STUBBORN SETS FOR REDUCED STATE SPACE GENERATION

Antti Valmari

Technical Research Centre of Finland
Computer Technology Laboratory
PO Box 201, SF–90571 OULU
FINLAND
Tel. int. +358 81 509 111[1]

ABSTRACT The "stubborn set" theory and method for generating reduced state spaces is presented. The theory takes advantage of concurrency, or more generally, of the lack of interaction between transitions, captured by the notion of stubborn sets. The basic method preserves all terminal states and the existence of nontermination. A more advanced version suited to the analysis of properties of reactive systems is developed. It is shown how the method can be used to detect violations of invariant properties. The method preserves the liveness (in Petri net sense) of transitions, and livelocks which cannot be exited. A modification of the method is given which preserves the language generated by the system. The theory is developed in an abstract variable/transition framework and adapted to elementary Petri nets, place/transition nets with infinite capacity of places, and coloured Petri nets.

Keywords system verification, analysis of behaviour of nets

CONTENTS

---

1 The original version of this paper was written while the author was visiting Telecom Australia Research Laboratories, 770 Blackburn Road, Clayton, Victoria 3168, AUSTRALIA.

0. INTRODUCTION

Reachability analysis (or state space generation) is a widely used method for analysing concurrent systems, even though it has serious performance problems. Significant increases in the performance of state space generation would have great practical value. Unfortunately, the number of states of even moderate systems is often astronomical. As we cannot expect an astronomical number of states to be generated in a reasonable time no matter how fast an algorithm we have, it seems that the performance problems cannot be solved without reducing the number of states that are generated.

In this paper a theory is presented facilitating reduction of the number of states without modifying the answers of certain (many) analysis questions. The theory takes advantage of concurrency. Concurrency offers much potential for state space reduction, as demonstrated by the following example. The system of n non-interacting processes each executing k steps sequentially before stopping has $(k+1)^n$ states. This is exponential in the number of processes. It seems, however, intuitively clear that not all of those states are really needed. Since the processes do not interact, the end result of their execution is independent of their order of execution. Furthermore, all projections of the global state of the system to each of the processes are found if we execute the processes in any order. One might thus claim that sufficient information for the analysis of the system is achieved by simulating the processes in one arbitrarily chosen order. Simulation in one order generates only $nk+1$ states, which is linear in n and k.

The n-process example above is oversimplified, because it ignores the fact that processes do interact in practice. Two questions arise:

- How can we take advantage of situations where processes do not interact without losing the consequences of their interaction?
- What properties of the system under analysis are preserved during state space reduction?

The theory developed in this paper answers the first question through the notion of *stubborn sets*. Roughly speaking, a stubborn set is a set of transitions closed with respect to mutual interactions; all transitions interfering with the transitions in the set belong to the set. The notion is dynamic, that is, which sets are stubborn depends on the current state. In the stubborn set state space reduction method a stubborn set is computed at every state (strictly speaking, at every state with enabled transitions), and only the enabled transitions in it are used when next states are generated.

A partial answer to the second question will be given by a set of theorems, which guarantee that all terminal states (that is, states with no enabled transitions) and nontermination are preserved by the method. Furthermore, there is a version of the method which preserves livelocks (defined as unintended cyclic terminal strong components of the state space), liveness of transitions (in the Petri net sense of the word) and facilitates the detection of violations of invariant properties, expressed as fact transitions. The method can be adapted to preserve the language generated by a system, where a symbol is associated to some (but not necessarily all) transitions and transitions write their symbols when they occur.

The stubborn set method is applicable to several models of concurrency. The notion "stubborn set" captures interaction relationships between transitions. Different concurrency models express information about transition interactions in different (and sometimes non-transparent) ways, and there is often more than one way to utilise this information in the definition of stubborn sets. Therefore it is better not to tie the theory to a particular established formalism, but to develop it in a general framework, from which it can be easily adapted to other formalisms. We develop the theory in the framework of *variable/transition systems*, apply it to

elementary and place/transition nets, and sketch two different applications to coloured Petri nets. As mentioned in the conclusions, the notion of variable/transition systems proved advantageous also when implementing the method.

The stubborn set theory was first introduced in [Valmari 88a], where the theory was developed directly for place/transition nets without capacity constraints. The paper also describes a linear algorithm for finding good (but not necessarily optimum) stubborn sets. The first attempt at developing the theory in a more general setting was [Valmari 88b], but the theory developed there is not well suited to Petri nets. The paper describes a quadratic algorithm for finding stubborn sets which produces optimum (in a certain sense) stubborn sets in many concurrency formalisms, including low level nets. Valmari's Ph.D. thesis [Valmari 88c] supplements the material in the papers mentioned and compares the stubborn set theory to some earlier attempts of utilising concurrency in state space reduction, including the classic static *virtual coarsening of atomic actions* (see [Pnueli 86]) and two methods by Overman [Overman 81]. [Valmari 89a] develops what is called in this paper the *weak* stubborn set theory of systems which are intended to terminate, and applies it to shared variable multi-process programs and elementary Petri nets.

The stubborn set method as developed in the above mentioned papers is suitable for analysing termination-oriented properties, that is, deadlocks, states corresponding to successful termination, and failure of termination. A certain kind of fairness problem (called *ignoring* in this paper) prevents the use of the method to analyse properties which are not directly related to termination. In this paper, after developing the basic stubborn set method, we present a solution to the ignoring problem. The solution renders possible the use of the stubborn set method to analyse liveness of transitions (in the Petri net sense of the word), livelocks, invariant properties (expressed as fact transitions) and the language generated by the system. To make the solution practical, an algorithm is described which can be embedded to reduced state space generation to ensure that ignoring is absent.

Another new result in this paper is the distinction of the *weak* and *strong* variants of the stubborn set theory. The weak theory generally leads to better state space reduction results. On the other hand, the implementation of the strong stubborn set theory is easier. For instance, the ignoring elimination algorithm given in this paper works with strong stubborn sets only.

This paper was originally published as a Petri Net Conference paper [Valmari 89b]. Changes were made to this version, most significantly the addition of Corollary 1.32 (livelock analysis) and Theorem 1.34 (language preservation). More recent development goes beyond this paper. There is now a stubborn set method which accepts a collection of linear temporal logic formulas (which are not allowed to use the "next state" operator), and preserves the truth values of them [Valmari 90]. There is also a version which preserves the "failure set semantics" [Brookes & 84] of systems.

Chapter 1 presents the theory in the variable/transition framework, and Chapter 2 demonstrates how it may be applied to Petri nets. A specific question of interest concerning Petri nets is how the stubborn set method relates to other Petri net state space reduction methods. Chapter 3 compares the stubborn set method to Jensen's equivalent marking method (see [Jensen 87]) with the aid of an example.

1. THEORY OF STUBBORN SETS

In this chapter we develop the stubborn set state space reduction method in the *variable/transition* system (*v/t-system*, for short) framework. V/T-systems resemble Petri nets, but the concepts of *token* and *place* have been replaced by the concept of *variable*. V/T-systems are defined in Section 1.1. The first key concept of the theory, namely the concept of *semi-stubborn* set of transitions is developed in Section 1.2. *Stubborn* sets are defined in Section 1.3. Section 1.4 gives the stubborn set reduced state space generation algorithm and shows

that it preserves all terminal states and the existence of nontermination. Section 1.5 discusses the *ignoring* problem which arises when stubborn sets are used to analyse properties not directly related to termination, and shows how it can be avoided if the strong version of the stubborn set theory is used. In Section 1.6 we take advantage of the solution to the ignoring problem and discuss the verification of various properties not directly related to termination.

1.1 Variable/Transition Systems

1.1.1 Formal Definition of V/T-Systems

Variable/transition systems (v/t-systems) can be thought of as abstractions of shared variable multi-process programs, where the concept of the location of the control of a process has been deprived of its special significance and is replaced by an ordinary variable. People working in temporal logic use quite similar models (see e.g. [Manna & 81], [Pnueli 86]), and so do Back and Kurki-Suonio in their *joint action* research [Back & 87]. V/T-systems can also be thought of as abstractions of Petri nets, where places and tokens have been replaced by variables and their values (one should note, however, that the abstraction is valid for interleaving semantics only). The formal definition of v/t-systems is as follows.

Definition 1.1 A *variable/transition system* (*v/t-system*, for short) is a five-tuple (V,T, *type*,*next*,ss_0), where

- V is a finite set of elements called *variables*
- T is a finite set of elements called *transitions*
- *type* is a function assigning a set called *type* to each variable, and
- the definition of *next* and ss_0 is deferred for a moment. □

Variables may be interpreted as programming language variables, program counters, message queues, or Petri net places. The type of a variable is a set, the elements of which are called *values*. A type may be interpreted as the type of a programming language variable, as the set of locations in a process where its control may reside, as the set of message sequences that may be stored in a message queue, as the set of numbers denoting the number of tokens in a place/transition net place, as the set of multisets denoting the contents of a high level net place, or something else, depending on the intended interpretation of the variable. The interpretation of transitions will be discussed in a moment.

Without loss of generality we assume that V is ordered. We can now complete the definition of v/t-systems and define the concept of *state*.

Definition 1.1, continued

- The Cartesian product of the types of the variables is called the *set of syntactic states* and is denoted by $\mathbb{S}$. Elements of $\mathbb{S}$ are called *states*.
- *next*: $\mathbb{S} \times T \rightarrow \mathbb{S} \cup \{undefined\}$, where *undefined* is a symbol not in $\mathbb{S}$. That is, *next* is a partial function from states and transitions to states. *next* is called the *next state function*.
- $ss_0 \in \mathbb{S}$ is a distinguished state called the *initial state*. □

Examples of states are the states of a concurrent program and the markings of a Petri net. The value of variable v at state s is denoted by $s(v)$. We denote the initial state by ss_0 instead of the perhaps more familiar s_0 since we want to use the latter symbol when sequences of states are discussed. The following definition gives the terms and notation associated with the occurrences of transitions.

Definition 1.2

- Transition t is *enabled* at state s, denoted by $en(s,t)$, if and only if $next(s,t) \neq undefined$. Otherwise t is *disabled*. The predicate "$next(s,t) \neq undefined$" is called the *enabling condition* of t.
- If $next(s,t) = s'$ where $s' \in \mathbb{S}$, we say that t may *occur* at s resulting in s'. We also use the notation $s -t\rightarrow s'$. □

An occurrence of a transition changes the state of the system. Transitions can be interpreted as atomic actions in a shared variable multi-process program, or as transitions of a Petri net, etc. Some concepts related to occurrence sequences, and the reachability relation between states are defined as follows.

Definition 1.3 Let $n \geq 0$, $s, s', s_0, \ldots, s_n \in \mathbb{S}$, and $t_1, \ldots, t_n \in T$.

- The sequence $\sigma = s_0 -t_1\rightarrow s_1 -t_2\rightarrow \ldots -t_n\rightarrow s_n$, where $s_{i-1} -t_i\rightarrow s_i$ for $i \in \{1,\ldots,n\}$ is called an *occurrence sequence*. The *length* of σ is n and is denoted by $|\sigma|$. The *first state* and *last state* of σ are s_0 and s_n and are denoted by $fs(\sigma)$ and $ls(\sigma)$, respectively.
- $s \rightarrow s' \Leftrightarrow \exists\, t \in T: s -t\rightarrow s'$.
- s' is *reachable from* s, denoted by $s \rightarrow^* s'$, if and only if there is an occurrence sequence σ such that $fs(\sigma) = s$ and $ls(\sigma) = s'$. (That is, "$\rightarrow^*$" is the reflexive and transitive closure of "$\rightarrow$".) □

The initial state and the next state function of a v/t-system define a labelled directed graph in a natural way:

Definition 1.4 The labelled directed graph (W,E,T) where

- $W = \{s \in \mathbb{S} \mid ss_0 \rightarrow^* s\}$
- $E = \{(s,t,s') \in W \times T \times W \mid s -t\rightarrow s'\}$

is called the *state space* of v/t-system $(V,T,type,next,ss_0)$. □

Assuming that two or more transitions cannot occur simultaneously, every possible finite or infinite execution of a v/t-system corresponds to a finite or infinite path in its state space. Execution models where only one transition can occur at a time are called *interleaving* models. The state space captures the interleaving semantics of v/t-systems.

We need to talk about the maximal strongly connected components (or strong components, for short) of the state space. Furthermore, we divide the strong components to terminal and nonterminal and to cyclic and acyclic ones.

Definition 1.5

- State s is a *terminal state*, if and only if no transition is enabled at s.
- $C \subseteq W$ is a *strong component*, if and only if there is $s \in C$ such that $C = \{s' \in W \mid s \rightarrow^* s' \wedge s' \rightarrow^* s\}$.
- Strong component C is *nonterminal*, if and only if $\exists\, s \in C: \exists\, s' \notin C: s \rightarrow s'$. Otherwise C is *terminal*.
- Strong component C is *cyclic*, if and only if $\exists\, (s,t,s') \in E: s \in C \wedge s' \in C$. Otherwise C is *acyclic*. □

An acyclic strong component contains exactly one state, but a cyclic one may contain one or more. Terminal states correspond to terminal acyclic strong components.

1.1.2 Connections of Variables and Transitions

So far we have not defined any notion of "connection" or "arc" between variables and transitions. That is, the next state function of a transition may be thought of as referring to every variable. In this section we define several notions that facilitate talking about different types of connections between variables and transitions, and help structure the next state function in a certain way.

The next definition assigns to each transition a test set, a write set, a read set and a connection set, the last one being the union of the other three. Informally, the test set is intended to represent the set of variables the transition refers to in its enabling condition. We do not, however, define it as *the* set of variables that the transition refers to in its enabling condition. Rather, we define it as a given set that must have a certain property; namely the property that if the variables in it assume equal values at two states, and the transition is enabled at one of the states, it is also enabled at the other. Clearly any set that has this property can be enlarged without violating the property. Therefore test sets can be defined in any convenient way, as long as they contain enough variables for the key property of test sets to be satisfied.

The reason for building this flexibility into the definition is that with shared variable multi-process programs it may be very difficult to compute the smallest possible legal test set. Consider a guarded programming language command with guard $(x \neq x)$. The guard is equivalent to **`false`**, so the reference to x in it is in a sense fake. It turns out that the empty set is a legal test set for the corresponding transition. However, it may be very difficult to distinguish between such fake references and true references. It is often easier to define that the test set is comprised of all variables appearing in the guard, plus the program counter of the process in question. This may lead to larger test sets than absolutely necessary, but promotes practicality.

Similar comments also hold for the write set and the read set. The write set of a transition is intended to contain (at least) the variables, the values of which the transition can modify when it occurs. That is, the values of variables outside the write set are not modified when the transition occurs. The read set is intended to contain (at least) the variables, the values of which are used by the transition when it determines the new values it assigns to (some) variables in its write set. To prevent the definition of the write set from interfering with the definition of the read set, the latter is made a bit complicated; it states that if a transition can occur in two states where the values of the variables in its read set agree, then for every variable the value of which is actually changed by either occurrence, the new value will be the same in both cases.

Definition 1.6 Let $(V,T,type,next,ss_0)$ be a v/t-system and let $t \in T$.

- $X \subseteq V$ has the *test set property* w.r.t. t if and only if for all states s and s':

  $$en(s,t) \wedge \forall\, v \in X\colon s'(v) = s(v) \;\Rightarrow\; en(s',t)$$

- $X \subseteq V$ has the *write set property* w.r.t. t if and only if for all states s and s':

  $$s \text{ -}t\rightarrow s' \;\Rightarrow\; \forall\, v \notin X\colon s'(v) = s(v)$$

- $X \subseteq V$ has the *read set property* w.r.t. t if and only if for all states s_1, s'_1, s_2 and s'_2:

  $$s_1 \text{ -}t\rightarrow s'_1 \wedge s_2 \text{ -}t\rightarrow s'_2 \wedge \forall\, v \in X\colon s_1(v) = s_2(v) \;\Rightarrow$$
  $$\forall\, v \in V\colon s'_1(v) = s'_2(v) \vee (\, s'_1(v) = s_1(v) \wedge s'_2(v) = s_2(v)\,)$$ □

From now on we assume that a unique *test set* of t satisfying the test set property is assigned to every transition t, and similarly a *write set* and a *read set*. The sets are denoted by $test(t)$, $wr(t)$ and $rd(t)$. We also define the *connection set* of t by $conn(t) = test(t) \cup wr(t) \cup rd(t)$.

The following definition allows us to look at the connections from the variables' point of view, and to talk about the connections of a set of variables or transitions.

Definition 1.7 Let v be a variable, $X \subseteq V$ or $X \subseteq T$, and *cset* be any of *test*, *wr*, *rd* and *conn*.

- $cset(v) = \{t \in T \mid v \in cset(t)\}$
- $cset(X) = \bigcup_{x \in X} cset(x)$ □

It is often the case that one can decide that a transition is disabled without knowing the values of all variables, or even the values of all the variables in its test set. For instance, a transition of a place/transition net is certainly disabled if it has an empty input place, independent of the markings of the other places. The stubborn set state space reduction theory takes advantage of this fact. This motivates the following definition.

Definition 1.8 Transition t is *enabled with respect to* a set of variables $U \subseteq V$ at state s, denoted by $en(s,t,U)$, if and only if

$$\exists\, s' \in \mathbb{S}\colon en(s',t) \wedge \forall\, v \in U\colon s'(v) = s(v)$$ □

The following properties of *en* are quite obvious:

$$\neg\, en(s,t,U) \;\Rightarrow\; \neg\, en(s,t)$$

$$en(s,t,U) \wedge U' \subseteq U \;\Rightarrow\; en(s,t,U')$$

$$en(s,t) \;\Leftrightarrow\; en(s,t,test(t)) \;\Leftrightarrow\; \forall\, U \subseteq V\colon en(s,t,U)$$

The stubborn set theory takes advantage of knowledge of transitions that can make a given transition enabled or disabled with respect to certain variables. In a place/transition net, for instance, assuming that there is an arc from place p to transition t but not vice versa, then only the transitions that increase the number of tokens in p can make t enabled with respect to $\{p\}$, and only the transitions that decrease the number of tokens in p can make t disabled with respect to $\{p\}$.

Definition 1.9 Let $t \in T$ and $U \subseteq V$.

- $X \subseteq wr(test(t))$ has the *write up set property* w.r.t. t and U if and only if for all states s and s' and transitions t':

  $$s\,[t'\rangle\, s' \wedge \neg\, en(s,t,U) \wedge t' \notin X \;\Rightarrow\; \neg\, en(s',t,U)$$

- $X \subseteq wr(test(t))$ has the *write down set property* w.r.t. t and U if and only if for all states s and s' and transitions t':

  $$s\,[t'\rangle\, s' \wedge en(s,t,U) \wedge t' \notin X \;\Rightarrow\; en(s',t,U)$$ □

The use of $wr(test(t))$ instead of the set of all transitions in Definition 1.9 is sound, because the transitions not in $wr(test(t))$ cannot make t enabled or disabled. From now on we assume that a unique *write up set* denoted by $wrup(t,U)$ and a unique *write down set* denoted by $wrdn(t,U)$ is attached to every $t \in T$ and $U \subseteq V$.

The final definition in this section is motivated by the fact that the enabling conditions of transitions can often be represented as conjunctions of conditions concerning small sets of variables. For instance, assuming that there are no capacity constraints, a place/transition net transition is enabled if and only if all its input places contain a sufficient number of tokens. This is the conjunction of the requirements that each individual place contains enough tokens.

Definition 1.10 Let $n \geq 1$. $\{V_1,\ldots,V_n\}$ is a *separation* of the enabling condition of transition t, denoted by $sep(t;V_1,\ldots,V_n)$, if and only if

- $V_1 \cup \ldots \cup V_n = test(t)$

- $V_i \cap V_j = \emptyset$ if $i \neq j$, and
- $\forall\, s \in \mathbb{S}$: ($en(s,t,V_1) \wedge \ldots \wedge en(s,t,V_n) \Rightarrow en(s,t)$) □

For simplicity, we will often talk of separations of transitions, when meaning separations of their enabling conditions. A separation always exists, because $sep(t;test(t))$ holds for every transition t. Because $en(s,t)$ implies $en(s,t,U)$ for every $U \subseteq V$, $sep(t;V_1,\ldots,V_n)$ implies

$$en(s,t,V_1) \wedge \ldots \wedge en(s,t,V_n) \;\Leftrightarrow\; en(s,t).$$

1.2 Semistubborn Sets of Transitions

A semistubborn set is a set of transitions satisfying a certain condition depending on the state. For the stubborn set state space reduction method to be practical, semistubborn sets should be defined statically, that is, in a way facilitating their computation using information about one state only. The following theorem plays a key role in the stubborn set theory, as it allows the permutation of occurrence sequences so that the permuted ones start with a transition belonging to a semistubborn set. It will make it possible to limit to a (certain kind of) semistubborn set when generating successors of the state. In this section we work backwards and define semistubborn sets so that the theorem can be proven.

Theorem 1.11 Let $T_s \subseteq T$ be semistubborn at state s_0, and let an occurrence sequence σ be given as below, where $t_1,\ldots,t_{n-1} \notin T_s$ and $t_n \in T_s$:

$$\sigma = s_0 -t_1\rightarrow s_1 -t_2\rightarrow \ldots -t_{n-1}\rightarrow s_{n-1} -t_n\rightarrow s_n$$

There is an occurrence sequence σ' as below, with the property that $s'_{n-1} = s_n$:

$$\sigma' = s_0 -t_n\rightarrow s'_0 -t_1\rightarrow s'_1 -t_2\rightarrow \ldots -t_{n-1}\rightarrow s'_{n-1}$$ □

Assuming that we are only interested in the states succeeding a future occurrence of a transition belonging to a semistubborn set, the theorem guarantees that we do not lose such states even if we restrict our attention to the semistubborn set when generating successors of the current state. The relation between σ and σ' can be illustrated graphically:

$$\begin{array}{ccccccccccc}
s_0 & -t_1\rightarrow & s_1 & -t_2\rightarrow & \ldots & -t_{n-2}\rightarrow & s_{n-2} & -t_{n-1}\rightarrow & s_{n-1} \\
| & & & & & & & & | \\
t_n & & & & & & & & t_n \\
\downarrow & & & & & & & & \downarrow \\
s'_0 & -t_1\rightarrow & s'_1 & -t_2\rightarrow & \ldots & -t_{n-2}\rightarrow & s'_{n-2} & -t_{n-1}\rightarrow & s_n = s'_{n-1}
\end{array}$$

We now proceed to define semistubborn sets such that Theorem 1.11 can be proven. To guarantee the existence of σ', we should establish three things:

(1) that t_n is enabled at s_0,

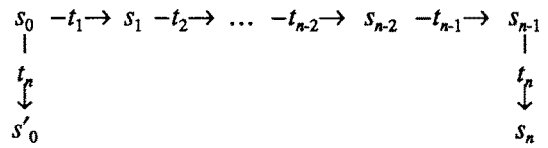
(2) that t_i is enabled at s'_{i-1} for $i = 1,\ldots,n-1$, and

(3) that $s'_{n-1} = s_n$ (the existence of s'_{n-1} is guaranteed by (1) and (2)).

Perhaps the most straightforward way to guarantee (1) would be to require that a semistubborn set contains only enabled transitions. It is the case, however, that to get a useful definition of semistubborn sets, we have to accept the presence of disabled transitions. Thus we should prevent a disabled transition from being the t_n of σ. This can be achieved by guaranteeing that a disabled transition cannot be enabled as a consequence of occurrences of transitions not belonging to T_s. The following condition is sufficient:

$$t \in T_s \wedge \neg\, en(s,t) \;\Rightarrow\; \exists\, U \subseteq V: \neg\, en(s,t,U) \wedge wrup(t,U) \subseteq T_s$$

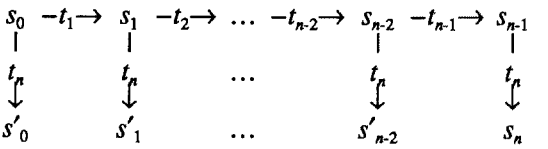
That is, if t is disabled at s, there is a subset of variables U so that t is not enabled at s with respect to U, and all transitions that can make t enabled with respect to U belong to T_s. The condition guarantees that we have at least the following figure:



As an intermediate step in establishing (2) and (3) we state a requirement guaranteeing that t_n is enabled at $s_1,\ldots,s_{n-2}$. It is sufficient to consider enabled transitions only since t_n is already guaranteed to be enabled at s_0. By the definition of σ, t_n is also enabled at s_{n-1}. There are two possibilities: we may either prevent an enabled transition from becoming disabled, or we may prevent it from becoming enabled again after becoming disabled. The following requirement is sufficient:

$$t \in T_s \wedge en(s,t) \;\Rightarrow\; \exists\, V_1,\ldots,V_m\colon sep(t;V_1,\ldots,V_m) \;\wedge\; (\,\forall\, j{=}1,\ldots,m\colon wrdn(t,V_j) \subseteq T_s \vee wrup(t,V_j) \subseteq T_s\,)$$

That is, a separation of the enabling condition of t is given, and for each set V_j of the separation, either no transition outside T_s can make t disabled with respect to V_j, or no transition outside T_s can make t enabled with respect to V_j. Let $i \in \{0,\ldots,n{-}1\}$. In the former case $en(s_i,t_n,V_j)$ holds, because t_n is enabled at s_0. In the latter case, because $en(s_{n-1},t_n,V_j)$ is true, we conclude $en(s_i,t_n,V_j)$. Thus at s_i, t_n is enabled with respect to every set in its separation, which implies by the definition of separation that t_n is enabled. The figure now looks like this:

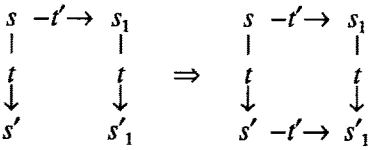


The final step is to require that the enabled transitions in a semistubborn set *accord left* the transitions not in the set in the sense of the following definition.

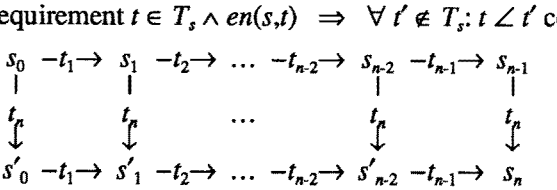
Definition 1.12 Transition t *accords left* transition t', denoted by $t \angle t'$, if and only if for every state s, s', s_1 and s'_1:

$$s \xrightarrow{t} s' \wedge s \xrightarrow{t'} s_1 \xrightarrow{t} s'_1 \;\Rightarrow\; s' \xrightarrow{t'} s'_1 \;\square$$

This definition can be illustrated graphically:



The requirement $t \in T_s \wedge en(s,t) \;\Rightarrow\; \forall\, t' \notin T_s\colon t \angle t'$ completes the figure:



Putting the parts together, we have the following definition of semistubborn sets:

Definition 1.13 A set of transitions $T_s \subseteq T$ is *semistubborn* in *weak* sense at state s, if and only if for every $t \in T_s$

(1) $\neg\, en(s,t) \;\Rightarrow\; \exists\, U \subseteq V\colon \neg\, en(s,t,U) \wedge wrup(t,U) \subseteq T_s$

(2) $en(s,t) \;\Rightarrow\; \forall\, t' \notin T_s\colon t \angle t' \wedge \exists\, V_1,\ldots,V_m\colon sep(t;V_1,\ldots,V_m) \;\wedge$
$(\,\forall\, j{=}1,\ldots,m\colon wrdn(t,V_j) \subseteq T_s \vee wrup(t,V_j) \subseteq T_s\,)$ □

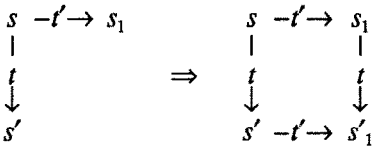
In the derivation above, when ensuring that t_n is enabled at s_1 to s_{n-2}, we allowed an enabled transition belonging to a semistubborn set being disabled by the transitions not in the set, provided that it cannot be enabled again by them. When developing decision procedures for liveness etc. in Section 1.5 we will discuss an algorithm which has a stricter requirement, namely that an enabled transition belonging to a semistubborn set cannot be disabled by the transitions not in the set. Unfortunately, strengthening the requirement removes some possibilities of optimizing semistubborn sets, and is thereby likely to lead to larger semistubborn sets and less state space reduction. Therefore we have chosen to develop two versions of the stubborn set theory, weak (leading to better state space reduction results) and strong (easier to implement, decision procedures for more properties), leaving the tradeoff between them to the implementer of the stubborn set method. When we do not specify which theory is used, the discussion is valid for both.

In conclusion, the difference between the weak and the strong theory is that in the latter enabled transitions belonging to a semistubborn set cannot be disabled by transitions outside the set. We will show in a moment that in the strong theory the relation "$\angle$" can be replaced by a simpler, more symmetric relation "$\leftrightarrow$" which is defined as follows:

Definition 1.14 Transition t *accords with* transition t', denoted by $t \leftrightarrow t'$, if and only if for every state s, s' and s_1 there is a state s'_1 such that

$$s \xrightarrow{t} s' \wedge s \xrightarrow{t'} s_1 \;\Rightarrow\; s' \xrightarrow{t'} s'_1 \wedge s_1 \xrightarrow{t} s'_1 \;\square$$

The definition is symmetric with respect to t and t'. A graphical illustration of the definition is:



According with is equivalent to the conjunction of according left and the requirement that t' cannot disable t. Therefore, in the strong theory it can replace according left in part (2) of the definition of semistubborn sets. The requirement that enabled transitions in a semistubborn set cannot be disabled by transitions not in the set is then automatically satisfied. The resulting strong definition of semistubborn sets is:

Definition 1.15 A set of transitions $T_s \subseteq T$ is *semistubborn* in *strong* sense at state s, if and only if for every $t \in T_s$

(1) $\neg\, en(s,t) \;\Rightarrow\; \exists\, U \subseteq V\colon \neg\, en(s,t,U) \wedge wrup(t,U) \subseteq T_s$

(2) $en(s,t) \;\Rightarrow\; \forall\, t' \notin T_s\colon t \leftrightarrow t'$ □

Although Definitions 1.13 and 1.15 are not equivalent, Theorem 1.11 is valid independent of which one is used.

Semistubborn sets remain semistubborn when transitions outside them occur, thus justifying the name *stubborn*:

Theorem 1.16 If T_s is semistubborn at state s and $s \xrightarrow{t'} s'$, where $t' \notin T_s$, then T_s is semistubborn at s'. □

Proof Let $t \in T_s$. If t is disabled at s, (1) of Definition 1.13 or 1.15 is valid and remains valid for t when t' occurs. If t is enabled at s and remains enabled when t' occurs, then (2) remains valid for t in both theories, as the right hand side of (2) is independent of the state. In the strong theory t cannot be disabled by the occurrence of t'. In the weak theory (Definition 1.13), if t is disabled by the occurrence of t', then by (2) at s' there is V_j such that $\neg\, en(s',t,V_j)$ and $wrup(t,V_j) \subseteq T_s$. Therefore (1) holds for t at s'. □

In both theories, at every state, there are at least two semistubborn sets of transitions, namely the empty set, and the set of all transitions T.

1.3 Stubborn Sets of Transitions

Theorem 1.11 allows the permutation of execution sequences in a certain way, provided that a transition belonging to a semistubborn set is going to occur in the future. However, there is no guarantee of such an occurrence. Quite the contrary: the empty set is always semistubborn. In this section the definition of semistubborn sets is augmented by a requirement which guarantees that there is an enabled transition in the set at least until a transition belonging to the set occurs.

Definition 1.17 A set of transitions $T_s \subseteq T$ is *stubborn* in *weak* sense at state s, if and only if T_s is semistubborn in weak sense at s, and there is a transition $t \in T_s$ such that

$$en(s,t) \wedge wrdn(t,test(t)) \subseteq T_s$$

Transitions with the above property are called *key transitions*. □

In the strong theory the definition of stubborn sets is simpler, because transitions outside a strong semistubborn set cannot disable transitions in the set.

Definition 1.18 A set of transitions $T_s \subseteq T$ is *stubborn* in *strong* sense at state s, if and only if T_s is semistubborn in strong sense at s, and T_s contains an enabled transition. Enabled transitions in T_s are called *key transitions*. □

According to the definitions, in both theories, a stubborn set contains at least one key transition, and key transitions are enabled, remain enabled and retain the key transition property at least until a transition belonging to the stubborn set occurs. Adding Theorem 1.16 to this gives the following result:

Theorem 1.19 If T_s is stubborn at state s and $s -t'\rightarrow s'$, where $t' \notin T_s$, then T_s is stubborn at s'. □

At every state at least T is semistubborn, and the key transition property is trivially satisfied for every enabled transition if $T_s = T$. Therefore:

Theorem 1.20 There are no stubborn sets if there are no enabled transitions. At least T is stubborn if there is an enabled transition. □

1.4 Reduced State Space Generation

In ordinary state space generation, for every encountered state all the transitions which are enabled in it are found and used to generate the immediate successors of the state. The act of doing this is sometimes called *expanding* the state. Expanding a state without enabled transitions amounts to doing nothing, so we limit the consideration to states with enabled transitions.

Algorithm 1.21 *Reduced state space generation* is the following modification of state space generation. When expanding a state with enabled transitions, instead of using all enabled transitions for generating immediate successor states, a stubborn set is found and only the enabled transitions in it are used. □

Because the set of all transitions, T, is stubborn if and only if there is an enabled transition, ordinary state space generation is a special case of reduced state space generation. However, always using T as the stubborn set leads to no reduction in the size of the state space. The intention is to use stubborn sets which contain less enabled transitions than T. This is often possible, as there are often many stubborn sets in a state.

Perhaps surprisingly, always choosing the stubborn set with as few enabled transitions as possible does not necessarily lead to maximal state space reduction results, as shown in [Valmari 88c]. It is not entirely clear what is the best way of choosing stubborn sets. Furthermore, we will encounter decision procedures which state some constraints to stubborn set selection. Fortunately, as long as the constraints are obeyed, the stubborn set selection does not affect the correctness of the analysis results. It affects only the amount of state space reduction achievable. This leaves the implementer of reduced state space generation room to choose the (constrained) stubborn sets in whatever convenient way, making a tradeoff between better reduction results and the ease of implementation. [Valmari 88a,b] contain a linear algorithm which produces "good" strong stubborn sets. In this paper we do not go further into the details of stubborn set selection, but in the (temporary?) absence of better sources refer to [Valmari 88a,b,c].

The notation used in the context of the *reduced state spaces* resulting from reduced state space generation is given below.

Definition 1.22 Let a rule be given specifying a unique stubborn set T_s for each $s \in \mathbb{S}$.

- t is *stubborn-enabled* at s, denoted by $\underline{en}(s,t)$, if and only if $en(s,t)$ and $t \in T_s$.
- $s -t\underline{\rightarrow} s'$ if and only if $s -t\rightarrow s'$ and $t \in T_s$.
- $s \underline{\rightarrow} s' \Leftrightarrow \exists t \in T_s: s -t\rightarrow s'$.
- s –key→ $s' \Leftrightarrow \exists t \in T_s: s -t\rightarrow s'$ and t is a key transition of T_s.
- "$\underline{\rightarrow}$*" is the reflexive and transitive closure of "$\underline{\rightarrow}$". "–key→*" is the reflexive and transitive closure of "–key→".
- The *reduced state space* is the labelled directed graph $(\underline{W},\underline{E},T)$, where

  $\underline{W} = \{s \in \mathbb{S} \mid ss_0 \underline{\rightarrow}^* s\}$

  $\underline{E} = \{(s,t,s') \in \underline{W} \times T \times \underline{W} \mid s -t\underline{\rightarrow} s'\}$
- The *key space* is the labelled directed graph $(\underline{W},E_K,T)$, where

  $E_K = \{(s,t,s') \in \underline{W} \times T \times \underline{W} \mid s -t\underline{\rightarrow} s' \wedge t$ is a key transition of $T_s\}$ □

The key space is a subset of the reduced state space which, in turn, is a subset of the ordinary state space in the sense that $\underline{W} \subseteq W$ and $E_K \subseteq \underline{E} \subseteq E$. In the strong theory all enabled transitions belonging to a stubborn set are its key transitions, and the key space is thus the same as the reduced state space.

The correctness of systems which are intended to terminate is often defined as consisting of two components: the system must terminate, and it must produce the right results upon termination. If the results produced by the system are considered as part of its state, then the latter requirement reduces to the requirement that the state of the system upon termination must satisfy certain properties. For instance, if we are analysing a protocol and request for one message transmission only, we expect the protocol to terminate, and we expect to see the transmitted message being available at the receiver side upon termination.

The following two theorems show that the stubborn set method preserves all terminal states, and the possibility of nontermination. Furthermore, for every occurrence sequence leading to termination, the stubborn set method preserves a sequence which is a permutation of the former. In this sense the stubborn set method preserves everything essential regarding the verification of systems which are intended to terminate. The theorems are valid in both the strong and the weak theory, and they are independent of how stubborn sets are found.

Theorem 1.23

(1) If $s \in W$ and s is a terminal state, then $s \in \underline{W}$ and s is a terminal state of $\underline{W}$. Furthermore, if σ is an occurrence sequence from ss_0 to s, then the reduced state space contains a permutation of σ leading from ss_0 to s.

(2) If s is a terminal state of $\underline{W}$, then $s \in W$ and s is a terminal state (of W). □

Proof (1) Let σ be an occurrence sequence from ss_0 to s. For $i = 0$ to $|\sigma|$ we construct s_i, $\underline{\sigma}_i$ and σ_i such that $fs(\underline{\sigma}_i) = ss_0$, $ls(\underline{\sigma}_i) = s_i = fs(\sigma_i)$, $ls(\sigma_i) = s$, $\underline{\sigma}_i$ belongs to the reduced state space, $|\underline{\sigma}_i| = i$ and $|\sigma_i| = |\sigma| - i$. Choose $s_0 = ss_0$, $\sigma_0 = \sigma$ and $\underline{\sigma}_0$ is the empty occurrence sequence starting at s_0. Now let $i > 0$. Because $|\sigma_{i-1}| > 0$, s_{i-1} has an enabled transition. Let T_{i-1} be the stubborn set used at s_{i-1}. By Theorem 1.20 T_{i-1} is not stubborn in s. By Theorem 1.19 σ_{i-1} contains at least one occurrence of a transition in T_{i-1}. Let t_i be the transition corresponding to the first such occurrence. By Theorem 1.11 there are s_i and σ_i such that $s_{i-1} -t_i\rightarrow s_i$, $fs(\sigma_i) = s_i$, $ls(\sigma_i) = s$, and $|\sigma_i| = |\sigma_{i-1}| - 1$. Because $t_i \in T_{i-1}$, $s_i \in \underline{W}$. $\underline{\sigma}_i$ is $\underline{\sigma}_{i-1}$ with the occurrence of t_i added to its end. $\underline{\sigma}_{|\sigma|}$ is a permutation of σ belonging to the reduced state space and leading from ss_0 to s. That s is a terminal state of $\underline{W}$ follows from $\underline{E} \subseteq E$.

(2) $s \in W$ because $\underline{W} \subseteq W$. s is terminal in W because otherwise Algorithm 1.21 would have chosen a stubborn set T_s at s and T_s contains an enabled transition by Definitions 1.17 and 1.18. □

Theorem 1.24 There is an infinite occurrence sequence in the reduced state space if and only if there is an infinite occurrence sequence in the ordinary state space. □

Proof The "only if" part is obvious, as $\underline{W} \subseteq W$ and $\underline{E} \subseteq E$. To prove the "if" part we construct for arbitrary n an occurrence sequence of length n belonging to the reduced state space using an argument resembling the one used in the proof of the previous theorem. Because T is finite, König's Lemma (see e.g. [Reisig 85] p. 141) gives then an infinite occurrence sequence in the reduced state space.

If there is an infinite occurrence sequence, then there is an infinite occurrence sequence starting at ss_0. Let σ be a prefix of such a sequence such that $|\sigma| = n$. Let s_i, $\underline{\sigma}_i$, σ_i and T_i be defined as in the proof of the previous theorem, with the exception that now $|\sigma_i| \geq |\sigma| - i$ and s is not defined. The difference to the proof of the previous theorem arises from the fact that there is now no guarantee that a transition belonging to T_{i-1} occurs in σ_{i-1}. In such a case, let t_i be a key transition of T_{i-1}. By the definition of key transitions, t_i is enabled at the end state of σ_{i-1}. Let σ'_{i-1} be σ_{i-1} with the occurrence of t_i added to its end. If a transition belonging to T_{i-1} does occur in σ_{i-1}, let $\sigma'_{i-1} = \sigma_{i-1}$. Theorem 1.11 can now be applied to σ'_{i-1}, and the proof continues as before. □

1.5 The "Ignoring" Problem

In the previous section we showed that the stubborn set method preserves sufficient information for the verification of systems which are intended to terminate. However, it is often the case that the system under analysis is not intended to terminate. Such a system is sometimes called a *reactive system*. Theorems 1.23 and 1.24 are of course valid for reactive systems, too, guaranteeing among other things that all deadlocks (that is, unintended terminal states) are preserved by the stubborn set method. On the other hand, there are important properties which are not preserved by the stubborn set method as presented so far.

In the proof of Theorem 1.24 we constructed from a given occurrence sequence an occurrence sequence belonging to the reduced state space. Each construction step consumed either one or zero transition occurrences from the original sequence in the sense of moving it from σ_{i-1} to $\underline{\sigma}_i$. The no consumption case arose when a key transition was added towards the end of σ_{i-1}, because then the key transition was the only one belonging to both T_{i-1} and σ'_{i-1}, and was consequently the one moved to $\underline{\sigma}_i$. Assuming that key transitions are added only a finite number of times, the reduced state space contains a permutation of some finite extension of

the original occurrence sequence. There is, however, no guarantee that key transitions may be added in a way that leads to an end. If the adding of key transitions never ends, the first unconsumed transition of the original sequence is *ignored* in the sense of the following definition.

Definition 1.25 Transition t is *ignored in state* s, if and only if $en(s,t)$ and $\forall\ s' \in \mathbf{S}$: (s –key→* $s' \Rightarrow \neg\ \underline{en}(s',t)$). t is *ignored*, if it is ignored in some $s \in \underline{W}$. □

Ignoring may limit seriously the coverage of the stubborn set method. For instance, if the system under analysis contains one process running in a loop and not interacting with the rest of the system, the stubborn set method may traverse once around the loop and then stop, leaving most of the behaviour of the system uninvestigated. This is cunning behaviour if one is interested in terminal states only, because the existence of such a loop guarantees that there are no terminal states. On the other hand, it is very undesirable behaviour when analysing reactive systems. In the remainder of this section we develop algorithms for detecting and eliminating ignoring. In Section 1.6 we show that many important system properties can be decided using the reduced state space when ignoring has been eliminated.

Assume t is ignored at s and t_k is a key transition at s. By Definitions 1.17 and 1.18 t_k remains enabled when t occurs. Then by the definitions of according left and according with t remains enabled when t_k occurs. As a consequence, t is ignored in the resulting state, too. Repeating the argument reveals that t is enabled and ignored in every state reachable from s in the key space. If the key space is finite, which is the case at least if the ordinary state space is finite, then it must contain a terminal strong component C such that t is enabled but not stubborn-enabled at its every state. Let $s \in C$. Because t is enabled at s there is a stubborn set at s. Thus s has a child in the key space and C is cyclic. On the other hand, if the key space contains a terminal strong component such that t is enabled at one of its states but not stubborn-enabled at any of its states, then t is obviously ignored in s. This gives the following theorem, and a corresponding algorithm for detecting ignoring.

Theorem 1.26 Assume the state space of the system is finite. Transition t is ignored if and only if the key space contains a terminal strong component C such that $\exists\ s \in C$: $en(s,t) \wedge \forall\ s \in C$: $\neg\ \underline{en}(s,t)$. If C exists, it is cyclic, and t is enabled in its every state. □

Algorithm 1.27 Assume the state space of the system is finite. Find the terminal strong components of the key space. For each of them, choose an arbitrary state and compute the set of transitions enabled in it. Subtract from this the set of all transitions which are stubborn-enabled in the component; this set is easily obtainable as it is the set of the transitions labelling the edges of the strong component. The union of the results of the subtractions for each terminal strong component is the set of ignored transitions. □

Strong components can be found in time linear in the number of states and edges in the key space using Tarjan's algorithm, described in [Aho & 74 Chapter 5]. Recognizing the terminal strong components is not difficult, because Tarjan's algorithm finds the strong components in depth first order. Tarjan's algorithm produces the set of states of each strong component, making it easy to compute the set of transitions labelling the output edges of the states. Therefore ignoring can be detected very cheaply, in linear time in the size of the reduced state space.

Detecting ignoring gives information about the coverage of the analysis and makes it possible to repeat the analysis with a modified stubborn set find algorithm, thereby forcing different branches of the state space to be investigated. However, this falls short from what we wish to achieve. In the strong theory of stubborn sets the key space is the same as the reduced state space. Together with some nice properties of Tarjan's algorithm, this fact enables cheap detection and remedy of ignoring at analysis time, leading to a reduced state space where ignoring does not occur.

Algorithm 1.28 Assume the state space of the system is finite, and the strong stubborn set definition is used. Generate the reduced state space in depth-first order and apply Tarjan's algorithm along with the generation. Attach an initially empty set of transitions T_x to each state generated. Tarjan's algorithm's stack of found nodes not belonging to a completed strong component will get extra short cut links as described below; therefore we rename it *T-list*. These links are used only for ignoring detection, and affect Tarjan's algorithm in no way.

Whenever a terminal strong component C is found, compute T_u, the set of transitions used in it as shown soon. Let s_c denote the current state, and T_e the set of enabled transitions in s_c. There are ignored transitions in the current branch of the reduced state space if and only if $T_e - T_u \neq \emptyset$. Mark the component ready as required by Tarjan's algorithm and backtrack from it only if there are no ignored transitions. If there are ignored transitions, add T_u to the T_x of s_c, and create a short cut link from the current top of the T-list to s_c (note that the states belonging to C are the states above and including s_c in the T-list). Then choose a new stubborn set such that at least one of its key transitions is ignored, that is, in $T_e - T_u$, and continue depth-first analysis.

The set of used transitions is computed by traversing the T-list from top to and including s_c, using short cut links where available. During the traversal, for states not adjacent to a short cut link, add their sets of stubborn-enabled transitions to T_u. For states adjacent to the tail but not to the head of a short cut link add nothing to T_u, and for states adjacent to the head of a short cut link add their T_x to T_u. (Note that the T_x sets and the short cut links are used at most once, and before the corresponding states are popped from the T-list by Tarjan's algorithm.) □

The new edges introduced to the reduced state space by this algorithm do not confuse Tarjan's algorithm, as they are introduced just before Tarjan's algorithm would become aware of their non-existence. The short cut links and the T_x sets are used for speeding up the computation of the set of ignored transitions, taking advantage of the previously computed T_u sets. Because of them, each state's set of stubborn-enabled transitions is computed at most once. Short cut links are used at most once. The remaining operations related to ignoring detection are done once for every ignoring recovery, and every ignoring recovery introduces at least one new edge to the reduced state space. Thus the cost of this algorithm is at most proportional to the size of the (final) reduced state space multiplied by the number of transitions in the v/t-system.

1.6 Stubborn Sets and Reactive Systems

Let us return to the argument in the proof of Theorem 1.24. Assuming that ignoring does not occur, it is always possible to select key transitions in such a way that as i grows, σ_i eventually contains an occurrence of a transition belonging to T_i. Continuing the procedure, the transition occurrences belonging to the original occurrence sequence are eventually all consumed, justifying the following theorem:

Theorem 1.29 Assume ignoring does not occur. Let σ be a finite occurrence sequence starting at s, where $s \in \underline{W}$. There are $s' \in \underline{W}$, a finite extension of σ called σ' leading from s to s', and a permutation of σ' called $\underline{\sigma}$ leading from s to s' and belonging to the reduced state space. Each transition occurrence in $\underline{\sigma}$ either has a corresponding transition occurrence in σ, or is the occurrence of a key transition. □

This theorem has corollaries which are important regarding the analysis and verification of reactive systems.

Corollary 1.30 If ignoring does not occur, a transition occurs in the reduced state space if and only if it occurs in the ordinary state space. □

This corollary allows the use of the well known Petri net fact technique (see e.g. [Reisig 85] p. 56) for verifying invariant properties with the stubborn set method, provided that ignoring is eliminated. A *fact* is a transition t whose enabling condition $E(s) = en(s,t)$ is expected to be never satisfied. That is, $\neg E$ is intended to be an invariant property. The generation of the current branch of the state space may be quitted when an enabled fact is found. The quitting causes the state violating the fact appear as a terminal state in the reduced state space, thus confusing the ignoring elimination algorithm (Algorithm 1.28). Because of this, if invariant properties are violated, it is guaranteed only that at least one violation is found. Furthermore, other analyses (terminal state detection etc.) are guaranteed to produce correct results only if there are no violations of invariant properties. As a consequence, some errors of the system under analysis are not necessarily reported, if there is a violated invariant. This is often sufficient, because the violation of the invariant is reported, and the remaining errors will be found after the error causing the violation of the invariant is fixed.

There is also a way to restore the ignoring elimination algorithm. This is done by not taking fact violation terminal states into account when checking in the algorithm whether a strong component is terminal. Key transitions leading from a state s to a fact violation terminal state cannot any more be considered key transitions in s. If the weak definition of stubborn sets is used, this may lead to the need of finding and using a new stubborn set in s to ensure that at every state with transitions which are enabled but not stubborn-enabled, there is a key transition leading to a proper successor state.

Transition t is *live* in the Petri net sense of the word, if and only if for every $s \in W$ there is an occurrence sequence starting at s and containing an occurrence of t. We can say that t is *live in the reduced state space* if and only if for every $s \in \underline{W}$ there is an occurrence sequence starting at s, belonging to the reduced state space and containing an occurrence of t.

Corollary 1.31 If ignoring does not occur, a transition is live (in the Petri net sense of the word) if and only if it is live in the reduced state space. That is, if ignoring does not occur, stubborn set state space reduction preserves liveness. □

Proof Assume t is live. Let $s \in \underline{W}$. Take an occurrence sequence σ starting at s and ending with the occurrence of t. By Theorem 1.29 the reduced state space contains a permutation of an extension of σ starting at s. Therefore it contains an occurrence sequence starting at s and containing the occurrence of t. t is thus live in the reduced state space. Assume now that t is not live. There is then a state s such that t is disabled in every state reachable from s. Take an occurrence sequence σ from ss_0 to s. By Theorem 1.29 there is a state $s' \in \underline{W}$ such that s' is reachable from s. t is disabled in every state reachable from s', thus it cannot occur in any occurrence sequence starting at s' in the reduced state space. As a consequence, t is not live in the reduced state space. □

A *livelock* can be thought of as a mode of operation where the system is doing something but what it is doing is unproductive. We can distinguish between two kinds of livelocks: those which can be exited, and those which cannot. The latter correspond to cyclic terminal strong components C of the state space of the system such that the set of transitions occurring at the states of C is not what is expected. The following corollary shows that such livelocks are preserved by the reduced state space generation method if ignoring does not occur.

Corollary 1.32 Assume the state space of the system is finite and ignoring does not occur.

(1) Let $C \subseteq W$ be a terminal strong component. There is a terminal strong component $\underline{C}$ in the reduced state space such that $\underline{C} \subseteq C$ and for every transition t, if and only if $\exists\, s, s' \in C$: $s \xrightarrow{t} s'$, then $\exists\, \underline{s}, \underline{s}' \in \underline{C}$: $\underline{s} \xrightarrow{t} \underline{s}'$.

(2) Let $\underline{C} \subseteq \underline{W}$ be a terminal strong component of the reduced state space. There is a terminal strong component C such that $\underline{C} \subseteq C$ and for every transition t, if and only if $\exists\, \underline{s}, \underline{s}' \in \underline{C}$: $\underline{s} \xrightarrow{t} \underline{s}'$, then $\exists\, s, s' \in C$: $s \xrightarrow{t} s'$. □

Proof (1) Let $s \in C$ and σ be an execution sequence from ss_0 to s. Every state reachable from s belongs to C. Consider the s' the existence of which is implied by Theorem 1.29. $s' \in C$ and $s' \in \underline{W}$. Let $\underline{C}$ be a terminal strong component of the reduced state space such that $\exists\, \underline{s} \in \underline{C}$: $s' \underline{\rightarrow}^* \underline{s}$. We see that $\underline{C} \subseteq C$. Now, let $\underline{s}$ be any element of $\underline{C}$ and let $s_1, s'_1 \in C$ and $t \in T$ such that $s_1 -t\rightarrow s'_1$. Because C is a strong component, $\underline{s} \rightarrow^* s_1 -t\rightarrow s'_1$. By Theorem 1.29 there are $\underline{s}_1, \underline{s}'_1 \in \underline{W}$ such that $\underline{s}_1 -t\underline{\rightarrow} \underline{s}'_1$. $\underline{s}_1, \underline{s}'_1 \in \underline{C}$, because $\underline{s} \in \underline{C}$ and $\underline{C}$ is a terminal strong component of the reduced state space. Now let $\underline{s}_1, \underline{s}'_1 \in \underline{C}$ such that $\underline{s}_1 -t\underline{\rightarrow} \underline{s}'_1$. $\underline{s}_1, \underline{s}'_1 \in C$ and $\underline{s}_1 -t\rightarrow \underline{s}'_1$ because $\underline{C} \subseteq C$ and $\underline{E} \subseteq E$.

(2) Choose $\underline{s} \in \underline{C}$. Let C be a terminal strong component of the ordinary state space such that $\exists\, s \in C$: $\underline{s} \rightarrow^* s$. As above, Theorem 1.29 implies that there is $\underline{s}' \in \underline{W} \cap C$ such that $\underline{s} \underline{\rightarrow}^* \underline{s}'$ and $s \rightarrow^* \underline{s}'$. Thus $\underline{s}' \in \underline{C}$, and because $\underline{C}$ is a strong component in the reduced state space, we get $\underline{s}' \underline{\rightarrow}^* \underline{s}$. Therefore $\underline{s}$, s and $\underline{s}'$ belong to the same strong component of the ordinary state space, that is, to C. We conclude $\underline{C} \subseteq C$. The claim "for every transition t, if and only if $\exists\, \underline{s}, \underline{s}' \in \underline{C}$: $\underline{s} -t\underline{\rightarrow} \underline{s}'$, then $\exists\, s, s' \in C$: $s -t\rightarrow s'$" is proven as above. □

It is known that the stubborn set method as presented so far does not preserve livelocks which can be exited. The stubborn set method can be modified to cover the analysis of such livelocks, too, but it is beyond the scope of this paper.

The last result in this chapter is about preserving the language generated by a system. We assume that some, but not necessarily all transitions of the system have been given a symbol from some alphabet. Then the language generated by the system is the set of strings generated by the occurrence sequences of the system.

Definition 1.33 Let Σ be a set of symbols. Let $(V,T,type,next,ss_0)$ be a v/t-system and α be a function from T to Σ^* such that $\forall\, t \in T$: $|\alpha(t)| \leq 1$. Let $\sigma = s_0 -t_1\rightarrow \ldots -t_n\rightarrow s_n$.

- $t \in T$ is *visible*, if and only if $|\alpha(t)| = 1$. Otherwise t is *invisible*. The set of visible transitions is denoted by T_V.
- The *word* generated by σ is the string $\alpha(t_1)\alpha(t_2)\ldots\alpha(t_n)$.
- The *language* generated by $(V,T,type,next,ss_0)$ with Σ and α is the set of words generated by all finite occurrence sequences of $(V,T,type,next,ss_0)$ starting at ss_0. □

Assuming that the selection of stubborn sets is constrained in a certain way and ignoring does not occur, the stubborn set method preserves the language generated by the system.

Theorem 1.34 Assume that ignoring does not occur, and the stubborn sets used by Algorithm 1.21 satisfy the following for every stubborn set T_s used at state s with an enabled transition:

$$(\exists\, t \in T_s \cap T_V\colon en(s,t)) \;\Rightarrow\; T_V \subseteq T_s.$$

If there is an occurrence sequence σ starting at ss_0 generating the word x, then there is an occurrence sequence $\underline{\sigma}$ in the reduced state space starting at ss_0 and generating x, and vice versa. □

Proof The "vice versa"-part is obvious. Regarding the other part, consider the $\underline{\sigma}$ of Theorem 1.29. We prove that in the construction of $\underline{\sigma}$ the order of occurrences of visible transitions does not change, implying that $\underline{\sigma}$ has a prefix generating x. Consider again the argument in the proof of Theorem 1.24. Each construction step moves a transition occurrence from σ'_{i-1} to $\underline{\sigma}_i$. If the transition is invisible, moving it does not change the order of occurrences of visible transitions. Assume now the moved transition t is visible. It is obviously enabled at $s_{i-1} = ls(\underline{\sigma}_{i-1})$ and belongs to T_{i-1}, where T_{i-1} is the stubborn set used at s_{i-1}. By the assumption of the theorem we get $T_V \subseteq T_{i-1}$. The moved transition is the first of those in σ'_{i-1} which belong to T_{i-1}, thus it is the first of those in σ'_{i-1} which belong to T_V. As a result, moving it does not change the order of occurrences of visible transitions. □

In conclusion, if the strong definition of stubborn sets is used, Algorithm 1.28 can be used to eliminate ignoring. When ignoring is eliminated, the stubborn set method preserves liveness in the Petri net sense of the word, and can be used to check invariant properties. Furthermore, it preserves livelocks defined as unintended terminal cyclic strong components. Assuming an extra constraint on the selection of stubborn sets the method preserves the language generated by the system.

2. STUBBORN SETS OF PETRI NETS

In this chapter the theory of Chapter 1 is applied to three different classes of Petri nets by adapting the definitions of stubborn sets (Definitions 1.17 and 1.18). The classes are elementary nets (see [Rozenberg & 86] or [Thiagarajan 87] for definition), place/transition nets with finite capacity of places ([Reisig 85] or [Reisig 87]), and coloured Petri nets ([Jensen 87]).

Since we have used the symbols s, s', s_1,... to denote states, we denote Petri net places by symbols starting with p, to avoid confusion. We continue to use the predicate *en* for denoting that a transition is enabled. Otherwise in this chapter we use familiar Petri net notation where convenient, such as $\bullet t$ and $t\bullet$ for the sets of the input and output places of transition t, respectively. In particular, markings (the Petri net equivalent of states) are denoted by the usual M, even when they are used mixed with variable/transition system notation. That is, we write, for instance, $next(M,t) = M'$.

2.1 Stubborn Sets of Elementary Nets

An elementary Petri net system as defined in [Rozenberg & 86] or [Thiagarajan 87] can be thought of as a variable/transition system where places correspond to variables, transitions correspond to transitions, markings correspond to states and the initial marking corresponds to the initial state. The type of each variable is $\{0,1\}$, representing the corresponding place being empty or marked, respectively. The topology of the net defines the next state function in an obvious way:

$next(M,t)$ = *undefined*, if and only if $\exists\, p \in \bullet t\colon M(p) = 0 \vee \exists\, p \in t\bullet\colon M(p) = 1$

otherwise $next(M,t) = M'$, where

$$\forall\, p \in \bullet t\colon M'(p) = 0 \;\wedge\; \forall\, p \in t\bullet\colon M'(p) = 1 \;\wedge\; \forall\, p \notin \bullet t \cup t\bullet\colon M'(p) = M(p)$$

Because of the form of the enabling condition of an elementary net transition, it has a separation (see Definition 1.10) consisting of single places:

$sep(t;\{p_1\},\ldots,\{p_n\})$, if $test(t) = \{p_1,\ldots,p_n\}$

The most natural choices for the test, write and read sets of transition t satisfying Definition 1.6 and the corresponding connection set are

if $\bullet t \cap t\bullet = \emptyset$: $test(t) = wr(t) = conn(t) = \bullet t \cup t\bullet \;\wedge\; rd(t) = \emptyset$

if $\bullet t \cap t\bullet \neq \emptyset$: $test(t) = wr(t) = rd(t) = conn(t) = \emptyset$

To keep our formulas simpler, from now on we assume that for all transitions t, $\bullet t \cap t\bullet = \emptyset$. (That is, we ban *side places*.) This is not an essential restriction, since transitions not satisfying this are never enabled.

Transition t is enabled with respect to a subset of places P' in the sense of Definition 1.8 if and only if

$$\forall\, p \in P' \cap \bullet t\colon M(p) = 1 \wedge \forall\, p \in P' \cap t\bullet\colon M(p) = 0$$

A possible choice for write up and write down sets of transition t (see Definition 1.9) is

$wrup(t,P') = \bullet(P' \cap \bullet t) \cup (P' \cap t\bullet)\bullet$

$wrdn(t,P') = (P' \cap \bullet t)\bullet \cup \bullet(P' \cap t\bullet)$

Every elementary net transition accords left (Definition 1.12) every other elementary net transition, as can be verified by algebraic manipulation based on the definitions. We can write weak definitions of semistubborn and stubborn sets of elementary net transitions (see Definitions 1.13 and 1.17):

Definition 2.1 Subset of transitions T_s of an elementary Petri net is *semistubborn* in the *weak* sense at marking M, if and only if for every $t \in T_s$

$$\neg\, en(M,t) \;\Rightarrow\; \exists\, p \in {\bullet}t\colon M(p) = 0 \wedge {\bullet}p \subseteq T_s \;\vee\; \exists\, p \in t{\bullet}\colon M(p) = 1 \wedge p{\bullet} \subseteq T_s$$

$$en(M,t) \;\Rightarrow\; \forall\, p \in {\bullet}t \cup t{\bullet}\colon {\bullet}p \subseteq T_s \vee p{\bullet} \subseteq T_s$$

T_s is *stubborn* in the *weak* sense at M, if and only if it is semistubborn in the weak sense at M, and

$$\exists\, t \in T_s\colon en(M,t) \wedge ({\bullet}t){\bullet} \cup {\bullet}(t{\bullet}) \subseteq T_s \;\square$$

Strong definitions of semistubborn and stubborn sets (see Definitions 1.15 and 1.18) have the following form:

Definition 2.2 Subset of transitions T_s of an elementary Petri net is *semistubborn* in the *strong* sense at marking M, if and only if for every $t \in T_s$

$$\neg\, en(M,t) \;\Rightarrow\; \exists\, p \in {\bullet}t\colon M(p) = 0 \wedge {\bullet}p \subseteq T_s \;\vee\; \exists\, p \in t{\bullet}\colon M(p) = 1 \wedge p{\bullet} \subseteq T_s$$

$$en(M,t) \;\Rightarrow\; ({\bullet}t){\bullet} \cup {\bullet}(t{\bullet}) \subseteq T_s$$

T_s is *stubborn* in the *strong* sense at M, if and only if it is semistubborn in the strong sense at M, and

$$\exists\, t \in T_s\colon en(M,t) \;\square$$

2.2 Stubborn Sets of Place/Transition Nets

The interpretation of place/transition nets (as defined in [Reisig 85] or [Reisig 87], for instance) as variable/transition systems follows the same guidelines as the interpretation of elementary nets. Since places may now contain more than one token, the type of the variable corresponding to place p is $\{0,\ldots,K(p)\}$ where $K(p)$ is the capacity of p, or $\{0,\ldots\}$ if the capacity of p is infinite. The next state function is defined by the transition rule of place/transition nets as follows. Let $W(x,y)$ be the weight of the arc from place or transition x to transition or place y, or 0, if there is no arc.

$$en(M,t) \;\Leftrightarrow\; \forall\, p \in P\colon W(p,t) \le M(p) \le K(p) - W(t,p)$$

$$en(M,t) \;\Rightarrow\; next(M,t) = M', \quad \text{where } \forall\, p \in P\colon M'(p) = M(p) - W(p,t) + W(t,p)$$

As with elementary nets, $sep(t;\{p_1\},\ldots,\{p_n\})$ holds for every transition t if $test(t) = \{p_1, \ldots, p_n\}$.

Let us denote the set of places with finite capacity by P_K. Test, read, write and connection sets may be defined as below:

$$test(t) = {\bullet}t \cup (t{\bullet} \cap P_K)$$

$$wr(t) = rd(t) = \{p \in P \mid W(p,t) \neq W(t,p)\}$$

$$conn(t) = {\bullet}t \cup t{\bullet}$$

Enabling with respect to a set of places can be defined as the restriction of the enabling condition to the set of places in question.

The definitions of smallest possible write down and write up sets of general place/transition nets are quite complicated, and thus so are also the definitions of (semi)stubborn sets. For simplicity, in the remainder of this section we assume that the capacity of each place is infinite. If stubborn sets of place/transition nets with capacity constraints are needed, one can use the general ideas of this paper to derive the necessary definitions.

With the assumption of the absence of capacity limitations, write up and write down sets with respect to individual places may be defined as follows:

$$wrup(t,\{p\}) = \{t' \in T \mid W(t',p) > W(p,t') < W(p,t)\}$$

$$wrdn(t,\{p\}) = \{t' \in T \mid W(p,t') > W(t',p) < W(p,t)\}$$

Theorem 2.3 Let t and t' be transitions of a place/transition net where the capacity of each place is infinite. $t \angle t'$, if and only if

$$\forall p \in P: W(t,p) \geq \min(\, W(p,t), W(p,t'), W(t',p)\,) \quad \square$$

Proof "if" part: Assume that $M \xrightarrow{t} M'$ and $M \xrightarrow{t'} M_1 \xrightarrow{t} M'_1$; we have to prove that $M' \xrightarrow{t'} M'_1$. By the assumption, for all $p \in P$, (1) $M(p) \geq W(p,t)$, (2) $M(p) \geq W(p,t')$, and (3) $M(p)-W(p,t')+W(t',p) \geq W(p,t)$. Also $M'(p) = M(p)-W(p,t)+W(t,p)$. If $W(t,p) \geq W(p,t)$, then $M'(p) \geq M(p) \geq W(p,t')$ by (2). If $W(t,p) \geq W(p,t')$, then $M'(p) \geq W(t,p) \geq W(p,t')$ by (1). If $W(t,p) \geq W(t',p)$, then $M'(p) \geq M(p)-W(p,t)+W(t',p) \geq W(p,t')$ by (3). Thus t' is enabled at M'. $M' \xrightarrow{t'} M'_1$, since the net result of the occurrence of two place/transition net transitions in sequence is independent of their order of occurrence.

"only if" part: Assume that there is a place p so that $W(t,p) < \min(\, W(p,t), W(p,t'), W(t',p)\,)$. Consider marking M where $M(p) = W(p,t)+W(p,t')-W(t,p)-1$, and the marking of the remaining places p' is $M(p') = \max(\, W(p',t), W(p',t'), W(p',t')-W(t',p')+W(p',t)\,)$. $M(p)$ is well defined, because $W(t,p) < W(p,t)$ implies $M(p) \geq W(p,t') \geq 0$. There are M', M_1 and M'_1 so that $M \xrightarrow{t} M'$ and $M \xrightarrow{t'} M_1 \xrightarrow{t} M'_1$, as can be verified by algebraic manipulation based on the assumption about $W(t,p)$. However, t' is not enabled at M' because of p. $\square$

Using the above definitions and Theorem 2.3, we get the following definitions:

Definition 2.4 Subset of transitions T_s of a place/transition net with infinite capacity of places is *semistubborn* in the *weak* sense at marking M, if and only if for every $t \in T_s$

$$\neg\, en(M,t) \;\Rightarrow\; \exists p \in P: M(p) < W(p,t) \wedge \forall t' \notin T_s: W(p,t') \geq \min(\, W(t',p), W(p,t)\,)$$

$$en(M,t) \;\Rightarrow\; \forall p \in P: (\, \forall t' \notin T_s: \min(\, W(t,p), W(t',p)\,) \geq \min(\, W(p,t), W(p,t')\,) \vee$$
$$\forall t' \notin T_s: \min(\, W(t,p), W(p,t')\,) \geq \min(\, W(p,t), W(t',p)\,)\,)$$

T_s is *stubborn* in the *weak* sense at M, if and only if it is semistubborn in the weak sense at M, and

$$\exists t \in T_s: en(M,t) \wedge \forall t' \notin T_s: \forall p \in P: W(t',p) \geq \min(\, W(p,t'), W(p,t)\,) \quad \square$$

Definition 2.5 Subset of transitions T_s of a place/transition net with infinite capacity of places is *semistubborn* in the *strong* sense at marking M, if and only if for every $t \in T_s$

$$\neg\, en(M,t) \;\Rightarrow\; \exists p \in P: M(p) < W(p,t) \wedge \forall t' \notin T_s: W(p,t') \geq \min(\, W(t',p), W(p,t)\,)$$

$$en(M,t) \;\Rightarrow\; \forall p \in P: \forall t' \notin T_s: \min(\, W(t,p), W(t',p)\,) \geq \min(\, W(p,t), W(p,t')\,)$$

T_s is *stubborn* in the *strong* sense at M, if and only if it is semistubborn in the strong sense at M, and

$$\exists t \in T_s: en(M,t) \quad \square$$

It is interesting to compare Definition 2.4 to Definitions 2.1 and 2.3 of [Valmari 88a], as both define semistubborn and stubborn sets of place/transition net transitions, and both assume that there are no capacity constraints. The definitions in [Valmari 88a] were derived directly for place/transition nets, using heuristics to develop different strategies for establishing conditions guaranteeing Theorem 1.11, and using algebraic manipulations to find the corresponding definition. The definitions in [Valmari 88a] are not intuitive, and, as a matter of fact, not equivalent to Definition 2.4. This is because stubborn set theory aims at finding a sufficient and statically computable condition for guaranteeing Theorem 1.11, and different approaches may lead to slightly different results.

2.3 Stubborn Sets of Coloured Petri Nets

Coloured Petri nets are a high level net class defined in [Jensen 87]. Tokens of a coloured Petri net may have an identity ("colour"). Transitions may have different occurrence modes (they, too, are called "colours"), and the tokens the transition consumes from its input places and produces for its output places may be defined by arbitrary functions from occurrence colours to multisets of token colours.

There are basically two approaches to interpreting coloured Petri nets as variable/transition systems, one concentrating to the places of the coloured net as such, the other to the places of the corresponding unfolded net. In the first approach each place is thought of as a single variable the type of which is defined so that it covers all the multisets of tokens that may be stored in the place. Each occurrence colour of each transition is thought of as a unique transition. Test, write, write down etc. sets and stubborn sets are defined putting the available information of the relationships between transitions and places to as good use as conveniently possible. For instance, an easy but crude way of defining the test, read, write and connection sets of transitions is to define $test(t) = wr(t) = rd(t) = conn(t) = {}^\bullet t \cup t^\bullet$ for each transition t. The observation that for a particular transition t and place p the functions defining the tokens consumed from and produced for p by t are equal, can be taken advantage of by removing p from $wr(t)$, because then t only tests the presence of some tokens in p without modifying the contents of p. An easy definition of write up and write down sets would be $wrup(t,P') = wrdn(t,P') = wr(test(t) \cap P')$ for each transition t and set of places P'. With these definitions, the weak and strong definition of semistubborn sets agree and lead to the following simple definition (originally [Valmari 88b]):

A set of transitions $T_s \subseteq T$ is *semistubborn* at marking M, if and only if for every $t \in T_s$

$$\neg\, en(M,t) \;\Rightarrow\; \exists\, P' \subseteq test(t): \neg\, en(M,t,P') \wedge wr(P') \subseteq T_s$$

$$en(M,t) \;\Rightarrow\; conn(wr(t)) \cup wr(conn(t)) \subseteq T_s$$

The definition of stubborn sets adds the requirement of the presence of an enabled transition in the set to this. This definition is quite simple, but it leads to unnecessarily large stubborn sets and thus does not give the best possible state space reduction results. Better reduction results are achieved if the information of the relationships between places and transitions is utilised more carefully.

In the other interpretation approach each place is thought of as consisting of several variables, each corresponding to one possible token colour. That is, the coloured Petri net is interpreted as being only a concise description of the corresponding unfolded place/transition net. Then the definitions of stubborn sets of place/transition nets are used. This approach does not imply that the unfolding should be actually done; rather, it states merely that the algorithm finding stubborn sets should interpret the coloured Petri net as if it were a condensed description of the corresponding unfolded place/transition net. The advantage of this approach over the first interpretation is that better reduction results may be achieved, but the disadvantage is that algorithms searching stubborn sets become more complicated and consume more time. At the time of writing it is an open research problem as to whether using this interpretation allows stubborn sets of coloured Petri net transitions to be computed with effort proportional to the size of the coloured Petri net rather than proportional to the size of the corresponding unfolded place/transition net.

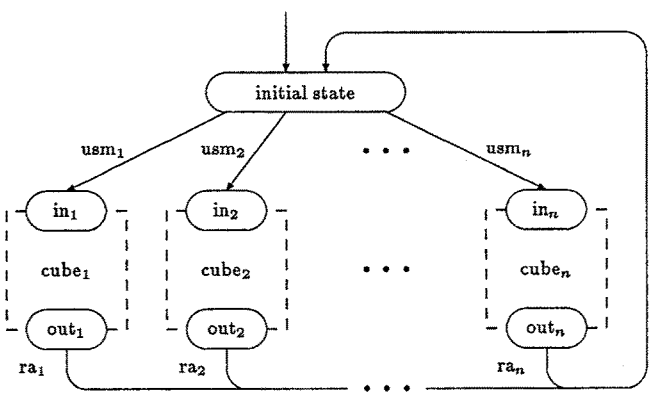
In conclusion, the best interpretation of coloured Petri nets as v/t-systems depends on the available information on what there may be in arc functions and transition guards. The ideas of this section should be applicable to most other high level net classes, too, including Genrich's predicate/transition nets [Genrich 87] and Numerical Petri Nets [Wheeler 85].

3. STUBBORN SETS AND EQUIVALENT MARKINGS

In this chapter we compare with the aid of an example, the stubborn set state space reduction method with Jensen's equivalent marking method as defined in [Jensen 87]. The example is a data base system originally presented by Genrich and Lautenbach. We use the version in [Jensen 87] p. 269. It consists of $n \geq 2$ data base managers, which modify the data base and send and receive messages to each other to ensure that they have the same idea about the contents of the data base. The model concentrates on the message exchange. In particular, the modification operations are not modelled.

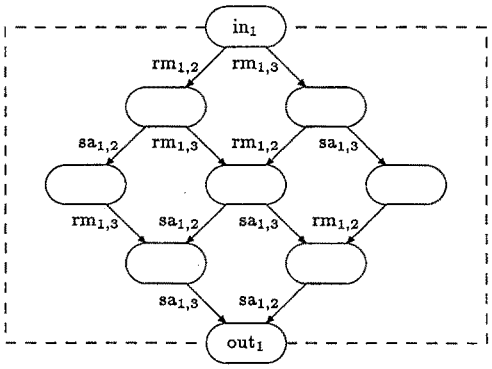
Initially all managers are in inactive state. Then one of them, any one, modifies his data base. This is modelled by transition "update and send messages" which reserves the data base for that manager and sends a message to every other manager. Then all the other managers concurrently perform a two step sequence, where the first step corresponds to the reception of the message, and the second step corresponds to the sending of an acknowledgement. When all acknowledgements are available, the manager who started the game reads them, releases the data base so that the other managers can modify it if they wish, and returns to the inactive state.

The state space of the data base system has a symmetrical structure which makes it easy to compute its size when no reduction method, the stubborn set method, the equivalent marking method or both are used. The ordinary state space (no reduction method used) is as in the figure below, where $\mathbf{usm}_i$ is the occurrence of transition "update and send messages" with occurrence colour i, $\mathbf{ra}_i$ is the occurrence of transition "receive acknowledgements" with occurrence colour i, $\mathbf{rm}_{i,j}$ is the occurrence of transition "receive message" with occurrence colour "j receives from i", $\mathbf{sa}_{i,j}$ is the occurrence of transition "send acknowledgement" with occurrence colour "j sends to i", $\mathbf{cube}_i$ is a subspace resembling an $n-1$ dimensional hypercube (see the example below), and $\mathbf{in}_i$ and $\mathbf{out}_i$ are the input and output states of $\mathbf{cube}_i$. $\mathbf{cube}_1$ for $n=3$ is shown in the following page.



An $n-1$ dimensional hypercube with edges of length 2 has 3^{n-1} vertices and $2(n-1)3^{n-2}$ edges. Therefore the state space of the example contains $n3^{n-1}+1$ vertices and $2n(n-1)3^{n-2}+2n$ edges.

When the stubborn set state space reduction method is used and stubborn sets are computed as if from the unfolded place/transition net corresponding to the coloured Petri net representing the data base system (see Section 2.3), the stubborn set method takes advantage of the fact that the receive message — send acknowledgement sequences of different managers are concurrent with each other. Therefore it simulates only one path through each $\mathbf{cube}_i$. This



is true independent of whether the weak or strong definition of stubborn sets is used. It is, however, required that the stubborn sets used are not larger than necessary. Under these assumptions, the numbers of vertices and edges are $2n^2-n+1$ and $2n^2$.

In this example, the equivalent marking method takes advantage of the fact that the system is symmetric with respect to the data base managers. That is, it is not necessary to know the identity of the manager that is at a given state; it is sufficient to know only the number of managers at each state. Therefore only one of $\mathbf{cube}_i$ is generated. Furthermore, within each $\mathbf{cube}_i$ only the states (j,k,l) are generated, where j is the number of managers which have not yet received the message, k is the number of managers which have received the message but have not acknowledged it, l is the number of managers which have acknowledged their message, and $j+k+l = n-1$. There are thus $(1/2)n^2+(1/2)n+1$ vertices. A bit more complicated analysis reveals that the number of edges is n^2-n+2.

Finally, when both reduction methods are used at the same time, only one of $\mathbf{cube}_i$ and only one path through it are generated. Therefore the number of vertices and edges are both $2n$. These results are summarized in the following table, where we have included only the most significant term of each formula:

	vertices	edges
no reduction	$n3^{n-1}$	$2n^2 3^{n-2}$
stubborn sets	$2n^2$	$2n^2$
equivalent markings	$\frac{1}{2}n^2$	n^2
both	$2n$	$2n$

In this example, the size of the state space is exponential in n if neither reduction method is used, quadratic in n if either reduction method is used, and linear in n if both are used at the same time. From this, one should not conclude that the stubborn set method and the equivalent marking method are roughly equally strong, but that they are incomparable, that they take advantage of different aspects of the system under analysis. The equivalent marking method works well when there is a suitable symmetry available; the stubborn set method works well when there is concurrency. In this example there are both, thus it is advantageous to use both reduction methods.

4. CONCLUSION

In this paper we have developed a general state space reduction theory and applied it to both low level and high level Petri nets. From the implementer's point of view the theory is very flexible. Several sets in the theory can be enlarged at will without invalidating the theorems

in this paper, including write down and write up sets. Therefore the implementer may choose how carefully the relationships between transitions are captured by the definition of stubborn sets. More detailed definitions give better state space reduction results, but if the ease of implementation is preferred, the use of crude definitions is perfectly legal.

There are two versions of the theory: strong and weak. With both versions, the reduced state space contains every terminal state of the system under analysis, and facilitates the detection of nontermination. This information is sufficient to verify the total correctness of systems which are intended to terminate. If nontermination occurs, a certain kind of fairness problem called *ignoring* can take place and limit the coverage of the analysis of reactive systems. Ignoring can be cheaply detected from the reduced state space. If the strong theory is used, ignoring can be eliminated altogether without undue cost using the algorithm given in the paper. If ignoring does not occur or is eliminated, reduced state space generation can be used to decide the liveness of transitions (in the Petri net sense of the word), to detect livelocks (defined as unintended terminal cyclic strong components of the state space) and to check invariant properties. Furthermore, with a small modification the stubborn set method can be forced to preserve the language generated by attaching a symbol to some (but not to all) transitions. The preserving of linear temporal logic formulas is discussed in [Valmari 90], and also the failure set semantics [Brookes & 84] of systems can be preserved.

We compared the stubborn set method to Jensen's equivalent marking method through an example. The conclusion was that both methods are capable of giving good reduction results (the size of the state space reduced from exponential to quadratic in system size with either method in the example), but they are incomparable in the sense that they take advantage of different aspects of the system under analysis. In the example the use of both methods at the same time led to even better results (linear).

As reported before [Valmari 88a], there is a test implementation which uses the strong form of the definitions and makes a very crude analysis of the relationship between transitions. A more serious attempt of applying the stubborn set method is currently being conducted by Telecom Australia. It is developing a protocol engineering tool called Toras [Wheeler & 90]. Among other features, it can generate ordinary and reduced state spaces using the strong stubborn set method. It contains an ignoring elimination algorithm resembling Algorithm 1.28. Furthermore, it contains the language preserving and the failure set semantics preserving stubborn set methods.

The state space generator tool of Toras has been divided to two modules, one embodying knowledge about state space generation and the various stubborn set methods, the other corresponding to the semantics of the concurrency model which is used to represent the systems under analysis. The idea is that the tool can be adapted to various concurrency formalisms by changing the latter module. The modules communicate with each other at the level of variable/transition systems. The above mentioned state space generation algorithms have been implemented, and at the time of writing there is a concurrency model module for place/transition systems without capacity constraints. Another for a certain version of Numerical Petri Nets [Wheeler 85] is partially implemented. In the initial speed tests a P/T-system version of the 100 philosopher system ($\approx 10^{47}$ states) was analysed in 20 minutes CPU on a Sun 3/60, with the result of generating 29 702 states using the basic strong stubborn set method [Wheeler & 90].

ACKNOWLEDGEMENTS

As many of the ideas of this paper have been developed during my Ph.D. thesis work, I would like to thank here my supervisor, referees and opponents of the thesis, namely Professor Kurki-Suonio of Tampere University of Technology and Drs Eike Best, Pekka Orponen and Joachim Parrow. Later the comments by Geoff Wheeler of Telecom Australia have been very valuable. The quality of this paper has improved also by the comments by the unknown referees of the Tenth Petri Net Conference, and after it by the two referees of this volume. I wrote the Petri Net Conference version of this paper [Valmari 89b] while I was visiting

Telecom Australia Research Laboratories supported by Telecom Australia. That the visit was possible and I had the chance to do this work is largely due to Jonathan Billington. The Technical Research Centre of Finland (VTT) and the Technology Development Centre of Finland (Tekes) have supported this work through its all stages, including the period in Australia.

REFERENCES

[Aho & 74] Aho, A. V., Hopcroft, J. E. & Ullman, J. D.: *The Design and Analysis of Computer Algorithms.* Addison-Wesley 1974. 470 p.

[Back & 87] Back, R. J. R. & Kurki-Suonio, R.: *Distributed Cooperation with Action Systems.* ACM Transactions on Programming Languages and Systems, Vol 10, No. 4 1988, pp. 513–554.

[Brauer & 87] Brauer, W., Reisig, W. & Rozenberg, G. (ed.): *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 1987. 480 p.

[Brookes & 84] Brookes, S. D., Hoare, C. A. R. & Roscoe, A. W.: *A Theory of Communicating Sequential Processes.* Journal of the ACM 31 (3) 1984, pp. 560–599.

[Genrich 87] Genrich, H.: *Predicate/Transition Nets.* In: [Brauer & 87], pp. 207–247.

[Jensen 87] Jensen, K.: *Coloured Petri Nets.* In: [Brauer & 87], pp. 248–299.

[Manna & 81] Manna, Z. & Pnueli, A.: *The Temporal Framework for Concurrent Programs.* In: Boyer, R. S. & Moore, J. S. (ed.): The Correctness Problem in Computer Science. Academic Press 1981, pp. 215–274.

[Overman 81] Overman, W. T.: *Verification of Concurrent Systems: Function and Timing.* Ph.D. Dissertation, University of California Los Angeles 1981. 174 p.

[Pnueli 86] Pnueli, A.: *Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends.* In: Current Trends in Concurrency, Lecture Notes in Computer Science 224, Springer 1986 pp. 510–584.

[Reisig 85] Reisig, W.: *Petri Nets, an Introduction.* Springer 1985. 161 p.

[Reisig 87] Reisig, W.: *Place/Transition Systems.* In: [Brauer & 87] pp. 117–141.

[Rozenberg & 86] Rozenberg, G. & Thiagarajan, P. S.: *Petri Nets: Basic Notions, Structure, Behaviour.* In: Current Trends in Concurrency, Lecture Notes in Computer Science 224, Springer 1986 pp. 585–668.

[Thiagarajan 87] Thiagarajan, P. S.: *Elementary Net Systems.* In: [Brauer & 87] pp. 26–59.

[Valmari 88a] Valmari, A.: *Error Detection By Reduced Reachability Graph Generation.* Proceedings of the Ninth European Workshop on Application and Theory of Petri Nets, Venice, Italy 1988 pp. 95–112.

[Valmari 88b] Valmari, A.: *Heuristics for Lazy State Generation Speeds up Analysis of Concurrent Systems.* Proceedings of the Finnish Artificial Intelligence Symposium STeP-88, Helsinki 1988. Volume 2 pp. 640–650.

[Valmari 88c] Valmari, A.: *State Space Generation: Efficiency and Practicality.* Ph.D. Thesis, Tampere University of Technology Publications 55, Tampere 1988. 169 p.

[Valmari 89a] 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 1989 pp. 89–103.

[Valmari 89b] Valmari, A.: *Stubborn Sets for Reduced State Space Generation.* Proceedings of the Tenth International Conference on Application and Theory of Petri Nets, Bonn, West Germany 1989 Vol. 2 pp. 1–22.

[Valmari 90] Valmari, A.: *A Stubborn Attack on State Explosion,* 15 p. In: Kurshan, R. & Clarke, E. M. (ed.): Proceedings of the Workshop on Computer-Aided Verification, DIMACS Technical Report 90-31, June 1990, Volume I.

[Wheeler 85] Wheeler, G. R.: *Numerical Petri Nets — A Definition.* Telecom Australia Research Laboratories Report 7780, 1985, 42 p.

[Wheeler & 90] Wheeler, G. R., Valmari, A. & Billington, J.: *Baby Toras Eats Philosophers but Thinks about Solitaire.* Proceedings of the Fifth Australian Software Engineering Conference, Sydney, NSW, Australia, 1990 pp. 283–288.