Ordered Counter-Abstraction

(Refinable Subword Relations for Parameterized Verification)

Pierre Ganty and Ahmed Rezine*

[1] IMDEA Software Institute, Spain
[2] Linköping University, Sweden

Abstract. We present an original refinable subword based symbolic representation for the verification of linearly ordered parameterized systems. Such a system consists of arbitrary many finite processes placed in an array. Processes communicate using global transitions constrained by their relative positions (i.e., priorities). The model can include binary communication, broadcast, shared variables or dynamic creation and deletion of processes. Configurations are finite words of arbitrary lengths. The successful monotonic abstraction approach uses the subword relation to define upward closed sets as symbolic representations for such systems. Natural and automatic refinements remained missing for such symbolic representations. For example, subword based relations (even in conjunction with constraints on resulting Parikh images) are simply too coarse for automatic forward verification of systems involving priorities. We remedy to this situation and introduce a symbolic representation based on an original combination of counter abstraction with subword based relations. This allows us to define an infinite family of relaxation operators that guarantee termination by a new well quasi ordering argument. The proposed automatic analysis is at least as precise and efficient as monotonic abstraction when performed backwards. It can also be successfully used in forward, something monotonic abstraction is incapable of. We implemented a prototype to illustrate the approach.

1 Introduction

We introduce in this paper an original adaptation of counter abstraction and use it for the verification of safety properties for linearly ordered parameterized systems. Typically, such a system consists of an arbitrary number of identical processes placed in a linear array. Each process is assumed to have a finite number of states (for example obtained by predicate abstraction). The arbitrary size of these systems results in an infinite number of possible configurations. Examples of linearly ordered parameterized systems include mutual exclusion algorithms, bus protocols, telecommunication protocols, and cache coherence protocols. The goal is to check correctness (here safety) regardless of the number of processes in the system.

Configurations of a parameterized system are finite words of arbitrary lengths over the finite set Q of process states. Processes change state using transitions that might involve

* supported in part by the CENIIT research organization (project 12.04).

universal or existential conditions. Transition t below is constrained by a universal condition. It states that a process (with array index) i may fire t only if all processes with indices $j > i$ (i.e., to the right, hence $\forall_R$) are in states $\{q_1, q_2, q_3\} \subseteq Q$.

$$t : q_5 \to q_6 : \forall_R \{q_1, q_2, q_3\} \tag{1}$$

An existential condition may require that some (instead of all) processes with indices $j > i$ are in certain states. Regular model checking [14, 9] is an important technique that has been used for the uniform verification of infinite state systems in general, and of linearly ordered parameterized systems in particular. This technique uses finite state automata to represent sets of configurations, and transducers (i.e., finite state automata over pairs of letters) to capture transitions of the system. Verification boils down to the repeated calculation of several automata-based constructions among which is the application of the transducers to (typically) heavier and heavier automata representing more and more complex sets of reachable configurations. Acceleration [3], widening [6, 18] and abstraction [7] methods are used to ease termination.
In order to combat this complexity, the framework of monotonic abstraction [2, 1] uses upward closed sets (wrt. a predefined pre-order) as symbolic representations. This introduces an over-approximation, as sets of states generated during the analysis are not necessarily upward closed. The advantage is to use minimal constraints (instead of arbitrary automata) to succinctly represent infinite sets of configurations. The approach typically adopts the subword relation as the pre-order for the kind of systems we consider in this work[3]. The analysis starts with upward closed sets representing the bad configurations and repeatedly approximates sets of predecessors by closing them upwards. Termination is guaranteed by well quasi ordering [13]. The scheme proved quite successful [2, 1] but did not propose refinements for eliminating false positives in ordered systems like the ones we consider here.
In this work, we describe an original integration of upward closed based symbolic representation and of threshold based counter abstraction. The resulting symbolic representation allows for the introduction of original relaxation operators that can be used in classical over-approximate-check-refine reachability schemes. The idea of counter abstraction [17, 11] is to keep track of the number of processes that satisfy a certain property. A typical property for a process is to be in some state in Q. A simple approach to ensure termination is then to count up to a prefixed threshold. After the threshold, any number of processes satisfying the property is assumed possible. This results in a finite state system that can be exhaustively explored. If the approximation is too coarse, the threshold can be augmented. For systems like those we consider in this paper, automatically finding the right properties and thresholds can get very challenging. Consider for instance the transition t above (1). It is part of Burns mutual exclusion algorithm, where q_6 models access to the critical section [2]. Suppose we want to compute the t-successors of configurations only containing processes in state q_5. These are in fact reachable in Burns algorithm. Plain counter abstraction would capture that all processes are at state q_5. After one step it would capture that there is one process at state q_6 and all other processes are at state q_5 (loosing that q_6 is at the right of all q_5).

[3] As a concrete example, if $q_5 \in Q$, then the word $q_5 q_5$ would represent all configurations in $(Q^* q_5 Q^* q_5 Q^*)$ since $q_5 q_5$ is subword of each one of them.

After the second step it would conclude that configurations with at least two q_6 are reachable (mutual exclusion violation). Observe that increasing the threshold will not help as it will not preserve the relative positions of the processes. Upward closure based representations will also result in a mutual exclusion violation if used in forward. Suppose we use $q_5 q_5$ as a minimal constraint. Upward closure wrt. to the subword relation would result in the set $(Q^* q_5 Q^* q_5 Q^*)$ which already allows two processes at state q_6 to coexist. Even when using the refined ordering of [1], upward closure would result in $(\{q_5\}^* \, q_5 \, \{q_5\}^* \, q_5 \, \{q_5\}^*)$. After one step, the obtained $(\{q_5\}^* \, q_5 \, \{q_5\}^* \, q_6)$ will be approximated with $(\{q_5, q_6\}^* \, q_5 \, \{q_5, q_6\}^* \, q_6 \, \{q_5, q_6\}^*)$, again violating mutual exclusion. Approximations are needed to ensure termination (the problem is undecidable in general [4]). Indeed, without approximation, one would differentiate among infinite numbers of sets, like in the following sequence:

$$(\{q_5\}^* \, q_6), \; (\{q_5\}^* \, q_6 \, \{q_5\}^* \, q_6), \ldots \; (\{q_5\}^* \, q_6 \, \{q_5\}^* \ldots \{q_5\}^* \, q_6) \qquad (2)$$

The idea of this work is to combine threshold-based counter abstraction with subword-based upward closure techniques in order to propose an infinite number of infinite abstract domains allowing increasing precision of the analysis while still ensuring termination. To achieve this, we introduce the notion of a *counted word*. A counted word has a *base* and a number of formulae (called *counters*). Like in monotonic abstraction, a base (a word in Q^*) is used as a minimal element and denotes all larger words wrt. the subword relation. In addition, the counters are used to constrain the denotation of the base. We associate two counters to each position in the base: a *left* and a *right counter*. For each state q in Q, the left counter of a position constrains how many of the processes to the left of the position can be in state q (i.e. constrains Parikh images of allowed prefixes). Right counters constrain allowed suffixes to the right of the positions. For example $(\{q_5\}^* \, q_6)$, which cannot be captured by usual upward closure or counter abstraction techniques, is captured by the counted word φ_1 defined below:

$$\begin{aligned}
\varphi_1 &= \left(\begin{bmatrix} v_{q_5} \geq 0 \\ \wedge v_{q_6} = 0 \end{bmatrix}, q_6, \begin{bmatrix} v_{q_5} = 0 \\ \wedge v_{q_6} = 0 \end{bmatrix} \right), \\
\varphi_2 &= \left(\begin{bmatrix} v_{q_5} \geq 0 \\ \wedge v_{q_6} = 0 \end{bmatrix}, q_6, \begin{bmatrix} v_{q_5} \geq 0 \\ \wedge v_{q_6} = 1 \end{bmatrix} \right) \left(\begin{bmatrix} v_{q_5} \geq 0 \\ \wedge v_{q_6} = 1 \end{bmatrix}, q_6, \begin{bmatrix} v_{q_5} = 0 \\ \wedge v_{q_6} = 0 \end{bmatrix} \right), \ldots \\
\varphi_k &= \left(\begin{bmatrix} v_{q_5} \geq 0 \\ \wedge v_{q_6} = 0 \end{bmatrix}, q_6, \begin{bmatrix} v_{q_5} \geq 0 \\ \wedge v_{q_6} = (k-1) \end{bmatrix} \right) \cdots \left(\begin{bmatrix} v_{q_5} \geq 0 \\ \wedge v_{q_6} = (k-1) \end{bmatrix}, q_6, \begin{bmatrix} v_{q_5} = 0 \\ \wedge v_{q_6} = 0 \end{bmatrix} \right)
\end{aligned}$$

In φ_1, the base q_6 denotes the set $(Q^* q_6 Q^*)$. This is constrained to $(\{q_5\}^* \, q_6 Q^*)$ by the right counter $\begin{bmatrix} v_{q_5} \geq 0 \\ \wedge v_{q_6} = 0 \end{bmatrix}$ and to $(\{q_5\}^* \, q_6)$ by the left counter $\begin{bmatrix} v_{q_5} = 0 \\ \wedge v_{q_6} = 0 \end{bmatrix}$. Sequence (2) can then be captured by the counted words $\varphi_1, \varphi_2, \ldots \varphi_k$. This gain in precision comes at the cost of loosing termination. We therefore propose *relaxation* operators. Each operator comes with a cut-off, i.e., thresholds associated to each state in Q. If a counter requires $(v_q = k)$ with k larger than the threshold imposed by the cut-off, we weaken $(v_q = k)$ into $(v_q \geq k)$. Using a well quasi ordering argument, we show that this is enough to ensure termination of the analysis that relaxes all generated representations. If a spurious trace is generated, we increase the thresholds in order to obtain a more precise relaxation that eliminates the spurious trace. We implemented a prototype that allows, for the first time, to verify by forward exploration classical linearly ordered parameterized systems using upward closure based representations.

Related work. Other verification efforts with a termination guarantee typically consider decidable subclasses [11, 10], or use approximations to obtain systems on which the analysis is decidable [17, 8, 16]. For example, the authors in [10] propose a forward framework with systematic refinement to decide safety properties for a decidable class. The problem we consider here is undecidable. The authors in [16] use heuristics to deduce cut-offs in order to check invariants on finite instances. In [17] the authors use counter abstraction and truncate the counters in order to obtain a finite state system. This might require manual insertion of auxiliary variables to capture the relative order of processes in the array. Environment abstraction [8] combines predicate and counter abstraction. It results in what is essentially a finite state approximated system. Hence, it can require considerable interaction and human ingenuity to find the right predicates. Our approach handles linearly ordered systems in a uniform manner. It automatically adds precision based on the spurious traces it might generate.

Outline. Section (2) gives preliminaries and formalizes the notion of counters. Section (3) uses these counters to define counted words and to state some of their properties that will be useful to build a symbolic representation for the verification of parameterized systems. Section (4) formally describes the considered class of parameterized systems, and reports on using counted words as a symbolic representation to solve their rechability problem. We conclude in Section (5).

2 Preliminaries

Preliminaries. Fix a finite alphabet Σ and let Σ^* be the set of finite words over Σ. Let $w \cdot w'$ be the concatenation of the words w and w', ϵ be the empty word, and $w \sqcup\!\sqcup\, w'$ be the shuffle set $\{w_1 \cdot w'_1 \cdot w_2 \cdots w'_m | \ w = w_1 \cdots w_n \text{ and } w' = w'_1 \cdots w'_m\}$. We use $\mathbb{N}$ for the set of natural numbers and $\overline{n}$, with $n \in \mathbb{N}$, to mean $\{1, \ldots, n\}$. Assume a word $w = \sigma_1 \cdots \sigma_n$ where $\sigma_i \in \Sigma$ for $i \in \overline{n}$. We write $|w|$ for the size n, $w_{[i,j]}$ to mean the word $\sigma_i \cdot \sigma_{i+1} \cdots \sigma_j$, $w_{[i]}$ for the letter σ_i, $hd(w)$ for the letter σ_1, $tl(w)$ for the suffix $w_{[2,n]}$, and $w^\bullet$ for the set $\{\sigma_1, \ldots, \sigma_n\}$. A multiset m is a mapping $\Sigma \to \mathbb{N}$. We write $m \preceq m'$ to mean that $m(\sigma) \leq m'(\sigma)$ for each $\sigma \in \Sigma$. We write $m \oplus m'$ to mean the multiset satisfying $(m \oplus m')(\sigma) = m(\sigma) + m'(\sigma)$ for each $\sigma \in \Sigma$. If $m' \preceq m$, then the multiset $m \ominus m'$ is defined and verifies $(m \ominus m')(\sigma) = m(\sigma) - m'(\sigma)$ for each σ in Σ. The Parikh image $w^\#$ of a word w is the multiset that gives the number of occurrences of each letter σ in w. Given a set S and a pre-order (i.e., a reflexive and transitive binary relation) $\sqsubseteq$ on S, the pair $(S, \sqsubseteq)$ is said to be a well quasi ordering (wqo for short) if there is no infinite sequence $e_1, e_2, \ldots$ of elements of S with $e_i \not\sqsubseteq e_j$ for all $1 \leq i < j$.

A counter over an alphabet Σ is a conjunction of simple constraints that denotes a set of multisets. We fix a set of integer variables V_Σ that is in a one to one correspondence with Σ. Each variable v is associated to a letter σ in Σ. We write v_σ to make the association clear. Intuitively, v_σ is used to count the number of occurrences of the associated letter σ in a word in Σ^*. A counter basically captures multisets over Σ by separately imposing a constraint on each letter in Σ. Indeed, we define a counter cr to be either [`false`] or a conjunction $[\wedge_{\sigma \in \Sigma}(v_\sigma \sim k)]$ where $\sim$ is in $\{=, \geq\}$, each v_σ is a variable ranging over

$\mathbb{N}$ and each k is a constant in $\mathbb{N}$. Assume in the following a counter cr. For a letter σ in Σ, we write $cr(\sigma)$ to mean the strongest predicate of the form $(v_\sigma \sim k)$ implied by the counter cr. We write $\mathbf{1}_\sigma$ (resp. $\mathbf{0}$) to mean the counter $[\wedge_{\sigma_i \in \Sigma}(v_{\sigma_i} = b_{\sigma_i})]$ with $b_{\sigma_i} = 1$ for $\sigma_i = \sigma$ and $b_{\sigma_i} = 0$ otherwise (resp. $b_{\sigma_i} = 0$ for all $\sigma_i \in \Sigma$). A substitution is a set $\{v_1 \leftarrow u_1, \ldots\}$ of pairs (s.t. $v_i \neq v_j$ if $i \neq j$) where $v_1, \ldots$ are variables, and $u_1, \ldots$ are either all variables or all natural numbers. Given a substitution S, we write $cr[S]$ to mean the formula obtained by replacing, for each pair $v_i \leftarrow u_i$, each occurrence of v_i in cr by u_i. We sometimes regard a multiset m as the substitution $\{v_\sigma \leftarrow m(\sigma) | \ \sigma \text{ in } \Sigma\}$. For a multiset m, the formula $cr[m]$ takes a Boolean value. In the case where it evaluates to true (resp. false), we say that m satisfies (resp. doesn't satisfy) the counter cr and that the counter cr accepts (resp. does not accept) the multiset m. Given a word w in Σ^*, we abuse notation and write $cr[w]$ to mean that $(w^\#)$ satisfies cr. We write $[\![cr]\!]$ to mean the set $\{m| \ cr[m] \text{ and } m \text{ is a multiset over } \Sigma\}$. We define the cut-off of cr, written $\kappa(cr)$, to be the multiset that associates to each letter σ in Σ the value $k+1$ if $cr(\sigma) = (v_\sigma = k)$ and 0 otherwise. Observe that if $\kappa(cr)(\sigma) \neq 0$ for all $\sigma \in \Sigma$, then cr accepts a single multiset, while if $\kappa(cr)(\sigma) = 0$ for all $\sigma \in \Sigma$, then cr accepts an upward closed set of multisets wrt. $\preceq$[4]. We write $\mathbb{C}$ for the set of counters over Σ. Given a natural k, we write $\mathbb{C}_k$ to mean $\{cr| \ \kappa(cr)(\sigma) \leq k \text{ for each } \sigma \in \Sigma\}$. Observe that for any counter $cr \in \mathbb{C}_k$, $((cr(\sigma) = (v_\sigma = k')) \implies k' < k)$.

Example 1. For the counter $cr = [v_a = 0 \wedge v_b = 2 \wedge v_c \geq 1]$ over $\Sigma = \{a, b, c\}$, we have that: $\kappa(cr)(a) = 1$, $\kappa(cr)(b) = 3$, and $\kappa(cr)(c) = 0$. In addition, cr is in $\mathbb{C}_3$.

Operations on counters. Assume two counters cr and cr'. The predicate $(cr \sqsubseteq_\mathbb{C} cr')$ is defined as the conjunction $\wedge_{\sigma \in \Sigma}(cr \sqsubseteq_\mathbb{C} cr')(\sigma)$, where $(cr \sqsubseteq_\mathbb{C} cr')(\sigma)$ is defined in Table (1). In addition, let cr'' be any of the counters $(cr \sqcap_\mathbb{C} cr')$, $(cr \ominus_\mathbb{C} cr')$, or $(cr \oplus_\mathbb{C} cr')$. The counter cr'' is defined as the conjunction $\wedge_{\sigma \in \Sigma} cr''(\sigma)$, where $cr''(\sigma)$ is stated in Table (1). Observe that $(cr \sqsubseteq_\mathbb{C} cr') \implies [\![cr']\!] \subseteq [\![cr]\!]$, that $[\![cr \sqcap_\mathbb{C} cr']\!] = [\![cr]\!] \cap [\![cr']\!]$, that $[\![cr \oplus_\mathbb{C} cr']\!] = \{m_1 \oplus_\mathbb{C} m_2 | \ cr[m_1] \text{ and } cr'[m_2]\}$, and that $[\![cr \ominus_\mathbb{C} cr']\!] = \{m_1 \ominus_\mathbb{C} m_2 | \ cr[m_1] \text{ and } cr'[m_2]\}$.

Table 1. Contribution of each $\sigma \in \Sigma$ to the predicate $cr \sqsubseteq_\mathbb{C} cr'$ and to the counters $cr \sqcap_\mathbb{C} cr'$, $cr \oplus_\mathbb{C} cr'$ and $cr \ominus_\mathbb{C} cr'$.

$cr(\sigma)$	$cr'(\sigma)$	$(cr \sqsubseteq_\mathbb{C} cr')(\sigma)$	$(cr \sqcap_\mathbb{C} cr')(\sigma)$	$(cr \oplus_\mathbb{C} cr')(\sigma)$	$(cr \ominus_\mathbb{C} cr')(\sigma)$
$v_\sigma = b$	$v_\sigma = b'$	$b = b'$	$(b = b')?v_\sigma = b : \texttt{false}$	$v_\sigma = b + b'$	$(b \geq b')?v_\sigma = b - b' : \texttt{false}$
	$v_\sigma \geq b'$	`false`	$(b \geq b')?v_\sigma = b : \texttt{false}$		
$v_\sigma \geq b$	$v_\sigma = b'$	$b' \geq b$	$(b' \geq b)?v_\sigma = b' : \texttt{false}$		$v_\sigma \geq max(0, b - b')$
	$v_\sigma \geq b'$		$v_\sigma \geq max(b, b')$		

Lemma 1. *For each $k \in \mathbb{N}$, $(\mathbb{C}_k, \sqsubseteq_\mathbb{C})$ is a well quasi ordering. In fact, from every infinite sequence $cr_1, cr_2, \ldots$ we can extract an infinite sequence $cr_{i_1} \sqsubseteq_\mathbb{C} cr_{i_2} \sqsubseteq_\mathbb{C} \ldots$*

Proof. Let $cr_1, cr_2, \ldots$ be an infinite sequence. Fix a letter σ. If the number of counters for which $cr_m(\sigma)$ is not an equality is infinite, then remove all the counters for which $cr_m(\sigma)$ is an equality. Otherwise, by definition of $\mathbb{C}_k$, there is a $b_0 < k$ such that the number of counters for which $cr_m(\sigma) = (v_\sigma = b_0)$ is infinite. Keep those counters and remove all others from the resulting sequence. By repeating this procedure for each

[4] A set M of multisets is upward closed wrt $\preceq$ if $m \preceq m'$ and $m \in M$ imlpy $m' \in M$

letter σ in Σ, we obtain a new infinite sequence of counters $cr_{m_1}, cr_{m_2}, \ldots$ for which, for each m_i, m_j, $cr_{m_i}(\sigma) = (v_\sigma = b)$ iff $cr_{m_j}(\sigma) = (v_\sigma = b)$. Fix a letter σ for which $cr_{m_1}(\sigma) = (v_\sigma \geq b)$. It is possible to extract from the resulting sequence another infinite sequence $cr_{n_1}, cr_{n_2}, \ldots$ such that if $cr_{n_i}(\sigma) = (v_\sigma \geq b_{n_i})$ and $cr_{n_j}(\sigma) = (v_\sigma \geq b_{n_j})$ with $n_i < n_j$, then $b_{n_i} \leq b_{n_j}$. By repeating this for each letter σ, we obtain an infinite sequence in which $cr_{i_1} \sqsubseteq_{\mathbb{C}} cr_{i_2} \sqsubseteq_{\mathbb{C}} \ldots$. □

3 Counted Words

A counted word φ is a finite sequence $(l_1, \sigma_1, r_1) \cdots (l_n, \sigma_n, r_n)$ in $(\mathbb{C} \times \Sigma \times \mathbb{C})^*$. The *base* of φ (written $\overline{\varphi}$) is the word $\sigma_1 \cdots \sigma_n$ in Σ^*. We write $\overleftarrow{\varphi}$ (resp. $\overrightarrow{\varphi}$) to mean the counter $[\wedge_{\sigma \in \Sigma}(v_\sigma \geq 0)]$ if $\varphi = \epsilon$, and l_1 (resp. r_n) otherwise. We refer to $l_1, \ldots l_n$ (resp. $r_1, \ldots r_n$) as the left (resp. right) counters of φ. The counted word φ is *well formed* if $l_i[(\overline{\varphi})_{[1,i-1]}]$ and $r_i[(\overline{\varphi})_{[i+1,n]}]$ evaluate to true for each $i \in \overline{n}$. We assume ϵ is *well formed*. The following lemma constrains the possible predicates in a well formed counted word.

Lemma 2 (Well formedness). *Let $\varphi = (l_1, \sigma_1, r_1) \cdots (l_n, \sigma_n, r_n)$ be well formed. For each $i \in \overline{n}$, $l_i(\sigma)$ (resp. $r_i(\sigma)$) either equals:*

- *$(v_\sigma = (\overline{\varphi}_{[1,i-1]})^\#(\sigma))$ (resp. $(v_q = (\overline{\varphi}_{[i+1,n]})^\#(\sigma))$), or*
- *$(v_\sigma \geq k)$ for some k in $\{0, \ldots (\overline{\varphi}_{[1,i-1]})^\#(\sigma)\}$ (resp. in $\{0, \ldots (\overline{\varphi}_{[i+1,n]})^\#(\sigma)\}$).*

Denotation. If $w = \sigma_1 \cdots \sigma_m$, $\varphi = (l_1, \sigma_1, r_1) \cdots (l_n, \sigma_n, r_n)$, and $h : \overline{n} \to \overline{m}$ is an increasing injection, we write $w \models^h \varphi$ to mean that all following three conditions hold for each $i \in \overline{n}$ i) $\overline{\varphi}_{[i]} = w_{[h(i)]}$, and ii) $l_i(w_{[1,h(i)-1]})$, and iii) $r_i(w_{[h(i)+1,n]})$. Intuitively, there is an injection h that ensures $\overline{\varphi}$ is subword of w, and s.t. words to the left and right of each image of h respectively respect corresponding left and right counters in φ. We write $w \models \varphi$ if $w \models^h \varphi$ for some injection h, and $[\![\varphi]\!]$ to mean $\{w |\ w \models \varphi\}$. We let $[\![\epsilon]\!] = \Sigma^*$. Observe that every well formed word has a non-empty denotation since $\overline{\varphi} \models \varphi$. We use $\mathbb{CW}$ to mean the set of well formed counted words.

Example 2. $\varphi = \left(\begin{bmatrix} v_a = 0 \\ \wedge v_b \geq 0 \end{bmatrix}, a, \begin{bmatrix} v_a \geq 0 \\ \wedge v_b \geq 0 \end{bmatrix}\right)\left(\begin{bmatrix} v_a = 1 \\ \wedge v_b = 0 \end{bmatrix}, a, \begin{bmatrix} v_a = 0 \\ \wedge v_b \geq 0 \end{bmatrix}\right)$ and $[\![\varphi]\!] = aab^*$.

Normalization of well formed words. Counters in a counted word are not independent. Consider for instance $\varphi = (l_1, a, r_1)(l_2, a, r_2)$ in Example (2). We can change $l_1(b)$ to $(v_b = 0)$ without affecting the denotation of φ. The reason is that any prefix accepted by l_1 will have to be allowed by l_2. It is therefore vacuous for l_1 to accept words containing b, and more generally to accept more than $l_2 \ominus_{\mathbb{C}} \mathbf{1}_a$ (defined by well formedness). Also, observe that l_2 and r_2 imply we can change $r_1(a)$ from $(v_a \geq 0)$ to $(v_a = 1)$. We strengthen a well formed word using the normalization rules depicted in Table (2).

Lemma 3 (Normalization rules). *Applying any of the rules of Table (2) on a well formed word φ does preserve its denotation, and hence its well formedness.*

$$\frac{\varphi_p \cdot (l_i, \sigma_i, r_i) \cdot \varphi_m \cdot (l_j, \sigma_j, r_j) \cdot \varphi_s}{\varphi_p \cdot (l_i \sqcap_{\mathbb{C}} l_{i,j}, \sigma_i, r_i) \cdot \varphi_m \cdot (l_j, \sigma_j, r_j) \cdot \varphi_s}\ \mathrm{left_{i<j}} \qquad \frac{\varphi_p \cdot (l_i, \sigma_i, r_i) \cdot \varphi_m \cdot (l_j, \sigma_j, r_j) \cdot \varphi_s}{\varphi_p \cdot (l_i, \sigma_i, r_i) \cdot \varphi_m \cdot (l_j, \sigma_j, r_j \sqcap_{\mathbb{C}} r_{j,i}) \cdot \varphi_s}\ \mathrm{right_{i<j}}$$

$$\frac{\varphi_p \cdot (l_i, \sigma_i, r_i) \cdot \varphi_m \cdot (l_j, \sigma_j, r_j) \cdot \varphi_s}{\varphi_p \cdot (l_i, \sigma_i, r_i) \cdot \varphi_m \cdot ((l_j \sqcap_{\mathbb{C}} l'_{j,i}), \sigma_j, r_j) \cdot \varphi_s}\ \mathrm{left'_{i\neq j}} \qquad \frac{\varphi_p \cdot (l_i, \sigma_i, r_i) \cdot \varphi_m \cdot (l_j, \sigma_j, r_j) \cdot \varphi_s}{\varphi_p \cdot (l_i, \sigma_i, r_i \sqcap_{\mathbb{C}} r'_{i,j}) \cdot \varphi_m \cdot (l_j, \sigma_j, r_j) \cdot \varphi_s}\ \mathrm{right'_{i\neq j}}$$

Table 2. Normalization rules. For example, the rule `left` states we can replace counter l_i in word $(\varphi_p \cdot (l_i, \sigma_i, r_i) \cdot \varphi_m \cdot (l_j, \sigma_j, r_j) \cdot \varphi_s)$ by $(l_i \sqcap_{\mathbb{C}} l_{i,j})$. The introduced counters are $l_{i,j} = (l_j \ominus_{\mathbb{C}} (\mathbf{1}_{\sigma_i} \oplus_{\mathbb{C}} \ldots \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_{j-1}}))$, $r_{j,i} = (r_i \ominus_{\mathbb{C}} (\mathbf{1}_{\sigma_{i+1}} \oplus_{\mathbb{C}} \ldots \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_j}))$, $l'_{j,i} = (l_i \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_i} \oplus_{\mathbb{C}} r_i) \ominus_{\mathbb{C}} (r_j \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_j})$, and $r'_{i,j} = (r_j \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_j} \oplus_{\mathbb{C}} l_j) \ominus_{\mathbb{C}} (l_i \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_i})$.

Proof. Sketch. Let φ' be the word obtained from φ by applying one of the above rules. Such a rule only strengthens the counters. Hence, $[\![\varphi]\!] \supseteq [\![\varphi']\!]$. Assume w in Σ^* with $w \models^h \varphi$. We show $w \models^h \varphi'$ holds. We describe the cases $\mathrm{left_{i<j}}$ and $\mathrm{right'_{i\neq j}}$. We start with $\mathrm{left_{i<j}}$ and show that $l_{i,j}(w_{[1,h(i)-1]})$. We know $l_j(w_{[1,h(j)-1]})$ from $w \models^h \varphi$. We also know $l_j(w_{[h(1)]} \cdot w_{[h(2)]} \cdots w_{[h(j-1)]})$ by well formedness of φ and $w \models^h \varphi$. Observe that due to the allowed predicates in the counters, if $cr[m]$ and $cr[m'']$ for some multisets $m \preceq m''$, then $cr[m']$ for any multiset $m \preceq m' \preceq m''$. Also, observe that: $(w_{[h(1)]} \cdot w_{[h(2)]} \cdots w_{[h(j-1)]})^{\#} \preceq (w_{[1,h(i)-1]} \cdot w_{[h(i)]} \cdot w_{[h(i+1)]} \cdots w_{[h(j-1)]})^{\#} \preceq (w_{[1,h(j)-1]})^{\#}$. We get that $(l_j \ominus_{\mathbb{C}} (\mathbf{1}_{\sigma_i} \oplus_{\mathbb{C}} \ldots \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_{j-1}}))(w_{[1,h(i)-1]})$ and hence the result. For $\mathrm{right'_{i\neq j}}$, we need to show that $r'_{i,j}(w_{[h(i)+1,|w|]})$. Observe that $w \models^h \varphi$ ensures $l_i(w_{[1,h(i)-1]})$, $\mathbf{1}_{\sigma_i}(w_{[h(i)]})$, $l_j(w_{[1,h(j)-1]})$, $\mathbf{1}_{\sigma_j}(w_{[h(j)]})$ and $r_j(w_{[h(j)+1,|w|]})$. Hence, $(l_j \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_j} \oplus_{\mathbb{C}} r_j)[w]$ and $(l_i \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_i})[w_{[1,h(i)]}]$. The result follows from that $w^{\#} = (w_{[1,h(i)]})^{\#} \oplus_{\mathbb{C}} (w_{[h(i)+1,|w|]})^{\#}$. □

Procedure Normalize($(l_1, \sigma_1, r_1) \cdots (l_n, \sigma_n, r_n)$)

```
repeat
    $(l'_1, \sigma'_1, r'_1) \cdots (l'_n, \sigma'_n, r'_n) \leftarrow (l_1, \sigma_1, r_1) \cdots (l_n, \sigma_n, r_n)$;
    for $i \leftarrow 1$ to $n$ do
        for $j \leftarrow 1$ to $i-1$ do
            $r_i \leftarrow r_i \sqcap_{\mathbb{C}} \Big(r_j \ominus_{\mathbb{C}} (\mathbf{1}_{\sigma_{j+1}} \oplus_{\mathbb{C}} \ldots \mathbf{1}_{\sigma_i})\Big)$ ;
            $l_i \leftarrow l_i \sqcap_{\mathbb{C}} \Big((l_j \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_j} \oplus_{\mathbb{C}} r_j) \ominus_{\mathbb{C}} (r_i \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_i})\Big)$;
            $r_i \leftarrow r_i \sqcap_{\mathbb{C}} \Big((l_j \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_j} \oplus_{\mathbb{C}} r_j) \ominus_{\mathbb{C}} (l_i \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_i})\Big)$;
        for $j \leftarrow i+1$ to $n$ do
            $l_i \leftarrow l_i \sqcap_{\mathbb{C}} \Big(l_j \ominus_{\mathbb{C}} (\mathbf{1}_{\sigma_{j+1}} \oplus_{\mathbb{C}} \ldots \mathbf{1}_{\sigma_i})\Big)$ ;
            $l_i \leftarrow l_i \sqcap_{\mathbb{C}} \Big((l_j \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_j} \oplus_{\mathbb{C}} r_j) \ominus_{\mathbb{C}} (r_i \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_i})\Big)$;
            $r_i \leftarrow r_i \sqcap_{\mathbb{C}} \Big((l_j \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_j} \oplus_{\mathbb{C}} r_j) \ominus_{\mathbb{C}} (l_i \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_i})\Big)$;
until $(l_1, \sigma_1, r_1) \cdots (l_n, \sigma_n, r_n) \sqsubseteq_{\mathbb{NCW}} (l'_1, \sigma'_1, r'_1) \cdots (l'_n, \sigma'_n, r'_n)$;
return $(l_1, \sigma_1, r_1) \cdots (l_n, \sigma_n, r_n)$;
```

Lemma 4 (Normalization). *Procedure Normalize repeatedly applies the rules of Table (2). It results in a counted word that is independent of the application order.*

Proof. Termination can be obtained as follows. At each rule, manipulated and obtained counted words are well formed. Using Lemma (2), we deduce that all counters belong to a finite lattice in which rules are monotonic functions that strengthen a counter and keep the other counters unchanged. Unicity can be obtained by contradiction. Suppose two different counted words are obtained as normalizations of the same well formed counted word. The words can only differ in their counters. Pick different corresponding counters. Given the allowed forms for the predicates (Lemmata (2) and (3)), we deduce that at least one predicate associated to some letter is strictly stronger in one of the counters. If we apply to the word with a weaker predicate, the sequence of rules that were applied to the word with a stronger predicate, we would get a strictly stronger predicate. This contradicts having reached a fixpoint for the word with a weaker predicate. □

Normalized words and entailment. We write $\mathbb{NCW}$ to mean the set of normalized words in $\mathbb{CW}$. Assume two normalized counted words $\varphi = (l_1, \sigma_1, r_1) \cdots (l_n, \sigma_n, r_n)$ and $\varphi' = (l'_1, \sigma'_1, r'_1) \cdots (l'_m, \sigma'_m, r'_m)$ in $\mathbb{NCW}$. We say that φ is h-entailed by φ' for some increasing injection $h : \overline{n} \to \overline{m}$, and write $\varphi \sqsubseteq^h_{\mathbb{NCW}} \varphi'$, to mean that the following three conditions hold for each $i \in \overline{n}$: $\overline{\varphi}_{[i]} = \overline{\varphi'}_{[h(i)]}$, $l_i \sqsubseteq_{\mathbb{C}} l'_{h(i)}$, and $r_i \sqsubseteq_{\mathbb{C}} r'_{h(i)}$. We write $\varphi \sqsubseteq_{\mathbb{NCW}} \varphi'$ to mean that $\varphi \sqsubseteq^h_{\mathbb{NCW}} \varphi'$ for some h. Observe that $([\,v_a \geq 0\,], a, [\,v_a = 0\,]) \not\sqsubseteq_{\mathbb{NCW}} ([\,v_a = 0\,], a, [\,v_a \geq 0\,])$, but $[\![([\,v_a \geq 0\,], a, [\,v_a = 0\,])]\!] = [\![([\,v_a = 0\,], a, [\,v_a \geq 0\,])]\!] = a^+$.

Lemma 5 (Entailment). *The relation $\sqsubseteq_{\mathbb{NCW}}$ on $\mathbb{NCW}$ is both reflexive and transitive. Moreover, it can be checked in linear time in the length of the counted words and $\varphi \sqsubseteq_{\mathbb{NCW}} \varphi'$ implies $[\![\varphi']\!] \subseteq [\![\varphi]\!]$.*

Word cut-offs. Similarly to the cut-offs defined in Section (2) for counters, the cut-off of a well formed word φ is a multiset $\kappa(\varphi)$. It associates to each letter σ the natural number $max\ \{\kappa(cr)(\sigma)|\ cr$ is a counter in $\varphi\}$. In Example (2), $\kappa(\varphi)(a) = 2$ and $\kappa(\varphi)(b) = 1$. We say that a counted word φ has a k-cut-off if all its counters are in $\mathbb{C}_k$. For example, counted words with a 0-cut-off only have inequalities in their counters (they denote upward closed sets with respect to the subword ordering). We write $\mathbb{CW}_k$ ($\mathbb{NCW}_k$) to mean the set of (normalized) well formed counted words that have a k-cut-off.

Theorem 1 (WQO). *For any fixed $k \in \mathbb{N}$, $(\mathbb{NCW}_k, \sqsubseteq_{\mathbb{NCW}})$ is a well quasi ordering.*

Proof. Higman's Lemma [13] states that if $(\Sigma, \preceq)$ is a wqo, then the pair $(\Sigma^*, \preceq^*)$ is also a wqo[5]. We let $\Gamma = \mathbb{C}_k \times \Sigma \times \mathbb{C}_k$ and $(l, \sigma, r) \preceq (l', \sigma', r')$ if $l \sqsubseteq_{\mathbb{C}} l'$ and $\sigma = \sigma'$ and $r \sqsubseteq_{\mathbb{C}} r'$. Observe that $\mathbb{NCW}_k \subseteq \Gamma^*$, and that $\preceq^*$ coincides with $\sqsubseteq_{\mathbb{NCW}}$. Hence, showing that $(\Gamma, \preceq)$ is a wqo establishes the result. Given an infinite sequence we can extract an infinite subsequence $(l_{m_1}, \sigma_{m_1}, r_{m_1}), (l_{m_2}, \sigma_{m_2}, r_{m_2}), \ldots$ in which $\sigma_{m_i} = \sigma_{m_j}$ for all $i \neq j$ and use Lemma (1). □

Meet of counted words. Given φ, φ' in $\mathbb{NCW}$, the result of Procedure (zip) is a set $(\varphi \sqcap_{\mathbb{NCW}} \varphi')$ of normalized counted words that entail both φ and φ' and whose denotation coincides with $[\![\varphi]\!] \cap [\![\varphi']\!]$. An empty set corresponds to an empty intersection. The procedure builds a constrained shuffle of φ and φ'. It is recursive and takes as arguments five counted words z, p, s, p', s', with $\varphi = (p \cdot s)$ and $\varphi' = (p' \cdot s')$. We write $(z, (p : s), (p' : s'))$ for clarity. Intuitively, each call tries to complete the first argument z in order to obtain a counted word that entails both $(p \cdot s)$ and $(p' \cdot s')$. The

[5] $\sigma_1 \cdots \sigma_n \preceq^* \sigma'_1 \cdots \sigma'_m$ iff there is a strictly increasing $h : \overline{n} \to \overline{m}$ with $\sigma_i \preceq \sigma'_{h(i)}$

procedure starts with $(\epsilon, (\epsilon : \varphi), (\epsilon : \varphi'))$ and collects all such counted words z. At each call, it considers contributions to z from $hd(s)$ (lines (2-4)), $hd(s')$ (lines (9-11)), or both $hd(s)$ and $hd(s')$ (lines (5-8)). The contributions are completed by further calls to Procedure zip, and the results are collected in the local variable `collect`. Lines (2-4) capture the situation where a state in z is mapped to $hd(s)$ and tolerated by (i.e., not forbidden by the counters of) φ' (test at line (3)). Lines (5-8) correspond to a state in z simultaneously mapped to $hd(s)$ and $hd(s')$. The words s and s' contain states that are still not treated. Termination is obtained with the ranking function $|s| + |s'|$. The following lemma establishes correctness of Procedure (zip).

Lemma 6 (intersection). *Given* φ, φ' *in* $\mathbb{NCW}$, $\texttt{zip}(\epsilon, (\epsilon : \varphi), (\epsilon : \varphi'))$ *returns a set* $\{\varphi_1, \ldots \varphi_n\}$ *s.t.* $(\varphi \sqsubseteq_{\mathbb{NCW}} \varphi_i)$, $(\varphi' \sqsubseteq_{\mathbb{NCW}} \varphi_i)$ *for each* $i \in \overline{n}$, *and* $\cup_{i \in \overline{n}} [\![\varphi_i]\!] = [\![\varphi]\!] \cap [\![\varphi']\!]$.

Procedure zip(z, (p:s), (p':s'))

```
1  collect := $\emptyset$;
2  **if** $(s \neq \epsilon)$ **then**
3      **if** $\kappa(\overrightarrow{p'})(\overline{hd(s)}) = 0$ *and* $\kappa(\overleftarrow{s'})(\overline{hd(s)}) = 0$ **then**
4          collect $\cup$:= zip($z \cdot hd(s), (p \cdot hd(s) : tl(s)), (p' : s')$)
5  **if** $(s \neq \epsilon$ *and* $s' \neq \epsilon)$ **then**
6      **if** $(\overleftarrow{hd(s)} \sqcap_{\mathbb{C}} \overleftarrow{hd(s')} \neq \texttt{false})$ *and* $(\overline{hd(s)} = \overline{hd(s')})$ *and* $(\overrightarrow{hd(s)} \sqcap_{\mathbb{C}} \overrightarrow{hd(s')} \neq \texttt{false})$ **then**
7          $e := (\overleftarrow{hd(s)} \sqcap_{\mathbb{C}} \overleftarrow{hd(s')}, \overline{hd(s)}, \overrightarrow{hd(s)} \sqcap_{\mathbb{C}} \overrightarrow{hd(s')})$;
8          collect $\cup$:= zip($z \cdot e, (p \cdot hd(s) : tl(s)), (p' \cdot hd(s') : tl(s'))$)
9  **if** $(s' \neq \epsilon)$ **then**
10     **if** $\kappa(\overrightarrow{p})(\overline{hd(s')}) = 0$ *and* $\kappa(\overleftarrow{s})(\overline{hd(s')}) = 0$ **then**
11         collect $\cup$:= zip($z \cdot hd(s'), (p : s), (p' \cdot hd(s') : tl(s'))$)
12 **if** $(s = \epsilon$ *and* $s' = \epsilon)$ **then**
13     collect := {Normalize($z$)}
14 **return** collect;
```

Relaxation. We use the notion of relaxing a counted word φ wrt. a multiset ρ. First, given a counter $cr = [\wedge_{\sigma \text{ in } \Sigma}(v_\sigma \sim k)]$, relaxing cr wrt. ρ, written $\nabla_\rho(cr)$, results in the counter $[\wedge_{\sigma \text{ in } \Sigma}(v_\sigma \sim' k)]$ s.t. $(v_\sigma \sim' k)$ is equal to $(v_\sigma \geq k)$ if $(v_\sigma \sim k)$ was $(v_\sigma = k)$ in cr with $k \geq \rho(\sigma)$, and to $(v_\sigma \sim k)$ otherwise. In other words, relaxation wrt. ρ replaces by inequalities those equalities that involve constants larger or equal to what is allowed by ρ. Relaxation of a counted word φ wrt. a multiset ρ is simply the word $\nabla_\rho(\varphi)$ obtained by normalizing the result of relaxing all counters in φ wrt. ρ. We let $\nabla_{\mathbb{NCW}}$ be the set $\{\nabla_\rho | \ \rho$ is a multiset over $\Sigma\}$.

Lemma 7 (Relaxation). $\nabla_\rho(\varphi) \sqsubseteq_{\mathbb{NCW}} \varphi$ *for any* $\varphi \in \mathbb{NCW}$ *and multiset* ρ. *In addition,* $\kappa(\nabla_\rho(\varphi))(\sigma) \leq max(0, 2\rho(\sigma) - 1)$ *for each* $\sigma \in \Sigma$.

Proof. Sketch. Suppose $\nabla_\rho(\varphi) \not\sqsubseteq_{\mathbb{NCW}} \varphi$, then there is a counter cr_φ in φ that does not entail a corresponding counter cr_∇ in $\nabla_\rho(\varphi)$. This is not possible. Indeed, before normalization, $\nabla_\rho(\varphi)$ and φ are both well formed with the same base and normalization in $\nabla_\rho(\varphi)$ starts with weaker counters than those in φ . By applying to φ the sequence of normalization rules used to normalize $\nabla_\rho(\varphi)$, we obtain (by monotonicity) that the counters in $\nabla_\rho(\varphi)$ are weaker than those in φ. The strongest cut-off $(2\rho(\sigma) - 1)$ is obtained when both left and right counters in some tuple (l, σ, r) associate the predicate $v_\sigma = (\rho(\sigma) - 1)$ to the letter σ. One can show by induction on the number of applications of the normalization rules, that for any letter σ', $\kappa(l \oplus_{\mathbb{C}} \mathbf{1}_\sigma \oplus_{\mathbb{C}} r)(\sigma') \leq max(0, 2\rho(\sigma') - 1)$. □

4 Reachability for Linear Parameterized Systems

Linear Parameterized Systems with Global Conditions. Such a system consists of arbitrary many finite processes placed in an array. Formally, a *linear parameterized system* is a pair $\mathcal{P} = (Q, T)$, where Q is a finite set of *local states* and T is a finite set of *transitions*. A transition is either *local* or *global*. A local transition is of the form $q \rightarrow q'$. It allows a process to change its local state from q to q' independently of the local states of the other processes. A global transition is of the form $q \rightarrow q' : \mathbb{Q}P$, where $\mathbb{Q} \in \{\exists_L, \exists_R, \exists_{LR}, \forall_L, \forall_R, \forall_{LR}\}$ and $P \subseteq Q$. For instance, the condition $\forall_L P$ means that "all processes to the left should be in local states that belong to P". This work is well suited for extensions involving binary or broadcast communication, shared variables or dynamic creation and deletion of processes. We omit them for clarity. A parameterized system (Q, T) induces an infinite-state transition system where $C = Q^*$ is the set of *configurations* and $\longrightarrow$ is a transition relation on C. For configurations $c = c_1 q c_2$, $c' = c_1 q' c_2$, and a transition $t \in T$, we write $c \longrightarrow_t c'$ to mean:

- t is a local transition of the form $q \rightarrow q'$, or
- t is a global transition $q \rightarrow q' : \mathbb{Q}P$, and one of the following conditions is satisfied:
  - either $\mathbb{Q}P = \exists_L P$ and ${c_1}^\bullet \cap P \neq \emptyset$, or $\mathbb{Q}P = \exists_R P$ and ${c_2}^\bullet \cap P \neq \emptyset$, or $\mathbb{Q}P = \exists_{LR} P$ and $({c_1}^\bullet \cup {c_2}^\bullet) \cap P \neq \emptyset$.
  - or $\mathbb{Q}P = \forall_L P$ and ${c_1}^\bullet \subseteq P$, or $\mathbb{Q}P = \forall_R P$ and ${c_2}^\bullet \subseteq P$, or $\mathbb{Q}P = \forall_{LR} P$ and $({c_1}^\bullet \cup {c_2}^\bullet) \subseteq P$.

We write $\longrightarrow$ to mean $\cup_{t \in T} \longrightarrow_t$ and use $\xrightarrow{*}$ to denote its reflexive transitive closure. We assume that prior to starting the execution of the system, each process is in an (identical) *initial* state. We use *Init* to denote the set of *initial* configurations. Notice that the set *Init* is infinite. It can be shown, using standard techniques (see e.g. [19]), that checking safety properties (expressed as regular languages) can be translated into instances of the following reachability problem: given $\mathcal{P} = (Q, T)$ and a possibly infnite set C_F of configurations, check whether $Init \xrightarrow{*} C_F$.

A counted word based refinable reachability scheme We use normalized counted words over Q as a symbolic representation for sets of configurations of (Q, T). For this purpose, we require *Init* and C_F to be captured using a (set of) counted words. We then proceed by repeatedly computing (lemma (8)) in forward (resp. backward) the set of successor (resp. predecessor) configurations starting from the counted words capturing *Init* (resp. C_F). We use lemma (6) to check the intersection with C_F (resp. *Init*). We use the $\sqsubseteq_{\mathbb{NCW}}$ relation (lemma (5)) to maintain a minimal set (i.e. a set of pairwise unrelated elements) capturing the set of configurations that are forward (resp. backward) reachable from *Init* (resp. C_F). To ensure termination, we systematically apply some relaxation ∇_ρ that imposes bounded cut-offs (lemma (7) and theorem (1)). We start with the cut-off $\rho = \mathbf{0}$ and strengthen it (i.e. increment the values associated to some of the letters in Q) in case the over-approximation induced by ∇_ρ results in a spurious trace. Strengthening the cut-off ρ results in a more precise (and hence more expensive) analysis. We use the following heuristic to eliminate encountered spurious traces without making the analysis unecessary expensive. We follow without relaxation, the trace obtained using ∇_ρ and identify the letters in Q for which the relaxation in ρ is responsible for generating the supious trace. We then only increase the cut-offs for those letters.

Lemma 8 (Post and Pre). *Given $\varphi \in \mathbb{NCW}$ and a transition t, we can compute two sets of counted words $post_t(\varphi)$ and $pre_t(\varphi)$ such that $(\cup_{\varphi' \in post_t(\varphi)} [\![\varphi']\!])$ and $(\cup_{\varphi' \in pre_t(\varphi)} [\![\varphi']\!])$ respectively equal $\{c' |\ c \longrightarrow_t c'$ with c in $[\![\varphi]\!]\}$ and $\{c' |\ c' \longrightarrow_t c$ with c in $[\![\varphi]\!]\}$.*

Experimental Results. We have implemented the counted words based reachability scheme in Ocaml and run experiments on an Intel Core 2 Duo 2.26 GHz laptop with 4GB of memory. Table (3) summarizes the results. We have considered four classical mutex algorithms, namely Burns [2], compact [5] and refined [15] versions of Szymanski's algorithm, and the related Gribomont-Zenner mutex [12]. The algorithms respectively appear under rows (I,II,III and IV) in Table (3). We instantiate the scheme both in forward and backward. For each instantiation and each algorithm, we give running times in seconds, the number of refinement steps, the number of generated counted words and the outcome of the analysis. We write "?" to mean a trace was found by the over-approximated analysis, and write "$\surd$" to mean unreachability (i.e., safety) is established. We allocate a budget of 20 minutes for each refinement step, and write $\times$ in case the analysis exhausted the allocated time. The backward instantiation is more precise and naturally generalizes using upward closed sets as symbolic representations (as in monotonic abstraction [2]). It is therefore not surprising that it managed to establish correctness for all algorithms. Upward closed sets (wrt. the subword relation) yield however a too imprecise forward analysis. Still, the new approach could establish mutual exclusion for both algorithms (I) and (II). The analysis exhausted its time budget for the two other algorithms. Backward analysis seems to profit from the fact that it starts from an upward closed set of configurations. Forward analysis does not have that advantage. We did experiment with simple non-approximated accelerations. While this boosted performance, we do not report it in Table (3) in order to simplify the presentation.

		refine	time	steps	words	safe
Forward	I	1	.01	0	1	?
		2	.03	6	241	?
		3	.11	17	875	$\surd$
	II	1	.01	0	1	?
		2	.02	7	343	?
		3	.02	8	323	?
		4	.02	9	241	?
		5	.15	11	297	?
		6	.04	12	105	?
		7	5.85	171	5143	$\surd$
	III	1	.01	0	1	?
		2	.03	8	356	?
		3	.04	10	406	?
		4	.32	24	1252	?
		5	.36	25	1248	?
		6	.81	35	1043	?
		7	.54	40	685	?
		8	.04	12	149	?
		9	39.35	532	11006	?
		10	> 1200	> 2000	> 68000	$\times$
	IV	1	.01	0	1	?
		2	.06	9	636	?
		3	.07	11	685	?
		4	.07	12	602	?
		5	.09	13	651	?
		6	.08	28	1695	?
		7	2.52	54	3003	?
		8	10.02	52	2758	?
		9	3.03	57	866	?
		10	1.80	81	1006	?
		11	>1200	>2800	> 120000	$\times$
Backward	I	1	.02	2	151	$\surd$
	II	1	.18	19	3026	$\surd$
	III	1	110.14	1166	169789	?
		2	158.29	1567	194425	?
		3	30.31	583	78942	$\surd$
	IV	1	138.10	932	233604	?
		2	34.33	434	129368	$\surd$

Table 3. $\mathbb{NCW}$ based forward and backward analysis of mutex algorithms.

5 Conclusions

We have introduced a new symbolic representation for the verification of parameterized systems where processes are organized in a linear array. The new representation combines counter abstraction together with upward closure based techniques. It allows for an approximated analysis with a threshold-based precision (or relaxation) that can be uniformly tuned. Based on the representation, we implemented a counter example based refinement scheme that illustrated the applicability and the relevance of the approach, both for forward and for backward analysis. Possible futur work can investigate more general representations to apply to heap or graph manipulating programs.

References

1. P. A. Abdulla, G. Delzanno, and A. Rezine. Approximated context-sensitive analysis for parameterized verification. In *FMOODS/FORTE*, volume 5522, pp 41–56. Springer, 2009.
2. P. A. Abdulla, N. B. Henda, G. Delzanno, and A. Rezine. Regular model checking without transducers. In *Proc. TACAS*, volume 4424 of *LNCS*, pp 721–736. Springer Verlag, 2007.
3. P. A. Abdulla, B. Jonsson, M. Nilsson, and J. d'Orso. Regular model checking made simple and efficient. In *Proc. CONCUR 2002*, volume 2421 of *LNCS*, pp 116–130, 2002.
4. K. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22:307–309, 1986.
5. T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. Zuck. Parameterized verification with automatically computed inductive assertions. In *Proc. CAV*, vol 2102 of *LNCS*, pp 221–234, 2001.
6. B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large. In *Proc.* 15^{th} *Int. Conf. on Computer Aided Verification*, volume 2725 of *LNCS*, pp 223–235, 2003.
7. A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *CAV04*, LNCS, pp 372–386, Boston, July 2004. Springer Verlag.
8. E. Clarke, M. Talupur, and H. Veith. Environment abstraction for parameterized verification. In *Proc. VMCAI '06*, volume 3855 of *LNCS*, pp 126–141, 2006.
9. D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification*, volume 2102 of *LNCS*, 2001.
10. G. Geeraerts, J.-F. Raskin, and L. Van Begin. Expand, Enlarge and Check: new algorithms for the coverability problem of WSTS. *Journal of Computer and System Sciences*, 72(1):180–203, 2006.
11. S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992.
12. E. Gribomont and G. Zenner. Automated verification of Szymanski's algorithm. In *Proc. TACAS '98*, volume 1384 of *LNCS*, pp 424–438, 1998.
13. G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc. (3)*, 2(7):326–336, 1952.
14. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *Proc. CAV '97*, volume 1254, pp 424–435, 1997.
15. Z. Manna and A. Pnueli. An exercise in the verification of multi – process programs. In *Beauty is Our Business*, pp 289–301. Springer Verlag, 1990.
16. A. Pnueli, S. Ruah, and L. Zuck. Automatic deductive verification with invisible invariants. In *Proc. TACAS '01*, volume 2031, pp 82–97, 2001.
17. A. Pnueli, J. Xu, and L. Zuck. Liveness with (0,1,infinity)-counter abstraction. In *Proc.* 14^{th} *Int. Conf. on Computer Aided Verification*, volume 2404 of *LNCS*, 2002.
18. T. Touili. Regular Model Checking using Widening Techniques. *ENTCS*, 50(4), 2001. Proc. Workshop on Verification of Parametrized Systems (VEPAS'01), Crete, July, 2001.
19. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. LICS '86, Int. Symp. on Logic in Computer Science*, June 1986.