Ordered Counter-Abstraction

(Refinable Subword Relations for Parameterized Verification)

Pierre Ganty and Ahmed Rezine[⋆],

[1] IMDEA Software Institute, Spain
pierre.ganty@imdea.org
[2] Linköping University, Sweden
ahmed.rezine@liu.se

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 and 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 instance, subword based relations 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 upper closure operators on the new symbolic representations. Each operator allows us to approximate reachable states using symbolic representations where termination is guaranteed by a new well quasi ordering argument. We propose a generic counter example based refinement scheme that builds on these symbolic representations. The proposed automatic analysis is at least as precise and efficient as monotonic abstraction when performed backwards. It can also be used in forward, something monotonic abstraction is incapable of. We implemented a prototype to support the previous claims.

Keywords: counter abstraction, well quasi ordering, reachability, parameterized verification

1 Introduction

In this paper, we introduce 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

[⋆] supported in part by the CENIIT research organization (project 12.04).

processes placed in a linear array. Each process is assumed to have a finite number of states (for example via predicate abstraction [11]). 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.

$$t : \sigma_5 \to \sigma_6 : \forall_R \{\sigma_1, \sigma_2, \sigma_3\} \tag{1}$$

Configurations of such a system are finite words of arbitrary lengths over the finite set Σ of process states. Processes change state using transitions that might involve universal or existential conditions. Transition t of (1) is constrained by a universal condition. Here, a process (with array index) i may fire t if all processes with indices $j > i$ (i.e., to the right, hence $\forall_R$) are in states $\{\sigma_1, \sigma_2, \sigma_3\} \subseteq \Sigma$. An existential condition requires that some (instead of all) processes are in certain states.

We build on the framework of monotonic abstraction [2, 1]. This framework 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[1]. 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 alarms 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 [18, 10] 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 Σ. A simple approach to ensure termination is then to count up to an a priori fixed 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 of (1). It is part of Burns mutual exclusion algorithm, where σ_6 models access to the critical section [2]. Suppose we want to compute the t-successors of configurations only containing processes in state σ_5. These are in fact reachable in Burns algorithm. Plain counter abstraction would

[1] As a concrete example, if $\sigma_5 \in \Sigma$, then the word $\sigma_5\sigma_5$ would represent all configurations in $(\Sigma^*\sigma_5\Sigma^*\sigma_5\Sigma^*)$ since $\sigma_5\sigma_5$ is subword of each one of them.

capture that all processes are at state σ_5. After one step it would capture that there is one process at state σ_6 and all other processes are at state σ_5 (loosing that σ_6 is at the right of all σ_5). After the second step it would conclude that configurations with at least two σ_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 $\sigma_5\sigma_5$ as a minimal constraint. Upward closure wrt. to the subword relation would result in the set $(\Sigma^*\sigma_5\Sigma^*\sigma_5\Sigma^*)$ which already allows two processes at state σ_6 to co-exist. Even when using the refined ordering of [1], upward closure would result in $(\{\sigma_5\}^*\,\sigma_5\,\{\sigma_5\}^*\,\sigma_5\,\{\sigma_5\}^*)$. After one step, the obtained $(\{\sigma_5\}^*\,\sigma_5\,\{\sigma_5\}^*\,\sigma_6)$ will be approximated with $(\{\sigma_5,\sigma_6\}^*\,\sigma_5\,\{\sigma_5,\sigma_6\}^*\,\sigma_6\,\{\sigma_5,\sigma_6\}^*)$, 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:

$$(\{\sigma_5\}^*\,\sigma_6),\ (\{\sigma_5\}^*\,\sigma_6\,\{\sigma_5\}^*\,\sigma_6),\ldots\ (\{\sigma_5\}^*\,\sigma_6\,\{\sigma_5\}^*\ldots\{\sigma_5\}^*\,\sigma_6) \qquad (2)$$

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 Σ^*) 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 σ in Σ, the left counter of a position constrains how many of the processes to the left of the position can be in state σ. Right counters constrain allowed suffixes to the right of the positions. For example $(\{\sigma_5\}^*\,\sigma_6)$ is captured by the counted word φ_1 defined below:

$$\begin{aligned}
\varphi_1 &= \left(\begin{bmatrix} v_{\sigma_5} \geq 0 \\ \wedge v_{\sigma_6} = 0 \end{bmatrix}, \sigma_6, \begin{bmatrix} v_{\sigma_5} = 0 \\ \wedge v_{\sigma_6} = 0 \end{bmatrix}\right),\\
\varphi_2 &= \left(\begin{bmatrix} v_{\sigma_5} \geq 0 \\ \wedge v_{\sigma_6} = 0 \end{bmatrix}, \sigma_6, \begin{bmatrix} v_{\sigma_5} \geq 0 \\ \wedge v_{\sigma_6} = 1 \end{bmatrix}\right)\left(\begin{bmatrix} v_{\sigma_5} \geq 0 \\ \wedge v_{\sigma_6} = 1 \end{bmatrix}, \sigma_6, \begin{bmatrix} v_{\sigma_5} = 0 \\ \wedge v_{\sigma_6} = 0 \end{bmatrix}\right), \ldots\\
\varphi_k &= \left(\begin{bmatrix} v_{\sigma_5} \geq 0 \\ \wedge v_{\sigma_6} = 0 \end{bmatrix}, \sigma_6, \begin{bmatrix} v_{\sigma_5} \geq 0 \\ \wedge v_{\sigma_6} = (k-1) \end{bmatrix}\right)\ldots\left(\begin{bmatrix} v_{\sigma_5} \geq 0 \\ \wedge v_{\sigma_6} = (k-1) \end{bmatrix}, \sigma_6, \begin{bmatrix} v_{\sigma_5} = 0 \\ \wedge v_{\sigma_6} = 0 \end{bmatrix}\right)
\end{aligned}$$

In φ_1, the base σ_6 denotes $(\Sigma^*\sigma_6\Sigma^*)$. This is constrained to $(\{\sigma_5\}^*\,\sigma_6\Sigma^*)$ by the right counter $\begin{bmatrix} v_{\sigma_5} \geq 0 \\ \wedge v_{\sigma_6} = 0 \end{bmatrix}$ and to $(\{\sigma_5\}^*\,\sigma_6)$ by the left counter $\begin{bmatrix} v_{\sigma_5} = 0 \\ \wedge v_{\sigma_6} = 0 \end{bmatrix}$. Sequence (2) can then be captured by the counted words $\varphi_1, \varphi_2, \ldots \varphi_k$. This gain in precision comes at the price of loosing termination. We therefore propose *relaxation* operators. Each operator comes with a cut-off, i.e., thresholds associated to each state in Σ. 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 use in a forward exploration scheme upward closure based representations to verify classical linearly ordered parameterized systems.

Related work. Other verification efforts with a termination guarantee typically consider decidable subclasses [10, 9], or use approximations to obtain systems on which the analysis is decidable [18, 8, 17]. For example, the authors in [9] propose a forward framework with systematic refinement to decide safety properties for a decidable class.

The problem we consider here is undecidable. Regular model checking [14, 3] is an important technique for the uniform verification of infinite state systems in general, and of linearly ordered parameterized systems in particular. It 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, 19] and abstraction [7] methods are used to ease termination. In order to combat this complexity, the framework of monotonic abstraction [2, 1] was proposed. Our work builds on this framework which uses upward closed sets (wrt. a predefined pre-order) as symbolic representations.

The authors in [17] use heuristics to deduce cut-offs in order to check invariants on finite instances. In [18] 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. The finite countermodel method [15] establishes safety of a parameterized systems by delegating to an SMT solver the burden to find a model for a suitable first-order encoding of the system.

Outline. Sect. 2 gives some preliminaries and Sect. 3 formalizes the notion of counters. Sect. 4 uses these counters to define counted words and Sect. 5 describes how to manipulate them. Sect. 6 formally describes the considered class of parameterized systems and Sect. 7 details the refinement based reachability algorithm. Finally, Sect. 8 reports on using counted words as a symbolic representation to solve rechability problems for four different parameterized systems. We conclude in Sect. 9.

2 Preliminaries

Preliminaries. Fix a finite alphabet Σ and let Σ^* be the set of finite words over Σ. We use (possibly primed or subscripted versions of) w (resp. σ) to mean a word in Σ^* (resp. a letter in Σ). We write $w \cdot w'$ to mean the concatenation of the words w and w', and ϵ to mean the empty word in Σ^*. We use $\mathbb{N}$ for the set of natural numbers and $[n : m]$, with $n, m \in \mathbb{N}$, to mean $\{n, n+1, \ldots, m\}$. Assume a word $w = \sigma_1 \cdots \sigma_n$ where $\sigma_i \in \Sigma$ for $i \in [1 : n]$. 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, and $w^\bullet$ for the set $\{\sigma_1, \ldots, \sigma_n\}$. We write $\mathcal{H}_n^m$ to mean the set of increasing injections from $[1 : n]$ to $[1 : m]$. We abuse notation and write $\mathcal{H}_u^w$ to mean the set $\left\{h |\ h \in \mathcal{H}_{|u|}^{|w|} \text{ and } (u)_i = (w)_{h(i)}\right\}$. For two words u, w, we write $u \preceq^h w$ to mean that $h \in \mathcal{H}_u^w$. We write $u \preceq w$ if $u \preceq^h w$ for some h. Observe that $u \preceq w$ iff u can be obtained by deleting letters from w. Given three words w, u and u', we say that w is the product of u and u' with respect to $h \in \mathcal{H}_u^w$ and $h' \in \mathcal{H}_{u'}^w$, written $w = u \overset{h,h'}{\otimes} u'$, to mean that both h and h' are defined, and that $[1 : |w|] = h([1 : |u|]) \cup h'([1 : |u'|])$. The product of two words, written $u \otimes u'$, is then the set $\left\{w |\ h \in \mathcal{H}_u^w \text{ and } h' \in \mathcal{H}_{u'}^w \text{ and } w = u \overset{h,h'}{\otimes} u'\right\}$. This set can be defined inductively as follows: $(w \otimes \epsilon) = \{w\} = (\epsilon \otimes w)$ and $(\sigma \cdot u) \otimes (\sigma' \cdot u')$ is the union of three sets, namely $\{\sigma \cdot w |\ w \in (u \otimes (\sigma' \cdot u'))\}$, $\{\sigma \cdot w |\ \sigma = \sigma' \text{ and } w \in (u \otimes u')\}$, and $\{\sigma' \cdot w |\ w \in ((\sigma \cdot u) \otimes u')\}$. For instance, $ab \otimes a = \{ab, aab, aba\}$ and $aba = (ab \overset{h,h'}{\otimes} a)$ with $h(1) = 1, h(2) = 2$ and $h'(1) = 3$. We write $f[a \leftarrow b]$ for a mapping $f : A \to B$, $a \in A$ and $b \in B$, to mean the mapping $g : A \to B$ such that $g(x) = b$ if $x = a$ and $g(x) = f(x)$ otherwise. 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 $(S, \trianglelefteq)$ 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 $e_i \ntrianglelefteq e_j$ for all $1 \leq i < j$.

3 Multisets Counters

We introduce in this section simple predicates we call counters. Counters denote multisets and we will use them in the remaining of this paper in order to capture the allowed letters in a word.

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 bijection 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` (sometimes written $\bot_\mathbb{C}$) or a conjunction

$[\wedge_{\sigma\in\Sigma}(v_\sigma \sim_\sigma k_\sigma)]$ where $\sim_\sigma$ is in $\{=,\geq\}$, each v_σ is a variable ranging over $\mathbb{N}$ and each k_σ are constants in $\mathbb{N}$. We sometimes write $\top_{\mathbb{C}}$ to mean the counter $[\wedge_{\sigma\in\Sigma}(v_\sigma \geq 0)]$. 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 $[\wedge_{\sigma'\in\Sigma}(v_{\sigma'} = b_{\sigma'})]$ with $b_{\sigma'} = 1$ for $\sigma' = \sigma$ and $b_{\sigma'} = 0$ otherwise (resp. $b_{\sigma'} = 0$ for all $\sigma' \in \Sigma$). A substitution is a set $\{v_1/u_1, \ldots\}$ 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/u_i, each occurrence of v_i in cr by u_i. We sometimes regard a multiset m as the substitution $\{v_\sigma \leftarrow m(\sigma) \mid \sigma \text{ in } \Sigma\}$. 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 $\{m \mid cr[m] \text{ and } m \text{ is a multiset over } \Sigma\}$. We define the cutoff 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, and if $\kappa(cr)(\sigma) = 0$ for all $\sigma \in \Sigma$ for some counter cr, then cr accepts an upward closed set of multisets.

Lemma 1 (convexity). *Given a counter cr and three multisets m, m' and m'' such that $m \preceq m' \preceq m''$. If $cr[m]$ and $cr[m']$ then $cr[m'']$.*

We write $\mathbb{C}$ for the set of counters over Σ. Given a natural k, we write $\mathbb{C}_k$ to mean $\{cr \mid \kappa(cr)(\sigma) \leq k \text{ for each } \sigma \in \Sigma\}$. Observe that for any counter $cr \in \mathbb{C}_k$, $((cr(\sigma) = (v_\sigma = k')) \implies k' < k)$.

Example 2. 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 $(cr \trianglelefteq_{\mathbb{C}} cr')$ is defined as the conjunction $\wedge_{\sigma\in\Sigma}(cr \trianglelefteq_{\mathbb{C}} cr')(\sigma)$, where $(cr \trianglelefteq_{\mathbb{C}} cr')(\sigma)$ is defined in Tab. 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 Tab. 1.

Lemma 3 (operations on counters). *Each of the following statements holds for any given counters cr, cr':*

1. $(cr \trianglelefteq_{\mathbb{C}} cr') \iff [\![cr']\!] \subseteq [\![cr]\!]$,
2. $[\![cr \sqcap_{\mathbb{C}} cr']\!] = [\![cr]\!] \cap [\![cr']\!]$,
3. $[\![cr \oplus_{\mathbb{C}} cr']\!] = \{m_1 \oplus m_2 \mid cr[m_1] \text{ and } cr'[m_2]\}$.
4. *$[\![cr \ominus_{\mathbb{C}} cr']\!]$ coincides with $\{m_1 \ominus m_2 \mid cr[m_1] \text{ and } cr'[m_2]\}$ in case $(cr \trianglelefteq_{\mathbb{C}} cr')$ and includes it otherwise.*

Observe that $cr \trianglelefteq_{\mathbb{C}} cr'$ is equivalent to $cr' \Rightarrow cr$. In fact, $cr \trianglelefteq_{\mathbb{C}} \perp_{\mathbb{C}}$ and $\top_{\mathbb{C}} \trianglelefteq_{\mathbb{C}} cr$ for any counter cr. We say that counter $\perp_{\mathbb{C}}$ is the strongest among all counters, no multiset is accepted by $\bot_{\mathbb{C}}$. The counter $\top_{\mathbb{C}}$ is the weakest counter, it accepts all multisets.

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

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

Lemma 4. *For $k \in \mathbb{N}$, $(\mathbb{C}_k, \trianglelefteq_{\mathbb{C}})$ is a well quasi ordering. From every infinite sequence $cr_1, cr_2, \ldots$ we can extract an infinite sequence $cr_{i_1} \trianglelefteq_{\mathbb{C}} cr_{i_2} \trianglelefteq_{\mathbb{C}} \ldots$*

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 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 and letter σ, either neither of $cr_{m_i}(\sigma)$ and $cr_{m_j}(\sigma)$ is an equality, or they are both the same equality. 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 $cr_{i_1} \trianglelefteq_{\mathbb{C}} cr_{i_2} \trianglelefteq_{\mathbb{C}} \ldots$. □

4 Counted Words

We introduce in this section counted words as a mean to symbollically represent (possibly infinite) sets of words. We state some of their properties (denotation, normalization and well quasi orederness) that are crucial to the construction of the reachability scheme we describe in Sect. 7.

A counted word φ is a finite word $(cr_1, \sigma_1, cr'_1) \cdots (cr_n, \sigma_n, cr'_n)$ in $(\mathbb{C} \times \Sigma \times \mathbb{C})^*$. The *base* of φ, written $b(\varphi)$, is the word $\sigma_1 \cdots \sigma_n$ in Σ^*. We also define mappings l and r from $(\mathbb{C} \times \Sigma \times \mathbb{C}) \cup \{\epsilon\}$ to $\mathbb{C}$. Both mappings associate $\top_{\mathbb{C}}$ to ϵ. In addition, l (resp. r) associates l (resp. r) to (l, σ, r) in $(\mathbb{C} \times \Sigma \times \mathbb{C})$. When clear from the context, we will write l_i and r_i to mean $l((\varphi)_i)$ and $r((\varphi)_i)$ resepctively. We refer to $l_1, \ldots l_{|\varphi|}$ (resp. $r_1, \ldots r_{|\varphi|}$) as the left (resp. right) counters of φ. The counted word φ is *well formed* if $l_i[b((\varphi)_{1,i-1})]$ and $r_i[b((\varphi)_{i+1,|\varphi|})]$ evaluate to true for each $i \in [1 : |\varphi|]$. We assume ϵ is *well formed.* We write $\mathtt{wf}(\varphi)$ to mean φ is well formed. The following lemma constrains the possible predicates in a well formed counted word.

Lemma 5 (Well formedness). *Assume a counted word φ. For each σ in Σ, let $k^l_\sigma = (b((\varphi)_{1,i-1}))^\#(\sigma)$ and $k^r_\sigma = (b((\varphi)_{i+1,|\varphi|}))^\#(\sigma)$. If φ is well formed, then $l_i(\sigma)$ is either the equality $(v_\sigma = k^l_\sigma)$ or some inequality $(v_\sigma \geq k)$ for some k in $[0 : k^l_\sigma]$. Similarly, each $r_i(\sigma)$ is either the equality $(v_\sigma = k^r_\sigma)$, or some inequality $(v_\sigma \geq k)$ for some k in $[0 : k^r_\sigma]$.*

Proof. Well formedness requires that each $l_i[b((\varphi)_{1,i-1})]$ and $r_i[b((\varphi)_{i+1,n})]$ hold. The rest follows from the allowed predicates in the counters.

Denotation. Assume a word w and a counted word φ. Let $h : [1 : |\varphi|] \to [1 : |w|]$ be an increasing injection. We write $w \models^h \varphi$ to mean that $b(\varphi) \preceq^h w$ and that both $l_i[(w)_{1,h(i)-1}]$ and $r_i[(w)_{h(i)+1,|w|}]$ hold. Intuitively, there is an increasing injection h that ensures $b(\varphi)$ is subword of w, and such that 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 $b(\varphi) \models \varphi$. We use $\mathbb{W}$ to mean the set of well formed counted words. For a set Φ of counted words, we write $[\![\Phi]\!]$ to mean the set $\cup_{\varphi \in \Phi} [\![\varphi]\!]$.

Example 6. $\varphi = \left(\begin{bmatrix} v_a = 0 \\ \wedge v_b \geq 0 \end{bmatrix}, a, \begin{bmatrix} v_a \geq 0 \\ \wedge v_b \geq 0 \end{bmatrix} \right) \left(\begin{bmatrix} v_a = 1 \\ \wedge v_b = 0 \end{bmatrix}, a, \begin{bmatrix} v_a = 0 \\ \wedge v_b \geq 0 \end{bmatrix} \right)$ and $[\![\varphi]\!] = aab^*$.

Normalization of well formed words. We are often interested in knowing whether occurences of some letter in Σ, besides those already present in the basis of some word φ, are allowed to the left or the right of some given position. Instead of going through all the left counters to the right of the position, and all the right counters to the left of the position, we make use of a saturation procedure that is applied once for each well formed counted word. This procedure strengthens each counter by taking into account the other counters in the counted word. Consider for instance the counted word φ in Example 6. 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 $l_2 \ominus_{\mathbb{C}} \mathbf{1}_a$ (which is defined by well formedness). Also, observe that l_2 and r_2 imply there are two occurences of a in any word belonging to $[\![\varphi]\!]$. We can therefore change $r_1(a)$ from $(v_a \geq 0)$ to $(v_a = 1)$. We normalize a well formed word using the normalization rules listed in Tab. 2.

Lemma 7 (Normalization rules). *Applying any of the rules of Tab. 2 on a well formed word φ does preserve its denotation, and hence its well formedness.*

Table 2. Given a counted word φ of length $n \geq 2$, each rule strengthens at most one counter. All rules preserve the base of φ. For instance $\mathrm{L(i,j)}$ returns an identical counted word except for the left counter at position i.

$$\frac{(\varphi)_{1,i-1} \cdot (l_i, \sigma_i, r_i) \cdot (\varphi)_{i+1,j-1} \cdot (l_j, \sigma_j, r_j) \cdot (\varphi)_{j+1,n}}{(\varphi)_{1,i-1} \cdot (l_i \sqcap_{\mathbb{C}} (l_j \ominus_{\mathbb{C}} (\mathbf{1}_{\sigma_i} \oplus_{\mathbb{C}} \ldots \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_{j-1}}), \sigma_i, r_i) \cdot (\varphi)_{i+1,j-1} \cdot (l_j, \sigma_j, r_j) \cdot (\varphi)_{j+1,n}} \ \mathrm{L(i,j)}$$

$$\frac{(\varphi)_{1,i-1} \cdot (l_i, \sigma_i, r_i) \cdot (\varphi)_{i+1,j-1} \cdot (l_j, \sigma_j, r_j) \cdot (\varphi)_{j+1,n}}{(\varphi)_{1,i-1} \cdot (l_i, \sigma_i, r_i) \cdot (\varphi)_{i+1,j-1} \cdot (l_j, \sigma_j, r_j \sqcap_{\mathbb{C}} (r_i \ominus_{\mathbb{C}} (\mathbf{1}_{\sigma_{i+1}} \oplus_{\mathbb{C}} \ldots \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_j}))) \cdot (\varphi)_{j+1,n}} \ \mathrm{R(i,j)}$$

$$\frac{(\varphi)_{1,i-1} \cdot (l_i, \sigma_i, r_i) \cdot (\varphi)_{i+1,j-1} \cdot (l_j, \sigma_j, r_j) \cdot (\varphi)_{j+1,n}}{(\varphi)_{1,i-1} \cdot (l_i, \sigma_i, r_i) \cdot (\varphi)_{i+1,j-1} \cdot ((l_j \sqcap_{\mathbb{C}} (l_i \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_i} \oplus_{\mathbb{C}} r_i) \ominus_{\mathbb{C}} (r_j \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_j})), \sigma_j, r_j) \cdot (\varphi)_{j+1,n}} \ \mathrm{LR(i,j)}$$

$$\frac{(\varphi)_{1,i-1} \cdot (l_i, \sigma_i, r_i) \cdot (\varphi)_{i+1,j-1} \cdot (l_j, \sigma_j, r_j) \cdot (\varphi)_{j+1,n}}{(\varphi)_{1,i-1} \cdot (l_i, \sigma_i, r_i \sqcap_{\mathbb{C}} ((r_j \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_j} \oplus_{\mathbb{C}} l_j) \ominus_{\mathbb{C}} (l_i \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_i}))) \cdot (\varphi)_{i+1,j-1} \cdot (l_j, \sigma_j, r_j) \cdot (\varphi)_{j+1,n}} \ \mathrm{RL(i,j)}$$

Proof. 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 L(i, j) and RL(i, j). We start with L(i, j) and show that $(l_j \ominus_{\mathbb{C}} (\mathbf{1}_{\sigma_i} \oplus_{\mathbb{C}} \ldots \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_{j-1}}))[(w)_{1,h(i)-1}]$. We have $l_j[(w)_{1,h(j)-1}]$ from $w \models^h \varphi$. We know $l_j[(w)_{h(1)} \cdot (w)_{h(2)} \cdots (w)_{h(j-1)}]$ by well formedness of φ and $w \models^h \varphi$. Notice that $((w)_{h(1)} \cdot (w)_{h(2)} \cdots (w)_{h(j-1)})^{\#} \preceq ((w)_{1,h(i)-1} \cdot (w)_{h(i)} \cdot (w)_{h(i+1)} \cdots (w)_{h(j-1)})^{\#} \preceq ((w)_{1,h(j)-1})^{\#}$. By the convexity property (lemma 1), we get $l_j[((w)_{1,h(i)-1} \cdot (w)_{h(i)} \cdot (w)_{h(i+1)} \cdots (w)_{h(j-1)})^{\#}]$. This ensures $(l_j \ominus_{\mathbb{C}} (\mathbf{1}_{\sigma_i} \oplus_{\mathbb{C}} \ldots \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_{j-1}}))[(w)_{1,h(i)-1}]$ and hence the result. For RL(i, j), we show $((r_j \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_j} \oplus_{\mathbb{C}} l_j) \ominus_{\mathbb{C}} (l_i \oplus_{\mathbb{C}} \mathbf{1}_{\sigma_i}))[(w)_{h(i)+1,|w|}]$. 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 $w^{\#} = ((w)_{1,h(i)})^{\#} \oplus ((w)_{h(i)+1,|w|})^{\#}$. □

Lemma 8 (Normalization). *Given a well formed word, the Procedure norm always returns the same counted word independently of the application order.*

Proof. First termination. At each rule, manipulated and obtained counted words are well formed by lemma 7. Using lemma 5, we deduce all counters belong to a finite lattice where each one of the individual rules corresponds to a monotonic function that is always enabled and that only strengthens one counter. Unicity can be obtained by contradiction. Suppose two different counted words are obtained as normalizations of the same well formed word. The words can only differ in their counters. Pick different corresponding counters. Given the allowed forms for the predicates (lemmas 5 and 7), 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. □

Procedure norm(φ)

```
   input  : A well formed counted word $\varphi$
   output: $\varphi$ with stronger counters and identical base and denotation
1  while rule($\varphi$) $\neq \varphi$ for some rule in Tab.2 do
2      $\varphi$ := rule($\varphi$)
3  return $\varphi$;
```

Entailment and minimal sets. We write $\mathbb{NW}$ to mean the set of normalized words in $\mathbb{W}$. Assume two normalized words $\varphi = (l_1, \sigma_1, r_1) \cdots (l_n, \sigma_n, r_n)$ and $\varphi' = (l'_1, {\sigma'}_1, r'_1) \cdots (l'_m, {\sigma'}_m, r'_m)$. We say that φ is h-entailed by φ' for some increasing injection $h : [1 : n] \to [1 : m]$, and write $\varphi \trianglelefteq^h_{\mathbb{NW}} \varphi'$, to mean that for each $i \in [1 : n]$, $(b(\varphi))_i = (b(\varphi'))_{h(i)}$, $l_i \trianglelefteq_{\mathbb{C}} l'_{h(i)}$, and $r_i \trianglelefteq_{\mathbb{C}} r'_{h(i)}$. We write $\varphi \trianglelefteq_{\mathbb{NW}} \varphi'$ if $\varphi \trianglelefteq^h_{\mathbb{NW}} \varphi'$ for some h. Observe that $([v_a \geq 0], a, [v_a = 0]) \not\trianglelefteq_{\mathbb{NW}} ([v_a = 0], a, [v_a \geq 0])$, but $[\![([v_a \geq 0], a, [v_a = 0])]\!] = [\![([v_a = 0], a, [v_a \geq 0])]\!] = a^+$. We say that a set $\{\varphi_1, \ldots \varphi_n\}$ of normalized words is *minimal* if $i \neq j$ implies $\varphi_i \not\trianglelefteq_{\mathbb{NW}} \varphi_j$.

Lemma 9 (Entailment). *$\trianglelefteq_{\mathbb{NW}}$ is reflexive and transitive. $\varphi \trianglelefteq_{\mathbb{NW}} \varphi'$ implies $[\![\varphi']\!] \subseteq [\![\varphi]\!]$ and it can be checked linearly in $|\varphi| + |\varphi'|$.*

Proof. Follows from reflexivity and transitivity of $\trianglelefteq_{\mathbb{C}}$. Also, if $w \models^h \varphi'$ and $\varphi \trianglelefteq^{h'} \varphi'$ then $w \models^{h \circ h'} \varphi$ with $\circ$ is function composition.

Word cutoffs. Similarly to the cutoffs defined in Sec. 2 for counters, the cutoff 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 $\varphi\}$. For instance, in example 6, $\kappa(\varphi)(a) = 2$ and $\kappa(\varphi)(b) = 1$. We say that a counted word φ has a k-cutoff if all its counters are in $\mathbb{C}_k$. For example, counted words with a 0-cutoff only have inequalities in their counters (i.e. denote all words from which some letters can be deleted in order to obtain the base of the counted word, i.e., upward closed sets of words). We write $\mathbb{W}_k$ ($\mathbb{NW}_k$) to mean the set of (normalized) well formed counted words that have a k-cutoff.

Theorem 10 (WQO). *For $k \in \mathbb{N}$, $(\mathbb{NW}_k, \trianglelefteq_{\mathbb{NW}})$ 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. We let $\Gamma = \mathbb{C}_k \times \Sigma \times \mathbb{C}_k$ and $(l, \sigma, r) \preceq (l', \sigma', r')$ if $l \trianglelefteq_{\mathbb{C}} l'$ and $\sigma = \sigma'$ and $r \trianglelefteq_{\mathbb{C}} r'$. Observe that $\mathbb{NW}_k \subseteq \Gamma^*$, and that $\preceq^*$ coincides with $\trianglelefteq_{\mathbb{NW}}$. Hence, showing $(\Gamma, \preceq)$ 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 4. □

The closure operator. We will make use of the fact that $(\mathbb{NW}_k, \trianglelefteq_{\mathbb{NW}})$ is a well quasi ordering (Thm. 10) in order to ensure termination of our reachability algorithm (Sect. 7). For this reason, the reachability algorithm systematically over-approximates generated counted words with elements in $\mathbb{NW}_k$ where k is a parameter of the reachability algorithm. We explain in the following how we compute over-approximations of a counted word using the *closure* operator. Relaxing a counter $cr = [\wedge_{\sigma \text{ in } \Sigma}(v_\sigma \sim k_\sigma)]$, wrt. a multiset m, written $\rho_m(cr)$, results in the counter $[\wedge_{\sigma \text{ in } \Sigma}(v_\sigma \sim' k_\sigma)]$ s.t. $(v_\sigma \sim' k_\sigma)$ equals $(v_\sigma \geq k_\sigma)$ if $(v_\sigma \sim k_\sigma)$ was $(v_\sigma = k_\sigma)$ in cr with $k_\sigma \geq m(\sigma)$, and $(v_\sigma \sim k_\sigma)$ otherwise. In other words, relaxation wrt. m replaces by inequalities those equalities that involve constants larger or equal to what is allowed by m. Observe that $\kappa(\rho_m(cr)) \preceq m$. for any multiset m over Σ. The *closure* of a counted word φ wrt. a multiset m is simply the word $\rho_m(\varphi)$ obtained by normalizing the result of relaxing all counters in φ wrt. m.

Lemma 11 (Closure). *Let φ, φ' be two normalized counted words and m, m' be two multisets. Each of the following holds:*

1. *(monotonicity)* $\varphi \trianglelefteq_{\mathbb{NW}} \varphi' \implies \rho_m(\varphi) \trianglelefteq_{\mathbb{NW}} \rho_m(\varphi')$
2. *(restrictivity)* $\rho_m(\varphi) \trianglelefteq_{\mathbb{NW}} \varphi$
3. *(idempotency)* $\rho_m(\rho_m(\varphi)) = \rho_m(\varphi)$
4. *(strengthening)* $m \preceq m' \implies \rho_m(\varphi) \trianglelefteq_{\mathbb{NW}} \rho_{m'}(\varphi)$
5. *(bounded cutoff)* $\kappa(\rho_m(\varphi))(\sigma) \leq max(0, 2m(\sigma) - 1)$ *for each* $\sigma \in \Sigma$.

Proof. We show restrictivity. The arguments for monotonicity and strengthening are similar. Suppose $\rho_m(\varphi) \not\trianglelefteq_{\mathbb{NW}} \varphi$, then there is a counter cr in φ such that $cr_\rho \not\trianglelefteq_{\mathbb{C}} cr$ where cr_ρ is the corresponding counter in $\rho_m(\varphi)$. This is not possible. Indeed, before normalizing, the relaxed φ and φ are both well formed with the same base. The normalization $\rho_m(\varphi)$ of the relaxed φ starts with weaker counters than those in φ. By applying to φ the sequence of normalization rules used on the relaxed φ, we obtain (by monotonicity of $\sqcap_{\mathbb{C}}$) that the counters in $\rho_m(\varphi)$ are weaker than those in φ. Idempotency is obtained by remarking that relaxing all counters wrt. m in φ and in $\rho_m(\varphi)$ yields the same well formed word. The strongest cutoff $(2m(\sigma) - 1)$ is obtained when both left and right counters in some tuple (l, σ, r) associate the predicate $v_\sigma = (m(\sigma) - 1)$ to the letter σ. One can show by induction on the number of applications of the normalization rules, that for any letter σ', $\kappa(l \oplus_{\mathbb{C}} \mathbf{1}_\sigma \oplus_{\mathbb{C}} r)(\sigma') \leq max(0, 2m(\sigma') - 1)$. □

5 Operations on Sets Counted Words

We define in this section operations to capture intersections and separations of two finite sets of counted words. These operations will be particularly useful in the refinement part of our generic scheme introduced in Sect. 7.

Intersection of counted words. In our reachability algorithm (Sect. 7), we will need to compute a set of counted words capturing the intersection of the denotations of two counted words. We use Proc. prod and Proc. inter in order to compute, for given normalized words φ and φ' in $\mathbb{NW}$, a set of normalized words (written inter(φ, φ')) such that the union of the denotation of its elements coincides with $[\![\varphi]\!] \cap [\![\varphi']\!]$. Intuitively, if a word w belongs to both $[\![\varphi]\!]$ and $[\![\varphi']\!]$

(i.e. $w \models^h \varphi$ and $w \models^{h'} \varphi'$) then the bases of φ and φ' can each be obtained by deleting some of the letters in w. In other words, w contains a word w' that is the product of the bases of φ and φ' with respect to two injections g and g' (see Fig. 3).

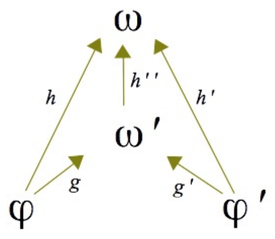


Fig. 1. A word w belonging to the denotation of two counted words φ and φ'

Proc. prod builds a product $\varphi \boxtimes \varphi'$ that is similar to the product $\otimes$ of Sect. 2. Formally, $\varphi \boxtimes \varphi'$ is the set $\{\pi \mid h \in \mathcal{H}_{|\varphi|}^{|\pi|}$ and $h' \in \mathcal{H}_{|\varphi'|}^{|\pi|}$ and $\pi = \varphi \overset{h,h'}{\boxtimes} \varphi'\}$; where $\pi = \varphi \overset{h,h'}{\boxtimes} \varphi'$ if $[1 : |\pi|] = h([1 : |\varphi|]) \cup h'([1 : |\varphi'|]$ and, for each $j : 1 \leq j \leq |\pi|$, one of the following conditions hold:

1. $h(i) \notin h'([1 : |\varphi'|])$ and $(\pi)_{h(i)} = (l((\varphi)_i), b((\varphi)_i), r((\varphi)_j))$
2. $h(i) = h'(i')$ and $b((\varphi)_i) = b((\varphi')_{i'})$ and $(\pi)_{h(i)}$ is equal to the triplet $(l((\varphi)_i) \sqcap_{\mathbb{C}} l((\varphi')_{i'}), b((\pi)_{h(i)}), r((\varphi)_i) \sqcap_{\mathbb{C}} r((\varphi')_{i'}))$
3. $h'(i') \notin h([1 : |\varphi|])$ and $(\pi)_{h'(i')} = (l((\varphi')_{i'}), b((\varphi')_i), r((\varphi')_{i'}))$

Intuitively, the product $\varphi \boxtimes \varphi'$ mimics the product $b(\varphi) \otimes b(\varphi')$ and considers contributions in the form of one letter at a time from φ (case 1 above), φ' (case 3), or both (case 2). A letter is simply copied if it comes from either φ or φ' (cases 1,3). If it comes from both, then the new counters result from the meet of the counters of φ and φ'.

```
Procedure prod($\pi, \varphi, \varphi'$)
input  : three counted words $\pi, \varphi, \varphi'$
output: $\{\pi \cdot \pi' | \ \pi' \in \varphi \boxtimes \varphi'\}$
1  let $\Pi := \emptyset$;
2  if $(\varphi = \epsilon)$ then return $\{\pi \cdot \varphi'\}$;
3  if $(\varphi' = \epsilon)$ then return $\{\pi \cdot \varphi\}$;
4  let $(l, \sigma, r), (l', \sigma', r') := (\varphi)_1, (\varphi')_1$;
5  if $\sigma = \sigma'$ then
6      let $p := (l \sqcap_{\mathbb{C}} l', \sigma, r \sqcap_{\mathbb{C}} r')$;
7      $\Pi \cup := \texttt{prod}(\pi \cdot p, (\varphi)_{2,|\varphi|}, (\varphi')_{2,|\varphi'|})$;
8  $\Pi \cup := \texttt{prod}(\pi \cdot (l, \sigma, r), (\varphi)_{2,|\varphi|}, \varphi')$;
9  $\Pi \cup := \texttt{prod}(\pi \cdot (l', \sigma', r'), \varphi, (\varphi')_{2,|\varphi'|})$;
10 return $\Pi$;
```

Lemma 12 establishes that Proc. prod can be used to build the set of all products for any given pair of counted words.

Lemma 12. *For any counted words φ, φ',* prod$(\epsilon, \varphi, \varphi')$ *returns the set $\varphi \boxtimes \varphi'$.*

Proof. By induction on $|\varphi| + |\varphi'|$. □

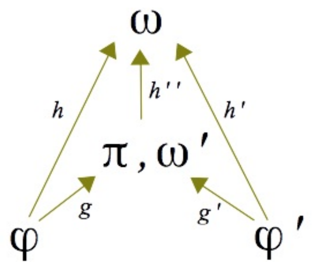
Proc. inter below takes as argument two counted words φ, φ' and computes a set of well formed and normalized counted words inter(φ, φ') such that the union of the denotation of its elements coincides with $[\![\varphi]\!] \cap [\![\varphi']\!]$. For this, it invokes Proc. prod in order to obtain the set $\varphi \boxtimes \varphi'$. It filters out the elements that are not well formed and returns the remaining ones after having normalized them.

```
Procedure inter($\varphi, \varphi'$)
input  : $\varphi, \varphi' \in \mathbb{NW}$
output: a set $\varphi \sqcap_{\mathbb{NW}} \varphi'$ of well formed normalized words
        $\{\varphi_1, \ldots, \varphi_n\}$ s.t. $\varphi \trianglelefteq_{\mathbb{NW}} \varphi_i$ and $\varphi' \trianglelefteq_{\mathbb{NW}} \varphi_i$ for
        $i : 1 \leq i \leq n$, and $[\![\varphi]\!] \cap [\![\varphi']\!] = \cup_{i \in [1:1]n} [\![\varphi_i]\!]$
1  let result := $\emptyset$;
2  foreach $\pi$ in prod$(\epsilon, \varphi, \varphi')$ do
3      if $\texttt{wf}(\pi)$ then
4          result $\cup :=$ norm$(\pi)$
5  return result;
```

Lemma 13 establishes correctness of Proc. inter.

Lemma 13 (intersection). *Given two counted words φ, φ' in $\mathbb{NW}$,* $\texttt{inter}(\varphi, \varphi')$ *gives a set $\{\varphi_1, \ldots \varphi_n\}$ s.t. $(\varphi \trianglelefteq_{\mathbb{NW}} \varphi_i)$, $(\varphi' \trianglelefteq_{\mathbb{NW}} \varphi_i)$ for each $i : 1 \leq i \leq n$, and $\cup_{i \ in \ [1:n]} [\![\varphi_i]\!] = [\![\varphi]\!] \cap [\![\varphi']\!]$.*

Proof. lemma 12 implies $\mathtt{inter}(\varphi, \varphi')$ returns $\{\mathtt{norm}(\pi) \mid \pi = \varphi \overset{h,h'}{\boxtimes} \varphi'$ for $h \in \mathcal{H}^{|\pi|}_{|\varphi|}$ and $h' \in \mathcal{H}^{|\pi|}_{|\varphi'|}$ and π is well formed$\}$. Assume π is a product $\varphi \overset{f,f'}{\boxtimes} \varphi'$. Observe the product $\boxtimes$ guarantees $b(\varphi) \preceq^f b(\pi)$ and for each $i \in [1 : |\varphi|]$, both $l((\varphi)_i) \trianglelefteq_{\mathbb{C}} l((\pi)_{f(i)})$ and $r((\varphi)_i) \trianglelefteq_{\mathbb{C}} r((\pi)_{f(i)})$ hold. The rules of Tab.2 only strengthen the counters of π, and hence do not violate these properties. We deduce $(\varphi \trianglelefteq^f_{\mathrm{NW}} \mathtt{norm}(\pi))$ and $(\varphi' \trianglelefteq^{f'}_{\mathrm{NW}} \mathtt{norm}(\pi))$. Hence, $[\![\pi]\!] \subseteq [\![\varphi]\!] \cap [\![\varphi']\!]$.


$$\begin{tikzcd} & \omega & \\ & \pi, \omega' \arrow[u, "h''"] & \\ \varphi \arrow[uur, "h"] \arrow[ur, "g"'] & & \varphi' \arrow[uul, "h'"'] \arrow[ul, "g'"] \end{tikzcd}$$

Moreover, assume a word w that satisfies both $(w \models^h \varphi)$ and $(w \models^{h'} \varphi')$ for some injections $h \in \mathcal{H}^{|w|}_{|\varphi|}$ and $h' \in \mathcal{H}^{|w|}_{|\varphi'|}$. We exhibit a well formed counted word π in $\varphi \boxtimes \varphi'$ such that $w \in [\![\pi]\!]$. By lemma 7, $w \in [\![\mathtt{norm}(\pi)]\!]$. We let $w' = (w)_{p_1} \cdots (w)_{p_k}$ be the word obtained by keeping exactly those indices, denoted $p_1, \ldots, p_k$ in increasing order, that belong to the union of the images of h and h'. In other words, $\{p_1, \ldots, p_k\} = h([1 : |\varphi|]) \cup h'([1 : |\varphi'|])$. This defines an increasing injection $h'' \in \mathcal{H}^w_{w'}$ with $h''(i) = p_i$ for each $i \in [1 : |w'|]$. Furthermore, we define the injections $g \in \mathcal{H}^{|w'|}_{|\varphi|}$ and $g' \in \mathcal{H}^{|w'|}_{|\varphi'|}$ with $g(i) = j$ when $h(i) = p_j$ for each $i \in [1 : |\varphi|]$, and $g'(i') = j$ when $h'(i') = p_j$ for each $i' \in [1 : |\varphi'|]$. By construction and well formedness of φ and φ', both $(w' \models^g \varphi)$ and $(w' \models^{g'} \varphi')$ hold. Observe that $g([1 : |\varphi|]) \cup g'([1 : |\varphi'|]) = [1 : |w'|]$. Let π be the product $\varphi \overset{g,g'}{\boxtimes} \varphi'$. π is well formed as otherwise $w' \not\models^g \varphi$ or $w' \not\models^{g'} \varphi'$. Observe that $b(\pi) = w'$. By lemma 12, $\mathtt{norm}(\pi)$ is in $\mathtt{inter}(\varphi, \varphi')$. We show $w \models^{h''} \pi$. Since $b(\pi) = w'$ and w' can be obtained from w by deleting those elements in w that are not in the image of h'', we deduce that $(b(\pi))_i = (w)_{h''(i)}$ for each $i \in [1 : |\pi|]$. In addition, the counters of π at position $i \in [1 : |\pi|]$ are either obtained as the counters of the position $g^{-1}(i)$ in φ if $i \notin g'([1 : |\varphi'|])$, those at position $(g')^{-1}(i)$ in φ' if $i \notin g([1 : |\varphi|])$, or as the meet of both in case $i \in g([1 : |\varphi|]) \cap g'([1 : |\varphi'|])$. Hence, $l((\pi)_i)[(w')_{1,h''(i)-1}]$ and $r((\pi)_i)[(w)_{h''(i)+1,|w'|}]$ for $i \in [1 : |\pi|]$. □

Procedure prodpair($\langle \pi, \varphi \rangle, \langle \breve{\pi}, \breve{\varphi} \rangle, \varphi'$)

```
input  : Five counted words $\pi, \varphi, \breve{\pi}, \breve{\varphi}, \varphi'$ s.t. $|\varphi| = |\breve{\varphi}|$
output: $\{\langle \pi \cdot p, \breve{\pi} \cdot \breve{p} \rangle \mid p = \varphi \overset{h,h'}{\boxtimes} \varphi'$ and $\breve{p} = \breve{\varphi} \overset{h,h'}{\boxtimes} \varphi'\}$
1  let $\mathit{\Pi} := \emptyset$;
2  if $(\varphi = \epsilon)$ then return $\{\langle \pi \cdot \varphi', \breve{\pi} \cdot \varphi' \rangle\}$;
3  if $(\varphi' = \epsilon)$ then return $\{\langle \pi \cdot \varphi, \breve{\pi} \cdot \breve{\varphi} \rangle\}$;
4  let $(l, \sigma, r), (l', \sigma', r'), (\breve{l}, \breve{\sigma}, \breve{r}) := (\varphi)_1, (\varphi')_1, (\breve{\varphi})_1$;
5  if $\sigma = \sigma'$ then
6      let $p, \breve{p} := (l \sqcap_{\mathbb{C}} l', \sigma, r \sqcap_{\mathbb{C}} r'), (\breve{l} \sqcap_{\mathbb{C}} l', \sigma, \breve{r} \sqcap_{\mathbb{C}} r')$;
7      $\mathit{\Pi} \cup :=$ prodpair($\left\langle \pi \cdot p, (\varphi)_{2,|\varphi|} \right\rangle, \left\langle \breve{\pi} \cdot \breve{p}, (\breve{\varphi})_{2,|\breve{\varphi}|} \right\rangle, (\varphi')_{2,|\varphi'|}$);
8  $\mathit{\Pi} \cup :=$ prodpair($\left\langle \pi \cdot (l, \sigma, r)), (\varphi)_{2,|\varphi|} \right\rangle, \left\langle \breve{\pi} \cdot (\breve{l}, \breve{\sigma}, \breve{r}), (\breve{\varphi})_{2,|\breve{\varphi}|} \right\rangle, \varphi'$);
9  $\mathit{\Pi} \cup :=$ prodpair($\langle \pi \cdot (l', \sigma', r'), \varphi \rangle, \langle \breve{\pi} \cdot (l, \sigma, r), \breve{\varphi} \rangle, (\varphi')_{2,|\varphi'|}$);
10 return $\mathit{\Pi}$;
```

Separation operator. The over-approximation entailed by the usage of closures of counted words during our reachability algorithm (Sect. 7) can yield false alarms in the form of traces. We follow such traces using non-approximated counted words and systematically check intersections with those obtained during the approximate analysis. If the trace is spurious, we can identify (algorithm 2 in Sect. 7) two counted words φ (capturing reachable configurations) and φ' (approximated configurations that violate the safety property) such that $\texttt{inter}(\varphi, \varphi') = \emptyset$ but $\texttt{inter}(\rho_m(\varphi), \varphi') \neq \emptyset$. We describe in the following a procedure to deduce a new multiset m' such that $\texttt{inter}(\rho_{m'}(\varphi), \varphi') = \emptyset$. One can choose m' to be $\kappa(\varphi)$, but that would result in large multisets that will slow down the analysis. Instead we derive a formula that exactly captures all multisets that are larger than m and that ensure $\texttt{inter}(\rho_{m'}(\varphi), \varphi') = \emptyset$.

Recall that the closure $\rho_m(\varphi)$ of φ with respect to m results in that some equalities of the form $v_\sigma = k_\sigma$ in the counters of φ are relaxed into inequalities $v_\sigma \geq k_\sigma$. The situation where $\texttt{inter}(\varphi, \varphi') = \emptyset$ but $\texttt{inter}(\rho_m(\varphi), \varphi') \neq \emptyset$ occurs because the relaxed equalities allow some of the products of $\rho_m(\varphi) \boxtimes \varphi'$ to be well formed when none of the products in $\varphi \boxtimes \varphi'$ was well formed. In other words, the more equalities in φ are preserved in $\rho_m(\varphi)$, the less products in $\rho_m(\varphi) \boxtimes \varphi'$ are well formed. If all equalities are preserved, we get $\rho_m(\varphi) = \varphi$ and $\rho_m(\varphi) \boxtimes \varphi'$. We describe Proc. separate that returns a formula avoid such that $\texttt{inter}(\rho_{m'}(\varphi), \varphi') = \emptyset$ for any multiset m' satisfying avoid. For this, Proc. separate starts by having avoid require any accepted multiset m' to be larger or equal to m (line 3,4). This ensures that if a product $\rho_m(\varphi) \overset{h,h'}{\boxtimes} \varphi'$ is well formed then the product $\rho_m(\varphi) \overset{h,h'}{\boxtimes} \varphi'$. is also well formed.

The procedure then invokes Proc. prodpair with the argument $(\langle \epsilon, \varphi \rangle, \varphi', \langle \epsilon, \rho_m(\varphi) \rangle)$. Intuitively, prodpair simultaneously mimics both prod$(\epsilon, \varphi, \varphi')$ and prod$(\epsilon, \rho_m(\varphi), \varphi')$; prodpair results in the set of pairs of products of the form $\langle \pi, \breve{\pi} \rangle$ in $(\varphi \boxtimes \varphi') \times$

$(\breve{\varphi} \boxtimes \varphi')$ where $\pi = \varphi \overset{h,h'}{\boxtimes} \varphi'$ and $\breve{\pi} = \rho_{m'}(\varphi) \overset{h,h'}{\boxtimes} \varphi'$ with respect to the same injections $h \in \mathcal{H}_{|\varphi|}^{|\pi|}$ and $h' \in \mathcal{H}_{|\varphi'|}^{|\pi|}$. For each well formed (and hence non empty) $\breve{\pi}$, Proc. separate builds a disjunction individual_avoid of constraints. Each one of these constraints aims at preserving one equality in φ that forbids well formedness of the product π and whose relaxations wrt. m allows the product $\breve{\pi}$ (conditions at lines 9, 11). To do that, the constraint requires that a multiset has to associate a strictly larger cuttoff to forbid the concerned relaxation (lines 10, 12). If a multiset m' satisfies one of these constraints (hence the disjunction at lines 10, 12), then the product $\rho_{m'}(\varphi) \overset{h,h'}{\boxtimes} \varphi'$ will not be well formed (lemma 11 and the tests at lines 9 and 11). The procedure conjuncts the disjunctions obtained for each word (line 13) hence eliminating all well formed words from the product. Lemma 14 establishes correctness of the separation procedure.

Procedure separate(φ, φ', m)

```
   input : $\varphi, \varphi' \in \mathbb{NW}$ and multiset $m$ s.t. $\texttt{inter}(\varphi, \varphi') = \emptyset$ but
           $\texttt{inter}(\rho_m(\varphi), \varphi') \neq \emptyset$
   output: A formula avoid s.t. avoid$[m']$ implies $\texttt{inter}(\rho_{m'}(\varphi), \varphi') \neq \emptyset$
 1 let $w := b(\pi)$;
 2 let avoid := $\texttt{true}$;
 3 foreach $\sigma$ in $\Sigma$ do
 4     avoid := avoid $\wedge$ $(v_\sigma \geq m(\sigma))$;
 5 foreach $\langle \pi, \breve{\pi} \rangle$ in prodpair$(\langle \epsilon, \varphi \rangle, \varphi', \langle \epsilon, \rho_m(\varphi) \rangle)$ do
 6     assert$(\neg\texttt{wf}(\pi))$;
 7     if $\texttt{wf}(\breve{\pi})$ then
 8         let individual_avoid := $\texttt{false}$;
 9         for $i := 1$ to $|\pi|$ do
10             foreach $\sigma$ in $\Sigma$ do
11                 if $l((\varphi)_i)(\sigma) = (v_\sigma = k_\sigma)$ and $k_\sigma < ((w)_{1,i-1})^{\#}(\sigma)$ then
12                     individual_avoid := individual_avoid $\vee$ $(v_\sigma > k_\sigma)$
13                 if $r((\varphi)_i)(\sigma) = (v_\sigma = k_\sigma)$ and $k_\sigma < ((w)_{i+1,|\pi|})^{\#}(\sigma)$ then
14                     individual_avoid := individual_avoid $\vee$ $(v_\sigma > k_\sigma)$
15         avoid := avoid $\wedge$ individual_avoid;
```

Lemma 14 (separation). *Assume* $\texttt{inter}(\varphi, \varphi') = \emptyset$ *and* $\texttt{inter}(\rho_\varphi(m), \varphi') \neq \emptyset$*. If* $\texttt{separate}(\varphi, \varphi', m)$ *returns formula* avoid *and multiset* m' *satisfies* avoid *then* $\texttt{inter}(\rho_{m'}(\varphi), \varphi') = \emptyset$*. In addition,* avoid$[m'] \wedge m' \preceq m''$ *implies* avoid$[m'']$.

Proof. Lemmas 5 and 11 ensure that given $m \preceq m'$ and some $i : 1 \leq i \leq |\varphi|$, if $l((\rho_m(\varphi))_i)(\sigma)$ is an equality then $l((\rho_{m'}(\varphi))_i)(\sigma)$ is the same equality. A similar fact can be deduced for $r((\rho_m(\varphi))_i)(\sigma)$ and $r((\rho_{m'}(\varphi))_i)(\sigma)$. Fix some m' satisfying avoid. Let $\pi = \varphi \overset{h,h'}{\boxtimes} \varphi'$, $\breve{\pi} = \rho_\varphi(m) \overset{h,h'}{\boxtimes} \varphi'$, and $\breve{\pi}' = \rho_\varphi(m') \overset{h,h'}{\boxtimes} \varphi'$. π cannot be well formed as $\texttt{inter}(\varphi, \varphi') = \emptyset$. Since avoid$[m']$ holds, then $m \preceq m'$ (lines 2-4, 15). If $\breve{\pi}$ is not well formed, then it is because of some equalities in its counters (inequalities where satisfied in φ and φ' and adding more letters does not violate them, so they cannot be the reason for non well formedness). $\breve{\pi}'$ has at least the same equalities as $\breve{\pi}$ because $m \preceq m'$. For this reason, $\breve{\pi}'$ is not well formed either. Suppose both $\breve{\pi}$ and $\breve{\pi}'$ are well formed. By construction, there is a position $i : 1 \leq i \leq |\pi|$ (line 9) and a letter σ (line 10) such such that the left (lines 11,12) or right (lines 13,14) counters violate well formedness of π as they do not accept the base of π. At least one of these counters is preserved when relaxing wrt. to m', and both π and $\breve{\pi}'$ have the same base. $\breve{\pi}'$ is therefore not well formed. In addition, avoid accepts an upward closed set of multisets since all atomic predicates in avoid are (strict) inequalities of the form $v \sim k$ with $\sim \in \{>, \geq\}$.

Let Φ, Φ' be two finite subsets of $\mathbb{NW}$. We write $\Phi \sqcap_{\mathbb{NW}} \Phi'$ to mean the union of each `inter`(φ, φ') for $\varphi \in \Phi$ and $\varphi' \in \Phi'$. In addition, if $\Phi \sqcap_{\mathbb{NW}} \Phi' = \emptyset$, we write separate$(\Phi, \Phi', m)$ to mean the conjunction $\bigwedge_{\varphi \in \Phi, \varphi' \in \Phi'}(\text{separate}(\varphi, \varphi', m))$.

6 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 $\mathcal{P} = (\Sigma, T)$, where Σ 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 $\sigma \to \sigma'$. It allows a process to change its local state from σ to σ' independently of the local states of the other processes. A global transition is of the form $\sigma \to \sigma' : \Box S$, where $\Box \in \{\exists_L, \exists_R, \exists_{LR}, \forall_L, \forall_R, \forall_{LR}\}$ and $S \subseteq \Sigma$. For instance, the condition $\forall_L S$ means that "all processes to the left should be in local states that belong to S". 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 (Σ, T) induces an infinite-state transition system where $C = \Sigma^*$ is the set of *configurations* and $\longrightarrow$ is a transition relation on C. For configurations $c = c_1 \sigma c_2$, $c' = c_1 \sigma' c_2$, and a transition $t \in T$, we write $c \longrightarrow_t c'$ to mean:

- t is a local transition of the form $\sigma \to \sigma'$, or
- t is a global transition $\sigma \to \sigma' : \Box S$, and one of the following holds:
  - either $\Box S = \exists_L S$ and $c_1{}^\bullet \cap S \neq \emptyset$, or $\Box S = \exists_R S$ and $c_2{}^\bullet \cap S \neq \emptyset$, or $\Box S = \exists_{LR} S$ and $(c_1{}^\bullet \cup c_2{}^\bullet) \cap S \neq \emptyset$.
  - or $\Box S = \forall_L S$ and $c_1{}^\bullet \subseteq S$, or $\Box S = \forall_R S$ and $c_2{}^\bullet \subseteq S$, or $\Box S = \forall_{LR} S$ and $(c_1{}^\bullet \cup c_2{}^\bullet) \subseteq S$.

We write $\longrightarrow$ to mean $\cup_{t \in T} \longrightarrow_t$ and use $\xrightarrow{*}$ to denote its reflexive transitive closure. We assume all processes have the same *initial* state. We use *Init* to denote the set of *initial* configurations. *Init* is infinite. Using standard techniques (see e.g. [20]), checking safety properties (expressed as regular languages) can be translated into instances of the following reachability problem: given $\mathcal{P} = (\Sigma, T)$ and a possibly infnite set C_F of configurations, check whether $\mathit{Init} \xrightarrow{*} C_F$.

Post and Pre operators. In the following, we describe how to compute counted words (written $\mathrm{post}_t(\varphi)$) exactly capturing the successors (for transition $t \in T$) of configurations denoted by a counted word φ in $\mathbb{NW}$. Assume the transition t is either local ($\sigma \to \sigma'$) or global ($\sigma \to \sigma' : \Box S$). First, we define an operator $\sigma \dotplus \varphi$ that takes a strengthened well formed word φ and a state σ and returns all pairs (φ', p) for $p : 1 \leq p \leq |\varphi| + 1$ such that either:

1. $\varphi' = \varphi$ with $p : 1 \leq p \leq |\varphi|$ and $b((\varphi)_p) = \sigma$, or
2. $\varphi' = (\varphi)_{1,p-1} \cdot (cr, \sigma, cr') \cdot (\varphi)_{p,|\varphi|}$ with $\kappa(cr)(\sigma) = \kappa(cr')(\sigma) = 0$ and where each one of the following two conditions hold:
   (a) if $p = 1$ then $cr' = \top_{\mathbb{C}}$ otherwise $cr' = r((\varphi)_{p-1})$
   (b) if $p = |\varphi| + 1$ then $cr = \top_{\mathbb{C}}$ otherwise $cr = l((\varphi)_{p+1})$

Intuitively, if it is possible to find or place the state σ at some position $p : 1 \leq p \leq |\varphi| + 1$ of φ, then there will be a pair (φ', p) in $\sigma \dotplus \varphi$ to capture that. In addition, given a state σ' and a pair (φ', p) such that $b((\varphi')_p) = \sigma$, we write $(\varphi', p)_{\sigma \to \sigma'}$ to mean the counted word φ'' of length $|\varphi'|$ and such that:

1. for each $i : 1 \leq i \leq p-1$, $(\varphi'')_i = (l((\varphi')_i), b((\varphi')_i), r((\varphi')_i) \oplus_{\mathbb{C}} \mathbf{1}_{\sigma'} \ominus_{\mathbb{C}} \mathbf{1}_{\sigma})$
2. for $i = p$, $(\varphi'')_i = (l((\varphi')_i), \sigma', r((\varphi')_i))$
3. for each $i : p+1 \leq i \leq |\varphi'|$, $(\varphi'')_i = (l((\varphi')_i) \oplus_{\mathbb{C}} \mathbf{1}_{\sigma'} \ominus_{\mathbb{C}} \mathbf{1}_{\sigma}, b((\varphi')_i), r((\varphi')_i))$

Intuitively, we know that $b((\varphi')_p) = \sigma$. The counted word $(\varphi', p)_{\sigma \to \sigma'}$ is obtained by replacing the state σ at position p in φ' with state σ', and by updating the right counters to the left of p, and the left counters to the right of p in order to take this replacement into account.

Assume a counted word φ. The set $\mathrm{post}_t(\varphi)$, is the smallest set containing strengthenings of all counted words ψ such that one of the following cases holds.

1. t is a local transition $\sigma \to \sigma'$ and ψ is equal to $(\varphi', p)_{\sigma \to \sigma'}$ for some (φ', p) in $\sigma \dotplus \varphi$, or
2. t is a global transition of the form $\sigma \to \sigma' : \forall_L S$, and there is a pair (φ, p) in $\sigma \dotplus \varphi'$ s.t. $b((\varphi')_{1,p-1})^{\bullet} \subseteq S$. In addition, let φ'' be a word of the same length as φ' and such that
   (a) $(\varphi'')_i = (l((\varphi')_i) \sqcap_{\mathbb{C}} \mathbf{0}_{\Sigma \setminus S}, b((\varphi')_i), r((\varphi')_i))$ for each $i : 1 \leq i \leq p$
   (b) $(\varphi'')_i = (l((\varphi')_i), b((\varphi')_i), r((\varphi')_i))$ for each $i : p+1 \leq i \leq |\varphi'|$
   ψ is then equal to $(\varphi'', p)_{\sigma \to \sigma'}$ Intuitively, we check first that there is at least a (possibly empty) prefix in S. If it is the case, we require that all accepted multisets to the left of σ only contain states in S. In addition, we replace σ by σ' at position p and update counters like in the previous case, or
3. t is a global transition of the form $\sigma \to \sigma' : \exists_L S$, and there is a pair (φ', p) in $\sigma \dotplus \varphi$ and a pair (φ'', p') in $\sigma'' \dotplus \varphi'$ for some $p' : 1 \leq p' < p$ and $\sigma'' \in S$. ψ is then equal to $(\varphi'', p+1)_{\sigma \to \sigma'}$. Intuitively, we make sure there is a witness σ'' in S to the left of σ. Then, we update the counters like for the previous cases.

The cases $\sigma \to \sigma' : \Box S$ where $\Box S$ is of the form $\forall_R S$ or $\forall_{LR} S$ are similar to case (2), and those where $\Box S$ is of the form $\exists_R S$ or $\exists_{LR} S$ are similar to case (3). Also, we let $\mathrm{pre}_t(\varphi)$ be $\mathrm{post}_{(\sigma' \to \sigma)}(\varphi)$ if $t = (\sigma \to \sigma')$ and $\mathrm{post}_{(\sigma' \to \sigma : \Box S)}(\varphi)$ if $t = (\sigma \to \sigma' : \Box S)$.

Lemma 15 (Post and Pre). *Given a strengthened word φ and a transition t we can compute a set of words $post_t(\varphi)$ (resp. $pre_t(\varphi)$) such that $post_t(\varphi) \trianglelefteq_{\mathbb{NW}} post_t(\varphi')$ (resp. $pre_t(\varphi) \trianglelefteq_{\mathbb{NW}} pre_t(\varphi')$) if $\varphi \trianglelefteq_{\mathbb{NW}} \varphi'$. In addition, $[\![post_t(\varphi)]\!]$ and $[\![pre_t(\varphi)]\!]$ respectively equal $\{c' \mid c \longrightarrow_t c'$ with c in $[\![\varphi]\!]\}$, and $\{c' \mid c' \longrightarrow_t c$ with c in $[\![\varphi]\!]\}$.*

For a finite subset Φ of $\mathbb{NW}$, we write $\mathrm{post}_t(\Phi)$ and $\mathrm{pre}_t(\Phi)$ to mean the unions $\cup_{\varphi \in \Phi} \mathrm{post}_t(\varphi)$ and $\cup_{\varphi \in \Phi} \mathrm{pre}_t(\varphi)$ respectively.

Example 16. $\mathrm{post}_t(\varphi)$ for $t = (a \rightarrow b : \exists_R \{a\})$ and $\varphi = \left(\begin{bmatrix} v_a \geq 0 \\ \wedge v_b = 0 \end{bmatrix}, a, \begin{bmatrix} v_a \geq 0 \\ \wedge v_b = 0 \end{bmatrix}\right)$ is $\left\{\left(\begin{bmatrix} v_a \geq 0 \\ \wedge v_b = 0 \end{bmatrix}, b, \begin{bmatrix} v_a \geq 1 \\ \wedge v_b = 0 \end{bmatrix}\right)\left(\begin{bmatrix} v_a \geq 0 \\ \wedge v_b = 1 \end{bmatrix}, a, \begin{bmatrix} v_a \geq 0 \\ \wedge v_b = 0 \end{bmatrix}\right)\right\}$

7 A Generic Refinement Scheme

Assume a parameterized system $\mathcal{P} = (\Sigma, T)$. We describe in this section a generic scheme for solving the reachability problem of Sect. 6. The scheme we introduce uses over-approximations to deduce unreachability. Each time the approximated analysis exhibits a sequence from the initial to the final configurations (i.e., a trace), we automatically follow the sequence in the original system. If it is possible we return it as a proof of reachability. Otherwise, the trace is a false positive and we automatically strengthen the approximation in order to prune the trace (Fig. 2).

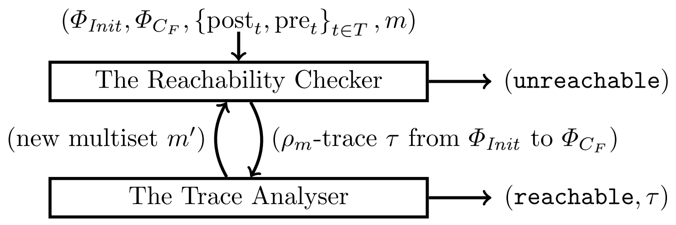

$(\Phi_{Init}, \Phi_{C_F}, \{\mathrm{post}_t, \mathrm{pre}_t\}_{t \in T}, m)$
The Reachability Checker
(`unreachable`)
(new multiset m')
(ρ_m-trace τ from Φ_{Init} to Φ_{C_F})
The Trace Analyser
(`reachable`, τ)

Fig. 2. A Generic scheme.

In the following, a forward analysis is described. For a backward analysis, simply switch Φ_{Init}, Φ_{C_F}, post, pre respectively with Φ_{C_F}, Φ_{Init}, pre, post in both algorithms 1 and 2.

The reachability checking algorithm. Algorithm 1 is a classical working list algorithm. It manipulates pairs (φ, τ) of constraints and *traces*. A trace τ wrt. to an closure ρ_m (or a ρ_m-trace for short) is a sequence $\varphi_0 \cdot t_1 \cdot \varphi_1 \cdots \varphi_n$ of $\mathbb{NW}$ elements $\varphi_0, \ldots \varphi_n$ and of transitions $t_1, \ldots t_n$ in T, such that φ_0 is in Φ_{Init} and $\varphi_{i+1} \in \rho_m(\mathrm{post}_{t_{i+1}}(\varphi_i))$ for each $i : 0 \leq i < n$. Each manipulated pair is of the form $(\varphi_n, \varphi_0 \cdot t_1 \cdots \varphi_n)$. The Algorithm maintains two sets $\mathcal{W}$ (working set) and $\mathcal{V}$ (visited set) such that $(\mathcal{W} \cup \mathcal{V})$ is minimal. The working set $\mathcal{W}$ collects pairs

(φ, τ) for which $\text{post}_t(\varphi)$ has still to be applied for each $t \in T$. The visited set $\mathcal{V}$ collects pairs (φ, τ) where $\text{post}_t(\varphi)$ has already been applied for each transition t in T (i.e., φ has been visited). Initially, $\mathcal{V}$ is empty and all members of Φ_{Init} (assumed minimal) are added to the working set (line 1). If there is a pair (φ_c, τ) in the working set, it is first removed from $\mathcal{W}$ (line 3). If its denotation intersects C_F, then we found a trace in the over-approximated system from the initial to the final configurations. In this case, the Algorithm returns τ as a proof of reachability (lines 4 and 5). Otherwise, the pair is added to the visited set (line 6) and $\text{post}_t(\varphi_c)$ is computed for each t in T. Each element in $\text{post}_t(\varphi_c)$ is relaxed (line

Algorithm 1: The reachability checker

```
input : minimal $\Phi_{Init}$ and $\Phi_{C_F}$, operators $\text{post}_t$ and $\text{pre}_t$ for each $t \in T$,
        and a multiset $m$
output: a trace $(\varphi_0 \cdot t_1 \cdot \varphi_2 \cdot t_2 \cdots \varphi_n)$ with $(\{\varphi_n\} \sqcap_{\mathbb{NW}} \Phi_{C_F}) \neq \emptyset$, or
        unreachable
1  $\mathcal{W} := \{(\varphi, \varphi) \mid \varphi \text{ in } \Phi_{Init}\}$, $\mathcal{V} := \emptyset$;
2  while $(\mathcal{W} \neq \emptyset)$ do
3      Pick and remove a pair $(\varphi_c, \tau)$ from $\mathcal{W}$;
4      if $(\{\varphi_c\} \sqcap_{\mathbb{NW}} \Phi_{C_F} \neq \emptyset)$ then
5          return $\tau$;
6      $\mathcal{V} := \{(\varphi_c, \tau)\} \cup \mathcal{V}$ ;
7      foreach $t \in T$ do
8          $New_t := \{\rho_m(\varphi) \mid \varphi \in \text{post}_t(\varphi_c)\}$;
9          foreach $\varphi \in New_t$ do
10             if for each $(\varphi_{old}, \tau_{old}) \in \mathcal{W} \cup \mathcal{V}.\ \varphi_{old} \not\trianglelefteq_{\mathbb{NW}} \varphi$ then
11                 $\mathcal{V} := \{(\varphi_{old}, \tau_{old}) \mid (\varphi_{old}, \tau_{old}) \in \mathcal{V} \wedge \varphi \not\trianglelefteq_{\mathbb{NW}} \varphi_{old}\}$;
12                 $\mathcal{W} := \{(\varphi_{old}, \tau_{old}) \mid (\varphi_{old}, \tau_{old}) \in \mathcal{W} \wedge \varphi \not\trianglelefteq_{\mathbb{NW}} \varphi_{old}\}$;
13                 $\mathcal{W} := \mathcal{W} \cup \{(\varphi, \tau \cdot t \cdot \varphi)\}$
14 return unreachable;
```

8) before being added to the set New_t. This closure is at the source of imprecision and will guarantee termination. Elements of New_t are pruned away if they do not add new configurations. Otherwise, they are used to remove redundant elements of $\mathcal{V} \cup \mathcal{W}$ before being added to $\mathcal{W}$ together with their updated traces (lines 11 and 12).

Lemma 17 (reachability). *Algorithm 1 always terminates. In case it returns* `unreachable`*, then* $(Init \xrightarrow{*} C_F)$ *does not hold for the parameterized system* $\mathcal{P} = (\Sigma, T)$*. Otherwise, it returns a trace* $\varphi_0 \cdot t_1 \cdot \varphi_1 \cdots \varphi_n$ *with* $\varphi_0 \in \Phi_{Init}$*,* $\{\varphi_n\} \sqcap_{\mathbb{NW}} \Phi_{C_F} \neq \emptyset$*, and* $\varphi_{i+1} \in \rho_m(post_{t_{i+1}}(\varphi_i))$ *for each* $i : 0 \leq i < n$*.*

Proof. Let $\mathcal{W}_k, \mathcal{V}_k$ be the sets $\mathcal{W}, \mathcal{V}$ obtained at (line 2) at the k^{th} iteration of the loop. We show the following four propositions by induction on k:

a) for each (φ, τ) in $\mathcal{V}_k \cup \mathcal{W}_k$, τ equals $\varphi_0 \cdot t_1 \cdots \varphi_n$ for some $n \leq k$, with $\varphi_0 \in \Phi_{Init}$, and $\varphi_{i+1} \in \rho_m(\text{post}_{t_{i+1}}(\varphi_i))$ for each $i : 0 \leq i < n$
b) *Init* is subset of $[\![\mathcal{V}_k]\!] \cup [\![\mathcal{W}_k]\!]$, and $\{c' | \; c \longrightarrow_t c'$ for some $c \in [\![\mathcal{V}_k]\!], t \in T\}$ (i.e., the successors of $[\![\mathcal{V}_k]\!]$) are in $[\![\mathcal{W}_k]\!] \cup [\![\mathcal{V}_k]\!]$.
c) for each φ in $\mathcal{V}$, $\{\varphi\} \sqcap_{\mathbb{N}\mathbb{W}} \Phi_{C_F}$ is empty
d) the set $\{\varphi | \; (\varphi, \tau) \in \mathcal{V}_k \cup \mathcal{W}_k\}$ is minimal wrt $\trianglelefteq_{\mathbb{N}\mathbb{W}}$, i.e., for any φ, φ', neither $\varphi \trianglelefteq_{\mathbb{N}\mathbb{W}} \varphi'$ nor $\varphi' \trianglelefteq_{\mathbb{N}\mathbb{W}} \varphi$ holds.

Base case: $k = 0$. $\mathcal{V}$ and $\mathcal{W}$ are initially respectively empty and minimal.
Suppose the propositions hold up to k. We show they hold for $k + 1$. Elements added to $\mathcal{V}$ need to fail the test at line 4. So proposition (c) holds. The test at line 10 and the conditions at lines 11 and 12 guarantee proposition (d). The form of the added tuple at line 13 ensures proposition (a).
For proposition (b), we show that i) $[\![\mathcal{V}_k \cup \mathcal{W}_k]\!] \subseteq [\![\mathcal{V}_{k+1} \cup \mathcal{W}_{k+1}]\!]$, and that ii) the successors of $[\![\varphi]\!]$, for any $\varphi \in \mathcal{V}_k \cup \mathcal{W}_k$, are also in $[\![\mathcal{V}_{k+1} \cup \mathcal{W}_{k+1}]\!]$. The only modifications to $\mathcal{V}_k \cup \mathcal{W}_k$ occur at lines 11,12 and 13. If φ' is removed at lines 11 and 12 from $\mathcal{V} \cup \mathcal{W}$, then the conditions at lines 11 and 12 ensure that $\varphi \trianglelefteq_{\mathbb{N}\mathbb{W}} \varphi'$ for some $\varphi \in New_t$. The element φ is added to $\mathcal{V} \cup \mathcal{W}$, hence $[\![\mathcal{V}_k \cup \mathcal{W}_k]\!] \subseteq [\![\mathcal{V}_{k+1} \cup \mathcal{W}_{k+1}]\!]$. For condition (ii), observe that lemmas 11 and 15 ensure that if $c \in [\![\varphi]\!]$ and $c \longrightarrow_t c'$, then there is $\varphi' \in New_t$ with $c' \in [\![\varphi']\!]$. The conditions at lines 10, 11 and 12 ensure that φ' is added to $\mathcal{V}$ unless it is entailed by an element in $\mathcal{V} \cup \mathcal{W}$.

Partial correctness. Suppose the Algorithm returns `unreachable`. Then $\mathcal{W}$ was empty at some iteration. Combined with proposition (b), we get that $[\![\mathcal{V}]\!]$ is a fixpoint that includes all reachable configurations. Proposition (c) ensures that $[\![\mathcal{V}]\!] \cap C_F$ is empty. If the algorithm returns $(\texttt{reach}, \tau)$, then the test at line 4 ensures that $\{\varphi_c\} \sqcap_{\mathbb{N}\mathbb{W}} \Phi_{C_F}$ is non empty. Moreover, line 3 together with proposition (a) ensure that $\tau = \varphi_0 \cdot t_1 \cdot \varphi_1 \cdots \varphi_n$ satisfies $\varphi_0 \in \Phi_{Init}$, $\varphi_n = \varphi_c$, and $\varphi_{i+1} \in \rho_m(\text{post}_{t_{i+1}}(\varphi_i))$ for each $i : 0 \leq i < n$.

Termination. Suppose the algorithm does not terminate. That means we add an infinite number of elements to $\mathcal{V}$. Consider a sequence $(\varphi_1, \varphi_2, \ldots)$ where each φ_k is some element added at iteration k. Proposition (d) and transitivity guarantee that in this sequence, $i < j$ implies that $\varphi_i \not\trianglelefteq_{\mathbb{N}\mathbb{W}} \varphi_j$. Indeed, if $\varphi_i \trianglelefteq_{\mathbb{N}\mathbb{W}} \varphi_j$ for some $i < j$, then $\varphi_i \notin \mathcal{V}_j$ when φ_j was added at line 11 to $\mathcal{V}_j$. This means that φ_i was removed by some element φ_k added to $\mathcal{V}_k$ at a later iteration $k : i < k < j$ such that $\varphi_k \trianglelefteq_{\mathbb{N}\mathbb{W}} \varphi_i$. By repeating this reasoning, and transitivity of $\trianglelefteq_{\mathbb{N}\mathbb{W}}$, we deduce that $\mathcal{V}_{j-1}$ had an element φ_{j-1} such that $\varphi_{j-1} \trianglelefteq_{\mathbb{N}\mathbb{W}} \varphi_j$. Thus, φ_j should not have passed the test at line 10, and hence not been added to $\mathcal{V}_j$. The existence of such an infinite sequence contradicts lemma 11 and theorem 10 since, according to the computation at line 8, all elements in $\mathcal{V}$ are of the form $\rho_m(\varphi)$ for some element φ.

Algorithm 2: The trace analyzer

```
   input : $\rho_m$-trace $(\varphi_0 \cdot t_1 \cdot \varphi_1 \cdots \varphi_n)$ with $\{\varphi_n\} \sqcap_{\mathbb{NW}} \Phi_{C_F} \neq \emptyset$
   output: reachable or a new multiset $m'$
1  $Curr := \{\varphi_n\} \sqcap_{\mathbb{NW}} \Phi_{C_F}$;
2  foreach $i = m - 1$ to $0$ do
3      $Pred := (\bigcup_{\varphi \in Curr} \mathrm{pre}_{t_{i+1}}(\varphi)) \sqcap_{\mathbb{NW}} \{\varphi_i\}$;
4      if $(Pred = \emptyset)$ then
5          return smallest $m'$ satisfying formulaseparate$(\mathrm{post}_{t_i}(\varphi_i), Curr, m)$
6      else $Curr := Pred$;
7  $Inter := Curr \sqcap_{\mathbb{NW}} \Phi_{Init}$;
8  if $Inter \neq \emptyset$ then
9      return reachable, $\tau$
10 else
11     return smallest $m'$ satisfying formula separate$(\{\varphi_0\}, \Phi_{Init}, \rho_m)$
```

The trace analyzer. Algorithm 2 simulates a trace backwards in order to check its possibility in the original system. Only the approximation resulting from the applications of the ρ_m closure can result in the analyzer failing to follow the supplied ρ_m-trace. If this happens (lines 4 or 10), the analyzer relies on the separation operator (lines 5 and 11) to supply a new multiset that results in a stronger closure operator that will prune the trace in future analysis. Otherwise, the analyzer will manage to reach the initial configurations at line 8. In this case, it returns the trace as a proof of reachability line 9.

Lemma 18 (Refinement). *Given a ρ_m-trace $\tau = \varphi_0 \cdot t_1 \cdots \varphi_n$ with $\{\varphi_n\} \sqcap_{\mathbb{NW}} \Phi_{C_F} \neq \emptyset$, Algorithm 2 terminates. If it returns $(\texttt{reachable}, \tau)$, then there are $c_0, \ldots c_n$ in C, with $c_0 \in Init$, $c_n \in C_F$ and s.t. $c_i \longrightarrow_{t_{i+1}} c_{i+1}$ for $i : 0 \leq i < n$. Otherwise, it returns a new multiset m' such no larger or equal multiset m'' can result in a $\rho_{m''}$-trace $\varphi_0' \cdot t_1 \cdots \varphi_n'$ with $\{\varphi_n'\} \sqcap_{\mathbb{NW}} \Phi_{C_F} \neq \emptyset$ and where $\varphi_i \trianglelefteq_{\mathbb{NW}} \varphi_i'$ for $i : 0 \leq i \leq n$.*

Proof. Termination is guaranteed by the fact that the number of iterations of the loop at line 2 is bounded by the finite size of the trace τ, and that the other operations are effective. The algorithm returns $(\texttt{reachable}, \tau)$ if it succeeds in building a sequence $(Curr_n \cdot t_{n-1} \cdots Curr_0)$ with $Curr_n = \{\varphi_n\} \sqcap_{\mathbb{NW}} \Phi_{C_F}$, and $Curr_i = \{\varphi_i\} \sqcap_{\mathbb{NW}} (\bigcup_{\varphi \in Curr_{i+1}} \mathrm{pre}_{t_{i+1}}(\varphi))$ for each $i : 0 \leq i < n$, and $Curr_0 \sqcap_{\mathbb{NW}} \Phi_{Init} \neq \emptyset$. Lemmas 13 and 15 on $\mathrm{pre}_t(.)$ and on $\sqcap_{\mathbb{NW}}$ guarantee the existence of configurations $c_0, \ldots c_n$ with c_0 in $[\![Curr_0]\!]$, and s.t. $c_i \longrightarrow_{t_{i+1}} c_{i+1}$ and c_{i+1} in $[\![Curr_{i+1}]\!]$ for each $i : 0 \leq i < n$. Suppose the Algorithm returns instead a multiset m' such that there exists a $\rho_{m''}$-trace $(\varphi_0' \cdot t_1 \cdots \varphi_n')$, for some m'' larger or equal than m', with $\varphi_0 \trianglelefteq_{\mathbb{NW}} \varphi_0'$, $\{\varphi_n'\} \sqcap_{\mathbb{NW}} \Phi_{C_F} \neq \emptyset$, and $\varphi_{i+1}' \in \rho_{m''}(\mathrm{post}_{t_{i+1}}(\varphi_i'))$ with $\varphi_{i+1} \trianglelefteq_{\mathbb{NW}} \varphi_{i+1}'$ for each $i : 0 \leq i < n$. This would happen at line 5 or at line 11.

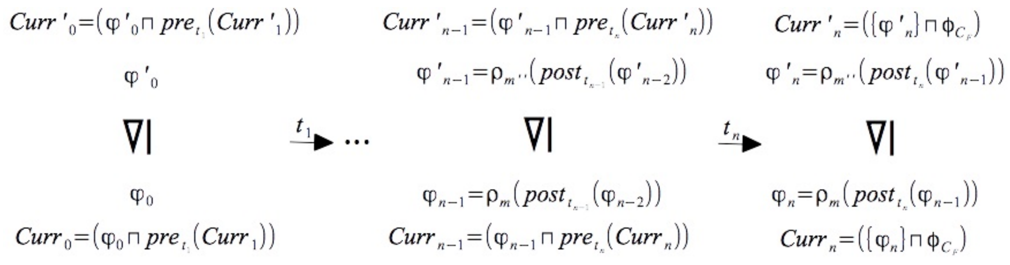

$Curr'_0 = (\varphi'_0 \sqcap pre_{t_1}(Curr'_1))$
φ'_0
$\trianglerighteq$
φ_0
$Curr_0 = (\varphi_0 \sqcap pre_{t_1}(Curr_1))$
t_1
$\cdots$
$Curr'_{n-1} = (\varphi'_{n-1} \sqcap pre_{t_n}(Curr'_n))$
$\varphi'_{n-1} = \rho_{m''}(post_{t_{n-1}}(\varphi'_{n-2}))$
$\trianglerighteq$
$\varphi_{n-1} = \rho_m(post_{t_{n-1}}(\varphi_{n-2}))$
$Curr_{n-1} = (\varphi_{n-1} \sqcap pre_{t_n}(Curr_n))$
t_n
$Curr'_n = (\{\varphi'_n\} \sqcap \phi_{C_F})$
$\varphi'_n = \rho_{m''}(post_{t_n}(\varphi'_{n-1}))$
$\trianglerighteq$
$\varphi_n = \rho_m(post_{t_n}(\varphi_{n-1}))$
$Curr_n = (\{\varphi_n\} \sqcap \phi_{C_F})$

Fig. 3. Suppose $\varphi'_0 \cdot t_1 \cdots \varphi'_n$ obtained with $\rho_{m'}$ after m' was deduced from the ρ_m trace $\varphi_0 \cdot t_1 \cdots \varphi_n$.

If at line 5, then let $Curr_n, Curr_{n-1}, \ldots$ be the sequence of $Curr$ sets manipulated at each iteration. In a similar manner, build the sequence $Curr'_n, Curr'_{n-1}, \ldots$ that we would obtain from the $\rho_{m''}$-trace $(\varphi'_0 {\cdot} t_1 \cdots \varphi'_n)$. Transitivity of $\trianglelefteq_{\mathbb{NW}}$ (lemmas 9) and lemma 15 guarantee that $Curr_j \trianglelefteq_{\mathbb{NW}} Curr'_j$ for each $j : 0 \leq j \leq n$. The fact that Algorithm 2 returned at line 5 says that there is a $i : 0 \leq i < n$ s.t. $\rho_{m'} = \text{separate}(\text{post}_{t_i}(\varphi_i), Curr_{i+1}, \rho_m)$ with $(\rho_{m'}(\text{post}_{t_i}(\varphi_i)) \sqcap_{\mathbb{NW}} Curr_{i+1} = \emptyset)$. Since $\varphi_i \trianglelefteq_{\mathbb{NW}} \varphi'_i$ we deduce (lemma 15) that $\text{post}_{t_i}(\varphi_i) \trianglelefteq_{\mathbb{NW}} \text{post}_{t_i}(\varphi'_i)$, and hence (lemma 11) that $\rho_{m''}(\text{post}_{t_i}(\varphi_i)) \trianglelefteq_{\mathbb{NW}} \rho_{m''}(\text{post}_{t_i}(\varphi'_i))$. Since $Curr_{i+1} \trianglelefteq_{\mathbb{NW}} Curr'_{i+1}$, we deduce $(\rho_{m''}(\text{post}_{t_i}(\varphi'_i)) \sqcap_{\mathbb{NW}} Curr'_{i+1} = \emptyset)$ as otherwise $\rho_{m'}(\text{post}_{t_i}(\varphi_i)) \sqcap_{\mathbb{NW}} Curr_{i+1} \neq \emptyset$. If at line 11, then $\rho_{m'} = \text{separate}(\{\varphi_0\}, \varPhi_{Init}, \rho_m)$, with $\rho_{m'}(\varphi_0) \sqcap_{\mathbb{NW}} \varPhi_{Init} = \emptyset$. Since $\varphi_0 \trianglelefteq_{\mathbb{NW}} \varphi'_0$, we deduce (lemma 11) $\rho_{m'}(\varphi_0) \trianglelefteq_{\mathbb{NW}} \rho_{m''}(\varphi'_0)$ and $\{\rho_{m''}(\varphi'_n)\} \sqcap_{\mathbb{NW}} \varPhi_{Init} = \emptyset$. Hence τ' would have $\{\varphi'_n\} \sqcap_{\mathbb{NW}} \varPhi_{C_F} = \emptyset$, which contradicts its definition.

By combining Lemmas (17) and (18), we get correctness of the Algorithm depicted in Figure (2).

Theorem 19. *Each iteration of the Algorithm depicted in Figure (2) terminates. If it returns* `unreachable`*, then Init* $\xrightarrow{*} C_F$ *does not hold. If it returns* $(\texttt{reachable}, \tau)$*, then Init* $\xrightarrow{*} C_F$ *via the transitions in* τ*. In addition, no trace is generated twice.*

8 Experimental Results

We have implemented the scheme of Fig.2 in OCaml[2] and run experiments on an Intel Core i7 2.93 GHz computer with 8GB of memory. We have considered four classical mutual exclusion algorithms, namely Burns [2], compact [5] and refined [16] versions of Szymanski's algorithm, and the related Gribomont-Zenner mutex [12]. In the following, we describe the algorithms and report on the obtained results for each one of them.

[2] Prototype "Pcw" available at http://www.ida.liu.se/ ahmre43

We did experiment with non-approximated simple accelerations. While these did boost performance, we do not report on these results as we follow the scheme of Sect. 7. One noticeable optimization was to select at line 3 in Alg.1, a pair (φ_c, τ) with shortest counted word φ_c. Combining with more systematic accelerations instead of taking one step at a time or adopting lazy strengthenings of closures applications in order to avoid trashing the explored state space can be the subject of natural extensions for this work.

Variables:
locations in $\{l_1, \dots, l_7\}$
f in $\{false, true\}$

Transitions:
$l_1[\mathtt{i}] \to l_2[\mathtt{i}] : f[\mathtt{i}] := false$

$l_2[\mathtt{i}] \to l_1[\mathtt{i}] : \exists_{\mathtt{j}<\mathtt{i}}.f[\mathtt{j}]$
$l_2[\mathtt{i}] \to l_3[\mathtt{i}] : \forall_{\mathtt{j}<\mathtt{i}}.\neg f[\mathtt{j}]$
$l_3[\mathtt{i}] \to l_4[\mathtt{i}] : f[\mathtt{i}] := true$
$l_4[\mathtt{i}] \to l_1[\mathtt{i}] : \exists_{\mathtt{j}<\mathtt{i}}.f[\mathtt{j}]$
$l_4[\mathtt{i}] \to l_5[\mathtt{i}] : \forall_{\mathtt{j}<\mathtt{i}}.\neg f[\mathtt{j}]$
$l_5[\mathtt{i}] \to l_6[\mathtt{i}] : \forall_{\mathtt{j}>\mathtt{i}}.\neg f[\mathtt{j}]$
$l_6[\mathtt{i}] \to l_7[\mathtt{i}] : f[\mathtt{i}] := false$
$l_7[\mathtt{i}] \to l_1[\mathtt{i}] : true$

$Init : \forall_{\mathtt{i}}(@l_1[\mathtt{i}] \wedge \neg f[\mathtt{i}])$
$C_F : \exists_{j \neq j'}(@l_6[j] \wedge @l_6[j'])$

States:
$\Sigma = \{\sigma_{1:0}, \sigma_{1:1}, \sigma_{2:0}, \sigma_{3:0}, \sigma_{4:1}, \sigma_{5:1}, \sigma_{6:1}, \sigma_{7:0}\}$

$Transitions$:
$\sigma_{1:0} \to \sigma_{2:0}$
$\sigma_{1:1} \to \sigma_{2:0}$
$\sigma_{2:0} \to \sigma_{1:0} : \exists_L \{\sigma_{1:1}, \sigma_{4:1}, \sigma_{5:1}, \sigma_{6:1}\}$
$\sigma_{2:0} \to \sigma_{3:0} : \forall_L \{\sigma_{1:0}, \sigma_{2:0}, \sigma_{3:0}, \sigma_{7:0}\}$
$\sigma_{3:0} \to \sigma_{4:1}$
$\sigma_{4:1} \to \sigma_{1:1} : \exists_L \{\sigma_{1:1}, \sigma_{4:1}, \sigma_{5:1}, \sigma_{6:1}\}$
$\sigma_{4:1} \to \sigma_{5:1} : \forall_L \{\sigma_{1:0}, \sigma_{2:0}, \sigma_{3:0}, \sigma_{7:0}\}$
$\sigma_{5:1} \to \sigma_{6:1} : \forall_R \{\sigma_{1:0}, \sigma_{2:0}, \sigma_{3:0}, \sigma_{7:0}\}$
$\sigma_{6:1} \to \sigma_{7:0}$
$\sigma_{7:0} \to \sigma_{1:0}$

$\Phi_{Init} = \{(cr_1, \sigma_{1:0}, cr_1)\}$
$\Phi_{C_F} = \{(cr_2, \sigma_{6:1}, cr_3) \cdot (cr_3, \sigma_{6:1}, cr_2)\}$

Fig. 4. Burns algorithm, with counters $cr_1 = \left[\bigwedge_{q \in Q \setminus \{\sigma_{1:0}\}} (v_q = 0) \wedge v_{\sigma_{1:0}} \geq 0\right]$, $cr_2 = \top_{\mathbb{C}}$ and $cr_3 = \bigwedge_{q \in Q \setminus \{\sigma_{1:0}\}} (v_q \geq 0) \wedge v_{\sigma_{1:0}} \geq 1$

Burns mutual exclusion algorithm. In this algorithm (left of Fig.4), each process has a local boolean flag `f` in addition to seven locations $\{l_1, \dots l_7\}$. A simple translation results in a parameterized system (right of Fig.4) with eight local states $\{\sigma_{1:0}, \sigma_{1:1}, \sigma_{2:0}, \sigma_{3:0}, \sigma_{4:1}, \sigma_{5:1}, \sigma_{6:1}, \sigma_{7:0}\}$ where each $\sigma_{s:b}$ corresponds to a process being at location σ_s with flag b. Each process interested in accessing its critical section (location l_6) checks twice to its left if there are other interested processes (i.e., with a flag set to 1). If there are, it returns to $\sigma_{(1:\_)}$. Otherwise, it waits untill all processes to its right are not competing to access the critical section. Mutual exclusion is violated in case more than one process is at state $\sigma_{6:1}$.

Table 3. $\mathbb{NW}$ based analysis of the Burns parameterized mutex.

Mode	m in ρ_m	$\mathrm{pre}_T, \mathrm{post}_T$ applications	Words	Time	Safe
Backward	$\forall \sigma \in \Sigma. m_0(\sigma) = 0$	3	145	$< 0.01 sec$	$\surd$
Forwad	$\forall \sigma \in \Sigma. m_0(\sigma) = 0$	0	1	$< 0.01 sec$	$\times$
	$m_1 = m_0[\sigma_{6:1} \leftarrow 1]$	6	201	$< 0.01 sec$	$\times$
	$m_2 = m_1[\sigma_{5:1} \leftarrow 1]$	14	525	$0.01 sec$	$\surd$

We describe in Tab.3 the results we obtained with our prototype implementation. We observe that the backwards analysis resulted in less refinements of the closure operator ρ_m when compared to the forward analysis. This observation is confirmed by all our experiments. In forward, the counted word capturing the initial configurations already intersects the bad configurations when relaxed with ρ_{m_0} where $\forall \sigma \in \Sigma . m_0(\sigma) = 0$. For this reason, the first refinement consists in excluding during closure processes at their critical section if they were not present in the original counted word.

Variables:
locations in $\{l_1, \ldots, l_9\}$
w, s in $\{false, true\}$

Transitions:
$l_1[\mathtt{i}] \rightarrow l_2[\mathtt{i}]$: non critical
$l_2[\mathtt{i}] \rightarrow l_3[\mathtt{i}] : \forall_{\mathtt{j} \neq \mathtt{i}} . (\neg s[\mathtt{j}])$
$l_3[\mathtt{i}] \rightarrow l_4[\mathtt{i}] : w[\mathtt{i}], s[\mathtt{i}] := true, true$
$l_4[\mathtt{i}] \rightarrow l_5[\mathtt{i}] : \exists_{\mathtt{j} \neq \mathtt{i}} . (@l_2[\mathtt{j}] \vee @l_3[\mathtt{j}])$
$\wedge s[\mathtt{i}] := false$
$l_4[\mathtt{i}] \rightarrow l_6[\mathtt{i}] : \forall_{\mathtt{j} \neq \mathtt{i}} . (\neg @l_2[\mathtt{j}] \wedge \neg @l_3[\mathtt{j}])$
$\wedge w[\mathtt{i}] := false$
$l_5[\mathtt{i}] \rightarrow l_6[\mathtt{i}] : \exists_{\mathtt{j} \neq \mathtt{i}} . (s[\mathtt{j}] \wedge \neg w[\mathtt{j}])$
$\wedge w[\mathtt{i}], s[\mathtt{i}] := false, true$
$l_6[\mathtt{i}] \rightarrow l_7[\mathtt{i}] : \forall_{\mathtt{j} \neq \mathtt{i}} . (\neg w[\mathtt{j}])$

$l_7[\mathtt{i}] \rightarrow l_8[\mathtt{i}] : \forall_{\mathtt{j} < \mathtt{i}} . (\neg s[\mathtt{j}] \wedge \neg w[\mathtt{j}])$
$l_8[\mathtt{i}] \rightarrow l_9[\mathtt{i}]$: critical
$l_9[\mathtt{i}] \rightarrow l_1[\mathtt{i}] : s[\mathtt{i}] := false$

$Init : \forall_i . (@l_1[i] \wedge \neg w[i] \wedge \neg s[i])$
$C_F : \exists_{j \neq j'} . (@l_8[j] \wedge @l_8[j'])$

States:
$\Sigma = \{\sigma_{1:0:0}, \sigma_{2:0:0}, \sigma_{3:0:0}\} \cup \{\sigma_{4:1:1}\} \cup \{\sigma_{5:1:0}\}$
$\cup \{\sigma_{6:0:1}, \sigma_{7:0:1}, \sigma_{8:0:1}, \sigma_{9:0:1}\}$

Transitions:
$\sigma_{1:0:0} \rightarrow \sigma_{2:0:0}$
$\sigma_{2:0:0} \rightarrow \sigma_{3:0:0} : \forall_{LR} \{\sigma_{1:0:0}, \sigma_{2:0:0}, \sigma_{3:0:0}\} \cup \{\sigma_{5:1:0}\}$
$\sigma_{3:0:0} \rightarrow \sigma_{4:1:1}$
$\sigma_{4:1:1} \rightarrow \sigma_{5:1:0} : \exists_{LR} \{\sigma_{2:0:0}, \sigma_{3:0:0}\}$

$\sigma_{4:1:1} \rightarrow \sigma_{6:0:1} : \forall_{LR} \{\sigma_{1:0:0}\} \cup \{\sigma_{4:1:1}\} \cup \{\sigma_{5:1:0}\}$
$\cup \{\sigma_{6:0:1}, \sigma_{7:0:1}, \sigma_{8:0:1}, \sigma_{9:0:1}\}$
$\sigma_{5:1:0} \rightarrow \sigma_{6:0:1} : \exists_{LR} \{\sigma_{6:0:1}, \sigma_{7:0:1}, \sigma_{8:0:1}, \sigma_{9:0:1}\}$

$\sigma_{6:0:1} \rightarrow \sigma_{7:0:1} : \forall_{LR} \{\sigma_{1:0:0}, \sigma_{2:0:0}, \sigma_{3:0:0}\}$
$\cup \{\sigma_{6:0:1}, \sigma_{7:0:1}, \sigma_{8:0:1}, \sigma_{9:0:1}\}$
$\sigma_{7:0:1} \rightarrow \sigma_{8:0:1} : \forall_{L} \{\sigma_{1:0:0}, \sigma_{2:0:0}, \sigma_{3:0:0}\}$
$\sigma_{8:0:1} \rightarrow \sigma_{9:0:1}$
$\sigma_{9:0:1} \rightarrow \sigma_{1:0:0}$

$\Phi_{Init} = \{(cr_1, \sigma_{1:0:0}, cr_1)\}$
$\Phi_{C_F} = \{(cr_2, \sigma_{8:0:1}, cr_3) \cdot (cr_3, \sigma_{8:0:1}, cr_2)\}$

Fig. 5. Compact version of Szymanski's algorithm [5], with counters $cr_1 = [(v_{\sigma_{1:0:0}} \geq 0) \wedge \wedge_{q \in Q \setminus \{\sigma_{1:0:0}\}} (v_q = 0)]$, $cr_2 = [\wedge_{q \in Q} (v_q \geq 0)]$, and $cr_3 = [(v_{\sigma_{8:0:1}} \geq 1) \wedge \wedge_{q \in Q \setminus \{\sigma_{8:0:1}\}} (v_q \geq 0)]$.

Compact version of Szymanski's Mutual Exclusion Algorithm. This version [5] is represented to the left of Fig.5. Each process has nine locations and two local boolean variables s and w. We use $@l_i[j]$ to mean the predicate evaluating to true if process j is at location l_i. This system can be easily flattened into the parameterized system to the right of Fig.5 with the states $\Sigma = \{\sigma_{1:0:0}, \sigma_{2:0:0}, \sigma_{3:0:0}\} \cup \{\sigma_{4:1:1}\} \cup \{\sigma_{5:1:0}\} \cup \{\sigma_{6:0:1}, \sigma_{7:0:1}, \sigma_{8:0:1}, \sigma_{9:0:1}\}$. Each state $\sigma_{s:b_1:b_2}$ corresponds to the state of a process at location l_s with variable w equal to b_1 and variable b_2 equal to s. Processes that reach location l_2 are guaranteed to eventually access their critical section at l_8. Processes move from location l_4 to location l_5 to wait for those processes that are still at locations l_2 or l_3. If there are no processes at locations l_2, l_3, then processes at l_4 move directly to location l_6. Once a process is at location l_6, those waiting processes at l_5 move to l_6 and no other process can move from l_2 to l_3 (the door is closed for this round). After gathering at l_6, the processes access the critical section, one at a time, with priority to the left most among processes at l_6, l_7, l_8 and l_9.

We describe in Tab.4 the results we obtained for the Compact version of the Szymanski mutex. The backward analysis did not involve any refinement steps, while the forward required seven refinements before it could conclude the parameterized system was safe.

Table 4. $\mathbb{NW}$ based analysis of the Compact Szymanski [5].

Mode	m in ρ_m	$\mathrm{pre}_T, \mathrm{post}_T$ applications	Words	Time	Safe
Backward	$\forall\sigma \in \Sigma. m_0(\sigma) = 0$	26	3581	$0.25 sec$	$\surd$
Forwad	$\forall\sigma \in \Sigma. m_0(\sigma) = 0$	0	1	$< 0.01 sec$	$\times$
	$m_1 = m_0[\sigma_{8:0:1} \leftarrow 1]$	7	311	$< 0.01 sec$	$\times$
	$m_2 = m_1[\sigma_{7:0:1} \leftarrow 1]$	9	372	$0.01 sec$	$\times$
	$m_3 = m_2[\sigma_{6:0:1} \leftarrow 1]$	10	350	$0.02 sec$	$\times$
	$m_4 = m_3[\sigma_{4:1:1} \leftarrow 1]$	10	185	$0.02 sec$	$\times$
	$m_5 = m_4[\sigma_{5:1:0} \leftarrow 1]$	11	112	$< 0.01 sec$	$\times$
	$m_6 = m_5[\sigma_{3:0:0} \leftarrow 1]$	245	3919	$1.91 sec$	$\times$
	$m_7 = m_6[\sigma_{9:0:1} \leftarrow 1]$	186	4806	$0.82 sec$	$\surd$

A refined version of Szymanski's Algorithm [16]. In this version, each process has thirteen locations and a local variable f that ranges over $\{0, 1, 2, 3, 4\}$. We flattened the local variable by encoding its value in process states. We use $\sigma_{s:k}$ to mean the state of a process at location l_s with local variable equal to k. Initially, all processes are at location is l_1 with $f = 0$. The critical section is at location l_{11}.

We describe in Tab.5 the results we obtained for the refined version of the Szymanski's mutex. The backward analysis involved three refinement steps, while the forward required nine refinements and several days before it could conclude the parameterized system was safe. We believe accelerations and further optimizations (in the way the minimal sets are organized) can drastically improve this delay.

Griboment-Zenner Mutual Exclusion Algorithm. This algorithm [12] is also derived from Szymanski's algorithm. Its transitions have a finer granularity when compared to the compact version [5], in the sense that tests and assignments are split in different atomic transitions. After flattening the resulting states of the processes range over $\{\sigma_{1:0:0:0}, \sigma_{2:0:0:0}\} \cup \{\sigma_{3:1:0:0}, \sigma_{4:1:0:0}\} \cup \{\sigma_{5:1:1:1}, \sigma_{6:1:1:1}, \sigma_{9:1:1:1}\} \cup \{\sigma_{7:1:1:0}, \sigma_{8:1:1:0}\} \cup \{\sigma_{10:1:0:1}, \sigma_{11:1:0:1}, \sigma_{12:1:0:1}, \sigma_{13:1:0:1}\}$. We write $\sigma_{s:b_1:b_2:b_3}$ to mean the state of a process at location l_s, with varibles a, w, s respectively evaluating to b_1, b_2, b_3.

We describe in Tab.6 the results we obtained for the Gribomont-Zenner mutex. The backward analysis did not involve any refinement steps, while the forward required seven refinements before it could conclude the parameterized system was safe.

Variables:
locations in $\{l_1, \dots l_{13}\}$
f in $\{0, 1, 2, 3, 4\}$

Transitions:
$l_1[\mathtt{i}] \rightarrow l_2[\mathtt{i}]$: non critical
$l_2[\mathtt{i}] \rightarrow l_3[\mathtt{i}]$: $f[\mathtt{i}] := 1$
$l_3[\mathtt{i}] \rightarrow l_4[\mathtt{i}]$: $\forall_{\mathtt{j}\neq\mathtt{i}}.f[\mathtt{j}] < 3$

$l_4[\mathtt{i}] \rightarrow l_5[\mathtt{i}]$: $f[\mathtt{i}] := 3$
$l_5[\mathtt{i}] \rightarrow l_6[\mathtt{i}]$: $\exists_{\mathtt{j}\neq\mathtt{i}}.f[j] = 1$
$l_5[\mathtt{i}] \rightarrow l_9[\mathtt{i}]$: $\forall_{\mathtt{j}\neq\mathtt{i}}.f[\mathtt{j}] \neq 1$

$l_6[\mathtt{i}] \rightarrow l_7[\mathtt{i}]$: $f[\mathtt{i}] := 2$
$l_7[\mathtt{i}] \rightarrow l_8[\mathtt{i}]$: $\exists_{\mathtt{j}\neq\mathtt{i}}.f[\mathtt{j}] = 4$
$l_8[\mathtt{i}] \rightarrow l_{10}[\mathtt{i}]$: $f[\mathtt{i}] := 4$
$l_9[\mathtt{i}] \rightarrow l_{10}[\mathtt{i}]$: $f[\mathtt{i}] := 4$
$l_{10}[\mathtt{i}] \rightarrow l_{11}[\mathtt{i}]$: $\forall_{\mathtt{j}<\mathtt{i}}.f[\mathtt{j}] < 2$
$l_{11}[\mathtt{i}] \rightarrow l_{12}[\mathtt{i}]$: *critical section*
$l_{12}[\mathtt{i}] \rightarrow l_{13}[\mathtt{i}]$: $\forall_{\mathtt{j}>\mathtt{i}}.f[\mathtt{j}] < 2 \vee f[\mathtt{j}] > 3$

$l_{13}[\mathtt{i}] \rightarrow l_1[\mathtt{i}]$: $f[\mathtt{i}] := 0$

$\mathit{Init} = \forall_{\mathtt{i}}.(@l_1[\mathtt{i}] \wedge f[\mathtt{i}] = 0)$
$C_F = \exists_{\mathtt{i}\neq\mathtt{j}}.(@l_{11}[\mathtt{i}] \wedge @l_{11}[\mathtt{j}])$

States:
$Q = \{\sigma_{1:0}, \sigma_{2:0}\} \cup \{\sigma_{3:1}, \sigma_{4:1}\} \cup \{\sigma_{7:2}, \sigma_{8:2}\}$
$\cup \{\sigma_{5:3}, \sigma_{6:3}, \sigma_{9:3}\}$
$\cup \{\sigma_{10:4}, \sigma_{11:4}, \sigma_{12:4}, \sigma_{13:4}\}$

Transitions:
$\sigma_{1:0} \rightarrow \sigma_{2:0}$
$\sigma_{2:0} \rightarrow \sigma_{3:1}$
$\sigma_{3:1} \rightarrow \sigma_{4:1}$: $\forall_{LR}\, \{\sigma_{1:0}, \sigma_{2:0}, \sigma_{3:1}, \sigma_{4:1}\}$
$\cup \{\sigma_{7:2}, \sigma_{8:2}\}$
$\sigma_{4:1} \rightarrow \sigma_{5:3}$
$\sigma_{5:3} \rightarrow \sigma_{6:3}$: $\exists_{LR}\{\sigma_{3:1}, \sigma_{4:1}\}$
$\sigma_{5:3} \rightarrow \sigma_{9:3}$: $\forall_{LR}\, \{\sigma_{1:0}, \sigma_{2:0}\} \cup \{\sigma_{7:2}, \sigma_{8:2}\}$
$\cup \{\sigma_{5:3}, \sigma_{6:3}, \sigma_{9:3}\}$
$\cup \{\sigma_{10:4}, \sigma_{11:4}, \sigma_{12:4}, \sigma_{13:4}\}$
$\sigma_{6:3} \rightarrow \sigma_{7:2}$
$\sigma_{7:2} \rightarrow \sigma_{8:2}$: $\exists_{LR}\{\sigma_{10:4}, \sigma_{11:4}, \sigma_{12:4}, \sigma_{13:4}\}$
$\sigma_{8:2} \rightarrow \sigma_{10:4}$
$\sigma_{9:3} \rightarrow \sigma_{10:4}$
$\sigma_{10:4} \rightarrow \sigma_{11:4}$: $\forall_L\, \{\sigma_{1:0}, \sigma_{2:0}, \sigma_{3:1}, \sigma_{4:1}\}$
$\sigma_{11:4} \rightarrow \sigma_{12:4}$
$\sigma_{12:4} \rightarrow \sigma_{13:4}$: $\forall_R\, \{\sigma_{1:0}, \sigma_{2:0}\} \cup \{\sigma_{3:1}, \sigma_{4:1}\}$
$\cup \{\sigma_{10:4}, \sigma_{11:4}, \sigma_{12:4}, \sigma_{13:4}\}$
$\sigma_{13:4} \rightarrow \sigma_{1:0}$

$\Phi_{\mathit{Init}} = \{(cr_1, \sigma_{1:0}, cr_1)\}$
$\Phi_{C_F} = \{(cr_2, \sigma_{11:4}, cr_3) \cdot (cr_3, \sigma_{11:4}, cr_2)\}$

Fig. 6. Refined version of Szymanski's algorithm [16] (left), and its parameterized model (right), with counters $cr_1 = [(v_{\sigma_{1:0}} \geq 0) \wedge \wedge_{q \in Q \setminus \{\sigma_{1:0}\}}(v_q = 0)]$, $cr_2 = [\wedge_{q \in Q}(v_q \geq 0)]$, and $cr_3 = [(v_{\sigma_{11:4}} \geq 1) \wedge \wedge_{q \in Q \setminus \{\sigma_{11:4}\}}(v_q \geq 0)]$.

Variables:
locations in $\{l_1, \dots l_{13}\}$
a, w, s in $\{false, true\}$

Transitions:
$l_1[\mathtt{i}] \to l_2[\mathtt{i}]$: non critical
$l_2[\mathtt{i}] \to l_3[\mathtt{i}] : a[\mathtt{i}] := true$
$l_3[\mathtt{i}] \to l_4[\mathtt{i}] : \forall_{\mathtt{j} \neq \mathtt{i}}. \neg s[\mathtt{j}]$

$l_4[\mathtt{i}] \to l_5[\mathtt{i}] : w[\mathtt{i}], s[\mathtt{i}] := true, true$
$l_5[\mathtt{i}] \to l_6[\mathtt{i}] : \exists_{\mathtt{j} \neq \mathtt{i}}. a[\mathtt{j}] \wedge \neg w[\mathtt{j}]$

$l_6[\mathtt{i}] \to l_7[\mathtt{i}] : s[\mathtt{i}] := false$
$l_7[\mathtt{i}] \to l_8[\mathtt{i}] : \forall_{\mathtt{j} \neq \mathtt{i}}. s[\mathtt{j}] \wedge \neg w[\mathtt{j}]$

$l_8[\mathtt{i}] \to l_9[\mathtt{i}] : s[\mathtt{i}] := true$
$l_5[\mathtt{i}] \to l_9[\mathtt{i}] : \forall_{\mathtt{j} \neq \mathtt{i}}. \neg a[\mathtt{j}] \vee w[\mathtt{j}]$

$l_9[\mathtt{i}] \to l_{10}[\mathtt{i}] : w[\mathtt{i}] := false$
$l_{10}[\mathtt{i}] \to l_{11}[\mathtt{i}] : \forall_{\mathtt{j} \neq \mathtt{i}}. \neg w[\mathtt{j}]$

$l_{11}[\mathtt{i}] \to l_{12}[\mathtt{i}] : \forall_{\mathtt{j} < \mathtt{i}}. \neg s[\mathtt{j}]$

$l_{12}[\mathtt{i}] \to l_{13}[\mathtt{i}]$: critical section
$l_{13}[\mathtt{i}] \to l_1[\mathtt{i}] : a[\mathtt{i}], s[\mathtt{i}] := false, false$

$Init = \forall_i . (@l_1[i] \wedge \neg w[i] \wedge \neg s[i])$
$C_F = \exists_{j \neq j'} . (@l_{12}[j] \wedge @l_{12}[j'])$

States:
$\Sigma = \{\sigma_{1:0:0:0}, \sigma_{2:0:0:0}\} \cup \{\sigma_{3:1:0:0}, \sigma_{4:1:0:0}\}$
$\cup \{\sigma_{5:1:1:1}, \sigma_{6:1:1:1}, \sigma_{9:1:1:1}\} \cup \{\sigma_{7:1:1:0}, \sigma_{8:1:1:0}\}$
$\cup \{\sigma_{10:1:0:1}, \sigma_{11:1:0:1}, \sigma_{12:1:0:1}, \sigma_{13:1:0:1}\}$

Transitions:
$\sigma_{1:0:0:0} \to \sigma_{2:0:0:0}$:
$\sigma_{2:0:0:0} \to \sigma_{3:1:0:0}$:
$\sigma_{3:1:0:0} \to \sigma_{4:1:0:0} : \forall_{LR} \{\sigma_{1:0:0:0}, \sigma_{2:0:0:0}\}$
$\cup \{\sigma_{3:1:0:0}, \sigma_{4:1:0:0}\}$
$\cup \{\sigma_{7:1:1:0}, \sigma_{8:1:1:0}\}$
$\sigma_{4:1:0:0} \to \sigma_{5:1:1:1}$:
$\sigma_{5:1:1:1} \to \sigma_{6:1:1:1} : \exists_{LR} \{\sigma_{3:1:0:0}, \sigma_{4:1:0:0}\}$
$\cup \{\sigma_{10:1:0:1}, \sigma_{11:1:0:1}, \sigma_{12:1:0:1}, \sigma_{13:1:0:1}\}$
$\sigma_{6:1:1:1} \to \sigma_{7:1:1:0}$:
$\sigma_{7:1:1:0} \to \sigma_{8:1:1:0} : \exists_{LR} \{\sigma_{10:1:0:1}, \sigma_{11:1:0:1}\}$
$\cup \{\sigma_{12:1:0:1}, \sigma_{13:1:0:1}\}$
$\sigma_{8:1:1:0} \to \sigma_{9:1:1:1}$:
$\sigma_{5:1:1:1} \to \sigma_{9:1:1:1} : \forall_{LR} \{\sigma_{1:0:0:0}, \sigma_{2:0:0:0}\}$
$\cup \{\sigma_{5:1:1:1}, \sigma_{6:1:1:1}, \sigma_{9:1:1:1}\} \cup \{\sigma_{7:1:1:0}, \sigma_{8:1:1:0}\}$
$\sigma_{9:1:1:1} \to \sigma_{10:1:0:1}$:
$\sigma_{10:1:0:1} \to \sigma_{11:1:0:1} : \forall_{LR} \{\sigma_{1:0:0:0}, \sigma_{2:0:0:0}\}$
$\cup \{\sigma_{3:1:0:0}, \sigma_{4:1:0:0}\}$
$\cup \{\sigma_{10:1:0:1}, \sigma_{11:1:0:1}, \sigma_{12:1:0:1}, \sigma_{13:1:0:1}\}$
$\sigma_{11:1:0:1} \to \sigma_{12:1:0:1} : \forall_{L} \{\sigma_{1:0:0:0}, \sigma_{2:0:0:0}\}$
$\cup \{\sigma_{3:1:0:0}, \sigma_{4:1:0:0}\} \cup \{\sigma_{7:1:1:0}, \sigma_{8:1:1:0}\}$
$\sigma_{12:1:0:1} \to \sigma_{13:1:0:1}$:
$\sigma_{13:1:0:1} \to \sigma_{1:0:0:0}$:

$\Phi_{Init} = \{(cr_1, \sigma_{1:0:0:0}, cr_1)\}$
$\Phi_{C_F} = \{(cr_2, \sigma_{12:1:0:1}, cr_3) \cdot (cr_3, \sigma_{12:1:0:1}, cr_2)\}$

Fig. 7. Gribomont-Zenner algorithm [12], with counters $cr_i = [(v_{\sigma_{1:0:0:0}} \geq 0) \wedge \wedge_{q \in Q \setminus \{\sigma_{1:0:0:0}\}} (v_q = 0)]$, $cr_f = [\wedge_{q \in Q} (v_q \geq 0)]$, and $cr'_f = [(v_{\sigma_{12:1:0:1}} \geq 1) \wedge \wedge_{q \in Q \setminus \{\sigma_{12:1:0:1}\}} (v_q \geq 0)]$

Table 5. $\mathbb{NW}$ based analysis of the Refined Szymanski [16].

Mode	m in ρ_m	$\text{pre}_T, \text{post}_T$ applications	Words	Time	Safe
Backward	$\forall \sigma \in \Sigma.m_0(\sigma) = 0$	1562	266 956	$74.1sec$	$\times$
	$m_1 = m_0[\sigma_{10:4} \leftarrow 1]$	2009	298 659	$91.1sec$	$\times$
	$m_2 = m_1[\sigma_{11:4} \leftarrow 1]$	2019	255211	$91.1sec$	$\times$
	$m_3 = m_2[\sigma_{12:4} \leftarrow 1]$	1843	335 656	$80.7sec$	$\surd$
Forwad	$\forall \sigma \in \Sigma.m_0(\sigma) = 0$	0	1	$< 0.01sec$	$\times$
	$m_1 = m_0[\sigma_{11:4} \leftarrow 1]$	8	418	$0.01sec$	$\times$
	$m_2 = m_1[\sigma_{10:4} \leftarrow 1]$	11	536	$0.02sec$	$\times$
	$m_3 = m_2[\sigma_{8:2} \leftarrow 1]$	27	1717	$0.32sec$	$\times$
	$m_4 = m_3[\sigma_{7:2} \leftarrow 1]$	43	1862	$0.25sec$	$\times$
	$m_5 = m_4[\sigma_{6:3} \leftarrow 1]$	12	357	$< 0.09sec$	$\times$
	$m_6 = m_5[\sigma_{5:3} \leftarrow 1]$	13	180	$0.02sec$	$\times$
	$m_7 = m_6[\sigma_{4:1} \leftarrow 1]$	614	15146	$30sec$	$\times$
	$m_8 = m_7[\sigma_{13:1} \leftarrow 1]$	646	12223	$42sec$	$\times$
	$m_9 = m_8[\sigma_{12:4} \leftarrow 1]$	46277	4956075	$6days$	$\surd$

Table 6. $\mathbb{NW}$ based analysis of the Gribomont-Zenner mutex [12].

Mode	m in ρ_m	$\text{pre}_T, \text{post}_T$ applications	Words	Time	Safe
Backward	$\forall \sigma \in \Sigma.m_0(\sigma) = 0$	932	233 604	$46sec$	$\times$
	$m_1 = m_0[\sigma_{10:1:0:1} \leftarrow 1]$	434	129 368	$11.6sec$	$\surd$
Forwad	$\forall \sigma \in \Sigma.m_0(\sigma) = 0$	0	1	$< 0.01sec$	$\times$
	$m_1 = m_0[\sigma_{12:1:0:1} \leftarrow 1]$	9	636	$0.01sec$	$\times$
	$m_2 = m_1[\sigma_{11:1:0:1} \leftarrow 1]$	11	685	$0.02sec$	$\times$
	$m_3 = m_2[\sigma_{10:1:0:1} \leftarrow 1]$	12	602	$0.02sec$	$\times$
	$m_4 = m_3[\sigma_{9:1:1:1} \leftarrow 1]$	13	651	$0.03sec$	$\times$
	$m_5 = m_4[\sigma_{8:1:1:0} \leftarrow 1]$	28	1695	$0.2sec$	$\times$
	$m_6 = m_5[\sigma_{7:1:1:0} \leftarrow 1]$	54	3003	$0.29sec$	$\times$
	$m_7 = m_6[\sigma_{6:1:1:1} \leftarrow 1]$	52	2758	$1.82sec$	$\times$
	$m_8 = m_7[\sigma_{5:1:1:1} \leftarrow 1]$	57	866	$0.24sec$	$\times$
	$m_9 = m_8[\sigma_{4:1:0:0} \leftarrow 1]$	81	1006	$0.18sec$	$\times$
	$m_{10} = m_9[\sigma_{13:1:0:1} \leftarrow 1]$	18265	1790082	$12239sec$	$\surd$

9 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 that can be uniformly tuned. Based on the representation, we implemented a counter example based refinement scheme that illustrates the applicability and the relevance of the approach, both for forward and for backward analysis. Possible future work can investigate more general representations to apply to heap or graph manipulating programs.

References

1. Abdulla, P.A., Delzanno, G., Rezine, A.: Approximated context-sensitive analysis for parameterized verification. In: Lee, D., Lopes, A., Poetzsch-Heffter, A. (eds.) FMOODS/FORTE. Lecture Notes in Computer Science, vol. 5522, pp. 41–56. Springer (2009)
2. Abdulla, P.A., Henda, N.B., Delzanno, G., Rezine, A.: Regular model checking without transducers (on efficient verification of parameterized systems). In: Proc. TACAS '07, 13^{th} Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 4424, pp. 721–736. Springer Verlag (2007)
3. Abdulla, P.A., Jonsson, B., Nilsson, M., d'Orso, J.: Regular model checking made simple and efficient. In: Proc. CONCUR 2002, 13^{th} Int. Conf. on Concurrency Theory. Lecture Notes in Computer Science, vol. 2421, pp. 116–130 (2002)
4. Apt, K., Kozen, D.: Limits for automatic verification of finite-state concurrent systems. Information Processing Letters 22, 307–309 (1986)
5. Arons, T., Pnueli, A., Ruah, S., Xu, J., Zuck, L.: Parameterized verification with automatically computed inductive assertions. In: Berry, Comon, Finkel (eds.) Proc. 13^{th} Int. Conf. on Computer Aided Verification. Lecture Notes in Computer Science, vol. 2102, pp. 221–234 (2001)
6. Boigelot, B., Legay, A., Wolper, P.: Iterating transducers in the large. In: Proc. 15^{th} Int. Conf. on Computer Aided Verification. Lecture Notes in Computer Science, vol. 2725, pp. 223–235 (2003)
7. Bouajjani, A., Habermehl, P., Vojnar, T.: Abstract regular model checking. In: CAV04. pp. 372–386. Lecture Notes in Computer Science, Springer Verlag, Boston (July 2004)
8. Clarke, E., Talupur, M., Veith, H.: Environment abstraction for parameterized verification. In: Proc. VMCAI '06, 7^{th} Int. Conf. on Verification, Model Checking, and Abstract Interpretation. Lecture Notes in Computer Science, vol. 3855, pp. 126–141 (2006)
9. Geeraerts, G., Raskin, J.F., Van Begin, L.: Expand, Enlarge and Check: new algorithms for the coverability problem of WSTS. Journal of Computer and System Sciences 72(1), 180–203 (2006), http://www.ulb.ac.be/di/ssd/ggeeraer/papers/eec-journal.pdf
10. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. Journal of the ACM 39(3), 675–735 (1992)
11. Graf, S., Saidi, H.: Construction of abstract state graphs with PVS. In: Proc. 9^{th} Int. Conf. on Computer Aided Verification. vol. 1254. Springer Verlag, Haifa, Israel (1997)
12. Gribomont, E., Zenner, G.: Automated verification of Szymanski's algorithm. In: Proc. TACAS '98, 4^{th} Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 1384, pp. 424–438 (1998)
13. Higman, G.: Ordering by divisibility in abstract algebras. Proc. London Math. Soc. (3) 2(7), 326–336 (1952)
14. Kesten, Y., Maler, O., Marcus, M., Pnueli, A., Shahar, E.: Symbolic model checking with rich assertional languages. In: Grumberg, O. (ed.) Proc. 9^{th} Int. Conf. on Computer Aided Verification. vol. 1254, pp. 424–435. Springer Verlag, Haifa, Israel (1997)

15. Lisitsa, A.: Finite reasons for safety - parameterized verification by finite model finding. J. Autom. Reasoning 51(4), 431–451 (2013)
16. Manna, Z., Pnueli, A.: An exercise in the verification of multi – process programs. In: Feijen, W., van Gasteren, A., Gries, D., Misra, J. (eds.) Beauty is Our Business. pp. 289–301. Springer Verlag (1990)
17. Pnueli, A., Ruah, S., Zuck, L.: Automatic deductive verification with invisible invariants. In: Proc. TACAS '01, 7^{th} Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. vol. 2031, pp. 82–97 (2001)
18. Pnueli, A., Xu, J., Zuck, L.: Liveness with (0,1,infinity)-counter abstraction. In: Proc. 14^{th} Int. Conf. on Computer Aided Verification. Lecture Notes in Computer Science, vol. 2404 (2002)
19. Touili, T.: Regular Model Checking using Widening Techniques. Electronic Notes in Theoretical Computer Science 50(4) (2001), proc. Workshop on Verification of Parametrized Systems (VEPAS'01), Crete, July, 2001
20. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Proc. LICS '86, 1^{st} IEEE Int. Symp. on Logic in Computer Science. pp. 332–344 (June 1986)