The Sleep Set Method Revisited

Kimmo Varpaaniemi

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

Abstract. *State space generation* is a powerful formal method for analysis of concurrent and distributed finite state systems. It suffers from the *state space explosion problem,* however: the state space of a system can be far too large to be completely generated. The *sleep set method* is one way to try to avoid generating all of the state space when verifying a given property. This paper is concentrated on the transition selection function in the sleep set method applied to a *labelled transition system* to verify a simple safety property or the existence of enabled infinite transition sequences. The conditions found for the function can be used for combining the sleep set method with other analysis techniques.
Topic: distributed systems

1 Introduction

State space generation is a powerful formal method for detecting errors in such concurrent and distributed systems that have a finite state space. It suffers from the so called *state space explosion problem,* however: the state space of the system can be far too large with respect to the time and other resources needed to inspect all states in the space. Fortunately, many properties can be verified without inspecting all reachable states of the system.

Godefroid's *sleep set method* [2, 3, 4, 5, 6, 7, 21] is a promising technique for alleviating the state space explosion problem. This method utilizes the *independence of transitions* to cut down on the number of states inspected during the verification of a property.

Labelled transition systems give a general framework for several models of concurrency. This paper is concentrated on the transition selection function in the sleep set method applied to a labelled transition system to verify a simple safety property or the existence of enabled infinite transition sequences. The conditions found for the function can be used for combining the sleep set method with other analysis techniques.

The rest of this paper has been organized as follows: in Section 2, we introduce labelled transition systems and define concepts related to them. In Section 3, we present the sleep set method and show some results concerning the method. We conclude in Section 4 by summarizing the results obtained and briefly discussing possible directions for future research.

2 Labelled Transition Systems

In this section we give definitions for *labelled transition systems.* As well-known, several models of concurrency can be described by means of labelled transition systems.

We shall use "iff" to denote "if and only if". The *power set* (the set of subsets) of a set A is denoted by 2^A. The set of *(total) functions* from a set A to a set B is denoted by $(A \to B)$. The set of natural numbers, including 0, is denoted by N. When we define finite sequences, we use ε to denote the empty sequence.

Definition 1. A *labelled transition system* (an *LTS* for short) is a quadruple $\langle S, \Sigma, \Delta, s_0 \rangle$ such that S and Σ are sets, $\Delta \subseteq S \times \Sigma \times S$, and $s_0 \in S$. We call S the *set of states, T* the *set of actions,* Δ the *set of transitions,* and s_0 the *initial state.* The set $\Sigma \cup \Delta$ is called the *set of events of the LTS.* The function α from Δ to Σ is defined by

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

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

Definition 2. Let $\langle S, \Sigma, \Delta, s_0 \rangle$ be an LTS. For any $\Gamma \subseteq \Sigma \cup \Delta$,

$$\begin{aligned} \Gamma^0 &= \{\varepsilon\}, \\ (\forall n \in N\ \Gamma^{n+1} &= \{wx \mid w \in \Gamma^n \wedge x \in \Gamma\}), \text{ and} \\ \Gamma^* &= \{w \mid \exists n \in N\ w \in \Gamma^n\}. \end{aligned}$$

The set Δ^* is called the *set of finite transition sequences of the LTS,* and the set $(\Sigma \cup \Delta)^*$ is called the *set of finite event sequences of the LTS. A finite event sequence w is firable from a state s to a state s'* iff $s[w\rangle s'$ where

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

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

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

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

Definition 3. Let $\langle S, \Sigma, \Delta, s_0 \rangle$ be an LTS. The set $(N \rightarrow \Delta)$ is called the *set of infinite transition sequences of the LTS.* The function ς from $(N \rightarrow \Delta) \times N$ to Δ^* is defined by

$$(\forall w \in (N \rightarrow \Delta)\ \varsigma(w, 0) = \varepsilon), \quad \text{and}$$
$$(\forall w \in (N \rightarrow \Delta)\ \forall n \in N\ \varsigma(w, n+1) = \varsigma(w, n)w(n)).$$

If w is an infinite transition sequence and $n \in N$, $\varsigma(w, n)$ is called the *prefix of length n of w. An infinite transition sequence w is enabled at a state s* ($s[w\rangle$ for short) iff for each $n \in N$, the prefix of length n of w is enabled at s. The function Ω from S to $2^{(N \rightarrow \Delta)}$ is defined by requiring that for each state s, $\Omega(s)$ is the set of those infinite transition sequences that are enabled at s. Let f be a function from S to 2^Δ. *An infinite transition sequence w is f-enabled at a state s* ($s[w\rangle_f$ for short) iff for each $n \in N$, the prefix of length n of w is f-enabled at s. We say that f is *tough-lived* iff for each reachable state s,

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

Definition 4. Let $\langle S, \Sigma, \Delta, s_0 \rangle$ be an LTS. A transition sequence v is an *alternative sequence of a finite transition sequence w from a state s to a state s'* iff v is a finite transition sequence, $s[w\rangle s'$, and $s[v\rangle s'$. A transition sequence v is a *length-secure alternative sequence of a finite transition sequence w from a state s to a state s'* iff v is an alternative sequence of w from s to s' and not longer than w. The function ϑ from $\Delta^* \times S \times S$ to $2^{(\Delta^*)}$ is defined by requiring that for each finite transition sequence w, and for each state s and s', $\vartheta(w, s, s')$ is the *set of length-secure alternative sequences of w from s to s'*. Let ψ be a truth-valued function on S. A state s is a *ψ-state* iff $\psi(s)$ is true. Let f be a function from S to 2^Δ. We say that *f represents all sets of alternative sequences to ψ-states* iff for each reachable state s and for each ψ-state s',

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

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

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

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

3 Sleep Set Method

In this section we present the sleep set method. We concentrate on a generalized version of Wolper's and Godefroid's terminal state detection algorithm [21]. The generalized version is in Figure 1. The intuitive idea of the algorithm is to eliminate such redundant transition sequences that are not eliminated by the transition selection function f. The LTS $\langle S, \Sigma, \Delta, s_0 \rangle$ is assumed to be such that $S \cup \Sigma$ is finite. The algorithm computes an LTS $\langle S, \Sigma, \nabla, s_0 \rangle$ such that $\nabla \subseteq \Delta$. From the finiteness of $S \cup \Sigma$ and from the fact that the set Act constructed during one visit to a state does not intersect with the sets Act constructed during the other visits to the state it follows that the execution of the algorithm takes a finite time only.

The function ψ can be any truth-valued function on S. To be practical, we can think that $\psi(s)$ is true iff a given simple safety property holds. By "simple" we mean that $\psi(s)$ can be computed without inspecting any other state than s. The construction of ∇ can be omitted if only the detection of reachable ψ-states is of interest.

We use actions much in the same way as Wolper and Godefroid use *program transitions* in their terminal state detection algorithm in [21]. One might think that our approach is thus more coarse than the approach used in [21]. However, if we have a *global LTS* of the form defined in [21], we can relabel each *global transition* [21] by the program transition of the global transition, and apply our algorithm to the resulting LTS. In practice, a global transition can be relabelled when used for the first time, whereas the unused global transitions need no relabelling.

Theorem 5. *Let $\langle S, \Sigma, \Delta, s_0 \rangle$ be an LTS such that $S \cup \Sigma$ is finite. Let ψ be a truth-valued function on S. Let f be a function from S to 2^{Δ} such that f represents all sets of length-secure alternative sequences to ψ-states. Then the algorithm in Figure 1 finds all reachable ψ-states.*

Proof. Let s_d be a reachable ψ-state.

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

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

```
make Stack empty; make $H$ empty; $\nabla = \emptyset$;
push $\langle s_0, \emptyset \rangle$ onto Stack;
while Stack is not empty do {
    pop $\langle s, \text{Sleep} \rangle$ from Stack;
    if $s$ is not in $H$ then {
        $\text{Trans} = \{x \in f(s) \mid s[x\rangle \wedge \alpha(x) \in \Sigma \setminus \text{Sleep}\}$;
        $\text{Act} = \{a \in \Sigma \mid \exists x \in \text{Trans}\ \alpha(x) = a\}$;
        $\text{Succ} = \{\langle a, S' \rangle \mid a \in \text{Act} \wedge S' = \{s' \in S \mid \langle s, a, s' \rangle \in \text{Trans}\}\}$;
        if $\psi(s)$ is true then print "$\psi$-state!";
        enter $\langle s, \text{a copy of Sleep} \rangle$ in $H$;
    }
    else {
        let hSleep be the set associated with $s$ in $H$;
        $\text{Act} = \text{hSleep} \setminus \text{Sleep}$;
        $\text{Succ} = \{\langle a, S' \rangle \mid a \in \text{Act} \wedge S' = \{s' \in S \mid s[a\rangle s'\}\}$;
        $\text{Sleep} = \text{hSleep} \cap \text{Sleep}$;
        substitute a copy of Sleep for the set associated with $s$ in $H$;
    }
    $\text{newSleep} = \emptyset$;
    for each $a$ in Act do {
        let $S'$ be the set for which $\langle a, S' \rangle \in \text{Succ}$;
        for each $s'$ in $S'$ do {
            $\text{xSleep} = \{a' \in \text{Sleep} \mid s'[a'\rangle \wedge (\forall s'' \in S\ s'[a'\rangle s'' \Rightarrow s[a'a\rangle s'')\} \cup$
                $\{a_2 \in \text{newSleep} \mid \exists S_2\ \langle a_2, S_2 \rangle \in \text{Succ} \wedge s'[a_2\rangle \wedge$
                    $(\forall s'' \in S\ s'[a_2\rangle s'' \Rightarrow (\exists s_2 \in S_2\ s_2[a\rangle s''))\}$;
            push $\langle s', \text{a copy of xSleep} \rangle$ onto Stack;
            $\nabla = \{\langle s, a, s' \rangle\} \cup \nabla$;
        }
        $\text{newSleep} = \{a\} \cup \text{newSleep}$;
    }
}
```

Fig. 1. An LTS reduction and a ψ-state detection algorithm.

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

Indeed, assume the opposite, i.e., there exists some transition x' such that

$\alpha(x')$ is in X', and for some finite transition sequence v', $x'v'$ is in $\vartheta(w', s', s_d)$. Clearly, then $x_1x'v'$ is in $\vartheta(w, s, s_d)$. If $\alpha(x')$ is in Sleep during the execution of the outermost "for-loop", then every state reachable from s' by $\alpha(x')$ is reachable from s by $\alpha(x')\alpha(x_1)$, so there exist transitions x_2 and x'' such that $\alpha(x_2) = \alpha(x')$, $\alpha(x'') = \alpha(x_1)$, and $x_2x''v'$ is in $\vartheta(w, s, s_d)$. From the condition satisfied by X it then follows that $\alpha(x_2)$ is not in X, a contradiction with the assumption that $\alpha(x') = \alpha(x_2)$ is in Sleep$= X$. The action $\alpha(x')$ thus cannot be in Sleep during the execution of the outermost "for-loop". This means that $\alpha(x')$ has been inserted into newSleep in the outermost "for-loop" before firing x_1. Moreover, every state reachable from s' by $\alpha(x')$ is reachable from some $s_2 \in S_2$ by $\alpha(x_1)$ where S_2 is the set associated with $\alpha(x')$ in Succ. Consequently, there exist transitions x_2 and x'' such that $\alpha(x_2) = \alpha(x')$, $\alpha(x'') = \alpha(x_1)$, $x_2x''v'$ is in $\vartheta(w, s, s_d)$, and x_2 is either x_1 itself or fired after x_1. The action $\alpha(x') = \alpha(x_2)$ is thus not in newSleep at the time when x_1 is fired. This is a contradiction. The inductive hypothesis can thus be used to establish that some element having s_d as the first component will be or has already been popped from the stack.

We now consider the case where s already appears in H. Let $Y \subseteq \Sigma$ be such that $\langle s, Y\rangle$ is in H. All those transitions that are enabled at s and have their actions in $Y \setminus X$ are fired. There are two situations: either some action in Y is the first action of some sequence in $\vartheta(w, s, s_d)$, or no such action exists. In the first situation, we can choose a transition analogous to the above x_1 and proceed as above.

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

(ii) The algorithm in Figure 1 starts by pushing $\langle s_0, \emptyset\rangle$ onto an empty stack. From the result shown in part (i) it thus follows that some element having s_d as the first component will be popped from the stack. □

If we choose ψ to be the characteristic function of the set of terminal states and f to be a function preserving all sets of length-secure alternative sequences to terminal states, Theorem 5 states that the algorithm in Figure 1 finds all reachable terminal states. For example, we can choose f to be any *stubborn set generator* [11, 12, 13, 14, 15, 16, 17, 18, 19, 20] since every stubborn set generator preserves all sets of length-secure alternative sequences to terminal states. This is the case for every definition of stubbornness we have seen, and also for *dynamic stubbornness* [11, 16, 19, 20] and *conditional stubbornness* [5] which are generalizations of stubbornness. The sleep set method can thus be combined with the stubborn set method in the detection of reachable terminal states without any assumption on the stubborn sets used.

Godefroid, Pirottin, and Wolper [5, 21] have shown the compatibility of the sleep set method and the stubborn set method in the detection of reachable terminal states in the case where the effective parts of the stubborn sets used are *persistent*. They have utilized the property that persistent set generators "represent all *conditional traces* [5, 8] to terminal states". As shown in [20], it is easy to define quite practical stubborn set generators which do not have that property.

Theorem 5 has the consequence that if for each encountered state, the transition selection function chooses all enabled transitions, then the algorithm in Figure 1 visits all reachable states. Koutny and Pietkiewicz-Koutny [9] have shown that the sleep set algorithm presented by Godefroid, Holzmann, and Pirottin in [3] does not have this property though claimed so in [3]. Moreover, the examples in [9] suggest that it is hard to find any interesting class of problems that could be completely solved using the sleep set algorithm in [3]. We thus obtain the following heuristic: a good sleep set algorithm visits all reachable states if the transition selection function selects all enabled transitions.

The report [20] contains an example which shows that the statement obtained from Theorem 5 by removing the word "length-secure" is not valid. The example utilized the fact that if the initial sleep set of a state is empty, further visits to the state have no effect. Infinite transition sequences can thus be fatal if we use the algorithm in Figure 1 without care. A more positive feature related to infinite transition sequences is stated in Theorem 6.

Theorem 6. *Let $\langle S, \Sigma, \Delta, s_0\rangle$ be an LTS such that $S \cup \Sigma$ is finite. Let f be a tough-lived function from S to 2^{Δ}. If no infinite transition sequence is enabled at s_0 in the LTS $\langle S, \Sigma, \nabla, s_0\rangle$ at the end of the execution of the algorithm in Figure 1, then no infinite transition sequence is enabled at s_0 in the LTS $\langle S, \Sigma, \Delta, s_0\rangle$.*

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

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

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

Indeed, assume the opposite, i.e., there exists some $v' \in \Omega(s')$ such that $\alpha(v'(0))$ is in X'. If $\alpha(v'(0))$ is in Sleep during the execution of the outermost "for-loop", then every state reachable from s' by $\alpha(v'(0))$ is reachable from s by $\alpha(v'(0))\alpha(x_1)$, so there exists an infinite transition sequence w such that w is in $\Omega(s)$, and $\alpha(w(0)) = \alpha(v'(0))$. From the condition satisfied by X it then follows that $\alpha(w(0))$ is not in X, a contradiction with the assumption that $\alpha(v'(0)) = \alpha(w(0))$ is in Sleep$= X$. The action $\alpha(v'(0))$ thus cannot be in Sleep during the execution of the outermost "for-loop". This means that $\alpha(v'(0))$ has been inserted into newSleep in the outermost "for-loop" before firing x_1. Moreover, every state reachable from s' by $\alpha(v'(0))$ is reachable from some $s_2 \in S_2$ by $\alpha(x_1)$ where S_2 is the set associated with $\alpha(v'(0))$ in Succ. Consequently, there exists an infinite transition sequence w such that w is in $\Omega(s)$, $\alpha(w(0)) = \alpha(v'(0))$, and $w(0)$ is either x_1 itself or fired after x_1. The action $\alpha(v'(0)) = \alpha(w(0))$ is thus not in newSleep at the time when x_1 is fired. This is a contradiction.

The case where s already appears in H can be handled by repeating the corresponding part of the proof of Theorem 5 with the exception that the expression "$\Omega(s)$" is in the place of the expression "$\vartheta(w, s, s_d)$".

(ii) The algorithm in Figure 1 starts by pushing $\langle s_0, \emptyset \rangle$ onto an empty stack. If $\Omega(s_0) \neq \emptyset$, using the result shown in part (i) we can construct an infinite transition sequence which is enabled at s_0 in the LTS $\langle S, \Sigma, \nabla, s_0 \rangle$ at the end of the execution of the algorithm. □

From Theorem 6 it follows that we can detect the existence of enabled infinite transition sequences from the reduced LTS. Alternatively, one can add on-the-fly loop detection to the algorithm in Figure 1 in such a way that the first loop of states found terminates the execution of the algorithm. Then there is no need to construct ∇ since the construction of ∇ affects neither the set of visited states nor the order of visiting. Since every stubborn set generator is tough-lived [12, 13, 14, 16, 17, 20], the stubborn set method and the sleep set method can be combined in the detection of the existence of enabled infinite transition sequences without any assumption on the stubborn sets used.

Let's consider the complexity of the algorithm in Figure 1. The cumulative time per state spent in the outermost "for-loop" is at most proportional to $\mu^4\rho^2$, where μ is the maximum number of states reachable from a state by an action, ρ is the maximum number of enabled actions of a state, and all visits to the state are counted. This is based on the fact that each sleep set associated with a state contains only actions that are enabled at the state. The time per visit to a state spent in the operations related to H is the time of the search for the state plus a time that is at most proportional to ρ. The searches in H are something that cannot be avoided easily whether or not we use sleep sets at all. Clearly, the time taken by the computation of $f(s)$ and $\psi(s)$ can be anything depending on f and ψ. If ψ is the characteristic function of the set of terminal states, then the expression "if $\psi(s)$ is true" in the algorithm in Figure 1 can be replaced by the expression "if Act and Sleep are both empty". It depends much on the LTS how many times a state is visited and how many simultaneous occurrences of a state there are in the stack. One stack element requires space for the state and at most ρ actions. It is not necessary to store copies of states and actions since pointers suffice.

4 Conclusions

We have presented weak but sufficient conditions for the transition selection function used in the sleep set method applied to a labelled transition system. We have shown that a function representing all sets of length-secure alternative sequences to ψ-states is sufficient in the detection of reachable ψ-states, and a tough-lived function suffices in the detection of the existence of enabled infinite transition sequences. These results can be used for combining the sleep set method with other analysis techniques. For example, the sleep set method can be combined with the stubborn set method in the detection of reachable terminal states and the existence of enabled infinite transition sequences without any assumption on the stubborn sets used.

The sleep set method is applicable to many purposes in addition to those considered in this paper. The conditions for the transition selection functions in the different versions of the sleep set method should be studied. One goal is to combine the sleep set method with the stubborn set method in *on-the-fly verification of linear time temporal logic formulae.* It is well-known that both methods alone can be used for that purpose [10, 18, 21].

Acknowledgements

This work has been funded by the Technology Development Centre of Finland (TEKES). The results in this paper have been obtained by extending the results presented in the research report [20].

References

1. Courcoubetis, C. (Ed.): Proceedings of the 5th International Conference on Computer-Aided Verification, Elounda, Greece, June/July 1993. Lecture Notes in Computer Science 697, Springer-Verlag, Berlin 1993, 504 p.
2. Godefroid, P.: *Using Partial Orders to Improve Automatic Verification Methods.* Clarke, E.M., and Kurshan, R.P. (Eds.), Proceedings of the 2nd International Workshop on Computer-Aided Verification, New Brunswick NJ, June 1990. Lecture Notes in Computer Science 531, Springer-Verlag, Berlin 1991, pp. 176–185.
3. Godefroid, P., Holzmann, G.J., and Pirottin, D.: *State Space Caching Revisited.* von Bochmann, G., and Probst, D.K. (Eds.), Proceedings of the 4th International Workshop on Computer-Aided Verification, Montreal, June 1992. Lecture Notes in Computer Science 663, Springer-Verlag, Berlin 1993, pp. 178–191.
4. Godefroid, P., and Kabanza, F.: *An Efficient Reactive Planner for Synthesizing Reactive Plans.* Proceedings of AAAI-91, Anaheim CA, July 1991, Vol. 2, pp. 640–645.

5. Godefroid, P., and Pirottin, D.: *Refining Dependencies Improves Partial-Order Verification Methods.* In [1], pp. 438–449.
6. Godefroid, P., and Wolper, P.: *Using Partial Orders for the Efficient Verification of Deadlock Freedom and Safety Properties.* Formal Methods in System Design 2 (1993) 2, pp. 149–164.
7. Holzmann, G.J., Godefroid, P., and Pirottin, D.: *Coverage Preserving Reduction Strategies for Reachability Analysis.* Linn, R.J., Jr., and Uyar, M.Ü. (Eds.), Proceedings of the 12th International IFIP WG 6.1 Symposium on Protocol Specification, Testing, and Verification, Lake Buena Vista FL, June 1992. IFIP Transactions C-8, North-Holland, Amsterdam 1992, pp. 349–363.
8. Katz, S., and Peled, D.: *Defining Conditional Independence Using Collapses.* Theoretical Computer Science 101 (1992) 2, pp. 337–359.
9. Koutny, M., and Pietkiewicz-Koutny, M.: *On the Sleep Sets Method for Partial Order Verification of Concurrent Systems.* Manuscript, January 1993, 14 p.
10. Peled, D.: *All from One, One for All: on Model Checking Using Representatives.* In [1], pp. 409–423.
11. Rauhamaa, M.: *A Comparative Study of Methods for Efficient Reachability Analysis.* Helsinki University of Technology, Digital Systems Laboratory Report A 14, Espoo, September 1990, 61 p.
12. Valmari, A.: *State Space Generation: Efficiency and Practicality.* Doctoral thesis, Tampere University of Technology Publications 55, Tampere 1988, 170 p.
13. Valmari, A.: *Eliminating Redundant Interleavings during Concurrent Program Verification.* Proceedings of Parallel Architectures and Languages Europe '89, Vol. 2. Lecture Notes in Computer Science 366, Springer-Verlag, Berlin 1989, pp. 89–103.
14. Valmari, A.: *Stubborn Sets for Reduced State Space Generation.* Rozenberg, G. (Ed.), Advances in Petri Nets 1990. Lecture Notes in Computer Science 483, Springer-Verlag, Berlin 1991, pp. 491–515.
15. Valmari, A.: *A Stubborn Attack on State Explosion.* Formal Methods in System Design 1 (1992) 4, pp. 297–322.
16. Valmari, A.: *Stubborn Sets of Coloured Petri Nets.* Proceedings of the 12th International Conference on Application and Theory of Petri Nets, Gjern, Denmark, June 1991, pp. 102–121.
17. Valmari, A.: *Alleviating State Explosion during Verification of Behavioural Equivalence.* University of Helsinki, Department of Computer Science, Report A-1992-4, Helsinki 1992, 57 p.
18. Valmari, A.: *On-the-Fly Verification with Stubborn Sets.* In [1], pp. 397–408.
19. Valmari, A., and Clegg, M.: *Reduced Labelled Transition Systems Save Verification Effort.* Baeten, J.C.M., and Groote, J.F. (Eds.), Proceedings of the 2nd International Conference on Concurrency Theory, Amsterdam, August 1991. Lecture Notes in Computer Science 527, Springer-Verlag, Berlin 1991, pp. 526–540.
20. Varpaaniemi, K.: *Efficient Detection of Deadlocks in Petri Nets.* Helsinki University of Technology, Digital Systems Laboratory Report A 26, Espoo, October 1993, 56 p.
21. Wolper, P., and Godefroid, P.: *Partial-Order Methods for Temporal Verification.* Best, E. (Ed.), Proceedings of the 4th International Conference on Concurrency Theory, Hildesheim, August 1993. Lecture Notes in Computer Science 715, Springer-Verlag, Berlin 1993, pp. 233–246.