A local checking algorithm for Boolean Equation Systems

J.Cl. Fernandez and Laurent Mounier*

Abstract

Given a boolean equation system $\mathcal{E}$ and one of its bound variables X_{init}, we propose a local algorithm for computing the solution $\delta(X_{\text{init}})$ of $\mathcal{E}$. This algorithm relies on depth-first traversals of the dependency graph of $\mathcal{E}$: the boolean equation system is solved during the depth-first search, and the algorithm terminates as soon as the value obtained for X_{init} is correct. Two applications are presented in the framework of program verification: bisimulation checking, and model-checking for the alternation-free μ-calculus. This algorithm has been implemented within the CÆSAR-ALDÉBARAN toolbox and experimental results on rather large examples demonstrated its practical interest.

1 Introduction

In 1991 we proposed an algorithm for computing various bisimulation relations between two labeled transition systems (lts) [FM91]. The main feature of this algorithm is to rely on a depth-first traversal of a synchronous product of the two ltss. Thus, it does not require to store in memory their whole set of states and transitions during the computation, which improved the memory efficiency of the usual methods based on partition refinement. This algorithm has been implemented within the CÆSAR-ALDÉBARAN toolbox, and in spite of its theoretical worst-case time complexity, it was successful in practice for on-the-fly verification of non trivial LOTOS programs.

From these interesting results, the initial aim of the present work was to re-use this algorithm for alternation-free μ-calculus [Koz83] model checking. Although previous works have already been carried out in this area (see for instance [Cle90, CS91, Lar92, BVW94, And92], our goal was twofold: first, to propose an algorithm detailed enough to allow a straightforward implementation, but also to obtain a convenient framework for designing on-the-fly algorithms for checking either equivalence relations or temporal logic formulae.

It turns out that *boolean equation systems* (bes), with mixed fixpoint equations, are a suitable framework to formalize such algorithms [And92, VWL94, AV95]. More precisely, we present in this paper a general algorithm for computing the solution $\delta(X_{\text{init}})$ of a given bes $\mathcal{E}$, where X_{init} is a distinguished variable of $\mathcal{E}$. This algorithm is restricted to so called mscc-consistent bess: each maximal strongly connected component of its dependency graph is either a least or greatest fixpoint equations set. The mscc-consistence ensures that there are no *alternating fixed point* [CS91, EL86]. Finally, two applications of this algorithm are proposed, respectively devoted to bisimulation checking and alternation-free μ-calculus [Koz83] model checking.

This algorithm is a straight generalization of the one presented in [FM91]: it relies on a depth-first traversal of the dependency graph of the bes under consideration starting from variable X_{init}. During this traversal a solution is computed for each variable of $\mathcal{E}$, following a postfixed order. The main idea

*Verimag, Miniparc - ZIRST, rue Lavoisier, 38330 Montbonnot, FRANCE; tel: (33) 76.90.96.42; e-mail: Jean-Claude.Fernandez@imag.fr,Laurent.Mounier@imag.fr

is then to minimize the number of traversals required to compute $\delta(X_{\text{init}})$. In particular, we show that when the dependency graph is reducible to a tree (i.e., when already computed results never need to be re-used), then a single traversal is always sufficient.

The paper is organized as follows: in section 2, we recall some definitions about boolean equation systems. We also give some propositions to establish the correctness of our algorithm. The algorithm itself is presented in section 3. We propose in section 4 two applications: equivalence-checking for bisimulation relations and model-checking for the μ-calculus. Finally, we present two case studies in the last section and we show, for one of them, that previous analysis results have been improved.

2 Boolean Equation Systems

We recall in this section some basic definitions on boolean equation systems, and we propose a resolution method leading to a local algorithm.

2.1 Definition

Let **Var** be a set of variables. The language of formulae is defined by the following abstract grammar where X ranges over **Var**:

$\Phi \quad ::= \quad \texttt{true} \mid \texttt{false} \mid X \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \neg\Phi$

In the sequel, we suppose, for technical convenience, that the formulae Φ are in positive normal form. For a given formula Φ, let $\mathbf{fv}(\Phi)$ be its set of *free variables* defined by structural induction: $\mathbf{fv}(\texttt{true}) = \mathbf{fv}(\texttt{false}) = \emptyset, \mathbf{fv}(X) = \{X\}, \mathbf{fv}(\neg\Phi) = \mathbf{fv}(\Phi), \mathbf{fv}(\Phi_1 \wedge \Phi_2) = \mathbf{fv}(\Phi_1 \vee \Phi_2) = \mathbf{fv}(\Phi_1) \cup \mathbf{fv}(\Phi_2)$.

A *Boolean Equation System* (bes, for short), $\mathcal{E}$, is a set $\{X_i =_{\sigma_i} \Phi_i\}_{i\in[1,n]}$, where $\sigma_i \in \{\mu, \nu\}$. An equation $X =_\mu \Phi$ (resp. $X =_\nu \Phi$) is called a least (resp. greatest) fixpoint equation. $\mathbf{bv}(\mathcal{E}) = \{X_1, ..., X_n\}$ is the set of bound variables of $\mathcal{E}$. Similarly, $\mathbf{fv}(\mathcal{E}) = \bigcup_{i=1}^{n} \mathbf{fv}(\Phi_i) \setminus \mathbf{bv}(\mathcal{E})$ is the set of free variables of $\mathcal{E}$. A bes is well-formed iff all left hand sides variables are different, i.e. if $X_i =_{\sigma_i} \Phi_i$ and $X_i =_{\sigma_j} \Phi_j$ then $i = j$. A bes is *closed* iff $\mathbf{fv}(\mathcal{E}) = \emptyset$. All the bess we consider are well-formed and closed.

Let $\mathcal{B} = (\{0,1\}, \vee, \wedge, <, \neg, 0, 1)$ be the uniquely complemented lattice of boolean values, where $0 < 1$, $\vee$ is the least upper bound, $\wedge$ the greatest lower bound and $\neg$ the negation operator.

For a formula Φ and an environment $\rho : \mathbf{Var} \to \mathcal{B}$, the boolean value $[[\Phi]](\rho)$ is defined by structural induction on Φ : $[[\texttt{true}]](\rho) = 1, [[\texttt{false}]](\rho) = 0, [[X]](\rho) = \rho(X), [[\Phi_1 \vee \Phi_2]](\rho) = [[\Phi_1]](\rho) \vee [[\Phi_2]](\rho), [[\Phi_1 \wedge \Phi_2]](\rho) = [[\Phi_1]](\rho) \wedge [[\Phi_2]](\rho), [[\neg\Phi]](\rho) = \neg[[\Phi]](\rho)$.

2.2 The direct graph of dependencies

Given a bes $\mathcal{E}$ and a bound variable X_{init} of $\mathcal{E}$, we consider in the following the lts $\mathcal{G}_\mathcal{E} = (\mathcal{A}_{X_{\text{init}}}, \rightarrow_\mathcal{E}, X_{\text{init}})$ defined as follows :

- X_{init} is the distinguished variable.
- $\rightarrow_\mathcal{E}$ is the dependency relation defined on the bound variables of $\mathcal{E}$: $X_i \rightarrow_\mathcal{E} X_j$ iff $X_j \in \mathbf{fv}(\Phi_i)$.
- $\mathcal{A}_{X_{\text{init}}}$ is the subset of $\mathcal{E}$ variables, reachable from X_{init} by transitive closure of $\rightarrow_\mathcal{E}$.

Let $<$ be the *postfixed* order produced by any depth-first traversal of $\mathcal{G}_\mathcal{E}$, starting from X_{init}: $X_i < X_j$ iff X_i is popped before X_j during the traversal. Note that X_{init} is the greatest element of $\mathcal{A}_{X_{\text{init}}}$ w.r.t. relation $<$.

We also consider the set of *Maximal Strongly Connected Components* (mscc for short) of lts $\mathcal{G}_\mathcal{E}$: two variables X_i, X_j of $\mathbf{bv}(\mathcal{E})$ belong to the same mscc iff $X_i \rightarrow_\mathcal{E}^* X_j$ and $X_j \rightarrow_\mathcal{E}^* X_i$.

The mscc relation is an equivalence relation on $\mathbf{bv}(\mathcal{E})$. Let $\mathcal{E}_1, ..., \mathcal{E}_k$ denote the partition of $\mathbf{bv}(\mathcal{E})$ w.r.t. this relation, and let $\mathcal{G}_\mathcal{E}^{\text{red}}$ be the quotient of lts $\mathcal{G}_\mathcal{E}$ modulo this relation: states of $\mathcal{G}_\mathcal{E}^{\text{red}}$ are msccs of $\mathcal{G}_\mathcal{E}$, and there is a transition from $\mathcal{E}_i$ to $\mathcal{E}_j$ iff there is a variable $X_i \in \mathcal{E}_i$ and a variable $X_j \in \mathcal{E}_j$ such that $X_i \rightarrow_\mathcal{E} X_j$. It is easy to see that $\mathcal{G}_\mathcal{E}^{\text{red}}$ is a direct acyclic graph (dag).

Each mscc $\mathcal{E}_l$ can be viewed as a bes $\{X_i =_{\sigma_i} \Phi_i\}_{X_i \in \mathcal{E}_l}$. In the sequel we denote by $\mathcal{E}_l$ either the mscc of $\mathcal{G}_\mathcal{E}$, or the bes it represents. Moreover, a bes $\mathcal{E}$ will be said *mscc-consistent* iff the following holds:

- each mscc $\mathcal{E}_l$ of $\mathcal{E}$ contains either only least fix-point (lfp) equations or only greatest fix-point equations (gfp), formally there exists $J_l \subseteq [1, n]$ such that $\mathcal{E}_l = \{X_i =_{\sigma_i} \Phi_i\}_{i \in J_l}$ and $\forall i_1, i_2 \in J_l \,.\, \sigma_{i_1} = \sigma_{i_2}$;
- negation operators only appear on free variables of a given mscc of $\mathcal{E}$.

In the sequel, the bes we consider are always supposed to be mscc-consistent.

For each bes $\mathcal{E}_l$ we also define a notion of *root* w.r.t. the $<$ relation:

- $\mathbf{root\_scc}(\mathcal{E}_l)$ denotes the maximal element of $\mathcal{E}_l$ w.r.t the $<$ relation, i.e. it is the first reached and the last analyzed during a depth-first traversal.
- $\mathbf{root}(\mathcal{E}_l)$ denotes the set of roots of each elementary cycle of $\mathcal{E}_l$:

$$\mathbf{root}(\mathcal{E}_l) = \{X_i \in \mathbf{bv}(\mathcal{E}_l) \mid \exists X_j \in \mathbf{bv}(\mathcal{E}_l) \,.\, X_j < X_i \ \wedge \ X_j \rightarrow_\mathcal{E} X_i\}$$

The relation $<$ between the variables of $\mathcal{E}$ can now be extended to mscc themselves, defining that $\mathcal{E}_i < \mathcal{E}_j$ iff $\mathbf{root\_scc}(\mathcal{E}_i) < \mathbf{root\_scc}(\mathcal{E}_j)$. Note that: $\mathcal{E}_i < \mathcal{E}_j \not\Rightarrow \forall X_i \in \mathcal{E}_i \,, \forall X_j \in \mathcal{E}_j \,, X_i < X_j$.

Finally, a mscc $\mathcal{E}_j$ is said *reducible to a tree* w.r.t. the relation $<$ iff each of its states has at most one predecessor by relation $\rightarrow_\mathcal{E}$ greater than itself:

$$\mathbf{reducible}(\mathcal{E}_j) \equiv \forall X_i, X_j, X_k \,.\, \neg(X_j \rightarrow_\mathcal{E} X_i \ \wedge \ X_k \rightarrow_\mathcal{E} X_i \ \wedge \ X_i < X_j \ \wedge \ X_i < X_k)$$

A dependency graph is said reducible to a tree iff each of its mscc is. Now, we have the following (straightforward) results:

Proposition 2.1

(i) *the set of free variables of a mscc $\mathcal{E}_i$ satisfies:* $\forall \mathcal{E}_i \,.\, \mathbf{fv}(\mathcal{E}_i) \subseteq \{\mathbf{bv}(\mathcal{E}_j) \mid \mathcal{E}_j < \mathcal{E}_i\}$

(ii) *If the lts $\mathcal{G}_\mathcal{E}$ is reducible to a tree:* $\forall \mathcal{E}_i \,.\, \mathbf{fv}(\mathcal{E}_i) \subseteq \{\mathbf{root\_scc}(\mathcal{E}_j) \mid \mathcal{E}_j < \mathcal{E}_i\}$

2.3 Solution of a bes

Let $\mathcal{E} = \{X_i =_{\sigma_i} \Phi_i\}_{i \in [1,n]}$ a mscc-consistent bes, and $\mathcal{E}_1, \ldots, \mathcal{E}_k$ its msccs.

The solution of $\mathcal{E}$ is defined as follows:

- the solution of an equation $X_i =_{\sigma_i} \Phi_i$ is a function $\delta : \mathtt{Var} \rightarrow \mathcal{B}$ such that, if $\sigma_i = \nu$ (resp. $\sigma_i = \mu$) then $\delta(X_i)$ is the largest (resp. the least) value satisfying $\delta(X_i) = [[\Phi_i]](\delta)$.

- δ is a (global) solution of $\mathcal{E}$ iff it is a solution of each of its equations.

The existence of such a solution is straightforward when $\mathcal{E}$ is mscc-consistent:

- For each mscc $\mathcal{E}_l$, boolean functions Φ_i appearing in right hand side of the equations of $\mathcal{E}_l$ are monotonic (since negation operators are only applied on free variables). Thus, given any environment $\rho_l : \mathbf{Var} \to \mathcal{B}$ assigning free variables of $\mathcal{E}_l$, there exists a unique solution δ_{l,ρ_l} for bes $\mathcal{E}_l$ (since $\mathcal{E}_l$ is either a greatest or a least fix-point equations set).

- According to proposition 2.1, each free variable of $\mathcal{E}_l$ is a bound variable of a mscc $\mathcal{E}_m$ with $\mathcal{E}_m < \mathcal{E}_l$, and $\mathbf{fv}(\mathcal{E}_1) = \emptyset$. Since relation $<$ is a total order, the global solution δ of $\mathcal{E}$ can be obtained by computing (in increasing order) each solution δ_{l,ρ_l}, where ρ_l is defined as the union of solutions δ_{m,ρ_m} computed so far:

$$\delta = \bigcup_{1 \leq l \leq k} \delta_{l,\rho_l} \text{ where } \rho_l = \bigcup_{\mathcal{E}_m < \mathcal{E}_l} \delta_{m,\rho_m}$$

To compute the solution δ_{l,ρ_l} of each mscc $\mathcal{E}_l$, the algorithm we propose in the next section relies on the *Gauss-Seidel* resolution method: δ_{l,ρ_l} is obtained as the limit of a sequence $(\delta^i_{l,\rho_l})_{i \geq 0}$ computed iteratively, each δ^i_{l,ρ_l} being obtained following the total order $<$ on variables of $\mathcal{E}_l$ (see below).

However, instead of computing in turn each δ_{l,ρ_l}, we consider the sequence of functions δ^i, which converges to δ, and defined as the union of solutions δ^i_{l,ρ_l} computed so far. The underlying idea is then to try to minimize the number of iteration steps required to obtain a *correct* value of $\delta^i(X_{\text{init}})$, i.e., equal to $\delta(X_{\text{init}})$.

This approach can be formalized as follows:

- Let $\mathcal{E}_l$ be a mscc of $\mathcal{E}$ and $\rho_l : \mathbf{Var} \to \mathcal{B}$ an environment such that $\rho_l(X)$ is defined for each variable X in $\mathbf{fv}(\mathcal{E}_l)$. The solution δ_{l,ρ_l} is the limit of the sequence $(\delta^i_{l,\rho_l})_{i \geq 0}$ where:

$$\begin{array}{lcl} \delta^0_{l,\rho_l}(X) & = & \left\{ \begin{array}{l} 0 \text{ if } X \notin \mathbf{fv}(\mathcal{E}_l) \text{ and } \mathcal{E}_l \text{ is a lfp} \\ 1 \text{ if } X \notin \mathbf{fv}(\mathcal{E}_l) \text{ and } \mathcal{E}_l \text{ is a gfp} \\ \rho_l(X) \text{ otherwise} \end{array} \right. \\ \\ \delta^{i+1}_{l,\rho_l}(X_j) & = & [\![\Phi_j]\!](\delta^{i,j}_{l,\rho_l}) \text{ where } \delta^{i,j}_{l,\rho_l}(X_k) = \left\{ \begin{array}{l} \delta^i_{l,\rho_l}(X_k) \text{ for } X_k \geq X_j \\ \delta^{i+1}_{l,\rho_l}(X_k) \text{ for } X_k < X_j \end{array} \right. \end{array}$$

- the global solution of a bes $\mathcal{E}$ of mscc $\mathcal{E}_1 \ldots \mathcal{E}_k$ is then the limit of the sequence $(\delta^i)_{i \geq 0}$ defined as follows:

$$\delta^i = \bigcup_{1 \leq l \leq k} \delta^i_{l,\rho_l} \text{ where } \rho_l = \bigcup_{\mathcal{E}_m < \mathcal{E}_l} \delta^i_{m,\rho_m}$$

  According to proposition 2.1, environment ρ_l is defined for each free variable of $\mathcal{E}_l$ as far as δ^i_{m,ρ_m} functions are computed in increasing order on the mscc of $\mathcal{E}$.

Intuitively, computation of function δ^i will require at most i traversals of lts $\mathcal{G}_\mathcal{E}$. However, the exact number of iterations steps required to compute $\delta(X_{\text{init}})$ is given in Section 2.4.

2.4 Stability

For each mscc $\mathcal{E}_l$ of a bes $\mathcal{E}$, we now define a notion of stability. Informally, $\mathcal{E}_l$ is stable at level i iff for all roots X of its elementary cycles, $\delta^i_{l,\rho_l}(X) = \delta_{l,\rho_l}(X)$.

More formally, predicates **stable** and **stable_scc** are defined as follows for all $i \geq 1$:

$$\begin{aligned} \mathbf{stable}^i(\mathcal{E}_l) &\equiv (\forall X \,.\, X \in \mathbf{root}(\mathcal{E}_l) \Rightarrow \delta^i_{l,\rho_l}(X) = \delta^{i-1}_{l,\rho_l}(X)) \\ \mathbf{stable\_scc}^i(\mathcal{E}_l) &\equiv (\forall m \,.\, \mathcal{E}_m < \mathcal{E}_l \Rightarrow \mathbf{stable}^i(\mathcal{E}_m)) \end{aligned}$$

Proposition 2.2 relates this notion of stability to the computation of the solution δ_{l,ρ_l} of a bes $\mathcal{E}_l$.

Proposition 2.2 *Let $\mathcal{E}$ be a (mscc-consistent) bes, and $\mathcal{E}_1, \mathcal{E}_2, \ldots \mathcal{E}_k$ its msccs. Then, for all mscc $\mathcal{E}_l$:*

(i) $\mathbf{fv}(\mathcal{E}_l) = \emptyset \Rightarrow \delta^1_{l,\rho_l} = \delta_{l,\rho_l}$

(ii) $\forall i \geq 1 \,.\, (\mathbf{stable\_scc}^i(\mathcal{E}_l) \wedge \mathbf{stable}^i(\mathcal{E}_l)) \Rightarrow \delta^i_{l,\rho_l} = \delta_{l,\rho_l}$

(iii)

$$\forall i \geq 1 \,.\, \mathbf{stable\_scc}^i(\mathcal{E}_l) \Rightarrow \forall X \in \mathcal{E}_l \,.\, \begin{cases} \delta^i_{l,\rho_l}(X) = 0 \;\Rightarrow\; \delta_{l,\rho_l}(X) = 0 & \text{if } \mathcal{E}_l \text{ is a gfp} \\ \delta^i_{l,\rho_l}(X) = 1 \;\Rightarrow\; \delta_{l,\rho_l}(X) = 1 & \text{if } \mathcal{E}_l \text{ is a lfp} \end{cases}$$

Proposition 2.3 provides an upper bound on the number of iterations required to compute locally the solution of a bes. However, as discussed in section 3, this bound will only correspond to a worst-case complexity of our algorithm.

Proposition 2.3 *Let $\mathcal{E}$ be a (mscc-consistent) bes, and $\mathcal{E}_1, \mathcal{E}_2, \ldots \mathcal{E}_k$ its msccs. Let n be the total number of roots of each msccs: $n = \sum_{i=1}^{k} \mid \mathbf{root}(\mathcal{E}_i) \mid$. Then, $\mathbf{stable\_scc}^n(\mathcal{E}_k) \wedge \mathbf{stable}^n(\mathcal{E}_k)$.*

Finally, when the dependence graph associated to $\mathcal{E}$ is reducible to a tree, the solution $\delta(X_{\text{init}})$ can always be computed in a single iteration step (i.e., a depth-first traversal of $\mathcal{G}_\mathcal{E}$):

Proposition 2.4 *Let $\mathcal{E}$ a (mscc-consistent) bes, and $\mathcal{E}_1, \mathcal{E}_2, \ldots \mathcal{E}_k$ its msccs. If the lts $\mathcal{G}_\mathcal{E}$ is reducible to a tree then $\delta^1(X_{init}) = \delta(X_{init})$.*

See [FM95] for a proof of these propositions.

3 Algorithm

We present in this section a local algorithm for computing the solution $\delta(X_{\text{init}})$ for a given bound variable X_{init} of a bes $\mathcal{E}$. The approach followed is the one described in the previous section: a sequence of functions $\delta^i(X)$ is computed by successive iterations, using a suitable order relation on bound variables of $\mathcal{E}$.

This algorithm proceeds by performing depth-first traversals of the dependency graph $\mathcal{G}_\mathcal{E}$ associated to bes $\mathcal{E}$: each traversal i of the algorithm computes the value of $\delta^i(X)$ for all bound variables X. However:

- Correctness of the results is guaranteed only if $\mathcal{E}$ is mscc-consistent, so this check has also to be performed (when not ensured by hypothesis on $\mathcal{E}$) ;
- The algorithm should terminate when the computed function δ^i is such that $\delta^i(X_{\text{init}}) = \delta(X_{\text{init}})$. This condition holds either for $i = 1$ if $\mathcal{G}_\mathcal{E}$ is reducible to a tree, or when the mscc $\mathcal{E}_k$ containing X_{init} satisfies $\mathbf{stable\_scc}^i(\mathcal{E}_k)$ and $\mathbf{stable}^i(\mathcal{E}_k)$. Therefore, these checks have also to be performed.

We show in the sequel that all these verifications can be carried out during the computations of functions δ^i. Moreover, since the algorithms only rely on depth first traversals, it does not require a prior generation of lts $\mathcal{G}_\mathcal{E}$, nor even to necessarily store its whole set of states and transitions.

Starting from an usual depth-first traversal algorithm, we add the detection of msccs and the computation of the solution.

To allow the computation of the functions δ^i during a dfs, several data structures are required:

- A stack Γ, with elements in $Q \times 2^Q \times \mathbb{N} \times \mathbb{N}$ (the final state X of the current execution sequence σ, its pending successors, its depth in σ and the minimal depth reachable from X). If $\gamma \in \Gamma$ then let $\gamma = (\text{state}(\gamma), \text{succ}(\gamma), \text{deep}(\gamma), \min(\gamma))$ and let γ_0 be the initial element. We write $X \in \Gamma$ for $\exists \gamma \in \Gamma \ . \ X = \text{state}(\gamma)$. The stack Γ is managed through usual operations "push", "pop" and "top".
- A set $V \subseteq Q$, used to store the visited states which are no longer in Γ.
- A set $\text{Scc} \subseteq Q$, intended to store the roots of each mscc of $\mathcal{G}_\mathcal{E}$ ($\text{Scc} = \bigcup_{\mathcal{E}_i} \mathbf{root\_scc}(\mathcal{E}_i)$).
- A set $R \subseteq Q$, intended to store the roots of each elementary cycle of $\mathcal{G}_\mathcal{E}$ ($R = \bigcup_{\mathcal{E}_i} \mathbf{root}(\mathcal{E}_i)$).
- Sets $\bot$ and $\top$, subsets of Q intended to store the states X whose values $\delta(X)$ are already known (and respectively equal to 0 and 1).
- a variable *depth* to keep track of the current depth of the stack.
- boolean variables *stable*, *stable_scc* and *reducible* those values are set according to the definition of previous section.

We denote by "post" the function delivering the successors list of a given state X of $\mathcal{G}_\mathcal{E}$: $\text{post}(X) = \{X' \mid X \rightarrow_\mathcal{E} X'\}$. Furthermore, we use the function $Fix : Q \rightarrow \{\nu, \mu\}$; for each variable X defined by the equation $X =_\sigma \Phi$, $Fix(X) = \sigma$.

The following function performs an iteration step of the algorithm: at the end of the i^{th} call a function $D(X)$ is available, such that for each state X, $D(X) = \delta^i(X)$. It also returns true if no more iterations are required, i.e., if $D(X_{\text{init}}) = \delta(X_{\text{init}})$.

Now, the algorithm steps are

step 1 The state Y belongs to Γ or V. In the last case, we do nothing ; In the first case, let γ' such that $\text{state}(\gamma') = Y$. We add Y to the root ($R := R \cup \{Y\}$) and we set D (Y) to 1 (resp. to 0) if Fix (Y) $= \nu$ (resp. Fix (Y) $= \mu$). If $\text{deep}(\gamma') < \min(\gamma)$ then $\min(\gamma) := \text{deep}(\gamma')$.

step 2: We perform the following statement:

- We compute D (X) $:= \Phi_X(D(X') \mid X' \in post(X))$.
- If (stable_scc and D (X) $= 1$ and Fix (X) $= \mu$) (resp. stable_scc and D (X) $= 0$ and Fix (X) $= \nu$) then we add X to $\top$ (resp. to $\bot$) and if ($X \in R$) we set stable to false.
- if $\text{depth}(\gamma) = \min(\gamma)$, then X is a root of a mscc. Thus, we set stable_scc to stable and we add X to the set Scc.
- if $\Gamma \neq \emptyset$ then $\min(\text{top}(\Gamma))$ is set to $\min(\min(\text{top}(\Gamma), \min(\gamma))$.

```
function evaluate (X_0 : state) returns boolean is
begin
   V := ∅ ; stable := true ; stable_scc := true ; reducible := true
   Γ := ∅ ; R := ∅ ;Scc := ∅
   depth := 1 ; push ((X_0, post (X_0),depth, depth), Γ)
   while (Γ ≠ ∅) loop
        γ := top (Γ) ; X := state(γ) ; S := succ (γ)
        if X ∈ ⊤ then D (X) := 1 elsif X ∈ ⊥ then D (X) := 0
        else
             if (S ≠ ∅) then
                  choose and remove Y in S
                  if (Y ∉ (V ∪ Γ)) then
                       depth := depth + 1 ; push ((Y, post (Y),depth, depth), Γ)
                  else
                       (* step 1 : check if Y is a mscc-root and update the depth
                       of the element in order to detect a mscc *)
                  end if
             else (* S = ∅ *)
                  depth := depth -1; pop (Γ) ; V := V ∪ {X}
                  (* step 2 :compute the solution associated with X, update the sets ⊤, ⊥, R
                  and update the variables stable, stable_scc and reducible. *)
             end if
        end if
   end loop
   return (reducible or stable_scc)
end
```

Finally, the local algorithm for computing $\delta(X_{\text{init}})$ is the following:

```
function compute_local_solution (X_0) returns boolean is
begin
   ⊥ := {X ∈ Q | post(X) = ∅ ∧ Φ_X = false}
   ⊤ := {X ∈ Q | post(X) = ∅ ∧ Φ_X = true}
   loop
        reliable := evaluate (X_0)
   while ¬ reliable
   end loop
   return (D(X_0))
end
```

The correctness of this algorithm is a consequence of the propositions established in the previous section, and its complexity is given by the following proposition:

Proposition 3.1 *Let $\mathcal{G}_\mathcal{E}$ be the dependency graph of a bes $\mathcal{E}$. If n, m and n_r denote respectively the number of states, transitions and roots of elementary cycles of $\mathcal{G}_\mathcal{E}$ then:*

- *The time complexity of function compute_local_solution is $O(m * n_r)$ in the general case, and $O(m)$ when $\mathcal{E}$ is reducible to a tree.*
- *Its memory complexity is $O(m)$.*

4 Applications

In this section, we propose two applications of this algorithm. More precisely, we show how bisimulation-checking and model-checking for the alternate μ-calculus are characterized by a mscc-consistent bes.

4.1 Bisimulation-checking

Let $S_i = (Q_i, A, \xrightarrow{a}, q_{0i})_{i=1,2}$ be two ltss. Bisimulation equivalence $\sim$ is defined as the greatest fix-point of $\mathcal{B}$: $2^{Q_1 \times Q_2} \longrightarrow 2^{Q_1 \times Q_2}$ where:

$$\begin{aligned}\mathcal{B}(R) \quad = \quad \{(p_1,p_2) \mid \forall a \in A \,.\, \forall q_1 \,.\, (p_1 \xrightarrow{a} q_1 \;\Rightarrow\; \exists q_2 \,.\, (p_2 \xrightarrow{a} q_2 \;\wedge\; (q_1,q_2) \in R)) \\ \forall q_2 \,.\, (p_2 \xrightarrow{a} q_2 \;\Rightarrow\; \exists q_1 \,.\, (p_1 \xrightarrow{a} q_1 \;\wedge\; (q_1,q_2) \in R))\}\end{aligned}$$

Let $\mathcal{A}\mathrm{ct}(p) = \{a \mid \exists q.p \xrightarrow{a} q\}$ and $\mathcal{E}$ be the bes with $X_{q_{0_1},q_{0_2}}$ as distinguished variable:

$$X_{p,q} =_\nu \texttt{false} \qquad \text{if } \mathcal{A}\mathrm{ct}(p) \neq \mathcal{A}\mathrm{ct}(q)$$

$$X_{p,q} =_\nu \bigwedge_{a \in A} \bigwedge_{p \xrightarrow{a} p'} \bigvee_{q \xrightarrow{a} q'} X_{p',q'} \;\wedge\; \bigwedge_{a \in A} \bigwedge_{q \xrightarrow{a} q'} \bigvee_{p \xrightarrow{a} p'} X_{p',q'} \qquad \text{if } \mathcal{A}\mathrm{ct}(p) = \mathcal{A}\mathrm{ct}(q)$$

The mscc-consistency is straightforward since each equation is a gfp one. Moreover, we can show that $\delta(X_{p,q}) = 1$ iff $(p,q) \in \nu R.\mathcal{B}(R)$ [FM95]. Such a characterization can be done for all bisimulation equivalence, simulation preorder and simulation equivalence.

4.2 Model-checking for the alternation free μ-calculus

Syntax

We consider the following abstract syntax :

$$\varphi ::= \text{true} \mid \text{false} \mid \varphi \;\vee\; \varphi \mid \varphi \;\wedge\; \varphi \mid < a > \varphi \mid [a]\varphi \mid X \mid \nu X.\varphi \mid \neg\nu X.\varphi \mid \mu X.\varphi \mid \neg\mu X.\varphi$$

Semantics

Formulae are interpreted over a lts $S = (Q, A, \rightarrow, q_0)$ with respect to a given environment ρ. Env denotes the set of environments: Env : Var $\rightarrow 2^Q$. For $\rho \in$ Env, X, $Y \in$ Var and $R \subseteq Q$ we adopt the usual convention: $\rho[R/X](Y) =$ if $(X = Y)$ then R else $\rho(Y)$.

The meaning of a formula φ, noted $[\![\varphi]\!]_\rho$, represents the set of states satisfying φ when its free variables are assigned by ρ.

Function $[\![.]\!] : \mathcal{F} \rightarrow$ Env $\rightarrow 2^Q$ describes the semantics of a formula and is defined as follows:

$$
\begin{aligned}
[\![\text{true}]\!]_\rho &= Q \\
[\![\text{false}]\!]_\rho &= \emptyset \\
[\![\varphi_1 \wedge \varphi_2]\!]_\rho &= [\![\varphi_1]\!]_\rho \cap [\![\varphi_2]\!]_\rho \\
[\![\varphi_1 \vee \varphi_2]\!]_\rho &= [\![\varphi_1]\!]_\rho \cup [\![\varphi_2]\!]_\rho \\
[\![X]\!]_\rho &= \rho(X) \\
[\![\neg\varphi]\!]_\rho &= Q \setminus [\![\varphi]\!] \\
[\![< a > \varphi]\!]_\rho &= \{q \mid \exists q' \,.\, q \xrightarrow{a} q' \wedge q' \in [\![\varphi]\!]_\rho\} \\
[\![[a]\varphi]\!]_\rho &= \{q \mid \forall q' \,.\, q \xrightarrow{a} q' \Rightarrow q' \in [\![\varphi]\!]_\rho\} \\
[\![\mu X.\varphi]\!]_\rho &= \bigcap \{R \mid [\![\varphi]\!]_{\rho[R/X]} \subseteq R\} \\
[\![\nu X.\varphi]\!]_\rho &= \bigcup \{R \mid R \subseteq [\![\varphi]\!]_{\rho[R/X]}\}
\end{aligned}
$$

Definition of the BES

Let $S = (Q, A, \xrightarrow{a}, q_0)$ be a lts and φ the formula to be checked on q_0. Let $\mathcal{E}$ be the bes defined according to the following rules with $X_{q_0,\varphi}$ as distinguished variable. For each equation $X_{p_1,\psi_1} =_\sigma \ldots$ obtained by rules (1)-(5), the value of σ is inherited as follows:

- if ψ_1 is not in the scope of a fixpoint formula (there is no context), then the value of σ is irrelevant,
- otherwise, there exists a variable $X_{p_2,\sigma x.\psi_2}$ such that ψ_1 is a subformula of ψ_2 and there exists no fixpoint on any syntactic path from ψ_2 to ψ_1.

$X_{p,\text{true}} =_\sigma \texttt{true}$ $\quad$ $X_{p,\text{false}} =_\sigma \texttt{false}$

$$
\begin{aligned}
X_{p,<a>\varphi} &=_\sigma \bigvee_{p \xrightarrow{a} p'} X_{p',\varphi} && (1)\\
X_{p,[a]\varphi} &=_\sigma \bigwedge_{p \xrightarrow{a} p'} X_{p',\varphi} && (2)\\
X_{p,\varphi_1 \wedge \varphi_2} &=_\sigma X_{p,\varphi_1} \wedge X_{p,\varphi_2} && (3)\\
X_{p,\varphi_1 \vee \varphi_2} &=_\sigma X_{p,\varphi_1} \vee X_{p,\varphi_2} && (4)\\
X_{p,\neg\varphi} &=_\sigma \neg X_{p,\varphi} && (5)\\
X_{p,\nu X.\varphi} &=_\nu X_{p,\varphi[\nu X.\varphi/X]} && (6)\\
X_{p,\mu X.\varphi} &=_\mu X_{p,\varphi[\mu X.\varphi/X]} && (7)
\end{aligned}
$$

Proposition 4.1 *Let φ be a closed formula and $S = (Q, A, \xrightarrow{a}, q_0)$ a lts. Then, $q_0 \in [\![\varphi]\!]$ iff the solution δ of the bes $\mathcal{E}_{q_0,\varphi}$ is such that $\delta(X_{q_0,\varphi}) = 1$*

The proof is done by structural induction on φ [FM95].

5 Implementation Issues

This algorithm is implemented in the CÆSAR-ALDÉBARAN toolbox [FGM+92], both for bisimulation checking and model checking for the μ-calculus.

The CÆSAR-ALDÉBARAN toolbox provides both an environment for the construction and the verification of communicating systems, and facilities to implement quickly new verification algorithms.

More precisely, it offers two kinds of tool: a compiler and verifier. The compiler (CÆSAR) allows the translation of a LOTOS program into several formalisms, such that Petri Nets or communicating ltss, whereas the verifier (ALDÉBARAN) is able to perform equivalence checking on such formalisms.

However, a third component, called OPEN-CÆSAR, has been developed by H. Garavel upon CÆSAR([Gar92]). Initially, the aim of this component was twofold: to offer an environment for easily implement new verification algorithms, and to allow on-the-fly verifications of LOTOS programs. More recently, OPEN-CÆSAR has been extended to also deal with programs described as communicating ltss.

The basic idea of OPEN-CÆSAR is to run the verification tool together with a simulator which generates a lts from a given LOTOS program as far as the verification progress. More precisely, the architecture of OPEN-CÆSAR relies on three separate modules:

- the *graph module*, providing a C representation of the lts (i.e., primitives to access the initial state and the successor list of a given state) ;
- the *storage module*, providing predefined data structures to store the lts informations (such that a stack, a state-table, ...etc.) ;
- the *exploration module*, which implements the verification algorithm in terms of a lts traversal.

Note that the graph and storage modules only depend on the program under verification and are automatically generated by OPEN-CÆSAR.

Within this toolbox, the model-checking algorithm has been implemented as an exploration module for OPEN-CÆSAR, whereas the equivalence-checking algorithm was implemented as a part of ALDÉBARAN.

6 Examples

We give in this section two examples of non trivial verifications performed using the implementation within CÆSAR-ALDÉBARAN of the two algorithms described in the previous section. After a brief presentation of these case-studies, we propose for both of them a formal specification of the requirements to be verified. Finally, the results of the verification are commented.

6.1 The *rel/REL* protocol

The *rel/REL* protocol [SE90] aims to support atomic communications between a transmitter and several receivers, in spite of an arbitrary number of failures from the stations involved in the communications. We focus here on a version of this protocol which preserves the order of the messages sent by the transmitter.

Protocol description

The service provided by this protocol consists of the two following (informal) properties:

atomicity: if a station E sends a message m to a group of stations G, then either all the functioning elements of G receive m, or none of them does, even if several crashes occur in the group $\{E\} \cup G$.

causality: if a station E sends a sequence of messages, in a defined order, to a group of stations G, then no functioning element of G may receive these messages in a different order.

The *rel/REL* protocol is built on a *transport* layer protocol which provides a reliable (i.e., atomic and causal) message transmission between any *pair* of stations. In case of crash, stations are supposed to have a *fail-silent behavior*: they stop to send and to accept messages. It is also assumed that, even if multiple crashes occur, the network remains strongly connected: all functioning stations may still exchange messages.

The protocol is based on the *two phase commit* algorithm: the transmitter sends two successive copies of the message to all receivers; each message is uniquely identified, and an additional label indicates whether it is the first or the second copy. On receipt of the first copy, a station S waits for the second one; if it does not arrive before the expiration of a delay, then S assumes that the transmitter crashed and that some of the receivers may have not received a copy of the message. Then, S relays the transmitter and multicasts the two copies of the message, still using the *rel/REL* protocol. To reduce the network traffic, a station stops to relay as soon as a second copy of the message is received from the transmitter or from any other receiver.

The full LOTOS specification can be found in [FM95].

Formal specification of atomicity

As defined above, atomicity means that "*an emitted message is either received by all the functioning receivers, or is not received by any of them*". If get_i denotes the receipt of a first copy of a message by station i, and $crash_i$ the crash of station i, this property can be rephrased in the following way:

"*for each station i and for each execution sequence not containing action get_i nor action $crash_i$ (*station i is still waiting for a message*), any occurrence of action get_j for a given $j \neq i$ (*station j received the message*) should be eventually followed either by action get_i (station i itself received the message), or action $crash_i$ (*station i stopped to function*), or action $crash_j$ (*station j stopped to function*)*".

Such a property can be expressed using the μ-calculus. For any formula f, g we use the following macros: $< * >$ denotes the next operator (i.e. $< * > f = \bigvee_{a \in A} < a > f$) ; $[*]$ is the dual of $< * >$; **A** is the conditional "always" operator (i.e. $\mathbf{A} f g = \nu X.g \wedge (f \Rightarrow [*]X)$) ; **Ev** is the "eventuality" operator : $\mathbf{Ev} f = \mu X.(f \vee ([*]X \wedge < * > T))$.

The atomicity property is then expressed by the following formula:

$$\bigwedge_i \mathbf{A}([get_i]F \wedge [crash_i]F)(\bigwedge_{i \neq j} [get_j](\mathbf{Ev}(< get_i > T \vee < crash_i > T \vee < crash_j > T)))$$

Formal specification of causality

The second service property concerns the preservation of the message order: "*messages from a given transmitter are received in the same order as they were sent*". This is a safety property expressing that the received messages respect some conditions, but not ensuring their receipt. As it associates a transmitter and a receiver, it is sufficient to verify it for any pair (transmitter, receiver).

This property can easily be expressed using a transition system. Assuming that messages sent by a transmitter are identified by unique numbers $1, 2, ..., n$ according to their emission order, the expected behavior of a receiver (modulo safety equivalence) can be represented by the simple sequence $S_n = get_1 \,;\, get_2 \,;\, \ldots \,;\, get_n$. The verification process then only consists in checking if the graph G of the LOTOS program is related with S_n modulo safety equivalence [BFG+91] when only get_i actions are made visible.

Verification

Both properties were verified for a LOTOS program describing a configuration with a single transmitter, two receivers, and three distinct messages sent. The causality was proved using the on-the-fly part of ALDÉBARAN. The verification time was about few minutes.

Moreover, the atomicity was proved for a configuration with a single transmitter, three receivers and three distinct messages sent. The verification time was about one hour on a SUN SS10 [1]. Notice that on the same workstation a lack of memory prevented to generate the complete graph from the LOTOS program. Therefore "classical" model-checking or equivalence-checking algorithms would have not been applicable on this example and previous results described in [BM90] have been improved.

6.2 A Transit-Node

This example was initially defined in the RACE project 2039 SPECS. It consists of a simple transit node where messages arrive, are routed and leave the node.

Informally, it can be viewed as a "black box" communicating with the environment through several data ports-in, several data ports-out, a control port-in and a control port-out. Two types of messages can be received by the node: *control* messages, allowing to modify the node configuration, and *data* messages, which have to be correctly routed from a data port-in to a data port-out.

Formal description

The LOTOS specification of the transit node [Mou94] consists in four communicating processes: a *Controller*, an *ErrorHandler*, a *DataPortInSet* and a *DataPortOutSet*.

- Process *Controller* accepts and treats the following control messages: open a data port, (re)-define a route, and send outside the node faulty data messages (e.g., data messages those transit time inside the node exceeded a given amount of time).
- Process *DataPortInSet* manages the data port-in set of the node. Indeed, data ports-in are modelized as independent processes, without interaction between each other. These processes accept and treat data messages. Data messages are labeled with a route number, indicating a

[1] but a "false" answer was instantaneously obtained on an erroneous version of the protocol ...

set of data port-out to which the message can be routed. If the route does not exist, or if the corresponding data ports-out are not open, then the message becomes faulty.

- Process *DataPortOutSet* manages the data port-in set of the node. Here again, data ports-out are modelized as independent processes, without interaction between each other. These processes simply store the data messages received from process *DataPortInSet*, and deliver them outside the node.

- Process *ErrHandler* stores all the faulty messages transmitted by the other processes and delivers them outside the node on indication of process *Controller* (see above).

Specification of the requirements

Several requirements were associated with the informal description of the node. We focus here on the routing of data messages. Further details on other requirements can be found in [Mou94].

Informally, and from a safety point of view, messages are correctly routed if and only if the following holds:

whenever a data message M is received with a route indication R, if S is the data port-out set associated to R, then message M cannot leave the node on a data port-out not belonging to S.

Assuming that action *add_route* (R, S) associates to route R the data port-out set S (route definition), action *data_in* (M, R) denotes the reception of a data message M with route indication R, and action *data_out* (M, P) denotes the delivery of message M through data port-out P, this requirement can be rephrased as follows:

For each (M, R, S), after any occurrence of action add_route (R, S) (i.e., route R is defined), no action data_in (M, R) can be followed by an action data_out (M, P) for a given $P \notin S$ until a new occurrence of action add_route (R, S') (i.e., route R is redefined).

To express this formula in the mu-calculus we use a *weak until* operator "**Wu** f g": $\mathbf{Wu}fg = \mu X.(g \vee (f \wedge < * > X))$. Then, the routing requirement can be formally specified as follows, where S, S' and S" are different data port-out sets, R is a route number, P a data port-out not belonging to S, and M a data message:

$$
\begin{array}{l}
\mathbf{A}\ \ (T) \\
\quad ([\text{add\_route}\,(R, S)] \\
\qquad \mathbf{Wu}([\text{data\_in}\,(M, R)] \quad \mathbf{Wu} \quad (\neg < \text{data\_out}\,(M, P) > T) \\
\qquad\qquad\qquad\qquad\qquad\quad (< \text{add\_route}\,(R, S') > T)) \\
\quad (\text{add\_route}\,(R, S")))
\end{array}
$$

Verification

This property was verified for a LOTOS program describing a node configuration consisting of two data ports and two data messages. Note that two data ports leads to four distinct port sets, and thus four routes can be defined using this configuration (a route is uniquely defined by its associated port set). The verification time was about twenty minutes on a SUN SS10.

7 Conclusion

We have extended an algorithm for bisimulation checking to solve general boolean equation systems with mixed fixpoint computations. This algorithm, already implemented for several equivalence and preorder relations, has been now implemented for the μ-calculus model-checking. Thus, new verification facilities have been introduced in CÆSAR-ALDÉBARAN, providing the user with both another specification formalism (the μ-calculus) and a new decision procedure.

The main interest of this algorithm is that it relies on depth-first traversals, thus allowing on-the-fly verification: the bes is solved as far as the traversal goes on. In the worst case, its complexity is similar to the already known algorithms. However, experimental results demonstrated its practical interest. Thus, thanks to the on-the-fly approach, non trivial μ-calculus formula could be verified on LOTOS programs those underlying lts was too large to be generated. This is even enforced when the formulae under verification is false, since only a small part of the program usually needs to be explored. Moreover, in this last case, a set of erroneous execution sequences is available, providing diagnostic elements. Such features are highly appreciable in the context of program verification, particularly for specification debugging (i.e., in the early stages of the verification process).

A local algorithm was already proposed by Andersen to solve alternation-free bes [And92], also based on a traversal of its dependency graph. However, the main difference is that whenever the value of a variable changes from 0 to 1 (or conversely) during the traversal, each variable kept in a "dependency list" is immediately updated (instead of performing another partial traversal, if necessary, as in our algorithm). Thus, although worst-case complexities of both algorithms are similar, our solution allows to save memory as much as possible.

Further applications could be considered for this algorithm. For instance model-checking for other logics than the μ-calculus (possibly by specializing it to particular operators), or equivalence-checking for non (bi)simulation-based relations (as far as they could be expressed in terms of boolean equation systems). But boolean equation systems appear also in the context of program flow analysis and this algorithm should be applied in this area.

References

[And92] H.R. Andersen. Model checking and boolean graphs. In *ESOP'92,LNCS 582*, 1992.

[AV95] H.R. Andersen and B. Vergauwen. Efficient checking of behavioural relations and modal assertions using fixed-point inversion. In *CAV'95, LNCS 939*, 1995.

[BFG+91] A. Bouajjani, J.C. Fernandez, S. Graf, C. Rodriguez, and J. Sifakis. Safety for branching time semantics. In *18th ICALP*. Springer Verlag, july 1991.

[BM90] S. Bainbridge and L. Mounier. Specification and verification of a reliable multicast protocol. Software Engineering Department Technical Report HPL-91-63, Hewlett-Packard Laboratories, Bristol, U.K, 1990.

[BVW94] O. Bernholtz, M. Vardi, and P. Wolper. An automata-theoritic approach to branching-time model checking. In *Workshop on Computer-Aided Verification 94, LNCS 818*, 1994.

[Cle90] R. Cleaveland. Tableau based model checking in the propositional μ-calculus. *Acta Informatica*, 1990.

[CS91] R. Cleaveland and B. Steffen. A linear-time model checking algorithm for the alternation free modal μ-calculus. In *Workshop on Computer-Aided Verification, LNCS 575*, July 1991.

[EL86] E. A. Emerson and C. L. Lei. Efficient model-checking in fragments of the propositional Mu-calculus. In *Symposium on Logic in Computer Science*, 1986.

[FGM+92] J.Cl. Fernandez, H. Garavel, L. Mounier, A. Rasse, C. Rodriguez, and J. Sifakis. A tool box for the verification of lotos programs. In *14th International Conference on software Engineering*, 11-15May 1992.

[FM91] J.-C. Fernandez and L. Mounier. "on the fly" verification of behavioural equivalences and preorders. In *Workshop on Computer-aided Verification, Aalborg University, Denmark.* LNCS 575, Springer Verlag, july 1–4 1991.

[FM95] J. Cl. Fernandez and L. Mounier. A local checking algorithm for boolean equation systems. Technical Report Spectre-95-07, Verimag, Grenoble-France, 1995.

[Gar92] Hubert Garavel. The open/cæsar reference manual. SPECTRE Technical Report C33, Laboratoire de Génie Informatique — Institut IMAG, Grenoble, May 1992. version 1.0.

[Koz83] D. Kozen. Results on the propositional μ-calculus. In *Theoretical Computer Science.* North-Holland, 1983.

[Lar92] K. Larsen. Efficient local correctness checking. In *Computer-Aided Verification, LNCS 630*, July 1992.

[Mou94] Laurent Mounier. A lotos specification of a transit-node. SPECTRE Research Report 94-8, Verimag, Grenoble, March 1994.

[SE90] Santosh K. Shrivastava and Paul. D. Ezhilchelvan. rel/rel: A family of reliable multicast protocol for high-speed networks. Technical report, University of Newcastle, Dept. of Computer Science, U.K, 1990.

[VWL94] B. Vergauwen, J. Wauman, and J. Levi. Efficient fixpoint computation. In *Static Analysis Symposium, SAS'94, LNCS 864*, 1994.