Introduction to Artificial Intelligence

Logical Reasoning

Henry Kautz

Outline

- Logic
- Efficient satisfiability testing by backtracking search
- Efficient satisfiability testing by local search
- Applications

Summary

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

Basic concepts of logic:

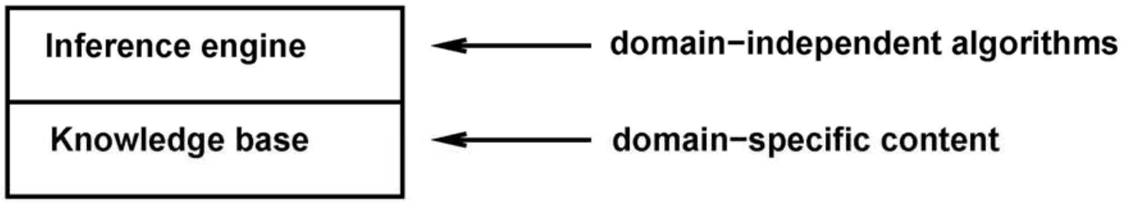
- syntax: formal structure of sentences
- semantics: truth of sentences wrt models
- entailment: necessary truth of one sentence given another
- inference: deriving sentences from other sentences
- soundess: derivations produce only entailed sentences
- completeness: derivations can produce all entailed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

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

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

Powerful & practical reasoning algorithms search through
space of partial or total truth assignments

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

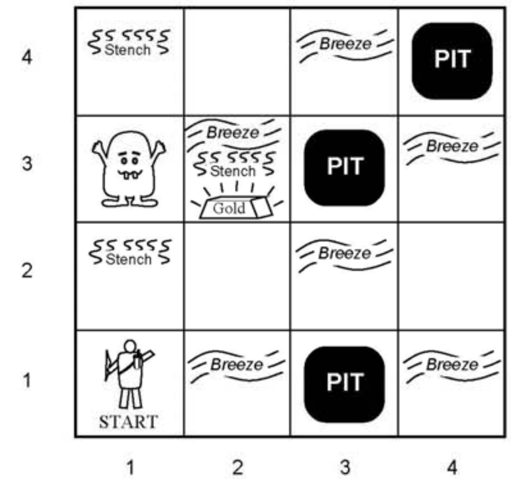
Glitter iff gold is in the same square

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



Sensors Breeze, Glitter, Smell

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

Wumpus world characterization

Observable?? No—only local perception

Deterministic?? Yes—outcomes exactly specified

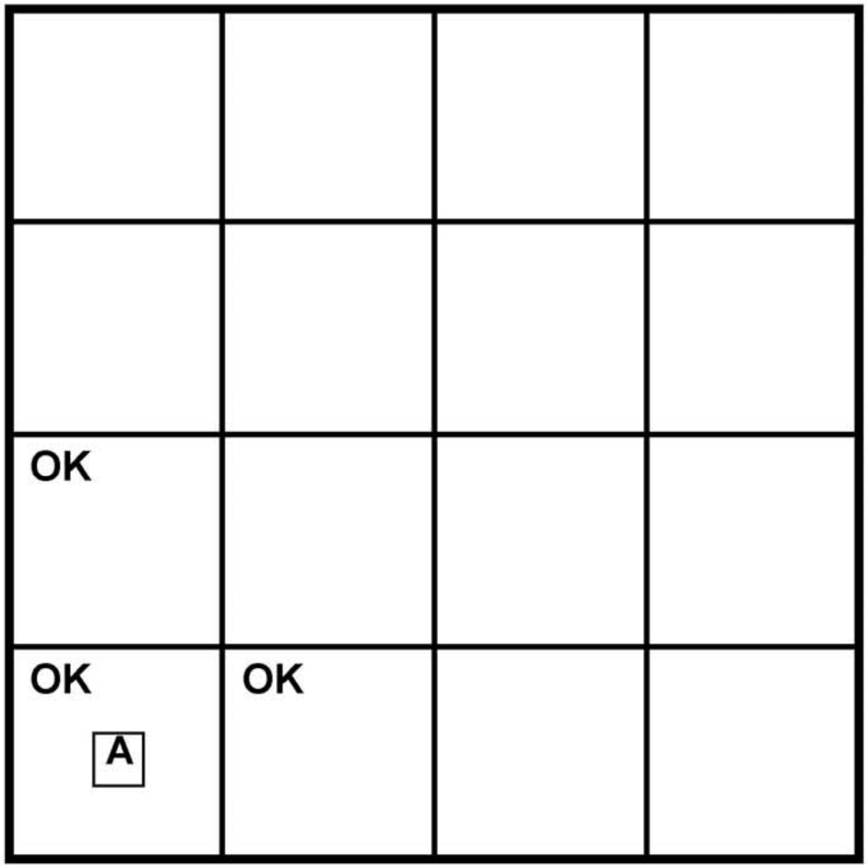
Episodic?? No—sequential at the level of actions

Static?? Yes—Wumpus and Pits do not move

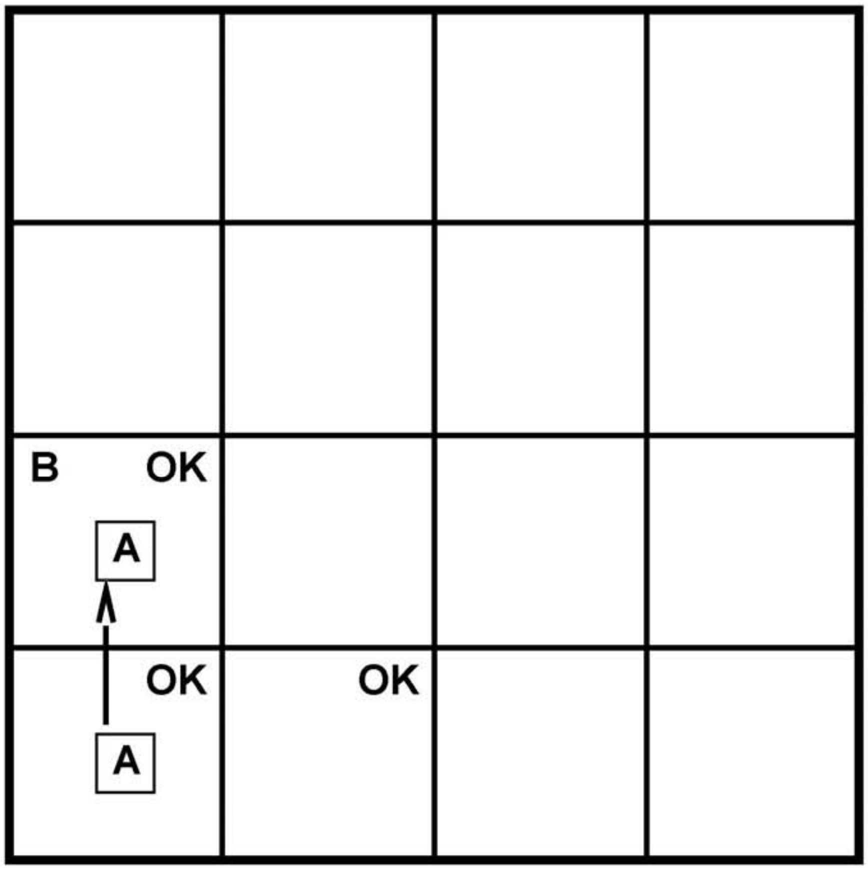
Discrete?? Yes

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

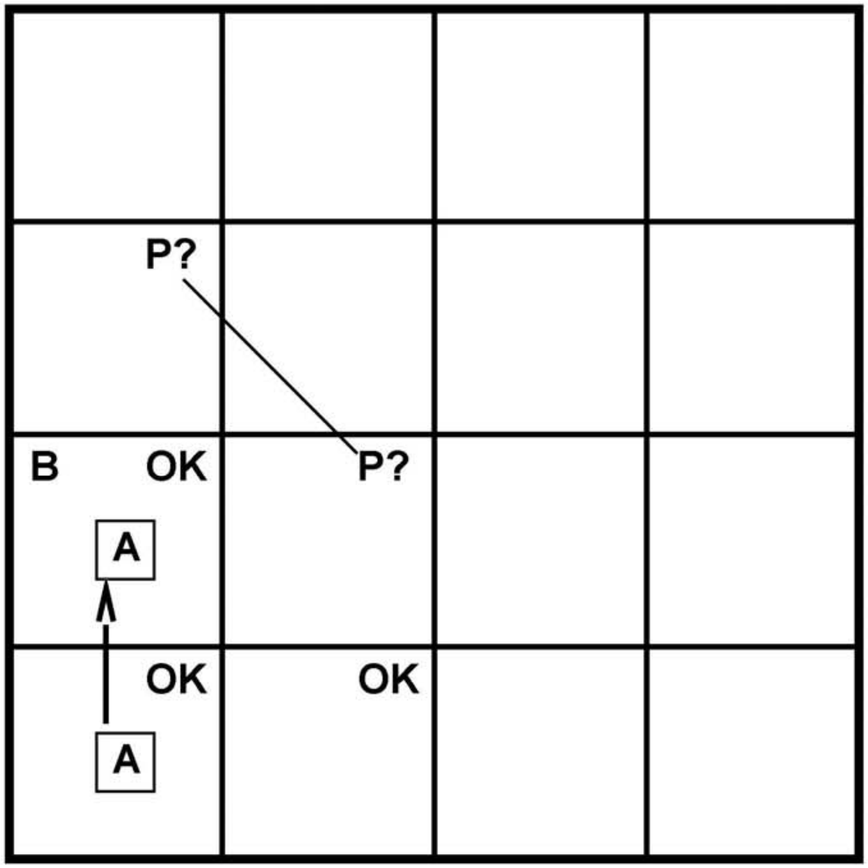
Exploring a wumpus world



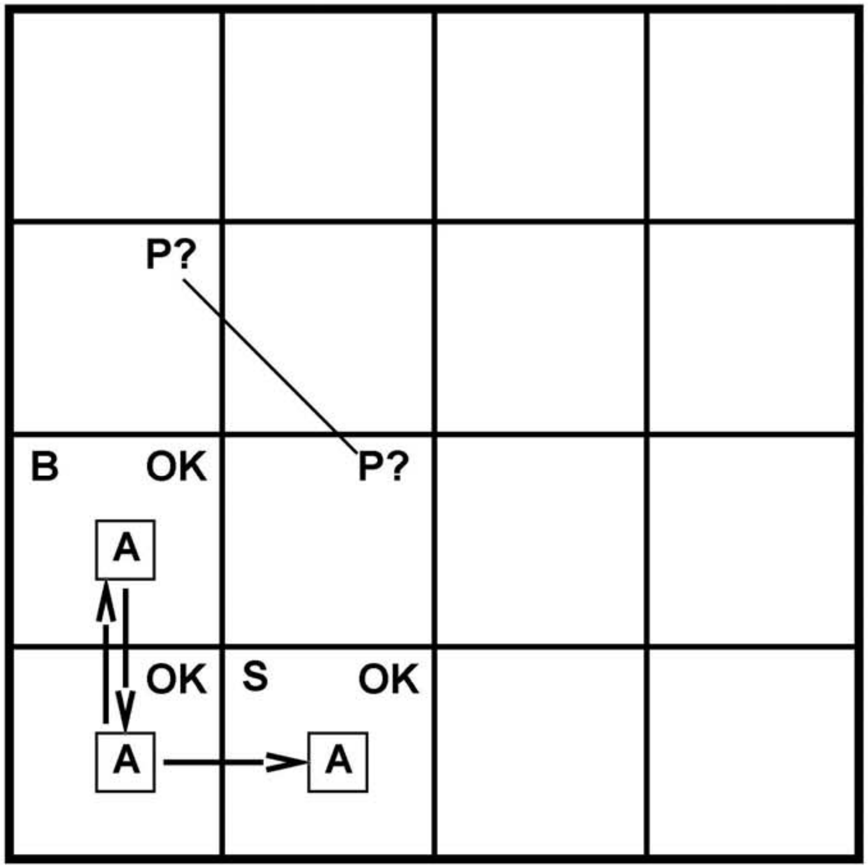
Exploring a wumpus world



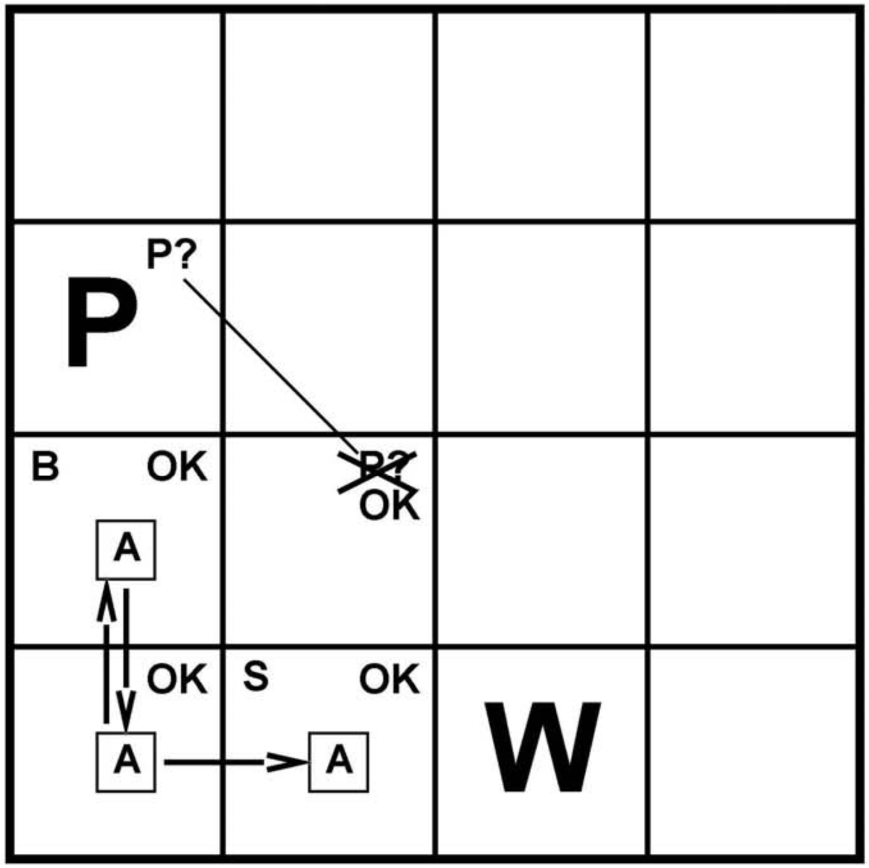
Exploring a wumpus world



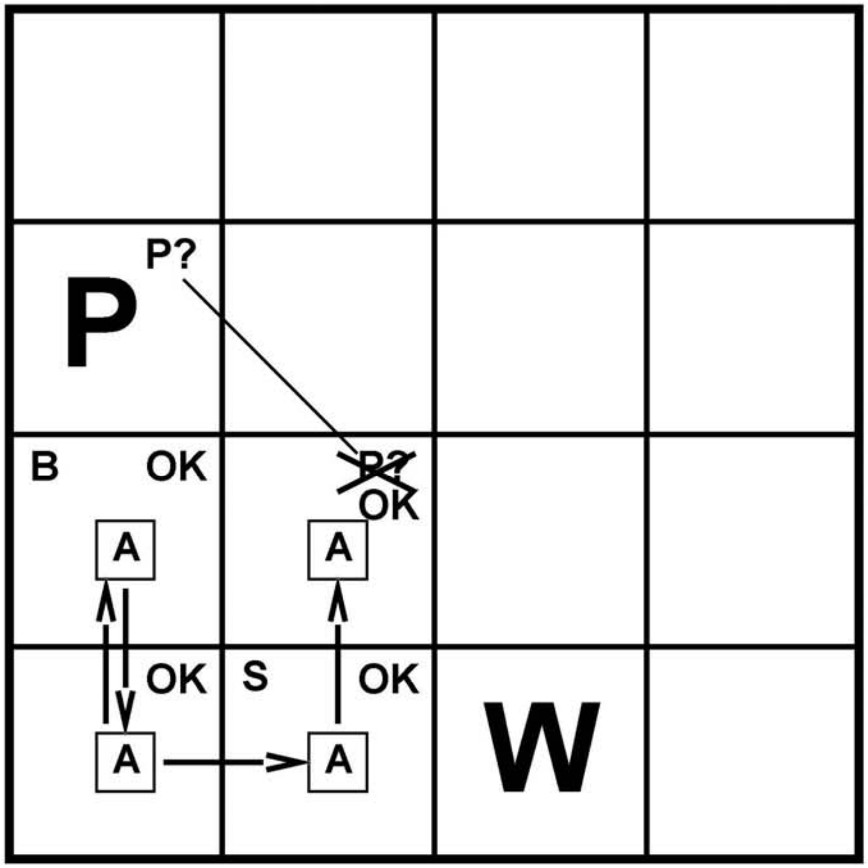
Exploring a wumpus world



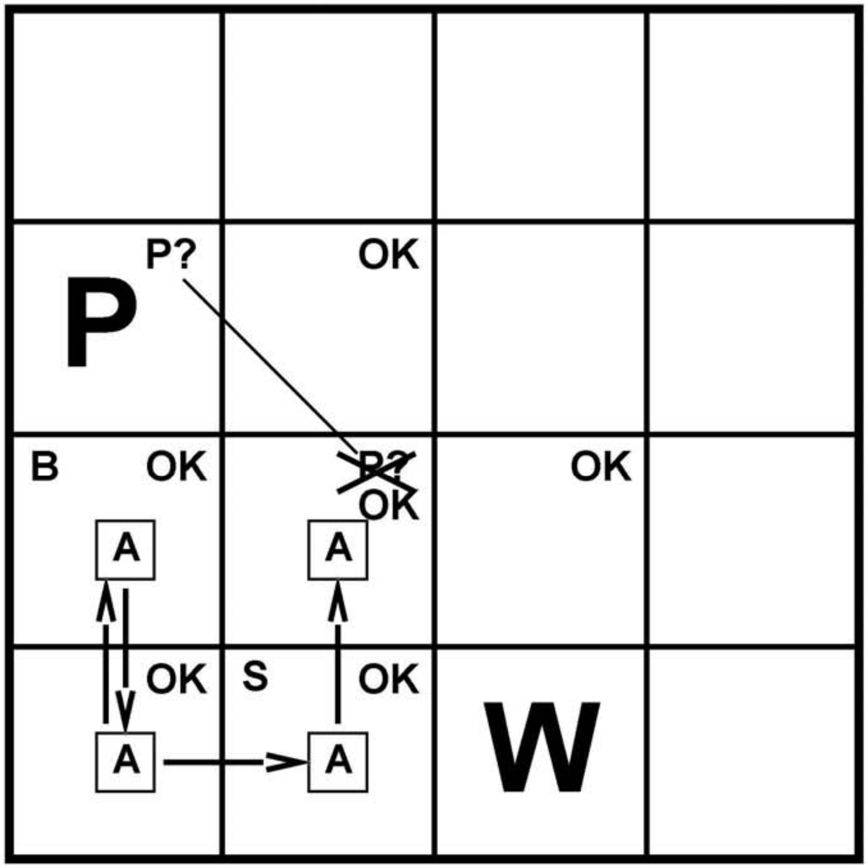
Exploring a wumpus world



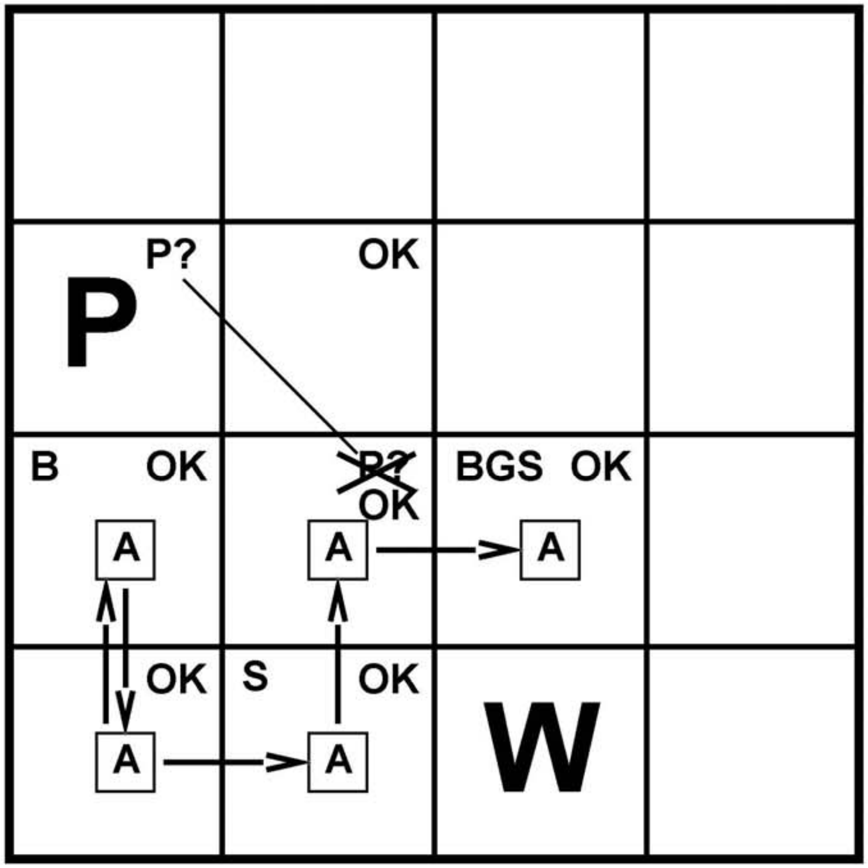
Exploring a wumpus world



Exploring a wumpus world



Exploring a wumpus world



Logic in general

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

Syntax defines the sentences in the language

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

E.g., the language of arithmetic

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

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

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

Entailment

Entailment means that one thing *follows from* another:

$$KB \models \alpha$$

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

E.g., the KB containing "the Giants won" and "the Reds won" entails "Either the Giants won or the Reds won"

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

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

Note: brains process *syntax* (of some sort)

Models

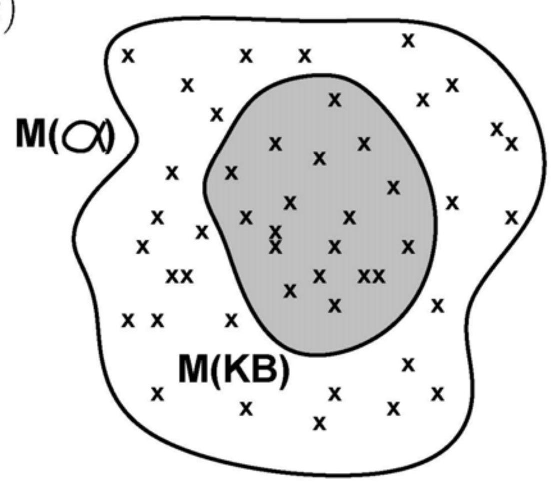
Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated

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

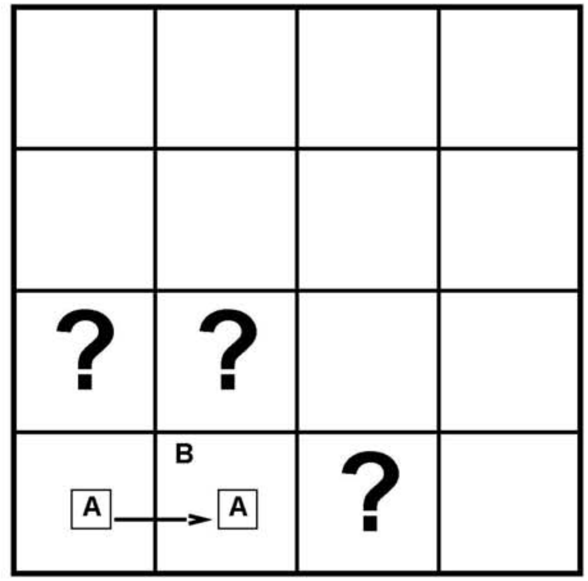
$M(\alpha)$ is the set of all models of α

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

E.g. $KB =$ Giants won and Reds won
$\alpha =$ Giants won



Entailment in the wumpus world

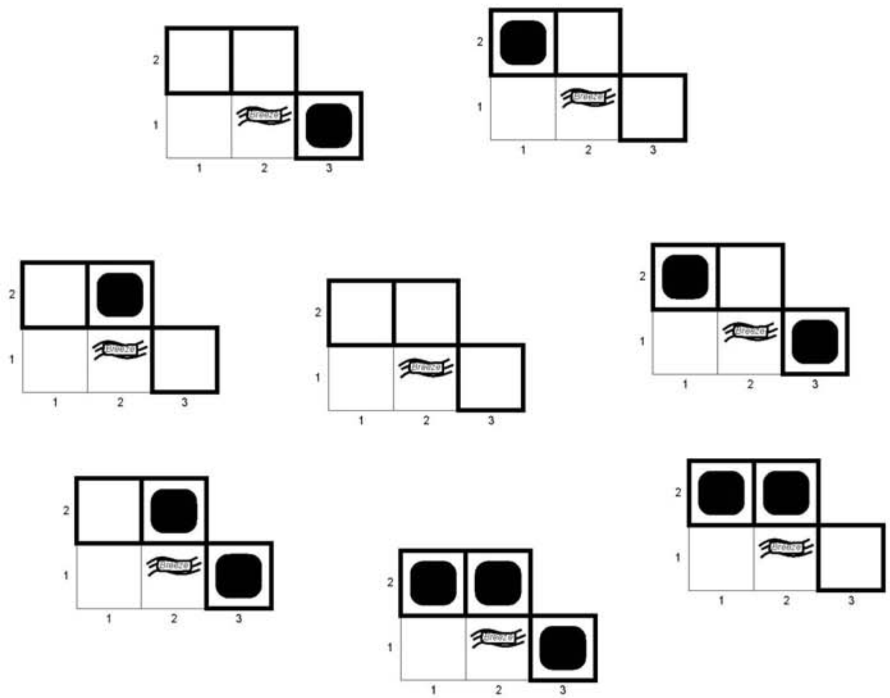


Situation after detecting nothing in [1,1], moving right, breeze in [2,1]

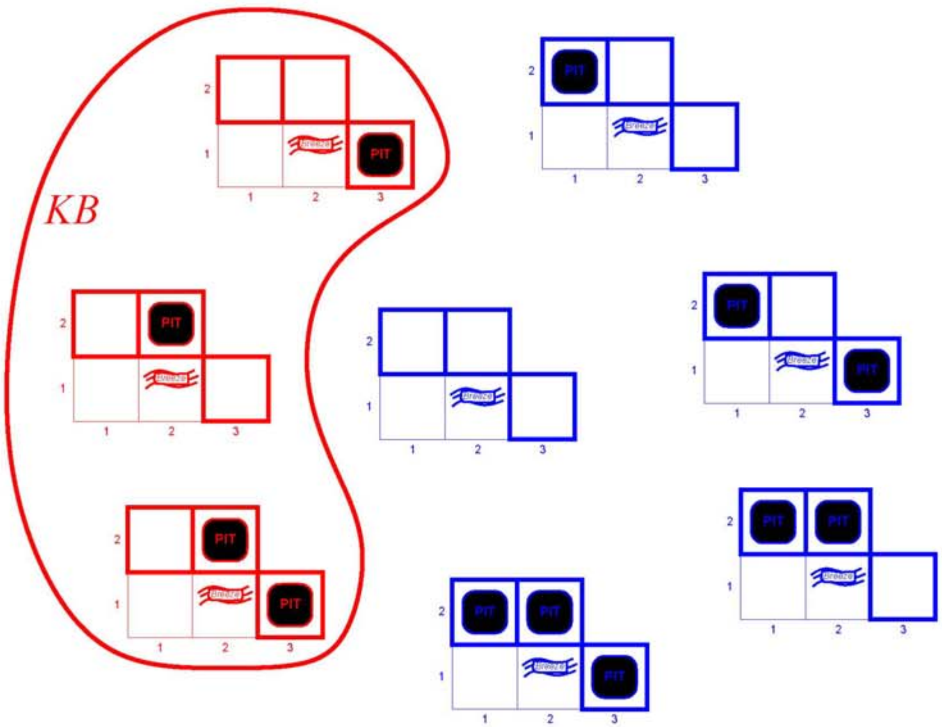
Consider possible models for ?s assuming only pits

3 Boolean choices $\Rightarrow$ 8 possible models

Wumpus models

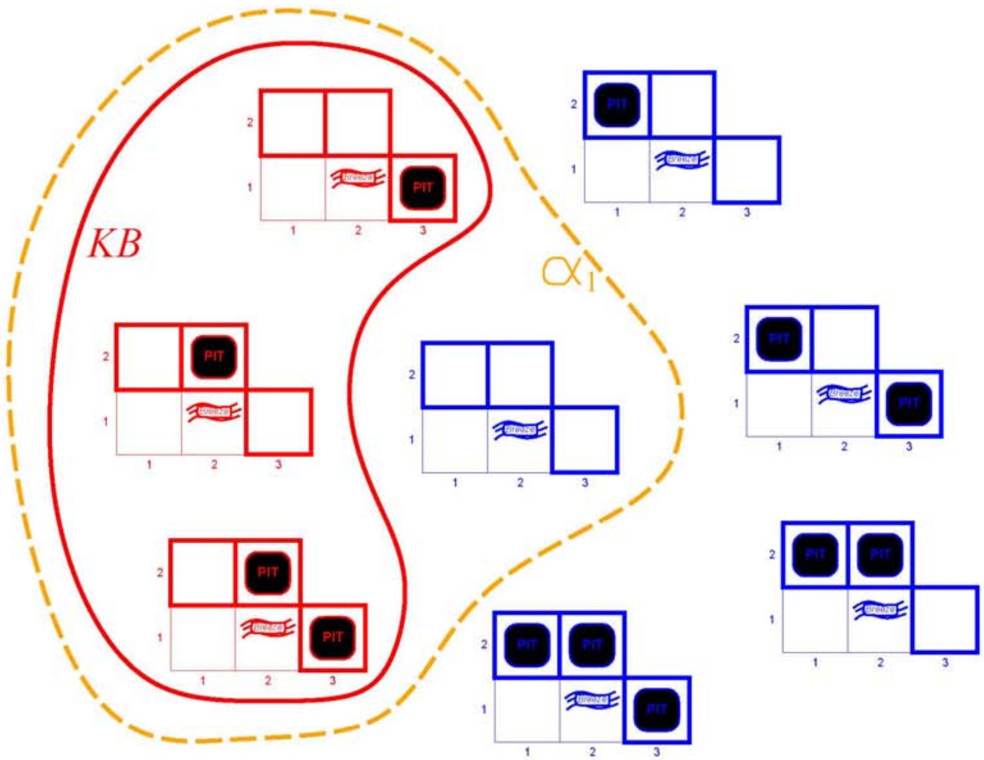


Wumpus models



KB = wumpus-world rules + observations

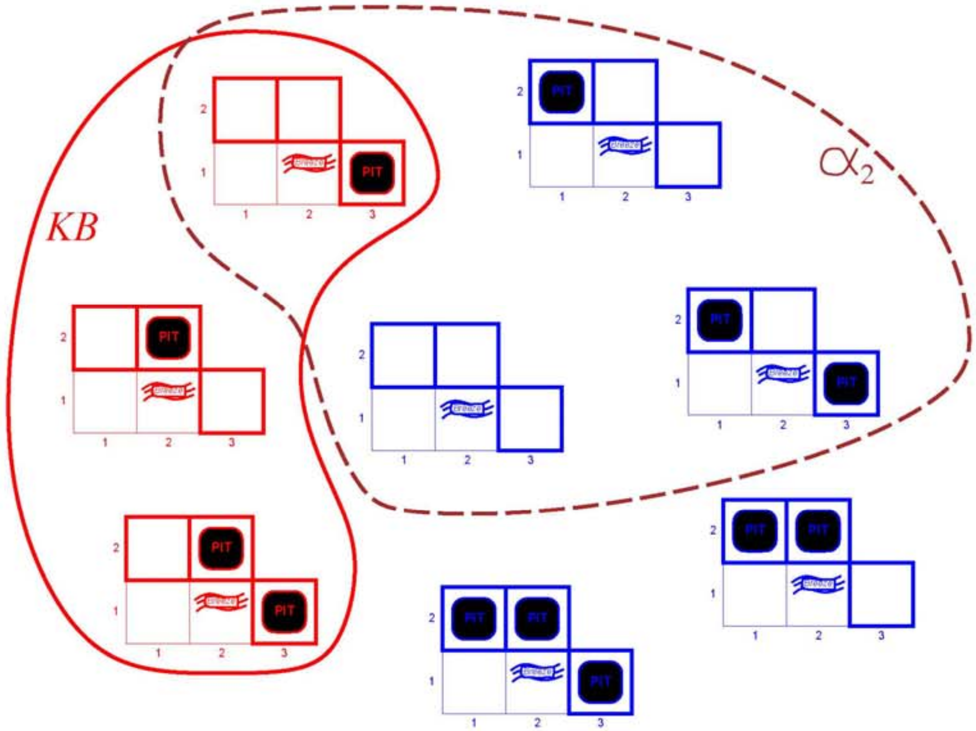
Wumpus models



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 $KB \vdash_i \alpha$

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

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

Rules for evaluating truth with respect to a model m:

$\neg S$	is true iff	S	is false		
$S_1 \wedge S_2$	is true iff	S_1	is true and	S_2	is true
$S_1 \vee S_2$	is true iff	S_1	is true or	S_2	is true
$S_1 \Rightarrow S_2$	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.,
$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = true \wedge (false \vee true) = true \wedge true = true$

Truth tables for connectives

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

Wumpus world sentences

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

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

"Pits cause breezes in adjacent squares"

Wumpus world sentences

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

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

"Pits cause breezes in adjacent squares"

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

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

Truth tables for inference

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	α_1
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*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*false*

Inference by enumeration

Depth-first enumeration of all models is sound and complete

function TT-ENTAILS?(KB, α) **returns** *true* or *false*

    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*))

Don't sweat the details: later we will see a much more efficient way of searching through model space!

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

Logical equivalence

Two sentences are logically equivalent iff true in same models:

$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

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

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., $A \land \neg A$

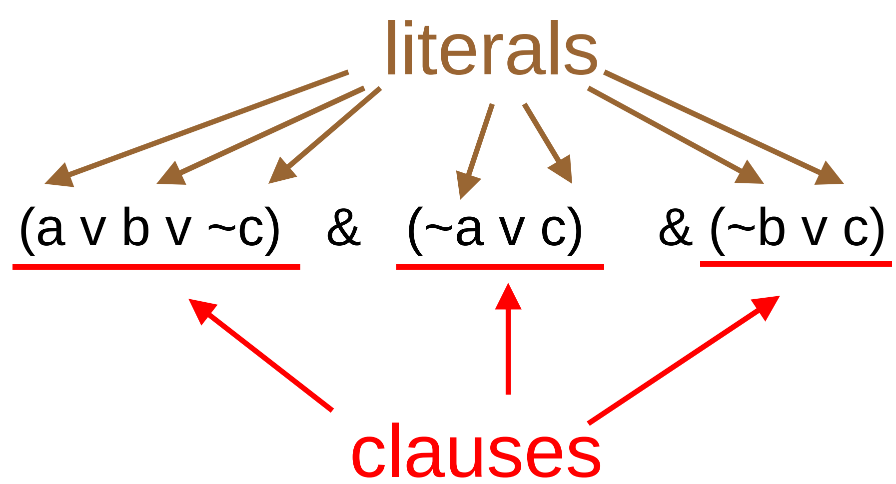
Satisfiability is connected to inference via the following:

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

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

Formal Computational Complexity

- SAT = Prototypical NP-complete problem:
  - Given a Boolean formula, is there a assignment of truth values to the Boolean variables that makes it true?
  - As hard as any problem where an answer can be *verified* in polynomial time
  - Still NP-complete if formulas are restricted to Conjunctive Normal Form:



Proof methods

Proof methods divide into (roughly) two kinds:

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

Model checking

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

Forward and backward chaining

Horn Form (restricted)

KB = *conjunction* of *Horn clauses*

Horn clause =

- ◊ proposition symbol; or
- ◊ (conjunction of symbols) $\Rightarrow$ symbol

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

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

Expert System for Automobile Diagnosis

Knowledge Base:

$GasInTank \wedge FuelLineOK \supset GasInEngine$

$GasInEngine \wedge GoodSpark \supset EngineRuns$

$PowerToPlugs \wedge PlugsClean \supset GoodSpark$

$BatteryCharged \wedge CablesOK \supset PowerToPlugs$

Observed:

$\neg$ EngineRuns,
GasInTank, PlugsClean, BatteryCharged

Prove:

$\neg FuelLineOK \vee \neg CablesOK$

Solution by Forward Chaining

Knowledge Base and Observations:

(~~¬ GasInTank~~ ∨ ~~¬ FuelLineOK~~ ∨ GasInEngine)

(~~¬ GasInEngine~~ ∨ ¬ GoodSpark ∨ ~~EngineRuns~~)

(¬ PowerToPlugs ∨ ~~¬ PlugsClean~~ ∨ ~~GoodSpark~~)

(~~¬ BatteryCharged~~ ∨ ~~¬ CablesOK~~ ∨ ~~PowerToPlugs~~)

(¬EngineRuns)

(GasInTank)

(PlugsClean)

(BatteryCharged)

Negation of Conclusion:

(FuelLineOK)

(CablesOK)

Resolution

Conjunctive Normal Form (CNF—universal)

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

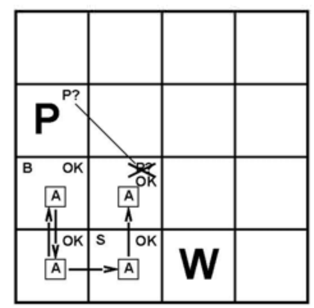
E.g., $(A \lor \neg B) \land (B \lor \neg C \lor \neg D)$

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

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

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

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



Resolution is sound and complete for propositional logic

Conversion to CNF

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

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

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

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 ($\vee$ over $\wedge$) and flatten:

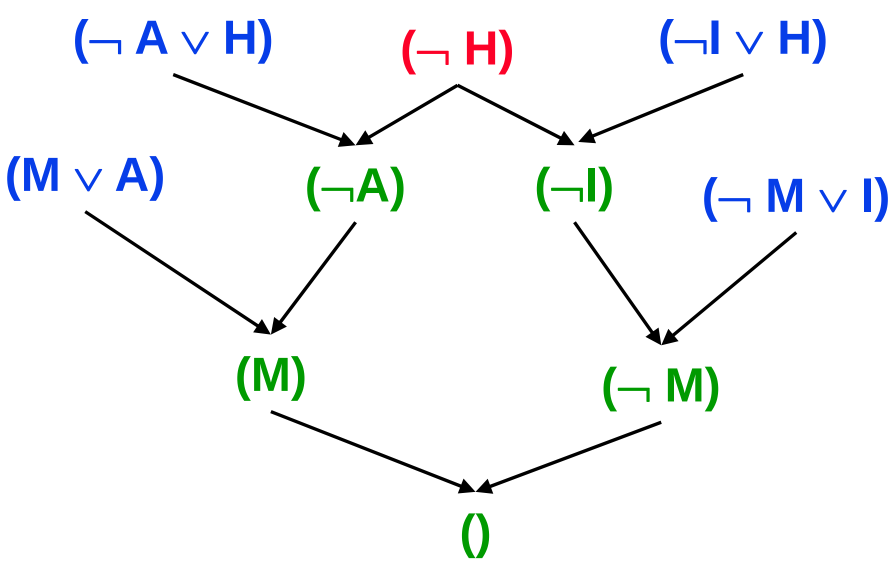
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Resolution Proof

DAG, where leaves are input clauses

Internal nodes are resolvants

Root is false (empty clause)



KB:

- If the unicorn is mythical, then it is immortal,
- if it is not mythical, it is an animal
- If the unicorn is either immortal or an animal, then it is horned.

Prove: the unicorn is horned.

THE CURIOUS INCIDENT OF THE DOG IN THE NIGHT

A racehorse was stolen from a stable, and a bookmaker Fitzroy Simpson was accused. Sherlock Holmes found the true thief by reasoning from the following premises:

1. The horse was stolen by Fitzroy or by the trainer, John Straker.
2. The thief entered the stable the night of the theft.
3. The dog barks if a stranger enters the stable.
4. Fitzroy was a stranger.
5. The dog did not bark.

Create a resolution refutation proof, using the propositions:

thief_fitzroy	thief_john
entered_fitzroy	entered_john
stranger_fitzroy	stranger_john
barks	

Efficient Local Search
for Satisfiability Testing

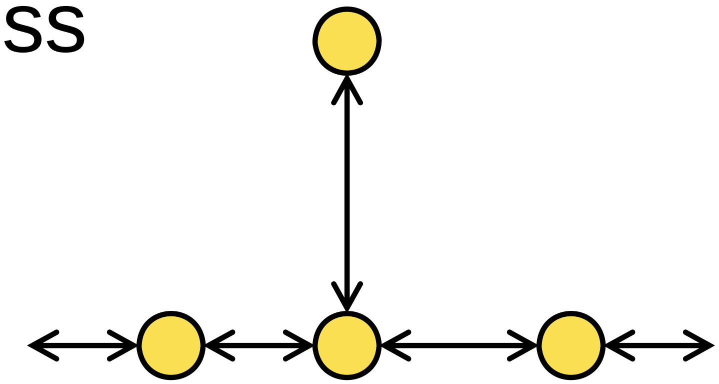
Greedy Local Search for SAT: GSAT

```
state = choose_start_state();
while ! GoalTest(state) do
        state := arg min { h(s) | s in Neighbors(state) }
end
return state;
```

- start = random truth assignment
- GoalTest = formula is satisfied
- h = number of false (unsatisfied) clauses
- neighbors = flip one variable (from true to false, or from false to true)

Smarter Noise Strategies

- For both random noise and simulated annealing, nearly all uphill moves are useless



- Can we find uphill moves that are more likely to be helpful?
- At least for SAT we can...

Random Walk for SAT

- Observation: if a clause is unsatisfied, at least one variable in the clause must be different in any global solution

(A v ~B v C)

- Suppose you randomly pick a variable from an unsatisfied clause to flip. What is the probability this was a good choice?

Random Walk for SAT

- Observation: if a clause is unsatisfied, at least one variable in the clause must be different in any global solution

(A v ~B v C)

- Suppose you randomly pick a variable from an unsatisfied clause to flip. What is the probability this was a good choice?

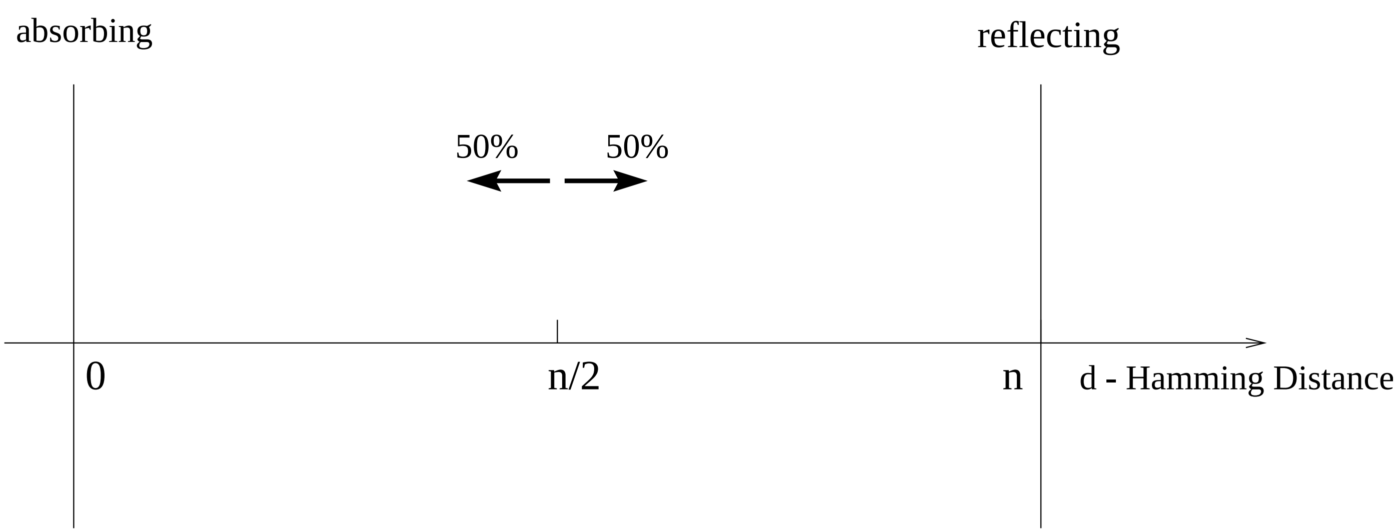
$$\Pr(\text{good choice}) \geq \frac{1}{\text{clause length}}$$

Random Walk Local Search

```
state = choose_start_state();
while ! GoalTest(state) do
    clause := random member { C | C is a clause of F and
                                  C is false in state }
    var := random member { x | x is a variable in clause }
    state[var] := 1 – state[var];
end
return state;
```

Properties of Random Walk

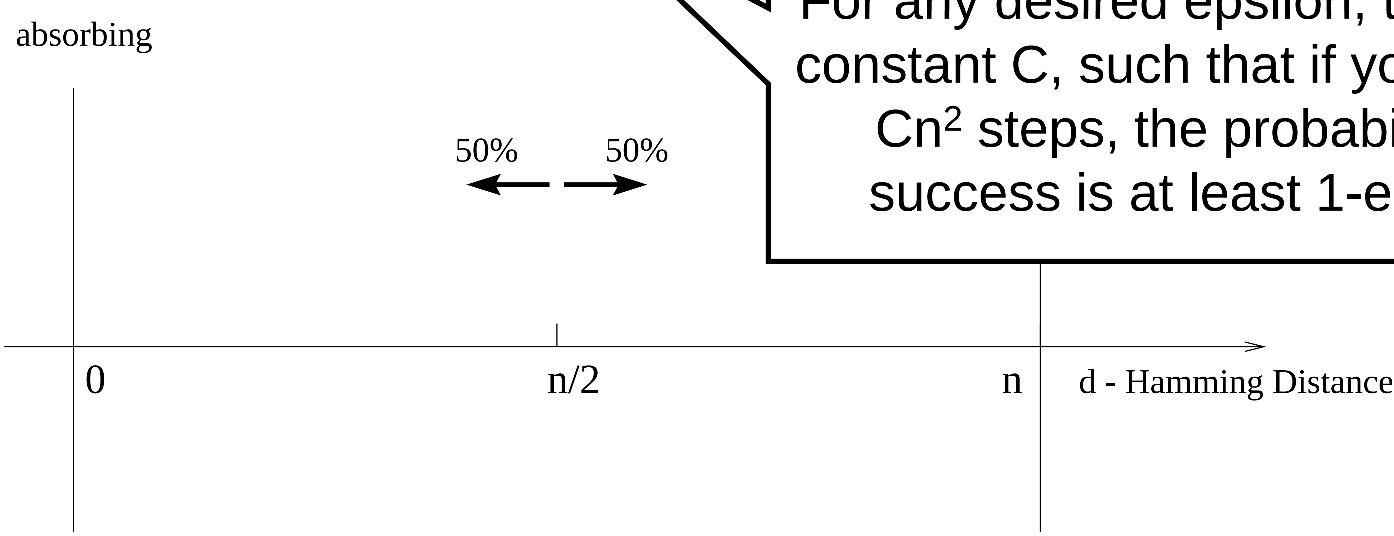
- If clause length = 2:
  - 50% chance of moving in the right direction
  - Converges to optimal with high probability in $O(n^2)$ time



Properties of Random Walk

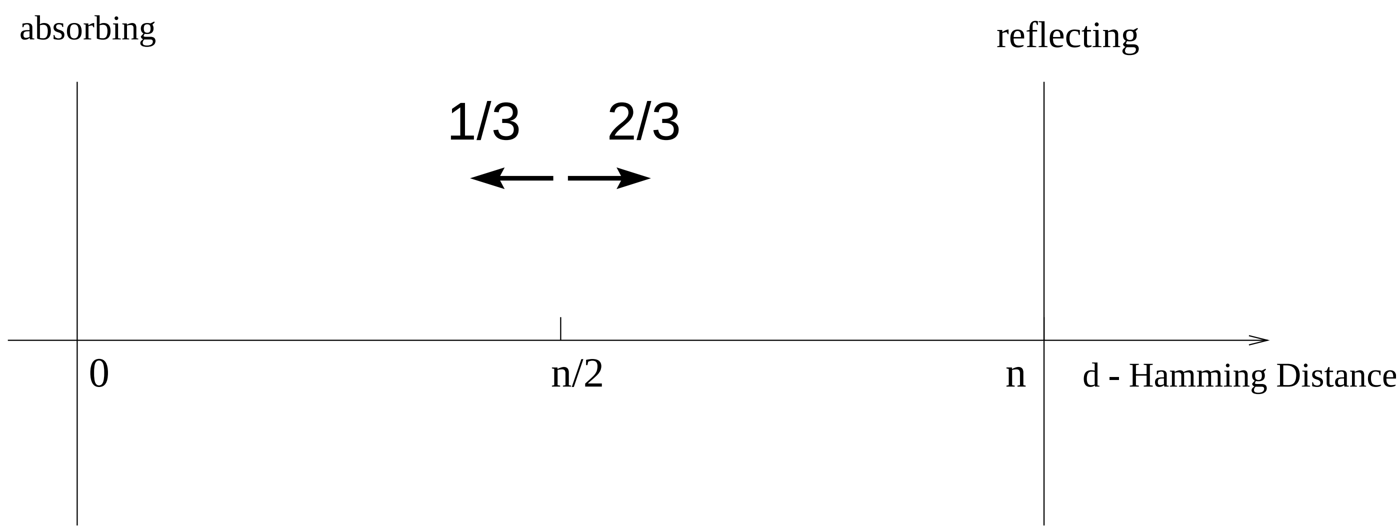
- If clause length = 2:
  - 50% chance of moving in the right direction
  - Converges to optimal with high probability in $O(n^2)$ time

For any desired epsilon, there is a constant C, such that if you run for Cn^2 steps, the probability of success is at least 1-epsilon



Properties of Random Walk

- If clause length = 3:
  - 1/3 chance of moving in the right direction
  - Exponential convergence
  - Compare pure noise: 1/(n-Hamming distance) chance of moving in the right direction
    - The closer you get to a solution, the more likely a noisy flip is bad



Greedy Random Walk

```
state = choose_start_state();
while ! GoalTest(state) do
    clause := random member { C | C is a clause of F and
                                  C is false in state };
    with probability noise do
        var := random member { x | x is a variable in clause };
    else
        var := arg x min { #unsat(s) | x is a variable in clause,
                                  s and state differ only on x};
    end
    state[var] := 1 – state[var];
end
return state;
```

Refining Greedy Random Walk

- Each flip
  - makes some false clauses become true
  - breaks some true clauses, that become false
- Suppose s1→s2 by flipping x. Then:

  #unsat(s2) = #unsat(s1) – make(s1,x) + break(s1,x)
- Idea 1: if a choice breaks nothing, it is very likely to be a good move
- Idea 2: near the solution, only the break count matters
  - the make count is usually 1

Walksat

```
state = random truth assignment;
while ! GoalTest(state) do
    clause := random member { C | C is false in state };
    for each x in clause do compute break[x];
    if exists x with break[x]=0 then var := x;
    else
        with probability noise do
            var := random member { x | x is in clause };
        else
            var := arg x min { break[x] | x is in clause };
    endif
    state[var] := 1 – state[var];
end
return state;
```

Put everything inside of a restart loop.
Parameters: noise, max_flips, max_runs

SAT Translation of N-Queens

- At least one queen each row:

  (Q11 v Q12 v Q13 v ... v Q18)

  (Q21 v Q22 v Q23 v ... v Q28)

  ...

  } $O(N^2)$ clauses

- No attacks:

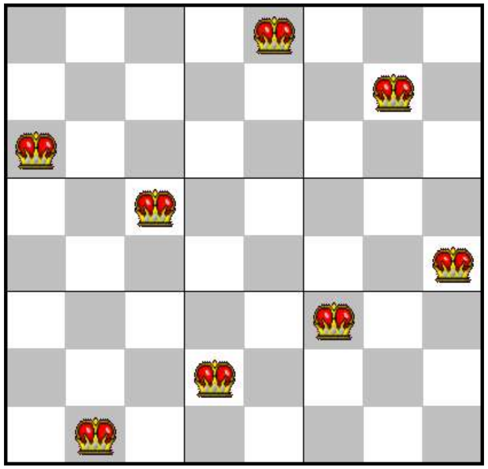
  (~Q11 v ~Q12)

  (~Q11 v ~Q22)

  (~Q11 v ~Q21)

  ...

  } $O(N^3)$ clauses



Demo:
Solving N-Queens with Walksat

Walksat Today

- Hard random 3-SAT: 100,000 vars, 15 minutes
  - Walksat (or slight variations) winner every year in "random formula" track of *International SAT Solver Competition*
  - Backtrack search methods: 700 variables
- Certain kinds of structured problems (graph coloring, Latin squares, n-queens, ...) ≈ 30,000 variables
  - But best systematic search routines better on certain other kinds of problems – e.g., verification
- Inspired huge body of research linking SAT testing to statistical physics (spin glasses)

Efficient Backtrack Search for Satisfiability Testing

Basic Backtrack Search for a Satisfying Model

```
Solve( F ): return Search(F, { });

Search( F, assigned ):
   if all variables in F are in assigned then
       if evaluate(F, assigned) then return assigned;
       else return FALSE;
   choose unassigned variable x;
   return      Search(F, assigned U {x=0}) ||
               Search(F, assigned U {x=1});
end;
```

State Space:

All partial or complete assignments of truth values to variables

Propagating Constraints

- Suppose formula contains

  (A v B v ~C)

  and we set A=0.
- What is the resulting constraint on the remaining variables B and C?

  (B v ~C)
- Suppose instead we set A=1. What is the resulting constraint on B and C?

  No constraint

Empty Clauses and Formulas

- Suppose a clause in F is shortened until it become empty. What does this mean about F and the partial assignment?

  F cannot be satisfied by any way of completing the assignment; must backtrack

- Suppose all the clauses in F disappear. What does this mean?

  F is satisfied by any completion of the partial assignment

Unit Propagation

- Suppose a clause in F is shortened to contain a single literal, such as

  (A)

  What should you do?

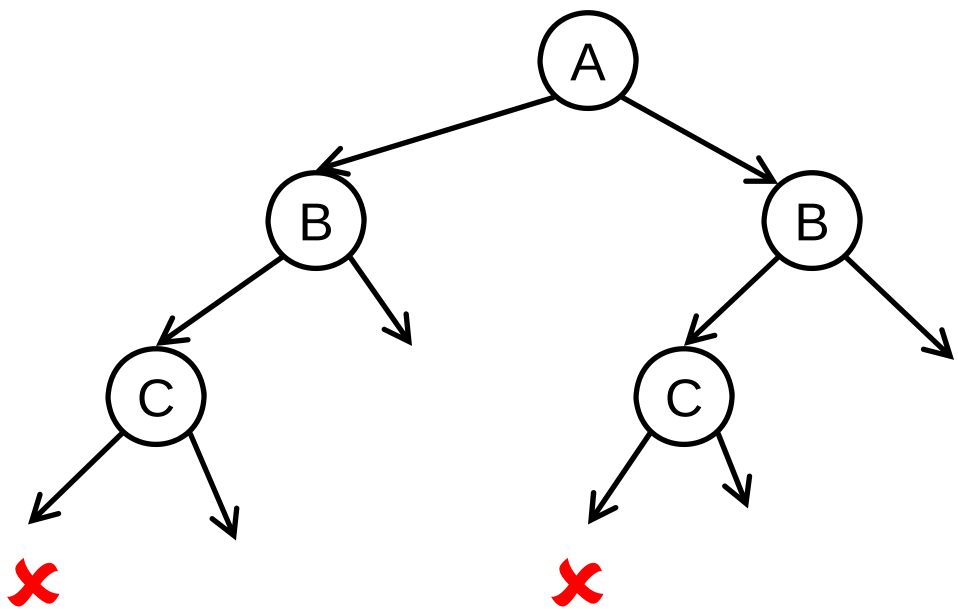
  Immediately add the literal to assigned. Repeat if another single-literal clause appears.
- Applying resolution where one clause is a single literal is called unit propagation

DPLL

```
DPLL( F, assigned ):
    while F has a unit clause (c) do
          assigned = assigned U {c};
          shorten clauses containing ~c;
          delete clauses containing c;
    end
    if F is empty then return assigned;
    if F contains an empty clause then return FALSE;
    choose an unassigned literal c; // variable and initial value
    return      Search(F U { (c) }, assigned)   ||
                Search(F U { (~c) }, assigned);
end;
```

Improving Efficiency: Clause Learning

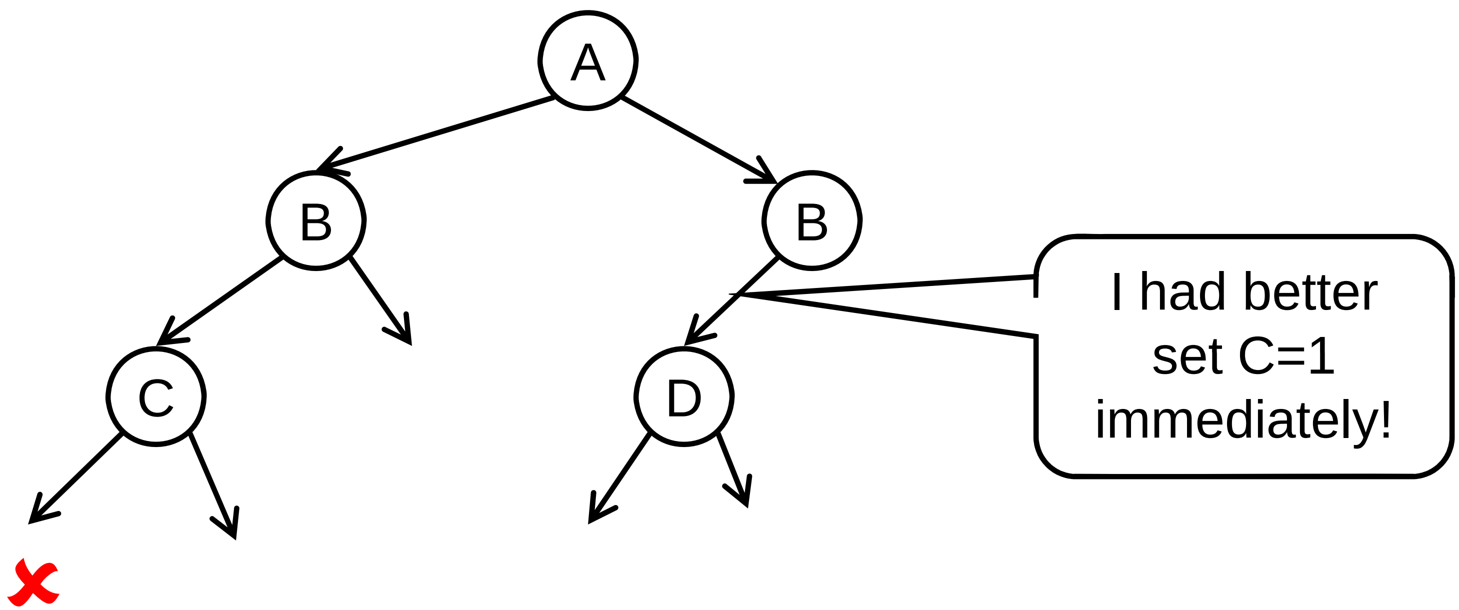
- Idea: backtrack search can repeatedly reach an empty clause (backtrack point) for the same reason



Example: Propagation from B=0 and C=0 leads to empty clause

Improving Efficiency: Clause Learning

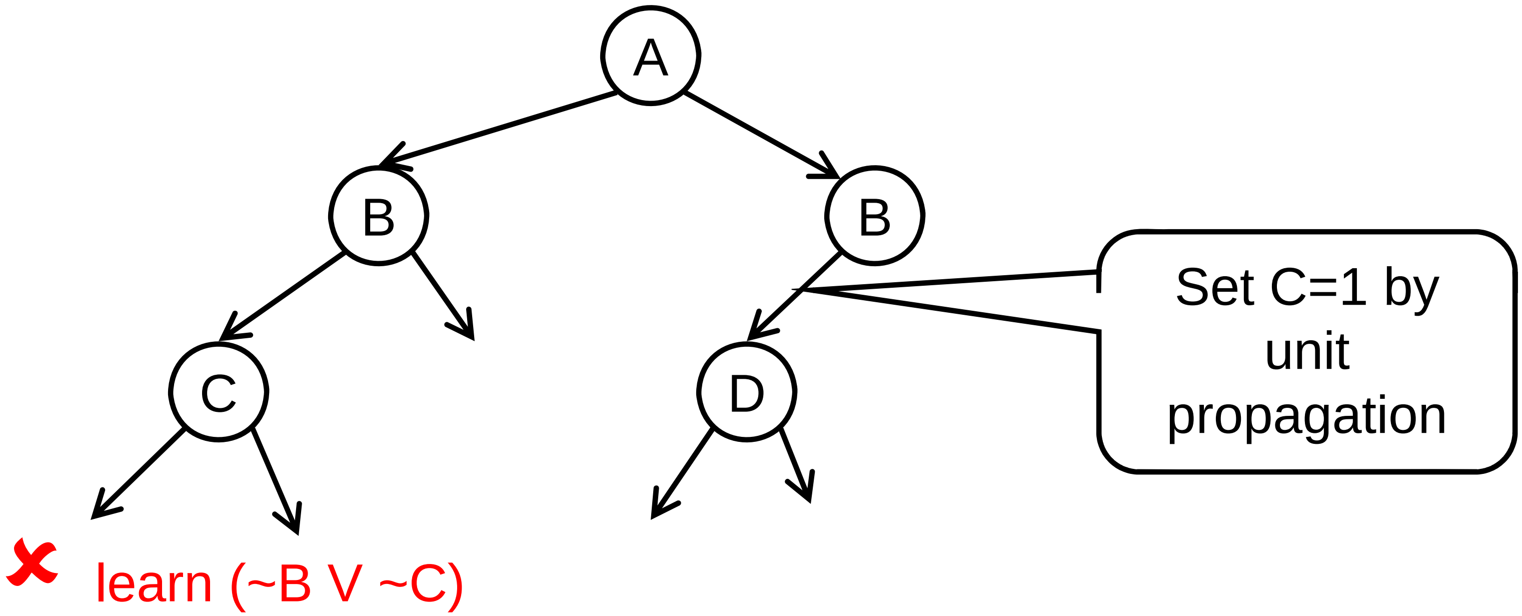
- If reason was remembered, then could avoid having to rediscover it



Example: Propagation from B=0 and C=0 leads to empty clause

Improving Efficiency: Clause Learning

- The reason can be remembered by adding a new learned clause to the formula



Example: Propagation from B=0 and C=0 leads to empty clause

Scaling Up

- Clause learning greatly enhances the power of unit propagation
- Tradeoff: memory needed for the learned clauses, time needed to check if they cause propagations
- Clever data structures enable modern SAT solvers to manage millions of learned clauses efficiently

What is BIG?

Consider a real world Boolean Satisfiability (SAT) problem

The instance bmc-ibm-6.cnf, IBM LSU 1997:

```
p cnf .
−1 7 0
−1 6 0
−1 5 0
−1 −4 0
−1 3 0
−1 2 0
−1 −8 0
−9 15 0
−9 14 0
−9 13 0
−9 −12 0
−9 11 0
−9 10 0
−9 −16 0
−17 23 0
−17 22 0
```

I.e., ((not x_1) or x_7)
((not x_1) or x_6)
etc.

x_1, x_2, x_3, etc. our Boolean variables
(set to True or False)

Set x_1 to False ??

10 pages later:

⋮

185 −9 0
185 −1 0
177 169 161 153 145 137 129 121 113 105 97
89 81 73 65 57 49 41
33 25 17 9 1 −185 0
186 −187 0
186 −188 0

…

I.e., (x_177 or x_169 or x_161 or x_153 …
x_33 or x_25 or x_17 or x_9 or x_1 or (not x_185))

clauses / constraints are getting more interesting…

Note x_1 …

4000 pages later:

10236 −10050 0
10236 −10051 0
10236 −10235 0
10008 10009 10010 10011 10012 10013 10014
10015 10016 10017 10018 10019 10020 10021
10022 10023 10024 10025 10026 10027 10028
10029 10030 10031 10032 10033 10034 10035
10036 10037 10086 10087 10088 10089 10090
10091 10092 10093 10094 10095 10096 10097
10098 10099 10100 10101 10102 10103 10104
10105 10106 10107 10108 −55 −54 53 −52 −51 50
10047 10048 10049 10050 10051 10235 −10236 0
10237 −10008 0
10237 −10009 0
10237 −10010 0

…

Finally, 15,000 pages later:

$-7\ 260\ 0$
$7\ -260\ 0$
$1072\ 1070\ 0$
$-15\ -14\ -13\ -12\ -11\ -10\ 0$
$-15\ -14\ -13\ -12\ -11\ 10\ 0$
$-15\ -14\ -13\ -12\ 11\ -10\ 0$
$-15\ -14\ -13\ -12\ 11\ 10\ 0$
$-7\ -6\ -5\ -4\ -3\ -2\ 0$
$-7\ -6\ -5\ -4\ -3\ 2\ 0$
$-7\ -6\ -5\ -4\ 3\ -2\ 0$
$-7\ -6\ -5\ -4\ 3\ 2\ 0$
$185\ 0$

Search space of truth assignments:

HOW?

$$2^{50000} \approx 3.160699437 \cdot 10^{15051}$$

Current SAT solvers solve this instance in approx. 1 minute!

Demo: SatPlan

Progress in SAT Solvers

Instance	Posit' 94	Grasp' 96	Sato' 98	Chaff' 01
ssa2670-136	40,66s	1,2s	0,95s	0,02s
bf1355-638	1805,21s	0,11s	0,04s	0,01s
pret150_25	>3000s	0,21s	0,09s	0,01s
dubois100	>3000s	11,85s	0,08s	0,01s
aim200-2_0-no-1	>3000s	0,01s	0s	0s
2dlx_..._bug005	>3000s	>3000s	>3000s	2,9s
c6288	>3000s	>3000s	>3000s	>3000s

Source: Marques Silva 2002