HELSINKI UNIVERSITY OF TECHNOLOGY
DIGITAL SYSTEMS LABORATORY
Series **A:** Research Reports
No. **51**; May 1998

ISSN 0783–5396
ISBN 951–22–4064–5

ON THE STUBBORN SET METHOD IN REDUCED STATE SPACE GENERATION

Kimmo Varpaaniemi

Digital Systems Laboratory
Department of Computer Science and Engineering
Helsinki University of Technology
Otaniemi, FINLAND

Helsinki University of Technology
Department of Computer Science and Engineering
Digital Systems Laboratory
P.O. Box 1100
FIN–02015 HUT, FINLAND

HELSINKI UNIVERSITY OF TECHNOLOGY
DIGITAL SYSTEMS LABORATORY
Series **A:** Research Reports
No. **51**; May 1998

ISSN 0783–5396
ISBN 951–22–4064–5

On the Stubborn Set Method in Reduced State Space Generation

Kimmo Varpaaniemi

Abstract: Reachability analysis is a powerful formal method for analysis of concurrent and distributed finite state systems. It suffers from the state space explosion problem, however, i.e. the state space of a system can be far too large to be completely generated. This thesis is concentrated on the application and theory of the stubborn set method which is one of the methods that try to relieve the state space explosion problem. A central topic in the thesis is the verification of nexttime-less LTL (linear time temporal logic) formulas. It is shown how the structure of a formula can be utilized when there is no fairness assumption. Another central topic is the basic problem how stubborn sets should be computed in order to get the best possible result w.r.t. the total time and space consumed in the state search. An algorithm for computing cardinality minimal or almost cardinality minimal (w.r.t. the number of enabled transitions) stubborn sets is presented, together with experiments that indicate that the algorithm is worth of consideration whenever one wants to get proper advantage of the stubborn set method. The thesis also considers how to cut down on space consumption in the stubborn set method and how well the method can be combined with another reduction technique, the sleep set method.

Keywords: reachability analysis, reduced state space generation, stubborn sets, verification of LTL formulas

Dissertation for the degree of Doctor of Technology to be presented with due permission for public examination and debate in Auditorium E at Helsinki University of Technology (Espoo, Finland) on the 29th of May, 1998, at 12 o'clock noon.

Printing: Picaset Oy; Lauttasaari 1998

Helsinki University of Technology
Department of Computer Science and Engineering
Digital Systems Laboratory
P.O. Box 1100
FIN–02015 HUT, FINLAND

Phone: $\frac{09}{+358\text{–}9}$ 4511
Telex: 125 161 htkk fi
Telefax: +358–9–451 3369
E-mail: lab@saturn.hut.fi

Preface

This work has been carried out in the Digital Systems Laboratory of Helsinki University of Technology. I am grateful to Professor Leo Ojala for his continuous support and to PhD Mikko Tiusanen, Dr.Tech. Johan Lilius and Dr.Tech. Nisse Husberg for their helpful comments. In addition, I would like to thank Lic.Tech. Marko Rauhamaa for discussions that partially affected the choice of the subject of this thesis.

I also greatly appreciate the financial support received from Helsinki Graduate School in Computer Science and Engineering, The Academy of Finland, The Emil Aaltonen Foundation, The Technology Development Centre of Finland, Nokia Telecommunications Oy, Telecom Finland Oy, and Commit Oy.

This document has been created by using the LaTeX Document Preparation System as well as the "what you see is *not* what you get" editors GNU Emacs and MS-DOS Editor and, in the X Window System, the drawing programs `idraw` and `xfig`.

The following advice applies to reading of any mathematically oriented publications: if you have difficulties in understanding a proof, try to construct a more understandable proof e.g. by using a pencil and paper. Trying to construct a counterexample is often helpful, too.

Contents

1 Introduction

Concurrent and distributed systems such as telecommunication protocols and process control systems influence and affect the lives of millions of people daily all over the world today. The design of these systems often involves so difficult problems related to timing that the traditional testing and analysis methods are not adequate. One possible solution to this problem is the discerning use of appropriate *formal methods*.

Reachability analysis, also known as *exhaustive simulation* or *state space generation*, is a powerful formal method for detecting errors in concurrent and distributed finite state systems. Strictly speaking, infinite state systems can be analyzed, too, but reachability analysis methods are typically such that they cannot process more than a finite set of states. Nevertheless, we can quite well try to find errors even in cases where we do not know whether or not the complete state space of the system is finite.

Anyway, reachability analysis suffers from the so called *state space explosion problem*, i.e. the complete state space of a system can be far too large w.r.t. the resources needed to inspect all states in the state space. Fortunately, in a variety of cases we do not have to inspect all reachable states of the system in order to get to know whether or not errors of a specified kind exist.

If we have a system consisting of sequential processes that interact with each other, we can imagine the system to be a global sequential system where an action is either a synchronizing action, i.e. a tuple of actions (from distinct processes but not necessarily from all processes) that must be executed simultaneously by the associated processes, or a non-synchronizing action, i.e. an internal action of some process. (Note that even in the case that there is no synchronizing action, a process can affect some condition that is necessary for an execution of an action in another process.) Every possible simultaneous execution of internal actions is simulated by executing the actions in question in every possible order. Any execution of an action sequence in the global sequential system can then be called an *interleaving* of local executions of action sequences in the processes. If two or more interleavings are sufficiently similar to each other, we can call all except one of them *redundant interleavings*.

The *stubborn set method* [68, 73, 74, 75, 76, 77, 78, 79, 80, 81] and the *sleep set method* [25, 26, 27, 28, 30, 31, 38, 49, 55, 99, 100] are state search techniques that are based on the idea that when two executions of action sequences are sufficiently similar to each other, it is not necessary to investigate both of the executions. *Persistent sets* [25, 26, 29, 31, 38, 98, 99, 100] and *ample sets* [51, 55, 56, 57] 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. This similarity is made explicit in [48] where a set is said to be a *stamper set* whenever the set is stubborn or ample or persistent in some way. Other closely related techniques have been presented in e.g. [2, 4, 23, 39, 44, 54, 60, 62, 83, 96, 98, 101]. (The classification of papers is much a matter of taste. For example, it is formally correct to say that the method introduced in [23] is a special form of the persistent method, but it is also formally correct to say that any method that simply generates the complete state space is a special form of the persistent set method.) This thesis is concentrated on the application and theory of the stubborn set method.

The rest of this chapter has been organized in such a way that each section is an

introduction to an equally named chapter.

1.1 Place/transition nets

Chapter 2 presents basic definitions related to *place/transition nets*, the primary formalism to which the stubborn set method is applied in this thesis. *Petri nets* [9, 10, 41, 59, 58, 64] are a widely used model for concurrent and distributed systems, and place/transition nets form a class of Petri nets. The main reason for choosing place/transition nets for the primary formalism is that there is hardly no simple and well-known formalism where the whole theory of the stubborn set method could be put into practice in a more fine-grained way. (For example, the difference between (general) *dynamic stubbornness* and *strong dynamic stubbornness* is significant in place/transition nets but does not seem to have any useful analogy in the theory of stubborn sets for process algebras [80].)

In Figure 1, we have a place/transition net that models the behaviour of two processes. Circles represent places while rectangles represent transitions. The positions of tokens, i.e. black dots in places indicate the local states of the processes. Both of the processes have exactly two possible local states. A combination of simultaneous local states is a global state. Figure 2 illustrates the full reachability graph, i.e. the complete state space, of the net. The vertices in the graph represent the possible global states while the edges show how a transition is able to change a global state. The number of tokens in a place s is denoted by $M(s)$.

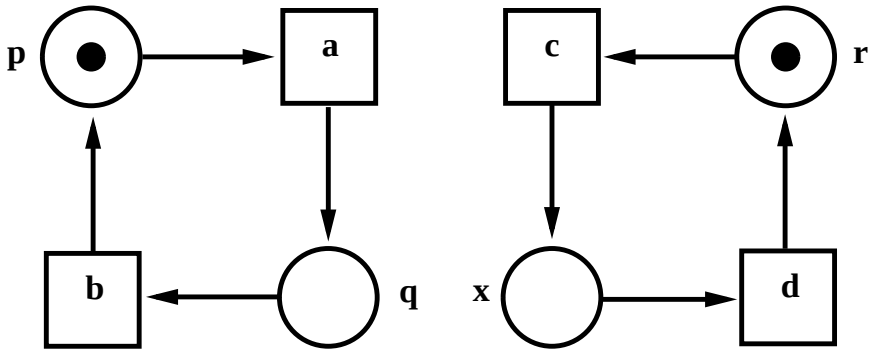


Figure 1: A place/transition net that models the behaviour of two processes.

Though the state space explosion problem is not concretely present in this small example, we can imagine a net where processes are similar to the above processes but the number of processes is n instead of 2. (Such a net can be obtained by duplicating a maximal connected subnet and by renaming the parts in the duplicates.) Then the full reachability graph is an n-dimensional hypercube that has 2^n vertices.

A reduced reachability graph can be constructed by starting from the initial global state and by taking into account some but not necessarily all possible transitions at those global states that become encountered. Assuming that we want a mechanically computed answer to the question if terminal vertices exist in the full reachability graph, we can let some algorithm construct a reduced reachability graph where one process changes its local state twice and the other processes do nothing. Such a

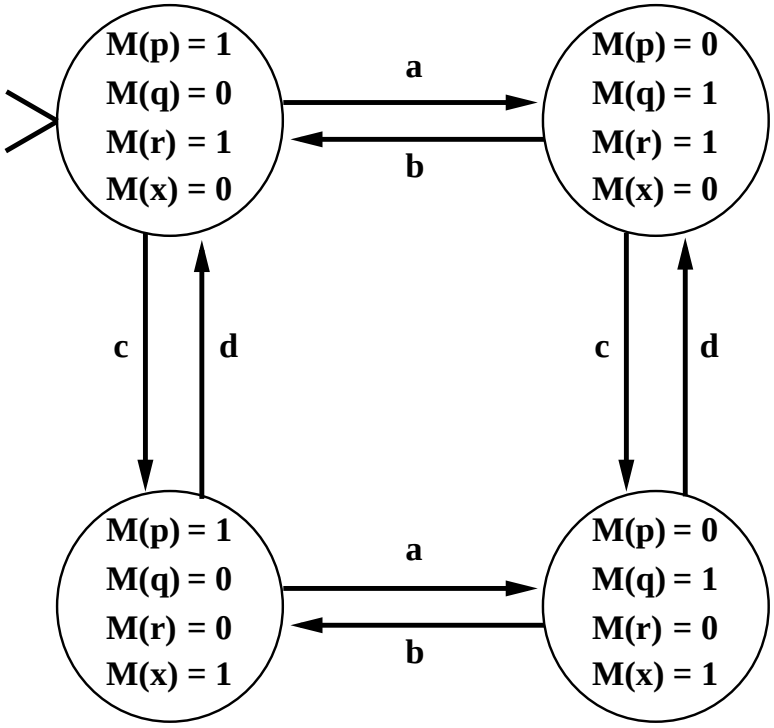


Figure 2: The full reachability graph of the net of Figure 1.

reduced reachability graph, constructible e.g. by using the stubborn set method, has no more than two vertices, independently of n. The obtained graph suffices for showing that the full reachability graph has no terminal vertex. The situation is more complicated if we want to know something else about the full reachability graph, but even then it is by no means likely that we should construct the full reachability graph explicitly.

From a modeller's point of view, place/transition nets may seem impractical because place/transition net models of actual systems tend to be very large. On the other hand, using *high-level nets* [9, 41] one can make compact models in a natural way. Fortunately, a high-level net can often be *unfolded* into a behaviourally equivalent finite place/transition net, and, using the inverse mapping of the unfolding mapping, the place/transition net can be folded back into the high-level net [22, 40]. This also provides a path of extending results on place/transition nets to high-level nets.

1.2 An LTL and associated automata

Linear time temporal logics [19] give us a straightforward though of course a limited way to express what should or should not happen in a concurrent or distributed system. Depending on the context, the abbreviation *LTL* refers either to a specific linear time temporal logic or to "a linear time temporal logic in general". In LTL, the satisfaction of a formula is measured w.r.t. an infinite or deadlock-ended execution. A formula is *valid at a state* iff the formula is satisfied by all those infinite and deadlock-ended executions that start from the state. (In this thesis, the word "deadlock" means "a state where no event is possible". Depending on the case, a deadlock may or may

not be an error.) Verifying a formula typically means showing that the formula is valid at the initial state of the system that is under analysis. Validity is sometimes redefined in such a way that the requirement of satisfaction is restricted to paths of a certain kind. *Fairness assumptions* [21] are one form of such a restriction. A definition of fairness expresses some kind of progress that is expected in situations of a certain kind.

On-the-fly verification of a property means that the property is verified during state space generation, in contrary to the traditional approach where properties are verified after state space generation. As soon as it is known whether the property holds, the generation of the state space can be stopped. Since an erroneous system can have much more states than the intended correct system, it is important to find errors as soon as possible. On the other hand, even in the case that all states become generated, the overhead caused by on-the-fly verification, compared to non-on-the-fly verification, is often negligible.

An LTL formula can be verified on-the-fly by means of a *Büchi automaton* [12, 16, 20, 24, 33, 84]. A Büchi automaton that accepts sequences satisfying the negation of the formula can be constructed automatically and intersected with the state space of the modelled system during the construction of the latter. The state space of the system can easily be thought of as a Büchi automaton. The formula is valid in the state space of the system iff the intersection to be computed, also a Büchi automaton, accepts no sequence. A *tester* [81] is an automaton that is used much in the same way as a Büchi automaton. A remarkable difference is that testers have some additional support for special cases but, to our knowledge, no published automatic construction from arbitrary formulas. Moreover, unlike a typical tester, a Büchi automaton is typically fully synchronized with the system being analyzed.

Chapter 3 presents one version of a linear time temporal logic and describes Büchi automata and testers. The presentation assumes that the system to be analyzed has a place/transition net model. This is sufficient for the later algorithmic considerations.

1.3 Stubbornness and dynamic stubbornness

Chapter 4 introduces stubborn sets and *dynamically stubborn sets*. For historical reasons, "stubbornness" without any preceding attribute is defined in a way that directly indicates how such sets can be computed. When one wants to show results concerning the theoretical properties of the stubborn set method, dynamic stubbornness is a more appropriate notion. When definitions are as they should be, stubbornness implies dynamic stubbornness but not vice versa.

Chapter 4 shows that in place/transition nets, persistence (as defined in the persistent set method) and *conditional stubbornness* [29] are special forms of dynamic stubbornness. Ample sets [51, 55, 56, 57] are considered, too. The concepts of ampleness and dynamic stubbornness have much in common though ampleness is strongly oriented towards verification with fairness assumptions.

1.4 Verification of linear time temporal properties

Chapter 5 is concentrated on the verification of *nexttime-less* LTL formulas with the aid of the stubborn set method. The presentation also covers the verification of basic termination properties, i.e. detecting of reachable deadlocks and deciding if an infinite execution exists.

In the fundamental presentation of stubborn sets in the verification of nexttime-less LTL-formulas [78], the computation of stubborn sets is directed by atomic formulas only, and the reduced state space can be used for verifying any nexttime-less LTL-formula that is constructible from those atomic formulas. Unfortunately, the state space generation algorithm in [78] tends to generate the complete state space when verification is done under some of the most typical fairness assumptions. (In [78], all reduction is gained by utilizing transitions that are "sufficiently uninteresting". A typical fairness assumption makes all transitions "too interesting" in this sense.) The approaches [55, 56] improve the approach of [78] by utilizing the structure of the formula and by allowing a fairness assumption. A weakness in [55, 56] is that the structure of the formula is utilized only in cases when fairness is assumed or the formula expresses a safety property. Chapter 5 improves the method by utilizing the structure of the formula when fairness is not assumed and the formula is arbitrary. (The expression "fairness is not assumed" should be read to mean "no kind of fairness is assumed" though the latter may sound like "unfairness is assumed".) Though the recently published alternative solution [48] can be considered more goal-oriented, it does not cover our approach.

Chapter 5 also considers the verification of nexttime-less LTL-formulas when fairness is assumed. For convenience, we concentrate on *operation fairness* [55], though we could in principle handle some of the weaker fairness assumptions mentioned by [55] in the same way.

The LTL verification approach in [78] can systematically be modified to handle fairness assumptions efficiently, and our approach can be modified quite similarly. It is by no means surprising that we essentially end up in an approach similar to those in [55, 56].

1.5 On heuristics for stubborn set computation

Chapter 6 considers the basic problem how stubborn sets should be computed in order to get the best possible result w.r.t. the total time and space consumed in the state search. The problem is inherently complex. The generating of minimal sized reduced reachability graphs is known to be an NP-hard problem [55] while e.g. the detection of deadlocks is known to be a PSPACE-hard problem [75].T So, we can hardly proceed in any other way than by designing heuristics for different kinds of cases. Chapter 6 presents an algorithm for computing cardinality minimal or almost cardinality minimal (w.r.t. the number of enabled transitions) stubborn sets. The chapter also contains experiments that indicate that the algorithm is worth of consideration whenever one wants to get proper advantage of the stubborn set method.

1.6 Removing redundancy from a stubbornly determined state space

In state space generation, the amount of available storage space is typically a more critical factor than the amount of time we are ready to spend in the generation. We thus need ways to cut down on space consumption, even with considerable additional costs in time consumption. Though *state space caching* [26, 27] is often a sufficient solution to this problem, it has the obvious disadvantage that it does not construct an explicit state space with which a later analysis would be possible. Chapter 7 suggests a way to avoid wasting storage space and still construct a state space where LTL-formulas can be verified later. The presented approach has much in common with the approach of [53], but some differences can still be observed.

1.7 On combining the stubborn set method with the sleep set method

Chapter 8 considers how the stubborn set method can be combined with the sleep set method in order to get a combined attack on the state space explosion. The basic idea of the combination can be found in [99], but there persistent sets are used instead of stubborn sets, and it is not necessarily easy to see how the idea can be extended to concern all stubborn sets. This extension is made explicit in Chapter 8.

Since the original combination was given for a more complicated formalism than place/transition nets, the new combination is presented for a formalism that, to our knowledge, covers all of the formalisms to which the stubborn set method, the sleep set method and the persistent set method have been applied in the literature. The compatibility result is shown by showing a more general result which gives a sufficient condition for a method to be compatible with the sleep set method in the verification of basic termination properties and simple safety properties. The question of compatibility in more challenging verification tasks is more or less an open problem, and before trying to solve it one could first try to solve the problem of what kind of verification tasks are actually supported by the sleep set method. For example, hardly nothing has been published about if or how the sleep set method could support the verification of an arbitrary nexttime-less LTL-formula when fairness is not assumed.

1.8 Other remarks on stubborn sets

Chapter 9 makes some remarks concerning *stubborn sets of high-level nets* [11, 79] and *stubborn sets in symbolic state space generation* [1, 35, 70, 71]. These two areas of research are among the most important as far as the stubborn set research is concerned since on one hand, models are often extremely large on a low level while on the other hand, state spaces are often too large to be handled without abstraction. The main motivation of the chapter is to clarify the limits of the considered approaches.

2 Place/transition nets

This chapter presents basic definitions related to *place/transition nets*, the primary formalism to which the stubborn set method is applied in this thesis. More precisely, we define *place/transition nets with infinite capacities* [64, 65]. (Capacities do not increase expression power and are typically eliminated anyway, so we do not include them in the definitions.) 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^∞) to denote the set of finite (respectively, infinite) words over the alphabet X, ε to denote the empty word and X^+ to denote $X^* \setminus \{\varepsilon\}$. For any alphabet X and for any $\rho \in X^\infty$, ρ is thought of as a function from N to X in such a way that $\rho = \rho(0)\rho(1)\rho(2)\ldots$.

2.1 Nets and reachability graphs

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*, $S \cap T = \emptyset$, 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. The net is *finite* iff $S \cup T$ is finite. 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. □

In our figures, places are circles, transitions are rectangles, and the initial marking is shown by the distribution of tokens, black dots, onto places. A directed arc, i.e. an arrow, is drawn from an element x to an element y iff x is an input element of y. Then $W(x,y)$ is called the *weight* of the arc. As usual, the weight is shown iff it is not equal to 1.

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^* (respectively, T^∞) is called the *set of finite* (respectively, *infinite*) *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

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

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

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. An infinite transition sequence σ is f-enabled

at a marking M ($M[\sigma\rangle_f$ for short) iff all finite prefixes of σ are f-enabled at M. 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 Ψ be the function from $\mathcal{M}$ to 2^T such that for each marking M, $\Psi(M) = T$. From now on in this thesis, we use a plain "$\rangle$" instead of "$\rangle_\Psi$", and as far as the notions of Definition 2.2 are concerned, we replace "Ψ-xxx" by "xxx" (where xxx is any word), with the exception that the Ψ-reachability graph of the net is called the *full reachability graph* of the net. (It is easy to see that this syntactic convention is consistent with Definition 2.1. For example, if t is a transition, the meaning of the expression "$[t\rangle$" is unique though the sub-expression "t" can be understood to mean either the transition t or the transition sequence t.) 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.

Definition 2.3 Let $\langle S, T, W, M_0\rangle$ be a place/transition net. Let f be a function from $\mathcal{M}$ to 2^T and let G be the f-reachability graph of the net. For any edge $\langle M, t, M'\rangle$ of G, t is called the *label* of the edge. (The labelling of the paths of G then follows by a natural extension.) A path of G is called a *terminal path* iff the path is finite and no nonempty transition sequence is f-enabled at the last vertex of the path. A finite path of G is called a *cycle* iff the path has at least one edge and the last vertex of the path is equal to the first vertex of the path. A finite path of G is called an *elementary cycle* iff the path is a cycle and no proper subpath of the path is a cycle. □

2.2 Permutations and equivalences

In the theory of place/transition nets, the concept of a redundant interleaving can be approached by defining permutations of and equivalences between transition sequences.

Definition 2.4 Let $\langle S, T, W, M_0\rangle$ be a place/transition net. Let $T_s \subseteq T$. A finite transition sequence, δ, *T_s-exhausts* a finite transition sequence σ iff for each $t \in T_s$, the number of t's in δ is greater than or equal to the number of t's in σ. The function $\Re$ from $(T^* \cup T^\infty) \times 2^T$ to $T^* \cup T^\infty$ is defined by requiring that for each $Y \in 2^T$, $\Re(\varepsilon, Y) = \varepsilon$, and for each $t_1 \in Y$, for each $t_2 \in T \setminus Y$, for each $\delta \in T^*$ and for each $\rho \in T^\infty$, $\Re(t_1\delta, Y) = t_1\Re(\delta, Y)$, $\Re(t_2\delta, Y) = \Re(\delta, Y)$, and $\Re(\rho, Y) = \Re(\rho(0), Y)\Re(\rho(1), Y)\Re(\rho(2), Y)\ldots$. For any $Y \in 2^T$ and for any $\sigma \in T^* \cup T^\infty$, $\Re(\sigma, Y)$ is called the *Y-restriction* of σ. Let $\Upsilon \subseteq 2^T$. A finite or an infinite transition sequence δ is *Υ-equivalent* to a finite or an infinite transition sequence σ iff for each $Y \in \Upsilon$, $\Re(\delta, Y) = \Re(\sigma, Y)$. Let $\mathcal{T} = \{\{t\} \mid t \in T\}$. A finite or an infinite transition sequence δ is a *permutation of a finite or an infinite transition sequence* σ iff δ is $\mathcal{T}$-equivalent to σ. □

The above Υ can be considered as a set of views to the behaviour of the net. If $T = \{a, b, c, d, e, f, g\}$ and $\Upsilon = \{\{a, b\}, \{c, d\}, \{d, f\}\}$ then $gbdcefa$ is Υ-equivalent to $badfc$ since both of these sequences have the $\{a, b\}$-restriction ba, the $\{c, d\}$-restriction dc, and the $\{d, f\}$-restriction df.

Note that in the case of infinite sequences, the above definition of a permutation does not pay any attention to the possible repeated patterns in the sequences. So, for example the sequence obtained by repeating $bbba$ infinitely many times is a permutation of the sequence obtained by repeating ab infinitely many times. At least our definition is mathematically sound since two finite or infinite transition sequences, σ_0 and σ_1, are permutations of each other iff the following holds: there exist bijections p_0 and p_1 from N to N such that p_1 is the inverse function of p_0 and for each $i \in \{0, 1\}$, $n \in N \setminus \{0\}$ and $t \in T$, if t occurs at least n times in σ_i, then t occurs at least n times in σ_{1-i} and p_i maps the position of the nth occurrence of t in σ_i to the position of the nth occurrence of t in σ_{1-i}.

In many formalisms, *independence of transitions* is the key to how redundant interleavings can be eliminated in verification. Independence is a useful abstraction in place/transition nets, too, though there it seems better to base the detection of redundancy to the recognition of enabled permutations than just to the recognition of independent transitions.

Definition 2.5 Let $\langle S, T, W, M_0\rangle$ be a place/transition net. Transitions t and t' *commute at a marking* M iff $M[tt'\rangle$ and $M[t't\rangle$. Transitions t and t' are *independent at a marking* M iff

$$\begin{array}{l}(M[tt'\rangle \wedge M[t't\rangle) \vee \\ (\neg M[t\rangle \wedge \neg M[t'\rangle) \vee \\ (M[t\rangle \wedge \neg M[t'\rangle \wedge \neg M[tt'\rangle) \vee \\ (M[t'\rangle \wedge \neg M[t\rangle \wedge \neg M[t't\rangle).\end{array}$$

Transitions t and t' are *globally independent* iff they are independent at all reachable markings. □

Our definition of plain independence corresponds to the definition in [29] for conditional independence which in turn is based on the corresponding definition in [45]. Our definition of independence can be obtained from the definition of valid conditional dependency relations, Definition 5 in [29], by taking the necessary conditions for a triple of two transitions and one state to be in the complement of a valid dependency relation, and substituting terms of place/transition nets for the terms of the model of concurrency in [29] in an obvious way.

Clearly, different transitions are independent at a marking iff neither of them can be fired at the marking making the other transition turn from enabled to disabled or from disabled to enabled.

Lemma 2.6 *Claims:*

- *A transition t commutes with itself at a marking iff tt is enabled at the marking.*
- *A transition t is independent of itself at a marking iff tt is enabled or t is disabled at the marking.*

- *Transitions commute at a marking iff they are enabled and independent at the marking.*

Proof. The results are obvious on the basis of Definition 2.5. □

We now give basic definitions for *conditional traces* and *strict traces* that in some form or another occur in e.g. [29, 45, 55]. The theory of conditional traces is a natural extension of the trace theory in [52].

Definition 2.7 Let $\langle S, T, W, M_0 \rangle$ be a place/transition net. A transition sequence δ is a *neighbour of a finite transition sequence* σ iff there exist transitions t and t', and finite transition sequences σ' and σ'' such that $\sigma = \sigma' t t' \sigma''$ and $\delta = \sigma' t' t \sigma''$. A transition sequence δ is a *strict neighbour of a finite transition sequence* σ iff there exist transitions t and t', and finite transition sequences σ' and σ'' such that t and t' are globally independent, $\sigma = \sigma' t t' \sigma''$ and $\delta = \sigma' t' t \sigma''$. Let M be a marking and R the binary relation on T^* such that $\sigma R \delta$ iff σ and δ are enabled at M and neighbours of each other. Let R'' be the reflexive-transitive closure of R. The *conditional trace of a finite transition sequence* σ *at* M is the set of finite transition sequences such that a sequence δ is in the conditional trace of σ at M iff σ is enabled at M and $\sigma R'' \delta$. Let R_1 be the binary relation on T^* such that $\sigma R_1 \delta$ iff σ and δ are enabled at M and strict neighbours of each other. Let R_2 be the reflexive-transitive closure of R_1. The *strict trace of a finite transition sequence* σ *at* M is the set of finite transition sequences such that a sequence δ is in the strict trace of σ at M iff σ is enabled at M and $\sigma R_2 \delta$. A set is an *ending conditional trace at* M iff the set is the conditional trace of some finite transition sequence at M. A set is an *ending strict trace at* M iff the set is the strict trace of some finite transition sequence at M. □

In other words, an ending conditional (respectively, strict) trace is a set of enabled finite transition sequences that can be obtained from each other by repeatedly interchanging adjacent independent (respectively, adjacent globally independent) transitions. The conditional or strict trace of a non-enabled finite transition sequence is empty. The reflexive-transitive closure of R (respectively, R_1) in Definition 2.7 is clearly an equivalence relation, and the conditional (respectively, strict) trace of an enabled finite transition sequence is the equivalence class of the sequence w.r.t. the equivalence relation. What we still need are conditional traces and strict traces for infinite transition sequences. We base the extended definition on finite prefixes.

Definition 2.8 Let $\langle S, T, W, M_0 \rangle$ be a place/transition net. A transition sequence σ_1 is in the *conditional* (respectively, *strict*) *trace of an infinite transition sequence* σ_0 *at a marking* M iff σ_1 is an infinite transition sequence and the following holds: for each $i \in \{0, 1\}$, $\lambda \in T^*$ and $\gamma \in T^\infty$, if $\sigma_i = \lambda\gamma$, then there exist $\mu \in T^*$, $\eta \in T^*$ and $\rho \in T^\infty$ such that $\sigma_{1-i} = \mu\rho$, and $\lambda\eta$ is in the conditional (respectively, strict) trace of μ at M. (Note that this condition does not contain any specific relationship between γ and η.) A set is an *endless conditional* (respectively, *strict*) *trace at* M iff the set is the conditional (respectively, strict) trace of some infinite transition sequence at M. □

Definition 2.8 corresponds to Definition 2.3 in [55] and is needed in the theory of *ample sets* [51, 55, 56, 57] that will be discussed in Chapter 4. An endless conditional

or strict trace can be seen to consist of enabled infinite transition sequences that are permutations of each other. A strict trace of a sequence is always a (proper or non-proper) subset of the conditional trace of the sequence. Also, the conditional or strict trace of an enabled infinite transition sequence is an equivalence class whereas the conditional or strict trace of a non-enabled infinite transition sequence is empty.

3 An LTL and associated automata

This chapter presents one version of a linear time temporal logic and describes Büchi automata and testers that support the verification of formulas of the logic. (Though no general automatic construction of a useful tester from a formula is known, a manual construction, possibly after some "harmless" modifications to the formula, often succeeds.) The presentation assumes that the system to be analyzed has a place/transition net model. This is sufficient for the later algorithmic considerations.

3.1 A linear time temporal logic

Our LTL has effectively the same syntax as the Propositional Linear Temporal Logic (PLTL) in [19]. The semantics are also effectively the same, with the exception that we consider finite executions, too. We make this difference because deadlock-ended executions are important to us whereas the semantic definitions for PLTL assume that every state has a successor.

A formula in our LTL is either atomic or of the form $\bot$, $(A) \Rightarrow (B)$, $\bigcirc(A)$ or $(A)\mathcal{U}(B)$ where A and B are formulas. The following are syntactic abbreviations: $\neg(A)$ means $(A) \Rightarrow (\bot)$, $\top$ means $\neg(\bot)$, $(A) \vee (B)$ means $(\neg(A)) \Rightarrow (B)$, $(A) \wedge (B)$ means $\neg((\neg(A)) \vee (\neg(B)))$, $\Diamond(A)$ means $(\top)\mathcal{U}(A)$, and $\Box(A)$ means $\neg(\Diamond(\neg(A)))$. An atomic formula is a subset of markings of the net, i.e. a subset of $\mathcal{M}$. In our examples, all atomic formulas are of the form "$M(s)$ op k" where $k \in N$, s is a place in the net, op is a comparison operator and the actual meaning of the formula "$M(s)$ op k" is $\{M \in \mathcal{M} \mid M(s) \text{ op } k\}$.

The operators $\bot$, $\Rightarrow$, $\neg$, $\top$, $\wedge$ and $\vee$ are called *propositional.* The other operators are then called non-propositional or *temporal.* Non-propositional operators have the following names: $\bigcirc$ is "nexttime", $\mathcal{U}$ is "until", $\Diamond$ is "eventually" and $\Box$ is "henceforth". A formula is *nexttime-less* iff the formula does not contain any $\bigcirc$. By a *Boolean combination* of formulas from a collection we mean a formula that can be constructed from the formulas of the collection by using propositional operators only. (A single formula can be used several times in the combination whereas it is not necessary to use all formulas of the collection.)

The rules of satisfaction of a formula are given w.r.t. finite and infinite paths in a (reduced or full) reachability graph of the net and are as follows. (We assume that a path always contains at least one vertex and starts with a vertex. Moreover, each finite path ends with a vertex. Also, paths x and y can be concatenated into a path xy iff x is finite and the last vertex of x is the first vertex of y. The path xy is then the path "x continued by y.")

- A path satisfies an atomic formula p iff the first vertex of the path is in p.
- No path satisfies $\bot$.
- A path satisfies $(A) \Rightarrow (B)$ iff the path satisfies B or does not satisfy A.

- A path x satisfies $\bigcirc(A)$ iff there is at least one edge in the path and A is satisfied by the path obtained from x by removing the first vertex and the first edge.
- A path x satisfies $(A)\mathcal{U}(B)$ iff there is a path z and a finite path y such that $x = yz$, z satisfies B, and for any finite paths v and u, $y = uv \neq u$ implies that vz satisfies A.

A formula is *valid at a marking in the graph* iff the formula is satisfied by all those infinite and terminal paths of the graph that start from the marking. (So, a formula $(A) \wedge (B)$ is valid at a marking iff both of A and B are valid at the marking. On the other hand, $(A) \vee (B)$ can be valid at a marking even in the case that neither A nor B is valid at the marking.) *Verifying a formula* means showing that the formula is valid at the initial marking in the full reachability graph of the net.

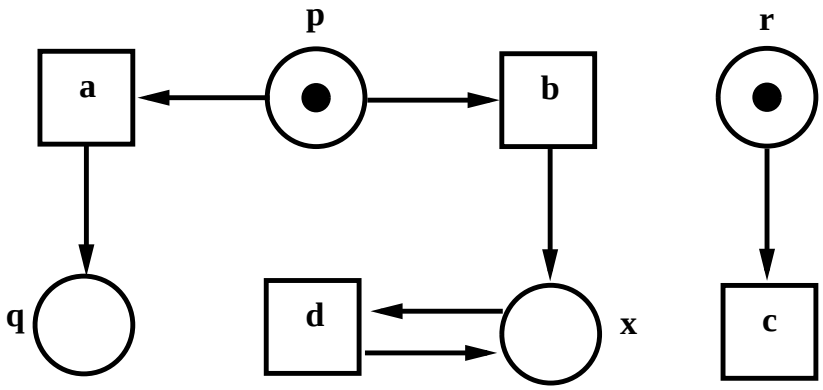


Figure 3: An example net used for LTL considerations.

Let us consider the net in Figure 3. The full reachability graph has exactly two terminal paths that start from the initial marking. The labels of these two paths are *ac* and *ca*. The labels of the infinite paths starting from the initial marking in the full reachability graph are *bddd*..., *cbddd*..., *bcddd*..., *bdcddd*..., *bddcddd*..., etc. All these terminal and infinite paths satisfy the formula

$$(\Diamond(M(x) = 1)) \vee (\Diamond(M(r) = 0)),$$

so the formula is valid at the initial marking in the full reachability graph, The same does not hold for the subformula $\Diamond(M(x) = 1)$, and not for the subformula $\Diamond(M(r) = 0)$ either. This is so because among the above mentioned paths, e.g. the path labelled by *ac* does not satisfy $\Diamond(M(x) = 1)$, whereas the path labelled by *bddd*... does not satisfy $\Diamond(M(r) = 0)$.

For convenience, validity is sometimes redefined in such a way that the requirement of satisfaction is restricted to paths of a certain kind. The restriction may or may not be expressible in LTL. *Fairness assumptions* [21] are one form of such a restriction. Fairness is basically an informal concept, and the choice of a formal definition depends much on the context. Anyway, a definition of fairness expresses some kind of progress that is expected in situations of a certain kind. Also, some definitions of fairness have turned out to be of general interest. To this thesis, we have chosen one of such definitions, *operation fairness* [55] that is a certain type of *strong fairness* [21].

Definition 3.1 Let $\langle S, T, W, M_0 \rangle$ be a place/transition net. A path in the full reachability graph of the net is *operation fair* iff the following holds for each transition t : if t is enabled infinitely many times on the path, then the path contains infinitely many occurrences of t. (Note that all finite paths are thus operation fair.) □

Operation fairness cannot be expressed in our LTL because our version of LTL has no general way to describe the occurrence of a transition in such a way that the description would match only that transition. On the other hand, as can be seen from [42, 55], operation fairness is easily expressible in action-oriented versions of LTL, at least if the net does not have infinitely many transitions.

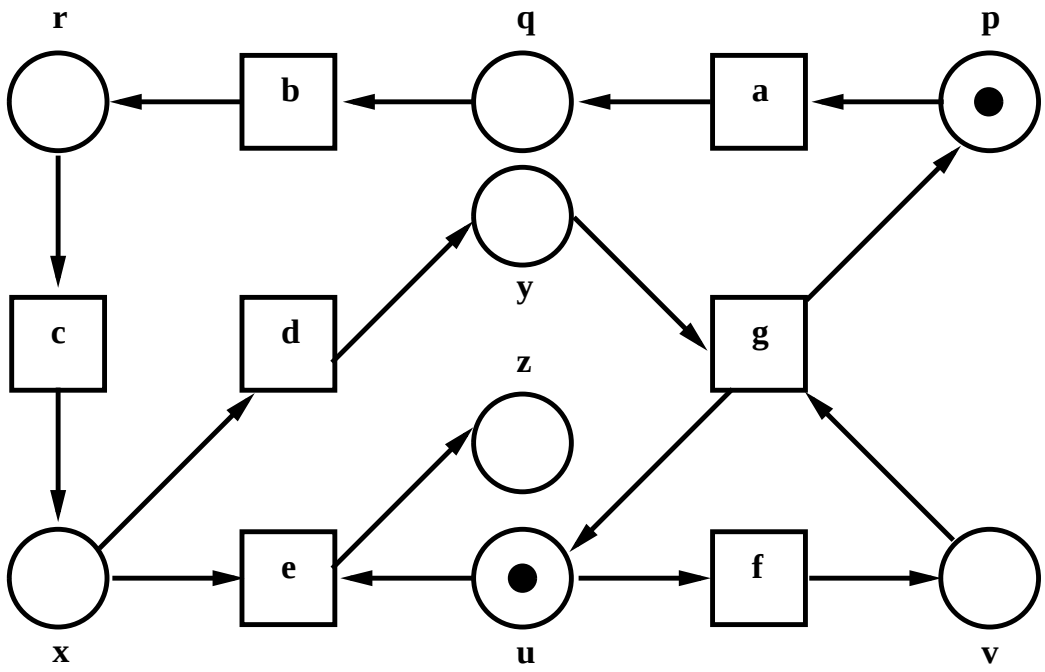


Figure 4: In the full reachability graph of this net, $(abfcdg)(abfcdg)(abfcdg)\ldots$ labels an operation fair path while $(abcfdg)(abcfdg)(abcfdg)\ldots$ does not.

Operation fairness is not guaranteed to be preserved when the order of firing of transitions is changed in such a way that the resulting path has no suffix that would be a suffix of the original path. In the net in Figure 4, transitions c and f are globally independent, and consequently, the infinite sequence $(abcfdg)(abcfdg)(abcfdg)\ldots$ is in the strict trace of the infinite sequence $(abfcdg)(abfcdg)(abfcdg)\ldots$ at the initial marking M_0. However, the path starting from M_0 and being labelled by $(abcfdg)(abcfdg)(abcfdg)\ldots$ is not operation fair though the path starting from M_0 and being labelled by $(abfcdg)(abfcdg)(abfcdg)\ldots$ is operation fair.

3.2 Büchi automata

An LTL-formula can be verified on-the-fly by constructing a *Büchi automaton* [12, 16, 20, 24, 33, 84] that corresponds to the negation of the formula and by computing the intersection of the automaton and a (full or appropriately reduced) reachability graph

in such a way that the intersection is computed simultaneously with the construction of the reduced graph, or even simultaneously with the construction of the automaton.

Definition 3.2 A *Büchi automaton* is a quintuple $B = \langle Q, \Sigma, \Delta, I, \mathcal{F} \rangle$ such that Q is the set of *states*, Σ is the *alphabet*, $\Delta \subseteq Q \times \Sigma \times Q$ is the set of *moves*, $I \subseteq Q$ is the set of *initial states* and $\mathcal{F} \subseteq 2^Q$ is the set of *acceptance sets*. B *accepts a word* σ *over the alphabet* Σ iff $\sigma \in \Sigma^\infty$ and there exist a function q from N to Q, a function ℓ from N to Σ, and a function κ from $N \times \mathcal{F}$ to N such that $q(0) \in I$ and for each $i \in N$, $\ell(i)$ is the $(i+1)$th letter in σ, $\langle q(i), \ell(i), q(i+1) \rangle \in \Delta$, and for each $F \in \mathcal{F}$, $\kappa(i, F) < \kappa(i+1, F)$ and $q(\kappa(i, F)) \in F$. The *language* of B, denoted by $L(B)$, is the set of all those words over Σ that are accepted by the automaton. B is a *finite Büchi automaton* iff $Q \cup \Sigma$ is finite. □

The states and moves in a Büchi automaton can be thought of as vertices and edges of a graph. The above acceptance condition can be described by saying that the word must be a label of an infinite path in the graph in such a way that the path starts from some initial state and contains infinitely many occurrences of members from each acceptance set. Definition 3.2 is a usual definition of a *generalized Büchi automaton* [16, 33]. A classic Büchi automaton becomes defined by requiring that there is exactly one initial state and exactly one acceptance set.

As shown in [16, 33], there is a simple way to map a generalized Büchi automaton that has only finitely many acceptance sets into a classic Büchi automaton that has the same language. (The requirement of a unique initial state in a classic automaton is sometimes ignored in the literature, but a unique initial state is easy to obtain by "duplicating" the moves from given initial states.)

Proposition 3.3, adapted from [33], gives us a clue to how Büchi automata can be used in verification. If a system and a property can be described by two finite Büchi automata in such a way that the language of the first automaton represents infinite executions of the system while the language of the second automaton defines undesirable formal sequences over the same alphabet, it then suffices to construct a Büchi automaton that accepts exactly the words in the intersection of the languages. The checking if that intersection is empty can be done efficiently during the construction [16]. If it turns out that the intersection is not empty, the construction can be stopped, and obtaining a word of the intersection before that stop can be guaranteed without essential loss in efficiency [16].

Proposition 3.3 has also a compositional aspect since if we can map a property into a language, then a disjunction can be mapped into a union of languages while a conjunction can be mapped into an intersection. The automaton B_3 in Proposition 3.3 will informally be called the *union of automata* B_1 and B_2 while B_4 will be called the *intersection of automata* B_1 and B_2. (From a formal point of view, this kind of naming is questionable since the union (respectively, intersection) of B_2 and B_1 should in principle be the same as the union (respectively, intersection) of B_1 and B_2.) Complements are considerably harder to handle than unions and intersections. To our knowledge, the best published universal algorithm for constructing a Büchi automaton for the language $\Sigma^\infty \setminus L(B)$ may construct an automaton where the 2-base logarithm of the number of states is proportional to $n \log_2 n$ when B is a Büchi automaton consisting of n states and having Σ as the alphabet [66]. However, the

worst case complexities do not tell the whole truth. For example, if we construct an automaton for an LTL formula in the way described in [84] and then another automaton for the negation of the formula, these two automata are structurally very close to each other.

Proposition 3.3 *Let* $B_1 = \langle Q_1, \Sigma_1, \Delta_1, I_1, \mathcal{F}_1 \rangle$, *and* $B_2 = \langle Q_2, \Sigma_2, \Delta_2, I_2, \mathcal{F}_2 \rangle$ *be Büchi automata and* $B_3 = \langle Q_3, \Sigma_3, \Delta_3, I_3, \mathcal{F}_3 \rangle$ *and* $B_4 = \langle Q_4, \Sigma_4, \Delta_4, I_4, \mathcal{F}_4 \rangle$ *be quintuples such that*

$$
\begin{aligned}
Q_3 &= (Q_1 \times \{1\}) \cup (Q_2 \times \{2\}), \\
\Sigma_3 &= \Sigma_1 \cup \Sigma_2, \\
\Delta_3 &= \{\langle\langle q_1, 1\rangle, x_1, \langle r_1, 1\rangle\rangle \mid \langle q_1, x_1, r_1\rangle \in \Delta_1\} \cup \\
&\quad \{\langle\langle q_2, 2\rangle, x_2, \langle r_2, 2\rangle\rangle \mid \langle q_2, x_2, r_2\rangle \in \Delta_2\}, \\
I_3 &= (I_1 \times \{1\}) \cup (I_2 \times \{2\}), \\
\mathcal{F}_3 &= \{(F_1 \times \{1\}) \cup (F_2 \times \{2\}) \mid (F_1 \in \mathcal{F}_1) \wedge (F_2 \in \mathcal{F}_2\}, \\
Q_4 &= Q_1 \times Q_2, \\
\Sigma_4 &= \Sigma_1 \cap \Sigma_2, \\
\Delta_4 &= \{\langle\langle q_1, q_2\rangle, x, \langle r_1, r_2\rangle\rangle \mid (\langle q_1, x, r_1\rangle \in \Delta_1) \wedge (\langle q_2, x, r_2\rangle \in \Delta_2)\}, \\
I_4 &= I_1 \times I_2, \ \textit{and} \\
\mathcal{F}_4 &= \{F_1 \times Q_2 \mid F_1 \in \mathcal{F}_1\} \cup \{Q_1 \times F_2 \mid F_2 \in \mathcal{F}_2\}.
\end{aligned}
$$

Claim: B_3 *and* B_4 *are Büchi automata such that* $L(B_3) = L(B_1) \cup L(B_2)$ *and* $L(B_4) = L(B_1) \cap L(B_2)$.

Proof. The case of B_3 is obvious since there can be no path between a state marked by 1 and a state marked by 2. What comes to B_4, the definition of moves makes sure that the finite prefixes in accepted words are as they should be while the definition of acceptance sets makes sure that the acceptance sets of B_1 and B_2 are appropriately represented. □

Compositional construction of a Büchi automaton for a formula is slightly confused by the virtual inflexibility of alphabets. Typically, the set of atomic formulas determines the alphabet, and only the necessary atomic formulas are taken into account. As a result, two "formula automata" are likely not to have equal alphabets. A simple solution to this problem is as follows. Classically, a member of an alphabet in a formula automaton is a subset of atomic formulas. We change this by defining that a member of such an alphabet is a pair of subsets of atomic formulas. In a move, the leftmost subset, X, in the label is the set of those atomic formulas that are "both important and true in the source state". The rightmost subset, Y, in the label is the set of those atomic formulas "the truth value of which is important". (So, X is a subset of Y by definition.) This kind of a move corresponds to a set of classic moves where the source states are equal, the target states are equal as well while the label of a move is Z such that $X = Z \cap Y$. This correspondence is then respected in defining unions and intersections between the new kind of automata. If $\Sigma_1 = 2^{K_1} \times 2^{O_1}$ and $\Sigma_2 = 2^{K_2} \times 2^{O_2}$, it is sufficient to modify Proposition 3.3 as follows:

$$\Sigma_4 = 2^{(K_1 \cup K_2)} \times 2^{(O_1 \cup O_2)}, \text{ and}$$

$$\Delta_4 = \{\langle\langle q_1, q_2\rangle, \langle X_1 \cup X_2, Y_1 \cup Y_2\rangle, \langle r_1, r_2\rangle\rangle \mid \begin{array}{l}\langle q_1, \langle X_1, Y_1\rangle, r_1\rangle \in \Delta_1, \\ \langle q_2, \langle X_2, Y_2\rangle, r_2\rangle \in \Delta_2, \text{ and} \\ (X_1 \cap Y_2) = (X_2 \cap Y_1)\}.\end{array}$$

$L(B_4)$ is no longer $L(B_1) \cap L(B_2)$, but by transforming the alphabets and moves into the classic form we see that the language of the transformation of B_4 is the intersection of the languages of the transformations of B_1 and B_2. From this it follows that $L(B_4)$ is empty iff $L(B_1) \cap L(B_2)$ is empty.

We still need a Büchi automaton that represents the behaviour of a place/transition net. Let G be the full or a reduced reachability graph that contains sufficiently many executions w.r.t. the formula to be verified. We can think of G as a Büchi automaton such that the initial marking of the net is the only initial state of the automaton, the set of markings in G form a trivial acceptance set, no other acceptance set exists, a member of the alphabet is a pair of subsets of atomic formulas whereas the set of moves is the set of edges of G relabelled as follows. The label of a move is a member of the alphabet. The leftmost subset in the label is the set of those atomic formulas that occur in the formula to be verified and contain the source marking. (These are the atomic subformulas that are satisfied by the paths that start from that marking). The rightmost subset in the label is the set of all atomic subformulas of the formula to be verified. The obtained "net automaton" is thus of the same form as the above described formula automaton.

Let us first assume that G has no terminal marking. We can then verify a formula w.r.t. G by constructing an automaton for the negation of the formula and by intersecting the result with the net automaton. The formula is valid at the initial marking in G iff the language of the intersection is empty.

Let us then consider the case that G has a terminal marking. In the verification of LTL-formulas, it is usual to transform deadlock-ended executions into infinite executions. After such a transformation, we have a reachability graph G' that has no terminal marking and can be handled accordingly. The transformation is something like the following. Let us consider an arbitrary place/transition net. We define a pseudo-transition that cannot lead from a marking to a different marking and is enabled iff no actual transition is enabled. Note that the enabledness of the pseudo-transition is defined by means of the enabledness of the actual transitions, so we do not have to worry about whether or not the new system could be simulated by an ordinary net. We do not let the pseudo-transition confuse any transition selecting procedure. At any marking, we imagine as long as possible that the pseudo-transition does not exist. If some actual transition is enabled, we proceed as if there really were no pseudo-transition. If no actual transition is enabled, we fire the pseudo-transition that leads from the marking to the marking itself.

The verification algorithm should of course somehow make a difference between true infinite executions and transformed deadlock-ended executions. One way to guarantee this in the Büchi automaton approach is to use a version of LTL that can describe both states and actions. Deadlock-ended executions can then be characterized in the formula to be verified by referring to the above pseudo-transition. If LTL is action-oriented, the net automaton must be redefined in such a way that the original label of an edge in the reachability graph affects the label of the corresponding move in

the automaton.

A similar action-oriented version of LTL can of course be used for describing the possible fairness assumptions. For example, the restriction of operation fairness to a single transition can be represented by a small Büchi automaton. An automaton representing operation fairness is then simply obtained by intersecting these automata with each other. By intersecting the resulting "fairness automaton" with an automaton that corresponds to the negation of the formula to be verified, we get an automaton that corresponds to the negation of "operation fairness implies the formula to be verified". The order in which the intersection operations are applied is not important, and intersections do not have to be realized in a brute force way since there are verification algorithms that only need to know the components of the intersection [16, 56].

3.3 Testers

In the above considerations, we looked at Büchi automata as if their main purpose were to assist us in the verification of LTL formulas. However, Büchi automata can equally well be thought of as direct specifications to what should not happen in a system. Such negative specifications can be designed without having to think in terms of formulas. To keep things simple, the set of actions of the system under analysis is typically chosen to be the alphabet of the automaton. Such a choice means that things must be expressed by means of actions even if the original informal specification would refer to the states of the system.

When the alphabet of a Büchi automaton is the same as the set of actions of the system under analysis, the intersection of the system and the automaton is much like a parallel composition of two systems. It is then natural to ask if the full synchronization of actions is necessary for the success of the verification task, provided that we are ready to modify the automaton when needed. The answer is negative, at least if the definition of a Büchi automaton is extended in certain directions. A *tester* [81] is one of such generalizations of a Büchi automaton.

When the system being analyzed is a place/transition net, without loss in analysis power we can essentially let a tester to be a place/transition net. When both the system and the tester are place/transition nets, they can be combined into a global system that is in turn essentially a place/transition net.

Definition 3.4 A *tester* is a 9-tuple $\langle S, T, W, M_0, \mathcal{M}_R, \mathcal{M}_D, \mathcal{M}_L, \mathcal{M}_I, T_v \rangle$ such that

- $\langle S, T, W, M_0 \rangle$ is a place/transition net,
- $\mathcal{M}_R \subseteq \mathcal{M}$ is the set of *reject states* where $\mathcal{M}$ is the set of markings of the net,
- $\mathcal{M}_D \subseteq \mathcal{M}$ is the set of *deadlock monitor states,*
- $\mathcal{M}_L \subseteq \mathcal{M}$ is the set of *livelock monitor states,*
- $\mathcal{M}_I \subseteq \mathcal{M}$ is the set of *infinite progress monitor states,* and
- $T_v \subseteq T$ is the set of *visible transitions.*

Let $M \in \mathcal{M}$. M is said to be a *monitor state* iff $M \in \mathcal{M}_R \cup \mathcal{M}_D \cup \mathcal{M}_L \cup \mathcal{M}_I$. M is said to be an *ordinary state* iff M is not a monitor state. Let σ be a finite or an infinite transition sequence of the net. The pair $\langle M, \sigma \rangle$ is a *reject history* iff σ is finite and leads from M to a reject state. The pair $\langle M, \sigma \rangle$ is a *monitored deadlock history* iff σ is finite and leads from M to a terminal deadlock monitor state. The pair $\langle M, \sigma \rangle$ is a *monitored livelock history* iff σ is infinite and there exist $\delta \in T^*$ and $\rho \in (T \setminus T_v)^\infty$ such that $\sigma = \delta\rho$ and δ leads from M to a livelock monitor state. The pair $\langle M, \sigma \rangle$ is a *monitored infinite progress* iff σ is infinite and the path starting from M and being labelled by σ in the full reachability graph of the net contains infinitely many occurrences of infinite progress monitor states and infinitely many occurrences of transitions from T_v. The *language of the tester* is the set of transition sequences σ' such that $\langle M_0, \sigma' \rangle$ is a reject history, a monitored deadlock history, a monitored livelock history or a monitored infinite progress. □

As described in [81], a Büchi automaton having a set of actions, i.e. transitions in our case, as an alphabet can be simulated by a tester where the sets of livelock monitor states and infinite progress monitor states coincide. We have replaced the term "infinite trace" of [81] by the term "infinite progress" since the infinite traces in question are more like plain transition sequences than traces of the form defined in Section 2.2.

We do not define operations between testers. Instead, we define a global system that combines a tester with the net under analysis. We start by defining how two plain place/transition nets are combined into a single net.

Definition 3.5 The *product of place/transition nets* $\langle S_1, T_1, W_1, M_1 \rangle$ and $\langle S_2, T_2, W_2, M_2 \rangle$ is defined iff $S_1 \cap S_2 = \emptyset$. When defined, the product is a place/transition net $\langle S_3, T_3, W_3, M_3 \rangle$ such that

- $S_3 = S_1 \cup S_2$,
- $T_3 = T_1 \cup T_2$,
- $\forall t \in T_1 \forall s \in S_1 \; W_3(s,t) = W_1(s,t) \wedge W_3(t,s) = W_1(t,s)$,
- $\forall t \in T_2 \forall s \in S_2 \; W_3(s,t) = W_2(s,t) \wedge W_3(t,s) = W_2(t,s)$,
- $\forall t \in T_1 \setminus T_2 \; \forall s \in S_2 \; W_3(s,t) = 0 \wedge W_3(t,s) = 0$,
- $\forall t \in T_2 \setminus T_1 \; \forall s \in S_1 \; W_3(s,t) = 0 \wedge W_3(t,s) = 0$,
- $\forall s \in S_1 M_3(s) = M_1(s)$, and
- $\forall s \in S_2 M_3(s) = M_2(s)$. □

Note that the input and output transitions of a place are the same in the product as in the original net. Consequently, every reachable marking of the product is essentially a pair of reachable markings of the component nets. The transitions in $T_1 \cap T_2$ are synchronized in the sense that for any $t \in T_1 \cap T_2$, $M[t\rangle$ in the product iff $M'[t\rangle$ and $M''[t\rangle$ where M' and M'' are the restrictions of M to the component nets. The other transitions behave as if there were no product at all.

Definition 3.6 The *global system* of a place/transition net $\langle S_1, T_1, W_1, M_1 \rangle$ and a tester $\langle S_2, T_2, W_2, M_2, \mathcal{M}_R, \mathcal{M}_D, \mathcal{M}_L, \mathcal{M}_I, T_v \rangle$ is defined iff $S_1 \cap S_2 = \emptyset$. When defined, the global system is a tester $\langle S_3, T_3, W_3, M_3, \mathcal{M}_{RR}, \mathcal{M}_{DD}, \mathcal{M}_{LL}, \mathcal{M}_{II}, T_{vv} \rangle$ such that

- $\langle S_3, T_3, W_3, M_3 \rangle$ is the product of $\langle S_1, T_1, W_1, M_1 \rangle$ and $\langle S_2, T_2, W_2, M_2 \rangle$,
- $\mathcal{M}_{RR} = \{M \in \mathcal{M} \mid \exists M' \in \mathcal{M}_R \; \forall s \in S_2 \; M'(s) = M(s)\}$ where $\mathcal{M}$ is the set of markings of the product,
- $\mathcal{M}_{DD} = \{M \in \mathcal{M} \mid \exists M' \in \mathcal{M}_D \; \forall s \in S_2 \; M'(s) = M(s)\}$,
- $\mathcal{M}_{LL} = \{M \in \mathcal{M} \mid \exists M' \in \mathcal{M}_L \; \forall s \in S_2 \; M'(s) = M(s)\}$,
- $\mathcal{M}_{II} = \{M \in \mathcal{M} \mid \exists M' \in \mathcal{M}_I \; \forall s \in S_2 \; M'(s) = M(s)\}$, and
- $T_{vv} = T_2$. (The set T_v thus does not affect the global system. This is an intentional trick.) □

In a tester-based verification task, the goal is to show that the language of the global system is empty. If we find a word that is in the language of the global system, the word is a counterexample to the goal and there is then no reason to continue. On-the-fly verification is easier with testers than with Büchi automata since the global system is nothing more than a place/transition net equipped with sets of monitor states and a set of visible transitions. The algorithms in [81] can be mechanically modified to apply to our formalism.

Büchi nets [20] have some similarity with the above global systems, but it is not clear how beneficial the similarity is. In [20], the intersection of a Büchi automaton (corresponding to a formula) and a reachability graph of a net is represented by a Büchi net which consists of an ordinary net and an acceptance condition. According to Definition 4 in [20], each transition in the Büchi net has a component representing a move of the automaton and another component representing a transition of the original net. A single transition in the original net can contribute to several transitions in the Büchi net. As explained in [20], an explosion in the number of transitions can be avoided by an alternative definition of a Büchi net where some or all transitions are connected to two fixed *scheduling places*. Unfortunately, as can be seen from the following chapters, such connections are not fruitful from the point of view of the stubborn set method. Moreover, Büchi nets do not have any obvious support for the concept of visible transitions. Consequently, it is by no means obvious how the theory in [81] could benefit from the application of Büchi nets.

4 Stubbornness and dynamic stubbornness

In this chapter we introduce stubborn and dynamically stubborn sets. Section 4.1 is devoted to dynamically stubborn sets. All the stubborn sets that have been defined in the literature for place/transition nets are known to be dynamically stubborn. Dynamically stubborn sets seem to have all the nice properties of (statically) stubborn sets except that the definition of dynamic stubbornness does not seem to imply a practical algorithm for computing dynamically stubborn sets. We base the definitions on the principles in [63]. We also present definitions, strongly corresponding to [29], of *persistent* and *conditionally stubborn* sets in the context of place/transition nets and show how persistence and conditional stubbornness can be rephrased in terms of dynamic stubbornness. We end Section 4.1 by defining *permutation-ampleness*, *conditional-trace-ampleness* and *strict-trace-ampleness* in the spirit of [55] and by showing some connections between dynamic stubbornness and these forms of ampleness.

True (or *static*) *stubbornness* is defined in Section 4.2. Our definition is the definition in [73] modified by taking advantage of the remarks in [73].

The *incremental algorithm* for computing stubborn sets [73, 75] is presented in Section 4.3, and the *deletion algorithm* [74, 75] in Section 4.4. The deletion algorithm is slower than the incremental algorithm, but the incremental algorithm is not guaranteed to produce minimal stubborn sets in any practical sense, unlike the deletion algorithm.

The question of what can be done with stubborn or dynamically stubborn sets will at least partially be answered in the remaining chapters.

4.1 Dynamic stubbornness

We define dynamic stubbornness on the basis of the principles in [63].

Definition 4.1 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 $T_s \subseteq T$ *fulfils the principle of conventional dynamic stubbornness* (*CD* for short) *at* M iff

$$\forall \sigma \in (T \setminus T_s)^* \; \forall \delta \in (T \setminus T_s)^* \; \forall t \in T_s \; M[\sigma \delta t\rangle \Rightarrow M[\sigma t \delta\rangle.$$

A set $T_s \subseteq T$ *fulfils the first principle of strong dynamic stubbornness* (*SD1* for short) *at* M iff

$$\forall \sigma \in (T \setminus T_s)^* \; \forall t \in T_s \; M[\sigma t\rangle \Rightarrow M[t\rangle.$$

A set $T_s \subseteq T$ *fulfils the second principle of strong dynamic stubbornness* (*SD2* for short) *at* M iff

$$\forall \sigma \in (T \setminus T_s)^* \; \forall t \in T_s \; (M[t\rangle \wedge M[\sigma\rangle) \Rightarrow (M[\sigma t\rangle \wedge M[t\sigma\rangle).$$

A set $T_s \subseteq T$ is *dynamically stubborn at* M iff T_s fulfils D1 and D2 at M. A set $T_s \subseteq T$ is *conventionally dynamically stubborn at* M iff T_s fulfils CD and D2 at M. A set $T_s \subseteq T$ is *unconventionally dynamically stubborn at* M iff T_s is dynamically stubborn but not conventionally dynamically stubborn at M. A set $T_s \subseteq T$ is *strongly dynamically stubborn at* M iff T_s fulfils SD1 and SD2 at M and $\exists t \in T_s \; M[t\rangle$. □

The principles D1, D2, CD, SD1, and SD2 are illustrated in Figure 5. The principles D1, D2, SD1, and SD2 are the principles 1*, 2*, 1, and 2 of [63], respectively.

Dynamic stubbornness has been defined in [79], too, but the definitions there are more limited than our definitions. We shall return to this subject later in this section.

Lemma 4.2 shows that the "complement criterion" in D1 could be modified to some extent without changing the meaning of the principle.

Lemma 4.2 *Assumptions:*

- $\langle S, T, W, M_0\rangle$ *is a place/transition net.*
- M *is a marking of the net.*
- *A set* $T_s \subseteq T$ *fulfils D1 at* M.
- *A sequence* σT^* *is enabled at* M.
- *A set* $L \subseteq T$ *is the set of those transitions that occur in* σ.

Claim: $T_s \cap L \neq \emptyset$ *iff* $\{t \in T_s \cap L \mid M[t\rangle\} \neq \emptyset$.

Proof. The "if" -part is obvious. We show the "only if"-part. Let $T_s \cap L \neq \emptyset$. From the definition of L it then follows that there exist $t'' \in T_s \cap L$, $\delta \in (L \setminus T_s)^*$ and $\delta' \in L^*$ in such a way that $\delta t'' \delta' = \sigma$. By D1, t'' is enabled at M. □

Lemma 4.3 shows how the dynamic key transitions of a dynamically stubborn set are able to "postpone" enabled finite and infinite sequences of transitions of the complement of the dynamically stubborn set. (Let us recall from the explanation immediately below Definition 2.2 that an infinite transition sequence is enabled at a marking iff all finite prefixes of the sequence are enabled at the marking.)

Lemma 4.3 *Assumptions:*

- $\langle S, T, W, M_0\rangle$ *is a place/transition net.*
- *A marking* M *is a nonterminal marking of the net.*
- *A set* $T_s \subseteq T$ *is dynamically stubborn at* M.

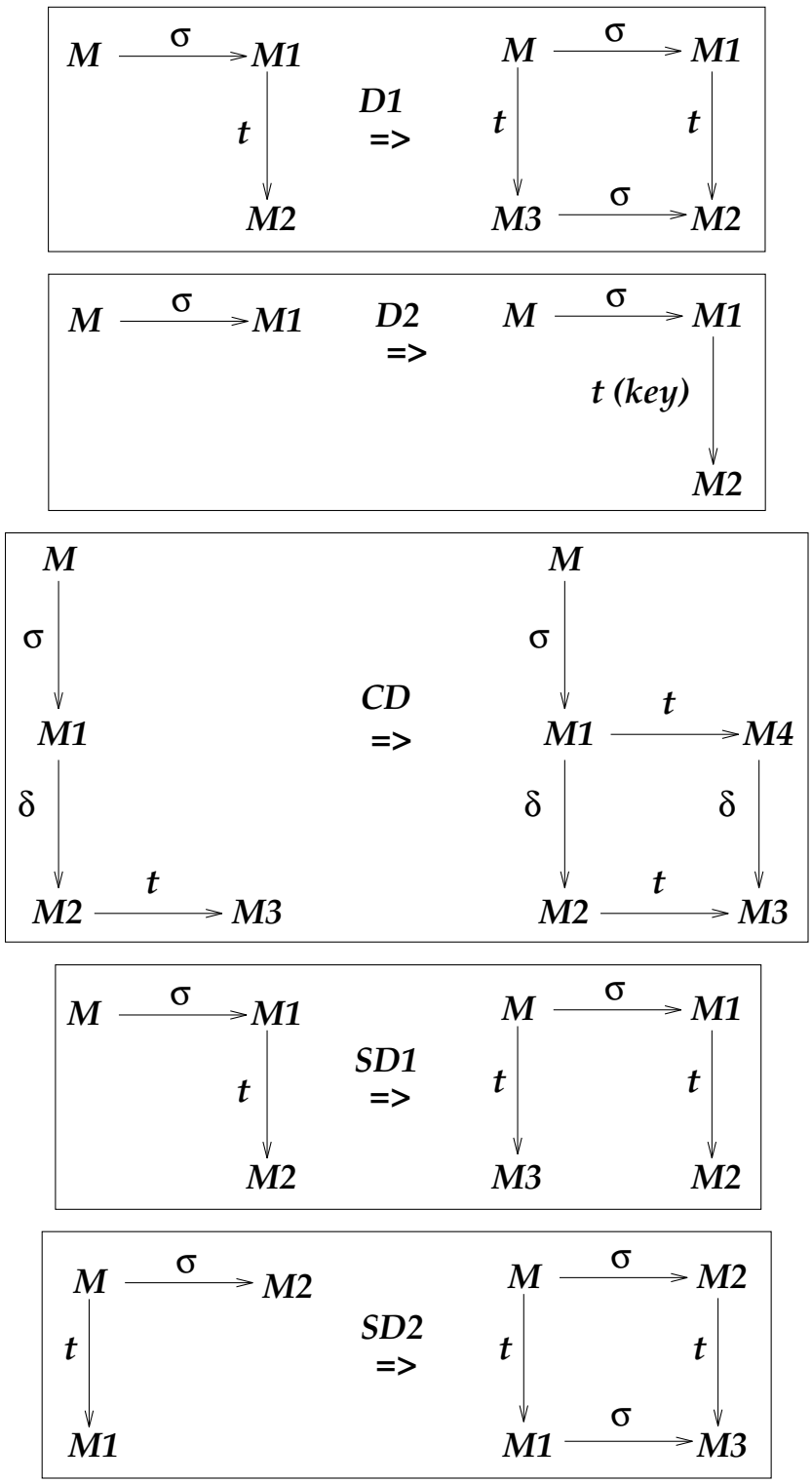


Figure 5: The principles of dynamic (D1 and D2), conventional dynamic (CD) and strong dynamic (SD1 and SD2) stubbornness.

- *A transition $t \in T_s$ is a dynamic key transition of T_s at M.*
- *A transition sequence σ is in $(T \setminus T_s)^* \cup (T \setminus T_s)^\infty$.*
- *The sequence σ is enabled at M.*

Claim: The sequence $t\sigma$ is enabled at M.

Proof. Let $\delta \in (T \setminus T_s)^*$ and $\rho \in (T \setminus T_s)^\infty \cup \{\varepsilon\}$ be any sequences such that $\delta\rho = \sigma$. (If σ is a finite sequence, then $\delta = \sigma$ and $\rho = \varepsilon$.) Since t is a dynamic key transition of T_s at M, the sequence δt is enabled at M. From D1 it then follows that the sequence $t\delta$ is enabled at M. □

We now start handling the different degrees of dynamic stubbornness.

Lemma 4.4 *If a set is conventionally dynamically stubborn at a marking, the set is dynamically stubborn at the marking. A set is strongly dynamically stubborn at a marking iff the set is dynamically stubborn at the marking and each enabled transition in the set is a dynamic key transition of the set at the marking.*

Proof. The results follow trivially from Definition 4.1. □

As one might expect, unconventionally dynamically stubborn sets exist. In the net in Figure 6, $\{a, b\}$ is dynamically stubborn but not conventionally dynamically stubborn at the initial marking since $M_0[cdeb\rangle$ and $\neg M_0[cdb\rangle$. The set $\{c\}$ is strongly dynamically stubborn at the initial marking.

A set can be conventionally dynamically stubborn without being strongly dynamically stubborn. In the net in Figure 7, the only dynamically stubborn sets at the initial marking are $\{t_0, t_1\}$, $\{t_1, t_2\}$, and $\{t_0, t_1, t_2\}$. The sets $\{t_0, t_1\}$ and $\{t_1, t_2\}$ are conventionally dynamically stubborn but not strongly dynamically stubborn at the initial marking since $\neg M_0[t_1t_2\rangle$ and $\neg M_0[t_1t_0\rangle$.

Lemma 4.5 *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 SD2 at M iff*

$$\forall\sigma \in (T \setminus T_s)^*\ \forall\delta \in (T \setminus T_s)^*\ \forall t \in T_s\ (M[t\rangle \wedge M[\sigma\delta\rangle) \Rightarrow M[\sigma t\delta\rangle.$$

Proof. The "if"-part is obvious since we can let $\sigma = \varepsilon$ in one context and $\delta = \varepsilon$ in another context. Let us prove the "only if" -part. Let a set $T_s \subseteq T$ fulfil SD2 at M. Let $\sigma \in (T \setminus T_s)^*$, $\delta \in (T \setminus T_s)^*$, $t \in T_s$, $M[t\rangle$, and $M[\sigma\delta\rangle$. Using SD2 for both $\sigma\delta$ and σ, we get $M[t\sigma\delta\rangle$ and $M[\sigma t\rangle$. As σt and $t\sigma$ lead to the same marking, we have $M[\sigma t\delta\rangle$. □

Lemma 4.6 *If a set is strongly dynamically stubborn at a marking, the set is conventionally dynamically stubborn at the marking.*

Proof. The result follows trivially from Definition 4.1 and Lemma 4.5. □

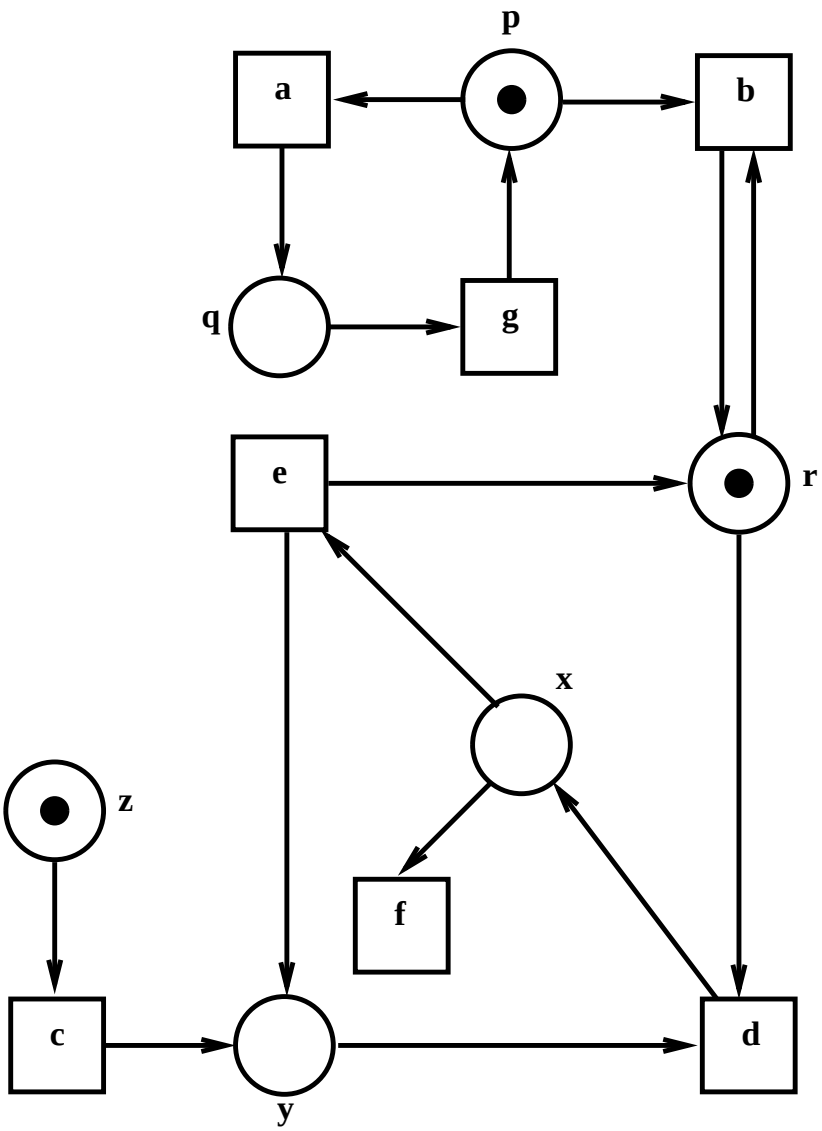


Figure 6: The set $\{a, b\}$ is unconventionally dynamically stubborn.

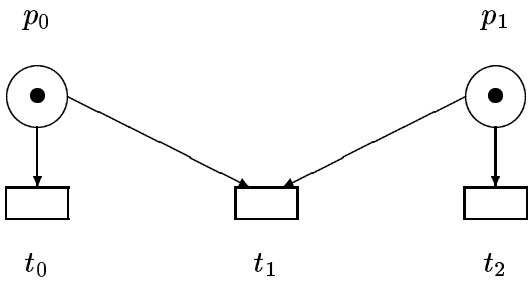


Figure 7: The set $\{t_0, t_1\}$ is conventionally but not strongly dynamically stubborn.

Lemma 4.7 *Let $\langle S, T, W, M_0 \rangle$ be a place/transition net. Let $M \in \mathcal{M}$. A set $T_s \subseteq T$ is strongly dynamically stubborn at M iff*

$$\forall \sigma \in (T \setminus T_s)^* \; \forall M' \in \mathcal{M}$$
$$M[\sigma\rangle M' \Rightarrow (T_s \textit{ is strongly dynamically stubborn at } M').$$

Proof. The "if"-part is obvious since $M[\varepsilon\rangle M$. Let us prove the "only if" -part. Let a set $T_s \subseteq T$ be strongly dynamically stubborn at M. Let $\sigma \in (T \setminus T_s)^*$, $\delta \in (T \setminus T_s)^*$, $t \in T_s$, $M' \in \mathcal{M}$, and $M[\sigma\rangle M'$. If $M'[\delta t\rangle$, we have $M[\sigma\delta t\rangle$, so by SD1 and SD2 for M it follows that $M[\sigma t\rangle$, which implies $M'[t\rangle$. The set T_s thus fulfils SD1 at M'. If $M'[t\rangle$ and $M'[\delta\rangle$, we have $M[\sigma t\rangle$ and $M[\sigma\delta\rangle$, so by SD1 and SD2 for M and Lemma 4.5 it follows that $M[\sigma\delta t\rangle$ and $M[\sigma t\delta\rangle$, which implies $M'[\delta t\rangle$ and $M'[t\delta\rangle$. The set T_s thus fulfils SD2 at M'. If $M[t\rangle$, SD2 for M implies $M[\sigma t\rangle$, so $M'[t\rangle$. Some transition in T_s is thus enabled at M' since some transition in T_s is enabled at M. □

Lemma 4.7 states that every sequence of such transitions that are not in a given strongly dynamically stubborn set leaves the set strongly dynamically stubborn. Lemma 4.8 states the similar result for conventionally dynamically stubborn sets.

Lemma 4.8 *Let $\langle S, T, W, M_0 \rangle$ be a place/transition net. Let $M \in \mathcal{M}$. A set $T_s \subseteq T$ is conventionally dynamically stubborn at M iff*

$$\forall \sigma \in (T \setminus T_s)^* \; \forall M' \in \mathcal{M}$$
$$M[\sigma\rangle M' \Rightarrow (T_s \textit{ is conventionally dynamically stubborn at } M').$$

Proof. The "if"-part is obvious since $M[\varepsilon\rangle M$. Let us prove the "only if" -part. Let a set $T_s \subseteq T$ be conventionally dynamically stubborn at M. Let $\sigma \in (T \setminus T_s)^*$, $\delta \in (T \setminus T_s)^*$, $\delta' \in (T \setminus T_s)^*$, $t \in T_s$, $M' \in \mathcal{M}$, and $M[\sigma\rangle M'$. If $M'[\delta\delta' t\rangle$, we have $M[\sigma\delta\delta' t\rangle$, so by CD for M it follows that $M[\sigma\delta t\delta'\rangle$, which implies $M'[\delta t\delta'\rangle$. The set T_s thus fulfils CD at M'. Let τ be a dynamic key transition of T_s at M. If $M'[\delta\rangle$, we have $M[\sigma\delta\rangle$, so by D2 for M it follows that $M[\sigma\delta\tau\rangle$. The transition τ is thus a dynamic key transition of T_s at M'. □

There is no lemma analogous to Lemmas 4.7 and 4.8 for all dynamically stubborn sets. Let $M_0[c\rangle M'$ and $M'[d\rangle M''$ in the net in Figure 6. The set $\{a, b\}$ is dynamically stubborn at M_0 and M' but not at M'' since $M''[eb\rangle$ and $\neg M''[b\rangle$.

Lemma 4.9 states that a set is conventionally dynamically stubborn iff the set is dynamically stubborn and every sequence of such transitions that are not in the set leaves the set dynamically stubborn.

Lemma 4.9 *Let $\langle S, T, W, M_0 \rangle$ be a place/transition net. Let $M \in \mathcal{M}$. A set $T_s \subseteq T$ is conventionally dynamically stubborn at M iff*

$$\forall \sigma \in (T \setminus T_s)^* \; \forall M' \in \mathcal{M}$$
$$M[\sigma\rangle M' \Rightarrow (T_s \textit{ is dynamically stubborn at } M').$$

Proof. The "only if" -part follows directly from Lemmas 4.4 and 4.8. Let us prove the "if"-part. Let $T_s \subseteq T$, and

$$\forall \sigma' \in (T \setminus T_s)^* \; \forall M' \in \mathcal{M} \; M[\sigma'\rangle M' \Rightarrow (T_s \text{ is dynamically stubborn at } M').$$

Let $\sigma \in (T \setminus T_s)^*$, $\delta \in (T \setminus T_s)^*$, $t \in T_s$, and $M' \in \mathcal{M}$ be such that $M[\sigma\delta t\rangle$ and $M[\sigma\rangle M'$. By D1 for M' we have $M'[t\delta\rangle$, so $M[\sigma t\delta\rangle$. The set T_s thus fulfils CD at M. Since $M[\varepsilon\rangle M$, T_s is dynamically stubborn at M. The set T_s thus fulfils D2 at M. □

Lemma 4.9 gives a useful alternative characterization of conventionally dynamically stubborn sets. We shall use this alternative characterization in Section 4.2.

Lemma 4.10 *Let $\langle S, T, W, M_0\rangle$ be a place/transition net. Let M be a marking of the net, and T_s and T_e subsets of T such that*

$$\{t \in T_s \mid M[t\rangle\} \subseteq T_e, \text{ and } T_e \subseteq T_s.$$

If T_s is dynamically stubborn at M, T_e is dynamically stubborn at M. If T_s is conventionally dynamically stubborn at M, T_e is conventionally dynamically stubborn at M. If T_s is strongly dynamically stubborn at M, T_e is strongly dynamically stubborn at M.

Proof. (i) Let T_s be dynamically stubborn at M. We show that

$$\forall \sigma \in (T \setminus T_e)^*\ M[\sigma\rangle \Rightarrow \sigma \in (T \setminus T_s)^*.$$

Let $\sigma \in (T \setminus T_e)^*$ and $\delta \in (T \setminus T_s)^*$ be such that $M[\sigma\rangle$ and δ is the longest finite prefix of σ not containing any transition in T_s. If $\delta \neq \sigma$, the first transition after δ in σ is enabled at M by D1 for T_s. Since no transition in $T_s \setminus T_e$ is enabled at M, we conclude that $\delta = \sigma$, so $\sigma \in (T \setminus T_s)^*$.

(ii) Since conventionally dynamically stubborn sets and strongly dynamically stubborn sets are dynamically stubborn by Lemma 4.4, the result of part (i) holds for them, too. Then D1 for T_s implies D1 for T_e, D2 for T_s implies D2 for T_e, CD for T_s, implies CD for T_e, SD1 for T_s implies SD1 for T_e, and SD2 for T_s, implies SD2 for T_e. □

Lemma 4.10 states that if we remove disabled transitions from a dynamically stubborn (conventionally dynamically stubborn, strongly dynamically stubborn) set, the remaining set is dynamically stubborn (conventionally dynamically stubborn, strongly dynamically stubborn). For example, if a dynamically stubborn set is minimal w.r.t. set inclusion, by Lemma 4.10 the set consists of enabled transitions only.

We define persistence and conditional stubbornness in such a way that the definitions correspond to the definitions in [29]. Our definitions can be obtained from the definitions 7 and 8 in [29] by substituting terms of place/transition nets for the terms of the model of concurrency in [29] in an obvious way.

Definition 4.11 Let $\langle S, T, W, M_0\rangle$ be a place/transition net. Let $M \in \mathcal{M}$. A set $T_s \subseteq T$ *fulfils the principle of persistence and conditional stubbornness* (*PE* for short) *at M* iff

$$\begin{aligned}&\forall \sigma \in (T \setminus T_s)^*\ \forall t \in T_s\ \forall t' \in T \setminus T_s\ \forall M' \in \mathcal{M}\\ &(M[t\rangle \wedge M[\sigma\rangle M' \wedge M'[t'\rangle) \Rightarrow\\ &(t \text{ and } t' \text{ are independent at } M').\end{aligned}$$

A set $T_s \subseteq T$ is *persistent at M* iff T_s fulfils PE at M and $\forall t \in T_s\ M[t\rangle$. A set $T_s \subseteq T$ is *conditionally stubborn at M* iff T_s fulfils SD1 and PE at M and $\exists t \in T_s\ M[t\rangle$. □

Clearly, the "$M[t\rangle\wedge$" in PE is redundant in the definition of persistence since all transitions in persistent sets are enabled. Looking at PE and proceeding inductively w.r.t. the length of σ, one observes that "commute" could be substituted for "are independent" in PE. Combining this observation with Lemma 4.5, one concludes that PE is nothing but SD2. Consequently, we rid ourselves of the concepts of persistence and conditional stubbornness:

Lemma 4.12 *A set fulfils PE at a marking iff the set fulfils SD2 at the marking. A set is conditionally stubborn at a marking iff the set is strongly dynamically stubborn at the marking.*

Proof. We show that PE is equivalent to SD2. The second statement then follows directly from Definitions 4.1 and 4.11. Let $\langle S, T, W, M_0\rangle$ be a place/transition net. Let $M \in \mathcal{M}$ and $T_s \subseteq T$.

(i) We prove that SD2 implies PE. Let T_s fulfil SD2 at M. Let $\sigma \in (T \setminus T_s)^*$, $t \in T_s$, $t' \in T \setminus T_s$, $M' \in \mathcal{M}$, $M[t\rangle$, $M[\sigma\rangle M'$, and $M'[t'\rangle$. By Lemma 4.5 we have both $M'[t't\rangle$ and $M'[tt'\rangle$. The transitions t and t' are thus independent at M'.

(ii) We prove that PE implies SD2. Let T_s fulfil PE at M. We use induction on the length of finite transition sequences to show that T_s fulfils SD2 at M. The principle SD2 is fulfilled trivially when restricted to ε. Our induction hypothesis is that SD2 is fulfilled when restricted to finite transition sequences of length $n \geq 0$. We show that SD2 is then fulfilled when restricted to finite transition sequences of length $n+1$. Let $\delta \in (T \setminus T_s)^*$, $t' \in T \setminus T_s$, and $t \in T_s$ be such that $M[t\rangle$, $M[\delta t'\rangle$, and δ is of length n. Let $M' \in \mathcal{M}$ be such that $M[\delta\rangle M'$. The transition t' is then enabled at M'. By the induction hypothesis we have $M[\delta t\rangle$ and $M[t\delta\rangle$. The transition t is thus enabled at M'. The principle PE then implies that t and t' are independent at M'. As we recall from Lemma 2.6, transitions commute at a marking iff they are enabled and independent at the marking. So t and t' commute at M'. Thus $M'[tt'\rangle$ and $M'[t't\rangle$, and consequently $M[\delta tt'\rangle$ and $M[\delta t't\rangle$. As already mentioned, we have $M[t\delta\rangle$, so $t\delta$ leads from M to the same marking as δt. We thus have $M[t\delta t'\rangle$. □

Lemma 4.13 *A set is a nonempty persistent set at a marking iff the set is a conditionally stubborn set at the marking and does not contain any transition that is disabled at the marking. The set of enabled transitions of any conditionally stubborn set is a nonempty persistent set.*

Proof. The first statement follows from the fact that a persistent set fulfils SD1 trivially since all its transitions are enabled. The second statement follows trivially from the first statement and lemmas 4.10 and 4.12. □

Lemma 4.14 *A set is a nonempty persistent set at a marking iff the set is a strongly dynamically stubborn set at the marking and does not contain any transition that is disabled at the marking. The set of enabled transitions of any strongly dynamically stubborn set is a nonempty persistent set.*

Proof. The result follows trivially from Lemmas 4.12 and 4.13. □

We now turn to the dynamically stubborn sets in [79]. The prefix AV used in the sequel comes from the name Antti Valmari.

Definition 4.15 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 principle of AV-strong dynamic stubbornness* (*AVSD* for short) *at* M iff

$$\forall t \in T_s \; \forall t' \in T \setminus T_s \; \forall M' \in \mathcal{M} \; (M[t\rangle \wedge M'[t\rangle \wedge M'[t'\rangle) \Rightarrow (M'[tt'\rangle \wedge M'[t't\rangle).$$

A set $T_s \subseteq T$ is *AV-strongly dynamically stubborn at* M iff T_s fulfils SD1 and AVSD at M and $\exists t \in T_s \; M[t\rangle$. □

Our AV-strong dynamic stubbornness is equivalent to the strong dynamic stubbornness defined in [79], because of the obvious equivalence between Definition 2.2 in [79] and our Definition 4.15. The principle AVSD is illustrated in Figure 8.

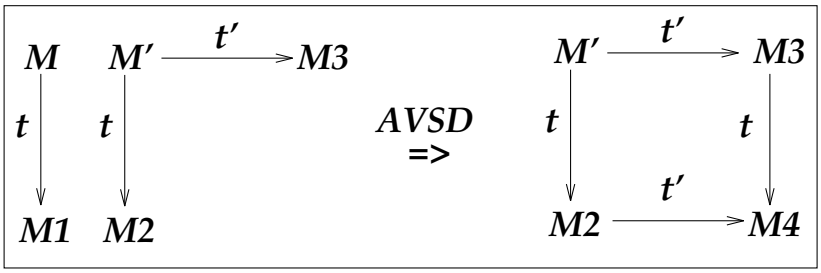


Figure 8: The principle of AV-strong dynamic stubbornness.

Lemma 4.16 *If a set is AV-strongly dynamically stubborn at a marking, the set is strongly dynamically stubborn at the marking.*

Proof. Theorem 2.5 in [79] shows that if a set is an AV-strongly dynamically stubborn set at a marking, the set fulfils D1 at the marking. An AV-strongly dynamically stubborn set contains an enabled transition by Definition 4.15. By Lemma 4.4 it then suffices to show that each enabled transition of an AV-strongly dynamically stubborn set at a marking is a dynamic key transition of the set at the marking. Let $\langle S, T, W, M_0 \rangle$ be a place/transition net. Let M be a marking of the net, and T_s a subset of T such that T_s is AV-strongly dynamically stubborn at M. Let a transition $t \in T_s$ be enabled at M. We show that

$$\forall \sigma \in (T \setminus T_s)^* \; M[\sigma\rangle \Rightarrow M[\sigma t\rangle.$$

We use induction on the length of σ. The claim holds trivially when restricted to $\sigma = \varepsilon$. Our induction hypothesis is that the claim holds when restricted to any σ of length $n \geq 0$. Let $\sigma \in (T \setminus T_s)^*$, $t' \in T \setminus T_s$, and $M' \in \mathcal{M}$ be such that σ is of length n, $M[\sigma\rangle M'$ and $M'[t'\rangle$. Using the induction hypothesis, we get $M[\sigma t\rangle$ which implies $M'[t\rangle$. We already have $M[t\rangle$ and $M'[t'\rangle$. Since T_s fulfils AVSD at M, $M'[t't\rangle$. Thus $M[\sigma t't\rangle$. □

The converse of Lemma 4.16 does not hold. In the net in Figure 9, $\{c\}$ is strongly dynamically stubborn but not AV-strongly dynamically stubborn at the initial marking since $M_0[cc\rangle$, $M_0[cd\rangle$, and $\neg M_0[ccd\rangle$.

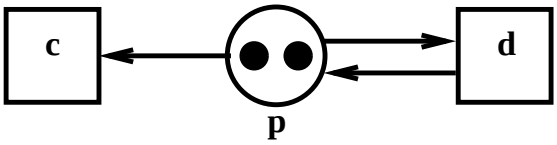


Figure 9: The set $\{c\}$ is strongly but not AV-strongly dynamically stubborn.

Figure 10 illustrates the classes of dynamically, conventionally dynamically, strongly dynamically, and AV-strongly dynamically stubborn sets at a marking M. The inclusions follow from Lemmas 4.4, 4.6 and 4.16. By the presented examples related to Figures 6, 7, and 9 we know that each of the inclusions can be strict.

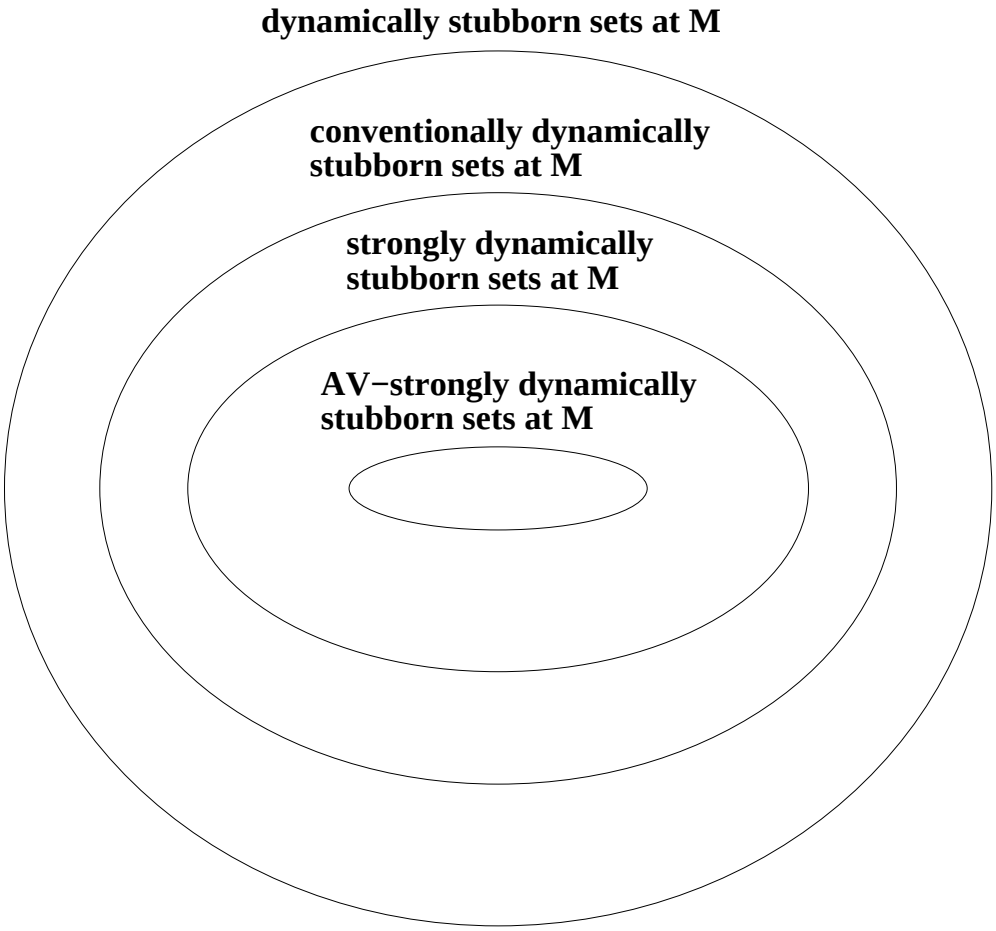


Figure 10: Four classes of dynamically stubborn sets at a marking.

To make the characterization of transition selection functions easier, Definition 4.17 extends the notion of dynamic stubbornness to concern functions.

Definition 4.17 Let $\langle S, T, W, M_0 \rangle$ be a place/transition net. Let f be a function from $\mathcal{M}$ to 2^T. Then we say that f is *dynamically stubborn* iff for each nonterminal marking M, $f(M)$ is dynamically stubborn. Correspondingly, f is *conventionally dynamically stubborn* iff for each nonterminal marking M, $f(M)$ is conventionally dynamically stubborn. Respectively, f is *unconventionally dynamically stubborn* iff f

is dynamically stubborn but not conventionally dynamically stubborn. Correspondingly, f is *strongly dynamically stubborn* iff for each nonterminal marking M, $f(M)$ is strongly dynamically stubborn. Finally, f is *AV-strongly dynamically stubborn* iff for each nonterminal marking M, $f(M)$ is AV-strongly dynamically stubborn. □

The *ample set method* [51, 55, 56, 57] is often presented using principles that are close to the principles of strong or at least conventional dynamic stubbornness. However, ampleness itself, according to the only explicit definition found [55], is not defined by means of such principles but instead by referring almost directly to what is to be preserved. Definition 4.18 has been obtained from the definition of ampleness in [55] by fixing the set of interesting paths to be the set of operation fair infinite or terminal paths. This fixing reflects the limitation that only interesting counterexamples are accepted in the verification of a formula.

Definition 4.18 Let $\langle S, T, W, M_0 \rangle$ be a place/transition net and M a reachable marking of the net. A set $T_s \subseteq T$ is *permutation-ample at* M iff for each operation fair infinite or terminal path starting from M in the full reachability graph, there exist $t \in T_s$, $\sigma \in T^* \cup T^\infty$ and $\delta \in T^* \cup T^\infty$ such that δ is the label of the path, $t\sigma$ is a permutation of δ and enabled at M, and moreover, the path starting from M and being labelled by $t\sigma$ is operation fair. Similarly, a set $T_s \subseteq T$ is *conditional-trace-ample* (respectively, *strict-trace-ample*) *at* M iff for each operation fair infinite or terminal path starting from M in the full reachability graph of $\langle S, T, W, M \rangle$, there exist $t \in T_s$, $\sigma \in T^* \cup T^\infty$ and $\delta \in T^* \cup T^\infty$ such that δ is the label of the path, $t\sigma$ is in the conditional (respectively, strict) trace of δ at M, and moreover, the path starting from M and being labelled by $t\sigma$ is operation fair. □

Clearly, strict-trace-ampleness implies conditional-trace-ampleness, which in turn implies permutation-ampleness. Note that even if the full reachability graph has infinite operation fair paths and we generate a reduced reachability graph by always firing the transitions in a chosen strict-trace-ample set, the reduced reachability graph does not necessarily have any infinite operation fair path. For example, if a net has two transitions but no place, any set consisting of a single transition is ample at the initial marking. If we fire only one transition at the initial marking, we get a reduced reachability graph where no other transition than the fired transition occurs. The reduced reachability graph has then no infinite operation fair path.

We now draw some connections between ampleness and dynamic stubbornness. Lemmas 4.19 and 4.20 should be interpreted with some care. More precisely, we should think of how practical the concept of ampleness would be if fairness assumptions were dropped. The three defined forms of ampleness can of course be redefined in such a way that all infinite paths are treated equally. If we then have a net consisting of nothing else but n transitions, all these transitions would have to be taken into an ample set whereas a dynamically stubborn set would not need more than one transition. A similar phenomenon would occur whenever a net would have some "sufficiently independent" parts.

Lemma 4.19 *If a set is dynamically stubborn at a reachable marking, the set is permutation-ample at the marking.*

Proof. Let δ be the label of an operation fair infinite or terminal path x starting from a marking M in the full reachability graph of a net. A terminal marking cannot have any key transition and thus not any dynamically stubborn set either. We can thus assume that M is not terminal. Let T_s be dynamically stubborn at M. Since x is operation fair and T_s has a dynamic key transition, there exist $t \in T_s$, $\sigma \in (T \setminus T_s)^*$ and $\rho \in T^* \cup T^\infty$ such that $\delta = \sigma t \rho$. Otherwise all dynamic key transitions of T_s would be enabled at all markings of the path x which would be a contradiction with the fact that x is either a terminal path or an operation fair infinite path. From D1 it directly follows that $t\sigma\rho$ is enabled at M. Since $t\sigma$ leads from M to the same marking as σt, the path starting from M and being labelled by $t\sigma\rho$ is operation fair. □

Lemma 4.20 *If a set is conventionally dynamically stubborn at a reachable marking, the set is conditional-trace-ample at the marking.*

Proof. Let δ be the label of an operation fair infinite or terminal path x starting from a marking M in the full reachability graph of a net. A terminal marking cannot have any key transition and thus not any conventionally dynamically stubborn set either. Like in the previous proof, we can thus assume that M is not terminal. Let T_s be conventionally dynamically stubborn at M. Since x is operation fair and T_s has a dynamic key transition, again, like in the previus proof, there exist $t \in T_s$, $\sigma \in (T \setminus T_s)^*$ and $\rho \in T^* \cup T^\infty$ such that $\delta = \sigma t \rho$. From CD it directly follows that $t\sigma\rho$ is in the conditional trace of $\sigma t\rho$ at M. For the same reason as above, the path starting from M and being labelled by $t\sigma\rho$ is operation fair. □

A set can be dynamically stubborn without being conditional-trace-ample. For example, the set $\{a, b\}$ is such a set at the initial marking in the net in Figure 6. This is so because the conditional trace of *cdebdf* at the initial marking does not contain any other sequence than the sequence *cdebdf* itself.

On the other hand, a set can quite well be strict-trace-ample without being dynamically stubborn or having any dynamically stubborn subset. Let us consider the net in Figure 11. The full reachability graph does not contain terminal paths, whereas all infinite paths in the graph are operation fair. The nonempty endless strict traces at the initial marking M_0 are $\{aceee\ldots, caeee\ldots\}$, $\{adeee\ldots, daeee\ldots\}$ and $\{bdeee\ldots, dbeee\ldots\}$ (because a is globally independent of c, d is globally independent of a whereas b is globally independent of d). Consequently, the set $\{a, d\}$ is strict-trace-ample at M_0. On the other hand, neither $\{a, d\}$ nor any of its subsets can be dynamically stubborn at M_0, because of the absence of a dynamic key transition.

Even if the condition of dynamic key transitions were relaxed, there would still be strict-trace-ample sets that are not dynamically stubborn and have no dynamically stubborn subset. Let us look at the net in Figure 12. Again, the full reachability graph does not contain terminal paths, and all infinite paths in the graph are operation fair. The nonempty endless strict traces at the initial marking M_0 are $\{acdeee\ldots, cadeee\ldots, cdaeee\ldots\}$, $\{adeee\ldots, daeee\ldots\}$ and $\{badeee\ldots, bdaeee\ldots, dbaeee\ldots\}$ (because a is globally independent of c, d is globally independent of a whereas b is globally independent of d). Consequently, the set $\{a, d\}$ is strict-trace-ample at M_0. However, neither $\{a, d\}$ nor any of its subsets can be dynamically stubborn at M_0 because either D1 would become violated or the set

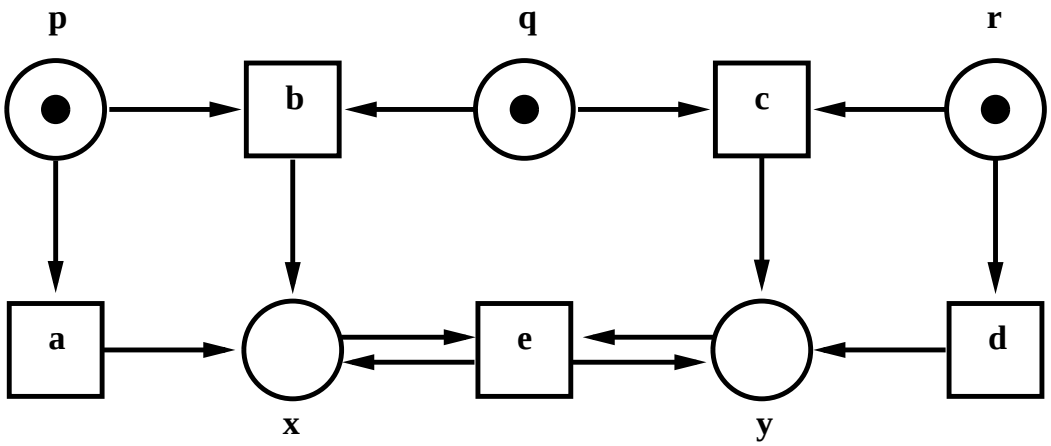


Figure 11: The set $\{a, d\}$ is strict-trace-ample but has no dynamic key transition.

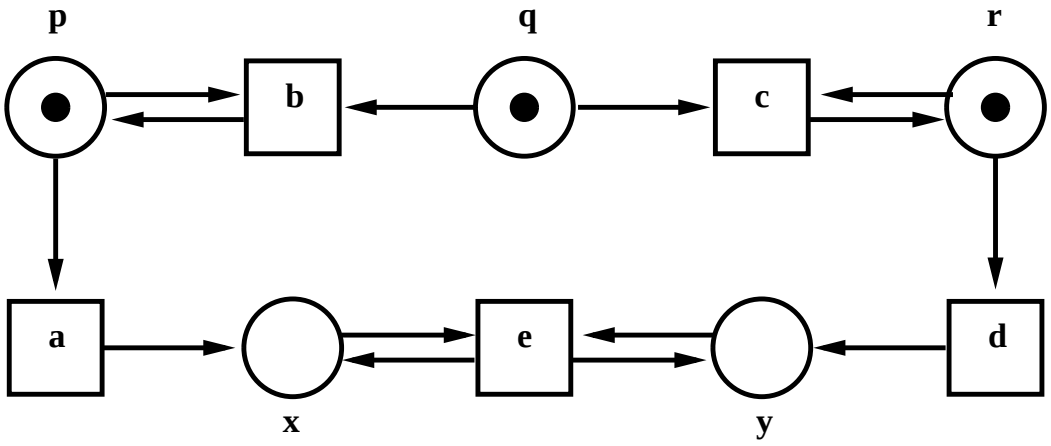


Figure 12: The set $\{a, d\}$ is strict-trace-ample but does not fulfil D1.

would be empty. To see this, consider the sequences ba and cd that are enabled at M_0.

4.2 Stubbornness

We define true stubbornness in place/transition nets. Since we are more interested in computing small stubborn sets in a moderate time than generating medium size or large stubborn sets fast, we have chosen a very weak definition of stubbornness.

Definition 4.21 Let $\langle S, T, W, M_0\rangle$ be a finite 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

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

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

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

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 flow to s than t has. Finally, $E_4(s)$ is the set of transitions that could decrease the contents of s.

Theorem 4.22 *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 [73] shows that our stubborn sets also fulfil D1. (We are not the first who observe this. Namely, [73] expresses the same thing below the proof in question.) $\square$

If we remove "$W(s,t) \leq W(t,s) \vee$" from Definition 4.21, we get the definition for the stubborn sets in [73]. Such a stubborn set is conventionally dynamically stubborn since by Lemma 2.5 in [73], every sequence of such transitions that are not in the set leaves the set stubborn, and we can then use our Theorem 4.22 and Lemma 4.9.

Let us return to some of the examples of Section 4.1. In the net in Figure 6, $\{a, b\}$ is stubborn but not conventionally dynamically stubborn, and $\{c\}$ is stubborn and strongly dynamically stubborn at the initial marking. In the net in Figure 7, $\{t_0, t_1\}$ and $\{t_1, t_2\}$ are stubborn and conventionally dynamically stubborn but not strongly dynamically stubborn at the initial marking. In the net in Figure 9, $\{c\}$ is stubborn and strongly dynamically stubborn but not AV-strongly stubborn at the initial marking.

As one might expect, a dynamically stubborn set is not necessarily stubborn. In the net in Figure 13, $\{a\}$ and $\{b\}$ are strongly dynamically stubborn but not stubborn at the initial marking.

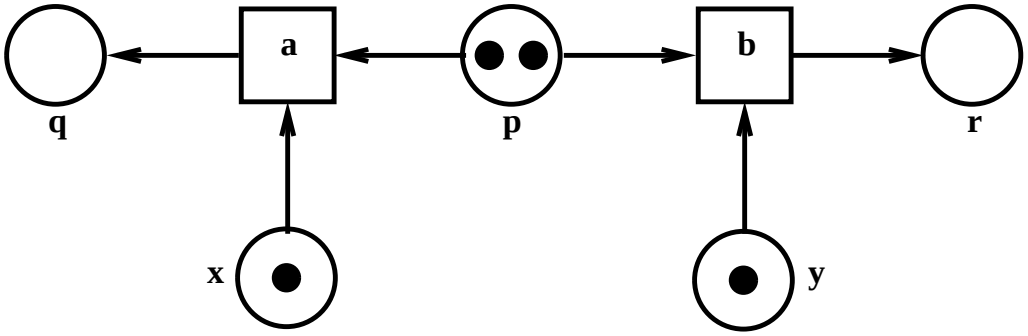


Figure 13: The set $\{a\}$ is strongly dynamically stubborn but not stubborn.

In [75], three basic algorithms for finding a suitable stubborn set are presented: the *candidate list algorithm*, the *incremental algorithm* and the *deletion algorithm*. Let us assume that T is the set of transitions and μ is the maximum number of input places of a transition. The candidate list algorithm selects the first stubborn set in a given candidate list $T_1, \ldots, T_n, T$. The time taken by an execution of the candidate list algorithm is at most proportional to $\mu \sum_{i=1}^{n} |T_i|$. The candidate list determines the size of the reduced reachability graph. We do not know good heuristics for automatic candidate list construction, so we concentrate on the incremental algorithm and the deletion algorithm that are automatic by nature. Both of these two algorithms contain nondeterministic choices, and there are cases where manual preliminary preparations can be useful.

4.3 The incremental algorithm

We now turn to the incremental algorithm [73, 75] for computing stubborn sets.

Definition 4.23 Let $\langle S, T, W, M_0 \rangle$ be a place/transition net. A place s is a *disabling place of a transition t at a marking M* iff $M(s) < W(s, t)$. A partial function f from $\mathcal{M} \times T$ to S is a *scapegoat generator* of the net iff for each marking M and each disabled transition t at M, $f(M, t)$ is a disabling place of t at M. Let G be the set of scapegoat generators of the net and B the set of functions from $\mathcal{M} \times T \times S$ to $\{2, 3\}$.

The function E_{14} from $G \times B \times \mathcal{M} \times T$ to 2^T is defined by

$$\forall f \in G\ \forall b \in B\ \forall M \in \mathcal{M}\ \forall t \in T$$
$$E_{14}(f,b,M,t) = \begin{cases} E_1(M, f(M,t)) & \text{if } \neg M[t\rangle, \\ \bigcup_{s \in \bullet t}(E_{b(M,t,s)}(M,t,s) \cup E_4(s)) & \text{if } M[t\rangle. \end{cases}$$

The function R_{14} from $G \times B \times \mathcal{M}$ to $2^{T \times T}$ is defined by

$$\forall f \in G\ \forall b \in B\ \forall M \in \mathcal{M}$$
$$R_{14}(f,b,M) = \{\langle t, t'\rangle \in T \times T \mid t' \in E_{14}(f,b,M,t)\}.$$

For each $f \in G$, $b \in B$, and $M \in \mathcal{M}$, the reflexive-transitive closure of $R_{14}(f,b,M)$ is denoted by $R^*_{14}(f,b,M)$). The function E^*_{14} from $G \times B \times \mathcal{M} \times T$ to 2^T is defined by

$$\forall f \in G\ \forall b \in B\ \forall M \in \mathcal{M}\ \forall t \in T$$
$$E^*_{14}(f,b,M,t) = \{t' \mid \langle t, t'\rangle \in R^*_{14}(f,b,M))\}.$$

For each $f \in G$, $b \in B$, and $M \in \mathcal{M}$, the $\langle f, b\rangle$-*dependency graph at* M is the pair $\langle V, A\rangle$ such that the set of vertices V is T, and the set of edges A is $R_{14}(f,b,M)$. □

Clearly, the set $E_{14}(f,b,M,t)$ is the set of transitions immediately succeeding t in the $\langle f, b\rangle$-dependency graph at M. Respectively, $E^*_{14}(f,b,M,t)$ is the set of transitions accessible from t in the $\langle f, b\rangle$-dependency graph at M. Note that Definition 4.21 implies $E_2(M,t,s) \cup E_4(s) = E_2(M,t,s)$.

Lemma 4.24 *Let $\langle S, T, W, M_0\rangle$ be a place/transition net. Let G be the set of scapegoat generators of the net and B the set of functions from $\mathcal{M} \times T \times S$ to $\{2,3\}$. Let $f \in G$, $b \in B$, $M \in \mathcal{M}$, $t \in T$, and $M[t\rangle$. Then $E^*_{14}(f,b,M,t)$ is both stubborn and strongly dynamically stubborn at M.*

Proof. Stubbornness is obvious. Theorem 4.22 then implies dynamic stubbornness. Strong dynamic stubbornness follows from dynamic stubbornness, from the definition of E_4 and from Lemma 4.4. □

The stubborn set in Lemma 4.24 is stubborn in the sense of the definition in [73], since the "$W(s,t) \leq W(t,s)\ \vee$" in Definition 4.21 is not utilized in Definition 4.23.

The incremental algorithm in [73] modified for our definitions can be described as follows. Let $\langle S, T, W, M_0\rangle$ be a finite place/transition net. Let G be the set of scapegoat generators of the net and B the set of functions from $\mathcal{M} \times T \times S$ to $\{2,3\}$. Let $f \in G$, $b \in B$, and $M \in \mathcal{M}$ be such that M is nonterminal. The algorithm produces a set T_s such that for some enabled transition τ at M, $T_s = E^*_{14}(f,b,M,\tau)$, and $\forall t \in T_s\ M[t\rangle \Rightarrow \tau \in E^*_{14}(f,b,M,t)$. The enabled transitions of T_s are in one maximal strongly connected component of the $\langle f, b\rangle$-dependency graph at M. The enabled transitions of T_s are found by traversing the $\langle f, b\rangle$-dependency graph in depth-first order, starting from an enabled transition, applying Tarjan's algorithm for computing maximal strongly connected components [67, 69], and stopping when the first maximal strongly connected component having an enabled transition is found. If b has the value 2 everywhere, our algorithm behaves as the algorithm in [73].

The time taken by an execution of this algorithm is at most proportional to $\mu\nu|T|$, where μ is the maximum number of input places of a transition, and ν is the maximum

number of adjacent transitions of a place. The number μ can be $|S|$, the number ν can be $|T|$, and $|S|$ can be far greater than $|T|$. We have made the practical assumption that the time taken by the computation of $f(M,t)$ is at most proportional to $\mu\nu$, and the time taken by the computation of $b(M,t,s)$ is at most proportional to ν. Even if we assumed that the computation of f and b takes no time, we would have the same time complexity for the incremental algorithm.

Without change in complexity, the incremental algorithm can be optimized [80, 95]: T_s is chosen to be such $E^*_{14}(f,b,M,\tau)$ that contains the least number of enabled transitions. (As above, τ has to be enabled at M.) All what is needed is to complete the depth-first search and application of Tarjan's algorithm in such a way that all enabled and only enabled transitions are checked in the outermost loop of the search. If the optimized incremental algorithm is used, the functions f and b are the only nondeterministic factors that affect the number of enabled transitions in T_s.

The reachability graph generation algorithm using the incremental algorithm produces a g-reachability graph of the net such that for each marking M in the graph, $g(M)$ is the T_s at M if there is an enabled transition at M. (If no transition is enabled at M, then any subset of T is valid for $g(M)$.)

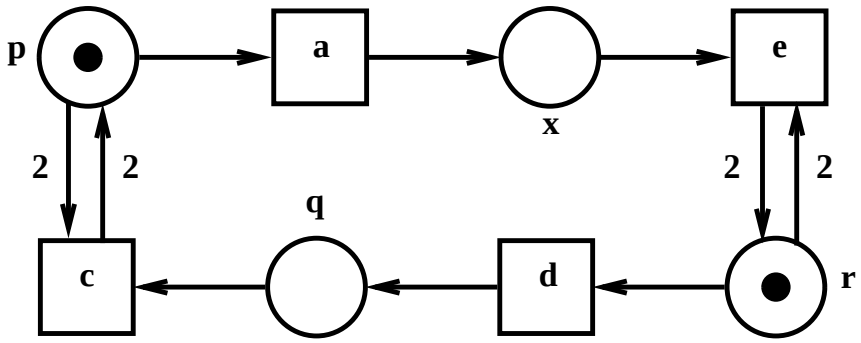


Figure 14: A net used for demonstrating the construction of a stubborn set.

Let us consider the construction of a stubborn set at the initial marking, M_0, of the net in Figure 14. We first observe that $E_1(M_0,p)=\emptyset$, $E_1(M_0,q)=\{d\}$, $E_1(M_0,r)=\emptyset$, $E_1(M_0,x)=\{a\}$, $E_4(p)=\{a\}$, $E_4(r)=\{d\}$, $E_2(M_0,a,p)=\{a,c\}$, $E_2(M_0,d,r)=\{d,e\}$, $E_3(M_0,a,p)=\emptyset$ and $E_3(M_0,d,r)=\emptyset$.

If $b(M_0,a,p)=b(M_0,d,r)=2$, $f(M_0,c)=q$ and $f(M_0,e)=x$. then the set of edges of the $\langle f,b,M_0\rangle$-dependency graph is $\{\langle a,a\rangle,\langle a,c\rangle,\langle c,d\rangle,\langle d,d\rangle,\langle d,e\rangle,\langle e,a\rangle\}$, so the dependency graph is strongly connected, and the set of all transitions thus becomes the computed stubborn set.

On the other hand, if $b(M_0,a,p)=b(M_0,d,r)=3$, then for any scapegoat generator f, $E^*_{14}(f,b,M_0,a)=\{a\}$ and $E^*_{14}(f,b,M_0,d)=\{d\}$, so either $\{a\}$ or $\{d\}$ becomes the computed stubborn set.

We conclude the example by considering the case where $b(M_0,a,p)=b(M_0,d,r)=2$, $f(M_0,c)=p$ and $f(M_0,e)=r$. In that case, the set of edges of the $\langle f,b,M_0\rangle$-dependency graph is $\{\langle a,a\rangle,\langle a,c\rangle,\langle d,d\rangle,\langle d,e\rangle\}$, so either $\{a,c\}$ or $\{d,e\}$ becomes the computed stubborn set. Since c and d are disabled, this choice is thus equally successful as the choice $b(M_0,a,p)=b(M_0,d,r)=3$.

The incremental algorithm sometimes produces stubborn sets that are not AV-strongly dynamically stubborn, even if the choice function b has the value 2 everywhere. Let us consider the net in Figure 9. If $b(M_0, c, p) = 2$, the computed stubborn set at M_0 is $\{c\}$, independently of the scapegoat generator. By the remark immediately after Lemma 4.24 we know that $\{c\}$ is stubborn also in the sense of the definition in [73]. We saw after Lemma 4.16 that the set $\{c\}$ is strongly dynamically stubborn but not AV-strongly dynamically stubborn at M_0 since $M_0[cc\rangle$, $M_0[cd\rangle$, and $\neg M_0[ccd\rangle$.

We end this section by describing what is meant by a pseudo-random scapegoat generator. An explicit description of a pseudo-random scapegoat generator would be far too complicated to be presented here. Therefore, only an informal description is given. Let r be a function from $(N \setminus \{0\}) \times (N \setminus \{0\}$ to $N \setminus \{0\}$ such that for each $n \in N \setminus \{0\}$, $r(n,1), r(n,2), r(n,3), \ldots$ is a sequence of *pseudo-random numbers* [67] in $\{1, \ldots, n\}$. (The ideal would be that the numbers in the sequence were uniformly distributed over $\{1, \ldots, n\}$.) Let f be the scapegoat generator to be defined. For each marking M and transition t, $f(M,t)$ is not defined earlier than necessary. If it is time for such definition, $f(M,t)$ is defined to be the $r(k,i)$th disabling place of t at M, where the list of disabling places of t at M is of length k and determined by some fixed list of the input places of t, whereas $i-1$ is the so far number of times when r has been "called with the first argument k".

4.4 The deletion algorithm

The stubborn sets computed by the incremental algorithm may contain unnecessarily many enabled transitions. The deletion algorithm [74, 75] is better in this sense.

Definition 4.25 Let $\langle S, T, W, M_0\rangle$ be a finite 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

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

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

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

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

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

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.

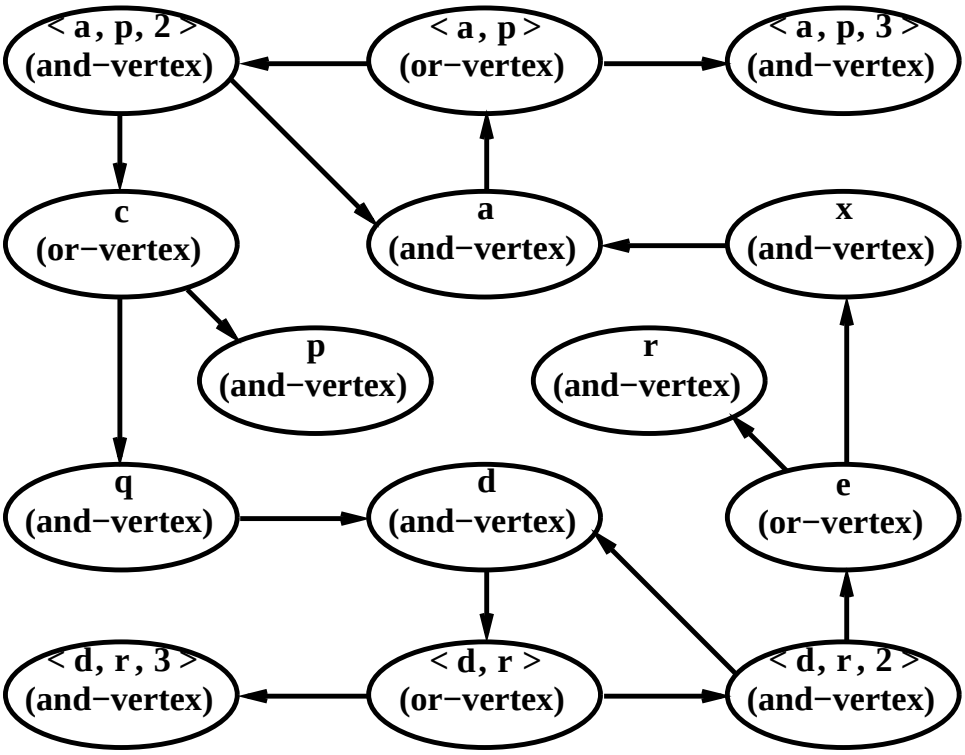


Figure 15: The and/or-graph of the net of Figure 14 at M_0.

Figure 15 displays the and/or-graph of the net of Figure 14 at the initial marking M_0. We observe that e.g. the set $\{a, \langle a, p\rangle, \langle a, p, 3\rangle\}$ is a legal set of vertices.

The pseudo-code language used in the algorithmic presentations in the sequel is a mixture of mathematical expressions, English expressions and the C programming language [46]. (Readers unfamiliar with the C language are strongly suggested to read, at least superficially, any of the several books written about the language.) Words belonging to the control structure of C are written in boldface. A construct of the form

"**for** (each x in A such that $\psi(x)$)",

where A is a set and $\psi(x)$ a truth-valued expression on x, is apparently against the syntax of the C language but corresponds to a valid "for-construct" where a cursor moves through a data structure and skips "uninteresting" elements. (This kind of moving and skipping can, regardless of the data structure, be implemented by means of function calls.) In order to support unambigous references to the code, we use comment-style line numbering throughout the presentations.

The basic deletion algorithm is presented in Figure 16. (We use the attribute "basic" here because some more sophisticated versions of the deletion algorithm are pre-

sented later in this thesis. The type **AO_vertex**, implementing each vertex of the and/or-graph, is assumed to have been defined appropriately.) The routine BasDelAlg 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 4.25 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 protection flag may seem useless but is needed because later algorithms in this thesis include calls to some of the routines in Figure 16.)

```
/*1*/   void Speculate(AO_vertex x) {
/*2*/       mark x grey;
/*3*/       for ( each white immediate predecessor vertex y of x )
/*4*/           if ( y is an and-vertex ) Speculate(y);
/*5*/           else {
/*6*/               subtract 1 from the counter of y;
/*7*/               if ( the counter of y is at 0 ) Speculate(y); } }
/*8*/   void Darken(AO_vertex x) {
/*9*/       mark x black;
/*10*/      for ( each grey immediate predecessor vertex y of x )
/*11*/          Darken(y); }
/*12*/  void Rehabilitate(AO_vertex x) {
/*13*/      mark x white;
/*14*/      for ( each non-black immediate predecessor vertex y of x ) {
/*15*/          if ( y is an or-vertex ) add 1 to the counter of y;
/*16*/          if ( y is grey ) Rehabilitate(y); } }
/*17*/  void Cnstr(void /* no argument */) {
/*18*/      for ( each protected enabled transition t )
/*19*/          set the root flag of t equal to 1;
/*20*/      while ( there are at least two enabled white transitions and
/*21*/              at least one enabled white transition has a zero root flag ) {
/*22*/          let t be some enabled white transition having a zero root flag;
/*23*/          set the root flag of t equal to 1; Speculate(t);
/*24*/          if ( the set of white transitions contains all protected transitions
/*25*/              and has a key transition) ) Darken(t);
/*26*/          else Rehabilitate(t); } }
/*27*/  void BasDelAlg(void /* no argument */) {
/*28*/      initialize the and/or-graph; make all transitions unprotected;
/*29*/      Cnstr(); }
```

Figure 16: The basic deletion algorithm.

The algorithm in Figure 16 can be characterized as follows: we first have a stubborn set that contains all enabled transitions. Then we try to remove each enabled transition in turn in order to obtain a smaller stubborn set. Any of such removals may force some other enabled transitions to be removed as well. A transition may be so "important" that no stubborn set can be obtained from the set of remaining transitions if the transition is removed. Therefore, we first speculate what would

happen if a certain transition were removed, and "rehabilitate" the transition if it cannot be removed successfully.

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 (lines /*20*/ and /*21*/) is checked.

The time taken by an execution of the basic 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 $\mu\nu|T|$.

Let us consider how the routine BasDelAlg constructs a stubborn set for the initial marking of the net in Figure 14. BasDelAlg first constructs the and/or-graph displayed in Figure 15 and initializes the records associated with the vertices. The routine Cnstr takes care of the rest. Since both of the enabled transitions, a and d, are initially white, the routine Speculate becomes called. Let us assume that $t = a$ when the line /*23*/ is entered for the first time. The routine Speculate then recursively goes from the and-vertex a to the and-vertex that has literally the name x. The counter of the or-vertex e is subtracted by one, but since the counter has the initial value 2, Speculate does not go to e. Then the program control quickly returns back to Cnstr, to the line /*24*/. The set of white transitions is $\{c, d, e\}$ at that moment. This set has the key transition d, so the grey vertices, a and x, become marked black. The "while-condition" is no longer true, so Cnstr finishes. The constructed stubborn set is thus $\{c, d, e\}$ which is effectively as good as $\{d\}$ since c and e are not enabled at the initial marking.

We now move onto a slightly higher abstraction level and consider how the basic deletion algorithm constructs a stubborn set for the initial marking of the net in Figure 7. All transitions are enabled, and the algorithm essentially just tries to remove them in some order. Let us assume that this order is t_0, t_1, t_2. Trying to remove t_0 is successful, and the removal is realized by marking t_0 black. Trying to remove t_1 is not succesful, i.e. t_1 must be rehabilated with the routine Rehabilitate. This rehabilitation restores the colours of the transitions to what they were before the removal speculation. Trying to remove t_2 is not succesful either. Clearly, the root flags are needed to make Cnstr stop. The constructed stubborn set is $\{t_1, t_2\}$. As we recall from the earlier considerations concerning this net, the set $\{t_1, t_2\}$ is not strongly dynamically stubborn at the initial marking. From Lemma 4.24 it thus follows that the incremental algorithm is unable to construct this set. Actually, $\{t_0, t_1, t_2\}$ is the only strongly dynamically stubborn set at the initial marking, so the incremental algorithm is somewhat naive in this case.

Considering the net in Figure 6, let us look at the marking M' such that $M_0[c\rangle M'$. At M', $\{a, b\}$ is the only inclusion minimal stubborn set and consists of enabled transitions only. For M', the basic deletion algorithm thus necessarily constructs a stubborn set that contains a and b but no other enabled transitions. Such a stubborn set is not even conventionally dynamically stubborn at M'.

4.5 Discussion

The most important result of this chapter is the rephrasing of persistence and conditional stubbornness in terms of strong dynamic stubbornness in place/transition nets. If a theorem is stated for persistent sets, we know that the theorem should not be applied to stubborn sets that are not strongly dynamically stubborn, unless the theorem is extended appropriately. On the other hand, any stubborn set computed by the incremental algorithm is strongly dynamically stubborn, so the set of enabled transitions in the computed set is persistent and any theorem concerning persistent sets applies to a reduced reachability graph constructed by using the incremental algorithm.

5 Verification of linear time temporal properties

This chapter considers the verification of nexttime-less LTL-formulas with the aid of the stubborn set method. A new preservation theorem is presented in Section 5.1, indicating how the structure of a formula can be utilized in verification when fairness is not assumed. The new algorithm for generating the reduced state space is presented in Section 5.2. Section 5.3 extends the theory of Section 5.1 to concern verification under the assumption of operation fairness.

5.1 A preservation theorem

Let us call a formula *directly temporal* iff the outermost operator of the formula is a non-propositional operator. A nexttime-less LTL-formula can be transformed into a nexttime-less LTL-formula where directly temporal subformulas are as short as possible [55]. Then a suitable reduced reachability graph can be generated by using the stubborn set method, provided that the conditions in Lemma 5.1 and Proposition 5.2 are satisfied. Note that any formula can be seen as a Boolean combination of directly temporal subformulas. The $\Box(\bigcirc(\top))$ formula occurring in Proposition 5.2 is satisfied by every infinite path whereas no terminal path satisfies it. Lemma 5.1 is a variant of a very well known result [56].

Lemma 5.1 *Assumptions:*

(P1) $\langle S, T, W, M_0 \rangle$ *is a place/transition net. (The net and the full reachability graph of the net can be finite or infinite.)*

(P2) $\mathcal{P}$ *is a (finite or an infinite) collection of atomic formulas, i.e.* $\mathcal{P} \subseteq 2^{\mathcal{M}}$. Γ *is the collection of all nexttime-less LTL-formulas that are constructible from the formulas of* $\mathcal{P}$.

(P3) Π *is a function from* $2^{\mathcal{M}}$ *to* 2^S *in such a way that whenever we have a subset* p *of* $\mathcal{M}$ *and markings* M *and* M' *for which* $M \in p$ *and* $M' \notin p$, *there exists* $s \in \Pi(p)$ *for which* $M(s) \neq M'(s)$.

(P4) Ξ *is a function from* Γ *to* 2^T *in such a way that for each* $\phi \in \Gamma$ *and for each atomic subformula* p *of* ϕ, $\{t \in T \mid \exists s \in \Pi(p)\ W(s,t) \neq W(t,s)\} \subseteq \Xi(\phi)$.

Claim: For each $\phi \in \Gamma$, *for each reachable marking* M, *and for each two paths* x *and* y *starting from* M *in the full reachability graph of the net, if* $\Re(x, \Xi(\phi)) = \Re(y, \Xi(\phi))$ *and* x *satisfies* ϕ, *then* y *satisfies* ϕ.

Proof.

Let $\phi \in \Gamma$ and M a reachable marking, and let paths x_0 and x_1 start from M in the full reachability graph in such a way that x_0 and x_1 have the same $\Xi(\phi)$-restriction and x_0 satisfies ϕ.

Let Θ be a function from $\mathcal{M}$ to $2^{\mathcal{P}}$ in such a way that for each marking M, $\Theta(M)$ contains exactly those atomic subformulas of ϕ that contain M. Let (finite or infinite) nonempty words α_0, α_1, β_0 and β_1 over the alphabet $2^{\mathcal{P}}$ be defined as follows.

- For each $i \in \{0,1\}$ and for each $n \in N \setminus \{0\}$, whenever x_i has at least n vertices, the word α_i has at least n characters, and the nth character in α_i is $\Theta(M')$ where M' is the nth vertex (all occurrences counted) in x.
- For each $i \in \{0,1\}$, β_i is the word obtained by replacing each maximal (finite or infinite) sequence of adjacent identical characters in α_i with a single occurrence of the character. (No other manipulation of α_i is thus included. If the set of distinct characters in some infinite suffix of α_i is a singleton set, then β_i is a finite word that has the member of that singleton set as the last character.)

From P3 and P4 it follows that $\beta_0 = \beta_1$. (Using the terminology in [48, 56], we could thus say that α_0 and α_1 are *propositional sequences* that are *equivalent up to stuttering.*) By looking at the satisfaction rules of our LTL, we conclude that y satisfies ϕ.

□

Proposition 5.2 *Assumptions: P1, P2, P3 and P4 of Lemma 5.1, and*

(P5) Φ *is a (proper or non-proper) subcollection of formulas from* Γ. *(*Φ *can be finite or infinite.)*

(P6) Υ *is a (finite or an infinite) subset of* 2^T *such that* $\{\Xi(\phi) \mid \phi \in \Phi\} \subseteq \Upsilon$.

(P7) f *is a function from* $\mathcal{M}$ *to* 2^T *in such a way that every terminal path in the* f*-reachability graph of the net is a terminal path of the full reachability graph of the net. (The* f*-reachability graph of the net can be finite or infinite.)*

(P8) For each terminal path starting from M_0 *in the full reachability graph, there exists a terminal path starting from* M_0 *in the* f*-reachability graph in such a way that the labels of the paths are* Υ*-equivalent.*

(P9) For each infinite path starting from M_0 *in the full reachability graph, there exists an infinite path starting from* M_0 *in the* f*-reachability graph in such a way that the labels of the paths are* Υ*-equivalent.*

Claim: For any boolean combination ϕ *of the formulas in* $\Phi \cup \{\Box(\bigcirc(\top))\}$, ϕ *is valid at* M_0 *in the full reachability graph of the net iff* ϕ *is valid at* M_0 *in the* f*-reachability graph of the net.*

Proof. The "only if" -part of the claim is obvious. The "if" -part follows directly from Lemma 5.1 and from the satisfaction rules. □

There is actually nothing new or amazing in Proposition 5.2, and its only purpose is to serve as an interface to Theorem 5.10, i.e. instead of talking about formulas we can talk about Υ-equivalence. Claims of Theorem 5.10 occur as assumptions in Proposition 5.2.

Theorem 5.10, the goal of this section, is a refinement of Theorem 2 of [78] and gives us better chances for reduction. The refinement is strongly inspired by [55, 56]. The new aspect in Theorem 5.10 is that we do not preserve all orders of *visible*

transitions. A transition is visible iff at least one member of the above defined Υ contains the transition. Roughly speaking, visible transitions are those transitions that determine the satisfaction of the atomic subformulas of the interesting formulas. In a verification task, if the original formula to be verified is ϕ_0 and an equivalent formula obtained by transformation is ϕ_1, then the collection of interesting formulas consists of directly temporal formulas such that ϕ_1 is a Boolean combination of the formulas in the collection. (If ϕ_1 itself is directly temporal, then the collection is simply $\{\phi_1\}$.)

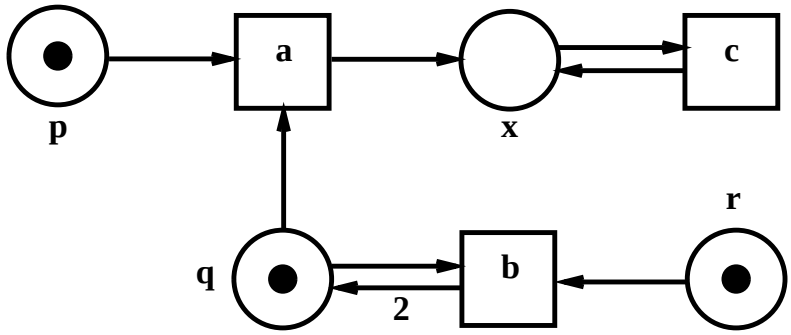


Figure 17: Both of a and b are visible w.r.t. the atomic formula "$M(q) = 0$".

Let us look at the net in Figure 17. Clearly, the full reachability graph of the net has no terminal path but has exactly two infinite paths that start from the initial marking. Among these two paths, the path labelled by $accc\ldots$ satisfies the formula $\Diamond(M(q) = 0)$ while the path labelled by $baccc\ldots$ does not. However, $\mathcal{M}$ has no markings M_1 and M_2 for which it would be that $M_1[b\rangle M_2$ and either $M_1(q) = 0 \neq M_2(q)$ or $M_2(q) = 0 \neq M_1(q)$. We still consider b as a visible transition w.r.t. the atomic formula "$M(q) = 0$", since in the sequel, transitions like this would anyway be treated like the "pedantically visible" transitions.

Since we do not preserve all orders of visible transitions, we can lose some *NDFD*- or *CFFD*-properties [42, 43, 80, 82]. Taking into account the title of [43], this may sound dangerous since the weakest equivalence meant by the title is NDFD. However, the title refers to preserving the validities/invalidities of all nexttime-less LTL-formulas constructible from a given set of atomic formulas. In the above verification task, we do not have to preserve that much. It suffices that the validities/invalidities of all Boolean combinations of the interesting formulas are preserved (since the original formula to be verified corresponds to one of such combinations).

The assumptions of Theorem 5.10 are the following.

(A1) $\langle S, T, W, M_0\rangle$ is a place/transition net, $\Upsilon \subseteq 2^T$, and $J = T \setminus \cup_{Y\in\Upsilon} Y$. (The net, Υ and the full reachability graph of the net can be finite or infinite.)

(A2) f is a dynamically stubborn function from $\mathcal{M}$ to 2^T. (The f-reachability graph of the net can be finite or infinite.)

(A3) For any $Y \in \Upsilon$ and for any marking M, $Y \subseteq f(M)$ or $\{t \in Y \cap f(M) \mid M[t\rangle\} = \emptyset$ (or both).

(A4) For any marking M, if $f(M)$ does not contain all those transitions that are enabled at M, then some transition in J is a dynamic key transition of $f(M)$ at M.

(A5) For any $t \in T \setminus J$, every infinite path (starting from a marking whatsoever) in the f-reachability graph of the net contains at least one marking M such that $t \in f(M)$.

Coarsely speaking, A3 prevents us from changing the order of transitions that are visible w.r.t. a single member of Φ while A4 and A5 prevent us from ignoring any member of Φ. The transitions in J are invisible w.r.t. all members of Φ. There is a following correspondence between A3 – A5 and the assumptions 2 – 4 of Theorem 2 of [78]: if $|\Upsilon| = 1$, n is between 3 and 5 and the f-reachability graph is finite, An becomes assumption $n - 1$ of [78].

Let us consider an example where we try to verify the formula

$$(\Diamond(M(q) = 1)) \vee (\Diamond(M(r) = 0))$$

about the net in Figure 3. We can let $\Upsilon = \{\{a\}, \{c\}\}$ ($|\Upsilon| = 2$) since the satisfaction of $M(q) = 1$ can be affected by a only whereas the satisfaction of $M(r) = 0$ can be affected by c only. Let us choose $\{a, b\}$ for the dynamically stubborn set at the initial marking. This choice respects all of A1 – A5. (Note that [78] would not accept such a choice but would require us to take all enabled transitions into the set. We have thus gained reduction w.r.t. [78].) At any other encountered nonterminal marking, we let the dynamically stubborn set contain all enabled transitions since A1 – A5 would otherwise be violated. (The same would have to be done if the conditions in [78] would have to be satisfied instead.) The reduced reachability graph has exactly one terminal path that starts from the initial marking, and the label of that path is ac. The labels of the infinite paths starting from the initial marking in the reduced reachability graph are $bddd\ldots$, $bcddd\ldots$, $bdcddd\ldots$, $bddcddd\ldots$, etc. From these paths the path labelled by $bddd\ldots$ invalidates the formula.

Let us then verify the formula

$$(\Diamond(M(x) = 1)) \vee (\Diamond(M(r) = 0)).$$

Using similar reasoning as above, we can let $\Upsilon = \{\{b\}, \{c\}\}$ ($|\Upsilon| = 2$). (Though d is connected to x, d cannot affect the satisfaction of $M(x) = 1$.) Proceeding as above, we actually get exactly the same reduced reachability graph, but that is merely a coincidence. Since there is no counterexample to the formula, we conclude that the formula is valid at the initial marking.

Let us also look what would be the consequences if some of A3 – A5 were dropped. Dropping A3 could make us draw a wrong conclusion about

$$\Box(((M(r) = 1) \vee (M(q) = 1)) \vee (\Box(M(q) = 0))).$$

When $\Upsilon = \{\{a, c\}\}$ ($|\Upsilon| = 1$), we could choose $\{a, b\}$ for the dynamically stubborn set at the initial marking. The only counterexample to the formula, i.e. the path starting from the initial marking and being labelled by ca, would then be lost.

Dropping A4 could make us draw a wrong conclusion about $\Diamond(M(r) = 0)$. When $\Upsilon = \{\{c\}\}$, we could choose $\{c\}$ for the dynamically stubborn set at the initial marking. The only counterexample to the formula, i.e. the path starting from the initial marking and being labelled by $bddd\ldots$, would then be lost.

Dropping A5 could make us draw a wrong conclusion about

$$\Box(((M(x) = 0) \vee (M(r) = 0)) \vee (\Box(M(r) = 1))).$$

When $\Upsilon = \{\{b, c\}\}$ ($|\Upsilon| = 1$), we could let $\{a, b, c\}$ be the dynamically stubborn set at the initial marking and choose $\{d\}$ to be the dynamically stubborn set at the marking to which b leads from the initial marking. All the counterexamples to the formula, i.e. the paths where c occurs after b, would then be lost.

In the net in Figure 18, omitting the attribute "dynamic key" in A4 could make us draw a wrong conclusion about

$$(\Diamond(M(q) = 1)) \vee (\Diamond(M(y) = 1)).$$

When $\Upsilon = \{\{a, d\}\}$ ($|\Upsilon| = 1$), we could choose $\{a, b, d\}$ for the dynamically stubborn set at the initial marking. The only counterexample to the formula, i.e. the path starting from the initial marking and being labelled by $ceee\ldots$, would then be lost.

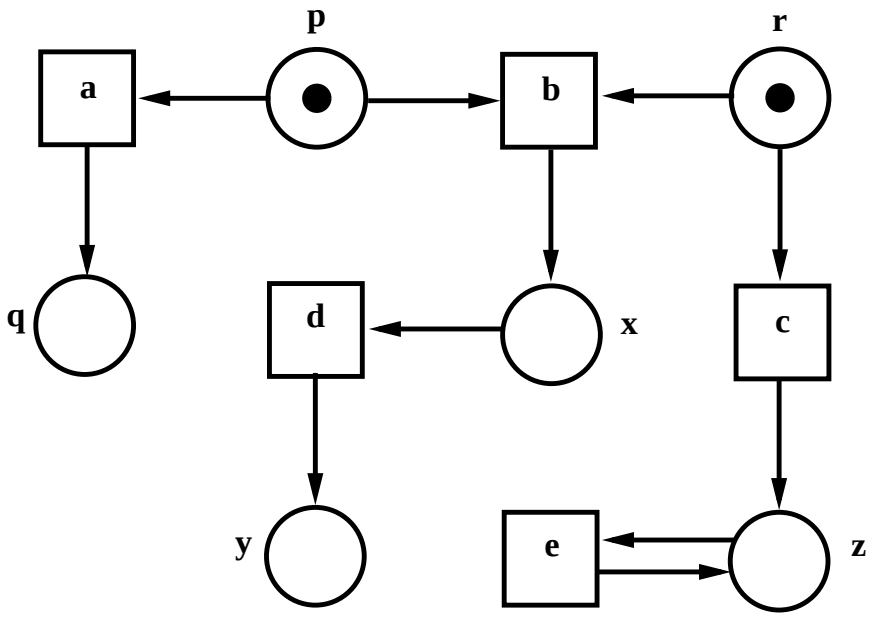


Figure 18: A net that motivates the assumption A4.

We now start working towards Theorem 5.10. Lemma 5.3 tells us that the used transition selection function respects the important orderings of transitions.

Lemma 5.3 *Assumptions: A1, A2 and A3.*

Claim: For each nonterminal marking M, for each t in $f(M)$ and for each σ in $(T \setminus f(M))^*$, if $M[\sigma t\rangle$, then $M[t\sigma\rangle$, and $t\sigma$ is Υ-equivalent to σt.

Proof. Let M be a nonterminal marking, $t \in f(M)$ and $\sigma \in (T \setminus f(M))^*$ in such a way that $M[\sigma t\rangle$. From D1 (and, as goes without saying, from A2) it follows that $M[t\sigma\rangle$. Let $Y \in \Upsilon$. If $Y \subseteq f(M)$, then $\sigma \in (T \setminus Y)^*$ and thus $\Re(t\sigma, Y) = t = \Re(\sigma t, Y)$. If $Y \not\subseteq f(M)$, then A3 has the effect that $t \notin Y$, so $\Re(t\sigma, Y) = \Re(\sigma, Y) = \Re(\sigma t, Y)$. □

Lemma 5.4 guarantees that the possible terminal paths of the full reachability graph are sufficiently represented in the reduced reachability graph.

Lemma 5.4 *Assumptions: A1, A2 and A3.*

Claim: For each finite transition sequence σ'' and for each marking M'', if σ'' leads from M'' to a terminal marking M_d, then there exists a permutation δ'' of σ'' in such a way that $M''[\delta''\rangle_f M_d$ and δ'' is Υ-equivalent to σ''.

Proof. We use induction on the length of σ''. The claim holds trivially when restricted to $\sigma'' = \varepsilon$. Our induction hypothesis is that the claim holds when restricted to any σ'' of length $n \geq 0$.

Let a finite transition sequence σ of length $n+1$ lead from a marking M to a terminal marking M_d. From D2 it follows that there exist $t \in f(M)$, $\delta \in (T \setminus f(M))^*$ and $\delta' \in T^*$ in such a way that $\sigma = \delta t \delta'$. From Lemma 5.3 it follows that there exists a marking M' in such a way that $M[t\rangle_f M'$, $M'[\delta\delta'\rangle M_d$, and $t\delta\delta'$ is Υ-equivalent to σ.

By the induction hypothesis, there exists a permutation δ'' of $\delta\delta'$ in such a way that $M'[\delta''\rangle_f M_d$ and δ'' is Υ-equivalent to $\delta\delta'$. So, $t\delta''$ is a permutation of σ in such a way that $M[t\delta''\rangle_f M_d$ and $t\delta''$ is Υ-equivalent to σ. □

Lemma 5.5 guarantees that the possible infinite invisible transition sequences are sufficiently represented in the reduced reachability graph.

Lemma 5.5 *Assumptions: A1, A2 and A4.*

Claim: For each $\sigma'' \in J^\infty$ and for each $M'' \in \mathcal{M}$, if $M''[\sigma''\rangle$ then there exists $\delta'' \in J^\infty$ in such a way that $M''[\delta''\rangle_f$.

Proof. Let $\sigma \in J^\infty$ and $M \in \mathcal{M}$ such that $M[\sigma\rangle$. We show by induction on $n \in N$ that we can define a function τ from N to J, a function μ from N to $\mathcal{M}$ and a function θ from N to J^∞ in such a way that $\mu(0) = M$ and for each $n \in N$, $\mu(n)[\theta(n)\rangle$, and either $n > 0$ and $\mu(n-1)[\tau(n-1)\rangle_f \mu(n)$ or $n = 0$. Let us denote this claim by B. (If B holds, the function τ represents an infinite transition sequence that is f-enabled at M.)

By letting $\mu(0) = M$ and $\theta(0) = \sigma$, we make B hold when restricted to $n = 0$. Let then $k \in N$. Our induction hypothesis is that B holds when restricted to $n = k$.

- Let us first consider the case that $\theta(k)$ contains a transition from $f(\mu(k))$. By the induction hypothesis, $\theta(k) \in J^\infty$. Let $t_k \in f(\mu(k)) \cap J$, $\gamma_k \in (J \setminus f(\mu(k)))^*$ and $\zeta_k \in J^\infty$ be such that $\gamma_k t_k \zeta_k = \theta(k)$. Since (because of the induction hypothesis) $\mu(k)[\gamma_k t_k \zeta_k\rangle$, from D1 it follows that $\mu(k)[t_k \gamma_k \zeta_k\rangle$. With all this information, we let $\tau(k) = t_k$, $\delta(k+1) = \delta(k) t_k$, $\theta(k+1) = \gamma_k \zeta_k$ and require that $\mu(k)[t_k\rangle \mu(k+1)$.

- Let us then consider the case that $\theta(k)$ does not contain any transition from $f(\mu(k))$. Since (because of the induction hypothesis) $f(\mu(k))$ does not contain all those transitions that are enabled at $f(\mu(k))$, A4 has the effect that we can choose $t_k \in f(\mu(k)) \cap J$ in such a way that t_k is a dynamic key transition of $f(\mu(k))$ at $\mu(k)$. From Lemma 4.3 it then follows that $\mu(k)[t_k\theta(k)\rangle$. With all this information, we let $\tau(k) = t_k$, $\delta(k+1) = \delta(k)t_k$, $\theta(k+1) = \theta(k)$ and require that $\mu(k)[t_k\rangle\mu(k+1)$. □

Lemma 5.6 shows that for a given enabled (finite or infinite) transition sequence $\sigma\rho$, on some infinite or terminal path in the reduced reachability graph, "the Υ-equivalence class of $\sigma\rho$ is enabled either forever or until some transition from σ becomes chosen." (The "forever" alternative may be the only alternative since A5 is not included in the assumptions of Lemma 5.6. To see this, look back to the "Dropping A5" example.)

Lemma 5.6 *Assumptions:*

- *A1, A2, A3 and A4.*
- $M \in \mathcal{M}$, $\sigma \in T^*$, $\rho \in T^* \cup T^\infty$ *and* $L \subseteq T$ *are such that* $M[\sigma\rho\rangle$ *and* L *is the set of those transitions that occur in* σ.

Claim: We can define a function β *from* N *to* $(T \setminus L)^*$, *functions* ξ *and* η *from* N *to* T^*, *a function* μ *from* N *to* $\mathcal{M}$ *and a function* θ *from* N *to* $T^* \cup T^\infty$ *in such a way that for each* $n \in N$,

- $\xi(n)\theta(n) = \rho$,
- $M[\beta(n)\rangle_f \mu(n)$,
- $\mu(n)[\sigma\eta(n)\theta(n)\rangle$,
- $\beta(n)\sigma\eta(n)$ *is* Υ*-equivalent to* $\sigma\xi(n)$, *and*
- $$\begin{aligned} & n = 0 \vee \\ & (n > 0 \wedge \ (\forall m \in N\ \forall t \in L \ \ \mu(m)[t\rangle_f \Rightarrow m \geq n) \wedge \\ & \qquad\qquad (\exists t_n \in T \setminus L \ \ \beta(n) = \beta(n-1)t_n)) \vee \\ & (n > 0 \wedge (\exists t_n \in L \ \ \mu(n-1)[t_n\rangle_f) \wedge \beta(n) = \beta(n-1)). \end{aligned}$$

Proof. We use induction on n. By letting $\beta(0) = \varepsilon$, $\xi(0) = \varepsilon$, $\eta(0) = \varepsilon$, $\mu(0) = M$ and $\theta(0) = \rho$, we make the claim hold when restricted to $n = 0$. Let then $k \in N$. Our induction hypothesis is that the claim holds when restricted to $n = k$.

- If there exists $\tau \in L$ such that $\mu(k)[\tau\rangle_f$, we let $\beta(k+1) = \beta(k)$, $\xi(k+1) = \xi(k)$, $\eta(k+1) = \eta(k)$, $\mu(k+1) = \mu(k)$ and $\theta(k+1) = \theta(k)$. The induction step is thus trivial in this case.

In the below considerations, no transition of L is f-enabled at $\mu(k)$. Using the induction hypothesis, we then immediately conclude that for each $m < k+1$, no transition of L is f-enabled at $\mu(m)$. Since we also know (by the induction hypothesis) that σ is enabled at $\mu(k)$, Lemma 4.2 has the effect that $\sigma \in (T \setminus f(\mu(k)))^*$.

- Let us consider the case where $\eta(k)$ contains a transition from $f(\mu(k))$. Then we can choose $\tau_k \in f(\mu(k))$, $\gamma_k \in (T \setminus f(\mu(k)))^*$ and $\zeta_k \in T^*$ in such a way that $\gamma_k\tau_k\zeta_k = \eta(k)$. From the induction hypothesis it follows that there exists a marking M such that $\mu(k)[\sigma\gamma_k\tau_k\rangle M$. Since $\sigma\gamma_k \in (T \setminus f(\mu(k)))^*$, Lemma 5.3 has the effect that $\mu(k)[\tau_k\sigma\gamma_k\rangle M$, and $\tau_k\sigma\gamma_k$ is Υ-equivalent to $\sigma\gamma_k\tau_k$. So, $\beta(k)\tau_k\sigma\gamma_k\zeta_k$ is Υ-equivalent to $\beta(k)\sigma\gamma_k\tau_k\zeta_k = \beta(k)\sigma\eta(k)$ which by the induction hypothesis is Υ-equivalent to $\sigma\xi(k)$. The induction step thus succeeds by letting $\beta(k+1) = \beta(k)\tau_k$, $\xi(k+1) = \xi(k)$, $\eta(k+1) = \gamma_k\zeta_k$ and $\theta(k+1) = \theta(k)$, and by requiring that $\mu(k)[\tau_k\rangle\mu(k+1)$.

- Let us then consider the case that $\theta(k)$ contains a transition from $f(\mu(k))$ but $\eta(k)$ does not. Then we can choose $\tau_k \in f(\mu(k))$, $\gamma_k \in (T \setminus f(\mu(k)))^*$ and $\zeta_k \in T^* \cup T^\infty$ in such a way that $\gamma_k\tau_k\zeta_k = \theta(k)$. From the induction hypothesis it follows that there exists a marking M such that $\mu(k)[\sigma\eta(k)\gamma_k\tau_k\rangle M$. Since $\sigma\eta(k)\gamma_k \in (T\setminus f(\mu(k)))^*$, Lemma 5.3 has the effect that $\mu(k)[\tau_k\sigma\eta(k)\gamma_k\rangle M$, and $\tau_k\sigma\eta(k)\gamma_k$ is Υ-equivalent to $\sigma\eta(k)\gamma_k\tau_k$. So, $\beta(k)\tau_k\sigma\eta(k)\gamma_k$ is Υ-equivalent to $\beta(k)\sigma\eta(k)\gamma_k\tau_k$ which by the induction hypothesis is Υ-equivalent to $\sigma\xi(k)\gamma_k\tau_k$. The induction step thus succeeds by letting $\beta(k+1) = \beta(k)\tau_k$, $\xi(k+1) = \xi(k)\gamma_k\tau_k$, $\eta(k+1) = \eta(k)\gamma_k$ and $\theta(k+1) = \zeta_k$, and by requiring that $\mu(k)[\tau_k\rangle\mu(k+1)$

- In the ultimate remaining case, $\sigma\eta(k)\theta(k) \in (T \setminus f(\mu(k)))^* \cup (T \setminus f(\mu(k)))^\infty$. Since $f(\mu(k))$ does not contain all those transitions that are enabled at $f(\mu(k))$, A4 has the effect that we can choose $\tau_k \in f(\mu(k)) \cap J$ in such a way that τ_k is a dynamic key transition of $f(\mu(k))$ at $\mu(k)$. From Lemma 4.3 it then follows that $\mu(k)[\tau_k\sigma\eta(k)\theta(k)\rangle$. Since $\tau_k \in J$, $\beta(k)\tau_k\sigma\eta(k)$ is Υ-equivalent to $\beta(k)\sigma\eta(k)$ which by the induction hypothesis is Υ-equivalent to $\sigma\xi(k)$. The induction step thus succeeds by letting $\beta(k+1) = \beta(k)\tau_k$, $\xi(k+1) = \xi(k)$, $\eta(k+1) = \eta(k)$ and $\theta(k+1) = \theta(k)$, and by requiring that $\mu(k)[\tau_k\rangle\mu(k+1)$ □

Lemma 5.7 states that if we have an infinite or a finite sequence in the full reachability graph, we can choose an arbitrary finite prefix of the sequence in such a way that there is a sequence that is Υ-equivalent to the original sequence and has a finite prefix that is f-enabled and covers the prefix we chose. (To remember the meaning of "exhausting", see Definition 2.4.)

Lemma 5.7 *Assumptions: A1, A2, A3, A4 and A5.*

Claim: For each $\sigma'' \in T^*$, for each $\rho' \in T^* \cup T^\infty$ and for each $M' \in \mathcal{M}$, if $M'[\sigma''\rho'\rangle$ and M' is f-reachable from M_0, then there exist $\gamma'' \in T^*$, $\delta_1 \in T^*$, $\delta_2 \in T^*$, $\rho'' \in T^* \cup T^\infty$ and $M'' \in \mathcal{M}$ in such a way that $\gamma''\rho'' = \rho'$, $M'[\delta_1\rangle_f M''$, $M''[\delta_2\rho''\rangle$, δ_1 $(T \setminus J)$-exhausts σ'', and $\delta_1\delta_2$ is Υ-equivalent to $\sigma''\gamma''$.

Proof. We use induction on the length of σ''. The claim holds trivially when restricted to $\sigma'' = \varepsilon$. Our induction hypothesis is that the claim holds when restricted to any σ'' of length $n \geq 0$. The claim holds trivially when restricted to any $\sigma'' \in J^*$ since in that case, ε is Υ-equivalent to σ'' and $(T \setminus J)$-exhausts σ'', so $\gamma'' = \delta_1 = \delta_2 = \varepsilon$, $\rho'' = \rho'$ and $M'' = M'$ are suitable choices for that case. Let then $\sigma \in T^* \setminus J^*$, $\rho \in T^* \cup T^\infty$ and $M \in \mathcal{M}$ be such that $M[\sigma\rho\rangle$, M is f-reachable from M_0 and the length of σ is $n+1$.

Let L be the set of those transitions that occur in σ. Then we can define a function β from N to $(T \setminus L)^*$, functions ξ and η from N to T^*, a function μ from N to $\mathcal{M}$ and a function θ from N to $T^* \cup T^\infty$ in the way stated in the claim of Lemma 5.6. We thus have that for each $k \in N$, $\xi(k)\theta(k) = \rho$, $M[\beta(k)\rangle_f \mu(k)$, $\mu(k)[\sigma\eta(k)\theta(k)\rangle$, and $\beta(k)\sigma\eta(k)$ is Υ-equivalent to $\sigma\xi(k)$.

Let us first assume that there are no $k' \in N$ and $\tau' \in L$ that would satisfy $\mu(k')[\tau'\rangle_f$. Let us call this assumption B. Since $\sigma \in T^* \setminus J^*$ and L is the set of those transitions that occur in σ, the set $L \setminus J$ is not empty. Let t be any transition in $L \setminus J$. From B and A5 it follows that there exists $k'' \in N$ such that $t \in f(\mu(k''))$. Consequently, there must be some $k_1 \leq k''$, $t' \in L \cap f(\mu(k_1))$, $\gamma \in (L \setminus f(\mu(k_1)))^*$ and $\gamma' \in L^*$ such that $\sigma = \gamma t' \gamma'$. Since $\mu(k_1)[\sigma\rangle$, from D1 it follows that $\mu(k_1)[t'\rangle_f$. We have thus reached a contradiction with B.

So, we can choose $k' \in N$ and $\tau' \in L$ such that $\mu(k')[\tau'\rangle_f$. Since $\mu(k')[\sigma\rangle$, there are some $t_1 \in f(\mu(k'))$, $\delta \in (T \setminus f(\mu(k')))^*$ and $\delta' \in T^*$ such that $\sigma = \delta t_1 \delta'$. Since $\mu(k')[\sigma\eta(k')\theta(k')\rangle$, from Lemma 5.3 it follows that there exists a marking M_1 such that $\mu(k')[t_1\rangle_f M_1$, $M_1[\delta\delta'\eta(k')\theta(k')\rangle$, and $t_1\delta\delta'$ is Υ-equivalent to σ. So, $\beta(k')t_1\delta\delta'\eta(k')$ is Υ-equivalent to $\sigma\xi(k')$ since $\beta(k')\sigma\eta(k')$ is Υ-equivalent to $\sigma\xi(k')$.

By the induction hypothesis, there exists $\gamma'' \in T^*$, $\delta_1 \in T^*$, $\delta_2 \in T^*$, $\rho'' \in T^* \cup T^\infty$ and $M_2 \in \mathcal{M}$ in such a way that $\gamma''\rho'' = \eta(k')\theta(k')$, $M_1[\delta_1\rangle_f M_2$, $M_2[\delta_2\rho''\rangle$, δ_1 $(T \setminus J)$-exhausts $\delta\delta'$, and $\delta_1\delta_2$ is Υ-equivalent to $\delta\delta'\gamma''$. Then $M[\beta(k')t_1\delta_1\rangle_f M_2$ and $\beta(k')t_1\delta_1$ $(T \setminus J)$-exhausts $\delta t_1 \delta' = \sigma$.

Let us first consider the case that γ'' is shorter than $\eta(k')$. Let $\delta_3 \in T^*$ be such that $\gamma''\delta_3 = \eta(k')$. Then $\delta_3\theta(k') = \rho''$. We thus have that $M_2[\delta_2\delta_3\theta(k')\rangle$. On the other hand, $\xi(k')\theta(k') = \rho$. Moreover, $\beta(k')t_1\delta_1\delta_2\delta_3$ is Υ-equivalent to $\sigma\xi(k')$ since $\delta_1\delta_2\delta_3$ is Υ-equivalent to $\delta\delta'\gamma''\delta_3 = \delta\delta'\eta(k')$ whereas $\beta(k')t_1\delta\delta'\eta(k')$ is Υ-equivalent to $\sigma\xi(k')$.

Let us then consider the case that γ'' is at least as long as $\eta(k')$. Let $\delta_4 \in T^*$ be such that $\eta(k')\delta_4 = \gamma''$. Then $\delta_4\rho'' = \theta(k')$. We thus have that $\xi(k')\delta_4\rho'' = \xi(k')\theta(k') = \rho$. On the other hand, $M_2[\delta_1\delta_2\rho''\rangle$. Moreover, $\beta(k')t_1\delta_1\delta_2$ is Υ-equivalent to $\sigma\xi(k')\delta_4$ since $\delta_1\delta_2$ is Υ-equivalent to $\delta\delta'\gamma'' = \delta\delta'\eta(k')\delta_4$ whereas $\beta(k')t_1\delta\delta'\eta(k')$ is Υ-equivalent to $\sigma\xi(k')$. □

If we have a series of finite prefixes of an infinite sequence in the full reachability graph, Lemma 5.7 gives us a series of finite sequences in the reduced reachability graph, but the series is not necessarily a series of prefixes of any single infinite sequence. There is still one thing we can do: we can move along a path in the reduced reachability graph and apply Lemma 5.7 to any of the markings on the path. Guaranteeing Υ-equivalence is then not difficult at all since for any infinite sequence, Lemma 5.7 just modifies some finite prefix of the sequence and leaves the rest of the infinite sequence untouched. We then just have to make sure that we choose a prefix that includes some of the so far untouched part of the original infinite sequence. This is the idea of the proof of the below Lemma 5.8.

Lemma 5.8 *Assumptions:*

- *A1, A2, A3, A4 and A5.*

- *$M \in \mathcal{M}$ and $\sigma \in T^\infty$ are such that $M[\sigma\rangle$, M is f-reachable from M_0 and σ contains infinitely many occurrences of transitions from $T \setminus J$.*

Claim: We can define a function μ from N to $\mathcal{M}$, functions β, λ, ζ and γ, from N to T^* and a function θ from N to T^∞ in such a way that for each $n \in N$,

- $\zeta(n)\gamma(n)\theta(n) = \sigma$,
- $M[\beta(n)\rangle_f \mu(n)$,
- $\mu(n)[\lambda(n)\theta(n)\rangle$,
- $\beta(n)$ *$(T \setminus J)$-exhausts* $\zeta(n)$,
- $\beta(n)\lambda(n)$ *is Υ-equivalent to* $\zeta(n)\gamma(n)$, *and*
- $n = 0 \vee$
  $(n > 0 \wedge \ (\exists \eta \in (T^+) \setminus (J^+) \ \ \beta(n) = \beta(n-1)\eta) \wedge$
  $(\exists \xi \in (T^+) \setminus (J^+) \ \ \zeta(n) = \zeta(n-1)\xi))$.

Proof. We use induction on n. By letting $\mu(0) = M$, $\beta(0) = \varepsilon$, $\lambda(0) = \varepsilon$, $\zeta(0) = \varepsilon$, $\gamma(0) = \varepsilon$ and $\theta(0) = \sigma$, we make the claim hold when restricted to $n = 0$. Let then $k \in N$. Our induction hypothesis is that the claim holds when restricted to $n = k$.

From the induction hypothesis it follows that $\mu(k)$ is f-reachable from M_0 and $\lambda(k)\theta(k)$ is enabled at $\mu(k)$. From the given assumption we know that σ contains infinitely many occurrences of transitions from $T \setminus J$. By Lemma 5.7 we can thus choose $\xi_1 \in (T^+) \setminus (J^+)$, $\xi_2 \in T^*$, $\rho \in T^\infty$, $\eta_1 \in T^*$, $\eta_2 \in T^*$ and $M_1 \in \mathcal{M}$ in such a way that $\xi_1\xi_2\rho = \theta(k)$, $\mu(k)[\eta_1\rangle_f M_1$, $M_1[\eta_2\rho\rangle$, η_1 $(T \setminus J)$-exhausts $\lambda(k)\xi_1$, and $\eta_1\eta_2$ is Υ-equivalent to $\lambda(k)\xi_1\xi_2$.

Now $\zeta(k)\gamma(k)\xi_1\xi_2\rho = \zeta(k)\gamma(k)\theta(k)$ which by the induction hypothesis is equal to σ. Since $\mu(k)[\eta_1\rangle_f M_1$ and (because of the induction hypothesis) $M[\beta(k)\rangle_f \mu(k)$, we conclude that $M[\beta(k)\eta_1\rangle_f M_1$. Since η_1 $(T \setminus J)$-exhausts $\lambda(k)\xi_1$ whereas (because of the induction hypothesis) $\beta(k)\lambda(k)$ is Υ-equivalent to $\zeta(k)\gamma(k)$, we conclude that $\beta(k)\eta_1$ $(T \setminus J)$-exhausts $\beta(k)\lambda(k)\xi_1$ which in turn $(T \setminus J)$-exhausts $\zeta(k)\gamma(k)\xi_1$. (The "exhausting" included in the induction hypothesis is not utilized.) Since $\eta_1\eta_2$ is Υ-equivalent to $\lambda(k)\xi_1\xi_2$ whereas $\beta(k)\lambda(k)$ is Υ-equivalent to $\zeta(k)\gamma(k)$, we conclude that $\beta(k)\eta_1\eta_2$ is Υ-equivalent to $\beta(k)\lambda(k)\xi_1\xi_2$ which in turn is Υ-equivalent to $\zeta(k)\gamma(k)\xi_1\xi_2$. Since $\xi_1 \in (T^+) \setminus (J^+)$ whereas η_1 $(T \setminus J)$-exhausts $\lambda(k)\xi_1$, we conclude that $\eta_1 \in (T^+) \setminus (J^+)$.

The induction step thus succeeds by letting $\mu(k+1) = M_1$, $\beta(k+1) = \beta(k)\eta_1$, $\lambda(k+1) = \eta_2$, $\zeta(k+1) = \zeta(k)\gamma(k)\xi_1$, $\gamma(k+1) = \xi_2$ and $\theta(k+1) = \rho$. □

Lemma 5.9 *Assumptions: A1, A2, A3, A4 and A5.*

Claim: For each $\sigma'' \in T^\infty$ and for each $M'' \in \mathcal{M}$, if $M''[\sigma''\rangle$, M'' is f-reachable from M_0 and σ'' contains infinitely many occurrences of transitions from $T \setminus J$, then there exists $\delta'' \in T^\infty$ in such a way that $M''[\delta''\rangle_f$ and δ'' is Υ-equivalent to σ''.

Proof. Let $\sigma \in T^\infty$ and $M \in \mathcal{M}$ be such that $M[\sigma\rangle$, M is f-reachable from M_0, and σ contains infinitely many occurrences of transitions from $T \setminus J$. We can then define a function μ from N to $\mathcal{M}$, functions β, λ, ζ and γ, from N to T^* and a function θ from N to T^∞ in the way stated in the claim of Lemma 5.8. The function β represents an infinite transition sequence that is f-enabled at M. Let ω be this infinite sequence.

- Let ξ be any finite prefix of σ. Then there must be $m \in N$ in such a way that ξ is a prefix of $\zeta(m)$. Now $\beta(m)$ $(T \setminus J)$-exhausts $\zeta(m)$ whereas $\beta(m)\lambda(m)$ is Υ-equivalent to $\zeta(m)\gamma(m)$. Consequently, for any $Y \in \Upsilon$, the Y-restriction of ξ is a prefix of or equal to the Y-restriction of $\beta(m)$. On the other hand, we know that $\beta(m)$ is a finite prefix of ω.
- Let η be any finite prefix of ω. Then there must be $n \in N$ in such a way that η is a prefix of $\beta(n)$. Now $\beta(n)\lambda(n)$ is Υ-equivalent to $\zeta(n)\gamma(n)$. Consequently, for any $Y \in \Upsilon$, the Y-restriction of η is a prefix of or equal to the Y-restriction of $\zeta(n)\gamma(n)$. On the other hand, we know that $\zeta(n)\gamma(n)$ is a finite prefix of σ.

From the above considerations it follows that the infinite sequence ω is Υ-equivalent to the infinite sequence σ. □

We are now ready to collect together the results we have obtained and prove the desired theorem. The task is simple since all the hard work has been done in proving the lemmas. Note that according to A1 – A5, "everything is possibly infinite".

Theorem 5.10 *Assumptions: A1, A2, A3, A4 and A5.*

Claims:

(C1) Every terminal path in the f-reachability graph of the net is a terminal path of the full reachability graph of the net.

(C2) For each terminal path starting from M_0 in the full reachability graph, there exists a terminal path starting from M_0 in the f-reachability graph in such a way that the labels of the paths are Υ-equivalent.

(C3) For each finite path starting from M_0 in the full reachability graph, there exists a finite path starting from M_0 in the f-reachability graph in such a way that the labels of the paths are Υ-equivalent.

(C4) For each infinite path starting from M_0 in the full reachability graph, there exists an infinite path starting from M_0 in the f-reachability graph in such a way that the labels of the paths are Υ-equivalent.

Proof. C1 follows trivially from D2. C2 is an immediate consequence of Lemma 5.4. C3 follows directly from Lemma 5.7, by letting $\rho' = \varepsilon$.

From Lemma 5.7, by letting $\rho' \in J^\infty$, and from Lemma 5.5 it directly follows that C4 holds when restricted to a path where some suffix of the label of the path is in J^∞. From Lemma 5.9 it immediately follows that C4 holds when restricted to a path where no suffix of the label is in J^∞. □

As we see from Proposition 5.2, C3 is actually not needed in our LTL verification problem. However, C3 is interesting by its own virtue, at least if Υ-equivalence is thought of as a behavioural equivalence.

5.2 An algorithm for generating a reduced state space

We are now ready to present a version of the deletion algorithm that produces stubborn sets for the refined LTL-preserving reachability graph generation algorithm. The advanced version of the deletion algorithm is presented in Figure 19.

The routine AdvDelAlg computes the stubborn set. As before, we assume that we are in a nonterminal marking. The and/or-graph is still the same as in Definition 4.25 and determines which vertices must be removed if a given vertex is removed. (Protection flags and other things belonging to a single transition record are "declared" in the discussion that precedes Figure 16.) The computed stubborn set is the remaining set of white transitions. The set $\Upsilon \subseteq 2^T$ is given to the algorithm as an input and is assumed to be like in Proposition 5.2. A member of Υ is presented by a structure consisting of a colour (with an arbitrary initial value) and a list of pointers to transitions. Each transition has a possibly empty owner list, i.e. a list of pointers to those members of Υ that contain the transition. As in the discussion after Proposition 5.2, we call a transition *visible* iff at least one member of Υ contains the transition.

If no invisible transition is enabled, the computed stubborn set contains all transitions. Otherwise the computed stubborn set has at least one invisible key transition. If the computed stubborn set contains an enabled transition from a member of Υ, then the computed stubborn set contains all transitions of that member. The assumptions of Theorem 5.10, except possibly A5, are thus satisfied.

Let us consider the case that at least one enabled invisible transition exists. Then the first priority of AdvDelAlg is to construct a stubborn set that has no enabled visible transition. (The first call to Prepare "removes all visible transitions" if possible, and otherwise returns zero, indicating failure.) Such a set is constructed whenever it is possible to do that. If visible transitions cannot be completely excluded, then some of the members of Υ are included "as whole blocks". If several members of Υ share some enabled transition, these members of Υ are treated as a single block. (Note that the set of enabled transitions depends on the current marking, so the forming of the blocks depends on the current marking, too.) If it is possible to construct a stubborn set that contains at most one of such blocks only, AdvDelAlg constructs such a stubborn set. Otherwise all enabled visible transitions are included. (We could do more work to find something between the "extreme" alternatives, but here we have made a compromise in order to keep the worst-case running time "tolerable".) A typical run of AdvDelAlg ends with a call to Improve that "improves" a stubborn set by constructing a stubborn set that includes the same "visible blocks" (if any) but possibly less enabled invisible transitions.

Above we mentioned that Υ is given as an input to the algorithm. As can be concluded from Proposition 5.2, the only real problem in constructing such Υ is how we obtain for each atomic subformula a good upper bound to the set of transitions that determine the satisfaction of that atomic subformula. One way to obtain upper

```
/*1*/    int Prepare(void /* no argument */) {
/*2*/       while ( there is at least one unprotected enabled visible white
/*3*/               transition ) {
/*4*/          let t be some unprotected enabled visible white transition;
/*5*/          Speculate(t) /* see Figure 16 */;
/*6*/          if ( the set of white transitions contains all protected transitions
/*7*/                  and has an invisible key transition )
/*8*/             Darken(t) /* see Figure 16 */;
/*9*/          else return 0;
/*10*/      }
/*11*/      return 1; }
/*12*/   void Improve(void /* no argument */) {
/*13*/      while ( there are at least two enabled invisible white transitions
/*14*/              and at least one enabled invisible white transition has
/*15*/              a zero root flag ) {
/*16*/         let t be some enabled invisible white transition having
/*17*/             a zero root flag;
/*18*/         set the root flag of t equal to 1; Speculate(t);
/*19*/         if ( the set of white transitions contains all protected transitions
/*20*/                 and has an invisible key transition )
/*21*/            Darken(t);
/*22*/         else Rehabilitate(t) /* see Figure 16 */; } }
/*23*/   void AdvDelAlg(void /* no argument */) {
/*24*/      initialize the and/or-graph;
/*25*/      if ( no invisible transition is enabled ) return;
/*26*/      make all transitions unprotected;
/*27*/      if ( Prepare() is equal to 1 ) { Improve(); return; }
/*28*/      mark all members of ϒ yellow;
/*29*/      while ( at least one member of ϒ is yellow and contains
/*30*/              at least one enabled transition ) {
/*31*/         let Y be some yellow member of ϒ that contains
/*32*/             at least one enabled transition;
/*33*/         for ( each enabled transition t in Y ) {
/*34*/             mark all owners of t orange;
/*35*/         }
/*36*/         initialize the and/or-graph;
/*37*/         make all transitions unprotected;
/*38*/         protect all transitions in orange members of ϒ;
/*39*/         if ( Prepare() is equal to 1 ) { Improve(); return; }
/*40*/         mark all orange members of ϒ blue;
/*41*/      }
/*42*/      initialize the and/or-graph;
/*43*/      make all transitions unprotected;
/*44*/      protect all transitions in blue members of ϒ;
/*45*/      Improve(); }
```

Figure 19: An advanced deletion algorithm.

bounds is as follows. Let p be an atomic formula, i.e. a set of markings. We first compute a subset of places, $B \subseteq S$, such that whenever we have markings M and M' for which $M \in p$ and $M' \notin p$, there exists $s \in B$ for which $M(s) \neq M'(s)$. Of course, we try to find an as small as possible B. Then we let

$$\{t \in T \mid \exists s \in B \;\; W(s,t) \neq W(t,s)\}$$

be the set of "transitions for p".

Above, the term "upper bound" is used intentionally since a set of markings may well have been defined implicitly, e.g. we may be able to evaluate quickly if a given marking is in p but we do not necessarily have any good way to list the members of p. Implicit definitions may occur e.g. when a transformation from a high-level modelling formalism into a place/transition net is used.

The amount of space required by the advanced deletion algorithm is at most proportional to $\zeta + \sum_{Y \in \Upsilon} |Y|$ where ζ is the amount of space required by the basic deletion algorithm. The time taken by an execution of the advanced deletion algorithm is at most proportional to $|\Upsilon|\xi + \sum_{Y \in \Upsilon} |Y|)$ where ξ is the time taken by an execution of the basic deletion algorithm. However, if a stubborn set with no enabled visible transition exists, the advanced deletion algorithm consumes about as much time as the basic deletion algorithm.

If the last line of AdvDelAlg is reached, then some proper subset of the set of enabled transitions of the computed stubborn set may be the set of enabled transitions of some equally acceptable stubborn set. We have not found any way that would eliminate this phenomenon completely without making the worst-case time complexity of the algorithm exponential in $|\Upsilon|$.

So far we have succeeded in satisfying the assumptions of Theorem 5.10, except A5. Since we do not know any efficient way that would satisfy the condition in Theorem 5.10 as weakly as possible, we satisfy the following condition: every elementary cycle in the reduced reachability graph contains at least one marking where the chosen stubborn set contains all visible transitions. This condition can be satisfied by using the same technique as in [78]. If this condition is satisfied and the reduced reachability graph is finite, A5 is satisfied.

A refined LTL-preserving algorithm for generating a reduced reachability graph of a net is presented in Figure 20. (The type **Marking** is assumed to be as the name suggests and is assumed to have been defined appropriately.) The routine Generate constructs the reduced reachability graph. The graph under construction is $\langle V, A \rangle$ where V is the set of vertices and A is the set of edges. Each generated marking has an associated expansion flag. The set K contains markings needed in the detection of cycles that would become created and violate the cycle condition if there were no recomputation of stubborn sets. On the outermost level the algorithm is very close to the corresponding algorithm in [78], and the correctness proof in [78] is essentially applicable as such to show that our algorithm really satisfies A5.

Let us return to the first example given after the definition of A1–A5. The reduced reachability graph discussed in the example becomes unavoidably constructed if we apply the algorithm in Figure 20, using the same Υ as in the example. The pseudo-code should not be too ambigous for readers to simulate the algorithm by themselves.

```
/*1*/    void Traverse(Marking M) {
/*2*/       mark M expanded;
/*3*/       if ( no transition is enabled at M ) return;
/*4*/       AdvDelAlg() /* see Figure 19 */;
/*5*/       if ( some enabled transition is not white, some visible transition
/*6*/               is not white, and some white transition leads from M
/*7*/               either to M itself or to some marking in K ) {
/*8*/          initialize the and/or-graph;
/*9*/          protect all visible transitions and make others unprotected;
/*10*/         Improve() /* see Figure 19 */;
/*11*/      }
/*12*/      for ( each enabled white transition t ) {
/*13*/         let M' be the marking to which t leads from M;
/*14*/         if ( M' is not in V ) {
/*15*/            insert M' into V and mark M' unexpanded;
/*16*/         }
/*17*/         insert ⟨M, t, M'⟩ into A;
/*18*/      }
/*19*/      if ( all enabled transitions are white or
/*20*/         all visible transitions are white ) return;
/*21*/      insert M into K;
/*22*/      for ( each edge of A starting from M and ending to
/*23*/              an unexpanded marking ) {
/*24*/         let M' be the marking to which the edge ends;
/*25*/         Traverse(M');
/*26*/      }
/*27*/      remove M from K; }
/*28*/   void Generate(void /* no argument */) {
/*29*/      make V, A and K empty;
/*30*/      insert the initial marking into V;
/*31*/      mark the initial marking unexpanded;
/*32*/      while ( V contains at least one unexpanded marking ) {
/*33*/         let M be some unexpanded marking in V;
/*34*/         Traverse(M); } }
```

Figure 20: A refined LTL-preserving reachability graph generation algorithm.

5.3 Treating operation fairness

We now consider verification under the assumption of operation fairness. In order to guarantee that operation fair paths are sufficiently retained in a reduction, we extend the assumptions A1–A5 by the following assumption A6 and then drop assumption A4 since A4 and A6 together would simply force us to generate the full reachability graph.

(A6) Let ϖ be a function from T to 2^T such that for each $t \in T$, $\{t' \in T \mid \exists s \in {}^\bullet t \;\; W(s,t') \neq W(t',s)\} \subseteq \varpi(t)$. Then $\mathcal{T} \cup \mathcal{Y} \subseteq \Upsilon$ where $\mathcal{T} = \{\{t\} \mid t \in T\}$ and $\mathcal{Y} = \{\varpi(t) \mid t \in T\}$.

From A6 it follows that all transitions are visible. The set $\varpi(t)$ contains at least all those transition that are "visible w.r.t. the enabledness of t", where visibility is understood in the same way as in the discussion after Proposition 5.2. The separation of $\mathcal{T}$ and $\mathcal{Y}$ reflects the fact that the definition of operation fairness does not say anything about what should happen if a transition is enabled at most finitely many times.

Lemma 5.11 has the effect that if two paths in the full reachability graph start from the same marking and have $(\mathcal{T} \cup \mathcal{Y})$-equivalent labels, then both of the paths are operation fair or neither of them is operation fair.

Lemma 5.11 *Assumptions:*

- *A1 and A6.*
- *M is a reachable marking, and x and y are paths that start from M in the full reachability graph of the net.*
- *The label of y is Υ-equivalent to the label of x.*
- *The path x is not operation fair.*

Claim: The path y is not operation fair either.

Proof. From A6 it follows that y is both $\mathcal{T}$-equivalent and $\mathcal{Y}$-equivalent to x. Since x is not operation fair, we conclude that x is an infinite path, and we can choose $t \in T$ in such a way that t is enabled infinitely many times on x but occurs at most finitely many times in x.

Since y is $\mathcal{T}$-equivalent to x, we conclude that y is an infinite path and t occurs at most finitely many times in y. Since y is $\mathcal{Y}$-equivalent to x, the $\varpi(t)$-restriction of the label of y is equal to the $\varpi(t)$-restriction of the label of x. From the definition of ϖ it thus follows that t is enabled infinitely times on y. □

Lemma 5.11 is apparently analogous to Lemma 5.1. If we had defined an action-oriented version of LTL [42, 55], operation fairness could have been expressed as

an ordinary formula (except possibly in the case that the set of transitions is infinite), and A6 would have been obtained as a side effect of the ordinary construction principles of Υ.

Note that $\mathcal{Y}$-equivalence does not imply $\mathcal{T}$-equivalence. If a net has transitions but no place, we can let $\varpi(t) = \emptyset$ for each transition t, with the consequence that any two transition sequences are $\mathcal{Y}$-equivalent. On the other hand, being in the same strict trace does not imply $\mathcal{Y}$-equivalence. If we return to the example concerning the net in Figure 4, we see that the sequences $(abcfdg)(abcfdg)(abcfdg)\ldots$ and $(abfcdg)(abfcdg)(abfcdg)\ldots$ are not $\mathcal{Y}$-equivalent since both of c and f must be in $\varpi(e)$.

Lemma 5.12 is much like Lemma 5.6. The effective difference is that the assumption A4 has been replaced by the assumption A6, the sequence being handled is definitely infinite and a label of an operation fair path.

Lemma 5.12 *Assumptions:*

- *A1, A2, A3 and A6.*
- *$M \in \mathcal{M}$, $\sigma \in T^*$, $\rho \in T^\infty$ and $L \subseteq T$ are such that M is a reachable marking, $M[\sigma\rho\rangle$ and L is the set of those transitions that occur in σ.*
- *The path starting from M and being labelled by $\sigma\rho$ in the full reachability graph is operation fair.*

Claim: We can define a function β from N to $(T \setminus L)^*$, functions ξ and η from N to T^*, a function μ from N to $\mathcal{M}$ and a function θ from N to T^∞ in such a way that for each $n \in N$,

- $\xi(n)\theta(n) = \rho$,
- $M[\beta(n)\rangle_f \mu(n)$,
- $\mu(n)[\sigma\eta(n)\theta(n)\rangle$,
- *$\beta(n)\sigma\eta(n)$ is Υ-equivalent to $\sigma\xi(n)$, and*
- $$\begin{array}{l} n = 0 \vee \\ (n > 0 \wedge \ (\forall m \in N\ \forall t \in L \ \ \mu(m)[t\rangle_f \Rightarrow m \geq n) \wedge \\ \qquad\qquad (\exists t_n \in T \setminus L \ \ \beta(n) = \beta(n-1)t_n)) \vee \\ (n > 0 \wedge (\exists t_n \in L \ \ \mu(n-1)[t_n\rangle_f) \wedge \beta(n) = \beta(n-1)). \end{array}$$

Proof. [[The proofs in this section repeat much of the text in the proofs in Section 5.1. The alternative would have been either to list the differences only, with the risk of fatal ambiguity, or to transform the lemmas in Section 5.1 into a more generic form, with the risk of fatal obscurity.]]

We use induction on n. By letting $\beta(0) = \varepsilon$, $\xi(0) = \varepsilon$, $\eta(0) = \varepsilon$, $\mu(0) = M$ and $\theta(0) = \rho$, we make the claim hold when restricted to $n = 0$. Let then $k \in N$. Our induction hypothesis is that the claim holds when restricted to $n = k$.

- If there exists $\tau \in L$ such that $\mu(k)[\tau\rangle_f$, we let $\beta(k+1) = \beta(k)$, $\xi(k+1) = \xi(k)$, $\eta(k+1) = \eta(k)$, $\mu(k+1) = \mu(k)$ and $\theta(k+1) = \theta(k)$. The induction step is thus trivial in this case.

In the below considerations, no transition of L is f-enabled at $\mu(k)$. Using the induction hypothesis, we then immediately conclude that for each $m < k+1$, no transition of L is f-enabled at $\mu(m)$. Since we also know (by the induction hypothesis) that σ is enabled at $\mu(k)$, Lemma 4.2 has the effect that $\sigma \in (T \setminus f(\mu(k)))^*$.

- Let us consider the case where $\eta(k)$ contains a transition from $f(\mu(k))$. Then we can choose $\tau_k \in f(\mu(k))$, $\gamma_k \in (T \setminus f(\mu(k)))^*$ and $\zeta_k \in T^*$ in such a way that $\gamma_k\tau_k\zeta_k = \eta(k)$. From the induction hypothesis it follows that there exists a marking M such that $\mu(k)[\sigma\gamma_k\tau_k\rangle M$. Since $\sigma\gamma_k \in (T \setminus f(\mu(k)))^*$, Lemma 5.3 has the effect that $\mu(k)[\tau_k\sigma\gamma_k\rangle M$, and $\tau_k\sigma\gamma_k$ is Υ-equivalent to $\sigma\gamma_k\tau_k$. So, $\beta(k)\tau_k\sigma\gamma_k\zeta_k$ is Υ-equivalent to $\beta(k)\sigma\gamma_k\tau_k\zeta_k = \beta(k)\sigma\eta(k)$ which by the induction hypothesis is Υ-equivalent to $\sigma\xi(k)$. The induction step thus succeeds by letting $\beta(k+1) = \beta(k)\tau_k$, $\xi(k+1) = \xi(k)$, $\eta(k+1) = \gamma_k\zeta_k$ and $\theta(k+1) = \theta(k)$, and by requiring that $\mu(k)[\tau_k\rangle\mu(k+1)$.

- Let us then consider the case that $\theta(k)$ contains a transition from $f(\mu(k))$ but $\eta(k)$ does not. Then we can choose $\tau_k \in f(\mu(k))$, $\gamma_k \in (T \setminus f(\mu(k)))^*$ and $\zeta_k \in T^* \cup T^\infty$ in such a way that $\gamma_k\tau_k\zeta_k = \theta(k)$. From the induction hypothesis it follows that there exists a marking M such that $\mu(k)[\sigma\eta(k)\gamma_k\tau_k\rangle M$. Since $\sigma\eta(k)\gamma_k \in (T\setminus f(\mu(k)))^*$, Lemma 5.3 has the effect that $\mu(k)[\tau_k\sigma\eta(k)\gamma_k\rangle M$, and $\tau_k\sigma\eta(k)\gamma_k$ is Υ-equivalent to $\sigma\eta(k)\gamma_k\tau_k$. So, $\beta(k)\tau_k\sigma\eta(k)\gamma_k$ is Υ-equivalent to $\beta(k)\sigma\eta(k)\gamma_k\tau_k$ which by the induction hypothesis is Υ-equivalent to $\sigma\xi(k)\gamma_k\tau_k$. The induction step thus succeeds by letting $\beta(k+1) = \beta(k)\tau_k$, $\xi(k+1) = \xi(k)\gamma_k\tau_k$, $\eta(k+1) = \eta(k)\gamma_k$ and $\theta(k+1) = \zeta_k$, and by requiring that $\mu(k)[\tau_k\rangle\mu(k+1)$

- In the ultimate remaining case, $\sigma\eta(k)\theta(k) \in (T \setminus f(\mu(k)))^\infty$. Let x be a path in the full reachability graph in such a way that x starts from $\mu(k)$ and is labelled by $\sigma\eta(k)\theta(k)$. (The induction hypothesis lets us define such a path.) Since (because of the induction hypothesis), $\beta(k)\sigma\eta(k)\theta(k)$ is Υ-equivalent to $\sigma\xi(k)\theta(k) = \sigma\rho$, Lemma 5.11 has the effect that the path starting from M and being labelled by $\beta(k)\sigma\eta(k)\theta(k)$ in the full reachability graph is operation fair. On the other hand, $M[\beta(k)\rangle\mu(k)$ (because of the induction hypothesis). Consequently, the path x is operation fair.

  By D2, $f(\mu(k))$ has at least one dynamic key transition at $\mu(k)$. Let t be any dynamic key transition of $f(\mu(k))$ at $\mu(k)$. Since $\sigma\eta(k)\theta(k) \in (T \setminus f(\mu(k)))^\infty$, t is enabled at all markings on x, but x does not contain any occurrence of t. So, x is not operation fair. We have thus reached a contradiction. This means that there is no case where we would have $\sigma\eta(k)\theta(k) \in (T \setminus f(\mu(k)))^\infty$. □

Lemma 5.13 is a similar analogy to Lemma 5.7 ad Lemma 5.12 is to Lemma 5.6. Again, the assumption A4 has been replaced by the assumption A6. The sequence to be transformed is definitely infinite and a label of an operation fair path, the result of the transformation is a permutation of the original sequence, and the prefix covering condition has been fixed according to the fact that $J = \emptyset$.

Lemma 5.13 *Assumptions: A1, A2, A3, A5 and A6.*

Claim: For each $\sigma'' \in T^*$, for each $\rho' \in T^\infty$ and for each $M' \in \mathcal{M}$, if $M'[\sigma''\rho'\rangle$, M' is f-reachable from M_0 and the path starting from M' and being labelled by $\sigma''\rho'$ in the full reachability graph is operation fair, then there exist $\gamma'' \in T^*$, $\delta_1 \in T^*$, $\delta_2 \in T^*$, $\rho'' \in T^\infty$ and $M'' \in \mathcal{M}$ in such a way that $\gamma''\rho'' = \rho'$, $M'[\delta_1\rangle_f M''$, $M''[\delta_2\rho''\rangle$, δ_1 T-exhausts σ'', and $\delta_1\delta_2$ is Υ-equivalent to $\sigma''\gamma''$.

Proof. [[It basically suffices to repeat the proof of Lemma 5.7 literally, with the following modifications: the induction hypothesis must refer to the claim of the lemma being proved, the reference to Lemma 5.6 is replaced by a reference to Lemma 5.12, and an observation concerning operation fairness is needed in the induction step.]]

Due to A6, the set J is empty. We use induction on the length of σ''. The claim holds trivially when restricted to $\sigma'' = \varepsilon$. Our induction hypothesis is that the claim holds when restricted to any σ'' of length $n \geq 0$. Let $\sigma \in T^*$, $\rho \in T^\infty$ and $M \in \mathcal{M}$ be such that $M[\sigma\rho\rangle$, M is f-reachable from M_0, the length of σ is $n+1$, and the path starting from M and being labelled by $\sigma\rho$ in the full reachability graph is operation fair.

Let L be the set of those transitions that occur in σ. Then we can define a function β from N to $(T \setminus L)^*$, functions ξ and η from N to T^*, a function μ from N to $\mathcal{M}$ and a function θ from N to $T^* \cup T^\infty$ in the way stated in the claim of Lemma 5.12. We thus have that for each $k \in N$, $\xi(k)\theta(k) = \rho$, $M[\beta(k)\rangle_f \mu(k)$, $\mu(k)[\sigma\eta(k)\theta(k)\rangle$, and $\beta(k)\sigma\eta(k)$ is Υ-equivalent to $\sigma\xi(k)$.

Let us first assume that there are no $k' \in N$ and $\tau' \in L$ that would satisfy $\mu(k')[\tau'\rangle_f$. Let us call this assumption B. Let t be any transition in L. From B and A5 and from the emptiness of J it follows that there exists $k'' \in N$ such that $t \in f(\mu(k''))$. Consequently, there must be some $k_1 \leq k''$, $t' \in L \cap f(\mu(k_1))$, $\gamma \in (L \setminus f(\mu(k_1)))^*$ and $\gamma' \in L^*$ such that $\sigma = \gamma t' \gamma'$. Since $\mu(k_1)[\sigma\rangle$, from D1 it follows that $\mu(k_1)[t'\rangle_f$. We have thus reached a contradiction with B.

So, we can choose $k' \in N$ and $\tau' \in L$ such that $\mu(k')[\tau'\rangle_f$. Since $\mu(k')[\sigma\rangle$, there are some $t_1 \in f(\mu(k'))$, $\delta \in (T \setminus f(\mu(k')))^*$ and $\delta' \in T^*$ such that $\sigma = \delta t_1 \delta'$. Since $\mu(k')[\sigma\eta(k')\theta(k')\rangle$, from Lemma 5.3 it follows that there exists a marking M_1 such that $\mu(k')[t_1\rangle_f M_1$, $M_1[\delta\delta'\eta(k')\theta(k')\rangle$, and $t_1\delta\delta'$ is Υ-equivalent to σ. So, $\beta(k')t_1\delta\delta'\eta(k')$ is Υ-equivalent to $\sigma\xi(k')$ since $\beta(k')\sigma\eta(k')$ is Υ-equivalent to $\sigma\xi(k')$.

Let x be the path starting from M and being labelled by $\sigma\rho$ in the full reachability graph. Respectively, let y be the path starting from M and being labelled by $\beta(k')t_1\delta\delta'\eta(k')\theta(k')$ in the full reachability graph. Finally, let z be the path starting from M_1 and being labelled by $\delta\delta'\eta(k')\theta(k')$ in the full reachability graph. Since x is operation fair and $\sigma\xi(k')\theta(k') = \sigma\rho$ is Υ-equivalent to $\beta(k')t_1\delta\delta'\eta(k')\theta(k')$, Lemma 5.11 has the effect that y is operation fair. Since $M[\beta(k')t_1\rangle M_1$, we conclude that z is operation fair, too.

By the induction hypothesis, there exists $\gamma'' \in T^*$, $\delta_1 \in T^*$, $\delta_2 \in T^*$, $\rho'' \in T^\infty$ and $M_2 \in \mathcal{M}$ in such a way that $\gamma''\rho'' = \eta(k')\theta(k')$, $M_1[\delta_1\rangle_f M_2$, $M_2[\delta_2\rho''\rangle$, δ_1 T-exhausts $\delta\delta'$, and $\delta_1\delta_2$ is Υ-equivalent to $\delta\delta'\gamma''$. Then $M[\beta(k')t_1\delta_1\rangle_f M_2$ and $\beta(k')t_1\delta_1$ T-exhausts $\delta t_1 \delta' = \sigma$.

Let us first consider the case that γ'' is shorter than $\eta(k')$. Let $\delta_3 \in T^*$ be such that $\gamma''\delta_3 = \eta(k')$. Then $\delta_3\theta(k') = \rho''$. We thus have that $M_2[\delta_2\delta_3\theta(k')\rangle$. On the other hand, $\xi(k')\theta(k') = \rho$. Moreover, $\beta(k')t_1\delta_1\delta_2\delta_3$ is Υ-equivalent to $\sigma\xi(k')$ since $\delta_1\delta_2\delta_3$ is Υ-equivalent to $\delta\delta'\gamma''\delta_3 = \delta\delta'\eta(k')$ whereas $\beta(k')t_1\delta\delta'\eta(k')$ is Υ-equivalent to $\sigma\xi(k')$.

Let us then consider the case that γ'' is at least as long as $\eta(k')$. Let $\delta_4 \in T^*$ be such that $\eta(k')\delta_4 = \gamma''$. Then $\delta_4\rho'' = \theta(k')$. We thus have that $\xi(k')\delta_4\rho'' = \xi(k')\theta(k') = \rho$. On the other hand, $M_2[\delta_1\delta_2\rho''\rangle$. Moreover, $\beta(k')t_1\delta_1\delta_2$ is Υ-equivalent to $\sigma\xi(k')\delta_4$ since $\delta_1\delta_2$ is Υ-equivalent to $\delta\delta'\gamma'' = \delta\delta'\eta(k')\delta_4$ whereas $\beta(k')t_1\delta\delta'\eta(k')$ is Υ-equivalent to $\sigma\xi(k')$. □

Lemmas 5.14 and 5.15 are similar continuations to Lemma 5.13 as lemmas 5.8 and 5.9 are to Lemma 5.7.

Lemma 5.14 *Assumptions:*

- *A1, A2, A3, A5 and A6.*
- *$M \in \mathcal{M}$ and $\sigma \in T^\infty$ are such that $M[\sigma\rangle$, and M is f-reachable from M_0.*
- *The path starting from M and being labelled by σ in the full reachability graph is operation fair.*

Claim: We can define a function μ from N to $\mathcal{M}$, functions β, λ, ζ and γ, from N to T^* and a function θ from N to T^∞ in such a way that for each $n \in N$,

- $\zeta(n)\gamma(n)\theta(n) = \sigma$,
- $M[\beta(n)\rangle_f \mu(n)$,
- $\mu(n)[\lambda(n)\theta(n)\rangle$,
- *the path starting from $\mu(n)$ and being labelled by $\lambda(n)\theta(n)$ in the full reachability graph is operation fair,*
- *$\beta(n)$ T-exhausts $\zeta(n)$,*
- *$\beta(n)\lambda(n)$ is Υ-equivalent to $\zeta(n)\gamma(n)$, and*
- $n = 0 \vee$
  $(n > 0 \wedge (\exists \eta \in T^+ \;\; \beta(n) = \beta(n-1)\eta) \wedge (\exists \xi \in T^+ \;\; \zeta(n) = \zeta(n-1)\xi))$.

Proof. [[It basically suffices to repeat the proof of Lemma 5.8 literally, with the following modifications: an observation concerning operation fairness is needed before applying any inductive argument, and the reference to Lemma 5.7 must be replaced by a reference to Lemma 5.13.]]

We use induction on n. By letting $\mu(0) = M$, $\beta(0) = \varepsilon$, $\lambda(0) = \varepsilon$, $\zeta(0) = \varepsilon$, $\gamma(0) = \varepsilon$ and $\theta(0) = \sigma$, we make the claim hold when restricted to $n = 0$. Let then $k \in N$. Our induction hypothesis is that the claim holds when restricted to $n = k$.

From the induction hypothesis it follows that $\mu(k)$ is f-reachable from M_0, $\lambda(k)\theta(k)$ is enabled at $\mu(k)$, and the path starting from $\mu(k)$ and being labelled by $\lambda(k)\theta(k)$ in the full reachability graph is operation fair. By Lemma 5.13 we can thus choose $\xi_1 \in T^+$, $\xi_2 \in T^*$, $\rho \in T^\infty$, $\eta_1 \in T^*$, $\eta_2 \in T^*$ and $M_1 \in \mathcal{M}$ in such a way that $\xi_1\xi_2\rho = \theta(k)$, $\mu(k)[\eta_1\rangle_f M_1$, $M_1[\eta_2\rho\rangle$, η_1 T-exhausts $\lambda(k)\xi_1$, and $\eta_1\eta_2$ is Υ-equivalent to $\lambda(k)\xi_1\xi_2$.

Now $\zeta(k)\gamma(k)\xi_1\xi_2\rho = \zeta(k)\gamma(k)\theta(k)$ which by the induction hypothesis is equal to σ. Since $\mu(k)[\eta_1\rangle_f M_1$ and (because of the induction hypothesis) $M[\beta(k)\rangle_f \mu(k)$, we conclude that $M[\beta(k)\eta_1\rangle_f M_1$. Since η_1 T-exhausts $\lambda(k)\xi_1$ whereas (because of the induction hypothesis) $\beta(k)\lambda(k)$ is Υ-equivalent to $\zeta(k)\gamma(k)$, we conclude that $\beta(k)\eta_1$ T-exhausts $\beta(k)\lambda(k)\xi_1$ which in turn T-exhausts $\zeta(k)\gamma(k)\xi_1$. (The "exhausting" included in the induction hypothesis is not utilized.) Since $\eta_1\eta_2$ is Υ-equivalent to $\lambda(k)\xi_1\xi_2$ whereas $\beta(k)\lambda(k)$ is Υ-equivalent to $\zeta(k)\gamma(k)$, we conclude that $\beta(k)\eta_1\eta_2$ is Υ-equivalent to $\beta(k)\lambda(k)\xi_1\xi_2$ which in turn is Υ-equivalent to $\zeta(k)\gamma(k)\xi_1\xi_2$. Since $\xi_1 \in T^+$ whereas η_1 T-exhausts $\lambda(k)\xi_1$, we conclude that $\eta_1 \in T^+$.

Let x be the path starting from $\mu(k)$ and being labelled by $\lambda(k)\theta(k)$ in the full reachability graph. Respectively, Let y be the path starting from $\mu(k)$ and being labelled by $\eta_1\eta_2\rho$ in the full reachability graph. Finally, let z be the path starting from M_1 and being labelled by $\eta_2\rho$ in the full reachability graph. Since the path x is operation fair and $\lambda(k)\xi_1\xi_2\rho = \lambda(k)\theta(k)$ is Υ-equivalent to $\eta_1\eta_2\rho$, Lemma 5.11 has the effect that y is operation fair. Since $\mu(k)[\eta_1\rangle M_1$, we conclude that z is operation fair, too.

The induction step thus succeeds by letting $\mu(k+1) = M_1$, $\beta(k+1) = \beta(k)\eta_1$, $\lambda(k+1) = \eta_2$, $\zeta(k+1) = \zeta(k)\gamma(k)\xi_1$, $\gamma(k+1) = \xi_2$ and $\theta(k+1) = \rho$. □

Lemma 5.15 *Assumptions: A1, A2, A3, A5 and A6.*

Claim: For each $\sigma'' \in T^\infty$ and for each $M'' \in \mathcal{M}$, if $M''[\sigma''\rangle$, M'' is f-reachable from M_0, and the path starting from M'' and being labelled by σ'' in the full reachability graph is operation fair, then there exists $\delta'' \in T^\infty$ in such a way that $M''[\delta''\rangle_f$ and δ'' is Υ-equivalent to σ''.

Proof. [[It basically suffices to repeat the proof of Lemma 5.9 literally, with the following modifications: the reference to Lemma 5.8 must be replaced by a reference to Lemma 5.14.]]

Let $\sigma \in T^\infty$ and $M \in \mathcal{M}$ be such that $M[\sigma\rangle$, M is f-reachable from M_0, and the path starting from M and being labelled by σ in the full reachability graph is operation fair. We can then define a function μ from N to $\mathcal{M}$, functions β, λ, ζ and γ, from N to T^* and a function θ from N to T^∞ in the way stated in the claim of Lemma 5.14. The function β represents an infinite transition sequence that is f-enabled at M. Let ω be this infinite sequence.

- Let ξ be any finite prefix of σ. Then there must be $m \in N$ in such a way that ξ is a prefix of $\zeta(m)$. Now $\beta(m)$ T-exhausts $\zeta(m)$ whereas $\beta(m)\lambda(m)$ is Υ-equivalent to $\zeta(m)\gamma(m)$. Consequently, for any $Y \in \Upsilon$, the Y-restriction of ξ is a prefix of or equal to the Y-restriction of $\beta(m)$. On the other hand, we know that $\beta(m)$ is a finite prefix of ω.

- Let η be any finite prefix of ω. Then there must be $n \in N$ in such a way that η is a prefix of $\beta(n)$. Now $\beta(n)\lambda(n)$ is Υ-equivalent to $\zeta(n)\gamma(n)$. Consequently, for any $Y \in \Upsilon$, the Y-restriction of η is a prefix of or equal to the Y-restriction of $\zeta(n)\gamma(n)$. On the other hand, we know that $\zeta(n)\gamma(n)$ is a finite prefix of σ.

From the above considerations it follows that the infinite sequence ω is Υ-equivalent to the infinite sequence σ. □

We are now ready to present a preservation theorem for operation fair paths.

Theorem 5.16 *Assumptions: A1, A2, A3, A5 and A6.*

Claims: C1 and C2 of Theorem 5.10 and

> *(C5) For each operation fair infinite path starting from M_0 in the full reachability graph, there exists an operation fair infinite path starting from M_0 in the f-reachability graph in such a way that the labels of the paths are Υ-equivalent.*

Proof. Again, C1 follows trivially from D2 and C2 is an immediate consequence of Lemma 5.4. C5 in turn follows directly from lemmas 5.11 and 5.15. □

Though $(\mathcal{T} \cup \mathcal{Y})$-equivalence preserves operation fairness and its negation, Theorem 5.16 does not promise anything that would concern the paths that are not operation fair. Let Υ' be an arbitrary subset of 2^T, thus not required to satisfy A6. By substituting Υ' for Υ in Theorem 5.10 and $\Upsilon' \cup \mathcal{T} \cup \mathcal{Y}$ for Υ in Theorem 5.16, one could present a corollary to be applied when operation fair counterexamples are expected but a total absence of operation fair counterexamples makes any counterexample acceptable. However, nothing prevents us from simply verifying a formula first under fairness assumptions and then without fairness assumptions. Such a simple approach is even recommendable since retaining several less than strictly related things during a single state space construction is one of the most typical ways to promote state space explosion.

We could now proceed into algorithms, but the truth is that Theorem 5.16 is effectively so close to the corresponding theorems in [55, 56] that it does not seem to imply any algorithmic contribution. A6 clearly forces the used dynamically stubborn sets to be strongly dynamically stubborn. This can in a sense be considered a justification for the explicit treatment of strongly dynamically stubborn sets.

Some comments concerning the *ignoring phenomenon* [77] are perhaps appropriate here. When used formally, the term "ignoring" means the negation of some fairness assumption. The definition used in [77] is oriented towards the verification of certain safety properties. However, since [77] is often referred to without explanations, it should be made clear that the definition of ignoring in [77] is inappropriate in the context of LTL. According to [77], a transition is *ignored at a marking in a reduced reachability graph* iff the transition is enabled at the marking but the graph has no path that would start from the marking and would contain an occurrence of the transition. To see the incompatibility with LTL, let us again return to the net in Figure 4. It is tempting to speculate that A1–A5, AV-strong dynamic stubbornness and absence of ignoring (with the meaning given in [77] and repeated above),

might together help us in verifying a formula under the assumption of operation fairness. Let $\Diamond(M(z) = 1)$ be the formula that we try to verify. We can then choose $\Upsilon = \{\{e\}\}$. Let M_0 be the initial marking, $M_0[a\rangle M_1$, $M_1[b\rangle M_2$, $M_2[c\rangle M_3$, $M_3[d\rangle M_4$, $M_3[e\rangle M_5$, $M_3[f\rangle M_6$, and $M_4[f\rangle M_7$. In order to guarantee A1–A5, AV-strong dynamic stubbornness of the chosen stubborn sets and absence of ignoring, it suffices to choose the stubborn sets in such a way that the chosen set is $\{a\}$ at M_0, $\{b\}$ at M_1, $\{c\}$ at M_2 and the set of all transitions at all other encountered nonterminal markings. Then the set of markings in the reduced reachability graph is $\{M_0, M_1, M_2, M_3, M_4, M_5, M_6, M_7\}$. The infinite paths starting from M_0 in the graph have labels where the sequences $abcdfg$ and $abcfdg$ are constantly repeated or arbitrarily alternated. None of these paths is operation fair. However, the above formula is not valid at M_0 in the full reachability graph under the assumption of operation fairness since the infinite path starting from M_0 and being labelled by $(abfcdg)(abfcdg)(abfcdg)\ldots$ in the full reachability graph is operation fair.

5.4 Discussion

Our algorithms can be used in on-the-fly verification as such, by computing an intersection of a Büchi automaton (that accepts exactly the sequences that satisfy the negation of the formula to be verified) and the reduced reachability graph simultaneously with the construction of the latter.

Without strong practical evidence, it is difficult to say anything firm about how efficient our approach is in practice when compared to some closely related techniques. When a formula to be verified without fairness assumptions is presentable as a Boolean combination of more than one usefully different formulas, our approach can be expected to produce smaller reduced reachability graphs than the approach in [78]. The time consumed in the computation of a single stubborn set depends on the number of members in Υ (as can be seen from the complexity considerations in Section 5.2) and on the basic way to compute stubborn sets (the deletion algorithhm or the incremental algorithm). When fairness is not assumed and the formula to be verified does not express a safety property, the algorithms in [56] are essentially on-the-fly verification versions of the algorithms of [78], whereas [55] does not recommend any special algorithm for such cases.

The tester technique in [81] can be considered more goal-oriented than our technique, but so far we have not found any automatic way to construct a useful tester for an arbitrary formula. The visibility relaxation heuristic of [48] improves the tester technique by utilizing the maximal strongly connected components of a tester state space. (The treatment of visible transitions can to some extent be relaxed simply by switching from the preferences in [81] to the preferences in [78, 80].) In [48], this heuristic is shown to apply very well to automatically constructible Büchi automata, too. However, this relaxation technique does not cover our approach. Let us consider a verification task where we need a Büchi automaton that accepts exactly the sequences that satisfy an $n$ary conjunction of formulas. (The formula to be verified then corresponds to an $n$ary disjunction. If an $n$ary conjunction were to be verified, we could verify it simply by verifying its conjuncts separately.) As can be seen from the construction description [24, 48] and from Lemma 6 of [48], all conjuncts become represented in every state of the automaton. Consequently, the visibility

relaxation heuristic in [48] does not take any obvious advantage of the fact that the $n$ary conjunction in question is a Boolean combination.

The idea to apply the stubborn set method to the verification of nexttime-less LTL-formulas is quite natural because the satisfaction of a formula is measured w.r.t. an execution that can be expected to contain some orders of actions that do not affect the satisfaction. Some kinds of a stubborn set method have been presented for the verification of branching time temporal properties, too, but the suggested solutions [23, 62, 98] place extremely strict assumptions on the stubborn sets used.

6 On heuristics for stubborn set computation

One critical factor in the incremental algorithm possibly producing unnecessarily large stubborn sets is the choice of a *scapegoat*. A scapegoat is a disabling place chosen during the execution of the incremental algorithm for a disabled transition. In Section 6.1, the gravity of the problem together with some heuristics for choosing a scapegoat are described in the context of a classical example.

From the basic deletion algorithm, 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 Section 6.2 is that though the obtained minimization algorithm is not practical as such, an incomplete version of the algorithm can be quite practical.

Many of the algorithms in this thesis have been implemented in a tool called PROD [94, 95]. All of the experimental observations mentioned in this chapter and later in this thesis have been obtained by using PROD.

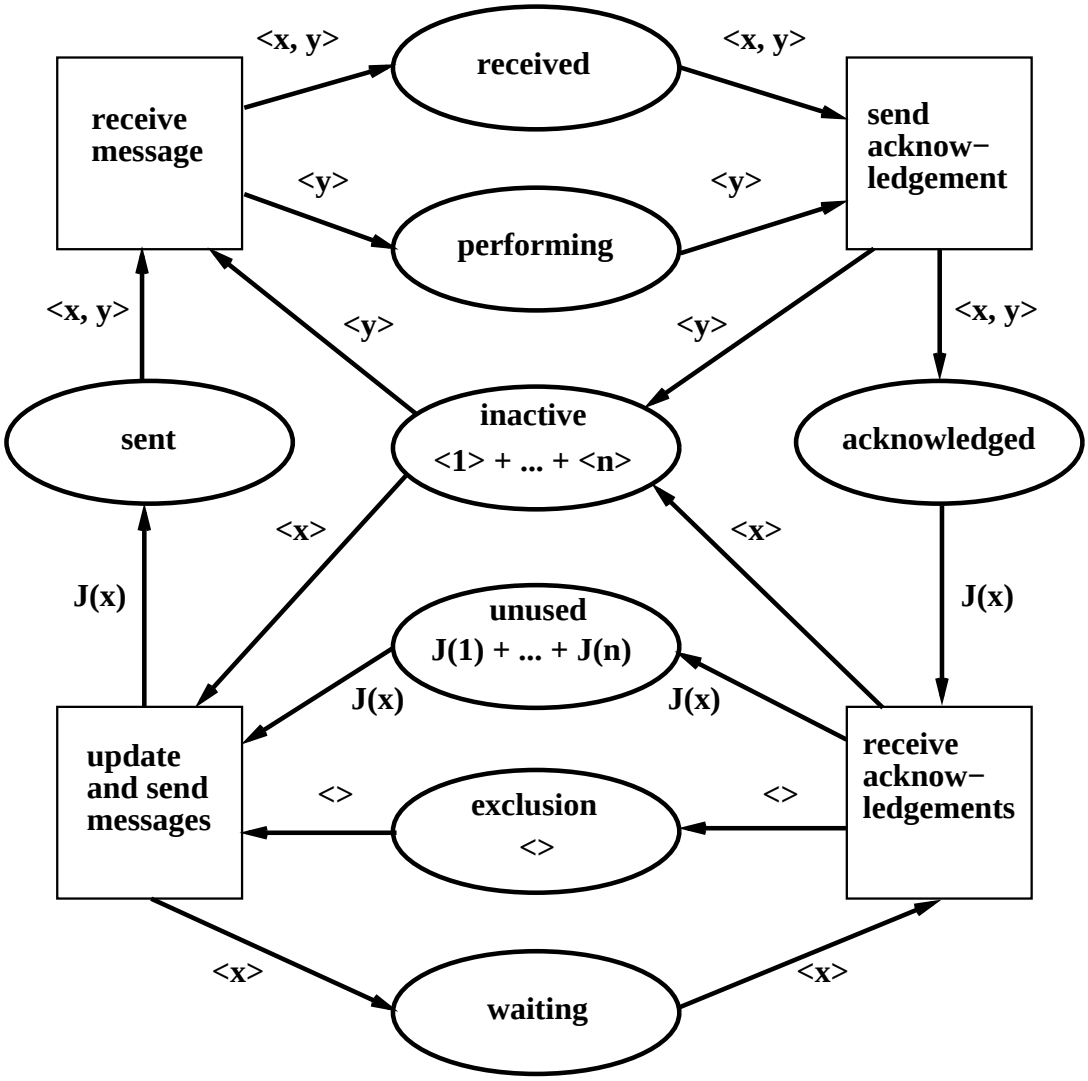
6.1 On choosing a scapegoat in the incremental algorithm

The next example shows that it is by no means exaggerated to say that the stubborn sets computed by the incremental algorithm may contain unnecessarily many enabled transitions.

Figure 21 presents a data base system of $n \geq 2$ data base managers. The predicate/transition net [22] in the figure is equivalent to the coloured Petri net in [40]. The contained resource allocation and a great amount of concurrency makes the system inherently very suitable for the stubborn set method. Let us assume the most obvious unfolding [22] into a place/transition net. The image of a transition instance of the predicate/transition net is a transition of the place/transition net, the image of a place-tuple pair of the predicate/transition net is a place of the place/transition net, etc. The full reachability graph of the place/transition net has $n \cdot 3^{n-1} + 1$ vertices and $2n(1 + (n-1) \cdot 3^{n-2})$ edges [75, 77]. The stubborn set method is capable of producing a reduced reachability graph having $2n^2 - n + 1$ vertices and $2n^2$ edges [75, 77]. In the reduced reachability graph in question (unique up to isomorphism), the vertex corresponding to the initial marking has n immediate successors. Every other vertex has one and only one immediate successor. From now on, this reduced reachability graph will be called the Λ-graph.

We shall now investigate the behaviour of the incremental algorithm in the above place/transition net. Definition 4.21 implies that for each marking M, if a transition t is enabled at M and s is an input place of t, then $E_2(M, t, s) = E_4(s)$. The function E_2 is thus certainly preferable to the function E_3 in the incremental algorithm as far as this net is concerned. So we choose the corresponding choice function b (that occurs in Definition 4.23) to be the function that has the value 2 everywhere.

We shall present four scapegoat generators, called α, β, γ, and δ. The scapegoat generator α is a pathological scapegoat generator not leading to any reduction. The scapegoat generators β, γ, and δ look much like α but γ and δ lead to the Λ-graph, and β leads close to the size of the Λ-graph. In addition, pseudo-random scapegoat generators are considered. They tend to be almost as bad as α.



(J(x') = J(x', 1) + ... + J(x', n−1))
(J(x', i) = <x', ((x'−1+i) mod n)+1>)

Figure 21: A data base system.

The scapegoat generator α is defined as follows: $\alpha(M', t')$ is defined iff the transition t' is disabled at the marking M'. Let a transition t be disabled at a marking M. Let $\ell(M, t, p_1, \ldots, p_j)$ denote the first element in a place list $p_1, \ldots, p_j$ that is a disabling place of t at M. Then

$$\alpha(M,t) = \begin{cases} \ell(M, t, \ \langle x' \rangle_{\text{inactive}}, \langle\rangle_{\text{exclusion}}, (J(x',1))_{\text{unused}}, \ldots, (J(x', n-1))_{\text{unused}}) \\ \qquad \text{if } t = \langle x' \rangle_{\text{update and send messages}}, \\ \ell(M, t, \ (J(x',1))_{\text{acknowledged}}, \ldots, (J(x', n-1))_{\text{acknowledged}}, \langle x' \rangle_{\text{waiting}}) \\ \qquad \text{if } t = \langle x' \rangle_{\text{receive acknowledgements}}, \\ \ell(M, t, \langle y' \rangle_{\text{inactive}}, \langle x', y' \rangle_{\text{sent}}) \qquad \text{if } t = \langle x', y' \rangle_{\text{receive message}}, \\ \ell(M, t, \langle y' \rangle_{\text{performing}}, \langle x', y' \rangle_{\text{received}}) \qquad \text{if } t = \langle x', y' \rangle_{\text{send acknowledgement}}. \end{cases}$$

Note that $\langle x', y' \rangle_{\text{receive message}}$ corresponds to the instance of the high-level transition "receive message" with $x = x'$ and $y = y'$ whereas $\langle x', y' \rangle_{\text{sent}}$ corresponds to the pair of the high-level place "sent" and tuple $\langle x', y' \rangle$. The other indexed tuples are analogous.

It can be shown that for each marking M that is reachable from the initial marking and for each transition t that is enabled at M, $E_{14}^*(\alpha, b, M, t)$ contains all transitions that are enabled at M. (In the proof, an exhaustive investigation of E_{14} is carried out manually. Formal parameters are used instead of concrete values.) This means that the incremental algorithm (optimized or not) has no reductive effect. The rotation from s to the next possible manager in the definition of $\alpha(M, \langle x' \rangle_{\text{receive acknowledgements}})$ is the actual pathological property of α. Experiments with PROD have indicated that this kind of stepping from a manager to another manager occurs very often when a pseudo-random scapegoat generator is used. As a result, pseudo-random generators tend to be almost as bad as α.

The pathological property of α can be eliminated by using a fixed manager when possible. Let $H(x', i)$ be $\langle x', i \rangle$ if $x' > i$, and $\langle x', i+1 \rangle$ if $x' \leq i$. The scapegoat generator β is defined as α with the following exception:

$$\beta(M,t) = \ell(M, t, (H(x',1))_{\text{acknowledged}}, \ldots, (H(x', n-1))_{\text{acknowledged}}, \langle x' \rangle_{\text{waiting}})$$

if $t = \langle x' \rangle_{\text{receive acknowledgements}}$ and t is disabled at M.

We conjecture that when the incremental algorithm (optimized or not) and β are used, the reduced reachability graph has $4(n-2)$ vertices and $8(n-2)$ edges more than the Λ-graph. (This conjecture has been obtained by letting PROD generate reduced reachability graphs for different values of n, and by inspecting the structure in each of the generated graphs.)

In the scapegoat generator γ, the places representing the phases of the managers have the highest priority. The scapegoat generator γ is defined as α with the following exception:

$$\gamma(M,t) = \ell(M, t, \langle x' \rangle_{\text{waiting}}, (J(x',1))_{\text{acknowledged}}, \ldots, (J(x', n-1))_{\text{acknowledged}})$$

if $t = \langle x' \rangle_{\text{receive acknowledgements}}$ and t is disabled at M.

The scapegoat generator δ has been got by considering that if a transition has a unique characteristic input place, then that place should have the highest priority.

The scapegoat generator δ is not pure in that sense but shows the sufficient interchange to transform α into an optimal scapegoat generator. The scapegoat generator δ is defined as α with the following exception:

$$\delta(M,t) = \ell(M,t,\langle x',y'\rangle_{\text{received}},\langle y'\rangle_{\text{performing}})$$

if $t = \langle x',y'\rangle_{\text{send acknowledgement}}$ and t is disabled at M.

In the same way as in the case of α, it can be shown that for each non-initial marking M that is reachable from the initial marking and for each transition t that is enabled at M, the set of enabled transitions in $E^*_{14}(\gamma,b,M,t)$ is $\{t\}$, and the set of enabled transitions in $E^*_{14}(\delta,b,M,t)$ is $\{t\}$. As a consequence, the incremental algorithm (optimized or not) produces the Λ-graph.

The above example suggests some heuristics for choosing a scapegoat. The scapegoat generator β suggests absolute ordering w.r.t. identity numbers, the scapegoat generator γ suggests giving a process control place the highest priority, and the scapegoat generator δ suggests giving a unique characteristic input place the highest priority. All these strategies are fixed order strategies in the sense that always the first possible alternative in a fixed list is chosen.

Some strategies work without knowledge of the modelled system. One of such strategies minimizes the number of enabled immediate successors of a vertex that are not in any maximal strongly connected component already found in the dependency graph. On the second priority level, it minimizes the number of all immediate successors of the vertex that are not in any maximal strongly connected component already found. On the third priority level, it minimizes the number of those immediate successors of the vertex that have not been visited yet.

A pseudo-random scapegoat generator is probably far from optimal if the incremental algorithm is as instable as in the above example. On the other hand, it is often useful to compare a given strategy to a pseudo-random strategy to see how good the strategy is. A pseudo-random strategy is a good measuring stick since it employs no knowledge of the system. If a strategy gives better (worse) results than a pseudo-random strategy, there must be something good (bad) in the strategy.

The deletion algorithm can also be used to estimate how good the incremental algorithm could be even though the deletion algorithm may sometimes compute a stubborn set containing more enabled transitions than a stubborn set computed by the incremental algorithm. The choice of an enabled transition to be deleted is a nondeterministic factor. In the case of the net in Figure 21, the nondeterminism associated with the deletion algorithm does not affect the number of enabled transitions in the computed stubborn set.

6.2 An incomplete minimization algorithm

Let us 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 a 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 22. 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 presented later in this section. The set returned by the function IncmplMin is the set of enabled transitions in the chosen stubborn set. (The type of the return values, **Set_of_transitions**, is assumed to have been defined appropriately.) 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.

```
/*1*/   Set_of_transitions IncmplMin(void /* no argument */) {
/*2*/     BasDelAlg() /* see Figure 16 */;
/*3*/     if ( the set of enabled white transitions contains all enabled
/*4*/          transitions or does not contain more than one enabled
/*5*/          transition ) return the set of enabled white transitions;
/*6*/     make a backup set from the set of enabled white transitions;
/*7*/     initialize a variable L to the size of the backup set;
/*8*/     if ( more than 5 transitions are enabled ) set L equal to 2;
/*9*/     initialize a variable i to 1;
/*10*/    while ( i < L ) {
/*11*/      for ( each subset T_e of enabled transitions of size i ) {
/*12*/        initialize the and/or-graph;
/*13*/        protect the transitions in T_e and make others unprotected;
/*14*/        Cnstr() /* see Figure 16 */;
/*15*/        if ( all enabled white transitions are protected )
/*16*/          { discard the existing backup set; return T_e; }
/*17*/        if ( L > the number of enabled white transitions ) {
/*18*/          discard the existing backup set;
/*19*/          make a backup set from the set of enabled white transitions;
/*20*/          set L equal to the size of the backup set;
/*21*/        } } add 1 to i; }
/*22*/    return the existing backup set; }
```

Figure 22: 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 $\max(\rho, 32)$ where ρ is the maximum number of enabled transitions at a marking.

The incomplete minimization algorithm can be extended to support the refined LTL-preserving stubborn set method of Chapter 5. However, we are more or less forced to make a compromise in what we try to minimize because in the algorithm in Figure 20, a preliminary stubborn set may have to be replaced by a stubborn set that contains all visible transitions. One feasible compromise is just to modify the algorithm in Figure 19 in such a way that we first try to minimize the number of enabled visible transitions if possible and then the number of enabled invisible transitions. (The algorithm in Figure 20 would not be modified in this compromise.)

We now consider three cases that have been studied with the aid of PROD. The first case is MULOG, a distributed mutual exclusion algorithm [26]. The second case is PFTP, a file transfer protocol [36]. The last case is YXA, a telephone protocol designed for educational purposes in Nokia Telecommunications Oy. The below statistics were obtained by running PROD in Linux on a 100 MHz Pentium with 64 Megabytes of RAM. The reduced reachability graphs that were under construction were kept in files.

n	algorithm	vertices	edges	user time	system time	elapsed time
3	IMA	1453	1476	14 s	1 s	15 s
3	DAA	1467	1490	15 s	1 s	16 s
3	IA1	4532	5534	12 s	2 s	15 s
3	IA2	4572	5590	12 s	2 s	14 s
4	IMA	15193	15534	295 s	7 s	317 s
4	DAA	15061	15402	310 s	7 s	338 s
4	IA1	77142	95980	380 s	58 s	568 s
4	IA2	77661	96721	383 s	56 s	525 s
5	IMA	174649	178584	6158 s	154 s	7139 s
5	DAA	172297	176232	6482 s	136 s	7140 s
5	IA1	1380020	1723416	12201 s	3063 s	70916 s
5	IA2	1388374	1734826	12426 s	3528 s	75648 s

Figure 23: Statistics of MULOG

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 24: The statistics of PFTP.

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 incomplete minimization algorithm (IMA for short), the deletion algorithm alone (DAA for short), and two versions of the incremental algorithm (IA1 and IA2 for short). IMA, DAA, IA1 and IA2 are actually on-the-fly verification algorithms close to the algorithms in [81]. Anyway, all these algorithms generate useful reduced reachability graphs completely when the property to be verified holds. In cases where all transitions are invisible and the property to be verified holds, IMA behaves exactly

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 25: The statistics of YXA.

like the algorithm in Figure 22 whereas DAA behaves exactly like the algorithm in Figure 16.

As described in Section 4.3, the incremental algorithm performs a search in a non-deterministically 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. All of the listed values for the user time, system time and elapsed time are raw numbers obtained from the computer in a single run of a batch program. We must admit that this is not the best possible way to report error-prone measurements. The batch program in question should be run tens of times under the same circumstances so that mean values and mean divergences could be reported. The author of this thesis has run that batch program a couple of times only but has still the opinion, based on some knowledge about Linux and other Unix-style operating systems, that the listed time values are reliable enough up to the precision of one digit.

In MULOG, there are n processes which communicate by sending messages. A message queue can have at most two messages at a time. There is exactly one visible transition, testing if more than one process could be in the critical section simultaneously. Each of IMA, DAA, IA1 and IA2 produces a reduced reachability graph where this transition does not occur. At most one process can thus be in the critical section at a time. Figure 23 shows the statistics of the MULOG case with different values of n. The relatively small but strange looking differences in user times between IMA and DAA are at least partially caused by the fact that the produced graphs are not equally easy to produce.

A detailed description of PFTP is given in Section 14.5 of [36]. 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. All transitions are invisible. Each of IMA, DAA, IA1 and IA2 result in 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 24 shows the statistics of PFTP.

As far as the PFTP case is concerned, the algorithms in PROD may seem less efficient than the algorithms in the SPIN tool [36, 37]. 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 [29, 56]. 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. All transitions are invisible. Again, each of IMA, DAA, IA1 and IA2 result in 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 25 shows the statistics of the YXA case with different values of k.

Note that even the statistics of MULOG and PFTP can be seen to be positive for the incomplete minimization algorithm since they indicate that the time taken by the algorithm to compute a stubborn set is not necessarily such a bottleneck in practice as could be feared on the basis of the worst case complexity. The number of enabled transitions of course matters but can, with a little thinking, often be kept small enough.

The design of the used models of MULOG, PFTP and YXA has supported rather DAA than IMA since if the orders of transitions used in deletions would have been mixed arbitrarily, there would most probably have been more branching in the reduced reachability graphs produced by DAA. No similar effort has been made to support IA1 or IA2 because, as can be concluded from Section 6.1, it is extremely difficult to support the incremental algorithm in nontrivial cases. Good ways for transition ordering in the deletion algorithm have been presented in [74].

6.3 Discussion

The incomplete minimization algorithm seems to be 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 [75] would suffice, let us consider a pathological case, the net in Figure 26. The only way to obtain a finite reduced reachability graph is to choose a set containing b and c for the stubborn set at some non-initial marking though $\{a\}$ would have less enabled transitions.

Actually, there is hardly no superior strategy [55, 75]. On the other hand, it is difficult to imagine a general heuristic that would try to minimize the size of the

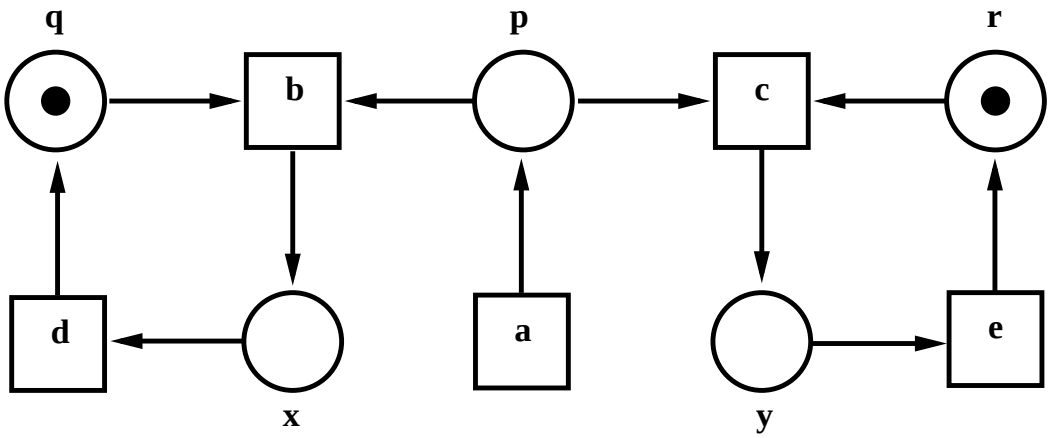


Figure 26: Minimization of branching is by no means guaranteed to result in a minimum-size reduced reachability graph.

reduced reachability graph without trying to minimize branching.

7 Removing redundancy from a stubbornly determined state space

Let us assume that we have a set of atomic propositions and we are to verify several nexttime-less LTL-formulas constructible from the atomic propositions. We say that a state of the system is *intermediate* iff exactly one transition is chosen to be executed at the state and the transition is invisible. In this chapter we show how almost all intermediate states can be eliminated during state space generation without affecting the result of the verification.

The elimination of intermediate states typically slows down state space generation since eliminated states can become generated many times. On the other hand, time is saved in the model-checking since the state space is smaller than the one that would have been obtained without the elimination. In the examples in this chapter, the elimination causes significant reduction in the size of the state space. Actually, the main motivation in the elimination of intermediate states is to save space resources.

7.1 Redefinitions

In order to proceed nicely, we rid ourselves from the requirement that the label of an edge in a reachability graph should be a single transition.

Definition 7.1 Let $\langle S, T, W, M_0\rangle$ be a place/transition net. The set T^* (respectively, T^∞) is called the *set of finite* (respectively, *infinite*) *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 state M to a state M'* iff $M[\sigma\rangle_f M'$, where

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

$$\forall M \in \mathcal{M}\ \forall M' \in \mathcal{M}\ \forall \delta \in T^+ \quad M[\delta\rangle_f M' \Leftrightarrow (\exists M'' \in \mathcal{M}\ \exists \gamma_1 \in T^*\ \exists \gamma_2 \in f(M'')\ \delta = \gamma_1\gamma_2 \wedge M[\gamma_1\rangle_f M'' \wedge M''[\gamma_2\rangle M').$$

A finite transition sequence σ is *f-enabled at a state M* ($M[\sigma\rangle_f$ for short) iff σ f-leads from M to some state. An infinite transition sequence σ is f-enabled at a state M ($M[\sigma\rangle_f$ for short) iff all finite prefixes of σ are f-enabled at M. A state M' is *f-reachable from a state M* iff some finite transition sequence f-leads from M to M'. A state M' is an *f-reachable state* 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 states, and the set of edges A is $\{\langle M, \sigma, M'\rangle \mid M \in V \wedge M' \in V \wedge \sigma \in f(M) \wedge M[\sigma\rangle_\Psi M'\}$. □

Again, 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. It is easy to see that when the target set of f consists of sequences of the length 1, Definition 7.1 makes no difference w.r.t. Definition 2.2.

Definition 2.3 suffices without modifications except that the function f should now be from $\mathcal{M}$ to $2^{(T^+)}$. Note that the definition of a cycle does not pay any attention to the labels of edges. We do not need further redefinitions since the remaining definitions in Chapter 2 do not need to be changed for the purposes of this chapter.

7.2 On the elimination of intermediate states

Proposition 7.2 together with Theorem 5.10 guide us in how intermediate states can be eliminated without affecting the result of a verification.

Proposition 7.2 *Let $\langle S, T, W, M_0 \rangle$ be a place/transition net. Let $\Upsilon \subseteq 2^T$. Let f be a function from $\mathcal{M}$ to 2^T and g a function from $\mathcal{M}$ to $2^{(T^+)}$ such that for any vertex M in the g-reachability graph and for any finite or infinite transition sequence, the sequence is g-enabled at M if (but not necessarily only if) the sequence is f-enabled at M.*

Claim: For each terminal path starting from M_0 in the f-reachability graph, there exists a terminal path starting from M_0 in the g-reachability graph in such a way that the labels of the paths are Υ-equivalent. For each infinite path starting from M_0 in the f-reachability graph, there exists an infinite path starting from M_0 in the g-reachability graph in such a way that the labels of the paths are Υ-equivalent.

Proof. The result follows trivially from the definitions. □

To guarantee that the g-reachability graph is totally reliable in verification, we thus only have to guarantee that the f-reachability graph is totally reliable and that no transition is enabled at those g-reachable states where no nonempty transition sequence is g-enabled. (The latter requirement is needed in order to avoid misleading terminal paths.)

Let us recall that a state encountered in the generation of a reachability graph is intermediate iff exactly one transition is chosen to be executed at the state and the transition is invisible. Note that being intermediate is a non-constant property since if we revisit an eliminated intermediate state, we may and sometimes even have to choose more than one transition to be executed. Moreover, even in the case that the state remains intermediate, the transition chosen is not necessarily the transition chosen during the previous visit.

We assume that intermediate states are eliminated on-the-fly in the context of an appropriate stubborn set based reachability graph generation algorithm. The rules of elimination are as follows.

- A finite acyclic path where all states except possibly the starting state and the ending state are intermediate can be replaced by an edge from the starting state to the ending state.
- An elementary cycle where at most one state is non-intermediate can be replaced by an edge from a state of the cycle to the state itself. If the cycle has a non-intermediate state, then that state is chosen. Otherwise any state of the cycle can be chosen.

In both of the above cases, the label of the replacing edge is defined to be the label of the eliminated path.

To ensure the assumptions of Proposition 7.2 and Theorem 5.10, we define the functions f (shared by the proposition and the theorem) and g as follows. (Note that the below presentation is purely proof theoretical, so we do not have to worry about the efficiency of the definition procedure.) For each encountered state M, if M has not remained intermediate throughout the construction, then $g(M)$ is the set of labels of the edges starting from M and resulting from the application of the above elimination rules. In the remaining states of $\mathcal{M}$ the value of g can be chosen arbitrarily. Let us then concentrate on the definition of f. For each $i \in N$, we define a function F_i from $\mathcal{M}$ to $2^{(2^T)}$ and a dynamically stubborn function f_i from $\mathcal{M}$ to 2^T. The definition of the latter follows from the definition of the former in such a way that $\forall i \in N\ \forall M \in \mathcal{M}\ f_i(M) = \cup_{Y \in F_i(M)} Y$.

We can assume that the construction of the g-reachability graph has been completed and that the graph is finite. For each nonterminal state M encountered during the construction, $F_0(M)$ is the collection of those stubborn sets that have been chosen for M during the visits to M. For each $M'' \in \mathcal{M}$ where $F_0(M'')$ does not become defined in this way, we let $F_0(M'') = \{T\}$. Since the g-reachability graph is finite, the f_0-reachability graph is finite, too. For each $i \in N$, we step from i to $i+1$ by considering an arbitrarily chosen problematic elementary cycle in the f_i-reachability graph as described below. If there is no such cycle, we finish the definition procedure by letting $f = f_i$ and for each $j > i$, $F_j = F_i$. Since the f_0-reachability graph is finite, there exists k such that the f_k-reachability graph has no problematic elementary cycle. The definition procedure thus takes a finite number of steps only.

The considered problematic cycle in the f_i-reachability graph is some elementary cycle that would make the ensuring of the assumptions of Proposition 7.2 impossible in the hypothetical case that the cycle would occur in the f-reachability graph. We choose a state M' from this cycle in such a way that a nonempty $F_{i+1}(M') \subseteq F_i(M')$ can be defined with the constraint that if the label of the edge starting from M' in the cycle is t, then $t \in T \setminus f_{i+1}(M')$ and either M' does not occur in the g-reachability graph or $\forall \sigma \in T^*\ \neg M'[t\sigma\rangle_g$. (This constraint can be satisfied because otherwise the cycle would not be problematic in the above specified way.) For each nonterminal state $M \neq M'$ in the f_i-reachability graph, we define $F_{i+1}(M) = F_i(M)$. For all other states $M'' \neq M'$ we let $F_{i+1}(M'') = \{T\}$.

7.3 Examples

We now look at three cases that have been analyzed with the aid of the PROD tool under the same circumstances as in Section 6.2. The cases are again MULOG, PFTP and YXA, and the used models are the same as the models used in Section 6.2.

Figure 27 shows 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 in two ways: the stubborn set method alone (SA for short) and combined with the elimination of intermediate states (ES for short). SA is actually the DAA of Section 6.2 whereas ES can be derived from SA in the way

described in Section 7.2.

The self-criticism presented in Section 6.2 and concerning the reporting of time values is applicable here, too, for similar reasons.

case	vertices	edges	user t.	system t.	elapsed t.
MULOG, $n = 4$, SA	15061	15402	292 s	6 s	298 s
MULOG, $n = 4$, ES	677	1018	385 s	4 s	390 s
MULOG, $n = 5$, SA	172297	176232	6058 s	105 s	6187 s
MULOG, $n = 5$, ES	7450	11385	8229 s	89 s	8553 s
PFTP, SA	5758	6615	949 s	9 s	987 s
PFTP, ES	858	1715	1375 s	11 s	1398 s
YXA, $k = 11$, SA	221935	237284	2475 s	135 s	2725 s
YXA, $k = 11$, ES	14212	29561	6752 s	142 s	6903 s
YXA, $k = 12$, SA	286974	305290	3361 s	198 s	3785 s
YXA, $k = 12$, ES	17021	35337	9446 s	214 s	9670 s
YXA, $k = 13$, SA	365224	386857	4264 s	270 s	4889 s
YXA, $k = 13$, ES	20170	41803	13430 s	282 s	13725 s

Figure 27: Statistics of case studies.

7.4 Discussion

It seems that intermediate states are quite usual when the stubborn set method is used for constructing a reduced state space. Consequently, from the elimination of intermediate states one can obtain substantial reduction in the size of the state space.

It should be emphasized that the elimination of intermediate states is not the same thing as *state space caching* [26, 27]. When we want to generate a reduced state space for a later use, state space caching does not help us since the caching technique intentionally forgets states that are not "needed at the moment".

Our approach differs from the approach in [53] in the following things:

- We can remove a state from a depth-first search stack without having to wait for the moment when the state would again be on the top of the stack. This is good since the state space explosion problem concerns search stacks, too. It is a common programming practice in depth-first search algorithms that a search stack is kept quickly accessible, e.g. in RAM, while at least some part of the rest of the data is allowed to be read from some more slowly accessible location, e.g. from a hard disk.
- We do not eliminate a state where the chosen stubborn set contains more than one enabled transition. Such elimination would be likely to cumulate in an explosion in the number of revisits to states.
- We do not estimate the probability of a revisit to a state. Let us quote [53]: "The preliminary DFS algorithm traverses the state space in a partial order manner, which is the exact same order used in the later reduction algorithm."

> It seems that this sentence has really been meant to be taken literally. The trick is that the preliminary algorithm is a state space caching algorithm.

Though we have not made experimental comparisons, it seems likely that the approach [53] is more efficient on average than our approach. There should still be certain kinds of applications where our approach is better.

Virtual coarsening of atomic actions, introduced by [2] and reviewed in [60], pp. 565–571, is a fast though also a somewhat limited way to eliminate intermediate states. As the name of the technique proposes, two or more actions are executed in a sequence without stopping in the middle. The method has a set of rules that decide what kind of actions can be combined in this sense.

In *covering step graph construction* [96, 97], redundant interleavings of actions are eliminated by choosing a set of independent transitions and executing them as a step, i.e. a virtually atomic sequence. This technique can thus be thought of as some kind of a stubborn set method combined with dynamic virtual coarsening of atomic actions.

Intermediate states sometimes reflect redundancy in a system description. *Net reductions* [5, 6] try to replace a net by a smaller net that is behaviourally close enough to the original net. Such reductions are still at most complementary to our approach, since e.g. many of the common ways to optimize the modelling of FIFO-queues [47] are disadvantageous for the stubborn set method. (The problem of modelling a FIFO-queue optimally for the stubborn set method is discussed in [73].)

8 On combining the stubborn set method with the sleep set method

This chapter considers the sleep set method [26] and is concentrated on the transition selection function when the method is applied to a labelled transition system to verify a basic termination property or a simple safety property. When used alone, i.e. with a transition selection function that always chooses all possible transitions, the sleep set method generates all reachable states though even then it can reduce the number of visits to the states. The conditions found in this chapter for the transition selection function can be used for combining the sleep set method with other analysis techniques and with the stubborn set method in particular. The reason for choosing labelled transition systems to be the underlying formalism instead of place/transition nets is that we want to generalize the compatibility results of [99] explicitly. The formalism used in [99] is more complicated than place/transition nets but can easily be thought of as a special kind of labelled transition systems.

8.1 Labelled transition systems

In this definition section we keep the notation as close as possible to the notation used in place/transition nets. Though there seems to be a tradition in the literature to associate labelled transition systems with some process algebraic framework, we shall not do that. The algorithmic considerations in this chapter do not force us to fix any specific framework.

Definition 8.1 A *labelled transition system* (an *LTS* for short) is a quadruple $\langle S, \Sigma, \Delta, s_0 \rangle$ such that S and Σ are sets, $\Delta \subseteq S \times \Sigma \times S$, and $s_0 \in S$. We call S the *set of states,* Σ the *set of actions,* Δ the *set of transitions,* and s_0 the *initial state.* The set $\Sigma \cup \Delta$ is called the *set of events of the LTS.* The function α from Δ to Σ is defined by

$$\forall s \in S\ \forall s' \in S\ \forall a \in \Sigma\ (\langle s, a, s' \rangle \in \Delta \Rightarrow \alpha(\langle s, a, s' \rangle) = a).$$

For any transition x, the action $\alpha(x)$ is called the *action of the transition* x. *An action* a *is fireable from a state* s *to a state* s' ($s[a\rangle s'$ for short) iff $\langle s, a, s' \rangle \in \Delta$. *A transition* x *is fireable from a state* s *to a state* s' ($s[x\rangle s'$ for short) iff $x = \langle s, \alpha(x), s' \rangle$. *An event* x *is enabled at a state* s iff x is fireable from s to some state. A state s is *terminal* iff no transition is enabled at s. □

We do not agree with the opinion expressed by [62] among others that labelled transition systems would not properly support the verification of state-based properties. The fact that we do not associate meanings with the states in the definition does not imply that the states could not be associated with meanings whatsoever. From the mathematical point of view, states can be assumed to exist before transitions are drawn between them.

Definition 8.2 Let $\langle S, \Sigma, \Delta, s_0 \rangle$ be an LTS. The set Δ^* is called the *set of finite transition sequences of the LTS,* and the set $(\Sigma \cup \Delta)^*$ is called the *set of finite event*

sequences of the LTS. A finite event sequence w is fireable from a state s to a state s' iff $s[w\rangle s'$ where

$$\begin{array}{l}\forall s \in S\ s[\varepsilon\rangle s, \text{ and}\\ \forall s \in S\ \forall s' \in S\ \forall v \in (\Sigma \cup \Delta)^*\ \forall x \in \Sigma \cup \Delta\\ s[vx\rangle s' \Leftrightarrow (\exists s'' \in S\ s[v\rangle s'' \wedge s''[x\rangle s').\end{array}$$

A finite event sequence w is enabled at a state s ($s[w\rangle$ for short) iff w is fireable from s to some state. A state s' is *reachable from a state s by a finite event sequence w* iff w is fireable from s to s'. A state s' is *reachable from a state s* iff some finite transition sequence is fireable from s to s'. A state s' is a *reachable state* iff s' is reachable from s_0. Let f be a function from S to 2^Δ. *A finite transition sequence w is f-fireable from a state s to a state s'* iff $s[w\rangle_f s'$, where

$$\begin{array}{l}\forall s \in S\ s[\varepsilon\rangle_f s, \text{ and}\\ \forall s \in S\ \forall s' \in S\ \forall v \in \Delta^*\ \forall x \in \Delta\\ s[vx\rangle_f s' \Leftrightarrow (\exists s'' \in S\ s[v\rangle_f s'' \wedge x \in f(s'') \wedge s''[x\rangle s').\end{array}$$

A finite transition sequence w is f-enabled at a state s ($s[w\rangle_f$ for short) iff w is f-fireable from s to some state. □

Definition 8.3 Let $\langle S, \Sigma, \Delta, s_0\rangle$ be an LTS. The set Δ^∞ is called the *set of infinite transition sequences of the LTS,* and the set $(\Sigma \cup \Delta)^\infty$ is called the *set of infinite event sequences of the LTS. An infinite event sequence w is enabled at a state s* ($s[w\rangle$ for short) iff there exists a function ξ from N to S in such a way that $\xi(0) = s$ and for each $i \in N$, $\xi(i)[w(i)\rangle\xi(i+1)$. The function Ω from S to 2^{Δ^∞} is defined by requiring that for each state s, $\Omega(s)$ is the set of those infinite transition sequences that are enabled at s. Let f be a function from S to 2^Δ. *An infinite event sequence w is f-enabled at a state s* ($s[w\rangle_f$ for short) iff there exists a function ξ from N to S in such a way that $\xi(0) = s$ and for each $i \in N$, $\xi(i)[w(i)\rangle_f\xi(i+1)$. We say that f is *tough-lived* iff for each reachable state s,

$$\Omega(s) \neq \emptyset \Rightarrow (\exists w \in \Omega(s)\ s[w\rangle_f).\ \square$$

Since any transition by definition has a unique target state, the following holds: if all finite prefixes of an infinite transition sequence w are enabled (respectively, f-enabled) at a state s, then w itself is enabled (respectively, f-enabled) at s. The same does not hold for an infinite event sequence, but that is no problem since in the rest of this chapter, all considered infinite event sequences are explicitly required to be transition sequences.

Definition 8.4 Let $\langle S, \Sigma, \Delta, s_0\rangle$ be an LTS. A transition sequence v is an *alternative sequence of a finite transition sequence w from a state s to a state s'* iff v is a finite transition sequence, $s[w\rangle s'$, and $s[v\rangle s'$. A transition sequence v is a *length-secure alternative sequence of a finite transition sequence w from a state s to a state s'* iff v is an alternative sequence of w from s to s' and not longer than w. The function ϑ from $\Delta^* \times S \times S$ to $2^{(\Delta^*)}$ is defined by requiring that for each finite transition sequence w, and for each state s and s', $\vartheta(w, s, s')$ is the *set of length-secure alternative sequences of w from s to s'*. Let ψ be a truth-valued function on S. A state s is a *ψ-state* iff

$\psi(s)$ is true. Let f be a function from S to 2^{Δ}. We say that f *represents all sets of alternative sequences to ψ-states* iff for each reachable state s and for each ψ-state s',

$$\forall w \in \Delta^* \; s[w\rangle s' \Rightarrow (\exists v \in \Delta^* \; s[v\rangle_f s').$$

Correspondingly, f *represents all sets of length-secure alternative sequences to ψ-states* iff for each reachable state s and for each ψ-state s',

$$\forall w \in \Delta^* \; s[w\rangle s' \Rightarrow (\exists v \in \vartheta(w, s, s') \; s[v\rangle_f). \quad \square$$

Clearly, a function representing all sets of length-secure alternative sequences to ψ-states represents all sets of alternative sequences to ψ-states. We say that a function f from S to 2^{Δ} *represents all sets of (length-secure) alternative sequences to terminal states* iff f represents all sets of (length-secure) alternative sequences to ψ-states in the case where ψ is the characteristic function of the set of terminal states.

Though it might be interesting to define dynamic stubbornness for labelled transition systems, we instead present the following axioms. This set of axioms is intended to be sound but not complete. In other words, any concrete definition of stubbornness should be consistent with these axioms, but the axioms do not express any sufficient condition for stubbornness.

(X1) Stubbornness is a property of a set of transitions and is defined w.r.t. a state.

(X2) A function f from S to 2^{Δ} is stubborn iff for each nonterminal reachable state s, $f(s)$ is stubborn at s.

(X3) Every stubborn function represents all sets of length-secure alternative sequences to terminal states.

(X4) Every stubborn function is tough-lived.

The full reachability graph of a place/transition net can be thought of as an LTS where transitions in the net sense are actions. More generally, the complete state space of a system modelled in any formalism can be thought of as an LTS. We can thus talk about the *LTS of a system.* We can extend a formalism-specific definition of stubbornness to concern the LTS of a system by defining that for any state s in the LTS, a subset Γ of transitions of the LTS is stubborn at s iff there exists a set B (a subset of an appropriate domain) in such a way that B is stubborn at s according to the formalism-specific definition, and

$$\Gamma = \{\langle s, c, s'\rangle \quad | \quad \langle s, c, s'\rangle \text{ is a transition that represents some execution of some member of } B\}.$$

To our knowledge, each definition of stubbornness, persistence or dynamic stubbornness found in the literature is such that if "persistent" or "dynamically stubborn" is replaced by "stubborn" and the definition is then extended to concern the LTS of a system in the above described way, the above axioms X1–X4 become satisfied.

8.2 The sleep set method

In this section we present the sleep set method. We concentrate on a generalized version of the terminal state detection algorithm [99]. The generalized version is in Figure 28. The intuitive idea of the algorithm is to eliminate such redundant transition sequences that are not eliminated by the transition selection function f. The LTS $\langle S, \Sigma, \Delta, s_0 \rangle$ is assumed to be such that $S \cup \Sigma$ is finite. The algorithm computes an LTS $\langle S, \Sigma, \nabla, s_0 \rangle$ such that $\nabla \subseteq \Delta$. (All states handled by the algorithm are reachable w.r.t. the input LTS. Also, the states occurring in ∇ are reachable w.r.t. the output LTS. We can still define both of the LTS's to have the same set of states since the algorithm concretely constructs only the set ∇.) From the finiteness of $S \cup \Sigma$ and from the fact that the set Act constructed during one visit to a state does not intersect with the sets Act constructed during the other visits to the state it follows that the execution of the algorithm takes a finite time only.

The function ψ can be any truth-valued function on S. To be practical, we can assume that $\psi(s)$ is computed without inspecting any other state than s. The construction of ∇ can be omitted if only the detection of reachable ψ-states is of interest.

The "code" of our algorithm is very close to the "code" of the algorithm in [99]. More precisely, the way in which we handle the sleep sets is effectively the same way in which [99] handles them. The easiest way to approach our algorithm is first to imagine that all actions are deterministic (i.e. no action can lead from a state to more than one state), and then to proceed to the general case. Namely, the virtual complexity of the presentation is almost solely caused by the fact that the actions are not necessarily deterministic.

Lines /*5*/, /*10*/ and /*13*/ and /*18*/ need the following explanation:
$H \subseteq S \times (2^{\Sigma})$, but we say that a state s_1 is in H iff s_1 is the state component of some (actual) element of H, i.e. $\exists \Sigma_1 \subseteq \Sigma \;\; \langle s_1, \Sigma_1 \rangle \in H$. The expression "the set associated with s_1 in H" assumes that only one element of H has s_1 as the state component. "Substituting Y for the set associated with s_1 in H" corresponds to a pseudo-statement "$H = (H \setminus \{\langle s_1, X \rangle\}) \cup \{\langle s_1, Y \rangle\}$" where $\langle s_1, X \rangle$ is the only element of H that has s_1 as the state component. A naive induction suffices for showing that in H, no state can be the state component of more than one element.

We use actions much in the same way as *program transitions* are used in the terminal state detection algorithm in [99]. One might think that our approach is thus more coarse than the approach used in [99]. However, if we have a *global LTS* of the form defined in [99], we can relabel each *global transition* [99] by the program transition of the global transition, and apply our algorithm to the resulting LTS. In practice, a global transition can be relabelled when used for the first time, whereas the unused global transitions need no relabelling.

Theorem 8.5 *Let $\langle S, \Sigma, \Delta, s_0 \rangle$ be an LTS such that $S \cup \Sigma$ is finite. Let ψ be a truth-valued function on S. Let f be a function from S to 2^{Δ} such that f represents all sets of length-secure alternative sequences to ψ-states. Then the algorithm in Figure 28 finds all reachable ψ-states.*

Proof. Let s_d be a reachable ψ-state.

```
/*1*/   make Stack empty; make H empty; ∇ = ∅;
/*2*/   push ⟨s0, ∅⟩ onto Stack;
/*3*/   while ( Stack is not empty ) {
/*4*/       pop ⟨s, Sleep⟩ from Stack;
/*5*/       if ( s is not in H ) {
/*6*/           Trans = {x ∈ f(s) | s[x⟩ ∧ α(x) ∈ Σ\ Sleep };
/*7*/           Act = {a ∈ Σ | ∃x ∈ Trans α(x) = a};
/*8*/           Succ = {⟨a, S'⟩ | a ∈ Act ∧S' = {s' ∈ S | ⟨s, a, s'⟩ ∈ Trans }};
/*9*/           if ( ψ(s) is true ) printf("ψ-state!");
/*10*/          enter ⟨s, a copy of Sleep⟩ in H;
/*11*/      }
/*12*/      else {
/*13*/          let hSleep be the set associated with s in H;
/*15*/          Act = hSleep \ Sleep;
/*16*/          Succ = {⟨a, S'⟩ | a ∈ Act ∧S' = {s' ∈ S | s[a⟩s'}};
/*17*/          Sleep = hSleep ∩ Sleep;
/*18*/          substitute a copy of Sleep for the set associated with s in H;
/*19*/      }
/*20*/      newSleep = ∅;
/*21*/      for ( each a in Act ) {
/*22*/          let S' be the set for which ⟨a, S'⟩ ∈ Succ;
/*23*/          for ( each s' in S' ) {
/*24*/              xSleep = {a' ∈ Sleep | s'[a'⟩ ∧ (∀s'' ∈ S  s'[a'⟩s'' ⇒ s[a'a⟩s'')}∪
/*25*/                       {a1 ∈ newSleep | ∃S1 ⟨a1, S1⟩ ∈ Succ ∧s'[a1⟩∧
/*26*/                                                (∀s'' ∈ S  s'[a1⟩s'' ⇒
/*27*/                                                    (∃s1 ∈ S1  s1[a⟩s''))};
/*28*/              push ⟨s', a copy of xSleep⟩ onto Stack;
/*29*/              ∇ = {⟨s, a, s'⟩} ∪ ∇;
/*30*/          }
/*31*/          newSleep = {a}∪ newSleep;
/*32*/      }
/*33*/  }
```

Figure 28: An LTS reduction and a ψ-state detection algorithm.

(i) We first prove that if $X \subseteq \Sigma$, a finite transition sequence w is fireable from a state s to s_d, and for each v in $\vartheta(w, s, s_d)$, the first action of v is not in X, then, if $\langle s, X\rangle$ is pushed onto the stack, some element having s_d as the first component will be or has already been popped from the stack. By ϑ we mean the ϑ of the LTS $\langle S, \Sigma, \Delta, s_0\rangle$. By the first action of a transition sequence we mean the action of the first transition of the sequence.

The proof proceeds by induction on the length of w. For $w = \varepsilon$, the result is immediate. Now, assume the proposition holds for finite transition sequences of length less than or equal to n, where $n \geq 0$, and let us prove that it holds for a finite transition sequence w of length $n + 1$. Let X be a subset of Σ, w be fireable from a a state s to s_d, and $\langle s, X\rangle$ have been pushed onto the stack. Let it also be the case that for each v in $\vartheta(w, s, s_d)$, the first action of v is not in X. Let us consider the steps immediately following the popping of $\langle s, X\rangle$ from the stack. If $s = s_d$, the element $\langle s_d, X\rangle$ has then been popped from the stack. From now on, we assume that $s \neq s_d$.

We first consider the case where s is not already in H. Since $s[w\rangle s_d$, $s \neq s_d$, and f represents all sets of length-secure alternative sequences to ψ-states, at least one transition in $f(s)$ is the first transition of some sequence in $\vartheta(w, s, s_d)$. Moreover, the action of such a transition is in $\Sigma \setminus X$, so all of such transitions are fired at s. (Firing of transitions takes place on line /*29*/ only.) Let x_1 be the first of such transitions in the firing order. Then there exists a finite transition sequence w' such that $x_1 w'$ is in $\vartheta(w, s, s_d)$. From the definition of ϑ it follows that $s[x_1 w'\rangle s_d$ and $x_1 w'$ is not longer than w. The length of w' is thus less than or equal to n. Let x_1 be fireable from s to a state s'. Then $s'[w'\rangle s_d$. Let $\langle s', X'\rangle$ be pushed onto the stack in the algorithmic step that immediately precedes the firing of x_1 from s to s'. We show that for each v in $\vartheta(w', s', s_d)$, the first action of v is not in X'.

Indeed, assume the opposite, i.e., there exists some transition x' such that $\alpha(x')$ is in X', and for some finite transition sequence v', $x'v'$ is in $\vartheta(w', s', s_d)$. Clearly, then $x_1 x' v'$ is in $\vartheta(w, s, s_d)$. If $\alpha(x')$ is in Sleep during the execution of the outermost "for-loop" (lines /*21*/–/*32*/), then (due to line /*24*/) every state reachable from s' by $\alpha(x')$ is reachable from s by $\alpha(x')\alpha(x_1)$, so there exist transitions x_2 and x'' such that $\alpha(x_2) = \alpha(x')$, $\alpha(x'') = \alpha(x_1)$, and $x_2 x'' v'$ is in $\vartheta(w, s, s_d)$. From the condition satisfied by X it then follows that $\alpha(x_2)$ is not in X, a contradiction with the assumption that $\alpha(x') = \alpha(x_2)$ is in Sleep $= X$. The action $\alpha(x')$ thus cannot be in Sleep during the execution of the outermost "for-loop". This means that $\alpha(x')$ has been inserted into newSleep in the outermost "for-loop" before firing x_1. Moreover (due to lines /*25*/–/*27*/), every state reachable from s' by $\alpha(x')$ is reachable from some $s_1 \in S_1$ by $\alpha(x_1)$ where S_1 is the set associated with $\alpha(x')$ in Succ. (Due to lines /*8*/ and /*16*/, S_1 is unique under circumstances whatsoever.) Consequently, there exist transitions x_2 and x'' such that $\alpha(x_2) = \alpha(x')$, $\alpha(x'') = \alpha(x_1)$, $x_2 x'' v'$ is in $\vartheta(w, s, s_d)$, and x_2 is either x_1 itself or fired after x_1. The action $\alpha(x') = \alpha(x_2)$ is thus not in newSleep at the time when x_1 is fired. This is a contradiction. The inductive hypothesis can thus be used to establish that some element having s_d as the first component will be or has already been popped from the stack.

We now consider the case where s already appears in H. Let $Y \subseteq \Sigma$ be such that $\langle s, Y\rangle$ is in H. All those transitions that are enabled at s and have their actions in $Y \setminus X$ are fired. There are two situations: either some action in Y is the first action of some sequence in $\vartheta(w, s, s_d)$, or no such action exists. In the first situation, we

can choose a transition analogous to the above x_1 and proceed as above.

Let us now turn to the second situation in which no action in Y is the first action of any sequence in $\vartheta(w, s, s_d)$. This can be the case either because no action in Y_0 is the first action of any sequence in $\vartheta(w, s, s_d)$ where Y_0 is the sleep set entered in H with s when s was inserted into H, or because there are some Y' and Z such that $\langle s, Z\rangle$ was popped from the stack before popping $\langle s, X\rangle$ from the stack, $\langle s, Y'\rangle$ was in H at the time of the popping of $\langle s, Z\rangle$ from the stack, some action in Y' is the first action of some sequence in $\vartheta(w, s, s_d)$, and no action in $Y' \cap Z$ is the first action of any sequence in $\vartheta(w, s, s_d)$. In the former case, we can proceed as above with Y_0 in the place of X. In the latter case, we can proceed as above with Z in the place of X, taking into account the fact that Sleep $= Y' \cap Z$ during the execution of the outermost "for-loop".

(ii) The algorithm in Figure 28 starts by pushing $\langle s_0, \emptyset\rangle$ onto an empty stack. From the result shown in part (i) it thus follows that some element having s_d as the first component will be popped from the stack. □

The proof of Theorem 8.5 resembles much the proof of Theorem 6 in [99]. However, the true difficulty in Theorem 8.5 is more in the formulation of the claim than in the proof itself.

From axiom X3 it follows that any stubborn function satisfies the conditions required from the transition selection function f when ψ is the characteristic function of terminal states. The sleep set method can thus be combined with the stubborn set method in the detection of reachable terminal states without any assumption on the stubborn sets used. This is clearly more than has been shown in [99] because the stubborn sets used in [99] are persistent, and by Lemma 4.14 we know that in place/transition nets, persistent sets are strongly dynamically stubborn.

Theorem 8.5 has also the consequence that if for each encountered state, the transition selection function f chooses all enabled transitions, then the algorithm in Figure 28 visits all reachable states. (To see this, one can define the value of ψ to be true at every state.)

Perhaps the most interesting thing in Theorem 8.5 is that it gives us a way to verify simple safety properties on-the-fly. If neither livelock monitor states nor infinite progress monitor states exist, the stubborn functions of [81] represent all sets of length-secure alternative sequences to reject states. (Without loss in generality we can assume that every reject state has a stubborn set that satisfies the same conditions as the stubborn sets of the ordinary states.) We then get a correct on-the-fly verification algorithm from the algorithm in Figure 28 simply by defining f to be a stubborn function of the kind used in [81], by defining ψ to be the characteristic function of reject states and terminal deadlock monitor states and by associating an automatic exit with the printing function.

We now turn to the checking of the possible existence of an infinite enabled transition sequence.

Theorem 8.6 *Let $\langle S, \Sigma, \Delta, s_0\rangle$ be an LTS such that $S \cup \Sigma$ is finite. Let f be a tough-lived function from S to 2^{Δ}. If no infinite transition sequence is enabled at s_0 in the*

LTS $\langle S, \Sigma, \nabla, s_0 \rangle$ *at the end of the execution of the algorithm in Figure 28, then no infinite transition sequence is enabled at* s_0 *in the LTS* $\langle S, \Sigma, \Delta, s_0 \rangle$.

Proof. (i) We first prove that if $X \subseteq \Sigma$, $s \in S$, $\Omega(s) \neq \emptyset$, for each v in $\Omega(s)$, $\alpha(v(0))$ is not in X, and $\langle s, X \rangle$ is pushed onto the stack, then there is a transition x, a set $X' \subseteq \Sigma$ and a state s' such that x will be or has already been fired from s to s', $\langle s', X' \rangle$ will be or has already been pushed onto the stack, $\Omega(s') \neq \emptyset$, and for each v in $\Omega(s')$, $\alpha(v(0))$ is not in X'. By Ω we mean the Ω of the LTS $\langle S, \Sigma, \Delta, s_0 \rangle$. (Firing of transitions takes place on line /*29*/ only.)

Let X be a subset of Σ, s be a state such that $\Omega(s) \neq \emptyset$, and $\langle s, X \rangle$ have been pushed onto the stack. Let it also be the case that for each v in $\Omega(s)$, $\alpha(v(0))$ is not in X. Let us consider the steps immediately following the popping of $\langle s, X \rangle$ from the stack.

We first consider the case where s is not already in H. Since $\Omega(s) \neq \emptyset$ and f is tough-lived, at least one transition in $f(s)$ is the first transition of some sequence in $\Omega(s)$. Moreover, the action of such transition is in $\Sigma \setminus X$, so all of such transitions are fired at s. Let x_1 be the first of such transitions in the firing order. Let x_1 be fireable from s to a state s'. Clearly, then $\Omega(s') \neq \emptyset$. Let $\langle s', X' \rangle$ be pushed onto the stack in the algorithmic step that immediately precedes the firing of x_1 from s to s'. We show that for each v in $\Omega(s')$, $\alpha(v(0))$ is not in X'.

Indeed, assume the opposite, i.e., there exists some $v' \in \Omega(s')$ such that $\alpha(v'(0))$ is in X'. If $\alpha(v'(0))$ is in Sleep during the execution of the outermost "for-loop" (lines /*21*/–/*32*/), then (due to line /*24*/) every state reachable from s' by $\alpha(v'(0))$ is reachable from s by $\alpha(v'(0))\alpha(x_1)$, so there exists an infinite transition sequence w such that w is in $\Omega(s)$, and $\alpha(w(0)) = \alpha(v'(0))$. From the condition satisfied by X it then follows that $\alpha(w(0))$ is not in X, a contradiction with the assumption that $\alpha(v'(0)) = \alpha(w(0))$ is in Sleep$= X$. The action $\alpha(v'(0))$ thus cannot be in Sleep during the execution of the outermost "for-loop". This means that $\alpha(v'(0))$ has been inserted into newSleep in the outermost "for-loop" before firing x_1. Moreover (due to lines /*25*/–/*27*/), every state reachable from s' by $\alpha(v'(0))$ is reachable from some $s_1 \in S_1$ by $\alpha(x_1)$ where S_1 is the set associated with $\alpha(v'(0))$ in Succ. (Due to lines /*8*/ and /*16*/, S_1 is unique under circumstances whatsoever.) Consequently, there exists an infinite transition sequence w such that w is in $\Omega(s)$, $\alpha(w(0)) = \alpha(v'(0))$, and $w(0)$ is either x_1 itself or fired after x_1. The action $\alpha(v'(0)) = \alpha(w(0))$ is thus not in newSleep at the time when x_1 is fired. This is a contradiction.

The case where s already appears in H can be handled by repeating the corresponding part of the proof of Theorem 8.5 with the exception that the expression "$\Omega(s)$" is in the place of the expression "$\vartheta(w, s, s_d)$".

(ii) The algorithm in Figure 28 starts by pushing $\langle s_0, \emptyset \rangle$ onto an empty stack. If $\Omega(s_0) \neq \emptyset$, using the result shown in part (i) we can construct an infinite transition sequence which is enabled at s_0 in the LTS $\langle S, \Sigma, \nabla, s_0 \rangle$ at the end of the execution of the algorithm. □

Even the proof of Theorem 8.6 resembles much the proof of Theorem 6 in [99], but again, the true difficulty is more in the formulation of the claim than in the proof itself.

From Theorem 8.6 it follows that we can detect the possible existence of enabled infinite transition sequences from the reduced LTS. Alternatively, one can add on-the-fly loop detection to the algorithm in Figure 28 in such a way that the first loop of states found terminates the execution of the algorithm. Then there is no need to construct ∇ since the construction of ∇ affects neither the set of visited states nor the order of visiting.

From axiom X4 it follows that any stubborn function satisfies the conditions required from the transition selection function f. The stubborn set method and the sleep set method can thus be combined in the detection of non-termination without any assumption on the stubborn sets used.

Let us consider the complexity of the algorithm in Figure 28. The cumulative time per state spent in the outermost "for-loop" is at most proportional to $\mu^4\rho^2$, where μ is the maximum number of states reachable from a state by an action, ρ is the maximum number of enabled actions of a state, and all visits to the state are counted. This is based on the fact that each sleep set associated with a state contains only actions that are enabled at the state. The time per visit to a state spent in the operations related to H is the time of the search for the state plus a time that is at most proportional to ρ. The searches in H are something that cannot be avoided easily whether or not we use sleep sets at all. Clearly, the time taken by the computation of $f(s)$ and $\psi(s)$ can be anything depending on f and ψ. If ψ is the characteristic function of the set of terminal states, then the expression "if $\psi(s)$ is true" in the algorithm in Figure 28 can be replaced by the expression "if Act and Sleep are both empty". It depends much on the LTS how many times a state is visited and how many simultaneous occurrences of a state there are in the stack. One stack element requires space for the state and at most ρ actions. It is not necessary to store copies of states and actions since pointers suffice.

In order to avoid ambiguity in the sequel in this section, transitions in the place/transition net sense will be called net transitions. In the following examples, stubbornness and dynamic stubbornness should be understood as defined in Chapter 4.

Let us consider an example which shows that the statement obtained from Theorem 8.5 by removing the word "length-secure" is not valid. Let $M_0[a\rangle M_1$, $M_1[d\rangle M_2$, and $M_2[bc\rangle M_3$ in the net in Figure 29. Let a function h from $\mathcal{M}$ to 2^T be defined by $h(M_0) = \{a\}$ and $h(M) = T$ when $M \neq M_0$. Let then
$f(M) = \{\langle M, t, M'\rangle \mid M[t\rangle_h M'\}$ at each state M. The state M_3 is the only reachable terminal state. The function f thus represents all sets of alternative sequences to terminal states. However, f does not represent all sets of length-secure alternative sequences to terminal states since $M_0[cd\rangle M_3$ while there is no sequence of net transitions that would h-lead from M to M_3 and be of length less than or equal to 2.

During the first visit to M_0, the algorithm in Figure 28 inserts $\langle M_0, \emptyset\rangle$ into H and pushes $\langle M_1, \emptyset\rangle$ onto the stack. The algorithm then visits M_1. The net transitions b and d are the enabled net transitions in $h(M_1) = T$ at M_1. Let b be fired before d at M_1. The algorithm pushes $\langle M_0, \emptyset\rangle$ and $\langle M_2, \{b\}\rangle$ onto the stack since b and d commute at M_1. The algorithm then visits M_2 but does not fire the sleeping b which is the only enabled net transition at M_2. No net transition is fired during the second visit to M_0 since the sleep set associated with M_0 in H is empty. The execution of the algorithm is then over. No terminal state was found though M_3 is a reachable

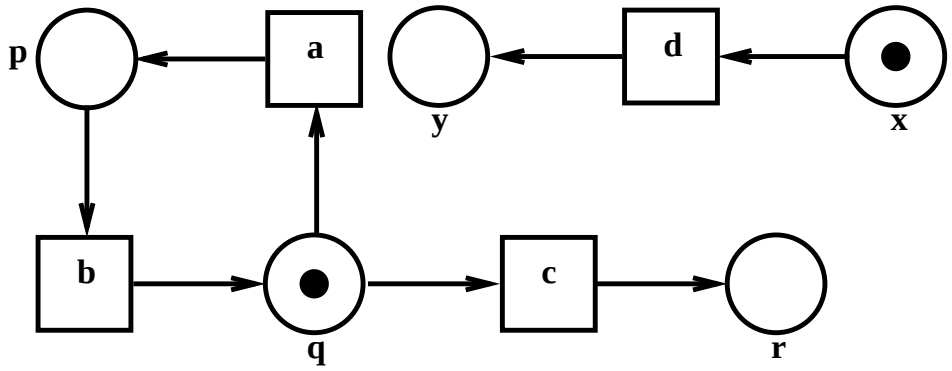


Figure 29: A deceptive f makes the algorithm in Figure 28 fail.

terminal state.

The combination of the sleep set method and the stubborn set method can really be better than the plain stubborn set method as far as the number of inspected states is concerned. The net in Figure 30 is a simple example showing this. The example is essentially the same as can be found in [100].

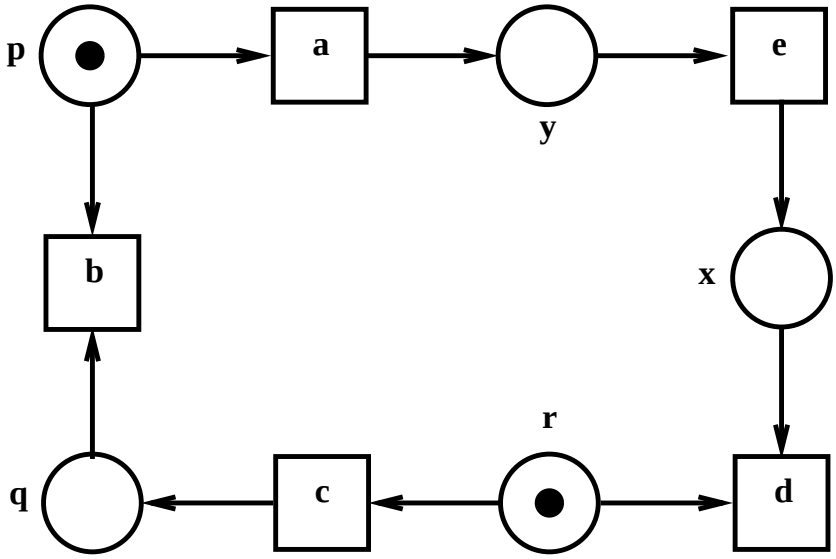


Figure 30: A net showing some of the power of the algorithm in Figure 28.

An exhaustive investigation shows that at each reachable nonterminal state of this net, there is one and only one dynamically stubborn set that is minimal w.r.t. set inclusion. By Lemma 4.10 we know that a dynamically stubborn set that is minimal w.r.t. set inclusion only contains enabled net transitions. Another exhaustive investigation shows that at each reachable nonterminal state of this net, the set of enabled net transitions of any stubborn set computed by the incremental algorithm or the deletion algorithm, using any of the definitions of stubbornness in [75, 77, 79], is a dynamically stubborn set that is minimal w.r.t. set inclusion. Let h be a dynamically stubborn function such that at any nonterminal state M, $h(M)$ is a stubborn set computed by the incremental algorithm. Then the set of enabled net transitions

in $h(M)$ is the only dynamically stubborn set at M that is minimal w.r.t. set inclusion. Thus, for each dynamically stubborn function g, the h-reachability graph is a subgraph of the g-reachability graph.

Let then $f(M) = \{\langle M, t, M' \rangle \mid M[t\rangle_h M'\}$ at each state M. We have $h(M_0) = \{a, c\}$. Let a be fired before c at M_0 in the algorithm in Figure 28. Let $M_0[c\rangle M'$. Since a and c commute at M_0 and a is fired before c, $\langle M', \{a\}\rangle$ is pushed onto the stack. Let us consider the visit to M' where $\langle M', \{a\}\rangle$ is popped from the stack. We have $h(M') = \{a, b\}$, but the sleeping a is not fired. Let $M'[a\rangle M''$. By executing the algorithm in Figure 28 completely, we see that M'' is never encountered, and no nonterminal state is visited more than once. The latter observation is important since it guarantees that all net transitions that are fired at a state M are in $h(M)$. The set of inspected states is thus a proper subset of the states of the h-reachability graph.

8.3 Discussion

A combination of the stubborn set method and the sleep set method in the verification of nexttime-less LTL-formulas has essentially been presented in e.g. [55]. From [26, 55] one can conclude that the persistent set method and the ample set method are compatible with the sleep set method in the verification of nexttime-less LTL-formulas, with the explicitly mentioned limitations. These limitations can basically be described by saying that we must have a fairness assumption of a certain kind. Nevertheless, what still remains to be studied is the compatibility of the stubborn set method with the sleep set method in the verification of nexttime-less LTL-formulas when fairness is not assumed. This does not seem an easy problem because to our knowledge, the problem has not even been solved in the case where the stubborn set always contains all enabled transitions. The limits of the applicability of the sleep set method should anyway be known better.

We have not heard of any experience where the use of sleep sets would have significantly reduced the number of visited states. (This has been tried with PROD, too, without success.) Instead, the statistics in [26] clearly indicate that the number of visits to the states can become reduced so much that the use of sleep sets pays off. This is obvious in state space caching where every avoided revisit to a state is important. The same holds for the elimination of intermediate states.

As pointed out in [49, 56, 80], some early publications about the sleep set method (at least [25] and [38] as well as the early version of [27] in [8], pp. 178–191, and the early version of [30] in [61], pp. 406–415) are flawed to some extent. However, as far as we know, most if not all of the flaws have been eliminated in later publications by the same authors [26, 27, 30, 31, 99].

9 Other remarks on stubborn sets

9.1 Stubborn sets of high-level nets

Place/transition nets of actual systems tend to be very large. On the other hand, using *high-level nets* [9, 41] one can make compact models in a natural way. Fortunately, a high-level net can often be *unfolded* into a behaviourally equivalent finite place/transition net, and, using the inverse mapping of the unfolding mapping, the place/transition net can be folded back into the high-level net [22, 40]. If such an unfolding exists, one can apply the stubborn set method to the result of the unfolding and then fold the reduced reachability graph. However, even a high-level net, the set of reachable markings of which is finite, may be difficult to unfold since the unfolding procedure usually needs explicit bounds on the possible transition instances. If suitable bounds are not known, they have to be estimated.

Unfolding can be implicit in the sense that no place/transition net is constructed but the transition instances of the high-level net are used just like transitions of a place/transition net. Implicit unfolding supports determining the enabled transition instances by unifying the arc expressions with the current marking.

In [79], the stubborn set method is applied to *coloured Petri nets* [41], mostly by means of implicit unfolding. The possibility to ignore the colour information is mentioned though not especially recommended [79]: "Some preliminary experiments have demonstrated that ignoring the colour information usually leads to grossly unnecessarily large stubborn sets." This kind of ignoring means that a high-level net is treated as if it were a low-level net with the same structure.

More sophisticated manipulation of stubborn sets is presented in [11]. The nets in question are *well formed coloured Petri nets*. General and well formed couloured Petri nets do not have much effective difference in description power since in practical analysis problems, we often do not have to express more complicated integer arithmetic operations than addition by one. The computation of stubborn sets in [11] is based on symbolic matrix calculations. The technique still does not work completely on the high level. Some of the transition instances, even some of those that are disabled, may have to be accessed one at a time, and in extreme cases the technique behaves in the same way as plain implicit unfolding.

What comes to the stubborn set method in high-level nets in general, it seems that the best way to proceed without any explicit or implicit unfolding is to define stubbornness on sets where the elements are appropriately chosen sequences of transition instances. Such sequences correspond to what is called *operations* in [29]. Stubbornness must be defined separately for each net, and the definition process is likely to require some thinking. However, the work done in the definition process may help the modeller to understand better the behaviour of the modelled system and thus pay off independently of how successful the actual state space generation is. An example of this kind of approach can be found in [29].

Quite recently, some experiments about applying the stubborn set method to high-level nets without unfolding have been reported [50]. We leave the evaluation of [50] for future.

Unfolding is still not to be avoided categorically. Many errors are of the kind that they occur with a little variation under all combinations of values of parameters of the modelled system, independently of how "realistic" the values are. Such errors can often be found by analyzing the system with a "minimal parameter configuration" that supports unfolding. The possible errors not found in this way can be tried to be found in other ways. On the other hand, it is often best to eliminate a found error from the model of the system before searching for more errors. The design and analysis of a model thus affect each other, and the designer may get feedback from the analyzer more quickly when the analysis is automatic as in the case of unfolding than when the analysis is semiautomatic as in [29].

9.2 Stubborn sets in symbolic state space generation

A *symbolic state space* is an indirect representation of a state space. A *symbolic state* then represents a set of states.

A *coverability graph* [64] of a place/transition net is a primitive example of a symbolic state space. A coverability graph is like a reachability graph, except that the vertices in the graph are virtual markings, i.e. functions from the set of places to $N \cup \{\omega\}$ where ω is a formal infinite number such that

$$\forall n \in N \; n < \omega \wedge \omega + n = \omega \wedge \omega - n = \omega.$$

Whenever there exist a place s and a virtual marking M in the graph in such a way that $M(s) = \omega$, M represents a set X of reachable ordinary markings in such a way that the set $\{M'(s) \mid M' \in X\}$ is infinite.

A place s is said to be *bounded* iff there exists $n \in N$ in such a way that for each reachable marking M, $M(s) \leq n$. A subset S' of places is said to be a set of *simultaneously unbounded places* iff for each $n \in N$, there exists a reachable marking M such that for each $s \in S'$, $M(s) > n$. A place is thus bounded iff the singleton set containing the place is not a set of simultaneously unbounded places. An ordinary or a virtual marking M' *covers* an ordinary or a virtual marking M iff for each place s in the net, $M'(s) \geq M(s)$. There exists a simple algorithm [64] that for any finite place/transition net constructs a finite coverability graph that satisfies the following: an ordinary marking M is covered by no reachable ordinary marking iff the constructed graph contains no virtual marking that covers M. From this it follows e.g. that all sets of simultaneously unbounded places can be detected from the constructed graph. Namely, a subset S' of places is a set of simultaneously unbounded places iff the constructed graph contains a virtual marking M such that for each $s \in S'$, $M(s) = \omega$.

Unfortunately, coverability graphs often do not contain sufficient information for solving a verification problem. For example, the coverability graph constructed for a net with a reachable terminal marking may be the same (up to isomorphism) as the coverability graph constructed for a net with no reachable terminal marking [58]. There are still some at least technically interesting classes of place/transition nets where coverability graphs can be used for deciding e.g. whether or not a reachable terminal marking exists. For such nets, the idea of constructing a reduced coverability

graph with the stubborn set method is quite natural. Stubbornness just has to be defined at a virtual marking. If we say that a subset T_s of transitions is stubborn at a virtual marking M in a coverability graph, then T_s should be such that for any reachable marking M' represented by M, T_s or some subset of T_s is stubborn at M'. Heuristics for computing stubborn sets for sets of markings are presented in [70, 71], and [35] suggests applying such heuristics to the construction of a reduced coverability graph.

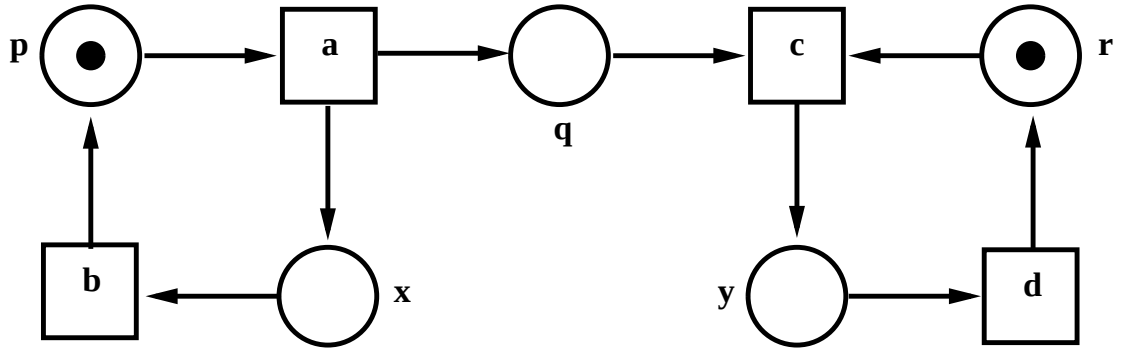


Figure 31: The place q may "look bounded" in a stubbornly reduced coverability graph.

The way suggested in [35] for constructing a reduced coverability graph cannot in general be used for the checking of boundedness of places. The net in Figure 31 satisfies all the special constraints listed in [35] for ensuring that the reduced coverability graph can be used for checking some basic things such as the existence or non-existence of reachable terminal markings. The checking of boundedness is not discussed in [35]. We immediately observe that the place q is not bounded. Let M_0 be the initial marking, $M_0[a\rangle M_1$, $M_1[c\rangle M_2$, and $M_2[b\rangle M_3$. Let stubbornness be defined as in [35]. (It is hard to imagine any reasonable definition of stubbornness that would make a difference w.r.t. this example.) Then $\{a\}$ is stubborn at M_0, $\{c\}$ is stubborn at M_1, $\{b\}$ is stubborn at M_2, and $\{d\}$ is stubborn at M_3. Consequently, the construction of a reduced coverability graph can result in a graph where all virtual markings are ordinary markings, namely the markings M_0, M_1, M_2 and M_3, while the only infinite path starting from M_0 in the graph is labelled by $(acbd)(acbd)(acbd)\ldots$.

We do not know any general solution to the problem how the stubborn set method could be revised to preserve boundedness information. In order to show that a place s is not bounded, it suffices to find from the full reachability graph a path containing markings M and M' such that M occurs before M', M' covers M and $M'(s) > M(s)$. The difficulty w.r.t. the stubborn set method is that all places are included in the covering condition. We can of course write LTL formulas of the form $\Box(M(s) \leq n)$.

Binary decision diagrams (*BDDs* for short) [13] and *graph encoded tuple sets* (*GETSes* for short) [32] are data structures that have turned out to be successful in encoding large state spaces or sets of states in a way that supports verification in a wide sense.

In the strictly BDD-based approach, a state space is described by a formula that is to be encoded into a BDD. Verification then reduces into manipulation of the BDD. The time consumed in such verification depends on the size of the BDD rather than on the size of the represented state space. In successful cases, a small BDD represents a

huge state space. BDDs are sometimes used simply as plain search structures where parts of the state space are kept. The benefit is that any algorithm working on the state space level is then directly applicable. GETSes have the same benefit. Unlike BDDs, GETSes are merely search structures, or at least we have not seen suggestions of using them in any other way.

The kind of a reduction obtained by means of the stubborn set method and similar techniques is to some extent orthogonal to the reduction obtained by a symbolic representation of states. The stubborn set method is able to cut down on redundancy caused by branching and can be "merged" with the strictly BDD-based verification approach as demonstrated in [1, 70, 71]. The merging requires that a prospective reduced state space to be represented can be expressed as a formula that can easily be encoded into a BDD. A given definition of stubbornness may have to be compromised in order to make the merging successful.

A drawback with BDDs is that the size of a BDD strongly depends on "uninteresting" things such as the chosen order of variables for the BDD. Consequently, the use of BDDs requires a lot of expertise. It somehow seems that BDDs are at best in cases where we have a deep and detailed understanding of the behaviour of a system and only need some formal way to show that the system is correct. We do not know how much GETSes differ from BDDs in these respects.

10 Conclusions

This thesis has considered relieving of the state space explosion problem that occurs in the analysis of concurrent and distributed systems. We have concentrated on one method for that purpose: the stubborn set method. We are fully aware of the fact that the stubborn set method has no special position among verification heuristics. It is also clear that in industrial-size cases, one method alone is typically almost useless. Our motivation is that whenever a method is used, it should be used reasonably.

The contributions of this thesis are as follows.

- In Chapter 4, we have rephrased persistence and conditional stubbornness in terms of strong dynamic stubbornness in place/transition nets.
- In Chapter 5, we have improved the LTL-preserving stubborn set method in such a way that it can utilize the structure of the formula when fairness is not assumed. At least the presented theory should be of more than temporary interest.
- In Chapter 6, we have designed an algorithm that computes stubborn sets that are cardinality minimal or almost cardinality minimal w.r.t. the number of enabled transitions. We have presented practical experiments that indicate that the algorithm is worth of consideration whenever one wants to get proper advantage of the stubborn set method.
- In Chapter 7, we have obtained practical evidence indicating that a stubborn reduction often produces a significant amount of intermediate states that can be eliminated without losing any important information. The way we have suggested for elimination is able to do something concrete in order to prevent an overflow in a depth-first search stack.
- In Chapter 8, we have shown that the stubborn set method can be combined with the sleep set method in the verification of basic termination properties and simple safety properties without having to place any assumptions on the stubborn sets used. We have also obtained very weak conditions for the sleep set method to be compatible with another method in such verification tasks. As an example, we have described one possible combination of the stubborn set method and the sleep set method to be used in on-the-fly verification of simple safety properties.

Though we have used place/transition nets as the main formalism, many of the results presented for place/transition nets can be extended, often by simple syntactic modification, to concern any other formalism where the stubborn set method is applicable.

The use of stubborn sets in various formalisms and logics is a fruitful area of future research. On the other hand, we should, by means of large case studies, try to find out what the central problems in the application of the method are and how these problems could be alleviated.

Acknowledgements

Discussions with Associate Professor Antti Valmari from Tampere University of Technology have affected the thesis in the following things: the definition of the and/or-graph (Definition 4.25), the assumption A4 (in Section 5.1) and the idea of intersecting an automatically constructed Büchi automaton with a reduced state space during the construction of the latter. The discussions in question took place in years 1992–1995.

The used model of YXA is based on a TNSDL description obtained from M.Sc.(Eng.) Esa Kettunen from Nokia Telecommunications Oy in 1994. The used model of MULOG is based on a PROMELA description obtained from Dr. Patrice Godefroid from AT&T in 1995.

Publications resulted from the work

The problems in choosing a scapegoat (Section 6.1) were discussed in [85, 86]. The connection between persistence and strong dynamic stubbornness was treated in [86, 87, 89]. The compatibility of the stubborn set method with the sleep set method was considered in [86, 87, 89, 91]. Preliminary comments concerning the verification of LTL-formulas were published in [88, 90]. The contribution of the above Chapter 5 is repeated in [93]. The incomplete minimization algorithm (Section 6.2) was introduced in [92]. For historical correctness concerning Chapter 7, we note that the algorithms SA and ES have been in the public release version of PROD since the end of the year 1995, and a paper similar to Chapter 7, written by the author of this thesis, was submitted to Euro-Par '96 in February 1996.

References

[1] Alur, R., Brayton, R.K., Henzinger, T.A., Qadeer, S., and Rajamani, S.K.: *Partial-Order Reduction in Symbolic State Space Exploration.* In [34], pp. 340–351.

[2] Ashcroft, E., and Manna, Z.: *Formalization of Properties of Parallel Programs.* Meltzer, B., and Michie, D. (Eds.), Machine Intelligence 6. Edinburgh University Press, Edinburgh, UK, 1971, pp. 17–42.

[3] Azéma, P., and Balbo, G. (Eds.): Proceedings of the 18th International Conference on Application and Theory of Petri Nets, Toulouse, France, June 1997. Lecture Notes in Computer Science 1248, Springer-Verlag, Berlin 1997, 467 p.

[4] Bause, F.: *Analysis of Petri Nets with a Dynamic Priority Method.* In [3], pp. 215–234.

[5] Berthelot, G.: *Transformations and Decompositions of Nets.* In [9], pp. 359–376.

[6] Berthelot, G., and Roucairol, G.: *Reduction of Petri-Nets.* Mathematical Foundations of Computer Science 1976. Lecture Notes in Computer Science 45, Springer-Verlag, Berlin 1976, pp. 202–209.

[7] Billington, J., and Reisig, W. (Eds.): Proceedings of the 17th International Conference on Application and Theory of Petri Nets, Osaka, Japan, June 1996. Lecture Notes in Computer Science 1091, Springer-Verlag, Berlin 1996, 549 p.

[8] von Bochmann, G., and Probst, D.K. (Eds.): Proceedings of the 4th International Workshop on Computer-Aided Verification, Montréal, Canada, June 1992. Lecture Notes in Computer Science 663, Springer-Verlag, Berlin 1993, 422 p.

[9] Brauer, W., Reisig, W., and Rozenberg, G. (Eds.): *Petri Nets: Central Models and Their Properties. Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, September 1986.* Lecture Notes in Computer Science 254, Springer-Verlag, Berlin 1987, 480 p.

[10] Brauer, W., Reisig, W., and Rozenberg, G. (Eds.): *Petri Nets: Applications and Relationships to Other Models of Concurrency. Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, September 1986.* Lecture Notes in Computer Science 255, Springer-Verlag, Berlin 1987, 516 p.

[11] Brgan, R., and Poitrenaud, D.: *An Efficient Algorithm for the Computation of Stubborn Sets of Well Formed Petri Nets.* De Michelis, G., and Diaz, M. (Eds.), Proceedings of the 16th International Conference on Application and Theory of Petri Nets, Turin, Italy, June 1995. Lecture Notes in Computer Science 935, Springer-Verlag, Berlin 1995, pp. 121–140.

[12] Büchi, J.R.: *On a Decision Method in Restricted Second Order Arithmetic.* Proceedings of the International Congress on Logic, Methodology and Philosophy of Science, 1960. Stanford University Press, Stanford CA, USA, 1962, pp. 1–12.

[13] Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., and Hwang, L.J.: *Symbolic Model Checking:* 10^{20} *States and Beyond.* Information and Computation 98 (1992) 2, pp. 142–170.

[14] Clarke, E.M., and Kurshan, R.P. (Eds.): Proceedings of the 2nd International Workshop on Computer-Aided Verification, New Brunswick NJ, USA, June 1990. Lecture Notes in Computer Science 531, Springer-Verlag, Berlin 1991, 372 p.

[15] Courcoubetis, C. (Ed.): Proceedings of the 5th International Conference on Computer-Aided Verification, Elounda, Greece, June/July 1993. Lecture Notes in Computer Science 697, Springer-Verlag, Berlin 1993, 504 p.

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

[17] Dembiński, P., and Średniawa, M. (Eds.): Proceedings of the 15th IFIP WG 6.1 Symposium on Protocol Specification, Testing and Verification, Warsaw, June 1995. Chapman & Hall, London 1996, 453 p.

[18] Desel, J., and Silva, M. (Eds.): Proceedings of the 19th International Conference on Application and Theory of Petri Nets, Lisbon, Portugal, June 1998. Lecture Notes in Computer Science, Springer-Verlag, Berlin 1998. (The contents of the book were fixed before April 1998.)

[19] Emerson, E.A.: *Temporal and Modal Logic.* van Leeuwen, J. (Ed.), Handbook of Theoretical Computer Science, Vol. B. Elsevier, Amsterdam 1990, pp. 995–1072.

[20] Esparza, J., and Melzer, S.: *Model Checking LTL Using Constraint Programming.* In [3], pp. 1–20.

[21] Francez, N.: *Fairness.* Springer-Verlag, Berlin 1986, 295 p.

[22] Genrich, H.J.: *Predicate/Transition Nets.* In [9], pp. 207–247.

[23] Gerth, R., Kuiper, R., Peled, D., and Penczek, W.: *A Partial Order Approach to Branching Time Logic Model Checking.* Proceedings of the 3rd Israel Symposium on Theory of Computing and Systems, Tel Aviv, Israel, 1995. IEEE Computer Society Press, Los Alamitos CA 1995, pp. 130–140.

[24] Gerth, R., Peled, D., Vardi, M.Y., and Wolper, P.: *Simple On-the-Fly Automatic Verification of Linear Temporal Logic.* In [17], pp. 3–18.

[25] Godefroid, P.: *Using Partial Orders to Improve Automatic Verification Methods.* In [14], pp. 176–185.

[26] Godefroid, P.: *Partial-Order Methods for the Verification of Concurrent Systems — An Approach to the State-Explosion Problem.* Lecture Notes in Computer Science 1032, Springer-Verlag, Berlin 1996, 143 p.

[27] Godefroid, P., Holzmann, G.J., and Pirottin, D.: *State-Space Caching Revisited.* Formal Methods in System Design 7 (1995) 3, pp. 227–241.

[28] Godefroid, P., and Kabanza, F.: *An Efficient Reactive Planner for Synthesizing Reactive Plans.* Proceedings of AAAI-91, Anaheim CA, USA, July 1991, Vol. 2, pp. 640–645.

[29] Godefroid, P., and Pirottin, D.: *Refining Dependencies Improves Partial-Order Verification Methods.* In [15], pp. 438–449.

[30] Godefroid, P., and Wolper, P.: *A Partial Approach to Model Checking.* Information and Computation 110 (1994) 2, pp. 305–326.

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

[32] Grégoire, J.-Ch.: *State Space Compression in SPIN with GETSs.* Proceedings of the 2nd International SPIN Verification Workshop, New Brunswick NJ, USA, August 1996, 19 p.

[33] Gribomont, E.P., and Wolper, P.: *Temporal Logic.* Thayse, A. (Ed.), From Modal Logic to Deductive Databases — Introducing a Logic Based Approach to Artificial Intelligence. John Wiley & Sons, New York NY, USA, 1989, pp. 165–233.

[34] Grumberg, O. (Ed.): Proceedings of the 9th International Conference on Computer-Aided Verification, Haifa, Israel, June 1997. Lecture Notes in Computer Science 1254, Springer-Verlag, Berlin 1997, 486 p.

[35] Hiraishi, K.: *Reduced State Space Representation for Unbounded Vector State Spaces.* In [7], pp. 230–248.

[36] Holzmann, G.J.: *Design and Validation of Computer Protocols.* Prentice-Hall, Englewood Cliffs NJ, USA, 1991, 500 p.

[37] Holzmann, G.J., and Peled, D.: *An Improvement in Formal Verification.* Hogrefe, D., and Leue, S. (Eds.), Proceedings of the 7th International IFIP WG 6.1 Conference on Formal Description Techniques, Bern, Switzerland, October 1994. Chapman & Hall, London 1995, pp. 177–191.

[38] Holzmann, G.J., Godefroid, P., and Pirottin, D.: *Coverage Preserving Reduction Strategies for Reachability Analysis.* Linn, R.J., Jr., and Uyar, M.Ü. (Eds.), Proceedings of the 12th International IFIP WG 6.1 Symposium on Protocol Specification, Testing and Verification, Lake Buena Vista FL, USA, June 1992. IFIP Transactions C-8, North-Holland, Amsterdam 1992, pp. 349–363.

[39] Janicki, R., and Koutny, M.: *Using Optimal Simulations to Reduce Reachability Graphs.* In [14], pp. 166–175.

[40] Jensen, K.: *Coloured Petri Nets and the Invariant Method.* Theoretical Computer Science 14 (1981), pp. 317–336.

[41] Jensen, K., and Rozenberg, G. (Eds.): *High-level Petri Nets. Theory and Application.* Springer-Verlag, Berlin 1991, 724 p.

[42] Kaivola, R.: *Equivalence, 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, Helsinki 1996, 185 p.

[43] Kaivola, R., and Valmari, A.: *The Weakest Compositional Semantic Equivalence Preserving Nexttime-less Linear Temporal Logic.* Cleaveland, W.R. (Ed.), Proceedings of the 3rd International Conference on Concurrency Theory, Stony Brook NY, USA, August 1992. Lecture Notes in Computer Science 630, Springer-Verlag, Berlin 1992, pp. 207–221.

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

[45] Katz, S., and Peled, D.: *Defining Conditional Independence Using Collapses.* Theoretical Computer Science 101 (1992) 2, pp. 337–359.

[46] Kernighan, B.W., and Ritchie, D.M.: *The C Programming Language.* 2nd edition. Prentice-Hall, Englewood Cliffs NJ, USA, 1988, 272 p.

[47] Kettunen, E., Montonen, E., and Tuuliniemi, T.: *A Comparison of Pr/T-Net Based FIFO Channel Models.* Helsinki University of Technology, Digital Systems Laboratory Report B 33, Espoo 1986, 18 p.

[48] Kokkarinen, I., Peled, D., and Valmari, A.: *Relaxed Visibility Enhances Partial Order Reduction.* In [34], pp. 328–339.

[49] Koutny, M., and Pietkiewicz-Koutny, M.: *On the Sleep Sets Method for Partial Order Verification of Concurrent Systems.* University of Newcastle upon Tyne, Department of Computing Science, Technical Report 495, Newcastle upon Tyne, UK, 1994.

[50] Kristensen, L.M., and Valmari, A.: *Finding Stubborn Sets of Coloured Petri Nets Without Unfolding.* In [18].

[51] Kurshan, R.P., Levin, V., Minea, M., Peled, D., and Yenigün, H.: *Static Partial Order Reduction.* Steffen, B. (Ed.), Proceedings of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Lisbon, Portugal, March/April 1998. Lecture Notes in Computer Science 1384, Springer-Verlag, Berlin 1998, pp. 345–357.

[52] Mazurkiewicz, A.: *Trace Theory.* In [10], pp. 279–324.

[53] Miller, H., and Katz, S.: *Saving Space by Fully Exploiting Invisible Transitions.* Alur, R., and Henzinger, T.A. (Eds.), Proceedings of the 8th International Conference on Computer-Aided Verification, New Brunswick NJ, USA, July/August 1996. Lecture Notes in Computer Science 1102, Springer-Verlag, Berlin 1996, pp. 336–347.

[54] Overman, W.T.: *Verification of Concurrent Systems: Function and Timing.* PhD thesis, University of California at Los Angeles, Los Angeles CA, USA, 1981, 174 p.

[55] Peled, D.: *All from One, One for All: on Model Checking Using Representatives.* In [15], pp. 409–423.

[56] Peled, D.: *Combining Partial Order Reductions with On-the-Fly Model-Checking.* Formal Methods in System Design 8 (1996) 1, pp. 39–64.

[57] Peled, D., and Penczek, W.: *Using Asynchronous Büchi Automata for Efficient Automatic Verification of Concurrent Systems.* In [17], pp. 115–130.

[58] Peterson, J.L.: *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Englewood Cliffs NJ, USA, 1981, 290 p.

[59] Petri, C.A.: *Kommunikation mit Automaten.* Schriften des IIM Nr. 2 (1962), Institut für Instrumentelle Mathematik, Bonn, Germany. English translation: *Communication with Automata.* Technical Report RADC-TR-65-377, Vol. 1, Suppl. 1, Griffith Air Force Base, New York NY, USA, 1966.

[60] Pnueli, A.: *Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends.* de Bakker, J.W., de Roever, W.-P., and Rozenberg, G. (Eds.), Current Trends in Concurrency, Overviews and Tutorials. Lecture Notes In Computer Science 224, Springer-Verlag, Berlin 1986, pp. 510–584.

[61] Proceedings of the 6th Annual IEEE Symposium on Logic in Computer Science, Amsterdam, July 1991. IEEE Computer Society Press, Los Alamitos CA, USA, 1991.

[62] Ramakrishna, Y.S., and Smolka, S.A.: *Partial-Order Reduction in the Weak Modal Mu-Calculus.* Mazurkiewicz, A., and Winkowski, J. (Eds.), Proceedings of the 8th International Conference on Concurrency Theory, Warsaw, July 1997. Lecture Notes in Computer Science 1243, Springer-Verlag, Berlin 1997, pp. 5–24.

[63] Rauhamaa, M.: *A Comparative Study of Methods for Efficient Reachability Analysis.* Helsinki University of Technology, Digital Systems Laboratory Report A 14, Espoo 1990, 61 p.

[64] Reisig, W.: *Petri Nets: An Introduction.* EATCS Monographs on Theoretical Computer Science 4, Springer-Verlag, Berlin 1985, 161 p.

[65] Reisig, W.: *Place/Transition Systems.* In [9], pp. 117–141.

[66] Safra, S.: *On the Complexity of ω-automata.* Proceedings of the 29th Annual Symposium on Foundations of Computer Science, White Plains NY, USA, October 1988. IEEE Computer Society Press, Washington DC, USA, 1988, pp. 319–327.

[67] Sedgewick, R.: *Algorithms.* Addison-Wesley, Reading MA, USA, 1983, 551 p.

[68] Sloan, R.H., and Buy, U.: *Stubborn Sets for Real-Time Petri Nets.* Formal Methods in System Design 11 (1997) 1, pp. 23–40.

[69] Tarjan, R.E.: *Depth-First Search and Linear Graph Algorithms.* SIAM Journal of Computing 1 (1972) 2, pp. 146–160.

[70] Tiusanen, M.: *Static Analysis of Ada Tasking Programs: Models and Algorithms.* PhD thesis, University of Illinois at Chicago, Chicago IL, USA, 1993, 161 p.

[71] Tiusanen, M.: *Symbolic, Symmetry, and Stubborn Set Searches.* In [72], pp. 511–530.

[72] Valette, R. (Ed.): Proceedings of the 15th International Conference on Application and Theory of Petri Nets, Zaragoza, Spain, June 1994. Lecture Notes in Computer Science 815, Springer-Verlag, Berlin 1994, 587 p.

[73] Valmari, A.: *Error Detection by Reduced Reachability Graph Generation.* Proceedings of the 9th European Workshop on Application and Theory of Petri Nets, Venice, Italy, June 1988, pp. 95–112.

[74] Valmari, A.: *Heuristics for Lazy State Space Generation Speeds up Analysis of Concurrent Systems.* Mäkelä, M., Linnainmaa, S., and Ukkonen, E. (Eds.), Proceedings of the Finnish Artificial Intelligence Symposium (Suomen tekoälytutkimuksen päivät), Vol. 2, Helsinki 1988, pp. 640–650.

[75] Valmari, A.: *State Space Generation: Efficiency and Practicality.* Doctoral thesis, Tampere University of Technology Publications 55, Tampere 1988, 170 p.

[76] Valmari, A.: *Eliminating Redundant Interleavings during Concurrent Program Verification.* Proceedings of Parallel Architectures and Languages Europe '89 Vol. 2. Lecture Notes in Computer Science 366, Springer-Verlag, Berlin 1989, pp. 89–103.

[77] Valmari, A.: *Stubborn Sets for Reduced State Space Generation.* Rozenberg, G. (Ed.), Advances in Petri Nets 1990. Lecture Notes in Computer Science 483, Springer-Verlag, Berlin 1991, pp. 491–515.

[78] Valmari, A.: *A Stubborn Attack on State Explosion.* Formal Methods in System Design 1 (1992) 4, pp. 297–322.

[79] Valmari, A.: *Stubborn Sets of Coloured Petri Nets.* Proceedings of the 12th International Conference on Application and Theory of Petri Nets, Gjern, Denmark, June 1991, pp. 102–121.

[80] Valmari, A.: *Alleviating State Explosion during Verification of Behavioural Equivalence.* University of Helsinki, Department of Computer Science, Report A-1992-4, Helsinki 1992, 57 p.

[81] Valmari, A.: *On-the-Fly Verification with Stubborn Sets.* In [15], pp. 397–408.

[82] Valmari, A., and Tienari, M.: *An Improved Failures Equivalence for Finite-State Systems with a Reduction Algorithm.* Jonsson, B., Parrow, J., and Pehrson, B. (Eds.), Proceedings of the 11th International IFIP WG 6.1 Symposium on Protocol Specification, Testing and Verification, Stockholm, June 1991. North-Holland, Amsterdam 1991, pp. 3–18.

[83] Valmari, A., and Tiusanen, M.: *A Graph Model for Efficient Reachability Analysis of Description Languages.* Proceedings of the 8th European Workshop on Application and Theory of Petri Nets, Zaragoza, Spain, June 1987, pp. 349–366.

[84] Vardi, M.Y., and Wolper, P.: *An Automata-Theoretic Approach to Automatic Program Verification.* Proceedings of the 1st IEEE Symposium on Logic in Computer Science, Cambridge, UK, June 1986. IEEE Computer Society Press, Washington DC, USA, 1986, pp. 322–344.

[85] Varpaaniemi, K.: *On Choosing a Scapegoat in the Stubborn Set Method.* Burkhard, H.-D., Starke, P.H., and Czaja, L. (Eds.), Proceedings of the 1st International Workshop on Concurrency, Specification and Programming, Berlin, November 1992. Fachbereich Informatik der Humboldt-Universität zu Berlin, Informatik-Preprint 22, Berlin 1993, pp. 163–171.

[86] Varpaaniemi, K.: *Efficient Detection of Deadlocks in Petri Nets.* Helsinki University of Technology, Digital Systems Laboratory Report A 26, Espoo 1993, 56 p.

[87] Varpaaniemi, K.: *Dynamically Stubborn Sets and the Sleep Set Method.* Burkhard, H.-D., Czaja, L., and Starke, P.H. (Eds.), Proceedings of the 2nd International Workshop on Concurrency, Specification and Programming, Nieborów, Poland, October 1993. Zakład Graficzny UW, zam. 261/94, Warsaw 1994, pp. 230–246.

[88] Varpaaniemi, K.: *On Computing Symmetries and Stubborn Sets.* Helsinki University of Technology, Digital Systems Laboratory Report B 12, Espoo 1994, 16 p.

[89] Varpaaniemi, K.: *On Combining the Stubborn Set Method with the Sleep Set Method.* In [72], pp. 548–567.

[90] Varpaaniemi, K.: *On-the-Fly Verification with PROD.* Desel, J., Oberweis, A., and Reisig, W. (Eds.), Proceedings of the "Algorithmen und Werkzeuge für Petrinetze" workshop, Berlin, October 1994. Universität Karlsruhe (TH), Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Bericht 309, Karlsruhe, Germany, 1994, pp. 80–83.

[91] Varpaaniemi, K.: *The Sleep Set Method Revisited.* Czaja, L., Burkhard, H.-D., and Starke, P.H. (Eds.), Proceedings of the 3rd International Workshop on Concurrency, Specification and Programming, Berlin, October 1994. Humboldt-Universität zu Berlin, Institut für Informatik, Informatik-Bericht 36, Berlin 1994, 10 p.

[92] Varpaaniemi, K.: *Finding Small Stubborn Sets Automatically.* Atalay, V., Halici, U., Inan, K., Yalabik, N., and Yazici, A. (Eds.), Proceedings of the 11th International Symposium on Computer and Information Sciences, Antalya, Turkey, November 1996, Vol. I. Middle East Technical University, Ankara, Turkey, 1996, ISBN 975-429-103-9, pp. 133–142.

[93] Varpaaniemi, K.: *On Stubborn Sets in the Verification of Linear Time Temporal Properties.* In [18].

[94] Varpaaniemi, K., Halme, J., Hiekkanen, K., and Pyssysalo, T.: *PROD Reference Manual.* Helsinki University of Technology, Digital Systems Laboratory Report B 13, Espoo 1995, 56 p.

[95] Varpaaniemi, K., and Rauhamaa, M.: *The Stubborn Set Method in Practice.* Jensen, K. (Ed.), Proceedings of the 13th International Conference on Application and Theory of Petri Nets, Sheffield, June 1992. Lecture Notes in Computer Science 616, Springer-Verlag, Berlin 1992, pp. 389–393.

[96] Vernadat, F., Azéma, P., and Michel, F.: *Covering Step Graph.* In [7], pp. 516–535.

[97] Vernadat, F., and Michel, F.: *Covering Step Graph Preserving Failure Semantics.* In [3], pp. 253–270.

[98] Willems, B., and Wolper, P.: *Partial-Order Methods for Model-Checking: from Linear Time to Branching Time.* Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick NJ, USA, July 1996. IEEE Computer Society Press, Los Alamitos CA, USA, 1996, pp. 294–303.

[99] Wolper, P., and Godefroid, P.: *Partial-Order Methods for Temporal Verification.* Best, E. (Ed.), Proceedings of the 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 1993. Lecture Notes in Computer Science 715, Springer-Verlag, Berlin 1993, pp. 233–246.

[100] Wolper, P., Godefroid, P., and Pirottin, D.: *A Tutorial on Partial-Order Methods for the Verification of Concurrent Systems.* Tutorial material of the 5th International Conference on Computer-Aided Verification, Elounda, Greece, June/July 1993, 85 p.

[101] Yoneda, T., Shibayama, A., Schlingloff, B.-H., and Clarke, E.M.: *Efficient Verification of Parallel Real-Time Systems.* In [15], pp. 321–332.