Introduction to Artificial Intelligence

8: Satisfiability and Model Construction

Luca Doria, KPH Mainz

Logical Deduction vs Satisfiability

Propositional Logic:

Logical deduction

- Given: A logical theory (set of propositions)
- Question: Does a proposition follow from this theory?
- Algorithm: reduction to unsatisfiability (coNP-complete problem)

Satisfiability (SAT)

- Given: A logical theory
- Target: A model for the theory
- Easier in general: we just need one model
- Wide applications range.

The SAT Problem

The SAT Problem

AKA as Boolean Satisfiability Problem.
The problem consists in determining if there exists an interpretation that satisfies a given Boolean formula.
More concretely, SAT asks whether the variables of a given Boolean formula can be consistently replaced by the values *true* or *false* in such a way that the formula evaluates to *true*.

k-SAT

Since every logical expression can be reduced to CNF, we define k-SAT as the SAT problem referred to clauses with k literals.

SAT can be reformulated as a constrained satisfaction problem (CSP) with symbols of the alphabet as variables and domain values {T,F}. Clauses are constraints.

Why is SAT important?

Complexity

SAT was the first problem to be shown to be NP-complete.
In particular, 2-SAT belongs to **P**, while k-SAT with k>2 is **NP-complete** (Cook-Levin Theorem, Cook, 1971 Levin, 1973).

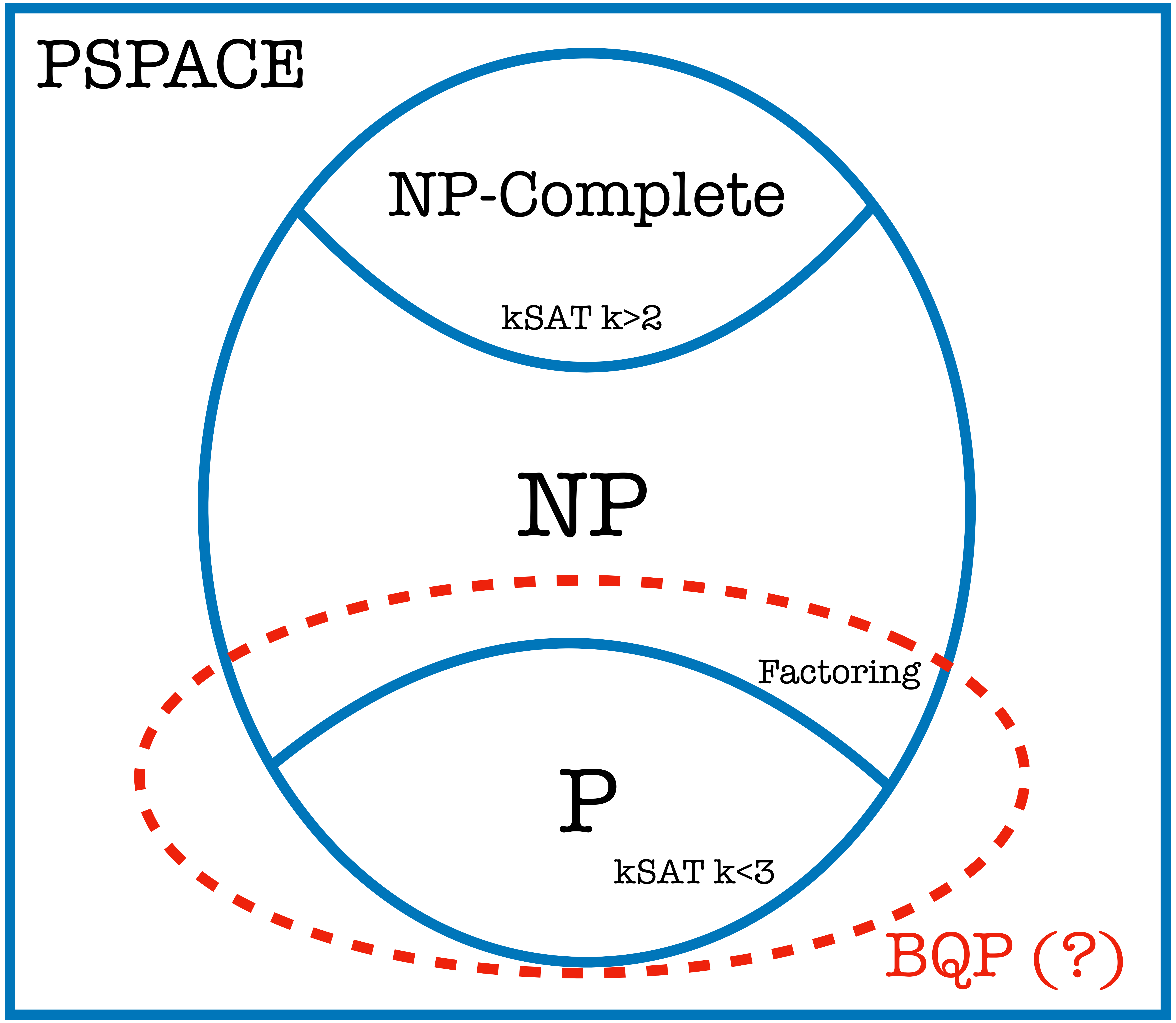
Notation:

$\mathbf{SAT_k(n,m)}$: SAT problem reduced in CNF with **m** clauses, **n** symbols and **k** different literals per clause.

Reducibility

We know that every problem in NP can be (polynomially) reduced to NP-complete problems, therefore SAT represents a paradigm for the problems in this class and NP-complete problems are the “most difficult” problems in NP.

Reminder: (Some) Complexity Classes



The DPLL Algorithm

The DPLL (Davis, Putnam, Logemann, Loveland, 1962) algorithm corresponds to **backtracking** with inference in constrained satisfaction problems.

In general, a SAT problem corresponds to a CSP if we regard the clauses as constraints.

How do we solved a CSP? With DF-search and backtracking.

Therefore: a similar procedure can be applied to solve SAT.

The DPLL Algorithm

The DPLL algorithm employs the following techniques for improving the search:

1) Early Termination: the algorithm tries to check if a sentence is true before even if the model is still not fully reconstructed. For example, a clause is true if one of the literals is true and the full sentence is true if all the clauses are true.
2) Pure symbol heuristic: A "pure symbol" is a literal that appears with the same "sign" in all the clauses. If there is a model, it has an assignment that makes the pure symbols true.
3) Unit clause heuristic: an unit clause consists of just one literal or it is a clause with only one literal unassigned and the others are false. Assigning an unit clause can imply a "cascade" of assignments called unit propagation.

The DPLL Algorithm

```
function DPLL_SAT(s) returns true or false
    clauses <— set of clauses from a CNF
    symbols <— propositions in s
    return DPLL(clauses, symbols, { })

function DPLL(clauses, symbols, model) returns true or false
    if every clause in clauses is true in model then return true
    if some clause in clauses is false in model then return false
    P, value <— FIND_PURE_SYMBOL(symbols, clauses, model)
    if P is non-null then return DPLL(clauses, symbols-P,model U {P=value})
    P, value <— FIND_UNIT_CLAUSE(clauses, model)
    if P is non-null then return DPLL(clauses, symbols-P, model U {P=value})
    P <— FIRST(symbols) ; rest <— REST(symbols)
    return DPLL(clauses, rest, model U {P=true}) or
           DPLL(clauses, rest, model U {P=false})
```

DPLL : Example

Consider the following CNF:

$$(\neg P_1 \lor P_2) \land (\neg P_1 \lor P_3 \lor P_5) \land (\neg P2 \lor P_4) \land (\neg P_3 \lor \neg P_4) \land (P_1 \lor P_5 \lor \neg P_2) \land (P_2 \lor P_3) \land (P_2 \lor \neg P_3 \lor P_7) \land (P_6 \lor \neg P_5)$$

corresponding to the satisfiability of the 8 clauses

$c_1 = (\neg P_1 \lor P_2)$

$c_2 = (\neg P_1 \lor P_3 \lor P_5)$

$c_3 = (\neg P_2 \lor P_4)$

$c_4 = (\neg P_3 \lor \neg P_4)$

$c_5 = (P_1 \lor P_5 \lor \neg P_2)$

$c_6 = (P_2 \lor P_3)$

$c_7 = (P_2 \lor \neg P_3 \lor P_7)$

$c_8 = (P_6 \lor \neg P_5)$

Let's apply DPLL…

DPLL : Example

$c_1 = (\neg P_1 \vee P_2)$

$c_2 = (\neg P_1 \vee P_3 \vee P_5)$

$c_3 = (\neg P_2 \vee P_4)$

$c_4 = (\neg P_3 \vee \neg P_4)$

$c_5 = (P_1 \vee P_5 \vee \neg P_2)$

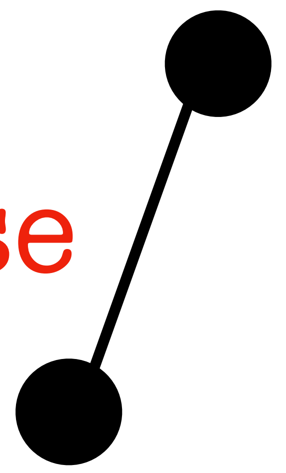
$c_6 = (P_2 \vee P_3)$

$c_7 = (P_2 \vee \neg P_3 \vee P_7)$

$c_8 = (P_6 \vee \neg P_5)$ This clause becomes a unit literal —> P_5 must be False

—> This clause is satisfied.

P_6= False



DPLL : Example

$c_1 = (\neg P_1 \vee P_2)$

$c_2 = (\neg P_1 \vee P_3 \vee P_5)$

$c_3 = (\neg P_2 \vee P_4)$

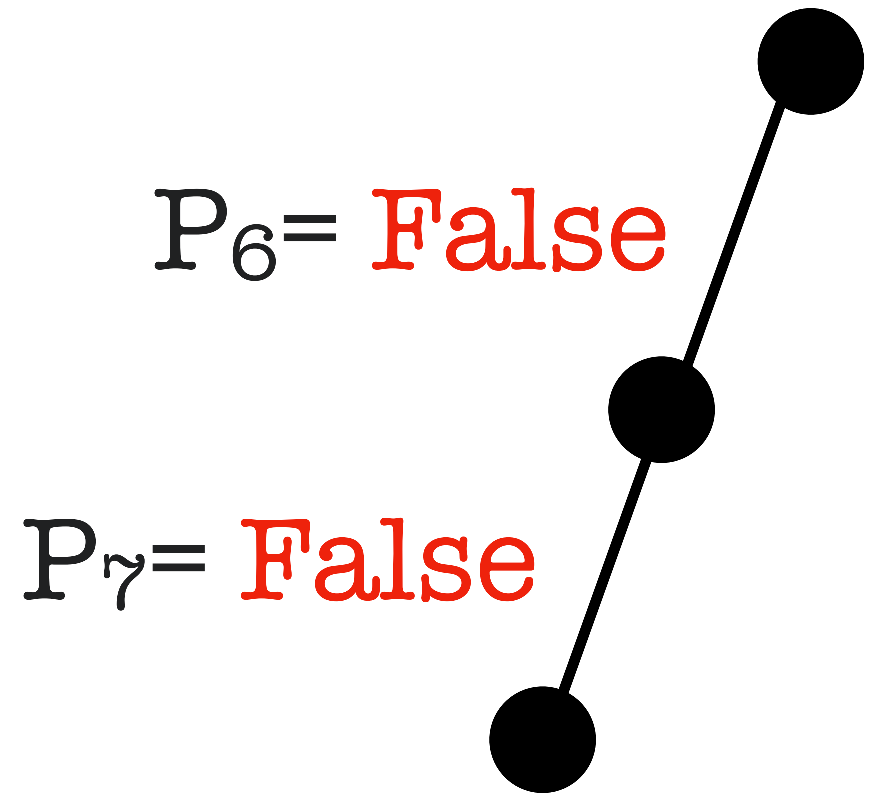
$c_4 = (\neg P_3 \vee \neg P_4)$

$c_5 = (P_1 \vee P_5 \vee \neg P_2)$

$c_6 = (P_2 \vee P_3)$

$c_7 = (P_2 \vee \neg P_3 \vee P_7)$

$c_8 = (P_6 \vee \neg P_5)$ Done



DPLL : Example

$c_1 = (\neg P_1 \vee P_2)$

$c_2 = (\neg P_1 \vee P_3 \vee P_5)$

$c_3 = (\neg P_2 \vee P_4)$

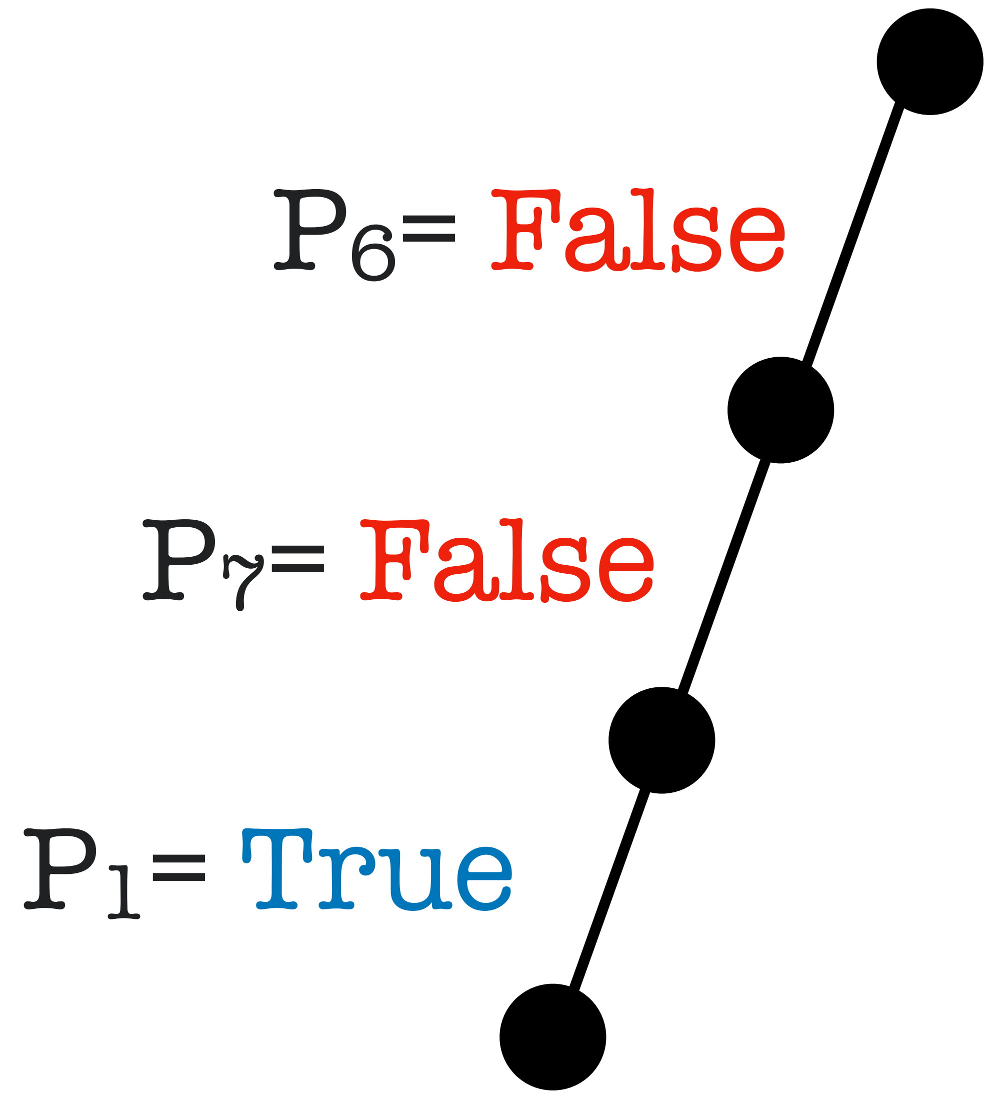
$c_4 = (\neg P_3 \vee \neg P_4)$

$c_5 = (P_1 \vee P_5 \vee \neg P_2)$ This clause is now true

$c_6 = (P_2 \vee P_3)$

$c_7 = (P_2 \vee \neg P_3 \vee P_7)$

$c_8 = (P_6 \vee \neg P_5)$ Done



DPLL : Example

$c_1 = (\neg P_1 \lor P_2)$

$c_2 = (\neg P_1 \lor P_3 \lor P_5)$ Unit propagation P_3=true

$c_3 = (\neg P_2 \lor P_4)$

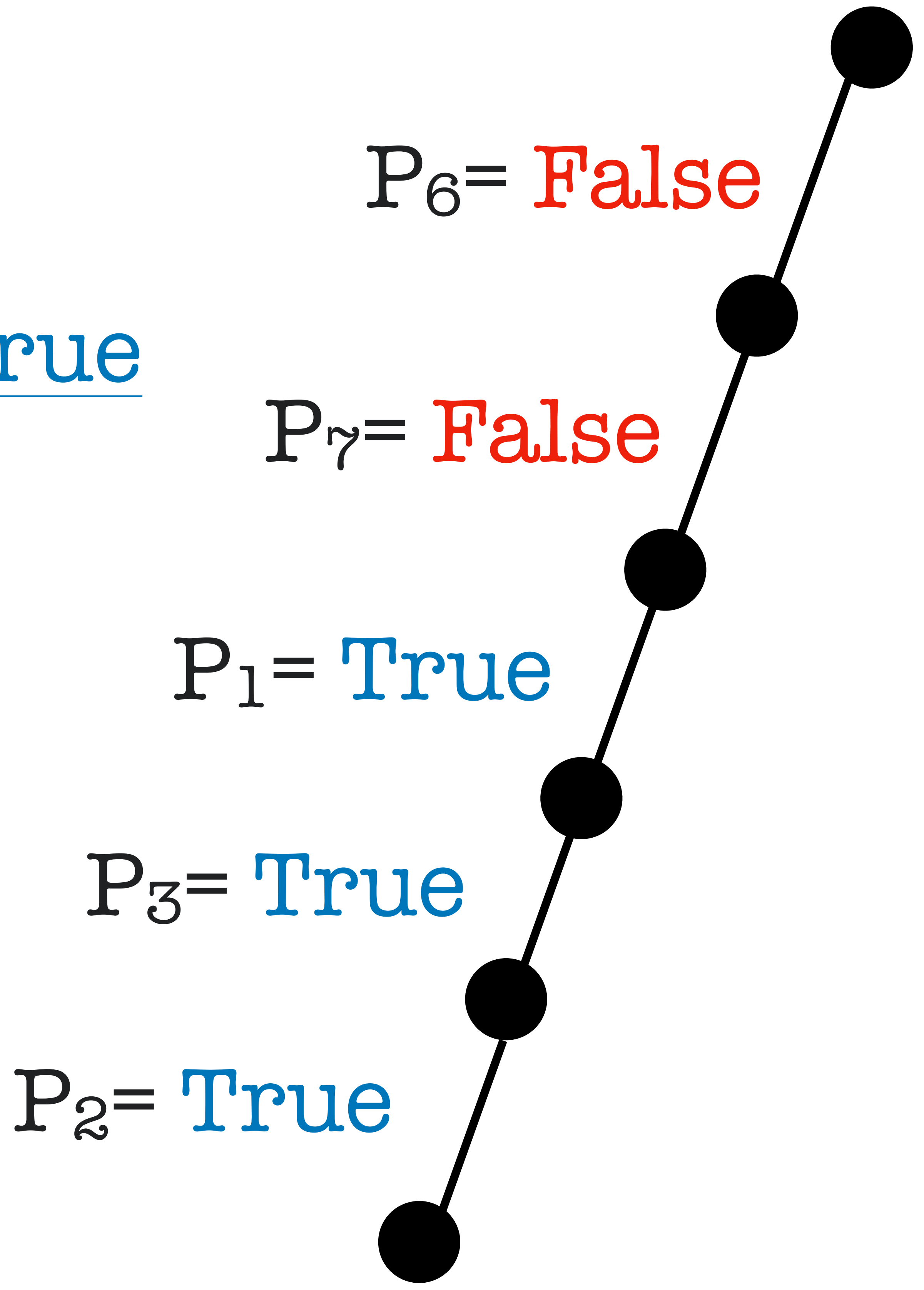
$c_4 = (\neg P_3 \lor \neg P_4)$

$c_5 = (P_1 \lor P_5 \lor \neg P_2)$ Done

$c_6 = (P_2 \lor P_3)$ Unit propagation

$c_7 = (P_2 \lor \neg P_3 \lor P_7)$

$c_8 = (P_6 \lor \neg P_5)$ Done



DPLL : Example

$c_1 = (\neg P_1 \lor P_2)$ Unit clause

$c_2 = (\neg P_1 \lor P_3 \lor P_5)$ Done

$c_3 = (\neg P_2 \lor P_4)$ P_4 must be true

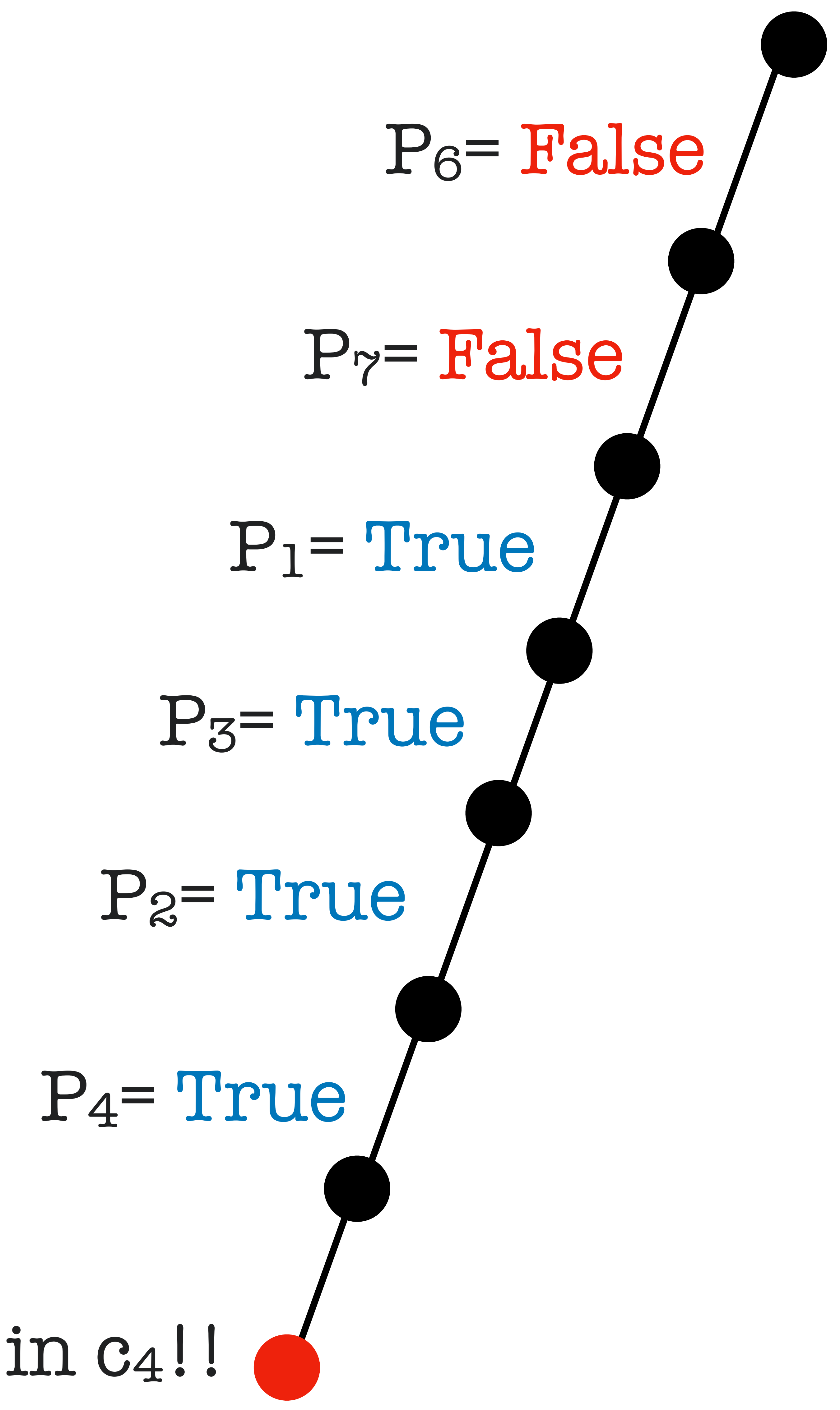
$c_4 = (\neg P_3 \lor \neg P_4)$ P_4 must be true

$c_5 = (P_1 \lor P_5 \lor \neg P_2)$ Done

$c_6 = (P_2 \lor P_3)$ Done

$c_7 = (P_2 \lor \neg P_3 \lor P_7)$

$c_8 = (P_6 \lor \neg P_5)$ Done



DPLL : Example

$c_1 = (\neg P_1 \lor P_2)$ Done

$c_2 = (\neg P_1 \lor P_3 \lor P_5)$ Done

$c_3 = (\neg P_2 \lor P_4)$ Done

$c_4 = (\neg P_3 \lor \neg P_4)$ Conflict

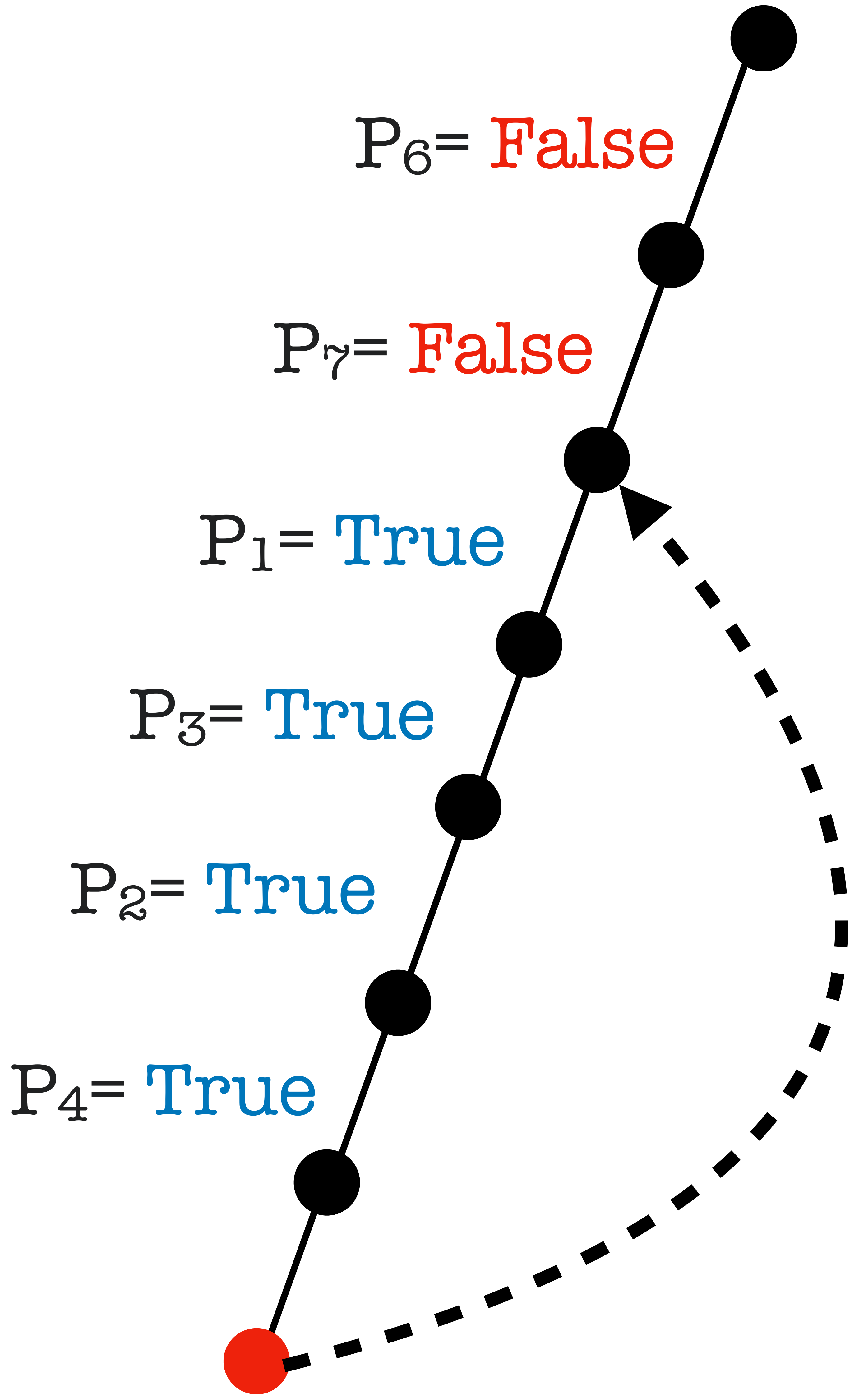
$c_5 = (P_1 \lor P_5 \lor \neg P_2)$ Done

$c_6 = (P_2 \lor P_3)$ Done

$c_7 = (P_2 \lor \neg P_3 \lor P_7)$

$c_8 = (P_6 \lor \neg P_5)$ Done

We took 3 decisions so far:
P_6, P_7, P_1, the other ones were forced.
BACKTRACK to the last.



DPLL : Example

$c_1 = (\neg P_1 \lor P_2)$ Done

$c_2 = (\neg P_1 \lor P_3 \lor P_5)$ Done

$c_3 = (\neg P_2 \lor P_4)$ Done

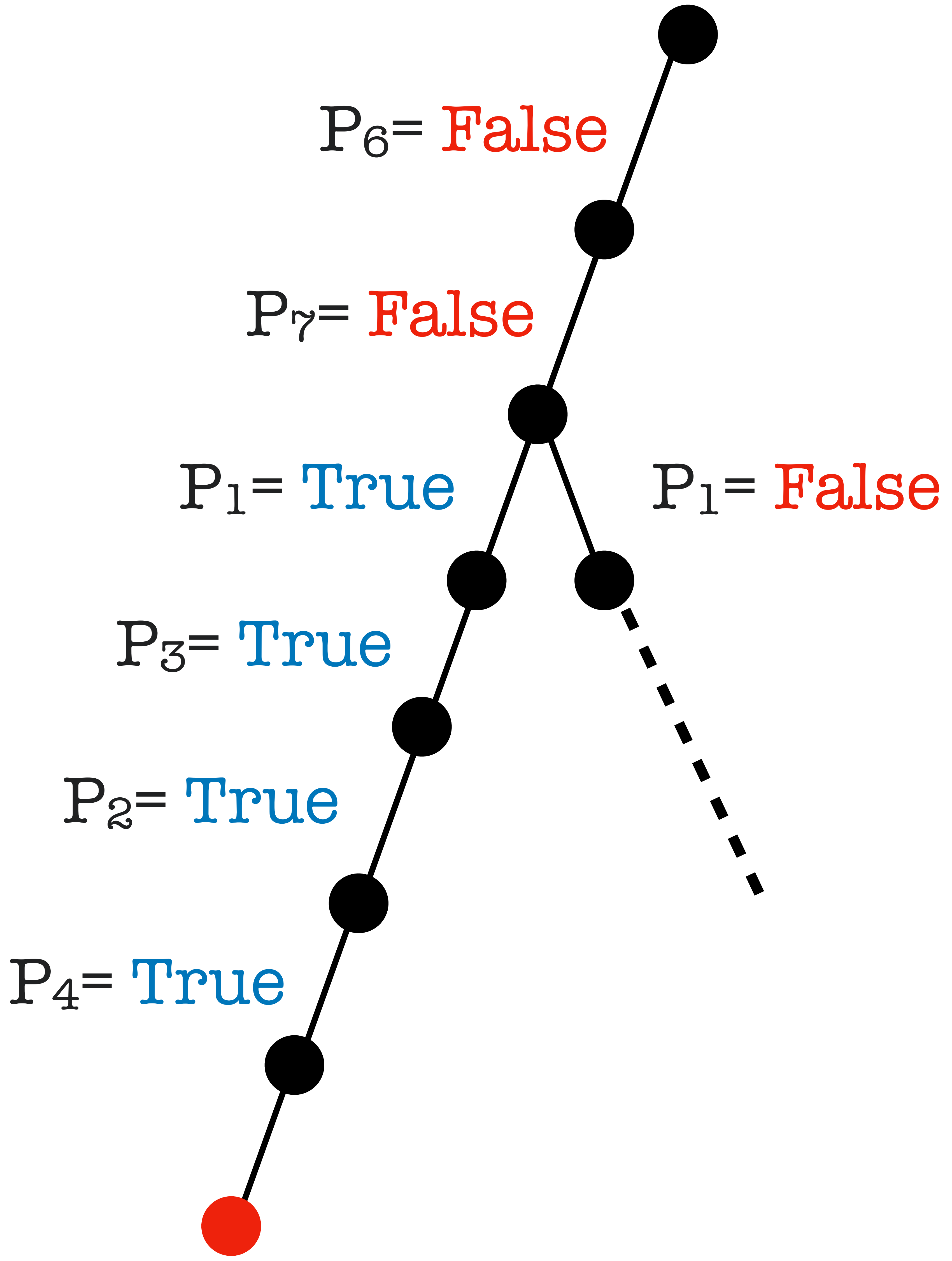
$c_4 = (\neg P_3 \lor \neg P_4)$ Conflict

$c_5 = (P_1 \lor P_5 \lor \neg P_2)$ Done

$c_6 = (P_2 \lor P_3)$ Done

$c_7 = (P_2 \lor \neg P_3 \lor P_7)$

$c_8 = (P_6 \lor \neg P_5)$ Done



DPLL Properties

- DPLL is complete, correct and always terminate.
- DPLL returns a model if one exists.
- DPLL requires in general exponential time (heuristics can help).
- DPLL requires polynomial time on Horn clauses (see next).
- In SAT competitions among algorithms, DPLL variants show excellent performances.

The average SAT instance

- SAT is NP-complete (exponential time in the worst case)
- True also for the best DPLL-style algorithm.
- Can we improve on the average case (not in the worst)?
- A first result (Goldberg, 1979): if the probability of a positive-, negative-, or non-appearance for a literal is 1/3, then <u>DPLL needs only average quadratic time</u>.

A digression: Phase Transitions

Phase transitions refer to when a substance transforms between a state of matter to another one.

Example: the boiling point of water (100^oC) or its freezing point (0^oC).

The change of phase is usually “fast” and it is controlled by a parameter (e.g. the temperature).

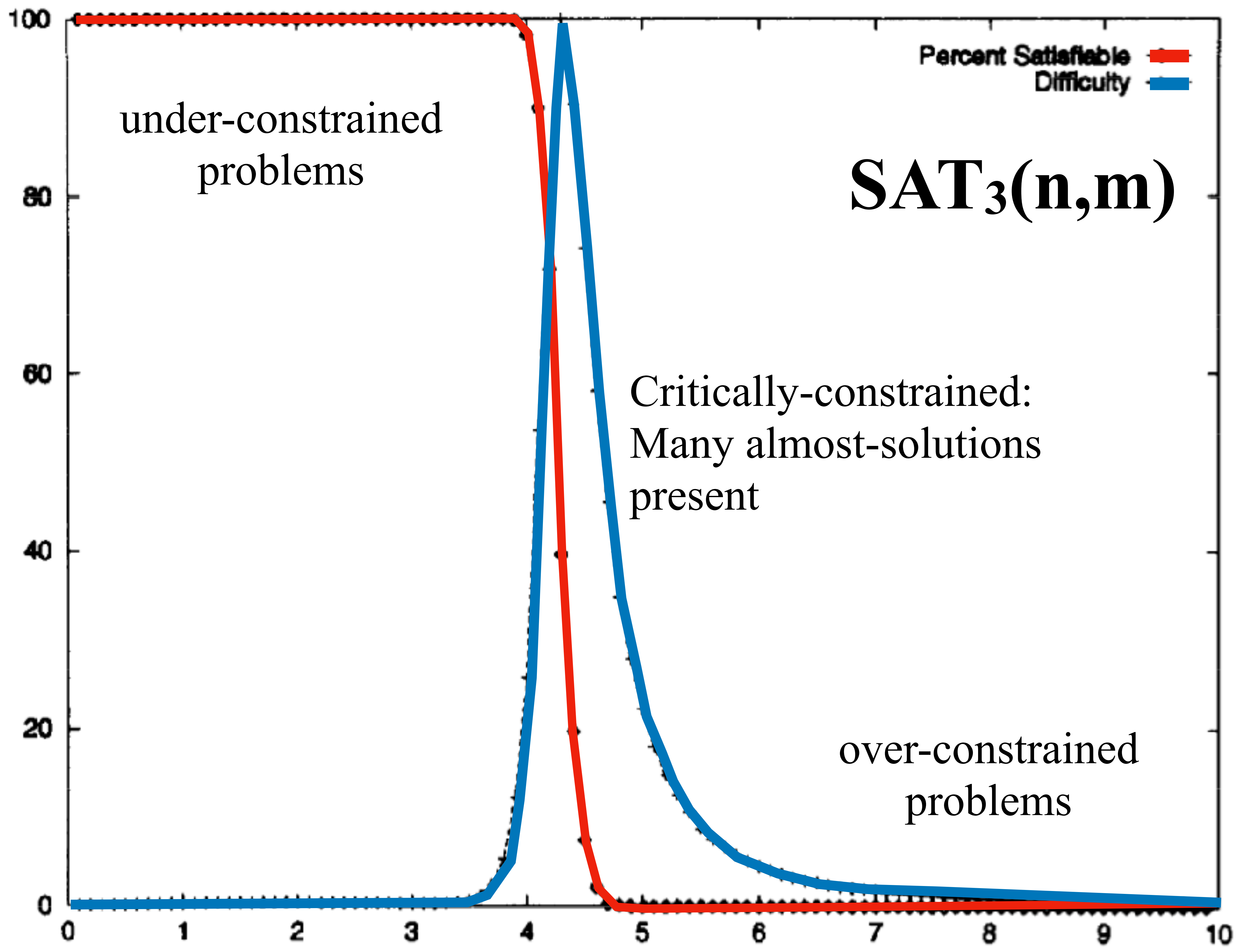
Phases display very different macroscopic properties while the “microscopic” structure is the same.

Phase transitions can happen also in algorithmic problems, as in the case of SAT!

Phase Transition in SAT

Usually the first backtracking path is successful

$\alpha \sim 4.26$



α = n/m (Clause/Variables)

The absence of a solution can be determined early in the search.

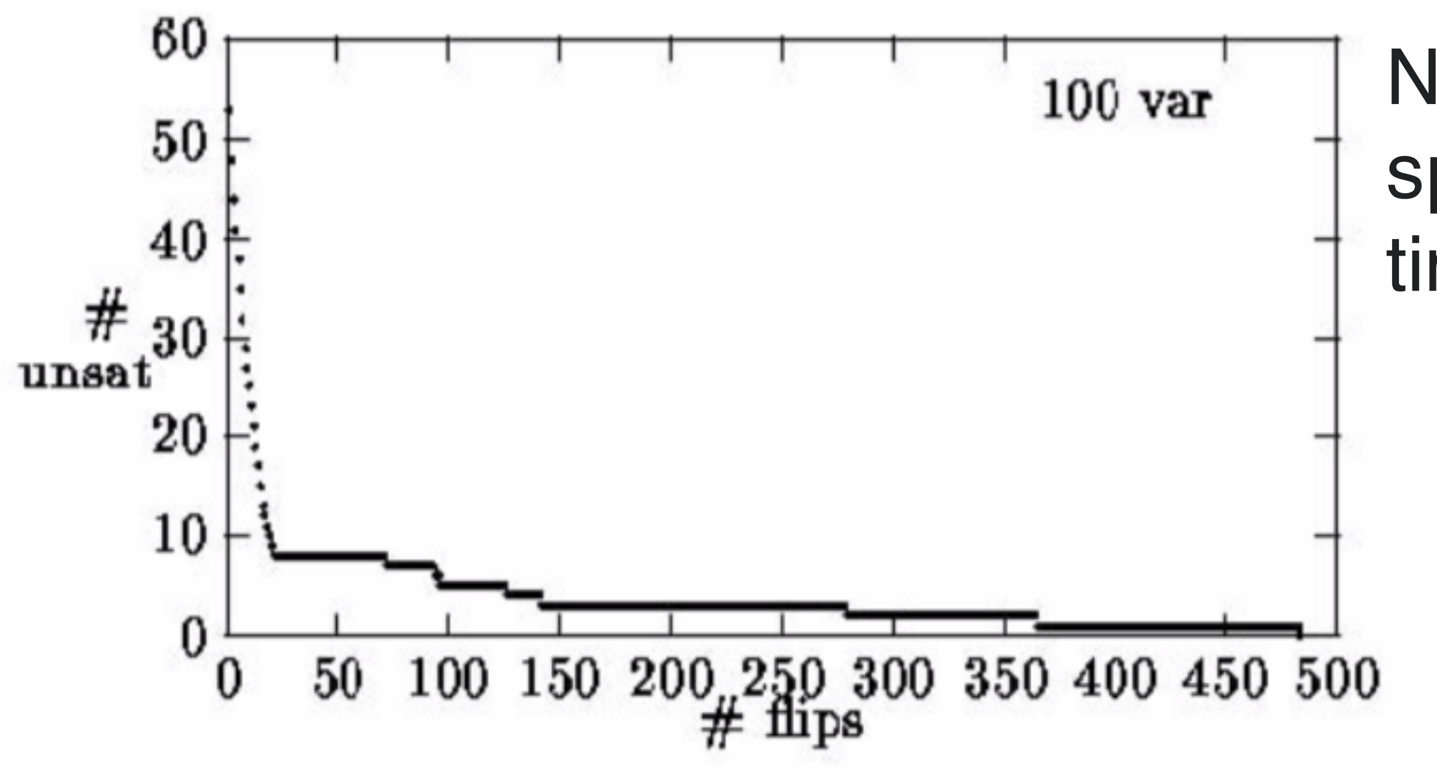
J.A.Crawford,L.D.Auton, Artif. Intell. 81,31 (1996)

SAT and Local Search

- SAT can be formulated as a local search problem:
- Make a local “move” and try to reach a better state.
- Needed: a function to minimise (evaluation function) associated to the problem.
- Example: Number of satisfied clauses.
- Problem: local maxima.
- In this context, many algorithms can be used (Simulated Annealing, (Reactive)Tabu Search, ….)

A local search algorithm: GSAT

```
function GSAT(clauses,Max-Flips, Max-Tries) returns truth assignment (if it exists)
    for i=1 to Max-Tries:
        T <— random truth assignment
        for j=1 to <Max-Flips:
            if T satisfies clauses then return T
            v <— propositional variable such that a change in its truth assignment
            gives the large increase in the # of clauses satisfied by T.
            T <— T with truth assignment v reversed.
    return "no assignment found"
```



Notice how GSAT spends a lot of time on plateaus.

Another variant: **WalkSAT.** Here the choice of which variable to flip is different. First picks a random clause which is unsatisfied by the current assignment T, then flips a variable within that clause. The variable is picked that will result in the **fewest previously satisfied clauses becoming unsatisfied**, with some probability of picking one of the variables at random. When picking a guessed-to-be-optimal variable, WalkSAT has to do less calculation than GSAT because it is considering fewer possibilities.

Practical Improvements of DPLL

Many ideas come from CSP algorithms:

Component Analysis: If we discover sets of clauses which do not share unassigned variables, we can decompose the original problem in two (or more) disjoined ones.

Variable/Value Ordering: Try first the unassigned variable which appears more frequently in the remaining clauses (degree heuristic).

Intelligent Backtracking: Backtrack directly to the source of conflicts. Frequently used: conflict clause learning (for not repeating the same "mistake" again).

Random restart: (remember hill climbing algorithms) in case of conflict, restart from a random place in the previous search tree.

Clever Indexing: optimisation of the data structures indexing the different variables/ clauses.

SAT Status

- Since the beginning of the 90s: SAT competitions
- http://www.satcompetition.org/
- Largest instances > 10^7 variables
- Complete solvers dominate the performance ranking
- Incomplete local search solvers are best on random satisfiable instances
- Best solvers use meta-algorithmic methods like algorithm configuration/selection, ….

Summary

- Algorithms solving SAT make strong use of resolution
- DPLL combines resolution with backtracking
    - Very efficient implementation techniques
    - Smart branching heuristics
    - Clause learning