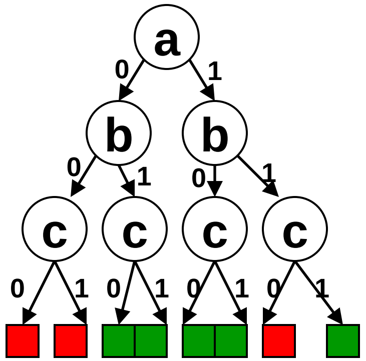
Satisfiability of Propositional Formulas

Mooly Sagiv

Slides by Sharad Malik, Ohad Shacham, Daniel Kroening and Ofer Strichman

The SAT Problem

- Given a propositional formula (Boolean function)
  - $\varphi = (a \lor b) \land (\neg a \lor \neg b \lor c)$
- Determine if φ is valid
- Determine if φ is satisfiable
  - Find a satisfying assignment or report that such does not exit
- For n variables, there are 2^n possible truth assignments to be checked



Why Bother?

- Core computational engine for major applications
  - Artificial Intelligence
    - Knowledge base deduction
    - Automatic theorem proving
  - Electronic Design Automaton
    - Testing and Verification
    - Logic synthesis
    - FPGA routing
    - Path delay analysis
    - And more…
  - Software Verification

Problem Representation

- Represent the formulas in Conjunctive Normal Form (CNF)
- Conversion to CNF is straightforward
  - $\mathbf{a \vee (b \wedge \neg(c \vee \neg d)) \equiv (a \vee (b \wedge \neg c \wedge \neg\neg d)) \equiv}$
    $\mathbf{(a \vee (b \wedge \neg c \wedge d)) \equiv (a \vee b) \wedge (a \vee \neg c) \wedge (a \vee d)}$
  - **May need to add variables**
- Notations
  - Literals
    - Variable or its negation
  - Clauses
    - Disjunction of literals
  - φ $\mathbf{= (a \vee b) \wedge (\neg a \vee \neg b \vee c) \equiv (a + b)(a' + b' + c)}$
- Advantages of CNF
  - Simple data structure
  - Compact
  - Compositional
  - All the clauses need to be satisfied

Complexity Results

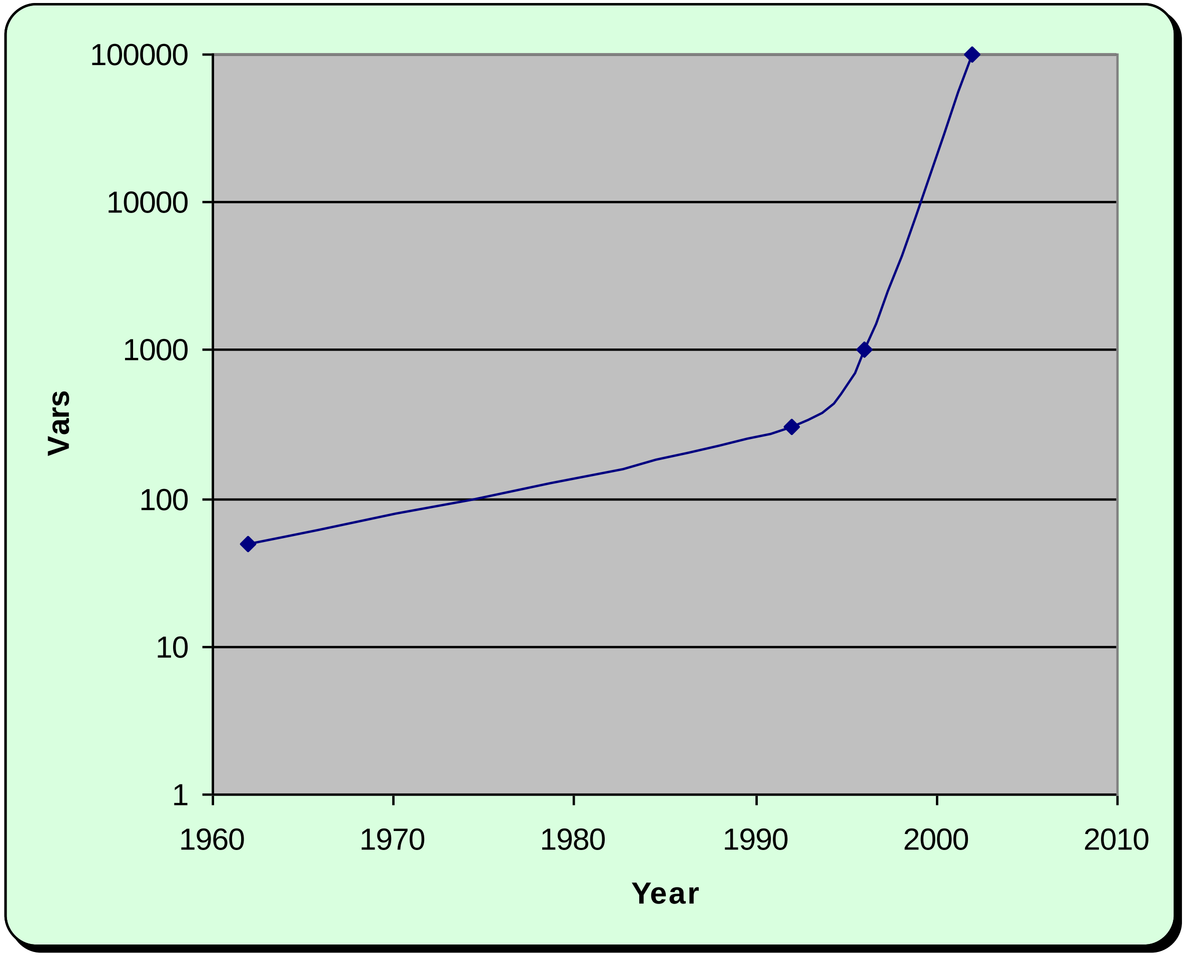
- First established NP-Complete problem
  - Even when at most 3 literals per clause (3-SAT)

  S. A. Cook, The complexity of theorem proving procedures, *Proceedings, Third Annual ACM Symp. on the Theory of Computing*,1971, 151-158
  - No polynomial algorithm for all instances unless P = NP
- Becomes polynomial when
  - At most two literals per clause (2-SAT)
  - At most one positive literal in every clause (Horn)

Goals

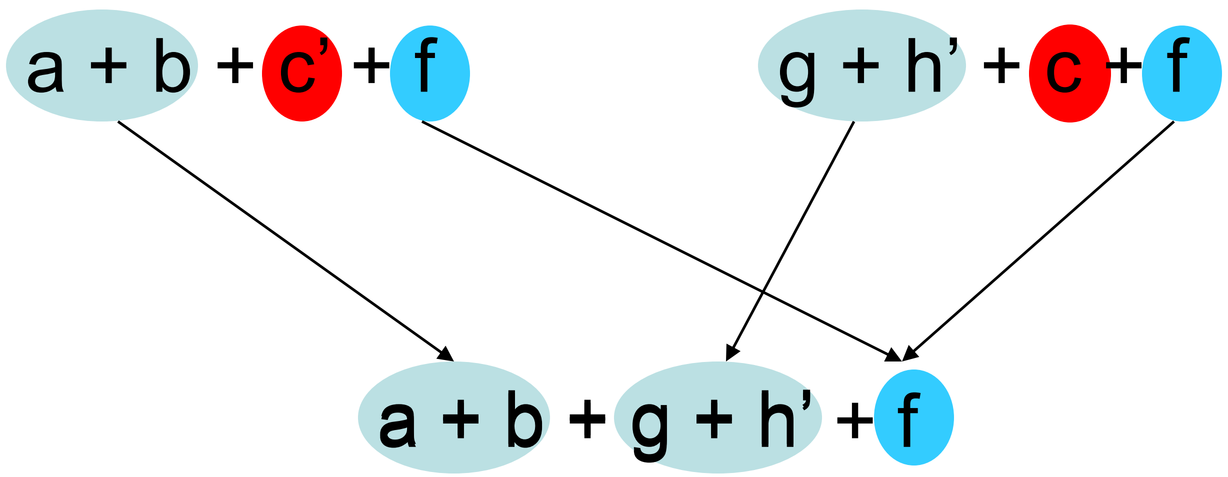
- Develop algorithms which solve all SAT instances
- Exponential worst case complexity
- But works well on many instances
  - Interesting Heuristics
  - Annual SAT conferences
  - SAT competitions
    - Randomly, Handmade, Industrial, AI
  - 10 Millions variables!

SAT made some progress…



Clause Resolution

- Resolution of a pair of clauses with exactly ONE incompatible variable

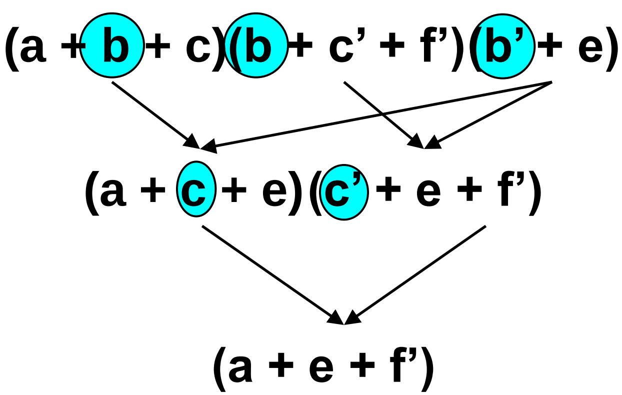


- What if more than one incompatible variables?

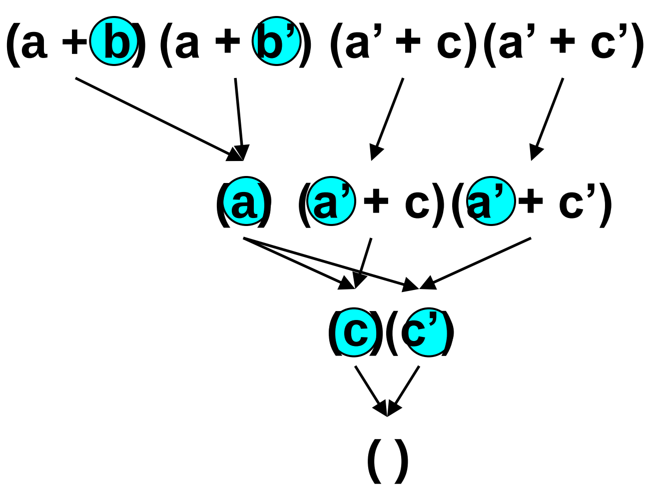
Davis Putnam Algorithm

M .Davis, H. Putnam, “A computing procedure for quantification theory", *J. of ACM*, Vol. 7, pp. 201-214, 1960

- Iteratively select a variable for resolution till no more variables are left
- Report UNSAT when the empty clause occurs
- Can discard resolved clauses after each iteration



SAT



UNSAT

Potential memory explosion problem!

Can we avoid using exponential space?

DLL Algorithm

- Davis, Logemann and Loveland

  M. Davis, G. Logemann and D. Loveland, “A Machine Program for Theorem-Proving", *Communications of ACM*, Vol. 5, No. 7, pp. 394-397, 1962

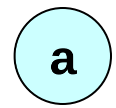
- Basic framework for many modern SAT solvers
- Also known as DPLL for historical reasons

Basic DLL Procedure - DFS

$(a' + b + c)$
$(a + c + d)$
$(a + c + d')$
$(a + c' + d)$
$(a + c' + d')$
$(b' + c' + d)$
$(a' + b + c')$
$(a' + b' + c)$

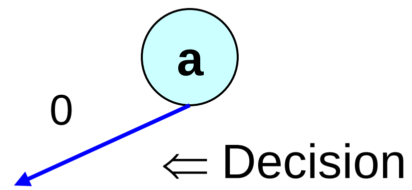
Basic DLL Procedure - DFS

(a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)



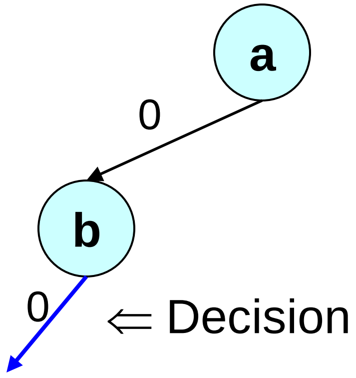
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)



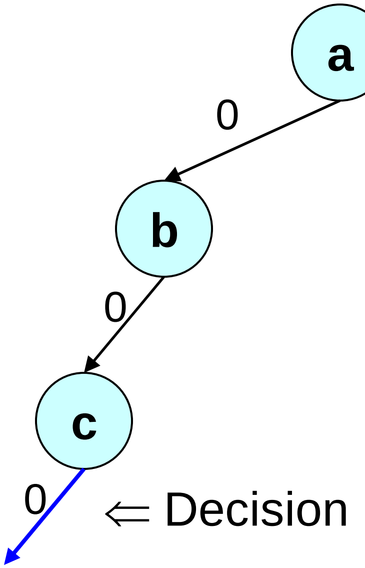
Basic DLL Procedure - DFS

(a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)



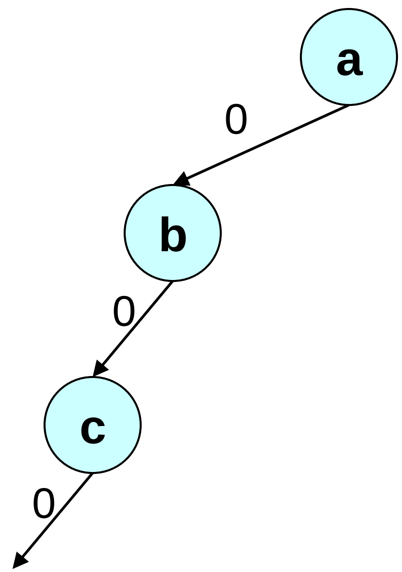
Basic DLL Procedure - DFS

(a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)

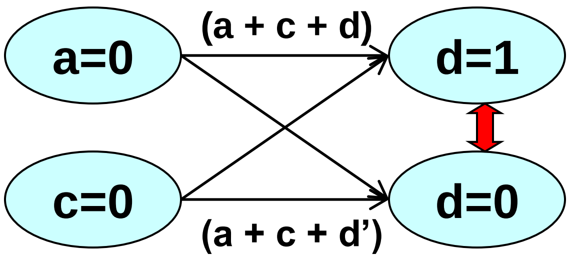


Basic DLL Procedure - DFS

(a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)



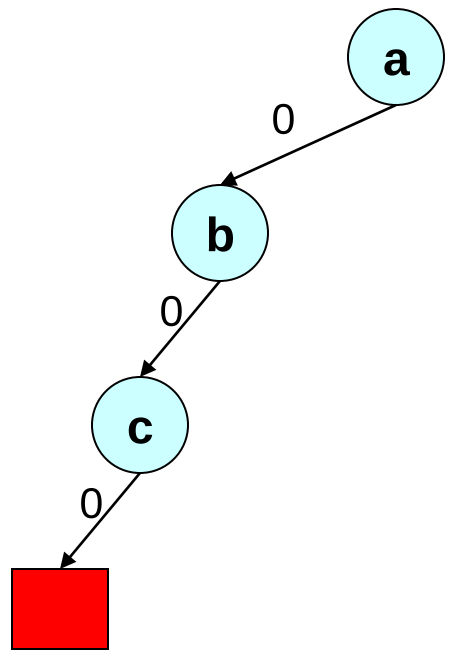
Implication Graph



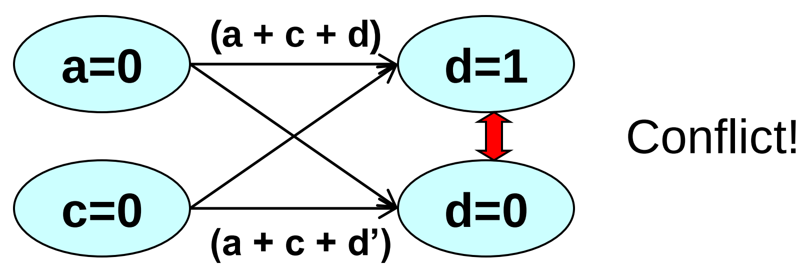
Conflict!

Basic DLL Procedure - DFS

(a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)

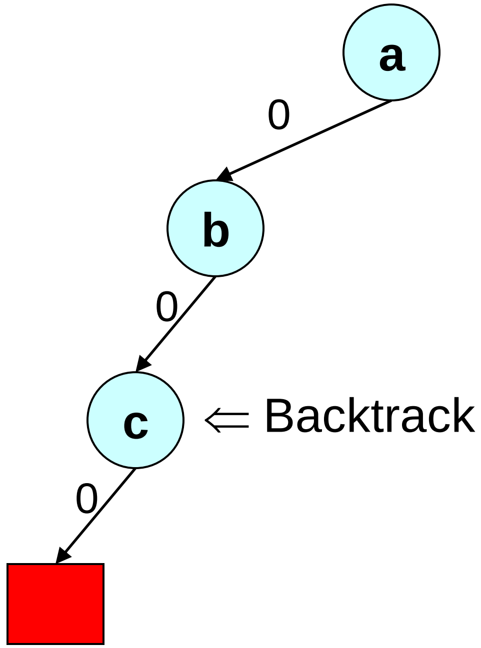


Implication Graph



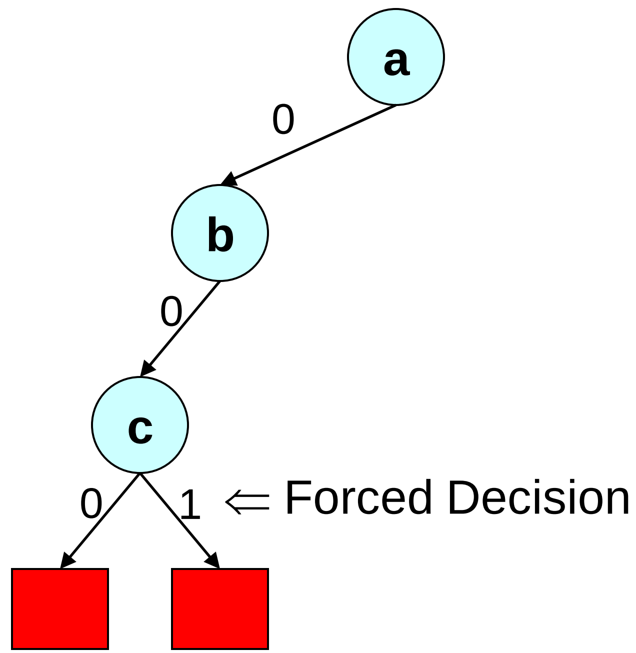
Basic DLL Procedure - DFS

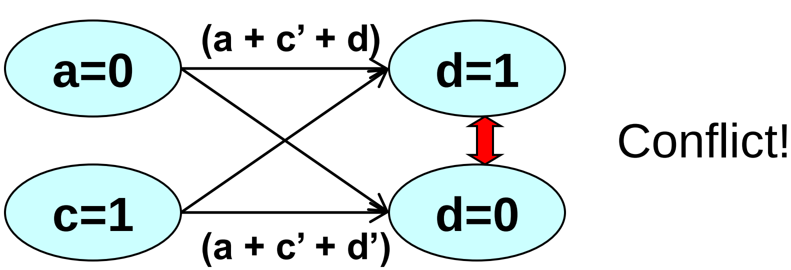
(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)



Basic DLL Procedure - DFS

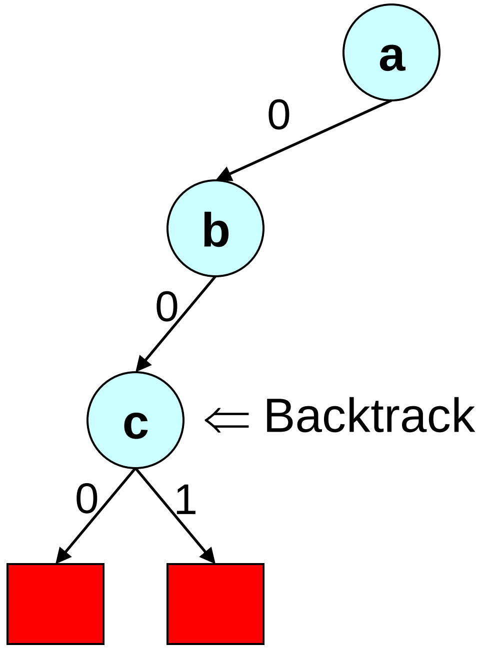
(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)





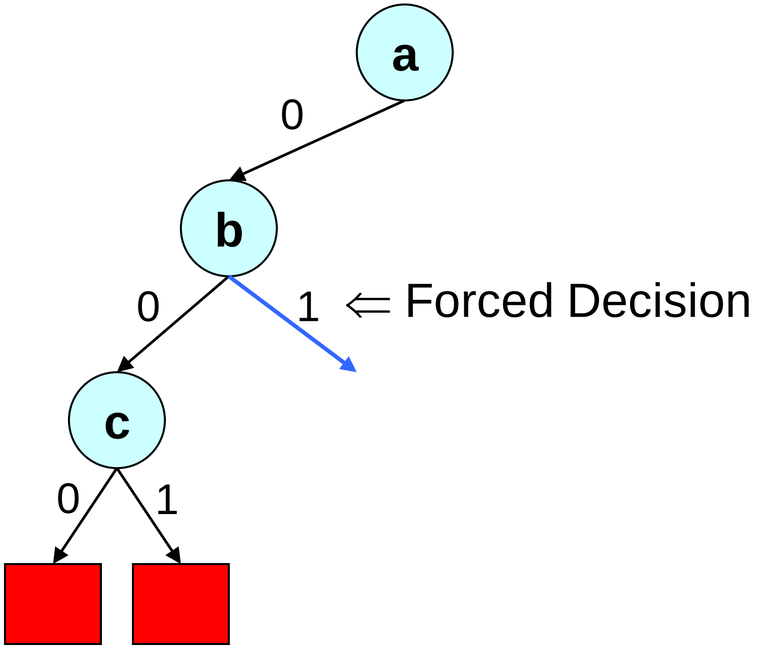
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)



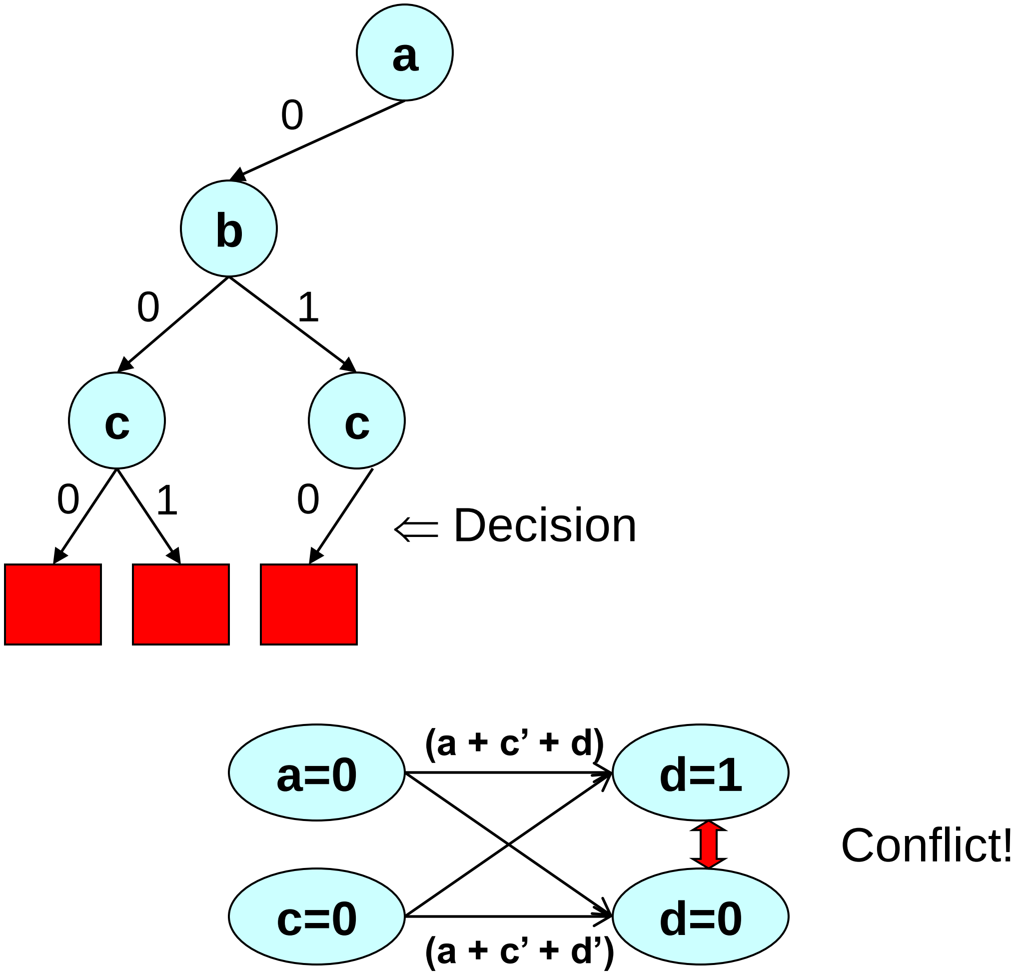
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)



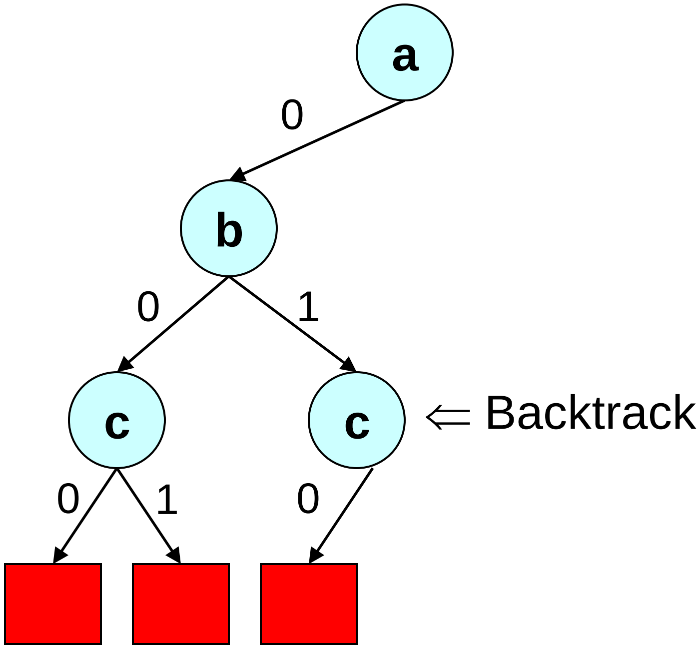
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)



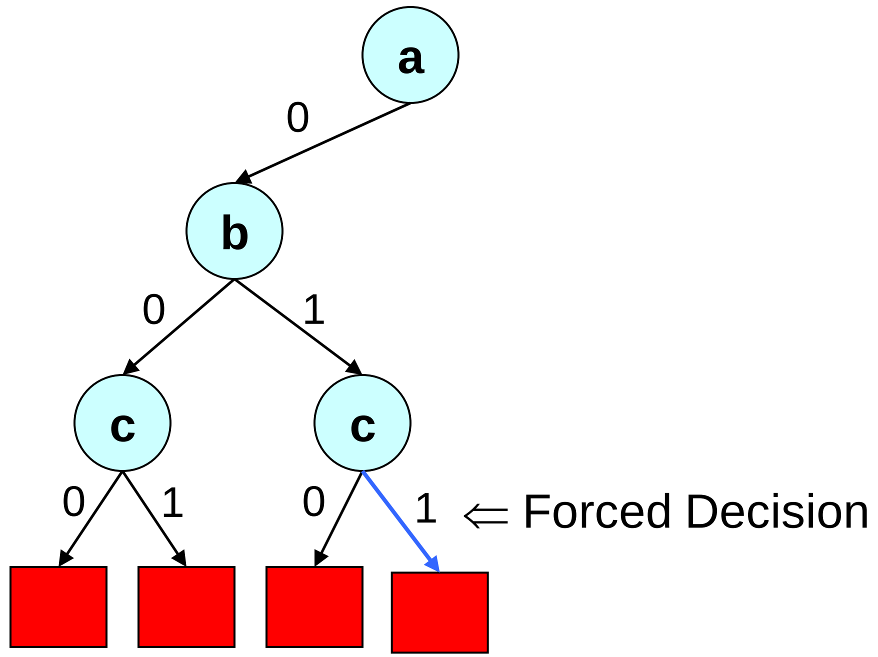
Basic DLL Procedure - DFS

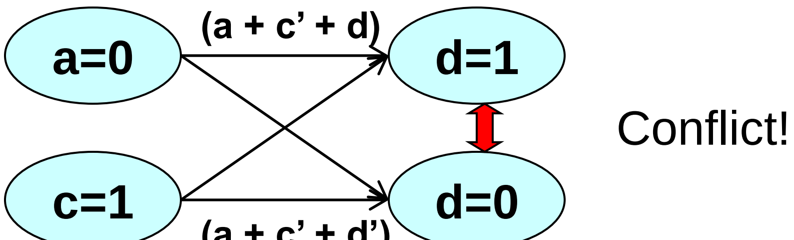
(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)



Basic DLL Procedure - DFS

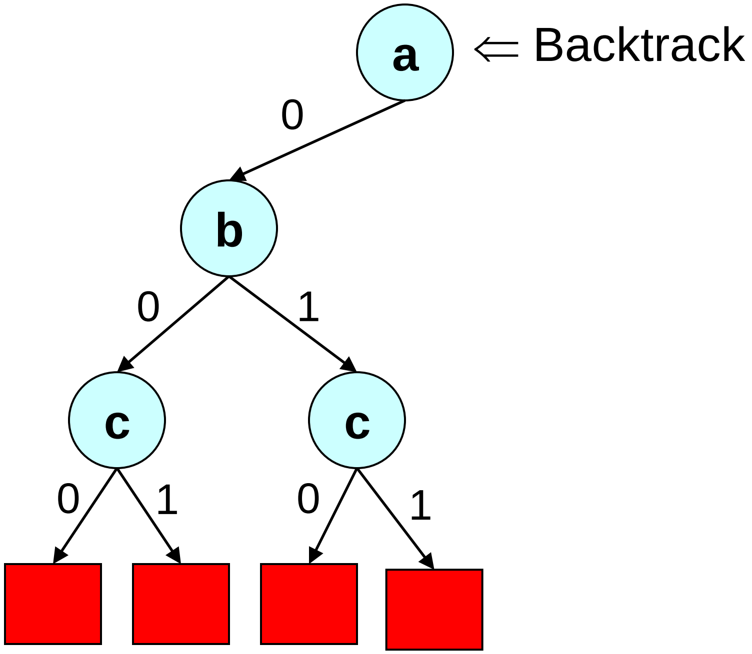
(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)





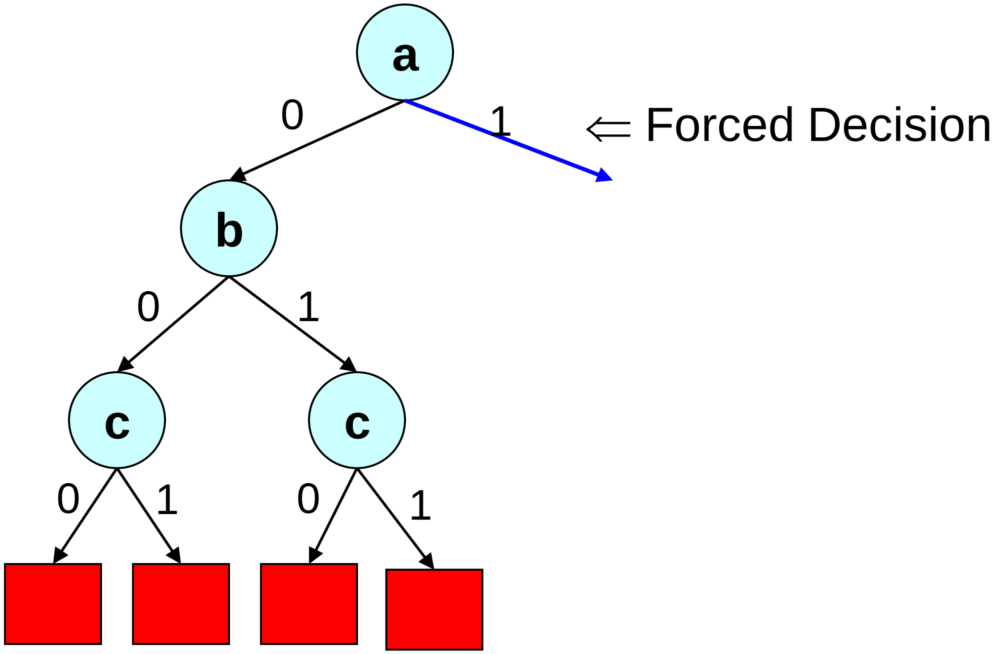
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)



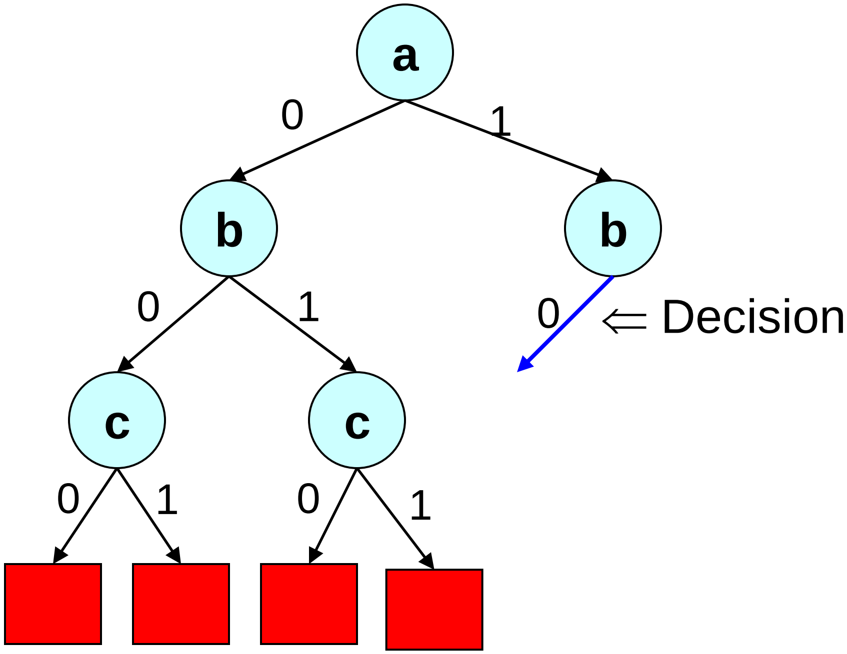
Basic DLL Procedure - DFS

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)



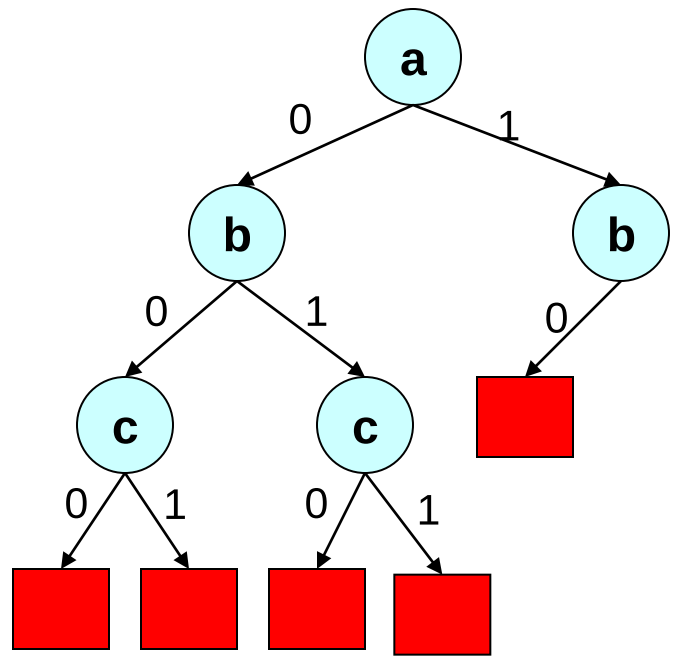
Basic DLL Procedure - DFS

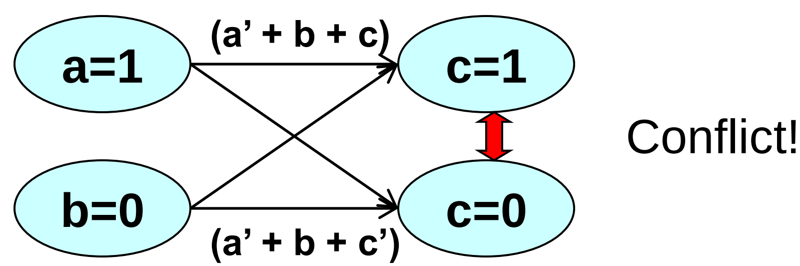
(a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)



Basic DLL Procedure - DFS

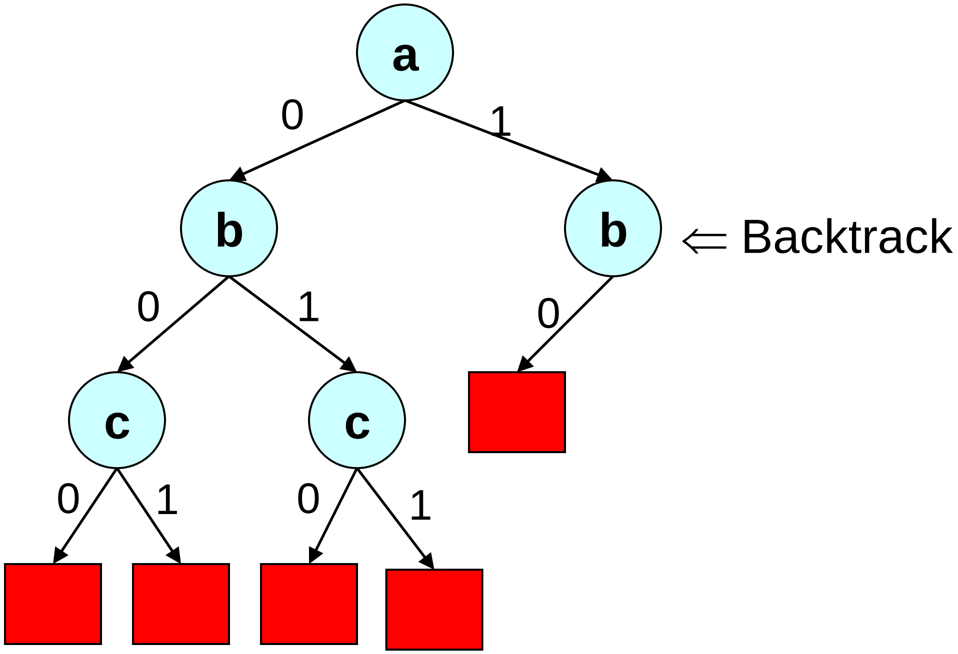
(a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)





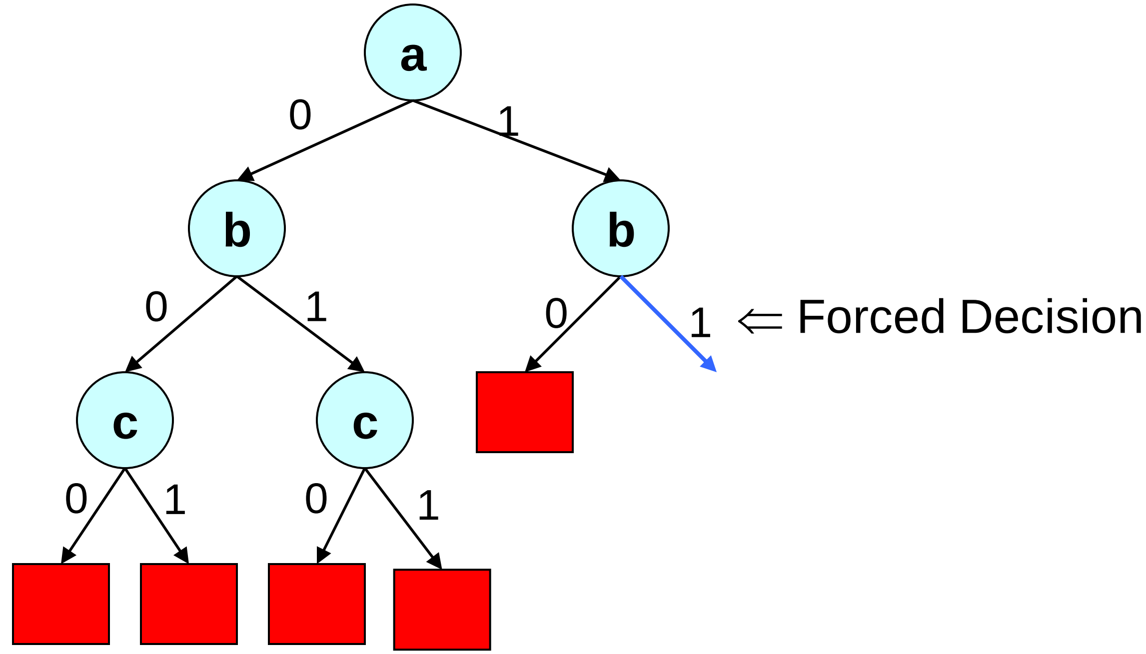
Basic DLL Procedure - DFS

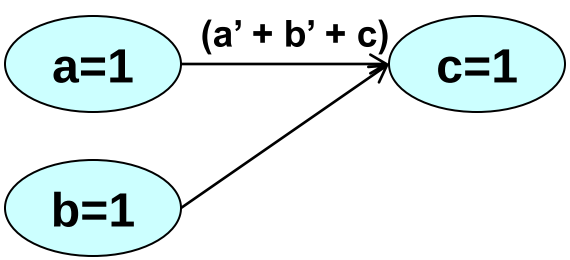
(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)



Basic DLL Procedure - DFS

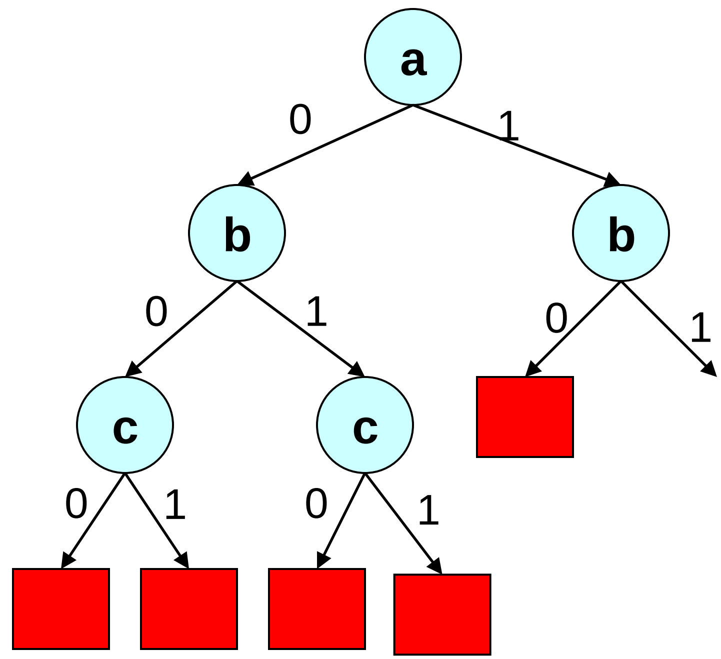
(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)

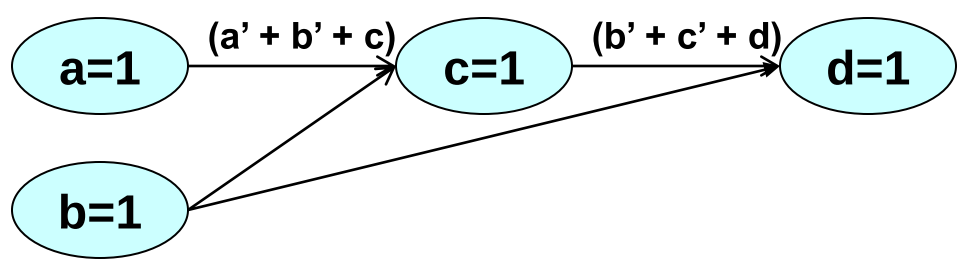




Basic DLL Procedure - DFS

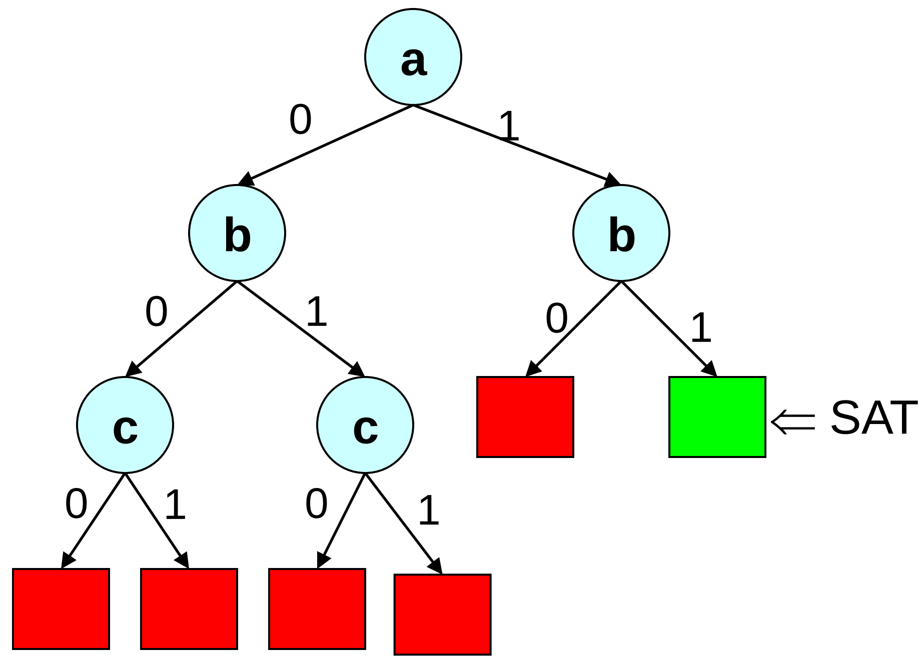
(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)

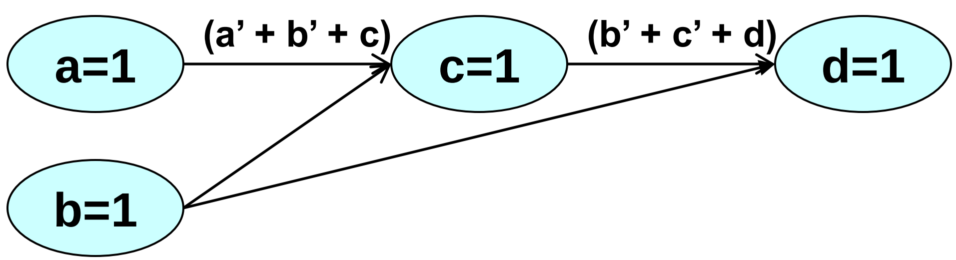




Basic DLL Procedure - DFS

(a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)





Implications and Boolean Constraint Propagation

- Implication
  - A variable is forced to be assigned to be True or False based on previous assignments
- **Unit** clause rule (rule for elimination of one literal clauses)
  - An unsatisfied clause is a unit clause if it has exactly one unassigned literal

$$(a + b' + c)(b + c')(a' + c')$$

a = T, b = T, c is unassigned

Satisfied Literal

Unsatisfied Literal

Unassigned Literal

  - The unassigned literal is implied because of the unit clause
- Boolean Constraint Propagation (BCP)
  - Iteratively apply the unit clause rule until there is no unit clause available
- Workhorse of DLL based algorithms

A Basic SAT algorithm

```
While (true)
{
        if (!Decide()) return (SAT)
        while (!BCP())
                if (!Resolve_Conflict()) return (UNSAT)
}
```

Choose the next variable and value. Return False if all variables are assigned

Apply repeatedly the *unit clause rule*. Return False if reached a conflict

Backtrack until no conflict. Return False if impossible

Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3’ + x8’

x1 + x8 + x12

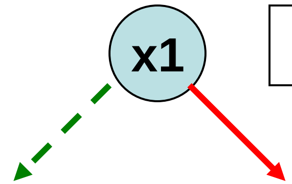
x2 + x11

x7’ + x3’ + x9

x7’ + x8 + x9’

x7 + x8 + x10’

x7 + x10 + x12’



x1=0

x1=0

Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3’ + x8’

x1 + x8 + x12

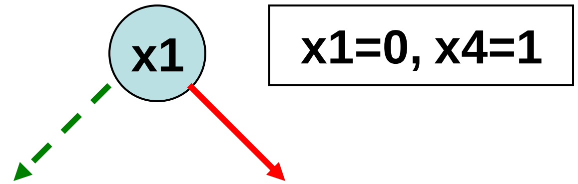
x2 + x11

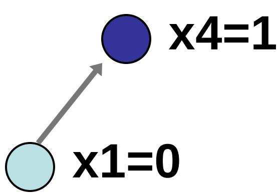
x7’ + x3’ + x9

x7’ + x8 + x9’

x7 + x8 + x10’

x7 + x10 + x12’





Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3’ + x8’

x1 + x8 + x12

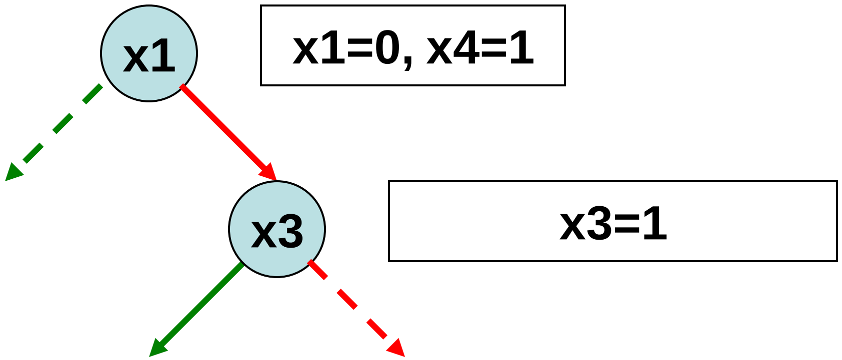
x2 + x11

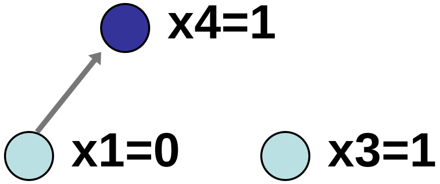
x7’ + x3’ + x9

x7’ + x8 + x9’

x7 + x8 + x10’

x7 + x10 + x12’





Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3' + x8'

x1 + x8 + x12

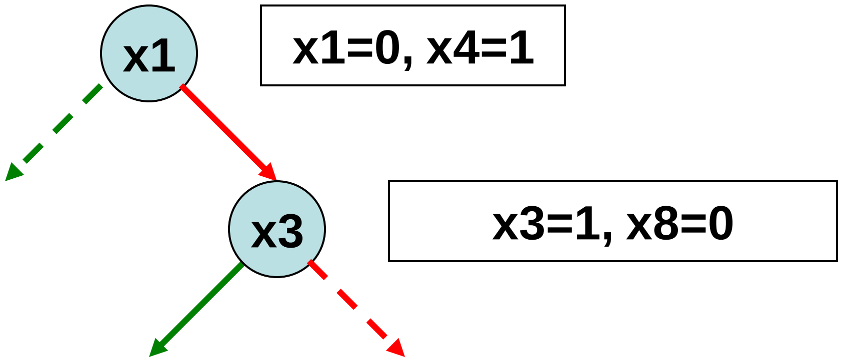
x2 + x11

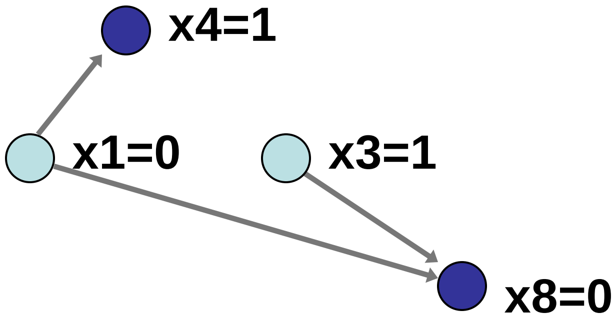
x7' + x3' + x9

x7' + x8 + x9'

x7 + x8 + x10'

x7 + x10 + x12'





Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3' + x8'

x1 + x8 + x12

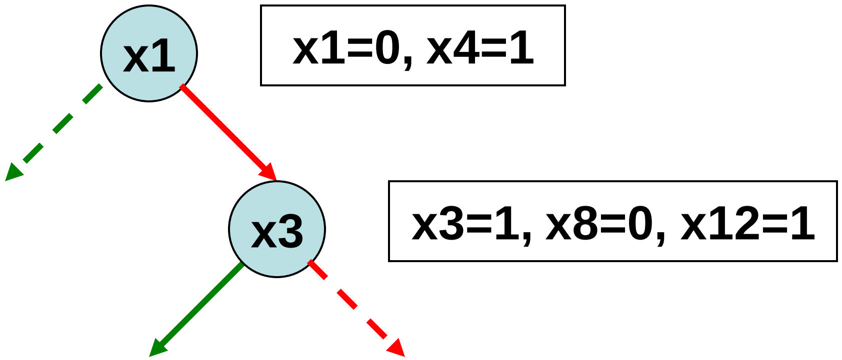
x2 + x11

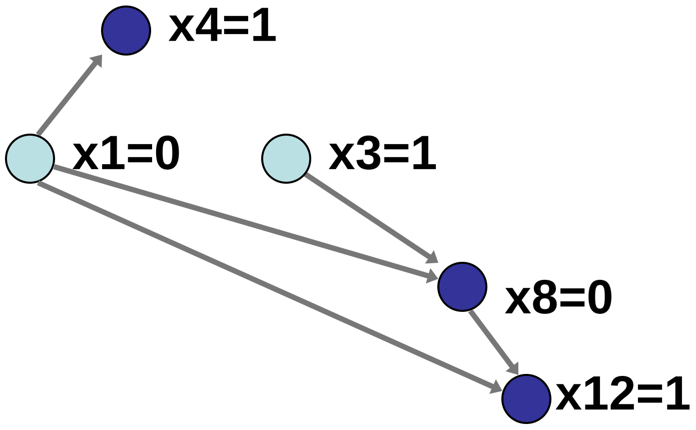
x7' + x3' + x9

x7' + x8 + x9'

x7 + x8 + x10'

x7 + x10 + x12'





Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3' + x8'

x1 + x8 + x12

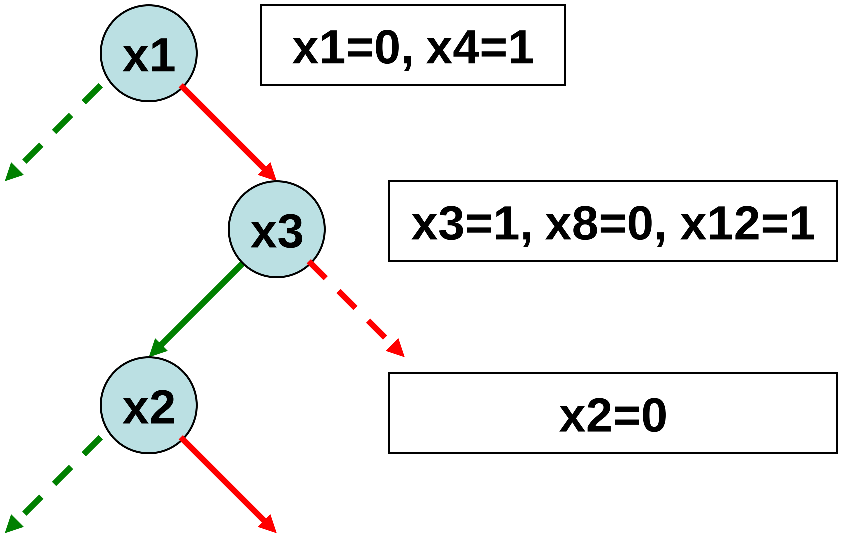
x2 + x11

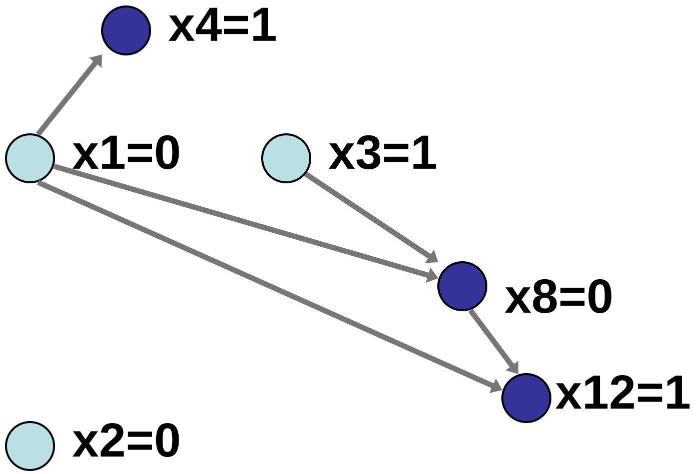
x7' + x3' + x9

x7' + x8 + x9'

x7 + x8 + x10'

x7 + x10 + x12'





Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3’ + x8’

x1 + x8 + x12

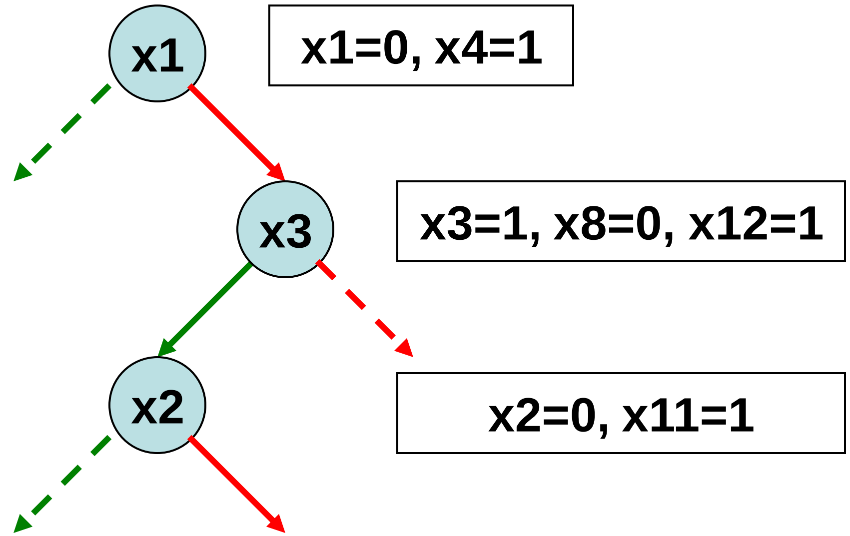
x2 + x11

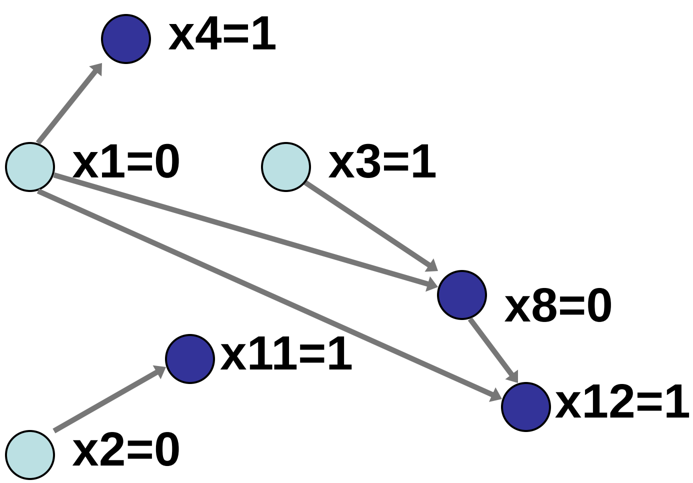
x7’ + x3’ + x9

x7’ + x8 + x9’

x7 + x8 + x10’

x7 + x10 + x12’





Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3' + x8'

x1 + x8 + x12

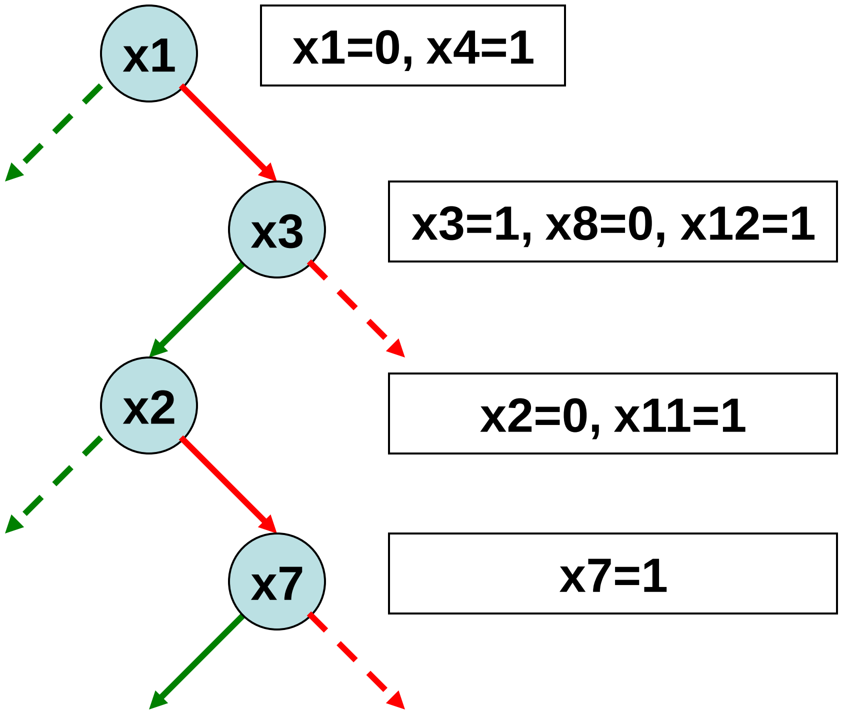
x2 + x11

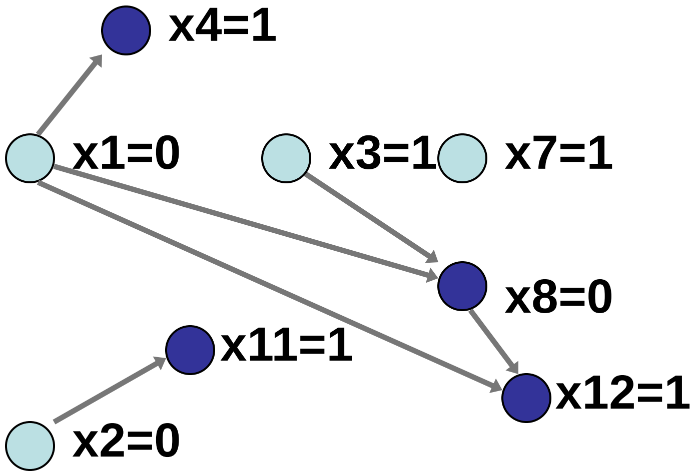
x7' + x3' + x9

x7' + x8 + x9'

x7 + x8 + x10'

x7 + x10 + x12'





Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3’ + x8’

x1 + x8 + x12

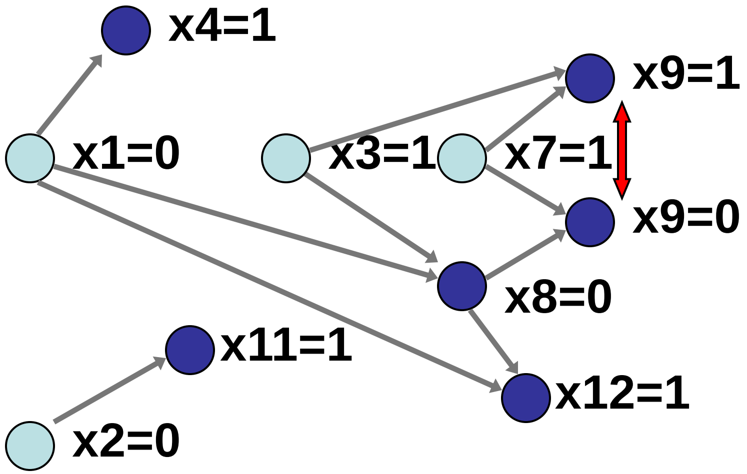
x2 + x11

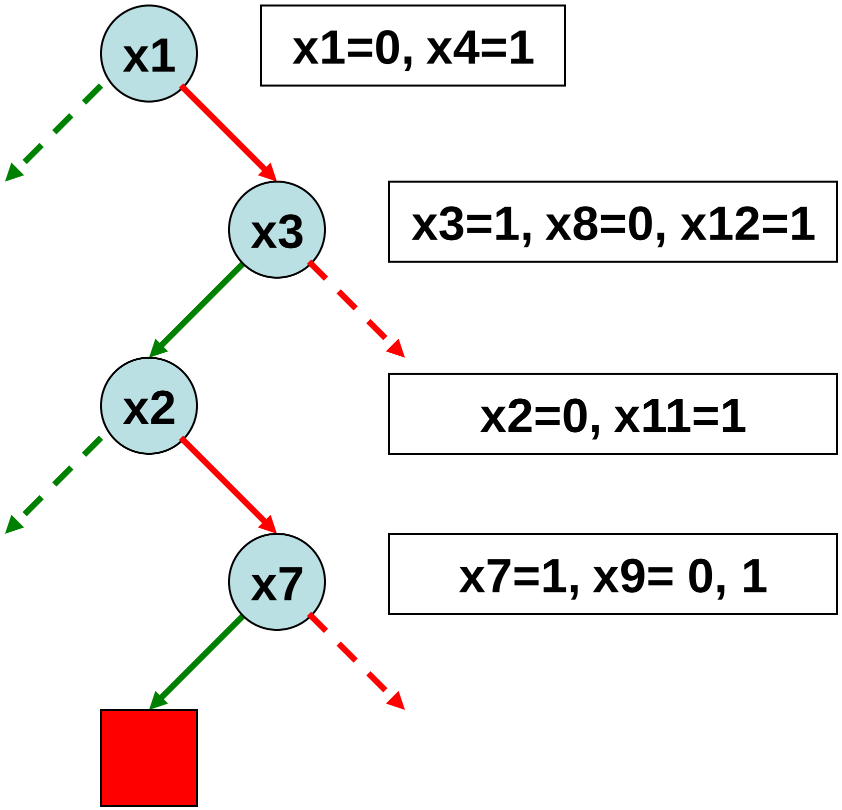
x7’ + x3’ + x9

x7’ + x8 + x9’

x7 + x8 + x10’

x7 + x10 + x12’





Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3' + x8'

x1 + x8 + x12

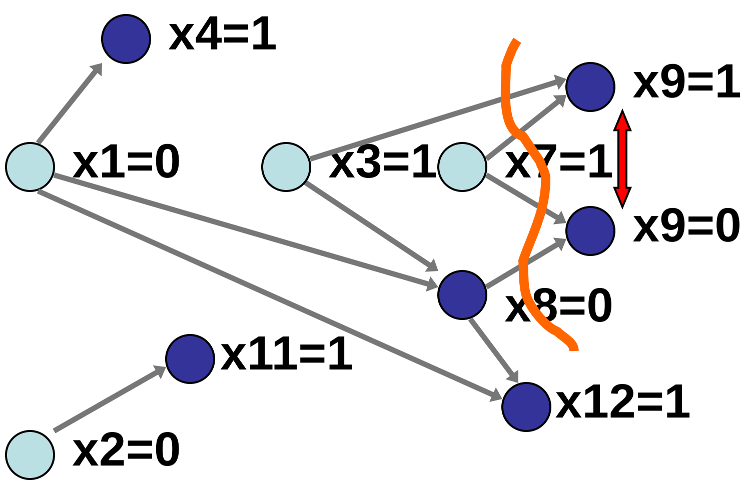
x2 + x11

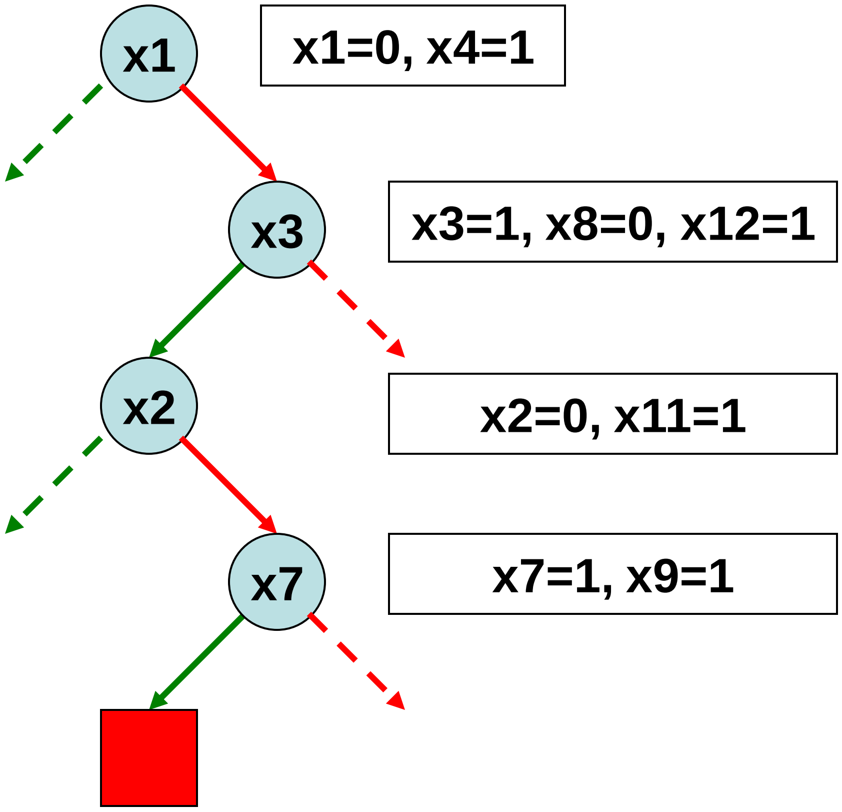
x7' + x3' + x9

x7' + x8 + x9'

x7 + x8 + x10'

x7 + x10 + x12'





x3=1∧x7=1∧x8=0 → conflict

Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3’ + x8’

x1 + x8 + x12

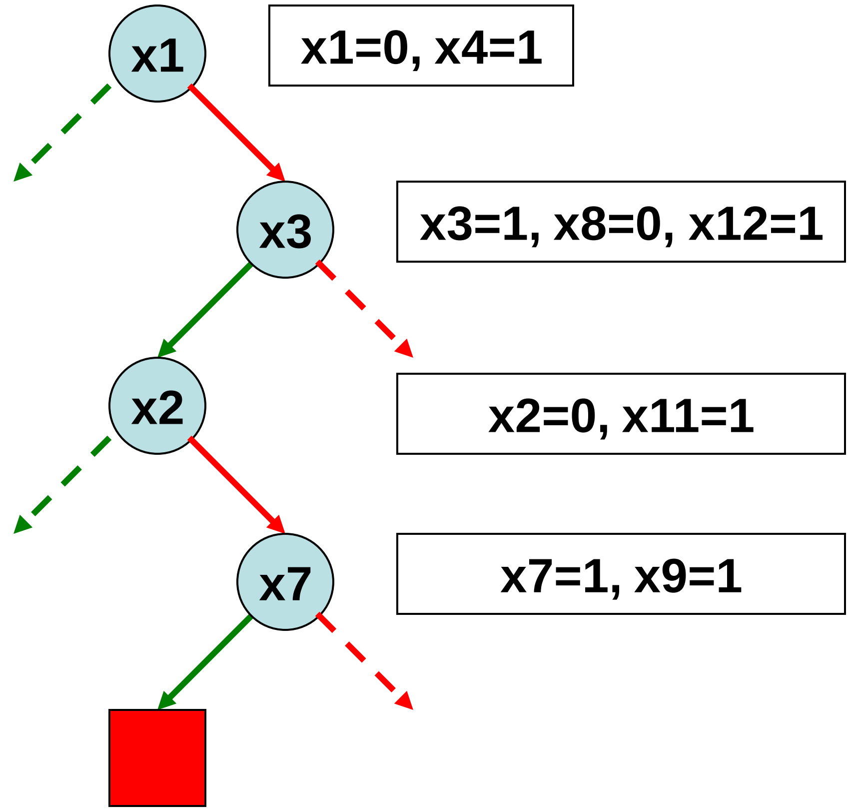
x2 + x11

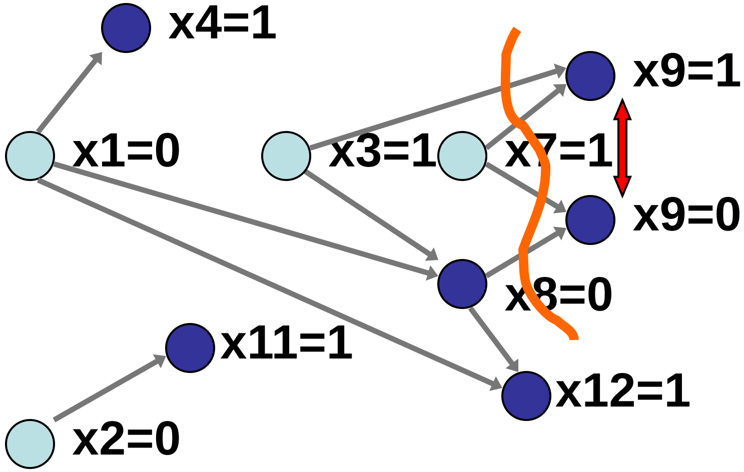
x7’ + x3’ + x9

x7’ + x8 + x9’

x7 + x8 + x10’

x7 + x10 + x12’





$x3=1 \wedge x7=1 \wedge x8=0 \rightarrow$ conflict
Add conflict clause: x3’+x7’+x8

Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3' + x8'

x1 + x8 + x12

x2 + x11

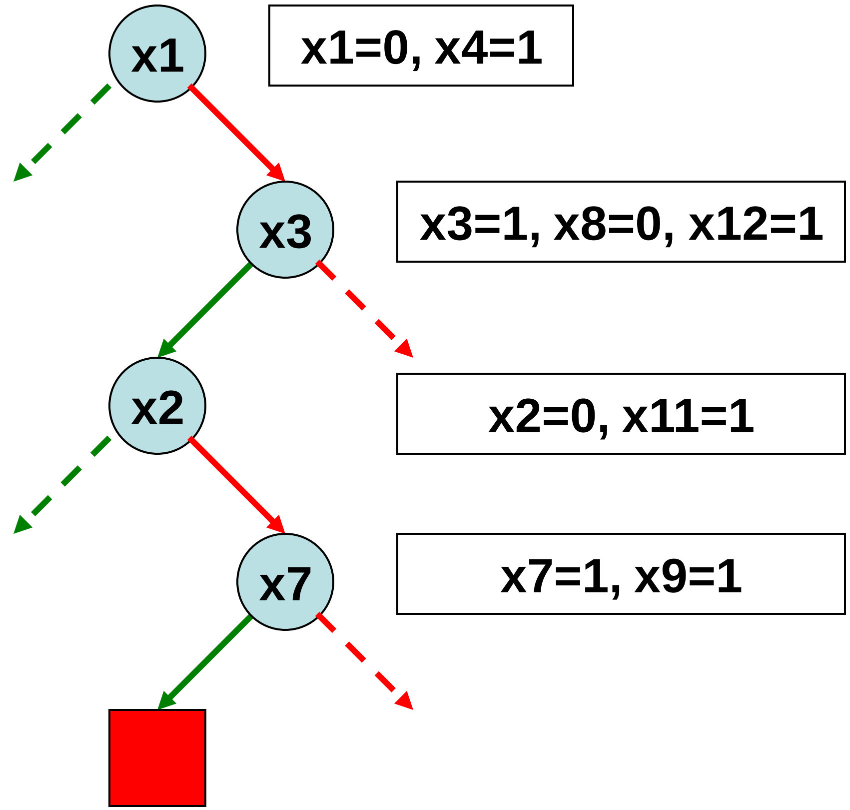
x7' + x3' + x9

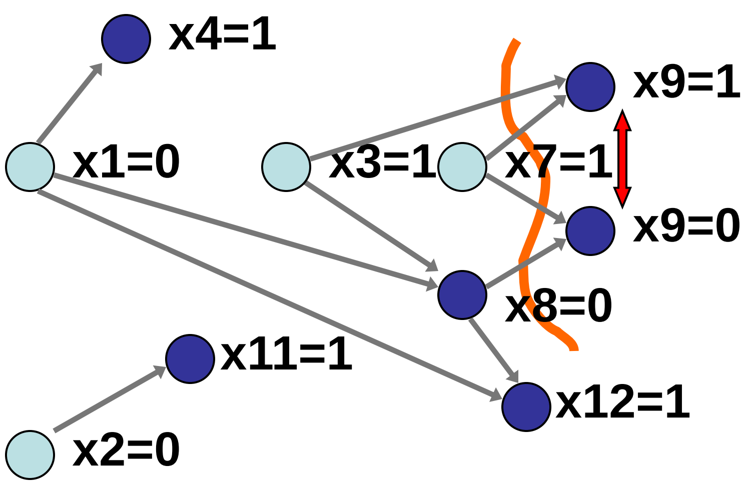
x7' + x8 + x9'

x3'+x7'+x8

x7 + x8 + x10'

x7 + x10 + x12'





x3=1∧x7=1∧x8=0 → conflict
Add conflict clause: x3'+x7'+x8

Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3’ + x8’

x1 + x8 + x12

x2 + x11

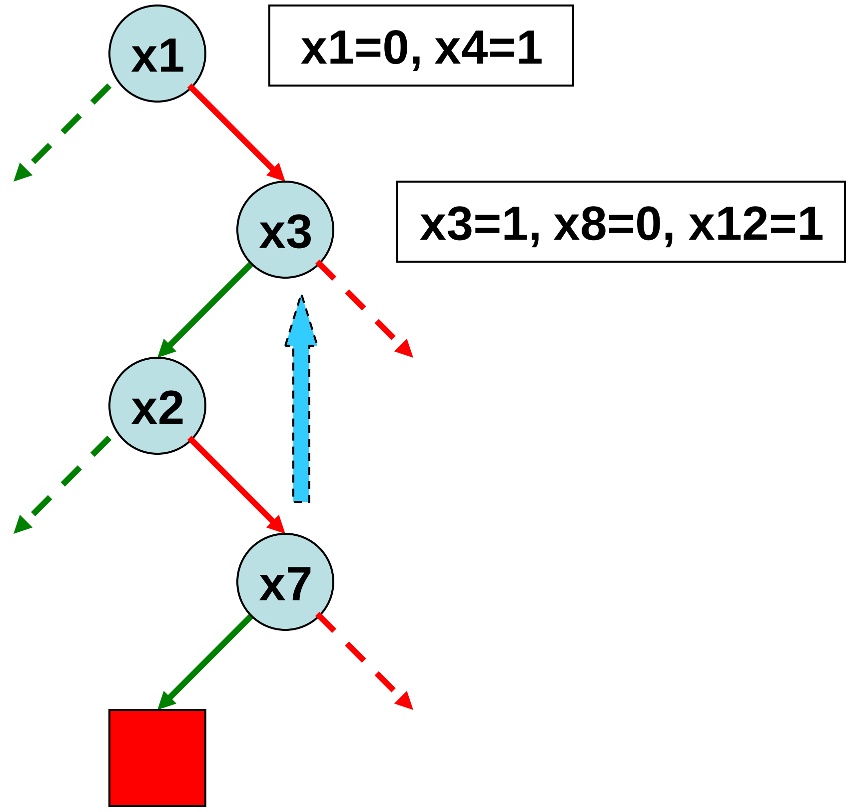
x7’ + x3’ + x9

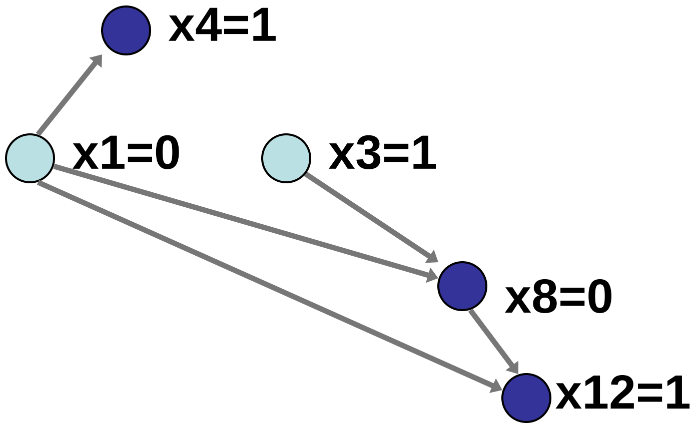
x7’ + x8 + x9’

x7 + x8 + x10’

x7 + x10 + x12’

x3’ + x7’ + x8





Backtrack to the decision level of x3=1
x7 = 0

Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3’ + x8’

x1 + x8 + x12

x2 + x11

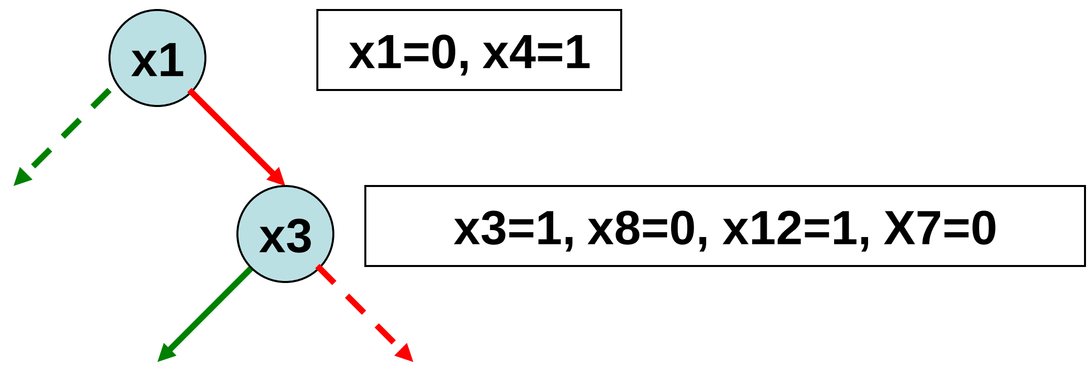
x7’ + x3’ + x9

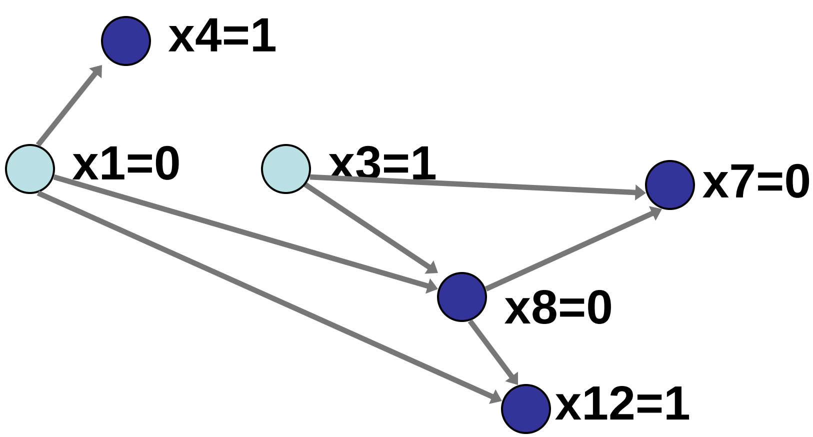
x7’ + x8 + x9’

x7 + x8 + x10’

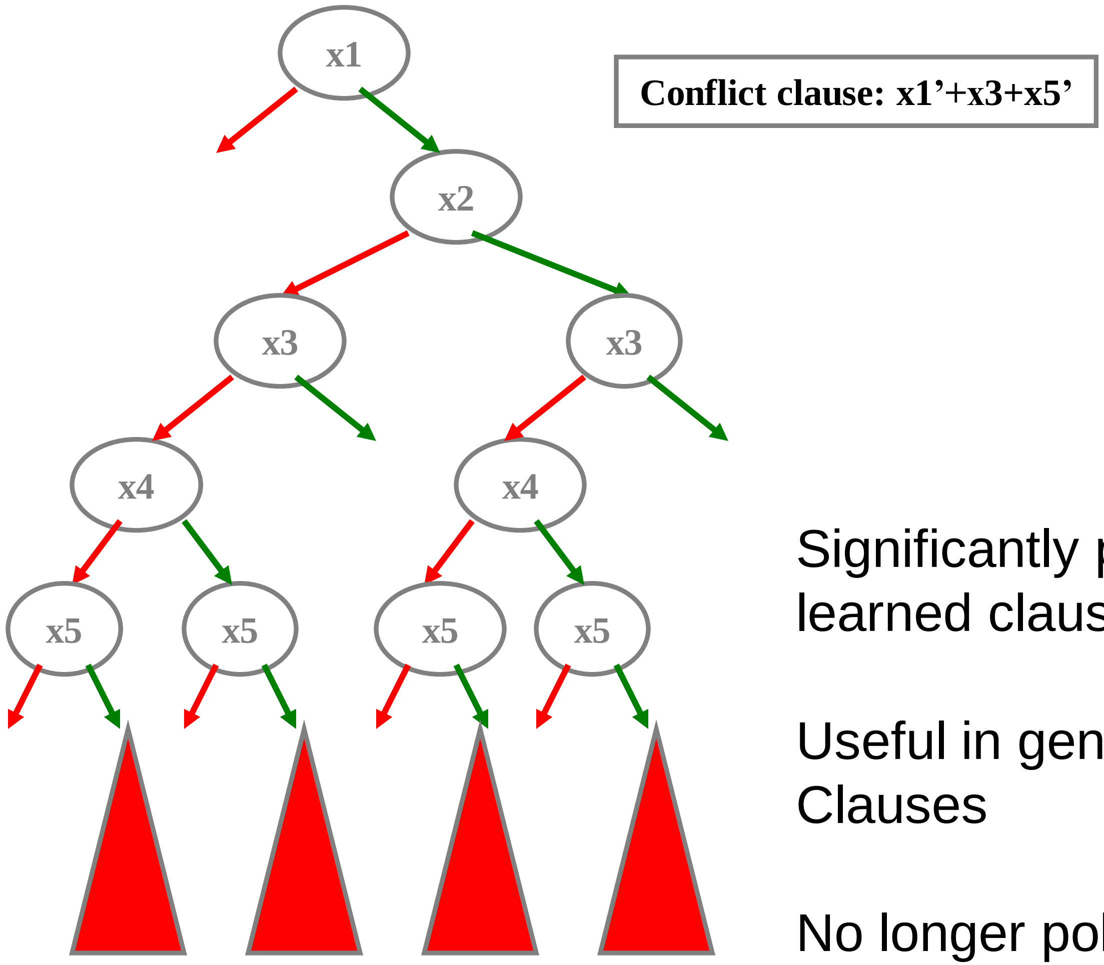
x7 + x10 + x12’

x3’ + x7 + x8’





What's the big deal?



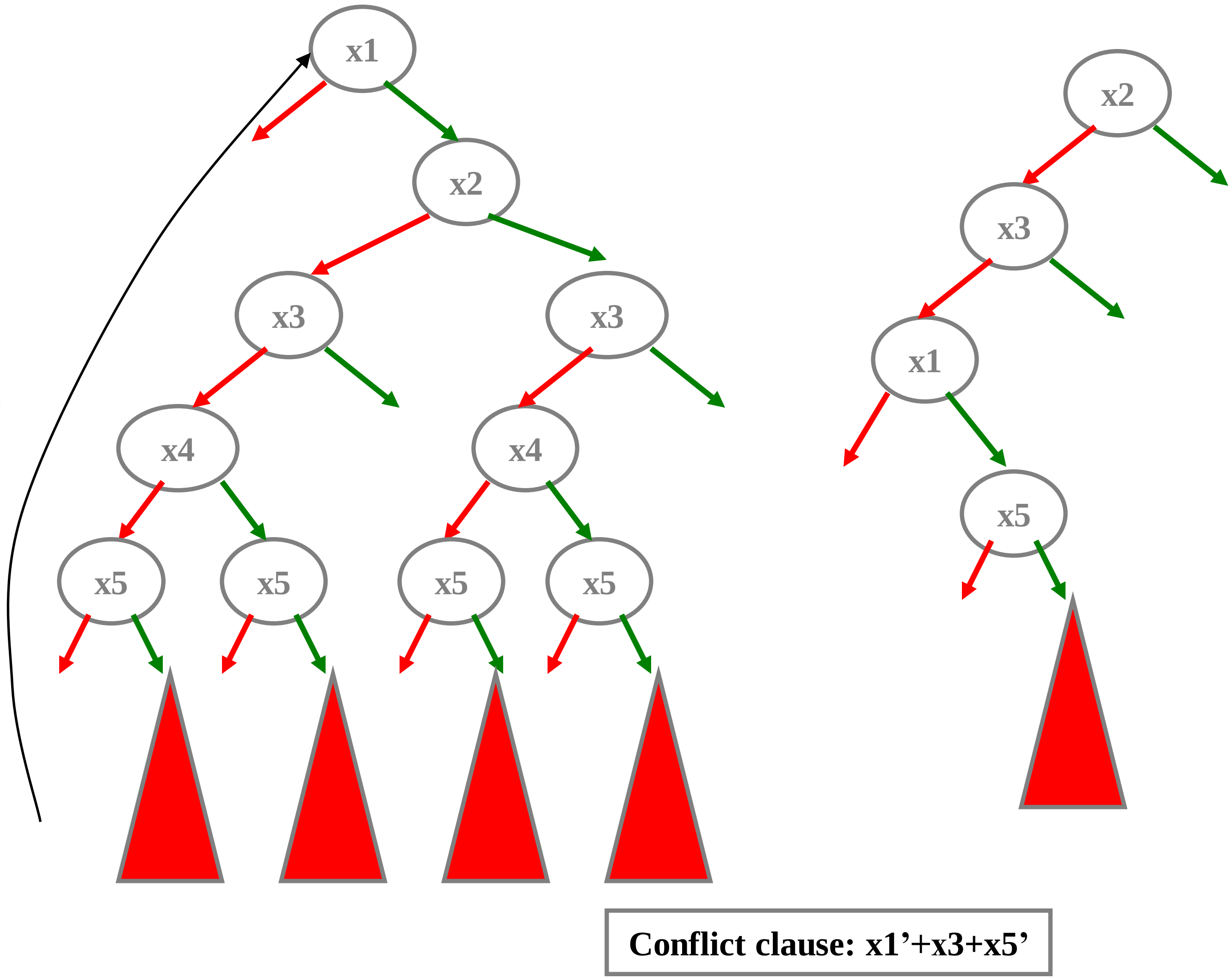
Significantly prune the search space – learned clause is useful forever!

Useful in generating future conflict Clauses

No longer polynomial space

Restart

- Abandon the current search tree and reconstruct a new one
- The clauses learned prior to the restart are *still there* after the restart and can help pruning the search space
- Adds to robustness in the solver



BCP Algorithm

- What “causes” an implication? When can it occur?
  - All literals in a clause but one are assigned to F
    - (v1 + v2 + v3): implied cases: (0 + 0 + v3) or (0 + v2 + 0) or (v1 + 0 + 0)
  - For an N-literal clause, this can only occur after N-1 of the literals have been assigned to F
  - So, (theoretically) we could completely ignore the first N-2 assignments to this clause
  - In reality, we pick two literals in each clause to “watch” and thus can ignore any assignments to the other literals in the clause
    - Example: (v1 + v2 + v3 + v4 + v5)
    - (**v1=X + v2=X** + v3=? {i.e. X or 0 or 1} + v4=? + v5=?)

Chaff Decision Heuristic - VSIDS

- Variable State Independent Decaying Sum
  - Rank variables by literal count in the initial clause database
  - Periodically, divide all counts by a constant
  - Only increment counts as new clauses are added
- Quasi-static:
  - Static because it doesn’t depend on var state
  - Not static because it gradually changes as new clauses are added
    - Decay causes bias toward *recent* conflicts

Finding a Solution to a SAT problem is can be viewed as 2 player game

- Player 1: tries to find satisfying assignment
- Player 2: tries to show that such assignment does not exist

```
Let A be an arbitrary assignment
while true:
    if A ⊨ C then return SAT
    if () ∈ C then return UNSAT

  let c ∈ C such that not A ⊨ c  and let A' such that A' ⊨ c
        A := A'
  ||
  let c' ∉ C such that C ⊨ c' and not A ⊨ c'
        C := C ∪ {c'}
```

Example Game 1

$(a + b)\,(a + b')\,(a' + c)(a' + c')$

Example Game 2

- $(a + b + c)(b + c' + f')(b' + e)$

Some Bibliography

- **Chaff: Engineering an Efficient SAT Solver** Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, Sharad Malik (DAC'01)
- **Efficient Conflict Driven Learning in a Boolean Satisfiability Solver** Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz (IJCAD’01)
- **A New Method for Solving Hard Satisfiability Problems** Bart Selman, Hector Levesque, David Mitchell (AAI’92)

Missing

- Post Chaff SAT solvers
  - BerkMin
  - Seige
  - miniSat
  - HaifaSAT
  - JeruSAT (Alex Nadel)
- The Stålmarck's algorithm
- Hyperresolution
- Local Search

Open Question

- Is there a subset of a useful propositional logic beyond Horn clauses which:
  - Allows polynomial SAT
  - Includes many of the practical instances
  - Some recent ideas in
    - Zack Newsham, Vijay Ganesh, Sebastian Fischmeister, Gilles Audemard, Laurent Simon:
      Impact of Community Structure on SAT Solver Performance. SAT 2014: 252-268

Summary

- Rich history of emphasis on practical efficiency
- Need to account for computation cost in search space pruning
- Need to match algorithms with underlying processing system architectures
- Specific problem classes can benefit from specialized algorithms
  - Identification of problem classes?
  - Dynamically adapting heuristics?
- We barely understand the tip of the iceberg here
  - much room to learn and improve