A sub-quadratic algorithm for conjunctive and disjunctive BESs

Jan Friso Groote[1,3] and Misa Keinänen[2]

[1]*Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

[2]*Dept. of Computer Science and Engineering, Lab. for Theoretical Comp. Science Helsinki University of Technology, P.O. Box 5400, FIN-02015 HUT, Finland*

[3]*CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*

J.F.Groote@tue.nl, Misa.Keinanen@hut.fi

Abstract

We present an algorithm for conjunctive and disjunctive Boolean equation systems (BESs), which arise frequently in the verification and analysis of finite state concurrent systems. In contrast to the previously best known $O(e^2)$ time solutions, our algorithm computes the solution of such a fixpoint equation system with size e and alternation depth d in $O(e \log d)$ time.

1 Introduction

A Boolean Equation System (BES) [1, 5, 6] is a sequence of boolean equations with minimal and maximal fixpoints. It gives a useful framework for the verification of finite state concurrent systems. This is due to the fact that many interesting properties of systems can naturally be specified in the modal μ-calculus [4]. The model checking problem says whether such a formula holds for a transition system. This problem, or more concretely a formula and a transition system, can be straightforwardly translated to a boolean equation system. So, a pleasant feature of a boolean equation system is that it gives a concise way of representing the model checking problem, laying bare the essential problem of computing the fixpoints.

We examine *conjunctive* and *disjunctive* fragments of boolean equation systems. Many practically relevant properties of systems can be expressed by means of fixed points that lead to boolean equations in either conjunctive or disjunctive forms. It

is therefore interesting to develop specific resolution techniques for these particular fragments.

All previous algorithms for solving conjunctive and disjunctive classes, including those from [2, 5], take at least quadratic time in the size of a boolean equation system in the worst case. For large boolean equations which are typically encountered in model checking and preorder/equivalence checking of realistic systems, these algorithms often lead to unacceptable running times. Up to now, it has been an open question whether or not the quadratic upper bound could be improved (e.g. see [2]).

The contribution of this paper is to resolve the question by presenting an especially fast algorithm for finding a solution for a boolean equation system in either conjunctive or disjunctive form. Given such a boolean equation system with size e and alternation depth d, our algorithm finds the solution using time $O(e \log d)$ in the worst case. This improves the best known upper bound and makes the verification of a large class of fixpoint expressions more tractable.

In this paper, we focus on devising the algorithm and giving the analysis of its asymptotic complexity. An evaluation of the performance on practical verification problems, and an empirical comparison between our approach and other related algorithms are left for future work.

Our algorithm combines essentially graph theoretic techniques for finding strong components [7, 8] and hierarchical clustering [9]. King, Kupferman and Vardi [3] recently found an algorithm in the realm of parity word automata. Their algorithm is very similar to ours and also resorts to the ideas in [9]. But their algoritm has a very different purpose, namely to decide the nonemptyness of parity word automata. Our approach differs from [3] in that we can employ a special structure of boolean equation systems, called *alternation depth* (see Def. 2.1), and our algorithm suggests even a slight improvement over [3].

This note is organised as follows. Section 2 introduces basic notions concerning boolean systems of equations. Section 3 formalises our idea of solving disjunctive (and conjunctive respectively) subclasses and presents the algorithm. Section 4 deals with the correctness and the complexity of the algorithm.

Acknowledgements. We would like to thank Moshe Vardi for pointing out to the results in [3] and Jaco van de Pol for useful comments. The financial supports of Academy of Finland (project 53695) and Helsinki Graduate School in Computer Science and Engineering are gratefully acknowledged.

2 Boolean equation systems

A Boolean equation system is an ordered sequence of fixed point equations of the form

$$(\sigma_1 x_1 = \alpha_1)(\sigma_2 x_2 = \alpha_2) \ldots (\sigma_n x_n = \alpha_n)$$

where all x_i are different. We generally use the letter $\mathcal{E}$ to represent a Boolean equation system, and let ϵ stand for the empty Boolean equation system. The symbol σ_i specifies the polarity of the fixed points. The symbol σ_i is μ if the i-th equation is a least fixed point equation and ν if it is a greatest fixed point equation. The order of equations in a Boolean equation system is very important, and we keep the order on variables and their indices in strict synchrony. We write $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ for the set of all Boolean variables. For each $1 \leq i \leq n$ we allow α_i to be a formula over Boolean variables and constants *false* and *true* and operators $\wedge$ and $\vee$, summarised by the grammar:

$$\alpha ::= \ \mathit{true} \mid \mathit{false} \mid x \in \mathcal{X} \mid \alpha_1 \wedge \alpha_2 \mid \alpha_1 \vee \alpha_2.$$

For non trivial Boolean equation systems (i.e. where the right hand side of interesting variables is not equal to *true* and *false*) *true* and *false* can be removed by simple propositional reasoning. Therefore, in the sequel, we assume that *true* and *false* do not occur in the Boolean equation systems that we consider.

Definition 2.1. Let

$$\mathcal{E} \equiv (\sigma_1 x_1 = \alpha_1)(\sigma_2 x_2 = \alpha_2) \ldots (\sigma_n x_n = \alpha_n)$$

be a Boolean equation system. The *alternation depth of* $\mathcal{E}$, notation $ad(\mathcal{E})$, is the number of variables x_j $(1 \leq j < n)$ such that $\sigma_j \neq \sigma_{j+1}$. An index j is a *ν-starting point of* $\mathcal{E}$ if $\sigma_j = \nu$, and either $j = 1$ or $\sigma_{j-1} = \mu$. If j is a ν-starting point then the *ν-segment of* j are those indices $j, j+1, \ldots, j+k$ such that $\sigma_{j+i} = \nu$ $(0 \leq i \leq k)$ and either $j + k = n$ or $\sigma_{j+k+1} = \mu$. Dual definitions can be given for μ.

Note that the alternation depth of a Boolean equation system is twice the number of ν-starting points of a Boolean equation system minus 0,1 or 2 depending on whether or not there are initial and trailing μ's. Furthermore, note that for each equation system $\mathcal{E}$ with variables from $\mathcal{X}$ we have that $ad(\mathcal{E}) \leq |\mathcal{X}|$. That is, the alternation depth of a Boolean equation system never exceeds the number of variables involved.

The semantics of Boolean equation systems provides a uniquely determined *solution*, to each Boolean equation system $\mathcal{E}$. A solution is a valuation assigning a constant value in $\{0, 1\}$ (with 0 standing for *false* and 1 for *true*) to all variables occurring in $\mathcal{E}$. Let $v, v_1, \ldots$ range over valuations, where each v is a function $v : \mathcal{X} \to \{0, 1\}$. We extend the definition of valuations to terms in the standard way. So, $v(\alpha)$ is the value of the term α after replacing each free variable x in α by $v(x)$. Let $v[x{:=}a]$ denote the valuation that coincides with v for all variables except x, which has the value a. We suppose that $[x{:=}a]$ has priority over all operations and $v[x{:=}a]$ stands for $(v[x{:=}a])$. Similarly, we apply $[x{:=}a]$ to terms; $\alpha[x{:=}a]$ indicates the term α where all occurrences of x have been replaced by a.

Definition 2.2 (*The solution of a Boolean equation system*). The solution of a Boolean equation system $\mathcal{E}$ relative to a valuation v, denoted by $[\![\mathcal{E}]\!]v$, is an assignment inductively defined by

$$[\![\epsilon]\!]v = v$$
$$[\![(\sigma_i x_i = \alpha_i)\mathcal{E}]\!]v = \begin{cases} [\![\mathcal{E}]\!]v[x_i{:=}MIN(x_i, \alpha_i, \mathcal{E}, v)] & \text{if } \sigma_i = \mu \\ [\![\mathcal{E}]\!]v[x_i{:=}MAX(x_i.\alpha_i, \mathcal{E}, v)] & \text{if } \sigma_i = \nu \end{cases}$$

where

$$MIN(x_i, \alpha_i, \mathcal{E}, v) = \bigwedge\{a | \alpha_i([\![\mathcal{E}]\!](v[x{:=}a])) \Rightarrow a\}$$
$$MAX(x_i, \alpha_i, \mathcal{E}, v) = \bigvee\{a | a \Rightarrow \alpha_i([\![\mathcal{E}]\!](v[x{:=}a]))\}.$$

As an example let us consider the following Boolean equation systems.

Example 2.3. Let $\mathcal{X}$ be the set $\{x_1, x_2, x_3\}$ and assume we are given a Boolean equation system

$$\mathcal{E}_1 \equiv ((\mu x_1 = x_1 \wedge x_2)(\mu x_2 = x_1 \vee x_3)(\mu x_3 = x_3)).$$

The system $\mathcal{E}_1$ is alternation-free, i.e. $ad(\mathcal{E}_1) = 0$, because it does not contain ν-starting points. The solution of $\mathcal{E}_1$ is given by the valuation $v : \mathcal{X} \to \{0, 1\}$ defined by $v(x_i) = 0$ for $i = 1, 2, 3$.

Example 2.4. Let $\mathcal{X}$ be the set $\{x_1, x_2, x_3\}$ and assume we are given a Boolean equation system

$$\mathcal{E}_2 \equiv ((\nu x_1 = x_2)(\mu x_2 = x_1 \vee x_3)(\nu x_3 = x_2 \wedge x_3)).$$

The system $\mathcal{E}_2$ is alternating, with alternation depth $ad(\mathcal{E}_2) = 2$. The solution of $\mathcal{E}_2$ is given by the valuation $v : \mathcal{X} \to \{0, 1\}$ defined by $v(x_i) = 1$ for $i = 1, 2, 3$.

So much for the preliminaries, which apply to all Boolean equation systems. We come now to the algorithm itself which refers to the disjunctive case only. The conjunctive case is fully dual and is therefore not treated explicitly.

3 The algorithm

We call a Boolean equation system *disjunctive* if no conjunction symbol ($\wedge$) appears in its right-hand side expressions. Given such a disjunctive system, we can view its variables as vertices of a graph and the dependencies between the variables as directed edges, obtaining another representation as defined below.

Definition 3.1. Let

$$\mathcal{E} \equiv (\sigma_1 x_1 = \alpha_1)(\sigma_2 x_2 = \alpha_2) \ldots (\sigma_n x_n = \alpha_n)$$

be a Boolean equation system. The *dependency graph* $G_{\mathcal{E}} = (V, E, \ell)$ *of* $\mathcal{E}$ is defined as follows

- $V = \{1, \ldots, n\}$,
- $E = \{\langle i, j\rangle \mid x_j \text{ occurs in } \alpha_i\}$,
- ℓ is a labelling function defined by $\ell(i) = \sigma_i$.

Definition 3.2. Let $G = (V, E, \ell)$ be a dependency graph and $k \in V$. We define the graph $G\restriction k = (V, E\restriction k, \ell)$ by taking

- $E\restriction k = \{\langle i, j\rangle \in E \mid i \geq k \text{ and } j \geq k\}$.

The following essential lemma comes from [2].

Lemma 3.3. Let $G_{\mathcal{E}} = (V, E, \ell)$ be the dependency graph of a disjunctive Boolean equation system $\mathcal{E}$. Let x_i be any variable in $\mathcal{E}$ and let valuation v be the solution of $\mathcal{E}$. Then the following are equivalent:

1. $v(x_i) = 1$
2. $\exists j \in V$ with $\ell(j) = \nu$ such that:
   (a) j is reachable from i, and
   (b) $G_{\mathcal{E}}$ contains a cycle of which the lowest index of a vertex on this cycle is j.

Finding cycles in graphs can be done in linear time using any algorithm to detect strongly connected components [7, 8]. A strongly connected component (SCC) in a graph $G = (V, E, \ell)$ is a set of vertices $W \subseteq V$ such that for each pair of vertices $k, l \in W$ it is possible to reach l from k by following directed edges in E. In the sequel we assume that all strongly connected components are maximal in the sense that there does not exist a larger set of vertices that is also a strongly connected component.

As we are generally interested in $v(x_1)$, we assume that all vertices in the dependency graph are reachable from vertex 1. So, the condition that needs to be checked is whether there is a cycle in $G_{\mathcal{E}}$ of which the lowest numbered vertex has label ν. The following algorithm performs this task efficiently. To apply the algorithm on boolean equation system with n equations, $\mathit{MinNuLoop}(k, n, G)$ must be executed where G is the dependency graph and k is the first ν-starting point. If such a starting point does not exist, $v(x_1) = 0$.

Algorithm. We define the algorithm $\mathit{MinNuLoop}(k_1, k_2, G)$ where k_1 and k_2 are indices such that $k_1 \leq k_2$, $G = (V, E, \ell)$ is a dependency graph, $\ell(k_1) = \nu$ and $|E| \geq |V|$. The algorithm *MinNuLoop* calculates whether there is an index k with $k_1 \leq k \leq k_3$, $\ell(k) = \nu$ and k is the smallest vertex on some cycle of G. The algorithm consists of the following steps:

1. Let s be the number of ν-starting points on $k_1, \ldots, k_2$. Let k_3 be the index of the $\lceil \frac{1}{2}s \rceil$-th ν-starting point on $k_1, \ldots, k_2$. Calculate the strongly connected components of $G \restriction k_3$. Let $C(k)$ represent the strongly connected component containing k. An SCC is called trivial if it consists of one vertex and has no self-loop. Check whether any vertex on the ν-segment of k_3 resides in a non-trivial strongly connected component. If so, report 'found' and stop.

2. Here and in 5 below we check vertices in the range $k_1, \ldots, k_3 - 2$. Calculate the graph $G' = (V', E', \ell')$ by

$$
\begin{array}{l}
V' = \{C(i) \mid i \in V \text{ and } \exists j.\langle i, j\rangle \in E \text{ and } C(i) \neq C(j)\}, \\
E' = \{\langle C(i), C(j)\rangle \in V' \times V' \mid \langle i, j\rangle \in E,\ C(i) \neq C(j)\} \text{ and} \\
\ell'(i) = \begin{cases} \ell(i) & \text{if } C(i) \text{ trivial,} \\ \mu & \text{otherwise.} \end{cases}
\end{array}
$$

3. Let k_4 be the smallest index of a ν-starting point larger than k_3. We check vertices in the range $k_4, \ldots, k_2$ (see also item 6). Calculate the graph $G'' = (V'', E'', \ell)$ by

$$
\begin{array}{l}
V'' = \{i \in V \mid C(i) \text{ is not trivial}\} \text{ and} \\
E'' = \{\langle i, j\rangle \in V'' \times V'' \mid \langle i, j\rangle \in E \text{ and } C(i) = C(j)\}.
\end{array}
$$

4. Forget G.

5. If $k_1 \leq k_3 - 2$, execute $\mathit{MinNuLoop}(k_1, k_3 - 2, G')$.

6. If $k_4 \leq k_2$, execute $\mathit{MinNuLoop}(k_4, k_2, G'')$.

The algorithm stops reporting 'found' iff a loop with a minimal ν-labelled vertex exists, or in other words iff variable x_1 holds in the boolean equation system.

4 Correctness and complexity

The correctness of the algorithm can be seen as follows. In step 1 it is straightforwardly checked whether any vertex in the ν-segment of k_3 is the smallest ν-labelled vertex on a cycle.

When investigating whether any of the vertices in the range $k_1, \ldots, k_3 - 2$ (vertex $k_3 - 1$ is μ-labelled) is the smallest ν-labelled node on a cycle, all strongly connected components calculated in step 1 occur in G' and can therefore be safely collapsed. Furthermore, nodes $C(i)$ in V' without outgoing edges cannot contribute to cycles, and can therefore be removed. Note that as all nodes without outgoing edges are removed from G', the precondition that $|E'| > |V'|$ to invoke *MinNuLoop* is met.

When investigating whether any vertex in the range $k_4, \ldots, k_2$ is the smallest, only nodes already on a cycle in $G \restriction k_3$ can contribute. Therefore, in G'' only those edges need to be considered.

The time complexity of the algorithm *MinNuLoop*(k_1, k_2, G) is $O(|E| \log A)$ where $G = (V, E, \ell)$ and A is the number of ν-starting points on $k_1, \ldots, k_2$. As noted elsewhere in this note $2A$ is approximately the alternation depth of the Boolean equation system that corresponds to G.

The time complexity has a nice justification. In step 1 of *MinNuLoop* it takes $O(|V|) \leq O(|E|)$ to determine k_3. Calculating $G{\restriction}k_3$, the strongly connected components and checking whether any vertex on the ν-segment of k_3 resides on a non-trivial strongly connected component requires $O(|E|)$ time.

In step 2, 3 and 4 the graphs G' and G'' are constructed to replace G. This can clearly be done in time $O(|E|)$.

A crucial observation is that for each edge $\langle i, j\rangle \in E$ at most one edge shows up in either E' or E'', depending on whether $C(i) = C(j)$. This means that $|E'|+|E''| \leq |E|$. Furthermore, if the number of ν-starting points in $k_1, \ldots, k_2$ is A, then there are at most $\frac{1}{2}A$ ν-starting points in both $k_1, \ldots, k_3-1$ and $k_4, \ldots, k_2$. So *MinNuLoop*(k_1, k_3-1, G') has time complexity $O(|E'| \log \frac{1}{2}A)$ and *MinNuLoop*(k_4, k_2, G'') has time complexity $O(|E''| \log \frac{1}{2}A)$. So, the time complexity of *MinNuLoop*(k_1, k_2, G) is

$$O(|E|) + O(|E'| \log \frac{1}{2}A) + O(|E''| \log \frac{1}{2}A) \leq O(|E| + |E| \log \frac{1}{2}A) = O(|E| \log A).$$

The time complexity for solving a boolean equation system also contains the generation of the dependency graph, and is easily seen to be $O(e \log d)$ where e is the size of the boolean equation system and d the alternation depth.

The space complexity of *MinNuLoop*(k_1, k_2, G) is $O(|E|)$. In order to see this it suffices to note that the graphs constructed in step 2 and 3 are together smaller than the graph G, which is thrown away in step 4. So, the memory footage is only reduced while executing the algorithm. As generating the dependency graph also takes linear space, solving a disjunctive BES also takes linear space.

A note about the alternation depth. We have taken a definition of alternation depth based on the sequential occurrences of μ's and ν's in a Boolean equation system. A definition of alternation depth that abstracts from the syntactical appearance can be found in Definition 3.34 of [5]. The idea is that to determine the alternation depth only chains of equations in a Boolean equation system must be followed that depend on each other. Using for instance Lemma 3.22 of [5] a Boolean equation system can be reordered such that our notion of alternation depth and the notion of [5] coincide.

A note about the open question in [2]. In [2] the following open question was stated. Given a directed graph of which the vertices are ordered and labelled with either red or green. Is there a sub quadratic algorithm to determine whether the graph contains a cycle of which the smallest vertex has label green? The algorithm presented here provides such an efficient algorithm.

Disjunctive and conjunctive straight BESs. In [2] so called disjunctive/conjunctive

straight BESs were introduced. These are BESs where variables with conjunction symbols in their right hand side do not mutually depend on variables with disjunctions in their right hand side. By partitioning such a BES our algorithm can be straightforwardly adapted to be applicable to this case also without loss of time or space complexity.

References

[1] H.R. Andersen. Model checking and boolean graphs. Theoretical Computer Science, 126 (1994) 3-30.

[2] J.F. Groote and M.K. Keinänen. Solving Disjunctive/Conjunctive Boolean Equation Systems with Alternating Fixed Points. In K. Jensen and A. Podelski, editors, Proc. 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2004), volume 2988 of Lecture Notes in Computer Science, pages 436-450. Springer, 2004.

[3] V. King, O. Kupferman and M.Y. Vardi. On the complexity of parity word automata. Proc. of 4th International Conference on Foundations of Software Science and Computation Structures, volume 2030 of Lecture Notes in Computer Science, pages 276–286. Springer, 2001.

[4] D. Kozen. Results on the propositional μ-calculus. Theoretical computer Science 27 (1983) 333-354.

[5] A. Mader. Verification of Modal Properties using Boolean Equation Systems. PhD thesis, Technical University of Munich, 1997.

[6] R. Mateescu. A Generic On-the-Fly Solver for Alternation-Free Boolean Equation Systems. In Proceedings of Conf. on Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science 2619 (Springer Verlag, 2003) 81-96.

[7] M. Sharir. A strong-connectivity algorithm and its application in data flow analysis. Computers and Mathematics with Applications 7(1):67-72, 1981.

[8] R.E. Tarjan. Depth first search and linear graph algorithms. SIAM Journal of Computing. 1(2):146-160, 1972.

[9] R.E. Tarjan. A hierarchical clustering algorithm using strong components. Information Processing Letters, 14(1):26-29, 1982.