Logic in AI

Chapter 7

Mausam

(Based on slides of Dan Weld, Stuart Russell, Subbarao Kambhampati, Dieter Fox, Henry Kautz…)

Logic: another thing that penguins aren't very good at.

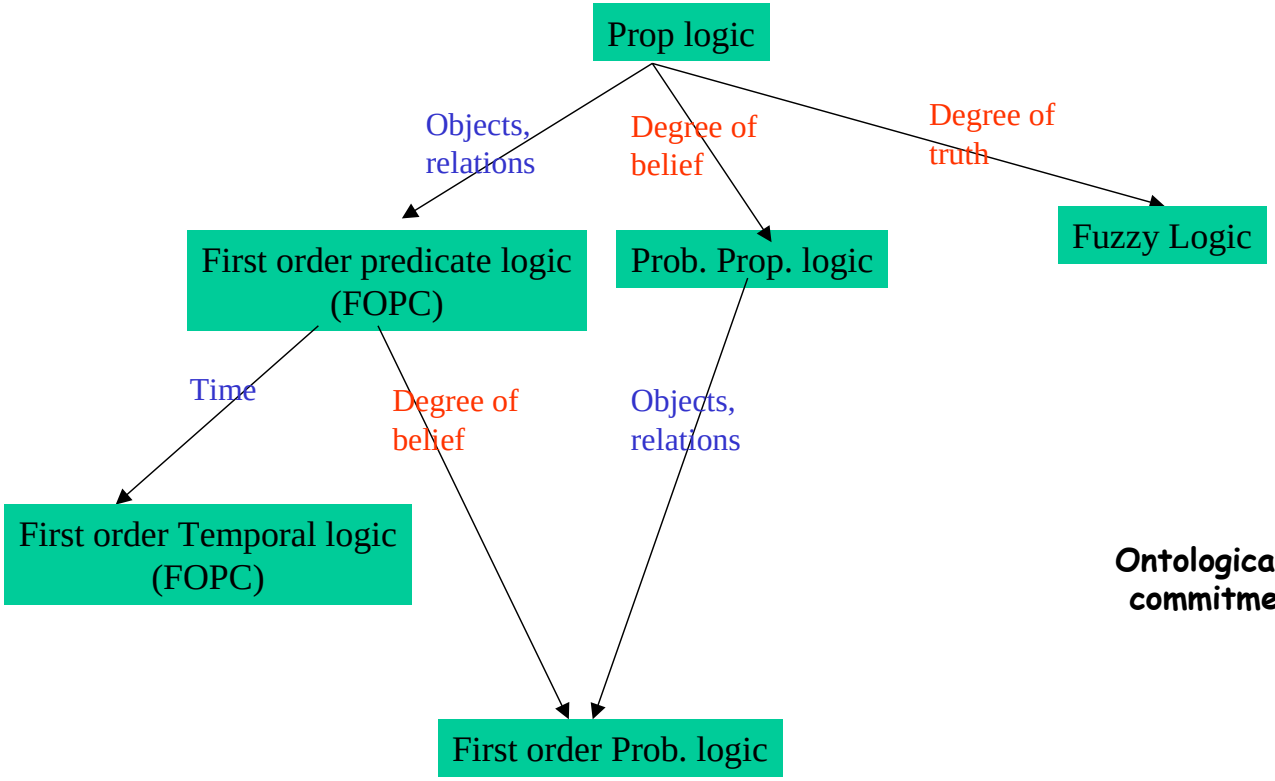
Knowledge Representation

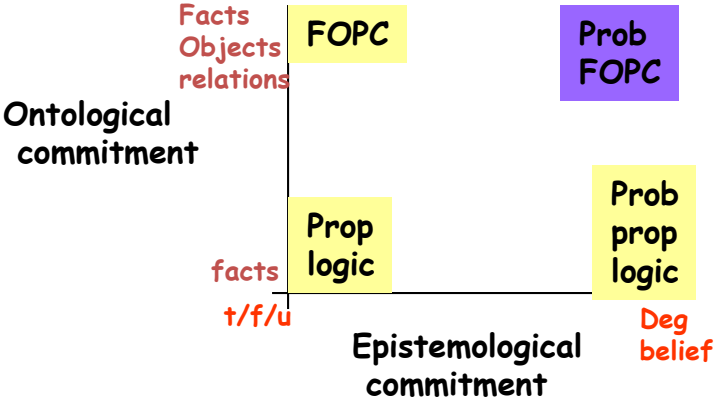
- *represent knowledge about the world in a manner that facilitates inferencing (i.e. drawing conclusions) from knowledge.*

- Example: Arithmetic logic
  - x >= 5

- In AI: typically based on
  - Logic
  - Probability
  - Logic and Probability

Common KR Languages





KR Languages

- Propositional Logic
- Predicate Calculus
- Frame Systems
- Rules with Certainty Factors
- Bayesian Belief Networks
- Influence Diagrams
- Ontologies
- Semantic Networks
- Concept Description Languages
- Non-monotonic Logic

Basic Idea of Logic

- By starting with true assumptions, you can deduce true conclusions.

Truth

- Francis Bacon (1561-1626)

No pleasure is comparable to the standing upon the vantage-ground of truth.

- Thomas Henry Huxley (1825-1895)

Irrationally held truths may be more harmful than reasoned errors.

- John Keats (1795-1821)

Beauty is truth, truth beauty; that is all ye know on earth, and all ye need to know.

- Blaise Pascal (1623-1662)

We know the truth, not only by the reason, but also by the heart.

- François Rabelais (c. 1490-1553)

Speak the truth and shame the Devil.

- Daniel Webster (1782-1852)

There is nothing so powerful as truth, and often nothing so strange.

Components of KR

- Syntax: defines the sentences in the language
- Semantics: defines the “meaning” to sentences
- Inference Procedure
  - Algorithm
  - Sound?
  - Complete?
  - Complexity
- Knowledge Base

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

Propositional Logic

- Syntax
  - Atomic sentences: P, Q, …
  - Connectives: $\wedge$, $\vee$, $\neg$, $\rightarrow$
- Semantics
  - Truth Tables
- Inference
  - Modus Ponens
  - Resolution
  - DPLL
  - GSAT

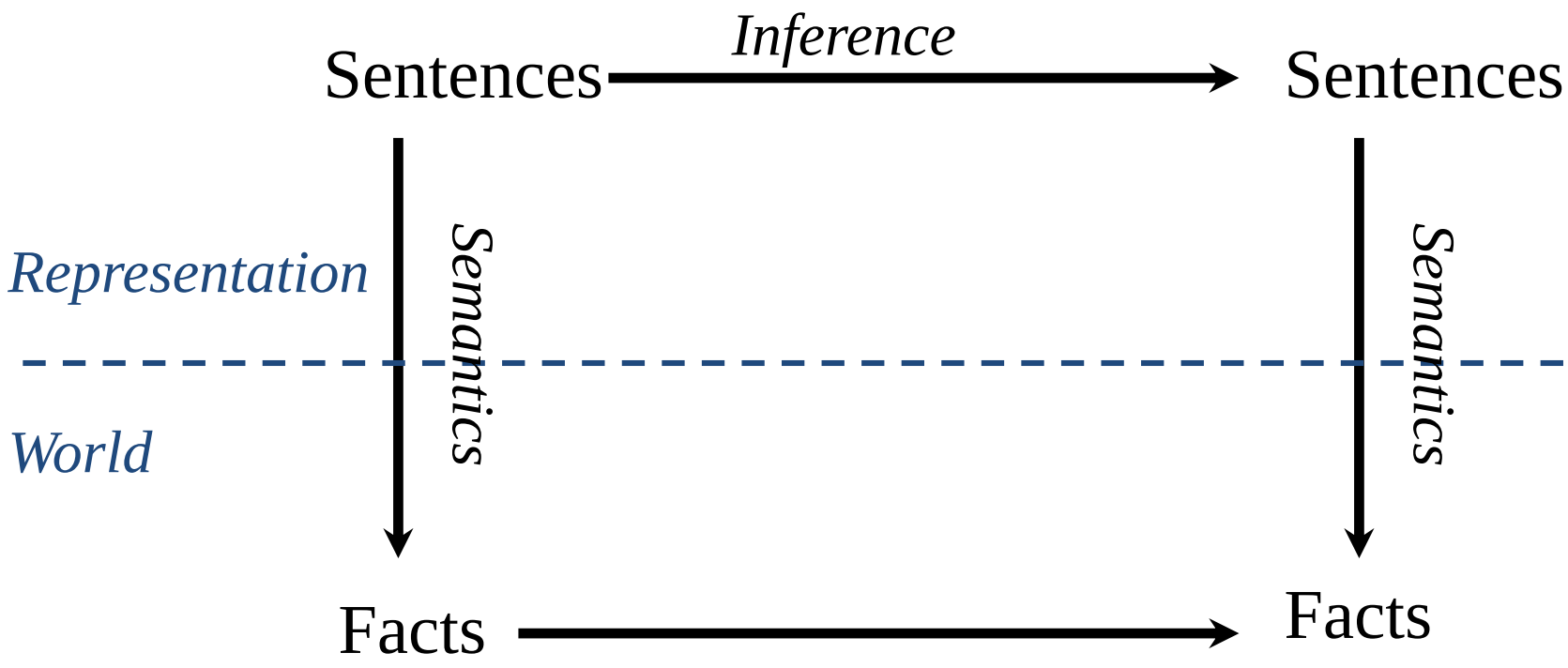
Propositional Logic: Syntax

- Atoms
  - P, Q, R, …
- Literals
  - P, $\neg$P
- Sentences
  - Any literal is a sentence
  - If S is a sentence
    - Then (S $\wedge$ S) is a sentence
    - Then (S $\vee$ S) is a sentence
- Conveniences

  P $\rightarrow$ Q same as $\neg$P $\vee$ Q

Semantics

- ***Syntax***: which arrangements of symbols are *legal*
  - (Def "sentences")
- ***Semantics***: what the symbols *mean* in the world
  - (Mapping between symbols and worlds)



Propositional Logic: **SEMANTICS**

- "Interpretation" (or "possible world")
  - Assignment to each variable either T or F
  - Assignment of T or F to each connective via defns

		Q	
		T	F
P	T	T	F
	F	F	F

$P \wedge Q$

		Q	
		T	F
P	T	T	T
	F	T	F

$P \vee Q$

Satisfiability, Validity, & Entailment

- S is **satisfiable** if it is true in ***some*** world
- S is **unsatisfiable** if it is false in ***all*** worlds
- S is **valid** if it is true in ***all*** worlds
- S1 **entails** S2 if ***wherever*** S1 is true S2 is also true

Examples

$P \rightarrow Q$

$R \rightarrow \neg R$

$S \wedge (W \wedge \neg S)$

$T \vee \neg T$

$X \rightarrow X$

Notation

$\Rightarrow$

$\supset$ } **Implication** (syntactic symbol)

$\rightarrow$

$\vdash$ **Proves:** S1 $\vdash_{ie}$ S2 if `ie' algorithm says `S2' from S1

$\models$ **Entails:** S1 $\models$ S2 if wherever S1 is true S2 is also true

- **Sound** $\vdash \rightarrow \models$
- **Complete** $\models \rightarrow \vdash$
- *(all truth & nothing but the truth)*

Reasoning Tasks

- Model finding

  KB = background knowledge

  S = description of problem

  Show (KB $\wedge$ S) is satisfiable

  A kind of constraint satisfaction

- Deduction

  S = question

  Prove that KB $\models$ S

  Two approaches:

  - Rules to derive new formulas from old (inference)
  - Show (KB $\wedge \neg$ S) is unsatisfiable

Special Syntactic Forms

- General Form:

  $((q \wedge \neg r) \rightarrow s)) \wedge \neg (s \wedge t)$

- Conjunction Normal Form (CNF)

  $(\neg q \vee r \vee s) \wedge (\neg s \vee \neg t)$

  Set notation: $\{ (\neg q, r, s), (\neg s, \neg t) \}$

  empty clause () = *false*

- Binary clauses: 1 or 2 literals per clause

  $(\neg q \vee r)$ $(\neg s \vee \neg t)$

- Horn clauses: 0 or 1 positive literal per clause

  $(\neg q \vee \neg r \vee s)$ $(\neg s \vee \neg t)$

  $(q \wedge r) \rightarrow s$ $(s \wedge t) \rightarrow$ *false*

Propositional Logic: **Inference**

A *mechanical* process for computing new sentences

1. Backward & Forward Chaining
2. Resolution (Proof by Contradiction)
3. SAT
    1. Davis Putnam
    2. WalkSat

Inference 1: Forward Chaining

Forward Chaining

Based on rule of ***modus ponens***

If know $P_1, \ldots, P_n$ & know $(P_1 \wedge \ldots \wedge P_n) \rightarrow Q$

Then can conclude Q

Backward Chaining: search

start from the query and go backwards

Analysis

- Sound?
- Complete?

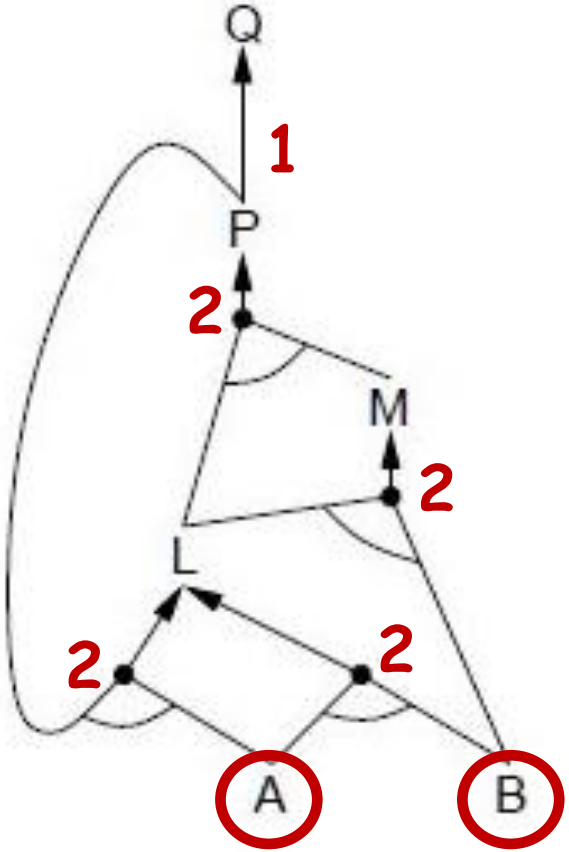
Can you prove

$$\{\ \} \models Q \vee \neg Q$$

- If KB has only Horn clauses & query is a single literal
  - Forward Chaining is complete
  - Runs linear in the size of the KB

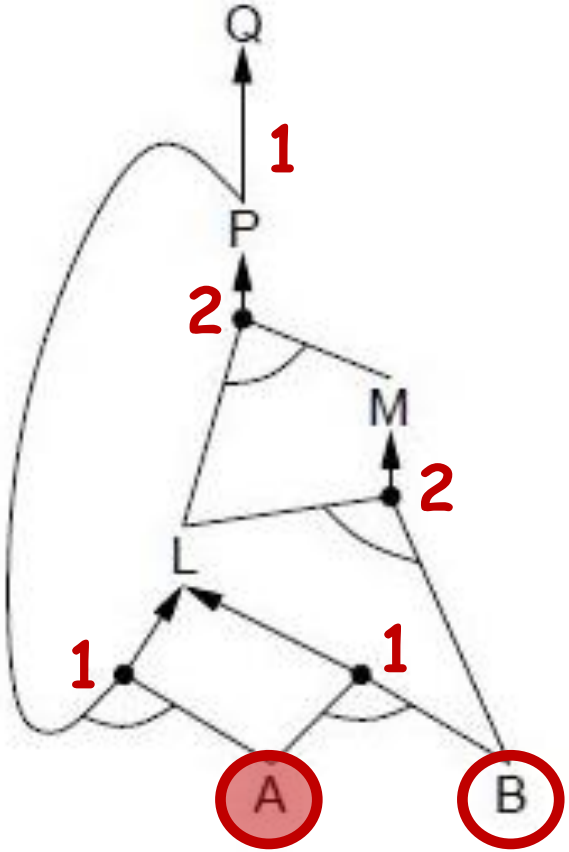
Example

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



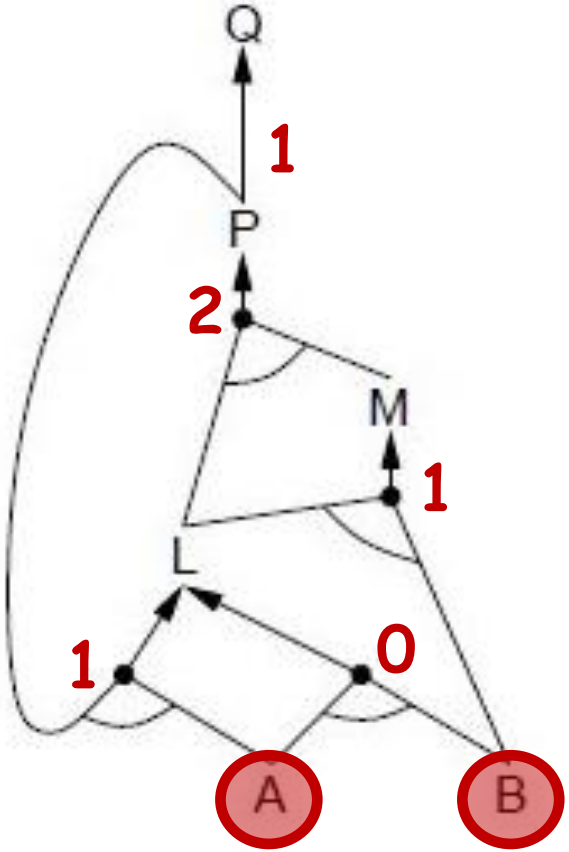
Example

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



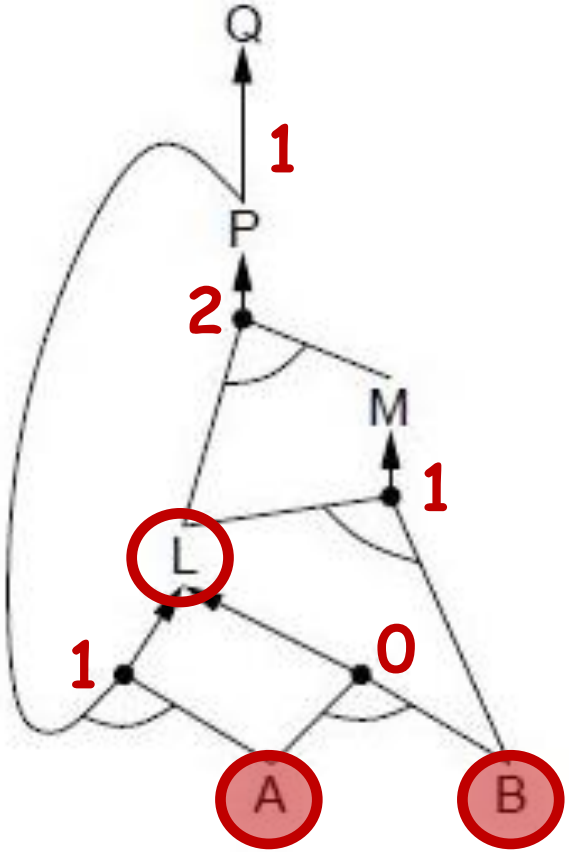
Example

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



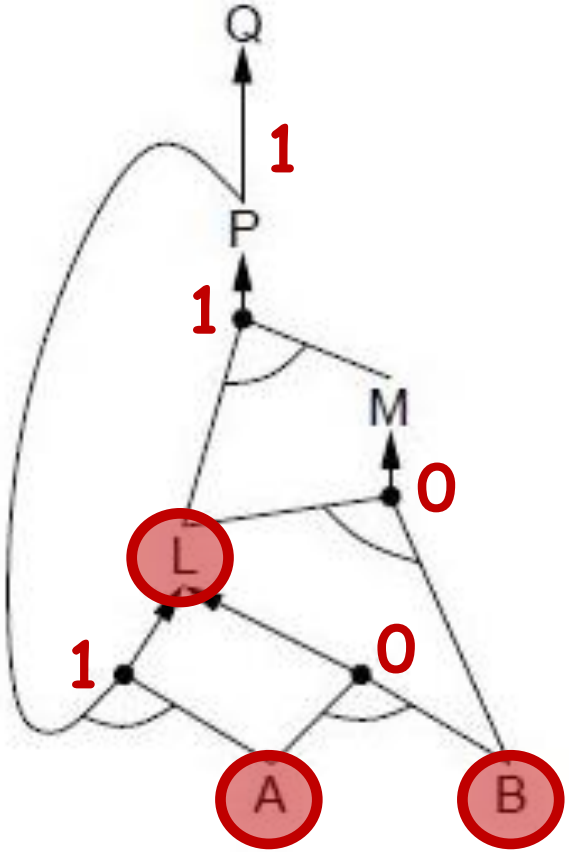
Example

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



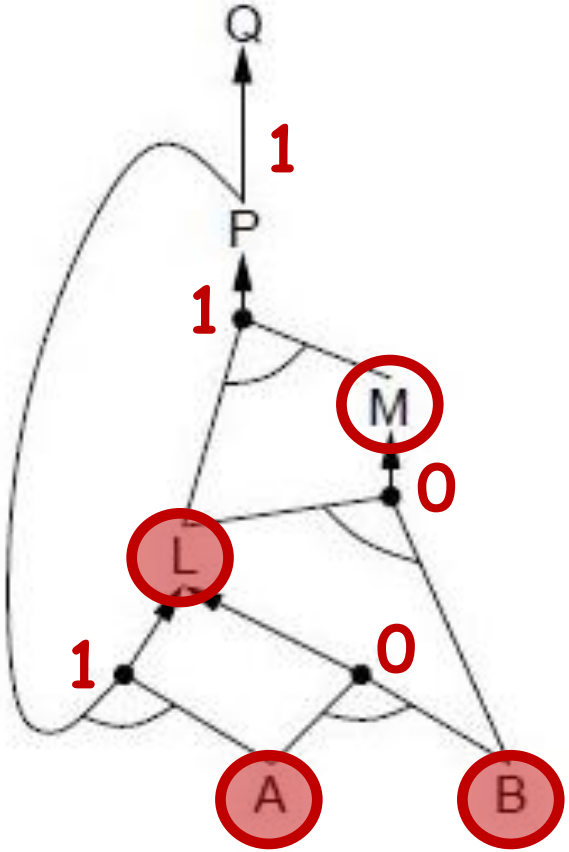
Example

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



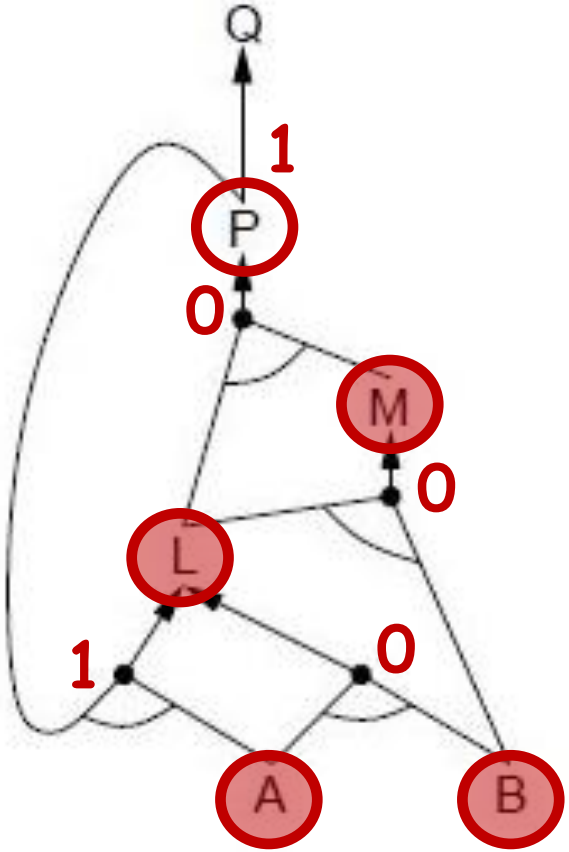
Example

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



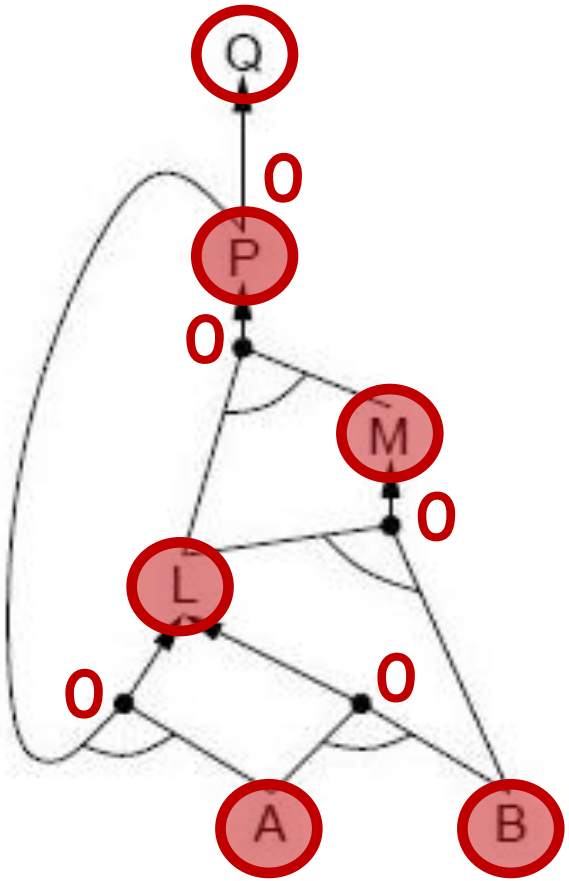
Example

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



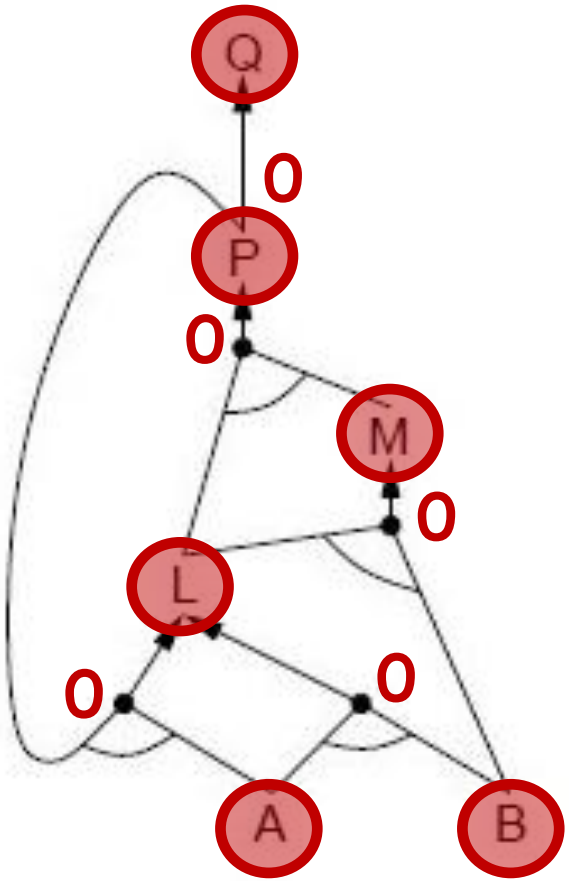
Example

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



Example

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



Propositional Logic: **Inference**

A *mechanical* process for computing new sentences

1. Backward & Forward Chaining
2. Resolution (Proof by Contradiction)
3. SAT
    1. Davis Putnam
    2. WalkSAT

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:

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

Inference 2: Resolution

[Robinson 1965]

$$\{ (p \vee \alpha), (\neg p \vee \beta \vee \gamma) \} \vdash_R (\alpha \vee \beta \vee \gamma)$$

Correctness

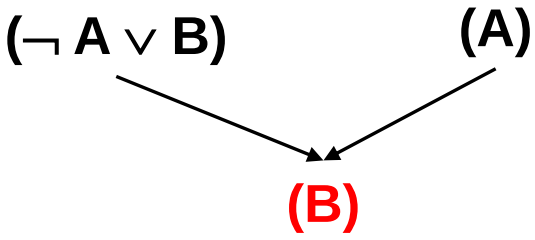
If $S1 \vdash_R S2$ then $S1 \models S2$

Refutation Completeness:

If S is unsatisfiable then $S \vdash_R ()$

Resolution subsumes Modus Ponens

$A \rightarrow B, A \models B$



If Will goes, Jane will go

$\sim W \vee J$

If doesn't go, Jane will still go

$W \vee J$

Will Jane go?

$\models J?$

$\boxed{J \vee J} = J$

Don't need to use other equivalences if we use resolution in refutation style

$\sim J$

$\sim W \vee J$

$W \vee J$

$\sim W$

J

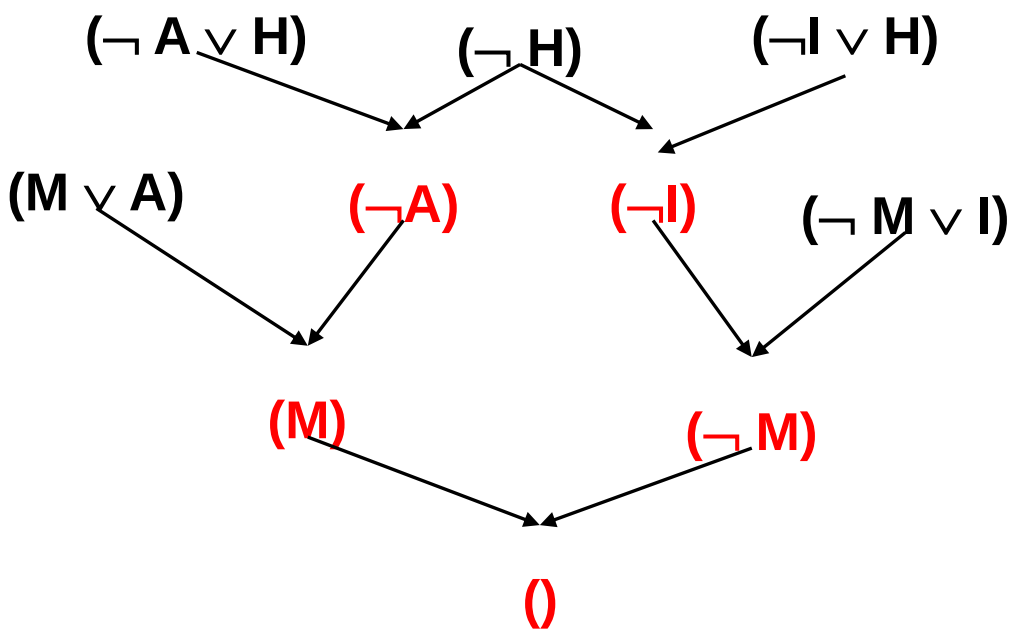
$\square$

Resolution

If the unicorn is mythical, then it is immortal, but if it is not mythical, it is a mammal. If the unicorn is either immortal or a mammal, then it is horned.

Prove: the unicorn is horned.

M = mythical
I = immortal
A = mammal
H = horned



Search in Resolution

- Convert the database into clausal form D_c
- Negate the goal first, and *then* convert it into clausal form D_G
- Let $D = D_c + D_G$
- Loop
  - Select a pair of Clauses C1 and C2 from D
    - Different control strategies can be used to select C1 and C2 to reduce number of resolutions tries
  - Resolve C1 and C2 to get C12
  - If C12 is empty clause, QED!! Return Success (We proved the theorem;)
  - D = D + C12

- Out of loop but no empty clause. Return "Failure"
  - Finiteness is guaranteed if we make sure that:
    - we never resolve the same pair of clauses more than once;
    - we use factoring, which removes multiple copies of literals from a clause (e.g. QVPVP => QVP)

Idea 1: Set of Support: At least one of C1 or C2 must be either the goal clause or a clause derived by doing resolutions on the goal clause (*COMPLETE*)

Idea 2: Linear input form: Atleast one of C1 or C2 must be one of the clauses in the input KB (*INCOMPLETE*)

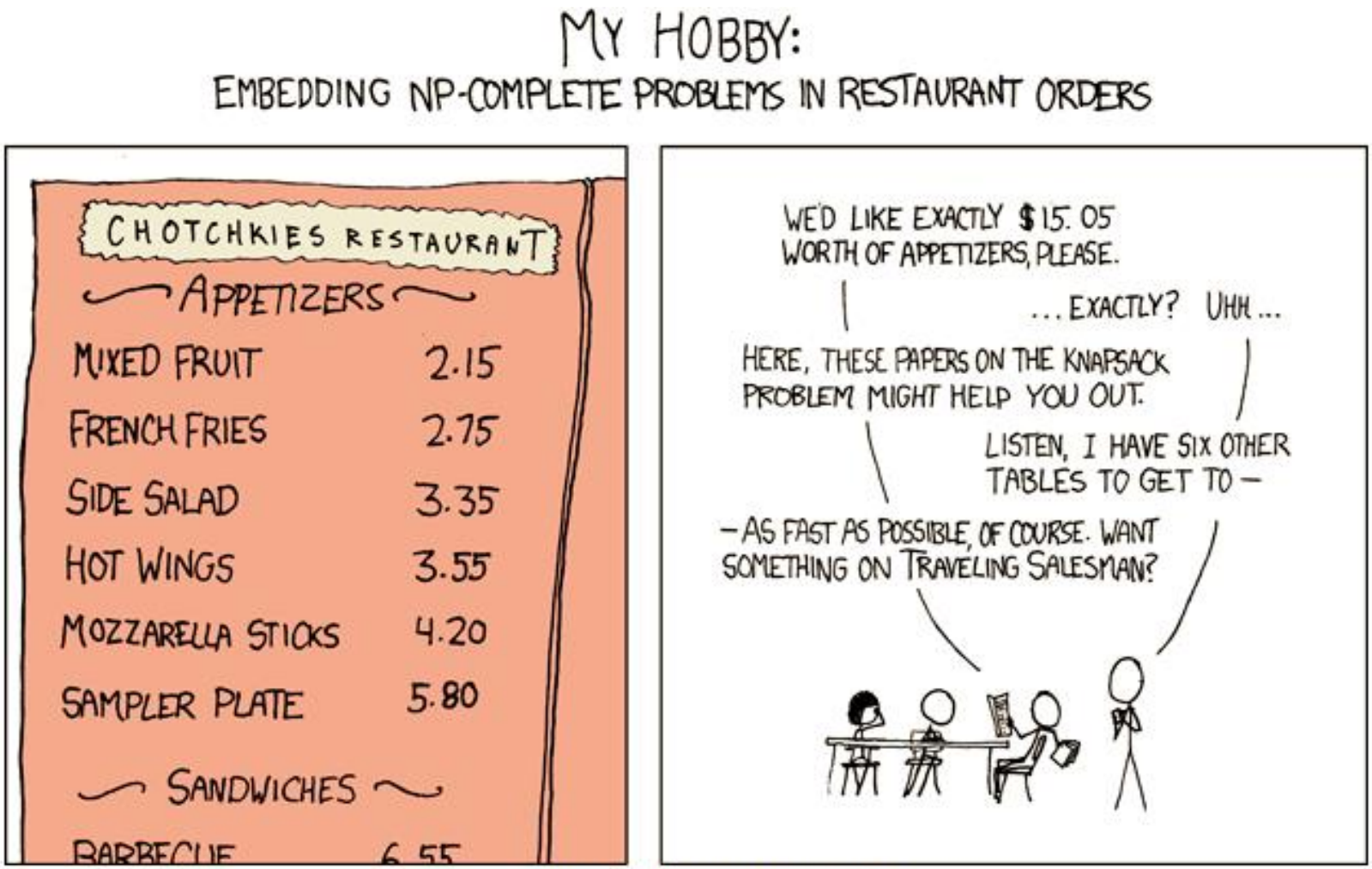
SAT: Model Finding

- Find assignments to variables that makes a formula true

Why study Satisfiability?

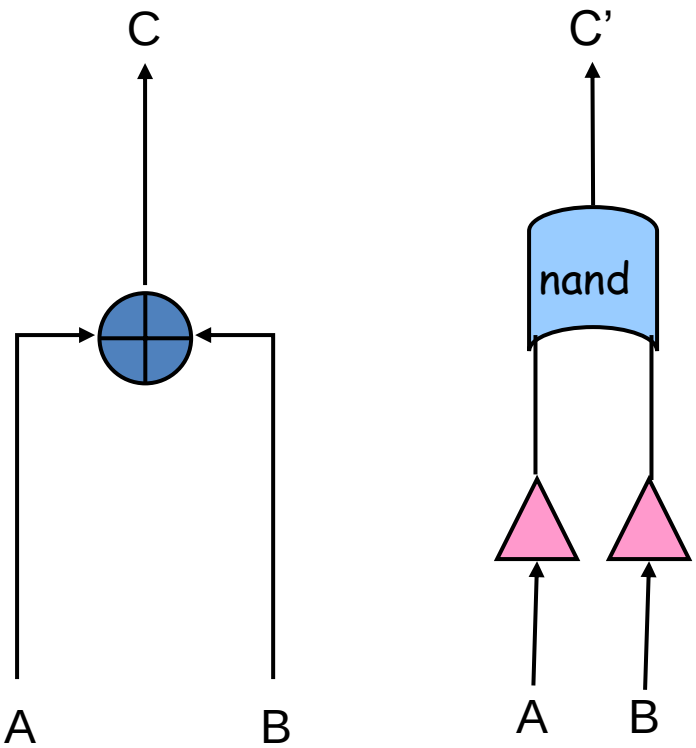
- Canonical NP complete problem.
  - several hard problems modeled as SAT
- Tonne of applications
- State-of-the-art solvers superfast

Tonne of Applications



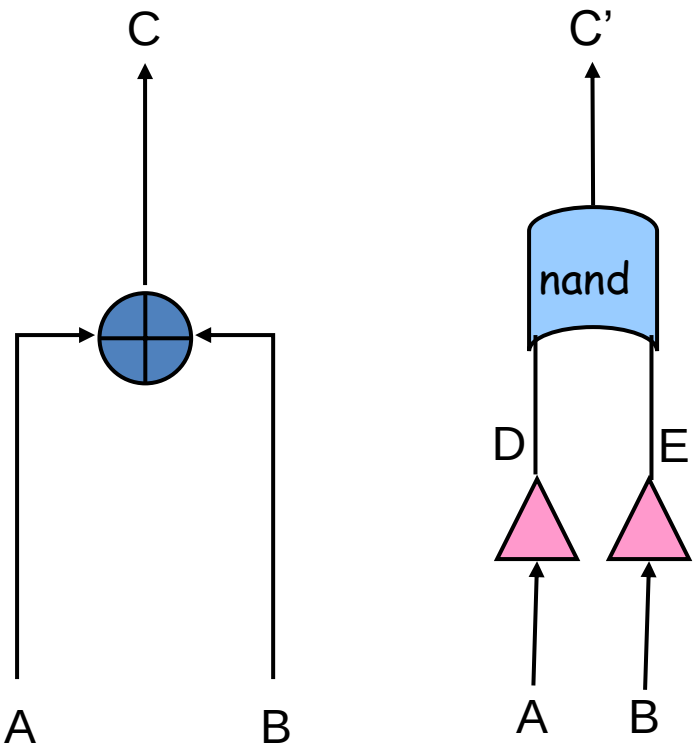
Title text: General solutions get you a 50% tip.

Testing Circuit Equivalence



- Do two circuits compute the same function?
- Circuit optimization
- Is there input for which the two circuits compute different values?

Testing Circuit Equivalence



$$C \equiv (A \vee B)$$

$$C' \equiv \neg(D \wedge E)$$

$$D \equiv \neg A$$

$$E \equiv \neg B$$

$$\neg(C \equiv C')$$

SAT Translation of N-Queens

- At least one queen each column:

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

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

  …

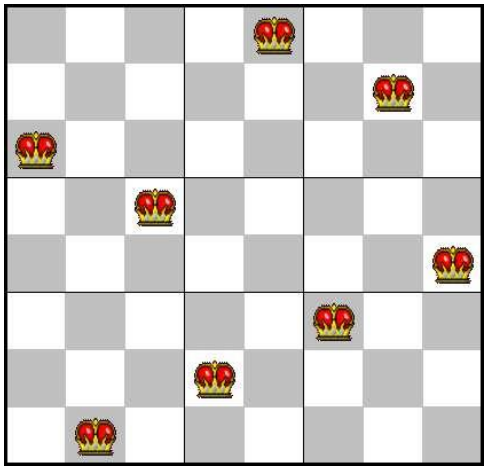
- No attacks:

  (~Q11 v ~Q12)

  (~Q11 v ~Q22)

  (~Q11 v ~Q21)

  …

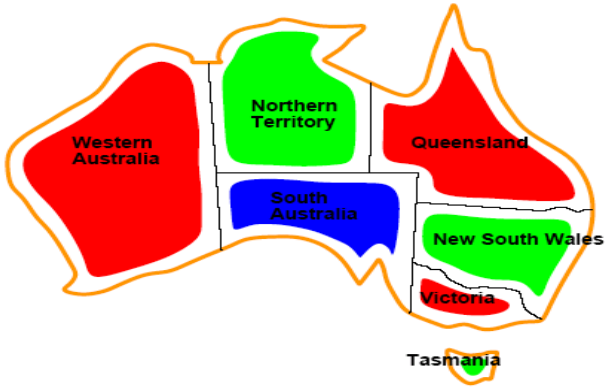


Graph Coloring

- A new SAT Variable for var-val pair

  $X_{WA\text{-}r}, X_{WA\text{-}g}, X_{WA\text{-}b}, X_{NT\text{-}r} \ldots$

- Each var has at least 1 value
  - $X_{WA\text{-}r} \vee X_{WA\text{-}g} \vee X_{WA\text{-}b}$
- No var has two values
  - $\sim X_{WA\text{-}r} \vee \sim X_{WA\text{-}g}$
  - $\sim X_{WA\text{-}r} \vee \sim X_{WA\text{-}b}$
- Constraints
  - $\sim X_{WA\text{-}r} \vee \sim X_{NT\text{-}r}$



Application: Diagnosis

- Problem: diagnosis a malfunctioning device
  - Car
  - Computer system
  - Spacecraft
- where
  - Design of the device is known
  - We can observe the state of only <u>certain parts</u> of the device – much is <u>hidden</u>

Model-Based, Consistency-Based Diagnosis

- Idea: create a logical formula that describes how the device <u>should</u> work
  - Associated with each "breakable" component C is a proposition that states "C is okay"
  - Sub-formulas about component C are all conditioned on C being okay
- A <u>diagnosis</u> is a smallest of "not okay" assumptions that are consistent with what is actually observed

Consistency-Based Diagnosis

1. Make some Observations O.
2. Initialize the Assumption Set A to assert that all components are working properly.
3. Check if the KB, A, O together are inconsistent (can deduce *false*).
4. If so, delete propositions from A until consistency is restored (cannot deduce *false*). The deleted propositions are a diagnosis.

There may be many possible diagnoses

Example: Automobile Diagnosis

- *Observable Propositions:*

  EngineRuns, GasInTank, ClockRuns

- *Assumable Propositions:*

  FuelLineOK, BatteryOK, CablesOK, ClockOK

- *Hidden (non-Assumable) Propositions:*

  GasInEngine, PowerToPlugs

- *Device Description F:*

  (GasInTank $\wedge$ FuelLineOK) $\rightarrow$ GasInEngine

  (GasInEngine $\wedge$ PowerToPlugs) $\rightarrow$ EngineRuns

  (BatteryOK $\wedge$ CablesOK) $\rightarrow$ PowerToPlugs

  (BatteryOK $\wedge$ ClockOK) $\rightarrow$ ClockRuns

- *Observations:*

  $\neg$ EngineRuns, GasInTank, ClockRuns

Example

- *Is* F ∪ Observations ∪ Assumptions *consistent?*

- F ∪ {¬EngineRuns, GasInTank, ClockRuns}
  ∪ { FuelLineOK, BatteryOK, CablesOK, ClockOK } → *false*
  - *Must restore consistency!*
- F ∪ {¬EngineRuns, GasInTank, ClockRuns}
  ∪ { BatteryOK, CablesOK, ClockOK } → *false*
  - ¬ *FuelLineOK is a diagnosis*
- F ∪ {¬EngineRuns, GasInTank, ClockRuns}
  ∪ {FuelLineOK, CablesOK, ClockOK } → *false*
  - ¬ *BatteryOK is <u>not</u> a diagnosis*

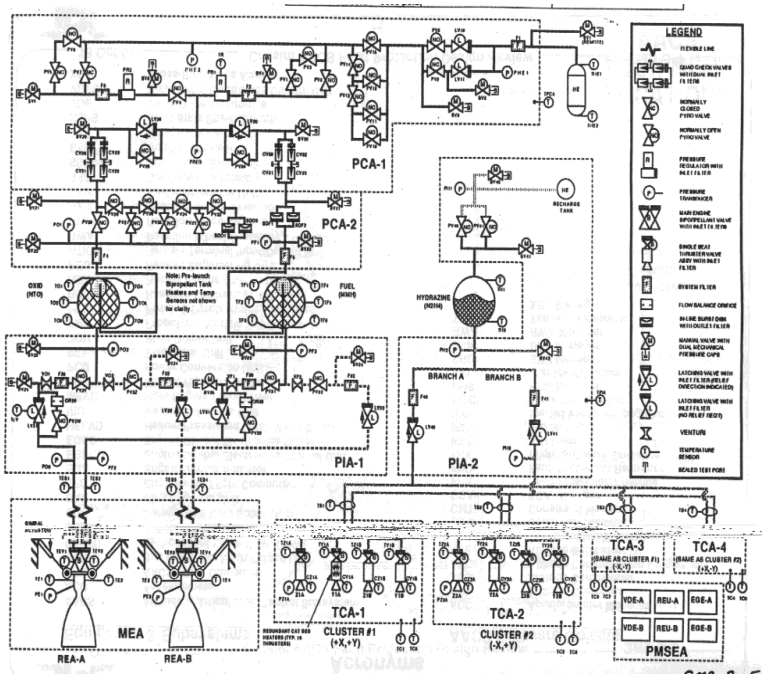
Complexity of Diagnosis

- If F is Horn, then each consistency test takes linear time – unit propagation is complete for Horn clauses.
- Complexity = ways to delete propositions from Assumption Set that are considered.
  - Single fault diagnosis – $O(n^2)$
  - Double fault diagnosis – $O(n^3)$
  - Triple fault diagnosis – $O(n^4)$

…

Deep Space One

- Autonomous diagnosis & repair “Remote Agent”
- Compiled systems schematic to 7,000 var SAT problem



Deep Space One

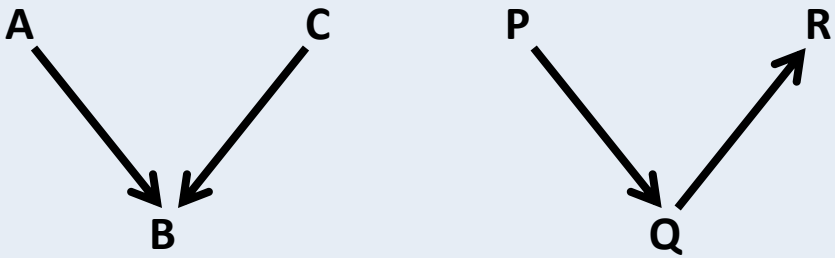
- a failed electronics unit
  - Remote Agent fixed by reactivating the unit.
- a failed sensor providing false information
  - Remote Agent recognized as unreliable and therefore correctly ignored.
- an attitude control thruster (a small engine for controlling the spacecraft's orientation) stuck in the "off" position
  - Remote Agent detected and compensated for by switching to a mode that did not rely on that thruster.

Assignment 2: Graph Subset Mapping

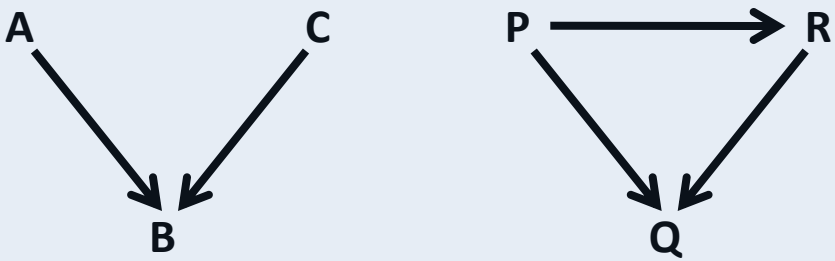
- Given two directed graphs G and G’
  - Check if G is a subset mapping to G’

- I.e. construct a one-one mapping (M) from all nodes of G to some nodes of G’ s.t.
  - (n1,n2) in G → (M(n1), M(n2)) in G’
  - (n1,n2) not in G → (M(n1), M(n2)) not in G’

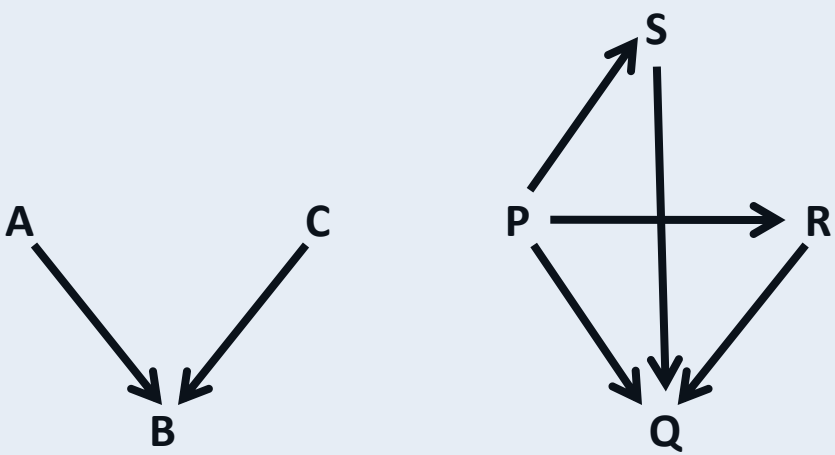
Graph G *Graph G'*



No, because the directionality of edges doesn't match.



No, because there is no edge between A and C in *G* whereas there is one between P and R in *G'*.



Yes. A mapping is: M(A) = S, M(B) = Q, M(C) = R

The edges from P to other nodes don't matter since no node in *G* got mapped to P.

SAT Model for Graph Subset Mapping

- If a mapping exists then SAT formula is satisfiable
- Else unsatisfiable

- The satisfying assignment suggests the mapping M

Inference 3: Model Enumeration

```
for (m in truth assignments){
   if  (m makes Φ true)
   then return "Sat!"
}
 return "Unsat!"
```

Inference 4: DPLL (Enumeration of ***Partial*** Models)

[Davis, Putnam, Loveland & Logemann 1962]

Version 1

```
dpll_1(pa){
  if (pa makes F false) return false;
  if (pa makes F true) return true;
  choose P in F;
  if (dpll_1(pa ∪ {P=0})) return true;
  return dpll_1(pa ∪ {P=1});
}
```

Returns true if F is satisfiable, false otherwise

DPLL Version 1

$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$

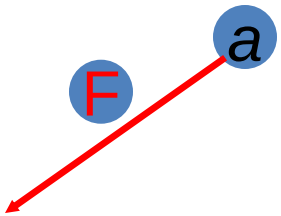
DPLL Version 1

$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$



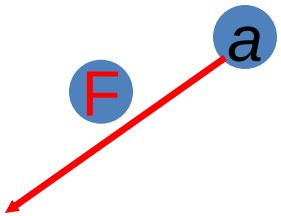
DPLL Version 1

$(F \lor b \lor c)$

$(F \lor \neg b)$

$(F \lor \neg c)$

$(T \lor c)$



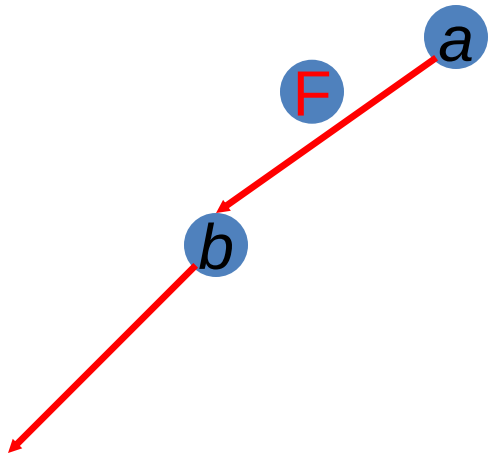
DPLL Version 1

$(F \vee F \vee c)$

$(F \vee T)$

$(F \vee \neg c)$

$(T \vee c)$



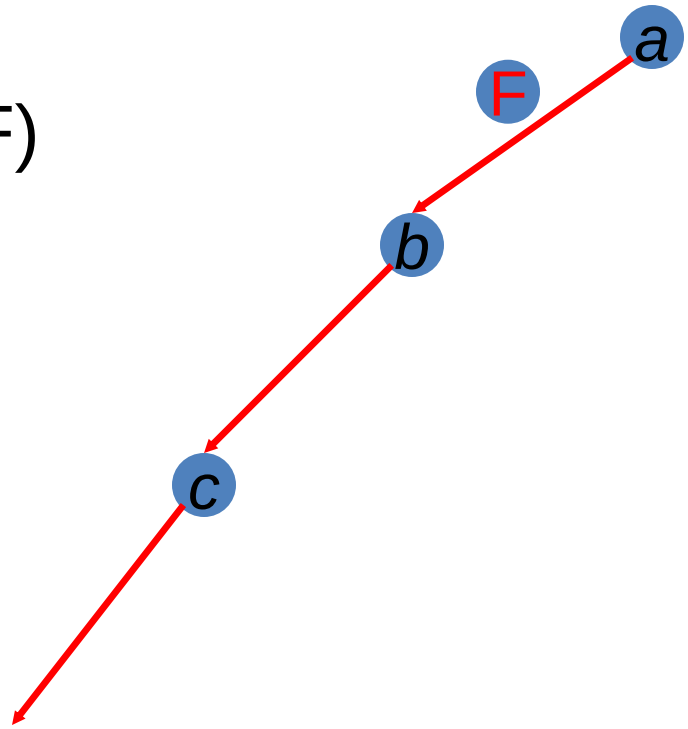
DPLL Version 1

$(F \vee F \vee F)$

$(F \vee T)$

$(F \vee T)$

$(T \vee F)$



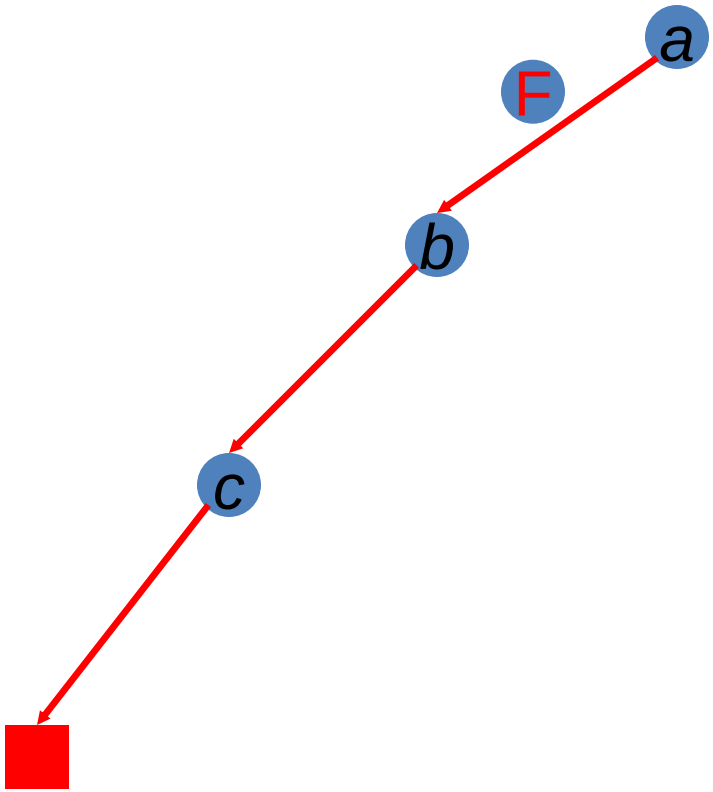
DPLL Version 1

F

T

T

T



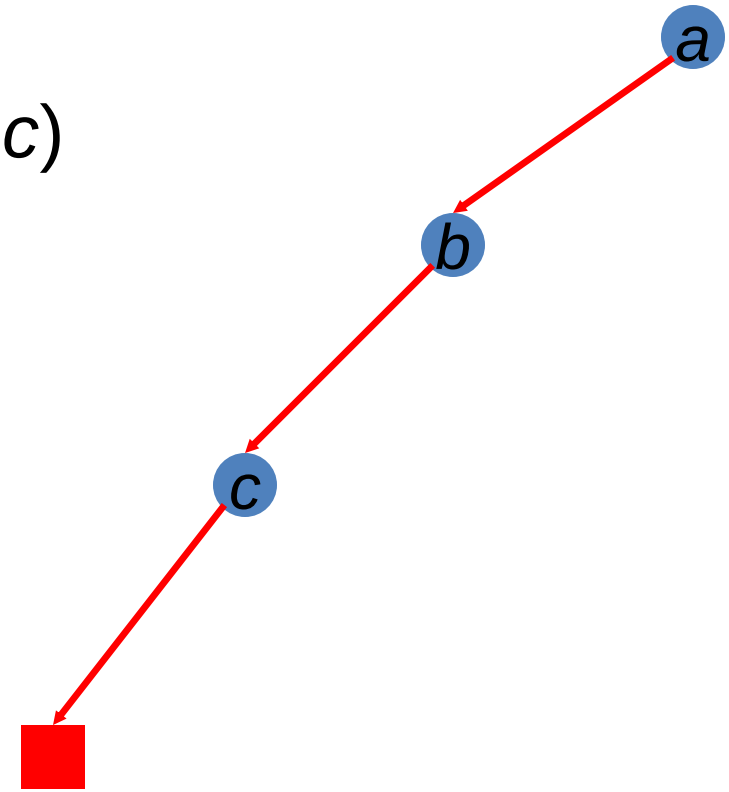
DPLL Version 1

$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$



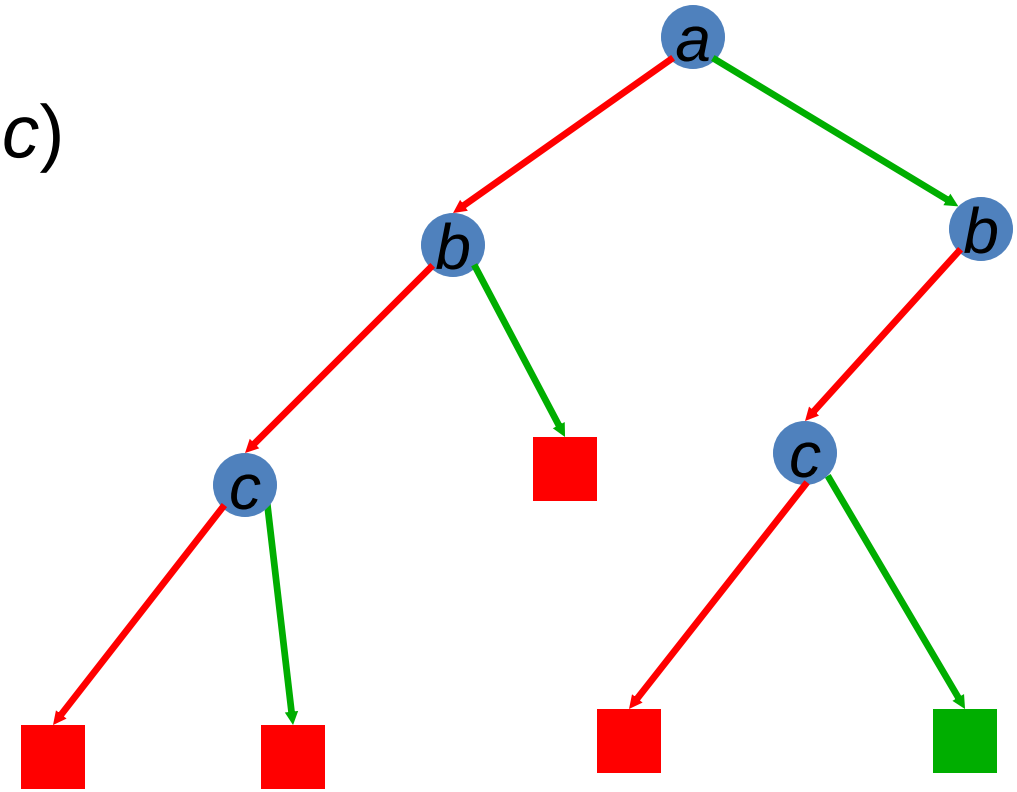
DPLL Version 1

$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$



DPLL as Search

- Search Space?
- Algorithm?

Improving DPLL

If literal L_1 is true, then clause $(L_1 \vee L_2 \vee \ldots)$ is true

If clause C_1 is true, then $C_1 \wedge C_2 \wedge C_3 \wedge \ldots$ has the same value as $C_2 \wedge C_3 \wedge \ldots$

Therefore: Okay to delete clauses containing true literals!

If literal L_1 is false, then clause $(L_1 \vee L_2 \vee L_3 \vee \ldots)$ has the same value as $(L_2 \vee L_3 \vee \ldots)$

Therefore: Okay to shorten clauses containing false literals!

If literal L_1 is false, then clause (L_1) is false

Therefore: the empty clause means false!

DPLL version 2

```
dpll_2(F, literal){
  remove clauses containing literal
  if (F contains no clauses)return true;
  shorten clauses containing ¬literal
  if (F contains empty clause)
     return false;
  choose V in F;
  if (dpll_2(F, ¬V))return true;
  return dpll_2(F, V);
}
```

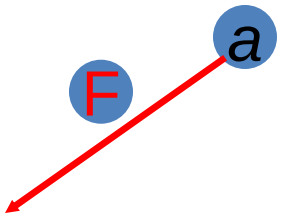
DPLL Version 2

$(F \vee b \vee c)$

$(F \vee \neg b)$

$(F \vee \neg c)$

$(T \vee c)$

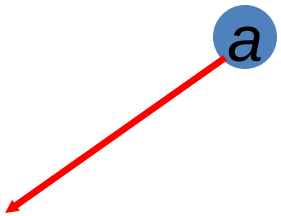


DPLL Version 2

$(b \vee c)$

$(\neg b)$

$(\neg c)$

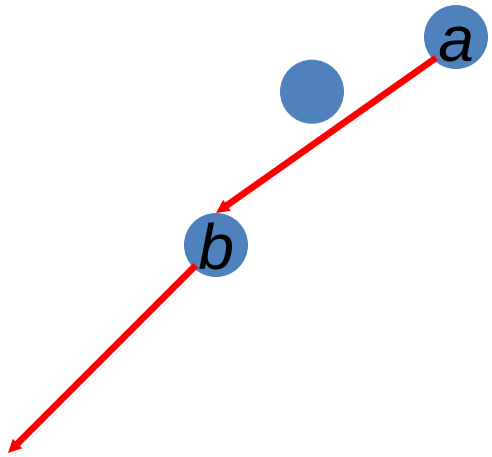


DPLL Version 2

(F $\vee$ c)

(T)

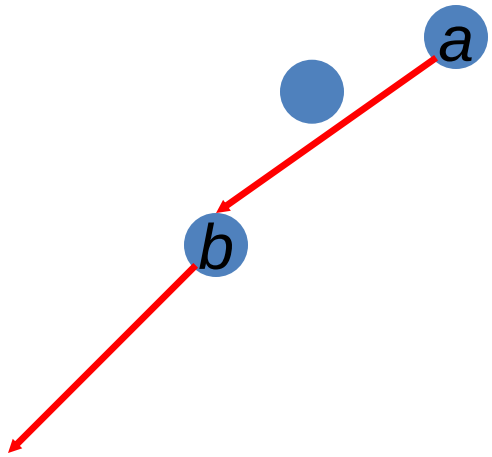
(¬c)



DPLL Version 2

(c)

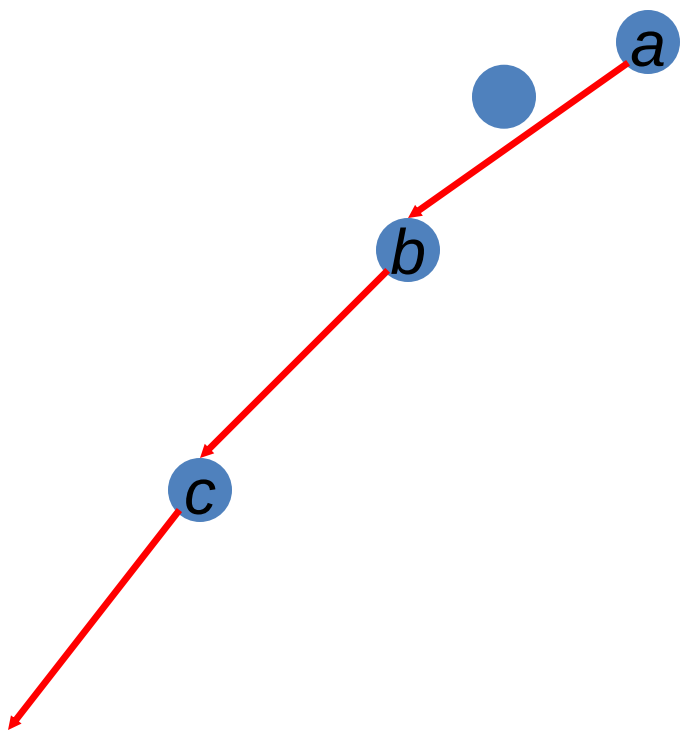
$(\neg c)$



DPLL Version 2

(F)

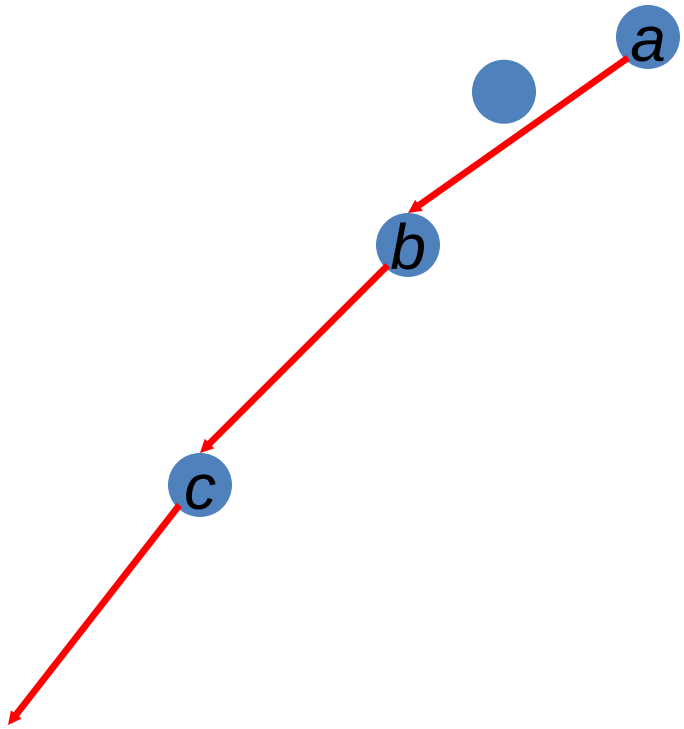
(T)



DPLL Version 2

Empty clause!

()



Structure in Clauses

- Unit Literals (unit propagation)

  A literal that appears in a singleton clause

  {{¬b c}{¬c}{a ¬b e}{d b}{e a ¬c}}

  Might as well set it true! *And simplify*

  {{¬b} {a ¬b e}{d b}}

  {{d}}

- Pure Literals
  - A symbol that always appears with same sign
  - {{a ¬b c}{¬c d ¬e}{¬a ¬b e}{d b}{e a ¬c}}

  Might as well set it true! *And simplify*

  {{a ¬b c} {¬a ¬b e} {e a ¬c}}

In Other Words

Formula $(L) \wedge C_2 \wedge C_3 \wedge \ldots$ is only true when literal L is true

Therefore: Branch immediately on unit literals!

May view this as adding constraint propagation techniques into play

In Other Words

Formula $(L) \wedge C_2 \wedge C_3 \wedge \ldots$ is only true when literal L is true

Therefore: Branch immediately on unit literals!

If literal L does not appear negated in formula F, then setting L true preserves satisfiability of F

Therefore: Branch immediately on pure literals!

May view this as adding constraint propagation techniques into play

DPLL (previous version)

Davis – Putnam – Loveland – Logemann

```
dpll(F, literal){
  remove clauses containing literal
  if (F contains no clauses) return true;
  shorten clauses containing ¬literal
  if (F contains empty clause)
     return false;


  choose V in F;
  if (dpll(F, ¬V))return true;
  return dpll(F, V);
}
```

DPLL (for real!)

Davis – Putnam – Loveland – Logemann

```
dpll(F, literal){
  remove clauses containing literal
  if (F contains no clauses) return true;
  shorten clauses containing ¬literal
  if (F contains empty clause)
     return false;
  if (F contains a unit or pure L)
     return dpll(F, L);
  choose V in F;
  if (dpll(F, ¬V))return true;
  return dpll(F, V);
}
```

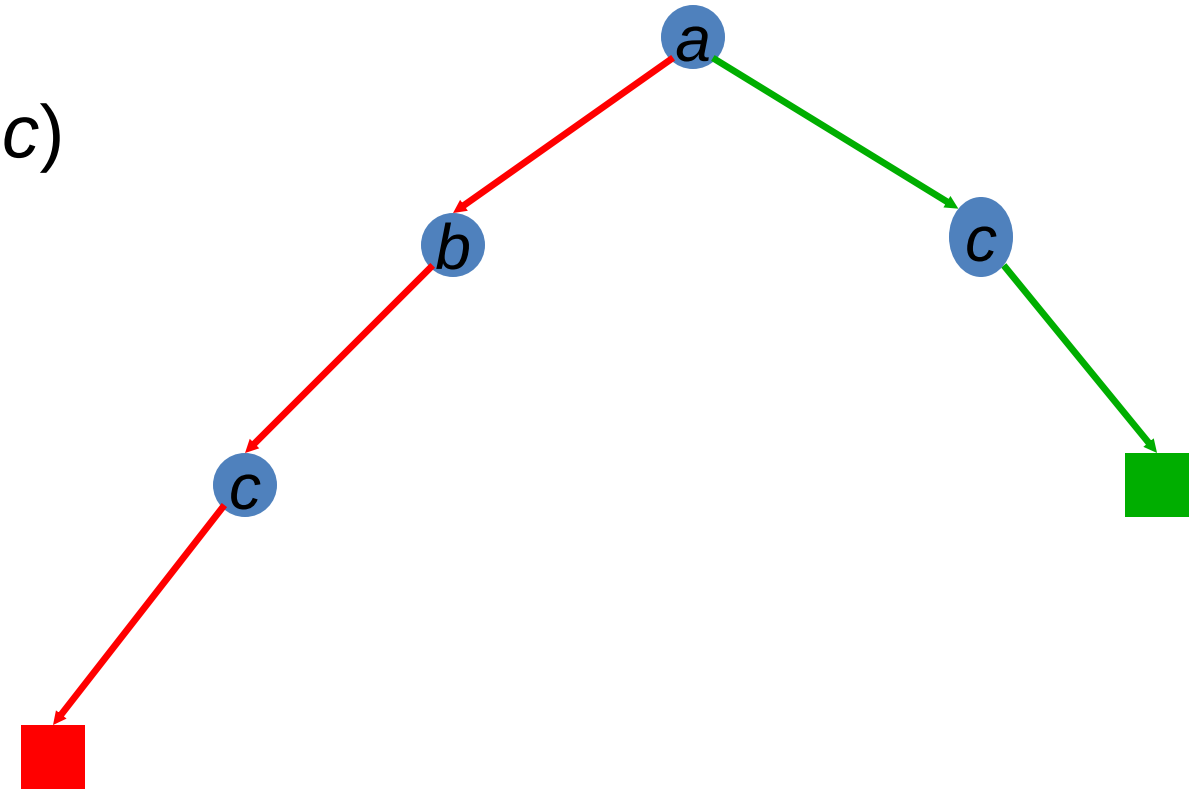
DPLL (for real)

$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$



DPLL (for real!)

Davis – Putnam – Loveland – Logemann

```
dpll(F, literal){
  remove clauses containing literal
  if (F contains no clauses) return true;
  shorten clauses containing ¬literal
  if (F contains empty clause)
      return false;
  if (F contains a unit or pure L)
      return dpll(F, L);
  choose V in F;
  if (dpll(F, ¬V))return true;
  return dpll(F, V);
}
```

Where could we use a heuristic to further improve performance?

Heuristic Search in DPLL

- Heuristics are used in DPLL to select a (non-unit, non-pure) proposition for branching

- Idea: identify a most constrained variable
  - Likely to create many unit clauses
- MOM's heuristic:
  - Most occurrences in clauses of minimum length

Success of DPLL

- 1962 – DPLL invented
- 1992 – 300 propositions
- 1997 – 600 propositions (satz)
- Additional techniques:
  - Learning conflict clauses at backtrack points
  - Randomized restarts
  - 2002 (zChaff) 1,000,000 propositions – encodings of hardware verification problems

GSAT

- ***Local*** search (Hill Climbing + Random Walk) over space of ***complete*** truth assignments
  - With prob p: flip any variable in any unsatisfied clause
  - With prob (1-p): flip best variable in any unsat clause
    - best = one which minimizes #unsatisfied clauses

- SAT encodings of N-Queens, scheduling
- Best algorithm for random K-SAT
  - Best DPLL: 700 variables
  - Walksat: 100,000 variables

Refining Greedy Random Walk

- Each flip
  - makes some false clauses become true
  - breaks some true clauses, that become false
- Suppose s1$\rightarrow$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 p do
            var := random member { x | x is in clause };
        else (probability 1-p)
            var := argmin_x { break[x] | x is in clause };
    endif
    state[var] := 1 – state[var];
end
return state;
```

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

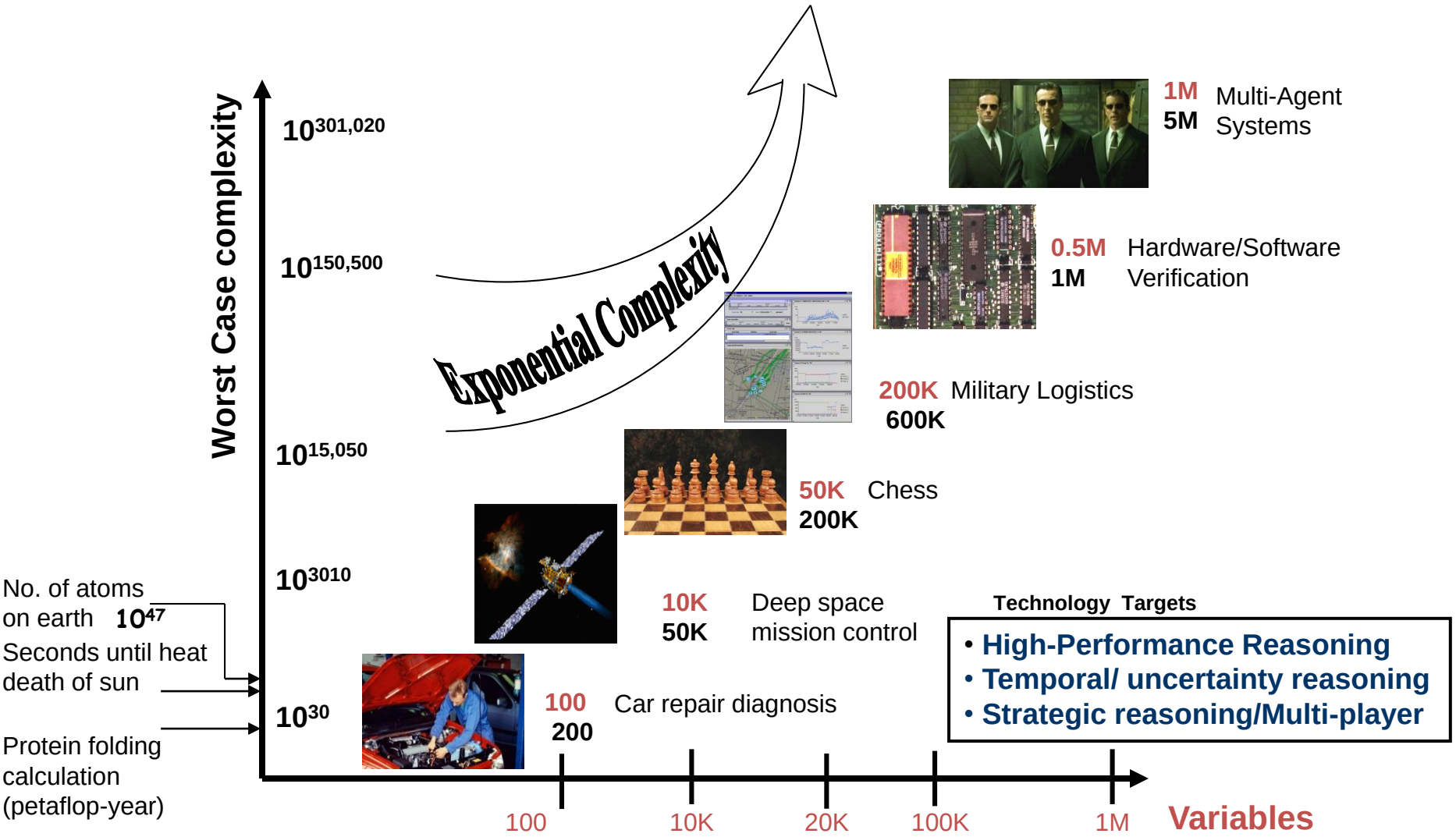
Advs of WalkSAT over GSAT

- WalkSat guaranteed to make at least 1 false clause (in random walk also)

- Number of evaluations small per move
  - does not iterate over all variables
  - only variables in the sampled clause

Real-World Reasoning

Tackling inherent computational complexity



Example domains cast in propositional reasoning system (variables, rules).

Symbolic Model Checking

- Any finite state machine is characterized by a transition function
  - CPU
  - Networking protocol
- Wish to prove some invariant holds for any possible inputs
- Bounded model checking: formula is sat *iff* invariant fails *k* steps in the future

$$\overline{S_t} = \text{ vector of Booleans representing state of machine at time } t$$

$$\rho : State \times Input \rightarrow State$$

$$\gamma : State \rightarrow \{0,1\}$$

$$\left(\bigwedge_{i=0}^{k-1} \left(\overline{S_{i+1}} \equiv \rho(\overline{S_i}, \overline{I_i}) \right) \right) \wedge S_o \wedge \neg\gamma\left(S_k\right)$$

A “real world” example

From “SATLIB”:

http://www.satlib.org/benchm.html

SAT-encoded bounded model checking instances (contributed by Ofer Shtrichman)

In Bounded Model Checking (BMC) [BCCZ99], a rather newly introduced problem in formal methods, the task is to check whether a given model M (typically a hardware design) satisfies a temporal property P in all paths with length less or equal to some bound k. The BMC problem can be efficiently reduced to a propositional satisfiability problem, and in fact if the property is in the form of an invariant (Invariants are the most common type of properties, and many other temporal properties can be reduced to their form. It has the form of 'it is always true that ... '.), it has a structure which is similar to many AI planning problems.

Bounded Model Checking instance

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

```
p cnf 51639 368352
-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)**
and ((not x_1) or x_6)
and … *etc.*

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
```

…

(x_{177} or x_{169} or x_{161} or x_{153} …
or x_{17} or x_9 or x_1 or (not x_{185}))

clauses / constraints are getting more interesting…

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
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

...

!!!

a 59-cnf clause…

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
```

What makes this possible?

Note that: $2^{50000} \approx 3.160699437 \cdot 10^{15051}$ ***… !!!***

The Chaff SAT solver (Princeton) solves this instance in less than one minute.

Progress in Last 20 years

- *Significant progress since the 1990's.* How much?
- Problem size: **We went from 100 variables, 200 constraints (early 90's)**
  to 1,000,000+ variables and 5,000,000+ constraints in 20 years

- Search space: from 10^30 to 10^300,000.
  [Aside: "one can encode quite a bit in 1M variables."]

- Is this just Moore's Law? It helped, but not much…
- – 2x faster computers does *not* mean can solve 2x larger instances
- – search difficulty does not scale linearly with problem size!
- **Tools**: 50+ competitive SAT solvers available

Forces Driving Faster, Better SAT Solvers

- **From academically interesting to practically relevant "Real" benchmarks**, with real interest in solving them

- Regular **SAT Solver Competitions** (Germany-89, Dimacs-93, China-96, SAT-02, SAT-03, …, SAT-07, SAT-09, SAT-2011)
  - "Industrial-instances-only" **SAT Races** (2008, 2010)
  - A tremendous resource! E.g., SAT Competition 2014:
    - 137 solvers submitted, downloadable, mostly open source
      - 79 teams, 14 countries
    - 500+ industrial benchmarks, 1000+ other benchmarks
    - 50,000+ benchmark instances available on the Internet
- *This constant improvement in SAT solvers is the key to the success of, e.g., SAT-based planning and verification*

Other Techniques: Nogood Learning

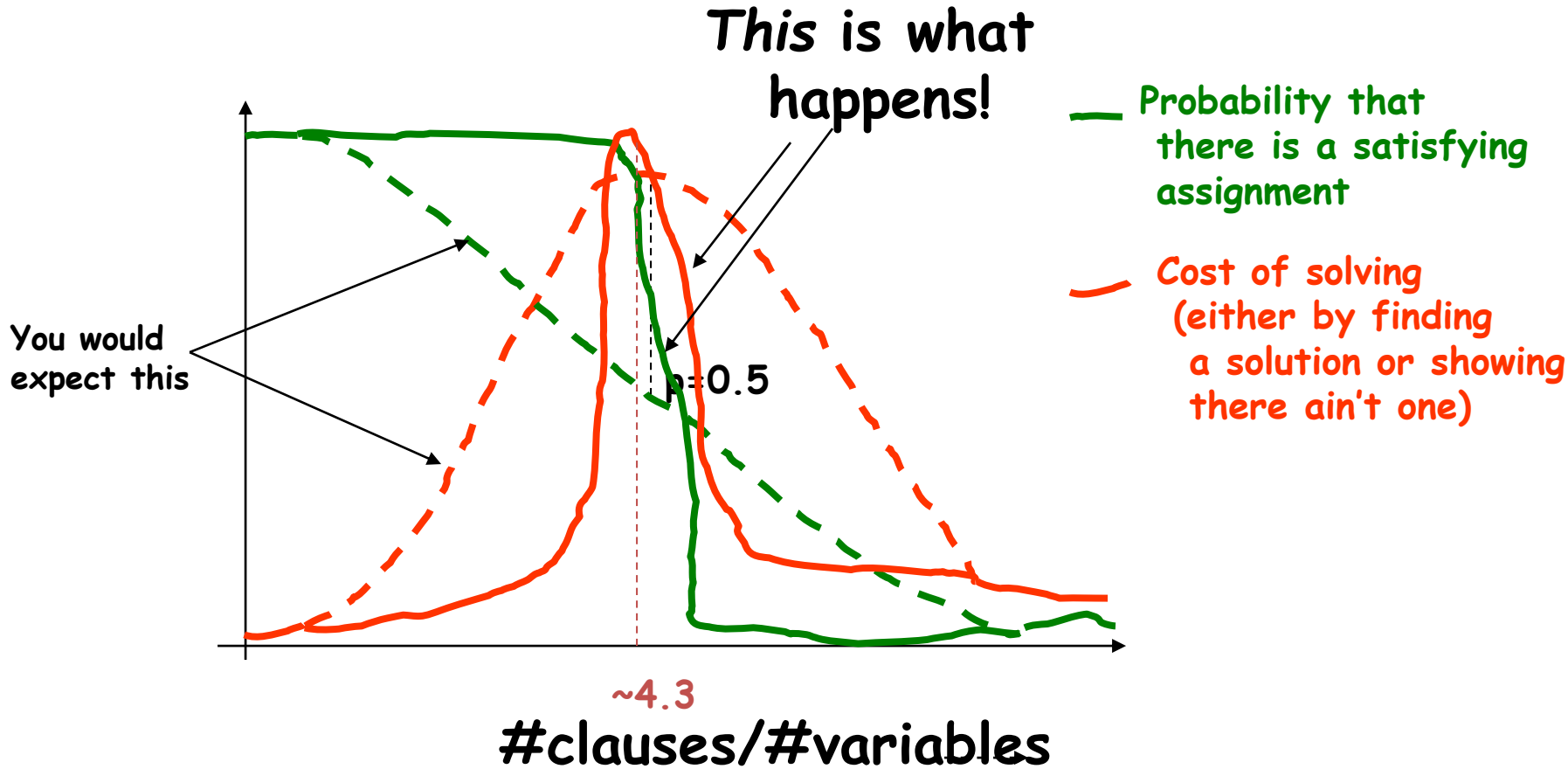
- Learn from mistakes *during* search
  - Nogood Learning: when DPLL backtracks,
- Learn a concise reason: what went wrong
  - avoid similar ‘mistakes’ in the future!
  - **Extremely powerful** in practice

Other Techniques: Machine Learning

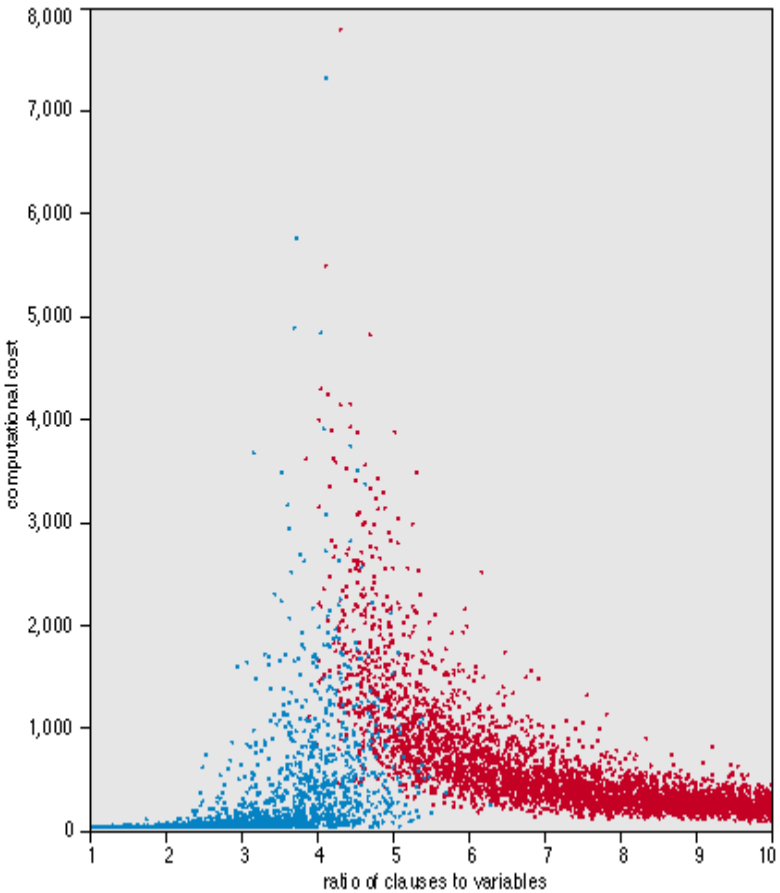
- Machine learning to build **algorithm portfolios**
  - Observation: no single SAT solver is good on every family of instances
  - Features of a given instance can be used to predict, with reasonable accuracy, which solver will work well on it!
  - **Solution**: design a portfolio solver using ML techniques
    - Based on runtime prediction models
    - Recent work – avoid complex models, use k-NN or clustering

- Automatic parameter tuning (generic and instance-specific)
  - SAT solvers are designed with many 'hardwired' parameters
  - Millions of parameter combinations – impossible to explore all by hand!
  - **Solution**: use automatic parameter tuning tools based on local search, genetic algorithms, etc.

Hardness of 3-sat as a function of #clauses/#variables



Random 3-SAT

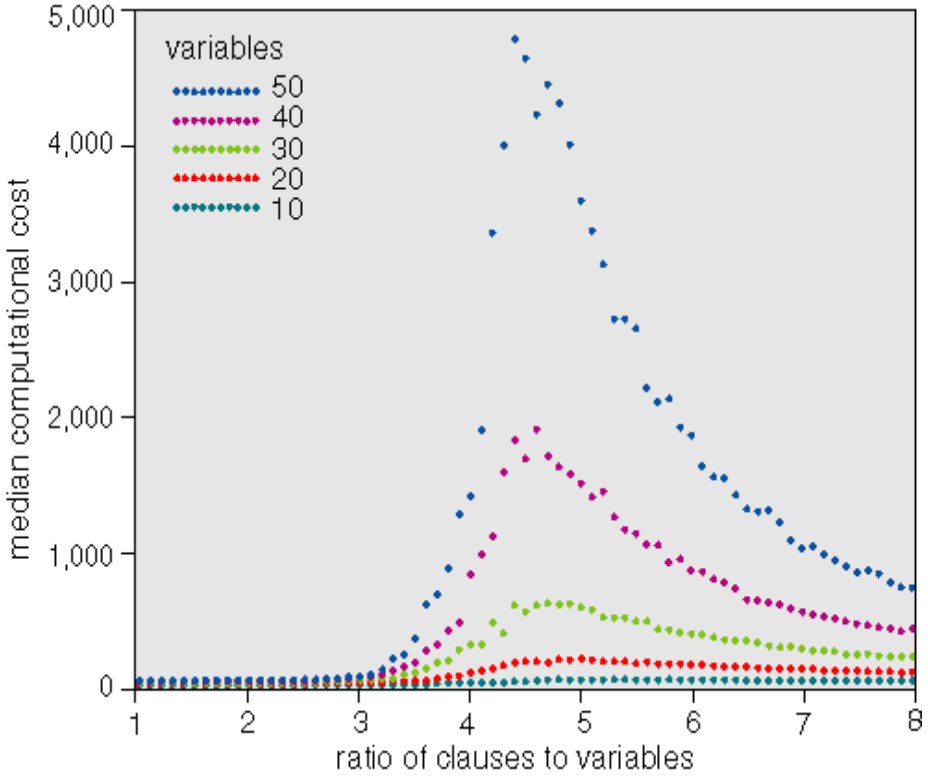


- Random 3-SAT
  - sample uniformly from space of all possible 3-clauses
  - *n* variables, *l* clauses
- Which are the hard instances?
  - around $l/n = 4.3$

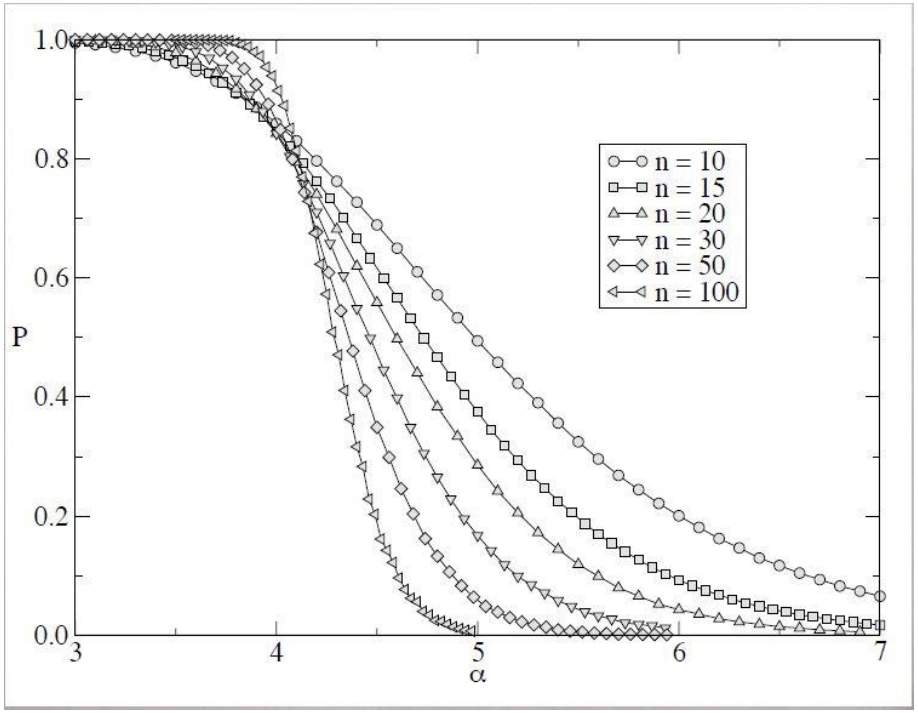
Random 3-SAT

- Varying problem size, *n*

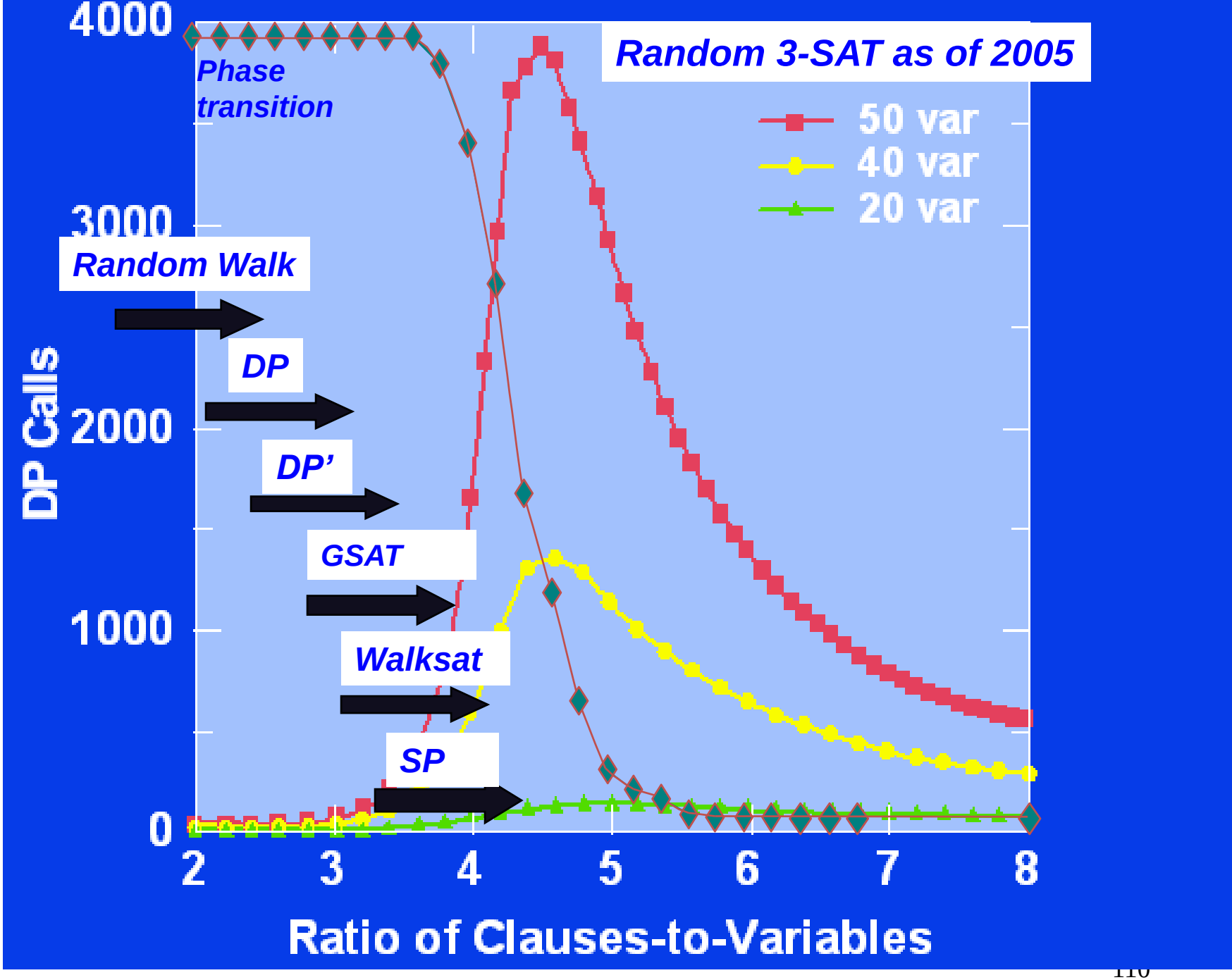
- Complexity peak appears to be largely invariant of algorithm



Random 3-SAT



- Complexity peak coincides with solubility transition
  - l/n < 4.3 problems under-constrained and SAT
  - l/n > 4.3 problems over-constrained and UNSAT
  - l/n=4.3, problems on “knife-edge” between SAT and UNSAT



Mitchell, Selman, and Levesque '92

Results: Random 3-SAT

- Random walk up to ratio 1.36 (Alekhnovich and Ben Sasson 03).
  empirically up to 2.5
- Davis Putnam (DP) up to 3.42 (Kaporis et al. ’02) ‘
  empirically up to 3.6
  approx. 400 vars at phase transition
- GSAT up till ratio 3.92 (Selman et al. ’92, Zecchina et al. ‘02)
  approx. 1,000 vars at phase transition
- Walksat up till ratio 4.1 (empirical, Selman et al. ’93)
  approx. 100,000 vars at phase transition
- Survey propagation (SP) up till 4.2
  (empirical, Mezard, Parisi, Zecchina ’02)
  approx. 1,000,000 vars near phase transition

3SAT phase transition

- Upper bounds (easier)
  - Typically by estimating count of solutions

3SAT phase transition

- Upper bounds (easier)
  - Typically by estimating count of solutions
  - E.g. Markov (or 1st moment) method

    For any statistic X

    $\text{prob}(X>=1) <= E[X]$

3SAT phase transition

- Upper bounds (easier)
  - Typically by estimating count of solutions
  - E.g. Markov (or 1st moment) method

For any statistic X

$$\text{prob}(X>=1) <= E[X]$$

$$\begin{aligned} E[X] &= 0.p(X=0) + 1.p(X=1) + 2.p(X=2) + 3.p(X=3) + \ldots \\ &>= \quad 1.p(X=1) + 1.p(X=2) + 1.p(X=3) + \ldots \\ &>= p(X>=1) \end{aligned}$$

3SAT phase transition

- Upper bounds (easier)
  - Typically by estimating count of solutions
  - E.g. Markov (or 1st moment) method

    For any statistic X

    prob(X>=1) <= E[X]

    No assumptions about the distribution of X *except non-negative!*

3SAT phase transition

- Upper bounds (easier)
  - Typically by estimating count of solutions
  - E.g. Markov (or 1st moment) method

    For any statistic X

    prob(X>=1) <= E[X]

    Let X be the number of satisfying assignments for a 3SAT problem

3SAT phase transition

- Upper bounds (easier)
  - Typically by estimating count of solutions
  - E.g. Markov (or 1st moment) method

    For any statistic X

    prob(X>=1) <= E[X]

    Let X be the number of satisfying assignments for a 3SAT problem

    The expected value of X can be easily calculated

3SAT phase transition

- Upper bounds (easier)
  - Typically by estimating count of solutions
  - E.g. Markov (or 1st moment) method

    For any statistic X

    prob(X>=1) <= E[X]

    Let X be the number of satisfying assignments for a 3SAT problem

    E[X] = 2^*n* * (7/8)^*l*

3SAT phase transition

- Upper bounds (easier)
  - Typically by estimating count of solutions
  - E.g. Markov (or 1st moment) method

    For any statistic X

    prob(X>=1) <= E[X]

    Let X be the number of satisfying assignments for a 3SAT problem

    E[X] = 2^*n* * (7/8)^*l*

    If E[X] < 1, then prob(X>=1) = prob(SAT) < 1

3SAT phase transition

- Upper bounds (easier)
  - Typically by estimating count of solutions
  - E.g. Markov (or 1st moment) method

    For any statistic X

    prob(X>=1) <= E[X]

    Let X be the number of satisfying assignments for a 3SAT problem

    E[X] = 2^*n* * (7/8)^*l*

    If E[X] < 1, then 2^*n* * (7/8)^*l* < 1

3SAT phase transition

- Upper bounds (easier)
  - Typically by estimating count of solutions
  - E.g. Markov (or 1st moment) method

    For any statistic X

    prob(X>=1) <= E[X]

    Let X be the number of satisfying assignments for a 3SAT problem

    E[X] = 2^*n* * (7/8)^*l*

    If E[X] < 1, then 2^*n* * (7/8)^*l* < 1

    n + *l* log2(7/8) < 0

3SAT phase transition

- Upper bounds (easier)
  - Typically by estimating count of solutions
  - E.g. Markov (or 1st moment) method

    For any statistic X

    prob(X>=1) <= E[X]

    Let X be the number of satisfying assignments for a 3SAT problem

    E[X] = 2^*n* * (7/8)^*l*

    If E[X] < 1, then 2^*n* * (7/8)^*l* < 1

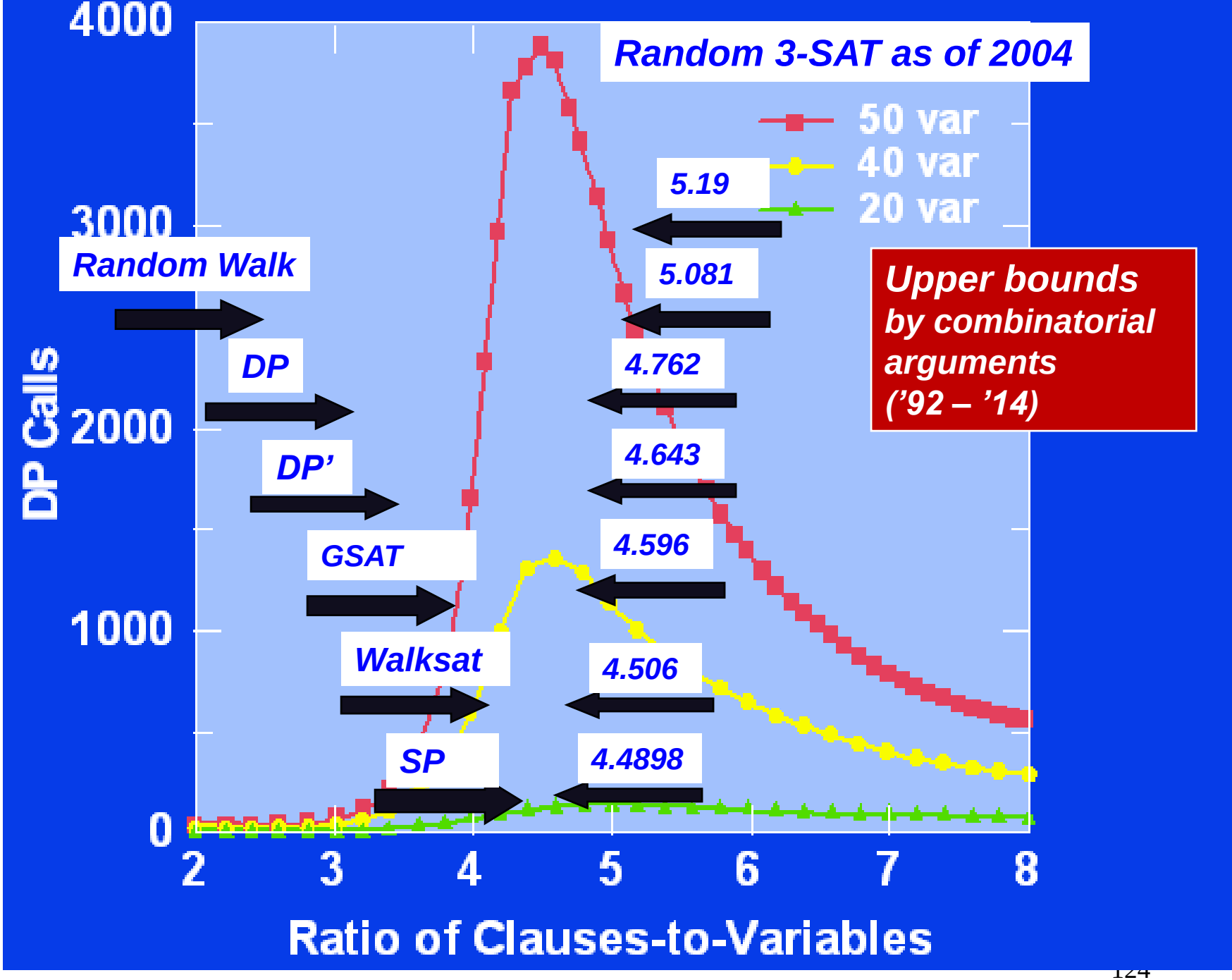
    n + *l* log2(7/8) < 0

    l/*n* > 1/log2(8/7) = 5.19…

Average vs Number

- But the transition is much lower at $l/n \sim 4.27$. What going on?

- In the range $4.27 < l/n < 5.19$,
  - the average no. of solutions is exponentially large.
- Occasionally, there are exponentially many...
  - ...but most of the time there are none!
- Large average doesn't prove satisfiability!


Random 3-SAT as of 2004
50 var
40 var
20 var
5.19
5.081
4.762
4.643
4.596
4.506
4.4898
Random Walk
DP
DP'
GSAT
Walksat
SP
Upper bounds by combinatorial arguments ('92 – '14)
DP Calls
4000
3000
2000
1000
0
2
3
4
5
6
7
8
Ratio of Clauses-to-Variables

2+*p*-SAT

Morph between 2-SAT and 3-SAT

– fraction *p* of 3-clauses

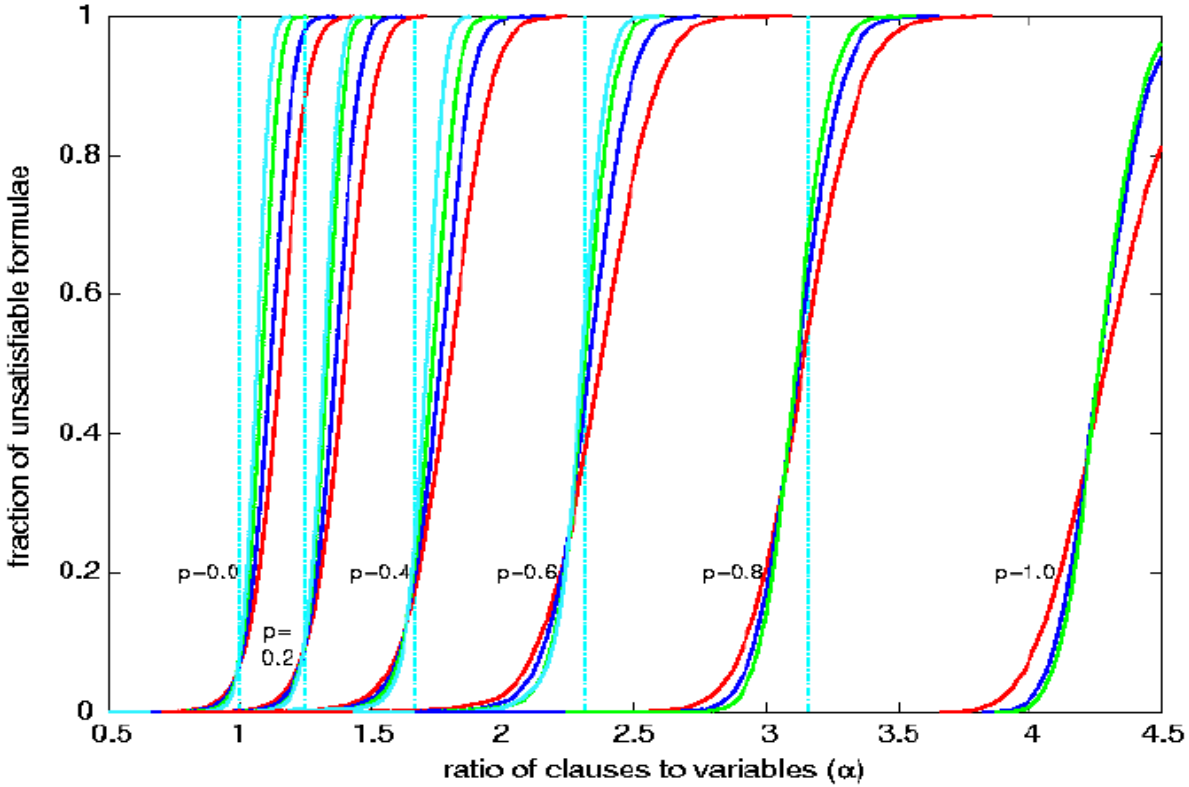
– fraction *(1-p)* of 2-clauses

[Monasson et al 1999]

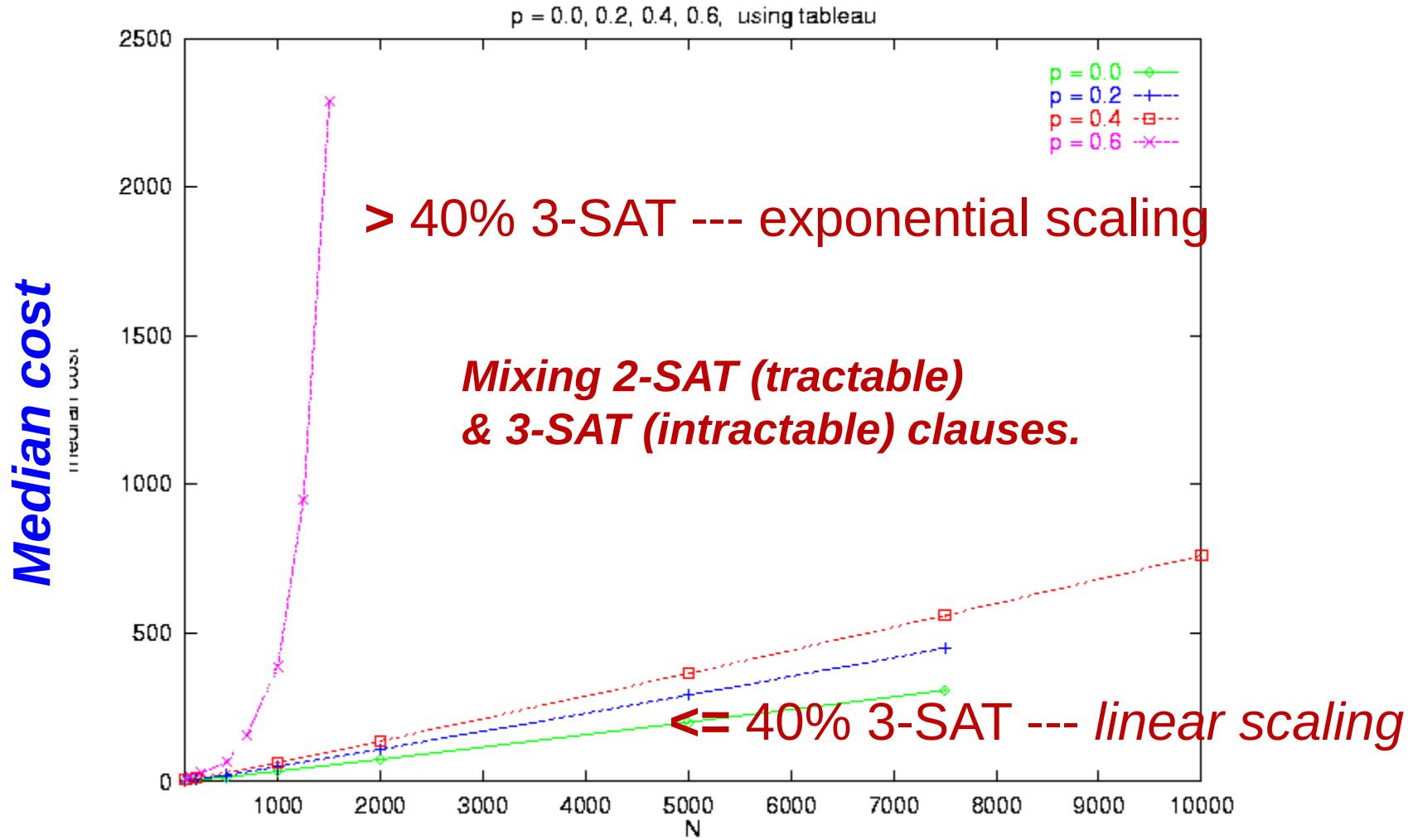
2+*p*-SAT

- Maps from P to NP
  - NP-complete for any *p>0*
  - Insight into change from P to NP *[Monasson et al 1999]*

2+*p*-SAT


fraction of unsatisfiable formulae
ratio of clauses to variables (α)
p=0.0
p=0.2
p=0.4
p=0.6
p=0.8
p=1.0
0
0.2
0.4
0.6
0.8
1
0.5
1
1.5
2
2.5
3
3.5
4
4.5

Computational Cost: 2+p-SAT
Tractable substructure can dominate!



(Monasson et al. 99; Achlioptas '00)