Finding Small Stubborn Sets Automatically

Kimmo Varpaaniemi
Helsinki University of Technology, Digital Systems Laboratory
Otakaari 1, FIN-02150 Espoo, Finland
E-mail: Kimmo.Varpaaniemi@hut.fi

Abstract. State space generation is often needed in the analysis of concurrent and distributed systems. The stubborn set method is one of the techniques that try to alleviate the state space explosion encountered in state space generation. An algorithm for finding a stubborn set having as few enabled transitions as possible is presented. Practicality of the algorithm is motivated with the aid of examples.

Keywords: verification of concurrent and distributed systems, state space generation, stubborn sets

1 Introduction

State space generation is often needed in the analysis of concurrent and distributed systems such as telecommunication protocols. The *stubborn set method* [20, 21, 22, 23, 24, 25] is one of the methods that try to relieve the state space explosion problem that occurs in state space generation. Somewhat similar techniques have been presented in e.g. [3, 5, 6, 7, 10, 12, 13, 14, 15, 16].

Using [21], it is easy to derive a non-brute-force algorithm that finds a stubborn set that contains the least number of enabled transitions. The point of this paper is that though the obtained minimization algorithm is not practical as such, an incomplete version of the algorithm can be quite practical.

The rest of this paper has been organized as follows: Section 2 presents the formalism to be used. Stubborn sets and algorithms are considered in Section 3. Section 4 is devoted to examples. Conclusions are then drawn in Section 5.

2 Formalism

The algorithms we consider are applied to *place/transition nets* (with infinite capacities) [19]. We shall use N to denote the set of non-negative integer numbers, 2^X to denote the set of subsets of the set X, X^* to denote the set of finite words over the alphabet X, and ε to denote the empty word. (We shall also use "iff" to denote "if and only if" and "w.r.t." to denote "with respect to".)

Definition 2.1 A *place/transition net* is a quadruple $\langle S, T, W, M_0 \rangle$ such that S is the set of *places*, T is the set of *transitions*, $S \cap T = \emptyset$, $S \cup T$ is finite, W is a function from $(S \times T) \cup (T \times S)$ to N, and M_0 is the *initial marking* (*initial state*), $M_0 \in \mathcal{M}$ where $\mathcal{M}$ is the set of *markings* (*states*), i.e. functions from S to N. If $x \in S \cup T$, then the set of *input elements* of x is ${}^\bullet x = \{y \mid W(y,x) > 0\}$, the set of *output elements* of x is $x^\bullet = \{y \mid W(x,y) > 0\}$, and the set of *adjacent elements* of x is $x^\bullet \cup {}^\bullet x$. A transition t *leads (can be fired) from a marking M to a marking M'* ($M[t\rangle M'$ for short) iff

$$\forall s \in S \; M(s) \geq W(s,t) \wedge M'(s) = M(s) - W(s,t) + W(t,s).$$

A transition t is *enabled at a marking M* iff t leads from M to some marking. A marking M is *terminal* iff no transition is enabled at M. $\square$

Transition sequences and reachability are introduced in Definition 2.2.

Definition 2.2 Let $\langle S, T, W, M_0 \rangle$ be a place/transition net. The set T^* is called the *set of finite transition sequences of the net.* Let f be a function from $\mathcal{M}$ to 2^T. A finite transition sequence σ *f-leads (can be f-fired) from a marking M to a marking M'* iff $M[\sigma\rangle_f M'$, where

$$\forall M \in \mathcal{M} \; M[\varepsilon\rangle_f M, \text{ and}$$

$$\begin{aligned}&\forall M \in \mathcal{M} \; \forall M' \in \mathcal{M} \; \forall \delta \in T^* \; \forall t \in T\\ &M[\delta t\rangle_f M' \Leftrightarrow (\exists M'' \in \mathcal{M} \; M[\delta\rangle_f M'' \wedge t \in f(M'') \wedge M''[t\rangle M').\end{aligned}$$

A finite transition sequence σ is *f-enabled at a marking M* ($M[\sigma\rangle_f$ for short) iff σ f-leads from M to some marking. A marking M' is *f-reachable from a marking M* iff some finite transition sequence f-leads from M to M'. A marking M' is an *f-reachable marking* iff M' is f-reachable from M_0. The *f-reachability graph* of the net is the pair $\langle V, A \rangle$ such that the set of vertices V is the set of f-reachable markings, and the set of edges A is $\{\langle M, t, M' \rangle \mid M \in V \wedge M' \in V \wedge t \in f(M) \wedge M[t\rangle M'\}$. Let then Ψ be the function from $\mathcal{M}$ to 2^T such that for each marking M, $\Psi(M) = T$. A finite transition sequence σ *leads (can be fired) from a marking M to a marking M'* ($M[\sigma\rangle M'$ for short) iff $M[\sigma\rangle_\Psi M'$. A finite transition sequence σ is *enabled at a marking M* ($M[\sigma\rangle$ for short) iff $M[\sigma\rangle_\Psi$. A marking M' is *reachable from a marking M* iff some finite transition sequence leads from M to M'. A marking M' is a *reachable marking* iff M' is reachable from M_0. The Ψ-reachability graph of the net is called the *full reachability graph* of the net. $\square$

When f is clear from the context or is implicitly assumed to exist and be of a kind that is clear from the context, then the f-reachability graph of the net is called the *reduced reachability graph* of the net.

3 Stubborn sets

When one wants to show results concerning the theoretical properties of the stubborn set method, it is often best to use a dynamic definition of stubbornness. The below principles D1 and D2 are the principles 1^* and 2^* of [18], respectively.

Definition 3.1 Let $\langle S, T, W, M_0 \rangle$ be a place/transition net. Let M be a marking of the net. A set $T_s \subseteq T$ *fulfils the first principle of dynamic stubbornness* (*D1* for short) *at* M iff $\forall \sigma \in (T \setminus T_s)^* \ \forall t \in T_s \ M[\sigma t\rangle \Rightarrow M[t\sigma\rangle$. A transition t is a *dynamic key transition of a set* $T_s \subseteq T$ *at* M iff $t \in T_s$ and $\forall \sigma \in (T \setminus T_s)^* \ M[\sigma\rangle \Rightarrow M[\sigma t\rangle$. A set $T_s \subseteq T$ *fulfils the second principle of dynamic stubbornness* (*D2* for short) *at* M iff T_s has a dynamic key transition at M. A set $T_s \subseteq T$ is *dynamically stubborn at* M iff T_s fulfils D1 and D2 at M. A function f from $\mathcal{M}$ to 2^T is *a dynamically stubborn function* iff for each marking M, either $f(M)$ is dynamically stubborn at M or no transition is enabled at M. $\square$

Theorem 3.2 *Let $\langle S, T, W, M_0 \rangle$ be a place/transition net. Let f be a dynamically stubborn function from $\mathcal{M}$ to 2^T. Then the f-reachability graph of the net contains all reachable terminal markings.*

Proof. The result is an immediate consequence of Theorem 12 of [27]. $\square$

The definition of dynamically stubborn sets does not suggest any algorithm for computing such sets. Therefore, we need a static definition for stubbornness. We shall use the following which is almost the same as in [20].

Definition 3.3 Let $\langle S, T, W, M_0 \rangle$ be a place/transition net. The function E_1 from $\mathcal{M} \times S$ to 2^T, the functions E_2 and E_3 from $\mathcal{M} \times T \times S$ to 2^T, and the function E_4 from S to 2^T are defined as follows: Let $M \in \mathcal{M}$, $t \in T$, and $s \in S$. Then

$$\begin{array}{rcl} E_1(M,s) & = & \{t' \in {}^\bullet s \mid M(s) \geq W(s,t') \wedge W(t',s) > W(s,t')\}, \\ E_2(M,t,s) & = & E_4(s) \cup \{t' \in s^\bullet \mid \ W(s,t) > W(t,s) \wedge \\ & & \qquad W(s,t') > M(s) - W(s,t) + W(t,s)\}, \\ E_3(M,t,s) & = & E_1(M,s) \cup \{t' \in {}^\bullet s \mid M(s) \geq W(s,t') \wedge W(t',s) > W(t,s)\}, \text{ and} \\ E_4(s) & = & \{t' \in s^\bullet \mid W(s,t') > W(t',s)\}. \end{array}$$

A transition t is a *key transition of a set* $T_s \subseteq T$ *at a marking* M iff $t \in T_s$, t is enabled at M, and $\forall s \in {}^\bullet t \ E_4(s) \subseteq T_s$. A set $T_s \subseteq T$ is *stubborn at a marking* M iff some transition is a key transition of T_s at M and each transition t in T_s satisfies

$$\begin{array}{l} (\exists s \in {}^\bullet t \ M(s) < W(s,t) \wedge E_1(M,s) \subseteq T_s) \vee \\ (M[t\rangle \wedge (\forall s \in {}^\bullet t \ W(s,t) \leq W(t,s) \vee E_2(M,t,s) \subseteq T_s \vee E_3(M,t,s) \subseteq T_s)). \quad \square \end{array}$$

Intuitively, $E_1(M,s)$ is the set of transitions that could increase the contents of s and are not disabled by s at M. Correspondingly, $E_2(M,t,s)$ is the set of transitions that could decrease the contents of s or get disabled because of the firing of t at M. Respectively, $E_3(M,t,s)$ is the set of transitions that are not disabled by s at M and could increase the contents of s or have a greater output weight to s than t has. Finally, $E_4(s)$ is the set of transitions that could decrease the contents of s.

Theorem 3.4 *If a transition is a key transition of a set at a marking, then the transition is a dynamic key transition of the set at the marking. If a set is stubborn at a marking, then the set is dynamically stubborn at the marking.*

Proof. The first claim follows easily from the definitions. Our stubborn sets thus fulfil D2. The proof of Theorem 2.2 of [20] shows that our stubborn sets also fulfil D1. $\square$

On the other hand, it is easy to find examples where a dynamic key transition is not a key transition or where a dynamically stubborn set is not stubborn.

We now turn to the so called *deletion algorithm* [21, 22] that computes stubborn sets that are inclusion minimal w.r.t. enabled transitions. Our deletion algorithm can be derived from the algorithm given in [21].

Definition 3.5 Let $\langle S, T, W, M_0 \rangle$ be a place/transition net and M a marking of the net. The *and/or-graph at* M is a triple $\langle V_\otimes, V_\oplus, A \rangle$ such that the set of *and-vertices* $V_\otimes$ is

$$\{s \mid \exists t \in T \; s \in {}^\bullet t \wedge M(s) < W(s,t)\} \cup \{t \in T \mid M[t\rangle\} \cup \\ \{\langle t, s, i\rangle \mid t \in T \wedge M[t\rangle \wedge s \in {}^\bullet t \wedge W(s,t) > W(t,s) \wedge i \in \{2,3\}\},$$

the set of *or-vertices* $V_\oplus$ is

$$\{t \in T \mid \neg M[t\rangle\} \cup \{\langle t, s\rangle \mid t \in T \wedge M[t\rangle \wedge s \in {}^\bullet t \wedge W(s,t) > W(t,s)\},$$

and the set of edges A is

$$\begin{aligned} &\{\langle s, t'\rangle \mid \exists t \in T \; s \in {}^\bullet t \wedge M(s) < W(s,t) \wedge t' \in E_1(M,s)\} \cup \\ &\{\langle t, \langle t, s\rangle\rangle \mid t \in T \wedge M[t\rangle \wedge s \in {}^\bullet t \wedge W(s,t) > W(t,s)\} \cup \\ &\{\langle \langle t, s, i\rangle, t'\rangle \mid \;\; t \in T \wedge M[t\rangle \wedge s \in {}^\bullet t \wedge W(s,t) > W(t,s) \wedge \\ &\qquad\qquad\qquad i \in \{2,3\} \wedge t' \in E_i(M,t,s)\} \cup \\ &\{\langle t, s\rangle \mid t \in T \wedge s \in {}^\bullet t \wedge M(s) < W(s,t)\} \cup \\ &\{\langle \langle t, s\rangle, \langle t, s, i\rangle\rangle \mid t \in T \wedge M[t\rangle \wedge s \in {}^\bullet t \wedge W(s,t) > W(t,s) \wedge i \in \{2,3\}\}. \end{aligned}$$

A set $V_s \subseteq V_\otimes \cup V_\oplus$ is *legal* iff

$$(\forall x \in V_s \cap V_\otimes \; \forall y \in V_\otimes \cup V_\oplus \; \langle x, y\rangle \in A \Rightarrow y \in V_s), \\ (\forall x \in V_s \cap V_\oplus \; \exists y \in V_s \; \langle x, y\rangle \in A), \text{ and}$$

some transition is a key transition of $V_s \cap T$ at M. □

The idea in defining the and/or-graph and legality is nothing else but to rephrase the definition of stubbornness. Consequently, the set of transitions of any legal set is stubborn. Also, for each stubborn set, there exists a legal set such that the set of transitions of the legal set is the stubborn set. Moreover, the set of vertices of the and/or-graph is legal iff the marking is nonterminal.

The deletion algorithm is presented in Figure 1. The procedure DelAlg computes the stubborn set. The and/or-graph is initialized in such a way that each vertex has links to the immediate predecessor vertices and each or-vertex has a counter initialized to the number of the immediate successor vertices. (From Definition 3.5 it follows that the number is not 0.) Also, each vertex has an associated colour that is initially white, and each transition has a root flag that is initially zero as well as a protection flag that has an arbitrary initial value.

The computed stubborn set is the remaining set of white transitions and is inclusion minimal w.r.t. enabled transitions. In other words, no proper subset of its enabled transitions can be the set of enabled transitions of any stubborn set. This can be shown by showing that the set of white vertices is legal each time when the while-condition in Cnstr is checked.

The time taken by an execution of the deletion algorithm is at most proportional to $\mu\nu\rho|T|$, where μ is the maximum number of input places of a transition, ν is the maximum number of adjacent transitions of a place, and ρ is the maximum number of enabled transitions at a marking. The amount of space required is at most proportional to $\mu\nu|T|$.

```
procedure Speculate(x) {
    mark x grey;
    for each white immediate predecessor vertex y of x do
        if y is an and-vertex then Speculate(y);
        else {
            subtract 1 from the counter of y;
            if the counter of y is at 0 then Speculate(y); } }
procedure Rehabilitate(x) {
    mark x white;
    for each non-black immediate predecessor vertex y of x do {
        if y is an or-vertex then add 1 to the counter of y;
        if y is grey then Rehabilitate(y); } }
procedure Cnstr {
    for each protected enabled transition t do
        set the root flag of t equal to 1;
    while (there are at least two enabled white transitions and at least one
            enabled white transition has a zero root flag) do {
        let t be some enabled white transition having a zero root flag;
        set the root flag of t equal to 1; Speculate(t);
        if (the set of white transitions contains all protected transitions
            and has a key transition) then mark all grey vertices black;
        else Rehabilitate(t); } }
procedure DelAlg {
    initialize the and/or-graph; make all transitions unprotected; Cnstr; }
```

Figure 1: The deletion algorithm.

Let's then consider the problem that we have some subset T_e of enabled transitions and we want to know if there exists a stubborn set that contains all transitions of T_e but no other enabled transitions. We can solve this problem simply by running the deletion algorithm with the constraint that members of T_e are not allowed to be removed. If a stubborn set of the desired kind exists, we get such. Otherwise we get a stubborn set that contains all transitions of T_e and at least one additional enabled transition.

By solving the above T_e-problem for each subset of enabled transitions in turn, we find a stubborn set that has the least number of enabled transitions. Such minimization is somewhat impractical since we cannot assume that the number of enabled transitions would always be sufficiently small. However, we can use an incomplete form that can be considered practical.

The *incomplete minimization algorithm* is presented in Figure 2. The presented form is not the most general possible but here we have decided not to introduce parameters that do not vary in the applications in Section 4. The set returned by the function IncmplMin is the set of enabled transitions in the chosen stubborn set. Complete minimization is performed if at most 5 transitions are enabled. Otherwise T_e varies only over sets of size 1, and in the case of "failure", one of the so far found stubborn sets is chosen in such a way that none of the so far found sets contains less enabled transitions.

function IncmplMin {
    DelAlg /* see Figure 1 */ ;
    if (the set of enabled white transitions contains all enabled
        transitions or does not contain more than one enabled
        transition) **then return** the set of enabled white transitions ;
    make a backup set from the set of enabled white transitions ;
    initialize a variable L to the size of the backup set ;
    if more than 5 transitions are enabled **then** set L equal to 2 ;
    initialize a variable i to 1 ;
    while $i < L$ **do** {
        for each subset T_e of enabled transitions of size i **do** {
            initialize the and/or-graph ;
            protect the transitions in T_e and make others unprotected ;
            Cnstr /* see Figure 1 */ ;
            if all enabled white transitions are protected **then**
                { discard the existing backup set ; **return** T_e ; }
            if $L >$ the number of enabled white transitions **then** {
                discard the existing backup set ;
                make a backup set from the set of enabled white transitions ;
                set L equal to the size of the backup set ;
            } } add 1 to i ; }
    return the existing backup set ; }

Figure 2: The incomplete minimization algorithm.

The incomplete minimization algorithm has the same worst-case space complexity as the deletion algorithm. The time taken by an execution of the incomplete minimization is at most proportional to the time taken by an execution of the deletion algorithm multiplied by max(ρ, 32) where ρ is the maximum number of enabled transitions at a marking.

4 Examples

The algorithms of Section 3 have been implemented in a tool called PROD [28, 29]. In this section we consider two cases that have been studied with the aid of PROD. The first case is PFTP, a file transfer protocol [8]. The second case is YXA, a telephone protocol designed for educational purposes in Nokia Telecommunications Oy. PROD can be obtained by ftp from /pub/prod in saturn.hut.fi, Internet address 192.26.133.104. The below statistics were obtained by running PROD in Linux on a Pentium with 64 Megabytes of RAM. The reduced reachability graphs that were under construction were kept in files.

algorithm	vertices	edges	user time	system time	elapsed time
IMA	5655	6418	962 s	8 s	985 s
DAA	5758	6615	964 s	9 s	1000 s
IA1	14148	19545	6895 s	33 s	7212 s
IA2	18928	27307	7770 s	14 s	7799 s

Figure 3: The statistics of PFTP.

k	algorithm	vertices	edges	user time	system time	elapsed time
9	IMA	27271	28961	282 s	11 s	314 s
9	DAA	125346	135735	1282 s	74 s	1525 s
9	IA1	638316	735463	1485 s	587 s	8000 s
9	IA2	497877	629506	1022 s	399 s	3602 s
10	IMA	35114	37085	379 s	15 s	424 s
10	DAA	168547	181260	1778 s	114 s	2129 s
10	IA1	897495	1015580	2294 s	1172 s	25661 s
10	IA2	671636	833660	1427 s	708 s	11065 s
11	IMA	44517	46798	533 s	21 s	600 s
11	DAA	221935	237284	2510 s	170 s	3034 s
11	IA1	1231401	1372943	2985 s	2383 s	67478 s
11	IA2	884726	1081029	2078 s	1407 s	31243 s

Figure 4: The statistics of YXA.

The statistics include the number of vertices and edges in the reduced reachability graph, as well as the so called user time, system time and elapsed time of the generation of the graph when generated with different stubborn set computation algorithms: the above described incomplete minimization algorithm (IMA for short), the above described deletion algorithm alone (DAA for short), and two versions of the so called *incremental algorithm* [20, 22, 24, 29] (IA1 and IA2 for short). The incremental algorithm performs a search in a nondeterministically chosen transition dependency graph. IA1 performs a full search in order to avoid unnecessarily many enabled transitions, whereas IA2 accepts the first stubborn set found. The worst-case time complexity of both IA1 and IA2 is the worst-case time complexity of DAA divided by the maximum number of enabled transitions at a marking. The worst-case space complexity of both IA1 and IA2 is the same as the worst-case space complexity of DAA.

Note that while the so called elapsed time measures the "real world" time, it is then also affected by the other programs run in the Linux system. However, the experiments considered in this section were arranged in such a way that the elapsed times were not considerably affected by the other programs, except those performing the associated hard disk operations.

A detailed description of PFTP is given in Section 14.5 of [8]. Here we assume that messages are neither lost nor duplicated. The window size is 2, and a message queue can have at most two messages at a time. The stubborn set method then produces a reduced reachability graph that has no terminal marking. From this it follows that under the above assumptions, the protocol is free of deadlocks. Figure 3 shows the statistics of the PFTP case.

As far as the PFTP case is concerned, the algorithms in PROD may seem less efficient than the algorithms in the SPIN tool [8, 9]. A partial explanation is that SPIN uses a different modelling formalism and a different definition of stubbornness that can utilize the special properties of some frequently needed operations [6, 15]. Another partial explanation is that SPIN stores states more compactly than PROD.

YXA is a simple telephone protocol which nevertheless includes most of the relevant aspects, such as charging, needed in any real world telephone protocol. In abstract terms, we again have processes that communicate by sending messages. A message queue can have at most k messages at a time. Because of the more or less inevitable state space explosion, we keep the number of processes at its least possible value. The stubborn set method then produces a reduced reachability graph that has no terminal marking. The protocol (actually a corrected version of the original YXA) is thus free of deadlocks. Figure 4 shows the statistics of the YXA case with different values of k.

5 Conclusions

The contribution of this paper is the incomplete minimization algorithm. The algorithm is worth of consideration whenever one wants to get proper advantage of the stubborn set method. Unfortunately, trying to minimize branching is no superior strategy for getting as small reduced reachability graphs as possible. Though the example given in [22] would suffice, let's consider a pathological case, a place/transition net $\langle S, T, W, M_0 \rangle$ such that $S = \{s\}$, $T = \{t_0, t_1, t_2\}$, $W(t_0, s) = W(s, t_1) = W(s, t_2) = 1$, $W(s, t_0) = W(t_1, s) = W(t_2, s) = 0$, and $M_0(s) = 0$. Then the only way to obtain a finite reduced reachability graph is to choose $\{t_1, t_2\}$ for the stubborn set at some non-initial marking though $\{t_0\}$ would contain less enabled transitions.

Actually, there is hardly no superior strategy [14, 22]. Nevertheless, if the set of enabled transitions of a stubborn set is a proper subset of enabled transitions of another stubborn set, it is clearly preferable to choose the former stubborn set [22]. On the other hand, it is difficult to imagine a general heuristic that would try to minimize the size of the redeuced reachability graph without trying to minimize branching.

When the storing of states is a critical problem, state space caching [5] can be used to relieve the situation. Such caching typically does not affect the way in which stubborn sets are computed.

Though we have presented the incomplete minimization algorithm for place/transition nets, a similar algorithm can actually be presented for any formalism where the stubborn set method is applicable. This is so because the idea in defining the and/or-graph and legality is nothing else but to rephrase the definition of stubbornness.

The algorithms in this paper can also be extended to on-the-fly verification of linear time temporal formulas that do not contain the next-state operator. The verification can follow the lines given in [25] or utilize Büchi automata [2, 4] and the preservation of CFFD-semantics [11, 24, 26]. Taking advantage of fairness assumptions should also be possible as suggested by the so called ample set approach in [14, 15].

Acknowledgements

This work has been supported by Helsinki Graduate School in Computer Science and Engineering, The Academy of Finland, The Technology Development Centre of Finland, Nokia Telecommunications Oy, Telecom Finland Oy, and Commit Oy.

The author would like to thank Associate Professor Antti Valmari from Tampere University of Technology for fruitful discussions on the subject of this research, and M.Sc. Esa Kettunen from Nokia Telecommunications Oy for a description of YXA.

References

[1] Courcoubetis C (ed): Proceedings of CAV '93. LNCS 697, Springer-Verlag 1993, 504 p.

[2] Courcoubetis C, Vardi MY, Wolper P, Yannakakis M: *Memory Efficient Algorithms for the Verification of Temporal Properties.* Formal Methods in System Design 1 (1992) 2/3, 275–288.

[3] Gerth R, Kuiper R, Peled D, Penczek W: *A Partial Order Approach to Branching Time Logic Model Checking.* Proceedings of the 3rd Israel Symposium on Theory of Computing and Systems, 1995.

[4] Gerth R, Peled D, Vardi MY, Wolper P: *Simple On-the-Fly Automatic Verification of Linear Temporal Logic.* In [17].

[5] Godefroid P: *Partial-Order Methods for the Verification of Concurrent Systems — An Approach to the State-Explosion Problem.* Doctoral thesis, University of Liège, 1994, 134 p.

[6] Godefroid P, Pirottin D: *Refining Dependencies Improves Partial-Order Verification Methods.* In [1], 438–449.

[7] Godefroid P, Wolper P: *Using Partial Orders for the Efficient Verification of Deadlock Freedom and Safety Properties.* Formal Methods in System Design 2 (1993) 2, 149–164.

[8] Holzmann GJ: *Design and Validation of Computer Protocols.* Prentice Hall 1991, 500 p.

[9] Holzmann GJ, Peled D: *An Improvement in Formal Verification.* Proceedings of the 7th International Conference on Formal Description Techniques, Bern 1994, 177–191.

[10] Janicki R, Koutny M: *Using Optimal Simulations to Reduce Reachability Graphs.* Clarke EM, Kurshan RP (eds), Proceedings of CAV '90. LNCS 531, Springer-Verlag 1991, 166–175.

[11] Kaivola R, Valmari A: *The Weakest Compositional Semantic Equivalence Preserving Nexttime-less Linear Temporal Logic.* Cleaveland WR (ed), Proceedings of CONCUR '92. LNCS 630, Springer-Verlag 1992, 207–221.

[12] Katz S, Peled D: *Verification of Distributed Programs Using Representative Interleaving Sequences.* Distributed Computing 6 (1992) 2, 107–120.

[13] Overman WT: *Verification of Concurrent Systems: Function and Timing.* PhD thesis, University of California at Los Angles, 1981, 174 p.
[14] Peled D: *All from One, One for All: on Model Checking Using Representatives.* In [1], 409–423.
[15] Peled D: *Combining Partial Order Reductions with On-the-Fly Model-Checking.* Dill DL (ed), Proceedings of CAV '94. LNCS 818, Springer-Verlag 1994, 377–390.
[16] Peled D, Penczek W: *Using Asynchronous Büchi Automata for Efficient Automatic Verification of Concurrent Systems.* In [17].
[17] Proceedings of the 15th IFIP WG 6.1 Symposium on Protocol Specification, Testing and Verification, Warsaw 1995. Published in the IFIP series by Chapman & Hall.
[18] Rauhamaa M: *A Comparative Study of Methods for Efficient Reachability Analysis.* Helsinki University of Technology, Digital Systems Laboratory Report A 14, 1990, 61 p.
[19] Reisig W: *Petri Nets: An Introduction.* EATCS Monographs on Theoretical Computer Science 4, Springer-Verlag 1985, 161 p.
[20] Valmari A: *Error Detection by Reduced Reachability Graph Generation.* Proceedings of the 9th European Workshop on Application and Theory of Petri Nets, Venice 1988, 95–112.
[21] Valmari A: *Heuristics for Lazy State Space Generation Speeds up Analysis of Concurrent Systems.* Mäkelä M, Linnainmaa S, Ukkonen E (eds), Proceedings of the Finnish Artificial Intelligence Symposium, Vol. 2, Helsinki 1988, 640–650.
[22] Valmari A: *State Space Generation: Efficiency and Practicality.* Doctoral thesis, Tampere University of Technology Publications 55, 1988, 170 p.
[23] Valmari A: *A Stubborn Attack on State Explosion.* Formal Methods in System Design 1 (1992) 4, 297–322.
[24] Valmari A: *Alleviating State Explosion during Verification of Behavioural Equivalence.* University of Helsinki, Department of Computer Science, Report A-1992-4, 1992, 57 p.
[25] Valmari A: *On-the-Fly Verification with Stubborn Sets.* In [1], 397–408.
[26] Valmari A, Tienari M: *An Improved Failures Equivalence for Finite-State Systems with a Reduction Algorithm.* Jonsson B, Parrow J, Pehrson B (eds), Proceedings of the 11th International IFIP WG 6.1 Symposium on Protocol Specification, Testing and Verification. North-Holland 1991, 3–18.
[27] Varpaaniemi K: *On Combining the Stubborn Set Method with the Sleep Set Method.* Valette R (ed), Proceedings of PN '94. LNCS 815, Springer-Verlag 1994, 548–567.
[28] Varpaaniemi K, Halme J, Hiekkanen K, Pyssysalo T: *PROD Reference Manual.* Helsinki University of Technology, Digital Systems Laboratory Report B 13, 1995, 56 p.
[29] Varpaaniemi K, Rauhamaa M: *The Stubborn Set Method in Practice.* Jensen K (ed), Proceedings of PN '92. LNCS 616, Springer-Verlag 1992, 389–393.