Appears in: CONCUR '95: Concurrency Theory, 7th Int. Conference, Pisa, Italy, Lecture Notes in Computer Science, Springer-Verlag, August, 1996.

An Algorithmic Approach for Checking Closure Properties of ω-Regular Languages

Doron Peled[1] and Thomas Wilke[2*] and Pierre Wolper[3]

[1] Bell Laboratories, 700 Mountain Avenue, Murray Hill, NJ 07974, USA
doron@research.att.com
[2] DIMACS, Rutgers University, Piscataway, NJ 08855-1179, USA
wilke@dimacs.rutgers.edu
[3] Université de Liège, Institut Montefiore, B28, 4000 Liège, Belgium
pw@montefiore.ulg.ac.be

Abstract. 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 formulae and show that for such properties there is a wide class of equivalence relations for which determining closure is decidable, in fact 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 *inde-*

* This author is supported by a DIMACS postdoctoral fellowship funded by NSF award 91-19999 and the New Jersey Commission on Science and Technology.

pendent, 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 [16] (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, 15], TrPTL [19] and TLC [1]. None of these logics is completely satisfying: they lack simplicity and/or cannot express all relevant properties. A full practical and complete solution to the problem has not 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 [12] 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* [22, 4, 26, 13, 14]. 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 equivalence class for trace equivalence [13] or stuttering equivalence [22, 14], 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. Recognizing projective closedness can also be used for improving the throughput of partial-order reduction [13, 12]. Projective properties are also preserved by sequential consistent [8] implementations of cache protocols [12].

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 [12]. 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 [23]. In the case of regular languages, we only assume that the equivalence relation is generated by a sequential relation [20], 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 10) 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 formulae, with the matching lower bound still holding for each of the concrete relations we consider.

The case of trace equivalence for ω-regular languages was also dealt with in [3]. There, 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.

2 The Concurrency Relations

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

Throughout this paper, Σ stands for a finite alphabet and Σ^∞ for $\Sigma^* \cup \Sigma^\omega$. The prefix order on Σ^∞ is denoted by $\sqsubset$.

Definition 1 (limit extension). The *limit extension* $\sim^{\lim} \subseteq \Sigma^\omega \times \Sigma^\omega$ of an equivalence relation $\sim \subseteq \Sigma^* \times \Sigma^*$ is defined as follows: $\alpha \sim^{\lim} \beta$ holds iff

- 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 $\sim^{\lim}$ is an equivalence relation on Σ^ω.*

Proof. The relation $\sim^{\lim}$ is a binary relation on Σ^ω, which is obviously reflexive and symmetric. Assume $\alpha \sim^{\lim} \beta \sim^{\lim} \gamma$ and let $u \sqsubset \alpha$. By definition of $\sim^{\lim}$, there exist $v \sqsubset \beta$ and v' such that $uv' \sim v$. On the other hand, since $\beta \sim^{\lim} \gamma$, 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 $\alpha \sim^{\lim} \gamma$. So $\sim^{\lim}$ is also transitive and thus an equivalence relation. $\square$

2.2 Finite and Infinite Traces

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

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

As in [6], the infinite words α and β are said to be *trace equivalent* if $\alpha \equiv^{\lim} \beta$.

2.3 Stutter and Projective 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 characters by a single copy of the character. For instance, $\natural(aabbbcaa) = abca$ and $\natural(aabbbc^\omega) = abc^\omega$. Words x and y are said to be *stutter equivalent* if $\natural x = \natural y$.

Let Σ_1 and Σ_2 be finite alphabets and $\Sigma = \Sigma_1 \times \Sigma_2$. The definitions below can be easily generalized to an n-component alphabet $\Sigma = \Sigma_1 \times \Sigma_2 \times \ldots \times \Sigma_n$. 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_1a_2, b_1b_2).

For each pair $c = (a, b) \in \Sigma$, let $c|_1 = a$ and $c|_2 = b$. For each word $x = c_1c_2c_3 \ldots \in \Sigma^\infty$, define the *projection* $x|_i$ $(i \in \{1, 2\})$ of x on the ith component to be $\natural(c_1|_ic_2|_ic_3|_i \ldots)$.

Notice that the projection operator also removes stuttering. Consider, for example, the word $u = (a_1, b_3)(a_1, b_5)(a_2, b_5)(a_9, b_5)$. Then $u|_1 = a_1a_2a_9$, and $u|_2 = b_3b_5$.

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

The following Lemma can be proved quite easily from the definitions of stutter and projective equivalence.

Lemma 3. *Both stutter and projective equivalence over infinite words are limit extensions of the corresponding equivalence relation over finite words.*

2.4 The Closure Problem

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

Given an alphabet Σ, a class $\mathcal{L}$ of languages of finite or infinite words over Σ, and an equivalence relation $\sim$ on, respectively, Σ^* or 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 properties. Recall that every LTL definable property is ω-regular [25].

3 The Closure Problem for Finite Words

We will prove that the closure problem is decidable for the set of regular languages of finite words over a given alphabet Σ and every equivalence relation that is generated by a sequential relation, i. e., a relation recognizable by a finite automaton in the sense we define below. In fact, we will show that, under these assumptions, the closure problem is in PSPACE. The finite version of all the concurrency relations we introduced in the previous section fit these assumptions. In addition, we show that, for these relations, the closure problems are PSPACE-hard and hence obtain a tight characterization of their complexity.

Let \$ be a symbol 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 symbol \$, not to be confused with the projection defined in Sect. 2). 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 4 [20]. A binary relation R on Σ^* is said to be *sequential* if $R_\$$ (viewed as a language) 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 $\overset{1}{\sim}$, that is, if $\sim$ is the smallest equivalence relation containing $\overset{1}{\sim}$.

The proof of the following lemma is straightforward:

Lemma 5. *Let $L \subseteq \Sigma^*$ and $\sim$ an equivalence relation on Σ^* generated by a symmetric relation $\stackrel{1}{\sim}$. Then L is not $\sim$-closed iff there exist words $u, v \in \Sigma^*$ such that $u \stackrel{1}{\sim} v$, $u \in L$, and $v \notin L$.*

For regular languages, it is immediate to extract from Lemma 5 a decision procedure that checks whether for a fixed equivalence relation $\sim$, generated by a sequential relation $\stackrel{1}{\sim}$, a given regular language L is $\sim$-closed. We assume the regular languages L are given by finite automata A, and, w.l.o.g., that $\stackrel{1}{\sim}$ is symmetric. (Note that the symmetric closure of a sequential relation is itself sequential). We build an automaton B accepting $R_\$$, where R is the relation containing all the pairs (u, v) that satisfy exactly the conditions of Lemma 5. The automaton B can be obtained as the product of three automata:

1. an automaton that checks that the first component of a given input belongs to $L\,\$^*$;
2. an automaton that checks that the second component of a given input does not belong to $L\,\$^*$; and
3. an automaton that checks that a given input belongs to $\stackrel{1}{\sim}_\$$ (viewed as a language).

If the automaton A is deterministic, the first and the second automaton can be chosen to be of size $O(|A|)$. If A is non-deterministic, the second automaton has size $O(2^{|A|})$. The third automaton is the automaton that recognizes $\stackrel{1}{\sim}_\$$ and has a fixed size.

Given that the non-emptiness problem for finite automata can be checked in linear time, we have the following results:

Theorem 6. *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, where $|A|$ stands for the size of an input automaton A.*

1. *The closure problem is decidable in time $O(|A|)$ for deterministic automata.*
2. *The closure problem is decidable in time $O(2^{|A|})$ for non-deterministic automata.*

An automaton for the complement of a language given by a non-deterministic automaton can be exponentially bigger. However, it need not be constructed to check the emptiness of the automaton B described above. We actually have:

Theorem 7. *Let $\sim$ be an equivalence relation generated by a sequential relation. The closure problem for $\sim$ and languages represented by non-deterministic finite automata is in PSPACE.*

Sketch of Proof. What pushes the complexity of our procedure up is that we have to check for nonacceptance by the automaton describing the language. However, it is well known that checking for the existence of a word not accepted by a non-deterministic automaton can be done in PSPACE. Basically, one does a non-deterministic search through the state-space of the deterministic version of the automaton. Since all one needs to remember is the current state, the procedure is in NPSPACE and hence in PSPACE. This procedure can easily be combined with a search through the other components of our automaton, yielding an overall PSPACE procedure. □

Returning to the concurrency equivalence 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 $\stackrel{1}{\equiv}$. Stutter equivalence is generated by the sequential relation $\{(uav, uaav) \mid u, v \in \Sigma^*,\ a \in \Sigma\}$. Projective equivalence is generated by the sequential relation that relates

- $u(a, b)v$ with $u(a, b)(a, b)v$,
- $u(a_1, b_1)(a_1, b_2)(a_2, b_3)v$ with $u(a_1, b_1)(a_2, b_2)(a_2, b_3)v$, and
- $u(a_1, b_1)(a_2, b_1)(a_3, b_2)v$ 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 $b_1, b_2, b_3 \in \Sigma_2$.

To sum up:

Corollary 8. *The closure problem for trace, stutter, and projective equivalence with respect to languages represented by non-deterministic finite automata is in PSPACE.*

Next, we will complement this result by proving a matching lower bound.

Theorem 9. *The closure problem for trace, stutter, and projective equivalence with respect to languages represented by non-deterministic finite automata is PSPACE-hard.*

Sketch of Proof. For every language $L \subseteq \Gamma^*$ accepted by a deterministic polynomial space Turing machine M, we can construct a polynomial time computable function $f\colon x \mapsto A_x$ that associates with every $x \in \Gamma^*$ a finite automaton A_x over Σ such that

- if $x \notin L$ then $L(A_x) = \Sigma^*$, and
- if $x \in L$ then $L(A_x) = \Sigma^* \setminus \{u\}$ for some $u \in ab\Sigma^*$. (The word u encodes the accepting computation of M on x, prefixed by ab).

This construction is similar to the one given in [7]. If we choose $(a, b) \notin D$, we have that $x \in L$ iff $L(A_x)$ is not trace closed. Similarly, we have that $x \in L$ iff $L(A_x)$ is not stutter-closed, or is not projective. This means that the complements of the problems in question are PSPACE-hard, and, hence, that so are the problems themselves. □

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 set $(ab)^*$. In this case, the $\sim$-closure of L is the set of strings which have as many occurrences of a as occurrences of b. But this set is not regular.

4 The Closure Problem for Infinite Words

We prove that the closure problem is decidable for the set of regular languages of infinite words for a large class of equivalence relations. These relations are "piecewise extensions" (see next definition) of relations that are generated by sequential relations. In fact, we show that the closure problem is in this case in PSPACE. The infinite versions of the concurrency relations from Sect. 2 are all particular instances of this.

Definition 10 (piecewise extension). Let $\sim$ be a binary relation on Σ^* (not necessarily an equivalence relation). We define the relations $\sim^{\omega}$ and $\sim^{\omega *}$ on Σ^{ω}.

We write $\alpha \sim^{\omega} \beta$ if there exist decompositions $\alpha = u_0 u_1 u_2 \ldots$ and $\beta = v_0 v_1 v_2 \ldots$ such that $u_i \sim v_i$ for every $i \geq 0$. The relation $\sim^{\omega *}$ is the transitive closure of $\sim^{\omega}$, and it is called the *piecewise extension* of $\sim$.

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

We denote the *syntactic congruence of a language* $L \subseteq \Sigma^{\omega}$ by $\approx_L$ (see [2]). It is a relation on finite words that is defined as follows: $u \approx_L v$ holds iff

- $xuyz^{\omega} \in L \leftrightarrow xvyz^{\omega} \in L$, for $x, y \in \Sigma^*$ and $z \in \Sigma^+$, and
- $x(uy)^{\omega} \in L \leftrightarrow x(vy)^{\omega} \in L$, for $x, y \in \Sigma^*$ with $uy \neq \varepsilon$ and $vy \neq \varepsilon$.

We will use the following property of the syntactic congruence, which is an immediate consequence of Lemma 2.2 in [2] and Proposition 2.3 in [11].

Proposition 11 [2, 11]. *Let $\sim$ be a congruence relation on Σ^* and $L \subseteq \Sigma^{\omega}$ an ω-regular language. Then L is closed under $\sim^{\omega *}$ iff $\sim \subseteq \approx_L$.*

We have:

Theorem 12. *Let $\sim$ be a congruence relation on Σ^* generated by a reflexive, symmetric relation $\stackrel{1}{\sim}$ and $L \subseteq \Sigma^{\omega}$ an ω-regular language. Then L is closed under $\sim^{\omega *}$ iff L is closed under $\stackrel{1}{\sim}^{\omega}$.*

Proof. For the non-trivial direction, assume L is not closed under $\sim^{\omega *}$. Then, according to Proposition 11, the syntactic congruence $\approx_L$ of L is not saturated by $\sim$, i.e., there exists an equivalence class of $\approx_L$ that is not closed under $\sim$. Hence, by Lemma 5, there are finite words u, v such that $u \stackrel{1}{\sim} v$ 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 $x(vy)^\omega \notin L$, where $uy \neq \varepsilon$ and $vy \neq \varepsilon$.

But $xuyz^\omega \stackrel{1}{\sim}^\omega xvyz^\omega$, as well as $x(uy)^\omega \stackrel{1}{\sim}^\omega x(vy)^\omega$. Thus, L is not $\stackrel{1}{\sim}^\omega$-closed. $\square$

It follows from Theorem 12 that an ω-regular language L is closed under $\sim^{\omega *}$ iff there exists no word $(\alpha, \beta) \in (\Sigma_\$ \times \Sigma_\$)^\omega$ such that $\alpha \downarrow \Sigma \in L$, $\beta \downarrow \Sigma \notin L$, and $(\alpha, \beta) \in (\stackrel{1}{\sim}_\$)^\omega$, where $(\stackrel{1}{\sim}_\$)^\omega$ stands for the ω iteration of the relation $\stackrel{1}{\sim}_\$$, viewed as a language of finite words.

We can now use an algorithm similar to the one in Sect. 3 to solve the closure problem for regular ω-languages and the piecewise extension $\sim^{\omega *}$ of a congruence relation $\sim$ that is generated by a sequential relation $\stackrel{1}{\sim}$. W.l.o.g., we can assume that $\stackrel{1}{\sim}$ is reflexive and symmetric, since the reflexive, symmetric closure of a sequential relation is, again, sequential.

One has to check the emptiness of an automaton obtained as the product of the following three automata:

1. an automaton that checks that the canonic projection on Σ of the first component of the input is in L;
2. an automaton that checks that the canonic projection on Σ of the second component of the input is not in L;
3. an automaton that checks that the input can be decomposed into infinitely many factors that are all elements of $\stackrel{1}{\sim}_\$$ (when viewed as a language).

The first automaton can be directly obtained from a Büchi automaton A for a given language L and has size $O(|A|)$; the second automaton can be constructed in the same way from a Büchi automaton for the complement of L. Using the results of [18], this yields the following complexity result.

Theorem 13. *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.*

Sketch of Proof. Similarly to the proof of Theorem 7, we use the fact that checking nonacceptance by a Büchi automaton can be done in PSPACE [18, 24]. $\square$

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

We first mention the following straightforward statement.

Lemma 14. *If $\sim$ is a congruence relation, then $\sim^{\omega *} \subseteq \sim^{\lim}$.*

A congruence relation $\sim$ is called *left cancellative* if $v \sim w$ whenever $uv \sim uw$. Trace, stutter, and projective equivalences are easily shown to be left cancellative. The following theorem, which is a generalization of a result concerning trace equivalence in [3], gives the converse of Lemma 14 for left cancellative congruences.

Theorem 15. *Let $\sim$ be a left cancellative congruence relation. Then $\sim^{\lim} = \sim^{\omega *}$.*

Proof. In view of Lemma 14, we only have to show $\sim^{\lim} \subseteq \sim^{\omega *}$.

Assume $\alpha \sim^{\lim} \beta$. We will 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

$$v_j \sim w_{2j} w_{2j+1}, \qquad \text{and} \tag{1}$$
$$u_j \sim w_{2j-1} w_{2j}, \qquad \text{for } j \geq 0, \tag{2}$$

where w_{-1} is the empty word ε. This implies $\alpha \sim^{\omega} w_0 w_1 w_2 \ldots \sim^{\omega} \beta$, which means $\alpha \sim^{\omega *} \beta$.

We will give an inductive definition for the u_i, v_i, and w_i. In step 0, we choose an arbitrary non-empty finite prefix $u_0 \sqsubset \alpha$, and set $w_0 = u_0$. In step $2i+1$, for $i \geq 0$, we will define v_i and w_{2i+1}; and in step $2i$, for $i \geq 1$, we will define u_i and w_{2i}.

We assume that before each step, (1) and (2) hold for words thus far defined; we have to make sure that the words to be defined in a new step don't lead to a violation of (1) or (2).

We describe only the steps $2i+1$; the description for the others is symmetric. Since (1) and (2) hold for words thus far defined, we have $u_0 u_1 \ldots u_i \sim w_0 w_1 \ldots w_{2i}$, where $u_0 u_1 \ldots u_i$ is a prefix of α, and $v_0 v_1 \ldots v_{i-1} \sim w_0 w_1 \ldots w_{2i-1}$, where $v_0 v_1 \ldots v_{i-1}$ is a prefix of β. 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 $u_0 u_1 \ldots u_i z \sim y$. We choose y to be long enough such that $v_0 v_1 \ldots v_{i-1}$ is a proper prefix of it. Let $x \in \Sigma^+$ be the string such that $y = v_0 v_1 \ldots v_{n-1} x$. Hence $w_0 w_1 \ldots w_{2i} z \sim u_0 u_1 \ldots u_i z \sim v_0 v_1 \ldots v_{i-1} x \sim w_0 w_1 \ldots w_{2i-1} x$. Set $v_i = x$ and $w_{2i+1} = z$. Then, since $\sim$ is cancellative, $v_i = x \sim w_{2i} w_{2i+1}$, i.e., (1) holds for $j = i$. $\square$

From Theorems 13 and 15 we have:

Corollary 16. *Let $\sim$ be a left cancellative 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 17. *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 9 to obtain:

Theorem 18. *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 focus now on the closure problem for the concurrency relations with respect to languages defined by LTL formulae. At first glance, it looks like this problem is harder than the closure problem for languages represented by Büchi automata. The reason is that there is an inherent exponential explosion in the conversion from LTL to Büchi automata [25]. 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 Γ_1 and Γ_2 of propositions; 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 [12]).

Theorem 19. *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 formulae is in PSPACE.*

Sketch of Proof. LTL formulae can be translated to Büchi automata [25]. The translation incurs an exponential blowup. However, as we did in the case of the automaton for the complement language in the proof of Theorem 13, it is not necessary to construct the automaton for a formula φ. Instead, one can conduct a search through its states, provided that each state and the successor relation between states can be calculated in time polynomial in the size of φ. Moreover, the complement of the automaton for φ is merely an automaton for $\neg\varphi$, eliminating the doubly exponential blowup in translating and complementing. $\square$

Theorem 20. *The closure problem for trace, stutter, and projective equivalence with respect to languages defined by LTL formulae is PSPACE-hard.*

Proof. Let $\sim$ be an arbitrary non-trivial concurrency relation. We will reduce the satisfiability problem for LTL formulas to the complement of the closure problem for $\sim$. Recall that LTL satisfiability is PSPACE-complete [17] and that PSPACE is closed under complementation.

As explained in the sketch of the proof of Theorem 9, there exist $a, b \in \Sigma$ such that $ab\Sigma^\omega$ is not $\sim$-closed. In fact, for such a and b and an arbitrary non-empty set $L \subseteq \Sigma^\omega$, abL is not $\sim$-closed.

Let ψ be a formula that defines $ab\Sigma^\omega$. Take, for instance,

$$\psi = \bigwedge_{p \in a} p \wedge \bigwedge_{p \in \Gamma \setminus a} \neg p \wedge \bigcirc\Big(\bigwedge_{p \in b} p \wedge \bigwedge_{p \in \Gamma \setminus b} \neg p\Big).$$

Consider the function that maps a given LTL formula ϕ to $\eta = \bigcirc\bigcirc\phi \wedge \psi$. 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.

Assume first ϕ is not satisfiable. Then $\bigcirc\bigcirc\phi$ is not satisfiable, and thus η is not satisfiable. So the set defined by η, the empty set, is $\sim$-closed. Assume now that ϕ is satisfiable, and let L be the set of ω-words that satisfy ϕ. Then abL is the set defined by η, and, as we have argued above, η is thus not $\sim$-closed. $\square$

Notice that the proof of Theorem 20 is general and can easily be adapted for a large class of equivalence relations.

6 Application of Recognizing Equivalence Closedness

6.1 Model-Checking Applications

Partial-order reduction methods is a generic name for a family of algorithms for generating a reduced state space of a concurrent program [22, 4, 26, 14]. These algorithms are used for model-checking properties of concurrent programs. 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 in these algorithms is that for most purposes, there is no need to distinguish between program execution sequences that are equivalent. Hence, a state space that includes at least one sequence per equivalence class can replace the full state space of a program.

However, most specification formalisms for sequences, e.g., temporal logics or Büchi automata, can specify properties that can distinguish between sequences that are trace equivalent. Although partial-order reductions are applicable also for such non-closed properties, the amount of reduction achieved typically degrades. The reason for this is that the correctness of the reduction is maintained by adding more dependencies to the system, forcing the property to become trace closed under a bigger dependency relation. These dependencies are usually added pessimistically, e.g., adding dependencies among all transitions that can potentially affect the truth of the checked property [22], or among subsets of such transitions, after applying some LTL rewriting rules [14].

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 the property to be checked will usually be trace closed. If the checked property is not closed, one can use our algorithm to guide the partial-order reduction algorithms as to which dependencies are needed to be added in order for the property to be checked reliably.

Checking for stutter equivalence is also important for similar applications. For instance, in the partial-order reduction methods of [22, 14], the reduction algorithm guarantees to generate at least one execution sequence from each stutter equivalence class. Hence, it is only usable for stutter-closed formulae. In [22, 14], this condition is enforced by restricting oneself to LTL formulae not containing the next-time operator. The decision procedure we have developed in this paper offers a more flexible alternative.

6.2 State-Based and Transition-Based Specifications

The above decision procedure, applied to trace equivalence, is based on a specification that expresses properties as sequences of atomic transitions. However, in most cases, the specification is given with respect to sequences of *states*, interpreted over a set of boolean *propositions*. We show how to change our decision procedure, when additional knowledge about the relation between the transitions and the propositions used in the specification is given.

Let Γ be a set of propositions. Each state is a subset of Γ. Each transition τ from a set of transitions Δ induces a relation $\delta(\tau) \subseteq 2^\Gamma \times 2^\Gamma$, which restricts the way the transition τ may change the propositions in Γ. We say that a sequence of transitions in $\rho \in \Delta^\omega$ *conforms* with a sequence of states $\sigma \in 2^{\Gamma^\omega}$ iff for each $i > 0$, $(\sigma(i-1), \sigma(i)) \in \delta(\rho(i))$. It is easy to see that the language $C \subseteq (\Delta \times 2^\Gamma)^\omega$ that recognizes pairs of conforming sequences is ω-regular.

Our modified decision procedure will check whether the checked property L can distinguish between sequences that conform with sequences of transitions that are trace equivalent.

We check the emptiness of the intersection of the following five automata over $2^\Gamma \times \Delta \times \Delta \times 2^\Gamma$. Each word of the automata is a quadruple $\mathrm{w} = (\sigma_1, \rho_1, \rho_2, \sigma_2)$. The five automata are as follows:

(1) Checks that the first word σ_1 is in L;
(2) Checks that the fourth word σ_2 is in the complement $\overline{L}$ of L;
(3) Checks that the pair (ρ_1, ρ_2) consists of infinitely many factors, such that each factor consists of a pair of words that are related by $\stackrel{1}{\equiv}$.
(4) Checks that ρ_1 conforms with σ_1.
(5) Checks that ρ_2 conforms with σ_2.

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.

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. 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.
12. 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.
13. 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.
14. Peled, D.: Combining partial-order reductions with on-the-fly model-checking. Formal Methods in System Design **8** (1996) 39–64.
15. Peled, D., Pnueli, A.: Proving partial-order properties. Theoretical Computer Science **126** (1994) 143–182.
16. Pnueli, A.: The temporal logic of programs. In Proc. 18th IEEE Symposium on Foundation of Computer Science, Providence, Rhode Island (1977) 46–57.
17. Sistla, A. P., Clarke, E. M.: The complexity of propositional linear temporal logics. Journal of the ACM **32** (1985) 733–749.
18. 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.
19. 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.
20. Thomas, W.: Automata and quantifier hierarchies: formal properties of finite automata and applications. In Proc. of LITP Spring School on Theoretical Com-

puter Science, J. E. Pin, ed. Lect. Notes in Comput. Sci., vol. 386, Springer (1989) 104–119.

21. Thomas, W.: Automata on infinite objects. In Handbook of Theoretical Computer Science, vol. B, J. van Leeuwen, ed., Elsevier, Amsterdam (1990) 133–191.
22. Valmari, A.: A stubborn attack on state explosion. Formal Methods in System Design **1** (1992) 297–322.
23. Vardi, M. Y., Wolper, P.: Automata-theoretic techniques for modal logics of programs. J. Comput. System Sci. **32** (1986) 182–221.
24. Vardi, M. Y., Wolper, P.: Reasoning about infinite computations. Information and Computation **115** (1994) 1–37.
25. Wolper, P.: Temporal logic can be more expressive. Information and Control **56** (1983) 72–99.
26. 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.