SAT-Solving: From Davis-Putnam to Zchaff and Beyond

Day 3: Recent Developments

Lintao Zhang

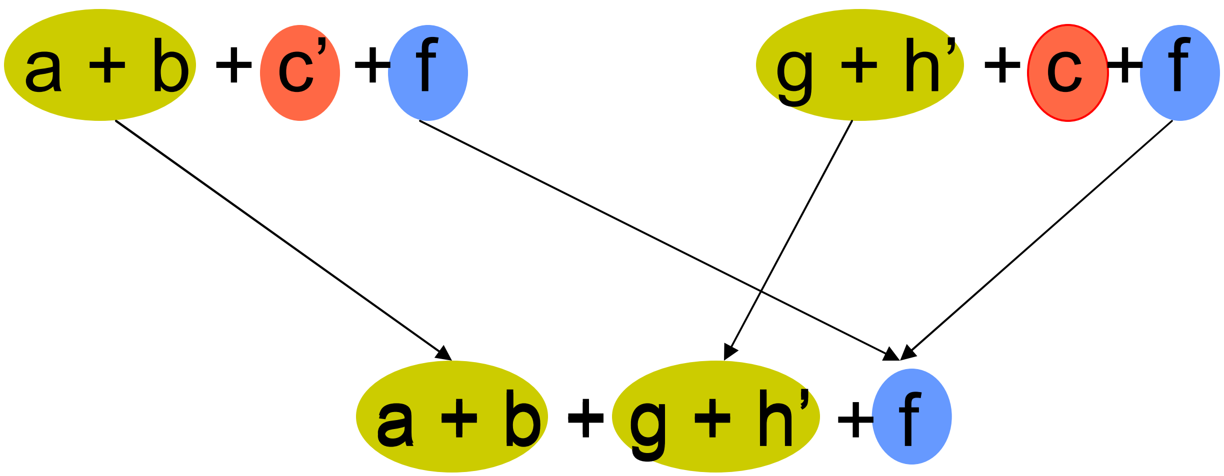
Microsoft Research

Requirements for SAT solvers in the Real World

- Fast & Robust
  - Given a problem instance, we want to solve it quickly
- Reliable
  - Can we depend on the SAT solver? i.e. is the solver bug free?
- Feature Rich
  - Incremental SAT Solving
  - Unsatisfiable Core Extraction
  - What are the other desirable features?
- Beyond SAT
  - Pseudo Boolean Constraint
  - Multi-Value SAT solving
  - Quantified Boolean Formula (QBF)

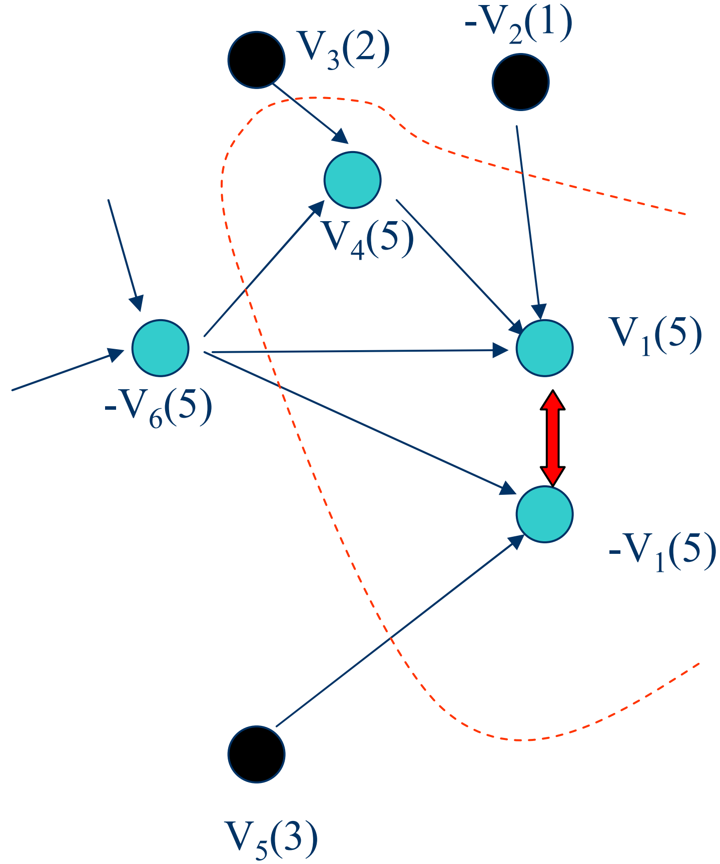
Resolution

- Resolution of a pair of clauses with exactly ONE incompatible variable
  - Two clauses are said to have distance 1



Lintao Zhang

Conflict Analysis as Resolution



$(V_2 + V_3' + V_5' + V_6)$

$(V_3' + V_6 + V_4)$

$(V_6 + V_5' + V_1')$

$(V_2 + V_4' + V_6 + V_1)$

$(V_2 + V_4' + V_6 + V_5')$

$(V_2 + V_3' + V_5' + V_6)$

Lintao Zhang

Microsoft Research

Key Observations

- DLL with learning is nothing but a resolution process
  - Has the same limitation as resolution
    - Certain class of instances require exponential sized resolution proof. Therefore, it will take exponential time for DLL SAT solver
- We can use this for
  - Certification / Correctness Checking
  - Unsatisfiable Core Extraction
  - Incremental SAT solving

Motivation for SAT Solver Validation

- Certify automatic reasoning tools:
  - Required for mission critical applications
    - Train Station Safety Check
  - Available for some theorem provers and model checkers
- Do we really need the validation?
  - Modern SAT solvers are intricate pieces of software (e.g. zchaff contains about 10,000 lines of code)
  - Bugs are abundant in SAT solvers

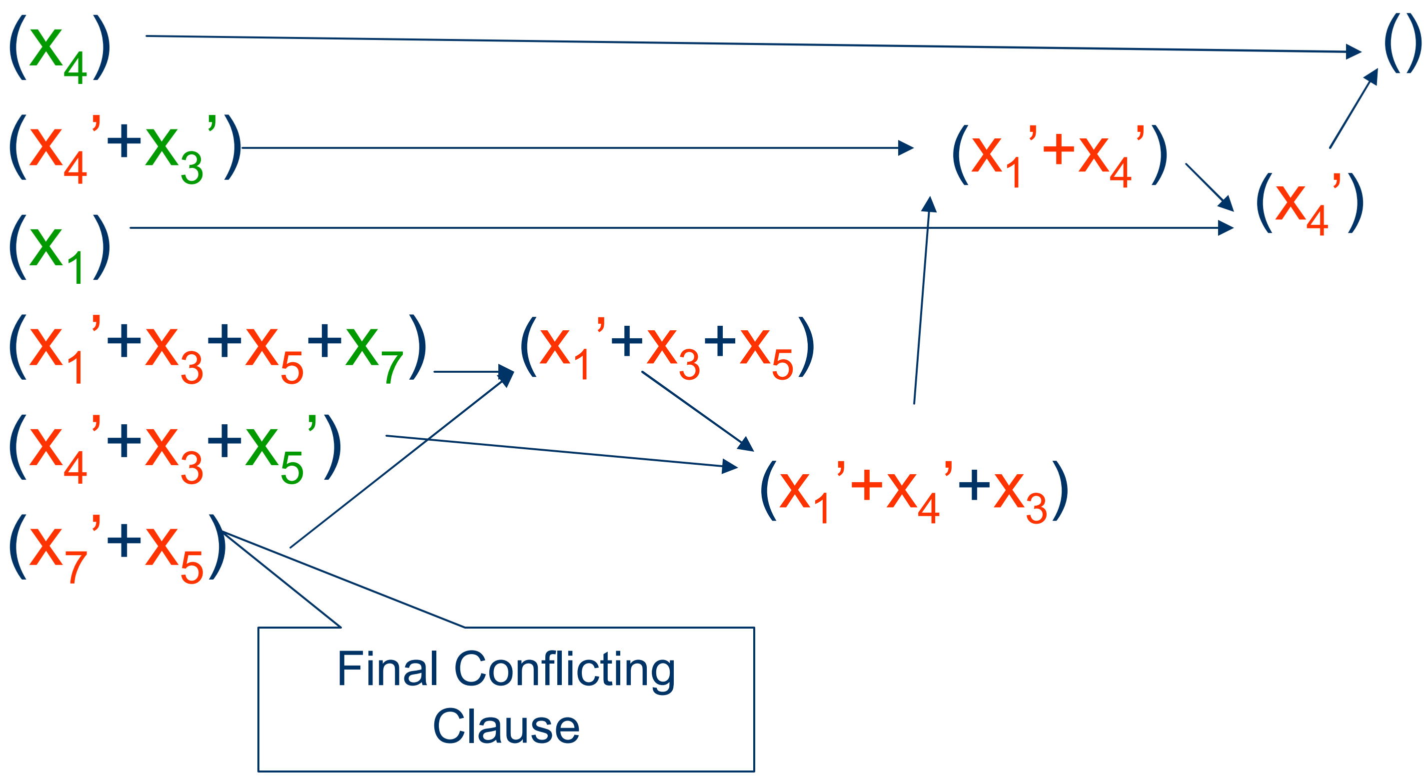
Certify a SAT Solver

- The correctness of a SAT solver:
  - If it claims the instance is satisfiable, it is easy to check the claim.
  - How about unsatisfiable claims?
    - Traditional method: run another SAT Solver
    - Time consuming, and cannot guarantee correctness
- Needs a formal check for the proof, similar to the check for the validity of a proof in math.
  - Must be automatic.
  - Must be able to work with current state-of-the-art SAT solvers

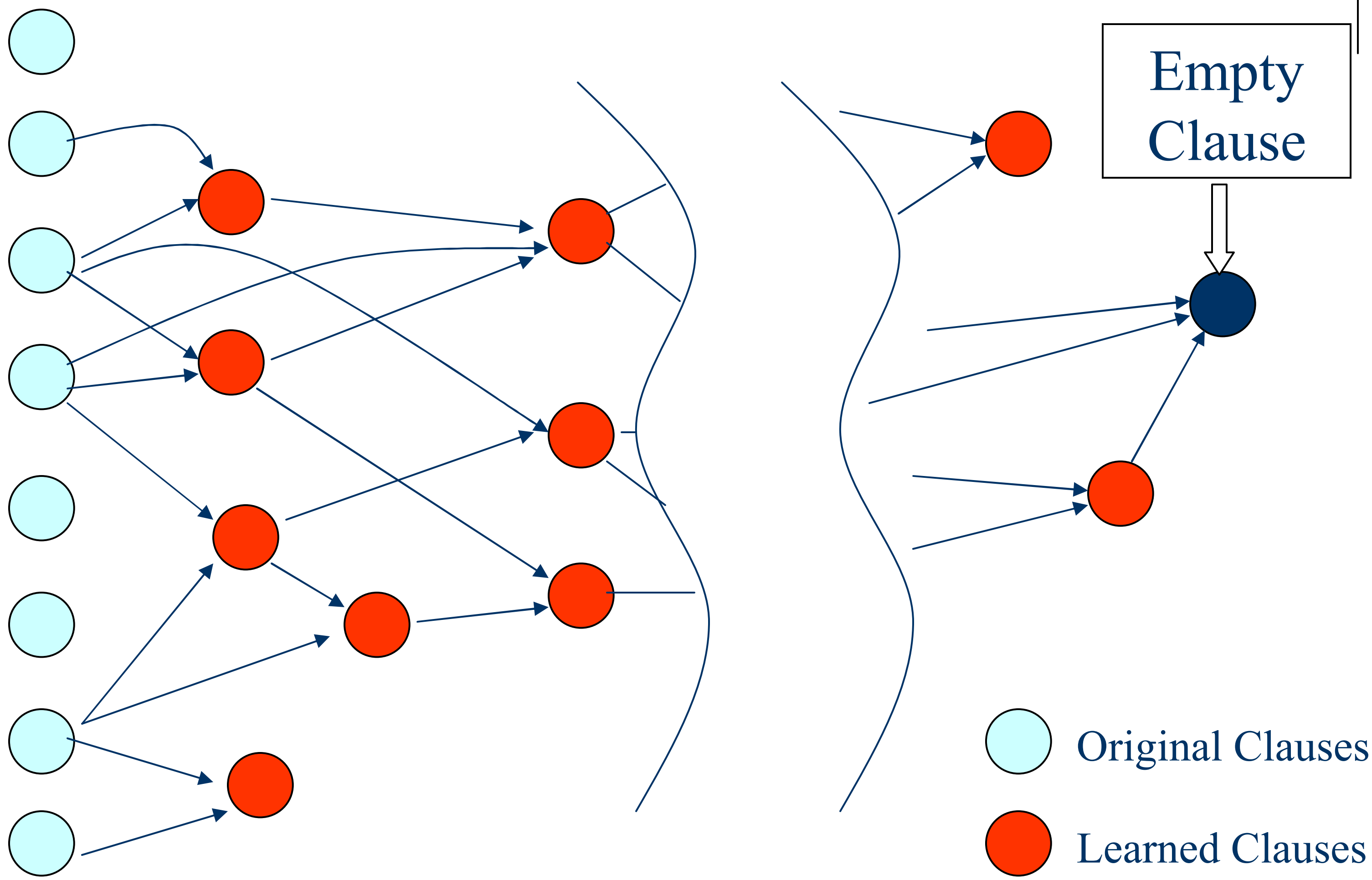
DLL with Learning

```
while(1) {
   if (decide_next_branch()) { //Branching
       while(deduce()==conflict) { //Deducing
              blevel = analyze_conflicts(); //Learning
              if (blevel < 0)
                     return UNSAT;
              else back_track(blevel); //Backtracking
       }
   else //no branch means all variables got assigned.
       return SATISFIABLE;
   }
```

Correct in Unsatisfiable Case



Resolution Graph



Microsoft Research

An Independent Checker

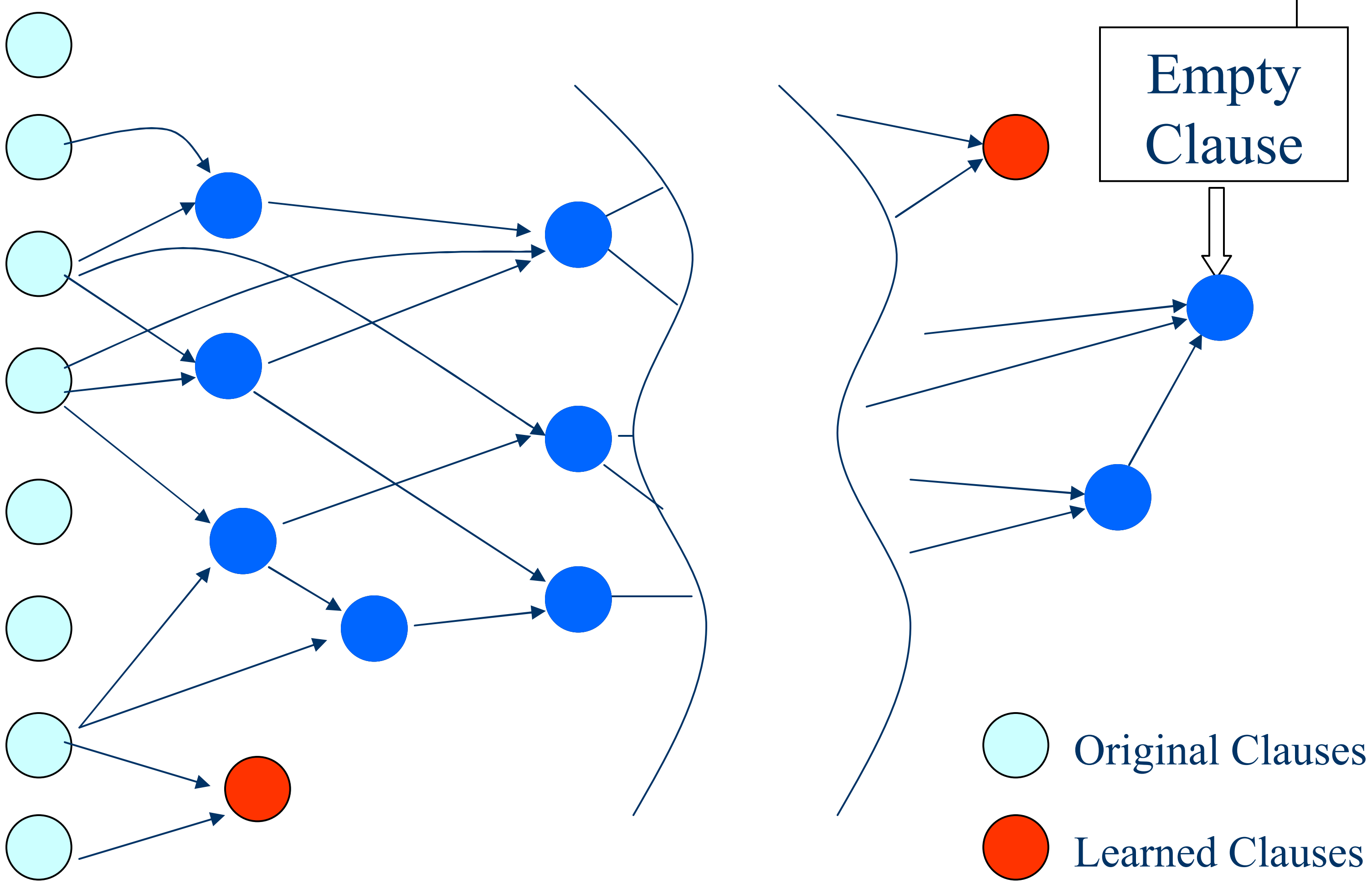
Strategy:

- SAT solver dump out a trace during the solving process representing the resolution graph
- Using a third party checker to construct the empty clause by resolution using the hint provided by the trace
- Trace only contain *resolve sources* of each learned clauses. Need to reconstruct the clause literals by resolution from original clauses

Practical Implementation: Depth First

- Start from the empty clause, recursively reconstruct all needed clause.
- Fast, because it only needs to reconstruct clauses that are needed for the proof.
- But may fail because of memory overflow on hard instances.

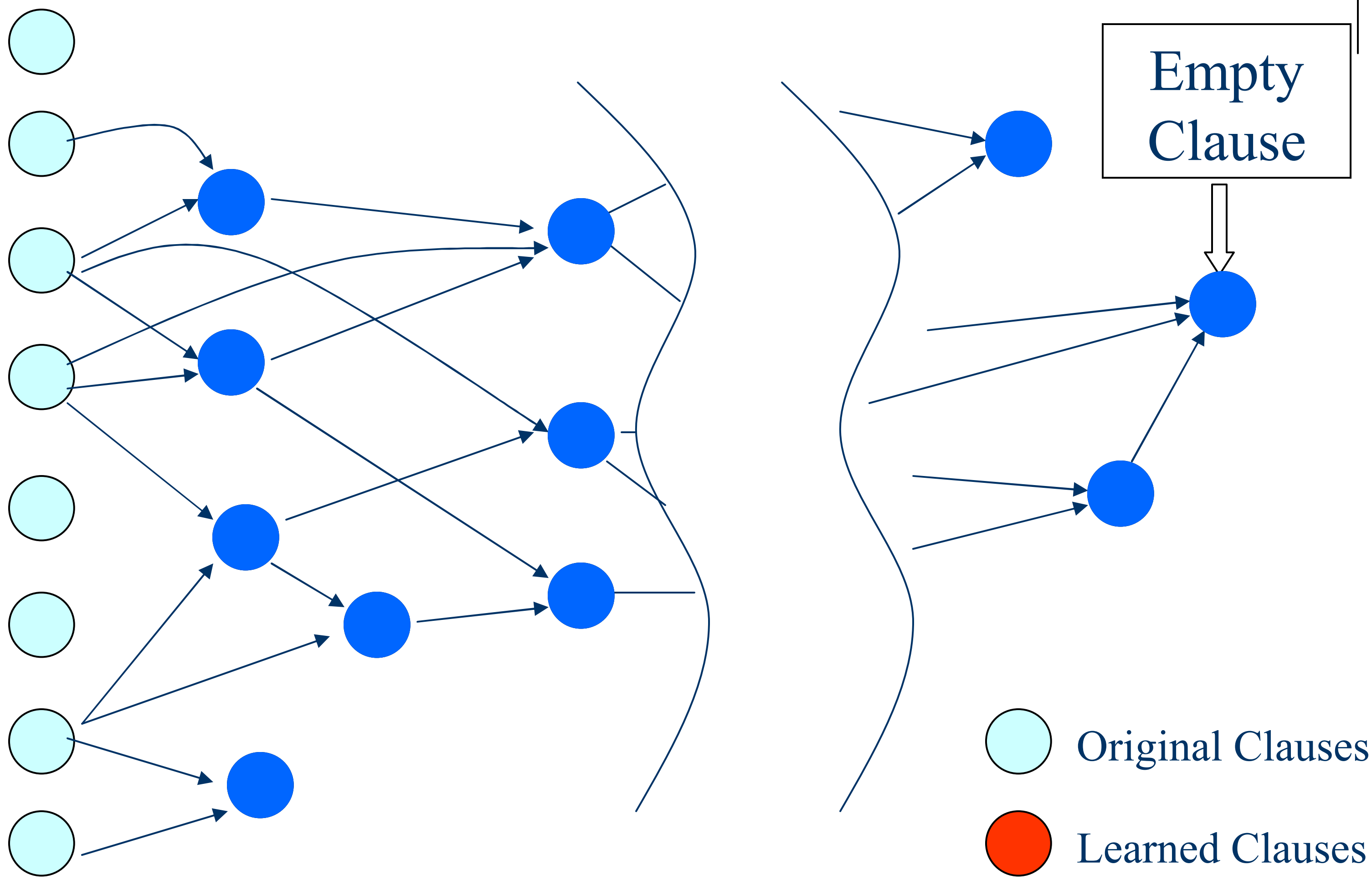
Depth First Approach



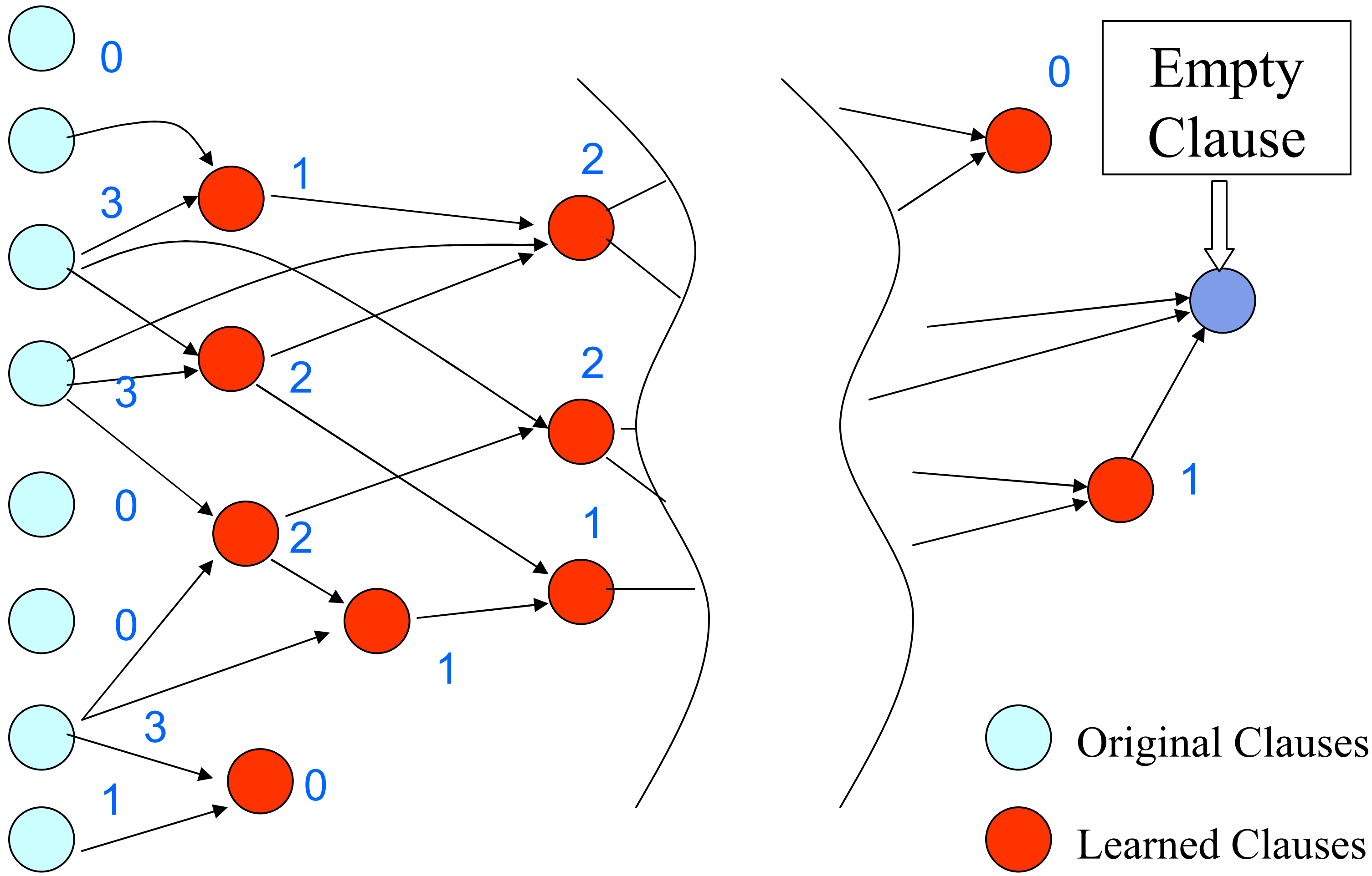
Practical Implementation: Breadth First

- Start from the original clauses and construct clauses in the same order as they appear
- Slower, because all the clauses need to be reconstructed
- No memory overflow problem if we delete clauses when they are not needed anymore.

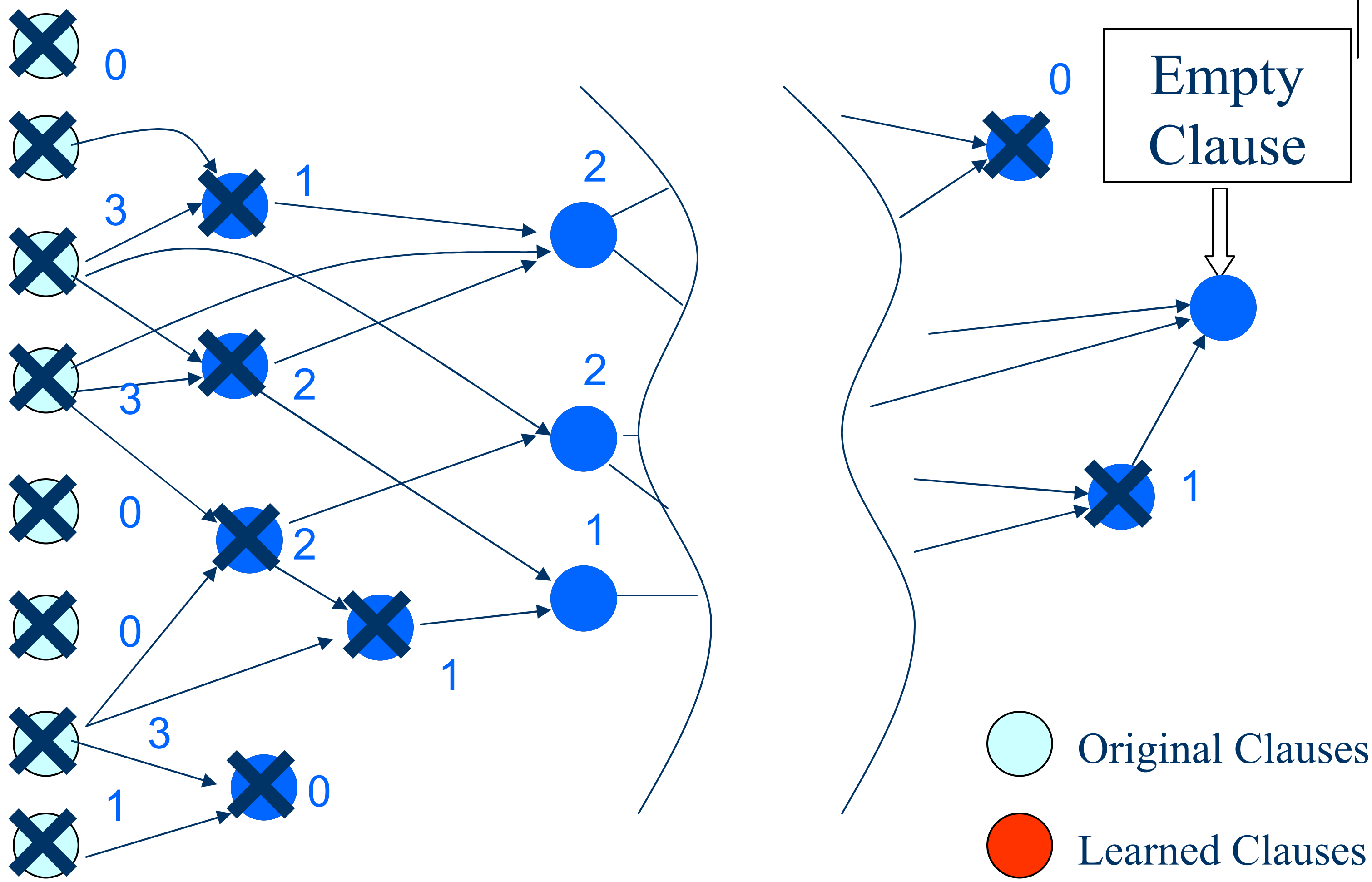
Breadth First Approach



Calculate Fan-outs in Breadth First Approach



Calculate Fan-outs in Breadth First Approach



Experimental Results

Instance Name	Num. Variables	Orig. Num. Clauses	Runtime	Trace Overhead
2dlx_cc_mc_ex_bp_f	4583	41704	3.3	11.89%
bw_large.d	5886	122412	5.9	9.12%
c5315	5399	15024	22.0	10.45%
too_largefs3w8v262	2946	50216	40.6	7.68%
c7552	7652	20423	64.4	8.76%
5pipe_5_ooo	10113	240892	118.8	4.51%
barrel9	8903	36606	238.2	4.51%
longmult12	5974	18645	296.7	6.17%
9vliw_bp_mc	20093	179492	376.0	4.26%
6pipe_6_ooo	17064	545612	1252.4	3.39%
6pipe	15800	394739	4106.7	2.77%
7pipe	23910	751118	13673.0	1.68%

* Experiments are carried out on a PIII 1.13Ghz Machine with 1G Mem

Experimental Results

Instance Name	Depth-First		Breadth-First	
	Time(s)	Mem(k)	Time(s)	Mem(k)
2dlx	0.84	7860	1.30	4652
bw_large.d	1.48	8720	2.44	9920
c5315	2.8	18108	5.19	3732
too_large	3.79	26752	5.47	6164
c7552	6.16	41420	11.44	5976
5pipe_5_ooo	6.6	50044	13.29	17936
barrel9	4.85	31456	10.46	6752
longmult12	25.9	154288	41.22	7488
9vliw_bp_mc	12.8	126752	33.81	17724
6pipe_6_ooo	38.5	249468	102.67	40136
6pipe	*	*	301.98	40248
7pipe	*	*	645.33	62620

Unsatisfiable Core Extraction: Problem Definition

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 $m \le n$

Example:

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

Unsatisfiable Core Extraction: Problem Definition

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 $m \le n$

Example:

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

Unsatisfiable Core Extraction: Problem Definition

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 $m \leq n$

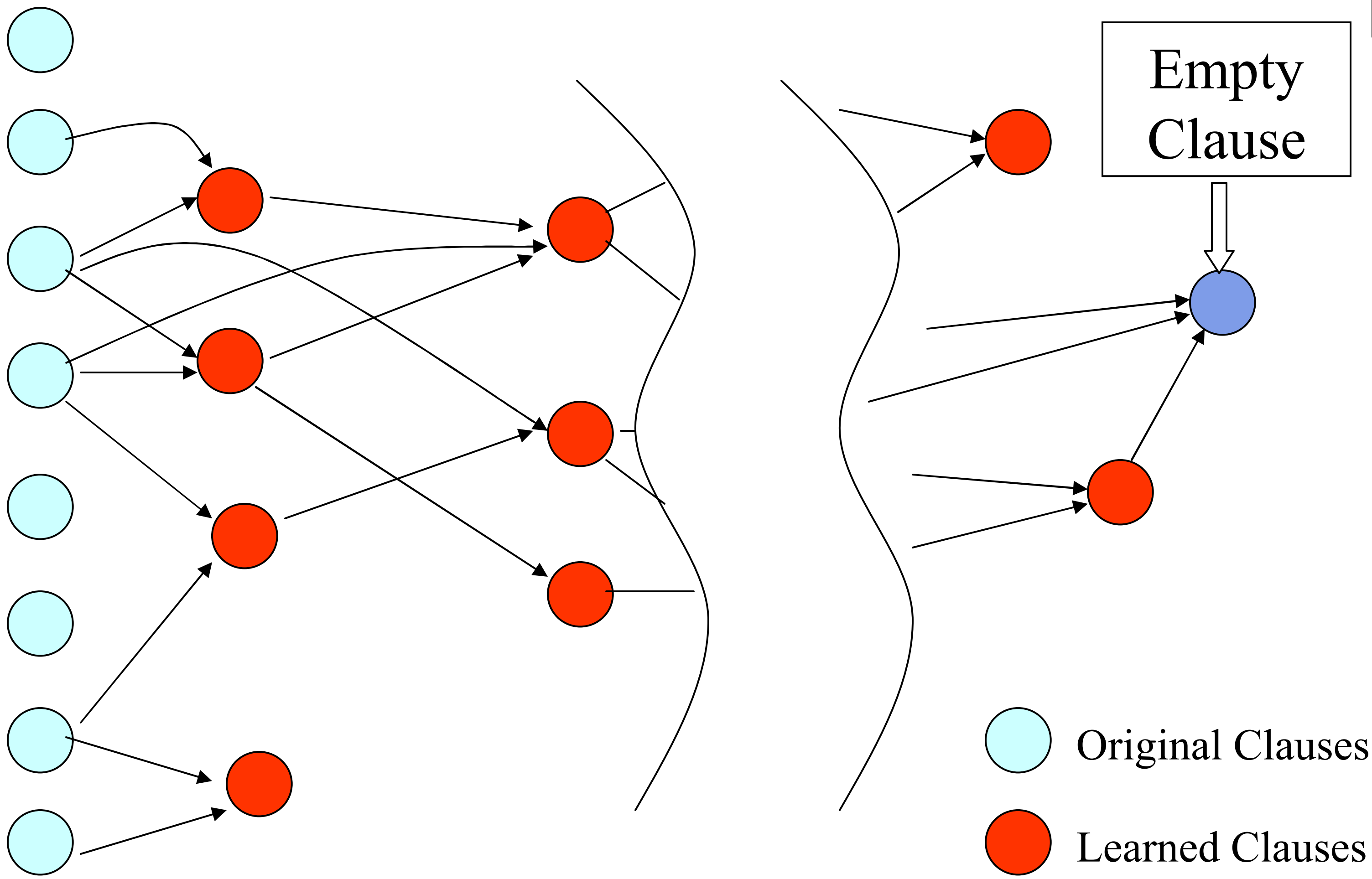
Example:

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

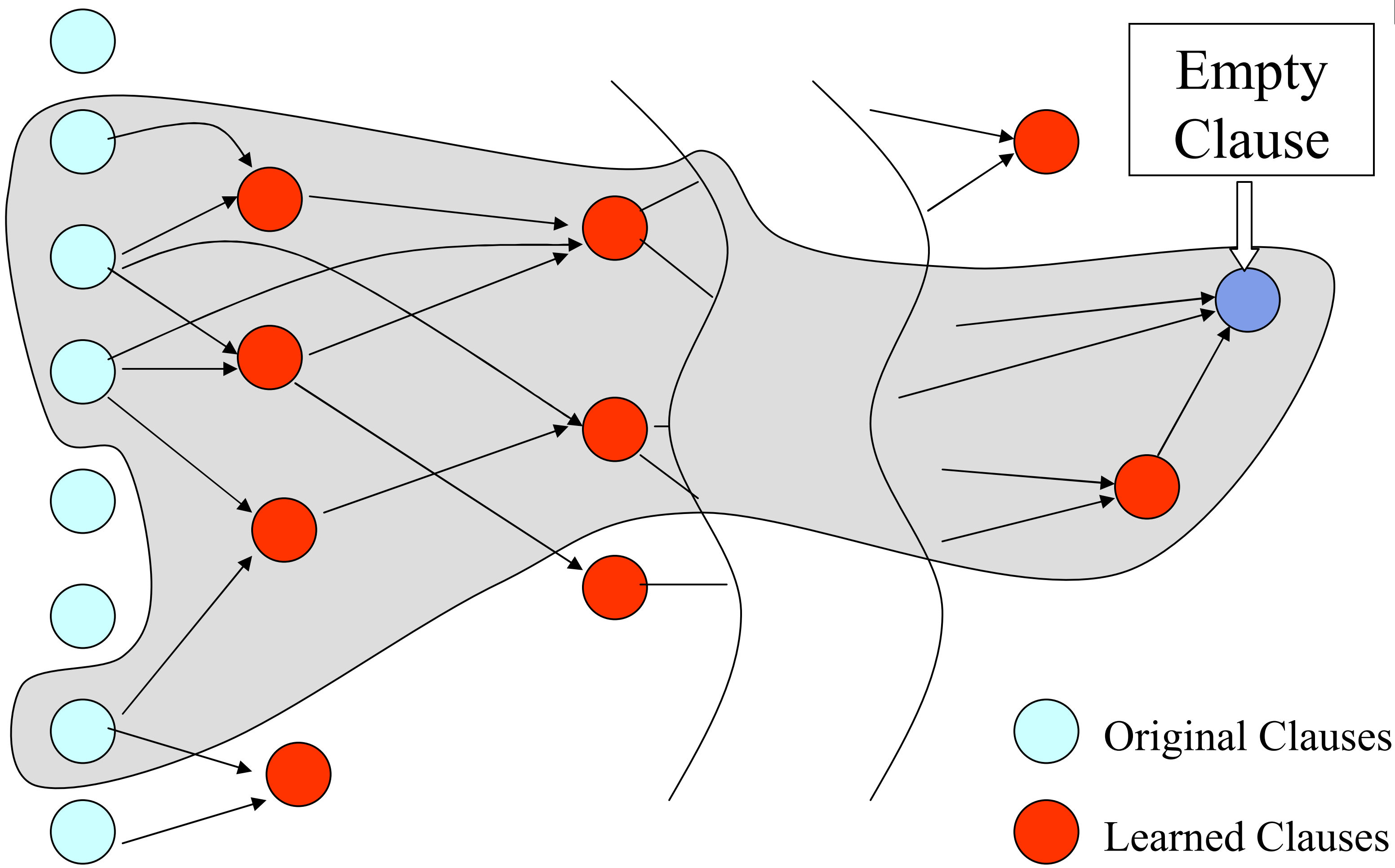
Motivation

- Debugging and redesign: SAT instances are often generated from real world applications with certain expected results:
  - If the expected results is unsatisfiable, but we found the instance to be satisfiable, then the solution is a “counter example” or “input vector” for debugging
    - Train station safety checking
    - Combinational Equivalence Checking
  - What if the expected results is satisfiable?
    - SAT Planning
    - FPGA Routing
- Relaxing constraints:
  - If several constraints make a safety property holds, are there any redundant constraints in the system that can be removed without violate the safety property?
  - Abstraction for model checking: Ken McMillan & Nina Alma, TACAS03; A. Gupta et al, ICCAD 2003

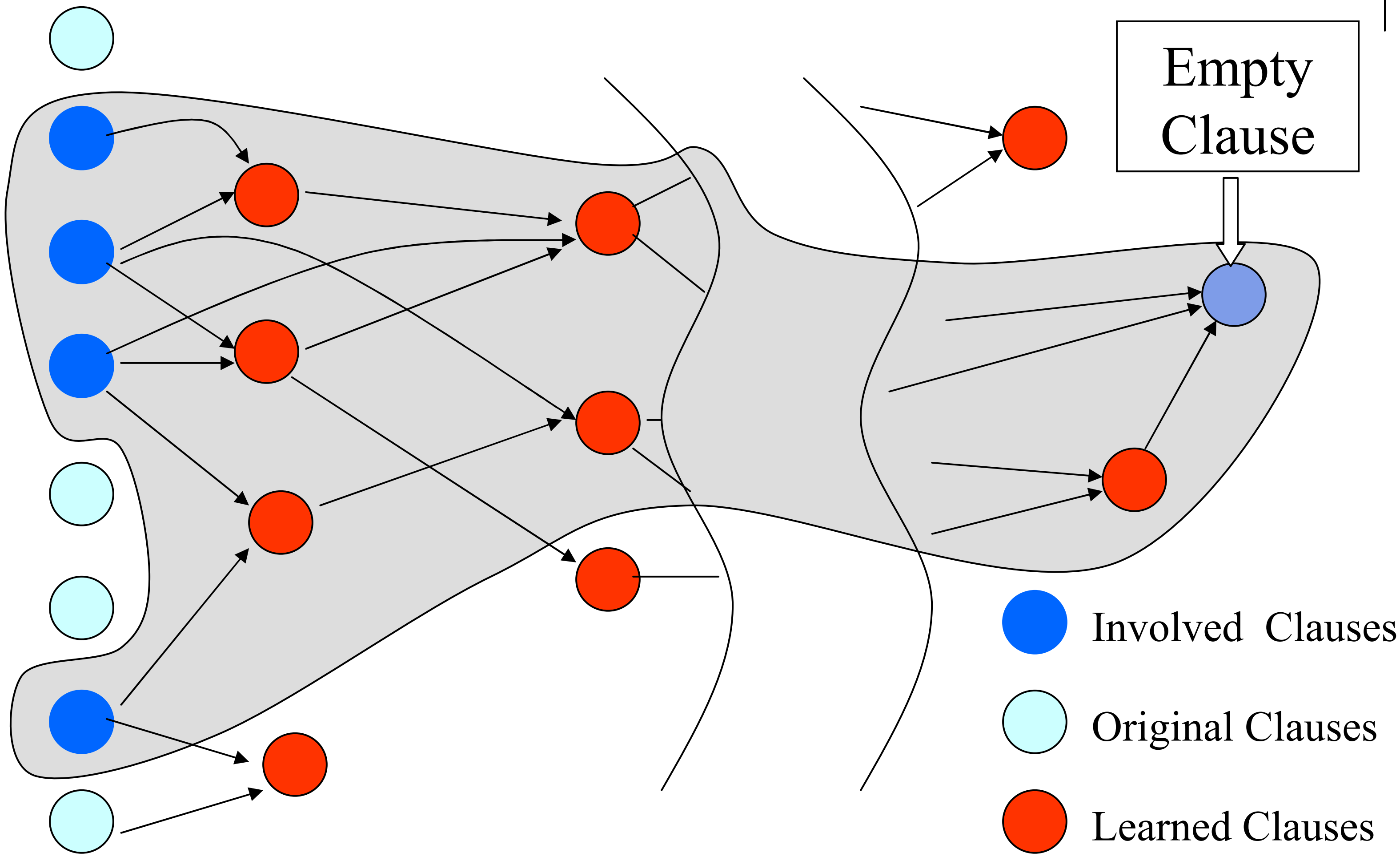
Proposed Approach



Proposed Approach



Proposed Approach



Lintao Zhang

Microsoft Research

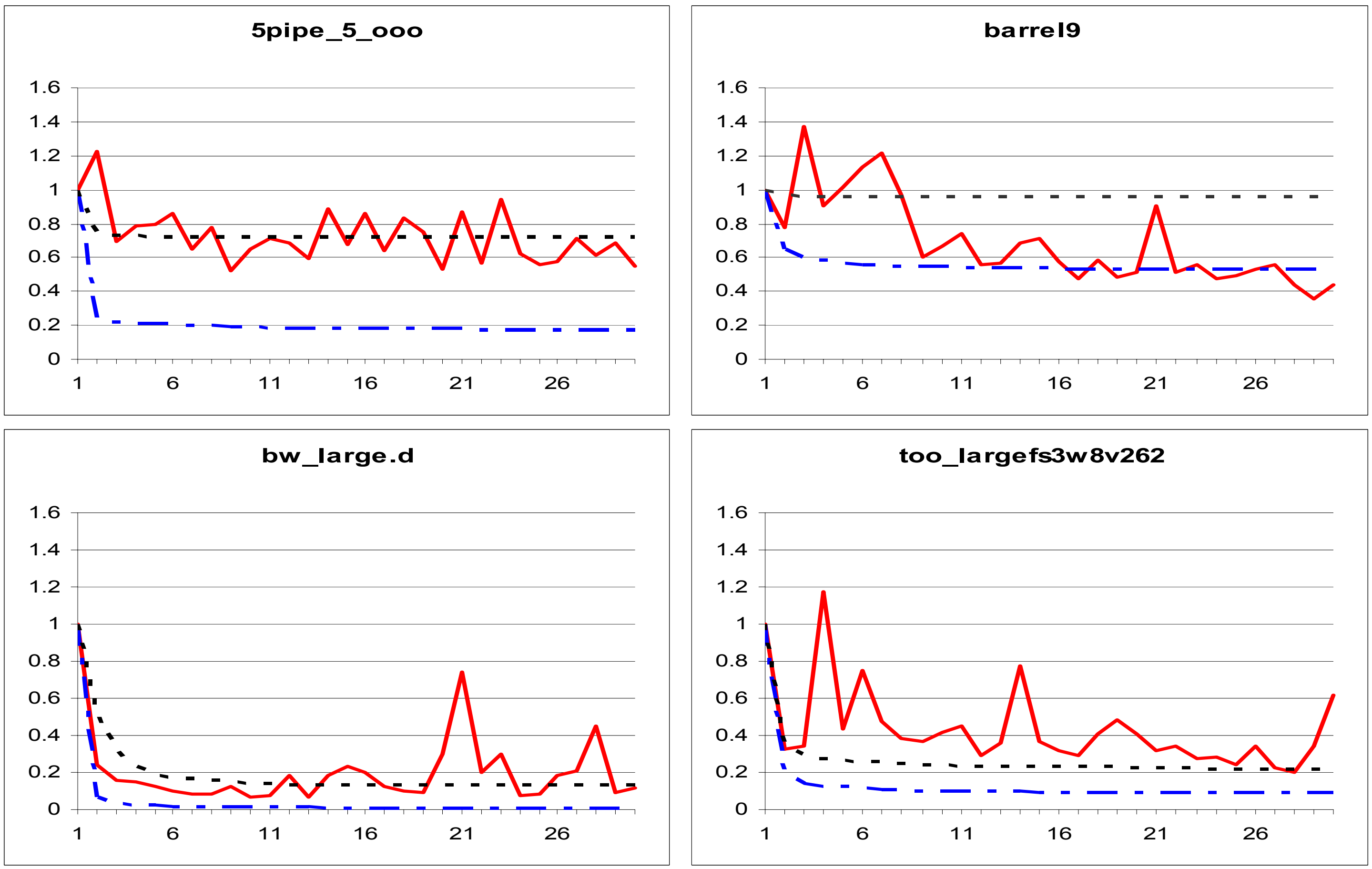
Implementation Issues

- No need to check the integrity of the graph
- Graph too large, have to traverse on disk
  - The nodes of the graph is already topologically ordered
  - But we need to reverse it
- Can iteratively run the procedure to obtain smaller cores
- Cannot guarantee the core to be minimal or minimum. Depends on the SAT solver for the quality of the core extracted

Experimental Results

Instance Name	Before		After		Clause Ratio	Run Time (s)
	Num Cls	Num.Vars	Num. Cls	Num. Vars		
2dlx	41704	4524	11169	3145	26.8%	0.85
bw_large.d	122412	5886	8151	3107	6.7%	1.54
C5315	15024	5399	14336	5399	95.4%	2.64
too_large	50216	2946	10060	2946	20.0%	2.65
C7552	20423	7651	19912	7651	97.5%	5.37
5pipe_5_ooo	240892	10113	57515	7494	23.9%	6.62
Barrel9	36606	8903	23870	8604	65.2%	4.66
longmult12	18645	5974	10727	4532	57.5%	21.31
9vliw_bp_mc	179492	19148	66458	16737	37.0%	10.68
6pipe_6_ooo	545612	17064	180559	12975	33.1%	37.14
6pipe	394739	15469	126469	13156	32.1%	99.96
7pipe	753118	23910	221070	20188	29.3%	152.71

Core Extracted After Several Iterations



Lintao Zhang

Microsoft Research

The Quality of the Core Extracted

- Start from the smallest core that can be extracted by the proposed method (i.e. run till fixed point), delete clauses one by one till no clause can be deleted without change the satisfiability of the formula.
- The resulting core is a *minimal* core for the formula.
- Finding minimal core is time consuming.

Benchmark Instance	Original #Cls	Extracted # Cls	Iterations	Minimal #Cls	Clause Ratio
2dlx_cc_mc_ex_bp_f	41704	8036	26	7882	1.020
Bw_large.d	122412	1352	35	1225	1.104
Too_largefs3w8v262	50416	4464	32	3765	1.186

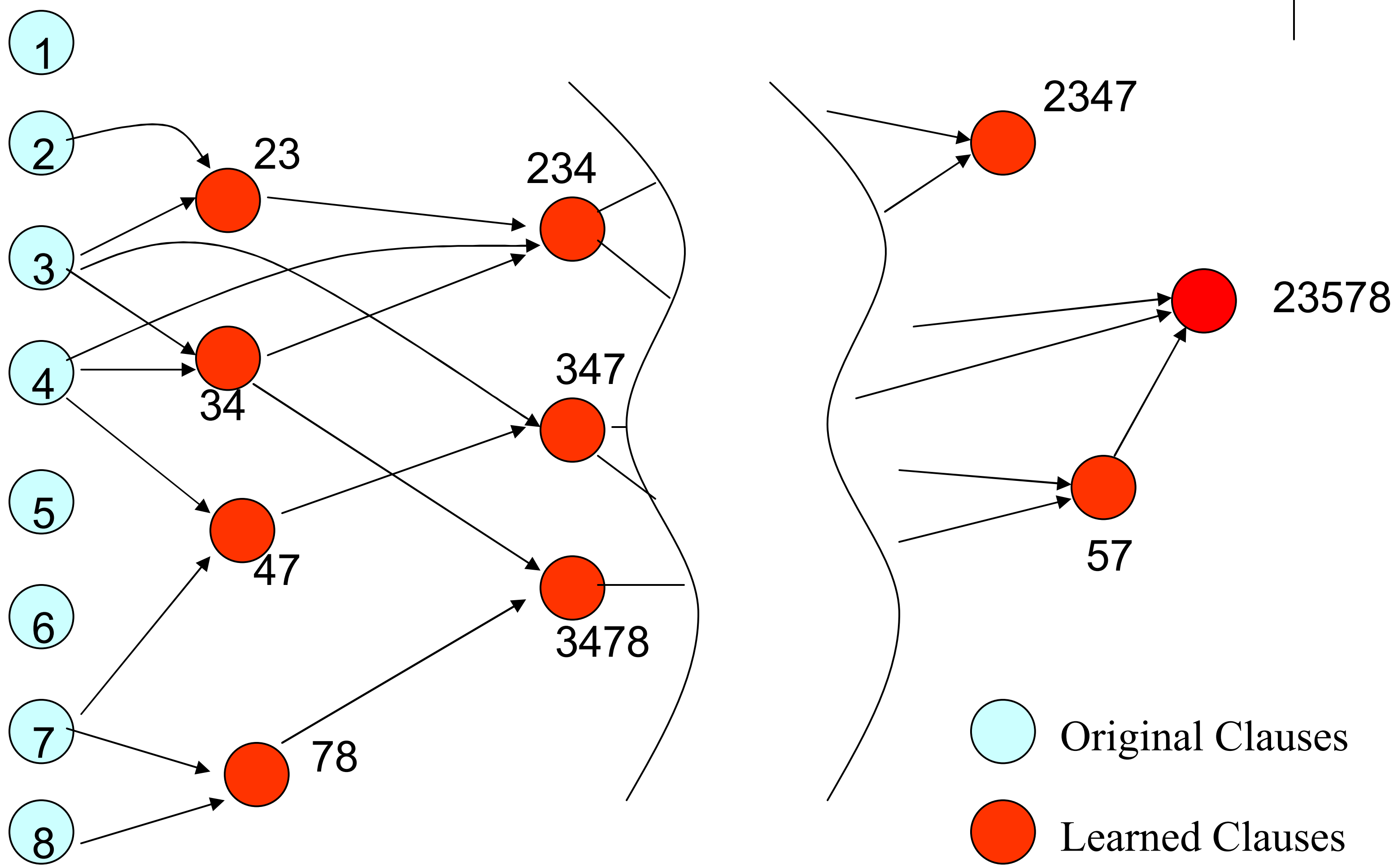
Incremental SAT Solving

- In real world, multiple SAT instances are generated to solve one problem
  - Combinational Equivalence Checking: equivalence multiple outputs are checked one by one
  - Bounded Model Checking: properties are checked at 1, 2, 3 … n cycles.
- These multiple SAT instances are often very similar, or have large common part
- Traditionally we solve them one by one independently
- Can we do better?

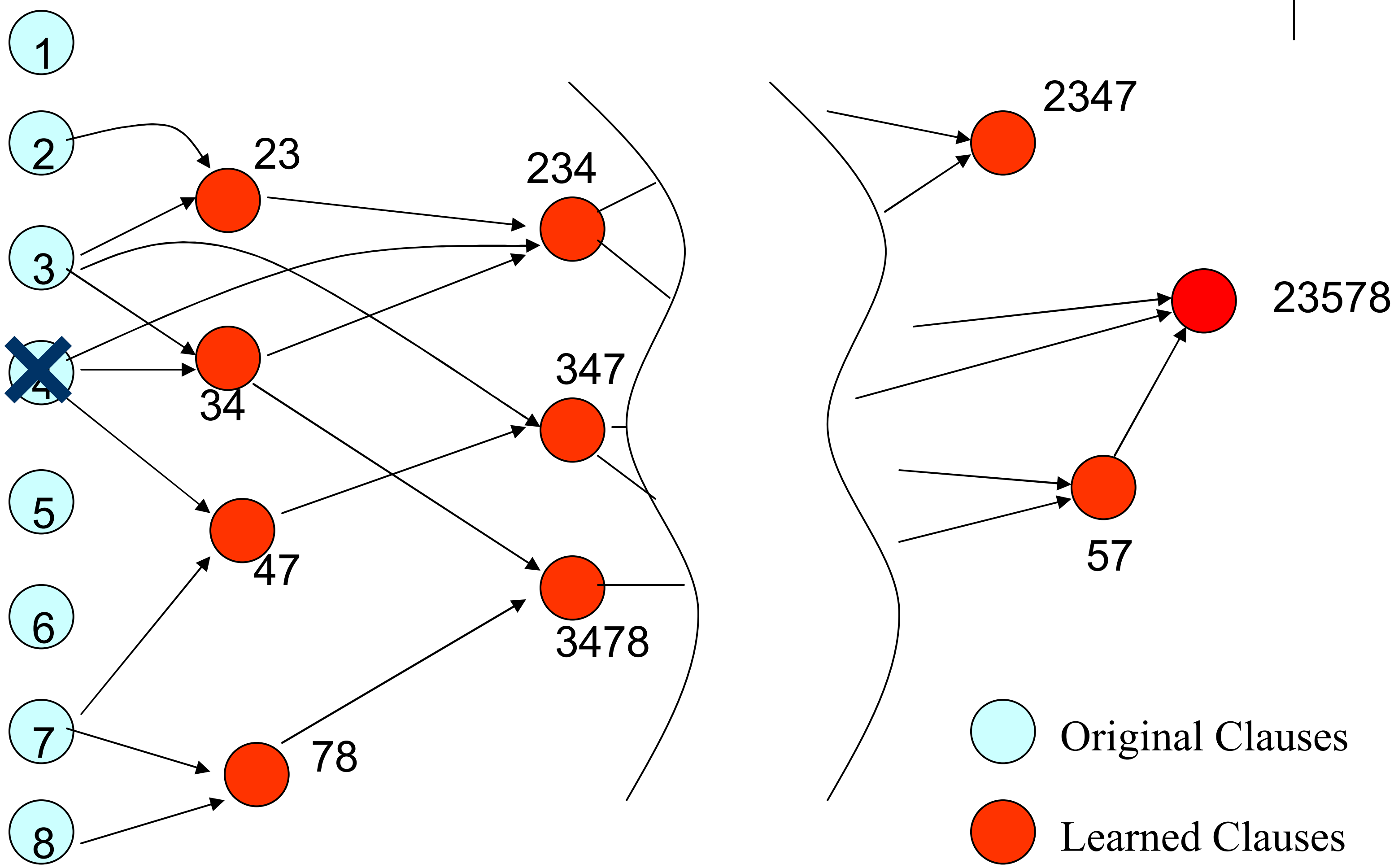
Incremental SAT Solving

- Previous efforts are recorded in the learned clauses
  - Try to keep the learned clauses
- To solve a series of SAT instances that differ only a little bit, we need to be able to
  - Add constraints to the clause database (easy)
    - If the original instance is UNSAT, the new instance is still UNSAT
    - If the original instance is SAT, while added constraint clauses are not conflicting under the satisfying assignment, then the new instance is still SAT
    - Otherwise, resolve the conflict and continue solve
  - Delete constraints from the database (tricky)
    - Some of the learned clauses are invalidated
    - Ofer Strichman, CAV2000

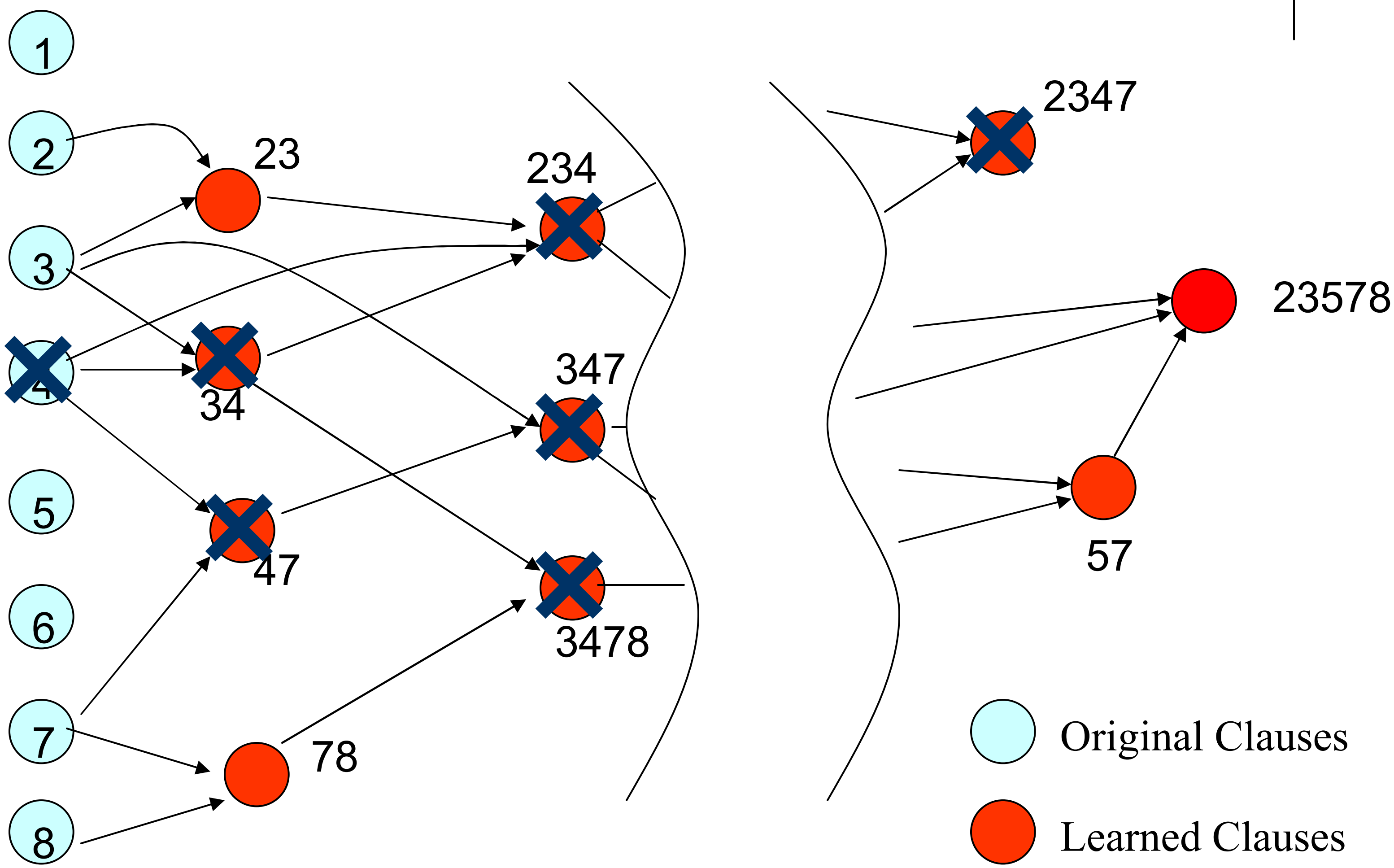
Find the Learned Clauses that are Invalidated



Find the Learned Clauses that are Invalidated



Find the Learned Clauses that are Invalidated



Engineering Questions

- How to do this efficiently?
  - It's too expensive for each learned clause to carry with it all it's resolve sources
  - Solution
    - Arrange the clauses into groups. Clauses are added and deleted a group at a time
    - Using bit vector for carrying the group information for each clause: 32 bit machine can have 32 groups of clauses, enough for most applications

Using SAT Techniques for Other types of Constraints

- SAT Limitations
  - Variables are in Boolean Domain
  - Constrains are expressed as clauses
    - E.g. at least one of the literals in each clause must be true
- SAT Advantages
  - DLL algorithm is highly polished, many well studied heuristics
  - Very fast BCP
  - Learning and non-chronological backtracking
- Can we remove some of the limitations while still retain the powerful SAT solving techniques?
  - Pseudo Boolean Constraint, Fadi Alou etc. PBS
  - Multi Value SAT, Cong Liu etc. CAMA
  - Quantified Boolean Solver

Pseudo Boolean (PB) Constraints Solver

- PB Constraints Definition

$$c_1 x_1 + \cdots + c_n x_n \sim g$$

$$c_i, g \in Z \qquad \sim \in \{=, \leq, \geq\} \qquad x_i \in Literals$$

- Examples:

$3\ x_1 + x_2 + 5\ x_3 = 2$

$4\ x_1 - 5\ x_3' >= 3$

Pseudo Boolean (PB) Constraints Solver

- Clauses can be generalized as a PB constraint:

  $(x \vee y) \longrightarrow (x + y \geq 1)$

- Convert arbitrary PB constraints to normal form:

  $c_1 x_1 + \cdots + c_n x_n \leq g$

  e.g. $-3x_1 + 2x_2 \geq -1$

  $-(-3x_1 + 2x_2) \leq -(-1)$

  $3x_1 - 2x_2 \leq 1$

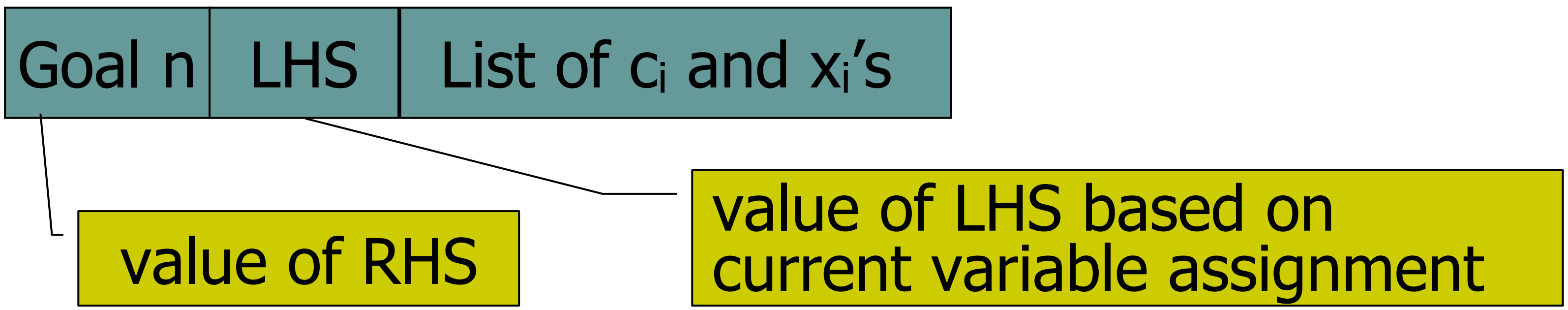
  $3x_1 - 2(1 - \bar{x}_2) \leq 1$

  $3x_1 + 2\bar{x}_2 \leq 3$

> • Positive coefficients
> • Constraint type: ≤
> ⇒ Faster manipulation

PB-SAT Algorithms

- Struct PBConstraint:



- For efficiency:
  - Sort the list of $c_i x_i$ in order of increasing c_i

PB-SAT Algorithms

- Assigning v_i to 1:
  For each literal x_i of v_i
  - If positive x_i, LHS += c_i
- Unassigning v_i from 1:
  For each literal x_i of v_i
  - If positive x_i, LHS -= c_i
- PB constraint state
  - UNS: LHS > goal

$5x_1+6x_2+3x_3 \leq 12$

LHS = 0

$5x_1+6x_2+3x_3 \leq 12$

LHS = 5

$5x_1+6x_2+3x_3 \leq 12$

LHS = 8

LHS < goal

SATISFIABLE

PB-SAT Algorithms

- Identifying implications
  - if c_i > goal – LHS, $x_i = 0$
  - Implied by literals in PB assigned to 1

$5x_1+6x_2+3x_3 \leq 12$

LHS = 0
goal - LHS = 12

$5x_1+6x_2+3x_3 \leq 12$

LHS = 8
goal - LHS = 4
Imply $x_2=0$

PB-SAT Algorithms

- Identifying conflicts
  - if LHS > goal
  - Conflicting assignment: consists of the subset of true literals whose sum of coefficients exceeds the goal.

$5x_1+6x_2+3x_3 \leq 7$

LHS = 0

$5x_1+6x_2+3x_3 \leq 7$

LHS = 14

LHS > goal

conflicting assignment = $\{x_1, x_2\}$

SAT Solving on Multi-Value Domain

- Let x_i denote a multi-valued variable with domain $P_i = \{0,1,\ldots,|P_i|-1\}$.
- x_i is assigned to a non-empty value set v_i, if x_i can take any value from $v_i \subseteq P_i$ but no value from $P_i \setminus v_i$
  - if $| v_i | = 1$, completely assigned, e.g. $x := \{2\}$
  - otherwise incompletely assigned, e.g. $x := \{0,2\}$
- A multi-valued literal $x_i^{s_i}$ is the Boolean function defined by

$$x_i^{s_i} \equiv (x_i = \gamma_1) + \cdots + (x_i = \gamma_k)$$

where $\gamma_i \in s_i \subseteq P_i$; s_i is the literal value set

Multi-Value SAT

- A multi-valued clause is a logical disjunction of one or more MV literals.
- A formula in multi-valued conjunctive normal form (MV-CNF) is the logical conjunction of a set of multi-valued clauses.
- A MV SAT problem given in MV-CNF is
  - Satisfiable if there exists a set of complete assignments to all variables such that the formula evaluates to true.
  - It is unsatisfiable if no such assignment exists.

Resolution

- Recall: Binary clause resolution:

$$\frac{(\sum l_i + x) \quad (\sum l_i' + \bar{x})}{(\sum l_i + \sum l_i')}$$

$$\frac{\begin{array}{c}(x_1' + x_3) \\ (x_2 + x_3' + x_4)\end{array}}{(x_1' + x_2 + x_4)}$$

  - variable x is eliminated
- MV resolution provides generalized case

$$\frac{(\sum x_i^{s_i} + x^s) \quad (\sum x'^{s'_j}_i{}' + x^{s'})}{(\sum x_i^{s_i} + \sum x'^{s'_j}_i{}' + x^{s \cap s'})}$$

$$\frac{\begin{array}{c}(x_1^{\{0,2\}} + x_3^{\{2,3\}}) \\ (x_1^{\{3\}} + x_2^{\{1,2\}} + x_3^{\{0,2\}})\end{array}}{(x_2^{\{1,2\}} + x_3^{\{0,2,3\}})}$$

  - take intersection of literal value sets of the resolving variable x

Decision

- Binary case: either 0 or 1 is assigned
- MV Case: ($2^{|P|}$-2) possible assignments initially
  - E.g. P={0,1,2}: {0}, {1}, {2}, {0,1}, {0,2}, {1,2}
- "Large decision" scheme:
  - Pick one value from value set of selected variable
  - Max depth of decision stack = # variables n
  - Learning by contra-positive only forbids one value
- "Small decision" scheme:
  - Exclude one value
  - Max depth of decision stack = $\sum_{i=1}^{n} |P_i|$

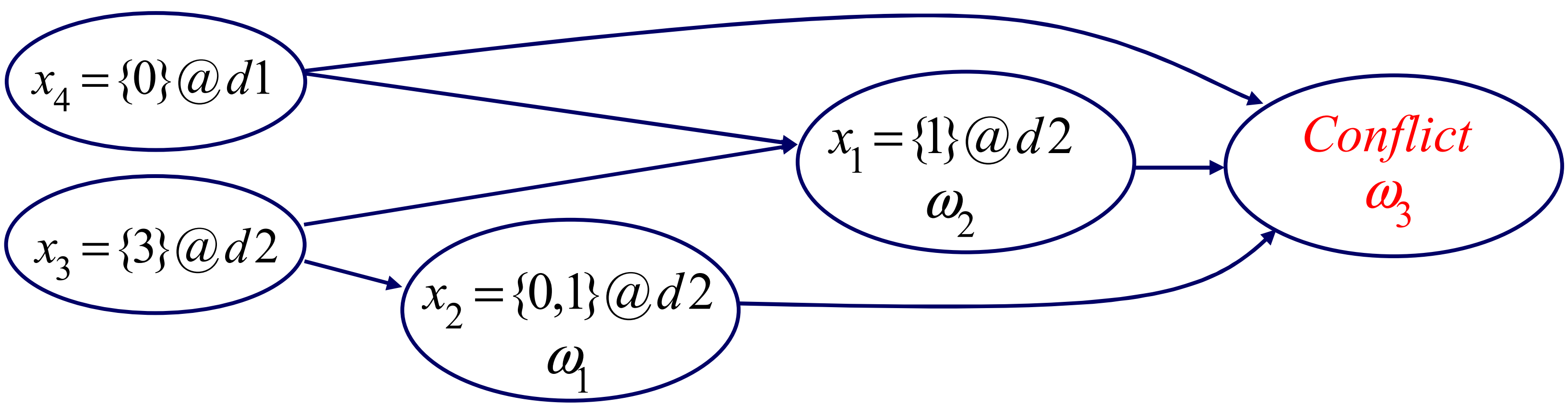
Boolean Constraint Propagation

$$\omega_1 = (x_3^{\{1\}} + x_2^{\{0,1\}})$$

$$\omega_2 = (x_4^{\{1,3\}} + x_3^{\{0\}} + x_1^{\{1\}})$$

$$\omega_3 = (x_2^{\{2\}} + x_1^{\{2,3\}} + x_4^{\{3\}})$$

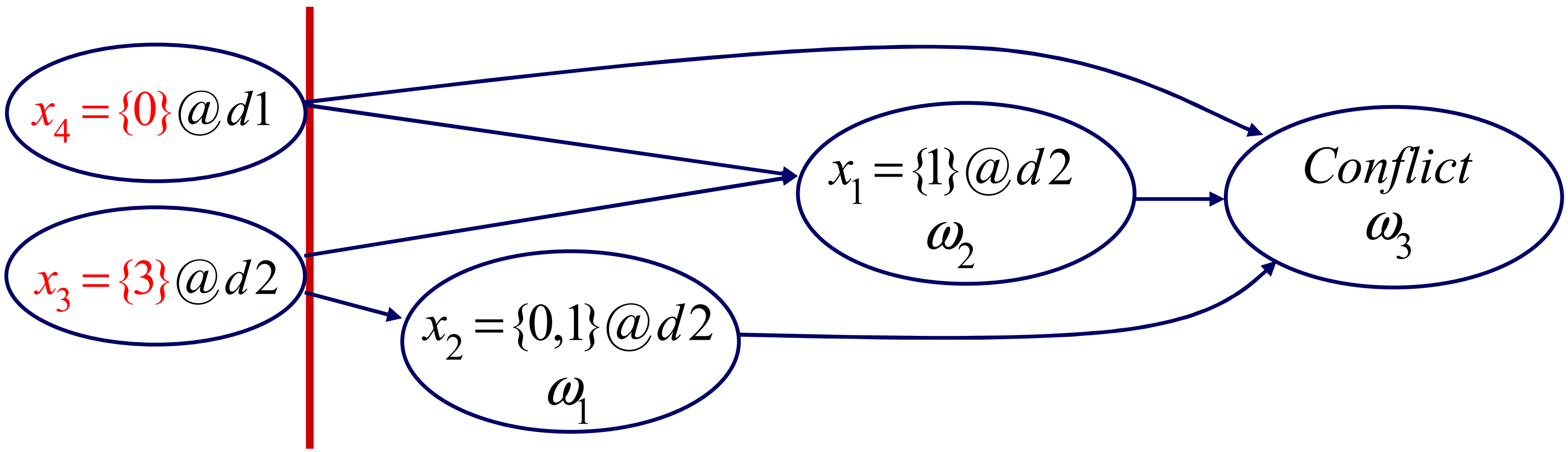
$$P_1 = P_2 = P_3 = P_4 = \{0, 1, 2, 3\}$$



Implication Graph (IG)

Conflict Analysis by Cut

- Cut at x3 and x4 as first UIP

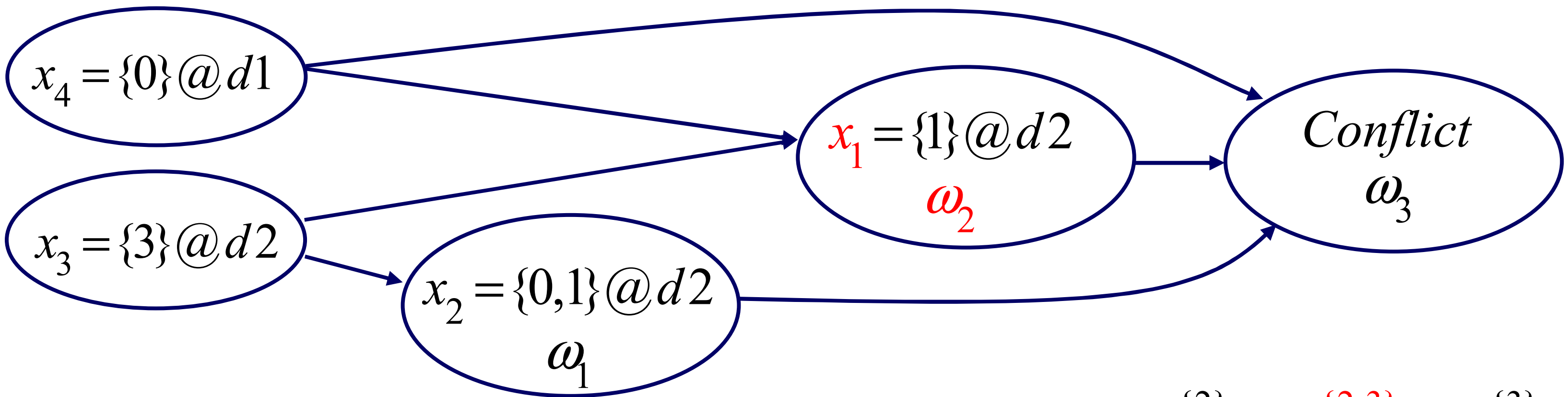


$$x_4 = \{0\} \wedge x_3 = \{3\} \Rightarrow Conflict$$

- By contra-positive we learn

$$\omega_{cut} = (x_4^{\{1,2,3\}} + x_3^{\{0,1,2\}})$$

Conflict Analysis by Resolution



$$\omega_1 = (x_3^{\{1\}} + x_2^{\{0,1\}})$$

$$\omega_2 = (x_4^{\{1,3\}} + x_3^{\{0\}} + x_1^{\{1\}})$$

$$\omega_3 = (x_2^{\{2\}} + x_1^{\{2,3\}} + x_4^{\{3\}})$$

$$\omega_{akk} = \omega_3 = (x_2^{\{2\}} + x_1^{\{2,3\}} + x_4^{\{3\}})$$

$$\omega_2 = (x_4^{\{1,3\}} + x_3^{\{0\}} + x_1^{\{1\}})$$

$$\omega'_{akk} = (x_2^{\{2\}} + x_4^{\{1,3\}} + x_3^{\{0\}})$$

$$\omega_1 = (x_3^{\{1\}} + x_2^{\{0,1\}})$$

$$\omega_{learn} = \omega'_{akk} = (x_4^{\{1,3\}} + x_3^{\{0,1\}})$$

MV-Conflict Analysis

- The learned clause is strictly "stronger" than the cut clause

$$\omega_{cut} = (x_4^{\{1,2,3\}} + x_3^{\{0,1,2\}})$$

$$\omega_{learn} = (x_4^{\{1,3\}} + x_3^{\{0,1\}})$$

Never visited before

- Cut clause forbids:

  $x_4 = \{0\} \wedge x_3 = \{3\}$

- Learned clause forbids:

  $x_4 = \{0\} \wedge x_3 = \{3\}, \quad x_4 = \{0\} \wedge x_3 = \{2\},$

  $x_4 = \{2\} \wedge x_3 = \{2\}, \quad x_4 = \{2\} \wedge x_3 = \{3\}$

- As if decisions were:

  $x_4 = \{0, 2\} @ d1; \; x_3 = \{2, 3\} @ d2$

Bigger search space

Pseudo Boolean Constraints and Multi-Value Constraints

They REALLY look like Boolean Satisfiability Solvers!!!

Lintao Zhang

Microsoft Research

QBF: Quantified Boolean Formula

- Quantified Boolean Formula

  $Q_1 x_1 \ldots\ldots Q_n x_n \ \varphi$

- Example:

  $\forall x \exists y (x+y')(x'+y)$

  $\exists de \forall xyz \exists abc \ f(a,b,c,d,e,x,y,z)$

- QBF Problem:

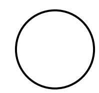
  Is F satisfiable?

Why Bother

- P-Space Complete, theoretically harder than NP-Complete problems such as SAT.
- Has practical Applications:
  - AI Planning
  - Sequential Circuit Verification
- Similarities with SAT
  - Leverage SAT techniques

Basic DLL Flow for QBF

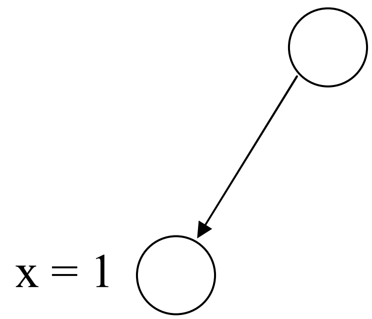
$$\exists x \forall y \ (x + y)(x' + y')$$

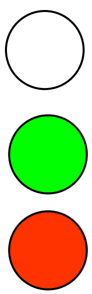


- Unknown
- True (1)
- False(0)

Basic DLL Flow for QBF

$$\exists x \forall y\ (x + y)(x' + y')$$





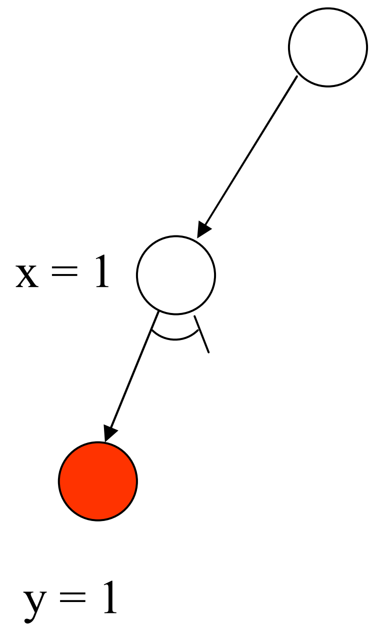
Unknown

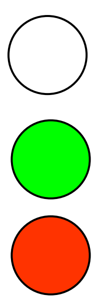
True (1)

False(0)

Basic DLL Flow for QBF

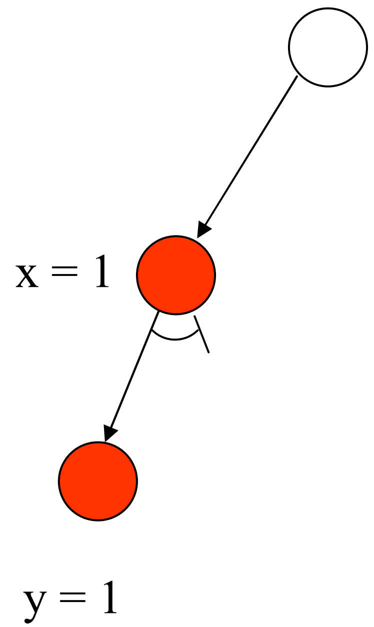
$\exists x \forall y \, (x + y)(x' + y')$

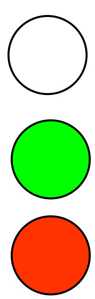




Basic DLL Flow for QBF

$\exists x \forall y\ (x + y)(x' + y')$



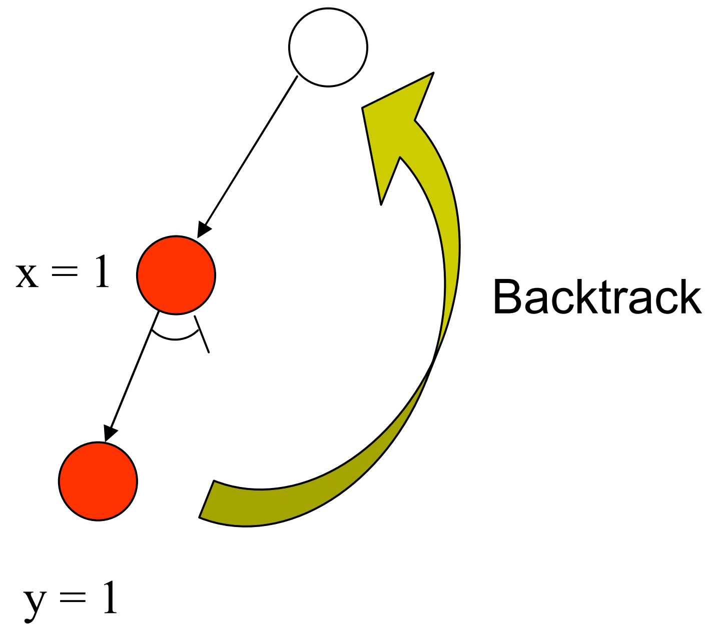


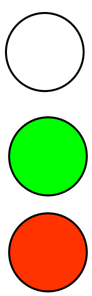
Lintao Zhang

Microsoft Research

Basic DLL Flow for QBF

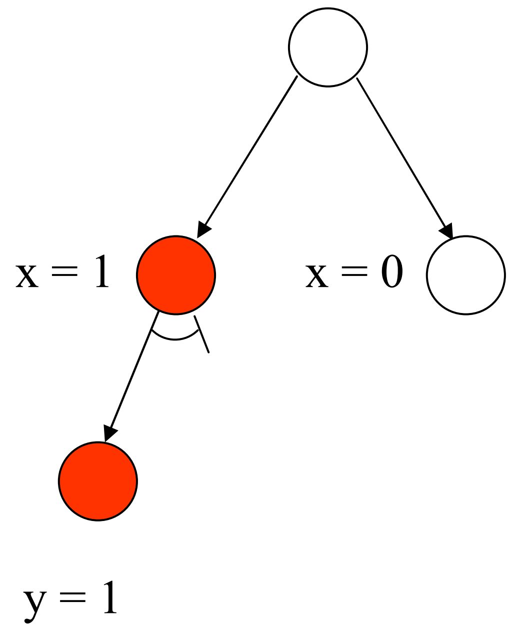
$\exists x \forall y\ (x + y)(x' + y')$

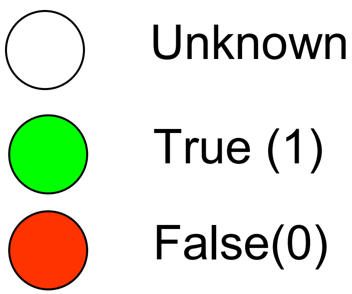




Basic DLL Flow for QBF

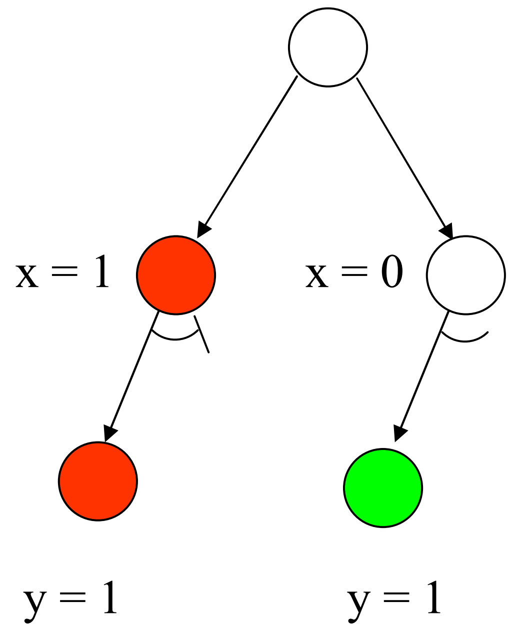
$$\exists x \forall y\ (x + y)(x' + y')$$

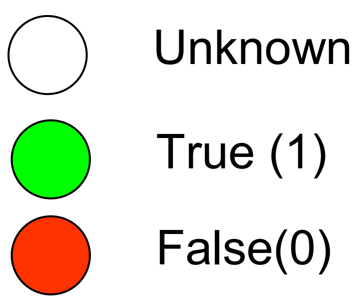




Basic DLL Flow for QBF

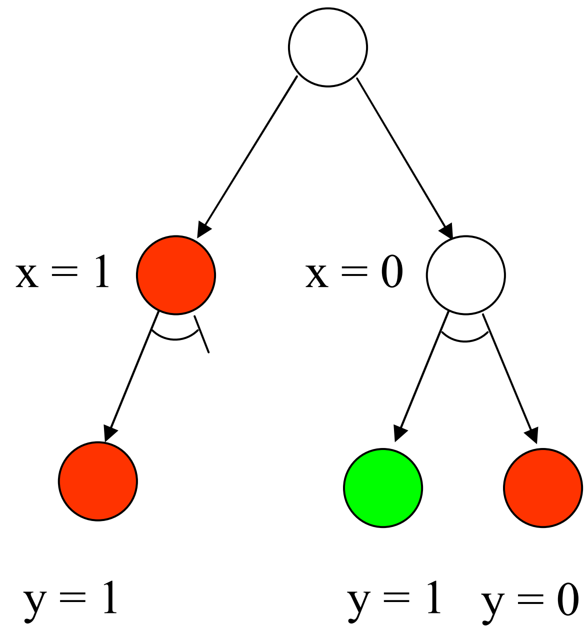
$\exists x \forall y\ (x + y)(x' + y')$

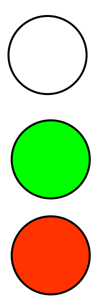




Basic DLL Flow for QBF

$\exists x \forall y\ (x + y)(x' + y')$



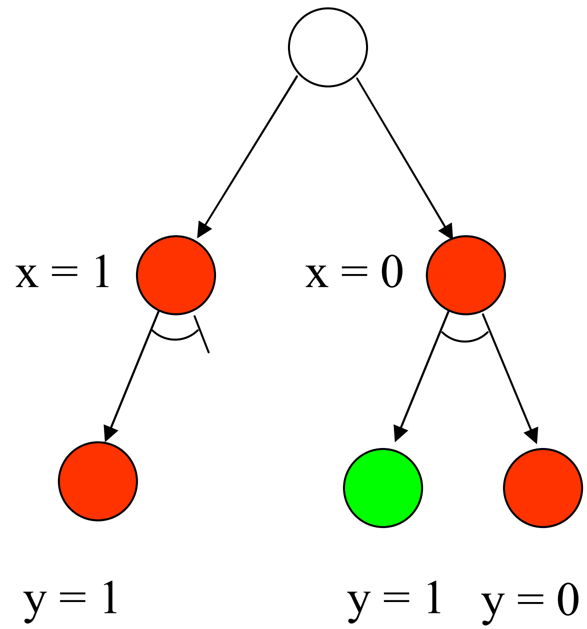


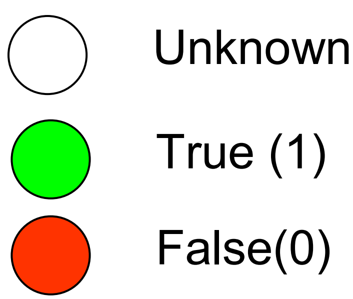
Lintao Zhang

Microsoft Research

Basic DLL Flow for QBF

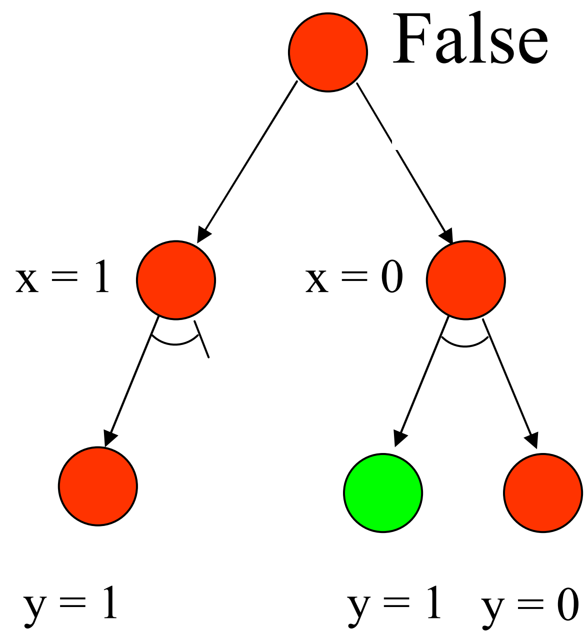
$$\exists x \forall y\ (x + y)(x' + y')$$

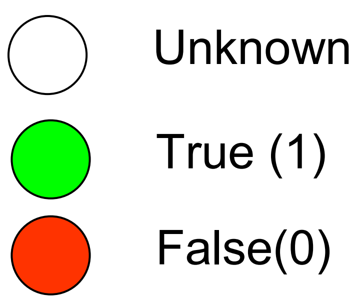




Basic DLL Flow for QBF

$\exists x \forall y\ (x + y)(x' + y')$





Basic DLL Flow for QBF

$$\forall x \exists y \, (x + y)(x' + y')$$

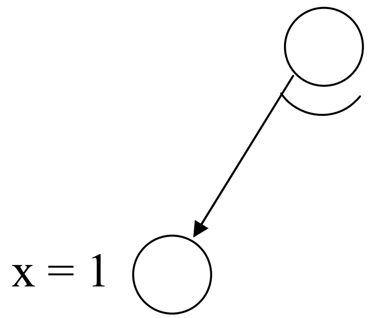
Unknown

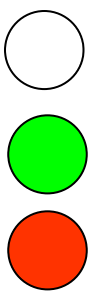
True (1)

False(0)

Basic DLL Flow for QBF

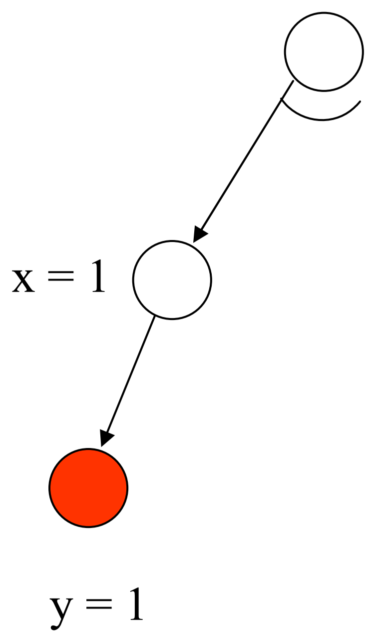
$\forall x \exists y\ (x + y)(x' + y')$

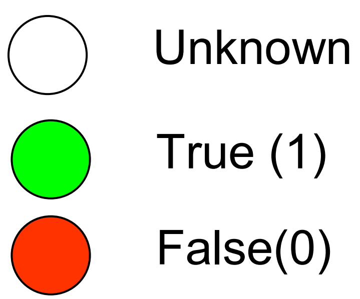




Basic DLL Flow for QBF

$\forall x \exists y\ (x + y)(x' + y')$



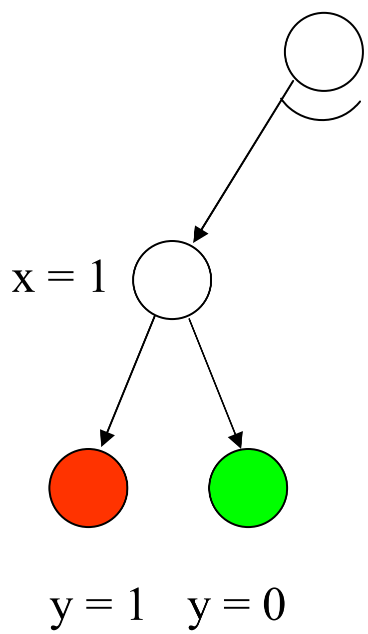


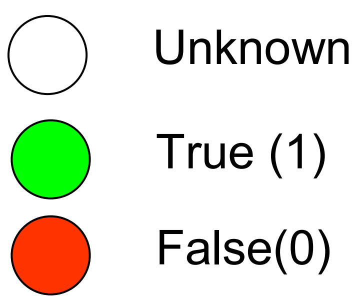
Lintao Zhang

Microsoft Research

Basic DLL Flow for QBF

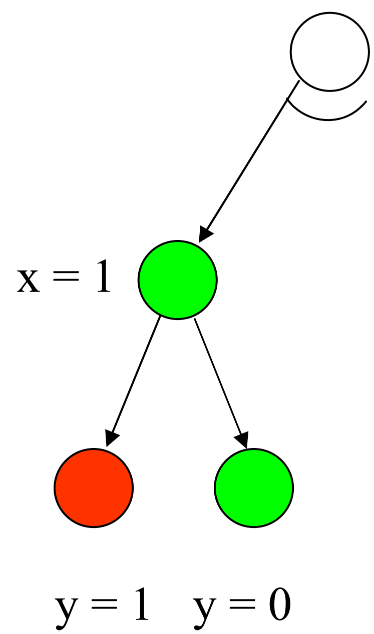
$\forall x \exists y\ (x + y)(x' + y')$

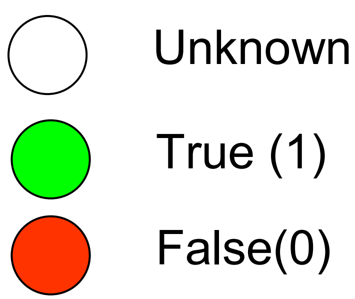




Basic DLL Flow for QBF

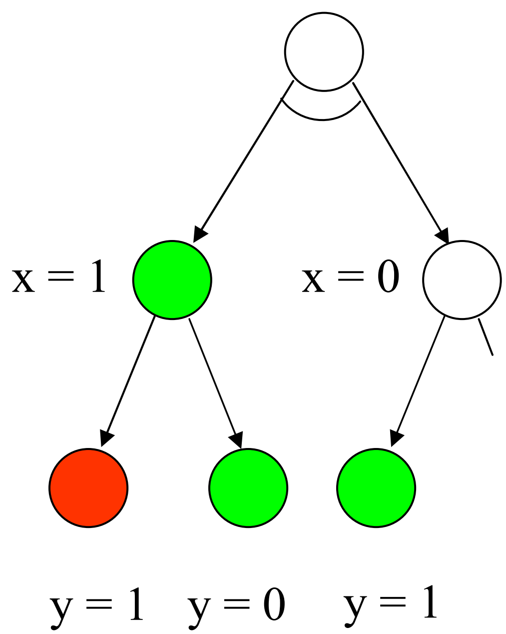
$\forall x \exists y\ (x + y)(x' + y')$

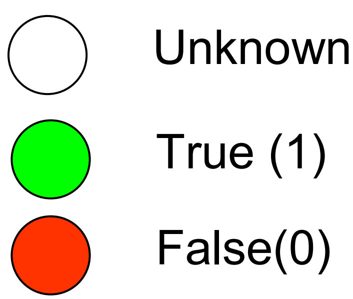




Basic DLL Flow for QBF

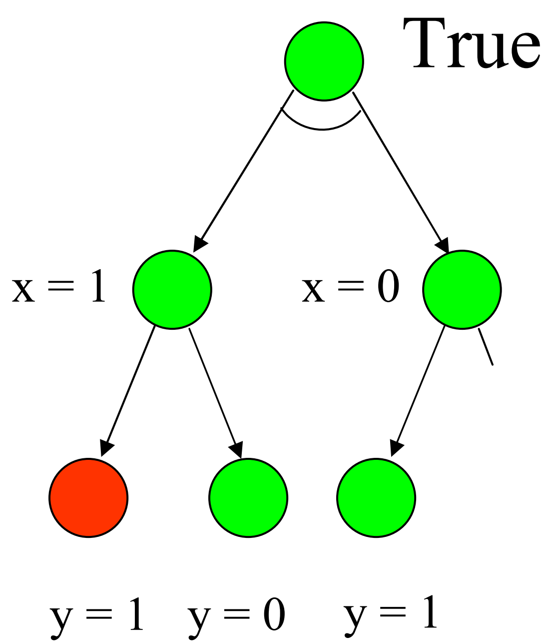
$\forall x \exists y\ (x + y)(x' + y')$

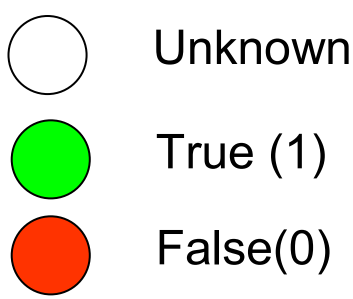




Basic DLL Flow for QBF

$\forall x \exists y \ (x + y)(x' + y')$





QBF: Conclusions

- QBF solving is much more difficult than SAT
  - No existing QBF solver is of much practical use
- It's possible to incorporate learning and non-chronological backtracking into a DLL based QBF solver
- Unlike SAT, where DLL seems to be the predominant solution, it's still unclear what is the most efficient approach to QBF
- Attracted a lot of interest recently, much more research effort is needed to make QBF solving practical
  - Chicken egg problem