Centrum voor Wiskunde en Informatica

REPORTRAPPORT

SEN

Software Engineering

Software ENgineering

SEN

Solving disjunctive/conjunctive boolean equation systems with alternating fixed points

Jan Friso Groote, Misa Keinänen

REPORT SEN-R0310 DECEMBER 9, 2003

CWI is the National Research Institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organization for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-369X

Solving disjunctive/conjunctive boolean equation systems with alternating fixed points

ABSTRACT

This paper presents a technique for the resolution of alternating disjunctive/conjunctive boolean equation systems. The technique can be used to solve various verification problems on finite-state concurrent systems, by encoding the problems as boolean equation systems and determining their local solutions. The main contribution of this paper is that a recent resolution technique from [13] for disjunctive/conjunctive boolean equation systems is extended to the more general disjunctive/conjunctive forms with alternation. Our technique has the time complexity $O(m+n^2)$, where m is the number of alternation free variables occurring in the equation system and n the number of alternating variables. We found that many μ-calculus formulas with alternating fixed points occurring in the literature can be encoded as boolean equation systems of disjunctive/conjunctive forms. Practical experiments show that we can verify alternating formulas on state spaces that are orders of magnitudes larger than reported up till now.

2000 Mathematics Subject Classification: 68Q60; 68Q85
1998 ACM Computing Classification System: D.2.4; F.2.2
Keywords and Phrases: Boolean Equation System; Algorithm; Disjunctive; Conjunctive

Solving disjunctive/conjunctive boolean equation systems with alternating fixed points

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

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

2. *Departement of Mathematics and Computer Science, Eindhoven University*
of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

3. *CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*
`J.F.Groote@tue.nl`, `Misa.Keinanen@hut.fi`

Abstract

This paper presents a technique for the resolution of alternating disjunctive/conjunctive boolean equation systems. The technique can be used to solve various verification problems on finite-state concurrent systems, by encoding the problems as boolean equation systems and determining their local solutions. The main contribution of this paper is that a recent resolution technique from [13] for disjunctive/conjunctive boolean equation systems is extended to the more general disjunctive/conjunctive forms with alternation. Our technique has the time complexity $O(m+n^2)$, where m is the number of alternation free variables occurring in the equation system and n the number of alternating variables. We found that many μ-calculus formulas with alternating fixed points occurring in the literature can be encoded as boolean equation systems of disjunctive/conjunctive forms. Practical experiments show that we can verify alternating formulas on state spaces that are orders of magnitudes larger than reported up till now.

2000 Mathematics Subject Classification: 68Q60; 68Q85
1998 ACM Computing Classification System: D.2.4; F.2.2
Keywords and Phrases: Boolean Equation System; Algorithm; Disjunctive; Conjunctive

1 Introduction

Modal μ-calculus [10] is an expressive logic for system verification, and most model checking logics can be encoded in the μ-calculus. Many important features of system models, like equivalence/preorder relations and fairness constraints, can be expressed with the logic, also. For these reasons, μ-calculus is a logic widely studied in the recent systems verification literature.

It is well-known that the μ-calculus model checking problem is in the complexity class NP $\cap$ co-NP. Emerson, Jutla, and Sistla [7, 8] showed the problem can be reduced to determining the winner in a parity game, and thus is in NP (and also by symmetry in co-NP). More recently, Jurdzinsky [9] showed that the problem is even in UP $\cap$ co-UP. Yet the complexity of μ-calculus model checking problem for the unrestricted logic is an open problem; no polynomial algorithm has been discovered.

[*] The work was supported by Academy of Finland (project 53695), Emil Aaltonen foundation and Helsinki Graduate School in Computer Science and Engineering.

Nevertheless, various effective model checking algorithms exist for expressive subsets. Arnold and Crubille [2] presented an algorithm for checking alternation depth 1 formulas of μ-calculus, which is linear in the size of the model and quadratic in the size of the formula. Cleaveland and Steffen [6] improved the result by making the algorithm linear also in the size of the formula. Andersen [1], and similarly Vergauwen and Lewi [16], showed how model checking alternation depth 1 formulas amounts to the evaluation of *boolean graphs*, resulting also in linear time techniques for model checking alternation depth 1 formulas. Even more expressive subsets of μ-calculus were investigated by Bhat and Cleaveland [5] as well as Emerson et al. [7, 8]. They presented polynomial time model checking algorithms for fragments L1 and L2, which may contain alternating fixed point formulas.

In this paper, instead of treating μ-calculus expressions together with their semantics, we prefer to work with the more flexible formalism of boolean equation systems [1, 12, 13, 17]. Boolean equation systems provide a useful framework for studying verification problems of finite-state concurrent systems, because μ-calculus expressions can be easily be translated into this simple formalism (see e.g. [3, 12, 13] for such translations).

We restrict the attention to boolean equation systems, which are either in disjunctive or in conjunctive form. We found that many practically relevant μ-calculus formulas (actually virtually all of them) can be encoded as boolean equation systems that are disjunctive, conjunctive, or disjunctive/conjunctive straight (see definition 3.2). For instance, the model checking problems for Hennessy-Milner logic (HML), Computation Tree Logic (CTL), and many equivalence/preorder checking problems result in alternation-free boolean equation systems in disjunctive/conjunctive forms (see for instance [13]). Moreover, encoding the L1 fragment of the μ-calculus (and similar subsets) or many fairness constraints as boolean equation systems result in alternating systems which are in disjunctive/conjunctive form.

Hence, the problem of solving disjunctive/conjunctive boolean equation systems with alternating fixed points is so important that developing special purpose solution techniques for these classes is worthwhile. Recently, the question has been addressed by Mateescu [13], who presented a resolution algorithm for disjunctive/conjunctive boolean equation systems. But, this approach is restricted to alternation-free systems. We are only aware of one sketch of an algorithm that is directed to alternating disjunctive/conjunctive boolean equation systems (proposition 6.5 and 6.6 of [12]). Here a $O(n^3)$[1] time and $O(n^2)$ space algorithm is provided where n is the number of variables. Our algorithm is a substantial improvement over this.

In this paper, we address the problem of solving alternating disjunctive/conjunctive straight boolean equation systems. The algorithm for the resolution of such equation systems is quite straightforward comparable to the alternation-free case presented in [13]. Essentially, the idea consists of computing simple kinds of dependencies between certain variables occurring in the equation systems. Our technique is such that it ensures linear-time worst case complexity of solving alternation-free boolean equation systems, and quadratic for the alternating systems. More precisely, we present resolution algorithms for the disjunctive/conjunctive classes which are of complexity $O(m+n^2)$, where m is the number of alternation-free variables and n the number of alternating variables occurring in the system. Hence, our approach preserves the best known worst case time complexity of model checking of many restricted but expressive fragments of the μ-calculus.

The paper is organized as follows. Section 2 introduces basic notions concerning boolean equation systems. Section 3 introduces the subclasses of disjunctive, conjunctive and disjunctive/conjunctive straight boolean equation systems and illustrates that many formulas with alternating fixed points fall into these classes. Section 4 presents the algorithm and section 5 provides some initial experimental results. In section 6 we wrap up and provide an open problem that we were unable to solve, but which – if solved – would eliminate the quadratic factor in the time complexity of our algorithm.

[1] This paper claims an $O(n^2)$ time algorithm, assuming the existence of an algorithm which allows union of (large) sets, and finding and deletion of elements in these in constant time. To our knowledge for this only a linear and most certainly no constant time algorithm exists.

2 Boolean equation systems

We give here a short introduction into boolean equation systems. A boolean equation system is an ordered sequence of fixed point equations like

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

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 $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$ 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 $\vee$, summarized by the grammar:

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

We write $x_i \in \alpha_j$ if x_i is a subterm of α_j.

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 $v, v_1, \dots$ 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 substituting 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 2.1. *(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

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

where $\mu x_i.\alpha([\![\mathcal{E}]\!]v) = \bigwedge\{a | \alpha_i([\![\mathcal{E}]\!]v[x{:=}a]) \Rightarrow a\}$ and $\nu x_i.\alpha([\![\mathcal{E}]\!]v) = \bigvee\{a | a \Rightarrow \alpha_i([\![\mathcal{E}]\!]v[x{:=}a])\}$.

It is said that a variable x_i *depends on* variable x_j, if α_i contains a reference to x_j, or to a variable x_k such that x_k depends on x_j. Two variables x_i and x_j are mutually dependent if x_i depends on x_j and vice versa.

A boolean equation system $\mathcal{E}$ is *alternation free* if, for any two variables x_i and x_j occurring in $\mathcal{E}$, x_i and x_j are mutually dependent implies $\sigma_i = \sigma_j$. Otherwise, system $\mathcal{E}$ is said to be *alternating* and it contains *alternating fixed points.*

Example 2.2. Let $\mathcal{X}$ be the set $\{x_1, x_2, x_3\}$ and assume we are given a boolean equation system

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

The system $\mathcal{E}_1$ is alternation-free, because it does not contain mutually dependent variables with different signs. Yet, note that variable x_3 with sign $\sigma_3 = \nu$ depends on variables x_1 and x_2 with different sign. A 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 2.3. Let $\mathcal{X}$ be the set $\{x_1, x_2, x_3\}$ and assume we are given a boolean equation system

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

The system $\mathcal{E}_2$ is alternating, because it contains mutually dependent variables with different signs, like x_1 and x_2 with $\sigma_1 \neq \sigma_2$. A solution of $\mathcal{E}_2$ is given by the valuation $v : \mathcal{X} \rightarrow \{0, 1\}$ defined by $v(x_i) = 1$ for $i = 1, 2, 3$.

In Mader [12] there are two lemmas that allow to solve boolean equation systems. As our proofs are based on these, we restate these here.

Lemma 2.4. *(Lemma 6.2 of [12]). Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be boolean equation systems and let $\sigma x = \alpha$ and $\sigma x = \alpha'$ be boolean equations where*

$$\alpha' = \begin{cases} \alpha[x{:=}\mathit{true}] & \text{if } \sigma = \nu, \\ \alpha[x{:=}\mathit{false}] & \text{if } \sigma = \mu. \end{cases}$$

Then $[\![\mathcal{E}_1(\sigma x = \alpha)\mathcal{E}_2]\!]v = [\![\mathcal{E}_1(\sigma x = \alpha')\mathcal{E}_2]\!]v$.

Lemma 2.5. *(Lemma 6.3 of [12]). Let $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$ be boolean equation systems and let $\sigma_1 x_1 = \alpha$, $\sigma_1 x_1 = \alpha'$ and $\sigma_2 x_2 = \beta$ be boolean equations where $\alpha' = \alpha[x_2{:=}\beta]$. Then*

$$[\![\mathcal{E}_1(\sigma_1 x_1 = \alpha)\mathcal{E}_2(\sigma_2 x_2 = \beta)\mathcal{E}_3]\!]v = [\![\mathcal{E}_1(\sigma_1 x_1 = \alpha')\mathcal{E}_2(\sigma_2 x_2 = \beta)\mathcal{E}_3]\!]v.$$

3 Disjunctive/conjunctive boolean equation systems

We introduce disjunctive/conjunctive form boolean equation systems in their most elementary form

Definition 3.1. Let $\sigma x = \alpha$ be a fixed point equation. We call this equation *disjunctive* if no conjunction symbol ($\wedge$) appears in α, and we call it *conjunctive* if no disjunction ($\vee$) symbol appears in α. Let $\mathcal{E}$ be a boolean equation system. We call $\mathcal{E}$ *conjunctive* (respectively *disjunctive*) iff each equation in $\mathcal{E}$ is conjunctive (respectively disjunctive).

But our algorithm applies to a much wider class of equation systems, namely those where the conjunction and disjunction symbol are not used in a nested way

Definition 3.2. Let $\mathcal{E}$ be a boolean equation system. We call $\mathcal{E}$ *disjunction/conjunction straight* (*DCS*) iff for all variables x_i and x_j in $\mathcal{E}$ that are mutually dependent, the equations $\sigma_i x_i = \alpha_i$ and $\sigma_j x_j = \alpha_j$ in $\mathcal{E}$ are both conjunctive or both disjunctive.

Observation I. The problem of solving disjunction/conjunctive straight boolean equation systems can be reduced to iteratively dealing with disjunctive or conjunctive boolean equation systems as follows. In a DCS boolean equation system the variables can be partitioned in blocks such that variables that mutually depend on each other belong to the same block. The dependency relation among variables can be extended to blocks in the sense that block B_i depends on block B_j if some variable $x_i \in B_i$ depends on some variable $x_j \in B_j$. This dependency relation is an ordering. We can start to find solutions for the variables in the last block, setting them to *true* or *false*. Using lemma 2.5 we can substitute the solutions for variables in blocks higher up in the ordering.

The following simplification rules can be used to simplify the equations

- $(\phi \wedge \mathit{true}) \mapsto \phi$
- $(\phi \wedge \mathit{false}) \mapsto \mathit{false}$
- $(\phi \vee \mathit{true}) \mapsto \mathit{true}$
- $(\phi \vee \mathit{false}) \mapsto \phi$

and the resulting equation system has the same solution. The rules allow to remove each occurrence of *true* and *false* in the right hand side of equations, except if the right hand side becomes equal to *true* or *false*, in which case yet another equation has been solved. By recursively applying these steps all non trivial occurrences of *true* and *false* can be removed from the equations and we call the resulting equations *purely disjunctive* or *purely conjunctive*.

Note that each substitution and simplification step reduces the number of occurrences of variables or the size of a right hand side, and therefore, only a linear number of such reductions are applicable.

After solving all equations in a block, and simplifying subsequent blocks the algorithm can be applied to the blocks higher up in the ordering iteratively solving them all.

Note that this allows us to restrict our attention to algorithms to solve purely disjunctive/conjunctive straight systems.

Example 3.3. Consider the boolean equation system $\mathcal{E}_2$ of example 2.3. The system $\mathcal{E}_2$ is not in conjunctive form. An equivalent conjunctive equation system $\mathcal{E}_3$ is obtained by replacing α_3 of $\mathcal{E}_2$ with *true* and propagating $x_3 = \mathit{true}$ throughout the formula using lemma 2.5. This results in the following sequence

$$\mathcal{E}_3 = ((\nu x_1 = x_2 \wedge x_1)(\mu x_2 = x_1)(\nu x_3 = \mathit{true}))$$

within which no disjunctions occur in right-hand sides of equations.

Observation II. We found that many formulas with apparently alternating fixed points lead to boolean equation systems that are disjunction/conjunction straight and therefore can be solved efficiently with our techniques.

Consider for instance the examples in section 3.5 in [4]. All formulas applied to any labelled transition systems yield disjunction/conjunction straight boolean equation systems, except for the modal formula

$$\mu Y.vZ.(P \wedge [a]Y) \vee (\neg P \wedge [a]Z).$$

But this formula is equivalent to the formula

$$\mu Y.(([a]Y \vee \nu Z.(\neg P \wedge [a]Z)))$$

which does lead to DCS equation systems.

As an illustration we explain the transformation of the last formula in the example section of [4]:

$$\nu X.\mu Y.\nu Z.[a]X \wedge (\langle a\rangle \mathit{true} \Rightarrow [-a]Y) \wedge [-a]Z.$$

If we consider a labeled transition system $M = (S, A, \longrightarrow)$ then the boolean equation system looks like:

$$\left.\begin{array}{l}\nu\, x_s = y_s\\ \mu\, y_s = z_s\\ \nu\, z_s = \bigwedge\limits_{s'\in\nabla(a,s)} x_{s'} \wedge (\bigwedge\limits_{s'\in\nabla(a,s)} \mathit{false} \vee \bigwedge\limits_{s'\in\nabla(\neg a,s)} y_{s'}) \wedge \bigwedge\limits_{s'\in\nabla(\neg a,s)} z_{s'}\end{array}\right\}\text{for all } s \in S.$$

Here, $\nabla(a, s) := \{s' | s \xrightarrow{a} s'\}$ and $\nabla(\neg a, s) := \{s' | s \xrightarrow{b} s' \text{ and } b \neq a\}$. On first sight these equations do not appear to be a conjunctive boolean equation system, as in the third group of equations a disjunction occurs. However, for each concrete labelled transition system the left side of this disjunction will either become true or false for each state $s \in S$. By applying the simplification rules the formula quickly becomes conjunctive.

4 The algorithm

We develop our resolution algorithm in terms of a *variable dependency graph* similar to those of *boolean graphs* [1], which provide a representation of the dependencies between variables occurring in equation systems.

Definition 4.1. *(Variable dependency graph).* Let $\mathcal{E} = ((\sigma_1 x_1 = \alpha_1)(\sigma_2 x_2 = \alpha_2) \dots (\sigma_n x_n = \alpha_n))$ be a disjunctive/conjunctive boolean equation system. The *dependency graph* of $\mathcal{E}$ is a triple $G_{\mathcal{E}} = (V, E, L)$ where

- $V = \{i \mid 1 \leq i \leq n\} \cup \{\bot, \top\}$ is the set of nodes
- $E \subseteq V \times V$ is the set of edges such that for all equations $\sigma_i\, x_i = \alpha_i$
  - $(i, j) \in E$, if a variable $x_j \in \alpha_i$
  - $(i, \bot) \in E$, if *false* occurs in α_i
  - $(i, \top) \in E$, if *true* occurs in α_i
  - $(\bot, \bot), (\top, \top) \in E$
- $L : V \to \{\mu, \nu\}$ is the node labeling defined by $L(i) = \sigma_i$ for $1 \leq i \leq n$, $L(\bot) = \mu$, and $L(\top) = \nu$.

Observe that in the definition above the sink nodes with self-loops, $\bot$ and $\top$, represent the constants *false* and *true*. The ordering on nodes (given by their sequence number) is extended to $\bot$ and $\top$ by putting them highest in the ordering.

The key idea of our technique is based on the following observation that to obtain local solutions of variables in disjunctive/conjunctive equation systems, it suffices to compute the existence of a cycle in the dependency graph with certain properties.

Lemma 4.2. *Let $G_{\mathcal{E}} = (V, E, L)$ be the dependency graph of a disjunctive (respectively conjunctive) 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. *$v(x_i) = 1$ (respectively $v(x_i) = 0$)*
2. *$\exists j \in V$ with $L(j) = \nu$ (respectively $L(j) = \mu$) such that:*
   (a) *j is reachable from i, and*
   (b) *$G_{\mathcal{E}}$ contains a cycle of which the lowest index of a node on this cycle is j.*

Proof. We only prove this lemma for disjunctive boolean equation systems. The case for conjunctive equation systems is dual and goes in the same way. First we show that (2) implies (1). If j lies on a cycle with all nodes with numbers larger than j, there are two possibilities. Either j equals $\top$ or $1 \leq j \leq n$. In the last case, there is a sub-equation system of $\mathcal{E}$ that looks as follows:

$$\begin{array}{lcl} \nu x_j & = & \alpha_j \\ & \vdots & \\ \sigma_{k_1} y_{k_1} & = & \alpha_{k_1} \\ \sigma_{k_2} y_{k_2} & = & \alpha_{k_2} \\ & \vdots & \\ \sigma_{k_n} y_{k_n} & = & \alpha_{k_n} \end{array}$$

where $x_j \in \alpha_j[y_{k_1} := \alpha_{k_1}][y_{k_2} := \alpha_{k_2}][y_{k_3} := \alpha_{k_3}] \dots [y_n := \alpha_{k_n}]$. Using lemma 2.5 we can rewrite the boolean equation system $\mathcal{E}$ to an equivalent one by replacing the equation $\nu x_j = \alpha_j$ by:

$$\nu x_j = \alpha_j[y_{k_1} := \alpha_{k_1}][y_{k_2} := \alpha_{k_2}][y_{k_3} := \alpha_{k_3}] \dots [y_n := \alpha_{k_n}].$$

Now note that the right hand side contains only disjunctions and the variable x_j at least once. Hence, by lemma 2.5 the equation reduces to

$$\nu x_j = \mathit{true}.$$

Now, as x_j is reachable from x_i, the equation $\sigma_i x_i = \alpha_i$ can similarly be replaced by $\sigma_i x_i = \mathit{true}$. Hence, for any solution v of $\mathcal{E}$, it holds that $v(x_i) = 1$. In case j equals $\top$, the term *true* is reachable from x_i. In a similar way using lemma 2.5 we can replace $\sigma_i x_i = \alpha_i$ by $\sigma_i x_i = \mathit{true}$.

Now we prove that (1) implies (2) by contraposition. So, assume that there is no j with $L(j) = v$ that is reachable from i such that j is on a cycle with only higher numbered nodes.

We prove with induction on $n - k$ that $\mathcal{E}$ is equivalent to the same boolean equation system where equations $\sigma_{k+1}x_{k+1} = \alpha_{k+1}, \ldots, \sigma_n x_n = \alpha_n$ that are reachable from x_i, have been replaced by $\sigma_{k+1}x_{k+1} = \beta_{k+1}, \ldots, \sigma_n x_n = \beta_n$ where all β_l are disjunctions of *false* and variables that stem from $x_1, \ldots, x_k$. If the inductive proof is finished, the lemma is also proven: consider the case where $n - k = n$. This says that $\mathcal{E}$ is equivalent to a boolean equation system where all right hand sides of equations reachable from x_i are equal to *false*. So, in particular $x_i = \mathit{false}$, or in other words, for every solution v of $\mathcal{E}$ it holds that $v(x_i) = 0$.

For $n - k = 0$ the induction hypothesis obviously holds. In particular *true* cannot occur in the right hand side of any equation reachable from x_i. So, consider some $n - k$ for which the induction hypothesis holds. We show that it also holds for $n - k + 1$. So, we must show if equation $\sigma_k x_k = \alpha_k$ is reachable from x_i, it can be replaced by an equation $\sigma_k x_k = \beta_k$ where in β_k only variables chosen from $x_1, \ldots, x_{k-1}$ and *false* can occur.

As x_k is reachable from x_i, all variables x_l occuring in α_k are also reachable from x_i. By the induction hypothesis the equations $\sigma_l x_l = \alpha_l$ for $l > k$ have been replaced by $\sigma_l x_l = \beta_l$ where in β_l only *false* and variables from $x_1, \ldots, x_k$ occur. Using lemma 2.5 such variables x_l can be replaced by β_l and hence, α_k is replaced by γ_k in which *false* and variables from $x_1, \ldots, x_k$ can occur.

What remains to be done is to remove x_k from γ_k assuming $x_k \in \gamma_k$. This can be done as follows. Suppose σ_k equals ν, then, as x_k occurs in γ_k, there must be a path in the dependency graph to a node $x_{l'}$ with $l' > k$ such that $x_k \in \alpha_{l'}$. But this means that the dependency graph has a cycle on which k is the lowest value. This contradicts the assumption. So, it cannot be that $\sigma_k = \nu$, so, $\sigma_k = \mu$. Now using lemma 2.4 the variable x_k in α_k can be replaced by *false* and subsequently be eliminated. This finalizes the induction step of the proof. □

Now consider a disjunctive/conjunctive straight boolean equation system $\mathcal{E}$. In order to find a solution for $\mathcal{E}$ we first partition the set of variables $\mathcal{X}$ into blocks such that variables are in the same block iff these are mutually dependent. As $\mathcal{E}$ is disjunctive/conjunctive straight, all variables in each block have defining equations that are either disjunctive or conjunctive. Using the well known algorithm [15] for the detection of strongly connected components, the partition can be constructed in linear time on the basis of the variable dependency graph. As argued earlier, the equations belonging to the variables in each block can be solved iteratively. If the variables in a block do not depend on unsolved variables, the equations in this block can be solved. So, we only have to concentrate on solving disjunctive or conjunctive equations belonging to variables in a single block.

So, we present here an algorithm to solve a disjunctive boolean equation system. The conjunctive case is dual and goes along exactly the same lines. Our algorithm is an extension of Tarjan's [15] algorithm to detect strongly connected components. It is given in figure 1 and explained below.

We assume that the boolean equation system has already been transformed into a variable dependency graph $G = (V, E, L)$. There are two main functions *solve* and *find*. The function *solve* takes the index i of a variable x_i of interest and solves it by reporting it to be either 0 or 1. The procedure *find*(k) constructs all the strongly connected components from node k and applies lemma 4.2 to them.

We use a standard adjacency-list representation and keep an array of lists of nodes. We assume that an array *sign* is given that indicates the label for each node. I.e. *sign*$[i] = \mathit{true}$ if the label of node i is ν, and *sign*$[i] = \mathit{false}$ if the label of node i is μ. We keep an integer array *value*, initially set to

```
int find(int k)
    if ($sign[k] = 1 \wedge$ adjacency list of $k$ contains $k$)
        report $x_i$ gets value 1; stop;
    $id := id + 1$;  $value[k] := id$;
    $min := id$;
    $stack[p] := k$;  $p := p + 1$;
    for all nodes $t$ in the adjacency list of $k$ do
        if ($value[t] = 0$)
            $m := find(t)$;
        else $m := value[t]$;
        if ($m < min$)
            $min := m$;
    od
    if ($min = value[k]$)
        $mu := false$;  $nu := false$;
        $S := \emptyset$;
        while ($stack[p] \neq k$) do
            $p := p - 1$;  $n := stack[p]$;
            if ($sign[n] = true$)
                $nu := true$;
            else $mu := true$;
            $S := S \cup \{n\}$;
        od
    if ($|S| > 1 \wedge mu = false$)
        report $x_i$ gets value 1; stop;
    if ($|S| > 1 \wedge mu = true \wedge nu = true$)
        for (all nodes $j$ in $S$ with $sign[j] = true$) do
            if ($cycle(G, S, j) = true$)
                report $x_i$ gets value 1; stop;
        od
return $min$;


void solve(int i)
    $find(i)$;
    if ($x_i$ is not yet reported 1)
        report $x_i$ gets value 0;
```

Figure 1: An algorithm for alternating, disjunctive boolean equation systems.

all zeros, containing numbers indicating the order in which nodes have been visited. If *value*$[i] = 0$, this indicates that node i has not yet been visited. In addition, we keep a stack of integers, *stack*, represented as an array of size $|V|$ with a stack pointer p initially set to zero. We have integers *id* (initially zero), *min*, and m for the detection of SCCs, which occur in a similar vein in the algorithm for the detection of SCCs in [15]. The variable *id* is used to number the nodes with consecutive numbers in the sequence they are visited by the algorithm. The variable *min* refers to an earlier visited node, reachable from node k. If no such node exists, *min* = *value*$[k]$ at the end of the first **for** loop and node k is the root of a strongly connected component that includes all higher numbered nodes residing on the stack. The variable m plays the role of a simple auxiliary store. Finally, we keep also a set S, integer n, and booleans *mu* and *nu* for processing the SCCs, explained below.

The procedure *solve* invokes the recursive procedure *find*. The procedure *find* first checks whether the node k being visited is labelled with ν and has a self-loop. If these hold, we have found a node that trivially satisfies conditions (2a) and (2b) of lemma 4.2, and the solution $v(x_i) = 1$ can be reported and the execution of the algorithm is terminated. Otherwise, *find* pushes the nodes onto a stack, and recursively searches for strongly connected components. If such a component is found (when *min* = *value*$[k]$), *find* puts all nodes in the component that reside on the stack in a set S. While doing so, it is checked whether all nodes in the component have the same label. If a label is 1, corresponding to the fixed point operator ν, the variable *nu* is set to *true*, and if a label is 0, corresponding to μ, the variable *mu* is set to true. If *mu* = *false* on a SCC with more than one node, all nodes have label ν and so, conditions (2a) and (2b) of lemma 4.2 are trivially satisfied, and solution of x_i can be reported to 1.

If both variables *nu* and *mu* are true, the component is alternating. In this case it must be checked whether the SCC contains a cycle of which the smallest numbered node j has label $L(j) = \nu$, according to lemma 4.2 to justify x_i to be set to 1. This is simply checked by applying a procedure $cycle(G_{\mathcal{E}}, S, j)$ to all nodes $j \in S$ with *sign*$[j] = 1$. The procedure *cycle* consists of a simple linear depth first search and is not given in detail here.

Finally, if no node j with $L(j) = \nu$ satisfying conditions (2a) and (2b) of lemma 4.2 was found, we can report at the end of the procedure *solve* the solution v of $\mathcal{E}$ be such that $v(x_i) = 0$.

We find that the algorithm is correct and works in polynomial time and space.

Theorem 4.3. *The algorithm for local resolution works correctly on any purely disjunctive/conjunctive system of boolean equations.*

In order to formally estimate the computational costs, denote the set of alternating variables in a system $\mathcal{E}$ with variables in $\mathcal{X}$ by $alt(\mathcal{E})$, and define it as a set $\{x_i \mid x_i \in \mathcal{X}$ and x_i is mutually dependent with some $x_j \in \mathcal{X}$ such that $\sigma_i \neq \sigma_j\}$. The set of alternation free variables is denoted by $af(\mathcal{E})$ and is defined as $af(\mathcal{E}) = \mathcal{X} - alt(\mathcal{E})$. Note that for alternation-free boolean equation systems it holds that $alt(\mathcal{E}) = \emptyset$, because there are no ocurrences of mutually dependent variables with different signs. Then, it is easy to see that:

Theorem 4.4. *The algorithm for local resolution of disjunctive/conjunctive boolean equation systems requires time $O(af(\mathcal{E}) + alt(\mathcal{E})^2)$ and space $O(|\mathcal{E}|)$.*

5 Some experiments

In this section, we describe an implementation of the resolution algorithm presented in the previous section. This prototype solver for alternating disjunctive/conjunctive boolean equation systems is implemented in C. To give an impression of the performance, we report experimental results on solving two verification problems using the tool.

As benchmarks we used two sets of μ-calculus model checking problems taken from [11] and [14], converted to boolean equation systems. We do not take exactly the same formulas because our

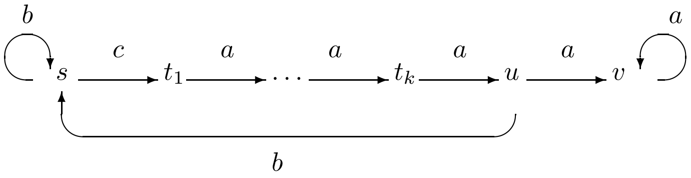

Figure 2: Process M_k for model checking the properties ϕ_1 and ϕ_2.

algorithm solves these in constant time, which would not give interesting results. The verification problems consist of checking μ-calculus formulas of alternation depth 2, on a sequence of regular labelled transition systems M_k of increasing size (see figure 2).

Suppose we want to check, at initial state s of process M_k, the property that transitions labeled b occur infinitely often along every infinite path of the process. This is expressed with alternating fixed-point formula:

$$\phi_1 \equiv \nu X.\mu Y.([b]X \wedge [-b]Y) \qquad (1)$$

The property is false at state s and we use the solver to find a counter-example for the formula. In second series of examples, we check the property that there is an execution in M_k starting from state s, where action a occurs infinitely often. This is expressed with the alternating fixed point formula

$$\phi_2 \equiv \nu X.\mu Y.(\langle a\rangle X \vee \langle -a\rangle Y) \qquad (2)$$

which is true at initial state s of the process M_k.

The problems of determining whether the system M_k satisfies the specifications ϕ_1 and ϕ_2 can be directly encoded as problems of solving the corresponding alternating boolean equation systems, which are in conjunctive and disjunctive forms. We report the times for the solver to find the local solutions corresponding to the local model checking problems of the formulas at state s.

The experimental results are given in table 1. The columns are explained below:

- Problem:
  - the process M_k, with $k + 3$ states
  - ϕ_1 the formula $\nu X.\mu Y.([b]X \wedge [-b]Y)$ to be checked
  - ϕ_2 the formula $\nu X.\mu Y.(\langle a\rangle X \vee \langle \neg a\rangle Y)$ to be checked
- n: the number of equations in the boolean equation system corresponding to the model checking problem
- Time: the time in seconds to find the local solution

The times reported are the time for the solver to find the local solutions measured as system time, on a 2.4Ghz Intel Xeon running linux (i.e. the times for the solver to read the equation systems from disk and build the internal data structure are excluded).

In the problem with the property ϕ_1, the solver found local solutions (and counterexamples) even without executing the quadratic part of the algorithm. In the problem with property ϕ_2, the quadratic computation needed to be performed only on very small portions of the equation systems. These facts are reflected in the performance of the solver, which exhibits linear growth in the execution times with increase in the size of the systems to be verified, in all of the experiments.

Problem		n	Time (sec)
$M_{5000000}$	ϕ_1	10 000 006	2.6
	ϕ_2	10 000 006	3.0
$M_{10000000}$	ϕ_1	20 000 006	5.5
	ϕ_2	20 000 006	6.4
$M_{15000000}$	ϕ_1	30 000 006	7.5
	ϕ_2	30 000 006	9.0

Table 1: Summary of execution times.

The benchmarks in [11] and [14] are essentially the only benchmarks in the literature for alternating boolean equation systems of which we are aware. These benchmarks have a quite simple structure, and therefore we must be careful in drawing general results from them Yet, it must be noted that our results are a *huge* improvement compared to these earlier reports, where the largest model only contained 1503 states and 3006 equations (if they used the same translation as we did) requiring times very comparable to ours. So, the results reported here provide an improvement in the order of a factor 10 000!

6 Discussion and conclusion

We argued that the verification of many formulas in the modal mu-calculus with alternating fixed points amounts to the verification of disjunctive/conjunctive straight boolean equation systems. Subsequently we provided an algorithm to solve these and showed that the performance of this algorithm on the standard benchmarks from the literature yield an improvement of many orders of magnitude. We believe that this makes the verification of a large class of formulas with alternating fixed points tractable, even for large, practical systems.

The algorithm that we obtain is for the large part linear, but contains an unpleasant quadratic factor. Despite several efforts, we have not been able to eliminate this. In essence this is due to the fact that we were not able to find a sub-quadratic algorithm for the following problem:

Open problem. Given a directed labelled graph $G = (V, E, L)$ of which the set of nodes is totally ordered. The labeling $L : V \to \{0, 1\}$ assigns to each node a value. Determine whether there exist a cycle in G of which the highest node has label 1.

As we believe that this problem has some interest by itself we provide it here.

Acknowledgements. We thank Michel Reniers for commenting a draft of this paper.

References

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

[2] A. Arnold and P. Crubille. A linear time algorithm to solve fixed-point equations on transition systems. Information Processing Letters, 29:57-66, 1988.

[3] A. Arnold and D. Niwinski. Rudiments of μ-calculus. Studies in Logic and the foundations of mathematics. Volume 146, Elsevier, 2001.

[4] J. Bradfield and C. Stirling. Modal Logicas and mu-Calculi: An introduction. Chapter 4 of Handbook of Process Algebra. J.A. Bergstra, A. Ponse and S.A. Smolka, editors. Elsevier, 2001.

[5] G. Bhat and R. Cleaveland. Efficient local model-checking for fragments of the modal μ-calculus. In Proceedings of the Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science 1055, pages 107-126, Springer Verlag 1996.

[6] R. Cleaveland and B. Steffen. Computing Behavioural relations logically. In proceedings of the 18 International Colloquium on Automata, Languages and Programming, Lecture Notes Computer Science 510, pages 127-138, Springer Verlag, 1991.

[7] E.A. Emerson, C. Jutla and A.P. Sistla. On model checking for fragments of the μ-calculus. In C. Courcoubetis, editor, Fifth Internat. Conf. on Computer Aided Verification, Elounda, Greece, Lecture Notes in Computer Science 697, pages 385-396, Springer Verlag, 1993.

[8] E.A. Emerson, C. Jutla, and A.P. Sistla. On model checking for the μ-calculus and its fragments. Theoretical Computer Science 258:491-522, 2001.

[9] M. Jurdzinski. Deciding the winner in parity games is in $UP \cap co - UP$. Information Processing Letters, 68:119-124, 1998.

[10] D. Kozen. Results on the propositional μ-calculus. Theoretical computer Science 27:333-354, 1983.

[11] X. Liu, X, C.R. Ramakrishnan and S.A. Smolka. Fully Local and Efficient Evaluation of Alternating Fixed Points. In B. Steffen, editor, Proceedings of TACAS'98, Lecture Notes in Computer Science 1384, Springer Verlag, 1988.

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

[13] R. Mateescu. A Generic On-the-Fly Solver for Alternation-Free Boolean Equation Systems. Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2003 (Warsaw, Poland), volume 2619 of Lecture Notes in Computer Science, pages 81-96. Springer Verlag, April 2003.

[14] B. Steffen, A. Classen, M. Klein, J. Knoop and T. Margaria. The fixpoint analysis machine. In I. Lee and S.A. Smolka, editors, Proceedings of the Sixth International Conference on Concurrency Theory (CONCUR '95), Lecture Notes in Computer Science 962, pages 72-87. Springer Verlag, 1995.

[15] R. Tarjan. Depth-First Search and Linear Graph Algorithms. SIAM J. Computing, Vol. 1, No. 2, June 1972.

[16] B. Vergauwen and J. Lewi. A linear algorithm for solving fixed-point equations on transition systems. In J.-C. Raoult, editor, CAAP'92, Lecture Notes Computer Science 581, pages 321-341, Springer Verlag, 1992.

[17] B. Vergauwen and J. Lewi. Efficient Local Correctness Checking for Single and Alternating Boolean Equation Systems. In proc. of ICALP'94.