Game playing

Chapter 6

Outline

◇ Games

◇ Perfect play
  – minimax decisions
  – α–β pruning

◇ Resource limits and approximate evaluation

◇ Games of chance

◇ Games of imperfect information

Games vs. search problems

"Unpredictable" opponent $\Rightarrow$ solution is a strategy
specifying a move for every possible opponent reply

Time limits $\Rightarrow$ unlikely to find goal, must approximate

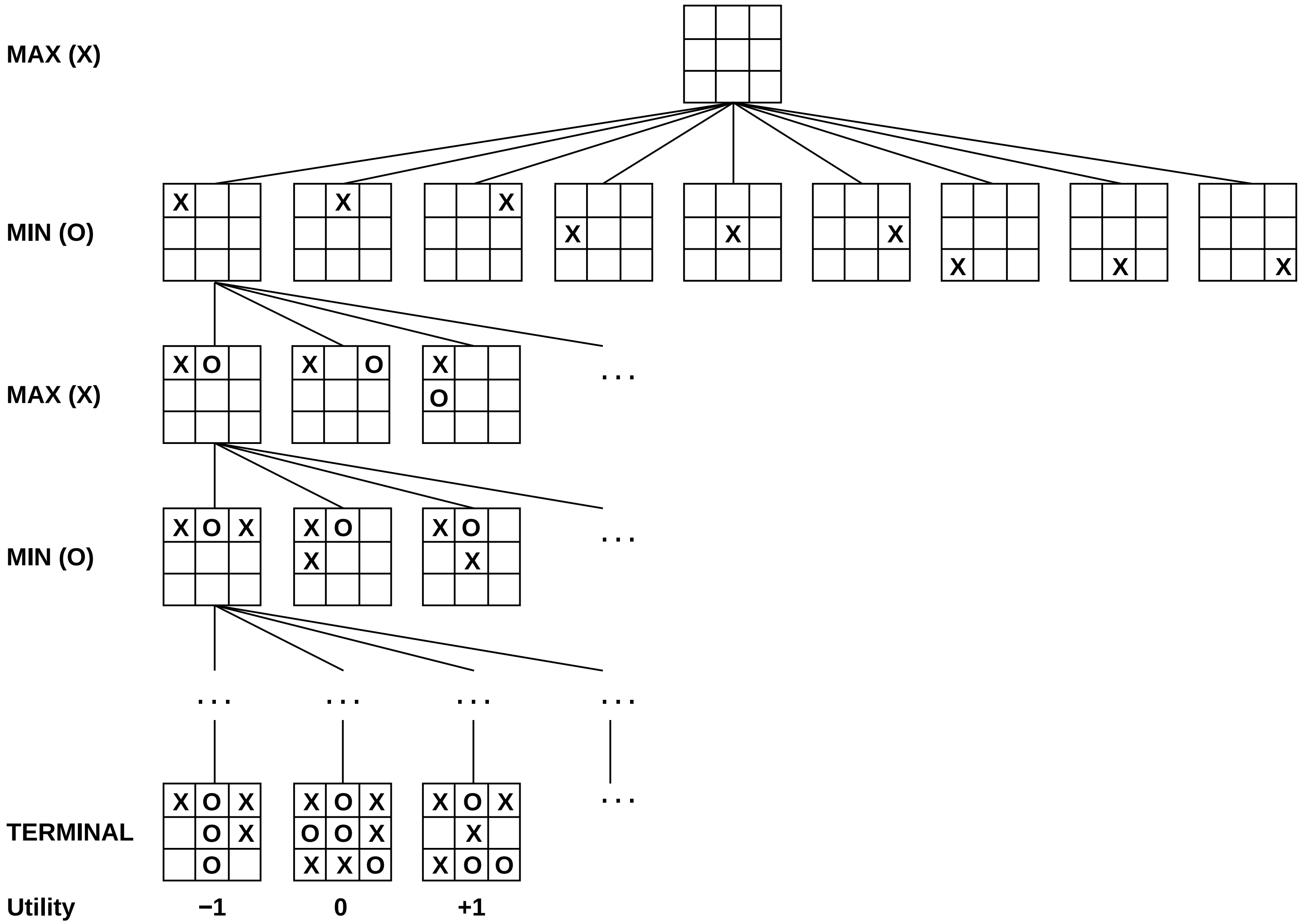
Plan of attack:

- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952–57)
- Pruning to allow deeper search (McCarthy, 1956)

Types of games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

Game tree (2-player, deterministic, turns)

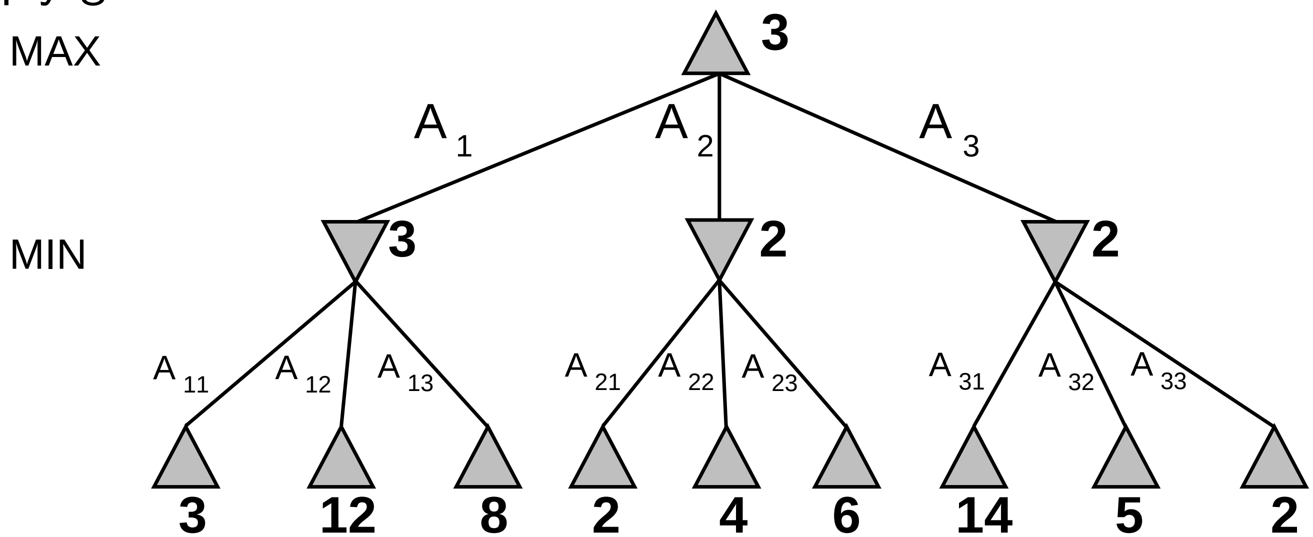


Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value
= best achievable payoff against best play

E.g., 2-ply game:



Minimax algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*
  inputs: *state*, current state in game

  return the a in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(a, *state*))

---

function MAX-VALUE(*state*) **returns** *a utility value*
  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))
  return v

---

function MIN-VALUE(*state*) **returns** *a utility value*
  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))
  return v

Properties of minimax

Complete??

Properties of minimax

Complete?? Only if tree is finite (chess has specific rules for this).
NB a finite strategy can exist even in an infinite tree!

Optimal??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity?? $O(b^m)$

Space complexity??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

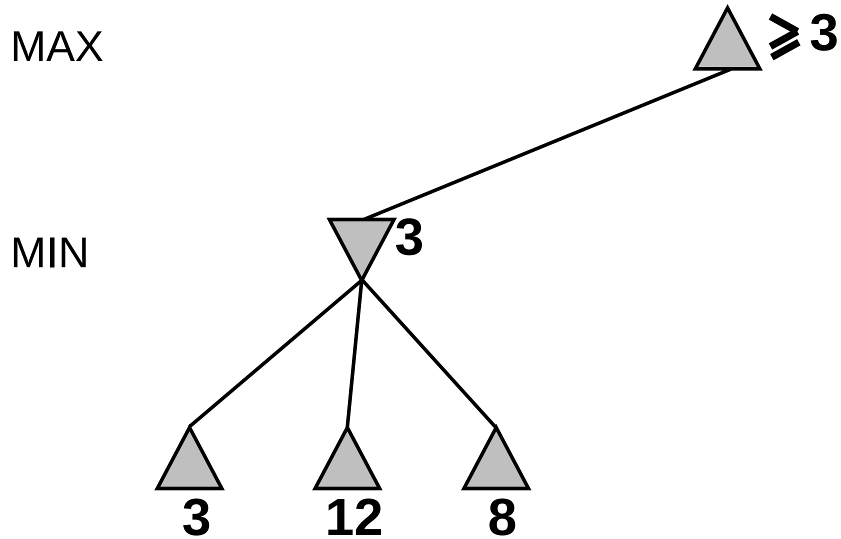
Time complexity?? $O(b^m)$

Space complexity?? $O(bm)$ (depth-first exploration)

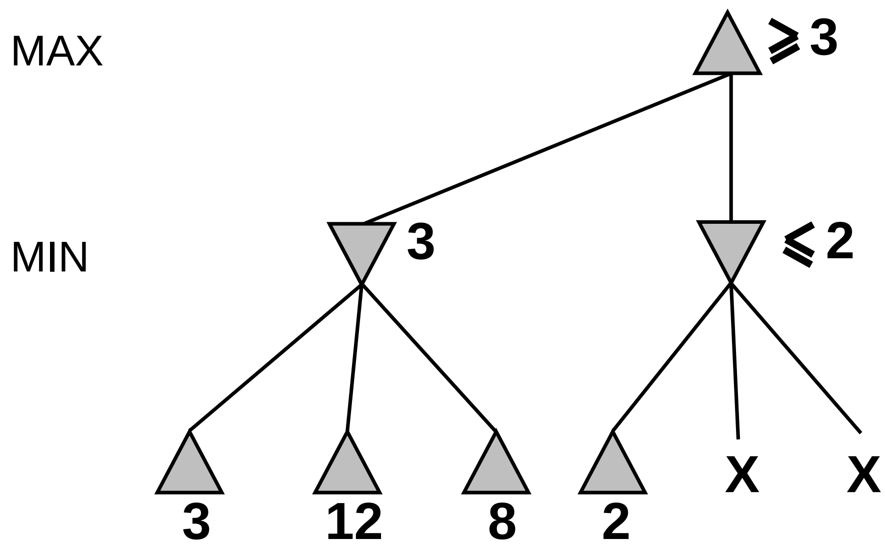
For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
$\Rightarrow$ exact solution completely infeasible

But do we need to explore every path?

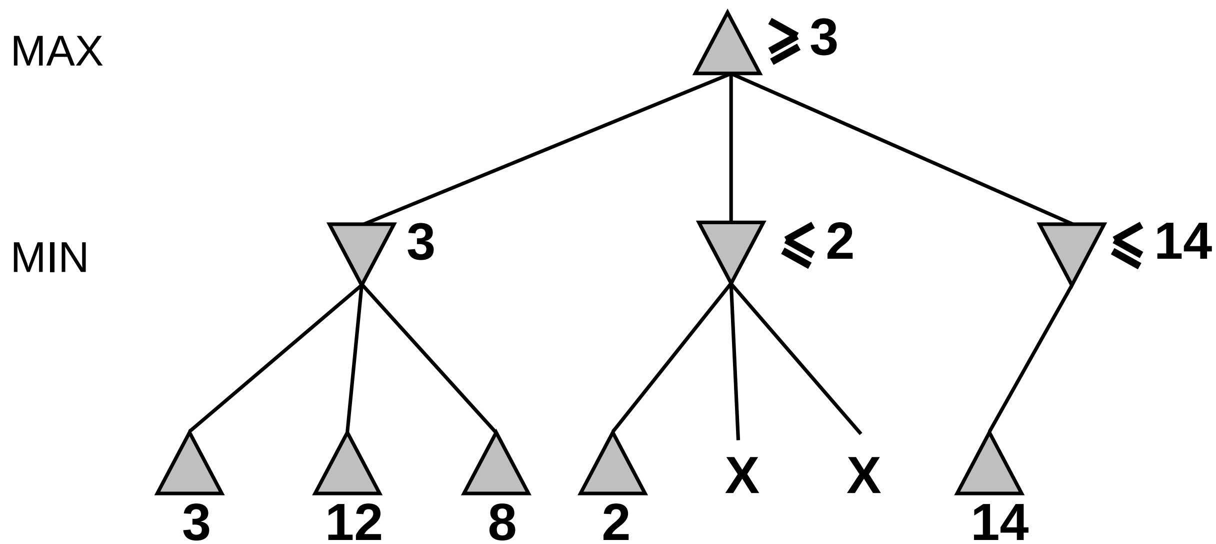
α–β pruning example



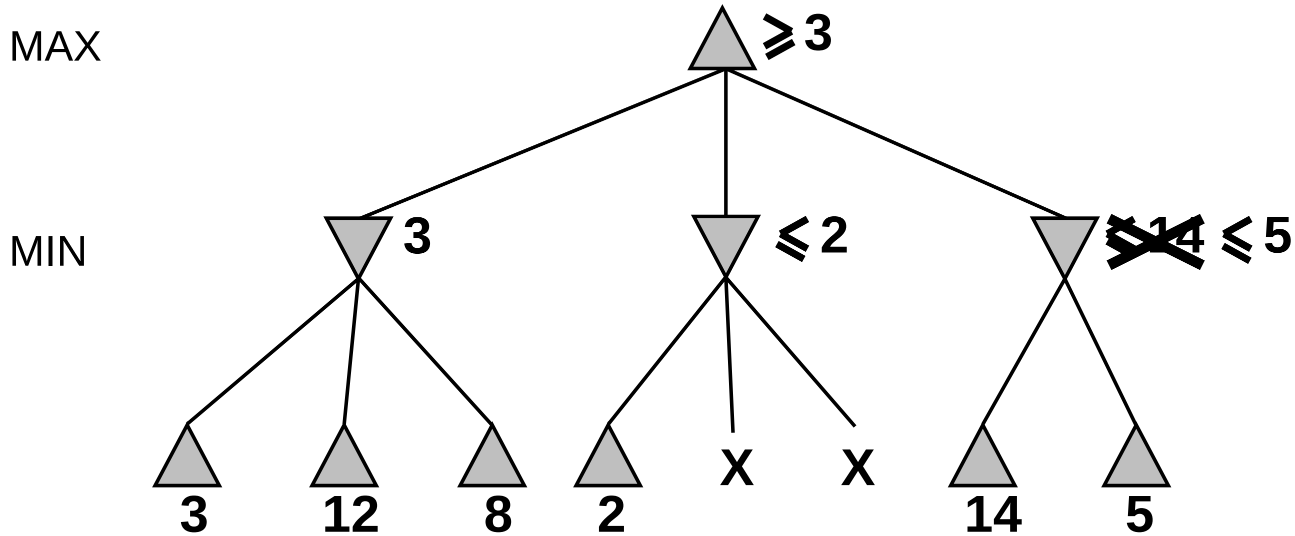
α–β pruning example



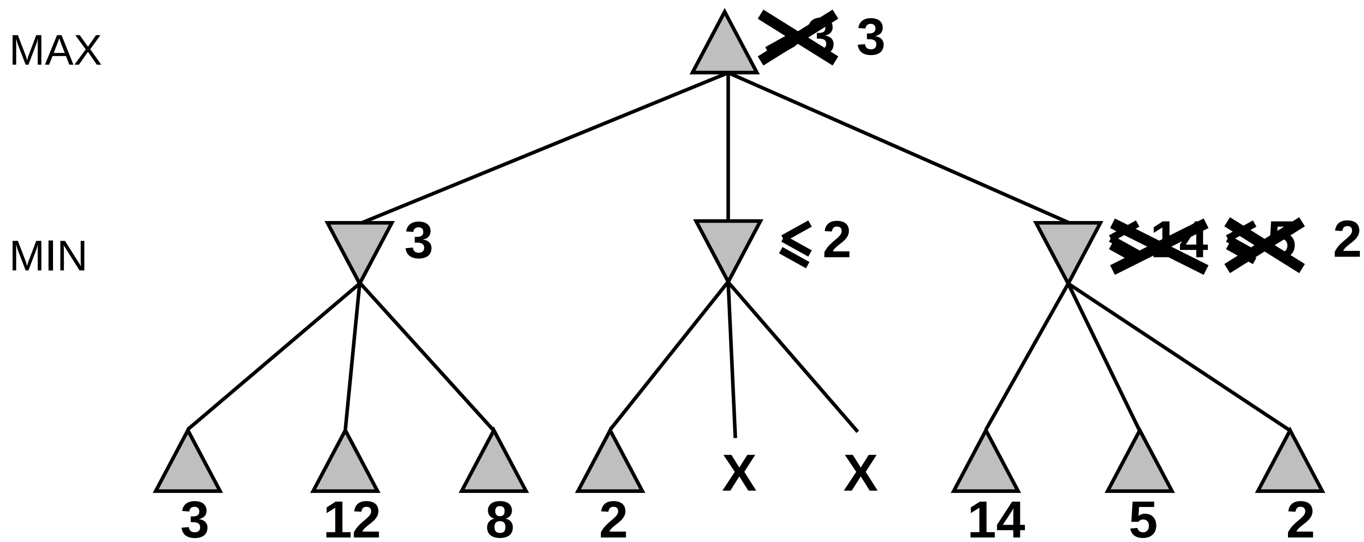
α–β pruning example



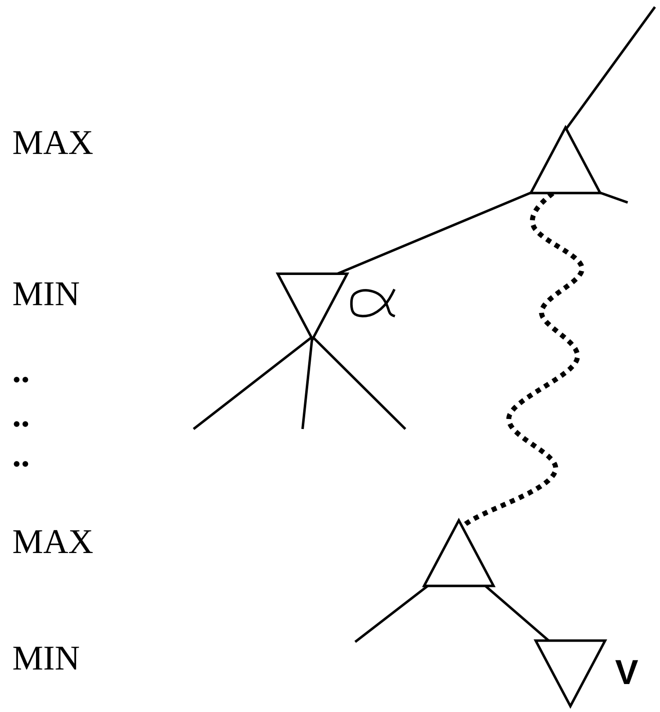
α–β pruning example



α–β pruning example



Why is it called α–β?



α is the best value (to MAX) found so far off the current path

If V is worse than α, MAX will avoid it $\Rightarrow$ prune that branch

Define β similarly for MIN

The α–β algorithm

function ALPHA-BETA-DECISION(*state*) **returns** an action
  return the a in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(a, *state*))

---

function MAX-VALUE(*state*, α, β) **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))
    if $v\ \geq\ \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  same as MAX-VALUE but with roles of α, β reversed

Properties of α–β

Pruning **does not** affect final result

Good move ordering improves effectiveness of pruning

With "perfect ordering," time complexity $= O(b^{m/2})$
$\Rightarrow$ **doubles** solvable depth

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

Unfortunately, 35^{50} is still impossible!

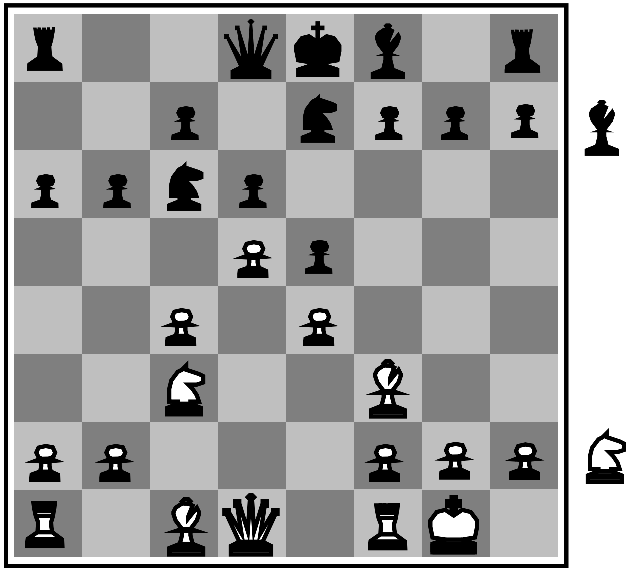
Resource limits

Standard approach:

- Use CUTOFF-TEST instead of TERMINAL-TEST
  e.g., depth limit (perhaps add quiescence search)
- Use EVAL instead of UTILITY
  i.e., evaluation function that estimates desirability of position

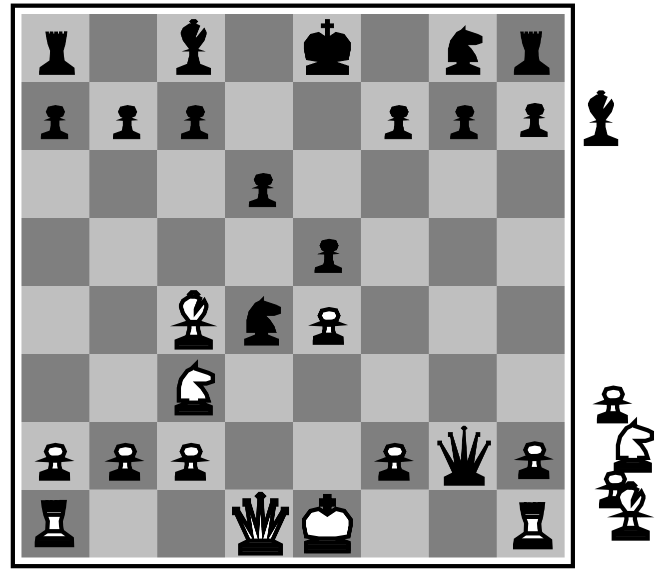
Suppose we have 100 seconds, explore 10^4 nodes/second
$\Rightarrow$ 10^6 nodes per move $\approx 35^{8/2}$
$\Rightarrow$ α–β reaches depth 8 $\Rightarrow$ pretty good chess program

Evaluation functions



Black to move

White slightly better



White to move

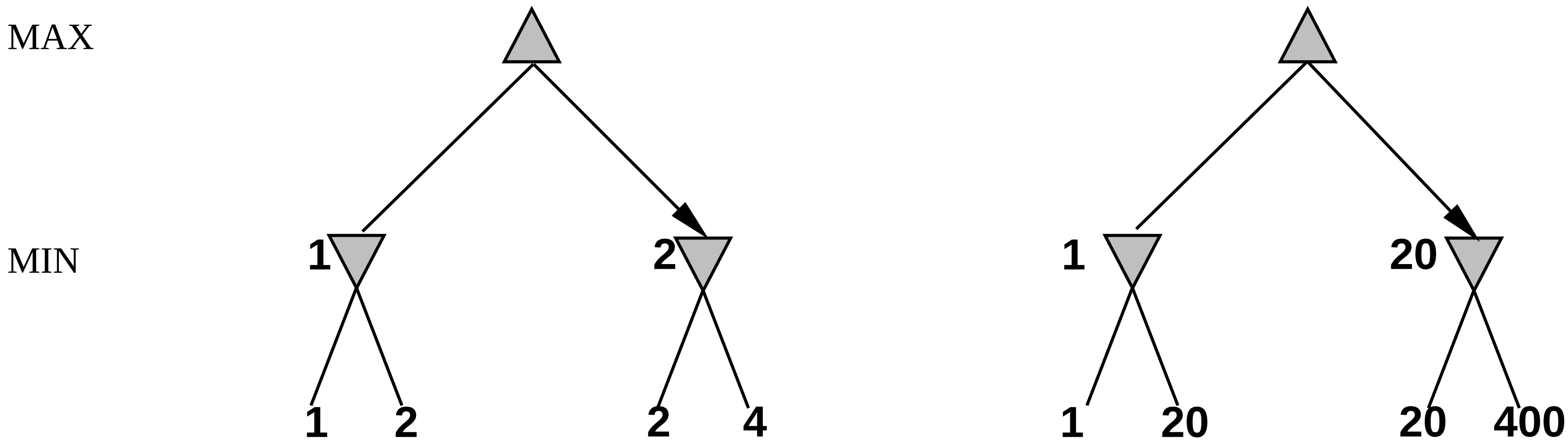
Black winning

For chess, typically linear weighted sum of features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g., $w_1 = 9$ with
$f_1(s) =$ (number of white queens) – (number of black queens), etc.

Digression: Exact values don't matter



Behaviour is preserved under any **monotonic** transformation of EVAL

Only the order matters:
payoff in deterministic games acts as an ordinal utility function

Deterministic games in practice

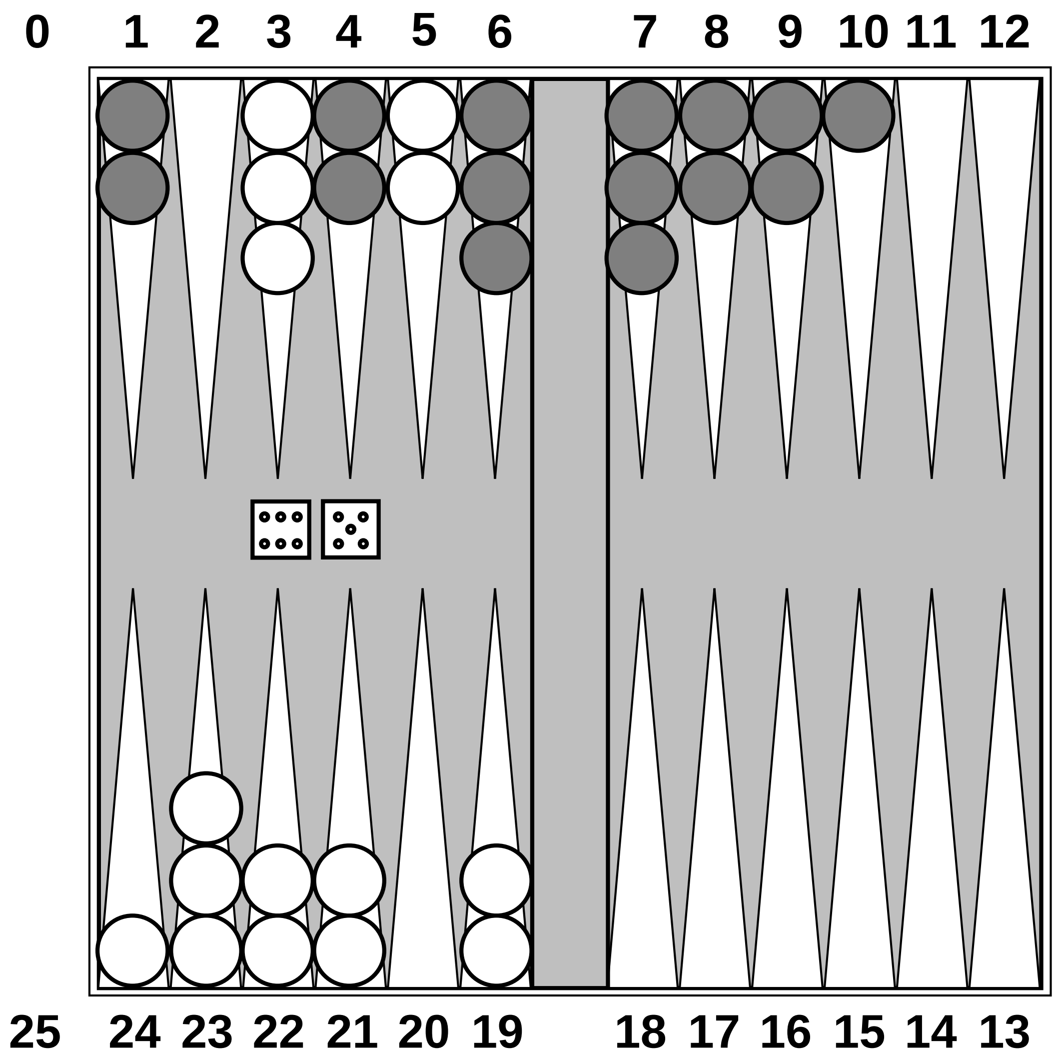
Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

Othello: human champions refuse to compete against computers, who are too good.

Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

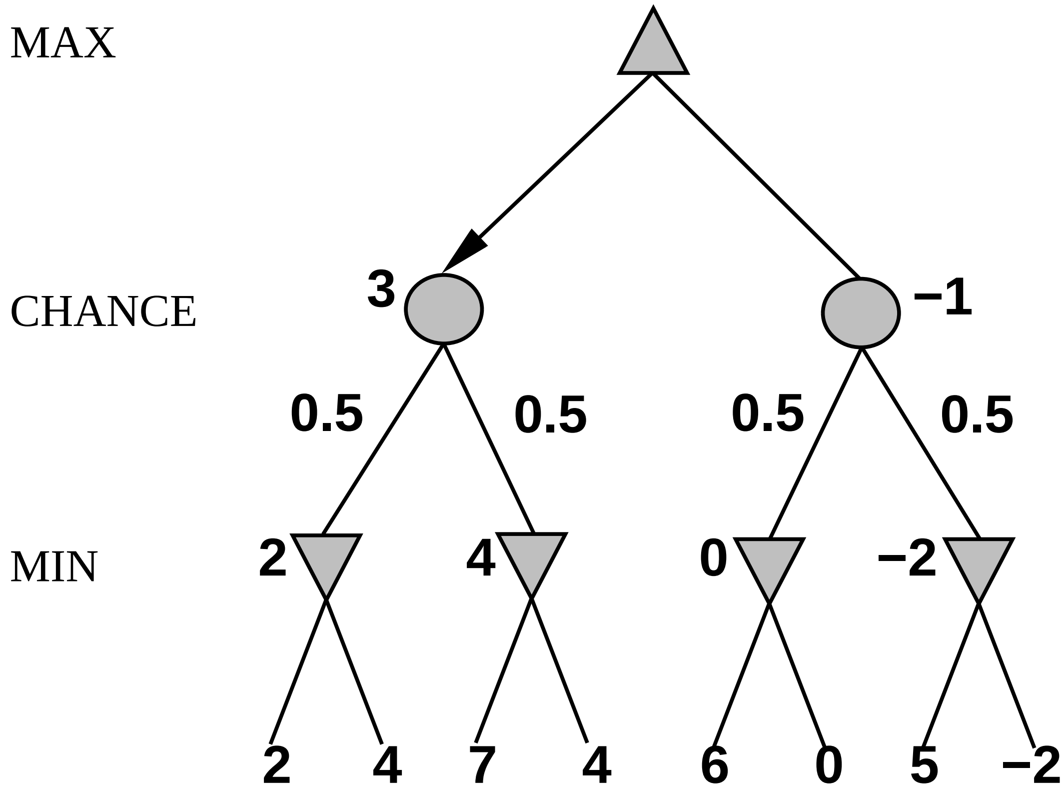
Nondeterministic games: backgammon



Nondeterministic games in general

In nondeterministic games, chance introduced by dice, card-shuffling

Simplified example with coin-flipping:



Algorithm for nondeterministic games

EXPECTIMINIMAX gives perfect play

Just like MINIMAX, except we must also handle chance nodes:

```
. . .
if state is a MAX node then
        return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a MIN node then
        return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a chance node then
        return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
. . .
```

Nondeterministic games in practice

Dice rolls increase b: 21 possible rolls with 2 dice
Backgammon $\approx$ 20 legal moves (can be 6,000 with 1-1 roll)

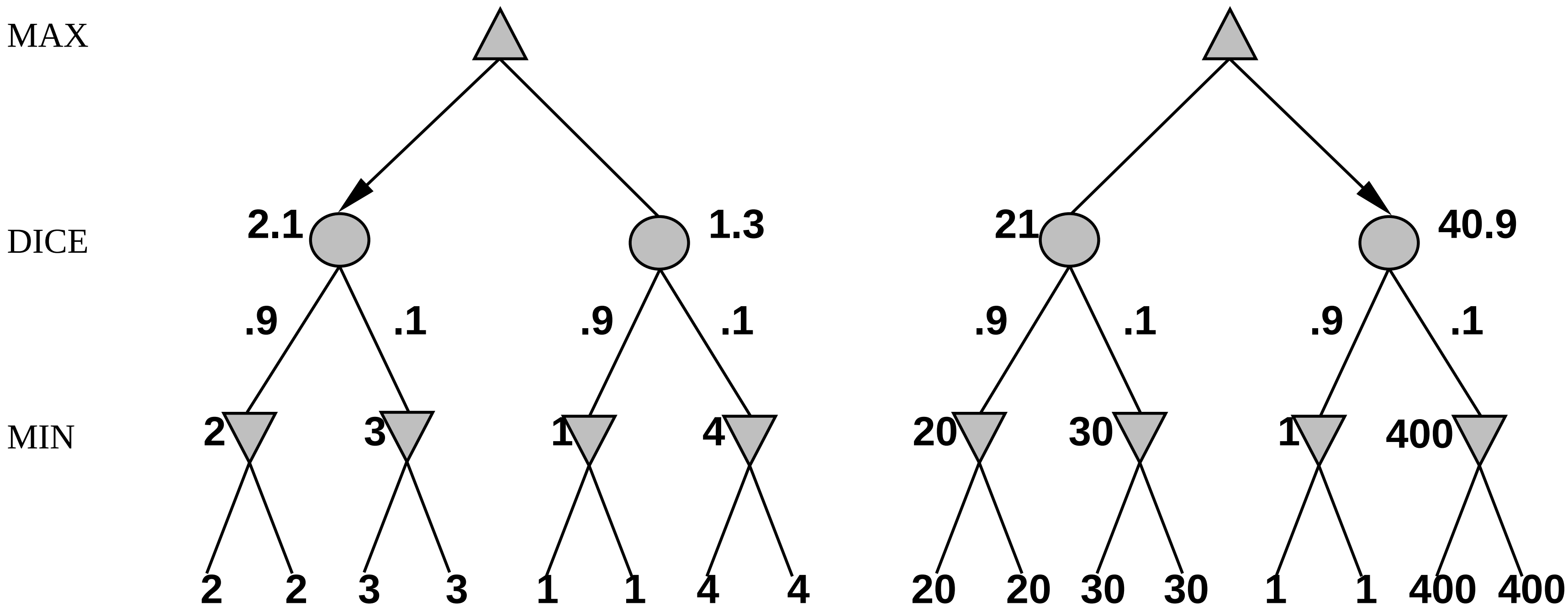
$$\text{depth } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$$

As depth increases, probability of reaching a given node shrinks
$\Rightarrow$ value of lookahead is diminished

α–β pruning is much less effective

TDGAMMON uses depth-2 search + very good EVAL
$\approx$ world-champion level

Digression: Exact values DO matter



Behaviour is preserved only by positive linear transformation of EVAL

Hence EVAL should be proportional to the expected payoff

Games of imperfect information

E.g., card games, where opponent's initial cards are unknown

Typically we can calculate a probability for each possible deal

Seems just like having one big dice roll at the beginning of the game*

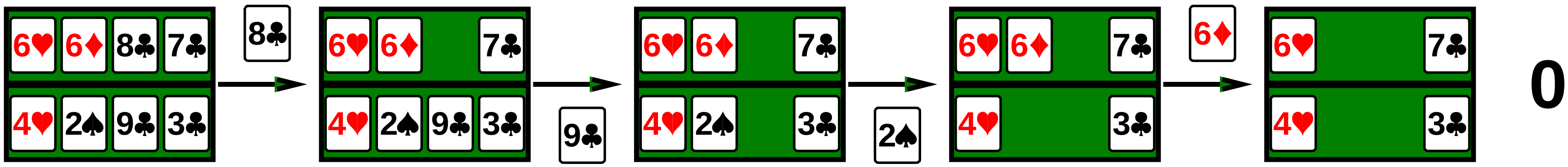
Idea: compute the minimax value of each action in each deal,
then choose the action with highest expected value over all deals*

Special case: if an action is optimal for all deals, it's optimal.*

GIB, current best bridge program, approximates this idea by
1) generating 100 deals consistent with bidding information
2) picking the action that wins most tricks on average

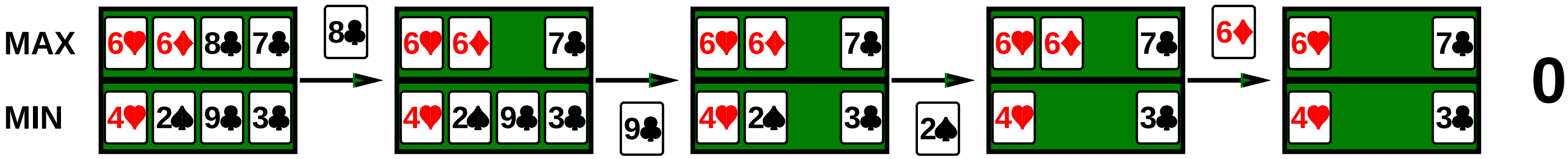
Example

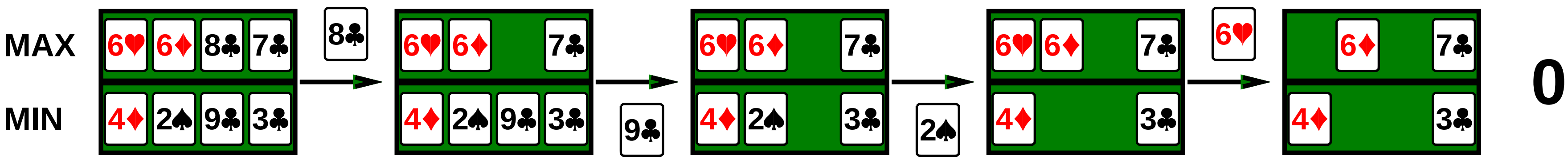
Four-card bridge/whist/hearts hand, MAX to play first



Example

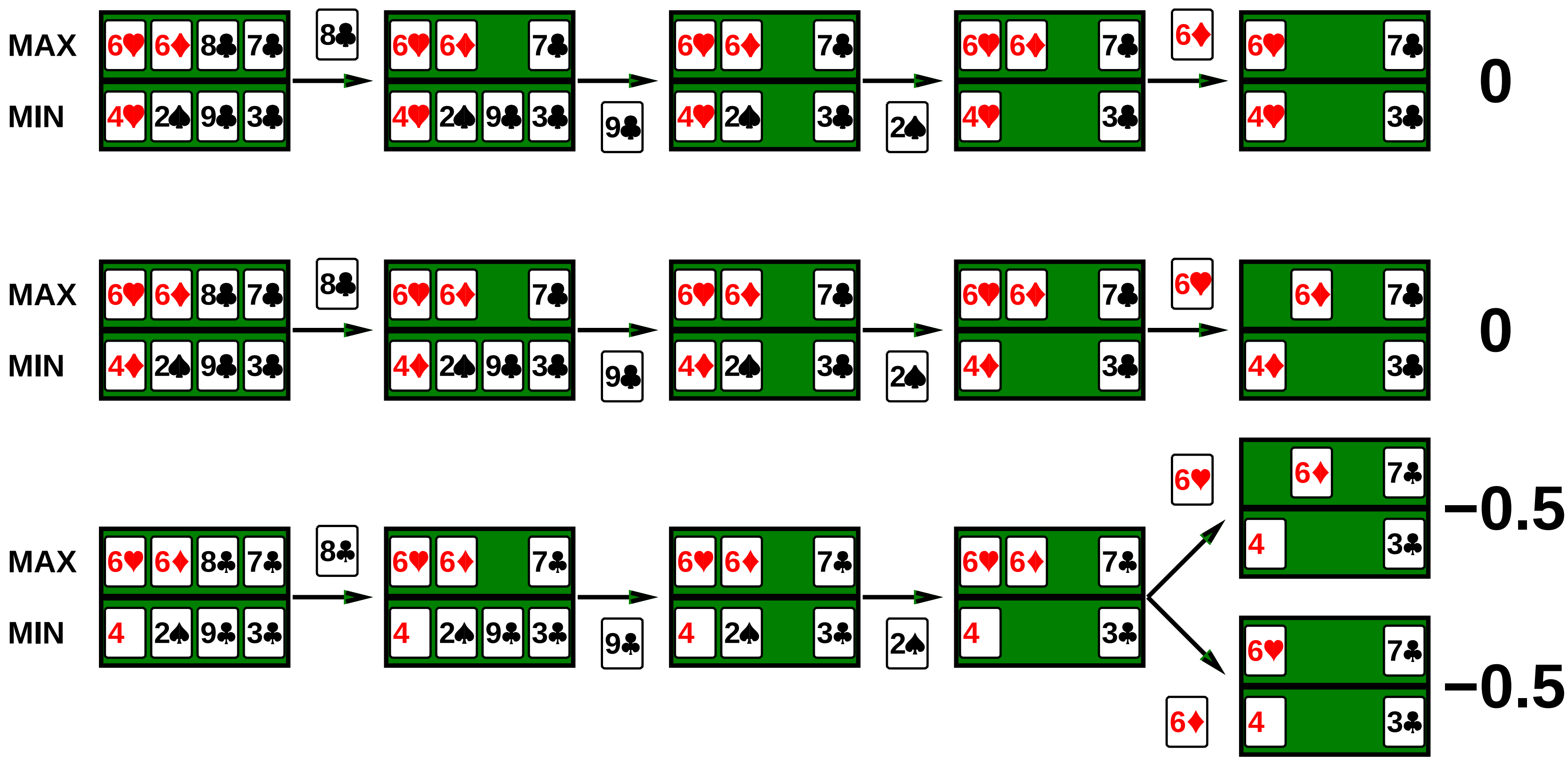
Four-card bridge/whist/hearts hand, MAX to play first





Example

Four-card bridge/whist/hearts hand, MAX to play first



Commonsense example

Road A leads to a small heap of gold pieces
Road B leads to a fork:

take the left fork and you'll find a mound of jewels;
take the right fork and you'll be run over by a bus.

Commonsense example

Road A leads to a small heap of gold pieces
Road B leads to a fork:
    take the left fork and you'll find a mound of jewels;
    take the right fork and you'll be run over by a bus.

Road A leads to a small heap of gold pieces
Road B leads to a fork:
    take the left fork and you'll be run over by a bus;
    take the right fork and you'll find a mound of jewels.

Commonsense example

Road A leads to a small heap of gold pieces
Road B leads to a fork:
  take the left fork and you'll find a mound of jewels;
  take the right fork and you'll be run over by a bus.

Road A leads to a small heap of gold pieces
Road B leads to a fork:
  take the left fork and you'll be run over by a bus;
  take the right fork and you'll find a mound of jewels.

Road A leads to a small heap of gold pieces
Road B leads to a fork:
  guess correctly and you'll find a mound of jewels;
  guess incorrectly and you'll be run over by a bus.

Proper analysis

* Intuition that the value of an action is the average of its values in all actual states is **WRONG**

With partial observability, value of an action depends on the information state or belief state the agent is in

Can generate and search a tree of information states

Leads to rational behaviors such as
- ♢ Acting to obtain information
- ♢ Signalling to one's partner
- ♢ Acting randomly to minimize information disclosure

Summary

Games are fun to work on! (and dangerous)

They illustrate several important points about AI

- ♢ perfection is unattainable $\Rightarrow$ must approximate
- ♢ good idea to think about what to think about
- ♢ uncertainty constrains the assignment of values to states
- ♢ optimal decisions depend on information state, not real state

Games are to AI as grand prix racing is to automobile design

Logical agents

Chapter 7, Sections 1–5

Outline

◊ Knowledge-based agents

◊ Example: The wumpus world

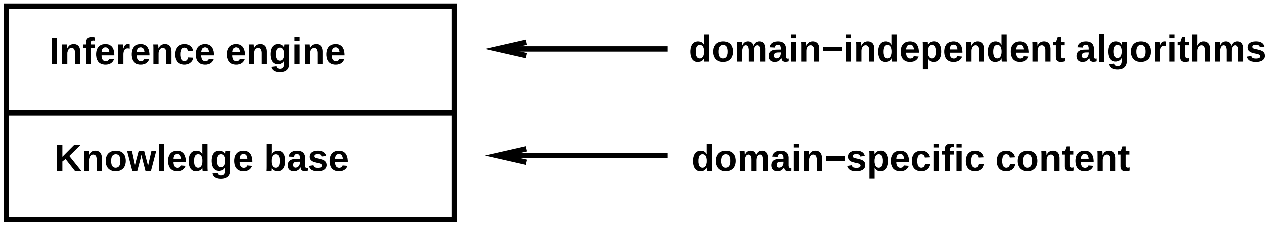
◊ Logic in general—models and entailment

◊ Propositional (Boolean) logic

◊ Equivalence, validity, satisfiability

◊ Inference rules and theorem proving
  - forward chaining
  - backward chaining
  - resolution

Knowledge bases



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

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

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

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

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

Wumpus world: PEAS description

Performance measure

gold $+1000$, death -1000

-1 per step, -10 for using the arrow

Environment

Squares adjacent to wumpus are smelly

Squares adjacent to pit are breezy

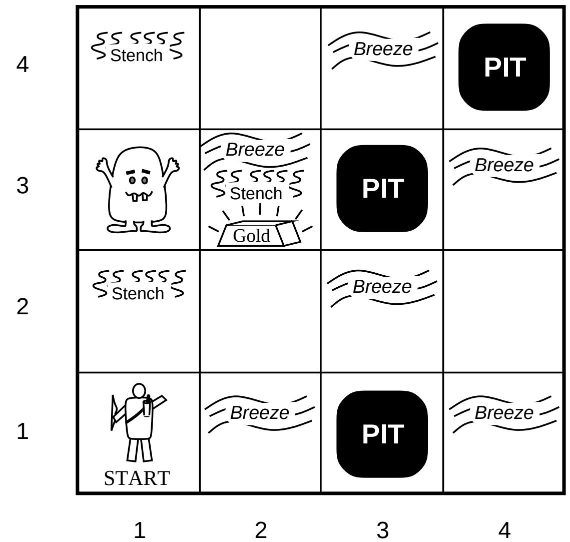
Glitter iff gold is in the same square

Shooting kills wumpus if you are facing it

Shooting uses up the only arrow

Grabbing picks up gold if in same square

Releasing drops the gold in same square



Actuators

Left turn, Right turn, Forward, Grab, Release, Shoot

Sensors

Breeze, Glitter, Smell

Wumpus world characterization

Observable?? No—only local perception

Deterministic?? Yes—outcomes exactly specified

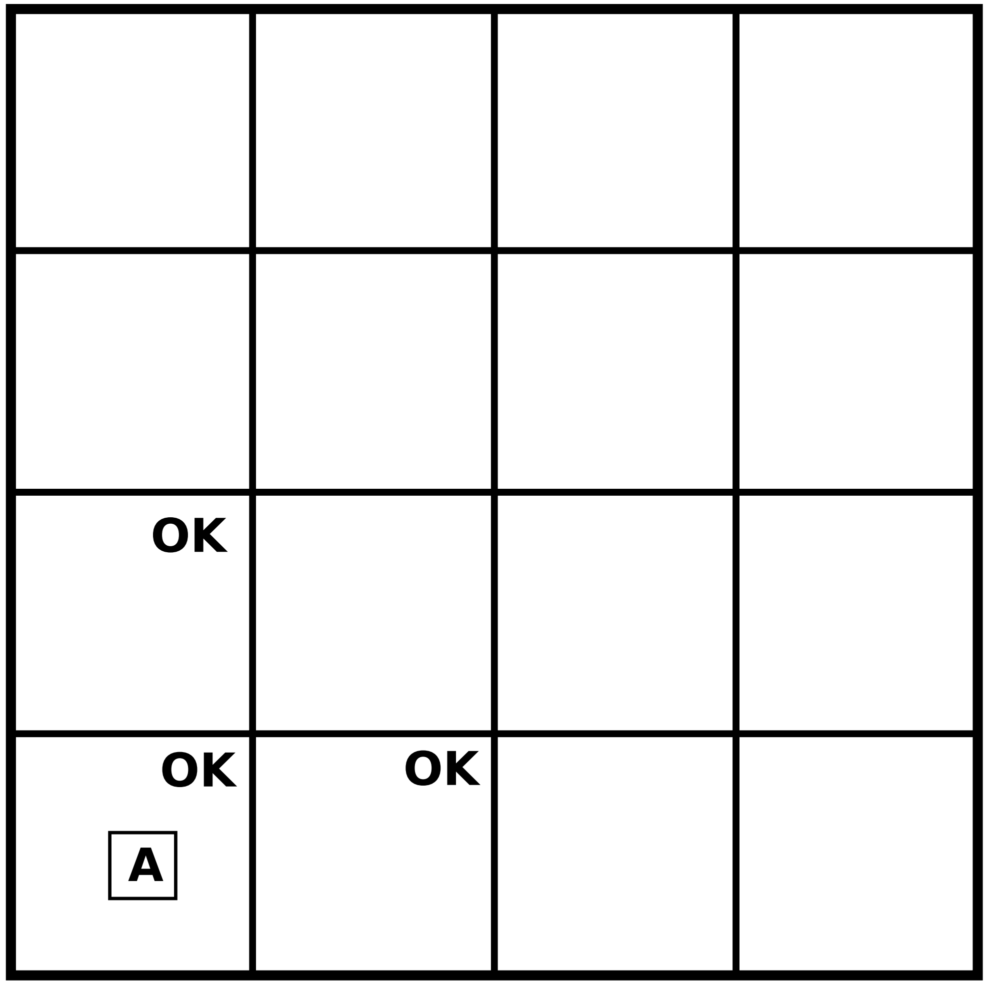
Episodic?? No—sequential at the level of actions

Static?? Yes—Wumpus and Pits do not move

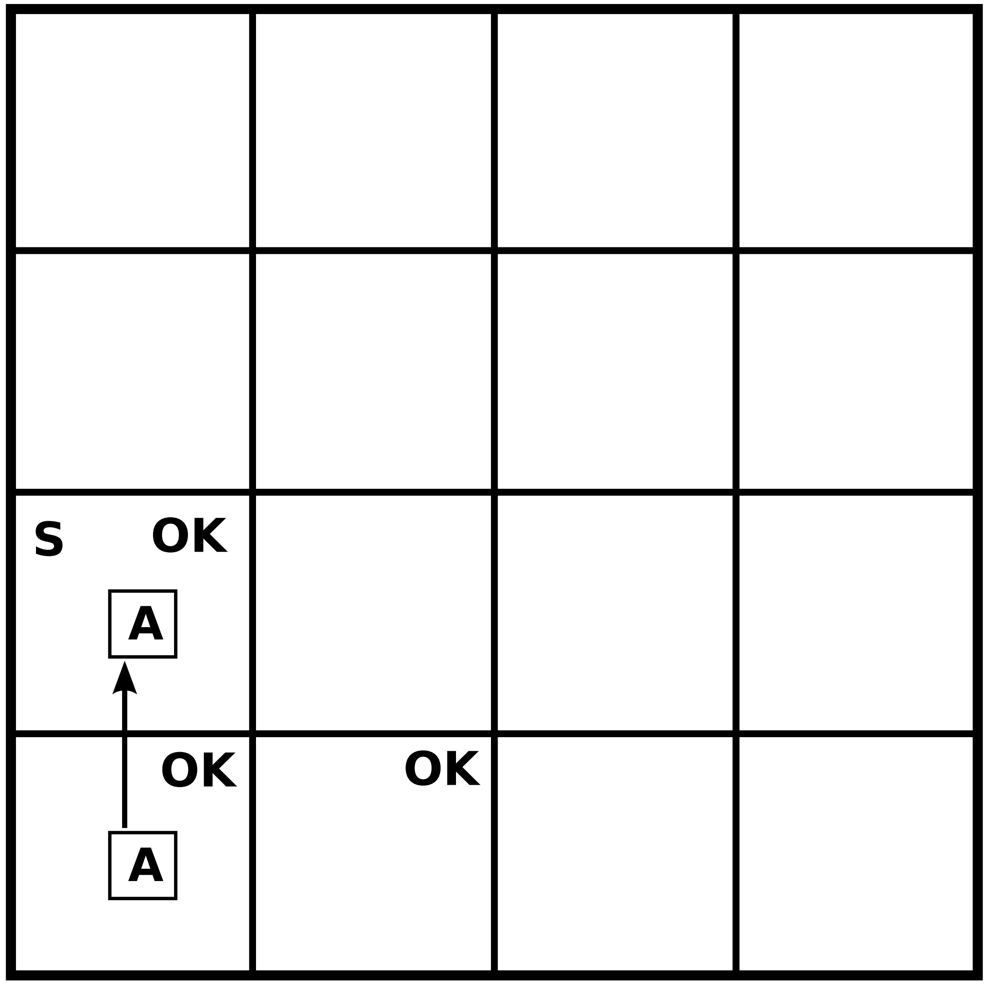
Discrete?? Yes

Single-agent?? Yes—Wumpus is essentially a natural feature

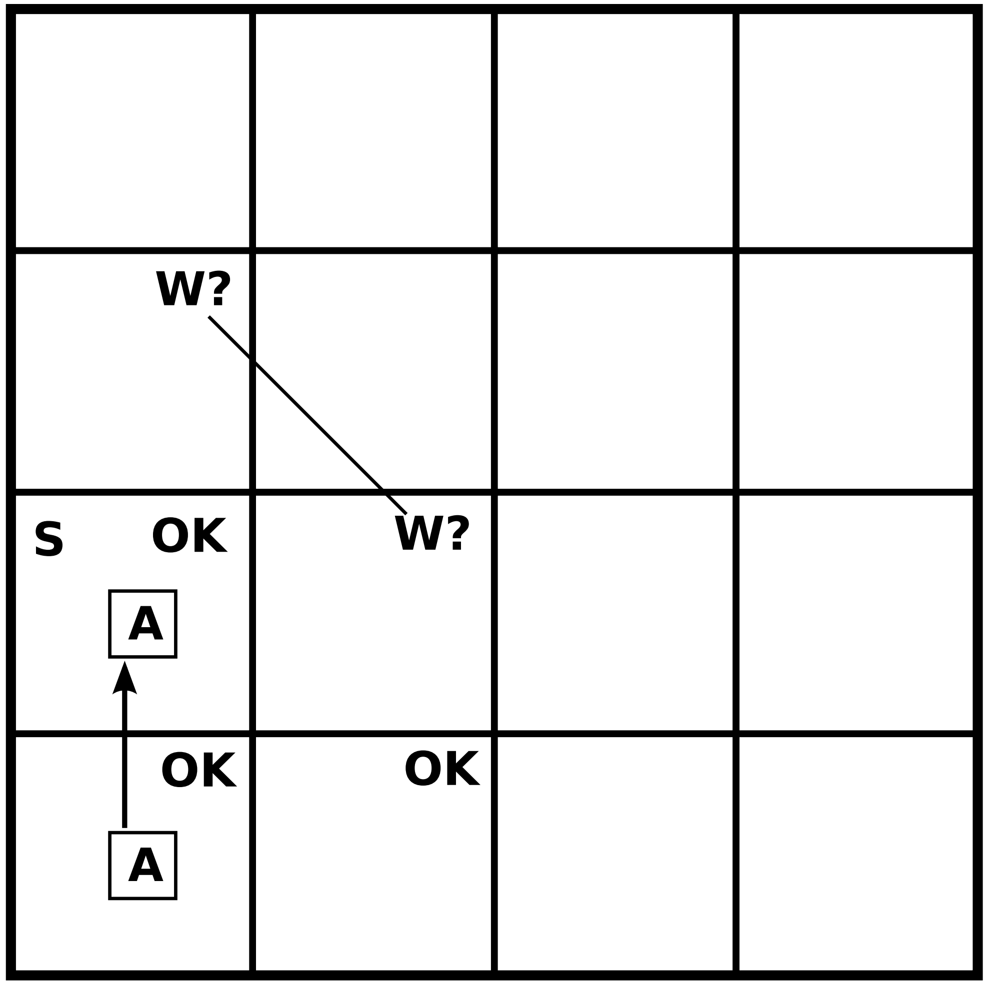
Exploring a wumpus world



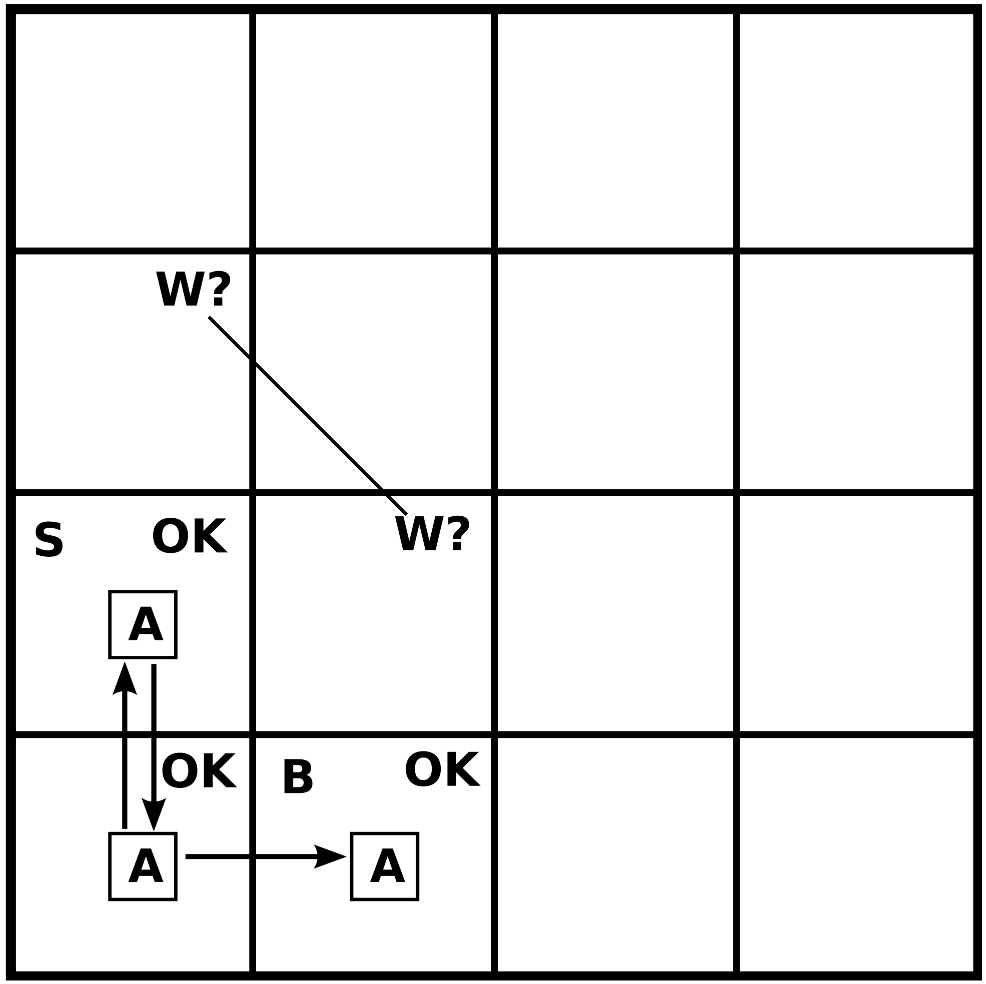
Exploring a wumpus world



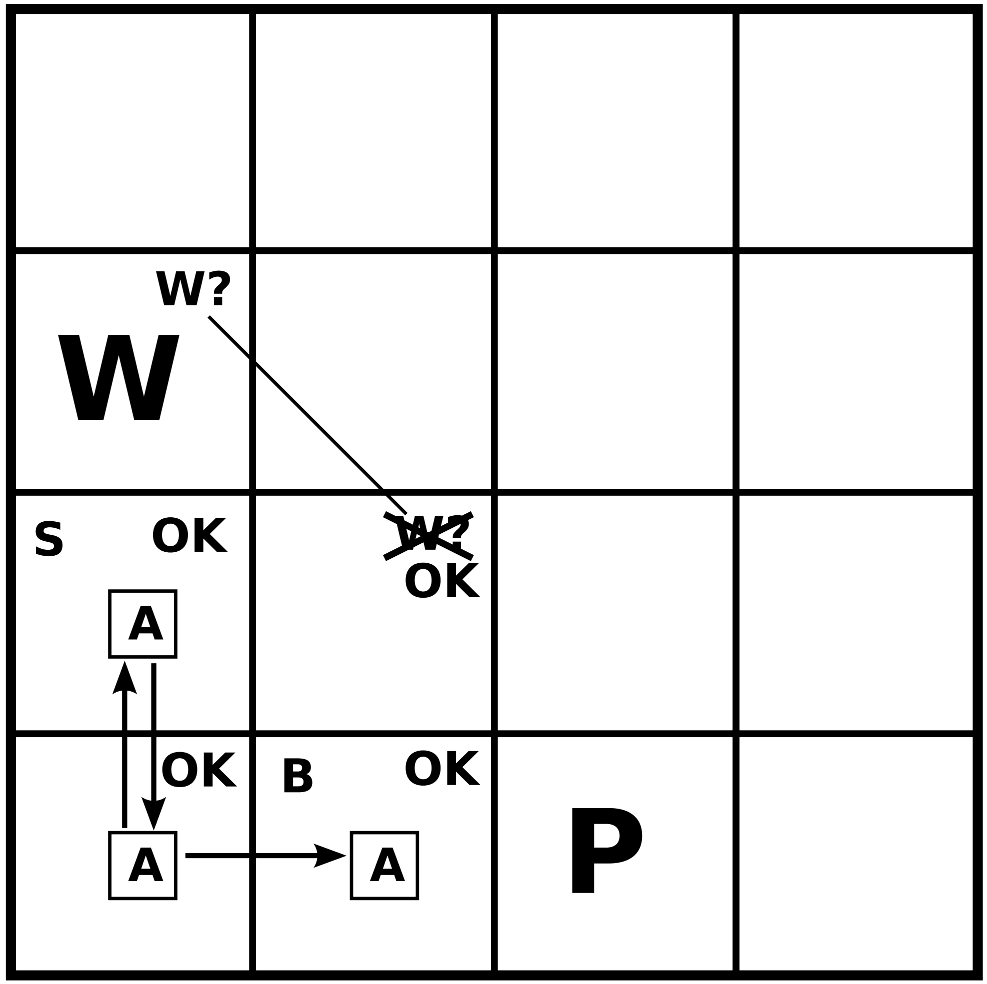
Exploring a wumpus world



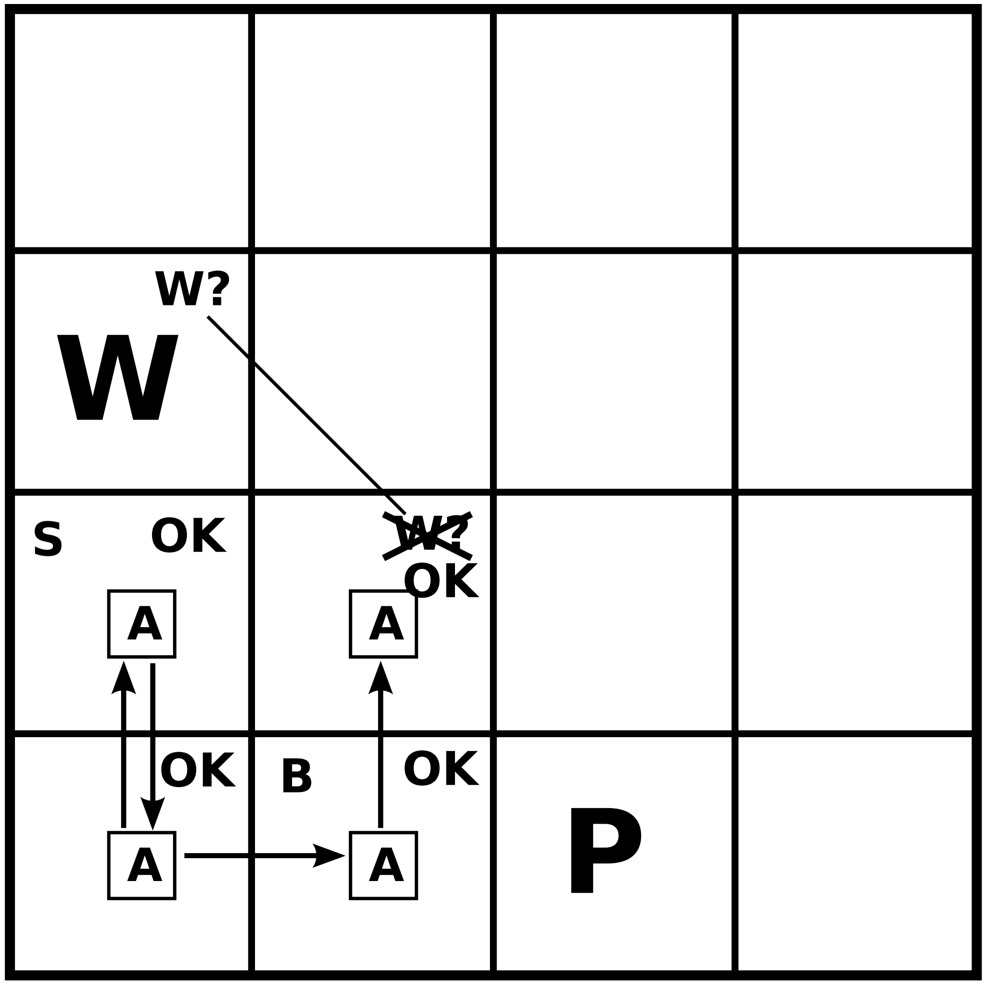
Exploring a wumpus world



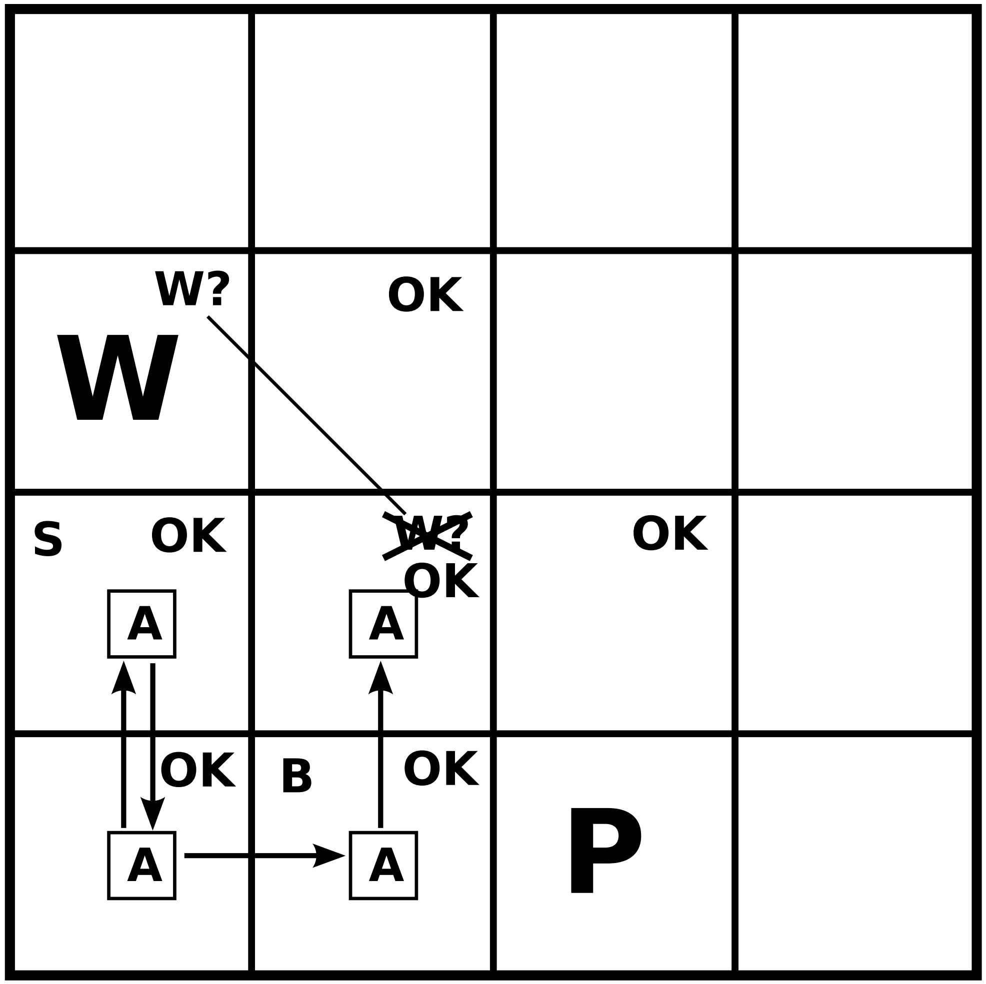
Exploring a wumpus world



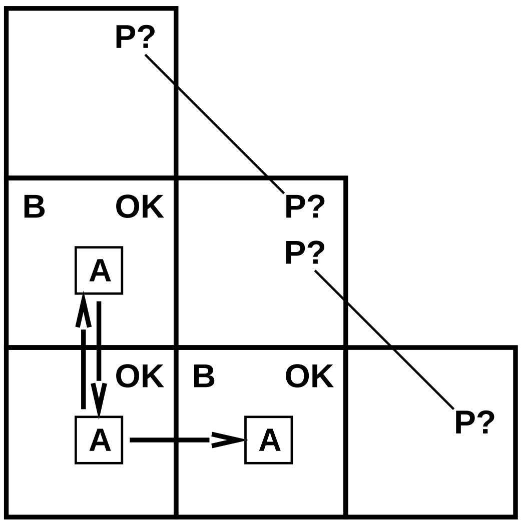
Exploring a wumpus world



Exploring a wumpus world



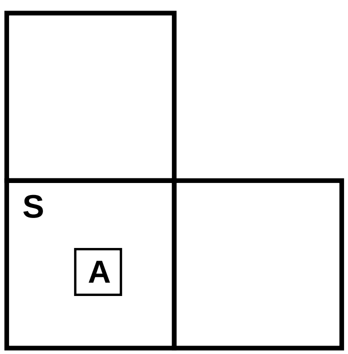
Other tight spots



Breeze in (1,2) and (2,1)

$\Rightarrow$ no safe actions

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



Smell in (1,1)

$\Rightarrow$ cannot move

Can use a strategy of coercion:

shoot straight ahead

wumpus was there $\Rightarrow$ dead $\Rightarrow$ safe

wumpus wasn't there $\Rightarrow$ safe

Logic in general

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

Syntax defines the sentences in the language

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

E.g., the language of arithmetic

$x + 2 \geq y$ is a sentence; $x2 + y >$ is not a sentence

$x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y

$x + 2 \geq y$ is true in a world where $x = 7,\ y = 1$
$x + 2 \geq y$ is false in a world where $x = 0,\ y = 6$

Entailment

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

$$KB \models \alpha$$

A knowledge base KB entails a sentence α
if and only if
α is true in all worlds where KB is true

E.g., the KB containing "There's a pit ahead" and "There's gold to the left" entails "Either there's a pit ahead or gold to the left"

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

Entailment is a relationship between sentences (i.e., **syntax**) that is based on **semantics**

Models

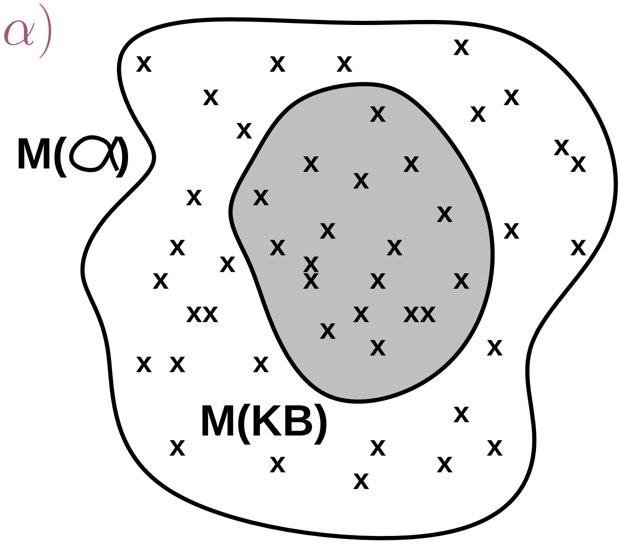
Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated

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

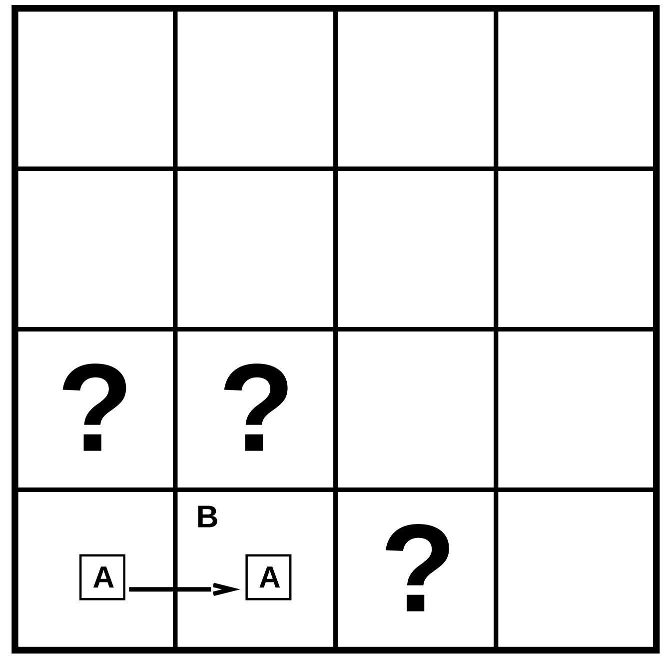
$M(\alpha)$ is the set of all models of α

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

E.g. $KB =$ { there's a pit ahead,
there's gold to the left }
$\alpha =$ there's gold to the left



Entailment in the wumpus world

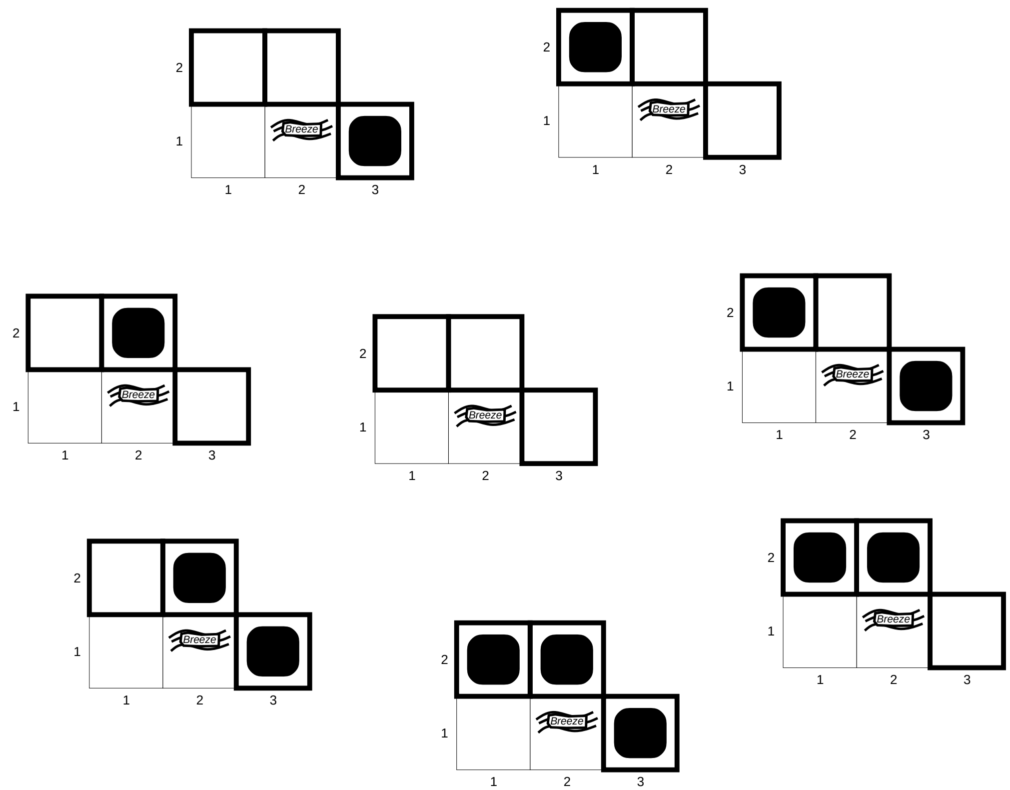


Situation after detecting nothing in [1,1], moving right, breeze in [2,1]

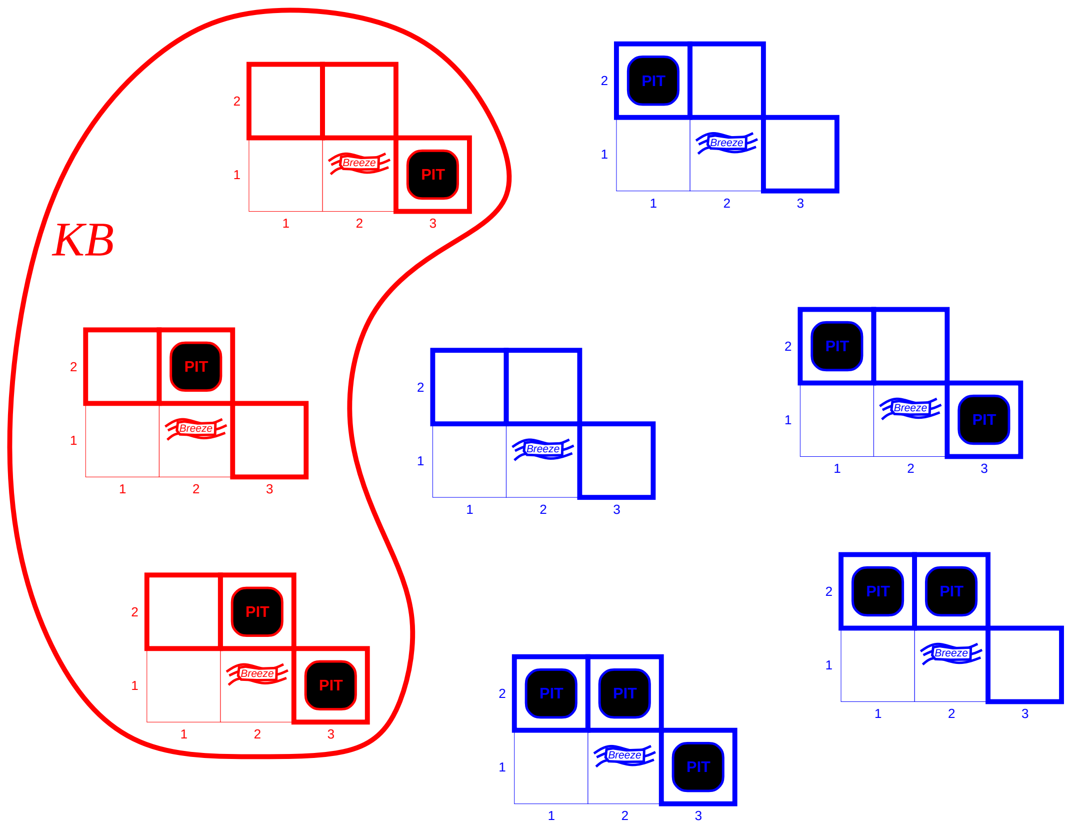
Consider possible models for ?s assuming only pits

3 Boolean choices $\Rightarrow$ 8 possible models

Wumpus models

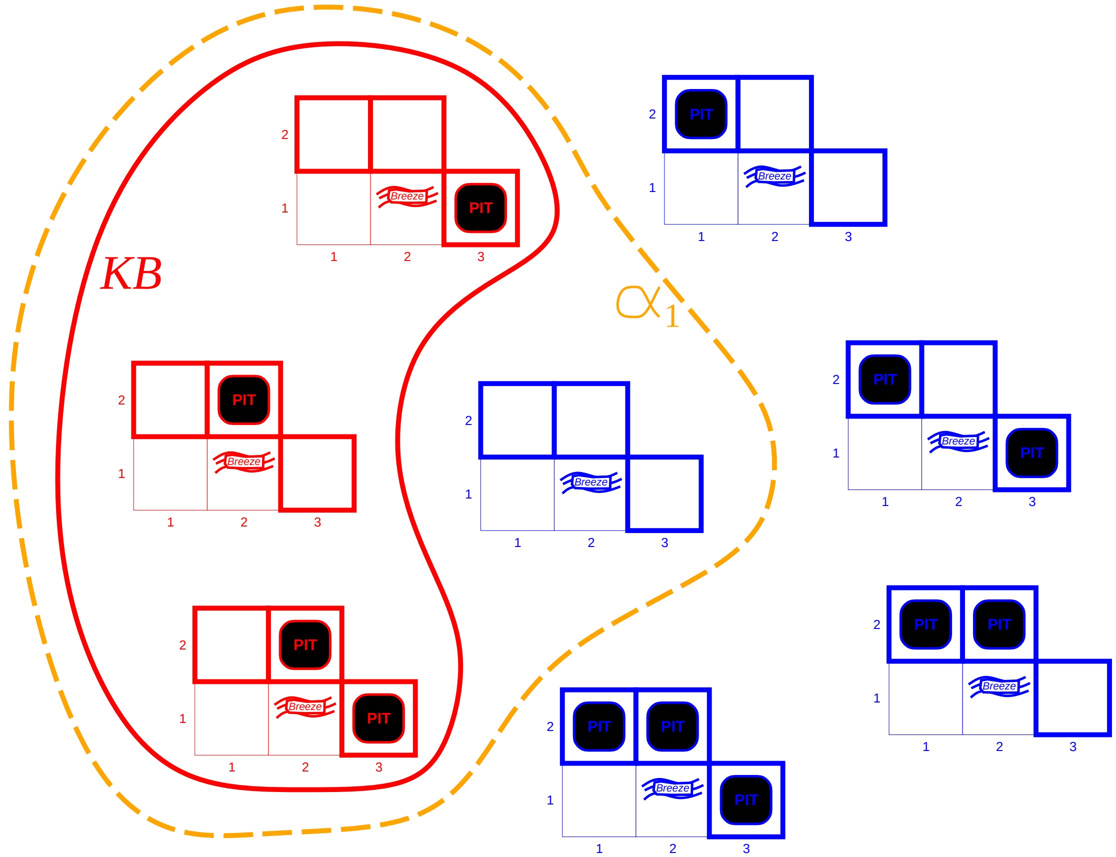


Wumpus models



KB = wumpus-world rules + observations

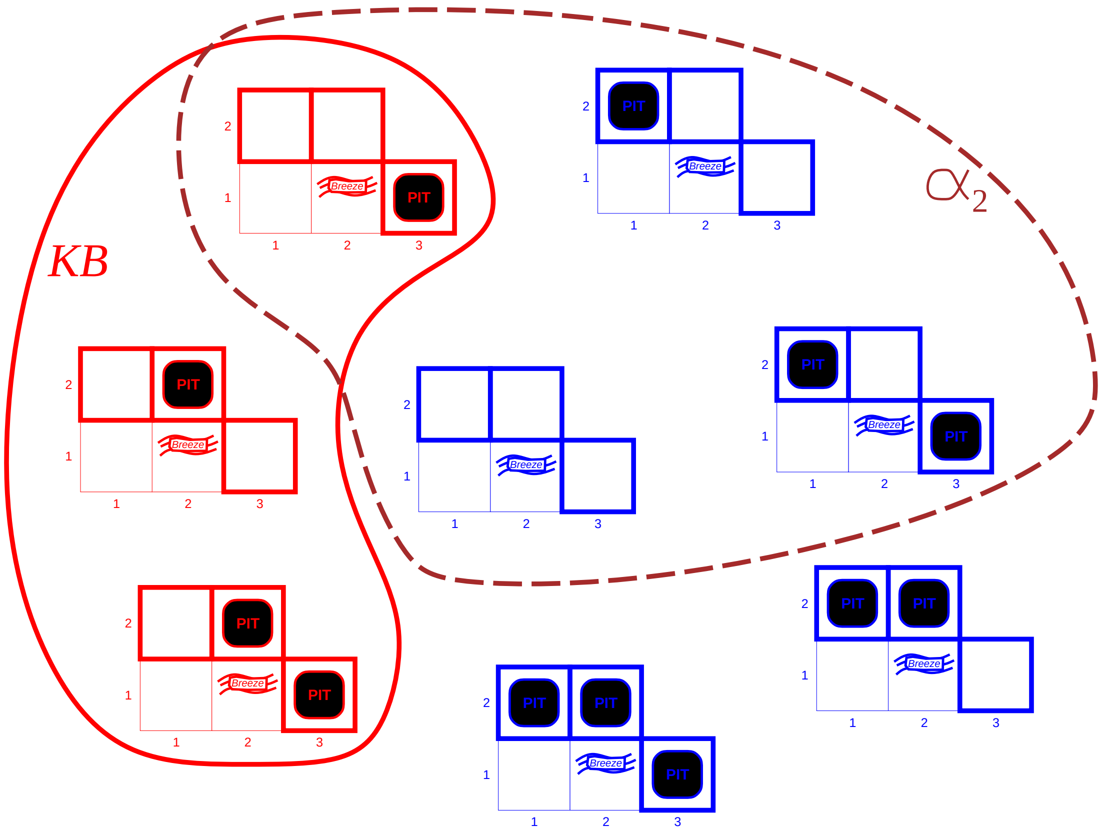
Wumpus models



KB = wumpus-world rules + observations

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

Wumpus models



KB = wumpus-world rules + observations

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

Inference

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

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

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

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

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

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

Propositional logic: Syntax

Propositional logic is the simplest logic—illustrates basic ideas

The proposition symbols P_1, P_2 etc are sentences

If S is a sentence, $\neg S$ is a sentence (negation)

If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)

If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)

If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)

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

Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

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

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

Rules for evaluating truth with respect to a model m:

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

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

Truth tables for connectives

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

Wumpus world sentences

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

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

How do we encode "pits cause breezes in adjacent squares"?

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

In other words, "a square is breezy **if and only if** there is an adjacent pit"

Truth tables for inference

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

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

Inference by enumeration

Depth-first enumeration of all models is sound and complete

```
function TT-ENTAILS?(KB, α) returns true or false
   inputs: KB, the knowledge base, a sentence in propositional logic
           α, the query, a sentence in propositional logic

   symbols ← a list of the proposition symbols in KB and α
   return TT-CHECK-ALL(KB, α, symbols, [ ])
```

```
function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
   if EMPTY?(symbols) then
      if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
      else return true
   else do
      P ← FIRST(symbols); rest ← REST(symbols)
      return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model)) and
                TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))
```

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

Logical equivalence

Two sentences are logically equivalent iff they are true in the same models:

$\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$

$(\alpha \land \beta)$	$\equiv (\beta \land \alpha)$	commutativity of $\land$
$(\alpha \lor \beta)$	$\equiv (\beta \lor \alpha)$	commutativity of $\lor$
$((\alpha \land \beta) \land \gamma)$	$\equiv (\alpha \land (\beta \land \gamma))$	associativity of $\land$
$((\alpha \lor \beta) \lor \gamma)$	$\equiv (\alpha \lor (\beta \lor \gamma))$	associativity of $\lor$
$\neg(\neg\alpha)$	$\equiv \alpha$	double-negation elimination
$(\alpha \Rightarrow \beta)$	$\equiv (\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta)$	$\equiv (\neg\alpha \lor \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta)$	$\equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \land \beta)$	$\equiv (\neg\alpha \lor \neg\beta)$	De Morgan
$\neg(\alpha \lor \beta)$	$\equiv (\neg\alpha \land \neg\beta)$	De Morgan
$(\alpha \land (\beta \lor \gamma))$	$\equiv ((\alpha \land \beta) \lor (\alpha \land \gamma))$	distributivity of $\land$ over $\lor$
$(\alpha \lor (\beta \land \gamma))$	$\equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma))$	distributivity of $\lor$ over $\land$

Validity and satisfiability

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

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:

$KB \models \alpha$ iff $(KB \land \neg\alpha)$ is unsatisfiable

i.e., prove α by *reductio ad absurdum*

Proof methods

Proof methods divide into (roughly) two kinds:

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

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

Forward and backward chaining

Horn Form (restricted)

KB = **conjunction** of **Horn clauses**

Horn clause =

– proposition symbol; or

– (conjunction of symbols) $\Rightarrow$ symbol

Example KB: $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

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

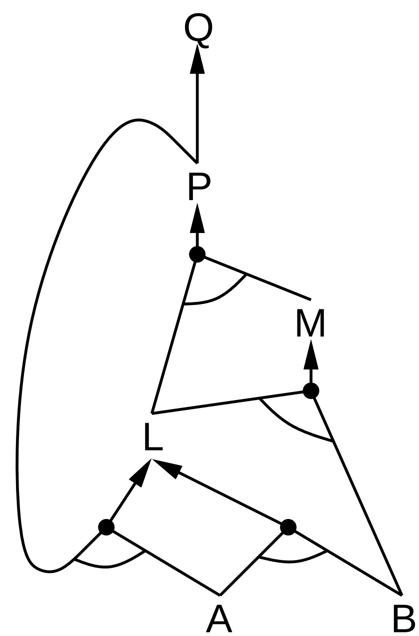
$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

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

Forward chaining

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

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

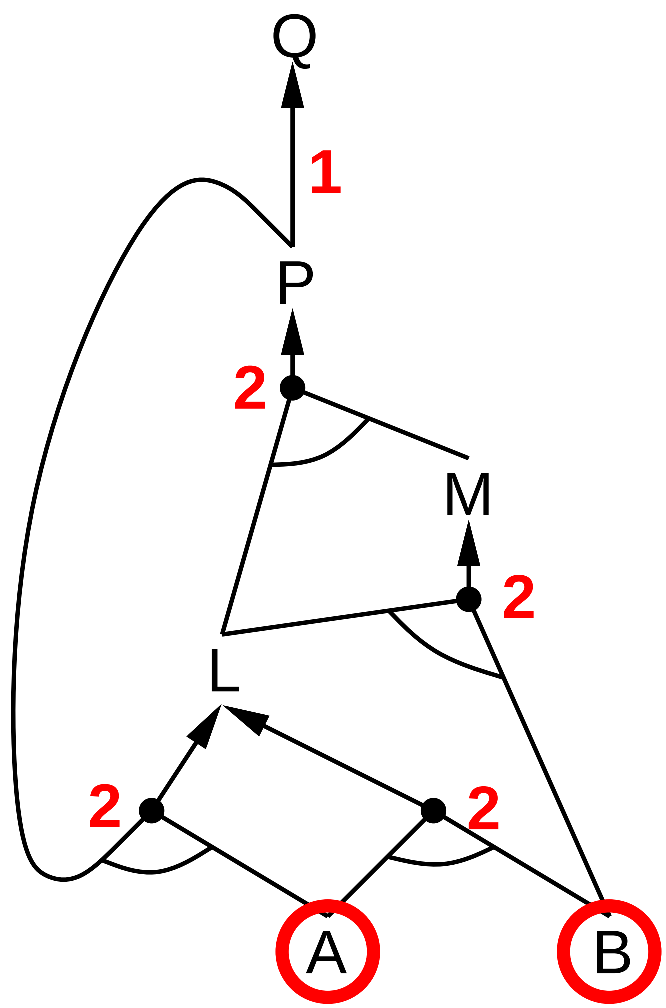


Forward chaining algorithm

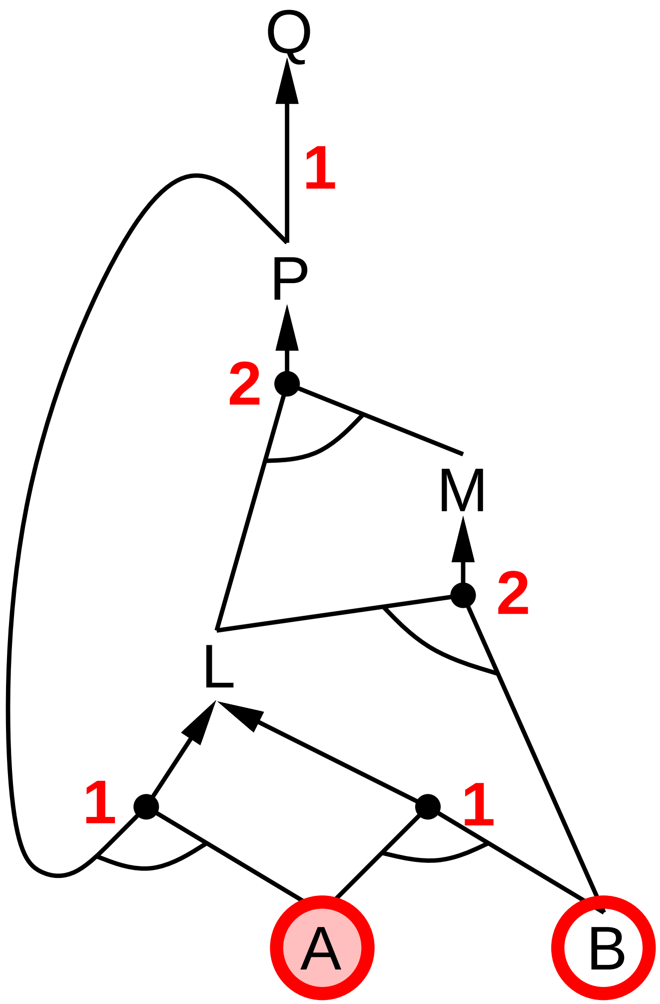
```
function PL-FC-Entails?(KB, q) returns true or false
   inputs: KB, the knowledge base, a set of propositional Horn clauses
           q, the query, a proposition symbol
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed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

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

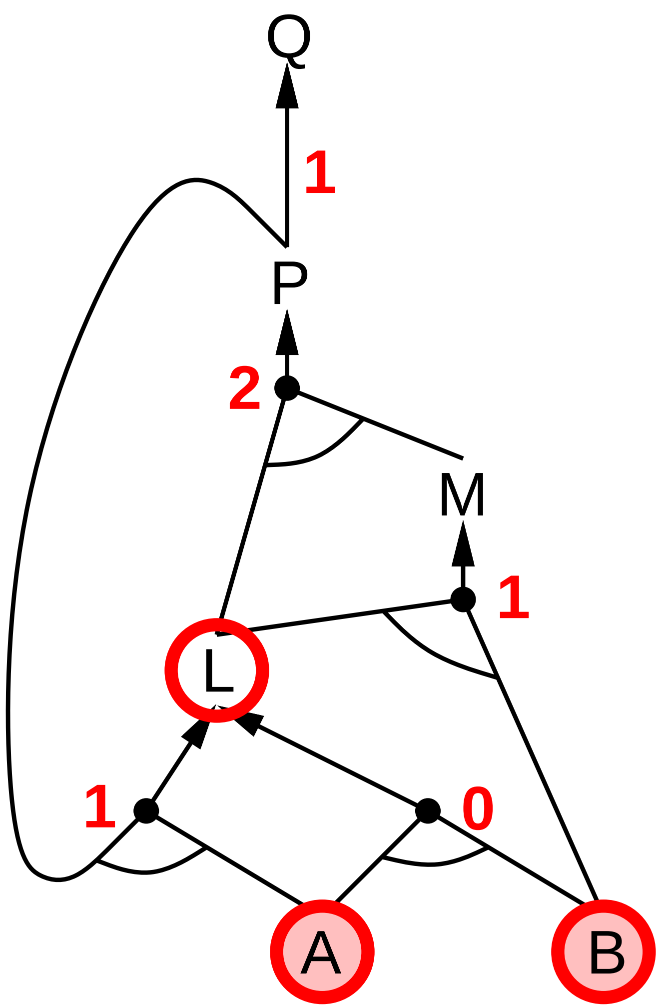
Forward chaining example



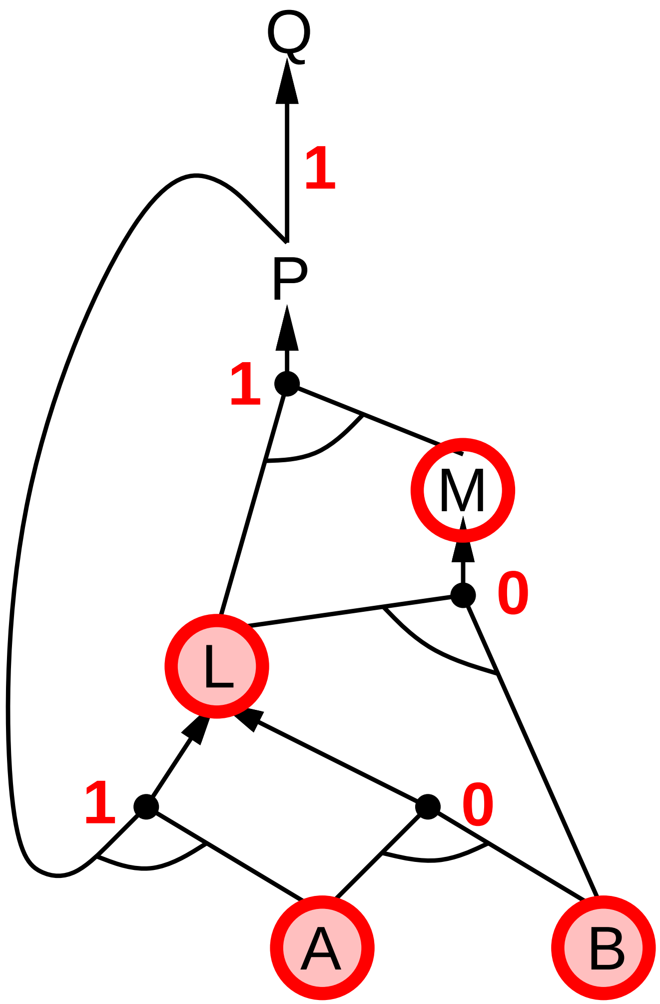
Forward chaining example



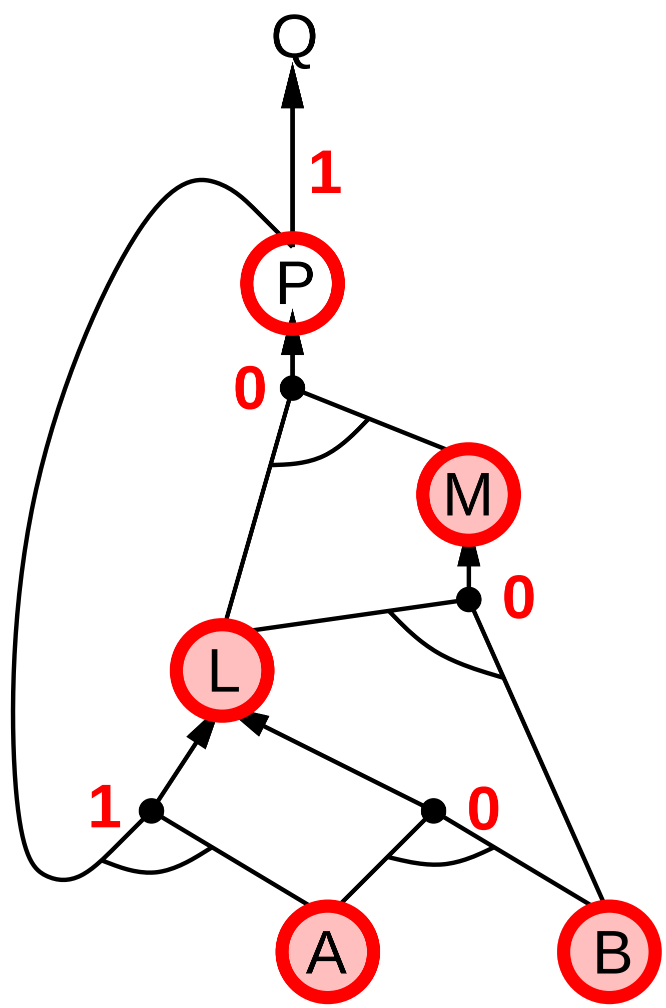
Forward chaining example



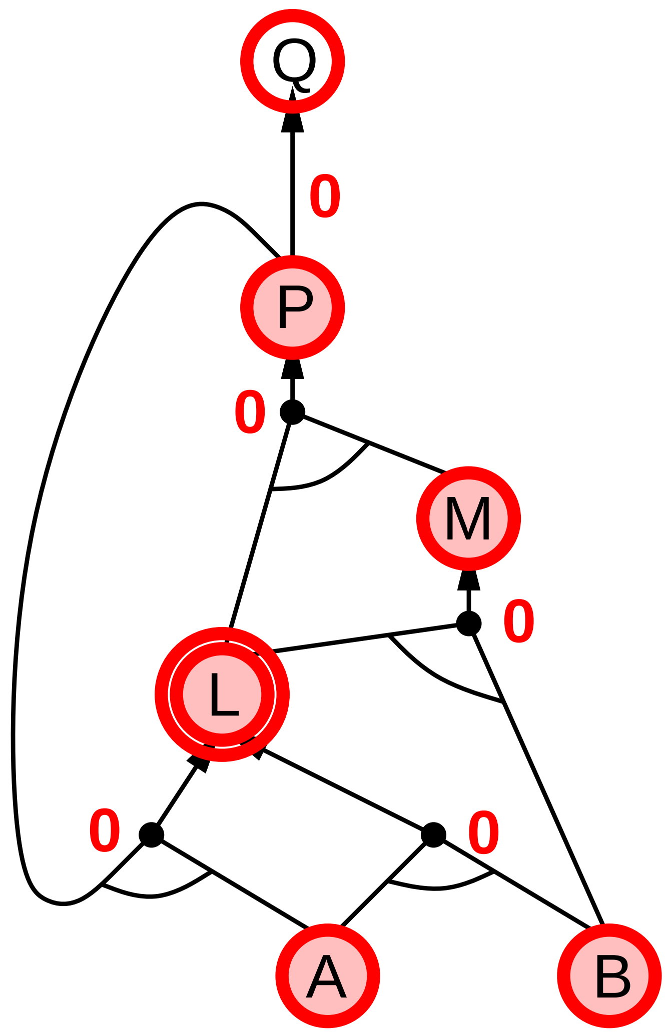
Forward chaining example



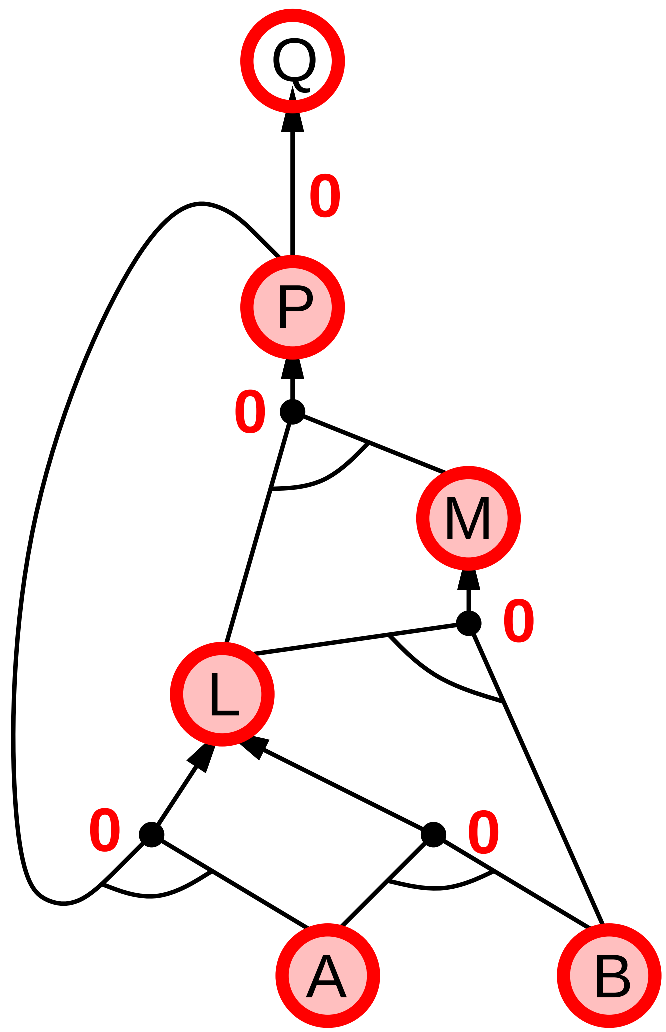
Forward chaining example



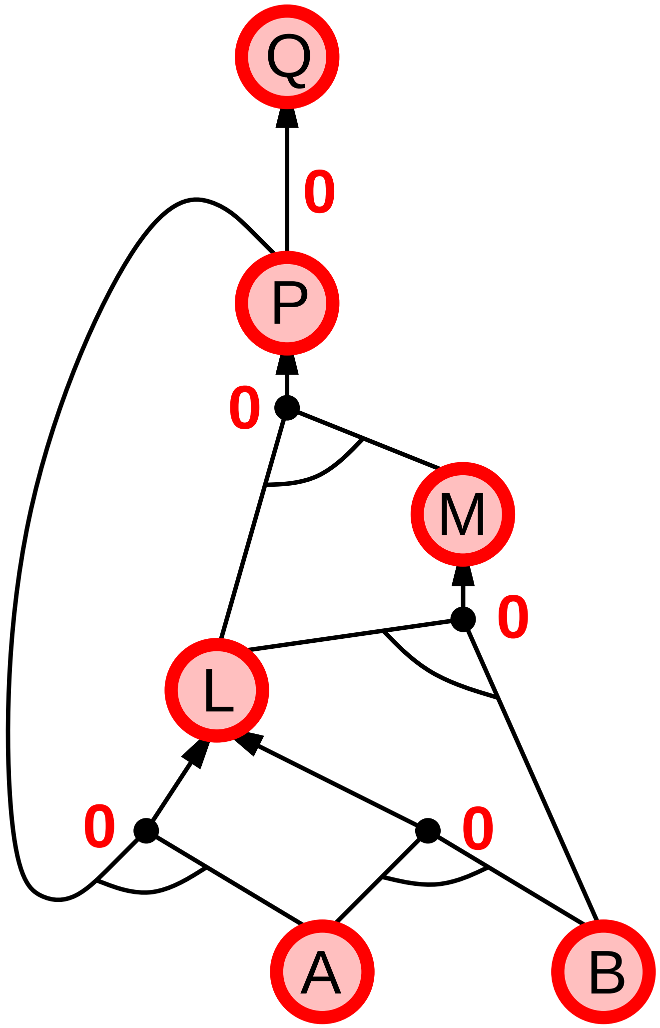
Forward chaining example



Forward chaining example



Forward chaining example



Proof of completeness

FC derives every atomic sentence that is entailed by KB

1. FC reaches a fixed point where no new atomic sentences are derived

2. Consider the final state as a model m, assigning true/false to symbols

3. Every clause in the original KB is true in m
   Proof: Suppose a clause $a_1 \land \ldots \land a_k \Rightarrow b$ is false in m
   Then $a_1 \land \ldots \land a_k$ is true in m and b is false in m
   Therefore the algorithm has not reached a fixed point!

4. Hence m is a model of KB

5. If $KB \models q$, then q is true in **every** model of KB, including m

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

Backward chaining

Idea: work backwards from the query q:

to prove q by BC,

check if q is known already, or

prove by BC all premises of some rule concluding q

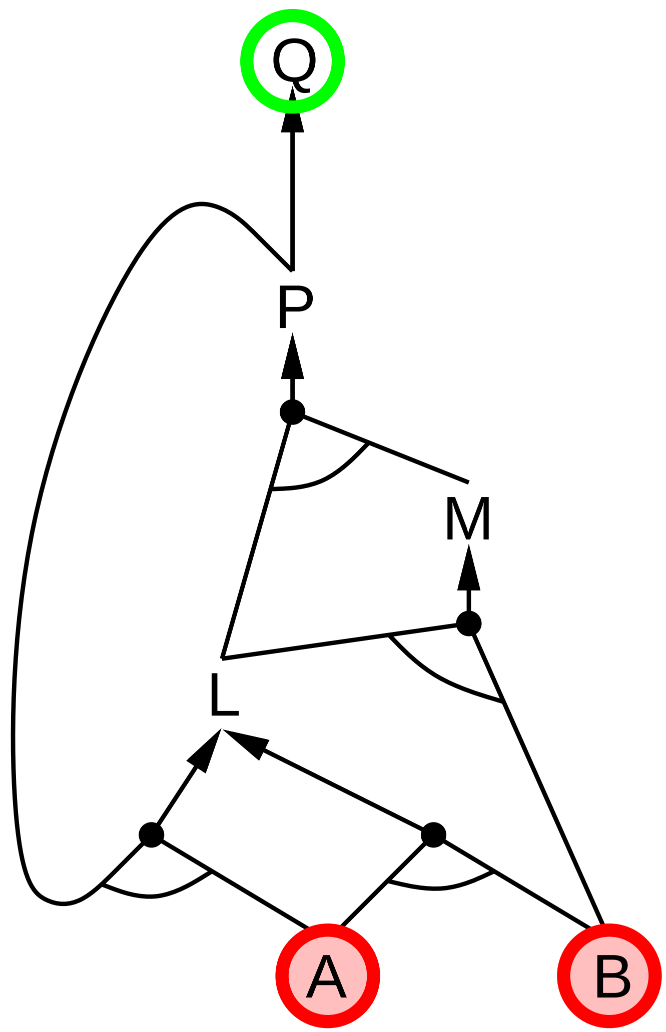
Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal

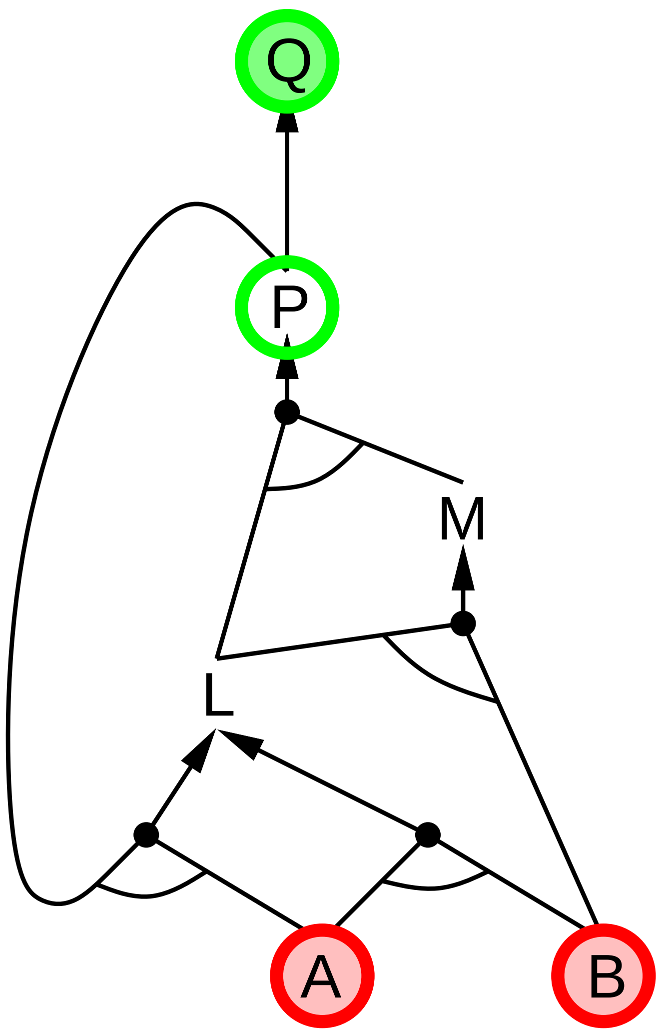
1) has already been proved true, or

2) has already failed

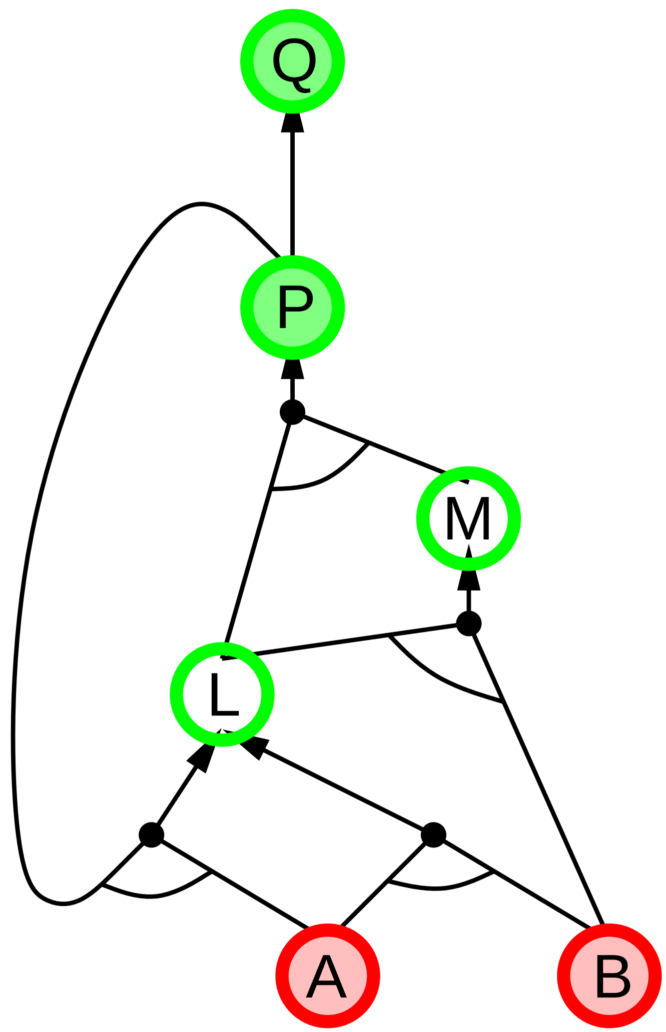
Backward chaining example



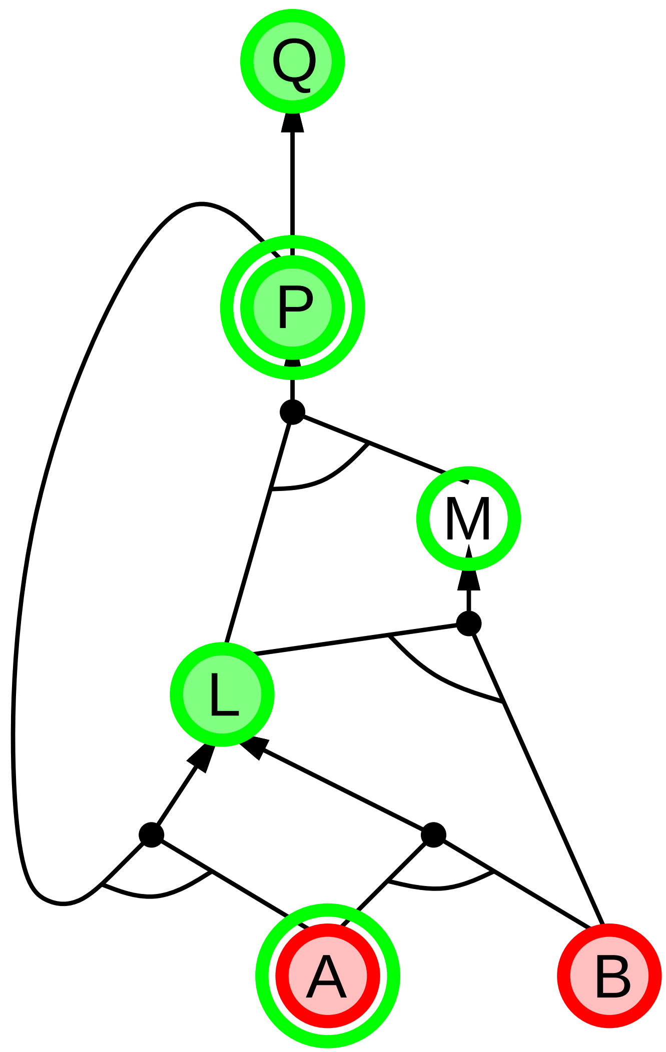
Backward chaining example



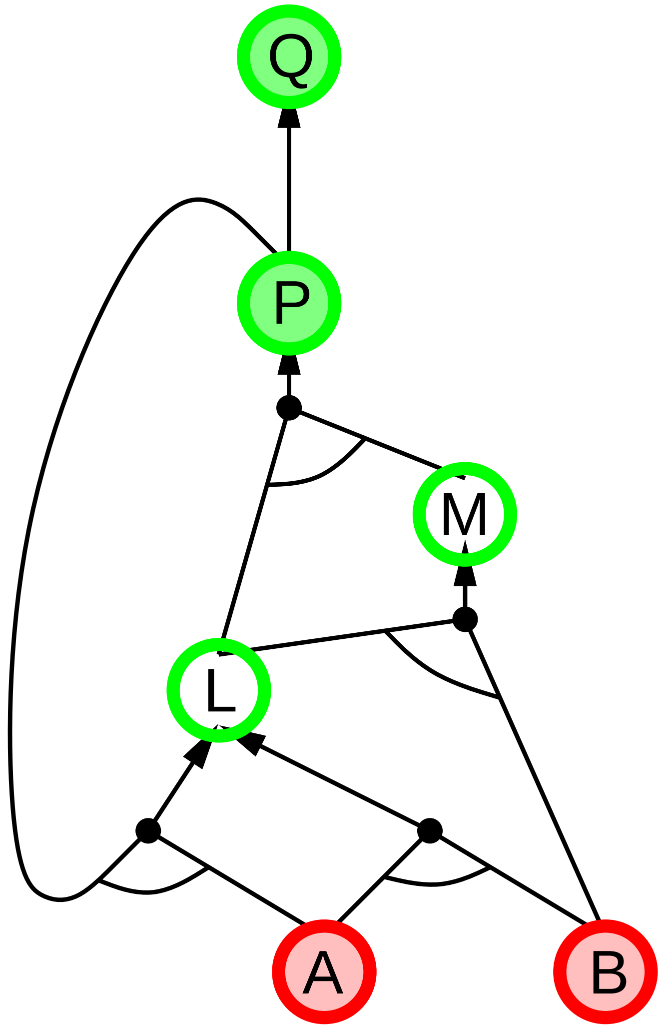
Backward chaining example



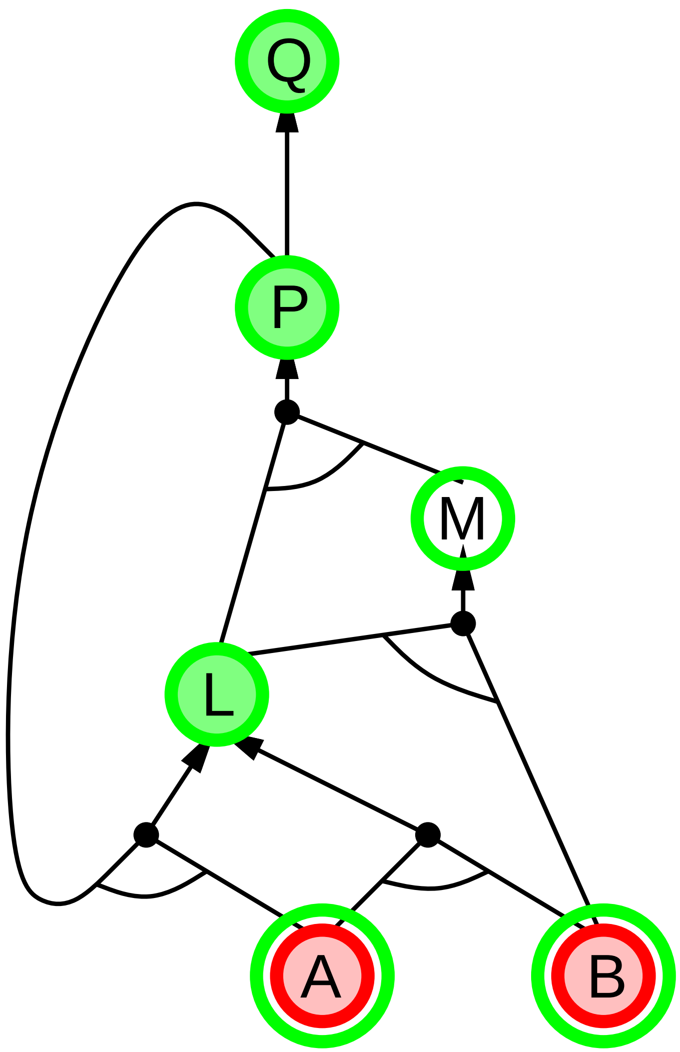
Backward chaining example



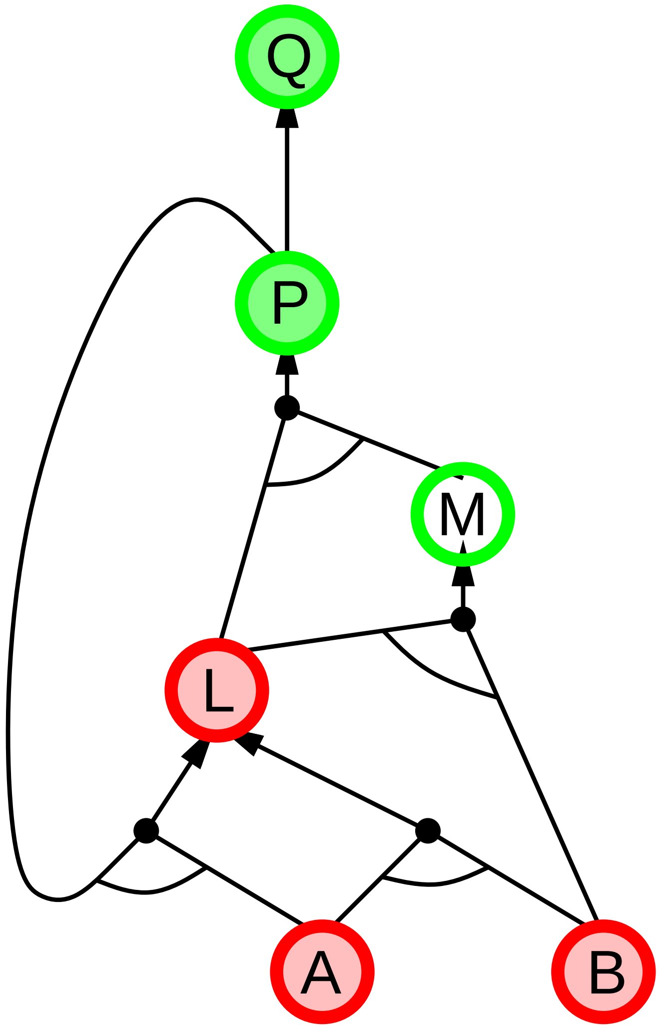
Backward chaining example



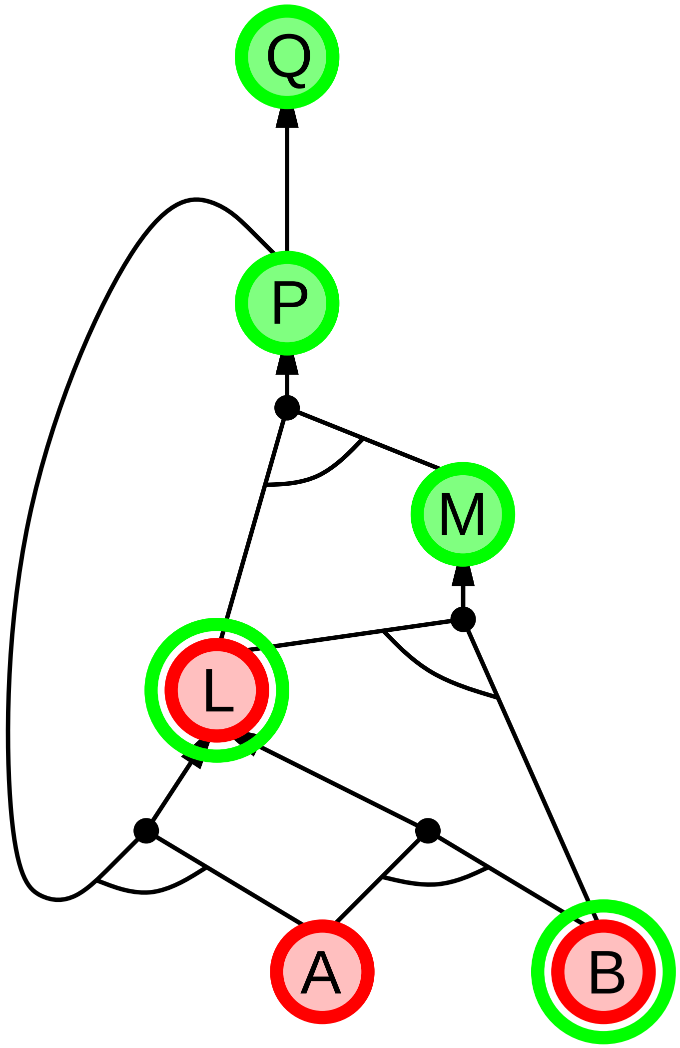
Backward chaining example



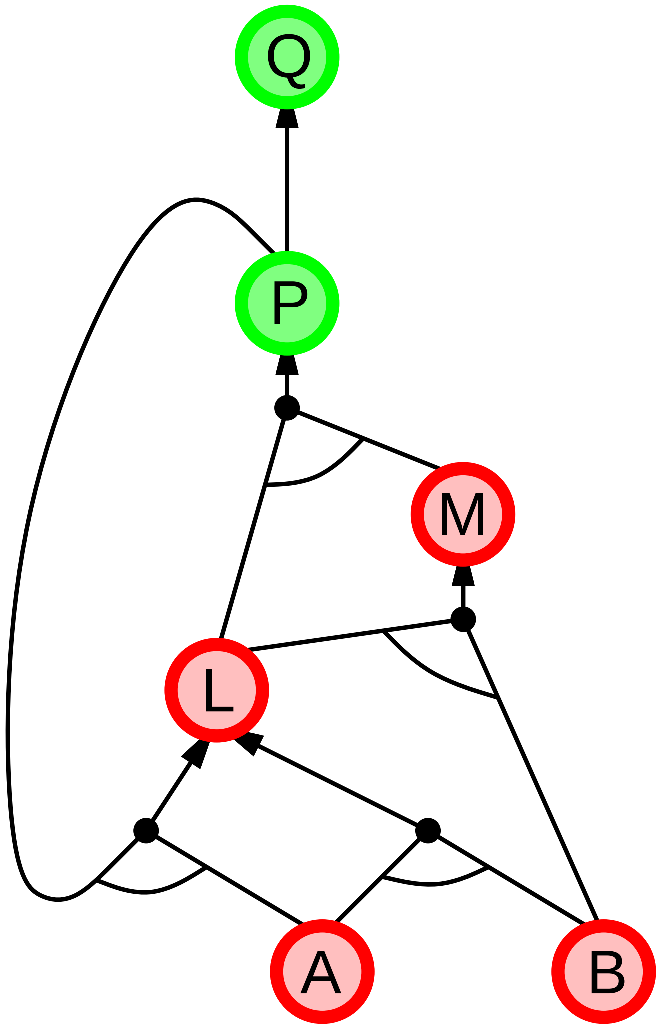
Backward chaining example



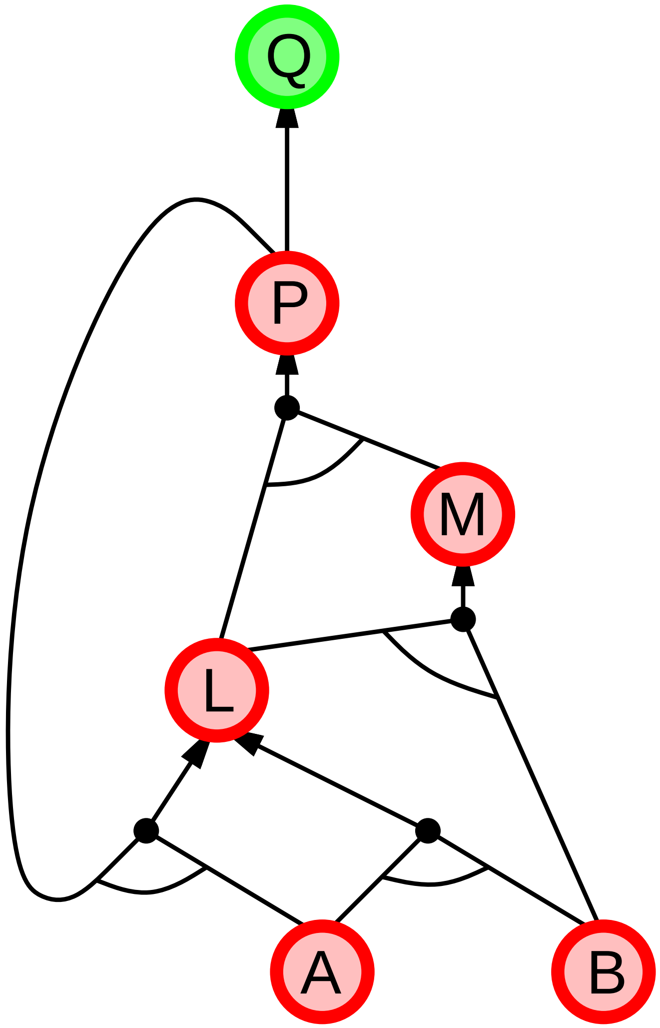
Backward chaining example



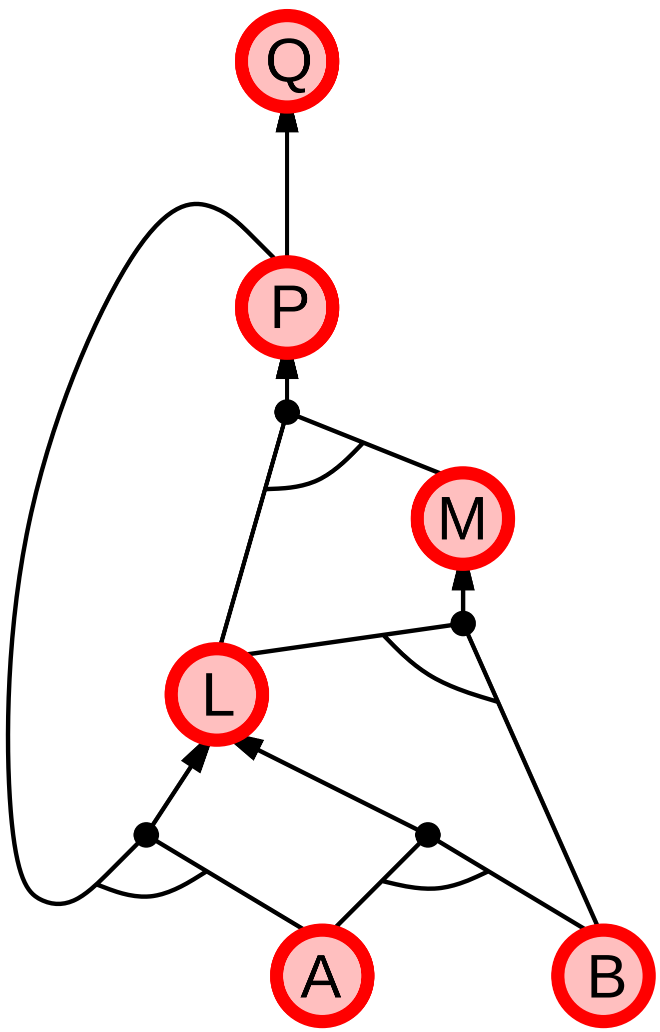
Backward chaining example



Backward chaining example



Backward chaining example



Forward vs. backward chaining

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

May do lots of work that is irrelevant to the goal

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

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

Resolution

Conjunctive Normal Form (CNF—universal)

$$\textbf{conjunction} \text{ of } \underbrace{\textbf{disjunctions} \text{ of } \textbf{literals}}_{\textbf{clauses}}$$

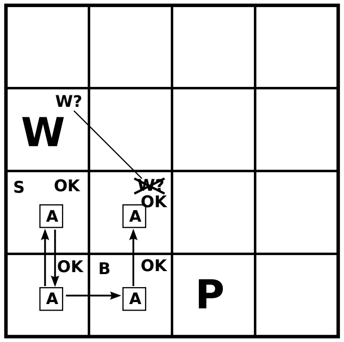
E.g., $(A \lor \neg B) \land (B \lor \neg C \lor \neg D)$

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

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

where $\ell_i \equiv \neg m_j$ are complementary literals. E.g.,

$$\frac{W_{1,3} \lor W_{2,2}, \qquad \neg W_{2,2}}{W_{1,3}}$$



Resolution is sound and complete for propositional logic

Conversion to CNF

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

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \;\Rightarrow\; \beta) \land (\beta \;\Rightarrow\; \alpha)$.

   $(B_{1,1} \;\Rightarrow\; (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \;\Rightarrow\; B_{1,1})$

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

   $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg(P_{1,2} \lor P_{2,1}) \lor B_{1,1})$

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

   $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1})$

4. Apply distributivity law ($\lor$ over $\land$) and flatten:

   $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})$

Resolution algorithm

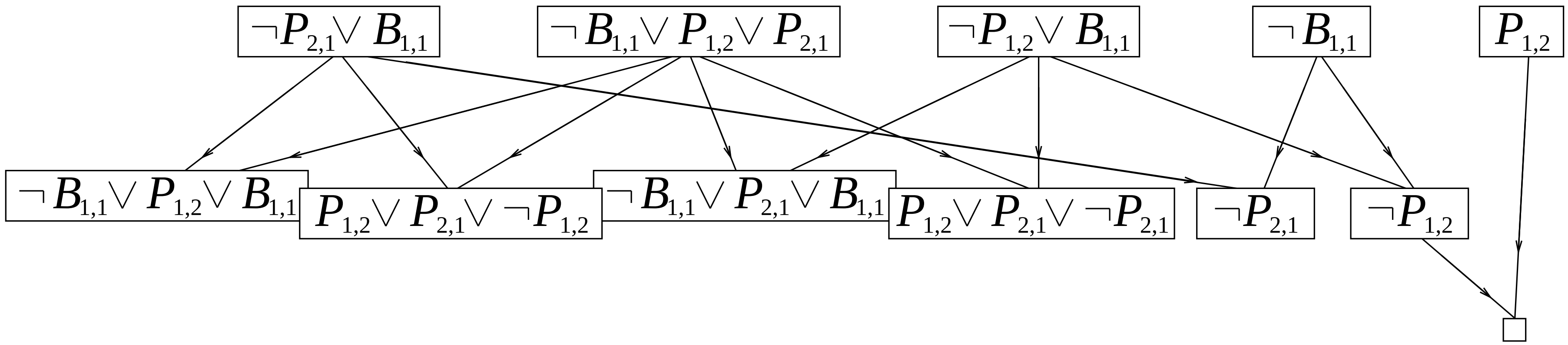
Proof by contradiction, i.e., show $KB \wedge \neg\alpha$ unsatisfiable

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

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

Resolution example

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



Summary

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

Basic concepts of logic:

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

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

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

Propositional logic lacks expressive power

First-order logic

Chapter 7

Outline

◇ Syntax and semantics of FOL

◇ Fun with sentences

◇ Wumpus world in FOL

Syntax of FOL: Basic elements

Constants	$KingJohn,\ 2,\ UCB, \ldots$
Predicates	$Brother,\ >, \ldots$
Functions	$Sqrt,\ LeftLegOf, \ldots$
Variables	$x,\ y,\ a,\ b, \ldots$
Connectives	$\wedge\ \vee\ \neg\ \Rightarrow\ \Leftrightarrow$
Equality	$=$
Quantifiers	$\forall\ \exists$

Atomic sentences

Atomic sentence $=$ $predicate(term_1, \ldots, term_n)$
or $term_1 = term_2$

Term $=$ $function(term_1, \ldots, term_n)$
or $constant$ or $variable$

E.g., $Brother(KingJohn, RichardTheLionheart)$
$> (Length(LeftLegOf(Richard)), Length(LeftLegOf(KingJohn)))$

Complex sentences

Complex sentences are made from atomic sentences using connectives

$$\neg S, \quad S_1 \wedge S_2, \quad S_1 \vee S_2, \quad S_1 \Rightarrow S_2, \quad S_1 \Leftrightarrow S_2$$

E.g. $Sibling(KingJohn, Richard) \Rightarrow Sibling(Richard, KingJohn)$
$>(1,2) \vee \leq(1,2)$
$>(1,2) \wedge \neg{>}(1,2)$

Truth in first-order logic

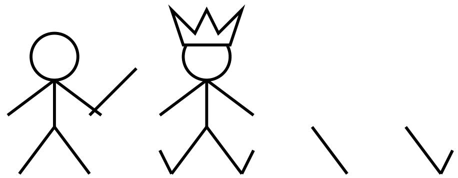
Sentences are true with respect to a model and an interpretation

Model contains objects and relations among them

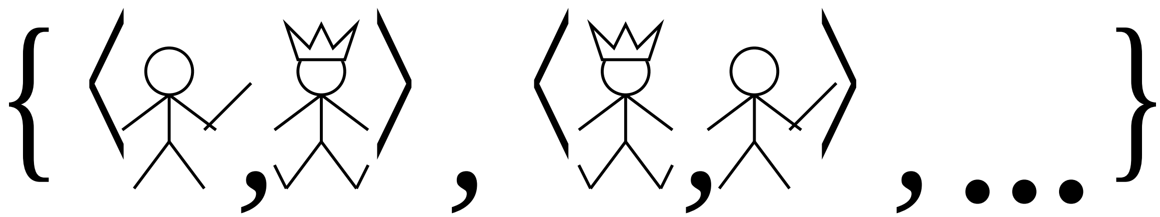
Interpretation specifies referents for
- $constant\ symbols \rightarrow$ objects
- $predicate\ symbols \rightarrow$ relations
- $function\ symbols \rightarrow$ functional relations

An atomic sentence $predicate(term_1, \ldots, term_n)$ is true
iff the objects referred to by $term_1, \ldots, term_n$
are in the relation referred to by $predicate$

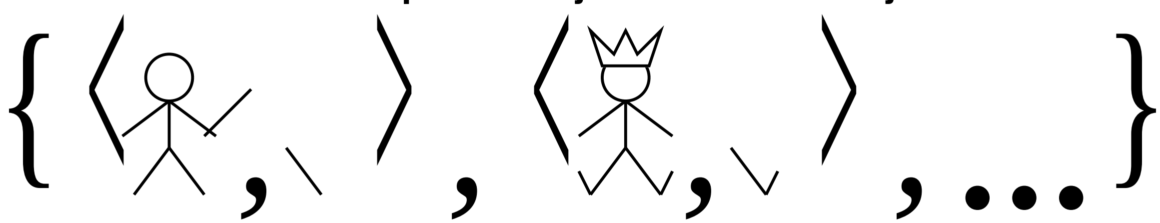
Models for FOL: Example

objects 

relations: sets of tuples of objects



functional relations: all tuples of objects + "value" object



Universal quantification

$\forall \langle variables \rangle \; \langle sentence \rangle$

Everyone at Berkeley is smart:
$\forall x \; At(x, Berkeley) \Rightarrow Smart(x)$

$\forall x \; P$ is equivalent to the conjunction of instantiations of P

$$\begin{aligned} & At(KingJohn, Berkeley) \Rightarrow Smart(KingJohn) \\ \wedge \; & At(Richard, Berkeley) \Rightarrow Smart(Richard) \\ \wedge \; & At(Berkeley, Berkeley) \Rightarrow Smart(Berkeley) \\ \wedge \; & \ldots \end{aligned}$$

Typically, $\Rightarrow$ is the main connective with $\forall$.
Common mistake: using $\wedge$ as the main connective with $\forall$:

$$\forall x \; At(x, Berkeley) \wedge Smart(x)$$

means "Everyone is at Berkeley and everyone is smart"

Existential quantification

$\exists \langle variables \rangle \;\; \langle sentence \rangle$

Someone at Stanford is smart:
$\exists x \;\; At(x, Stanford) \land Smart(x)$

$\exists x \;\; P$ is equivalent to the <u>disjunction</u> of <u>instantiations</u> of P

$$
\begin{aligned}
& At(KingJohn, Stanford) \land Smart(KingJohn) \\
\lor \;\; & At(Richard, Stanford) \land Smart(Richard) \\
\lor \;\; & At(Stanford, Stanford) \land Smart(Stanford) \\
\lor \;\; & \ldots
\end{aligned}
$$

Typically, $\land$ is the main connective with $\exists$.
Common mistake: using $\Rightarrow$ as the main connective with $\exists$:

$$\exists x \;\; At(x, Stanford) \;\Rightarrow\; Smart(x)$$

is true if there is anyone who is not at Stanford!

Properties of quantifiers

$\forall x\ \forall y$ is the same as $\forall y\ \forall x$ (why??)

$\exists x\ \exists y$ is the same as $\exists y\ \exists x$ (why??)

$\exists x\ \forall y$ is not the same as $\forall y\ \exists x$

$\exists x\ \forall y\ Loves(x, y)$
"There is a person who loves everyone in the world"

$\forall y\ \exists x\ Loves(x, y)$
"Everyone in the world is loved by at least one person"

Quantifier duality: each can be expressed using the other

$\forall x\ Likes(x, IceCream) \qquad \neg\exists x\ \neg Likes(x, IceCream)$

$\exists x\ Likes(x, Broccoli) \qquad \neg\forall x\ \neg Likes(x, Broccoli)$

Fun with sentences

Brothers are siblings

.

"Sibling" is reflexive

.

One's mother is one's female parent

.

A first cousin is a child of a parent's sibling

.

.

$$\forall x, y \;\; Brother(x, y) \;\Leftrightarrow\; Sibling(x, y).$$

.

$$\forall x, y \;\; Sibling(x, y) \;\Leftrightarrow\; Sibling(y, x)$$

.

$$\forall x, y \;\; Mother(x, y) \;\Leftrightarrow\; (Female(x) and Parent(x, y))$$

.

$$\forall x, y \;\; FirstCousin(x, y) \;\Leftrightarrow\; \exists p, ps \;\; Parent(p, x) \land Sibling(ps, p) \land Parent(ps, y)$$

Equality

$term_1 = term_2$ is true under a given interpretation
if and only if $term_1$ and $term_2$ refer to the same object

E.g., $1 = 2$ and $\forall\, x \;\; \times(Sqrt(x), Sqrt(x)) = x$ are satisfiable
$2 = 2$ is valid

E.g., definition of (full) $Sibling$ in terms of $Parent$:

$$\forall\, x, y \;\; Sibling(x,y) \;\Leftrightarrow\; [\neg(x\!=\!y) \wedge \exists\, m, f \;\; \neg(m\!=\!f) \wedge Parent(m,x) \wedge Parent(f,x) \wedge Parent(m,y) \wedge Parent(f,y)]$$

Interacting with FOL KBs

Suppose a wumpus-world agent is using an FOL KB
and perceives a smell and a breeze (but no glitter) at $t = 5$:

$\textsc{Tell}(KB, Percept([Smell, Breeze, None], 5))$
$\textsc{Ask}(KB, \exists\, a \;\; Action(a, 5))$

I.e., does the KB entail any particular actions at $t = 5$?

Answer: $Yes,\ \{a/Shoot\}$ $\quad\leftarrow$ <u>substitution</u> (binding list)

Given a sentence S and a substitution σ,
$S\sigma$ denotes the result of plugging σ into S; e.g.,
$S = Smarter(x, y)$
$\sigma = \{x/Hillary, y/Bill\}$
$S\sigma = Smarter(Hillary, Bill)$

$\textsc{Ask}(\mathrm{KB}, \mathrm{S})$ returns some/all σ such that $KB \models S\sigma$

Knowledge base for the wumpus world

"Perception"
$\forall\, b, g, t \;\; Percept([Smell, b, g], t) \;\Rightarrow\; Smelt(t)$
$\forall\, s, b, t \;\; Percept([s, b, Glitter], t) \;\Rightarrow\; AtGold(t)$

Reflex: $\forall\, t \;\; AtGold(t) \;\Rightarrow\; Action(Grab, t)$

Reflex with internal state: do we have the gold already?
$\forall\, t \;\; AtGold(t) \land \neg Holding(Gold, t) \;\Rightarrow\; Action(Grab, t)$

$Holding(Gold, t)$ cannot be observed
$\Rightarrow$ keeping track of change is essential

Deducing hidden properties

Properties of locations:

$\forall\, l, t \;\; At(Agent, l, t) \land Smelt(t) \;\Rightarrow\; Smelly(l)$

$\forall\, l, t \;\; At(Agent, l, t) \land Breeze(t) \;\Rightarrow\; Breezy(l)$

Squares are breezy near a pit:

Diagnostic rule—infer cause from effect

$$\forall\, y \;\; Breezy(y) \;\Rightarrow\; \exists\, x \;\; Pit(x) \land Adjacent(x, y)$$

Causal rule—infer effect from cause

$$\forall\, x, y \;\; Pit(x) \land Adjacent(x, y) \;\Rightarrow\; Breezy(y)$$

Neither of these is complete—e.g., the causal rule doesn't say whether squares far away from pits can be breezy

Definition for the $Breezy$ predicate:

$$\forall\, y \;\; Breezy(y) \;\Leftrightarrow\; [\exists\, x \;\; Pit(x) \land Adjacent(x, y)]$$

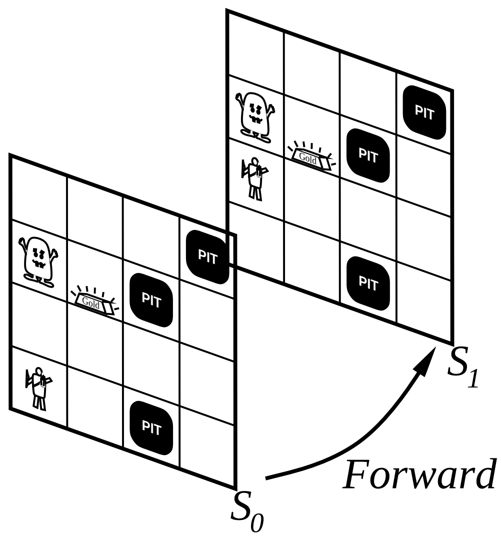
Keeping track of change

Facts hold in situations, rather than eternally
E.g., $Holding(Gold, Now)$ rather than just $Holding(Gold)$

Situation calculus is one way to represent change in FOL:

Adds a situation argument to each non-eternal predicate
E.g., Now in $Holding(Gold, Now)$ denotes a situation

Situations are connected by the $Result$ function
$Result(a, s)$ is the situation that results from doing a is s



Describing actions I

"Effect" axiom—describe changes due to action
$\forall s\;\; AtGold(s) \;\Rightarrow\; Holding(Gold, Result(Grab, s))$

"Frame" axiom—describe non-changes due to action
$\forall s\;\; HaveArrow(s) \;\Rightarrow\; HaveArrow(Result(Grab, s))$

Frame problem: find an elegant way to handle non-change
(a) representation—avoid frame axioms
(b) inference—avoid repeated "copy-overs" to keep track of state

Qualification problem: true descriptions of real actions require endless caveats—what if gold is slippery or nailed down or ...

Ramification problem: real actions have many secondary consequences—what about the dust on the gold, wear and tear on gloves, ...

Describing actions II

Successor-state axioms solve the representational frame problem

Each axiom is "about" a predicate (not an action per se):

$$\begin{array}{lcl} \text{P true afterwards} & \Leftrightarrow & [\text{an action made P true} \\ & \vee & \text{P true already and no action made P false}] \end{array}$$

For holding the gold:

$$\begin{array}{l} \forall\, a, s \;\; Holding(Gold, Result(a,s)) \;\Leftrightarrow \\ \quad [(a = Grab \wedge AtGold(s)) \\ \quad\; \vee\, (Holding(Gold, s) \wedge a \neq Release)] \end{array}$$

Making plans

Initial condition in KB:

$At(Agent, [1,1], S_0)$
$At(Gold, [1,2], S_0)$

Query: $\text{Ask}(KB, \exists\, s \;\; Holding(Gold, s))$
i.e., in what situation will I be holding the gold?

Answer: $\{s/Result(Grab, Result(Forward, S_0))\}$
i.e., go forward and then grab the gold

This assumes that the agent is interested in plans starting at S_0 and that S_0 is the only situation described in the KB

Making plans: A better way

Represent plans as action sequences $[a_1, a_2, \ldots, a_n]$

$PlanResult(p, s)$ is the result of executing p in s

Then the query $\text{Ask}(KB, \exists\, p \;\; Holding(Gold, PlanResult(p, S_0)))$ has the solution $\{p/[Forward, Grab]\}$

Definition of $PlanResult$ in terms of $Result$:

$$\forall\, s \;\; PlanResult([\,], s) = s$$
$$\forall\, a, p, s \;\; PlanResult([a|p], s) = PlanResult(p, Result(a, s))$$

Planning systems are special-purpose reasoners designed to do this type of inference more efficiently than a general-purpose reasoner

Summary

First-order logic:
- objects and relations are semantic primitives
- syntax: constants, functions, predicates, equality, quantifiers

Increased expressive power: sufficient to define wumpus world

Situation calculus:
- conventions for describing actions and change in FOL
- can formulate planning as inference on a situation calculus KB

Inference in first-order logic

Chapter 9, Sections 1–5

Outline

◇ Reducing first-order inference to propositional inference

◇ Unification

◇ Generalized Modus Ponens

◇ Forward and backward chaining

◇ Resolution

A brief history of reasoning

450B.C.	Stoics	propositional logic, inference (maybe)
322B.C.	Aristotle	"syllogisms" (inference rules), quantifiers
1565	Cardano	probability theory (propositional logic + uncertainty)
1847	Boole	propositional logic (again)
1879	Frege	first-order logic
1922	Wittgenstein	proof by truth tables
1930	Gödel	$\exists$ complete algorithm for FOL
1930	Herbrand	complete algorithm for FOL (reduce to propositional)
1931	Gödel	$\neg\exists$ complete algorithm for arithmetic
1960	Davis/Putnam	"practical" algorithm for propositional logic
1965	Robinson	"practical" algorithm for FOL—resolution

Universal instantiation (UI)

Every instantiation of a universally quantified sentence is entailed by it:

$$\frac{\forall\, v \quad \alpha}{\textsc{Subst}(\{v/g\}, \alpha)}$$

for any variable v and ground term g

E.g., $\forall\, x \quad King(x) \land Greedy(x) \;\Rightarrow\; Evil(x)$ yields

$$King(John) \land Greedy(John) \;\Rightarrow\; Evil(John)$$
$$King(Richard) \land Greedy(Richard) \;\Rightarrow\; Evil(Richard)$$
$$King(Father(John)) \land Greedy(Father(John)) \;\Rightarrow\; Evil(Father(John))$$
$$\vdots$$

Existential instantiation (EI)

For any sentence α, variable v, and constant symbol k
that does not appear elsewhere in the knowledge base:

$$\frac{\exists\, v \;\; \alpha}{\textsc{Subst}(\{v/k\}, \alpha)}$$

E.g., $\exists\, x \;\; Crown(x) \land OnHead(x, John)$ yields

$$Crown(C_1) \land OnHead(C_1, John)$$

provided C_1 is a new constant symbol, called a Skolem constant

Another example: from $\exists\, x \;\; d(x^y)/dy = x^y$ we obtain

$$d(e^y)/dy = e^y$$

provided e is a new constant symbol

Existential instantiation contd.

UI can be applied several times to **add** new sentences;
the new KB is logically equivalent to the old

EI can be applied once to **replace** the existential sentence;
the new KB is **not** equivalent to the old,
but is satisfiable iff the old KB was satisfiable

Reduction to propositional inference

Suppose the KB contains just the following:

$$\forall x \;\; King(x) \land Greedy(x) \;\Rightarrow\; Evil(x)$$
$$King(John)$$
$$Greedy(John)$$
$$Brother(Richard, John)$$

Instantiating the universal sentence in **all possible** ways, we have

$$King(John) \land Greedy(John) \;\Rightarrow\; Evil(John)$$
$$King(Richard) \land Greedy(Richard) \;\Rightarrow\; Evil(Richard)$$
$$King(John)$$
$$Greedy(John)$$
$$Brother(Richard, John)$$

The new KB is propositionalized: proposition symbols are

$King(John),\; Greedy(John),\; Evil(John), King(Richard)$ etc.

Reduction contd.

Claim: a ground sentence is entailed by new KB iff entailed by original KB

Claim: every FOL KB can be propositionalized so as to preserve entailment

Idea: propositionalize KB and query, apply resolution, return result

Problem: with function symbols, there are infinitely many ground terms,
e.g., $Father(Father(Father(John)))$

Theorem: Herbrand (1930). If a sentence α is entailed by an FOL KB,
it is entailed by a **finite** subset of the propositional KB

Idea: For $n = 0$ to ∞ do
create a propositional KB by instantiating with depth-n terms
see if α is entailed by this KB

Problem: works if α is entailed, loops if α is not entailed

Theorem: Turing (1936), Church (1936), entailment in FOL is semidecidable

Problems with propositionalization

Propositionalization seems to generate lots of irrelevant sentences.
E.g., from

$$\forall x \;\; King(x) \wedge Greedy(x) \;\Rightarrow\; Evil(x)$$
$$King(John)$$
$$\forall y \;\; Greedy(y)$$
$$Brother(Richard, John)$$

it seems obvious that $Evil(John)$, but propositionalization produces lots of facts such as $Greedy(Richard)$ that are irrelevant

With p k-ary predicates and n constants, there are $p \cdot n^k$ instantiations

With function symbols, it gets much much worse!

Unification

We can get the inference immediately if we can find a substitution θ such that $King(x)$ and $Greedy(x)$ match $King(John)$ and $Greedy(y)$

E.g., $\theta = \{x/John, y/John\}$ works

$\textsc{Unify}(\alpha, \beta) = \theta$ where $\alpha\theta = \beta\theta$

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, Bill)$	$\{x/Bill, y/John\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$
$Knows(John, x)$	$Knows(x, Elizabeth)$	$fail$

Standardizing apart eliminates overlap of variables, e.g., x/z_{17} in q:

$Knows(John, x)$	$Knows(z_{17}, Elizabeth)$	$\{x/Elizabeth, z_{17}/John\}$

Generalized Modus Ponens (GMP)

$$\frac{p_1{}',\ \ p_2{}',\ \ldots,\ p_n{}',\ \ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta}$$ where $p_i{}'\theta = p_i\theta$ for all i

$p_1{}'$ is $King(John)$ p_1 is $King(x)$
$p_2{}'$ is $Greedy(y)$ p_2 is $Greedy(x)$
θ is $\{x/John, y/John\}$ q is $Evil(x)$
$q\theta$ is $Evil(John)$

GMP is used with a KB of definite clauses (**exactly** one positive literal)
All variables are assumed to be universally quantified

Theorem: GMP is sound

Soundness of GMP

We need to show that

$$p_1', \ \ldots, \ p_n', \ \ (p_1 \land \ldots \land p_n \Rightarrow q) \models q\theta$$

provided that $p_i'\theta = p_i\theta$ for all i

Lemma: For any definite clause p, we have $p \models p\theta$ by UI

1. $(p_1 \land \ldots \land p_n \Rightarrow q) \models (p_1 \land \ldots \land p_n \Rightarrow q)\theta = (p_1\theta \land \ldots \land p_n\theta \Rightarrow q\theta)$
2. $p_1', \ \ldots, \ p_n' \models p_1' \land \ldots \land p_n' \models p_1'\theta \land \ldots \land p_n'\theta$
3. From 1 and 2, $q\theta$ follows by ordinary Modus Ponens

Example knowledge base

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

Prove that Colonel West is a criminal.

Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:

$$American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$$

Nono ... has some missiles, i.e., $\exists\, x\ Owns(Nono, x) \land Missile(x)$:

$Owns(Nono, M_1)$ and $Missile(M_1)$

... all of its missiles were sold to it by Colonel West:

$$\forall\, x\ \ Missile(x) \land Owns(Nono, x) \ \Rightarrow\ Sells(West, x, Nono)$$

Missiles are weapons:

$$Missile(x) \Rightarrow Weapon(x)$$

An enemy of America counts as "hostile":

$$Enemy(x, America) \ \Rightarrow\ Hostile(x)$$

West, who is American ...

$$American(West)$$

The country Nono, an enemy of America ...

$$Enemy(Nono, America)$$

Forward chaining algorithm

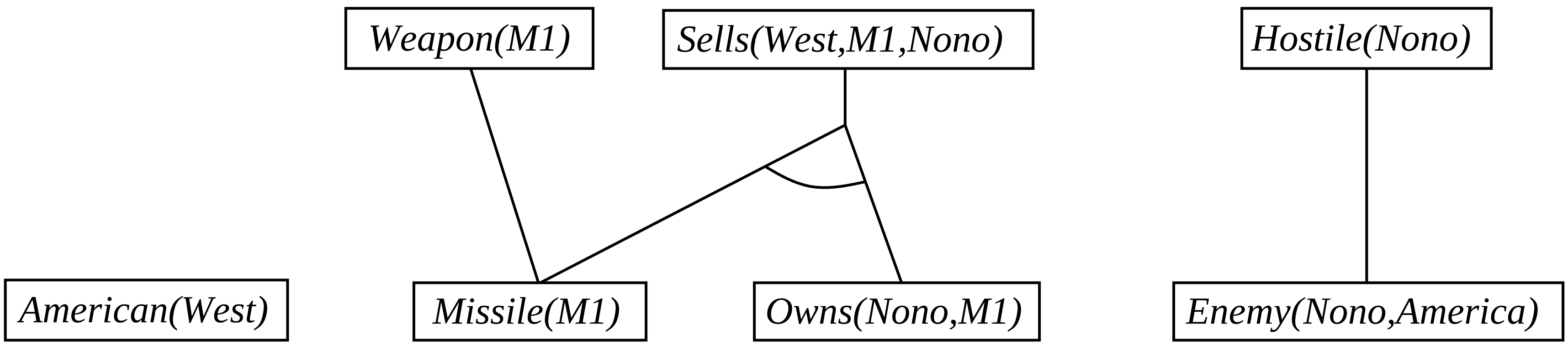
```
function FOL-FC-ASK(KB, α) returns a substitution or false

    repeat until new is empty
        new ← { }
        for each sentence r in KB do
            ( p1 ∧ . . . ∧ pn ⇒ q) ← STANDARDIZE-APART(r)
            for each θ such that (p1 ∧ . . . ∧ pn)θ = (p′1 ∧ . . . ∧ p′n)θ
                            for some p′1, . . . , p′n in KB
                q′ ← SUBST(θ, q)
                if q′ is not a renaming of a sentence already in KB or new then do
                    add q′ to new
                    φ ← UNIFY(q′, α)
                    if φ is not fail then return φ
        add new to KB
    return false
```

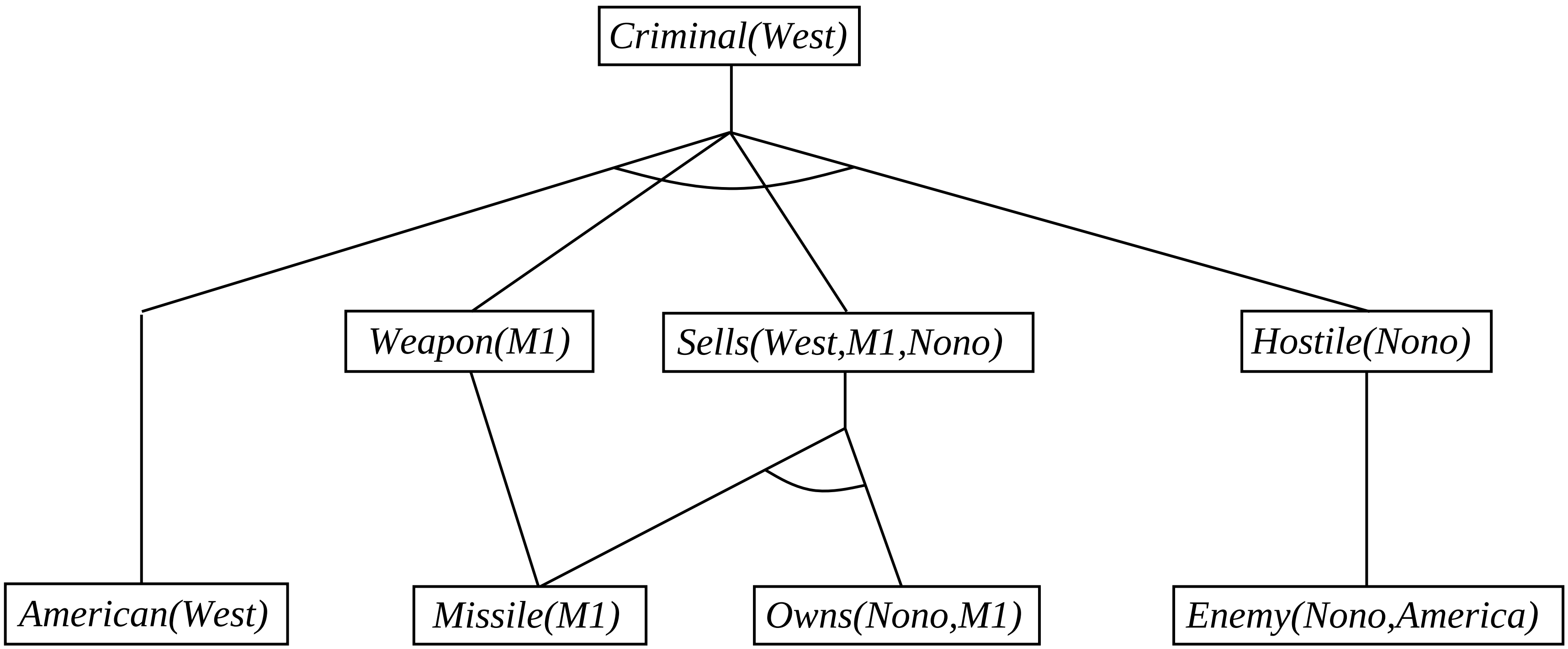
Forward chaining proof

American(West) *Missile(M1)* *Owns(Nono,M1)* *Enemy(Nono,America)*

Forward chaining proof



Forward chaining proof



Properties of forward chaining

Sound and complete for first-order definite clauses
(proof similar to propositional proof)

Datalog = first-order definite clauses + **no functions** (e.g., crime KB)
FC terminates for Datalog in polynomial time: at most $p \cdot n^k$ literals

May not terminate in general if α is not entailed

This is unavoidable: entailment with definite clauses is semidecidable

Backward chaining algorithm

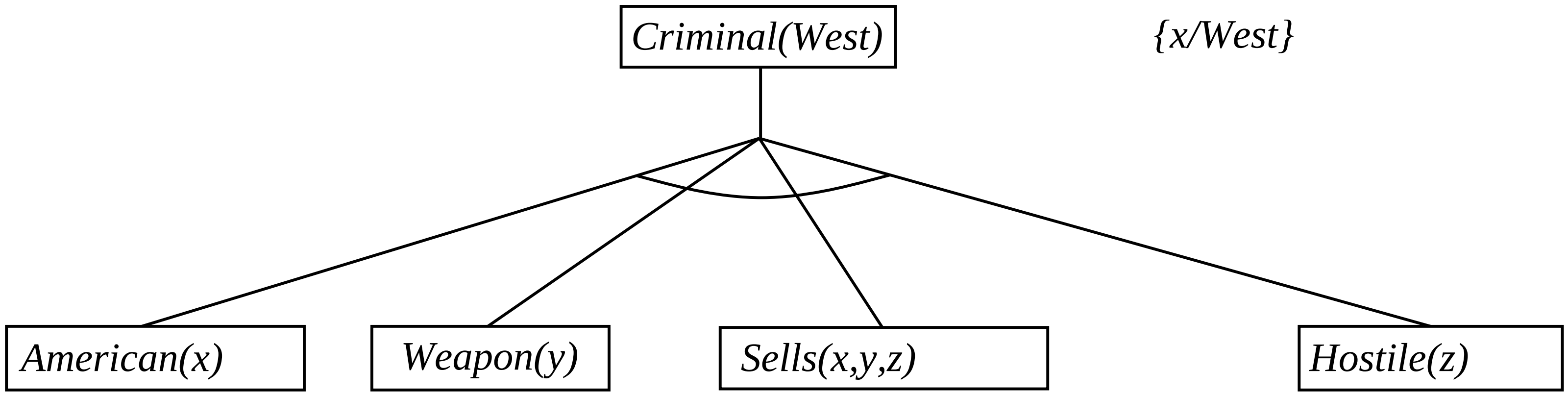
function FOL-BC-ASK($KB, goals, \theta$) **returns** a set of substitutions
inputs: KB, a knowledge base
$goals$, a list of conjuncts forming a query (θ already applied)
θ, the current substitution, initially the empty substitution $\{\,\}$
local variables: $answers$, a set of substitutions, initially empty

if $goals$ is empty **then return** $\{\theta\}$
$q' \leftarrow \textsc{Subst}(\theta, \textsc{First}(goals))$
for each sentence r **in** KB
where $\textsc{Standardize-Apart}(r) = (\,p_1 \;\wedge\; \ldots \;\wedge\; p_n \;\Rightarrow\; q)$
and $\theta' \leftarrow \textsc{Unify}(q, q')$ succeeds
$new\_goals \leftarrow [\,p_1, \ldots, p_n | \textsc{Rest}(goals)]$
$answers \leftarrow \text{FOL-BC-}\textsc{Ask}(KB, new\_goals, \textsc{Compose}(\theta', \theta)) \cup answers$
return $answers$

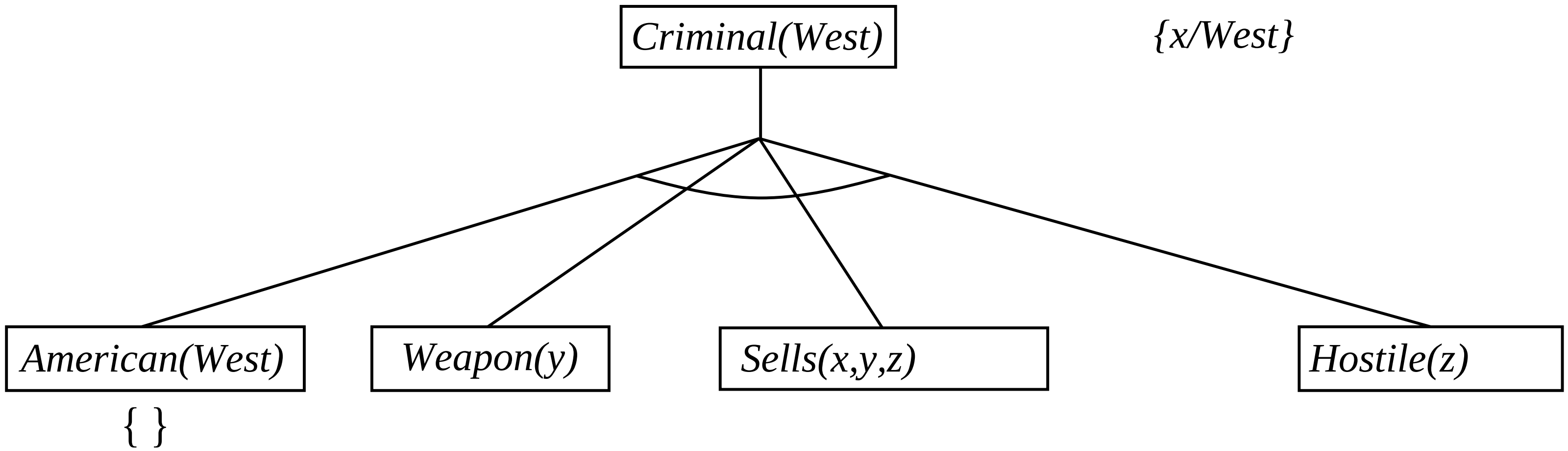
Backward chaining example

$\boxed{Criminal(West)}$

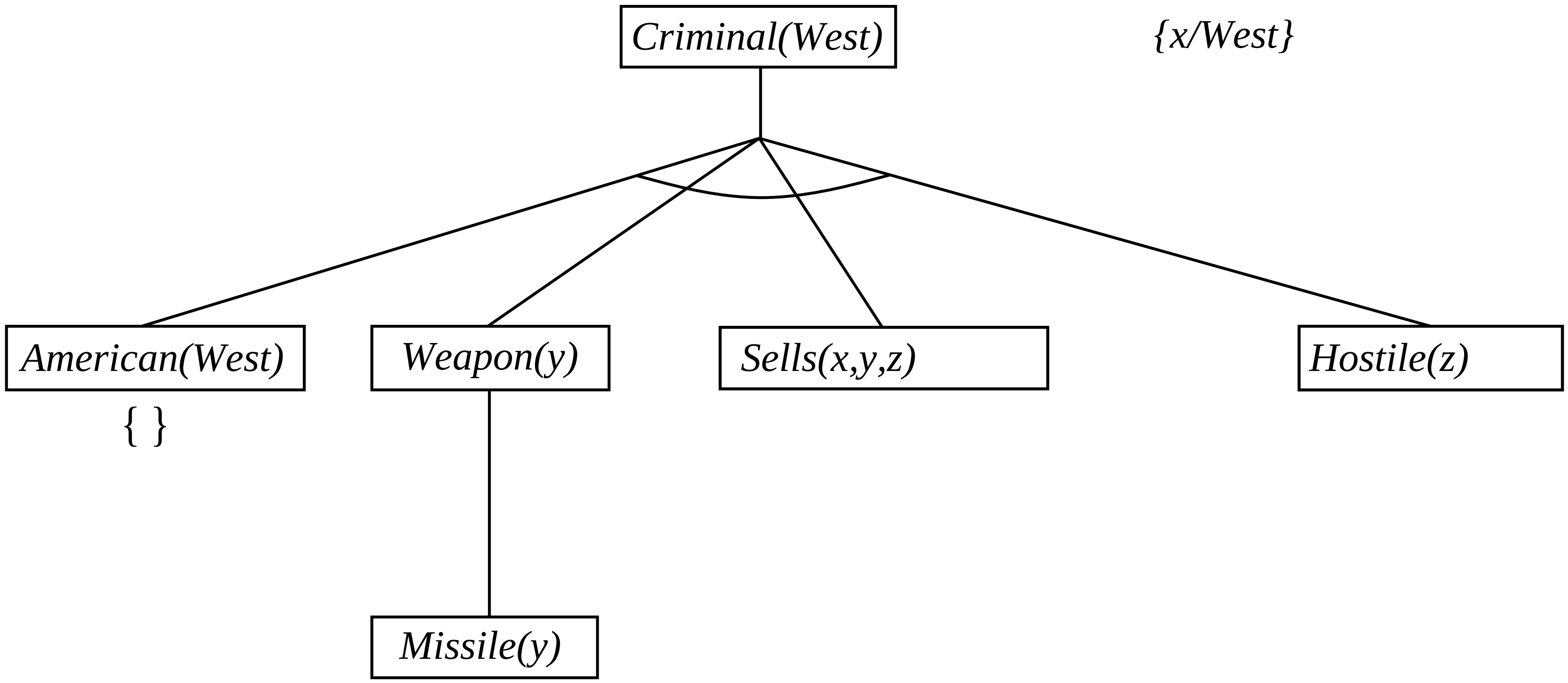
Backward chaining example



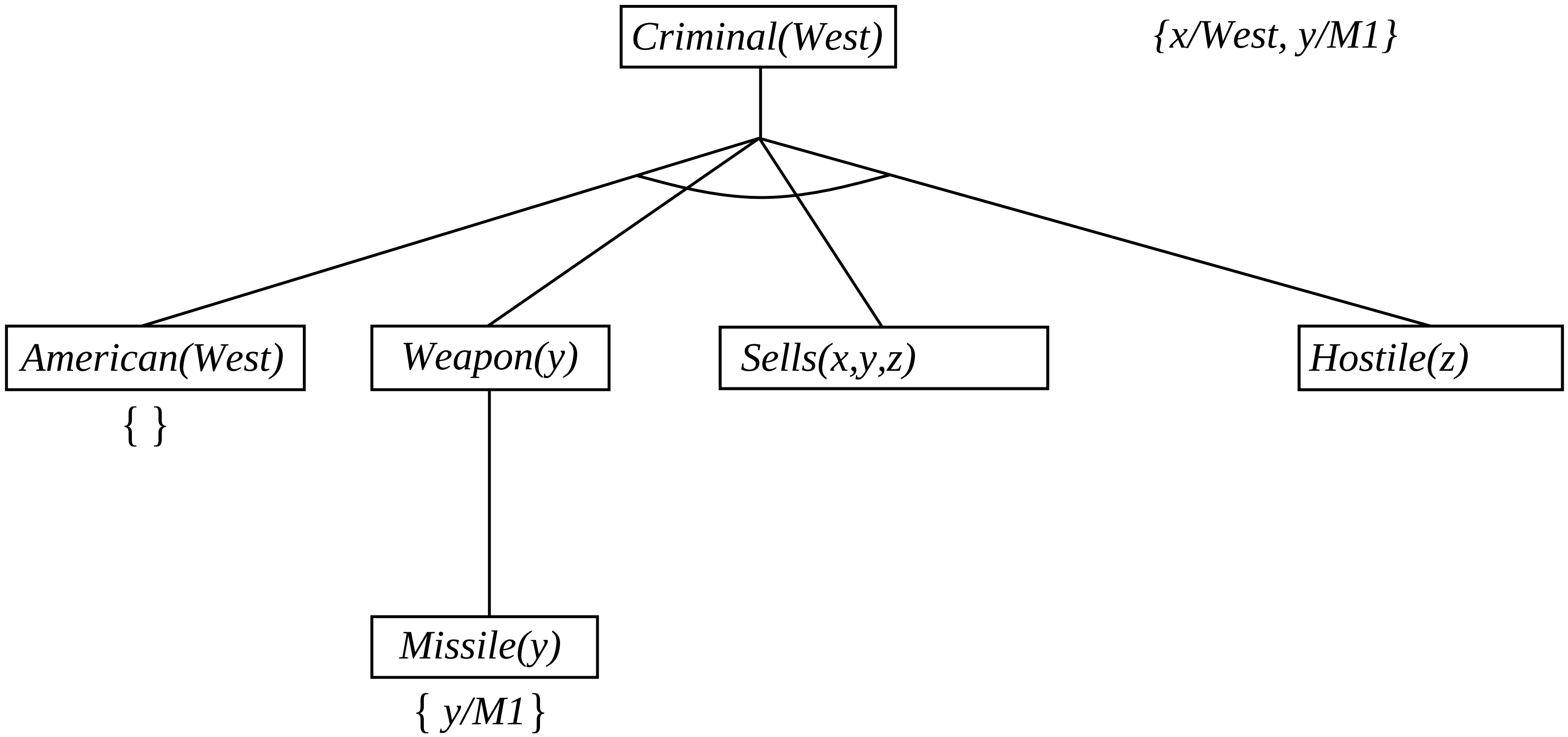
Backward chaining example



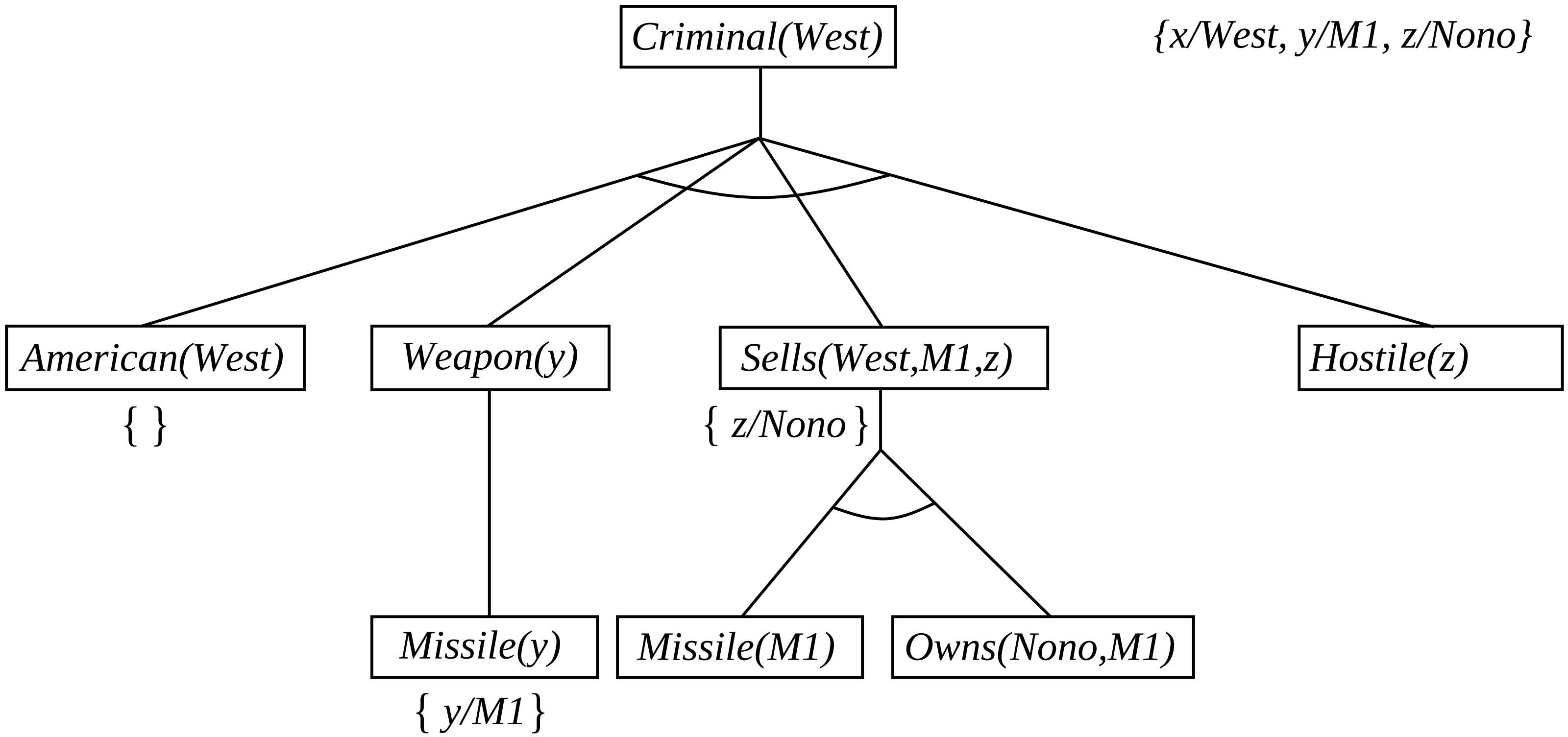
Backward chaining example



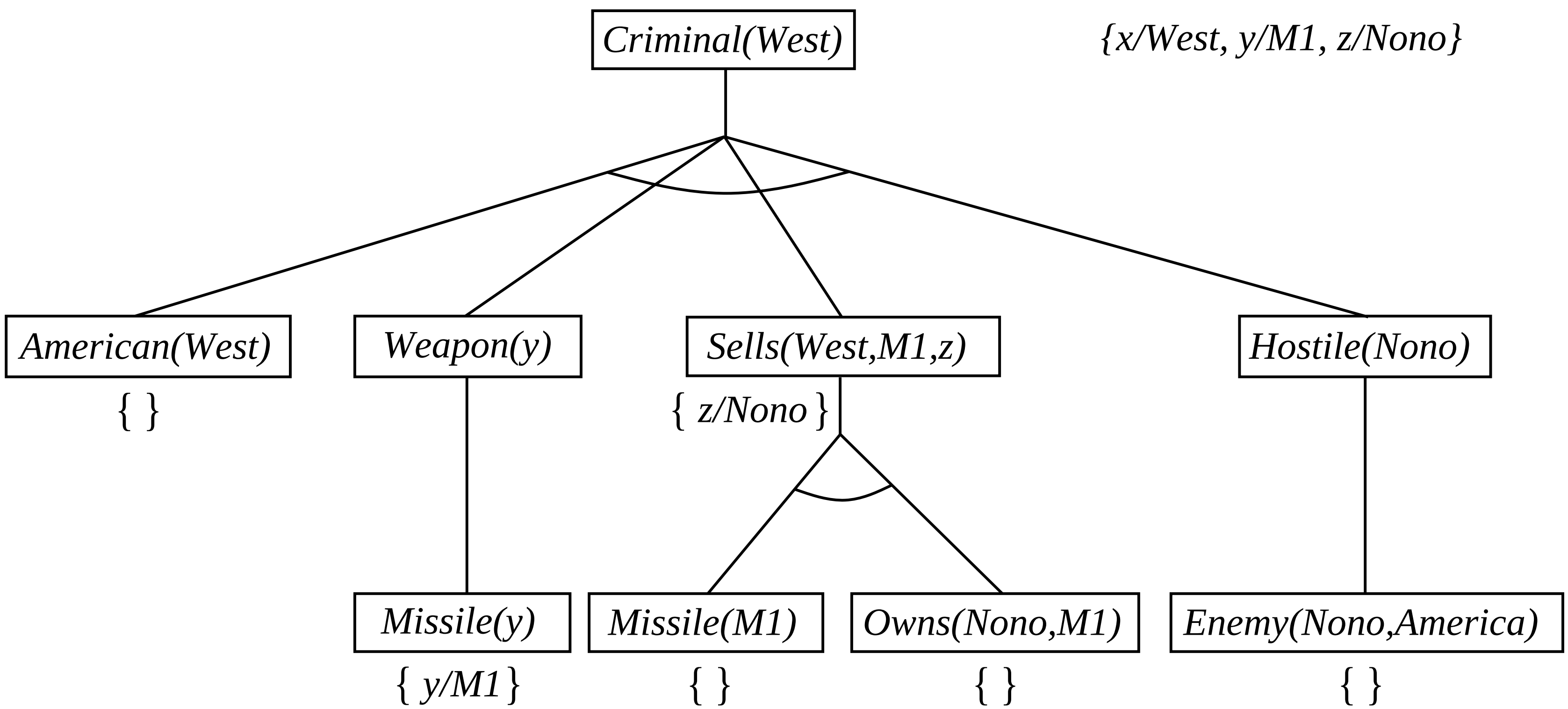
Backward chaining example



Backward chaining example



Backward chaining example



Properties of backward chaining

Depth-first recursive proof search: space is linear in size of proof

Incomplete due to infinite loops
$\Rightarrow$ fix by checking current goal against every goal on stack

Inefficient due to repeated subgoals (both success and failure)
$\Rightarrow$ fix using caching of previous results (extra space!)

Widely used (without improvements!) for logic programming

Resolution: brief summary

Full first-order version:

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

where $\textsc{Unify}(\ell_i, \neg m_j) = \theta$.

For example,

$$\frac{\begin{array}{l}\neg Rich(x) \vee Unhappy(x) \\ Rich(Ken)\end{array}}{Unhappy(Ken)}$$

with $\theta = \{x/Ken\}$

Apply resolution steps to $CNF(KB \wedge \neg\alpha)$; complete for FOL

Conversion to CNF

Everyone who loves all animals is loved by someone:

$$\forall x\ \ [\forall y\ \ Animal(y) \ \Rightarrow\ Loves(x, y)] \ \Rightarrow\ [\exists y\ \ Loves(y, x)]$$

1. Eliminate biconditionals and implications

$$\forall x\ \ [\neg\forall y\ \ \neg Animal(y) \lor Loves(x, y)] \lor [\exists y\ \ Loves(y, x)]$$

2. Move $\neg$ inwards: $\neg\forall x, p\ \ \equiv \exists x\ \ \neg p$, $\ \neg\exists x, p\ \ \equiv \forall x\ \ \neg p$:

$$\forall x\ \ [\exists y\ \ \neg(\neg Animal(y) \lor Loves(x, y))] \lor [\exists y\ \ Loves(y, x)]$$

$$\forall x\ \ [\exists y\ \ \neg\neg Animal(y) \land \neg Loves(x, y)] \lor [\exists y\ \ Loves(y, x)]$$

$$\forall x\ \ [\exists y\ \ Animal(y) \land \neg Loves(x, y)] \lor [\exists y\ \ Loves(y, x)]$$

Conversion to CNF contd.

3. Standardize variables: each quantifier should use a different one

$$\forall\, x \;\; [\exists\, y \;\; Animal(y) \land \neg Loves(x, y)] \lor [\exists\, z \;\; Loves(z, x)]$$

4. Skolemize: a more general form of existential instantiation.
   Each existential variable is replaced by a Skolem function
   of the enclosing universally quantified variables:

$$\forall\, x \;\; [Animal(F(x)) \land \neg Loves(x, F(x))] \lor Loves(G(x), x)$$

5. Drop universal quantifiers:

$$[Animal(F(x)) \land \neg Loves(x, F(x))] \lor Loves(G(x), x)$$

6. Distribute $\land$ over $\lor$:

$$[Animal(F(x)) \lor Loves(G(x), x)] \land [\neg Loves(x, F(x)) \lor Loves(G(x), x)]$$

Resolution proof: definite clauses