An Algorithmic Approach for Checking Closure Properties of Temporal Logic Specifications and ω-Regular Languages

Doron Peled

Bell Laboratories, Lucent Technologies
700 Mountain Ave.
Murray Hill, NJ 07974-0636, USA

Thomas Wilke[1]

Christian-Albrechts-Universität zu Kiel
Institut für Informatik und Praktische Mathematik
24098 Kiel, Germany

Pierre Wolper

Université de Liège
Institut Montefiore, B28
4000 Liège, Belgium

In concurrency theory, there are several examples where the interleaved model of concurrency can distinguish between execution sequences which are not significantly different. One such example is sequences that differ from each other by stuttering, i. e., the number of times a state can adjacently repeat. Another example is executions that differ only by the ordering of independently executed events. Considering these sequences as different is semantically rather meaningless. Nevertheless, specification languages that are based on interleaving semantics, such as linear temporal logic (LTL), can distinguish between them. This situation has led to several attempts to define languages that cannot distinguish between such equivalent sequences. In this paper, we take a different approach to this problem: we develop algorithms for deciding if a property cannot distinguish between equivalent sequences, i. e., is *closed* under the equivalence relation. We focus on properties represented by regular languages, ω-regular languages, or propositional LTL formulas and show that for such properties there is a wide class of equivalence relations for which determining closure is decidable, in fact is in PSPACE. Hence, checking the closure of a specification is no more difficult than checking satisfiability of a temporal formula. Among the closure properties we are able to handle, one finds trace closedness, stutter closedness and projective closedness, for all of which we are also able to prove a PSPACE lower bound. Being able to check that a property is closed under an equivalence relation has an immediate application in state-space exploration based verification. Indeed, the knowledge that the specification does not distinguish between equivalent execution sequences allows constructing a reduced state space where it is sufficient that at least one sequence per equivalence class is represented.

1 Introduction

In the total order model of concurrency, the atomic actions of the various processes are *interleaved* into totally ordered executions. If two actions are *independent*, they will be interleaved in both possible ways, but will appear in each execution in a specific, though arbitrary, order. Distinguishing between sequences that differ from each other only by the order of concurrently executed events is artificial and mostly meaningless. It is thus usual and useful to group such sequences into equivalence classes. A well-known way of doing this is the concept of *traces* due to Mazurkiewicz [10]: a trace is a set of interleaving sequences that can be obtained from each other by successively commuting independent adjacent actions. Traces are equivalence classes, and sequences belonging to the same trace are said to be *trace equivalent*. Unfortunately, most common specification languages, such as linear-time temporal logic [18] (LTL), naturally allow the specification of properties that are not trace closed, i. e., that can distinguish between trace equivalent sequences. One will usually not specify such properties, but since they can be expressed, every property should be treated as if it might not be trace closed. To work around this, several attempts have been made in the past to define logics that can only define trace-closed properties, e.g., ISTL [5,16], TrPTL [22], TLC [1], and recently LTrL [23]. None of these logics is completely satisfying: they lack simplicity, cannot express all relevant properties, or do not have a known elementary decision procedure. A practical and complete solution to the problem has not yet been given.

Another equivalence that is useful in studying concurrent systems is *stutter equivalence*: a pair of sequences are considered to be equivalent if they differ in at most the number of times a state may adjacently repeat [9]. Although next-time operator free linear-time temporal logic formulas are naturally stutter closed, i. e., cannot distinguish between stutter equivalent sequences, the use of a next-time operator does not preclude stutter closure and can be convenient. Finally, projective equivalence [13] is an extension of stutter equivalence that requires stutter equivalence of various projections of a sequence.

One context in which knowing that a property is closed is valuable is that of *partial-order verification algorithms* [26,4,30,14,15]. These algorithms proceed by checking a property on a reduced state space obtained by only exploring selected interleaving sequences. The reduction is based on the observation that it is not necessary to explore different interleavings that vary from each other only by the relative order of occurrence of independent (concurrent) transitions. Since it is guaranteed that at least one sequence is selected out of each

[1] Part of the research reported here was conducted while the author was visiting DIMACS as part of the Special Year on Logic and Algorithms.

equivalence class for trace equivalence [14] or stuttering equivalence [26,15], one needs to ensure that the checked property is closed with respect to the equivalence relation exploited. If the property is not closed, one has to be more restrictive as to which transitions can be viewed as independent, with the consequence that a smaller reduction is obtained. Thus, being able to check whether a property is closed for a given equivalence relation is important for achieving good partial-order reductions. Furthermore, the same also applies in the context of theorem proving based formal verification [5]. Finally, recognizing projective closedness can also be used for improving the effectiveness of partial-order reduction [14,13]. Projective properties are also preserved by sequential consistent [8] implementations of cache protocols [13].

In this paper, we study the problem of determining whether a property is closed under various equivalence relations, including what we will call concurrency relations, namely, trace equivalence, stutter equivalence, and projective equivalence [13]. We exhibit sufficient conditions that the equivalence relation should satisfy in order for the problem to be decidable for regular and ω-regular properties and hence also for linear-time temporal logic properties [27]. In the case of regular languages, we only assume that the equivalence relation is generated by a sequential relation [24], that is, by a relation 'recognized' by a finite automaton; in the case of ω-regular languages, we assume that the equivalence relation is what we call a "piecewise extension" (see Definition 13) of an equivalence relation on finite words generated by a sequential relation.

We first show how closure under an equivalence relation generated from a sequential relation can be decided for finite word languages. We obtain a polynomial time decision procedure for languages represented by deterministic automata and show that the problem is in PSPACE when non-deterministic automata are used as a representation. Furthermore, we obtain a matching lower bound for each of the concurrency relations. Extending the decision procedure to ω-regular languages is more difficult, but is achieved by using the characterization of ω-regular languages in terms of congruences [2]. Furthermore, the problem remains in PSPACE both for languages specified by non-deterministic automata and by LTL formulas, with the matching lower bound still holding for each of the concrete relations we consider.

Closure of ω-regular languages with respect to trace equivalence is also dealt with in [3] and [11]. In the first paper, the authors prove that it is decidable whether a regular ω-language is trace-closed. However, the decision procedure that is suggested by their proof has worst-case space complexity at least exponential. In the independent work [11], it is shown that determining whether an ω-regular language given by a so-called "I-diamond" Muller or Büchi automaton is trace-closed is PSPACE-complete. As I-diamond Büchi automata are restricted Büchi automata, the hardness result from [11] is stronger than what we prove. On the other hand, the PSPACE upper bound from [11] refers only to I-diamond Büchi and Muller automata, which means it is weaker than what we prove.

2 Concurrency Relations and the Closure Problem

We recall the notions of trace, stutter, and projective equivalence, which will also be referred to as *concurrency relations*. We also introduce the notion of the limit extension of an equivalence relation. This allows us to view the infinite versions of the concurrency relations as uniform extensions of the corresponding finite versions.

2.1 The Limit Extension of an Equivalence Relation

Throughout this paper, Σ stands for a finite alphabet and Σ^∞ for $\Sigma^* \cup \Sigma^\omega$. The prefix order on Σ^∞ is denoted by $\sqsubset$, and the first and last letter of a word $v \in \Sigma^+$ are denoted by $\mathrm{fst}(v)$ and $\mathrm{lst}(v)$, respectively.

Definition 1 (limit extension) *The* limit extension $\sim^{\mathrm{lim}} \subseteq \Sigma^\omega \times \Sigma^\omega$ *of an equivalence relation* $\sim \subseteq \Sigma^* \times \Sigma^*$ *is defined by:* $\alpha \sim^{\mathrm{lim}} \beta$ *if and only if*

- *for every* $u \in \Sigma^*$ *such that* $u \sqsubset \alpha$ *there exist* v, $v' \in \Sigma^*$ *such that* $v \sqsubset \beta$ *and* $uv' \sim v$, *and*
- *for every* $u \in \Sigma^*$ *such that* $u \sqsubset \beta$ *there exist* v, $v' \in \Sigma^*$ *such that* $v \sqsubset \alpha$ *and* $uv' \sim v$.

Recall that an equivalence relation $\sim$ on Σ^* is called a congruence relation if $uv \sim u'v'$ whenever $u \sim u'$ and $v \sim v'$.

Lemma 2 *If* $\sim$ *is a congruence relation on* Σ^*, *then* $\sim^{\mathrm{lim}}$ *is an equivalence relation on* Σ^ω.

PROOF. The relation $\sim^{\mathrm{lim}}$ is a binary relation on Σ^ω and obviously reflexive and symmetric.

To show transitivity, assume $\alpha \sim^{\mathrm{lim}} \beta \sim^{\mathrm{lim}} \gamma$ and let $u \sqsubset \alpha$. By definition of $\sim^{\mathrm{lim}}$, there exist $v \sqsubset \beta$ and v' such that $uv' \sim v$. On the other hand, since $\beta \sim^{\mathrm{lim}} \gamma$, there exist $w \sqsubset \gamma$ and w' such that $vw' \sim w$. Thus $uv'w' \sim w$, since $\sim$ is a congruence relation. Symmetrically, for $u \sqsubset \gamma$ we can find v, v', w, w' such that $w \sqsubset \alpha$ and $uv'w' \sim w$. Hence $\alpha \sim^{\mathrm{lim}} \gamma$. $\square$

2.2 Trace Equivalence

A *dependency relation* is a reflexive and symmetric relation $D \subseteq \Sigma \times \Sigma$. The pair $(\Sigma,\ D)$ is also called a *dependency graph.*

For two words $u, v \in \Sigma^*$, write $u \overset{1}{\equiv} v$ if there exist words w_1, w_2 and letters a, b such that $(a, b) \notin D$, $u = w_1abw_2$ and $v = w_1baw_2$, i. e., if u is obtained from v by exchanging the order of two adjacent independent letters. Let $\equiv$ be the reflexive and transitive closure of the relation $\overset{1}{\equiv}$. We say that u and v are *trace equivalent* [10] over (Σ, D) if $u \equiv v$. That is, u is trace equivalent to v if u can be obtained from v by repeatedly commuting adjacent independent letters.

Following [6], ω-words α and β are said to be *trace equivalent* if $\alpha \equiv^{\lim} \beta$.

2.3 Stutter Equivalence

The *stutter removal* operator $\natural \colon \Sigma^\infty \mapsto \Sigma^\infty$ maps every word x to the word that is obtained from x by replacing every maximal finite substring of identical letters by a single copy of the letter. For instance, $\natural(aabbbcaa) = abca$, $\natural(aabbbc^\omega) = abc^\omega$, and $\natural((aab)^\omega) = (ab)^\omega$. Words x and y are said to be *stutter equivalent* if $\natural(x) = \natural(y)$.

Even though stutter equivalence is not defined as a limit extension, it could actually be done so:

Lemma 3 *Stutter equivalence over ω-words is the limit extension of stutter equivalence over finite words.*

PROOF. We have to show that if $\sim$ denotes stutter equivalence over finite words, then $\sim^{\lim}$ denotes stutter equivalence over ω-words.

Let α and β be ω-words, and let $\alpha = u_1u_2u_3\ldots$ and $\beta = v_1v_2v_3\ldots$ be the decompositions of α and β into maximal subwords of identical letters. In case any of these decompositions should be finite, modify it so it becomes infinite, by chopping the last factor into letters.

First, suppose α and β are stutter equivalent ω-words. Then $\natural(u_i) = \natural(v_i)$ for every $i \geq 0$. Let $u \sqsubset \alpha$. Let n be such that $u \sqsubset u_1 \ldots u_n$. Then there exists v' such that $uv' = u_1 \ldots u_n$. On the other hand, $u_1 \ldots u_n \sim v_1 \ldots v_n$. Thus $uv' \sim v \sqsubset \beta$ where $v = v_1 \ldots v_n$. Symmetrically, if $v \sqsubset \beta$, there exist u' and u such that $vu' \sim u \sqsubset \alpha$. Hence $\alpha \sim^{\lim} \beta$.

Next, suppose $\alpha \sim^{\lim} \beta$. For every n there exist v and v' such that $u_1 \dots u_n v' \sim v \sqsubset \beta$, which means $\natural(u_1 \dots u_n v') = \natural(v) \sqsubset \natural(\beta)$, and therefore $\natural(u_1 \dots u_n) \sqsubset \natural(\beta)$. Hence $\natural(\alpha) \sqsubset \natural(\beta)$. Symmetrically, $\natural(\beta) \sqsubset \natural(\alpha)$. Thus, $\natural(\alpha) = \natural(\beta)$, i. e., α and β are stutter equivalent. $\square$

2.4 Projective Equivalence

Projective equivalence [13] is the natural extension of stutter equivalence to component alphabets. For simplicity in notation, the definitions and statements below are for an alphabet with two components only but easily generalize to alphabets with any (but finite) number of components.

Let Σ_1 and Σ_2 be finite alphabets and $\Sigma = \Sigma_1 \times \Sigma_2$. As usual, we identify the elements of Σ^∞ with elements of $\Sigma_1^\infty \times \Sigma_2^\infty$ in the natural way, e. g., we identify $(a_1, b_1)(a_2, b_2)$ with $(a_1 a_2, b_1 b_2)$.

For each pair $c = (a, b) \in \Sigma$, let $c|_1 = a$ and $c|_2 = b$. Accordingly, for each word $x = c_1 c_2 c_3 \ldots \in \Sigma^\infty$, define $x|_i = c_1|_i \, c_2|_i \, c_3|_i \ldots$, $i \in \{1, 2\}$.

Words x and y are called *projective equivalent* if $\natural(x|_1) = \natural(y|_1)$ and $\natural(x|_2) = \natural(y|_2)$.

As with stutter equivalence, we obtain:

Lemma 4 *Projective equivalence over ω-words is the limit extension of projective equivalence over finite words.*

PROOF. We have to show that if $\sim$ denotes projective equivalence over finite words, then $\sim^{\lim}$ denotes projective equivalence over ω-words.

First, assume α and β are projective equivalent ω-words. Let $u \sqsubset \alpha$. Then $u|_1 \sqsubset \alpha|_1$ and $u|_2 \sqsubset \alpha|_2$. Lemma 3 applied to $\alpha|_1$ and $\beta|_1$ yields that there exist $v_1 \sqsubset \beta|_1$ and v_1' such that $u|_1 v_1'$ and v_1 are stutter equivalent. Similarly, there exist $v_2 \sqsubset \beta|_2$ and v_2' such that $u|_2 v_2'$ and v_2 are stutter equivalent. W. l. o. g., assume $|v_1| \geq |v_2|$. Let w be such that $|w| = |v_1| - |v_2|$ and $v_2 w \sqsubset \beta|_2$. Then $u|_2 v_2' w \sim v_2 w \sqsubset \beta|_2$. Let $a = \mathrm{lst}(v_1')$, $b = \mathrm{lst}(v_2' w)$, and define v' and v to be $(v_1' a^{|v_2' w|}, v_2' w b^{|v_1'|})$ and $(v_1, v_2 w)$, respectively. Then $uv' \sim v \sqsubset \beta$. By symmetry, we obtain $\alpha \sim^{\lim} \beta$.

Next, assume $\alpha \sim^{\lim} \beta$. Then $\alpha|_1$ and $\beta|_1$ are equivalent with respect to the limit extension of stutter equivalence. Accordingly, the same holds for $\alpha|_2$ and $\beta|_2$. Thus, by Lemma 3, $\natural(\alpha|_1) = \natural(\beta|_1)$ and $\natural(\alpha|_2) = \natural(\beta|_2)$, hence α and β are projective equivalent. $\square$

Given an equivalence relation $\sim$ over a set M and a subset $M' \subseteq M$, we say that M' is *$\sim$-closed* or *closed under* $\sim$ if M' is a union of equivalence classes of $\sim$. In particular, we will say that a language is trace closed (with respect to a given dependency alphabet), stutter closed, or projective closed if it is closed under the corresponding trace, stutter, or projective equivalence relation, respectively.

Given an alphabet Σ, a class $\mathcal{L}$ of languages of finite or ω-words over Σ, and an equivalence relation $\sim$ on Σ^* respectively Σ^ω, the *closure problem* is to determine whether a given language $L \in \mathcal{L}$ is $\sim$-closed.

The term 'closure problem' can be explained as follows. Given an equivalence relation on Σ, the mapping $L \mapsto \{x \mid \exists y\,(x \sim y \wedge y \in L)\}$ defines a closure operator on the set of all languages over Σ. The closure problem is to determine whether the closure of a given language L is L itself.

We are mainly interested in those closure problems where $\sim$ is one of the concurrency relations and $\mathcal{L}$ is the class of regular or ω-regular languages, or LTL-definable ω-languages. Recall that every LTL-definable ω-language is ω-regular [29].

3 The Closure Problem for Finite Words

We prove that the closure problem is in PSPACE for the set of regular languages of finite words over a given alphabet Σ with respect to every equivalence relation generated by a sequential relation (for a definition of sequential, see below). The finite version of all the concurrency relations introduced in the previous section are, in fact, generated by sequential relations, which means their closure problems are in PSPACE. In addition, we show that for these relations the closure problems are PSPACE-hard and hence obtain a tight characterization.

3.1 The General Situation

Let $\$$ be a letter not belonging to the alphabet Σ, write $\Sigma_\$$ for $\Sigma \cup \{\$\}$ and $x \downarrow \Sigma$ for the *canonic projection* $\Sigma_\$^\infty \to \Sigma^\infty$ (the one erasing the letter $\$$). For a binary relation R on Σ^*, let $R_\$$ be the relation that, for $(u, v) \in R$, contains $(u\,\$^{|v|-|u|}, v)$ if $|u| \leq |v|$ and $(u, v\,\$^{|u|-|v|})$ otherwise. Note that any two words that are related by $R_\$$ have the same length, and recall that we identify every pair in $R_\$$ with a word over $\Sigma_\$ \times \Sigma_\$$, and $R_\$$ itself with a language over $\Sigma_\$ \times \Sigma_\$$.

Definition 5 ([24]) *A binary relation R on Σ^* is* sequential *if $R_\$$ (viewed as a formal language over $\Sigma_\$ \times \Sigma_\$$) is regular.*

We say that an equivalence relation $\sim$ is *generated* by a relation $\overset{1}{\sim}$ if $\sim$ is the reflexive, symmetric, transitive closure of $\overset{1}{\sim}$, i. e., if $\sim$ is the smallest equivalence relation containing $\overset{1}{\sim}$.

The closure problem for an equivalence relation $\sim$ generated by a relation $\overset{1}{\sim}$ can be solved by looking only at $\overset{1}{\sim}$:

Lemma 6 *Let $L \subseteq \Sigma^*$ and $\sim$ an equivalence relation on Σ^* generated by a symmetric relation $\overset{1}{\sim}$. Then L is not $\sim$-closed if and only if there exist words $u, v \in \Sigma^*$ such that $u \overset{1}{\sim} v$, $u \in L$, and $v \notin L$.*

PROOF. For the non-trivial direction assume $u \in L$ iff $v \in L$ for all u and v with $u \overset{1}{\sim} v$. Let u' and v' be distinct words such that $u' \sim v'$. Then there exist words $u_0, \ldots, u_n$ such that $u_0 = u'$, $u_n = v'$, and $u_i \overset{1}{\sim} u_{i+1}$ for $i < n$. As we have $u_i \in L$ iff $u_{i+1} \in L$, we obtain $u_0 \in L$ iff $u_n \in L$, hence $u' \in L$ iff $v' \in L$, i. e., L is $\sim$-closed. □

Suppose $\overset{1}{\sim}$ from Lemma 6 is sequential and given by a deterministic finite automaton C. Let A_1 be a finite automaton over $\Sigma_\$ \times \Sigma_\$$ accepting exactly all words u with $u|_1 \in L\$^*$ and $u|_2 \in \Sigma^*\$^*$. Similarly, let A_2 be an automaton accepting exactly all words u with $u|_1 \in \Sigma^*\$^*$ and $u|_2 \notin L\$^*$. Then, by Lemma 6, L is $\sim$-closed if and only if

$$L(C \times A_1 \times A_2) = \emptyset \tag{1}$$

where $\times$ denotes automata-theoretic product.

This leads to the following result concerning the time complexity of the closure problem.

Theorem 7 *Let $\sim$ be an equivalence relation generated by a sequential relation, and consider the closure problem for $\sim$ with respect to languages represented by deterministic or non-deterministic finite automata.*

(i) The closure problem is decidable in time $\mathcal{O}(|A|^2)$ for deterministic automata.

(ii) The closure problem is decidable in time $\mathcal{O}(|A|\, 2^{|A|})$ for non-deterministic automata.

Here, $|A|$ stands for the size of an input automaton A.

PROOF. First note that w. l. o. g. we can assume that $\overset{1}{\sim}$ is symmetric, for the symmetric closure of a sequential relation is sequential. Also, recall that the emptiness problem for finite automata is solvable in linear time.

(i) Slight modifications of A yield deterministic automata A_1 and A_2 as specified above. Both A_1 and A_2 can be constructed in time $\mathcal{O}(|A|)$. Hence $C \times A_1 \times A_2$ can be constructed in time $\mathcal{O}(|C|\, |A|^2)$. Condition (1) can therefore be checked in time $\mathcal{O}(|C|\, |A|^2)$, which is identical with $\mathcal{O}(|A|^2)$, as C is considered constant.

(ii) Slight modifications of A yield a non-deterministic automaton A_1 and a non-deterministic automaton A_2' recognizing the complement of $L(A_2)$. Determinization (using the subset construction) and a following complementation (by complementing the final state set) of A_2' lead to an automaton A_2 as desired. The construction of A_1 and A_2 can be carried out in time $\mathcal{O}(|A|)$ and $\mathcal{O}(2^{|A|})$, respectively. Thus $C \times A_1 \times A_2$ can be constructed in time $\mathcal{O}(|C|\, |A|\, 2^{|A|})$, hence (1) can be checked in time $\mathcal{O}(|A|\, 2^{|A|})$. □

We obtain the following bound for the space complexity of the closure problem.

Theorem 8 *Let $\sim$ be an equivalence relation generated by a sequential relation.*

The closure problem for $\sim$ with respect to languages represented by non-deterministic finite automata is in PSPACE.

PROOF. In general, the product automaton $C \times A_1 \times A_2$ cannot be constructed in space polynomial in $\mathcal{O}(|A|)$. It is, however, possible to check for its emptiness within this complexity bound.

The state space of $C \times A_1 \times A_2$ (as described in the proof of Theorem 7, part (ii)) is the Cartesian product of three state spaces of size $\mathcal{O}(1)$, $\mathcal{O}(|A|)$, and $\mathcal{O}(2^{|A|})$. Each state of the product automaton can be represented in space $\mathcal{O}(|A|)$. Moreover, given two states, one can determine in space $\mathcal{O}(|A|)$ whether one is the successor of the other. The following non-deterministic algorithm for checking the emptiness of $C \times A_1 \times A_2$ can thus be implemented in space $\mathcal{O}(|A|)$:

1. let q be the initial state of $C \times A_1 \times A_2$
2. choose a state q'
3. if q' is a successor of q, let $q = q'$, else halt (without accepting)
4. if q is final, accept, else goto 2

From Savitch's theorem we can conclude that this can be done in deterministic polynomial space as well. □

3.2 Application to the Concurrency Relations

Returning to the concurrency relations presented in Sect. 2, it is enough to show that each one of the three equivalence relations is generated by a sequential relation. Trace equivalence $\equiv$ is already defined to be the transitive closure of a sequential relation, namely $\overset{1}{\equiv}$. Stutter equivalence is generated by the sequential relation $\{(uav, uaav) \mid u, v \in \Sigma^*,\ a \in \Sigma\}$. For projective equivalence, it is slightly more difficult to find the right representation:

Lemma 9 *Projective equivalence is generated by the sequential relation that relates*

(i) $u(a, b)v$ *with* $u(a, b)(a, b)v$,
(ii) $u(a_1, b_1)(a_1, b_2)(a_2, b_3)v$ *with* $u(a_1, b_1)(a_2, b_2)(a_2, b_3)v$, *and*
(iii) $u(a_1, b_1)(a_2, b_1)(a_3, b_2)v$ *with* $u(a_1, b_1)(a_2, b_2)(a_3, b_2)v$,

for $u, v \in \Sigma^*$, $a_1, a_2, a_3 \in \Sigma_1$, *and* $b_1, b_2, b_3 \in \Sigma_2$.

PROOF. Let $\sim$ denote the equivalence relation generated by the relation defined in the lemma. Obviously, if $u \sim v$ then u and v are projective equivalent. For the other direction, assume u and v are projective equivalent words. We have to show $u \sim v$. If one of the words is empty then so is the other and there is nothing to show. So for the rest of the proof assume u and v are non-empty. Let $u_i = u|_i$ and $v_i = v|_i$ for $i = 1, 2$. Since u and v are projective equivalent, they end in the same letter, say (a, b). By (i), it is enough to show $u(a, b) \sim v(a, b)$.

Using repeatedly (ii), we get

$$u(a, b) \sim (\natural(u_1)a^{|u_1|-|\natural(u_1)|+1}, u_2 b) \ .$$

Using repeatedly (iii), we see that the right hand side of this relation is $\sim$-equivalent to

$$(\natural(u_1)a^{|u_1|-|\natural(u_1)|+1}, \natural(u_2)b^{|u_2|-|\natural(u_2)|+1}) \ .$$

This, in turn, is $\sim$-equivalent to

$$(\natural(u_1)a^{|v|+|u_1|-|\natural(u_1)|+1}, \natural(u_2)b^{|v|+|u_2|-|\natural(u_2)|+1})$$

in view of (i). As we have $|u_1| = |u_2| = |u|$, we conclude

$$u(a,b) \sim (\natural(u_1)a^{|u|+|v|-|\natural(u_1)|+1}, \natural(u_2)b^{|u|+|v|-|\natural(u_2)|+1}) \ , \tag{2}$$

and, by symmetry,

$$v(a,b) \sim (\natural(v_1)a^{|u|+|v|-|\natural(v_1)|+1}, \natural(v_2)b^{|u|+|v|-|\natural(v_2)|+1}) \ . \tag{3}$$

Now notice that the right hand sides of (2) and (3) are identical, since $\natural(u_1) = \natural(v_1)$ and $\natural(u_2) = \natural(v_2)$ (recall that u and v are projective equivalent). $\square$

In view of Theorem 8, we have:

Corollary 10 *The closure problem for trace, stutter, and projective equivalence with respect to languages represented by non-deterministic finite automata is in PSPACE.*

Next, we complete this result by proving a matching lower bound. We say that a concurrency relation is *non-trivial* if the underlying alphabet Σ contains at least two letters and, in case of trace equivalence, there is at least one pair of independent letters.

Theorem 11 *The closure problem for trace, stutter, and projective equivalence (over non-trivial alphabets) with respect to languages represented by non-deterministic finite automata is PSPACE-hard.*

PROOF. Lemma 3.2.3. from [7] states that the problem of determining whether the intersection of languages recognized by deterministic finite automata is empty is PSPACE-hard. By modifying the proof of the lemma in a straightforward way, one proves that for an arbitrary alphabet Σ with at least two letters, say a and b, the following problem is PSPACE hard: given a non-deterministic finite automaton A recognizing either Σ^* or $\Sigma^* \setminus \{u\}$ for some $u \in ab\Sigma^*$, determine whether or not A recognizes Σ^*.

Now, assume (Σ, D) is a trace alphabet such that $(a,b) \notin D$ and A is an automaton such that either $L(A) = \Sigma^*$ or $L(A) = \Sigma^* \setminus \{u\}$ for some $u \in ab\Sigma^*$. Then $L(A) = \Sigma^*$ iff $L(A)$ is trace closed. This means the identity function is a reduction from the above problem to trace closedness. Hence, trace closedness is PSPACE-hard. As one easily sees, the identity functions also reduces the above problem to stutter closedness and projective closedness. $\square$

Corollary 12 *The closure problem for trace, stutter, and projective equivalence (over non-trivial alphabets) with respect to languages represented by nondeterministic finite automata is PSPACE-complete.*

Note that, in general, not every equivalence relation on Σ^* is generated by a sequential relation. Moreover, the $\sim$-closure of a regular language is not necessarily regular again. Consider, for instance, the case where $\Sigma = \{a, b\}$, $\sim$ is the trace equivalence relation with respect to the dependency alphabet $D = \{(a, a), (b, b)\}$, and L is the regular language denoted by $(ab)^*$. In this case, the $\sim$-closure of L is the set of words which have as many occurrences of a as occurrences of b. But this language is not regular.

4 The Closure Problem for Infinite Words

We prove that the closure problem is decidable for regular languages of ω-words with respect to a considerable number of equivalence relations, namely, relations that are "piecewise extensions" (see next definition) of relations generated by sequential relations. In fact, we show that the closure problem is in PSPACE for such relations. The infinite versions of the concurrency relations from Sect. 2 are among these relations.

4.1 Piecewise Extensions

Definition 13 (piecewise extension) *Let $\sim$ be a binary relation on Σ^*. The relations $\sim^\omega, \sim^{\omega*} \subseteq \Sigma^\omega \times \Sigma^\omega$ are defined by:*

(i) $\alpha \sim^\omega \beta$ iff there exist decompositions $\alpha = u_0u_1u_2\dots$ and $\beta = v_0v_1v_2\dots$ such that $u_i \sim v_i$ for every $i \geq 0$,
(ii) $\sim^{\omega*}$ is the transitive closure of $\sim^\omega$.

The relation $\sim^{\omega*}$ is called the piecewise extension *of $\sim$.*

It is easy to see that by definition $\sim^{\omega*}$ is an equivalence relation, provided $\sim$ is reflexive and symmetric.

We write $\approx_L$ for the *syntactic congruence of a language $L \subseteq \Sigma^\omega$* [2]. It is the relation over Σ^* defined by: $u \approx_L v$ iff

$$\begin{aligned} &xuyz^\omega \in L \leftrightarrow xvyz^\omega \in L, \text{ and} \\ &x(uy)^\omega \in L \leftrightarrow x(vy)^\omega \in L, \text{ where } uy \neq \varepsilon \text{ and } vy \neq \varepsilon \end{aligned} \tag{4}$$

for $x, y \in \Sigma^*$ and $z \in \Sigma^+$.

The following property of the syntactic congruence, which is an immediate consequence of Lemma 2.2 in [2] and Proposition 2.3 in [12], is of interest to us.

Proposition 14 ([2,12]) *Let $\sim$ be a congruence relation on Σ^* and $L \subseteq \Sigma^\omega$ an ω-regular language. Then L is closed under $\sim^{\omega *}$ if and only if $\sim \subseteq \approx_L$.*

W.l.o.g. we thence assume that $\overset{1}{\sim}$ is reflexive and symmetric. If not, it can easily be closed under reflexivity and symmetry, with the transitive closure of the obtained relation being the same as the transitive closure of the original one. We obtain:

Theorem 15 *Let $\sim$ be a congruence relation on Σ^* generated by a reflexive, symmetric relation $\overset{1}{\sim}$ and $L \subseteq \Sigma^\omega$ an ω-regular language. Then L is closed under $\sim^{\omega *}$ if and only if L is closed under $\overset{1}{\sim}^{\omega}$.*

PROOF. For the non-trivial direction, assume L is not closed under $\sim^{\omega *}$. Then, according to Proposition 14, there exists an equivalence class of $\approx_L$ that is not closed under $\sim$. Hence, by Lemma 6, there are finite words u, v such that $u \overset{1}{\sim} v$ but $u \not\approx_L v$. So there exist x, y, z such that $xuyz^\omega \in L$ and $xvyz^\omega \notin L$, or $x(uy)^\omega \in L$ and $x(vy)^\omega \notin L$. But $xuyz^\omega \overset{1}{\sim}^{\omega} xvyz^\omega$, as well as $x(uy)^\omega \overset{1}{\sim}^{\omega} x(vy)^\omega$. Thus, L is not $\overset{1}{\sim}^{\omega}$-closed. $\square$

To obtain an algorithm for piecewise extensions we now argue as in Section 3.

Suppose $\overset{1}{\sim}$ from Theorem 13 is a sequential relation and C is an automaton for $\overset{1}{\sim}_\$$. It is then easy to construct a Büchi automaton C' recognizing $(\overset{1}{\sim}_\$)^\omega$. Let A_1 and A_2 be Büchi automata over $\Sigma_\$ \times \Sigma_\$$ such that $(\alpha, \beta) \in L(A_1)$ iff $\alpha \downarrow \Sigma \in L$ and $(\alpha, \beta) \in L(A_2)$ iff $\beta \downarrow \Sigma \notin L$. Then, by Theorem 15, L is $\sim$-closed iff

$$L(C' \times A_1 \times A_2) = \emptyset \tag{5}$$

where the product operation $\times$ is defined in the obvious way.

From this, we derive:

Theorem 16 *Let $\sim$ be a congruence relation on Σ^* that is generated by a sequential relation. The closure problem for $\sim^{\omega *}$ with respect to languages represented by Büchi automata is in PSPACE.*

PROOF. The argument from the proof of Theorem 8 goes through with minor modifications:

First, instead of C the automaton C' is used. Second, A_2 is constructed using Safra's method [19]; a state of A_2 can then be represented in space $\mathcal{O}(|A| \log |A|)$. Third, the algorithm is adapted as follows (in order to check whether there is a computation $q_0, \ldots, q_i, \ldots, q_n$ such that q_0 is initial, q_i is final, and $q_i = q_n$).

1. let q be the initial state of $C' \times A_1 \times A_2$
2. choose a state q'
3. if q' is a successor of q, let $q = q'$, else halt (without accepting)
4. if q is final, goto 5 or 2, else goto 2
5. let $q_F = q$
6. choose a state q'
7. if q' is a successor of q, let $q = q'$, else halt (without accepting)
8. if $q = q_F$, accept, else goto 6

This can be implemented by a non-deterministic and thus also by a deterministic polyspace Turing machine.

Alternatively to Safra's construction one can use the method introduced in [21,28]. □

4.2 Limit Extensions and Concurrency Relations

If we can now show that trace, stutter and projective equivalences are piecewise extensions of congruence relations generated by sequential relations, we can apply Theorem 16 to obtain the desired decidability and complexity results.

We first mention the following straightforward claim.

Lemma 17 *If $\sim$ is a congruence relation, then $\sim^{\omega *} \subseteq \sim^{\lim}$.*

PROOF. Since $\sim^{\lim}$ is transitive, we only need to show $\sim^{\omega} \subseteq \sim^{\lim}$.

Suppose $\alpha \sim^{\omega} \beta$, say $\alpha = u_0 u_1 u_2 \ldots$ and $\beta = v_0 v_1 v_2 \ldots$ such that $u_i \sim v_i$ for $i \geq 0$, and $u \sqsubset \alpha$. Let k be such that $u \sqsubset u_0 \ldots u_k$ and let v' be such that $uv' = u_0 \ldots u_k$. Then $uv' \sim v_0 \ldots v_k$, since $\sim$ is a congruence relation. □

Definition 18 (flexible relation) *Let g be a function $\Sigma \times \Sigma \mapsto \mathcal{F}$, where $\mathcal{F}$ is some finite set. A congruence relation $\sim$ is called* flexible with respect to g *if the following conditions hold for all $v, v', w, w' \in \Sigma^+$, $a \in \Sigma$:*

(i) *If $v \sim v'$, $vw \sim v'w'$ and $g(lst(v), fst(w)) = g(lst(v'), fst(w'))$, then $w \sim w'$.*
(ii) *If $v \sim v'$, then $g(lst(v), a) = g(lst(v'), a)$.*

If $\sim$ is flexible with respect to some function g, then we say that $\sim$ is flexible.

The definition of flexibility weakens the notion of left cancellativeness. The latter requires that if $v \sim v'$ and $vw \sim v'w'$ then $w \sim w'$. This does not hold for the stuttering and projective equivalences. To see this, consider the case where $v = v' = a$, $w = ab$ and $w' = b$. What prevents left cancellativeness in this case is that the boundary between v and w and that between v' and w' correspond to different cases: while $lst(v) = fst(w)$, we have $lst(v') \neq fst(w')$. Juxtaposed, the first pair of letters belong to an adjacent repetition of the same letter, while the second pair differs. There is a finite number of cases for the boundary letters (two in this example), represented by the function g. Condition (i) guarantees that if the boundary letters for the two pairs belong to the same case, then $w \sim w'$. This limits the way ab and aab can be broken into pairs of equivalent components. Condition (ii) requires that two equivalent words v, v' would behave in the same way w.r.t. the same first letter a.

Notice that if g is a constant function, then (ii) is trivial and (i) means that $\sim$ is left cancellative.

Let us first see that the concurrency relations are flexible.

Lemma 19 *Each concurrency relation is flexible.*

PROOF. As trace equivalence is known to be left cancellative, trace equivalence is flexible with respect to any constant function.

For stutter equivalence, we take $g\colon \Sigma \times \Sigma \rightarrow \{\textsc{same}, \textsc{different}\}$ with $g(a, b) = \textsc{same}$ if $a = b$ and else $\textsc{different}$.

We have to show that conditions (i) and (ii) are satisfied. To do this, let $\sim$ denote stutter equivalence.

Let $v, v', w, w' \in \Sigma^+$, $a \in \Sigma$.

(i) Suppose $v \sim v'$ and $vw \sim v'w'$. By way of contradiction, suppose $\mathrm{lst}(v) = \mathrm{fst}(w)$ and $\mathrm{lst}(v') \neq \mathrm{fst}(w')$. Then $\natural(vw) \in \natural(v)(\Sigma \setminus \{\mathrm{fst}(w)\})\Sigma^*$ and $\natural(v'w') \in \natural(v')\mathrm{fst}(w')\Sigma^*$. This implies $\natural(vw) \neq \natural(v'w')$, as we have $\natural(v) = \natural(v')$—a contradiction.

(ii) This is satisfied, because we have $\mathrm{lst}(u) = \mathrm{lst}(u')$ whenever u and u' are

non-empty $\sim$-equivalent words.

For projective equivalence, we take a function g with a range that includes four values, namely NONE, BOTH, LEFT and RIGHT, indicating which of the components agree. We set

$$g((a,b),(c,d)) = \begin{cases} \text{BOTH} & \text{if } a = c \text{ and } b = d, \\ \text{NONE} & \text{if } a \neq c \text{ and } b \neq d, \\ \text{LEFT} & \text{if } a = c \text{ and } b \neq d, \\ \text{RIGHT} & \text{if } a \neq c \text{ and } b \neq d. \end{cases}$$

That this is a correct choice can be proved just as in the case of stutter equivalence. $\Box$

The following theorem, which is a generalization of a result for trace equivalence in [3], gives the converse of Lemma 17 for flexible congruences.

Theorem 20 *Let $\sim$ be a flexible congruence relation. Then $\sim^{\lim} = \sim^{\omega *}$.*

PROOF. In view of Lemma 17, we only have to show $\sim^{\lim} \subseteq \sim^{\omega *}$. Let $\sim$ be flexible with respect to g. Assume $\alpha \sim^{\lim} \beta$. We show that there are decompositions $\alpha = u_0 u_1 u_2 \ldots$ and $\beta = v_0 v_1 v_2 \ldots$ and an infinite sequence of words $w_0, w_1, w_2 \ldots$ such that $u_0 = w_0$, and

$$g(\mathrm{lst}(w_{2j}),\, \mathrm{fst}(w_{2j+1})) = g(\mathrm{lst}(u_j),\, \mathrm{fst}(u_{j+1})) \tag{6}$$

and

$$v_j \sim w_{2j} w_{2j+1} \tag{7}$$

hold for $j \geq 0$, and

$$g(\mathrm{lst}(w_{2j-1}),\, \mathrm{fst}(w_{2j})) = g(\mathrm{lst}(v_{j-1}),\, \mathrm{fst}(v_j)) \tag{8}$$

and

$$u_j \sim w_{2j-1} w_{2j} \tag{9}$$

hold for $j \geq 1$ (for a graphical illustration, see Figure 1). This implies $\alpha \sim^{\omega} w_0 w_1 w_2 \ldots \sim^{\omega} \beta$, which means $\alpha \sim^{\omega *} \beta$.

We give an inductive definition for the u_i, v_i, and w_i. In step $2i+1$, we will define v_i and w_{2i+1}; and in step $2i$, we will define u_i and w_{2i}. This will be done in such a way that

(*) after step $2i$ ($i > 0$), (6) and (7) hold for $0 \leq j < i$ and (8) and (9) hold for $1 \leq j \leq i$, and

(**) after step $2i+1$, (6) and (7) hold for $0 \leq j \leq i$ and (8) and (9) hold for $1 \leq j \leq i$.

Notice that after step $2i$ (respectively $2i+1$) the words v_i (respectively u_{i+1}) are not yet defined; nevertheless, (6) (respectively (8)) make sense, as $\mathrm{fst}(u_{i+1})$ (respectively $\mathrm{fst}(v_i)$) will have to be defined as the first letter of what remains from α (respectively β) after removing the prefix $u_0 \ldots u_i$ (respectively $v_0 \ldots v_{i-1}$) which is already defined.

Step 0. We choose an arbitrary non-empty finite prefix $u_0 \sqsubset \alpha$, and set $w_0 = u_0$.

Step $2i{+}1$. By assumption, see (*), (7) and (9), $u_0u_1 \ldots u_i \sim w_0w_1 \ldots w_{2i}$ where $u_0u_1 \ldots u_i \sqsubset \alpha$, and $v_0v_1 \ldots v_{i-1} \sim w_0w_1 \ldots w_{2i-1}$ where $v_0v_1 \ldots v_{i-1} \sqsubset \beta$. Let $a \in \Sigma$ be such that $u_0u_1 \ldots u_ia \sqsubset \alpha$, i.e., a is going to be $\mathrm{fst}(u_{i+1})$. Since $\alpha \sim^{\lim} \beta$, it follows from Definition 1 that there are finite non-empty strings y and z, such that $y \sqsubset \beta$ and $u_0u_1 \ldots u_iaz \sim y$. We choose y to be long enough so that $v_0v_1 \ldots v_{i-1}$ is a proper prefix of it. (We can do so for the following reason: let y be any string such that $y \sqsubset \beta$, say $\beta = y\beta'$, and $u_0u_1 \ldots u_iaz \sim y$. Then for every $y' \sqsubset \beta'$, we have $yy' \sqsubset \beta$ and $u_0u_1 \ldots u_iazy' \sim yy'$. Thus, we can use yy' instead of y, and yy' can be made long enough.)

Let $v_i \in \Sigma^+$ be the word such that $y = v_0v_1 \ldots v_{i-1}v_i$ and set $w_{2i+1} = az$. We now have to show that the new requirements are fulfilled, namely (6) and (7) for $j = i$.

We have $u_i \sim w_{2i-1}w_{2i}$. So, by Definition 18, part (ii), we have $g(\mathrm{lst}(w_{2i}), a) = g(\mathrm{lst}(u_i), a)$. Also, $\mathrm{fst}(w_{2i+1}) = a = \mathrm{fst}(u_{i+1})$ (remember that u_{i+1} will only be defined in the next step). Thus, $g(\mathrm{lst}(w_{2i}), \mathrm{fst}(w_{2i+1})) = g(\mathrm{lst}(u_i), \mathrm{fst}(u_{i+1}))$, which is (6) for $j = i$.

From (**), we get $w_0w_1 \ldots w_{2i+1} = w_0 \ldots w_{2i}az \sim u_0 \ldots u_iaz \sim v_0 \ldots v_{i-1}v_i \sim w_0 \ldots w_{2i-1}v_i$. Also, $w_0 \ldots w_{2i-1} \sim v_0 \ldots v_{i-1}$. From Definition 18, part (i), we can now conclude $w_{2i}w_{2i+1} \sim v_i$, as we have $g(\mathrm{lst}(w_{2i-1}), \mathrm{fst}(w_{2i})) = g(\mathrm{lst}(v_{i-1}), \mathrm{fst}(v_i))$ by (9). Thus we have (7) for $j = i$.

Step $2i$. We proceed symmetrically. □

From Theorems 16 and 20, we can now conclude:

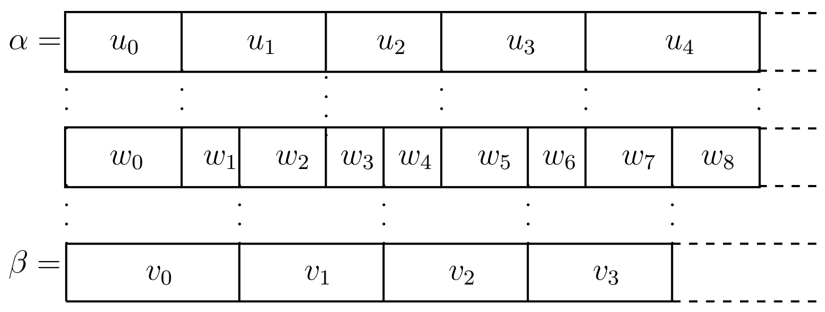

Fig. 1. Construction for Theorem 20

Corollary 21 *Let $\sim$ be a flexible congruence relation generated by a sequential relation. The closure problem for the limit extension of $\sim$ with respect to languages represented by Büchi automata is in PSPACE.*

Since the finite version of each concurrency relation is a left-cancellative congruence, and the infinite version of each concurrency relation is the limit extensions of its finite version, the previous corollary allows us to state:

Theorem 22 *The closure problem for trace, stutter and projective equivalence with respect to languages represented by Büchi automata is in PSPACE.*

It is now straightforward to adapt the proof of Theorem 11 to obtain:

Theorem 23 *The closure problem for trace, stutter and projective equivalence with respect to languages represented by Büchi automata is PSPACE-hard.*

5 The Closure Problem for LTL Properties

We now focus on the closure problem for languages defined by LTL formulas. At first glance, this problem looks harder than the closure problem for languages represented by Büchi automata. For there is an inherent exponential explosion in the conversion from LTL to Büchi automata [29]. However, we show that for the class of equivalence relations we are interested in, the problem is still in PSPACE, even when the property is described using an LTL formula.

Notice that a propositional LTL formula φ uses a set of propositions Γ and that the corresponding alphabet of the language described by φ is $\Sigma = 2^{\Gamma}$. Notice also that projective closedness is defined with respect to a partition of Γ into two sets of propositions, Γ_1 and Γ_2; the alphabets Σ_1 and Σ_2 are 2^{Γ_1} and 2^{Γ_2}, respectively, and every $c \in \Sigma$ is identified with the letter $(c \cap \Gamma_1, c \cap \Gamma_2)$ from $\Sigma_1 \times \Sigma_2$ (see [13]).

Theorem 24 *Let $\sim$ be a congruence relation on Σ^* that is generated by a* *sequential relation. The closure problem for $\sim^{\omega *}$ with respect to languages represented by LTL formulas is in PSPACE.*

PROOF. Recall from [29] that every LTL formula φ can be translated into an equivalent Büchi automaton of size $\mathcal{O}(2^{|\varphi|})$, each state of which can be represented in space $\mathcal{O}(|\varphi|)$.

Let φ be a formula and L the language defined by φ. In order to determine whether L is $\sim^{\omega *}$-closed we can proceed as in the previous section: we only have to check (5). To do this we can follow the steps in the proof of Theorem 16. There are, however, two modifications to be made: first, A_1 is constructed from φ using the construction from [29]; second, A_2 is constructed from $\neg\varphi$ using the construction from [29]. Each state of the product automaton will be representable in space $\mathcal{O}(|\varphi| + |\varphi|)$, which is identical with $\mathcal{O}(|\varphi|)$. The test for emptiness (as explained at the end of the proof of Theorem 16) can thus be implemented in polynomial space. $\square$

Theorem 25 *The closure problem for trace, stutter, and projective equivalence (over non-trivial alphabets) with respect to languages defined by LTL formulas is PSPACE-hard.*

PROOF. Let $\sim$ be an arbitrary non-trivial concurrency relation over Σ. We will reduce the satisfiability problem for LTL formulas over Σ to the complement of the closure problem for $\sim$. This is enough, for LTL satisfiability is PSPACE-complete [20] (for any non-trivial alphabet) and PSPACE is closed under complementation.

Since $\sim$ is assumed to be a non-trivial concurrency relation, there exist $a, b \in \Sigma$ such that for every language $L \subseteq \Sigma^*$, the language abL is $\sim$-closed iff $L = \emptyset$. (If $\sim$ denotes projective equivalence or stutter equivalence, it is enough to choose a and b distinct; if $\sim$ denotes trace equivalence, one has to choose a and b independent.)

Let ψ be the formula

$$\psi = \bigwedge_{p \in a} p \wedge \bigwedge_{p \in \Gamma \setminus a} \neg p \wedge \bigcirc \left(\bigwedge_{p \in b} p \wedge \bigwedge_{p \in \Gamma \setminus b} \neg p \right),$$

which defines $ab\Sigma^*$.

Consider the function that maps a given LTL formula φ to $\eta = \bigcirc \bigcirc \varphi \wedge \psi$. Clearly, this mapping is computable in polynomial time. Thus to conclude the proof, we only have to show that φ is satisfiable iff the set defined by η is not

$\sim$-closed. But this is trivial, because by construction η defines the language abL where L denotes the language defined by ψ. □

6 Applications

6.1 The Context: Model Checking and Partial-Order Methods

Partial-order reduction methods is a generic name for a family of algorithms for generating a reduced state space of a concurrent program [26,4,30,15]. They are based on a modified depth-first search, where at each state in the search only a subset of the transitions that can be taken (i. e., are *enabled*) are chosen. The main observation behind these algorithms is that for most purposes, there is no need to distinguish between program execution sequences that are trace equivalent. Hence, a state space that includes at least one sequence per equivalence class can replace the full state space of a program.

However, the reduced state spaces produced by partial-order methods cannot be used without further precautions for model checking specifications given as temporal logic formulas or as Büchi automata. Indeed these formalisms can express properties that do distinguish between sequences that are trace equivalent. The approaches proposed so far to solve this problem consist of considering more transitions as being dependent, and hence reducing the size of the trace equivalence classes. Concretely, correctness is ensured by being pessimistic about which transitions need to considered dependent, e.g., one adds dependencies among all transitions that can potentially affect the truth of the checked property [26], or among subsets of such transitions, after applying some LTL rewriting rules [15]. Of course, this has the negative effect of substantially limiting the reduction of the size of the state space that can be achieved by partial-order methods.

The results of this paper offer an interesting alternative: check that the formula to be verified is trace closed and use the partial-order technique without any additional dependencies. Note that one can expect a well specified property to be trace closed. If the property nevertheless turns out not to be trace closed and really should be checked as such, one can use our algorithm to guide the partial-order reduction algorithms as to which dependencies need to be added in order for the property to be checked reliably. Indeed, when a property is not trace closed, our algorithm produces a pair of independent actions whose permutation causes a sequence satisfying the property to become one that no longer satisfies it. By adding this pair of actions to the dependency relation and repeating the procedure until the property is trace closed, one obtains a minimal dependency relation for which partial-order methods check the property reliably.

Checking for stutter equivalence is also important for similar applications. For instance, in the partial-order reduction methods of [26,15], the reduction algorithm guarantees to generate at least one execution sequence from each stutter equivalence class. Hence, it is only usable for stutter-closed formulas. In [26,15], this condition is enforced by restricting oneself to LTL formulas not containing the next-time operator. The decision procedure we have developed in this paper offers a more flexible alternative. Similarly, the reduction method of [14] guarantees to generate at least one execution sequence for each trace equivalence class and thus can benefit from a decision procedure that can check whether the specification is trace closed.

6.2 Matching States and Transitions

We now turn to the problem of applying our decision procedure for checking closure properties in the context of model checking. The difficulty is that the decision procedures we have given assume that both the specification and the equivalence relation for which closure is checked are expressed in terms of the same set of atomic actions. However, in the context of model checking, it is common to have a specification expressed in terms of Boolean propositions interpreted in *states* of the program whereas some equivalences among computations, e.g., trace equivalence, are expressed in terms of *transitions* of the program. We thus need to adapt our decision procedure to take this into account.

Concretely, we have an LTL specification φ built over a set of atomic propositions Γ as well as a program P. The program P is defined by a set of states S, a set of transitions $\Delta \subseteq S \times S$, and an interpretation function $v : S \to \Gamma$. The problem is to determine whether the property φ is closed for P, i.e. whether there are, or not, two sequences of transitions from Δ that are equivalent with respect to $\sim^{\omega *}$ ($\overset{1}{\sim}$ is a given sequential relation on transition sequences) and that generate sequences of states for which the LTL formula φ has different values.

Unfortunately, solving this problem exactly requires exploring the state space of P since the possible effect of transitions on the truth values of state propositions can depend on which states are actually reachable. However, exploring the state space of P would defeat the practical purpose of checking for closure, which is precisely to allow the computation of a reduced state space. We thus turn to an approximate solution and use a nondeterministic representation of the relation between the sequence of transitions and the truth values of the propositions in Γ. Concretely, for each transition $\tau \in \Delta$, we extract from the program text a relation $\delta(\tau) \subseteq 2^\Gamma \times 2^\Gamma$ that represents all possible ways in which the transition τ can affect the truth values of the propositions in Γ. Furthermore, we say that a sequence $\sigma \in (2^\Gamma)^\omega$ *conforms* with a sequence of transitions $\rho \in \Delta^\omega$ iff for each $i > 0$, $(\sigma(i-1), \sigma(i)) \in \delta(\rho(i))$. So, in practice we will check whether there are two sequences of transitions from Δ equivalent with respect to $\sim^{\omega *}$ and for which there are conforming sequences from $(2^\Gamma)^\omega$ on which the LTL formula φ has different values. We will call this notion *conformance closure*. Notice that checking for conformance closure yields a potentially pessimistic result since sequences conforming to a transition sequence are not necessarily possible. The advantage is that conformance closure avoids any additional complexity linked to relating states and transitions as shown by the following Theorem.

Theorem 26 *Let Δ be the set of transitions of a program P. Let $\sim$ be a congruence relation on Δ^* that is generated by a non-trivial sequential relation $\overset{1}{\sim}$. The conformance closure problem for $\sim^{\omega *}$ with respect to a property expressed by a temporal logic formula φ is in PSPACE.*

PROOF. We are thus given a temporal logic formula φ over a set of propositions Γ and a program P, described by its set of transitions Δ as well as the relations $\delta(\tau) \subseteq 2^\Gamma \times 2^\Gamma$ describing the effect of the transitions on the propositions in Γ.

To check for conformance closure, we build an automaton operating on infinite words over the alphabet $2^\Gamma \times \Delta \times \Delta \times 2^\Gamma$. An infinite word over this alphabet can be viewed as a quadruple $\mathrm{w} = (\sigma_1, \rho_1, \rho_2, \sigma_2)$ where $\sigma_1, \sigma_2 \in (2^\Gamma)^\omega$ and $\rho_1, \rho_2 \in \Delta^\omega$. The automaton we build is obtained by taking the product of the five following components.

(1) An automaton checking that $\sigma_1 \in L(\varphi)$ where $L(\varphi)$ is the set of sequences satisfying the formula φ.
(2) An automaton checking that $\sigma_2 \in \overline{L(\varphi)}$.
(3) an automaton that checks that the input can be decomposed into infinitely many factors that are all elements of $\overset{1}{\sim}_\$$.
(4) An automaton that checks that ρ_1 conforms with σ_1.
(5) An automaton that checks that ρ_2 conforms with σ_2.

The automata (1) and (2) are exponential in the size of φ but can be built and explored in PSPACE (see Theorem 16). The automaton (3) is of size linear in the size of the automaton defining $\overset{1}{\sim}_\$$. The automata (4) and (5) are of size $|\Delta| \times 2^{(2|\Gamma|)}$, but can be built and explored using space polynomial in the size of Δ and Γ.

Note that, for trace equivalence, conformance closure is in fact PSPACE-complete. Indeed, the hardness follows from the similar result established in Theorem 25.

7 Conclusions

Being trace closed or stutter closed is a natural property of specifications for concurrent systems. Yet, many specification languages can specify properties that violate it, e.g., LTL and finite automata recognizable languages. We have proposed an algorithm for a family of equivalence relations, including trace, stutter and projective equivalence, which decides closedness for regular and ω-regular languages, and LTL specifications. This allows exploiting the simplicity of such languages, while using the decision procedure to restrict the specifications to closed ones.

Furthermore, it is perfectly realistic to expect to be able to use our algorithm in the context of verification tools that use partial-order state space reductions. Indeed, it involves no other construction than those routinely used in model-checking, and has the same complexity in terms of the LTL formula, namely PSPACE-complete.

Acknowledgments

We are grateful to Anca Muscholl for pointing out a bug in the proof of Theorem 20 in the conference version [17] of this paper, and to P. S. Thiagarajan for sending us a preprint of [23].

References

[1] Alur, R., Peled, D., Penczek, W.: Model-checking of causality properties. In Proc. 10th IEEE Symposium on Logic in Computer Science, San Diego, California (1995) 90–100.

[2] Arnold, A.: A syntactic congruence for rational ω-languages. Theoretical Computer Science **39** (1985) 333–335.

[3] Diekert, V., Gastin, P., Petit, A.: Rational and recognizable trace languages. Information and Computation **116** (1995) 134–153.

[4] Godefroid, P.: Using partial orders to improve automatic verification methods. In Proc. 2nd Workshop on Computer Aided Verification, New Brunswick, NJ. Lect. Notes in Comput. Sci., vol. 531, Springer (1990) 176–185.

[5] Katz, S., Peled, D.: Verification of distributed programs using representative interleaving sequences. Distributed Computing **6** (1992) 107–120.

[6] Kwiatkowska, M. Z.: Event fairness and non-interleaving concurrency. Formal Aspects of Computing **1** (1989) 213–228.

[7] Kozen, D.: Lower bounds for natural proof systems. 18th IEEE Symposium on Foundations of Computer Science, Providence, Rhode Island (1977) 254–266.

[8] Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Transactions on Computers **28** (1979) 690–691.

[9] Lamport, L.: What good is temporal logic? In Proc. IFIP Congr. on Information Processing, Elsevier (1983) 657–668.

[10] Mazurkiewicz, A.: Trace theory. In Proc. Advances in Petri Nets 1986, Bad Honnef, Germany. Lect. Notes in Comput. Sci., vol. 255, Springer (1987) 279–324.

[11] Muscholl, A.: Über die Erkennbarkeit unendlicher Spuren, doctoral thesis, Stuttgart University, 1994. Appeared also as vol. 17 of Teubner-Texte zur Informatik, Teubner, Stuttgart, Leipzig, 1996.

[12] Pécuchet, J.-P.: Etude Syntaxique des parties reconnaissable de mots infinis. In Automata, Languages, and Programming: 13th Intern. Coll. Rennes, France. Lect. Notes in Comput. Sci., vol. 226, Springer (1986) 294–303.

[13] Peled, D.: On projective and separable properties. In Proc. Colloquium on Trees in Algebra and Programming, Edinburgh, Scotland. Lect. Notes in Comput. Sci., vol. 787, Springer (1994) 291–307.

[14] Peled, D.: All from One, One from All: on Model Checking using representatives, In Proc. 5th International Conference on Computer Aided Verification, Elounda, Greece, Lect. Notes in Comput. Sci., vol. 697, Springer (1993) 409–423.

[15] Peled, D.: Combining partial-order reductions with on-the-fly model-checking. Formal Methods in System Design **8** (1996) 39–64.

[16] Peled, D., Pnueli, A.: Proving partial-order properties. Theoretical Computer Science **126** (1994) 143–182.

[17] Peled, D., Wilke Th., Wolper P.: An Algorithmic Approach for Checking Closure Properties of ω-Regular Languages. CONCUR'96, 7th International Conference on Concurrency Theory, 1996, Pisa, Italy, Springer Verlag, Lect. Notes in Comput. Sci., vol. 1119, Springer (1996) 596–610.

[18] Pnueli, A.: The temporal logic of programs. In Proc. 18th IEEE Symposium on Foundation of Computer Science, Providence, Rhode Island (1977) 46–57.

[19] Safra, S.: On the complexity of ω-automata. In Proc. 29th Annual Symposium on Foundations of Computer Science, White Plains, New York (1988) 319–327.

[20] Sistla, A. P., Clarke, E. M.: The complexity of propositional linear temporal logics. Journal of the ACM **32** (1985) 733–749.

[21] Sistla, A. P., Vardi, M. Y., Wolper, P.: The complementation problem for Büchi automata with applications to temporal logic. Theoretical Computer Science **49** (1987) 217–237.

[22] Thiagarajan, P. S.: A trace based extension of linear time temporal logic. In Proc. 10th IEEE Symposium on Logic in Computer Science, Paris, France (1994) 438–447.

[23] Thiagarajan, P. S., Walukiewicz, I.: An Expressively Complete Linear Time Temporal Logic for Mazurkiewicz Traces. In Proc. 13th IEEE Symposium on Logic in Computer Science, Warsaw, Poland (1997). To appear.

[24] Thomas, W.: Automata and quantifier hierarchies: formal properties of finite automata and applications. In Proc. of LITP Spring School on Theoretical Computer Science, J. E. Pin, ed. Lect. Notes in Comput. Sci., vol. 386, Springer (1989) 104–119.

[25] Thomas, W.: Automata on infinite objects. In Handbook of Theoretical Computer Science, vol. B, J. van Leeuwen, ed., Elsevier, Amsterdam (1990) 133–191.

[26] Valmari, A.: A stubborn attack on state explosion. Formal Methods in System Design **1** (1992) 297–322.

[27] Vardi, M. Y., Wolper, P.: Automata-theoretic techniques for modal logics of programs. J. Comput. System Sci. **32** (1986) 182–221.

[28] Vardi, M. Y., Wolper, P.: Reasoning about infinite computations. Information and Computation **115** (1994) 1–37.

[29] Wolper, P.: Temporal logic can be more expressive. Information and Control **56** (1983) 72–99.

[30] Wolper, P., Godefroid, P.: Partial-order methods for temporal verification. In Proc. CONCUR, 4th Conference on Concurrency Theory, Hildesheim, Germany. Lect. Notes in Comput. Sci., vol. 715, Springer (1993) 233–246.