

Efficient Checking of Behavioural Relations and Modal Assertions using Fixed-Point Inversion

Henrik Reif Andersen¹ and Bart Vergauwen²

¹ Department of Computer Science, Technical University of Denmark,
Bld. 344, DK-2800 Lyngby, Denmark.

E-mail: hra@id.dtu.dk, WWW: <http://www.id.dtu.dk/~hra>

² Department of Computer Science, K.U. Leuven,
Celestijnenlaan 200A, B-3001 Leuven, Belgium.

E-mail: bartv@cs.kuleuven.ac.be

Abstract. This paper presents an algorithm for solving Boolean fixed-point equations containing one level of nesting of minimum and maximum fixed points. The algorithm assumes that the equations of the inner fixed point is of a certain restricted kind and has a worst-case time- and space-complexity that is linear in the size of the equation system. By observing that a range of behavioral relations – in particular weak bisimulation – and modal assertions can be checked using equation systems of this restricted form, the algorithm improves on existing *ad hoc* constructed algorithms.

Finally, we show how the key idea of *inverting a fixed point* can be used in decreasing the number of fixed-point iterations needed in BDD-based methods for solving the same class of problems.

1 Introduction

Transition systems play a central role as formal models for reactive and concurrent systems. This paper deals with the problem of automatically verifying the correctness of finite transition systems. Verification is the process of comparing a transition system with a system specification. System specifications usually fall into two categories: *logical* versus *behavioural* specifications. Logical specifications describe properties that the system is supposed to satisfy, e.g. deadlock freedom, mutual exclusion, liveness properties, etc. Temporal and modal logics are popular tools for expressing such properties. A behavioural specification, on the other hand, usually takes the form of another transition system describing the system behaviour at a more abstract level. Verification then boils down to comparing two transition systems (an abstract and a concrete one) w.r.t. a given equivalence or preorder relation.

We will formulate all these verification problems as a question of finding solutions to *Boolean equation systems*: A sequence of monotone equations over Boolean variables of two different kinds, either minimum or maximum. For a restricted class of such systems we give a linear time algorithm for computing the solution and show how this algorithm gives rise to efficient algorithms for

a range of verification problems, including weak bisimulation equivalence and checking of fairness properties expressed in the modal μ -calculus.

2 Boolean Equation Systems

To express monotone Boolean equations we assume a set of variables $Vars$ ranged over by X, Y, Z, \dots . As right-hand sides B we allow finite disjunctions and conjunctions of variables:

$$B ::= \bigvee \{X_1, X_2, \dots, X_n\} \mid \bigwedge \{X_1, X_2, \dots, X_n\},$$

where $n \geq 0$. An empty disjunction is written as 0 and an empty conjunction as 1. From these monotone Boolean expressions we form Boolean equation systems E :

$$E ::= \epsilon \mid X =_{\mu} B \ E \mid X =_{\nu} B \ E,$$

where ϵ is the empty equation system. The first equation is a *minimum fixed-point equation*, the second a *maximum fixed-point equation*. We shall often use σ to range over $\{\mu, \nu\}$. For a Boolean equation system E to be well-formed we assume that all left-hand side variables of E are different. We denote by $bv(E)$ the left-hand side variables of E , which are said to be *bound* by E . Similarly, the set of *free variables* $fv(E)$ of E is the set of variables occurring in right-hand sides which are not bound by E . We shall sometimes need to restrict an equation system E to a set of variables $S \subseteq bv(E)$. The right-hand sides are unchanged. We denoted the restricted equation system by $E \upharpoonright S$.

The solution to a Boolean equation system is a function $\delta : bv(E) \rightarrow \mathbb{O}$ from the bound variables of E to the two-point lattice $\mathbb{O} = (\{0, 1\}, \wedge, \vee)$ of truth-values. The solution will be given relative to an environment $\rho : Vars \rightarrow \mathbb{O}$ giving meaning to (at least) the free variables $fv(E)$ of E . We will make use of the following notation for environments: $[]$ is the empty environment, $[v/X]$ is the environment that assigns v to X , and for two environments ρ and ρ' with disjoint variable domains, $\rho\rho'$ is the union of them. Hence, for example $[] [v/X] = [v/X] []$ and $[u/X][v/Y] = [v/Y][u/X]$. The meaning of a right-hand side B is now trivially defined:

$$\begin{aligned} \llbracket \bigvee \{X_1, X_2, \dots, X_n\} \rrbracket \rho &= \bigvee \{\rho(X_1), \rho(X_2), \dots, \rho(X_n)\} \\ \llbracket \bigwedge \{X_1, X_2, \dots, X_n\} \rrbracket \rho &= \bigwedge \{\rho(X_1), \rho(X_2), \dots, \rho(X_n)\} \end{aligned}$$

The solution $\llbracket E \rrbracket \rho$ is defined by:

$$\begin{aligned} \llbracket \epsilon \rrbracket \rho &= [] \\ \llbracket X =_{\sigma} B \ E \rrbracket \rho &= (\llbracket E \rrbracket \rho [v/X]) [v/X] \\ &\quad \text{where } v = \sigma u. \llbracket B \rrbracket (\rho [u/X] (\llbracket E \rrbracket \rho [u/X])) \end{aligned}$$

Here $\sigma u.f(u)$ denotes the extremal σ fixed point of the monotone function f on the lattice \mathbb{O} . Since \mathbb{O} only contains two elements it is easy to see that

$\mu u.f(u) = f(0)$ and $\nu u.f(u) = f(1)$. Using this observation, the given semantics provides a straightforward recursive procedure for computing the solution. It is extremely inefficient³ and a better way is to find the solutions in *blocks*. A σ -*block* E is a Boolean equation system containing only σ -equations. The solution to a σ -block with n variables can be found simultaneously for all variables as a fixed point in \mathbb{O}^n in time $O(|E|)$ ([And94, VL92]). We shall henceforth refer to this algorithm as the *linear-time fixed-point finder*.

For a sequence of k alternating blocks $E_1 E_2 \dots E_k$ the solution can be found in time $O(|E|^k)$ ([And94, And93, CDS92]). In fact, this bound is quite pessimistic. The actual complexity is better but difficult to express. However, for two alternating blocks $E_1 E_2$ a more precise bound (of the algorithm in [And93]) is $O(|E_1| + |fv(E_2) \cap bv(E_1)||E_2|)$. The factor $|fv(E_2) \cap bv(E_1)|$ is the number of bound variables from E_1 that appears in E_2 and it arises from recomputation of the inner block.

We will consider the special case of two alternating blocks when the inner block possesses a property of being *SCC-consistent*. In this case, we will in linear time change the inner σ -block to a block of the opposite type and of size no larger than the original. The solution to the two blocks can now be found in linear time by the linear-time fixed-point finder. Despite the simplicity of the resulting algorithm it has some striking applications which we shall discuss in a later section.

When discussing time and space complexity of algorithms in this paper, the underlying computational model is the standard — a RAM under the “uniform cost”-criterion [AHU74]. We assume that all Boolean equation systems are given by a *standard representation* when used as input to the algorithms. In a standard representation variables are numbers 1 through n . Right-hand sides are represented by three arrays *tags*, *type* and *vars*. The first is an array of tags indicating whether the right-hand side is disjunctive (\vee) or conjunctive (\wedge). The second is an array of types indicating whether the equation is of type μ or ν , and the third is an array of linked lists of variables appearing as arguments. For a Boolean equation system E with n variables and a total of m variables appearing on the right-hand sides, this gives a representation of size $O(n + m)$. Throughout the paper we shall use $n + m$ as the measure of the size of E and refer to it as $|E|$.

3 Fixed-Point Inversion

Let \rightsquigarrow_E be a relation on variables of E defined by

$$X \rightsquigarrow_E X' \Leftrightarrow_{\text{def}} X \text{ occurs in the right-hand side of the bound variable } X'.$$

Let \rightsquigarrow_E^* be the reflexive, transitive closure of \rightsquigarrow_E .

Recall, that a *strongly connected component* (SCC) for \rightsquigarrow_E is a set of variables S such that for all $X, Y \in S$, $X \rightsquigarrow_E^* Y \rightsquigarrow_E^* X$, hence there is a cycle containing

³ It is highly exponential, see [And93] for elaboration.

both X and Y . A *maximal SCC* is an SCC that does not contain any other SCC. Strongly connected components play a crucial rôle in the fixed-point inversion where a block must give consistent solutions to all variables of the same maximal SCC. A property we refer to as being *SCC-consistent*:

Definition 1. A σ -block E is *SCC-consistent* if for all maximal strongly connected components S of \rightsquigarrow_E the following holds: For all X, X' in S and for all environments ρ assigning values to the free variables of $E \upharpoonright S$, $(\llbracket E \upharpoonright S \rrbracket \rho)(X) = (\llbracket E \upharpoonright S \rrbracket \rho)(X')$.

Notice that SCC-consistency is a property of a σ -block that is independent of whether σ is μ or ν .

In the following proposition characterizing some classes of SCC-consistent blocks, we shall say that a variable bound in E is *disjunctive (conjunctive)* if its right-hand side is a disjunction (conjunction). Let $scc(Y) = \{X \mid X \rightsquigarrow_E^* Y \rightsquigarrow_E^* X\}$, the set of variables belonging to the same SCC as Y .

Proposition 1 *The following conditions on σ -blocks E ensure SCC-consistency:*

1. All cycles of \rightsquigarrow_E contain only disjunctive variables.
2. All cycles of \rightsquigarrow_E contain only conjunctive variables.
3. If $\sigma = \nu$ and for all conjunctive Y in $bv(E)$ the following holds:
 - (a) $\{X \mid X \rightsquigarrow_E Y\} \subseteq scc(Y)$
 - (b) if $X, X' \in scc(Y)$ are disjunctive then X and X' belong to a cycle of \rightsquigarrow_E containing only disjunctive variables.

The next section will show examples of equation systems falling in the first two classes. We have no concrete examples of the third but included it to emphasize that there might be other useful classes of SCC-consistent blocks.

Lemma 1 (Compaction Algorithm) *Assume E is an SCC-consistent σ -block. Let $S \subseteq bv(E)$ be a maximal SCC of \rightsquigarrow_E . Then there exists a Boolean equation system E' of size $O(|E \upharpoonright S|)$ with $bv(E') = S$ such that*

$$\begin{aligned} fv(E') &\subseteq fv(E \upharpoonright S), \\ \forall \rho : fv(E \upharpoonright S) \rightarrow \mathbb{O}. \llbracket E \upharpoonright S \rrbracket \rho &= \llbracket E' \rrbracket \rho, \\ \rightsquigarrow_{E'} &\text{ is acyclic.} \end{aligned}$$

Moreover, if E is given by a standard representation and membership of S can be tested in time $O(1)$, E' can be found in time $O(|E \upharpoonright S|)$.

Proof. (Sketch) Take $S_\wedge = \{X \in S \mid X \text{ is conjunctive in } E\}$, and $S_\vee = \{X \in S \mid X \text{ is disjunctive in } E\}$. Trivially, $S = S_\wedge \cup S_\vee$. The following property can be shown:

If $\sigma = \mu$ and \rightsquigarrow_E is cyclic on S , then $\llbracket E \upharpoonright S \rrbracket \rho(X) = 1$ for all $X \in S$, if and only if, there exists $Y \in S_\vee$ and $Z \notin S$, such that $Z \rightsquigarrow_E Y$ and $\rho(Z) = 1$.

Notice that if \rightsquigarrow_E is acyclic on the SCC S , S must be a singleton $\{X\}$ that does not appear in its rhs. (A test that of course can be performed in linear time.) The direction from right to left is obvious from SCC-consistency ($\rho(Z) = 1$ enforces $\llbracket E \upharpoonright S \rrbracket \rho(Y) = 1$ which in turn implies $\llbracket E \upharpoonright S \rrbracket \rho(X) = 1$ for all $X \in S$). For the other direction we assume by way of contradiction that there is no Y in S_\vee with a $Z \notin S$ such that $Z \rightsquigarrow_E Y$ and $\rho(Z) = 1$, and then argue that $\llbracket E \upharpoonright S \rrbracket \rho(X) = 0$ for all $X \in S$. First we look at all $Z \notin S$ that appear in the rhs. of a conjunctive variable Y . Clearly if just one of these have $\rho(Z) = 0$ then $\llbracket E \upharpoonright S \rrbracket \rho(Y) = 0$ and by SCC-consistency, $\llbracket E \upharpoonright S \rrbracket \rho(X) = 0$ for all $X \in S$. If they all have $\rho(Z) = 1$ they could, as far as evaluation goes, just as well be removed. Also the $Z \notin S$ appearing in right-hand sides of disjunctive Y could be removed since they must by assumption have $\rho(Z) = 0$. The \rightsquigarrow_E -graph that remains after removing these two classes of variable would be an SCC with no free variables. The minimum fixed point of any such SCC, that is not a singleton with rhs. $\bigwedge \emptyset$, is clearly 0.

A dual property holds for $\sigma = \nu$. The new Boolean equation system E' is now constructed by choosing some variable $X \in S$ and let its rhs. compute the value of all variables in S . The right-hand sides for the variables in $S \setminus \{X\}$ will simply be $\bigvee \{X\}$ or equivalently $\bigwedge \{X\}$. The rhs. of X is found as follows. If S is acyclic the single rhs. is unchanged. Otherwise, if $\sigma = \mu$ the rhs. will be a disjunction (conjunction if $\sigma = \nu$) over the set of $Z \notin S$ that occurs in right-hand sides of some $Y \in S_\vee$ ($Y \in S_\wedge$ if $\sigma = \nu$). Correctness of this transformation follows directly from the above property and its dual.

Computationally we first perform the linear time test for acyclicity and in case of failure simply collect all the rhs.'s of disjunctive (respectively conjunctive) variables of S in a list without removing duplicates, iterate through the list and remove all variables that belong to S . By assumption membership of S can be tested in constant time, and therefore the whole construction can be performed in linear time. \square

Theorem 1 (Inversion Algorithm) *If E is an SCC-consistent μ -block (ν -block) then it is possible in time $O(|E|)$ to find a ν -block (μ -block) \bar{E} of size $O(|E|)$ with free variables among $fv(E)$ such that for all environments ρ assigning values to the free variables of E we have $\llbracket E \rrbracket \rho = \llbracket \bar{E} \rrbracket \rho$.*

Proof. (Sketch) The algorithm proceeds as follows. First, the graph \rightsquigarrow_E is constructed in linear time. Second, the maximal strongly connected components are found in linear time using Tarjan's algorithm [Tar72]. The result of the algorithm is a table scc which makes it possible in constant time to get the identity $scc[X]$ of the strongly connected component of any variable. (The identity is a number between 0 and $k - 1$ if the graph contains k maximal strongly connected components.)

Thirdly, each strongly connected components is collapsed by the Compaction Algorithm. The resulting Boolean equation system \bar{E} has no cycles: This would violate the assumption of the strongly connected components being maximal. Moreover, this is all done in time $O(|E|)$ and the resulting Boolean equation system has size $O(|E|)$.

Relation	F.P. Inversion	Other	Reference
Weak bisimulation	$nm + \min(ln, m)m_\tau$	$lm^2 \log n$ $ln^2 \log n + ln^{2.376}$	[ESTT91] See note 1
Observation congruence	$nm + \min(ln, m)m_\tau$		
Simulation preorder	$nm + \min(ln, m)m_\tau$?	
Branching bisimulation	$(n + m_\tau)m$	nm	[GV90]
Weak prebisimulation	m^2	$n^4 m^2$	See note 2
Formula			
$\nu X. \mu Y. [a]((Q \wedge X) \vee (\neg Q \wedge Y))$			
$\nu X. \mu Y. (a)((Q \wedge X) \vee (\neg Q \wedge Y))$	m	nm	See note 3
Notes:			
1. From [KS90]. They give the bound $O(mn^2 \log n + mn^{2.376})$ but according to their proof m is used for bounding l .			
2. Naive fixed-point iteration. The solution using a Boolean equation system in [And93] gives $O(n^2 m^2)$.			
3. From [And94]. Similar fairness properties can also be checked for CTL in time $O(m)$ [CES86].			

Table 1. A comparison of the worst-case asymptotic upper bounds of algorithms obtained by using fixed-point inversion with algorithms from the literature. The following abbreviations are used: $n = |S|$ is the number of states, $m = |\rightarrow|$ is the number of transitions (assumed to exceed n), $m_\tau = |\overset{\tau}{\rightarrow}|$ is the number of τ -transitions, and $l = |L|$ is the number of labels.

Since $\leadsto_{\bar{E}}$ is acyclic it is not hard to see that \bar{E} will give the same value no matter whether it is interpreted as a minimum or a maximum fixed point. \square

We can use the Inversion Algorithm to efficiently compute alternating blocks of fixed points:

Theorem 2 (Inversion in Alternating Fixed Points) *If $E = E_\nu E_\mu$ is a Boolean equation system consisting of a ν -block E_ν and a μ -block E_μ such that E_μ is SCC-consistent, then the solution $\llbracket E \rrbracket \rho$ can be found in time $O(|E|)$.*

Proof. Use the inversion algorithm to convert E_μ into a ν -block \bar{E}_μ of size $O(|E_\mu|)$ in time $O(|E_\mu|)$. Then compute $\llbracket E_\nu \bar{E}_\mu \rrbracket \rho$ in time $O(|E_\nu \bar{E}_\mu|) = O(|E|)$ using the linear-time fixed-point finder. \square

4 Applications

In this section we show some important applications of the inversion algorithm. The results are summarized in table 1. We shall discuss some of them in detail below.

4.1 Weak Bisimulation

Given a labelled transition system $t = (S, L, \rightarrow)$ where S is a set of *states*, L a set of *labels* containing the special label τ , and $\rightarrow \subseteq S \times L \times S$ a transition relation

we can define a notion of weak bisimulation equivalence as the maximum fixed point νF of the monotonic function F on the powerset of $S \times S$ defined by

$$(p, q) \in F(R) \Leftrightarrow_{\text{def}} \begin{array}{l} \forall a \in L, p'. p \xrightarrow{a} p' \Rightarrow \exists q'. q \xrightarrow{a} q' \text{ and } (p', q') \in R, \\ \forall a \in L, q'. q \xrightarrow{a} q' \Rightarrow \exists p'. p \xrightarrow{a} p' \text{ and } (p', q') \in R. \end{array}$$

Here $\xrightarrow{\tau} =_{\text{def}} \xrightarrow{\tau^*}$, and for $a \neq \tau$, $\xrightarrow{a} =_{\text{def}} \xrightarrow{\tau} \xrightarrow{a} \xrightarrow{\tau}$. (See [Mil89] for details.)⁴ We can formulate the question of whether a particular pair (p, q) is in νF , i.e. whether p and q are weakly bisimilar, as a question of whether the variable $X_{p,q}$ has the solution 1 in the following Boolean equation system E_{\approx} with two alternating blocks (the ranges of indices are shown to the right):

$$\begin{array}{l} X_{p,q} = \nu \wedge \{Y_{a,p',q}^0 \mid a \neq \tau, p \xrightarrow{a} p'\} \cup \{Y_{p',q}^1 \mid p \xrightarrow{\tau} p'\} \cup \{X_{q,p}\} \\ \hspace{20em} ((p, q) \in S \times S) \\ Y_{a,p',q}^0 = \mu \vee \{Y_{p',q'}^1 \mid q \xrightarrow{a} q'\} \cup \{Y_{a,p',q'}^0 \mid q \xrightarrow{\tau} q'\} \\ \hspace{20em} (\exists p. p \xrightarrow{a} p', q \in S) \\ Y_{p,q}^1 = \mu \vee \{Y_{p,q'}^1 \mid q \xrightarrow{\tau} q'\} \cup \{X_{p,q}\} \\ \hspace{20em} ((p, q) \in S \times S) \end{array}$$

It is not hard to see that the size of E_{\approx} is $O(nm + \min(ln, m)m_{\tau})$ where $n = |S|$, $m = |\rightarrow|$, $m_{\tau} = |\xrightarrow{\tau}|$. Moreover, provided the transition system is represented as an edge-labelled directed graph using an array of adjacency lists, it is possible to compute a representation of E_{\approx} of the required form in linear time. We shall discuss this point in the next section.

Since the inner μ -block consists entirely of disjunctions it is SCC-consistent by proposition 1. Therefore the solution can be found in time $O(nm + \min(ln, m)m_{\tau})$ (assuming $n \leq m$) by theorem 2. We have chosen to make explicit the dependency of the algorithms on the number of labels. As can be seen from table 1 for (very) sparse graphs with $m = O(n)$ our algorithm has complexity $O(n^2)$ – independent of l – which improves on both of the other algorithms. For moderately sparse graphs with $m = O(n \log n)$ we get $\min(ln^2 \log n, n^2 \log^2 n)$ which is at least as good as [KS90] and better if l is asymptotically greater than $\log n$.

With a slight modification of the equations it is easy to obtain observation congruence and (weak) prebisimulation. Milner’s simulation preorder [Mil89, p.208] is defined by taking only the top clause in weak bisimulation above. Thus we can compute it within the same bound as for weak bisimulation.

The equations for branching bisimulation are attached as an appendix.

Relations of alternation depth one can be computed directly without fixed-point inversion in time linear in the size of the corresponding Boolean equation system. This yields for example an $O(mn)$ -algorithm for ready (bi-)simulation

⁴ Weak bisimulation is often defined between two different transition systems. However, it becomes more cumbersome to faithfully express the complexity of the algorithm in this case. If one of the transition systems is small, which could very well happen if it was a “specification” and the other an “implementation”, the complexity is actually better than the bound indicates.

which compares favourably with the $O(lmn)$ -algorithm of Bloom and Paige [BP92].

4.2 Implementational Aspects

When applying the Inversion Algorithm to compute behavioural relations it is necessary that the standard representation of the corresponding Boolean equation system be computed in linear time. This, of course, can be more or less difficult to achieve. For all the applications mentioned in this paper we claim that it *can* be done. The formal proof requires a lot of tedious and elaborate coding details. Here, we shall only sketch the basic mechanisms making it possible. (In each individual case of a behavioural relation it will probably often be possible to find simpler and more direct approaches.)

We shall assume that states, labels and transitions are represented by numbers 1 to n , 1 to l (the label τ having the number 1) and 1 to m . Three arrays *src*, *lbl*, and *tgt* represent the source, label, and target of each transition e . This representation requires $O(m)$ space on a RAM-model under the “uniform cost”-criterion [AHU74].

From such a representation it is possible in linear time (assuming $n \leq m$) to compute some auxiliary structures for the labelled transition system. First, however, it is convenient to introduce the concept of a *backward section*. A backward section s is a pair (a, p') such that there exists some p with $p \xrightarrow{a} p'$. Each section will be given a number from 1 to k . Notice that $k \leq \min(ln, m)$. Backward sections will be stored in arrays *lbl*, *btgt* and *brng* such that *lbl*[s] is the label a of section s , *btgt*[s] is the target p' of section s and *brng*[s] is a linked list representation of the set $\{p \mid p \xrightarrow{a} p'\}$. Since labels are represented by numbers from 1 to l these arrays can be constructed in time $O(k + l + m)$ using Radix sort [AHU74] to sort the edges on their labels before inserting them into *lbl* and *brng*. Furthermore we can construct in linear time an array *bwd* that associates each state p with a linked list of its backward sections $bwd[p] = \{s \mid p \in brng[s]\}$.

In order to construct a standard representation of the Boolean equation system E we basically need to find an address $@X$ for each variable of E , the contents of which is a tag $tag[@X] \in \{\vee, \wedge\}$ indicating whether X is disjunctive or conjunctive, and a linked list of (addresses of) the variables appearing in the right-hand side of X , *vars*[$@X$]. No matter the organization of the addresses $@X$ in memory, it is not difficult in linear time to change such a structure to an array of tags and variable lists as required.

We construct the addresses by considering the indexing sets of the variables. If the variable is of the form $X_{p,q}$ where p, q ranges freely over S , we take an $n \times n$ two-dimensional array X of memory capable of storing a tag and pointer to a linked list. If the variable is of the form $Y_{a,p',q}$ where q ranges freely over S and (a, p') over the backward sections, we take an $n \times k$ two-dimensional array Y , such that $Y[q, s]$ is the address $@Y_{a,p',q}$ for the variable $Y_{a,p',q}$ with *btgt*[s] = p' , *lbl*[s] = a . If the variable is of the form $Y_{p,a,p',q}$ where $p \xrightarrow{a} p'$ (for an example, see appendix) we take an $n \times m$ two-dimensional array Y such that $Y[q, e]$ is the address $@Y_{p,a,p',q}$ of the variable $Y_{p,a,p',q}$ with *src*[e] = p , *lbl*[e] = a , *tgt*[e] = p' .

What remains is to argue how each right-hand side can be found in linear time, i.e. how the list of *addresses* of variables can be found in linear time. This will be done by traversing one of the auxiliary structures depending on the form of the right-hand side. Consider for example the situation where the rhs. contains a list of variables of the form $\{Y_{a,p',q} \mid a \neq \tau, p \xrightarrow{a} p'\}$. The addresses for these variables are $Y[q, s]$ where s range over the backward sections $bwd[p]$ with $bbl[s] \neq \tau$. Clearly this list can be constructed in linear time.

4.3 Modal Assertions

The last application we shall comment on is model checking of assertions expressed in the modal μ -calculus [Koz83]. The task is to decide for a given assertion A and labelled transition system t whether t satisfies A . In [And94, VL92] it is shown how to translate this problem into a question of solving a Boolean equation system of size $O(|A||t|)$. We will not repeat the translation in detail but remark that box-modalities end up being conjunctions and diamond-modalities disjunctions. Thus for assertions as

$$\nu X. \mu Y. [a]((Q \wedge X) \vee (\neg Q \wedge Y)),$$

and

$$\nu X. \mu Y. \langle a \rangle((Q \wedge X) \vee (\neg Q \wedge Y)),$$

where Q is a constant, we will get a Boolean equation system consisting of an outer ν -block and an inner μ -block. For the first assertion each right-hand side in the μ -block will be a conjunction (arising from the box-modality) of variables from either the ν - or the μ -block depending on the valuation for the constant Q . The disjunction simply disappears. According to proposition 1 this makes the μ -block SCC-consistent. Similarly, for the second assertion each right-hand side will be a disjunction of variables from the ν - or the μ -block. This also gives rise to a SCC-consistent μ -block.

The above assertions, expressing the fairness properties that “along all a -paths Q will infinitely often hold” and “there exists an a -path along which Q holds infinitely often”, can then be checked in time $O(|t|)$ using the Inversion Algorithm. In general, any assertion of alternation depth two that after the translation yields an SCC-consistent inner block can be verified in linear time.

5 Fixed-Point Inversion with BDDs

The fixed-point inversion can also be used on symbolic representations of Boolean equation systems. We will describe how to do this using Binary Decision Diagrams (BDDs) [Bry92].

A Boolean equation system E will be given as a triple $(b_E, conj_E, disj_E)$ of BDD's representing the characteristic functions for \rightsquigarrow_E (actually $\rightsquigarrow_{E^{-1}}$) and the set of bound variables split into two sets of conjunctive respectively disjunctive variables. More precisely we assume given an encoding $enc : fv(E) \cup bv(E) \rightarrow \mathbb{O}^n$

of the variables of E using n bits. Using the variables $\{x_1, \dots, x_n, y_1, \dots, y_n\}$ in the BDD b_E and $\{x_1, \dots, x_n\}$ in BDD's $conj_E$ and $disj_E$ the conditions are that

$$b_E[enc(X)/\vec{x}, enc(Y)/\vec{y}] \text{ is valid, if and only if, } Y \rightsquigarrow_E X,$$

$conj_E[enc(X)/\vec{x}]$ is valid, if and only if, X is bound and conjunctive in E , and

$disj_E[enc(X)/\vec{x}]$ is valid, if and only if, X is bound and disjunctive in E .

A BDD b_E^* for the reflexive, transitive closure of \rightsquigarrow_E can be computed by a standard fixed-point iteration or by using iterative squaring [BCM⁺90] such that

$$b_E^*[enc(X)/\vec{x}, enc(Y)/\vec{y}]$$

holds, if and only if, there is an \rightsquigarrow_E -path from Y to X in E . Now, two variables X and Y will belong to the same strongly connected component, if and only if, $b_{scc}[enc(X)/\vec{x}, enc(Y)/\vec{y}]$ is valid, where

$$b_{scc} = b_E^* \wedge b_E^*[\vec{y}/\vec{x}, \vec{x}/\vec{y}].$$

(The last operation involves a re-ordering of the variables.) If σ is μ , the triple of BDD's for the fixed-point inverted equation system \bar{E} is now found by

$$b_{\bar{E}} = (\neg c \wedge b_E) \vee (c \wedge \neg b_{scc} \wedge \exists \vec{z}. b_{scc}[\vec{z}/\vec{y}] \wedge b_E[\vec{z}/\vec{x}] \wedge disj_E[\vec{z}/\vec{x}]),$$

$$conj_{\bar{E}} = conj_E \wedge \neg c$$

$$disj_{\bar{E}} = disj_E \vee (conj_E \wedge c),$$

where $c = \exists \vec{y}. b_E \wedge b_{scc}$ is the set of \vec{x} appearing on a cycle. The definition of $b_{\bar{E}}$ follows closely the proof of correctness of the Compaction Algorithm.

If σ is ν the defining clause for $conj_{\bar{E}}$ is $conj_E \vee (disj_E \wedge c)$ and for $disj_{\bar{E}}$ it is $disj_E \wedge \neg c$.

For an alternating equation system $E = E_\nu E_\mu$ consisting of two blocks represented by triples of BDDs we can use fixed-point inversion to find the solution efficiently. We proceed as follows. First E_μ is inverted yielding a triple of BDDs $(b_{\bar{E}_\mu}, conj_{\bar{E}_\mu}, disj_{\bar{E}_\mu})$ as described above. We then take $b_E = b_{E_\nu} \vee b_{\bar{E}_\mu}$, $conj_E = conj_{E_\nu} \vee conj_{\bar{E}_\mu}$ and $disj_E = disj_{E_\nu} \vee disj_{\bar{E}_\mu}$. The solution to E is now found iteratively from the initial approximation $b_\rho = 1$ representing the environment $\rho(X) = 1$ for all $X \in bv(E)$. In doing this we need an operation $eval(b_E, b_\rho)$ that finds a BDD for $\llbracket E \rrbracket \rho$ where E is the Boolean equation system represented by $(b_E, conj_E, disj_E)$ and ρ the environment represented by b_ρ . It can be defined as follows:

$$eval(b_E, b_\rho) = (conj_E \wedge \forall \vec{y}. b_E \rightarrow b_\rho[\vec{y}/\vec{x}]) \vee (disj_E \wedge \exists \vec{y}. b_E \wedge b_\rho[\vec{y}/\vec{x}]).$$

The iteration is now a simple loop:

$$b_\rho := 1$$

$$\mathbf{repeat} \ b'_\rho := b_\rho \quad b_\rho := eval(b_E, b_\rho) \quad \mathbf{until} \ b_\rho = b'_\rho$$

What we have achieved is that the solution of $E = E_\nu E_\mu$ can be found by *two* fixed-point iterations (one of them using iterative squaring), instead of the potential need of $|fv(E_\mu) \cap bv(E_\nu)|$ iterations of the innermost fixed point in the direct approach.

6 Conclusion

We have described an algorithm for finding in linear time solutions to Boolean equation systems consisting of two alternating blocks when the inner block is SCC-consistent. The algorithm is based on the idea of inverting a fixed point. We have shown a range of applications of the algorithm ranging from behavioural relations to model checking of modal assertions. This included an algorithm for checking weak bisimulation equivalence with the best known worst-case complexity for sparse graphs.

We further showed how fixed-point inversion can be used in BDD-based methods.

In [EJS93] Tarjan's algorithm for strongly connected components is also used in giving an improved model checker for fragments of the modal μ -calculus. However, in quite a different manner. Emerson et al finds " ν -" and " μ -cycles" to reduce the need for recomputation of inner fixed points. They also reach a different result. They focus on identifying a syntactic subset of the modal μ -calculus for which model checking can be done efficiently. We base our algorithm on the more semantic notion of SCC-consistent blocks covering a larger sublogic. In fact, we answer the question from their conclusion of whether there is a model checker for their logics L_1 and L_2 running in time $O(|A||t|)$ instead of $O(|A|^2|t|)$, in the affirmative – at least for alternation depth two. (The generalization to higher alternation depths could be interesting to carry out although of little practical interest.) It is unclear whether the algorithm of Emerson et al is amenable to BDD-based methods.

References

- [AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- [And93] Henrik R. Andersen. *Verification of Temporal Properties of Concurrent Systems*. PhD thesis, Department of Computer Science, Aarhus University, Denmark, June 1993. PB-445.
- [And94] Henrik R. Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1):3–30, April 1994.
- [BCM⁺90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 428–439. IEEE Computer Society Press, 1990.
- [BP92] Bard Bloom and Robert Paige. Computing ready simulations efficiently. 1992.
- [Bry92] Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *Computing Surveys*, 24(3):293–318, September 1992.
- [CDS92] Rance Cleaveland, Marion Dreimüller, and Bernhard Steffen. Faster model checking for the modal mu-calculus. In G. v. Bochmann and D. K. Probst, editors, *Proceedings of the 4th Workshop on Computer Aided Verification, CAV'92, June 29 - July 1, 1992, Montreal, Quebec, Canada*, volume 663 of *LNCS*, pages 383–394. Springer-Verlag, 1992.

- [CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
- [EJS93] E.A. Emerson, C.S. Jutla, and A.P. Sistla. On model-checking for fragments of μ -calculus. In Costas Courcoubetis, editor, *Proceedings of the 5th International Conference on Computer Aided Verification, CAV'93*, volume 697 of *LNCS*, pages 385–396. Springer-Verlag, 1993.
- [ESTT91] Klaus Estenfeld, Hans-Albert Schneider, Dirk Taubner, and Erik Tidén. Computer aided verification of parallel processes. In A. Pfitzmann and E. Raubold, editors, *VIS '91 Verlässliche Informationssysteme*, volume 271 of *Informatik Fachberichte*, pages 208–226, Darmstadt, 1991. Springer-Verlag.
- [GV90] J.F. Groote and F. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In M.S. Paterson, editor, *Proceedings of ICALP*, volume 443 of *LNCS*. Springer-Verlag, 1990.
- [Koz83] Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27, 1983.
- [KS90] P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.
- [Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.
- [Tar72] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 2(1), June 1972.
- [VL92] Bart Vergauwen and Johan Lewi. A linear algorithm for solving fixed-point equations on transition systems. In J.-C. Raoult, editor, *Proceedings of 17'th Colloquium on Trees in Algebra and Programming, CAAP'92, Rennes, France*, volume 581 of *LNCS*, pages 322–341. Springer-Verlag, 1992.

Appendix. Boolean Equation System for Branching Bisimulation

Branching bisimulation equivalence [GV90] is the largest fixed-point νF of the function on the powerset of $S \times S$ defined by:

$$\begin{aligned}
 (p, q) \in F(R) &\Leftrightarrow_{\text{def}} \\
 &\forall a \in L, p'. p \xrightarrow{a} p' \Rightarrow \\
 &\quad (a = \tau \text{ and } (p', q) \in R) \text{ or} \\
 &\quad (\exists q', q''. q \xrightarrow{\tau} q' \xrightarrow{a} q'' \text{ and } (p, q') \in R \text{ and } (p', q'') \in R) \\
 &\forall a \in L, q'. q \xrightarrow{a} q' \Rightarrow \\
 &\quad (a = \tau \text{ and } (p, q') \in R) \text{ or} \\
 &\quad (\exists p', p''. p \xrightarrow{\tau} p' \xrightarrow{a} p'' \text{ and } (p', q) \in R \text{ and } (p'', q') \in R)
 \end{aligned}$$

Again, we shall need an inner μ -block to express the “weak” transitions $\xrightarrow{\tau}$ and use a ν -block for the overall νF . The required Boolean equation system has

variables

$$\begin{aligned}
& \{X_{p,q}^0 \mid p, q \in S\} \cup \\
& \{X_{p,q}^1 \mid p, q \in S\} \cup \\
& \{X_{p,p',q}^2 \mid p, p', q \in S \text{ and } p \xrightarrow{\tau} p'\} \cup \\
& \{Y_{p,a,p',q}^0 \mid p, p', q \in S, a \in L \text{ and } p \xrightarrow{a} p'\} \cup \\
& \{Y_{p,a,p',q}^1 \mid p, p', q \in S, a \in L \text{ and } p \xrightarrow{a} p'\} \cup \\
& \{Y_{p',a,q}^2 \mid a \in L, q \in S, \exists p \in S. p \xrightarrow{a} p'\}.
\end{aligned}$$

A total of, $|S|^2 + |S|^2 + |\xrightarrow{\tau}||S| + |\rightarrow||S| + |\rightarrow||S| + |\rightarrow||S| = O(|\rightarrow||S|)$ variables.
The right-hand sides are defined as follows:

$$\begin{aligned}
X_{p,q}^0 &= \nu X_{p,q}^1 \wedge X_{q,p}^1 \\
X_{p,q}^1 &= \nu \bigwedge \{X_{p,p',q}^2 \mid p \xrightarrow{\tau} p'\} \cup \{Y_{p,a,p',q}^0 \mid p \xrightarrow{a} p' \text{ and } a \neq \tau\} \\
X_{p,p',q}^2 &= \nu X_{p',q}^0 \vee Y_{p,\tau,p',q}^0 \\
Y_{p,a,p',q}^0 &= \mu \bigvee \{Y_{p,a,p',q'}^0 \mid q \xrightarrow{\tau} q'\} \cup \{Y_{p,a,p',q}^1\} \\
Y_{p,a,p',q}^1 &= \mu X_{p,q}^0 \wedge Y_{p',a,q}^2 \\
Y_{p',a,q}^2 &= \mu \bigvee \{X_{p',q'}^0 \mid q \xrightarrow{a} q'\}
\end{aligned}$$

The total size of the Boolean equation system is $O(|\rightarrow|(|\xrightarrow{\tau}| + |S|))$. This means that the Inversion Algorithm will solve it in time $O(|\rightarrow|(|\xrightarrow{\tau}| + |S|))$.