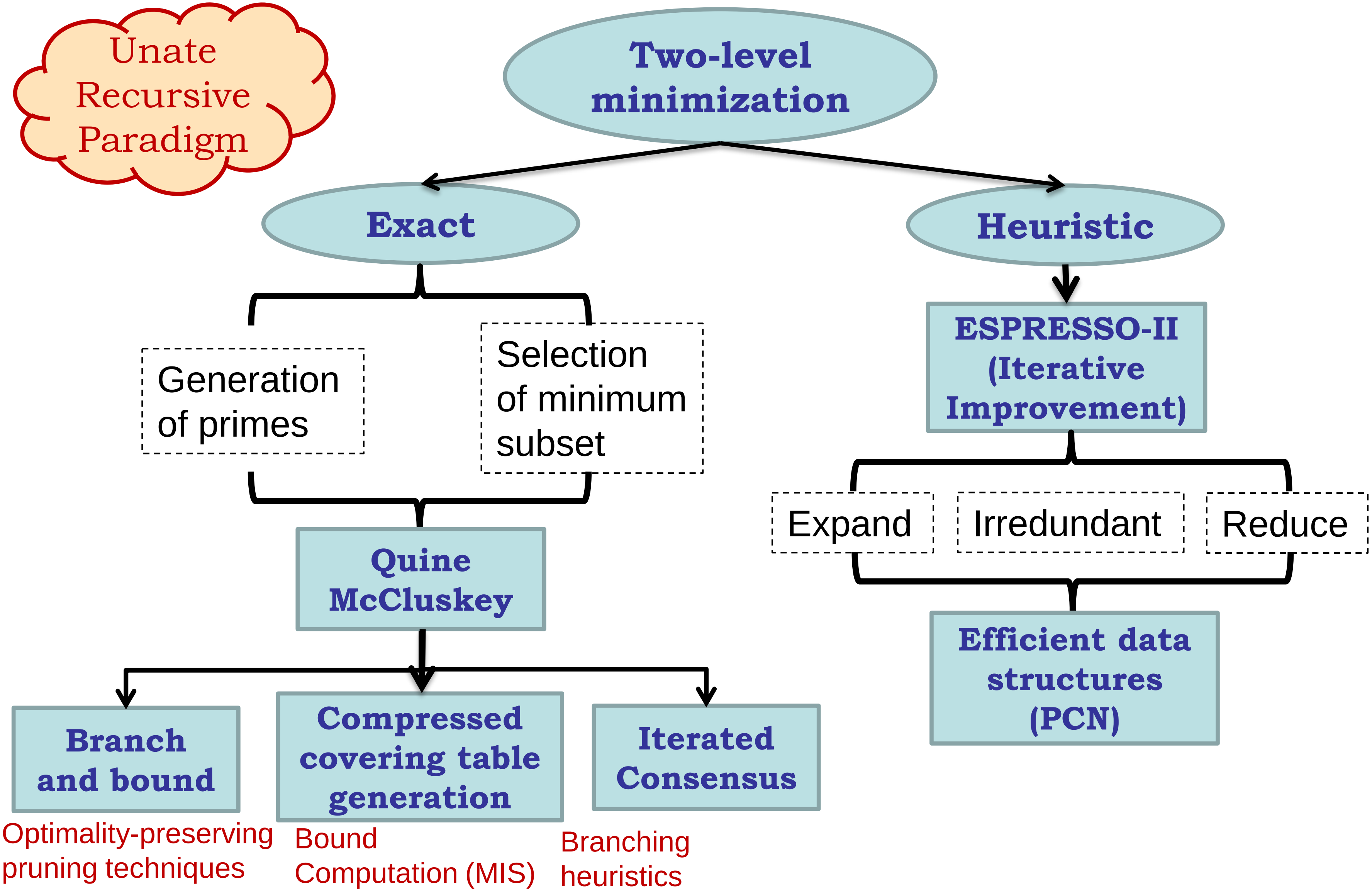
Two-level Minimization Techniques: Summary



Unate Recursive Paradigm

Two-level minimization

Exact

Heuristic

Generation of primes

Selection of minimum subset

Quine McCluskey

Branch and bound

Optimality-preserving pruning techniques

Compressed covering table generation

Bound Computation (MIS)

Iterated Consensus

Branching heuristics

ESPRESSO-II (Iterative Improvement)

Expand

Irredundant

Reduce

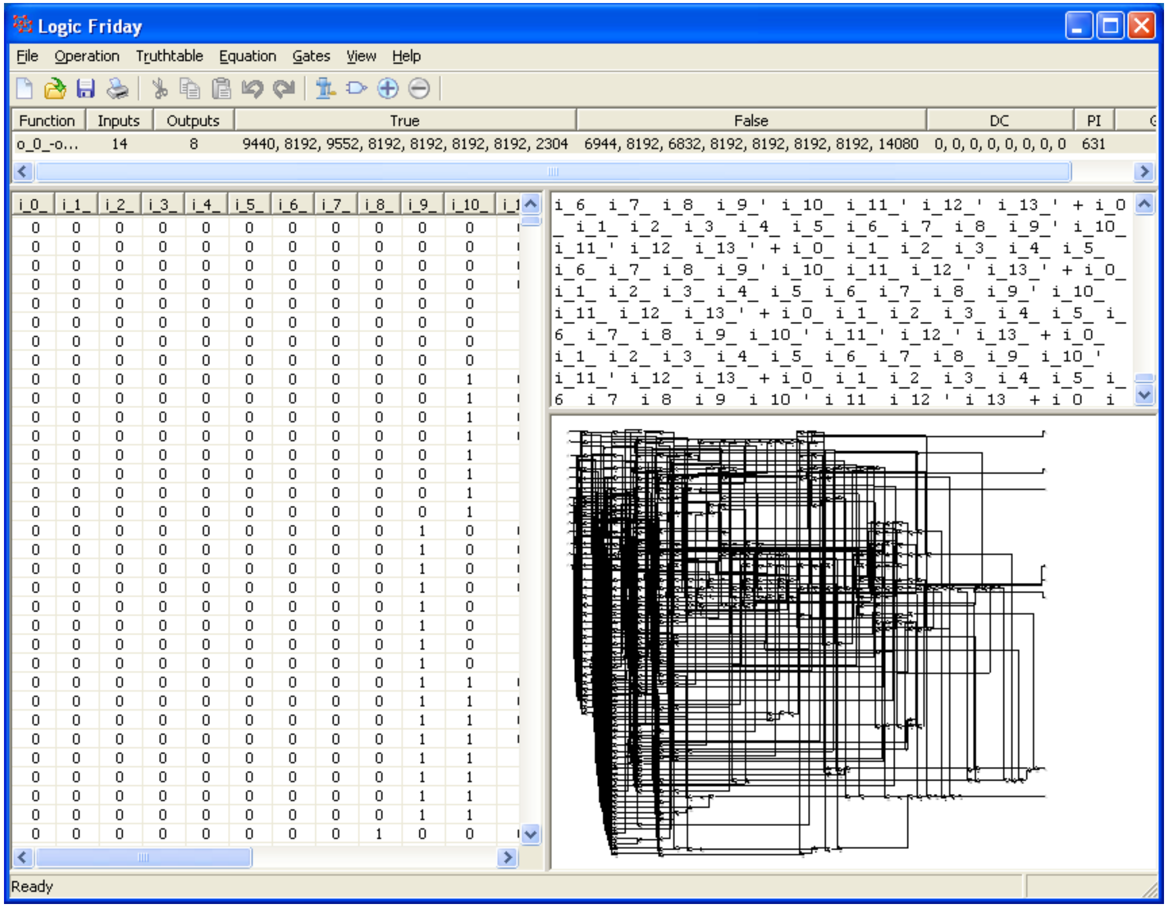
Efficient data structures (PCN)

Further reading in two-level minimization

- Using BDDs to implicitly represent and solve the covering problem
  - "A New Viewpoint on Two-Level Logic Minimization," O. Coudert, J. C. Madre, and H. Fraisse, Proc. Design Automation Conference, 1993, pp. 625-630.
- Better lower bound computation and pruning techniques
  - "New Ideas for Solving Covering Problems," O. Coudert, J. C. Madre, Proc. Design Automation Conference, 1995, pp. 641-646.
- Using Linear Programming for Lower Bound Computation
  - "Solving Covering Problems Using LPR-Based Lower Bounds," S. Liao and S. Devadas, Proc. Design Automation Conference, pp. 117-120, 1997.
- Do not generate primes that will not appear in the minimum solution
  - P. McGeer, J. Sanghavi, R. Brayton, and A. Sangiovanni-Vincentelli. Espresso-Signature: A New Exact Minimizer for Logic Functions. Design Automation Conference, pp. 618-624, 1993.
- Overview papers
  - "Complexity of two-level logic minimization," C. Umans, T. Villa, and A. L. Sangiovanni-Vincentelli, IEEE Trans. On Computer-Aided Design, vol. 25, no. 7, pp. 1230-1246, July 2006.
  - "Two-level logic minimization: An overview," O. Coudert, Integration – The VLSI Journal, vol. 17, no. 2, pp. 97-140, October 1994.

Demonstration: Logic Friday

- Free logic synthesis tool for students, hobbyists, and engineers who work on digital logic circuits (http://www.sontrak.com)
- Based on ESPRESSO and MIS II packages from U.C. Berkeley



Function	Inputs	Outputs	True	False	DC	PI
o_0_-o...	14	8	9440, 8192, 9552, 8192, 8192, 8192, 8192, 2304	6944, 8192, 6832, 8192, 8192, 8192, 8192, 14080	0, 0, 0, 0, 0, 0, 0, 0	631

i_0_	i_1_	i_2_	i_3_	i_4_	i_5_	i_6_	i_7_	i_8_	i_9_	i_10_
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	1	0	0

```
i_6_  i_7_  i_8_  i_9_' i_10_  i_11_' i_12_' i_13_' + i_0
_  i_1_  i_2_  i_3_  i_4_  i_5_  i_6_  i_7_  i_8_  i_9_' i_10_
i_11_' i_12_  i_13_' + i_0_  i_1_  i_2_  i_3_  i_4_  i_5_
i_6_  i_7_  i_8_  i_9_' i_10_  i_11_  i_12_' i_13_' + i_0_
i_1_  i_2_  i_3_  i_4_  i_5_  i_6_  i_7_  i_8_  i_9_' i_10_
i_11_  i_12_  i_13_' + i_0_  i_1_  i_2_  i_3_  i_4_  i_5_  i_
6_  i_7_  i_8_  i_9_  i_10_' i_11_' i_12_' i_13_  + i_0_
i_1_  i_2_  i_3_  i_4_  i_5_  i_6_  i_7_  i_8_  i_9_  i_10_'
i_11_' i_12_  i_13_  + i_0_  i_1_  i_2_  i_3_  i_4_  i_5_  i_
6  i 7  i 8  i 9  i 10 ' i 11  i 12 ' i 13  + i 0  i
```

Ready

Logic Friday: What can it do?

- With Logic Friday you can
  - Enter and view a logic function as a truth table, an equation, or a gate diagram
  - Enter functions with up to 16 inputs and 16 outputs
  - Minimize a function with options of fast or exact minimization
  - Automatically generate a multi-level gate diagram using gates chosen from a library
  - Automatically minimize the number of standard gate packages
  - Trace the logic state of each gate's inputs and outputs for a given input vector
  - Compare logic functions
  - Generate new functions as logical combinations of others
  - Generate efficient, compact C code lookup functions from logic functions
  - Save functions and gate diagram images to files
  - Export and import truth tables as CSV files for editing in spreadsheet applications.

Example of ESPRESSO Input/Output

$$f(A,B,C,D) = \sum m(4,5,6,8,9,10,13) + \sum d(0,7,15)$$

Espresso Input

```
.i 4             -- # inputs
.o 1             -- # outputs
.ilb a b c d     -- input names
.ob f            -- output name
.p 10            -- number of product terms
0100    1        -- A'BC'D'
0101    1        -- A'BC'D
0110    1        -- A'BCD'
1000    1        -- AB'C'D'
1001    1        -- AB'C'D
1010    1        -- AB'CD'
1101    1        -- ABC'D
0000    -        -- A'B'C'D' don't care
0111    -        -- A'BCD don't care
1111    -        -- ABCD don't care
.e               -- end of list
```

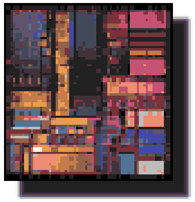
Espresso Output

```
.i 4
.o 1
.ilb a b c d
.ob f
.p 3
1-01   1
10-0   1
01--   1
.e
```

f =

ECE 595Z
Digital VLSI Design Automation

Module 4 (Lectures 11-13): Boolean Satisfiability



Anand Raghunathan
MSEE 348
raghunathan@purdue.edu

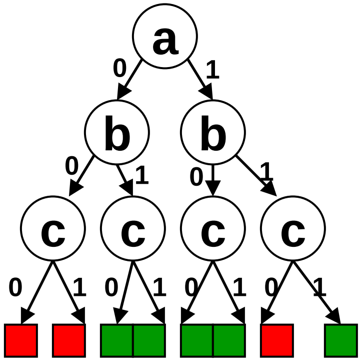
Acknowledgment : Prof. Sharad Malik, Princeton University

SAT in a Nutshell

- Given a Boolean formula, find a variable assignment such that the formula evaluates to 1, or prove that no such assignment exists.

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

- For n variables, there are 2^n possible truth assignments to be checked.

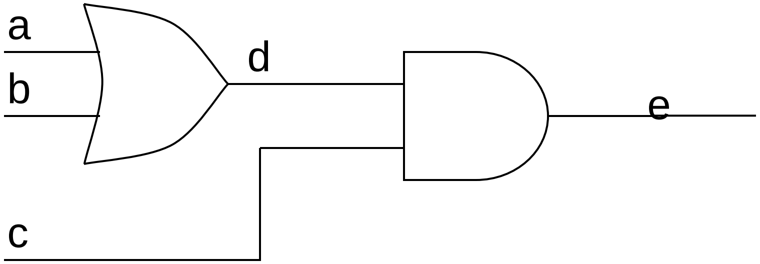


- First established NP-Complete problem.

  S. A. Cook, The complexity of theorem proving procedures, *Proceedings, Third Annual ACM Symp. on the Theory of Computing,* 1971, 151-158

Problem Representation

- Conjunctive Normal Form (CNF)
  - (a + b)(a’ + b’ + c)
- Logic circuit representation
  - Circuits have structural and direction information
- Circuit – CNF conversion is straightforward – Tseitin Transformation



$d \equiv (a + b)$

(a + b + d’)

(a’ + d)

(b’ + d)

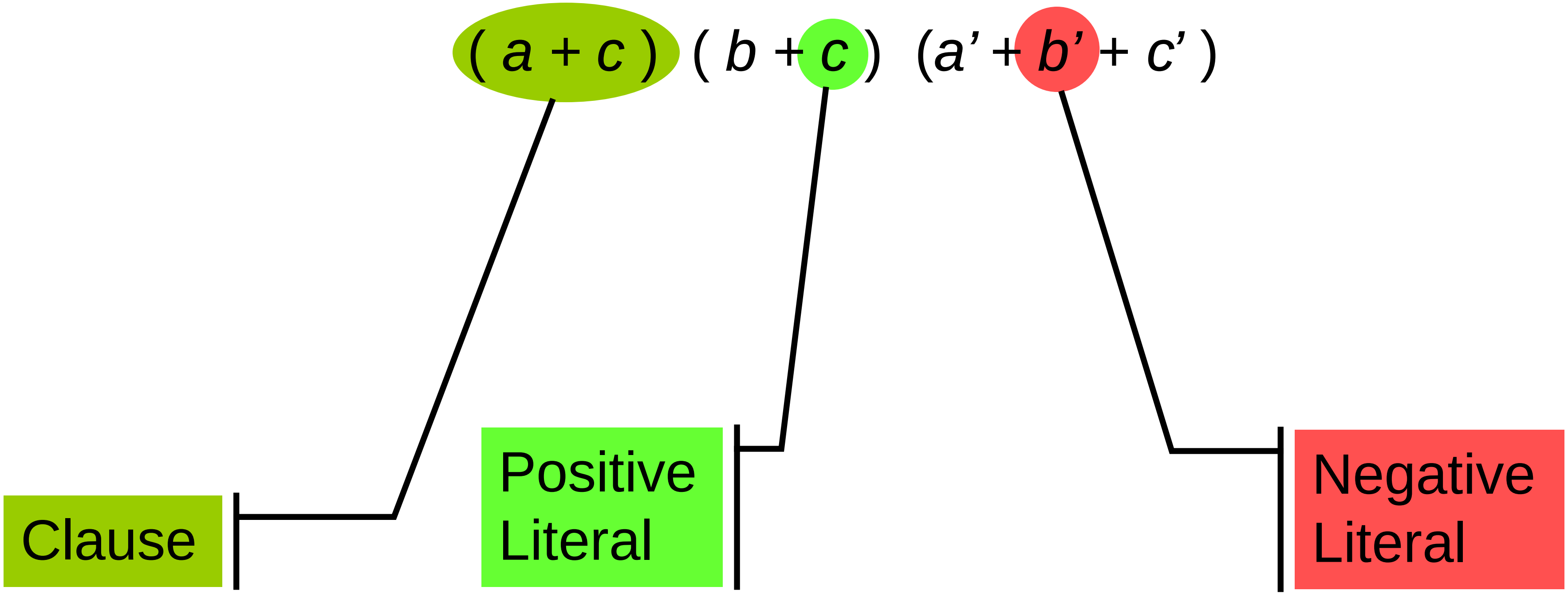
$e \equiv (c \cdot d)$

(c’ + d’ + e)

(d + e’)

(c + e’)

Terminology



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

Clause

Positive Literal

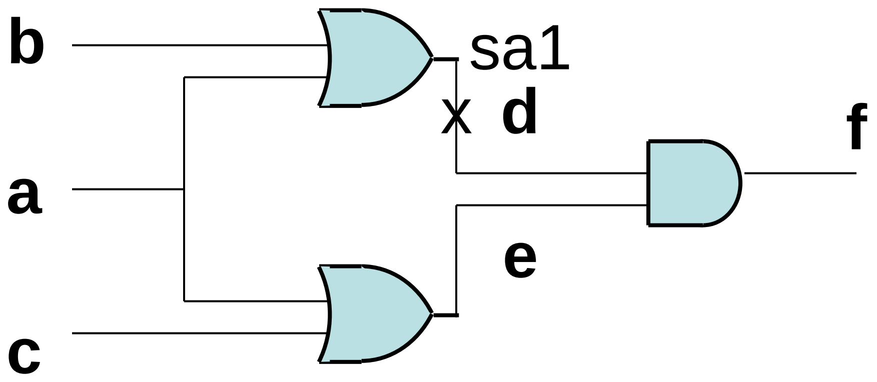
Negative Literal

Why Bother?

- Core computational engine for major applications
  - AI
    - Knowledge base deduction
    - Automatic theorem proving
  - EDA
    - Testing
    - Verification
    - Timing analysis
    - Power analysis
    - and more…

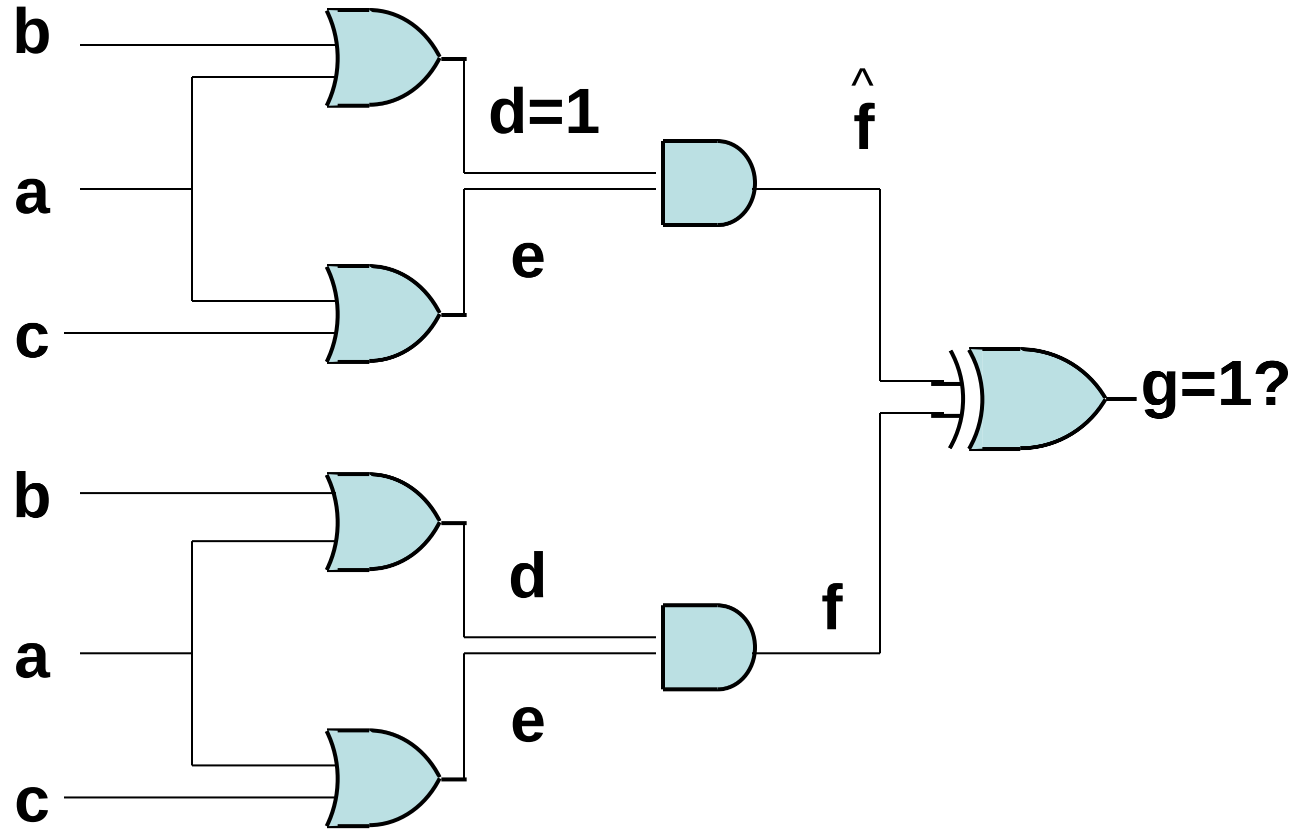
EDA Drivers for SAT

- Automatic Test Pattern Generation (ATPG)
  - Example: Manufacturing defects modeled as Stuck-at faults



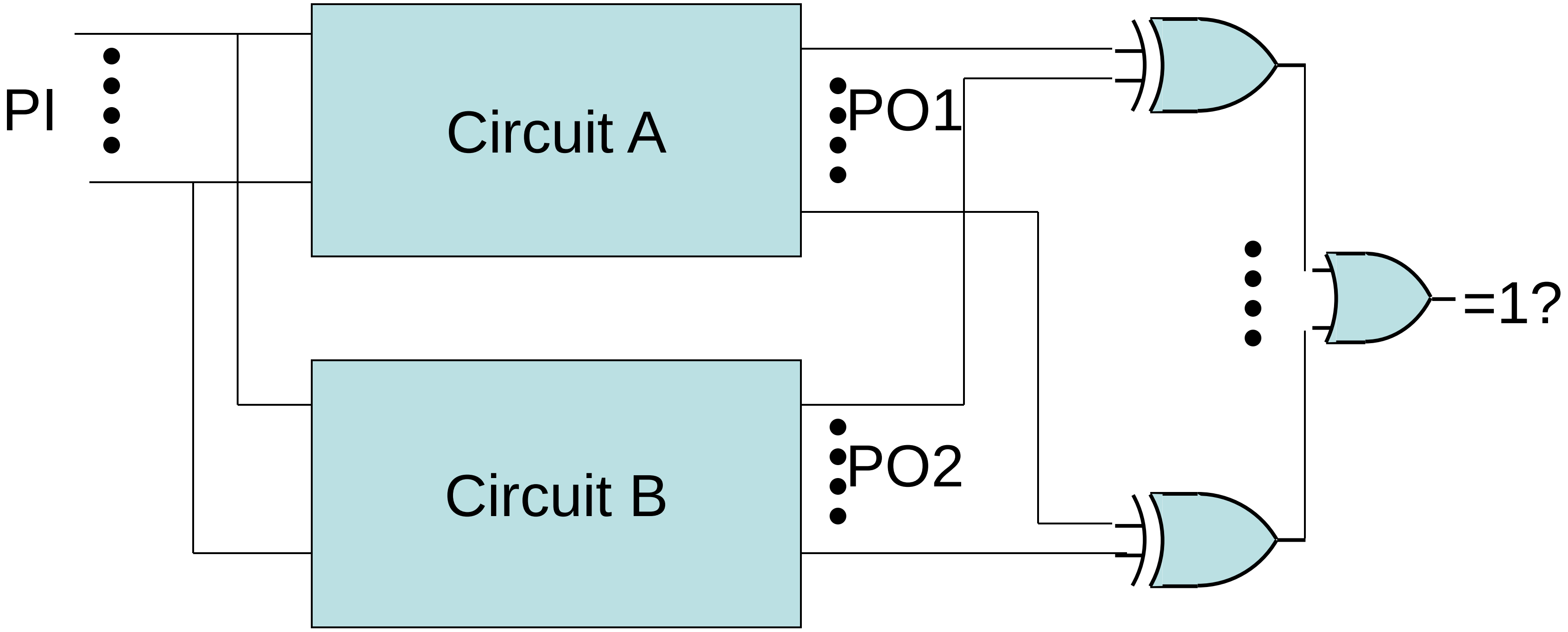
EDA Drivers for SAT

- ATPG
  - Miter : Two copies of a circuit feeding an XOR



EDA Drivers for SAT

- Combinational Equivalence Checking



History of SAT solvers

1869: William Stanley Jevons: Logic Machine

W S Jevons, *On the Mechanical Performance of Logical Inference*, In Philosophical Transactions of the Royal Society, Vol. 160, Part II, pp. 497-518, Oct. 1869.

W S Jevons; *Pure Logic and Other Minor Works, Pure Logic or the Logic of Quality Apart From the Quantity*; Macmillan and Co., London, 1890

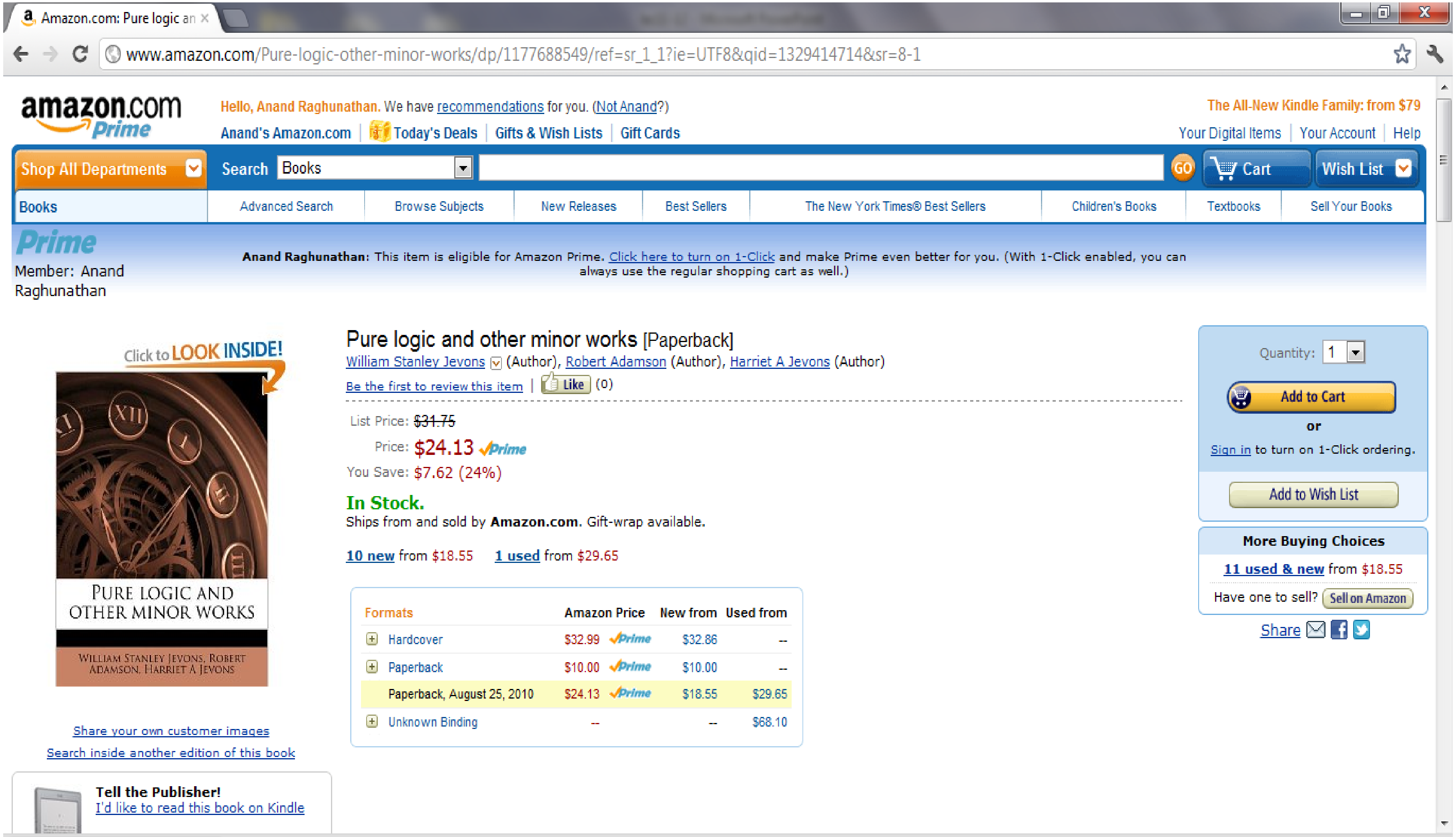
The Logical Machine

- First attempt to construct a “reasoning” machine
  - Based on principle of “substitution of similars”
- Better known for his contributions to economics – marginal utility theory

Jevons’ logical machine, exhibited before the Royal Society of London (1870)

William Stanley Jevons (1835-1882) economist and logician

For sale!



Amazon.com: Pure logic an ×

www.amazon.com/Pure-logic-other-minor-works/dp/1177688549/ref=sr_1_1?ie=UTF8&qid=1329414714&sr=8-1

amazon.com Prime

Hello, Anand Raghunathan. We have recommendations for you. (Not Anand?)

Anand's Amazon.com | Today's Deals | Gifts & Wish Lists | Gift Cards

The All-New Kindle Family: from $79

Your Digital Items | Your Account | Help

Shop All Departments | Search: Books | GO | Cart | Wish List

Books | Advanced Search | Browse Subjects | New Releases | Best Sellers | The New York Times® Best Sellers | Children's Books | Textbooks | Sell Your Books

Prime
Member: Anand Raghunathan

Anand Raghunathan: This item is eligible for Amazon Prime. Click here to turn on 1-Click and make Prime even better for you. (With 1-Click enabled, you can always use the regular shopping cart as well.)

Click to LOOK INSIDE!

PURE LOGIC AND OTHER MINOR WORKS

WILLIAM STANLEY JEVONS, ROBERT ADAMSON, HARRIET A JEVONS

Share your own customer images

Search inside another edition of this book

Tell the Publisher!
I'd like to read this book on Kindle

Pure logic and other minor works [Paperback]

William Stanley Jevons (Author), Robert Adamson (Author), Harriet A Jevons (Author)

Be the first to review this item | Like (0)

List Price: ~~$31.75~~

Price: $24.13 ✓Prime

You Save: $7.62 (24%)

In Stock.

Ships from and sold by **Amazon.com**. Gift-wrap available.

10 new from $18.55 1 used from $29.65

Formats	Amazon Price		New from	Used from
⊞ Hardcover	$32.99	✓Prime	$32.86	--
⊞ Paperback	$10.00	✓Prime	$10.00	--
Paperback, August 25, 2010	$24.13	✓Prime	$18.55	$29.65
⊞ Unknown Binding	--		--	$68.10

Quantity: 1

Add to Cart

or

Sign in to turn on 1-Click ordering.

Add to Wish List

More Buying Choices

11 used & new from $18.55

Have one to sell? Sell on Amazon

Share

History of SAT solvers

1952
Quine
Iterated Consensus
≈10 variables

W. V. Quine, “The problem of simplifying truth functions”, *Amer. Math Monthly* Vol. 59, pp. 521-531, 1952.

Recall Iterated Consensus?

- Iterated consensus generates all prime implicants.
  - Starting point is Disjunctive Normal Form (DNF) or SOP
- Iterated consensus can be used to check tautology of a DNF formula
  - For a tautological formula, the only prime is 1
- Tautology checking on DNF is the dual problem of satisfiability checking for CNF
  - A SAT Checking Procedure!

Iterated Consensus

CNF formula

(a + b + c)(b + c’ + f’)(b’ + e)

CNF formula

(a + b) (a + b’) (a’ + c)(a’ + c’)

Iterated Consensus

CNF formula

(a + b + c)(b + c’ + f’)(b’ + e)

DNF for complement

a’b’c’ + b’cf + be’

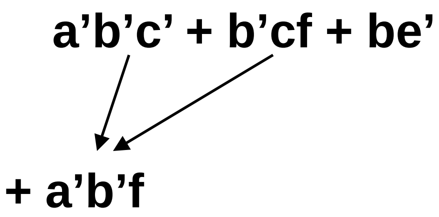
CNF formula

(a + b) (a + b’) (a’ + c)(a’ + c’)

DNF for complement

a’b’ + a’b + ac’ + ac

Iterated Consensus

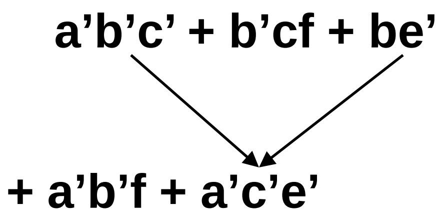


a’b’c’ + b’cf + be’

+ a’b’f

a’b’ + a’b + ac’ + ac

Iterated Consensus

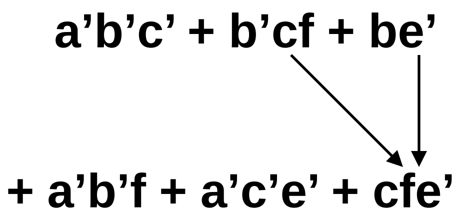


a’b’c’ + b’cf + be’

+ a’b’f + a’c’e’

a’b’ + a’b + ac’ + ac

Iterated Consensus

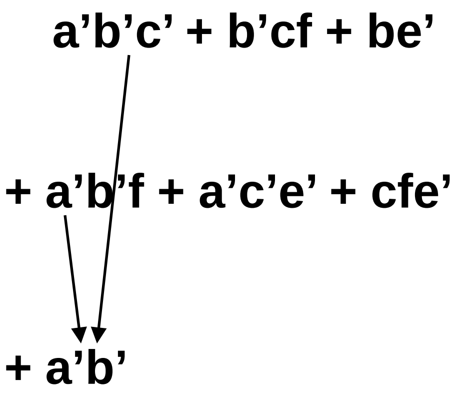


a’b’c’ + b’cf + be’

+ a’b’f + a’c’e’ + cfe’

a’b’ + a’b + ac’ + ac

Iterated Consensus



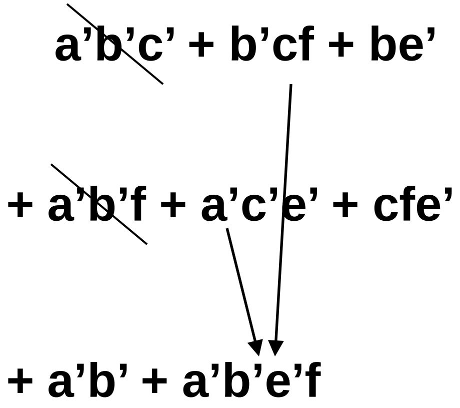
a’b’c’ + b’cf + be’

+ a’b’f + a’c’e’ + cfe’

+ a’b’

a’b’ + a’b + ac’ + ac

Iterated Consensus



~~a’b’c’~~ + b’cf + be’

+ ~~a’b’f~~ + a’c’e’ + cfe’

+ a’b’ + a’b’e’f

a’b’ + a’b + ac’ + ac

Iterated Consensus

~~a’b’c’~~ + b’cf + be’

+ ~~a’b’f~~ + a’c’e’ + cfe’

+ a’b’ + ~~a’b’e’f~~

a’b’ + a’b + ac’ + ac

Iterated Consensus

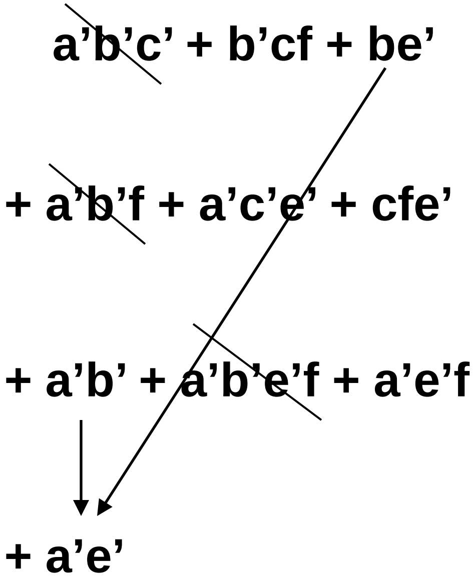
$\cancel{a'b'c'} + b'cf + be'$

$a'b' + a'b + ac' + ac$

$+ \cancel{a'b'f} + a'c'e' + cfe'$

$+ a'b' + \cancel{a'b'e'f} + a'e'f$

Iterated Consensus



a’~~b’c’~~ + b’cf + be’

+ a’~~b’f~~ + a’c’e’ + cfe’

+ a’b’ + ~~a’b’e’f~~ + a’e’f

+ a’e’

a’b’ + a’b + ac’ + ac

Iterated Consensus

~~a’b’c’~~ + b’cf + be’

a’b’ + a’b + ac’ + ac

+ ~~a’b’f~~ + ~~a’c’e’~~ + cfe’

+ a’b’ + ~~a’b’e’f~~ + ~~a’e’f~~

+ a’e’

Iterated Consensus

~~a’b’c’~~ + b’cf + be’

a’b’ + a’b + ac’ + ac

+ ~~a’b’f~~ + ~~a’c’e’~~ + cfe’

+ a’b’ + ~~a’b’e’f~~ + ~~a’e’f~~

+ a’e’

No more implicants
can be generated,
not a tautology

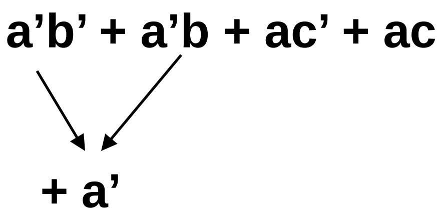
Iterated Consensus

~~a’b’c’~~ + b’cf + be’

+ ~~a’b’f~~ + ~~a’c’e’~~ + cfe’

+ a’b’ + ~~a’b’e’f~~ + ~~a’e’f~~

+ a’e’



a’b’ + a’b + ac’ + ac

+ a’

No more implicants
can be generated,
not a tautology

Iterated Consensus

~~a’b’c’~~ + b’cf + be’

+ ~~a’b’f~~ + ~~a’c’e’~~ + cfe’

+ a’b’ + ~~a’b’e’f~~ + ~~a’e’f~~

+ a’e’

No more implicants
can be generated,
not a tautology

~~a’b’~~ + ~~a’b~~ + ac’ + ac

+ a’

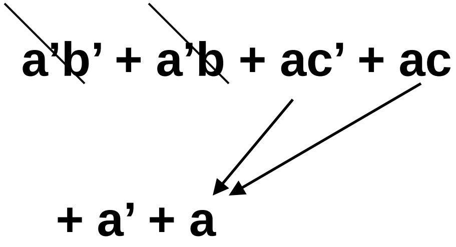
Iterated Consensus

$\cancel{a'b'c'} + b'cf + be'$

$+ \cancel{a'b'f} + \cancel{a'c'e'} + cfe'$

$+ a'b' + \cancel{a'b'e'f} + \cancel{a'e'f}$

$+ a'e'$



$\cancel{a'b'} + \cancel{a'b} + ac' + ac$

$+ a' + a$

No more implicants
can be generated,
not a tautology

Iterated Consensus

~~a’b’c’~~ + b’cf + be’

+ ~~a’b’f~~ + ~~a’c’e’~~ + cfe’

+ a’b’ + ~~a’b’e’f~~ + ~~a’e’f~~

+ a’e’

~~a’b’~~ + ~~a’b~~ + ~~ac’~~ + ~~ac~~

+ a’ + a

No more implicants
can be generated,
not a tautology

Iterated Consensus

$\cancel{a'b'c'} + b'cf + be'$

$+ \cancel{a'b'f} + \cancel{a'c'e'} + cfe'$

$+ a'b' + \cancel{a'b'e'f} + \cancel{a'e'f}$

$+ a'e'$

$\cancel{a'b'} + \cancel{a'b} + \cancel{ac'} + \cancel{ac}$

$$\begin{tikzcd} + a' \arrow[d] & + a \arrow[dl] \\ + 1 & \end{tikzcd}$$

No more implicants can be generated, not a tautology

Iterated Consensus

~~a’b’c’~~ + b’cf + be’

+ ~~a’b’f~~ + ~~a’c’e’~~ + cfe’

+ a’b’ + ~~a’b’e’f~~ + ~~a’e’f~~

+ a’e’

No more implicants can be generated, not a tautology

~~a’b’~~ + ~~a’b~~ + ~~ac’~~ + ~~ac~~

+ ~~a’~~ + ~~a~~

+ 1

Iterated Consensus

$\cancel{a'b'c'} + b'cf + be'$

$+ \cancel{a'b'f} + \cancel{a'c'e'} + cfe'$

$+ a'b' + \cancel{a'b'e'f} + \cancel{a'e'f}$

$+ a'e'$

No more implicants can be generated, not a tautology

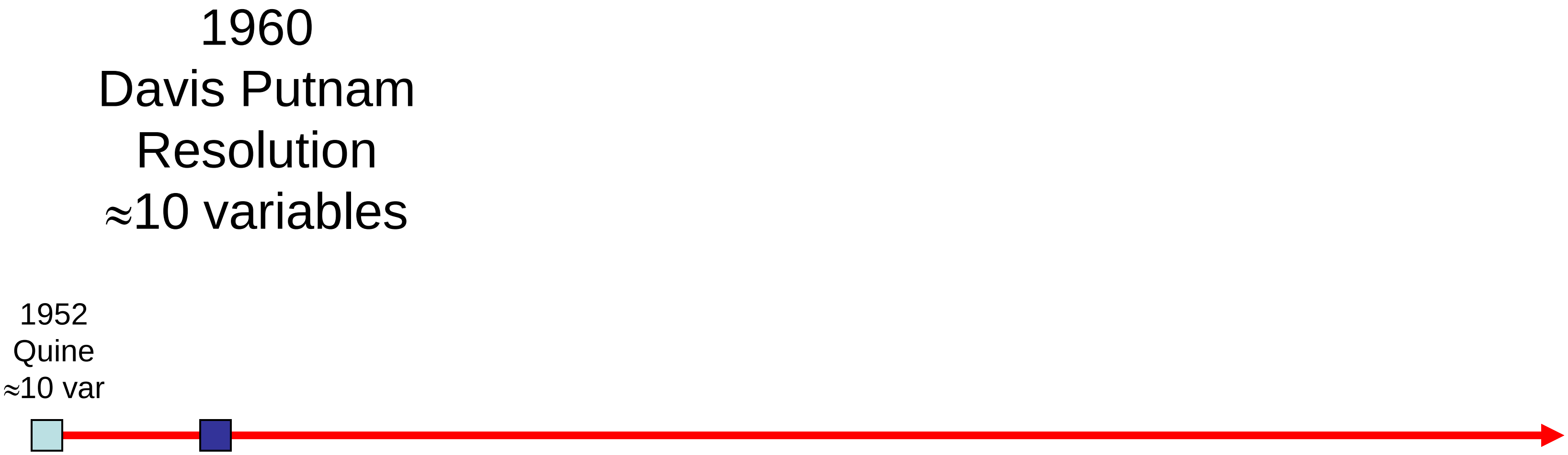
$\cancel{a'b'} + \cancel{a'b} + \cancel{ac'} + \cancel{ac}$

$+ \cancel{a'} + \cancel{a}$

$+ 1$

Tautology

The Timeline



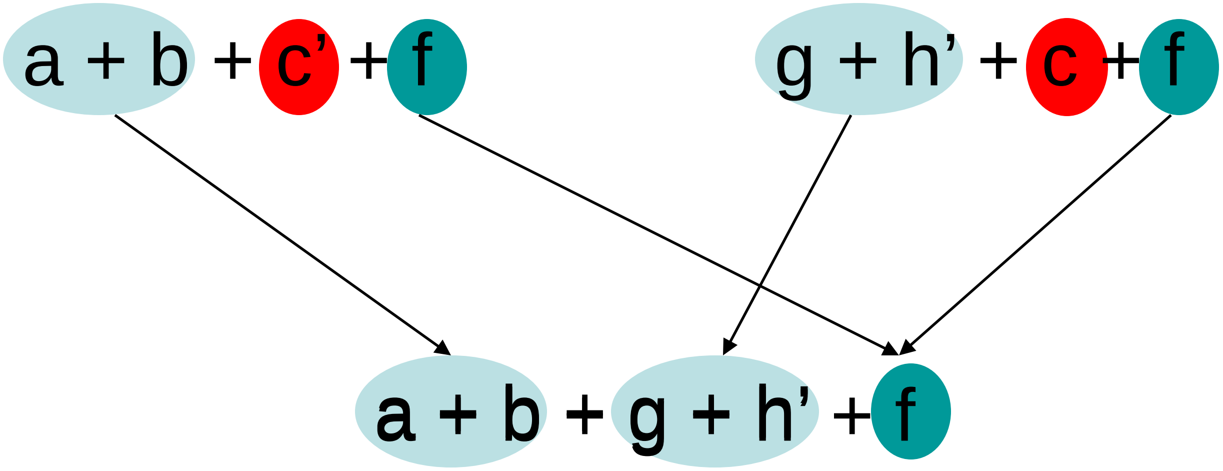
1960
Davis Putnam
Resolution
≈10 variables

1952
Quine
≈10 var

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

Resolution

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



a + b + c’ + f

g + h’ + c + f

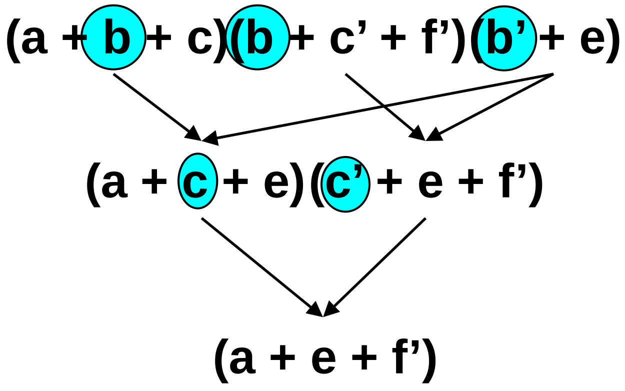
a + b + g + h’ + f

Resolution is the dual of consensus!

Davis Putnam Algorithm

Iteratively select a variable for resolution till no more variables are left.

- Can discard all original clauses after each iteration.

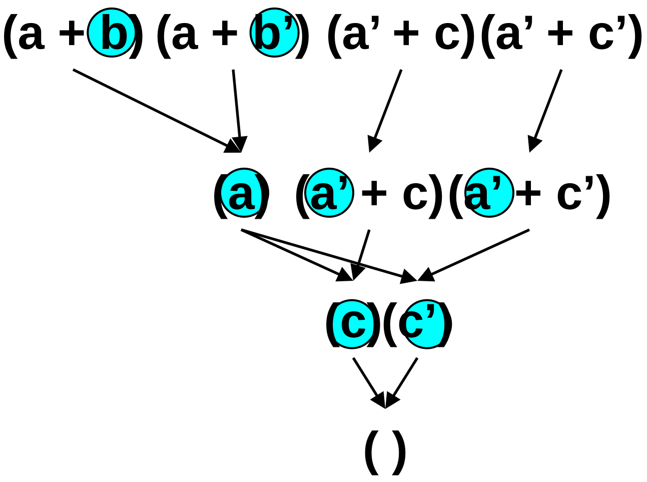


(a + b + c)(b + c’ + f’)(b’ + e)

(a + c + e)(c’ + e + f’)

(a + e + f’)

SAT



(a + b) (a + b’) (a’ + c)(a’ + c’)

(a) (a’ + c)(a’ + c’)

(c)(c’)

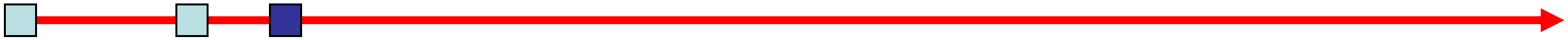
( )

UNSAT

Potential memory explosion problem!

The Timeline

1962
Davis Logemann Loveland
Depth First Search
≈ 10 var



1960
DP
≈ 10 var

1952
Quine
≈ 10 var

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

DLL Algorithm

- Davis, Logemann and Loveland
- Basic framework for many modern SAT solvers
- Also known as DPLL for historical reasons

Basic DLL Procedure - DFS

$(a' + b + c)$

$(a + c + d)$

$(a + c + d')$

$(a + c' + d)$

$(a + c' + d')$

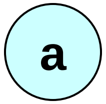
$(b' + c' + d)$

$(a' + b + c')$

$(a' + b' + c)$

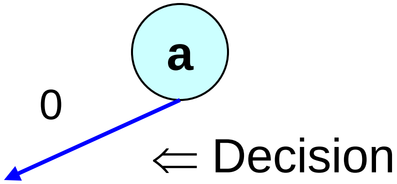
Basic DLL Procedure - DFS

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



Basic DLL Procedure - DFS

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



Basic DLL Procedure - DFS

(a’ + b + c)

(a + c + d)

(a + c + d’)

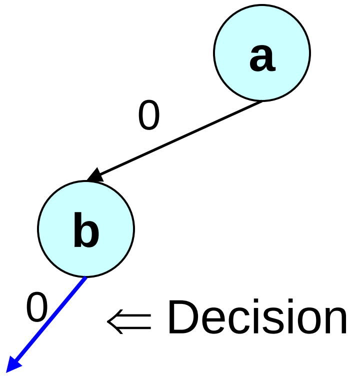
(a + c’ + d)

(a + c’ + d’)

(b’ + c’ + d)

(a’ + b + c’)

(a’ + b’ + c)



Basic DLL Procedure - DFS

(a’ + b + c)

(a + c + d)

(a + c + d’)

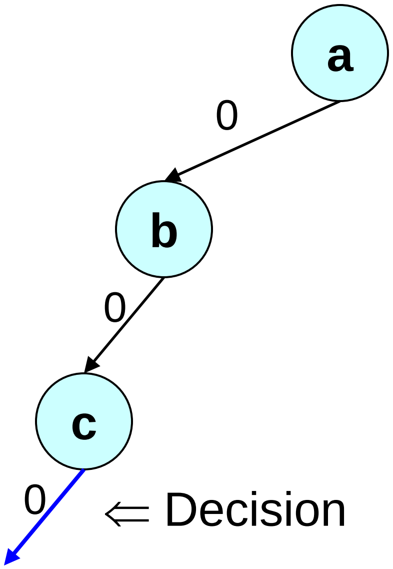
(a + c’ + d)

(a + c’ + d’)

(b’ + c’ + d)

(a’ + b + c’)

(a’ + b’ + c)



⇐ Decision

Basic DLL Procedure - DFS

(a' + b + c)

(a + c + d)

(a + c + d')

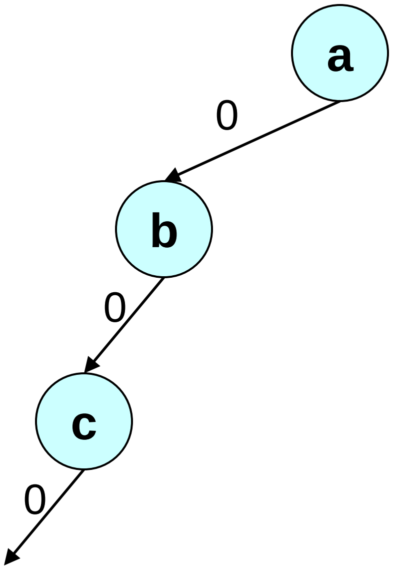
(a + c' + d)

(a + c' + d')

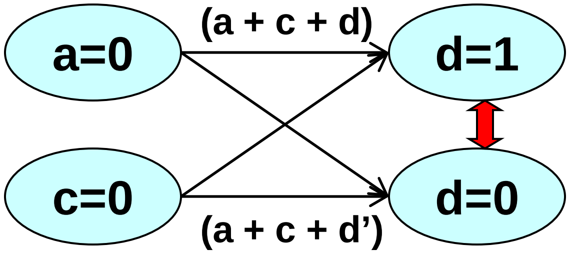
(b' + c' + d)

(a' + b + c')

(a' + b' + c)



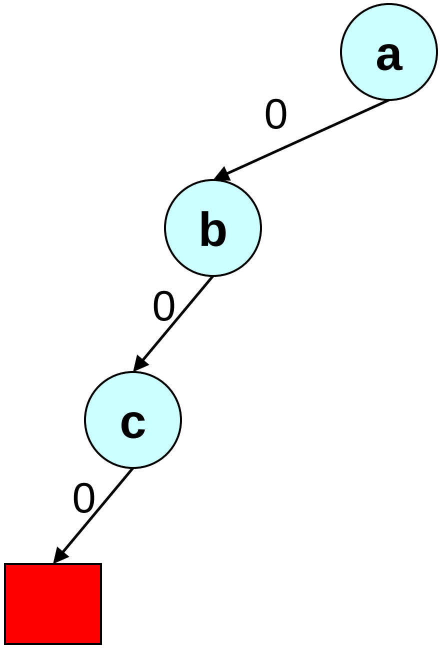
Implication Graph



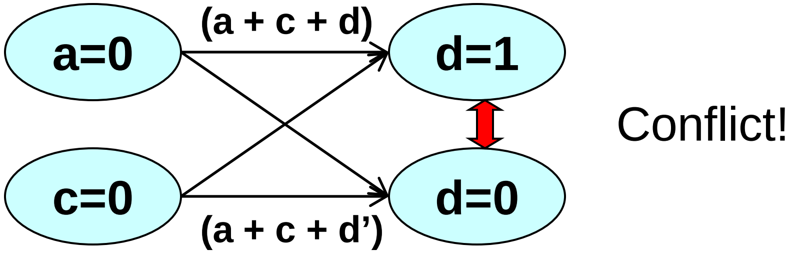
Conflict!

Basic DLL Procedure - DFS

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

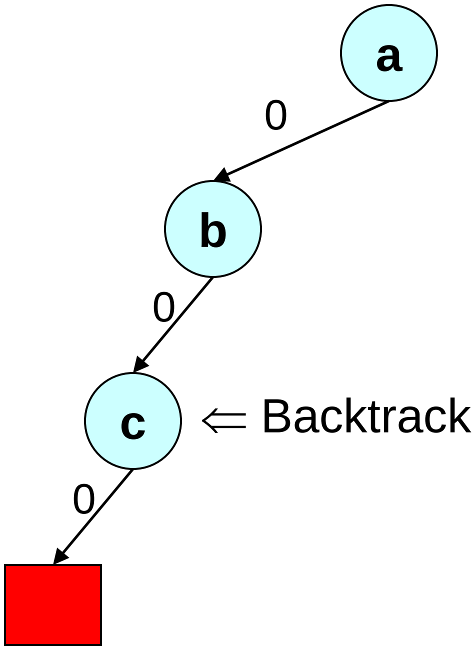


Implication Graph



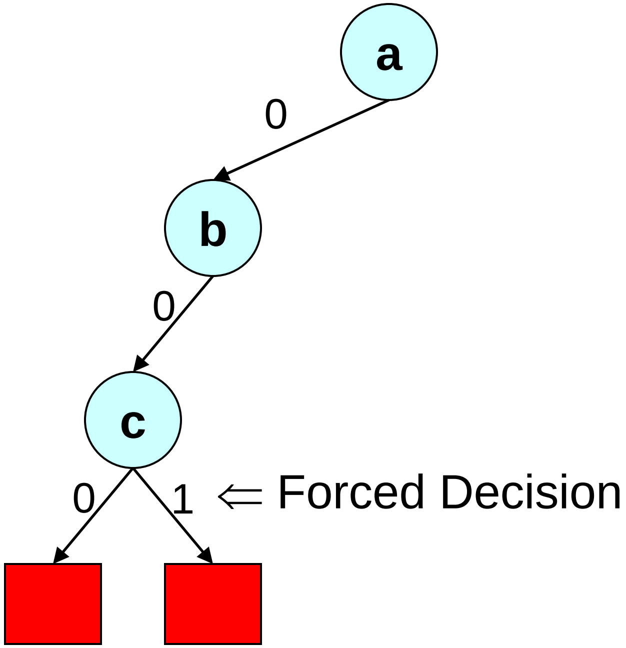
Basic DLL Procedure - DFS

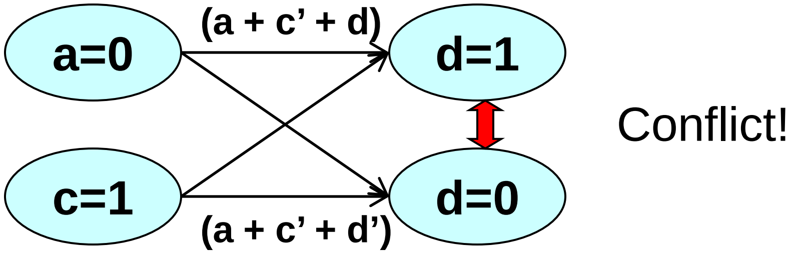
(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)



Basic DLL Procedure - DFS

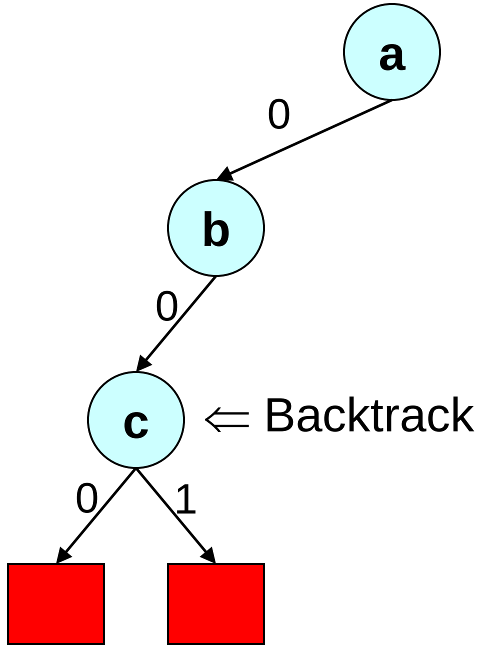
(a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)





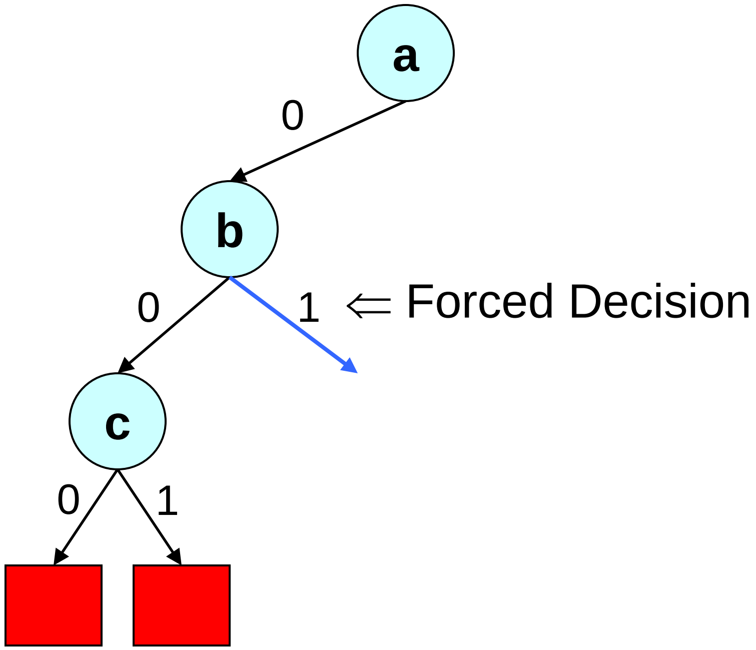
Basic DLL Procedure - DFS

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



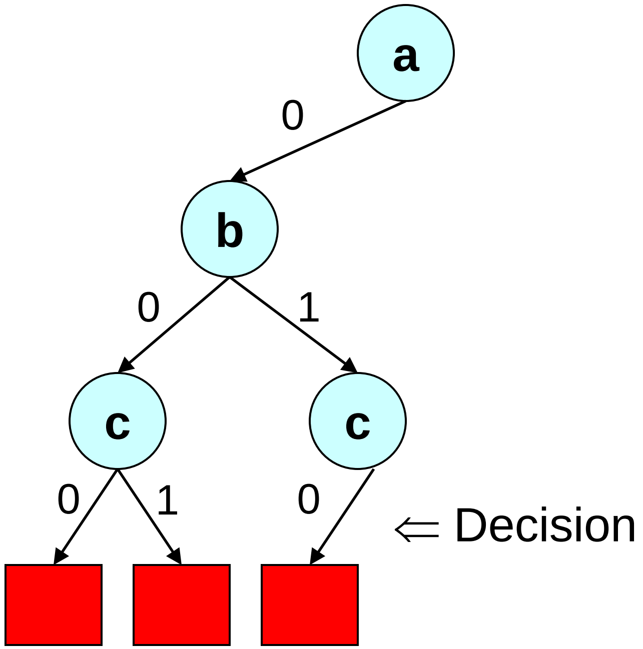
Basic DLL Procedure - DFS

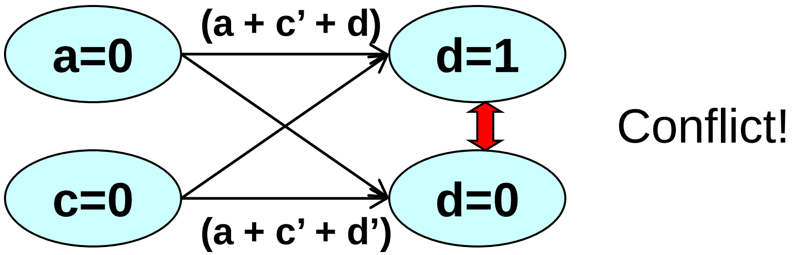
(a’ + b + c)
(a + c + d)
(a + c + d’)
(a + c’ + d)
(a + c’ + d’)
(b’ + c’ + d)
(a’ + b + c’)
(a’ + b’ + c)



Basic DLL Procedure - DFS

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





Basic DLL Procedure - DFS

(a’ + b + c)

(a + c + d)

(a + c + d’)

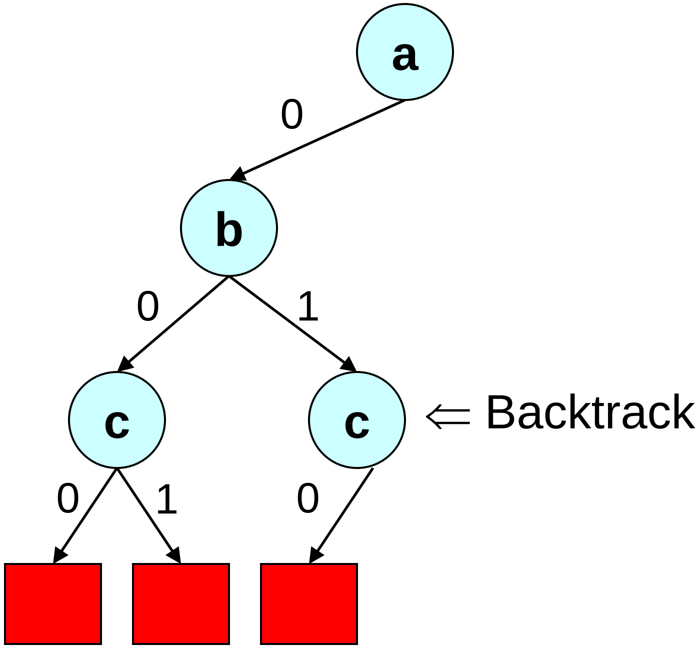
(a + c’ + d)

(a + c’ + d’)

(b’ + c’ + d)

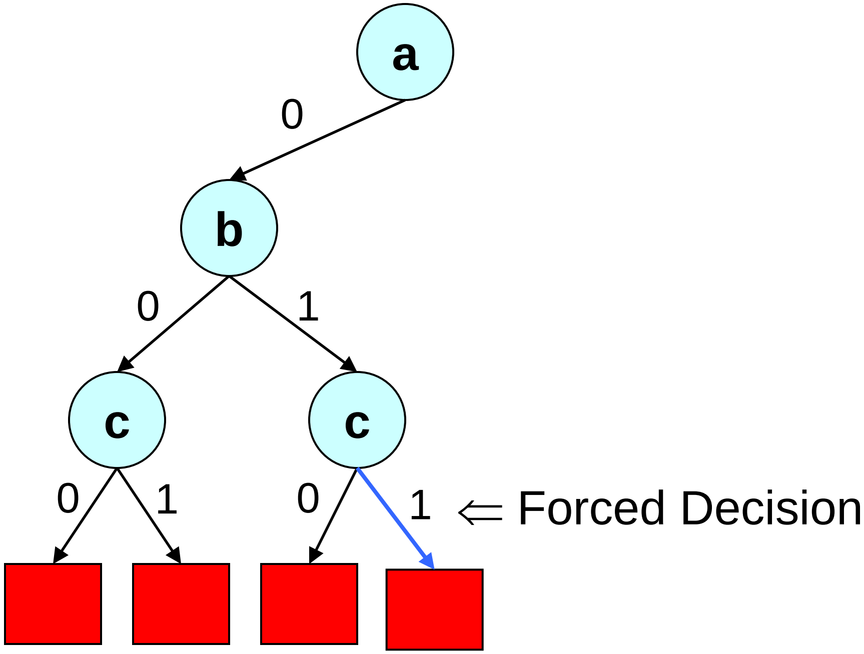
(a’ + b + c’)

(a’ + b’ + c)

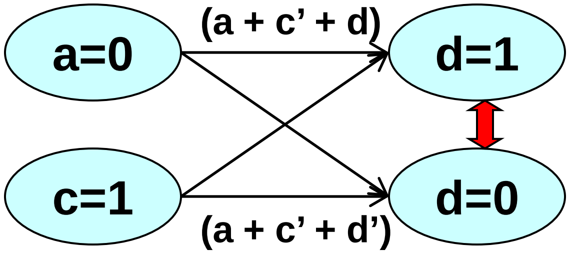


Basic DLL Procedure - DFS

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



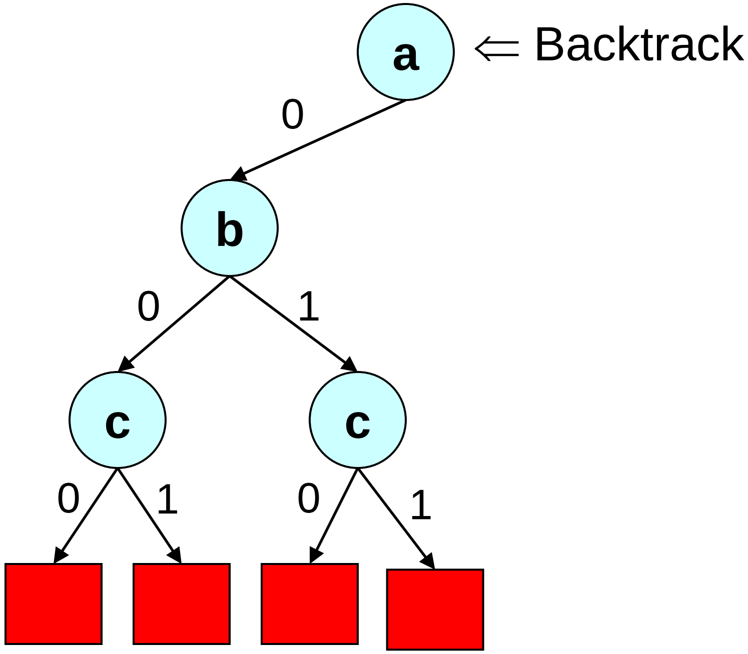
⇐ Forced Decision



Conflict!

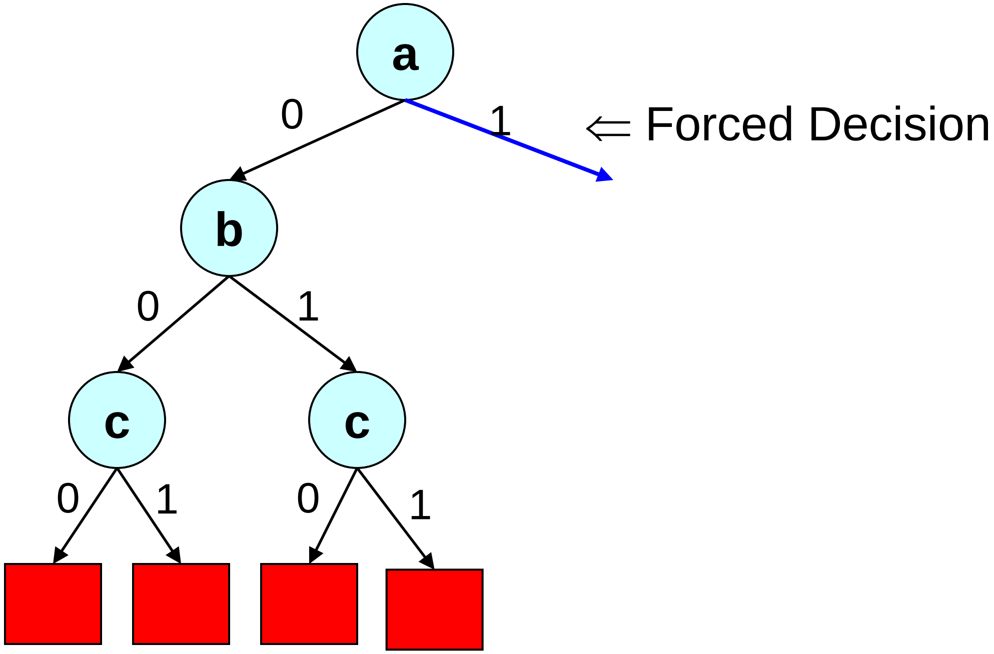
Basic DLL Procedure - DFS

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



Basic DLL Procedure - DFS

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



Basic DLL Procedure - DFS

(a’ + b + c)

(a + c + d)

(a + c + d’)

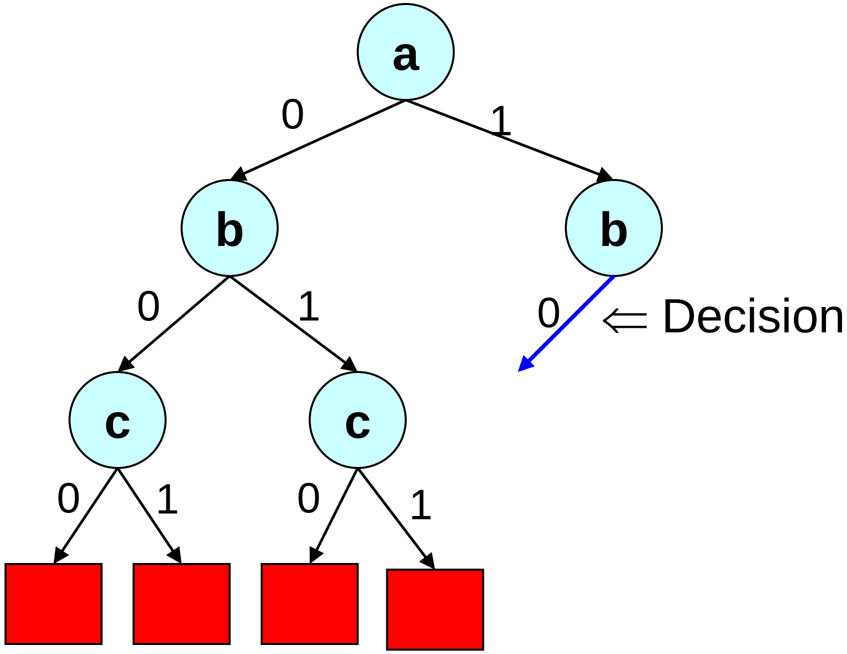
(a + c’ + d)

(a + c’ + d’)

(b’ + c’ + d)

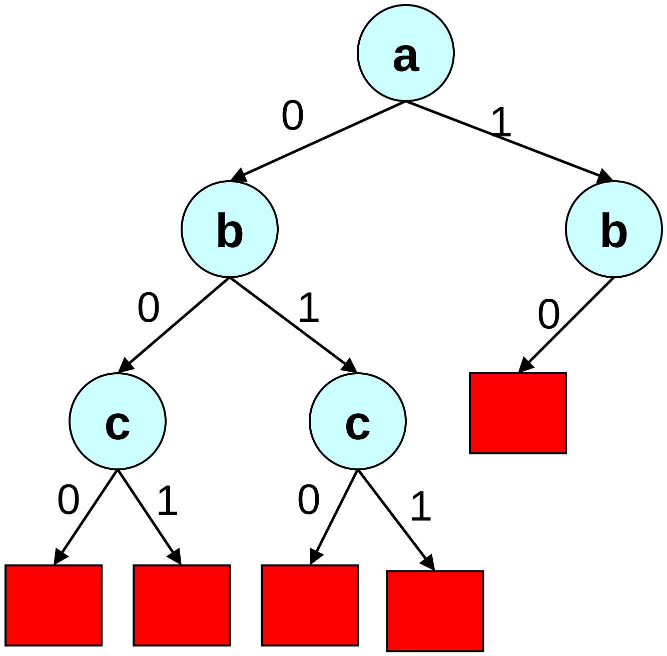
(a’ + b + c’)

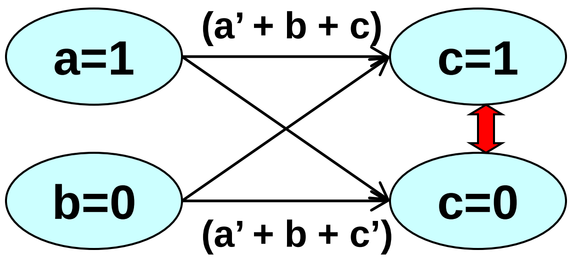
(a’ + b’ + c)



Basic DLL Procedure - DFS

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

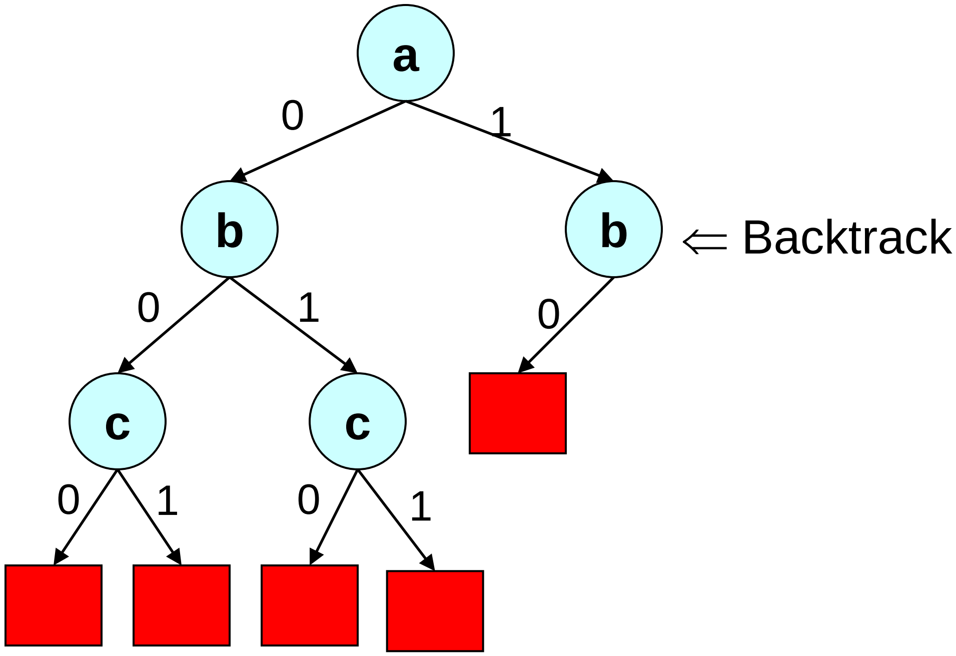




Conflict!

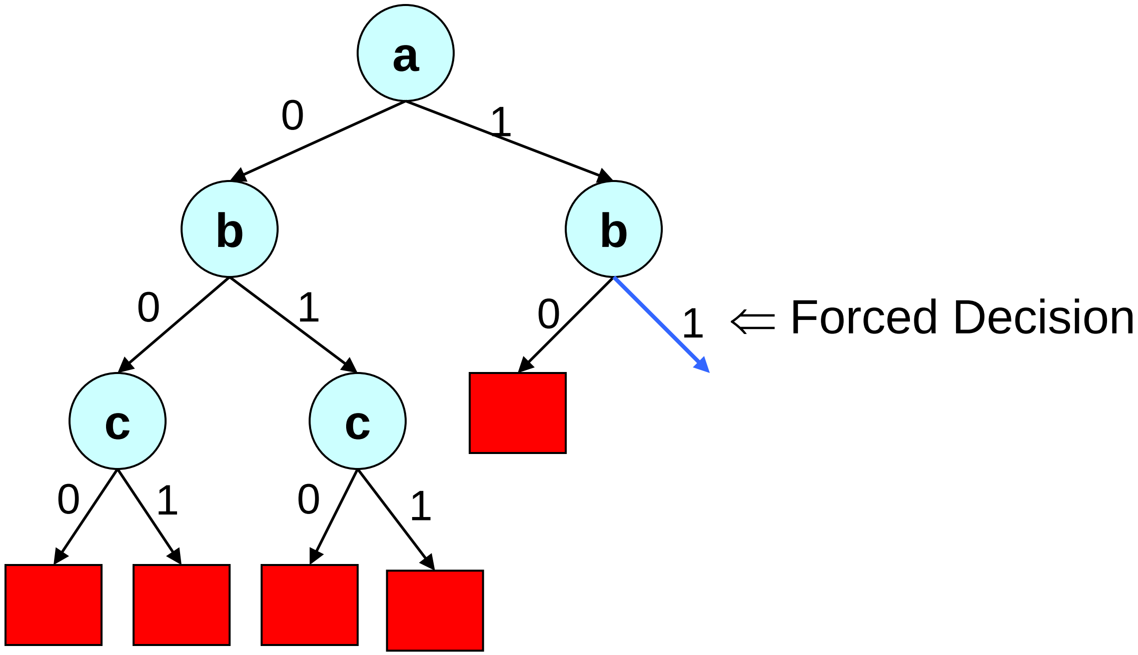
Basic DLL Procedure - DFS

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

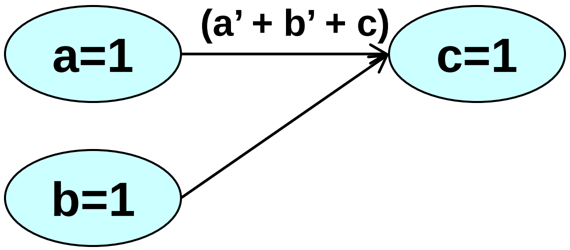


Basic DLL Procedure - DFS

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



⇐ Forced Decision



Basic DLL Procedure - DFS

(a' + b + c)

(a + c + d)

(a + c + d')

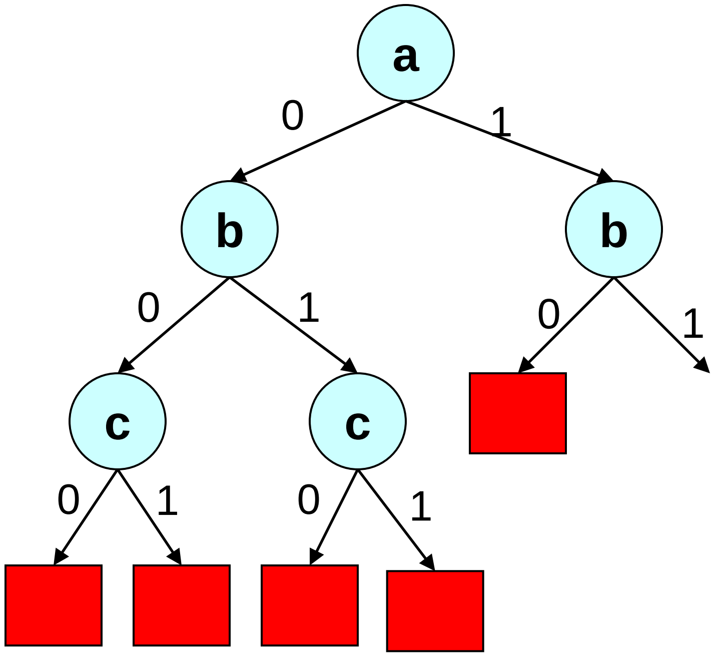
(a + c' + d)

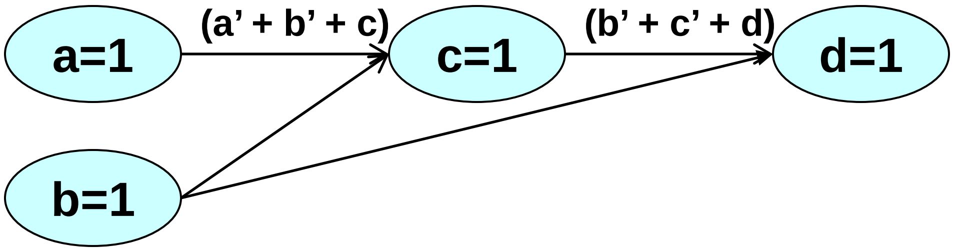
(a + c' + d')

(b' + c' + d)

(a' + b + c')

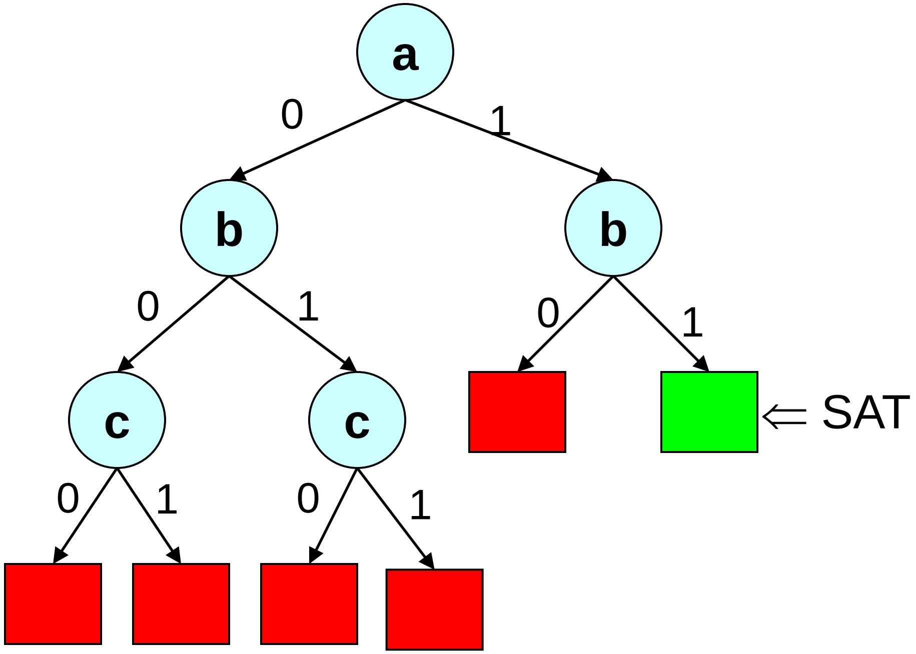
(a' + b' + c)

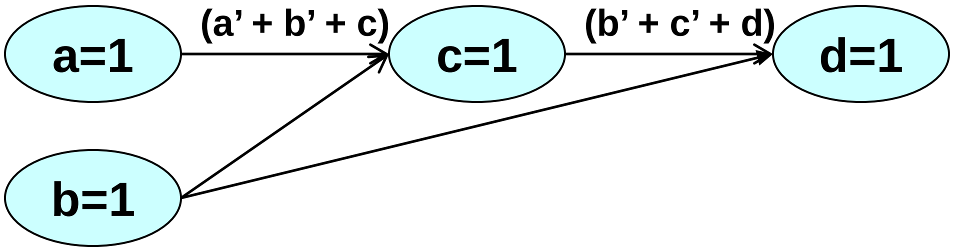




Basic DLL Procedure - DFS

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


a
0
1
b
b
0
1
0
1
c
c
⇐ SAT
0
1
0
1


a=1
(a’ + b’ + c)
c=1
(b’ + c’ + d)
d=1
b=1