

Boolean Satisfiability in Electronic Design Automation

João Marques-Silva
Informatics Department
Technical University of Lisbon
IST/VINESC, CEL

Karem A. Sakallah
EECS Department
University of Michigan

Context

- SAT is the quintessential NP-complete problem
- Theoretically well-studied
- Practical algorithms for large problem instances started emerging in the last five years
- Has many applications in EDA and other fields
- Can potentially have similar impact on EDA as BDDs
- EDA professionals should have good working knowledge of SAT formulations and algorithms

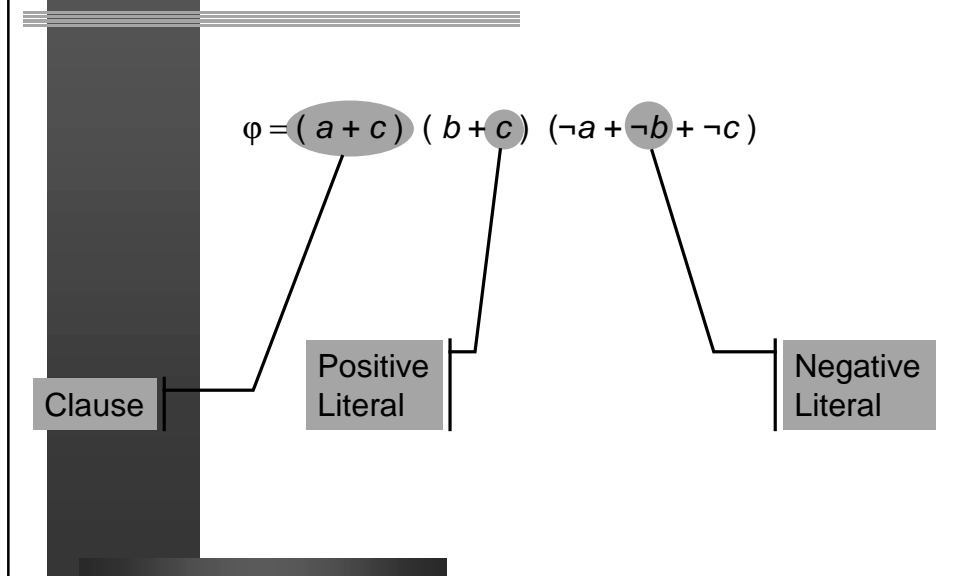
Outline

- Boolean Satisfiability (SAT)
- Basic Algorithms
- Representative EDA Applications
- Taxonomy of Modern SAT Algorithms
- Advanced Backtrack Search Techniques
- Experimental Evidence
- Conclusions

Boolean Satisfiability

- Given a suitable representation for a Boolean function $f(X)$:
 - Find an assignment X^* such that $f(X^*) = 1$
 - *Or prove that such an assignment does not exist* (i.e. $f(X) = 0$ for all possible assignments)
- In the “classical” SAT problem, $f(X)$ is represented in product-of-sums (POS) or conjunctive normal form (CNF)
- Many decision (yes/no) problems can be formulated either directly or indirectly in terms of Boolean Satisfiability

Conjunctive Normal Form (CNF)



Basics

- Implication

$$\begin{aligned}x \rightarrow y &= \neg x + y \\ &= \neg(\neg y) + (\neg x) \\ &= \neg y \rightarrow \neg x \text{ (contra positive)}\end{aligned}$$

- Assignments: $\{a = 0, b = 1\} = \neg a b$

- Partial (some variables still unassigned)
 - Complete (all variables assigned)
 - Conflicting (imply $\neg\varphi$)
- $$\begin{aligned}\varphi &= (a + c)(b + c)(\neg a + \neg b + \neg c) \\ \varphi &\rightarrow (a + c) \\ \neg(a + c) &\rightarrow \neg\varphi \\ \neg a \neg c &\rightarrow \neg\varphi\end{aligned}$$

Consensus

- General technique for deriving new clauses

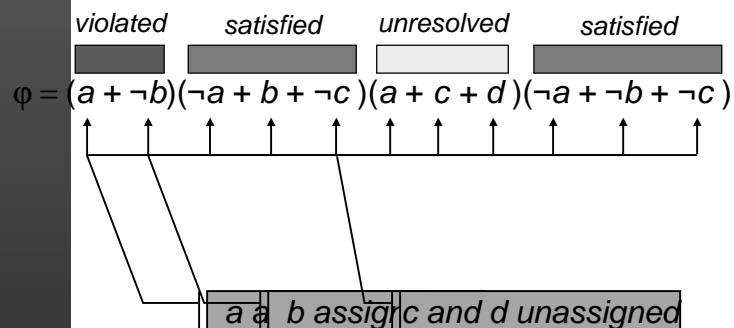
Example: $\omega_1 = (\neg a + b + c)$, $\omega_2 = (a + b + d)$

Consensus:

$$\text{con}(\omega_1, \omega_2, a) = (b + c + d)$$

- Complete procedure for satisfiability [Davis, JACM'60]
- Impractical for real-world problem instances
- Application of restricted forms has been successful!
 - E.g., always apply restricted consensus
 - $\text{con}((\neg a + \alpha), (a + \alpha), a) = (\alpha)$
 - α is a disjunction of literals

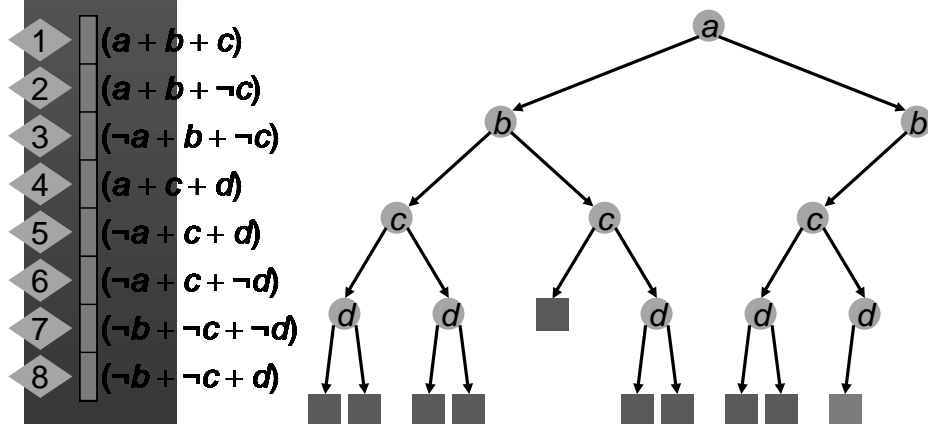
Literal & Clause Classification



Outline

- Boolean Satisfiability (SAT)
- Basic Algorithms
- Representative EDA Applications
- Taxonomy of Modern SAT Algorithms
- Advanced Backtrack Search Techniques
- Experimental Evidence
- Conclusions

Basic Backtracking Search



Unit Clause Rule - Implications

- An unresolved clause is *unit* if it has exactly one unassigned literal

$$\varphi = (a + c)(b + c)(\neg a + \neg b + \neg c)$$

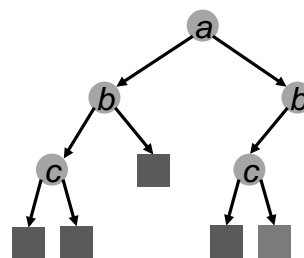
- A unit clause has exactly one option for being satisfied

$$a b \rightarrow \neg c$$

i.e. c must be set to 0.

Basic Search with Implications

- 1 $(a + b + c)$
- 2 $(a + b + \neg c)$
- 3 $(\neg a + b + \neg c)$
- 4 $(a + c + d)$
- 5 $(\neg a + c + d)$
- 6 $(\neg a + c + \neg d)$
- 7 $(\neg b + \neg c + \neg d)$
- 8 $(\neg b + \neg c + d)$



4

4

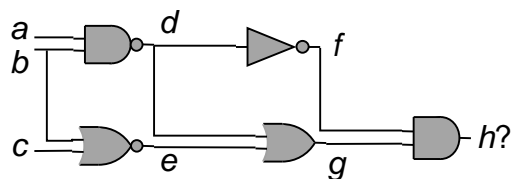
Pure Literal Rule

- A variable is *pure* if its literals are either all positive or all negative
- Satisfiability of a formula is unaffected by assigning pure variables the values that satisfy all the clauses containing them

$$\varphi = (a + c)(b + c)(b + \neg d)(\neg a + \neg b + d)$$

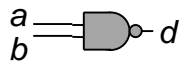
- Set c to 1; if φ becomes unsatisfiable, then it is also unsatisfiable when c is set to 0.

Circuit Satisfiability



$$\varphi = h [d = \neg(ab)] [e = \neg(b+c)] [f = \neg d] [g = d+e] [h = fg]$$

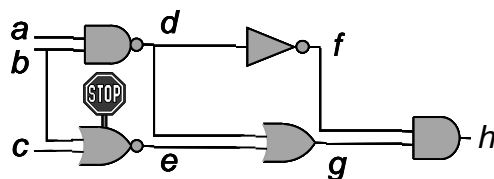
Gate CNF



$$\begin{aligned}
 \varphi_d &= [d = \neg(a b)] \\
 &= \neg[d \oplus \neg(a b)] \\
 &= \neg[\neg(a b) \neg d + a b d] \\
 &= \neg[\neg a \neg d + \neg b \neg d + a b d] \\
 &= (a + d)(b + d)(\neg a + \neg b + \neg d)
 \end{aligned}$$

$$\begin{aligned}
 \varphi_d &= [d = \neg(a b)][\neg d = a b] \\
 &= [d = \neg a + \neg b][\neg d = a b] \\
 &= (\neg a \rightarrow d)(\neg b \rightarrow d)(a b \rightarrow \neg d) \\
 &= (a + d)(b + d)(\neg a + \neg b + \neg d)
 \end{aligned}$$

Circuit Satisfiability



$$\begin{aligned}
 \varphi &= h [d = \neg(ab)] [e = \neg(b+c)] [f = \neg d] [g = d+e] [h = fg] \\
 &= h
 \end{aligned}$$

$$\begin{aligned}
 &(a + d)(b + d)(\neg a + \neg b + \neg d) \\
 &(\neg b + \neg e)(\neg c + \neg e)(b + c + e) \\
 &(\neg d + \neg f)(d + f) \\
 &(\neg d + g)(\neg e + g)(d + e + \neg g) \\
 &(f + \neg h)(g + \neg h)(\neg f + \neg g + h)
 \end{aligned}$$

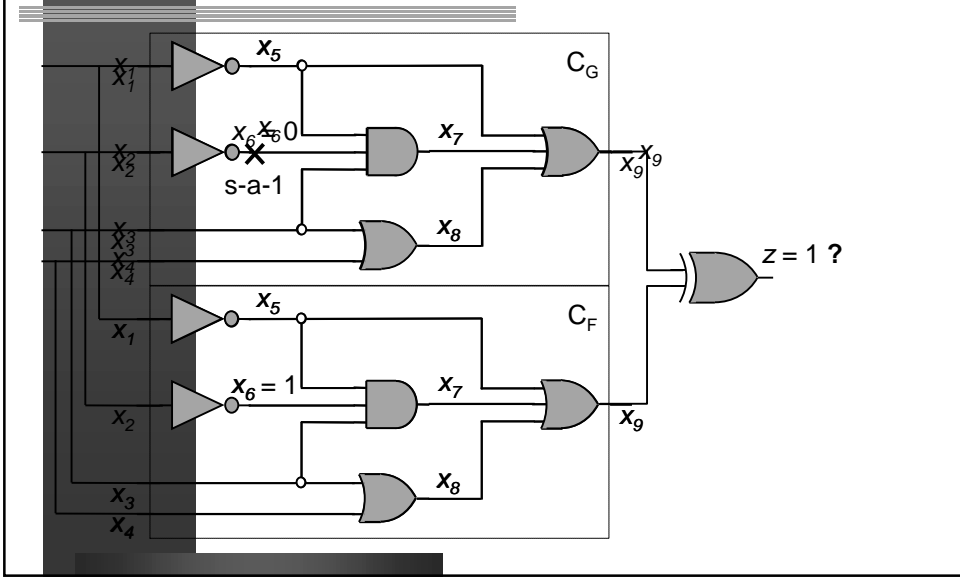
Outline

- Boolean Satisfiability (SAT)
- Basic Algorithms
- Representative EDA Applications
- Taxonomy of Modern SAT Algorithms
- Advanced Backtrack Search Techniques
- Experimental Evidence
- Conclusions

Applications of SAT in EDA

- Test Pattern Generation:
 - Stuck-at, Delay faults, etc.
 - Redundancy Removal
- Circuit Delay Computation
- Combinational Equivalence Checking
- Bounded Model Checking
- Superscalar processor verification
- FPGA routing
- Noise analysis

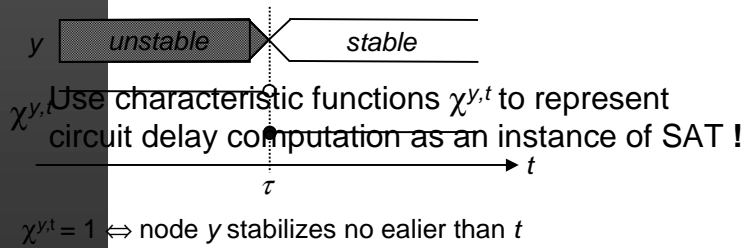
ATPG



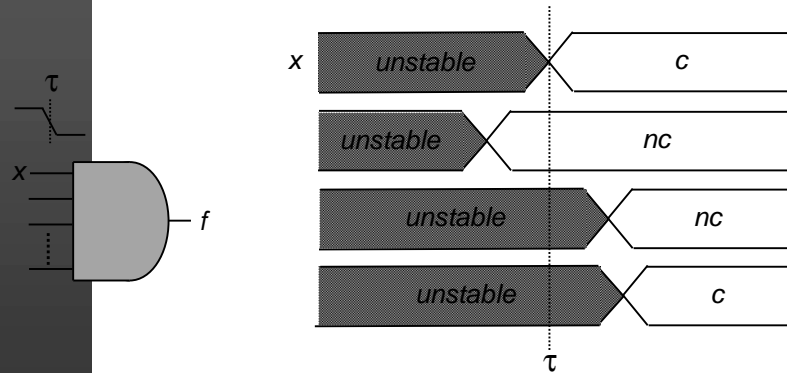
Delay Computation Using SAT

Can circuit delay be $\geq \Delta$?

Characteristic Function [McGeer, ICCAD'91]

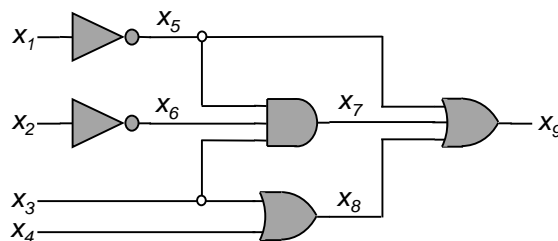


Delay Computation Using SAT



$$\chi^{f,t} = \sum_{g \in I(f)} \chi^{g,t-d(g,f)} \cdot \prod_{h \in I(f)} (\chi^{h,t-d(h,f)} + [h = nc(f)])$$

An Example

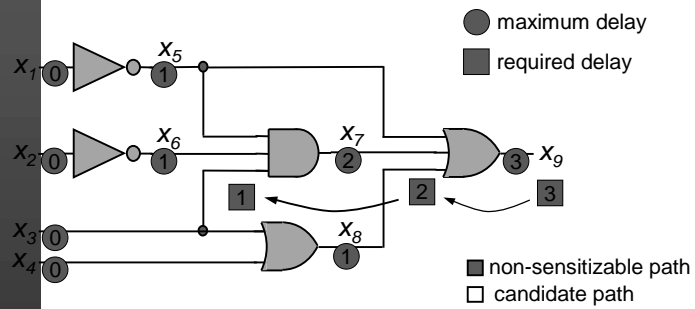


Q: Is the circuit delay greater than or equal to $\Delta = 3$?

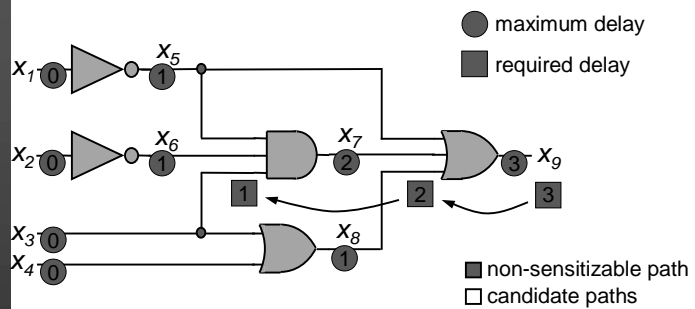
\equiv

Q: Is there any input vector $\mathbf{x}=(x_1, x_2, x_3, x_4)$, such that $\chi^{x_9-3}(\mathbf{x})=1$?

An Example



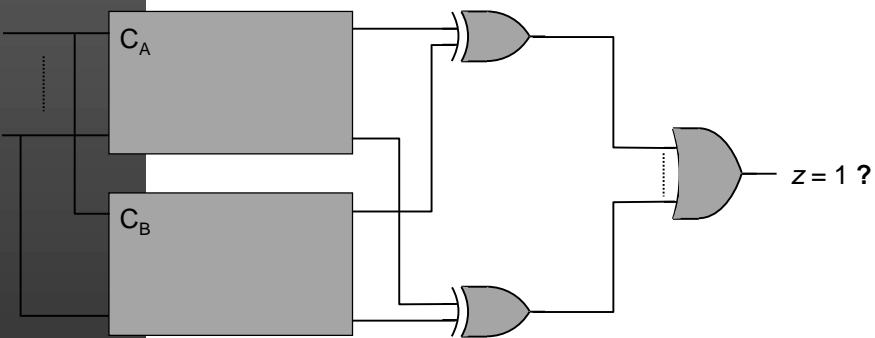
An Example



solving the SAT problem we obtain: $\chi_{x_9=3}(\mathbf{x})=0$

\therefore The critical delay Δ is smaller than or equal to 3 !

Equivalence Checking



z = 1 ?

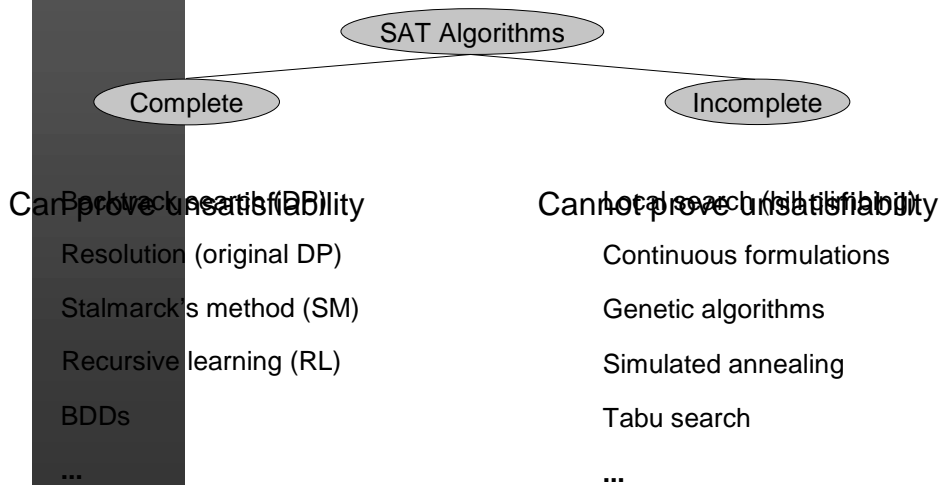
Bounded Model Checking

- Problem formulation,
 - System property P does not hold in one of the first k states following initial state I_0
- $$I_0 \wedge p(0,1) \wedge p(1,2) \wedge \dots \wedge p(k-1,k) \wedge (\neg P_0 \vee \neg P_1 \dots \vee \neg P_k)$$
- Create SMV-compatible model and create instance of SAT, in CNF format

Outline

- Boolean Satisfiability (SAT)
- Basic Algorithms
- Representative EDA Applications
- Taxonomy of Modern SAT Algorithms
- Advanced Backtrack Search Techniques
- Experimental Evidence
- Conclusions

A Taxonomy of SAT Algorithms



Resolution (original DP)

- Iteratively apply resolution (consensus) to eliminate one variable each time
 - i.e., resolution between all pairs of clauses containing x and $\neg x$
 - formula satisfiability is preserved
- Stop applying resolution when,
 - Either empty clause is derived \Rightarrow instance is unsatisfiable
 - Or only clauses satisfied or with pure literals are obtained \Rightarrow instance is satisfiable

$$\varphi = (a + c)(b + c)(d + c)(\neg a + \neg b + \neg c)$$

Eliminate variable c

$$\begin{aligned} \varphi_1 &= (a + \neg a + \neg b)(b + \neg a + \neg b)(d + \neg a + \neg b) \\ &= (d + \neg a + \neg b) \end{aligned}$$

Instance is SAT !

Stalmarck's Method (SM) in CNF

- Recursive application of the branch-merge rule to each variable with the goal of identifying common conclusions

$$\varphi = (a + b)(\neg a + c)(\neg b + d)(\neg c + d)$$

$$\text{Try } a = 0: \quad (a = 0) \Rightarrow (b = 1) \Rightarrow (d = 1) \quad C(a = 0) = \{a = 0, b = 1, d = 1\}$$

$$\text{Try } a = 1: \quad (a = 1) \Rightarrow (c = 1) \Rightarrow (d = 1) \quad C(a = 1) = \{a = 1, c = 1, d = 1\}$$

$$C(a = 0) \cap C(a = 1) = \{d = 1\} \quad \text{Any assignment to variable } a \text{ implies } d = 1. \\ \text{Hence, } d = 1 \text{ is a necessary assignment !}$$

An Alternative Explanation for SM

$$\varphi = (a + b)(\neg a + c)(\neg b + d)(\neg c + d)$$

resolution

$$(b + c)$$

resolution

$$(c + d)$$

resolution

$$(d)$$

Sequence of resolution operations for finding necessary assignments

Comment: SM provides a mechanism for identifying suitable resolution operations

Recursive Learning (RL) in CNF

- Recursive evaluation of clause satisfiability requirements for identifying common assignments

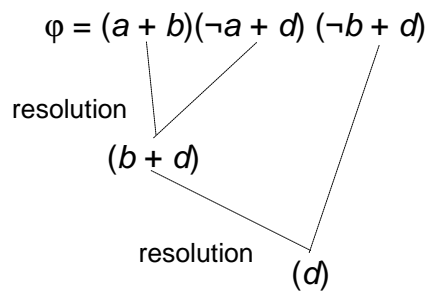
$$\varphi = (a + b)(\neg a + d)(\neg b + d)$$

Try $a = 1$: $(a = 1) \Rightarrow (d = 1)$ $C(a = 1) = \{a = 1, d = 1\}$

Try $b = 1$: $(b = 1) \Rightarrow (d = 1)$ $C(b = 1) = \{b = 1, d = 1\}$

$C(a = 1) \cap C(b = 1) = \{d = 1\}$ Every way of satisfying $(a + b)$ implies $d = 1$.
Hence, $d = 1$ is a necessary assignment !

An Alternative Explanation for RL



Sequence of resolution operations for finding necessary assignments

Comment: RL provides yet another mechanism for identifying suitable resolution operations

SM vs. RL

- Both complete procedures for SAT
- Stalmarck's method (in CNF):
 - hypothetical reasoning based on variables
- Recursive learning (in CNF):
 - hypothetical reasoning based on clauses
- Both can be viewed as the process of identifying selective resolution operations

- Both can be integrated into backtrack search algorithms

Local Search - GSAT

- Repeat M times:
 - Randomly pick complete assignment
 - Repeat K times (and while exist unsatisfied clauses):
 - Flip variable that will satisfy largest number of unsat clauses

$\varphi = (a + b)(\neg a + c)(\neg b + d)(\neg c + d)$ Pick random assignment

$\varphi = (a + b)(\neg a + c)(\neg b + d)(\neg c + d)$ Flip assignment on d

$\varphi = (a + b)(\neg a + c)(\neg b + d)(\neg c + d)$ Instance is satisfied !

Local Search - WalkSAT

- With probability p , flip variable in unsatisfied clause
- With probability $1 - p$, apply GSAT procedure

- Better than GSAT for hard, structured, satisfiable problem instances

Comparison

- Local search is incomplete
 - If instances are known to be SAT, local search can be competitive
- Resolution is in general impractical
- Stalmarck's Method (SM) and Recursive Learning (RL) are in general slow, though robust
 - SM and RL can derive too much unnecessary information
- For most EDA applications backtrack search (DP) is currently the most promising approach !
 - **Augmented with techniques for inferring new clauses/implicates (i.e. learning) !**

Outline

- Boolean Satisfiability (SAT)
- Basic Algorithms
- Representative EDA Applications
- Taxonomy of Modern SAT Algorithms
- Advanced Backtrack Search Techniques
- Experimental Evidence
- Conclusions

Techniques for Backtrack Search

- Conflict analysis
 - Clause/implicate recording
 - Non-chronological backtracking
- Incorporate and extend ideas from:
 - Resolution
 - Recursive learning
 - Stalmarck's method
- Formula simplification & Clause inference
- Randomization & Restarts

Clause Recording

- During backtrack search, for each conflict create clause that explains and prevents recurrence of same conflict

$$\varphi = (a + b)(\neg b + c + d)(\neg b + e)(\neg d + \neg e + f) \dots$$

Assume (decisions) $c = 0$ and $f = 0$

Assign $a = 0$ and imply assignments

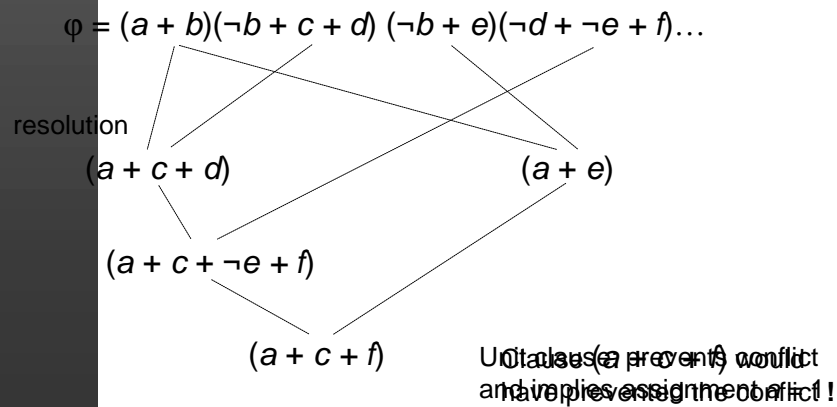
A conflict is reached: $(\neg d + \neg e + f)$ is unsat

$$(a = 0) \wedge (c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$$

$$(\varphi = 1) \Rightarrow (a = 1) \vee (c = 1) \vee (f = 1)$$

Clause Recording

- Clauses derived from conflicts can also be viewed as the result of applying selective resolution



More on Clause Recording

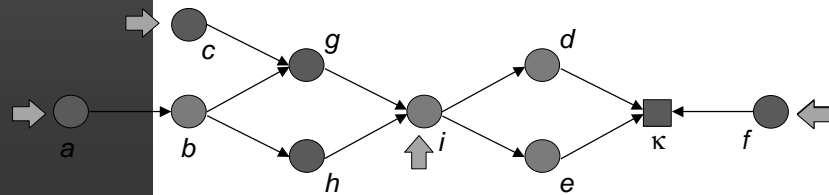
- Clause recording can be made polynomial
 - For each conflict 1 clause is recorded
 - Keep clauses of size $\leq K$
 - Larger clauses get deleted when (become) unresolved
 - Growth in the number of clauses is polynomial in K
- Relevance-based learning
 - Delete large unresolved clauses with $\geq M$ free literals

Conflict-Induced Assignments

- Exploit structure of conflicting implication sequences for identifying more necessary assignments

$$\phi = (a + b)(\neg b + c + \neg g)(\neg b + \neg h)(g + h + i)(\neg i + d)(\neg i + e)(\neg d + \neg e + f) \dots$$

Assume (decisions) $c = 0$, $f = 0$, and $a = 0$, and imply assignments



crossed clause $(a + c + \neg i)$ assigned $a = 0$ and $c = 1$

Ideas from other Approaches

- Resolution, Stalmarck's method and recursive learning can be incorporated into backtrack search (DP)
 - create additional clauses/implicates
 - anticipate and prevent conflicting conditions
 - identify necessary assignments
 - allow for non-chronological backtracking

Resolution within DP:

$$\begin{array}{ccc} (a + b + c) & & (\neg a + b + d) \\ \text{resolution} \swarrow & & \searrow \\ & (b + c + d) & \end{array}$$

Unit clause

Clause provides explanation for necessary assignment $b = 1$

Stalmarck's Method within DP

$$\varphi = (a + b + e)(a + c + f)(\neg b + d)(\neg c + d + g)$$

Implications:

$$(a = 0) \wedge (e = 0) \Rightarrow (b = 1) \Rightarrow (d = 1)$$

$$(a = 1) \wedge (f = 0) \Rightarrow (c = 1)$$

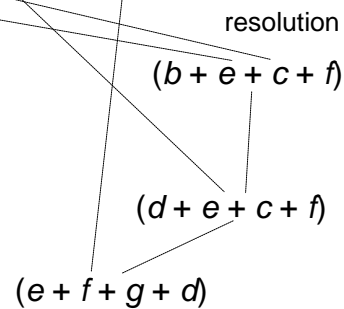
$$(c = 1) \wedge (g = 0) \Rightarrow (d = 1)$$

$$(e = 0) \wedge (f = 0) \wedge (g = 0) \Rightarrow (d = 1)$$

Clausal form:

[Redacted]

Clause provides explanation
for necessary assignment $d = 1$



Recursive Learning within DP

$$\varphi = (a + b + c)(\neg a + d + e)(\neg b + d + c)$$

Implications:

$$(a = 1) \wedge (e = 0) \Rightarrow (d = 1)$$

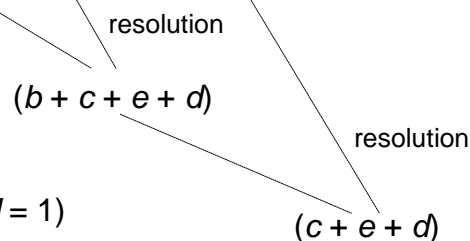
$$(b = 1) \wedge (c = 0) \Rightarrow (d = 1)$$

$$(c = 0) \wedge ((e = 0) \wedge (c = 0)) \Rightarrow (d = 1)$$

Clausal form:

[Redacted]

Clause provides explanation
for necessary assignment $d = 1$



Formula Simplification

- Eliminate clauses and variables

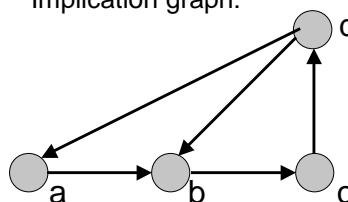
- If $(x + \neg y)$ and $(\neg x + y)$ exist, then x and y are equivalent, $(x \leftrightarrow y)$
 - eliminate y , and replace by x
 - remove satisfied clauses
- Utilize 2CNF sub-formula for identifying equivalent variables

$$(\neg a + b)(\neg b + c)(\neg c + d)(\neg d + b)(\neg d + a)$$

$$\equiv (a \rightarrow b)(b \rightarrow c)(c \rightarrow d)(d \rightarrow b)(d \rightarrow a)$$

a, b, c and d are pairwise equivalent
 \therefore replace all variables by a

Implication graph:



Support-Set Equivalence

- Existence of CNF sub-formulas such that $x = f(a,b)$
- If $x = f(a,b)$ and $y = f(a,b)$, then $x \leftrightarrow y$

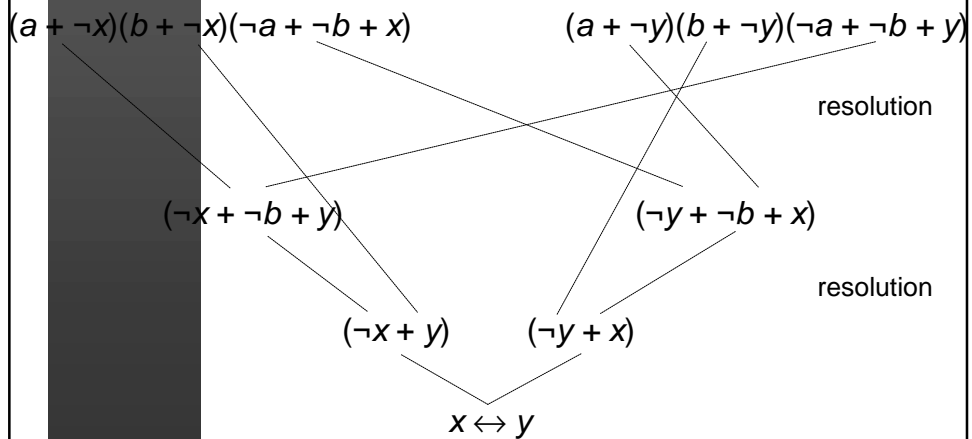
$$x = (a \wedge b) \quad \text{is represented as } (a + \neg x)(b + \neg x)(\neg a + \neg b + x)$$

$$y = (a \wedge b) \quad \text{is represented as } (a + \neg y)(b + \neg y)(\neg a + \neg b + y)$$

Can use resolution to obtain: $(x + \neg y)(y + \neg x)$

Hence, $x \leftrightarrow y$

2-Variable Equivalence



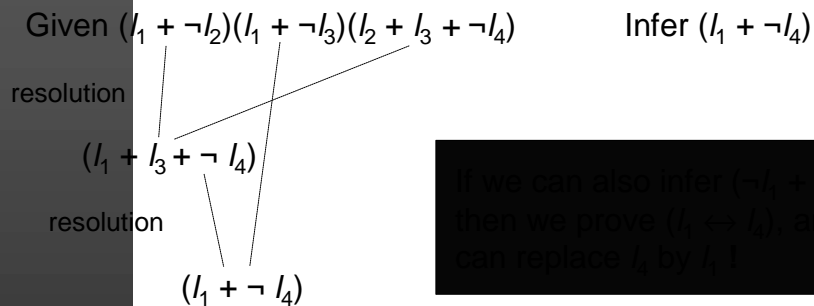
Clause Inference Conditions

- Support-set equivalence can be viewed as the derivation of two binary clauses:

$$(\neg x + y)(\neg y + x)$$

- Can use pattern matching techniques for inferring single binary/unit clauses
 - To establish 2-variable equivalence (pair of binary clauses)
 - To identify implication relations (single binary/unit clause)

Clause Inference Conditions



Type of Inference: 2 Binary / 1 Ternary (2B/1T) Clauses

Other types: 1B/1T, 1B/2T, 3B/1T, 2B/1T, 0B/4T

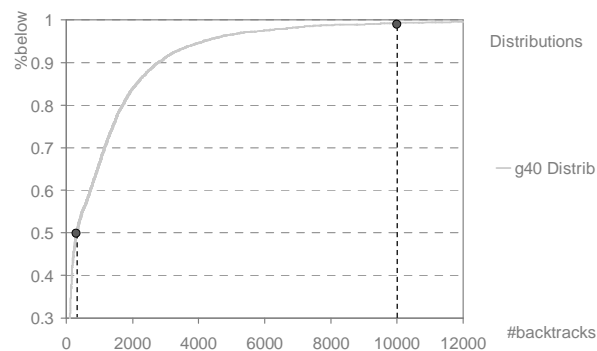
The Power of Resolution

- Most search pruning techniques can be explained as particular ways of applying selective resolution
 - Conflict-based clause recording
 - Non-chronological backtracking
 - Extending Stalmarck's method to backtrack search
 - Extending recursive learning to backtrack search
 - Clause inference conditions
- General resolution is computationally too expensive !
- Most techniques indirectly identify which resolution operations to apply !
 - To create new clauses/implicates
 - To identify necessary assignments

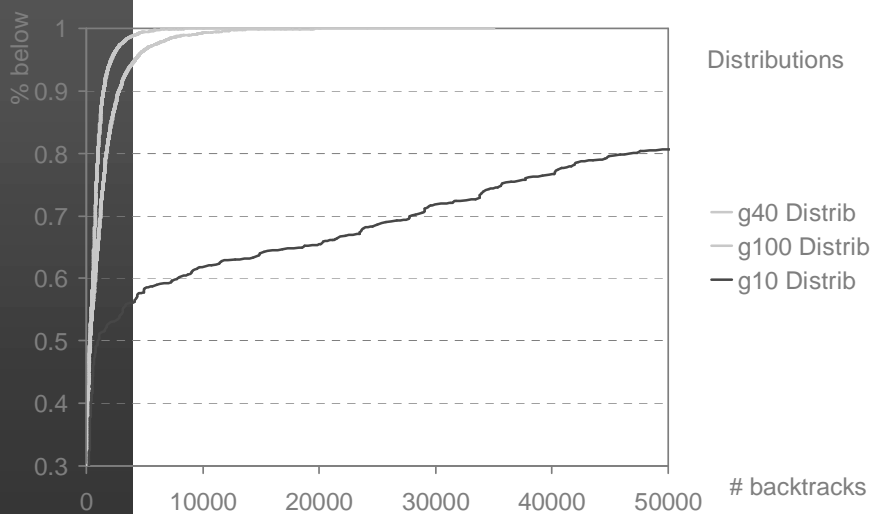
Randomization & Restarts

- Run times of backtrack search SAT solvers characterized by heavy-tail distributions
 - For a fixed problem instance, run times can exhibit large variations with different branching heuristics and/or branching randomization

Processor verification instance w/ branching randomization and 10000 runs

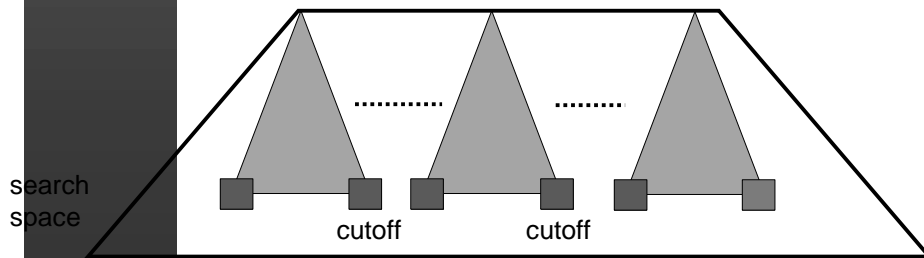


Heavy Tails & Learning



Randomization & Restarts

- Search strategy: Rapid Randomized Restarts
 - Randomize variable selection heuristic
 - Utilize a “small” backtrack cutoff value
 - Repeatedly restart the search each time backtrack cutoff reached
 - Use randomization to explore different paths in search tree



Randomization & Restarts

- Can make the search strategy complete
 - Increase backtrack cutoff value after each restart
- Can utilize learning
 - Useful for proving unsatisfiability
- Can utilize portfolios of algorithms and/or algorithm configurations
 - Either, run K algorithms (or algorithm configurations)
 - concurrently, in different processors, or
 - sequentially, in a single processor
 - Or, after each restart, pick an algorithm from a portfolio
 - Also useful for proving unsatisfiability

Outline

- Boolean Satisfiability (SAT)
- Basic Algorithms
- Representative EDA Applications
- Taxonomy of Modern SAT Algorithms
- Advanced Backtrack Search Techniques
- Experimental Evidence
- Conclusions

Empirical Evidence (in EDA)

- Illustrate scalability of modern SAT solvers
 - Ability to solve large problem instances
- Illustrate practical application of the techniques described for backtrack search
 - Clause recording and non-chronological backtracking
 - Recursive Learning / Stalmarck's Method
 - CNF formula simplification
 - Randomization and restarts
 - Portfolio of algorithm configurations
- Utilize modern backtrack search SAT algorithm, GRASP

Empirical Evidence (in EDA)

Can solve large problem instances

domain	instance	variables	clauses	CPU time
FPGA routing	bigkeyv1w6	25896	91400	11.9

Empirical Evidence (in EDA)

Non-chronological backtracking (NCB)
and clause recording (CR)
can be observed often and can be crucial

domain	instance	w/o NCB and CR		w/ NCB and w/o CR				w/ NCB and CR			
		B	T	B	NCB	LJ	T	B	NCB	LJ	T
testing	ssa2670-130	> 11250000	> 10000	7142	3538	20	60.3	100	42	15	0.65

Empirical Evidence (in EDA)

SM and RL can be useful

domain	instance	w/o RL				w/ RL			
		B	NCB	LJ	T	B	NCB	LJ	T
CEC	bench903	19024	2075	37	> 2,000	2494	812	42	117

Empirical Evidence (in EDA)

Formula simplification can be significant

domain	instance	before		after	
		variables	clauses	variables	clauses
BMC	barrel7	3523	13765	805	3467

Empirical Evidence (in EDA)

Randomization & Restarts can be effective

domain	instance	w/o restarts				w/ restarts				
		B	NCB	LJ	T	B	NCB	LJ	T	R
verification	2dlx_cc_bug54	42645	15156	35	2299.3	222	147	36	53.5	3

Restart search every 100 backtracks
Keep recorded clauses of size ≤ 10

Empirical Evidence (in EDA)

Portfolio of algorithm configurations can be essential

domain	instance	w/o portfolio				w/ portfolio			
		B	NCB	LJ	T	B	NCB	LJ	T
verification	dlx2_cc	2273327	688891	39	> 100000	100498	32066	70	2032

Configurations:
- Clauses recorded: sizes 15 to 30
- Clauses kept: sizes 10 to 20
- Branching heuristics: GRASP's
- Each configuration w/ equal probability
- 4 different configurations used
- Initial cutoff = 250; increments = 50

Conclusions

- Many recent SAT algorithms and (EDA) applications
- Hard Applications
 - Bounded Model Checking
 - Combinational Equivalence Checking
 - Superscalar processor verification
 - FPGA routing
- “Easy” Applications
 - Test Pattern Generation: Stuck-at, Delay faults, etc.
 - Redundancy Removal
 - Circuit Delay Computation
- Other Applications
 - Noise analysis, etc.

Conclusions

- Complete vs. Incomplete algorithms
 - Backtrack search (DP)
 - Resolution (original DP)
 - Stalmarck’s method
 - Recursive learning
 - Local search
- Techniques for backtrack search (infer implicates)
 - conflict-induced clause recording
 - non-chronological backtracking
 - resolution, SM and RL within backtrack search
 - formula simplification & clause inference conditions
 - randomization & restarts

Research Directions

- Algorithms:
 - Explore relation between different techniques
 - backtrack search; conflict analysis; recursive learning; branch-merge rule; randomization & restarts; clause inference; local search (?); BDDs (?)
 - Address specific solvers (circuits, incremental, etc.)
 - Develop visualization aids for helping to better understand problem hardness
- Applications:
 - Industry has applied SAT solvers to different applications
 - SAT research requires challenging and representative publicly available benchmark instances !

More Information on SAT in EDA

- <http://algos.inesc.pt/grasp>
- <http://algos.inesc.pt/sat>
- <http://algos.inesc.pt/~jpms> (jpms@inesc.pt)

- http://andante.eecs.umich.edu/grasp_public
- <http://nexus6.cs.ucla.edu/GSRC/bookshelf/Slots/SAT/GRASP>
- <http://eecs.umich.edu/~karem> (karem@umich.edu)

- SATLIB

References

- Resolution
 - Davis&Putnam, JACM'60
- Backtrack Search
 - Davis et. al, CACM'62
 - Non-chronological backtracking and clause recording
 - Marques-Silva&Sakallah, ICCAD'96; Bayardo&Schrag, AAI'97; Zhang, CADE'97
 - Relevance-based learning
 - Bayardo&Schrag, AAI'97
 - Conflict-induced necessary assignments
 - Marques-Silva&Sakallah, ICCAD'96

References (Cont'd)

- Backtrack Search (Cont'd)
 - Randomization and restarts
 - Gomes&Selman, AAI'98; Baptista&Marques-Silva, CP'2000
 - Formula simplification
 - Li, AAI'2000; Marques-Silva, CP'2000
- Stalmarck's Method
 - Stalmarck, Patent'89; Groote&Warners, CWI TechRep'1999
- Recursive Learning
 - Kunz&Pradhan, ITC'92; Marques-Silva&Glass, DATE'99
- Local Search
 - Selman&Kautz, IJCAI'93