Solving Alternating Boolean Equation Systems in Answer Set Programming

Misa Keinänen[1,2] and Ilkka Niemelä[1]

[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] *Department of Computer Science, CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*
`Ilkka.Niemala@hut.fi,Misa.Keinanen@hut.fi`

Abstract. In this paper we apply answer set programming to solve alternating Boolean equation systems. We develop a novel characterization of solutions for variables in disjunctive and conjunctive Boolean equation systems. Based on this we devise a mapping from Boolean equation systems with alternating fixed points to normal logic programs such that the solution of a given variable of an equation system can be determined by the existence of a stable model of the corresponding logic program. Our translation is such that it ensures the computational complexity of solving important subclasses of equation systems, like linear time for solving alternation-free equation systems and polynomial time for disjunctive/conjunctive alternating systems. The technique can be used to model check alternating formulas of μ-calculus logic.

1 Introduction

The correctness of finite-state concurrent systems can be formalized as model checking problems. Model checking is a verification technique aimed at determining whether a system specification model satisfies desired properties expressed as temporal logic formulas. In recent years, research on model checking has addressed large scale verification problems, which are often solved by special purpose verification tools.

Yet it has been demonstrated that also logic programming systems can successively be applied to the construction of practical model checkers, like e.g. in [8, 4, 11]. In the present paper, we continue this line of research and restrict the attention to the model checking problem of modal μ-calculus [10], and in particular to its formulation as *Boolean equation systems* [1, 13, 16]. The research topic belongs to the area of formal verification, but more specifically it addresses effective ways of solving systems of fixed point equations.

The modal *μ-calculus* is an expressive logic for systems verification, and has been widely studied in the recent model checking literature (e.g. [2] gives a general exposition). Boolean equation systems provide here a useful framework, because μ-calculus expressions can easily be translated into this more flexible formalism (see [1, 2, 13] for the standard translations). The complexity of μ-calculus model checking is an important open problem; no polynomial time algorithm has been discovered. On the other hand, it is shown in [5, 6] that the problem is in the complexity class NP $\cap$ co-NP (and is known to be even in UP $\cap$ co-UP [9]). Hence, the problem appears to be directly solvable with any answer set programming system capable of handling NP-complete problems. In this paper, we propose an answer set programming based approach for solving alternating Boolean equation systems. We suggest the method provides a basis for a model checking technique for alternating fragment of μ-calculus logic.

Previously, answer set programming has been applied to solve Boolean equation systems in [11] where it is argued that alternating Boolean equation systems can be solved by computing certain *preferred stable models* of propositional normal logic programs corresponding to Boolean equation systems. Moreover, it is shown in [11] how alternation-free Boolean equation systems can be mapped to stratified logic programs, which can be directly solved in linear time, preserving the complexity [15] of model checking alternation-free fragment of μ-calculus. However, the approach proposed in [11] does not preserve the polynomial time complexity of solving disjunctive and conjunctive Boolean equation systems.

We reduce the problem of solving alternating Boolean equation systems to computing stable models of normal logic programs. This is achieved by devising an alternative mapping from Boolean equation systems to normal logic programs so the solution for a given variable in an equation system can be determined by the existence of a stable model of the corresponding logic program. Our translation is such that it ensures polynomial time complexity of solving both disjunctive and conjunctive alternating systems, and hence preserves the complexity of model checking many important fragments of μ-calculus, like $L1$ and $L2$ investigated in [3, 5, 6].

The paper is organized as follows. In the following section we introduce basic notions of Boolean equation systems. In Section 3 we state some properties of Boolean equation systems which are important in solving them. In Section 4 we review stable model semantics of normal logic programs. In Section 5 we show how alternating Boolean equation systems can be solved using answer set programming techniques. In Section 6 we discuss some initial experimental results. Finally, Section 7 contains conclusive remarks.

2 Boolean Equation Systems

We will give in this section a short presentation of Boolean equation systems. Essentially, a Boolean equation system is an ordered sequence of fixed point equations over Boolean variables, with associated signs, μ and ν, specifying the polarity of the fixed points. The equations are of the form $\sigma x = \alpha$, where α is a positive Boolean formula. The sign, σ, is μ if the equation is a least fixed point equation and ν if it is a greatest fixed point equation.

Let $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ be a set of Boolean variables. The set of *positive Boolean expressions* over $\mathcal{X}$ is denoted by $B^+(\mathcal{X})$, and given 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 define the syntax of Boolean equation systems as follows.

Definition 1 (The syntax of a Boolean equation system). *A Boolean equation is of the form $\sigma_i x_i = \alpha_i$, where $\sigma_i \in \{\mu, \nu\}$, $x_i \in \mathcal{X}$, and $\alpha_i \in B^+(\mathcal{X})$.*

A Boolean equation system is an ordered sequence of Boolean equations

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

where the left-hand sides of the equations are all different. We assume that the order on variables and equations are in synchrony.

The semantical interpretation of Boolean equation systems is such that each system $\mathcal{E}$ has a uniquely determined solution, which is a valuation assigning a constant value in $\{0, 1\}$ (0 standing for $false$ and 1 for $true$) to variables occurring in $\mathcal{E}$. More precisely, the solution is a truth assignment to the variables $\{x_1, x_2, ...\}$ satisfying the fixed-point equations such that the right-most equations have higher priority over left-most equations.

Formally, we consider positive Boolean expressions α as functions over Boolean lattice $(\{0, 1\}, \leq)$ with $0 < 1$. Let $v, v_1, \ldots$ range over valuations, where each v is a function $v : \mathcal{X} \to \{0, 1\}$. A function α can be applied to a valuation v, and the result $\alpha(v)$ is the value of α after substituting each free variable x of α by $v(x)$. We denote by $v[x/a]$ the valuation that coincides with v for all variables except x. We suppose that $[x/a]$ has priority over all operations and $v[x/a]$ stands for $(v[x/a])$. Then, the semantics is defined as follows:

Definition 2 (The solution of a Boolean equation system). *Let $(\sigma x = \alpha)\mathcal{E}$ be a Boolean equation system, and $v : \mathcal{X} \to \{0, 1\}$ a valuation. The solution of $(\sigma x = \alpha)\mathcal{E}$ relative to v, denoted by $[\![(\sigma x = \alpha)\mathcal{E}]\!]v$, is an assignment inductively defined by*

$$[\![\epsilon]\!]v = v$$
$$[\![(\sigma x = \alpha)\mathcal{E}]\!]v = \begin{cases} [\![\mathcal{E}]\!]v[x/\mu x.\alpha([\![\mathcal{E}]\!]v)] & \textit{if } \sigma = \mu \\ [\![\mathcal{E}]\!]v[x/\nu x.\alpha([\![\mathcal{E}]\!]v)] & \textit{if } \sigma = \nu \end{cases}$$

where $\mu x.\alpha([\![\mathcal{E}]\!]v) = \bigwedge\{a | a \geq \alpha([\![\mathcal{E}]\!]v[x/a])\}$ *and* $\nu x.\alpha([\![\mathcal{E}]\!]v) = \bigvee\{a | a \leq \alpha([\![\mathcal{E}]\!]v[x/a])\}$.

Notice in the above definition that we present an empty equation system as ϵ, and we denote the *greatest lower bound* of the complete lattice $(\{0, 1\}, \leq)$ by operator $\bigwedge$ and the *least upper bound* by $\bigvee$.

Example 1. Let $\mathcal{X}$ be the set $\{x_1, x_2, x_3\}$ and assume we are given a Boolean equation system

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

The solution $[\![\mathcal{E}_1]\!]v$ of $\mathcal{E}_1$ is given by the valuation $v : \mathcal{X} \to \{0, 1\}$ defined by $v(x_i) = 1$ for $i = 1, 2, 3$.

3 Properties of Boolean Equation Systems

In this section, we discuss important notions of Boolean equation systems. We also state some facts about Boolean equation systems, which turn out to be useful in the computation of their solutions.

The *size* of Boolean equation systems is inductively defined as $|\epsilon| = 0$ and $|(\sigma x = \alpha)\mathcal{E}| = 1 + |\alpha| + |\mathcal{E}|$, where $|\alpha|$ is the number of variables and constants in α.

A Boolean equation system $\mathcal{E}$ is in *standard form* if each right-hand side expression α_i consists of a disjunction $x_i \vee x_j$, conjunction $x_i \wedge x_j$, or a single variable x_i. As pointed out in [13], for each system $\mathcal{E}$ there is another system $\mathcal{E}'$ in *standard form* such that $\mathcal{E}'$ preserves the solution of $\mathcal{E}$ and has size linear in the size of $\mathcal{E}$. We may thus restrict to standard form Boolean equation systems.

We say 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. We say that a variable x_i is *self-dependent*, if x_i depends on itself such that no variable x_j with $j < i$ occurs in this chain of dependencies. It is said that two variables x_i and x_j are *mutually dependent* if x_i depends on x_j and vice versa. A Boolean equation system is *alternation free* if x_i and x_j are mutually dependent implies that $\sigma_i = \sigma_j$ holds. Otherwise, the Boolean equation system is said to be *alternating*.

Example 2. Consider the Boolean equation system $\mathcal{E}_1$ of Example 1. The system $\mathcal{E}_1$ is in standard form and is alternating, because it contains alternating fixed points with mutually dependent variables having different signs, like x_1 and x_2 with $\sigma_1 \neq \sigma_2$. The variables x_1 and x_3 of $\mathcal{E}_1$ are self-dependent.

The variables of a standard form Boolean equation system can be partitioned in *blocks* such that any two distinct variables belong to the same block iff they are mutually dependent. The dependency relation among variables extends to blocks such that block B_i depends on another block B_j if some variable occurring in block B_i depends on another variable in block B_j. The resulting dependency relation among blocks is an ordering. In Mader [13], there are two useful lemmas (Lemma 6.2 and Lemma 6.3) which allow us to solve all blocks of standard form Boolean equation systems one at a time, starting from the last block and then substituting the solutions to the blocks higher up in the ordering. Alternation-free blocks of standard form Boolean equation systems can be trivially solved in linear time in the size of the blocks. Thus, we restrict here to devise a technique to solve alternating blocks of standard form Boolean equation systems, for which no polynomial time solution technique is known.

We call an equation $\sigma_i x_i = \alpha_i$ *disjunctive* if its right-hand side α_i is a disjunction. A standard form Boolean equation system is said to be *disjunctive* if all its equations are either *disjunctive* equations or single variables. Similarly, a Boolean equation $\sigma_i x_i = \alpha_i$ is *conjunctive* if its right-hand side α_i is a conjunction. A standard form Boolean equation system is *conjunctive* if all its equations are *conjunctive* or single variables.

The following lemmas form the basis for our answer set programming based technique to solve standard form Boolean equation systems with alternating fixed points. For a disjunctive (conjunctive respectively) form Boolean equation systems we have that:

Lemma 1. *Let $\mathcal{E}$ be a disjunctive (conjunctive) Boolean equation system in standard form. Let x_i be any variable of $\mathcal{E}$, and v the solution $\mathcal{E}$. Then the following are equivalent:*

1. $v(x_i) = 1$ *(or* $v(x_i) = 0$ *respectively)*

2. there is a variable x_j in $\mathcal{E}$ such that $\sigma_j = \nu$ ($\sigma_j = \mu$) and:
   (a) x_i depends on x_j and
   (b) x_j is self-dependent

From each Boolean equation system $\mathcal{E}$ containing both disjunctive and conjunctive equations we may construct a new Boolean equation system $\mathcal{E}'$, which is either in a disjunctive or in a conjunctive form. In order to obtain from $\mathcal{E}$ a disjunctive form system $\mathcal{E}'$, we remove in every conjunctive equation of $\mathcal{E}$ exactly one conjunct; otherwise the system $\mathcal{E}$ is unchanged. The dual case is similar. For any standard form Boolean equation system having both disjunctive and conjunctive equations we have the property:

Lemma 2. *Let $\mathcal{E}$ be a standard form Boolean equation system, x_i any variable occurring in $\mathcal{E}$, and v the solution of $\mathcal{E}$. Then the following are equivalent:*

1. $v(x_i) = 0$ (or $v(x_i) = 1$ respectively)
2. from $\mathcal{E}$ can be constructed a disjunctive (conjunctive) system $\mathcal{E}'$ with the solution v' s.t. $v'(x_i) = 0$ ($v'(x_i) = 1$ respectively).

In Section 5 we will see the application of the above lemmas to give a compact encoding of the problem of solving alternating Boolean equation systems as the problem of finding certain stable models of normal logic programs.

4 Stable Models of Normal Logic Programs

For encoding Boolean equation systems we use normal logic programs with the stable model semantics [7]. A normal rule is of the form

$$a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n. \tag{1}$$

where each a, b_i, c_j is a ground atom. Models of a program are sets of ground atoms. A set of atoms Δ is said to satisfy an atom a if $a \in \Delta$ and a negative literal not a if $a \notin \Delta$. A rule r of the form (1) is satisfied by Δ if the head a is satisfied whenever every body literal $b_1, \ldots, b_m$, not $c_1, \ldots,$ not c_n is satisfied by Δ and a program Π is satisfied by Δ if each rule in Π is satisfied by Δ.

Stable models of a program are sets of ground atoms which satisfy all the rules of the program and are justified by the rules. This is captured using the concept of a *reduct*. For a program Π and a set of atoms Δ, the reduct Π^Δ is defined by

$$\Pi^\Delta = \{a \leftarrow b_1, \ldots, b_m. \mid a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n. \in \Pi, \\ \{c_1, \ldots, c_n\} \cap \Delta = \emptyset\}$$

i.e., a reduct Π^Δ does not contain any negative literals and, hence, has a unique subset minimal set of atoms satisfying it.

Definition 3. *A set of atoms Δ is a stable model of a program Π iff Δ is the unique minimal set of atoms satisfying Π^Δ.*

We employ two extensions which can be seen as compact shorthands for normal rules. We use *integrity constraints*, i.e., rules

$$\leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n. \tag{2}$$

with an empty head. Such a constraint can be taken as a shorthand for a rule

$$f \leftarrow \text{not } f, b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n.$$

where f is a new atom. Notice that a stable model Δ satisfies an integrity constraint (2) only if at least one of its body literals is not satisfied by Δ.

For expressing the choice of selecting exactly one atom from two possibilities we use *choose-1-of-2 rules* on the left which correspond to the normal rules on the right:

$$1\ \{a_1, a_2\}\ 1. \qquad \begin{array}{l} a_1 \leftarrow \text{not } a_2. \\ a_2 \leftarrow \text{not } a_1. \\ \leftarrow a_1, a_2. \end{array}$$

Choose-1-of-2 rules are a simple subclass of cardinality constraint rules [14]. The `Smodels` system (http://www.tcs.hut.fi/Software/smodels/) provides an implementation for cardinality constraint rules and includes primitives supporting directly such constraints without translating them first to corresponding normal rules.

5 Solving Boolean Equation Systems with Stable Models

The overall idea of our approach is as follows. Given a standard form Boolean equation system $\mathcal{E}$, we partition its variables into blocks so that variables are in the same block iff they are mutually dependent. The partition can be constructed in linear time on the basis of the dependencies between the variables. Like argued in Section 3, the variables belonging to the same blocks can be solved iteratively one block at a time.

If all variables in a single block have the same sign, i.e. the block is alternation-free, the variables in this block can be trivially solved in linear time. So we only need to concentrate on solving alternating blocks containing mutually dependent variables with different signs. Consequently, we present here a technique to solve an alternating Boolean equation system which applies Lemmas 1-2 from Section 3.

In order to reduce the resolution of alternating Boolean equation systems to the problem of computing stable models of logic programs we define a translation from equation systems to normal logic programs. Consider a standard form, alternating Boolean equation system $\mathcal{E}$ and a variable x_k of $\mathcal{E}$. We construct a logic program $\Pi(\mathcal{E}, x_k)$ which captures the solution $v(x_k)$ of x_k. Suppose that the number of conjunctive equations of $\mathcal{E}$ is less than (or equal to) the number of disjunctive equations, or that no conjunction symbols occur in the right-hand sides of $\mathcal{E}$. The dual case goes along exactly the same lines and is omitted.[3] The idea is that $\Pi(\mathcal{E}, x_k)$ is a ground program which is polynomial in the size of the equation system $\mathcal{E}$. We give a compact description of $\Pi(\mathcal{E}, x_k)$

[3] This is the case where the number of disjunctive equations of $\mathcal{E}$ is less than the number of conjunctive equations, or where no disjunction symbols occur in the right-hand sides of $\mathcal{E}$.

as a program with variables. This program consists of the rules

$$solve(k). \tag{3}$$
$$depends(Y) \leftarrow dep(X,Y), solve(X). \tag{4}$$
$$depends(Y) \leftarrow depends(X), dep(X,Y). \tag{5}$$
$$reached(X,Y) \leftarrow nu(X), dep(X,Y), Y \geq X. \tag{6}$$
$$reached(X,Y) \leftarrow reached(X,Z), dep(Z,Y), Y \geq X. \tag{7}$$
$$\leftarrow depends(Y), reached(Y,Y), nu(Y). \tag{8}$$

extended for each equation $\sigma_i x_i = \alpha_i$ of $\mathcal{E}$ by

- $dep(i,j)$., if $\alpha_i = x_j$
- $dep(i,j)$. and $dep(i,k)$., if $\alpha_i = (x_j \vee x_k)$
- $1\ \{dep(i,j), dep(i,k)\}\ 1$., if $\alpha_i = (x_j \wedge x_k)$

and by $nu(i)$. for each variable x_i such that $\sigma_i = \nu$.

The idea is that for the solution v of $\mathcal{E}$, $v(x_k) = 0$ iff $\Pi(\mathcal{E}, x_k)$ has a stable model. This is captured in the following way. The system $\mathcal{E}$ is turned effectively to a disjunctive system by making a choice between $dep(i,j)$ and $dep(i,k)$ for each conjunctive equation $x_i = (x_j \wedge x_k)$. Hence, each stable model corresponds to a disjunctive system constructed from $\mathcal{E}$ and vice versa. By Lemma 2 $v(x_k) = 0$ iff there is such a disjunctive system for which $v'(x_k) = 0$. By Lemma 1 for a disjunctive system $v'(x_k) = 1$ iff there is a variable x_j such $\sigma_j = \nu$ and x_k depends on x_j and x_j is self-dependent. The program $\Pi(\mathcal{E}, x_k)$ effectively rules out each stable model where $v'(x_k) = 1$ in the corresponding disjunctive system. This is done by eliminating models where there is a variable x_j such that $\sigma_j = \nu$ and x_k depends on x_j and x_j is self-dependent. Hence, $\Pi(\mathcal{E}, x_k)$ has a stable model iff there is a disjunctive system constructed from $\mathcal{E}$ such that $v'(x_k) \neq 1$, i.e., $v'(x_k) = 0$. Hence, we have our main result.

Theorem 1. *Let $\mathcal{E}$ be standard form, alternating Boolean equation system. Let x_k be any variable and v the solution of $\mathcal{E}$. Then $v(x_k) = 0$ iff $\Pi(\mathcal{E}, x_k)$ has a stable model.*

Similar property holds also for the dual program, which allows us to solve all alternating blocks of standard form Boolean equation systems.

Although $\Pi(\mathcal{E}, x_k)$ is given using variables, for the theorem above a finite ground instantiation of it is sufficient. For explaining the ground instantiation we introduce a relation *depDom* such that $depDom(i,j)$ holds iff there is an equation $\sigma_i x_i = \alpha_i$ of $\mathcal{E}$ with x_j occurring in α_i. Now the sufficient ground instantiation is obtained by substituting variables X, Y in the rules (4–6) with all pairs i, j such that $depDom(i,j)$ holds, substituting variables X, Y, Z in rule (7) with all triples l, i, j such that $nu(l)$ and $depDom(i,j)$ hold and variable Y in rule (8) with every i such that $nu(i)$ holds. This means also that such conditions can be added as domain predicates to the rules without compromising the correctness of the translation. For example, rule (7) could be replaced by $reached(X,Y) \leftarrow nu(X), depDom(Z,Y), reached(X,Z), dep(Z,Y), Y \geq X$. Notice that such conditions make the rules domain restricted as required, e.g., by the `Smodels` system.

$$\left.\begin{array}{l}\nu\, x_1 = x_2 \wedge x_n \\ \mu\, x_2 = x_1 \vee x_n \\ \nu\, x_3 = x_2 \wedge x_n \\ \mu\, x_4 = x_3 \vee x_n \\ \ldots \\ \nu\, x_{n-3} = x_{n-4} \wedge x_n \\ \mu\, x_{n-2} = x_{n-3} \vee x_n \\ \nu\, x_{n-1} = x_{n-2} \wedge x_n \\ \mu\, x_n = x_{n-1} \vee x_{n/2} \end{array}\right\} \text{for } n \in 2\mathbb{N}$$

Problem (n)	Time (sec)
1800	27.9
2000	34.9
2200	42.3
2400	50.5
2600	59.8

Fig. 1. The Boolean equation system in [13, p.91] and experimental results.

6 Experiments

In this section, we describe some experimental results on solving alternating Boolean equations systems with the approach presented in the previous section. We demonstrate the technique on two series of examples. The times reported are the average of 3 runs of the time for `Smodels` 2.27 to find the solutions as reported by the $/usr/bin/time$ command on a 2.0Ghz AMD Athlon running Linux.

The first series deals with solving alternating Boolean equation systems of increasing size and alternation depth. The problem is taken from [13, p.91] and consists of finding solution $v(x_1)$ of the left-most variable x_1 occurring in a sequence of alternating fixed point equations given in Fig.1. The example is such that a Boolean equation system with n equations has the alternation depth n. The solution of the system is such that $v(x_1) = 1$ which can be obtained by determing the existence of a stable model of the corresponding logic program. The experimental results are summarised in Fig.1. Our benchmarks are essentially the only results in the literature for alternating Boolean equation systems with the alternation depth $n \geq 4$ of which we are aware. Notice that our benchmarks have the alternation depths $1800 \leq n \leq 2600$. Like pointed out in [13], known algorithms based on the approximation technique are exponential in the size of the equation system in Fig.1, because a maximal number of backtracking steps is always needed to solve the left-most equation.

In second series of examples we used a set of μ-calculus model checking problems taken from [12], converted to alternating Boolean equation systems. The problems consist of checking a μ-calculus formula of alternation depth 2, on a sequence of models $M_k = (S, A, \longrightarrow)$ of increasing size (see Fig.2 in [12]). Suppose that all transitions of process M_k in [12] are labeled with a and we want to check, at initial state s, the property that a is enabled infinitely often along all infinite paths. This is expressed with alternating fixed point formula:

$$\phi \equiv \nu X.\mu Y.([-].(\langle a\rangle true \wedge X) \vee Y) \qquad (9)$$

which is true at initial state s of the process M_k. The problem can be directly encoded as the problem of solving the corresponding alternating equation system in Fig.2. The results are given in Fig. 2. The columns are:

$$\left.\begin{aligned}\nu\, x_s &= y_s\\ \mu\, y_s &= \bigwedge_{s'\in\nabla(t,s)} z_{s'} \vee y_s\\ \mu\, z_s &= \bigvee_{s'\in\nabla(a,s)} true \wedge x_s\end{aligned}\right\}\text{for all } s\in S.$$

where $\nabla(t,s) := \{s' | s \xrightarrow{i} s' \wedge i \in A\}$
and $\nabla(a,s) := \{s' | s \xrightarrow{a} s'\}$.

Problem	$\lvert s\rvert$	$\lvert\longrightarrow\rvert$	n	Time (sec)
M_{500}	503	505	1006	1.9
M_{1000}	1003	1005	2006	7.3
M_{1500}	1503	1505	3006	16.4

Fig. 2. The Boolean equation system and experimental results.

- Problem: Process $M_k = (S, A, \longrightarrow)$ from [12].
- $|S|$: Number of states in M_k.
- $|\longrightarrow|$: Number of transitions in M_k.
- n: Number of equations in the corresponding Boolean equation system.
- Time: The time in seconds to solve variable x_s.

The benchmarks in [12] have a quite simple structure and no general results can be drawn from them. A more involved practical evaluation of our approach is highly desirable, and benchmarking on real world systems is left for future work.

7 Conclusion

We presented an aswer set programming based method for computing the solutions of alternating Boolean equation systems. We developed a novel characterization of solutions for variables in Boolean equation systems and devised a mapping from systems with alternating fixed points to normal logic programs. Our translation is such that the solution of a given variable of an equation system can be determined by the existence of a stable model of the corresponding logic program. This result provides the basis to verify μ-calculus formulas with alternating fixpoints, using answer set programming techniques.

The experimental results indicate that stable model computation is quite competitive approach to solve Boolean equations systems in which the number of alternation is relatively large. The alternation of fixpoint operators give more expressive power in μ-calculus, but all known model checking algorithms are exponential in the alternation. Consequently, our approach is expected to be quite effective in the verification tasks, where there is a need of formulas with great expressive power.

Acknowledgements. We would like to thank Keijo Heljanko for valuable discussions. The financial supports of Academy of Finland (project 53695), Emil Aaltonen foundation and Helsinki Graduate School in Computer Science and Engineering are gratefully acknowledged.

References

1. H.R. Andersen. Model checking and Boolean graphs. Theoretical Computer Science, 126:3-30, 1994.
2. A. Arnold and D. Niwinski. Rudiments of μ-calculus. Studies in Logic and the foundations of mathematics. Volume 146, Elsevier, 2001.
3. 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.
4. G. Delzanno and A. Podelski. Model checking in CLP. In Proceedings of the Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science 1579, pp. 223-239, 1999.
5. 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.
6. 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.
7. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming*, pages 1070–1080, Seattle, USA, August 1988. The MIT Press.
8. K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. Forthcoming in Theory and Practice of Logic Programming (TPLP), Cambridge University Press, 2003.
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. K. N. Kumar, C. R. Ramakrishnan, and S. A. Smolka. Alternating fixed points in Boolean equation systems as preferred stable models. In proceedings of 17th International Conference of Logic Programming, Lecture Notes in Computer Science 2237, 2001.
12. 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, 1998.
13. A. Mader. Verification of Modal Properties using Boolean Equation Systems. PhD thesis, Technical University of Munich, 1997.
14. P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.
15. 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.
16. B. Vergauwen and J. Lewi. Efficient Local Correctness Checking for Single and Alternating Boolean Equation Systems. In proc. of ICALP'94.