Minimizing the Number of Successor States in the Stubborn Set Method

Kimmo Varpaaniemi *

Helsinki University of Technology
Laboratory for Theoretical Computer Science
P.O. Box 9700, FIN-02015 HUT, Finland
kimmo.varpaaniemi@hut.fi

Abstract. Combinatorial explosion which occurs in parallel compositions of LTSs can be alleviated by letting the stubborn set method construct on-the-fly a reduced LTS that is CFFD- or CSP-equivalent to the actual parallel composition. This paper considers the problem of minimizing the number of successor states of a given state in the reduced LTS. The problem can be solved by constructing an and/or-graph with weighted vertices and by finding a set of vertices that satisfies a certain constraint such that no set of vertices satisfying the constraint has a smaller sum of weights. Without weights, the and/or-graph can be constructed in low-degree polynomial time w.r.t. the "length of the input of the problem". However, since actions can be nondeterministic and transitions can share target states, it is not known whether the weights are generally computable in polynomial time. Consequently, it is an open problem whether minimizing the number of successor states is as "easy" as minimizing the number of successor transitions.
Keywords: LTSs, CFFD-equivalence, CSP-equivalence, stubborn sets, and/or-graphs

1 Introduction

In process-algebraic approaches to formal verification, it is usual to describe behaviors of actual systems by defining *LTSs* (*labelled transition systems*) and operations which produce LTSs from LTSs. For example, there are several ways to define a *parallel composition* of LTSs. If correspondence between actions and atomic propositions is defined appropriately, *CFFD-equivalence* (*chaos free failures divergences equivalence*) guarantees preservation of system-level satisfiability/unsatisfiability of any linear time temporal logic formula that does not contain the next-time operator. Since CFFD-equivalence is also a congruence w.r.t. certain typical parallel composition operations as well as certain other typical LTS operations, it is particularly suitable for compositional verification. More information about equivalences and preorders related to CFFD-equivalence can

* This work has been funded by the National Technology Agency of Finland, Nokia Research Center, Nokia Networks, Elisa Communications, and Finnish Rail Administration.

be found e.g. from [3, 13]. The famous *CSP-equivalence* [2, 13] is closely related to CFFD-equivalence. CFFD-equivalence implies CSP-equivalence.

Combinatorial explosion tends to occur in parallel compositions, but the explosion can be alleviated by eliminating redundant interleavings. The *stubborn set method* is able to do such elimination by constructing on-the-fly a reduced LTS that is CFFD- or CSP-equivalent to the actual parallel composition [11, 12]. (Ensuring CFFD-equivalence needs more constraints in the method than ensuring CSP-equivalence.) The method has been classified as one of the *partial order methods* [6] and has turned out [8, 12] successful also in the verification of branching time temporal properties which in turn are related to the CCS theory [4]. *Persistent sets* [1] and *ample sets* [5, 7] are strikingly similar to stubborn sets, at least if we consider the actual construction algorithms that have been suggested for stubborn, persistent and ample sets.

It is a system-independent and intuitively appealing heuristic to minimize the number of successor states of each state in the reduced LTS though it is well known that the resulting reduced LTS itself is not necessarily minimal w.r.t. the number of states. On the other hand, it is possible to compromise the heuristic in such a way that "nice" branching is obtained by means of "good" stubborn sets.

And/or-graphs are used for various purposes e.g. in the research of artificial intelligence. The idea of using and/or-graphs in stubborn set optimization problems was introduced in [10]. Many of the system-independent heuristics suggested for the computation of stubborn sets in a sense work on the abstraction level of and/or-graphs, regardless of whether such a graph is explicitly constructed.

The number of successor states of a given state can be minimized by constructing an and/or-graph with weighted vertices and by finding a set of vertices that satisfies a certain constraint such that no set of vertices satisfying the constraint has a smaller sum of weights. Without weights, the and/or-graph can be constructed in low-degree polynomial time w.r.t. the "length of the input of the problem". However, since actions can be nondeterministic and transitions can share target states, it is not known whether the weights are generally computable in polynomial time. Consequently, it is an open problem whether minimizing the number of successor states is as "easy" as minimizing the number of successor transitions.

The rest of the paper has been structured as follows. Section 2 presents results concerning and/or-graphs and makes remarks on two tools that have been used for solving some and/or-graph problems that are of interest in this paper. Some basic LTS theory, including CFFD- and CSP-equivalence and one form of a parallel composition, is presented in Section 3. Section 4 defines CSP- and CFFD-oriented versions of stubbornness and then proceeds to the main topic of the paper.

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^* (respectively, X^ω) to denote the set of finite (respectively, infinite) words over the alphabet X, and ε to denote the empty word. For any word σ, $\sigma(i)$ denotes the element in the $(i+1)$th position of σ. For any sets A and B, for any $R \subseteq A \times B$ and for any $x \in A$, $R(x)$ is the set $\{y \in B \mid \langle x, y \rangle \in R\}$. For any $a \in N$, $N_{<a} = \{i \in N \mid i < a\}$.

2 And/or-graphs

Definition 2.1. An *and/or-graph* is a quadruple $\langle V_\otimes, V_\oplus, \kappa, F \rangle$ such that $V_\otimes$ is the set of *and-vertices*, $V_\oplus$ is the set of *or-vertices*, $V_\otimes \cap V_\oplus = \emptyset$, $V_\otimes \cup V_\oplus$ is finite, $\kappa \in V_\otimes \cup V_\oplus$, $F \subseteq (V_\otimes \cup V_\oplus) \times (V_\otimes \cup V_\oplus)$ is the set of *edges*, and
$\forall\, y \in V_\oplus :\ F(y) \neq \emptyset$. A set $L \subseteq V_\otimes \cup V_\oplus$ is *legal* iff
$(\kappa \in L) \wedge (\forall\, x \in V_\otimes \cap L :\ F(x) \subseteq L) \wedge (\forall\, y \in V_\oplus \cap L :\ F(y) \cap L \neq \emptyset)$.
For any $H \subseteq V_\otimes \cup V_\oplus$, a set $P \subseteq V_\otimes \cup V_\oplus$ is *H-solid* iff there exists a legal set $L \subseteq V_\otimes \cup V_\oplus$ such that $P = H \cap L$. □

Let SOLIDOPTW be the following optimization problem: "Given an and/or-graph, a subset H of its vertices and a function w from H to N, find an H-solid set A such that for each H-solid set P, $\sum_{a \in A} w(a) \leq \sum_{p \in P} w(p)$." Every instance of SOLIDOPTW has a solution. In the worst case, H itself is a solution. Note also that in our considerations concerning optimization problems, there is no need to know whether an obtained solution is unique.

Lemma 2.1. *Let $\langle V_\otimes, V_\oplus, \kappa, F \rangle$ be an and/or-graph and $H \subseteq V_\otimes \cup V_\oplus$.*
Let $\vartheta = \lceil \log_2(|V_\otimes| + |V_\oplus| + 1) \rceil$. Within $O(|V_\otimes| + |V_\oplus| + |F|\vartheta)$ elementary time units, it is possible to check whether a given subset P of vertices is H-solid. Any instance of SOLIDOPTW concerning this and/or-graph and this H can be solved within $O(((|V_\otimes| + |V_\oplus|)\chi + |F|\vartheta)2^{|H|})$ elementary time units where
$\chi = \lceil \log_2(1 + \sum_{h \in H} w(h)) \rceil$.

Proof. The first claim is just a combination of the results of Lemmas 2.1 and 2.2 of [15]. The claim concerning SOLIDOPTW follows from the observation that any instance of SOLIDOPTW concerning this and/or-graph and this H can be solved by going through the subsets of H in such a way that for each subset P of H, if P is H-solid, a so-far-minimum variable is compared to $\sum_{p \in P} w(p)$ and updated if needed. □

IMA, i.e. the so called *incomplete minimization algorithm*, solves some instances of SOLIDOPTW whereas for any of the other instances, IMA computes an H-solid set that has the least $\sum_{h \in H} w(h)$ among the H-solid sets recognized when trying to find a solution to the instance. IMA uses the brute force idea expressed in the proof of Lemma 2.1, but instead of doing an H-solidity check for a candidate P, IMA computes an H-solid set "inspired by P" by using a variant of the so called *deletion algorithm* [10]. For any candidate P, IMA computes such a set within $O(((|V_\otimes| + |V_\oplus|)\chi + |F|\vartheta)|H|)$ elementary time units. The implementation of IMA for the needs of the stubborn set method of the high-level Petri net reachability analysis tool PROD [16], concerning the case $\forall\, h \in H :\ w(h) = 1$, is described in [14]. In that implementation, the candidates

$P \subseteq H$ are processed in the order of increasing cardinality, and a solution to the instance of SOLIDOPTW in question is guaranteed whenever a solution of cardinality ≤ 1 exists or $|H| \leq 5$. The experiments reported in [14] and the experience obtained so far indicate the following.

- In some reachability analysis tasks, using IMA for the tasks reduces the total analysis times significantly when compared to any of the tried alternatives that are available in PROD for the same tasks.
- Though IMA has been used for several industry project reachability analysis tasks, in such tasks IMA has never significantly slowed down the analysis when compared to the alternative where each needed H-solid set is computed by running the deletion algorithm only once. The same holds for all those "examples from the literature" that have been fed into PROD.

For complete minimization, PROD has an option which makes PROD call the logic program analysis tool Smodels [9, 17] whenever an H-solid set is to be computed. The translation presented in [15] from and/or-graphs to logic programs is as such applicable for expressing any instance of SOLIDOPTW in such a way that Smodels is able to compute a solution. The experience obtained so far indicates the following.

- The most typical instances of SOLIDOPTW arising in the context of PROD are so easy that PROD's own heuristics find solutions to them, much quicker than Smodels.
- Smodels tends to process the candidates $P \subseteq H$ in the order of decreasing cardinality and therefore sometimes "gets stuck" in the middle of the cardinalities even if a solution of cardinality ≤ 1 exists.
- There is no fundamental reason why Smodels could not be made to process the candidates in the order of increasing cardinality.
- Smodels has algorithms for goals less demanding than minimization, and it is apparently recommendable to try to reach some of such goals before trying minimization.

Let SOLIDCARD be the following decision problem: "Given an and/or-graph, a subset H of its vertices and a number $n \in N$, is there an H-solid set P such that $|P| \leq n$?" Respectively, let SOLIDWEIGHT be the following decision problem: "Given $n \in N$, an and/or-graph, a subset H of its vertices and a function w from H to N, is there an H-solid set P such that $\sum_{p \in P} w(p) \leq n$?"

Clearly, any instance of SOLIDCARD can be solved by solving an instance of SOLIDWEIGHT, whereas any instance of SOLIDWEIGHT can be solved by solving an instance of SOLIDOPTW. On the other hand, any instance of SOLIDOPTW can be solved by solving at most $\lceil \log_2(1 + \sum_{h \in H} w(h)) \rceil$ instances of SOLIDWEIGHT, by carrying out a binary search where n varies between 0 and $\sum_{h \in H} w(h)$.

Theorem 2.1. *SOLIDCARD and SOLIDWEIGHT are NP-complete.*

Proof. NP-completeness of SOLIDCARD is the precise result of Theorem 5.2 in [15]. Lemma 2.1 essentially implies that SOLIDWEIGHT is in NP. NP-hardness of SOLIDWEIGHT is a direct consequence of the NP-hardness of SOLIDCARD. □

3 LTSs, CFFD-equivalence, CSP-equivalence, and the parallel composition

Definition 3.1. A *labelled transition system* (an *LTS*) is a quintuple $L = \langle S, \mho, \Im, \Delta, \iota \rangle$ such that S is the set of *states*, $\mho$ is the set of *visible actions*, $\Im$ is the set of *invisible actions*, $\mho \cap \Im = \emptyset = S \cap (\mho \cup \Im)$, $\Delta \subseteq S \times (\mho \cup \Im) \times S$ is the set of *transitions*. and $\iota \in S$ is the *initial state*. For each transition $\langle x, a, y \rangle$, a is the *label of the transition* and y is the *target state of the transition*. A transition $\langle x, a, y \rangle$ is a *successor transition* of a state x' iff $x' = x$. A state y is a *successor state* of a state x iff there is some action a such that $\langle x, a, y \rangle$ is a transition. L is a *finite LTS* iff $S \cup \mho \cup \Im$ is finite. An action a is *enabled at a state* x iff there is a state y such that $\langle x, a, y \rangle \in \Delta$. The set of actions enabled at a state x is denoted by $\eta(x)$. A state x is *terminal* iff $\eta(x) = \emptyset$. An action a is *deterministic* iff $\forall x \in S : \ \forall y \in S : \ \forall z \in S : \ ((\langle x, a, y \rangle \in \Delta) \wedge (\langle x, a, z \rangle \in \Delta)) \Rightarrow y = z$.
L is a *deterministic LTS* iff all actions are deterministic. □

Definition 3.2. Let $L = \langle S, \mho, \Im, \Delta, \iota \rangle$ be an LTS. For any $\rho \in (S \cup \mho \cup \Im)^*$, ρ is a *finite path-like sequence* of L iff $|\rho|$ is an odd number, for each even number $i \in N_{<|\rho|}$, $\rho(i) \in S$, and for each odd number $i \in N_{<|\rho|}$, $\rho(i) \in \mho \cup \Im$. Respectively, for any $\rho \in (S \cup \mho \cup \Im)^\omega$, ρ is an *infinite path-like sequence* of L iff for each even number $i \in N$, $\rho(i) \in S$, and for each odd number $i \in N$, $\rho(i) \in \mho \cup \Im$. A finite path-like sequence ρ is a *finite path* of L iff for each odd number $i \in N_{<|\rho|}$, $\langle \rho(i-1), \rho(i), \rho(i+1) \rangle \in \Delta$. (The definition of finite path-like sequences guarantees that $i + 1 < |\rho|$.) Respectively, an infinite path-like sequence ρ is an *infinite path* of L iff for each odd number $i \in N$, $\langle \rho(i-1), \rho(i), \rho(i+1) \rangle \in \Delta$. A state y is *reachable* from a state x iff L has a finite path ρ such that $\rho(0) = x$ and $\rho(|\rho| - 1) = y$. For any finite path-like sequence ρ, the *label* of ρ is the word $\sigma \in (\mho \cup \Im)^*$ such that $2|\sigma| + 1 = |\rho|$ and for each $i \in N_{<|\sigma|}$, $\sigma(i) = \rho(2i+1)$. Respectively, for any infinite path-like sequence ρ, the label of ρ is the word $\sigma \in (\mho \cup \Im)^\omega$ such that for each $i \in N$, $\sigma(i) = \rho(2i+1)$. For any finite or infinite path-like sequence ρ, $\zeta(\rho)$ denotes the label of ρ. □

Definition 3.3. Let $L = \langle S, \mho, \Im, \Delta, \iota \rangle$ be an LTS. For L, the function υ, also called the *visible projection*, from $(\mho \cup \Im)^* \cup (\mho \cup \Im)^\omega$ to $\mho^* \cup \mho^\omega$ is defined as follows.

- $\upsilon(\varepsilon) = \varepsilon$, whereas for any $a \in \mho$, $\upsilon(a) = a$, and for any $b \in \Im$, $\upsilon(b) = \varepsilon$.
- For any $a \in \mho \cup \Im$ and for any $\sigma \in (\mho \cup \Im)^*$, $\upsilon(a\sigma) = \upsilon(a)\upsilon(\sigma)$.
- For any $\sigma \in (\mho \cup \Im)^\omega$, $\upsilon(\sigma) = \upsilon(\sigma(0))\upsilon(\sigma(1))\upsilon(\sigma(2))\ldots$. (From this it follows that $\upsilon(\sigma)$ is not necessarily in $\mho^\omega$.) □

Definition 3.4. Let $L = \langle S, \mho, \Im, \Delta, \iota\rangle$ be an LTS. A state x is a *stable state* iff no invisible action is enabled at x. L is a *stable LTS* iff the initial state is stable. The *stability indicator* of L is an integer number such that the indicator is 1 if L is stable and 0 if L is not stable. A finite word σ of visible actions is a *finite trace* of L iff L has a finite path ρ such that $\rho(0) = \iota$ and $\upsilon(\zeta(\rho)) = \sigma$. Respectively, an infinite word σ of visible actions is an *infinite trace* of L iff L has an infinite path ρ such that $\rho(0) = \iota$ and $\upsilon(\zeta(\rho)) = \sigma$. A finite word σ of visible actions is a *divergence trace* of L iff L has an infinite path ρ such that $\rho(0) = \iota$ and $\upsilon(\zeta(\rho)) = \sigma$. A finite word σ of visible actions is a *CSP-divergence trace* of L iff some divergence trace of L is a (proper or improper) prefix of σ. For any $\sigma \in \mho^*$ and for any $A \subseteq \mho$, the pair $\langle\sigma, A\rangle$ is a *stable failure* of L iff L has a finite path ρ such that $\rho(0) = \iota$, $\upsilon(\zeta(\rho)) = \sigma$, and $\eta(\rho(|\rho| - 1)) \subseteq \mho \setminus A$. For any $\sigma \in \mho^*$ and for any $A \subseteq \mho$, the pair $\langle\sigma, A\rangle$ is a *CSP-failure* of L iff $\langle\sigma, A\rangle$ is a stable failure of L or σ is a CSP-divergence trace of L. (In order to be explicit, we also require that all finite traces, divergence traces and CSP-divergence traces are finite words of visible actions, all infinite traces are infinite words of visible actions, and all stable failures and CSP-failures are members of $\mho^* \times 2^{\mho}$.) Two LTSs are *chaos-free failures divergences equivalent* (*CFFD-equivalent*) iff they have the same visible actions, the same value of the stability indicator, the same infinite traces, the same divergence traces and the same stable failures. Two LTSs are *CSP-equivalent* iff they have the same visible actions, the same CSP-divergence traces and the same CSP-failures. $\square$

Definition 3.5. Let $n \in N \setminus \{0, 1\}$. Let $\mathcal{L} = \{L_0, \ldots, L_{n-1}\}$ be such that for each $i \in N_{<n}$, $L_i = \langle S_i, \mho_i, \Im_i, \Delta_i, \iota_i\rangle$ is an LTS. $\mathcal{L}$ is *action-consistent* iff for all different i and k in $N_{<n}$, $\mho_i \cap \Im_k = \emptyset$ and $(S_0 \times \ldots \times S_{n-1}) \cap (\mho_i \cup \Im_i) = \emptyset$. $\mathcal{L}$ is *strictly action-consistent* iff $\mathcal{L}$ is action-consistent and for all different i and k in $N_{<n}$, $\Im_i \cap \Im_k = \emptyset$. If $\mathcal{L}$ is action-consistent, the *parallel composition* of $L_0, \ldots, L_{n-1}$ is the LTS $L = \langle S, \mho, \Im, \Delta, \iota\rangle$ such that $S = S_0 \times \ldots \times S_{n-1}$, $\mho = \mho_0 \cup \ldots \cup \mho_{n-1}$, $\Im = \Im_0 \cup \ldots \cup \Im_{n-1}$, $\iota = \langle\iota_0, \ldots, \iota_{n-1}\rangle$, for any $a \in \mho \cup \Im$, for any $x = \langle x_0, \ldots, x_{n-1}\rangle \in S$ and for any $y = \langle y_0, \ldots, y_{n-1}\rangle \in S$, $\langle x, a, y\rangle \in \Delta$ iff $\forall i \in N_{<n} : \ ((a \in \mho_i \cup \Im_i) \wedge (\langle x_i, a, y_i\rangle \in \Delta_i)) \vee ((a \notin \mho_i \cup \Im_i) \wedge x_i = y_i)$. $\square$

Definition 3.5 conforms to [12]. The parallel composition operation can be thought to carry out hiding after fine-grained alphabet-based synchronization. The synchronized invisible actions are the "hidden" actions. There is no need for renaming because several invisible actions are allowed. Proposition 3.1 is based on the considerations in [13].

Proposition 3.1. *Let $n \in N \setminus \{0, 1\}$, and let $\{K_0, \ldots, K_{n-1}\}$ and $\{L_0, \ldots, L_{n-1}\}$ be strictly action-consistent sets of LTSs. If for each $i \in N_{<n}$, K_i is CFFD-equivalent to L_i. then the parallel composition of $K_0, \ldots, K_{n-1}$ is CFFD-equivalent to the parallel composition of $L_0, \ldots, L_{n-1}$. Respectively, if for each $i \in N_{<n}$, K_i is CSP-equivalent to L_i, then the parallel composition of $K_0, \ldots, K_{n-1}$ is CSP-equivalent to the parallel composition of $L_0, \ldots, L_{n-1}$.* $\square$

Since Definition 3.5 allows synchronization of invisible actions, the claim obtained from Proposition 3.1 by omitting the word "strictly" is not valid. However, there is a good reason to allow synchronization of invisible actions. Namely, the connection, mentioned in the beginning of Introduction, between CFFD-equivalence and preservation of linear time temporal properties does not require CFFD-equivalence to be any kind of a congruence. The less we have visible actions, the better are the chances to get significant reduction by using the stubborn set method, On the other hand, it is up to the user of the method to decide which actions in a component LTS are visible.

4 Stubborn sets

Throughout this section, we assume the following.

- $n \in N \setminus \{0,1\}$, and for each $i \in N_{<n}$, $L_i = \langle S_i, \mho_i, \Im_i, \Delta_i, \iota_i \rangle$ is an LTS, and η_i is the "η of L_i" w.r.t. Definition 3.1.
- $\{L_0, \ldots, L_{n-1}\}$ is action-consistent, $L = \langle S, \mho, \Im, \Delta, \iota \rangle$ is the parallel composition of $L_0, \ldots, L_{n-1}$, and η_L is the "η of L" w.r.t. Definition 3.1.
- $s = \langle s_0, \ldots, s_{n-1} \rangle \in S$ is an unstable state, and $(\mho \cup \Im) \cap 2^N = \emptyset$.

Let us recall from Definition 3.4 that a state x is stable iff no invisible action is enabled at x. Definition 4.1 is restricted to unstable states because the underlying theory [11, 12] suggests that for any stable state x, any potential CSP-stubborn set at x would have to contain all the actions that are enabled at x.

Definition 4.1. Let $A \subseteq \mho \cup \Im$. A is *CSP-stubborn* at s iff the following four conditions hold.

- $\Im \cap \eta_L(s) \cap A \neq \emptyset$. In other words, A contains at least one invisible action that is enabled at s.
- $\mho \cap \eta_L(s) \cap A = \emptyset$ or $\mho \subseteq A$. In other words, if A contains some visible action that is enabled at s, then A contains all visible actions.
- For each $a \in A \cap \eta_L(s)$ and for each $i \in N_{<n}$, $(a \notin \mho_i \cup \Im_i) \vee (\eta_i(s_i) \subseteq A)$.
- For each $a \in A \setminus \eta_L(s)$, there is some $i \in N_{<n}$ such that $a \in (\mho_i \cup \Im_i) \setminus \eta_i(s_i)$ and $\eta_i(s_i) \subseteq A$.

A is *CSP-eligible* at s iff there is some $Q \subseteq \mho \cup \Im$ such that Q is CSP-stubborn at s and $A = Q \cap \eta_L(s)$. Let $X \subseteq S$. A is *CFFD-stubborn* at $\langle s, X \rangle$ iff A is CSP-stubborn at s and the following condition holds.

- $(\{s\} \times (\Im \cap A) \times (X \cup \{s\})) \cap \Delta = \emptyset$ or $\mho \subseteq A$. In other words, if some successor transition of s leads to some state of $X \cup \{s\}$ and the label of the transition is an invisible action of A, then A contains all visible actions.

A is *CFFD-eligible* at $\langle s, X \rangle$ iff there is some $Q \subseteq \mho \cup \Im$ such that Q is CFFD-stubborn at $\langle s, X \rangle$ and $A = Q \cap \eta_L(s)$. $\square$

Lemma 4.1. *$\mho \cup \Im$ is CSP-stubborn at s and CFFD-stubborn at $\langle s, X \rangle$ for any $X \subseteq S$. If s is not a successor state of itself, then the set of CSP-stubborn sets at s is the same as the set of CFFD-stubborn sets at $\langle s, \emptyset \rangle$. Let $i \in N_{<n}$ and $X \subseteq S$. If $\{L_0, \ldots, L_{n-1}\}$ is strictly action-consistent, $\mho = \emptyset$ and $\eta_i(s_i) \neq \emptyset$, then $\Im_i$ is CSP-stubborn at s and CFFD-stubborn at $\langle s, X \rangle$.*

Proof. The claims are direct consequences of Definition 4.1. □

The optimization considerations in the sequel consider Definition 4.1 simply as a "given hopefully useful definition of constrained stubbornness", without worrying about the reasons for the definition. Proposition 4.1 is due to [12].

Proposition 4.1. *Let f be a function from S to $2^{\mho \cup \Im}$ such that for each state x reachable from ι, either x is stable and $f(x) = \eta_L(x)$ or x is unstable and $f(x)$ is CSP-eligible at L. Let L' be the LTS which has the set of transitions $\{\langle x, a, y \rangle \in \Delta \mid a \in f(x)\}$ and is otherwise the same as L. Then L' is CSP-equivalent to L.* □

There is also a connection between CFFD-eligibility and CFFD-equivalence, but it is difficult to express the connection without presenting an algorithm that in the case of a finite L, constructs an LTS that is CFFD-equivalent to L. The set X of Definition 4.1 represents the "current" contents of a certain depth-first search stack. We omit the details.

From now on, we assume that L is finite. We define the decision problems CSPELIGTR, CFFDELIGTR, CSPELIGST and CFFDELIGST, and the corresponding optimization problems CSPELIGTROPT, CFFDELIGTROPT, CSPELIGSTOPT and CFFDELIGSTOPT as follows.

CSPELIGTR: "Given $m \in N$, does s have a CSP-eligible set A such that the number of successor transitions of s labelled by actions of A is less than or equal to m?" CSPELIGTROPT: "Find a CSP-eligible set A at s such that the number of successor transitions of s labelled by actions of A is the least possible."

CFFDELIGTR: "Given $X \subseteq S$ and $m \in N$, does $\langle s, X \rangle$ have a CFFD-eligible set A such that the number of successor transitions of s labelled by actions of A is less than or equal to m?" CFFDELIGTROPT: "Given $X \subseteq S$, find a CFFD-eligible set A at $\langle s, X \rangle$ such that the number of successor transitions of s labelled by actions of A is the least possible."

CSPELIGST: "Given $m \in N$, does s have a CSP-eligible set A such that the number of successor states of s via transitions labelled by actions of A is less than or equal to m?". CSPELIGSTOPT: "Find a CSP-eligible set A at s such that the number of successor states of s via transitions labelled by actions of A is the least possible."

CFFDELIGST: "Given $X \subseteq S$ and $m \in N$, does $\langle s, X \rangle$ have a CFFD-eligible set A such that the number of successor transitions of s via transitions labelled by actions of A is less than or equal to m?" CFFDELIGSTOPT: "Given $X \subseteq S$, find a CFFD-eligible set A at $\langle s, X \rangle$ such that the number of successor states of s via transitions labelled by actions of A is the least possible."

Theorem 4.1 is a "moral corollary" of Theorem 5.1 of [15]. At this point, the distinction between NP-hardness and NP-completeness is important. We shall return to the question of NP-completeness later in this section.

Theorem 4.1. *The problems CSPELIGTR, CFFDELIGTR, CSPELIGST and CFFDELIGST are NP-hard.*

Proof. The famous NP-complete problem SAT can be reduced simultaneously to all of the four problems, much in the same way as SAT is reduced to the problem ELIGCARD in [15]. The key to success in [15] is the construction of the so called characteristic net. It is possible to construct a strictly action-consistent set of small deterministic LTSs such that their parallel composition is isomorphic to the reachability graph of the characteristic net, the initial state is the reference state and not a successor of itself, all enabled actions are invisible, the successor transitions of the initial state do not share any target state, and there is a bijective cardinality-preserving mapping from the eligible sets of the characteristic net to the CSP-eligible sets of the parallel composition. By Lemma 4.1 we then know that the CSP-eligible sets at the initial state are the same as the CFFD-eligible sets at the pair of the initial state and the empty set. □

In order to organize the transformation of the above mentioned decision and optimization problems into and/or-graph problems, we extend the concept of stubbornness by utilizing the fact that in a sense, Definition 4.1 treats the component LTSs as indivisible units. As can be seen e.g. from Algorithm 2 in Chapter 4 of [1], it is mostly a matter of taste how the domain of elements in stubborn sets is chosen. In our extension, the elements are *blocks* as defined below.

Definition 4.2. The *dependency graph* at s is the undirected graph where the set of vertices is $N_{<n}$ and there is an edge between i and k iff $i \neq k$ and $(\Im_i \cap \Im_k \cap \eta_L(s) \neq \emptyset) \vee ((\mho_i \cap \eta_L(s) \neq \emptyset) \wedge (\mho_k \cap \eta_L(s) \neq \emptyset))$. Φ is the function from $N_{<n}$ to $2^{N_{<n}}$ such that for each $i \in N_{<n}$. $\Phi(i)$ is the set of vertices of the maximal connected subgraph of the dependency graph at s such that $i \in \Phi(i)$. The set of *blocks* at s is $\mathcal{B} = \{\Phi(i) \mid i \in N_{<n}\}$. (Consequently, blocks are pairwise disjoint and form a partition of $N_{<n}$.) Γ is the function from $\mho \cup \Im$ to $2^{N_{<n}}$ such that for each $a \in \mho \cup \Im$, $\Gamma(a) = \{i \in N_{<n} \mid a \in \mho_i \cup \Im_i\}$. ℓ is the function from $2^{N_{<n}}$ to $\mho \cup \Im$ such that for each $Y \subseteq N_{<n}$, $\ell(Y) = \cup_{i \in Y}(\mho_i \cup \Im_i)$. A set $Y \subseteq N_{<n}$ *contributes to a transition* $\langle x, a, y\rangle \in \Delta$ iff $x = s$ and $Y \cap \Gamma(a) \neq \emptyset$. A set $Y \subseteq N_{<n}$ *contributes to a state* $y \in S$ iff there is some $a \in \mho \cup \Im$ such that $\langle s, a, y\rangle \in \Delta$ and Y contributes to $\langle s, a, y\rangle$. ∂ is the function from $2^{N_{<n}}$ to N such that for each $Y \subseteq N_{<n}$, $\partial(Y)$ is the number of those transitions in Δ to which Y contributes. $\sharp$ is the function from $(2^S) \times (2^{N_{<n}})$ to N such that for each $U \subseteq S$ and for each $Y \subseteq N_{<n}$, $\sharp(U, Y)$ is the number of those states in U to which Y contributes. □

Lemma 4.2. *(i) Let $B \in \mathcal{B}$ and $a \in \ell(B) \cap \eta_L(s)$. Then $\Gamma(a) \subseteq B$. (ii) With the same B and a, let $C \in \mathcal{B} \setminus \{B\}$, $q \in \ell(C)$, $y \in S$, $\langle s, a, y\rangle \in \Delta$ and $\langle s, q, y\rangle \in \Delta$. Then $y = s$.*

Proof. The first claim is an obvious consequence of Definition 4.2. So, since B and C are disjoint, it must be the case that $\Gamma(a)$ and $\Gamma(q)$ are disjoint and for each $i \in \Gamma(a) \cup \Gamma(q)$, $y_i = s_i$ where $y = \langle y_0, \ldots, y_{n-1} \rangle$. □

Proposition 4.2. *For any* $\mathcal{Y} \subseteq \mathcal{B}$, $\sharp(S \setminus \{s\}, \cup_{Y \in \mathcal{Y}} Y) = \sum_{Y \in \mathcal{Y}} \sharp(S \setminus \{s\}, Y)$ *and* $\partial(\cup_{Y \in \mathcal{Y}} Y) = \sum_{Y \in \mathcal{Y}} \partial(Y)$. *For any* $B \in \mathcal{B}$,
$\sharp(S, B) \leq \partial(B) = \sum_{a \in \ell(B) \cap \eta_L(s)} \prod_{i \in \Gamma(a)} |(\{s_i\} \times \{a\} \times S_i) \cap \Delta_i|$.

Proof. The claims are obvious consequences of Definition 4.2 and Lemma 4.2. □

Definition 4.3. The set of *visible blocks* at s is $\Omega = \{B \in \mathcal{B} \mid \ell(B) \cap \mho \neq \emptyset\}$. The set of *enabled blocks* at s is $\mathcal{H} = \{B \in \mathcal{B} \mid \ell(B) \cap \eta_L(s) \neq \emptyset\}$. The set of *visibly enabled blocks* at s is $\mathcal{H}_\mho = \{B \in \mathcal{B} \mid \ell(B) \cap \mho \cap \eta_L(s) \neq \emptyset\}$. (Though $\mathcal{H}_\mho \subseteq \Omega \cap \mathcal{H}$, $(\Omega \cap \mathcal{H}) \setminus \mathcal{H}_\mho$ can be nonempty.) The set of *invisibly enabled blocks* at s is $\mathcal{H}_\Im = \{B \in \mathcal{B} \mid \ell(B) \cap \Im \cap \eta_L(s) \neq \emptyset\}$. ($\mathcal{H}_\Im \neq \emptyset$ because throughout this section, we assume s to be unstable.) Θ is the function from $\mho \cup \Im$ to $2^\mathcal{B}$ such that $\forall\, a \in \mho \cup \Im :\ \Theta(a) = \{\Phi(i) \mid (i \in \Gamma(a)) \wedge (a \notin \eta_i(s_i))\}$. $\surd$ is the function from $2^\mathcal{B}$ to $2^{\mho \cup \Im}$ such that $\forall\, \mathcal{Y} \subseteq \mathcal{B} :\ \surd(\mathcal{Y}) = \eta_L(s) \cap (\cup_{Y \in \mathcal{Y}} \ell(Y))$. ¶ is the function from $2^{\mho \cup \Im}$ to $2^\mathcal{B}$ such that $\forall\, A \subseteq \mho \cup \Im :\ \P(A) = \{B \in \mathcal{B} \mid \ell(B) \cap A \neq \emptyset\}$. Let $\mathcal{Y} \subseteq \mathcal{B}$. $\mathcal{Y}$ is *block-CSP-stubborn* at s iff the following three conditions hold.

- $\mathcal{Y} \cap \mathcal{H}_\Im \neq \emptyset$.
- If $\mathcal{Y} \cap \mathcal{H}_\mho \neq \emptyset$, then $\Omega \subseteq \mathcal{Y}$.
- For each $a \in \cup_{Y \in \mathcal{Y}} \cup_{i \in Y} (\eta_i(s_i) \setminus \eta_L(s))$, $\mathcal{Y} \cap \Theta(a) \neq \emptyset$.

$\mathcal{Y}$ is *block-CSP-eligible* at s iff there is some $\mathcal{Z} \subseteq \mathcal{B}$ such that $\mathcal{Z}$ is block-CSP-stubborn at s and $\mathcal{Y} = \mathcal{Z} \cap \mathcal{H}$. Ψ is the function from 2^S to $2^\mathcal{B}$ such that $\forall\, X \subseteq S :\ \Psi(X) = \{\Phi(i) \mid (i \in N_{<n}) \wedge ((\{s\} \times \Im_i \times (X \cup \{s\})) \cap \Delta \neq \emptyset)\}$. (Consequently, $\Psi(X) \subseteq \mathcal{H}_\Im$.) Let $X \subseteq S$. $\mathcal{Y}$ is *block-CFFD-stubborn* at $\langle s, X \rangle$ iff $\mathcal{Y}$ is block-CSP-stubborn at s and the following condition holds.

- If $\mathcal{Y} \cap \Psi(X) \neq \emptyset$, then $\Omega \subseteq \mathcal{Y}$.

$\mathcal{Y}$ is *block-CFFD-eligible* at $\langle s, X \rangle$ iff there is some $\mathcal{Z} \subseteq \mathcal{B}$ such that $\mathcal{Z}$ is block-CFFD-stubborn at $\langle s, X \rangle$ and $\mathcal{Y} = \mathcal{Z} \cap \mathcal{H}$. □

Proposition 4.3. *The following hold for each* $\mathcal{Y} \subseteq \mathcal{B}$, *for each* $A \subseteq \mho \cup \Im$ *and for each* $X \subseteq S$.

- *If* $\mathcal{Y}$ *is block-CSP-eligible at* s, *then* $\P(\surd(\mathcal{Y})) = \mathcal{Y}$ *and* $\surd(\mathcal{Y})$ *is CSP-eligible at* s.
- *If* $\mathcal{Y}$ *is block-CFFD-eligible at* $\langle s, X \rangle$, *then* $\P(\surd(\mathcal{Y})) = \mathcal{Y}$ *and* $\surd(\mathcal{Y})$ *is CFFD-eligible at* $\langle s, X \rangle$.
- *If* A *is CSP-eligible at* s, *then* $\surd(\P(A)) = A$ *and* $\P(A)$ *is block-CSP-eligible at* s.
- *If* A *is CFFD-eligible at* $\langle s, X \rangle$, *then* $\surd(\P(A)) = A$ *and* $\P(A)$ *is block-CFFD-eligible at* $\langle s, X \rangle$.

Proof. A brute force comparison between Definitions 4.1 and 4.3 suffices. □

Definition 4.4. For any and/or-graph $G = \langle V_\otimes, V_\oplus, \kappa, F\rangle$ and for any $\mathcal{Z} \subseteq \mathcal{B}$, G *reflects* $\langle s, \mathcal{Z}\rangle$ iff there is $\mu \in V_\otimes \setminus (\{s\} \cup \mathcal{B})$ such that the following hold.

- $V_\otimes = \{\mu, s\} \cup \mathcal{B}$ and $V_\oplus = (\{\kappa\} \cup \mho \cup \Im) \setminus \eta_L(s)$.
- $F(\kappa) = \mathcal{H}_\Im$, $F(\mu) = \Omega$, and $F(s) = \emptyset$.
- For any $a \in (\mho \cup \Im) \setminus \eta_L(s)$, $F(a) = \Theta(a)$.
- For any $B \in \mathcal{B}$, $F(B) \subseteq \{\mu, s\} \cup \mho \cup \Im$.
- For any $B \in \mathcal{B}$, $F(B) \cap (\mho \cup \Im) = \cup_{i \in B}(\eta_i(s_i) \setminus \eta_L(s))$.
- For any $B \in \mathcal{B}$, $\mu \in F(B)$ iff $B \in \mathcal{H}_\mho \cup \mathcal{Z}$.
- For any $B \in \mathcal{B}$, $s \in F(B)$ iff B contributes to s. □

Proposition 4.4. *Let $X \subseteq S$ and $\mathcal{Y} \subseteq \mathcal{B}$. Let G_1 be any and/or-graph that reflects $\langle s, \emptyset\rangle$, and let G_2 be any and/or-graph that reflects $\langle s, \Psi(X)\rangle$. Then the following hold.*

- *$\mathcal{Y}$ is block-CSP-eligible at s iff $\mathcal{Y} \cup \{s\}$ is $(\mathcal{H} \cup \{s\})$-solid w.r.t. G_1.*
- *$\mathcal{Y}$ is block-CFFD-eligible at $\langle s, X\rangle$ iff $\mathcal{Y} \cup \{s\}$ is $(\mathcal{H} \cup \{s\})$-solid w.r.t. G_2.*
- *If $\mathcal{Y}$ is $(\mathcal{H} \cup \{s\})$-solid w.r.t. G_1, then $\mathcal{Y} \cup \{s\}$ is $(\mathcal{H} \cup \{s\})$-solid w.r.t. G_1.*
- *If $\mathcal{Y}$ is $(\mathcal{H} \cup \{s\})$-solid w.r.t. G_2, then $\mathcal{Y} \cup \{s\}$ is $(\mathcal{H} \cup \{s\})$-solid w.r.t. G_2.*

Proof. A brute force comparison of Definitions 2.1 and 4.3 suffices. □

Propositions 4.2, 4.3 and 4.4 give us a way to reduce each of CSPELIGTR, CFFDELIGTR CSPELIGST and CFFDELIGST to SOLIDWEIGHT and to reduce each of the corresponding optimization problems to SOLIDOPTW. It is obvious that G_1 of the above kind can be constructed in low-degree polynomial time w.r.t. to the length of the input of any of the problems CSPELIGTR, CSPELIGTROPT, CSPELIGST and CSPELIGSTOPT, independently of the way in which the input is encoded.

Respectively, G_2 of the above kind can be constructed in low-degree polynomial time w.r.t. to the length of the input of any of the problems CFFDELIGTR, CFFDELIGTROPT, CFFDELIGST and CFFDELIGSTOPT. However, since X is a part of the input while the number of successor states of s can be exponential w.r.t. the length of the input, the polynomial time requirement necessitates enumeration of X instead of enumerating the successor states of s. (In the case of deterministic LTSs, it is possible to construct in low-degree polynomial time an alternative and/or-graph that expresses Definition 4.1 directly and has all successor states of s as vertices analogous to the above vertex s.)

The reduction of the problems needs appropriate weights for the "reference vertices of solidity" but does not add any constraint to SOLIDWEIGHT or SOLIDOPTW. For CSPELIGTR, CFFDELIGTR, CSPELIGTROPT and CFFDELIGTROPT, the weight of an enabled block B is $\partial(B)$, and the weight of s is 0. An array representing the weight function can be computed in low-degree polynomial time. Due to Lemma 2.1 and Theorem 4.1, we thus get Theorem 4.2.

Theorem 4.2. *CSPELIGTR and CFFDELIGTR are NP-complete.* □

In the case of the problems CSPELIGST, CFFDELIGST, CSPELIGSTOPT and CFFDELIGSTOPT, the weight of an enabled block B is $\sharp(S \setminus \{s\}, B)$, the weight of s is 1. Unfortunately, here we do not have any general polynomial time claim and thus do not know any analogy to Theorem 4.2 either.

5 Conclusions

The contributions of this paper to the stubborn set method are the following. (i) The problem of minimizing the number of successor states was transformed into an and/or-graph problem. By using the translation presented in [15], the and/or-graph problem in question can be transformed into a logic program problem that e.g. Smodels [9, 17] is ready to solve. (ii) The potential difference in complexity between the problem of minimizing the number of successor states and the problem of minimizing the number of successor transitions was taken into discussion. (iii) By means of the difference between CSP- and CFFD-stubbornness, the role of constraints in the and/or-graph construction was concretized.

References

1. P. Godefroid, *Partial-Order Methods for the Verification of Concurrent Systems — An Approach to the State-Explosion Problem*, LNCS 1032, Springer, 1996, 143 p.
2. C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, 1985, 256 p.
3. R. Kaivola, *Equivalences, Preorders and Compositional Verification for Linear Time Temporal Logic and Concurrent Systems* (Doctoral thesis), University of Helsinki, Department of Computer Science, Report A-1996-1, 1996, 185 p.
4. R. Milner, *A Calculus of Communicating Systems*, LNCS 92, Springer, 1980, 171 p.
5. D.A. Peled, "All from One, One for All: on Model Checking Using Representatives," in C. Courcoubetis (Ed.), *Computer Aided Verification (CAV '93)*, LNCS 697, Springer, 1993, pp. 409–423.
6. D.A. Peled, V.R. Pratt, and G.J. Holzmann (Eds.), *Partial Order Methods in Verification*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 29, American Mathematical Society, 1997, 357 p.
7. W. Penczek, M. Szreter, R. Gerth, and R. Kuiper, "Improving Partial Order Reductions for Universal Branching Time Properties," *Fundamenta Informaticae*, Vol. 43, No. 1–4, 2000, pp. 245–267.
8. K. Schmidt, "Stubborn Sets for Model Checking the EF/AG Fragment of CTL," *Fundamenta Informaticae*, Vol. 43, No. 1–4, 2000, pp. 331–341.
9. P. Simons, *Extending and Implementing the Stable Model Semantics* (Doctoral thesis), Helsinki University of Technology, Report HUT-TCS-A58, 2000, 91 p.
10. A. Valmari, "Heuristics for Lazy State Space Generation Speeds up Analysis of Concurrent Systems," in M. Mäkelä, S. Linnainmaa, and E. Ukkonen (Eds.), *STeP-88 (Proceedings of the Finnish Artificial Intelligence Symposium)*, Vol. 2, Helsinki 1988, pp. 640–650.
11. A. Valmari, *Alleviating State Explosion during Verification of Behavioural Equivalence*, Univ. of Helsinki, Dept. of Computer Science, Report A-1992-4, 1992, 57 p.
12. A. Valmari, "Stubborn Set Methods for Process Algebras," in [6], pp. 213–231.
13. A. Valmari and M. Tienari, "Compositional Failure-Based Semantic Models for Basic LOTOS," *Formal Aspects of Computing*, Vol. 7, No. 4, 1995, pp. 440–468.
14. K. Varpaaniemi, *On the Stubborn Set Method in Reduced State Space Generation* (Doctoral thesis), Helsinki University of Technology, Digital Systems Laboratory Report A 51, 1998, 105 p.
15. K. Varpaaniemi, "Stable Models for Stubborn Sets," *Fundamenta Informaticae*, Vol. 43, No. 1–4, 2000, pp. 355–375.
16. Worldwide web, page http://www.tcs.hut.fi/Software/prod/index.html.
17. Worldwide web, page http://www.tcs.hut.fi/Software/smodels/index.html.