

SAT: Propositional Satisfiability

22c:145 Artificial Intelligence
Russell & Norvig, Ch. 7.6

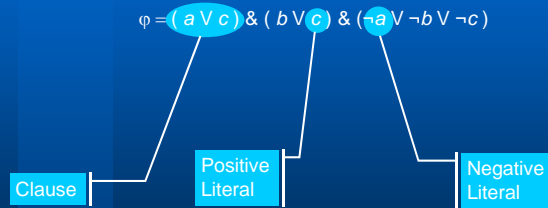
Validity vs. Satisfiability

- Validity:
 - A sentence is **valid** if it is true in every interpretation (every interpretation is a model).
 - A sentence s is a **valid** consequence of a set S of sentences if $(S \Rightarrow s)$ is valid.
 - Proof methods: True Table and Inference Rules
- Satisfiability:
 - A set of sentences is **satisfiable** if there exists an interpretation in which every sentence is true (it has at least one model).
 - Proof Methods: The Davis-Putnam-Logeman-Loveland procedure (DPLL).

SAT: Propositional Satisfiability

- An instance of SAT is defined as (X, S)
 - X : A set of 0-1 (propositional) variables
 - S : A set of sentences (formulas) on X
- Goal: Find an assignment $f: X \rightarrow \{0, 1\}$ so that every sentence becomes true.
- SAT is the first NP-complete problem.
 - **Good News**: Thousands of problems can be transformed into SAT
 - **Bad News**: There are no efficient algorithms for SAT

Conjunctive Normal Form (CNF)



Propositional Clauses

- Every propositional constraint can be converted into a set of equivalent **clauses**.
 - $S = \{C_1, C_2, \dots, C_m\} = C_1 \& C_2 \& \dots \& C_m$
- A **clause** is a disjunction of **literals**.
 - $C = (L_1 \vee L_2 \vee \dots \vee L_n)$
- A **literal** is either a **variable** or the negation of a variable.
 - $L = x$ or $L = \neg x$
- A set of clauses is also said to be in Conjunctive Normal Form (CNF).

Gate CNF



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

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

DIMACS Format

c This is an example of
c an SAT instance in DIMACS format

p cnf 3 5
1 2 0
1 3 0
-1 -2 0
-1 -3 0
-2 -3 0

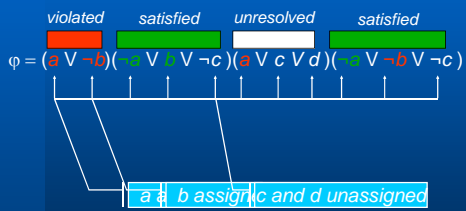
X1 V X2
X1 V X3
-X1 V -X2
-X1 V -X3
-X2 V -X3

DIMACS: Discrete Mathematics and Computer Science

More on Assignments

- Assignments: $\{a = 0, b = 1\} = \neg a \ \& \ b$
 - Partial (some variables still unassigned)
 - Complete (all variables assigned)
 - Conflicting (imply $\neg\phi$)
 - $\phi = (a \vee c) \ \& \ (b \vee c) \ \& \ (\neg a \vee \neg b \vee \neg c)$
 - $\phi \rightarrow (a \vee c)$
 - $\neg(a \vee c) \rightarrow \neg\phi$
 - $\neg a \ \& \ \neg c \rightarrow \neg\phi$

Literal & Clause Classification



Unit Clause Rule - Implications

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

$$\varphi = (\dots \vee c)(\dots \vee c)(\neg a \vee \neg b \vee \neg c)$$

- A unit clause has exactly one option for being satisfied

$$a \vee b \rightarrow \neg c$$

i.e. c must be set to 0.

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

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

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

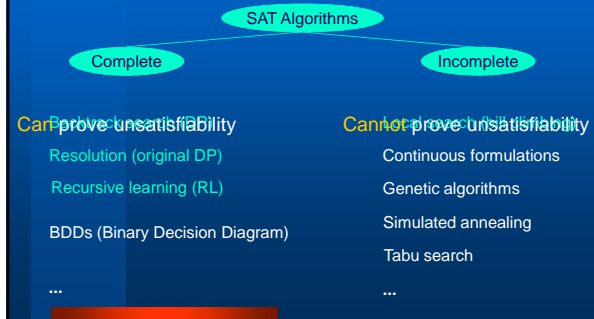
Resolution/Consensus

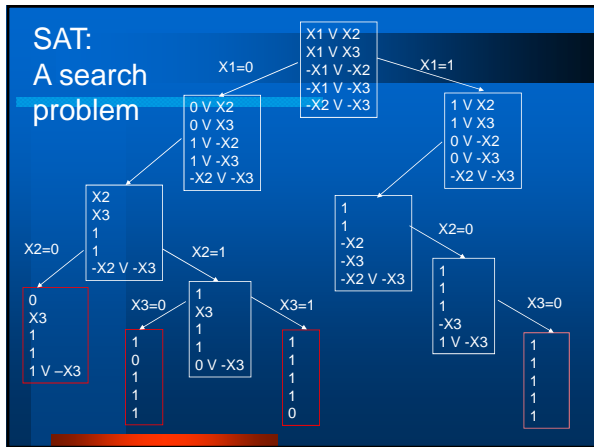
- General technique for deriving new clauses
Example: $\omega_1 = (\neg a \vee b \vee c)$, $\omega_2 = (a \vee b \vee d)$
Resolution:

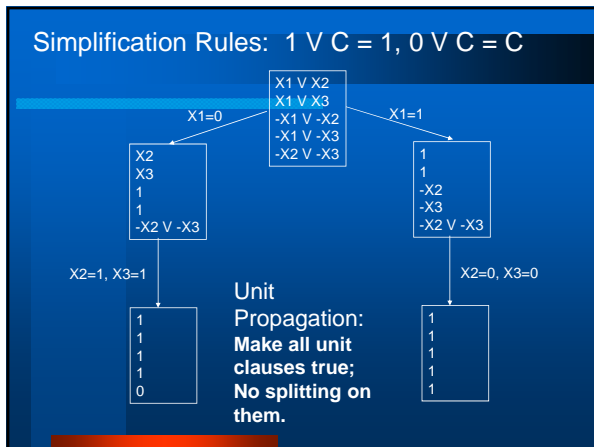
$$\text{res}(\omega_1, \omega_2, a) = (b \vee c \vee d)$$

- 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 resolution*
 - $\text{res}(\neg a \vee \alpha, a \vee \alpha, a) = \alpha$
 - α is a disjunction of literals

A Taxonomy of SAT Algorithms







The Davis-Putnam-Logemann-Loveland Algorithm (1960)

```
function Satisfiable ( clause set S ) return { 0, 1 }
repeat /* unit propagation */
  for each unit clause L in S do
    delete from S every clause containing L
    delete -L from each C in S in which -L occurs
  if S is empty then return 1
  else if a clause in S is empty then return 0
until no more new unit clauses changes

choose a literal L occurring in S /* splitting */
if Satisfiable( S U { L } ) then return 1
else if Satisfiable( S U { -L } ) then return 1
else return 0
```

DPLL uses Depth-First-Search

- DFS with Backtrack: Instead of maintaining a path of nodes, only one node is maintained. The node is modified when going down and everything is undone when going up.
- The branching factor is dictated by the splitting rule.

DPLL uses Backtrack Search

- Implicit enumeration
- Iterated unit-clause rule
 - Boolean constraint propagation
- Pure-literal rule
- Chronological backtracking in presence of conflicts
- The worst-time complexity is exponential in terms of the number of variables.

Implementing The DPLL Algorithm

- A destructive data structure is needed for clauses: Instead of copying clauses, modify them and then undo modification when backtracking.
- Efficient algorithms for unit-propagation.
- There are many choices for selecting a literal to split (heuristics are needed).

Satbox's Results on n-queen prob.

n	solutions	1 soln.	all soln.
8	92	0.00	0.01
10	724	0.00	0.04
20	>100,000	0.00	240
40	-	0.16	-
60	-	1.30	-
80	-	4.30	-
100	-	11.20	-

- Times are in seconds.
- There are 1646800 clauses for n=100

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**

$$\phi = (a \vee c)(b \vee c)(d \vee c)(\neg a \vee \neg b \vee \neg c) \quad \text{Eliminate variable } c$$

$$\phi_1 = (a \vee \neg a \vee \neg b)(b \vee \neg a \vee \neg b)(d \vee \neg a \vee \neg b)$$

$$= (d \vee \neg a \vee \neg b) \quad \text{Instance is SAT !}$$

Recursive Learning (RL) in CNF

- Recursive evaluation of **clause satisfiability** requirements for identifying **common assignments**

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

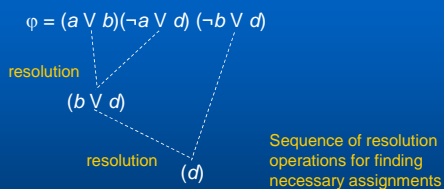
Try $a = 1$: $(a = 1) \Rightarrow (d = 1)$ $C(a = 1) = \{a = 1, d = 1\}$

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 $(a \vee b)$ implies $d = 1$. Hence, $d = 1$ is a **necessary** assignment !

Recursion can be of arbitrary depth

An Alternative Explanation for RL



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

Comparison

- Local search is **incomplete**
 - If instances are known to be SAT, local search can be competitive
- Resolution is in general **impractical**
- Recursive Learning (RL) are in general **slow**, though **robust**
 - RL can derive too much **unnecessary** information

Techniques for Backtrack Search

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

Clause Recording

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

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

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

Assign $a = 0$ and imply assignments

A conflict is reached: $(\neg d \vee \neg e \vee f)$ is unsatisfiable

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

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

\therefore create new clause: $(a \vee c \vee f)$

Clause Recording

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

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

resolution

$$(a \vee c \vee d)$$

$$(a \vee e)$$

$$(a \vee c \vee \neg e \vee f)$$

$$(a \vee c \vee f)$$

• Clause $(a \vee c \vee f)$ explains conflict and implies assignment $a = 1$!

Recursive Learning within DP

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

Implications:

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

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

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

$$(b \vee c \vee e \vee d)$$

$$(c \vee e \vee d)$$

Clausal form:

$$(c \vee e \vee d) \text{ Unit clause !}$$

Clause provides explanation for necessary assignment $d = 1$

Formula Simplification

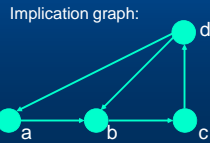
- Eliminate **clauses** and **variables**

- If $(x \vee \neg y)$ and $(\neg x \vee y)$ 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

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

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

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



2-Variable Equivalence

$$(a \vee \neg x)(b \vee \neg x)(\neg a \vee \neg b \vee x)$$

$$(a \vee \neg y)(b \vee \neg y)(\neg a \vee \neg b \vee y)$$

$$(\neg x \vee \neg b \vee y)$$

$$(\neg y \vee \neg b \vee x)$$

$$(\neg x \vee y)$$

$$(\neg y \vee x)$$

$$x \leftrightarrow y$$

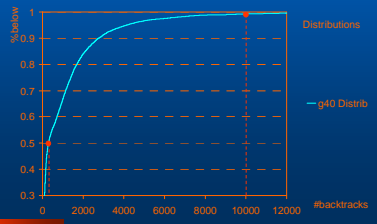
The Power of Resolution

- Most search pruning techniques can be explained as particular ways of applying selective resolution
 - Conflict-based clause recording
 - Non-chronological backtracking
 - Extending 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

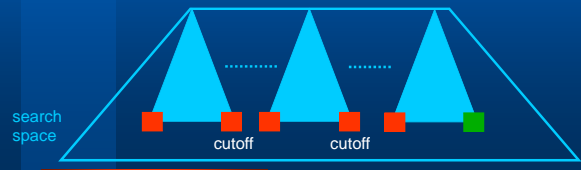
- Run times of backtrack search SAT solvers characterized by **heavy-tail distributions**
 - For a fixed problem instance, run times can exhibit **large variations** with different branching heuristics and/or branching randomization

Processor verification instance w/ branching randomization and 10000 runs



Randomization & Restarts

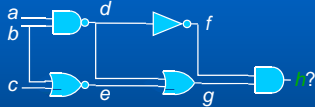
- Search strategy: **Rapid Randomized Restarts**
 - Randomize variable selection heuristic
 - Utilize a "small" backtrack **cutoff** value
 - Repeatedly restart the search each time backtrack cutoff reached
 - Use randomization to explore different paths in search tree



Randomization & Restarts

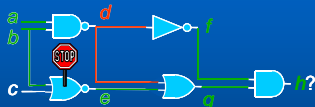
- Can make the search strategy **complete**
 - Increase backtrack cutoff value after each restart
- Can utilize **learning**
 - Useful for proving unsatisfiability
- Can utilize **portfolios** of algorithms and/or algorithm configurations
 - **Either**, run K algorithms (or algorithm configurations)
 - 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

Circuit Satisfiability



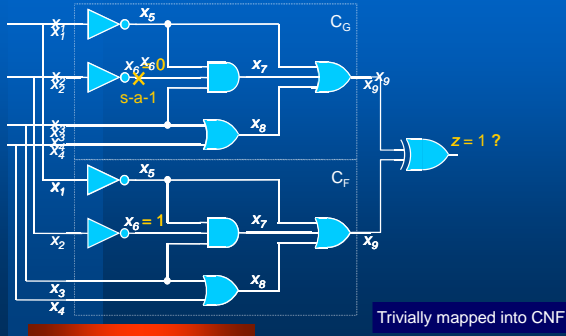
$$\varphi = [d \equiv \neg(ab)] [e \equiv \neg(bVc)] [f \equiv \neg d] [g \equiv dVe] [h \equiv fg] h$$

Circuit Satisfiability



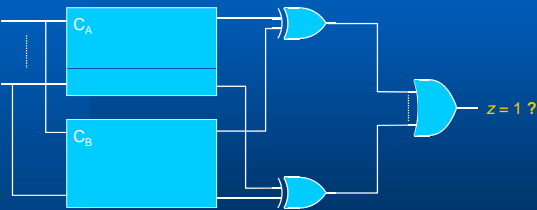
$$\begin{aligned} \varphi &= h [d \equiv \neg(ab)] [e \equiv \neg(bVc)] [f \equiv \neg d] [g \equiv dVe] [h \equiv fg] \\ &= h \\ &= (aVd)(bVd)(\neg aV\neg bV\neg d) \\ &= (\neg bV\neg e)(\neg cV\neg e)(bVcVe) \\ &= (\neg dV\neg f)(dVf) \\ &= (\neg dVg)(\neg eVg)(dVeV\neg g) \\ &= (fV\neg h)(gV\neg h)(\neg fV\neg gVh) \end{aligned}$$

ATPG (Automatic Test Pattern Generation)



Equivalence Checking

- Combinational Circuits:



If $z = 1$ is unsatisfiable, the two circuits are equivalent !

Bounded Model Checking

- Sequential Circuits:

- Problem formulation,
 - System property P does not hold in one of the first k states following initial state I_0
 - Transition function ρ

$$I_0 \wedge \rho(0,1) \wedge \rho(1,2) \wedge \dots \wedge \rho(k-1,k) \wedge (\neg P_0 \vee \neg P_1 \dots \vee \neg P_k)$$

- Represent formula as an instance of SAT in CNF format

Comparison

- CEC is **harder** than ATPG
 - In ATPG the two circuits are **known** to be similar
 - In CEC the two circuits can be fairly different
 - Hard instances of CEC are unsatisfiable
- BMC is **hard**
 - Large number of (apparently) unrelated variables
 - Identification of conflicts may require large number of decisions

SAT Problem Hardness in EDA

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



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
 - Formula simplification
 - Randomization and restarts
 - Portfolio of algorithm configurations
- Utilize modern backtrack search SAT algorithm, www.satlive.org

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)

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				R
		B	NCB	LJ	T	B	NCB	LJ	T	
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
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)
 - 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 !

A Few References

- EDA Applications
 - Marques-Silva&Sakallah, DAC'00
- Resolution
 - Davis&Putnam, JACM'60
- Backtrack Search
 - Davis et. al, CACM'62
 - Non-chronological backtracking and clause recording
 - Marques-Silva&Sakallah, ICCAD'96; Bayardo&Schrage, AAAI'97; Zhang, CADE'97
 - Relevance-based learning
 - Bayardo&Schrage, AAAI'97
 - Conflict-induced necessary assignments
 - Marques-Silva&Sakallah, ICCAD'96

A Few References (Cont'd)

- Backtrack Search (Cont'd)
 - Randomization and restarts
 - Gomes&Selman, AAAI'98; Baptista&Marques-Silva, CP'2000
 - Formula simplification
 - Li, AAAI'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
