neg C \vee \neg D)$

Resolution assumes sentences are in *Conjunctive Normal Form (CNF):*

sentence = conjunction of clauses

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

Conversion to CNF

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

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta)\wedge(\beta \Rightarrow \alpha)$.
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

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 ($\wedge$ over $\vee$) and flatten:
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

   This is in CNF – Done!

Resolution motivation

There is a pit in [1,3] or
There is a pit in [2,2]

There is no pit in [2,2]

There is a pit in [1,3]

More generally,

$$\frac{\ell_1 \vee \ldots \vee \ell_k \qquad \neg \ell_i}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k}$$

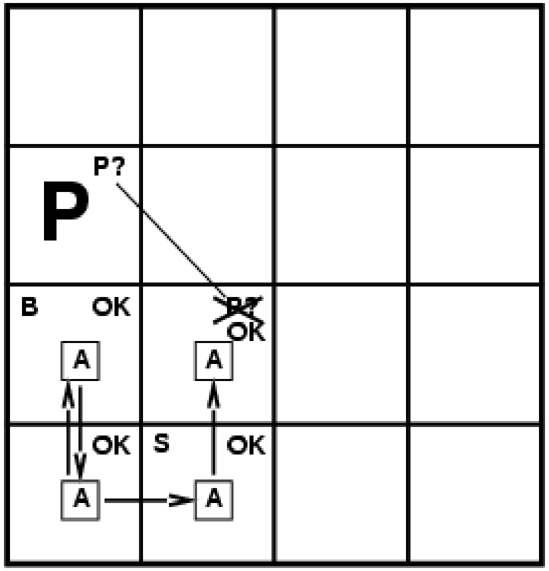
Inference Technique: Resolution

General Resolution inference rule (for CNF):

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

where ℓ_i and m_j are complementary literals ($\ell_i = \neg m_j$)

E.g., $\dfrac{P_{1,3} \vee P_{2,2} \qquad\qquad \neg P_{2,2}}{P_{1,3}}$



Resolution is sound and complete for propositional logic

Soundness

Proof of soundness of resolution inference rule:

$$\neg(l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow l_i$$

$$\neg m_j \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)$$

$$\neg(l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)$$

(since $l_i = \neg m_j$)

Resolution algorithm

To show KB $\models$ α, use proof by contradiction,
i.e., show *KB* $\wedge \neg\, \alpha$ unsatisfiable

function PL-RESOLUTION(KB, α) **returns** *true* or *false*

  clauses ← the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new ← { }
  loop do
    for each C_i, C_j **in** *clauses* **do**
      resolvents ← PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      new ← *new* ∪ *resolvents*
    if *new* ⊆ *clauses* **then return** *false*
    clauses ← *clauses* ∪ *new*

PL-RESOLUTION can be shown to be complete (see text)

Resolution example

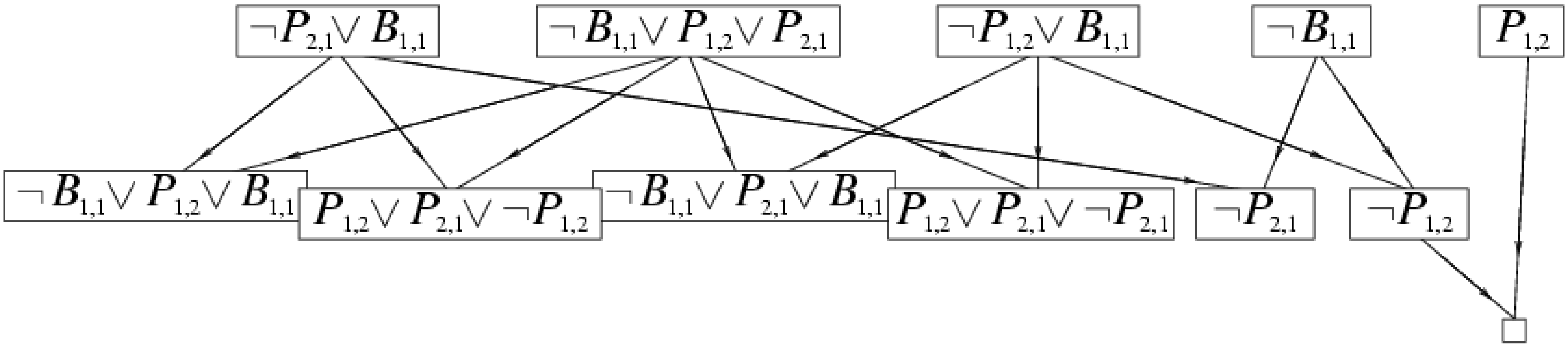
Given no breeze in [1,1], prove there's no pit in [1,2]

$KB = (B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1}$ and $\alpha = \neg P_{1,2}$

Resolution: Convert to CNF and show $KB \land \neg \alpha$ is unsatisfiable

Resolution example

Resolution example



Empty clause

(i.e., KB $\wedge \neg \alpha$ unsatisfiable)

Inference Technique II: Forward/Backward Chaining

Require sentences to be in Horn Form:

KB = conjunction of Horn clauses

- Horn clause =
  - proposition symbol or
  - "(conjunction of symbols) $\Rightarrow$ symbol"

    (i.e. clause with at most 1 positive literal)
- E.g., KB = $C \wedge (B \Rightarrow A) \wedge ((C \wedge D) \Rightarrow B)$

F/B chaining is based on "Modus Ponens" rule:

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

- Complete for Horn clauses

Very natural and linear time complexity in size of KB

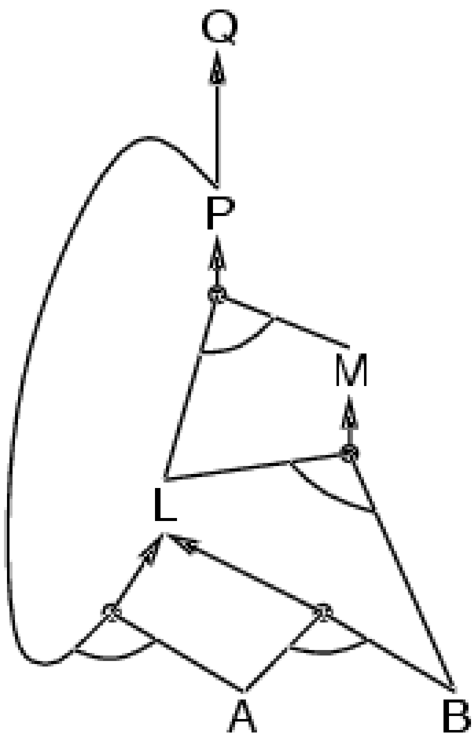
Forward chaining

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

KB:

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

Query: **"Is Q true?"**



AND-OR Graph for KB

Forward chaining algorithm

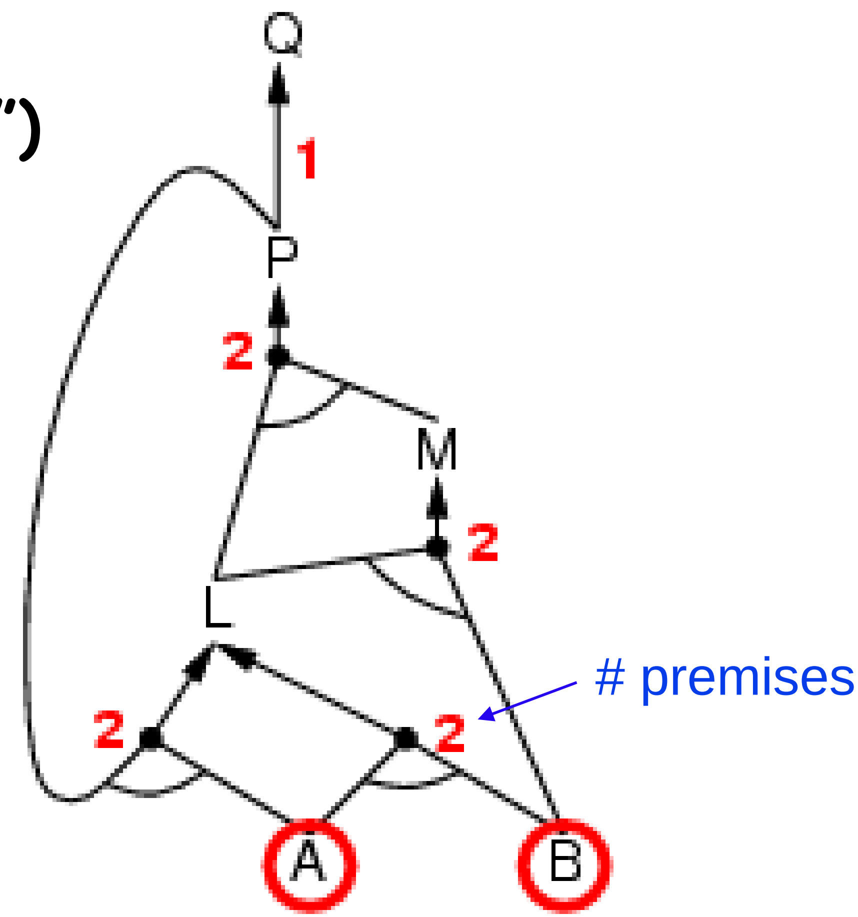
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known to be true

   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]     // Decrement # premises
               if count[c] = 0 then do    // All premises satisfied
                   if HEAD[c] = q then return true
                   PUSH(HEAD[c], agenda)
   return false
```

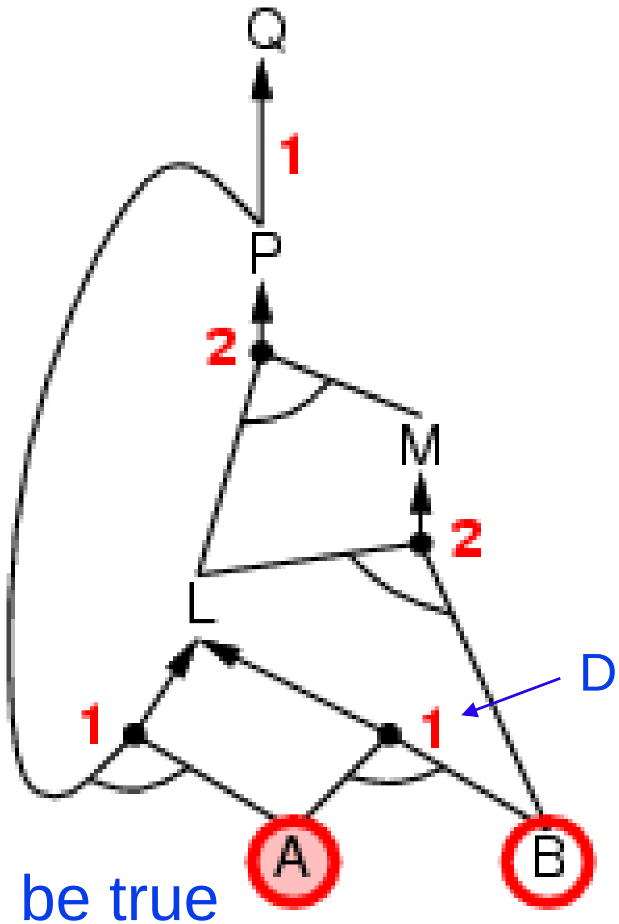
Forward chaining is sound & complete for Horn KB

Forward chaining example

Query = Q
(i.e. "Is Q true?")



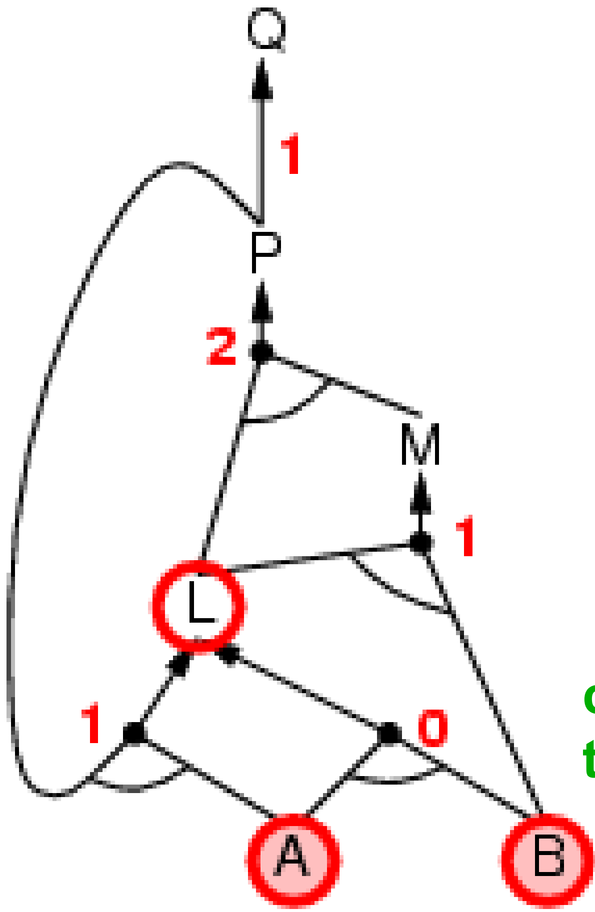
Forward chaining example



Decrement count

A is known to be true

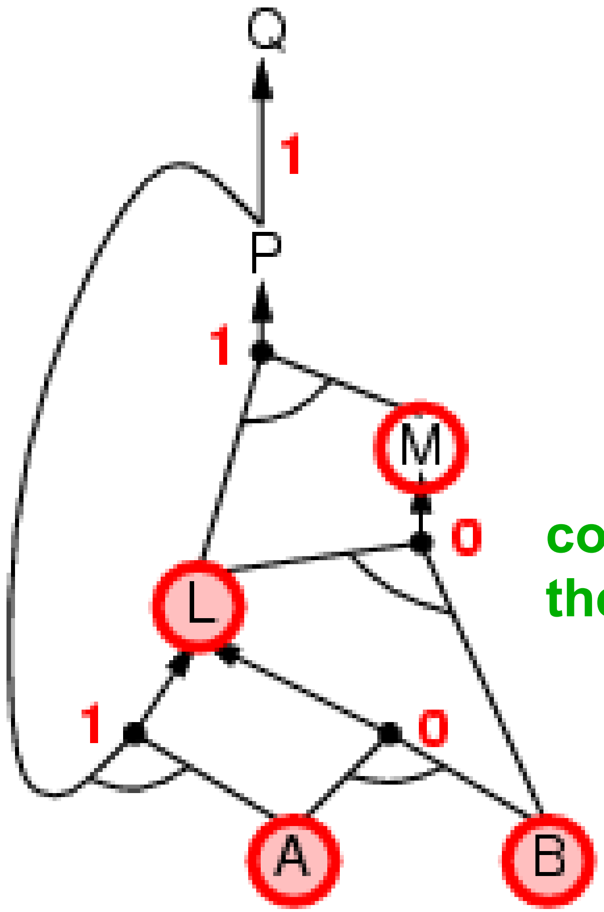
Forward chaining example



count = 0;
therefore, L is true

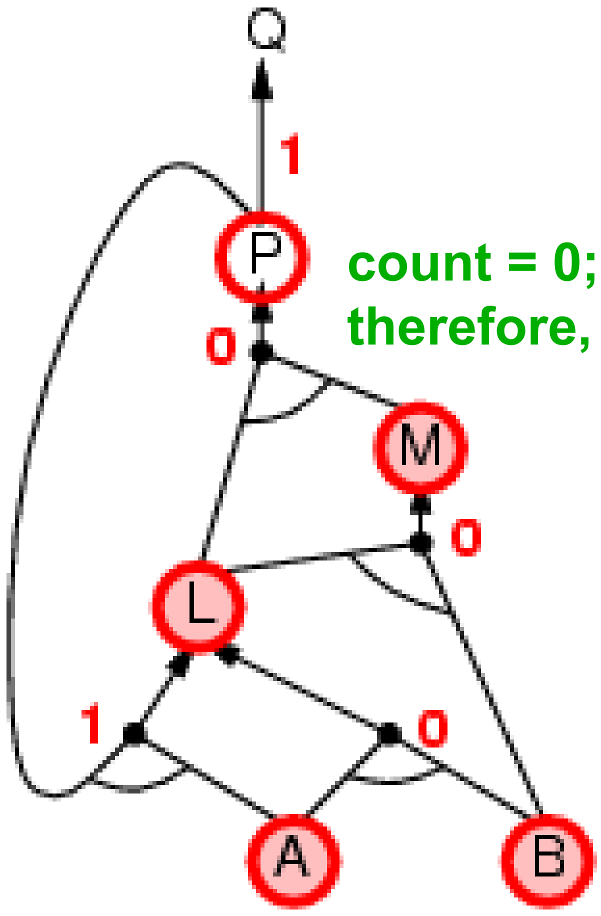
B is also known to be true

Forward chaining example



count = 0;
therefore, M is true

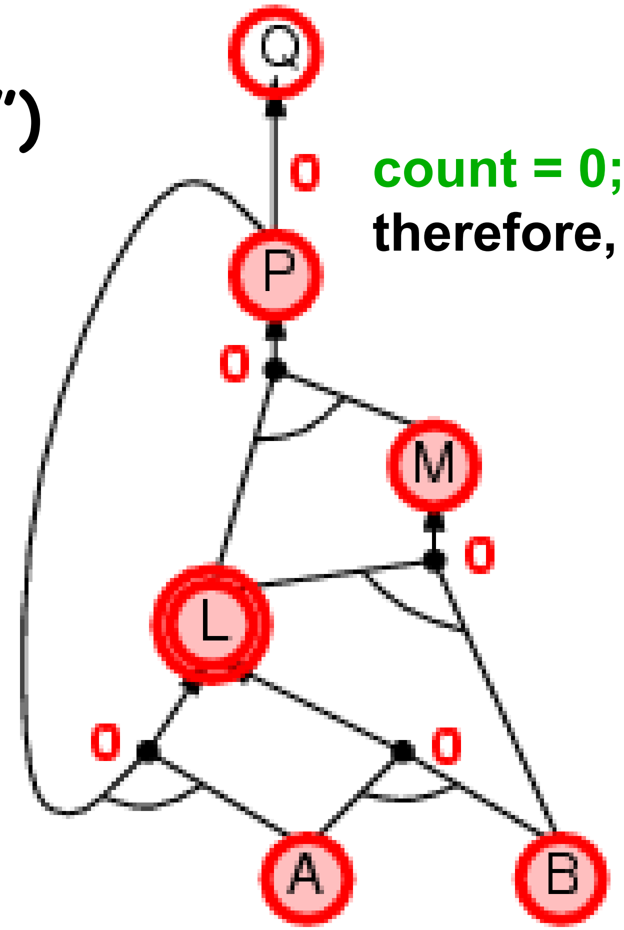
Forward chaining example



count = 0;
therefore, P is true

Forward chaining example

Query = Q
(i.e. "Is Q true?")



count = 0;
therefore, Q is true

Backward chaining

Idea: work backwards from the query q

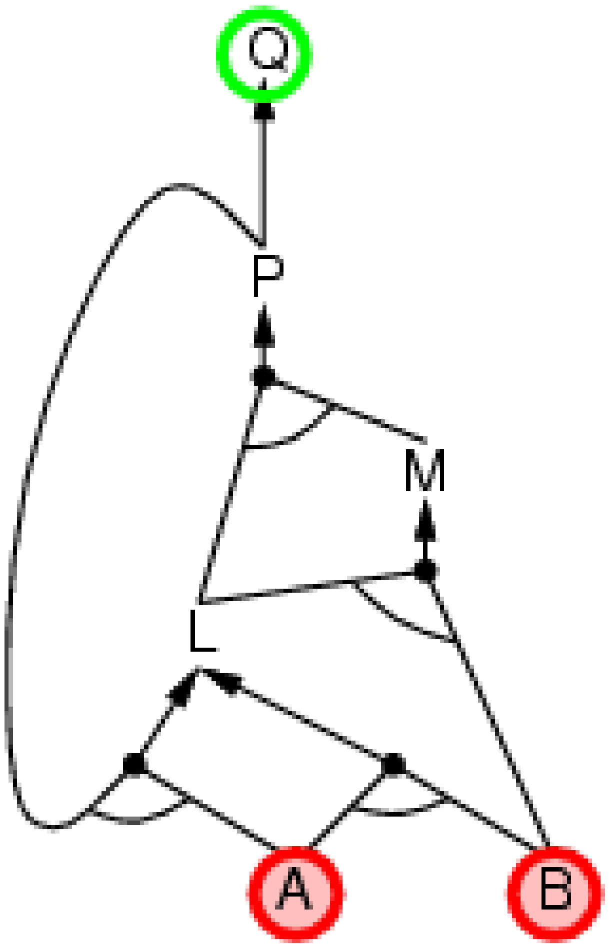
- to prove q:
  - check if q is known already, OR
  - prove by backward chaining all premises of some rule concluding q

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

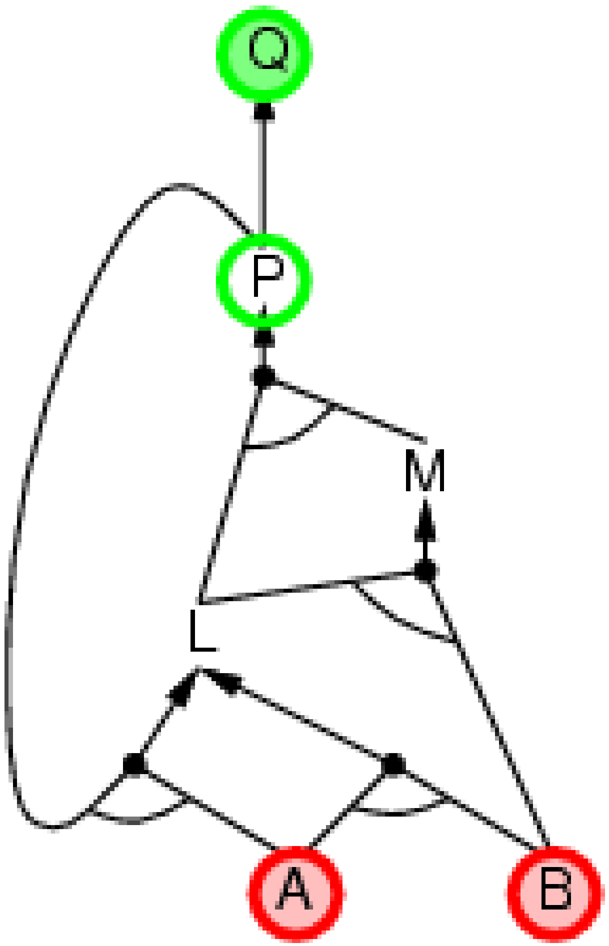
Avoid repeated work: check if new subgoal

1. has already been proved true, or
2. has already failed

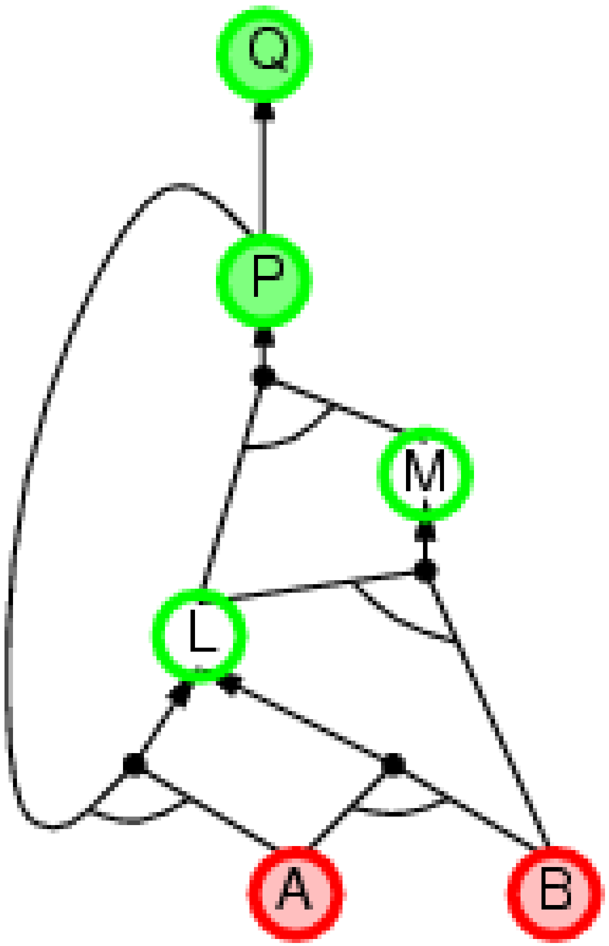
Backward chaining example



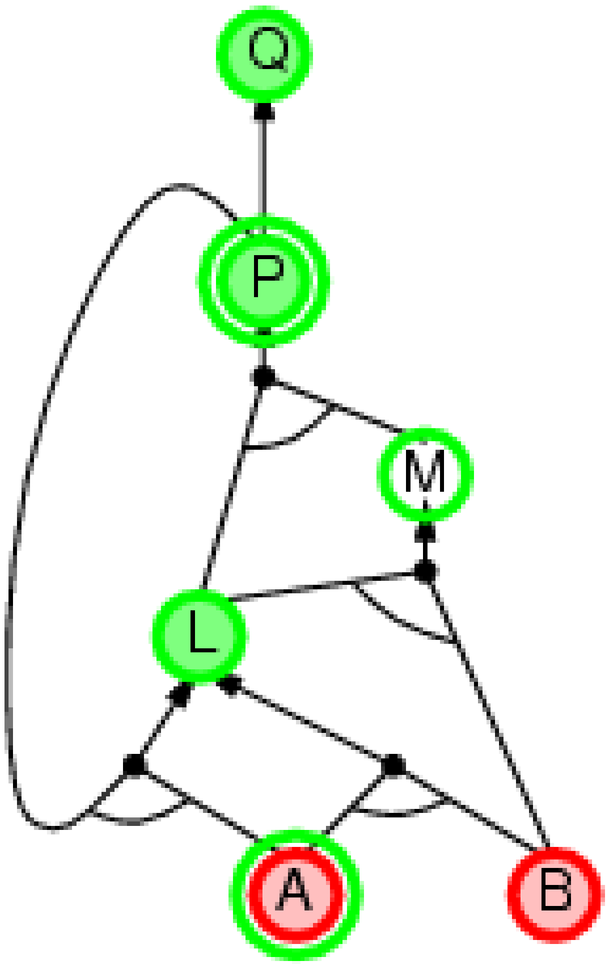
Backward chaining example



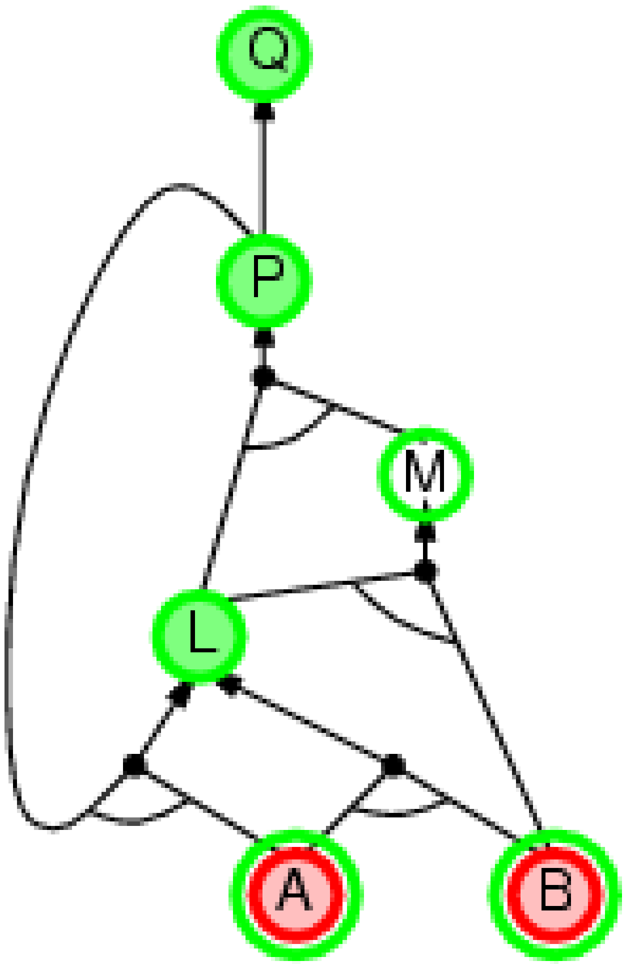
Backward chaining example



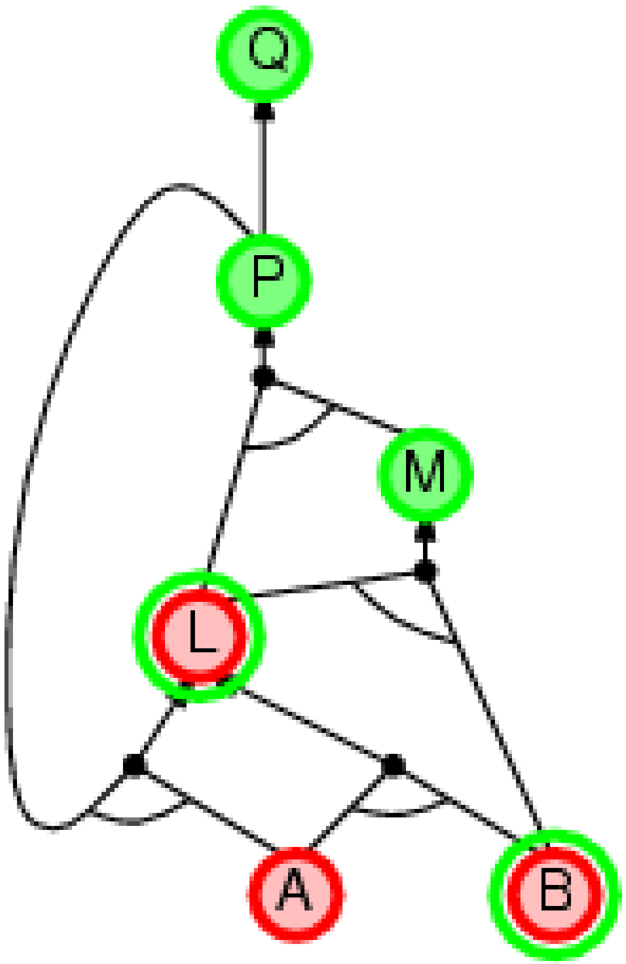
Backward chaining example



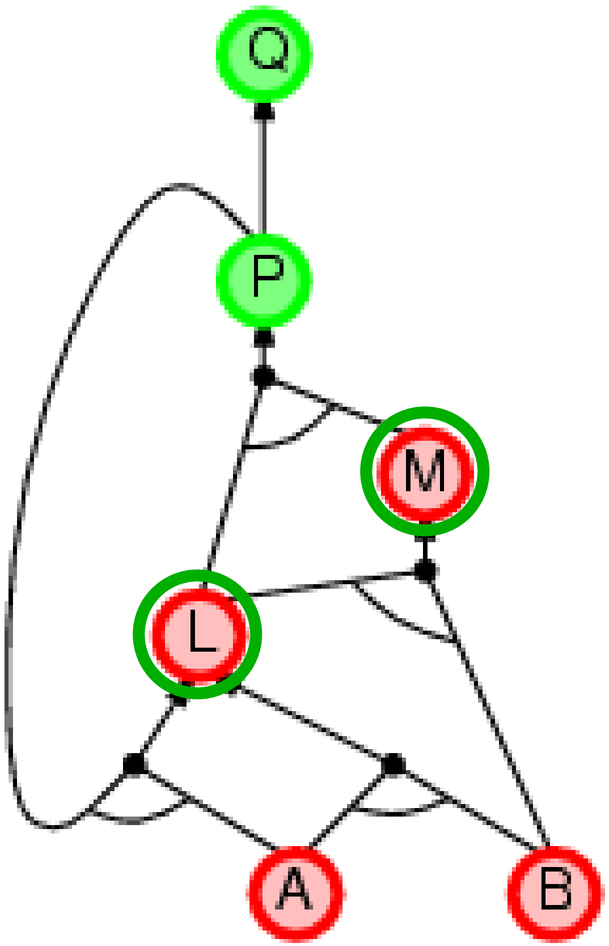
Backward chaining example



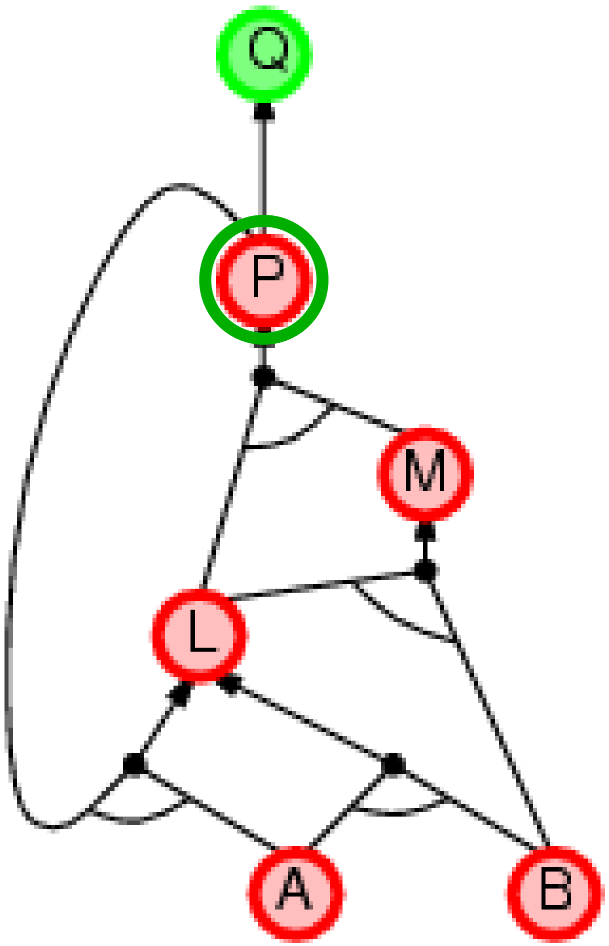
Backward chaining example



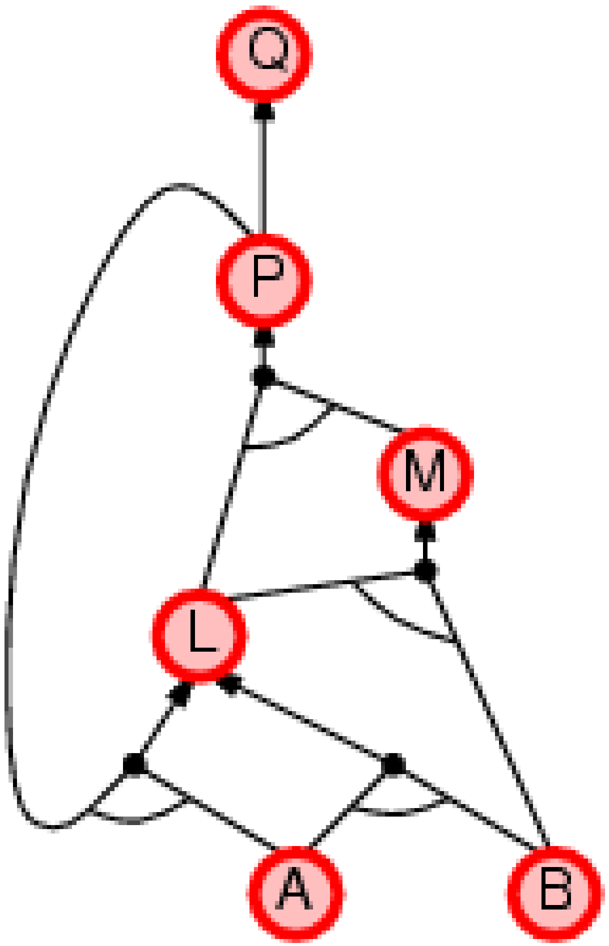
Backward chaining example



Backward chaining example



Backward chaining example



Forward vs. backward chaining

FC is data-driven, automatic, unconscious processing

- e.g., object recognition, routine decisions

FC may do lots of work that is irrelevant to the goal

BC is goal-driven, appropriate for problem-solving

- e.g., How do I get an A in this class?
- e.g., What is my best exit strategy out of the classroom?
- e.g., How can I impress my date tonight?

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

Next Class: More logic

&

Uncertainty

Note: No homework this week, HW #2 will be assigned next week