# Not Checking for Closure under Stuttering

Gerard J. Holzmann and Orna Kupferman
Bell Laboratories
700 Mountain Avenue
Murray Hill, NJ 07974
{gerard, ok}@research.att.com

June 15, 1996

**Abstract**

The model checker SPIN works better with specifications that are *closed under stuttering*. Checking such specifications, SPIN can use its partial-order reductions. It is hard to check whether a given specification is closed under stuttering and it is pity to give up SPIN's partial-order reductions. We suggest an algorithm that, given a program $P$ and a specification $N$ of bad behaviors for $P$, checks the correctness of $P$ with respect to an extension $N'$ of $N$ that is guaranteed to be closed under stuttering. In this check, SPIN can use its partial-order reductions. If $P$ is correct with respect to $N'$, we conclude that it is correct also with respect to $N$. If $P$ is not correct with respect to $N'$, we use the counter-example that SPIN provides to determine whether the program is correct with respect to $N$ or that $N$ is not closed under stuttering.

## 1 Introduction

To perform verifications with the model checker SPIN [Hol91], the user should provide two specifications. The first formalizes the behavior of the system to be verified; the second formalizes all possible violations of the correctness requirements. The first specification is usually given as a set of asynchronous processes, specified in the language of the verifier (PROMELA). During the verification itself, the interleaving product of these processes is computed as a *Büchi word automaton*. We refer to this product as the *program automaton* $P$. The second specification is given directly as a Büchi word automaton, in the syntax of PROMELA. We refer to this automaton as the *never-claim* $N$.

Verification is now reduced to checking the emptiness of the intersection of the automata $P$ and $N$ [VW86]. If the intersection is empty, it is guaranteed that no computation of the system violates the correctness requirement. If the intersection is not empty, Spin generates an infinite word in the intersection, and presents it as a counter-example to the correctness requirement.

The intersection of $P$ and $N$ is computed *on-the-fly*, using a depth-first search algorithm. When a standard exhaustive depth-first search algorithm is used, it is irrelevant whether or not the language of $N$ has special properties. For instance, it does not matter whether or not this language is *closed under stuttering*. There are, however, also optimized depth-first search algorithms that do make use of special properties that $N$ might have. In particular, this is true for the *partial-order* reductions used in Spin version 2. These reductions are guaranteed to preserve safety and liveness properties for an automaton $N$ whose language is closed under stuttering [HP94].

Given a never-claim automaton $N$, a naive algorithm would test $N$ for closure under stuttering and, according to the result, decide whether or not partial-order reductions can be applied. Testing $N$ for closure under stuttering, however, is hard. Indeed, the problem of testing Büchi automata for closure under stuttering is PSPACE-complete [PWW96]. Ideally, we would like to use a method that avoids the test, but can still apply the partial order reduction algorithm in as many cases as possible. This paper presents such a method.

## 2 Preliminaries

Given an alphabet $\Sigma$, a *block over* $\Sigma$ is a word in $\Sigma^*$ all of whose letters are the same. We can view a word $w \in \Sigma^\omega$ as an infinite sequence of blocks and represent it as a function $w : \mathbb{N}^+ \rightarrow \Sigma \times \mathbb{N}^+$, such that the $i$'th block of a word $w$ with $w(i) = \langle \sigma, n \rangle$ is $\sigma^n$. So, for example, the word $101100111000\ldots$ is represented by $w(i) = \langle i \bmod 2, i \operatorname{div} 2 \rangle$. We use $w_\sigma$ and $w_n$ to denote $w$ projected on its $\Sigma$ and $\mathbb{N}^+$ elements, respectively; thus $w(i) = \langle w_\sigma(i), w_n(i) \rangle$. Note that we do not require that successive blocks are of different letters. A *normal form* for the word $w$ is a function $w : \mathbb{N}^+ \rightarrow \Sigma \times \mathbb{N}^+$ where for every $i \in \mathbb{N}^+$ we have

$$w_\sigma(i) = w_\sigma(i+1) \rightarrow \text{ for all } j \geq i \text{ we have } w_\sigma(j) = w_\sigma(i) \text{ and } w_n(j) = 1.$$

That is, in a normal form, successive blocks are of different letters, except for the case where $w$ has a suffix $\sigma^\omega$. Then, the suffix is presented by infinitely many blocks of length 1. In the sequel, we assume that words are presented in this normal form.

We define an equivalent relation $S \subseteq \Sigma^\omega \times \Sigma^\omega$ as follows. For two words $w$ and $w'$, we have $S(w, w')$ iff $w_\sigma \equiv w'_\sigma$. Thus, two words are related by $S$ if they agree on the labels of their blocks and may only disagree on the lengths of their blocks. If $S(w, w')$, we say that $w'$ *stutters* $w$. We write $[w]$ to denote the set of all words $w'$ for which $S(w, w')$. A language $\mathcal{L} \subseteq \Sigma^\omega$ is *closed under stuttering* iff for every word $w \in \Sigma^\omega$, we have that $w \in \mathcal{L}$ iff $[w] \subseteq \mathcal{L}$. Thus, if $w$ is in $\mathcal{L}$, so are all words obtained from $w$ by either *shrinking* or *stretching* each of its blocks.

For technical convenience we present our algorithm in terms of Büchi transition systems, rather than Büchi word automata. One can translate transition systems to word automata by moving labels from states to transitions. Similarly, by moving labels from transitions to states, one can translate word automata to transition systems.

A *Büchi transition system* $A = \langle \Sigma, W, R, W_0, L, F \rangle$ consists of an alphabet $\Sigma$, a set $W$ of states, a total transition relation $R \subseteq W \times W$ (i.e., for every $w \in W$ there exists $w' \in W$ such that $R(w, w')$), a set $W_0$ of initial states, a labeling function $L : W \to \Sigma$, and an acceptance condition $F \subseteq W$. A *computation* of $A$ is a sequence $\pi = w_0, w_1, w_2, \ldots$ of states such that for every $i \geq 0$ we have $R(w_i, w_{i+1})$. For a computation $\pi = w_0, w_1, w_2, \ldots$, let $\mathit{Inf}(\pi)$ denote the set of states that $\pi$ visits infinitely often. That is,

$$\mathit{Inf}(\pi) = \{w \in W \ : \ \text{for infinitely many } i \geq 0, \text{ we have } w_i = w\}.$$

We say that a computation is *accepting* iff $\mathit{Inf}(\pi) \cap F \neq \emptyset$.

For a system $A$, we use $\mathcal{L}(A)$ to denote the set of all words $\sigma_0 \cdot \sigma_1 \cdots \in \Sigma^\omega$ for which there exists an accepting computation $w_0, w_1, \ldots$ with $w_0 \in W_0$ and with $L(w_i) = \sigma_i$ for all $i \geq 0$. Thus, each transition system defines a subset of $\Sigma^\omega$. We sometimes say that $A$ accepts $w$, meaning that $w \in \mathcal{L}(A)$. We say that a transition system $A$ is *empty* iff $\mathcal{L}(A) = \emptyset$; i.e., $A$ has no accepting computation. We say that a transition system $A$ is closed under stuttering iff $\mathcal{L}(A)$ is closed under stuttering.

## 3 Closing a Büchi Transition System under Stuttering

Given a Büchi transition system $A$, we construct a Büchi transition system $A'$ such that $w' \in \mathcal{L}(A')$ iff there exists $w \in [w'] \cap \mathcal{L}(A)$. That is, every word accepted by $A$ is also accepted by $A'$. In addition, $A'$ also accepts all the words that stutter some word accepted by $A$. We call $A'$ the *closure of $A$ under stuttering.* The system $A'$ should be more flexible than $A$ in two senses. First, when $A$ moves to a certain state, $A'$ may prefer to stay in the current state, reflecting its tolerance towards "stretched" blocks. Second, when $A$ proceeds to a certain state, $A'$ may prefer jump to states that $A$ reaches in several steps, reflecting its tolerance towards "shrunken" blocks. To preserve the acceptance condition we should further make sure that $A'$ does not miss or add infinitely many visits to states in $F$.

Let $A = \langle \Sigma, W, R, W_0, L, F\rangle$. Before defining $A'$, we need to define the *closure* of $R$ and its *$F$-closure*.

- For two states $w$ and $w'$, we have that $cl(R)(w, w')$ holds iff there exists $n \geq 1$ and $w_1, w_2, \ldots, w_n$ such that $w_1 = w$, $w_n = w'$, and for all $1 \leq i \leq n-1$ we have $R(w_i, w_{i+1})$ and $L(w_i) = L(w_1)$.

- For two states $w$ and $w'$, we have that $cl_F(R)(w, w')$ holds iff there exists $n \geq 1$, $1 \leq j \leq n$, and $w_1, w_2, \ldots, w_n$ such that $w_1 = w$, $w_j \in F$, $w_n = w'$, and for all $1 \leq i \leq n-1$ we have $R(w_i, w_{i+1})$ and $L(w_i) = L(w_1)$.

Intuitively, $cl(R)$ relates two states $w$ and $w'$ iff there exists a path from $w$ to $w'$ all of whose states are labeled as $w$. The relation $cl_F(R)$ is the same, only that at least one accepting state is visited in the path from $w$ to $w'$.

Given $A$, let $A' = \langle \Sigma, W \times \{0,1\}, R', W_0 \times \{0\}, L', W \times \{1\}\rangle$, where for every $w \in W$ we have that $L'(\langle w, 0\rangle) = L'(\langle w, 1\rangle) = L(w)$ and $R'(\langle w, c\rangle, \langle w', c'\rangle)$ holds iff one of the following holds.

1. $w = w'$ and $c = c' = 0$.

2. $cl(R)(w, w')$ and $c' = 0$.

3. $cl_F(R)(w, w')$ and $c' = 1$.

That is, the system $A'$ handles tolerance towards stretched blocks by self loops, and it handles tolerance towards shrunken blocks by proceeding according to the closure. The distinction between $cl(R)$ and $cl_F(R)$ as well as the duplication of the states, serves two goals. First, $A'$ can loop only in states that are not accepting. Otherwise, it might have accepted also words with suffixes $\sigma^\omega$ that stutter no word accepted by $A$. Second, when $A'$ reads a block of length 1 whose corresponding block in a word accepted by $A$ visits $F$, it can visit an accepting state and "keep tracking" $A$ in its next block.

**Theorem 3.1** *For every Büchi transition system $A$ and its closure under stuttering $A'$, the following hold.*

**(1)** $\mathcal{L}(A) \subseteq \mathcal{L}(A')$.

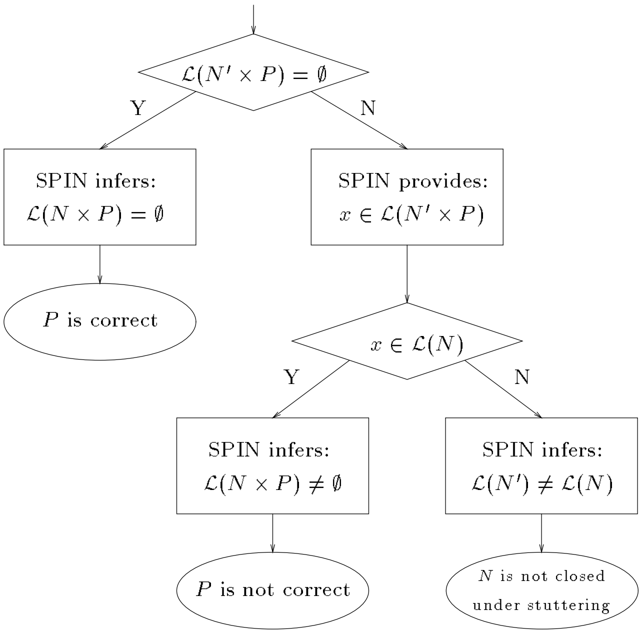
**(2)** *$A'$ is closed under stuttering.*

**(3)** *$A$ is closed under stuttering iff $\mathcal{L}(A) = \mathcal{L}(A')$.*

Note that the number of reachable states of $A'$ is at most twice that of $A$. Recall that SPIN gets its input as a word automaton (rather than a transition system). In this case, the construction of $N'$ is slightly more complicated. We may have to add up to $|W| \times |\Sigma|$ new states (one state for each transition in $N$). The following section shows how, in cases where the size of $N'$ remains close to that of $N$, SPIN can use $N'$ (with partial-order reduction), in order to check $N$, without first checking $N$ for closure under stuttering.

# 4 The Algorithm

Given $P$ and $N$, let $N'$ be the closure of $N$ under stuttering. Our algorithm is summarized in the figure below.



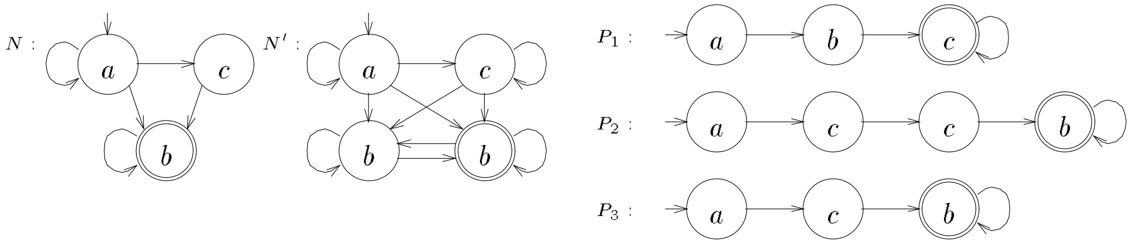

Instead of using $N$, SPIN uses $N'$ and computes its intersection with $P$. Since the automaton $N'$ is known to be closed under stuttering, this can be done using partial-order reductions. If the intersection is empty then, as $\mathcal{L}(N) \subseteq \mathcal{L}(N')$, it is guaranteed that the intersection of $N$ and $P$ is empty as well. Hence, $P$ is proven to be correct with respect to $N$.

If the intersection of $N'$ and $P$ is not empty, SPIN generates a counter-example $x$ accepted by both $N'$ and $P$. SPIN now checks whether $x$ is accepted by $N$. Since $x$ is a single word, this can be done easily by projecting the computations of $N$ on $x$. If $x$ is accepted by $N$ then, as it is also accepted by $P$, we found a violation of the correctness, and $P$ is proven to be incorrect with respect to $N$. If $x$ is not accepted by $N$ then, as it is accepted by $N'$, we proved that $N$ is not closed under stuttering. The user can then choose to repeat the verification with a revised claim, that is closed under stuttering, or to check the original $N$ without partial-order reduction.

Note that in some cases, the proof of $P$ with respect to $N$ can proceed successfully using $N'$ and the optimized partial-order reduction algorithm, even when the original $N$ is not closed under stuttering. To check merely the closure of $N$ under stuttering, our algorithm would be neither efficient nor helpful. First, $P$ is usually much bigger than $N$. Second, when $P$ is proved to be either correct or incorrect, no information on being $N$ closed under stuttering is obtained. The question we want to answer, however, is not whether $N$ is closed under stuttering, but whether $P$ is correct with respect to $N$, and we want to do so as efficiently as possible.

## 5 An Example

The figure below shows a Büchi transition system $N$ and its closure $N'$ under stuttering. On the right are three sample programs $P_1$, $P_2$, and $P_3$.




We demonstrate the execution of our algorithm with respect to these programs. The intersection with $N'$ is empty for program $P_1$, proving $P_1$'s correctness with respect to $N$ without further checks. For the other two programs, the intersection with $N'$ is not empty. Our algorithm now checks the membership of a counter-example (which in this case equals the original program) in $N$. For $P_2$, membership in $N$ holds, proving that $N$ is not closed under stuttering. For $P_3$, membership does not hold, proving that $P$ is not correct with respect to $N$. In the case of both $P_1$ and $P_3$ we can establish the program's (in)correctness, using $N'$ with partial-order reduction, despite the fact that $N$ itself is not closed under stuttering. In the case of $P_2$, we obtain an answer about the closure of $N$ under stuttering.

## References


[Hol91] G.J. Holzmann, *Design and Validation of Computer Protocols.* Prentice Hall, Software Series, 1991.

[HP94] G.J. Holzmann, D. Peled, An Improvement in Formal Verification, *Proc. 7th Int. Conf. on Formal Description Techniques*, Berne, Switzerland, 1994, 177–194.

[PWW96] D. Peled, T. Wilke, and P. Wolper, An Algorithmic Approach for Checking Closure Properties of $\omega$-Regular Languages. *Proc. of the 7th Conferance on Concurrency Theory*, August 1996.

[VW86] M.Y. Vardi and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. *Proc. of the First Symposium on Logic in Computer Science*, 322–331, Cambridge, June 1986.