SAT Solvers:
A Condensed History

Sharad Malik
Princeton University

COS 598d
3/1/2010

Where are we today?

- Intractability of the problem no longer daunting
  - Can regularly handle practical instances with millions of variables and constraints
- SAT has matured from theoretical interest to practical impact
  - Electronic Design Automation (EDA)
    - Widely used in many aspects of chip design
      - Automated logic design, verification
      - Used by every EDA vendor
        - Cadence, Synopsys, Mentor Graphics…
      - Used in in-house tools at leading semiconductor vendors
        - Intel, IBM, …
  - Increasing use in software verification
    - Reported commercial use at Microsoft, NEC,…

Where are we today (contd.)

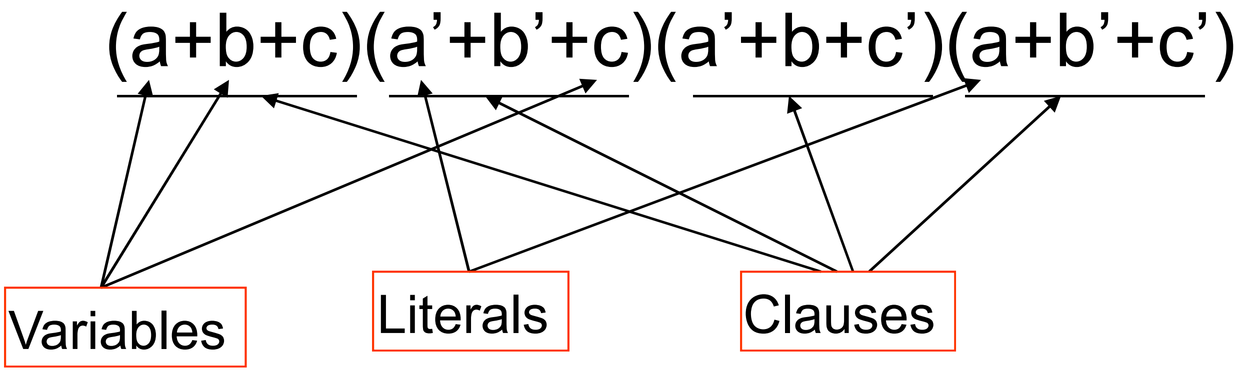
- Significant SAT community
  - SatLive and SAT competitions
  - SAT Conference
- Emboldened researchers to take on even harder problems
  - Satisfiability Modulo Theories (SMT)
  - Max-SAT
  - Quantified Boolean Formulas (QBF)

SAT Solvers:
A Condensed History

- Deductive
  - Davis-Putnam 1960 [DP]
  - Iterative existential quantification by “resolution”
- Backtrack Search
  - Davis, Logemann and Loveland 1962 [DLL]
  - Exhaustive search for satisfying assignment
- Conflict Driven Clause Learning [CDCL]
  - GRASP: Integrate a constraint learning procedure, 1996
- Locality Based Search
  - Emphasis on exhausting local sub-spaces, e.g. Chaff, Berkmin, miniSAT and others, 2001 onwards
  - Added focus on efficient implementation
    - Boolean Constraint Propagation, Decision Heuristics, …

Problem Representation

- Conjunctive Normal Form
  - Representation of choice for modern SAT solvers



(a+b+c)(a’+b’+c)(a’+b+c’)(a+b’+c’)

Variables

Literals

Clauses

Circuit to CNF Conversion

- Tseitin Transformation

$d \equiv (a + b)$

$(a + b + d')$

$(a' + d)$

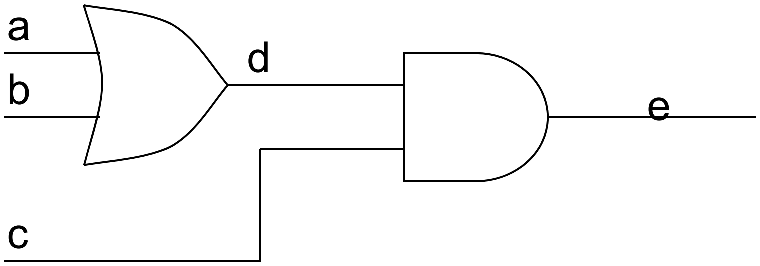
$(b' + d)$

$e \equiv (c \cdot d)$

$(c' + d' + e)$

$(d + e')$

$(c + e')$



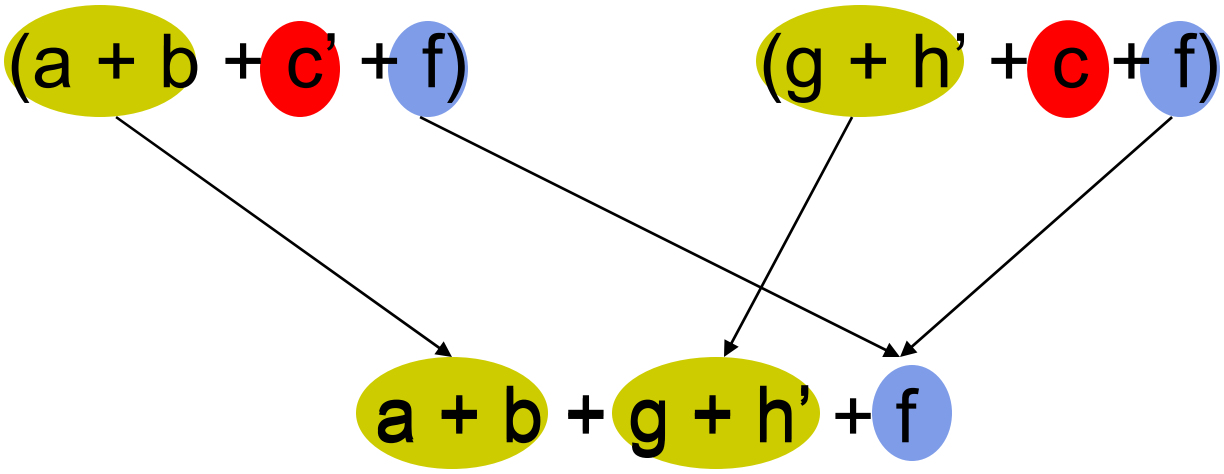
Consistency conditions for circuit variables

- Can ‘e’ ever become true?

Is $(e)(a + b + d')(a'+d)(b'+d)(c'+d+e)(d+e')(c+e')$ satisfiable?

Resolution

- Resolution of a pair of distance-one clauses



(a + b + c’ + f)   (g + h’ + c + f)

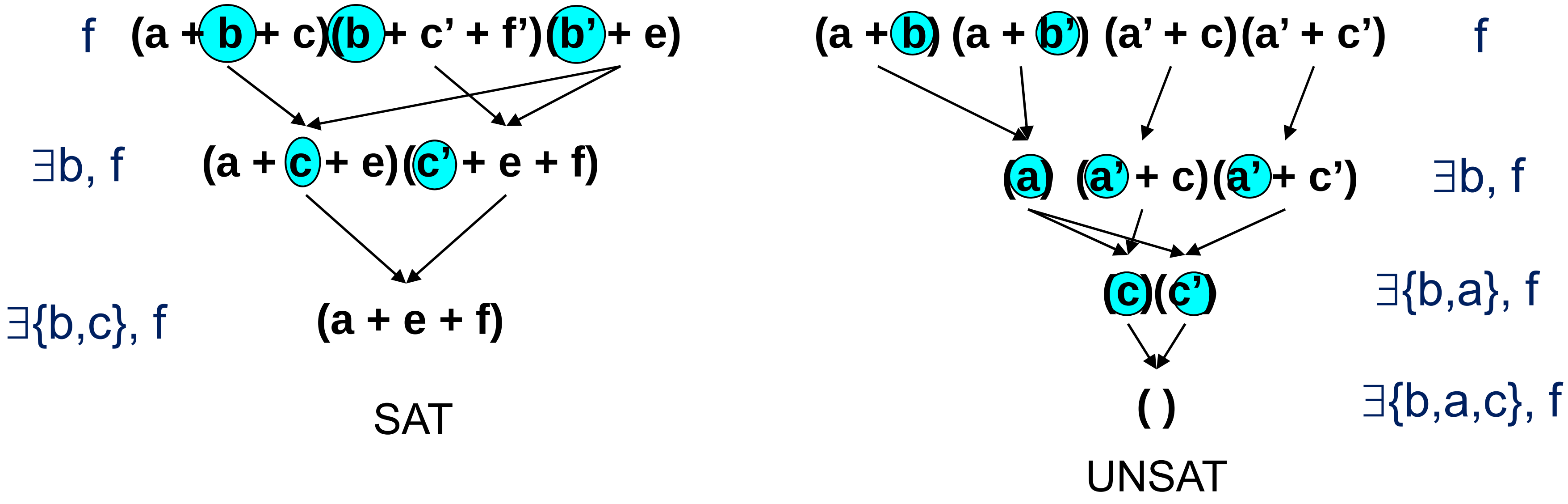
a + b + g + h’ + f

Resolvent implied by the original clauses

Davis Putnam Algorithm

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

- Iterative existential quantification of variables



f **(a + b + c)(b + c’ + f’)(b’ + e)**

∃b, f **(a + c + e)(c’ + e + f)**

∃{b,c}, f **(a + e + f)**

SAT

(a + b) (a + b’) (a’ + c)(a’ + c’) f

(a) (a’ + c)(a’ + c’) ∃b, f

(c)(c’) ∃{b,a}, f

( ) ∃{b,a,c}, f

UNSAT

Potential memory explosion problem!

SAT Solvers:
A Condensed History

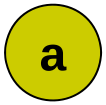
- Deductive
  - Davis-Putnam 1960 [DP]
  - Iterative existential quantification by “resolution”
- Backtrack Search
  - Davis, Logemann and Loveland 1962 [DLL]
  - Exhaustive search for satisfying assignment
- Conflict Driven Clause Learning [CDCL]
  - GRASP: Integrate a constraint learning procedure, 1996
- Locality Based Search
  - Emphasis on exhausting local sub-spaces, e.g. Chaff, Berkmin, miniSAT and others, 2001 onwards
  - Added focus on efficient implementation
    - Boolean Constraint Propagation, Decision Heuristics, …

Basic DLL Search

(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 Search

(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 Search

→ (a’ + b + c)

(a + c + d)

(a + c + d’)

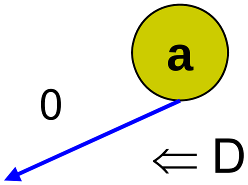
(a + c’ + d)

(a + c’ + d’)

(b’ + c’ + d)

→ (a’ + b + c’)

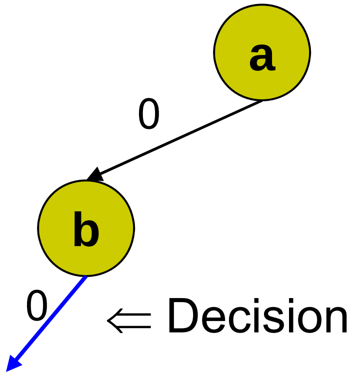
→ (a’ + b’ + c)



⇐ Decision

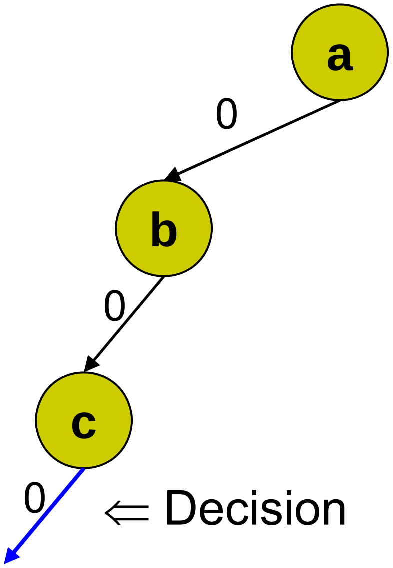
Basic DLL Search

(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 Search

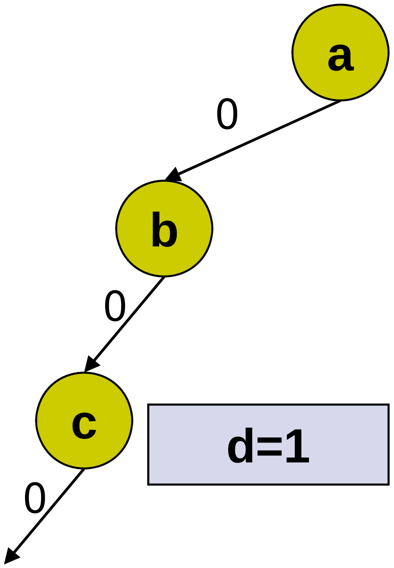
(a’ + b + c)
(a + c + d)
(a + c + d’)
→ (a + c’ + d)
→ (a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)



⇐ Decision

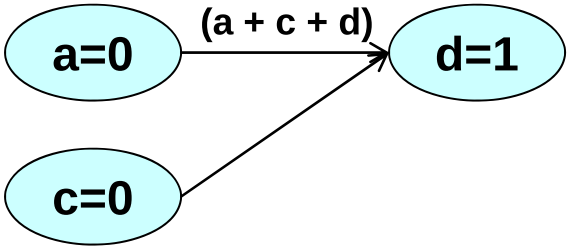
Basic DLL Search

(a’ + b + c)
→ **(a + c + d)** ⇐ Unit
→ **(a + c + d’)**
(a + c’ + d)
(a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)



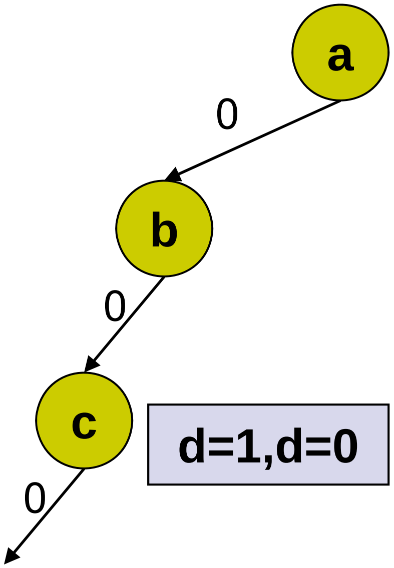
Unit Clause Rule

Implication Graph

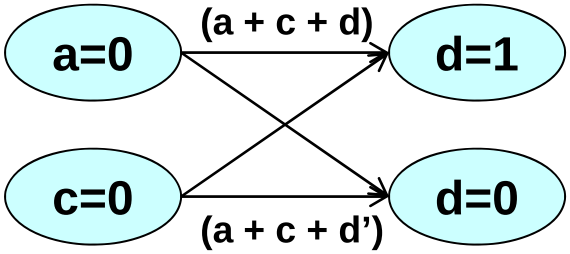


Basic DLL Search

(a' + b + c)
→ (a + c + d)
→ (a + c + d') ⇐ Unit
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)



Implication Graph



Basic DLL Search

(a’ + b + c)

→ (a + c + d)

→ (a + c + d’)

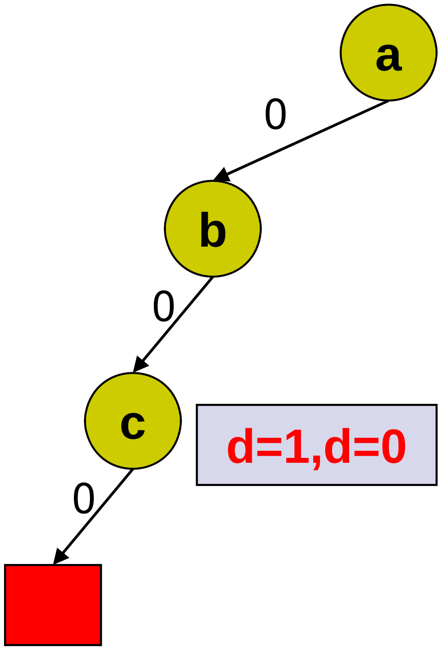
(a + c’ + d)

(a + c’ + d’)

(b’ + c’ + d)

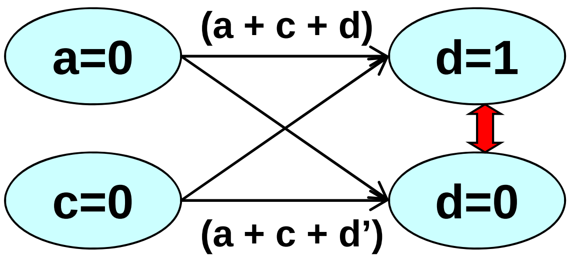
(a’ + b + c’)

(a’ + b’ + c)



d=1,d=0

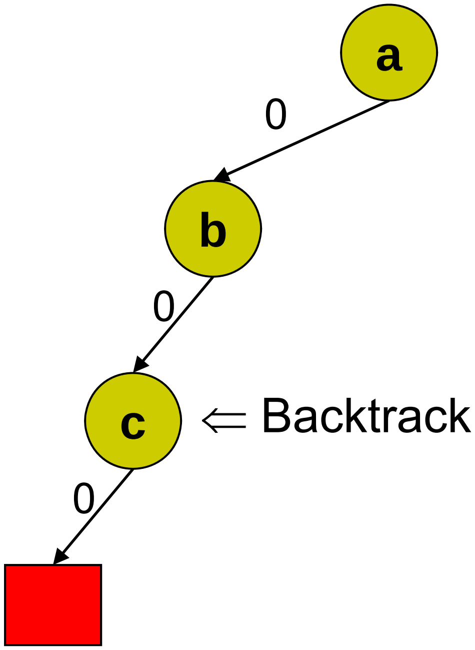
Implication Graph



Conflict!

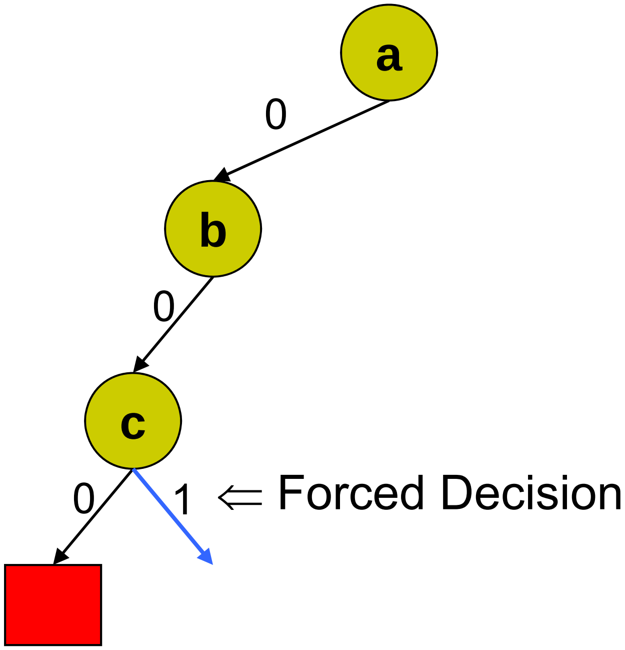
Basic DLL Search

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



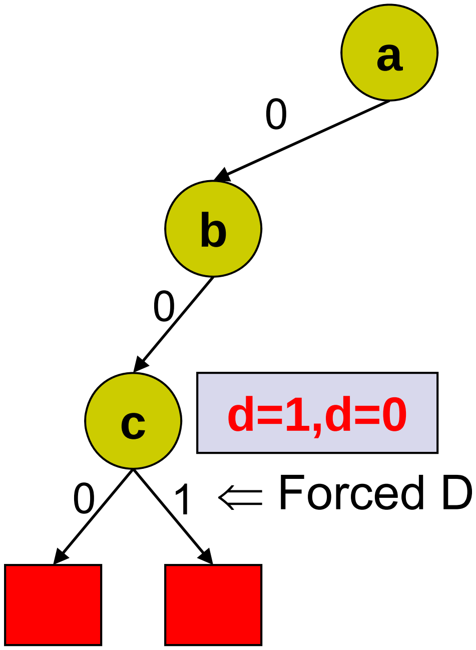
Basic DLL Search

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



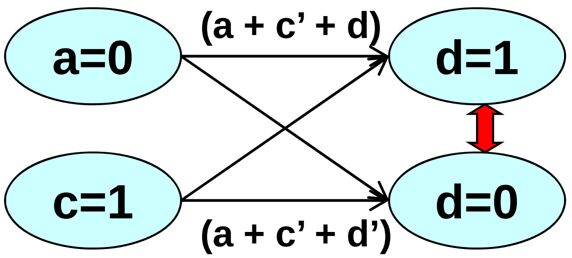
Basic DLL Search

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



⇐ Forced Decision

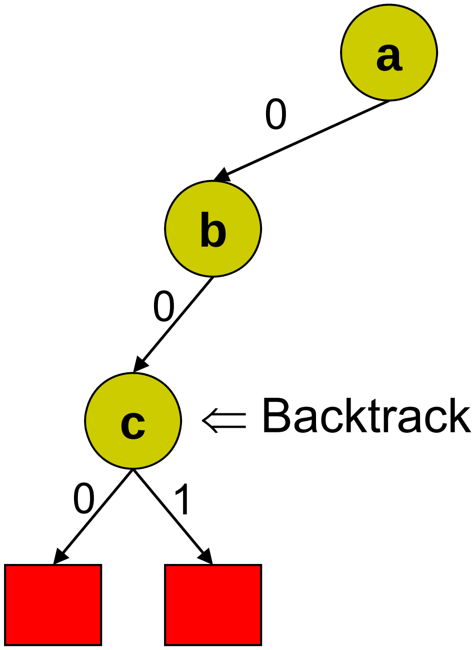
Implication Graph



Conflict!

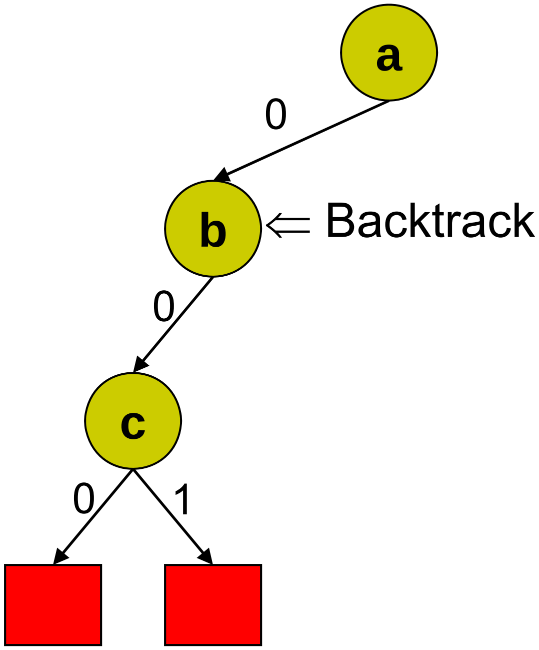
Basic DLL Search

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



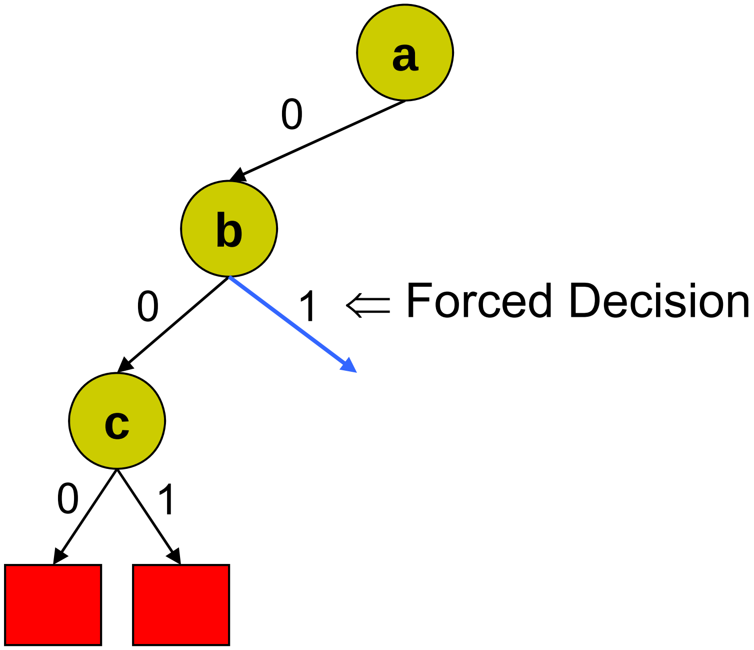
Basic DLL Search

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



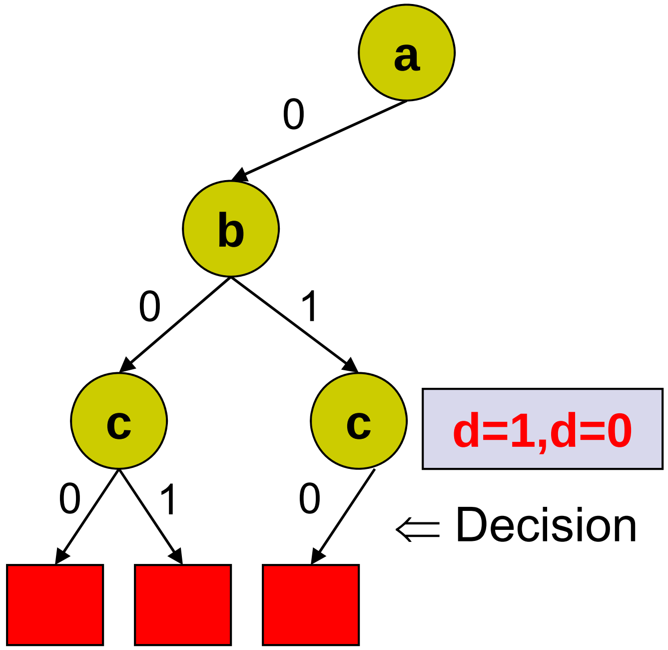
Basic DLL Search

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



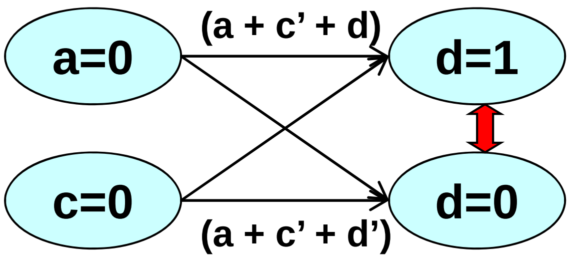
Basic DLL Search

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



⇐ Decision

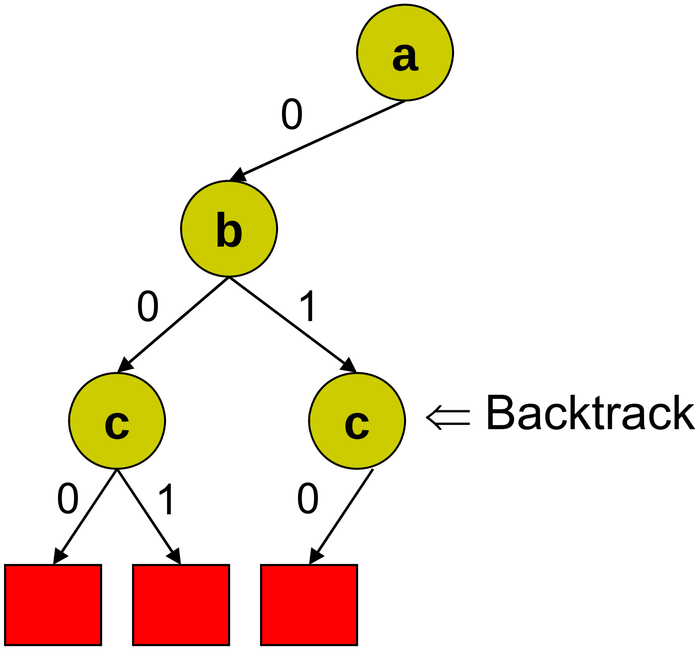
Implication Graph



Conflict!

Basic DLL Search

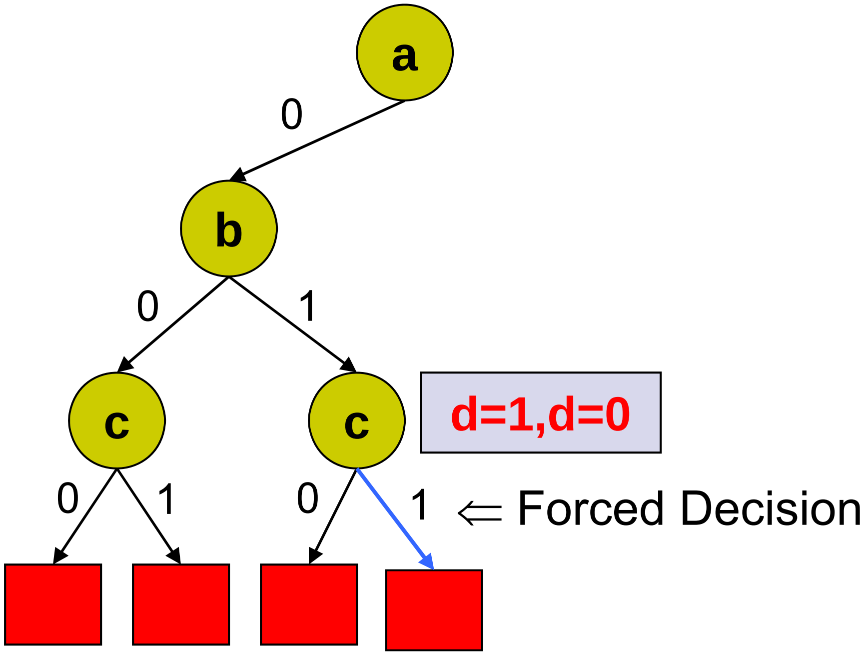
(a' + b + c)
→ (a + c + d)
→ (a + c + d')
→ (a + c' + d)
→ (a + c' + d')
→ (b' + c' + d)
(a' + b + c')
(a' + b' + c)



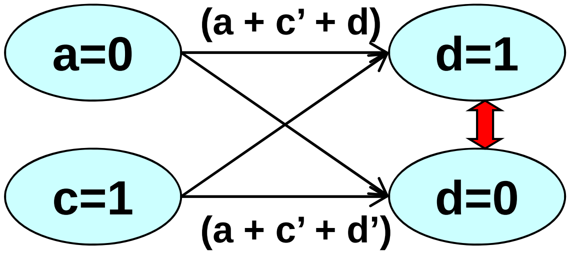
⇐ Backtrack

Basic DLL Search

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



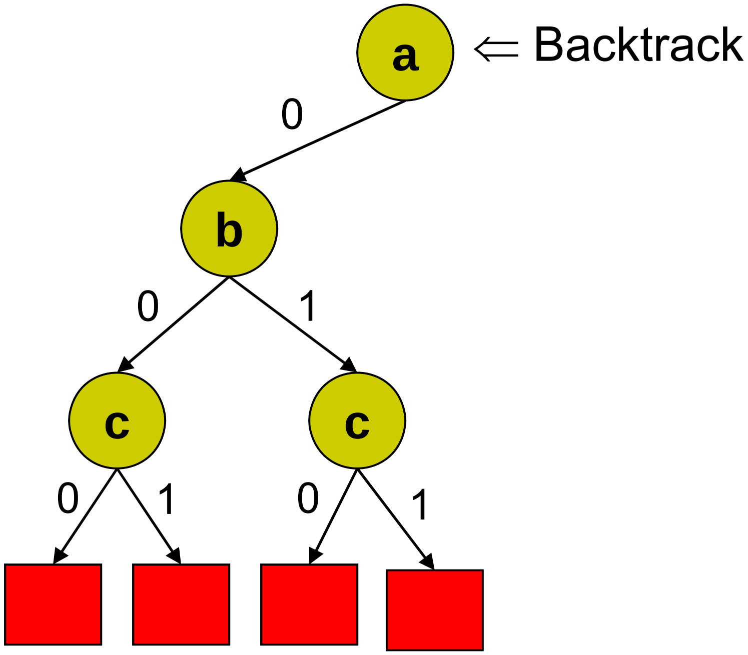
Implication Graph



Conflict!

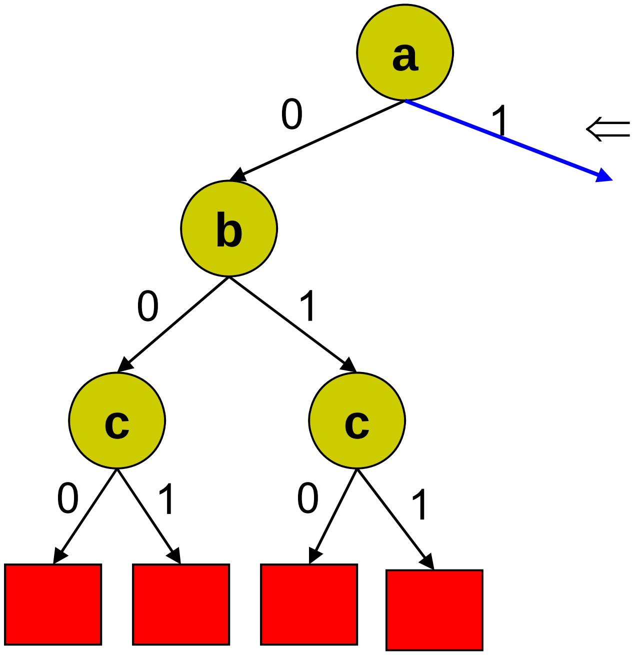
Basic DLL Search

→ (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 Search

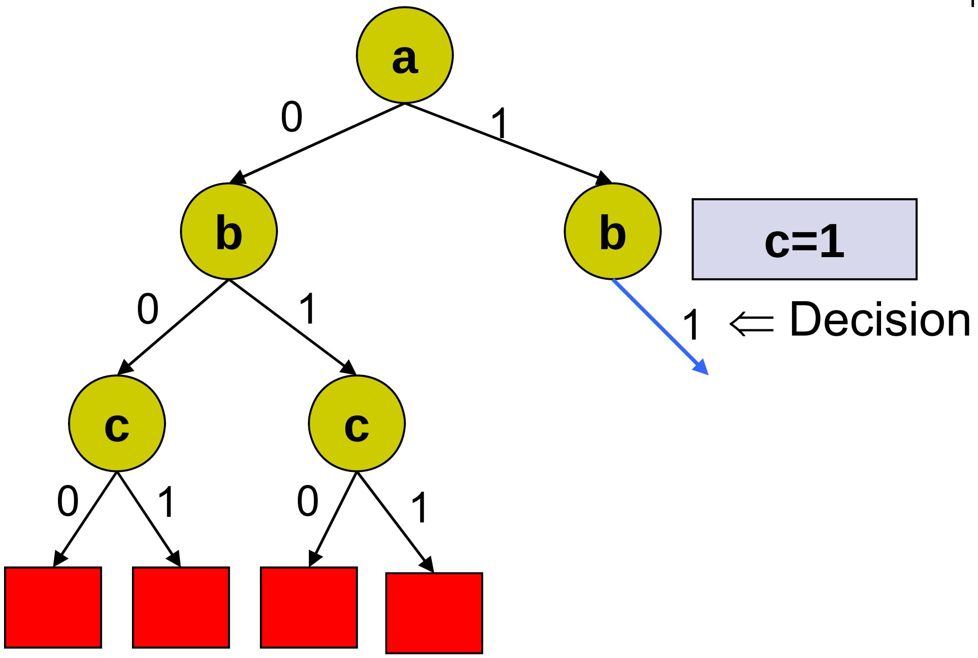
(a’ + b + c)
→ (a + c + d)
→ (a + c + d’)
→ (a + c’ + d)
→ (a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)



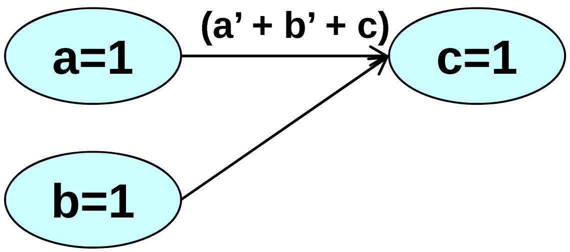
⇐ Forced Decision

Basic DLL Search

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

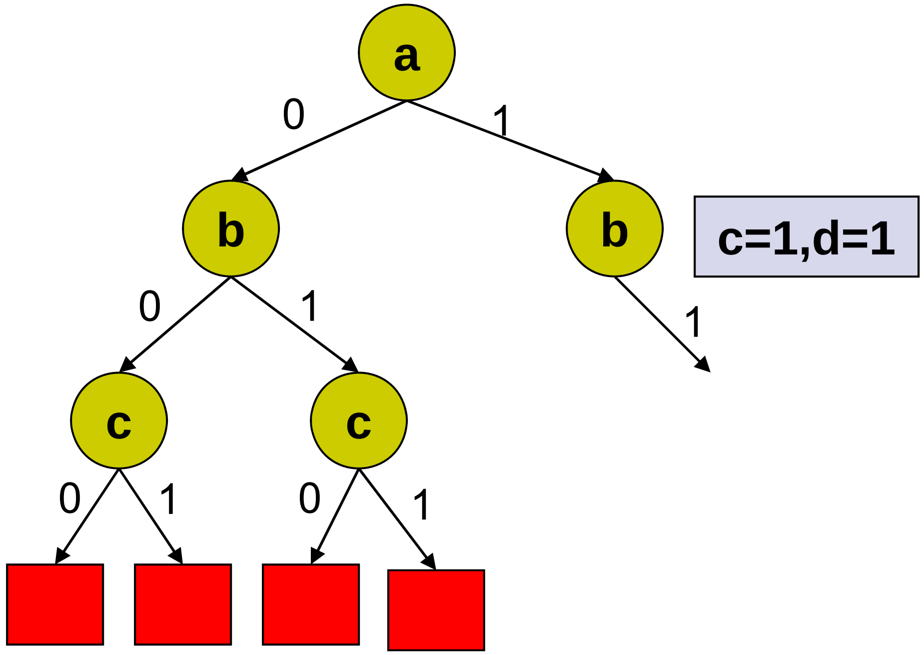


Implication Graph

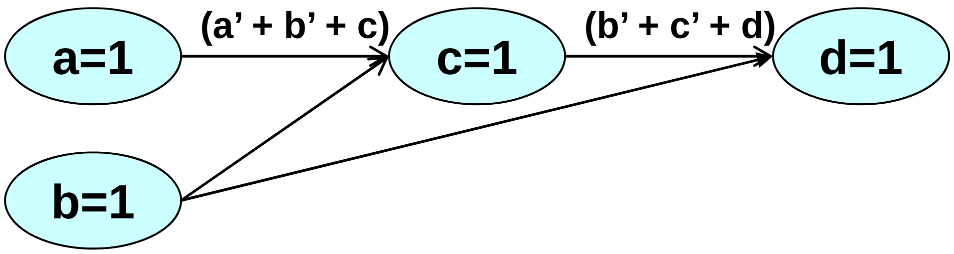


Basic DLL Search

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

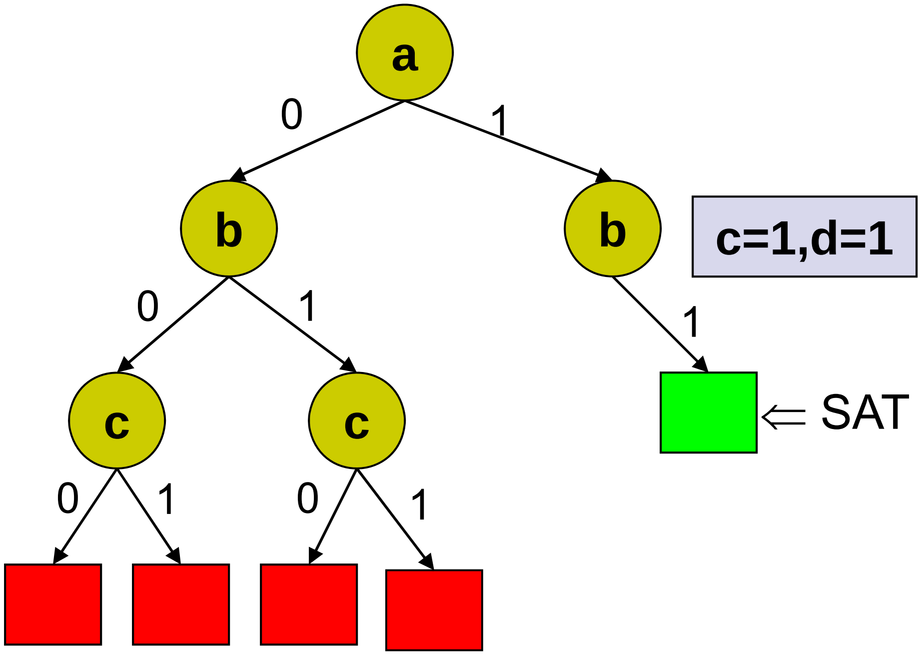


Implication Graph

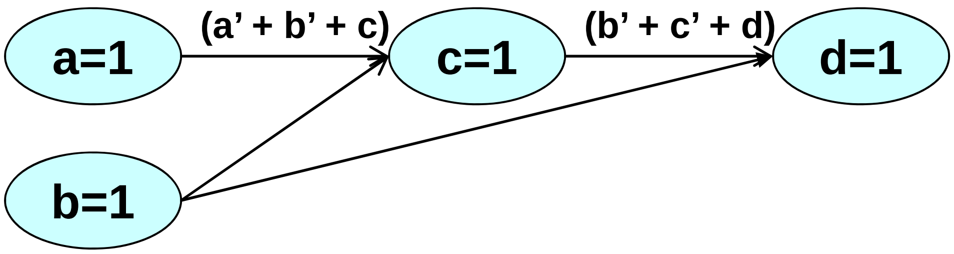


Basic DLL Search

(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



SAT Solvers:
A Condensed History

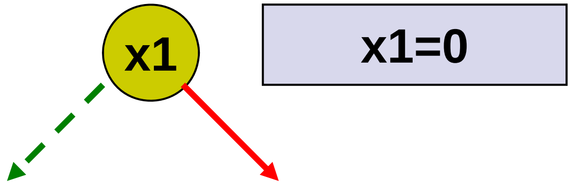
- Deductive
  - Davis-Putnam 1960 [DP]
  - Iterative existential quantification by “resolution”
- Backtracking Search
  - Davis, Logemann and Loveland 1962 [DLL]
  - Exhaustive search for satisfying assignment
- Conflict Driven Clause Learning [CDCL]
  - GRASP: Integrate a constraint learning procedure, 1996
- Locality Based Search
  - Emphasis on exhausting local sub-spaces, e.g. Chaff, Berkmin, miniSAT and others, 2001 onwards
  - Added focus on efficient implementation
    - Boolean Constraint Propagation, Decision Heuristics, …

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

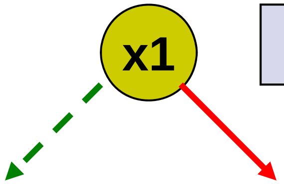
x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’



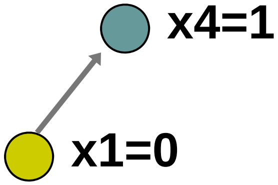
● x1=0

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’

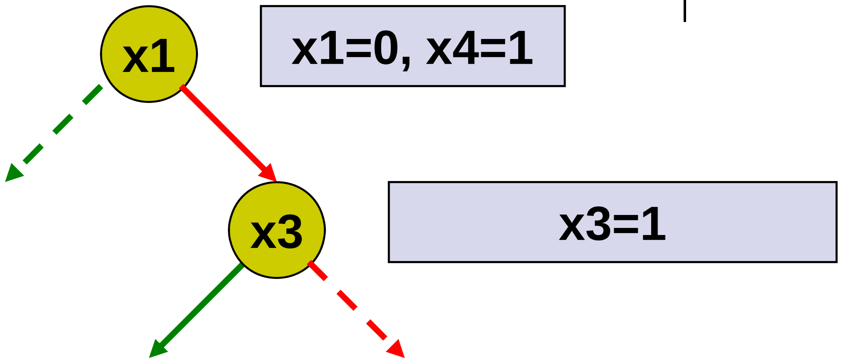


x1=0, x4=1



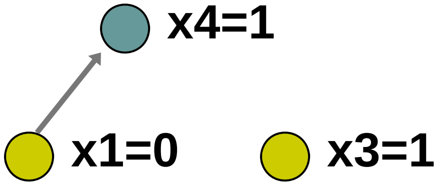
Conflict Driven Learning and Non-chronological Backtracking

x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’

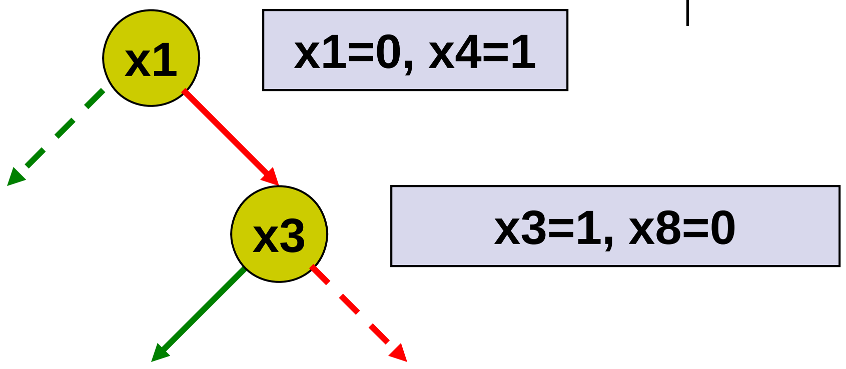


x1=0, x4=1

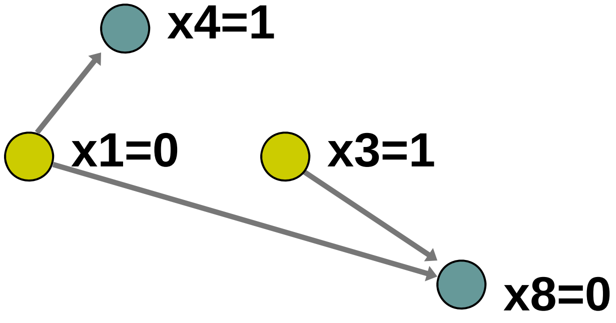
x3=1



Conflict Driven Learning and Non-chronological Backtracking

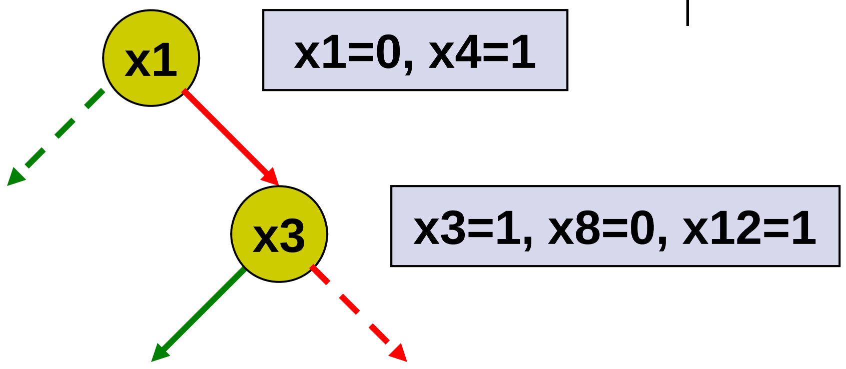


x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’



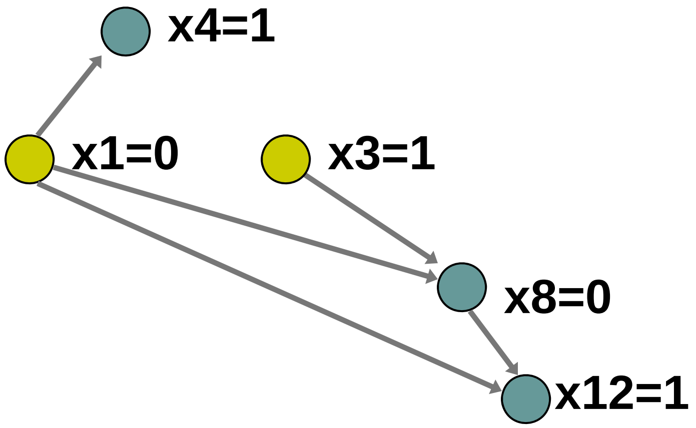
Conflict Driven Learning and Non-chronological Backtracking

x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’



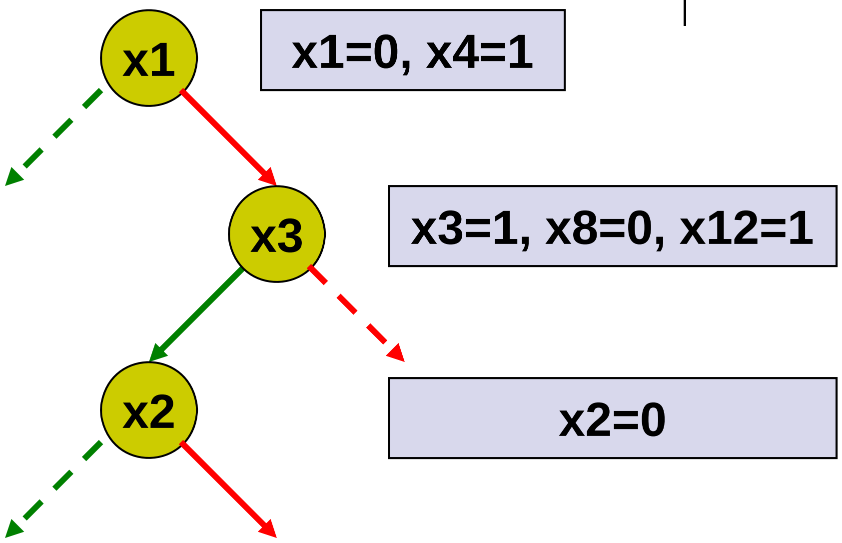
x1=0, x4=1

x3=1, x8=0, x12=1



Conflict Driven Learning and Non-chronological Backtracking

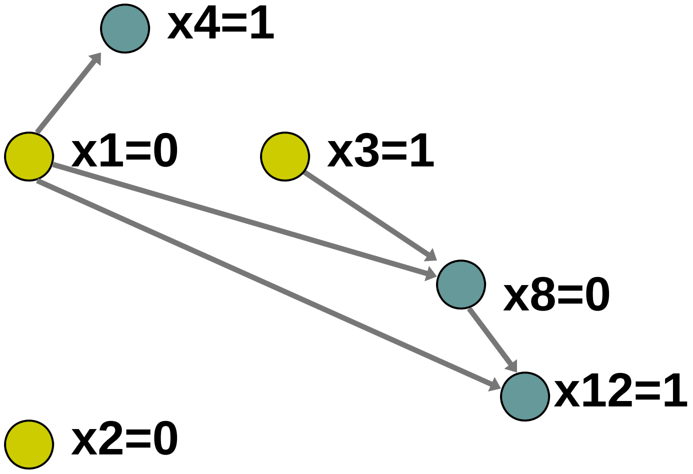
x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’



x1=0, x4=1

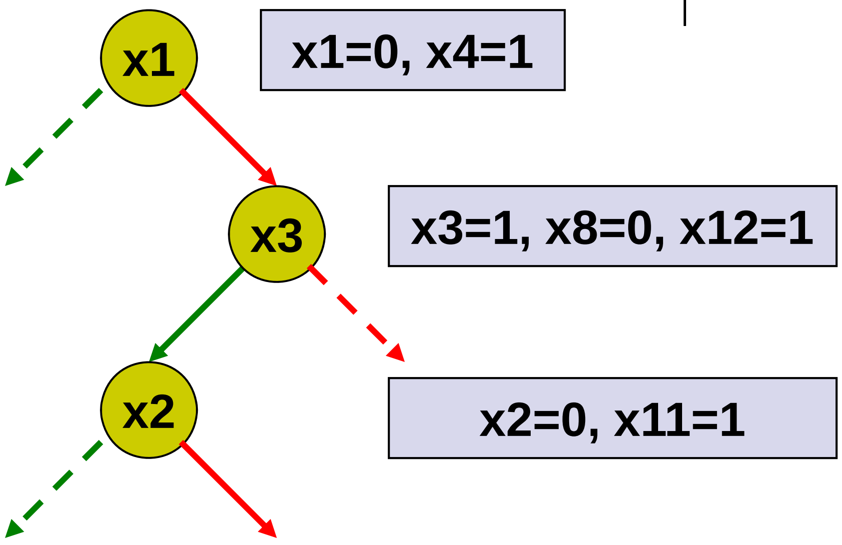
x3=1, x8=0, x12=1

x2=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’



x1

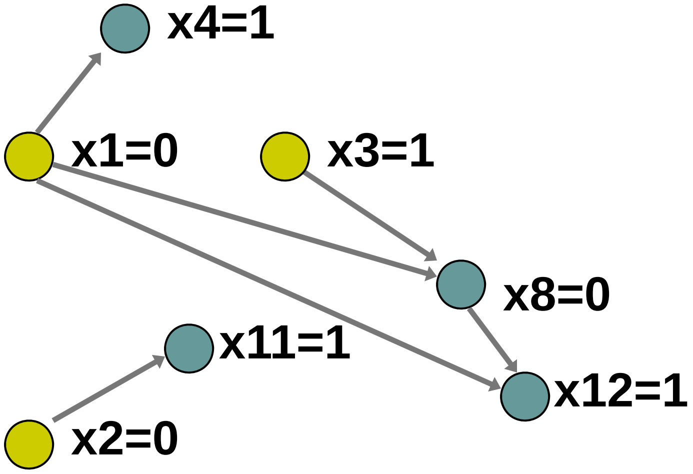
x1=0, x4=1

x3

x3=1, x8=0, x12=1

x2

x2=0, x11=1



x4=1

x1=0

x3=1

x8=0

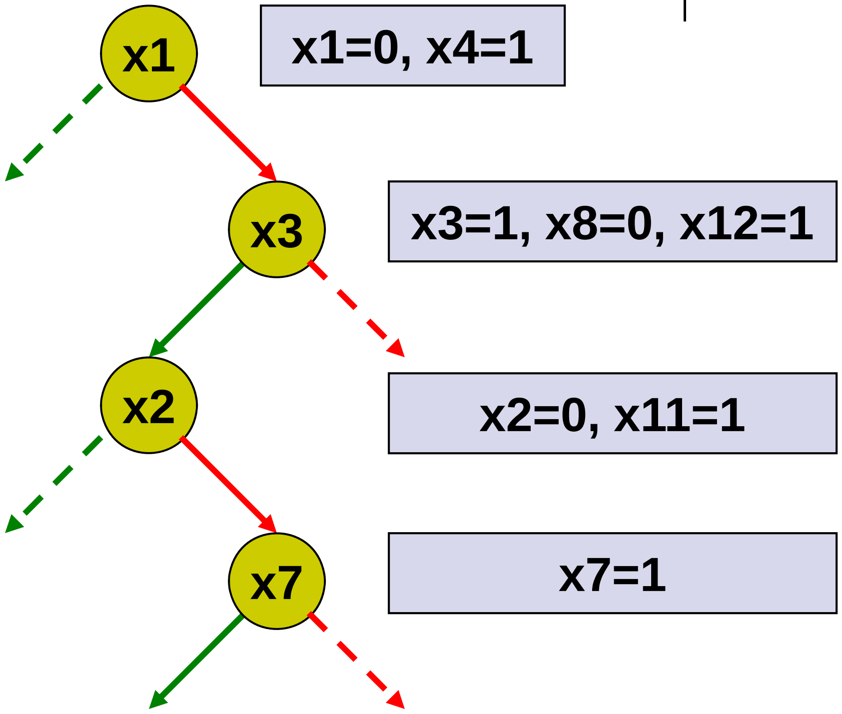
x11=1

x12=1

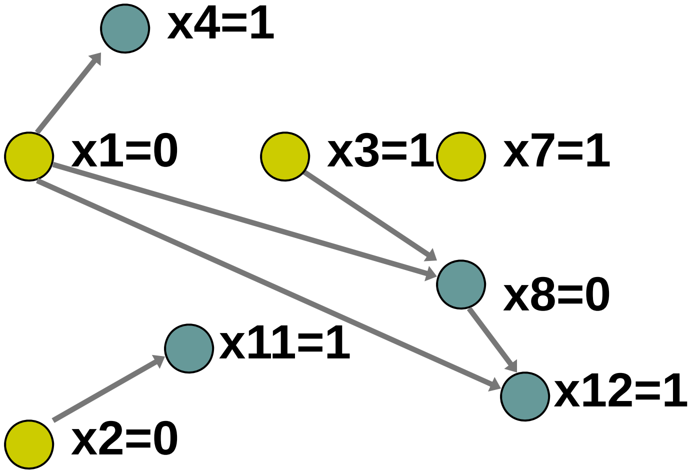
x2=0

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’

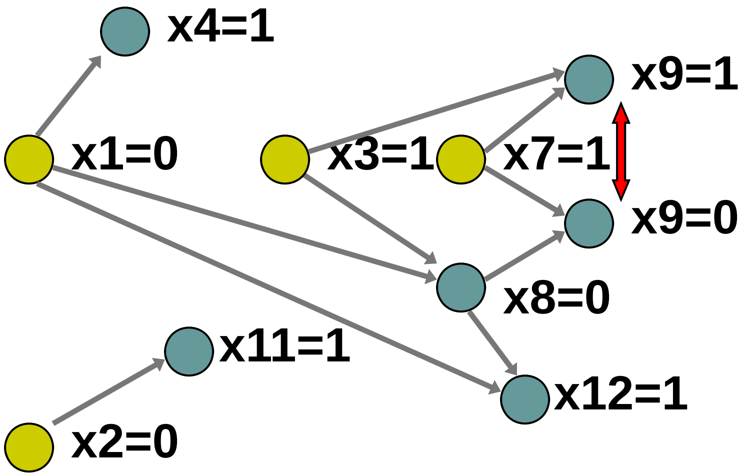


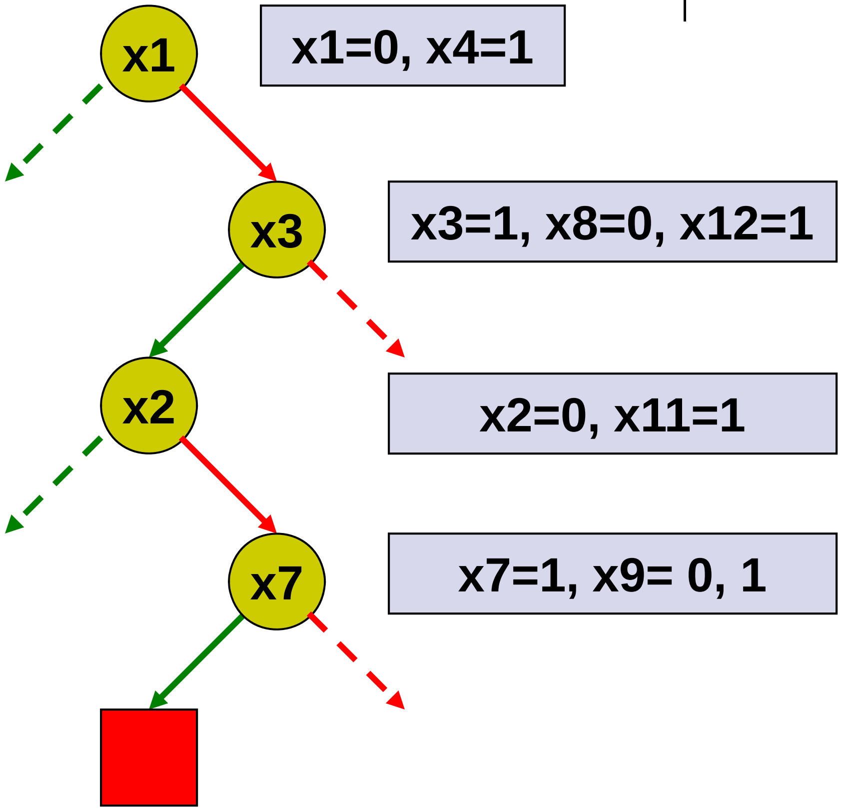
x1=0, x4=1
x3=1, x8=0, x12=1
x2=0, x11=1
x7=1



Conflict Driven Learning and Non-chronological Backtracking

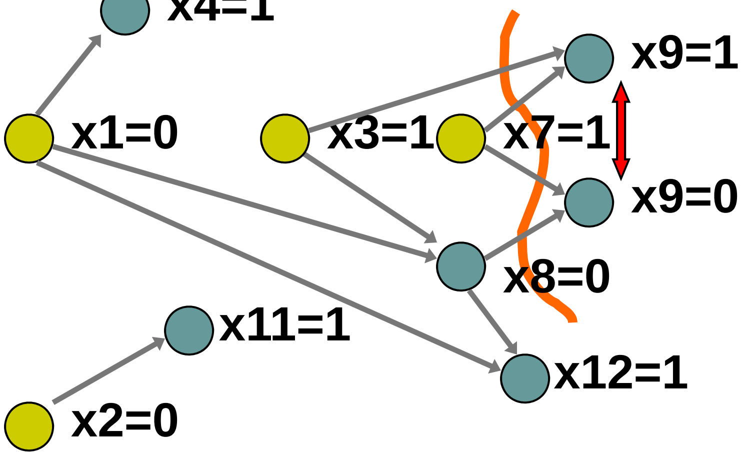
x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’

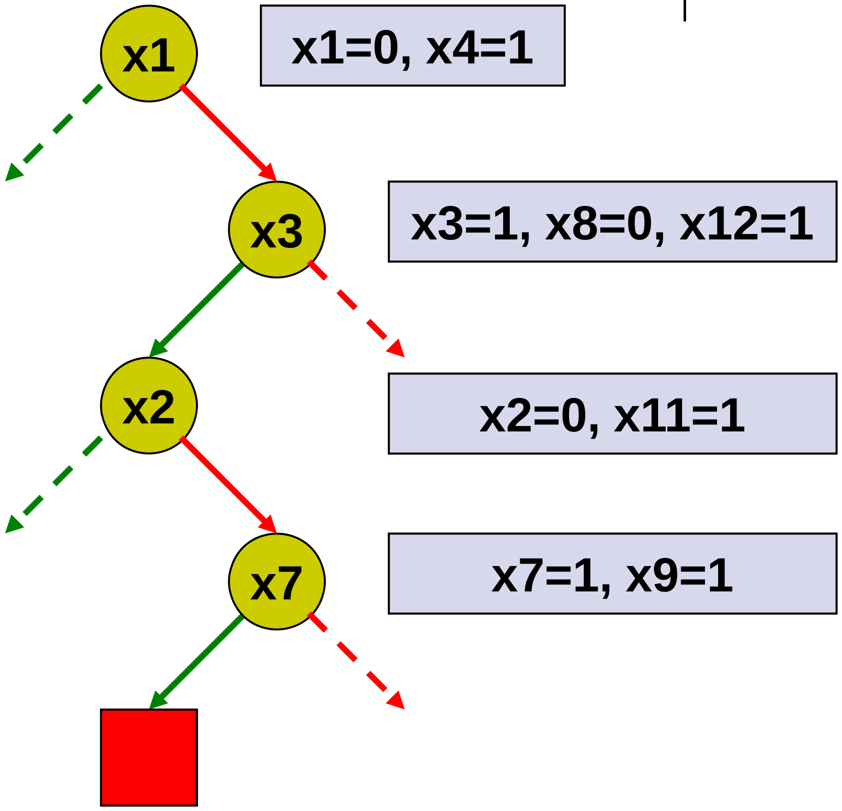




Conflict Driven Learning and Non-chronological Backtracking

x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’





x1=0, x4=1

x3=1, x8=0, x12=1

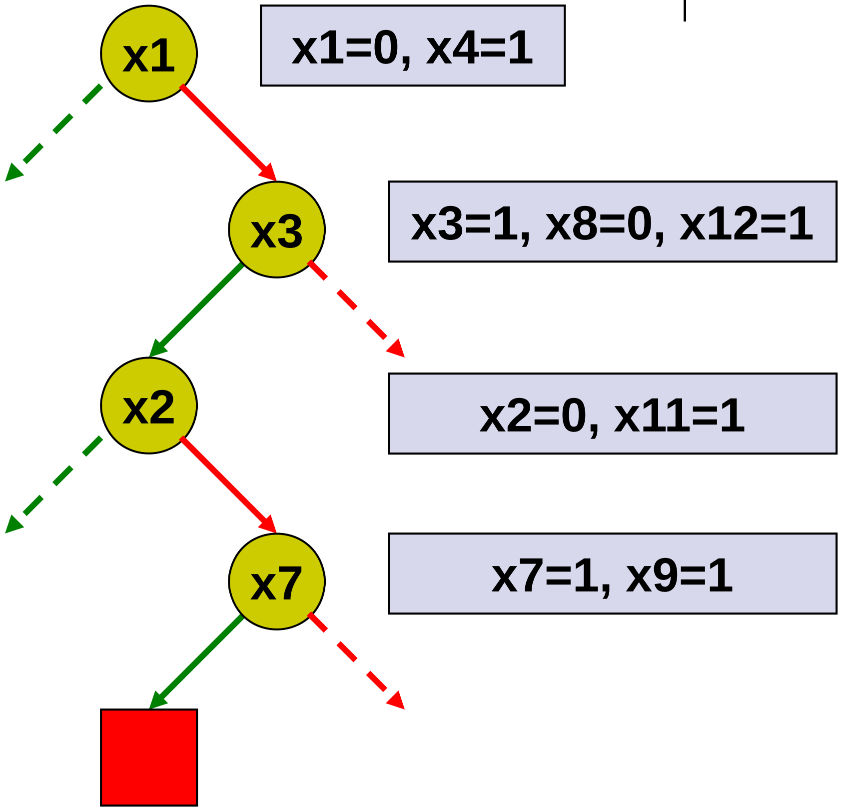
x2=0, x11=1

x7=1, x9=1

$x3=1 \wedge x7=1 \wedge x8=0 \rightarrow$ 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’

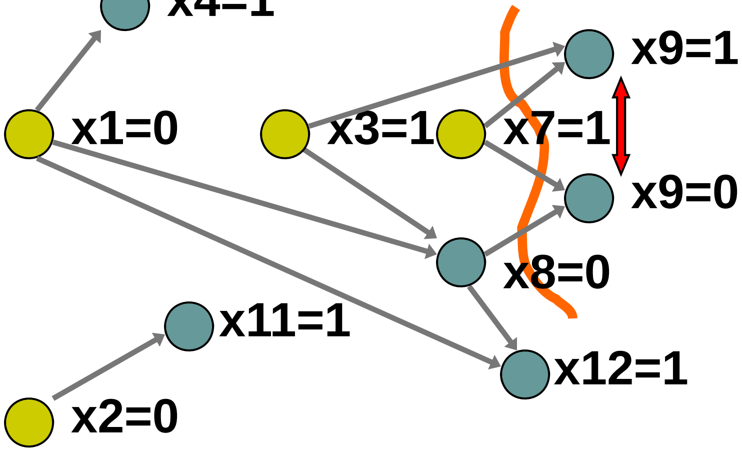


x1=0, x4=1

x3=1, x8=0, x12=1

x2=0, x11=1

x7=1, x9=1



$x3=1 \wedge x7=1 \wedge x8=0 \rightarrow$ conflict

Add conflict clause: x3’+x7’+x8

Conflict Driven Learning and Non-chronological Backtracking

x1 + x4

x1 + x3’ + x8’

x1 + x8 + x12

x2 + x11

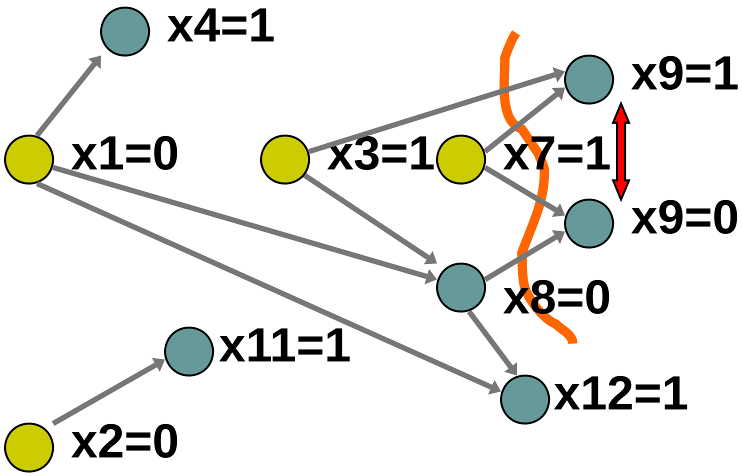
x7’ + x3’ + x9

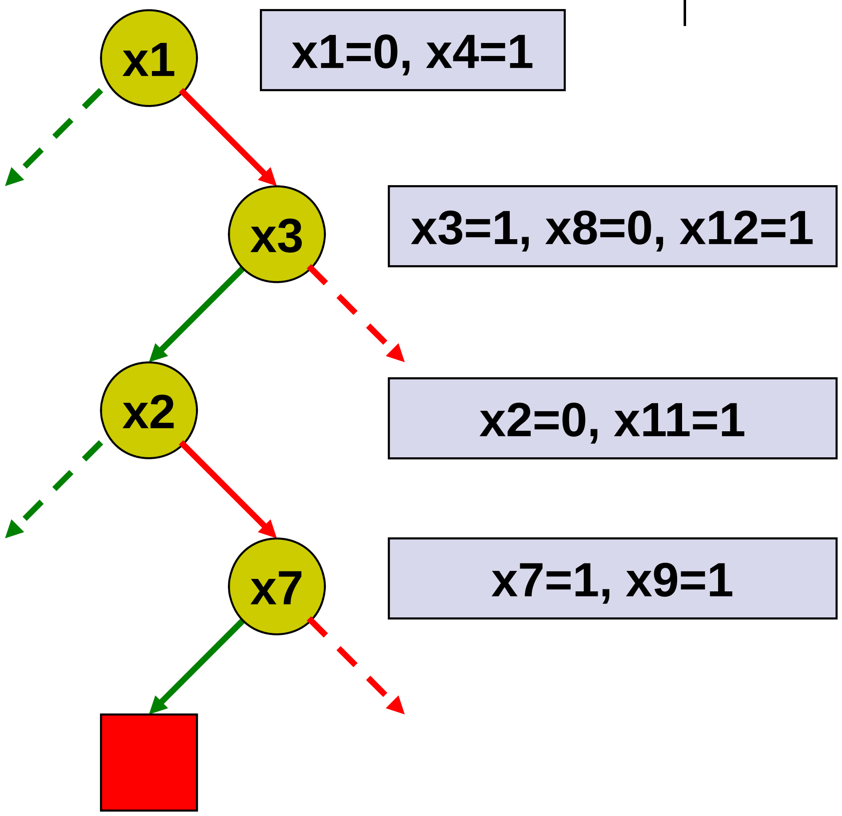
x7’ + x8 + x9’

x7 + x8 + x10’

x7 + x10 + x12’

x3’+x7’+x8





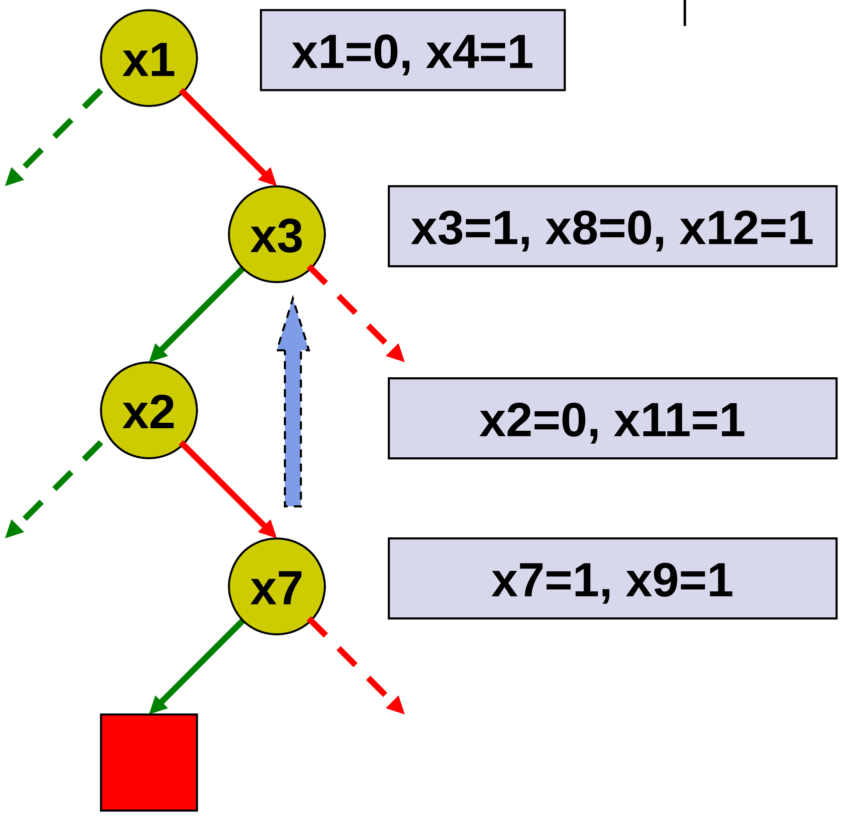
x3=1∧x7=1∧x8=0 → conflict

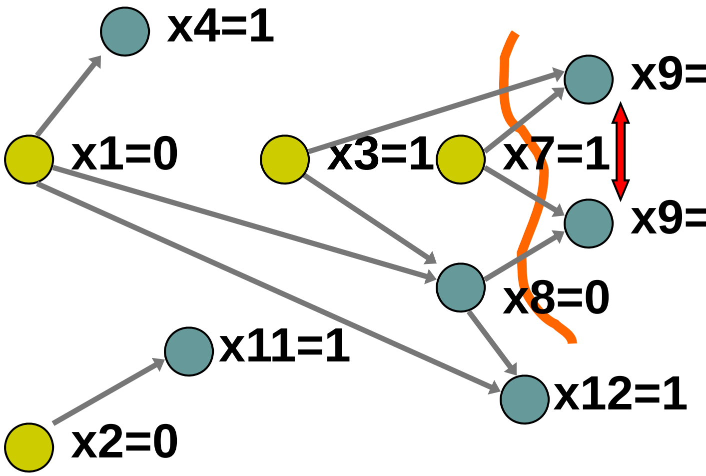
Add conflict clause: x3’+x7’+x8

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

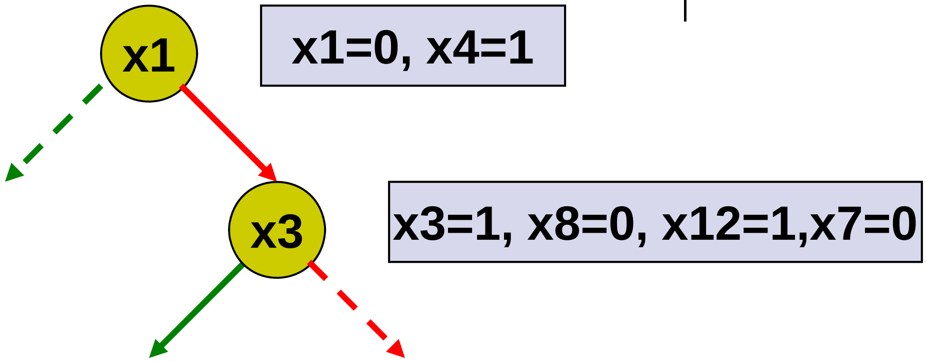


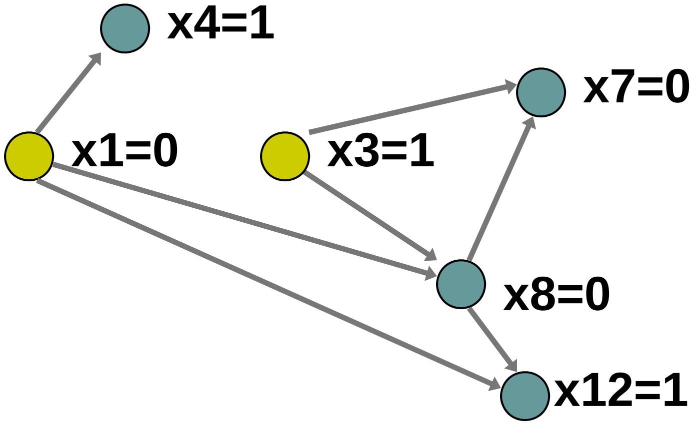


Backtrack to the decision level of x3=1

Conflict Driven Learning and Non-chronological Backtracking

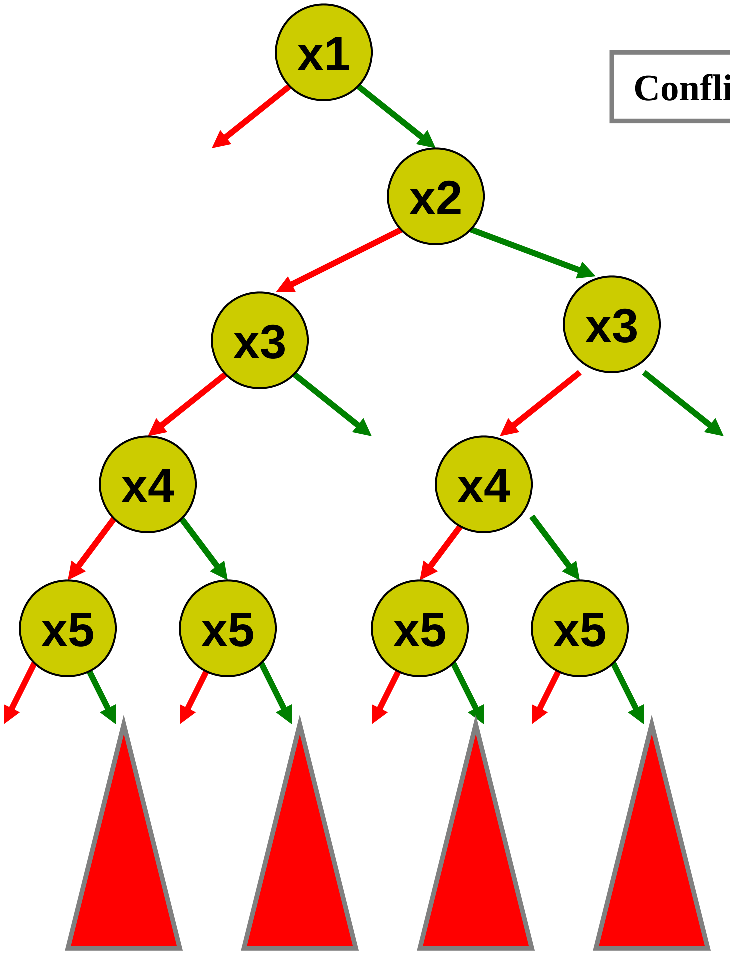
x1 + x4
x1 + x3’ + x8’
x1 + x8 + x12
x2 + x11
x7’ + x3’ + x9
x7’ + x8 + x9’
x7 + x8 + x10’
x7 + x10 + x12’
x3’ + x7’ + x8 ←new clause





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

What’s the big deal?



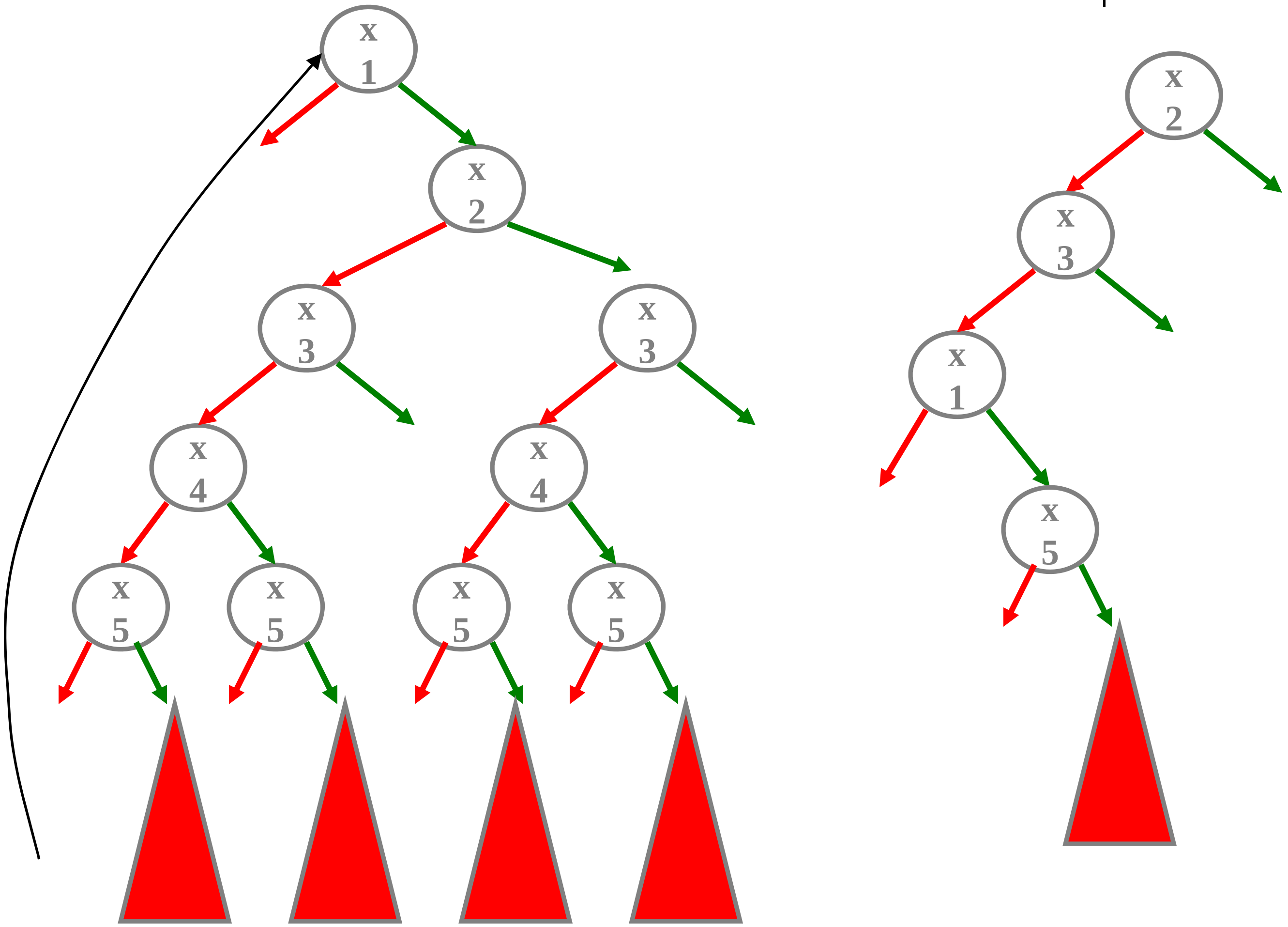
Conflict clause: x1’+x3+x5’

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

Useful in generating future conflict clauses.

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



Conflict clause: x1’+x3+x5’

SAT Solvers:
A Condensed History

- Deductive
  - Davis-Putnam 1960 [DP]
  - Iterative existential quantification by “resolution”
- Backtracking Search
  - Davis, Logemann and Loveland 1962 [DLL]
  - Exhaustive search for satisfying assignment
- Conflict Driven Clause Learning [CDCL]
  - GRASP: Integrate a constraint learning procedure, 1996
- Locality Based Search
  - Emphasis on exhausting local sub-spaces, e.g. Chaff, Berkmin, miniSAT and others, 2001 onwards
  - Added focus on efficient implementation
    - Boolean Constraint Propagation, Decision Heuristics, …

Success with Chaff (2000)

- First major instance: Tough
- Industrial Processor Verification
  - Bounded Model Checking, 14 cycle behavior
- Statistics
  - 1 million variables
  - 10 million literals initially
    - 200 million literals including added clauses
    - 30 million literals finally
  - 4 million clauses (initially)
    - 200K clauses added
  - 1.5 million decisions
  - 3 hour run time

[MMZ+01]

Constants Matter

Motivating Metrics: Decisions, Instructions, Cache Performance and Run Time

	1dlx_c_mc_ex_bp_f
Num Variables	**776**
Num Clauses	**3725**
Num Literals	**10045**

	zChaff	SATO	GRASP
# Decisions	3166	3771	1795
# Instructions	86.6M	630.4M	1415.9M
# L1/L2 accesses	24M / 1.7M	188M / 79M	416M / 153M
% L1/L2 misses	4.8% / 4.6%	36.8% / 9.7%	32.9% / 50.3%
# Seconds	0.22	4.41	11.78

Chaff Contribution 1: 2 Literal Watching

- N-literal clause can be unit or conflicting only after N-1 of the literals have been assigned to F
  - (v1 + v2 + v3): implied cases: (0 + 0 + v3) or (0 + v2 + 0) or (v1 + 0 + 0)
- 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=?)
- If a clause can become newly implied via any sequence of assignments, then this sequence will include an assignment of one of the watched literals to F

Decision Heuristics – Conventional Wisdom

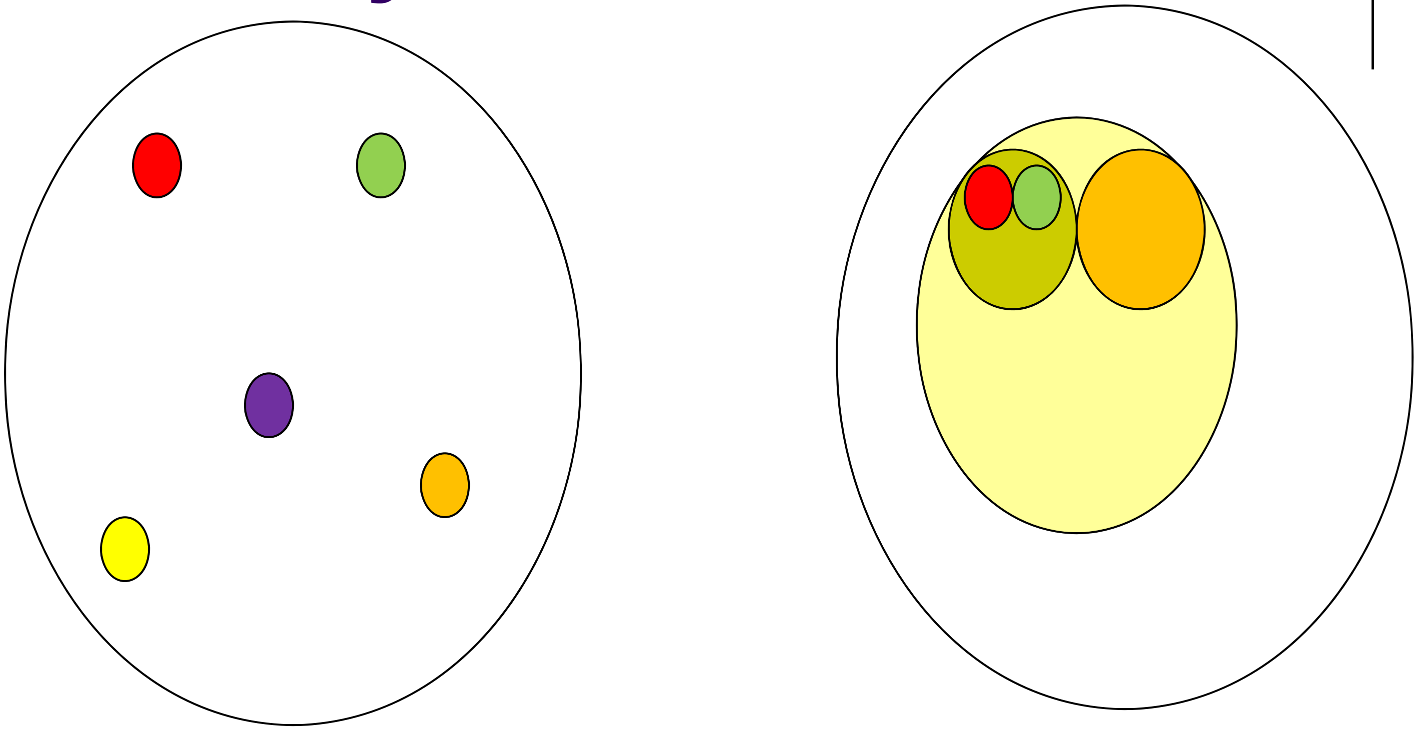
- “Assign most tightly constrained variable” : e.g. DLIS (Dynamic Largest Individual Sum)
  - Simple and intuitive: At each decision simply choose the assignment that satisfies the most unsatisfied clauses.
  - Expensive book-keeping operations required
    - Must touch *every* clause that contains a literal that has been set to true. Often restricted to initial (not learned) clauses.
    - Need to reverse the process for un-assignment.
- Look ahead algorithms even more compute intensive

  C. Li, Anbulagan, “Look-ahead versus look-back for satisfiability problems” *Proc. of CP*, 1997.
- Take a more “global” view of the problem

Chaff Contribution 2: Modern Decision Heuristics – Variable Activity Based

- VSIDS: **V**ariable **S**tate **I**ndependent **D**ecaying **S**um
  - Rank variables by literal count in the initial clause database
  - Only increment counts as new (learnt) clauses are added
  - Periodically, divide all counts by a constant
- Quasi-static:
  - Static because it doesn't depend on variable state
  - Not static because it gradually changes as new clauses are added
    - Decay causes bias toward *recent* conflicts.
    - Has a beneficial interaction with 2-literal watching

Activity Based Heuristics and Locality Based Search



- By focusing on a sub-space, the covered spaces tend to coalesce
    - More opportunities for resolution since most of the variables are common.
    - Variable activity based heuristics lead to locality based search

SAT Solvers: Related Capabilities

- Independent Checkers and UNSAT Cores
- minCostSAT
- partialMaxSAT

Extracting an Unsatisfiable Core: Motivation

- Debugging and redesign: SAT instances are often generated from real world applications with certain expected results:
  - If the expected result is unsatisfiable, but the instance is satisfiable, then the solution is a “stimulus” or “input vector” or “counter-example” for debugging
    - Combinational Equivalence Checking
    - Bounded Model Checking
  - What if the expected results is satisfiable?
    - SAT Planning
    - FPGA Routing
- Relax constraints:
  - If several constraints make a safety property hold, are there any redundant constraints in the system that can be removed without violating the safety property?

Extract Unsatisfiable Core

Given an unsatisfiable Boolean Formula in CNF

$$F=C_1C_2......C_n$$

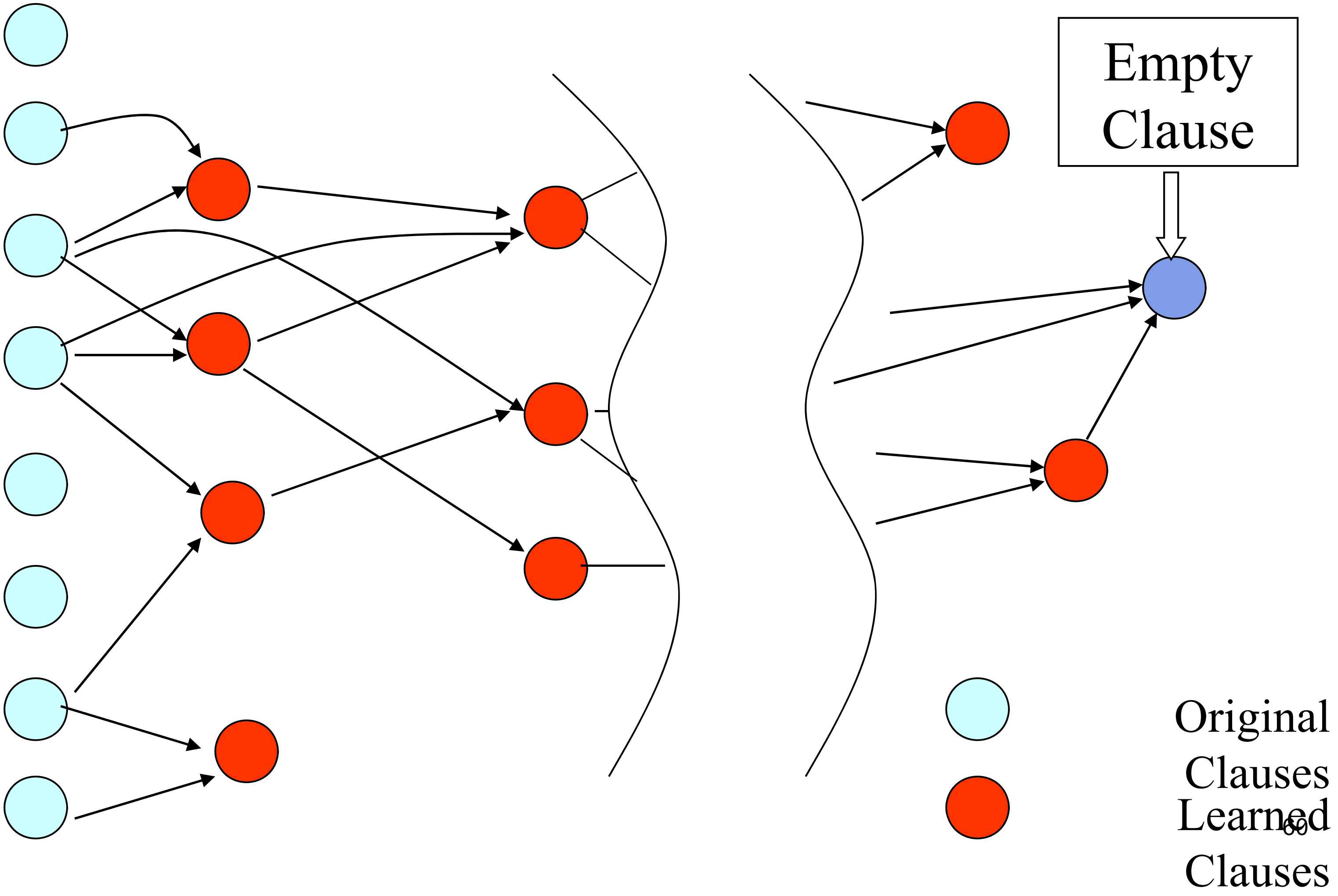
Find a formula

$$G=C_1'C_2'......C_m'$$

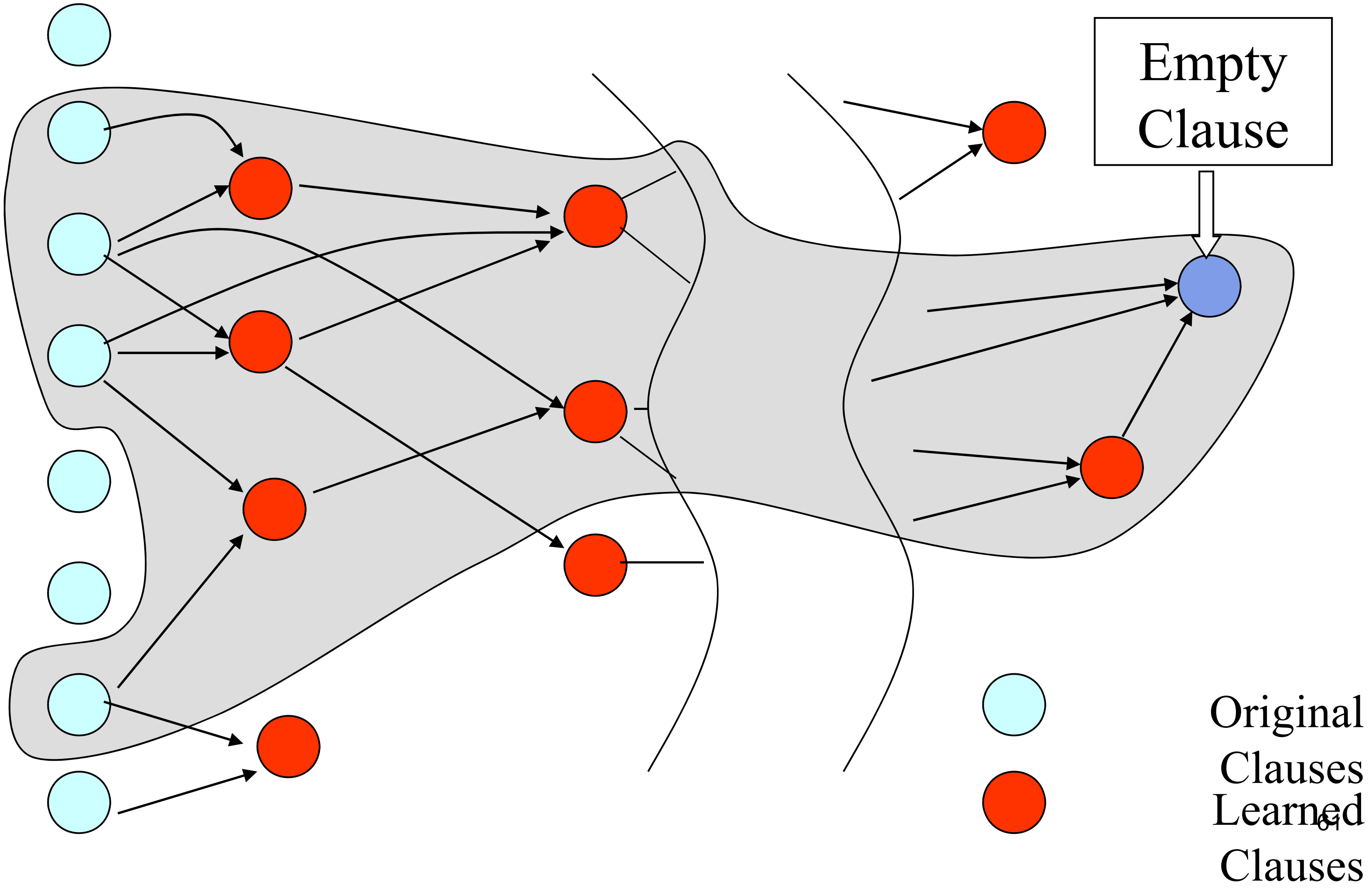
Such that G is unsatisfiable, $C_i' \in \{C_i \mid i=1...n\}$ with $m \leq n$

Proof of Unsatisfiability and Unsat Core

Resolution Graph for UNSAT Instance



Proof of Unsatisfiability and Unsat Core

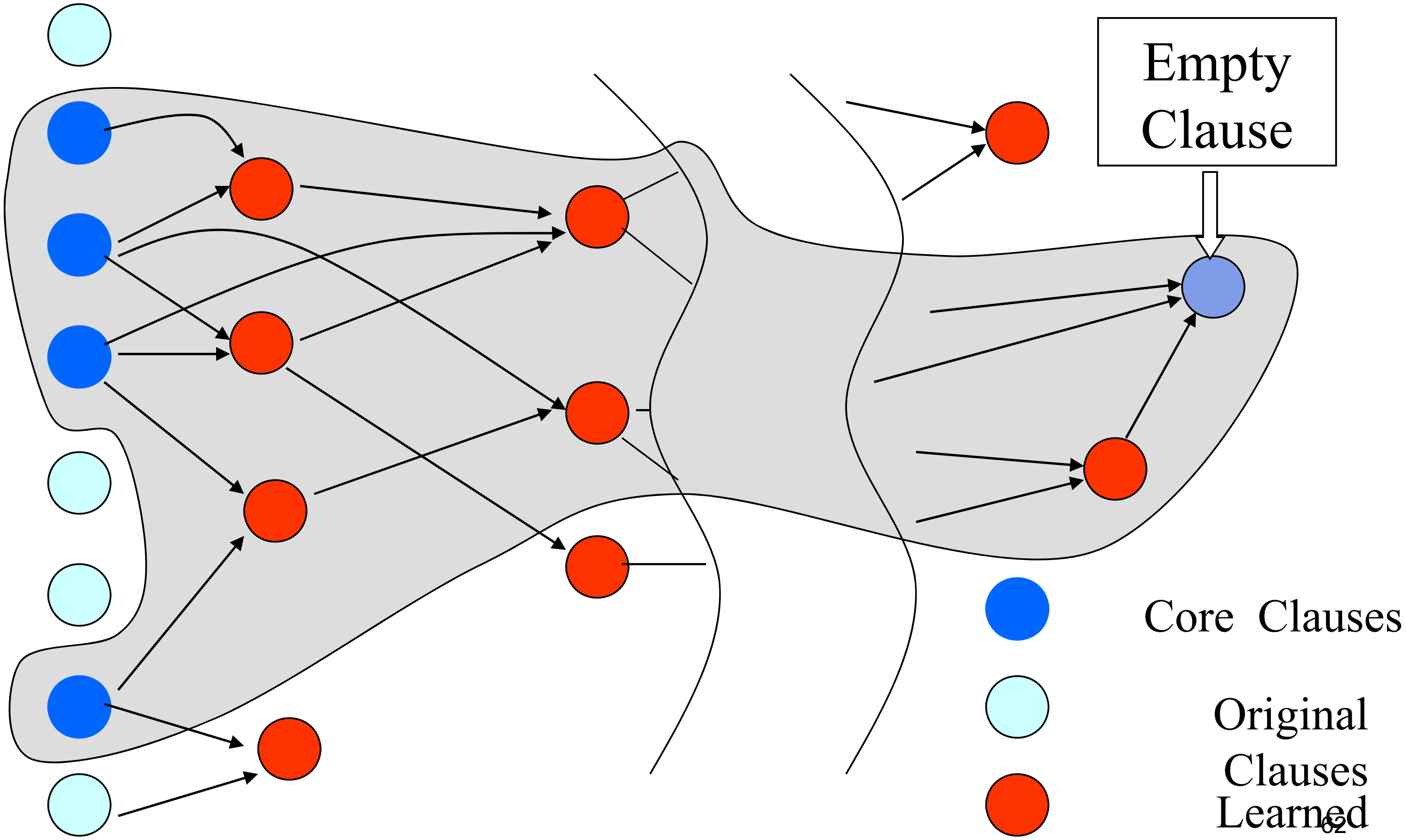


Empty Clause

Original Clauses

Learned Clauses

Proof of Unsatisfiability and Unsat Core



Empty Clause

● Core Clauses

○ Original Clauses

● Learned Clauses

Problem Definition

- MinCostSAT: Given a Boolean formula φ with
  - n variables $x_1, x_2, \ldots, x_n$ $x_i \in \{0,1\}$ each costs $c_i \geq 0$ $1 \leq i \leq n$
- Find a variable assignment $X \in \{0,1\}^n$
  - X satisfies φ
  - X minimizes

$$C = \sum_{i=1}^{n} c_i x_i$$

Partial MAX-SAT (PM-SAT) Problem Definition

- Two sets of clauses
  - Non-relaxable or hard
  - Relaxable or soft

$(x'_1 + x_2)(x'_1 + x'_2)[x_1 + x_3][x_1]$

- Objective – A truth assignment that
  - Satisfies all non-relaxable clauses
  - Satisfies maximum number of relaxable clauses

$x_1 = 0, x_2 = 0, x_3 = 1$

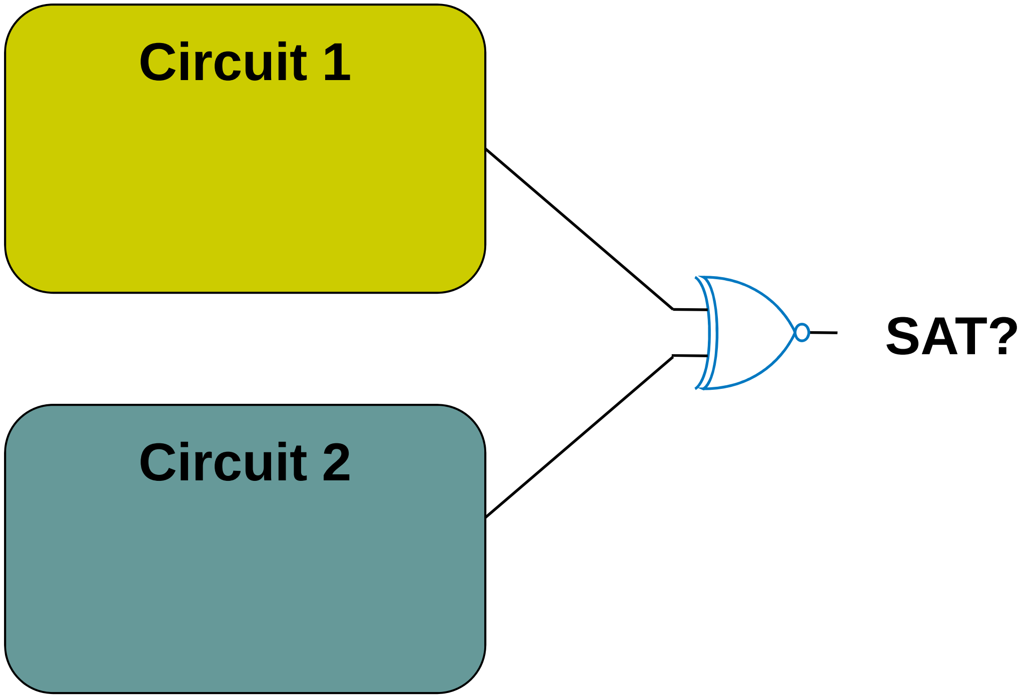
- Along the spectrum of SAT and MAX-SAT
  - All clauses are non-relaxable → Classical SAT
  - All clauses are relaxable → MAX-SAT

SAT Applications:

Equivalence Checking, Model Checking, Bounded Model Checking

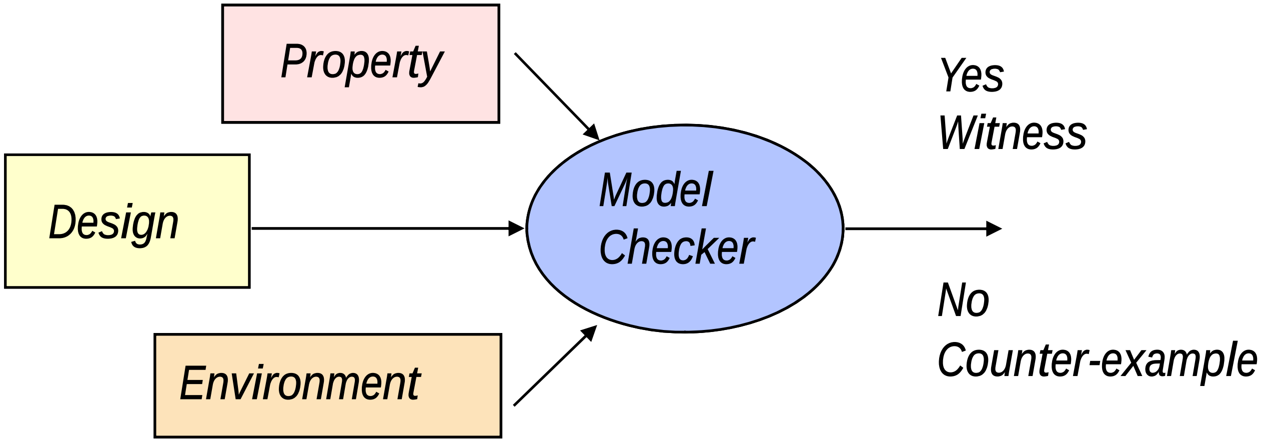
in Hardware and Software Verification

Equivalence Checking



Model Checking Finite State Systems (MC)

(Slides courtesy:
A. Gupta, NEC)



- **Model checker:** **Checks whether the design satisfies the property by exhaustive state space traversal** [Clarke *et al.* 82]
- **Advantages**
  - **Automatic verification method**
  - **Provides error traces for debugging**
  - **No test vectors required: all inputs are automatically considered**
  - **Sound and complete (no false proofs, no false bugs)**
- **Practical Issues**
  - **State space explosion (exponential in number of state elements)**
  - **The system needs to be closed**

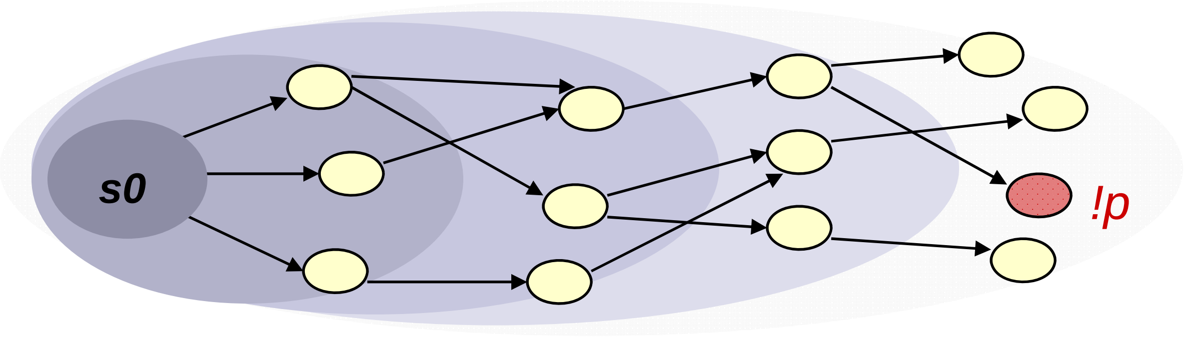
    i.e. we need to model the environment (constraints on design inputs, or models)

Automatic Property Verification

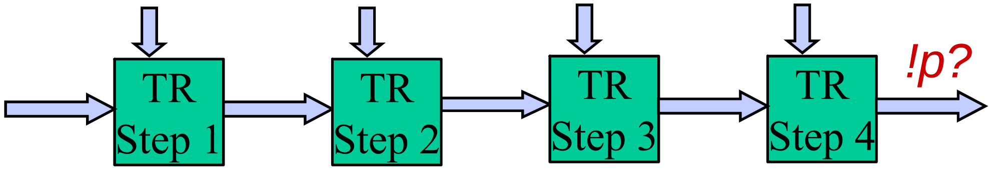
(Slides courtesy:
A. Gupta, NEC)

- ☐ **Verification Approach: e.g. Model Checking**
  - **Exhaustive state space exploration**
  - **Maintains a representation of visited states (explicit states, BDDs, ckt graphs …)**
  - **Expensive, need abstractions and approximations**
- ☐ **Falsification Approach: e.g. *Bounded* Model Checking**
  - **State space search for bugs (counter-examples) or test case inputs**
  - **Typically does not maintain representation of visited states**
  - **Less expensive, but need good search heuristics**

Model Checking AG p
Does the set of states
reachable from s0
contain a bad state(s)?



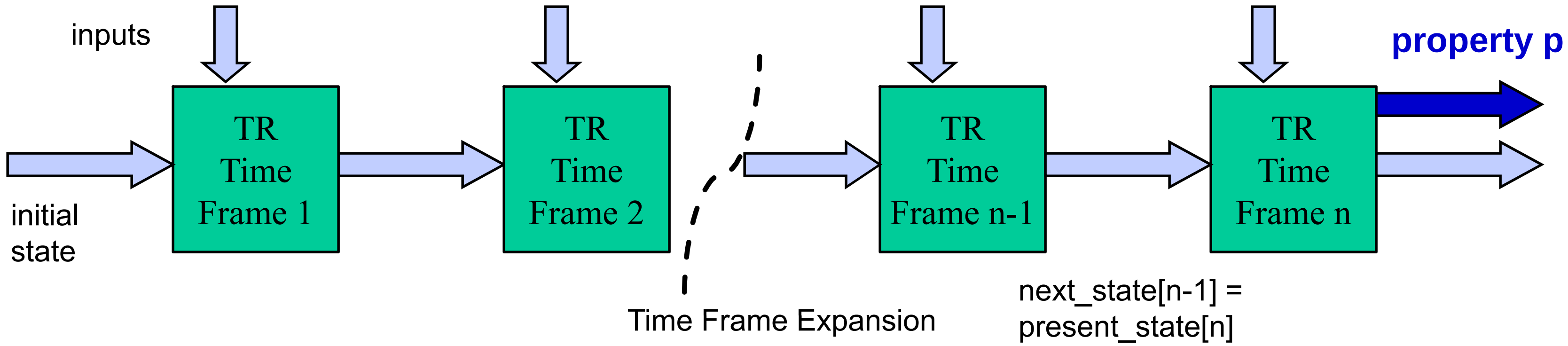
Bounded Model Checking
Is there is a path from
the initial state s0
to the bad state(s)?



Bounded Model Checking (BMC)

(Slides courtesy:
A. Gupta, NEC)

- **Main idea: Unroll transition relation logic up to some *bounded length* to check p**



inputs

property p

TR Time Frame 1

TR Time Frame 2

TR Time Frame n-1

TR Time Frame n

initial state

Time Frame Expansion

next_state[n-1] =
present_state[n]

- **BMC problem translated to a Boolean formula *f*** [Biere *et al.* 00]
  - A bug exists of length *k* ⇔ $SAT(f_k)$ (formula is satisfiable)
  - Satisfiability of f_k is checked by a standard SAT solver

- **Falsification: Can check for bounded length bugs**
  - Scales much better than use of Binary Decision Diagrams (BDDs)
    - BDDs: 100s of state elements
    - SAT-based BMC: 10k of state elements

- **SMT-based BMC can potentially improve performance due to word-level reasoning**

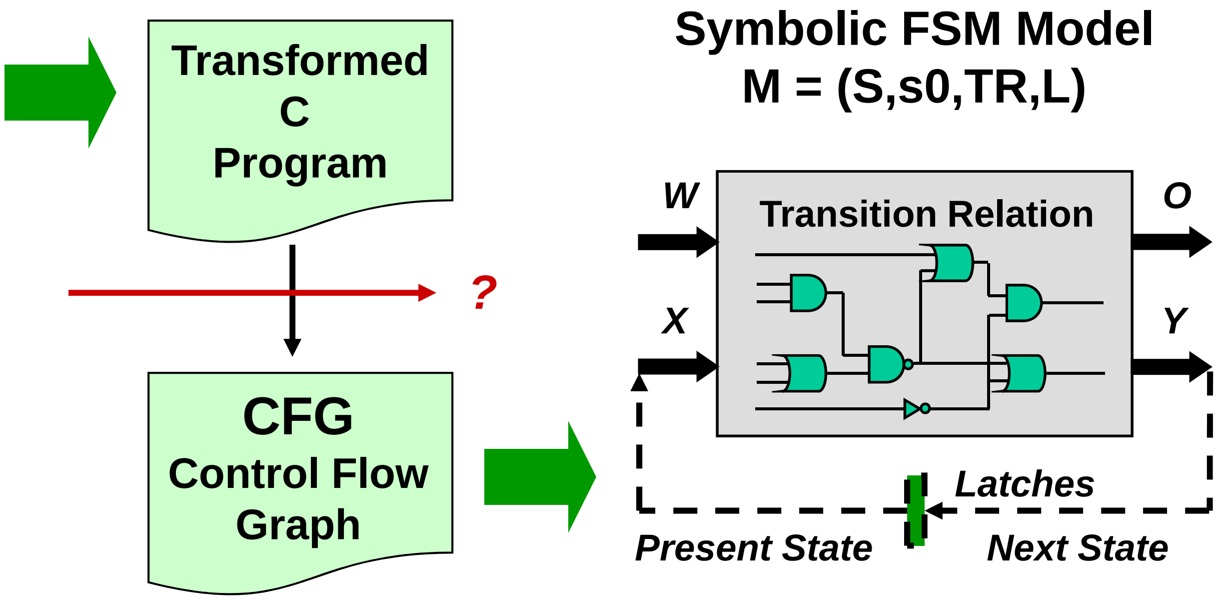
From HW Verification to SW Verification

(Slides courtesy:
A. Gupta, NEC)

C Program

```
void bar() {
    int x = 3 , y = x-3 ;
    while ( x <= 4 ) {
        y++ ;
        x = foo(x);
    }
    y = foo(y);
}

int foo ( int l ) {
    int t = l+2 ;
    if ( t>6 )
        t - = 3;
    else
        t --;
    return t;
}
```


Symbolic FSM Model
M = (S,s0,TR,L)

- ☐ **Source-to-source transformations**
  - For modeling pointers, arrays, structures …
- ☐ **Control Flow Graph: Intermediate Representation**
  - Well-studied optimizations provide simplification and reduction in size of verification models
  - Allows separation of model *building* phase from model *checking* phase

References

- [GJ79] Michael R. Garey and David S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, W. H. Freeman and Company, San Francisco, 1979
- [DP 60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM, 7:201–215, 1960*
- [DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM, 5:394–397, 1962*
- [SS99] J. P. Marques-Silva and Karem A. Sakallah, “GRASP: A Search Algorithm for Propositional Satisfiability”, *IEEE Trans. Computers*, C-48, 5:506-521, 1999.
- [BS97] R. J. Bayardo Jr. and R. C. Schrag “Using CSP look-back techniques to solve real world SAT instances.” *Proc. AAAI*, pp. 203-208, 1997
- [BS00] Luís Baptista and João Marques-Silva, “Using Randomization and Learning to Solve Hard Real-World Instances of Satisfiability,” In *Principles and Practice of Constraint Programming – CP 2000,* 2000*.*

References

- [H07] J. Huang, “The effect of restarts on the efficiency of clause learning,” Proceedings of the Twentieth International Joint Conference on Automated Reasoning, 2007
- [MMZ+01] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang and S. Malik. Chaff: Engineering and efficient sat solver. In *Proc., 38th Design Automation Conference (DAC2001), June* 2001.
- [ZS96] H. Zhang, M. Stickel, “An efficient algorithm for unit-propagation” In Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics,1996
- [ES03] N. Een and N. Sorensson. An extensible SAT solver. In SAT-2003
- [B02] F. Bacchus “Exploring the Computational Tradeoff of more Reasoning and Less Searching”, *Proc. 5th Int. Symp. Theory and Applications of Satisfiability Testing*, pp. 7-16, 2002.
- [GN02] E.Goldberg and Y.Novikov. BerkMin: a fast and robust SAT-solver. In *Proc., DATE-2002, pages 142–149, 2002.*

References

- [R04] L. Ryan, Efficient algorithms for clause-learning SAT solvers, M. Sc. Thesis, Simon Fraser University, 2002.
- [EB05] N. Eén and A. Biere. Effective Preprocessing in SAT through Variable and Clause Elimination, In Proceedings of SAT 2005
- [ZM03] L. Zhang and S. Malik, Validating SAT solvers using an independent resolution-based checker: practical implementations and other applications, In Proceedings of Design Automation and Test in Europe, 2003.
- [LSB07] M. Lewis, T. Schubert, B. Becker, Multithreaded SAT Solving, In Proceedings of the 2007 Conference on Asia South Pacific Design Automation
- [HJS08] Youssef Hamadi, Said Jabbour, and Lakhdar Sais, ManySat: solver description, Microsoft Research-TR-2008-83