A Linear-time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas

by

Aspvall, B., Plass, F.M. and Tarjan, R.E.

Information Processing Letters, Vol 8, No 3, March 1979

Presentation of

by

Martin Richards

mr@cl.cam.ac.uk

http://www.cl.cam.ac.uk/users/mr/

University Computer Laboratory
New Museum Site
Pembroke Street
Cambridge, CB2 3QG

The Problem

Evaluate the formula:

$$Q_1 x_1 Q_2 x_2 \cdots Q_n x_n \quad C$$

where each Q_i is either $\forall$ or $\exists$

and C is in conjunctive normal form with at most two literals per clause.

For example

$$\exists a \exists b \forall c \exists d (a \vee b) \wedge (b \vee \bar{c}) \wedge (\bar{b} \vee \bar{d}) \wedge (b \vee d) \wedge (d \vee a)$$

Note

$a \vee b$	is equivalent to	$\bar{a} \longrightarrow b$
	or	$\bar{b} \longrightarrow a$
$\bar{a} \vee b$	is equivalent to	$a \longrightarrow b$
	or	$\bar{b} \longrightarrow \bar{a}$
$a \vee \bar{b}$	is equivalent to	$\bar{a} \longrightarrow \bar{b}$
	or	$b \longrightarrow a$
$\bar{a} \vee \bar{b}$	is equivalent to	$a \longrightarrow \bar{b}$
	or	$b \longrightarrow \bar{a}$

Graph Formation

Without loss of generality assume that all clauses of C have two literals, since $(x) = (x \vee x)$.

Assuming the formula contains n variables $v_1, \ldots, v_n$, create $2n$ vertices named $v_1, \ldots, v_n$ and $\bar{v_1}, \ldots, \bar{v_n}$.

For each clause of the form: $(a \vee b)$, add edges $\bar{a} \rightarrow b$ and $\bar{b} \rightarrow a$.

Add appropriate edges for clauses of the form:

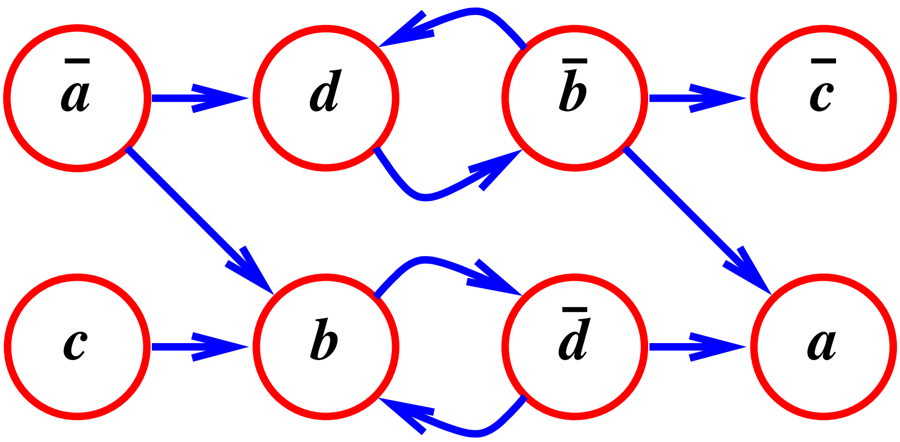
$$(a \vee \bar{b}),\ (\bar{a} \vee b) \text{ and } (\bar{a} \vee \bar{b}).$$

Example

If C is

$$(a \vee b) \wedge (b \vee \bar{c}) \wedge (\bar{b} \vee \bar{d}) \wedge (b \vee d) \wedge (d \vee a)$$

The graph is:



Duality Property

The constructed graph is isomorphic to one obtained by reversing all edges and complementing the names of the vertices.

Strongly Connected Components

A strongly connected component is a maximal subset of the vertices for which paths exists from any vertex to any other vertex.

All vertices in a strongly connected component must be assigned the same truth value
($x \longrightarrow y \longrightarrow \cdots \longrightarrow x$ implies $x = y$).

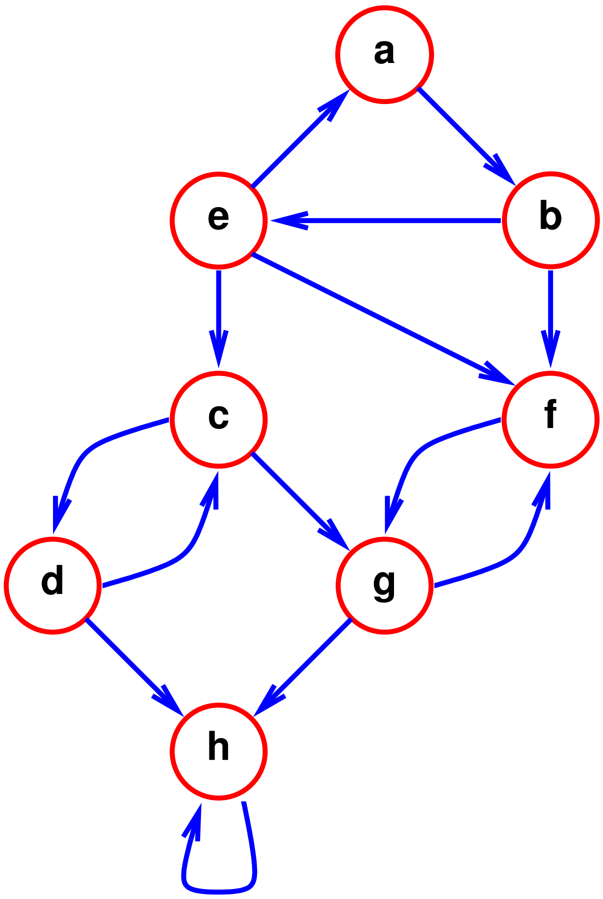
If a strongly connected component contain a variable x and its complement $\bar{x}$ then there is an inconsistency.

All the strongly connected components can be found in linear time.

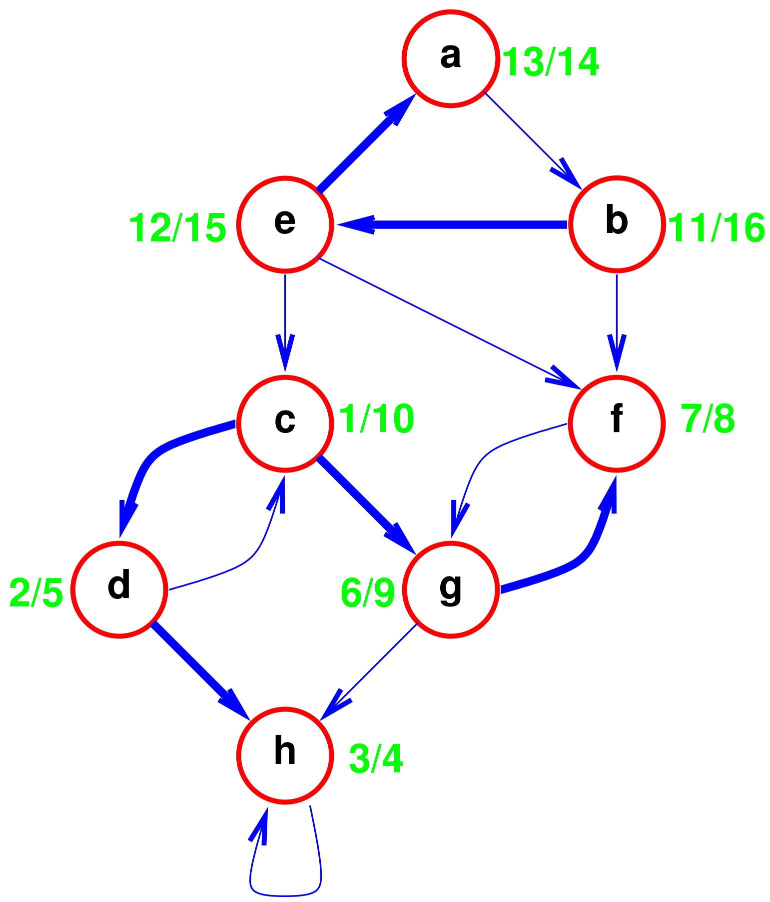
Components Algorithm

1) Perform depth first search on graph G(V,E), attaching the discovery time ($d[u]$) and finishing time ($f[u]$) to every vertex (u) of G.

For example, consider



After DFS



Forefather

Define $\phi(u) = w$ (the forefather of u)

where $w \epsilon V$

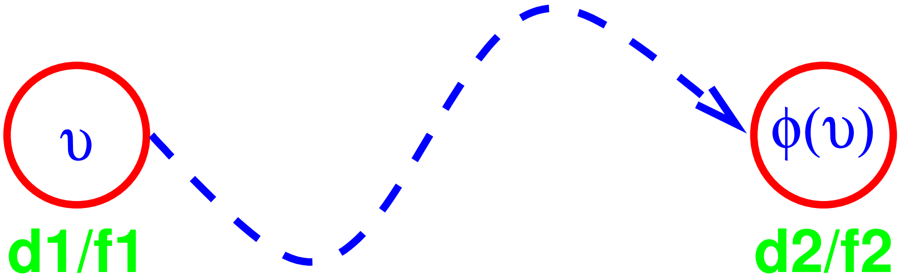
and $u \longrightarrow \cdots \longrightarrow w$

and $\forall w' (u \longrightarrow \cdots \longrightarrow w' \Rightarrow f[w'] \leq f[w])$

Clearly, since $u \longrightarrow \cdots \longrightarrow u$

$$f[u] \leq f[\phi(u)] \qquad (1)$$

Forefather Property



Either $u = \phi(u)$

or $u \neq \phi(u)$

if $d2 < f2 < d1 < f1$

or $d1 < d2 < f2 < f1$

contradicts of (1)

if $d1 < f1 < d2 < f2$

contradicts $u \longrightarrow \cdots \longrightarrow \phi(u)$

so $d2 < d1 < f1 < f2$

ie $\phi(u) \longrightarrow \cdots \longrightarrow u$

so u and $\phi(u)$ are in the same strongly connected component.

Algorithm Continued

2) Find the vertex r with largest $f[r]$ that is not in any strongly connected component so far identified. Note that r is a forefather.

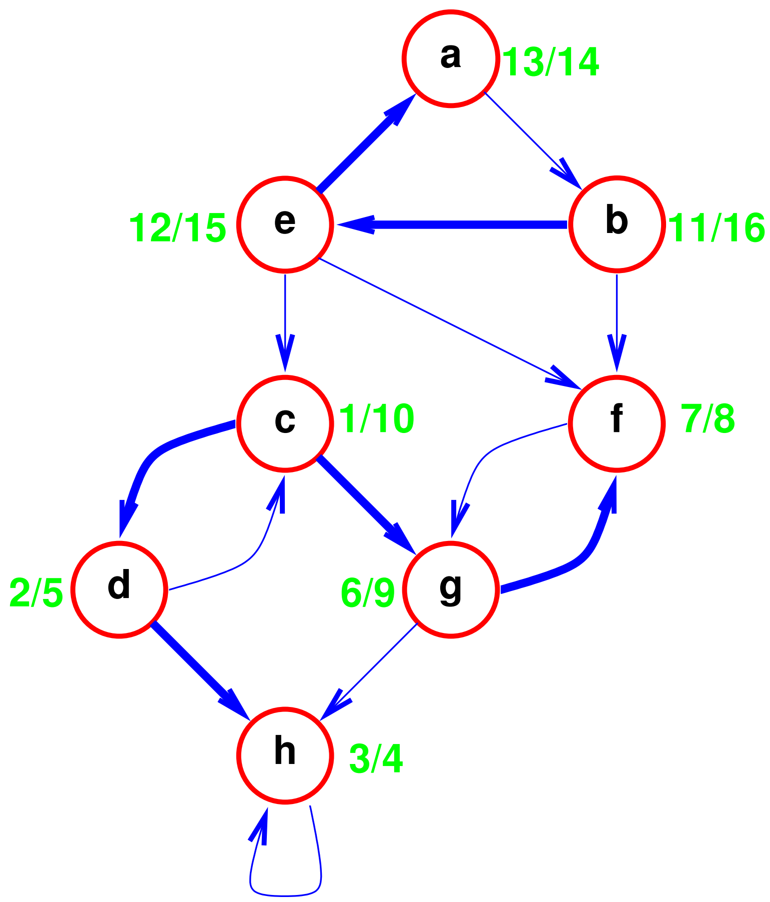
3) Form the set of vertices $\{u|\phi(u) = r\}$ – i.e. the strongly connected component containing r. This set is the same as $\{u|u \longrightarrow \cdots \longrightarrow r\}$

   This set is the set of vertices reachable from r in the graph $G^T = G$ with all its edges reversed. This set can be found using DFS on G^T.

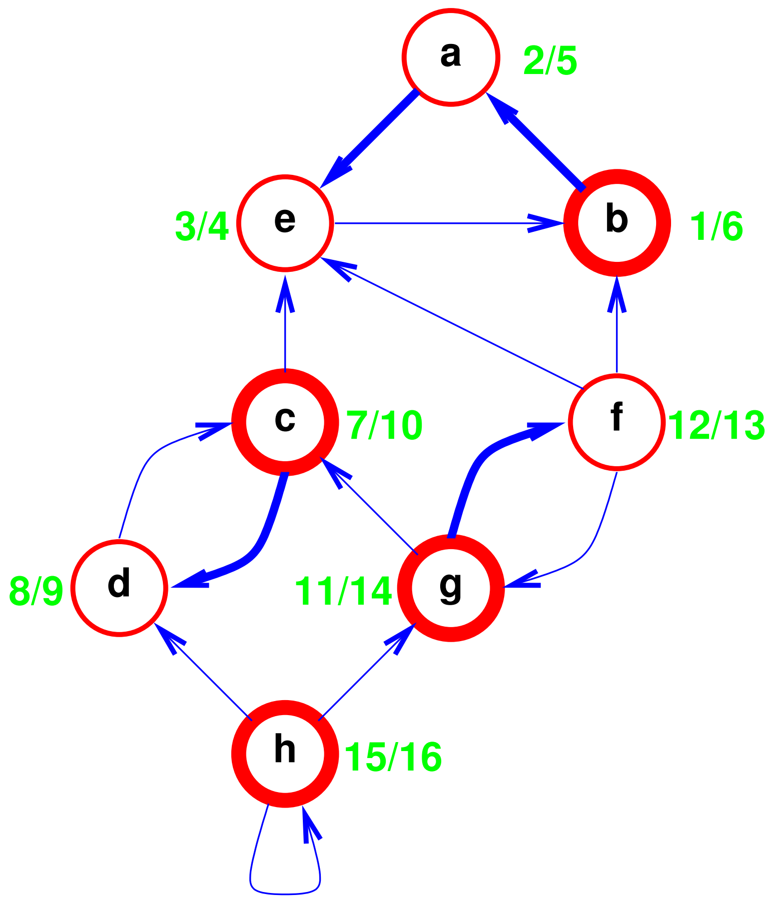
4) Repeat from (2) until all components have been found.

The complexity is $O(|V| + |E|)$.

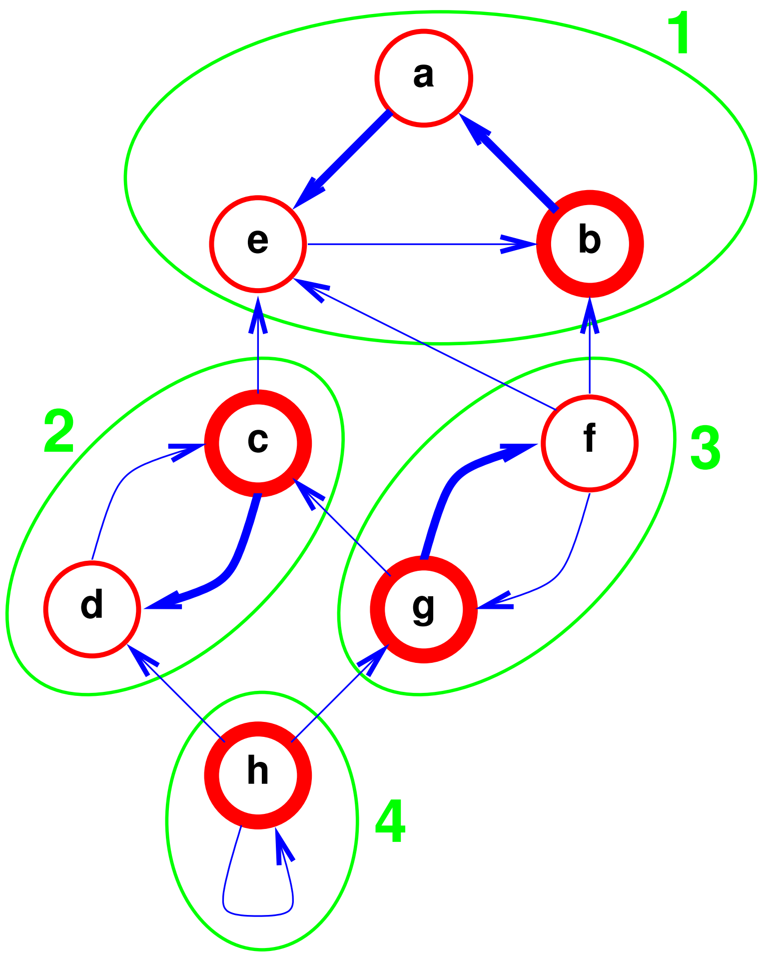
Recall



After DFS on G^T



Components



Satisfiability

If all the quantifiers are $\exists$, then we just have to determine whether it is possible to assign truth values to the vertices with the following properties:

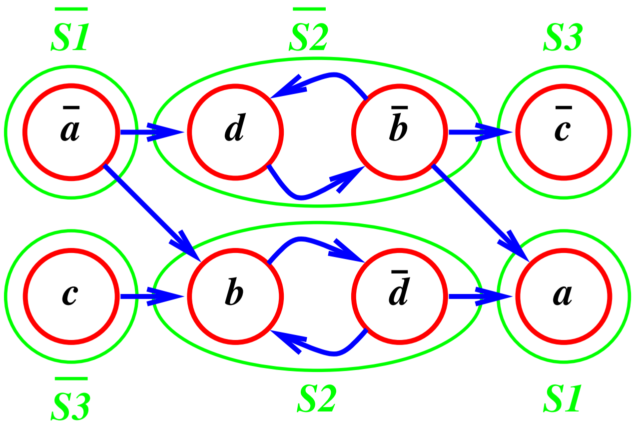
- Complementary vertices must be assigned compementary truth values,
- no edge $u \rightarrow v$ can have u assigned true and v assigned false.

Example (again)

If C is

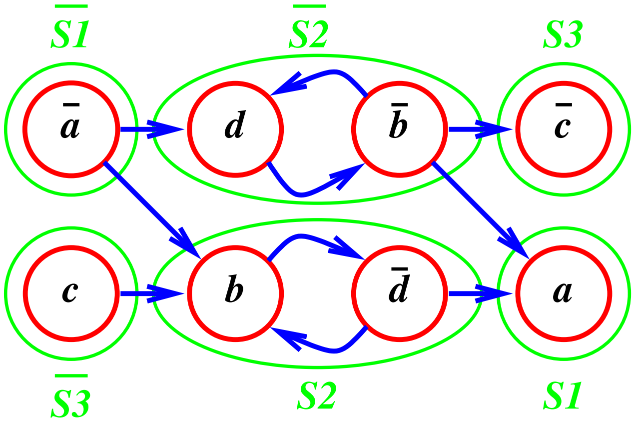
$$(a \vee b) \wedge (b \vee \bar{c}) \wedge (\bar{b} \vee \bar{d}) \wedge (b \vee d) \wedge (d \vee a)$$

The graph is:



C is satisfiable if and only if no vertex x is in the same strong component as $\bar{x}$.

Assignment Algorithm



Consider strongly connected components in reverse topological order:

$S1$	Mark $S1$ true and $\bar{S}1$ false,
$S2$	Mark $S2$ true and $\bar{S}2$ false,
$S3$	Mark $S3$ true and $\bar{S}3$ false,
$\bar{S}2$	Already marked,
$\bar{S}1$	Already marked,
$\bar{S}3$	Already marked.

General Case

We call a vertex universal if the corresponding variable is universally qualified, and existential otherwise.

A formula F is true if and only if none of the following conditions holds:

1. A vertex x is in the same strongly connected component as its complement $\bar{x}$.
2. An existential vertex x occurs in the same strongly connected component as a universally declared vertex y, with x declared before y. For example: $\cdots \exists x \cdots \forall y \cdots \quad C$.
3. There is a path from a universal vertex x to another universal vertex y. For example: $\cdots \forall x \cdots \forall y \cdots \quad C$.

The Algorithm

Initially all the strongly connected components are unmarked, but become marked true, false or contingent as the algorithm proceeds.

1) Let S be the next unmarked strongly connected component, chosen is reverse topological order. If there is no such S return "F is true". If S contains a variable x and its complement $\bar{x}$, return "F is false".

2) If S has some false or contingent successor,

   2.1) If S contains at least one universal vertex, return "F is false".

   2.2) If $\bar{S} \rightarrow \cdots \rightarrow S$, return "F is false". Mark S false and $\bar{S}$ true then goto (1).

3) // All successors, if any, are marked true. If S contains two or more universal vertices, return "F is false".

The Algorithm (cont.)

4) // All successors, if any, are marked true and
   // S contains less than two universal vertices.
   If S contains no universal vertices, mark S true and $\bar{S}$ false then goto (1).

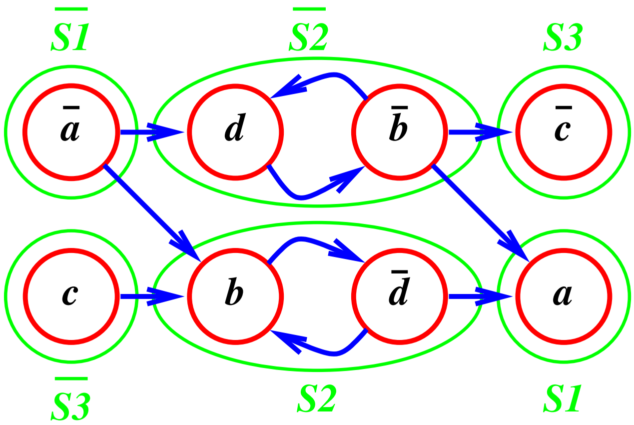
5) // S has just one universal vertex, y say.
   If S also contains an existential vertex x declared before the universal vertex y,
   (i.e. $\cdots \exists x \cdots \forall y \cdots C$), return "F is false".
   If $\bar{S} \rightarrow \cdots \rightarrow S$, return "F is false".
   Mark S and $\bar{S}$ contingent and goto (1).

Example (again)

If the formula is

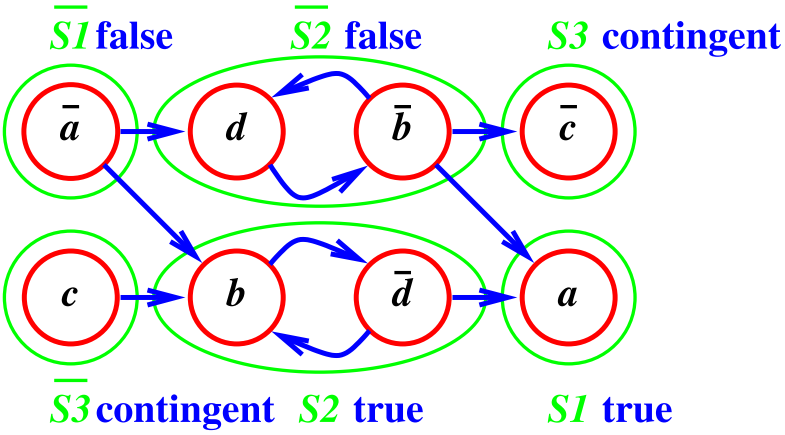
$$\exists a \exists b \forall c \exists d (a \vee b) \wedge (b \vee \bar{c}) \wedge (\bar{b} \vee \bar{d}) \wedge (b \vee d) \wedge (d \vee a)$$

The graph is:



Algorithm Trace

Formula: $\exists a \exists b \forall c \exists d \quad C$



Consider strongly connected components in reverse topological order:

$S1$	Mark $S1$ true and $\bar{S}1$ false,
$S2$	Mark $S2$ true and $\bar{S}2$ false,
$S3$	Mark $S3$ and $\bar{S}3$ contingent,
$\bar{S}2$, $\bar{S}1$, $\bar{S}3$	Already marked.

So the formula is true.