Checking Inherently Fair Linear-Time Properties in a Non-Naïve Way[⋆]

Thierry Nicola, Frank Nießner and Ulrich Ultes-Nitsche

Department of Computer Science, University of Fribourg
Boulevard de Pérolles 90, CH-1700 Fribourg, Switzerland

Abstract. In [9], an alternative verification relation for linear-time properties is introduced which uses an inherent fairness condition. That relation is specifically tailored to the verification of distributed systems under a relaxed version of strong fairness. We will call it the *inherently fair linear-time verification* relation in this paper, or IFLTV relation for short. We present an analysis of the mathematical structure of the IFLTV relation, which enables us to obtain an improved non-naïve procedure for checking the IFLTV relation.

1 Introduction

The results of [9] yield an immediate procedure for checking the satisfaction of the IFLTV relation on Büchi automata describing the behavior of a system and a linear-time property, respectively. We refer to that procedure as the *naïve* procedure. Even though the IFLTV relation is practically applicable as it comes with an abstraction concept based on so-called *weakly continuation-closed* abstractions [9], which can be computed efficiently from trace reductions of the automaton representing the behavior [10, 12], the naïve procedure for checking the IFLTV relation is sub-optimal and somehow limiting the applicability of the entire verification approach.

In this paper we analyze the IFLTV relation and apply the results to the structure of the involved automaton representations of system behavior and property. By doing so we define a procedure for checking IFLTV which requires fewer computation steps than the naïve one. This procedure is the main result of this paper and we will refer it as being the *non-naïve* procedure.

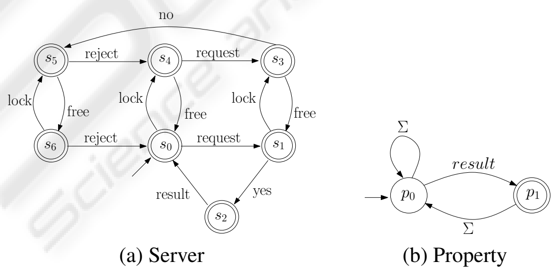
This paper is structured as follows: Sect. 2 gives a very brief motivation for the definition of the IFLTV relation; more motivating information can be found in [9]. After introducing some preliminary concepts in Sect. 3, we define in Sect. 4 the IFLTV relation formally. In Sect. 5, the IFLTV relation is analyzed to the extent that we can construct from it an IFLTV checking procedure for automaton representations of behavior and property in Sect. 6, analyzing its correctness in Sect. 7 and discussing its complexity in Sect. 8. Finally Sect. 9 concludes the paper.

⋆ Supported by the Swiss National Science Foundation under grant # 200021-103985/1 and by the Hasler Foundation under grant # 1922.

2 Motivation

Assume two systems, both randomly selecting initially an unbounded positive integer n. The first system will operate n steps and then stop. The second system will either operate n steps and stop, or may decide nondeterministically to operate forever. Only infinitely long observations could distinguish the two systems, which are apparently practically impossible. So, system one is as good as system two from the point of view of an outside observer. Linear-time verification, however, distinguishes the two systems as the second system does not satisfy the property 'always performing only finitely many operations' where the first one does. Besides being insensitive to such differences requiring infinite observations, IFLTV is as powerful as the usual linear-time verification relation. The two verification relations coincide, for instance, in safety properties [9] and differ only in liveness properties [2]. We consider IFLTV the more *practical* verification relation, as it is related to only those differences in system behavior which can practically (in a finite amount of time) be observed.

Consider the following abstract model of a server, see Fig. 1a. If the server resource is *free*, the server reacts to a *request* by an internal event *yes*, followed by sending *result*, but if the resource is locked, then a *request* is rejected by the server. A desired property of a server is that it eventually returns a result. This property can be expressed by the automaton in Fig. 1b. However, our model does not satisfy this property using the classical linear-time verification relation, see for instance [14, 13], because the model, and any reasonable model of such a system, contains the extreme execution scenario in which whenever a request is made the resource is locked. In our model, for instance $lock \cdot (request \cdot no \cdot reject)^{\omega}$ is such a behavior violating the linear-time satisfaction of the property. However, such a behavior is highly unfair, and such extreme executions are normally ignored by using an explicit fairness assumption restricting the allowed executions of the system. In our example assuming strong fairness [6] would do the job. For linear-time satisfaction it is therefore necessary to add such an explicit fairness condition.



(a) Server (b) Property

Fig. 1. An abstract server model and a property automaton.

Applying IFLTV frees one from the need of finding an explicit fairness constraint on the system model by having the fairness assumption inherently in the verification relation's definition. Therefore, not having to deal with fairness explicitly, can make IFLTV quite useful in practice.

3 Preliminaries

We assume the reader is familiar with the most common notions of formal languages and automata theory [8, 11]. For a finite set of system *events* Σ, let Σ^* be the set of all finite strings over Σ, let Σ^ω be the set of all infinite strings, and let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. A language is a set $L \subseteq \Sigma^\infty$.

Let L be a language. Then the set of all finite prefixes of L is $pre(L) = \{w \in \Sigma^* | \exists x \in \Sigma^\infty : wx \in L\}$. The continuation of $w \in pre(L)$ in language L is $cont(w, L) = \{x \in \Sigma^\infty | wx \in L\}$, and the set of all events $en(w, L)$ which are enabled in L after event sequence w occured is $en(w, L) = pre(cont(w, L)) \cap \Sigma$.

Let $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton, where Q is a finite set of states, Σ is a finite set of symbols, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of final states, and $\delta : Q \times \Sigma \to 2^Q$ is a transition relation. If $\delta(q, a) \ni p$ then we call (q, a, p) a transition. For the rest of this paper, we assume the transition relation to be extended to $\delta : 2^Q \times \Sigma^* \to 2^Q$ in the usual way.

Let $w = \alpha_1\alpha_2 \ldots \in \Sigma^\omega$ be an infinite string. A run of automaton $\mathscr{A}$ on w is a sequence $\rho(w) = r_0 r_1 \ldots$, such that $r_{i+1} \in \delta(r_i, \alpha_{i+1})$ for $i \geq 0$. A run is called successful if $r_0 = q_0$ and $\omega(\rho(w)) \cap F \neq \emptyset$ where $\omega(\rho(w))$ is the set of all states that occur infinitely often in $\rho(w)$. Automaton $\mathscr{A}$ Büchi-accepts w if there is a successful run of $\mathscr{A}$ on w. The infinite-string language accepted by Büchi automaton $\mathscr{A}$ is $L(\mathscr{A}) = \{w \in \Sigma^\omega | \mathscr{A}$ *Büchi-accepts* $w\}$[3].

A state $q \in Q$ of the automaton $\mathscr{A}$ is said to be reachable if there exists a path from q_0 to q in $\mathscr{A}$. A state $q \in Q$ of $\mathscr{A}$ is said to be co-reachable if there exists a run $\rho(w) = q_1 q_2 \ldots$ of $\mathscr{A}$ on some infinite string w such that $q_1 = q$ and $\omega(\rho(w)) \cap F \neq \emptyset$.

The *behavior* of a system and the properties are sets of infinitely long event sequences over Σ. We consider here behaviors and properties which can be represented by Büchi automata. A behavior B satisfies the linear-time property P (written: '$B \models P$') if and only if $B \subseteq P$. The satisfaction relation defined in this way is called linear-time satisfaction relation. It follows that $B \cap P = B$ is an equivalent definition of linear-time satisfaction, since B can only be a subset of P if $B = B \cap P$.

Throughout this paper, let P be a property and $\mathscr{P} = (Q_P, \Sigma, \delta_P, q_P, F_P)$ be a Büchi automaton such that $L(\mathscr{P}) = P$, and let B be a behavior and $\mathscr{B} = (Q_B, \Sigma, \delta_B, q_B, F_B)$ be a Büchi automaton such that $L(\mathscr{B}) = B$. An additional requirement for automaton $\mathscr{B}$ is that all its states are accepting states, in other words $F_B = Q_B$,[1] and it can therefore be assumed to be deterministic.

4 The IFLTV Relation and its Naïve Checking Procedure

Instead of demanding $B = B \cap P$ as in the linear-time satisfaction relation, the definition of the IFLTV relation is obtained by relaxing the definition, requiring only that finite prefixes of B have to equal the finite prefixes of $B \cap P$. Thus, we define IFLTV as

$$B \text{ satifies } P \text{ inherently fairly} \Leftrightarrow pre(B) = pre(B \cap P). \qquad (1)$$

[1] The behavior is the Eilenberg-limit of a prefix-closed regular language, which immediately implies that the corresponding Büchi automaton contains only accepting states.

Because $pre(B \cap P) \subseteq pre(B)$ always holds (since $B \cap P \subseteq B$), we can refine the above definition to

$$B \text{ satifies } P \text{ inherently fairly} \Leftrightarrow pre(B) \subseteq pre(B \cap P). \tag{2}$$

Informally, this definition states that a behavior satisfies a property inherently fairly, if and only if all finite behaviors can be continued to an infinite behavior satisfying the property. This observation, written more formally, gives us another way of representing the IFLTV relation:

$$B \text{ satifies } P \text{ inherently fairly} \Leftrightarrow \forall v \in pre(B) : \exists x \in cont(v, B) : vx \in P. \tag{3}$$

Formal proofs of the equivalence of the above-mentioned definitions can be found in [9]. If B satisfies P inherently fairly, we write '$B \Vdash P$'.

The second condition of checking whether or not $pre(B) \subseteq pre(B \cap P)$ is equivalent to checking whether or not $pre(B) \cap \Sigma^* \setminus pre(B \cap P) = \emptyset$. Therefore we can check the IFLTV relation naïvely by (note that the following automata are interpreted as Büchi or finite-string automata as appropriate):

1. Computing the product of $\mathscr{B}$ and $\mathscr{P}$ and reducing it if necessary (yielding $B \cap P$).
2. Changing all states in the resulting automaton to be final states (yielding $pre(B \cap P)$)
3. Determinizing this automaton, making it complete, and interchanging final and non-final states (yielding the complement automaton $\Sigma^* \setminus pre(B \cap P)$).
4. Computing the product of the resulting automaton and $\mathscr{B}$ (yielding $pre(B) \cap (\Sigma^* \setminus pre(B \cap P))$).
5. Checking whether or not the resulting automaton is empty (yielding the test $pre(B) \cap (\Sigma^* \setminus pre(B \cap P)) = \emptyset$).

5 Checking Whether Linear-Time Properties are Satisfied Inherently Fairly

With the help of some auxiliary lemmas, we will prove that the IFLTV relation can be represented by the following conditions on a behavior B and a property P (the result is stated in Corollary 2):

1. $\forall v \in pre(B) : en(v, B) \cap en(v, P) = en(v, B)$ and
2. $\forall v \in (pre(B) \cap pre(P)) : cont(v, B \cap P) \neq \emptyset$.

So checking IFLTV can be reduced to checking enabled conditions of individual events in the behavior and the property (first condition), and checking the possible continuation of finite behaviors in the correct sub-behavior $B \cap P$ of behavior B (second condition).

We first prove that IFLTV implies the following subset condition.

Lemma 1. $B \Vdash P \quad \Rightarrow \quad pre(B) \subseteq pre(P)$.

Proof. Let $v \in pre(B)$. Then, by definition of "$\Vdash$", there is a $w \in cont(v, B)$ such that $vw \in P$. Hence $v \in pre(P)$. □

We prove now that the subset condition is equivalent to the enabling condition we want to show for the IFLTV relation.

Lemma 2. $pre(B) \subseteq pre(P) \quad \Leftrightarrow \quad \forall v \in pre(B) : en(v, B) \cap en(v, P) = en(v, B)$.

Proof. "$\Rightarrow$": Let $v \in pre(B)$ and let $a \in en(v, B)$. Then $va \in pre(B)$ and hence, by Lemma 1, $va \in pre(P)$, implying $a \in en(v, P)$. Thus $en(v, B) \subseteq en(v, P)$, implying $en(v, B) \cap en(v, P) = en(v, B)$.

"$\Leftarrow$": Let $v \in pre(B)$ and let $a \in en(v, B)$. Because $en(v, B) \cap en(v, P) = en(v, B)$, we get $a \in en(v, P)$. Therefore $va \in pre(P)$ and consequently $v \in pre(P)$. Hence $pre(B) \subseteq pre(P)$. □

Lemmas 1 and 2 together result in the following corollary:

Corollary 1. $B \Vdash P \quad \Rightarrow \quad \forall v \in pre(B) : en(v, B) \cap en(v, P) = en(v, B)$.

To complete the prove in one direction, it remains to show that the second condition is a consequence of the IFLTV relation, too.

Lemma 3. $B \Vdash P \quad \Rightarrow \quad \forall v \in (pre(B) \cap pre(P)) : cont(v, B \cap P) \neq \emptyset$.

Proof. Let $v \in (pre(B) \cap pre(P))$. Then in particular $v \in pre(B)$. Then, by definition of "$\Vdash$", there exist $w \in cont(v, B)$ such that $vw \in P$. Hence $vw \in B \cap P$, or equivalently: $w \in cont(v, B \cap P)$. Therefore $cont(v, B \cap P)$ is not empty. □

Having shown that the two conditions stated at the beginning of this section are a consequence of the IFLTV relation, it remains to show that the two conditions together also imply the IFLTV relation.

Lemma 4. *Conditions*

1. $\forall v \in pre(B) : en(v, B) \cap en(v, P) = en(v, B)$ *and*
2. $\forall v \in (pre(B) \cap pre(P)) : cont(v, B \cap P) \neq \emptyset$.

imply $B \Vdash P$.

Proof. Rewriting $\forall v \in (pre(B) \cap pre(P)) : cont(v, B \cap P) \neq \emptyset$ yields

$$\forall v \in (pre(B) \cap pre(P)) : \exists w \in cont(v, B) : vw \in P. \tag{4}$$

From $\forall v \in pre(B) : en(v, B) \cap en(v, P) = en(v, B)$, taking into account Lemma 2, we get $pre(B) \subseteq pre(P)$, implying $pre(B) \cap pre(P) = pre(B)$. Then, substituting $pre(B)$ for $pre(B) \cap pre(P)$ in Equation (4) yields $\forall v \in pre(B) : \exists w \in cont(v, B) : vw \in P$, which is just the definition of $B \Vdash P$. □

Summarizing the previous results, we establish a main result of this paper, which we will use to construct a non-naïve checking procedure for the IFLTV relation.

Corollary 2. $B \Vdash P$ *if and only if*

1. $\forall v \in pre(B) : en(v, B) \cap en(v, P) = en(v, B)$
2. $\forall v \in (pre(B) \cap pre(P)) : cont(v, B \cap P) \neq \emptyset$.

Proof. This corollary follows immediately from Corollary 1 and Lemmas 3 and 4. □

6 The Non-Naïve IFLTV Procedure

Both the behavior and the property are specified by automata. Since it is neccessary to decide a subset condition on the intersection of these languages, we consider an automaton $\mathscr{A}_{B\cap P}$ which accepts $B \cap P$ in order to check the IFLTV relation. So initially, we construct the automaton $\mathscr{A}_{B\cap P}$ and throughout this construction, we check whether for all prefixes of the behavior the enabled actions in the behavior are enabled actions in the property as well, i.e., we check for a potential violation of the first condition given in Corollary 2. If this condition is violated, the construction immediately aborts with the result $\mathit{pre}(B) \not\subseteq \mathit{pre}(B \cap P)$, i.e. B does not satisfy P inherently fairly. Otherwise the construction succeeds and we obtain the automaton $\mathscr{A}_{B\cap P}$. During the reduction of $\mathscr{A}_{B\cap P}$, we verify the second condition of Corollary 2.

For the purpose of checking the conditions of Corollary 2 we compute in parallel, during the construction of $\mathscr{A}_{B\cap P}$, sets of states of the property automaton. These subsets correspond to the states of a 'determinized' version of $\mathscr{P}$. They indicate whether or not Corollary 2 is violated.

Let $\mathscr{B} = (Q_B, \Sigma, \delta_B, q_0, Q_B)$ be the deterministic behavior automaton (recall that behavior automata are always deterministic) and $\mathscr{P} = (Q_P, \Sigma, \delta_P, p_0, F_P)$ be the property automaton (potentially nondeterministic). Then

$$\mathscr{A}_{B\cap P} = (Q_B \times Q_P, \Sigma, \delta, (q_0, p_0), Q_B \times F_P),$$

where δ is defined by:

$$\begin{aligned}&\delta((u,r),a) \ni (v,s) \Leftrightarrow \\ &\delta_B(u,a) = v \quad \text{and} \quad \delta_P(r,a) \ni s, \quad \forall u,v \in Q_B, \quad r,s \in Q_P, \quad a \in \Sigma.\end{aligned}$$

We construct $\mathscr{A}_{B\cap P}$ stepwise starting with the initial state (q_0, p_0) and proceed as follows: Let (q, r) be a state of $\mathscr{A}_{B\cap P}$ not yet considered (at the beginning, there is just the state (q_0, p_0)). Now, for every transition (q, a, p) of $\mathscr{B}$ and every matching transition (r, a, s) of $\mathscr{P}$ we add the transition $((q, r), a, (p, s))$ to $\mathscr{A}_{B\cap P}$. If (p, s) is not yet a state of $\mathscr{A}_{B\cap P}$, we add it as well. If there exists no transition in (q, r) matching (q, a, p) and furthermore, there is no alternative state (q, r') in $\mathscr{A}_{B\cap P}$ with a matching transition, then we abort, because in that case there exists a $v \in \mathit{pre}(B)$ such that $v \notin \mathit{pre}(B \cap P)$, to wit $B \not\Vdash P$. Here, an alternative state (q, r') is a state that can be reached by the same $v \in \Sigma^*$ which passes $\mathscr{A}_{B\cap P}$ into (q, r). Due to the determinism of the behavior automaton, the first component of each state that represents an alternative to state (q, r) must still be q. Thus, we just have to collect the set of states of $\mathscr{P}$ that may occur in the second entry, i.e., we maintain sets $R_q \subseteq 2^{Q_P}$ such that $r' \in R_q$ if and only if there exists a state (q, r') in $\mathscr{A}_{B\cap P}$ which is an alternative to (q, r). Of course, the set which corresponds to the state q_0 is initially $\{\{p_0\}\}_{q_0}$.

There are two possible termination scenarios: Either no more states and transitions can be added, then the construction of the product automaton $\mathscr{A}_{B\cap P}$ is completed, or we may have aborted the construction since we know already that B does not satisfy P inherently fairly.

To check the second condition of Corollary 2, we remove from $\mathscr{A}_{B\cap P}$ all states that are not co-reachable. The resulting automaton is denoted by $\mathscr{A}_{B\cap P,red}$. Note that due to our construction, each state in $\mathscr{A}_{B\cap P}$ is reachable. Detection of states which are not co-reachable can be performed by using a combination of standard algorithms that determine the strongly connected components of the transition graph of $\mathscr{A}_{B\cap P}$ and standard algorithms that reduce finite automata (see, for instance, [1, 5]). Now assume the algorithm has determined the set of useless[2] states and has removed each of its elements from $\mathscr{A}_{B\cap P}$. Thus, after the reduction process, each state of $\mathscr{A}_{B\cap P,red}$ is reachable by a $v \in \Sigma^*$ such that $cont(v, B \cap P) \neq \emptyset$.

The last step of our IFLTV algorithm consists of updating the sets R_q of sets of states of $\mathscr{P}$ that are attached to each state of $A_{B\cap P}$. Of course, if a set R_q contains r' such that (q, r') has been identified to be useless, then we have to remove r' from each set in R_q since (q, r') no longer represents an alternative state. Now, if during this deletion process a set R_q of second-component-states becomes empty, we have identified a $v \in (pre(B) \cap pre(P))$ such that $cont(v, B \cap P) = \emptyset$, violating Corollary 2 and thus $B \not\Vdash P$. Otherwise, the reduction step terminates and all state sets R_q contain at least one state. Then we have $B \Vdash P$.

7 Correctness of the Construction

Given automata that accept the behavior B and the property P respectively, we apply the usual product construction to obtain the automaton $\mathscr{A}_{B\cap P}$ which accepts the intersection $B \cap P$. Note that the computation of the sets R_q of alternative second components of states in $\mathscr{A}_{B\cap P}$ does affect in no way the final product automaton $\mathscr{A}_{B\cap P}$ but rather is supplementary. Hence $L(\mathscr{A}_{B\cap P}) = B \cap P$.

Mainly, we will prove that our construction implements accurately the two conditions of Corollary 2 and for this reason verifies correctly whether or not B satisfies P inherently fairly. So we verify first

Theorem 1. *For all $v \in pre(B)$: $en(v, B) \cap en(v, P) = en(v, B) \Leftrightarrow$ The construction of $\mathscr{A}_{B\cap P}$ succeeds.*

Proof. "$\Rightarrow$" To each $v \in pre(B)$ there is a state $q \in Q_B$ such that $\delta_B(q_B, v) = q$. The set $en(v, B)$ contains all symbols $a \in \Sigma$ such that $\delta_B(q, a) \neq \emptyset$. Furthermore, these symbols are elements of $en(v, P)$ as well. It follows that in $\mathscr{P}$, there is at least one state r such that $\delta_P(q_P, v) \ni r$ and $\delta_P(r, a) \neq \emptyset$. But then, due to the definition, r is in the set of alternative second-component-states R_q and $\delta_P(R_q, a) \neq \emptyset$. This shows the existence of a matching transition. Since this holds for all $v \in pre(B)$, the construction of $\mathscr{A}_{B\cap P}$ succeeds.
"$\Leftarrow$" For all $v \in pre(B)$ there is a state $q \in Q_B$ such that $\delta_B(q_B, v) = q$. Moreover, $en(v, B) \neq \emptyset$ and for all $a \in en(v, B)$ there is a $p \in Q_B$ and $\delta_B(q, a) = p$. Since the construction of $\mathscr{A}_{B\cap P}$ succeeded and was not aborted due to lack of a matching transition, it follows that each transition of $\mathscr{B}$ has at least one corresponding transition in $\mathscr{A}_{B\cap P}$. Consequently, there exist states (q, r) and (p, s) of $\mathscr{A}_{B\cap P}$ such that $\delta((q_B, q_P), v) \ni (q, r)$ and $\delta((q, r), a) \ni (p, s)$. This implies that in $\mathscr{P}$ we have $\delta_P(q_P, v) \ni r$ and $\delta_P(r, a) \ni s$. Hence, $a \in en(v, P)$ implying $en(v, B) \cap en(v, P) = en(v, B)$ for all $v \in pre(B)$. $\square$

[2] We refer to a state as being useless, whenever this state is not co-reachable.

Theorem 2. *For all $v \in (pre(B) \cap pre(P)) : cont(v, B \cap P) \neq \emptyset \Leftrightarrow$ No set R_q, $q \in Q_B$, becomes empty after removing useless states.*

Proof. "$\Rightarrow$" For all $v \in (pre(B) \cap pre(P))$, we have $\delta((q_B, q_P), v) \neq \emptyset$ due to the construction of $\mathscr{A}_{B\cap P}$. Since $L(\mathscr{A}_{B\cap P}) = B \cap P$ and there is a $w \in cont(v, B \cap P)$, at least one element of $\delta((q_B, q_P), v)$ is co-reachable. Thus, there exists a successful run $\rho(vw)$ of $\mathscr{A}_{B\cap P}$ on vw. Obviously, none of the states of $\rho(vw)$ will be removed by the reduction process, or in other words, all corresponding sets of second-entry-states contain at least one element after $\mathscr{A}_{B\cap P}$ has been reduced and the sets R_q have been updated. According to the construction of $\mathscr{A}_{B\cap P}$, each of its states can be reached by a $v \in (pre(B) \cap pre(P))$, and thus occurs in a successful run. This means that each of the sets R_q of second-component-states, which are attached to a state in that run, contains at least one element even after the reduction- and update-process.
"$\Leftarrow$" It follows that for each $v \in (pre(B) \cap pre(P))$ that leads $\mathscr{A}_{B\cap P}$ to one of its states, there is at least one state in the reduced automaton $\mathscr{A}_{B\cap P}$ that is reached by v. Since each state of the reduced $\mathscr{A}_{B\cap P}$ is co-reachable and the reduced automaton $\mathscr{A}_{B\cap P}$ still recognizes $B \cap P$, it implies the existence of a $w \in cont(v, B \cap P)$. Furthermore, due to the construction of $\mathscr{A}_{B\cap P}$, every $v \in (pre(B) \cap pre(P))$ leads $\mathscr{A}_{B\cap P}$ to one of its states. Thus, for all $v \in (pre(B) \cap pre(P))$ there is a $w \in cont(v, B \cap P)$. □

Corollary 3. *The presented IFLTV procedure verifies correctly whether or not B satisfies P inherently fairly.*

Proof. Follows immediately from the theorems above and Corollary 2. □

8 Complexity of the Procedure

It has been shown that deciding IFLTV is PSPACE-complete [9]. So the only complexity criterion for evaluating our algorithm is whether or not it requires less computation steps than the previously known algorithms, even though it remains inefficient in the worst case. However, it can be seen that the presented procedure is more efficient than the naïve one, and, we think that it is worth being considered as an alternative to current model-checking algorithms for verifying liveness properties under fairness constraints.

Our procedure for the IFLTV relation is divided into two major steps, construction of the product automaton and reduction of this automaton. During the construction of the product automaton, we compute the sets we need for the verification task. These sets corresponds to the states of a determinized product automaton, so what we do is in fact a determinization of the product automaton. The second step of reduction, is obtained by computing the strongly connected components of the product automaton.

Compared to the naïve algorithm for IFLTV, which needs to compute the product automata twice, build the complement automaton, determinize once and do an emptiness check, the described procedure is more efficient, which only needs to compute one product automaton, one determinization and one reduction of the automaton. Further, our procedure has an integrated on-the-fly check, which means the procedure may stop whenever a violation of one of the two criteria mentioned above has been detected.

9 Conclusions and Outlook

The presented construction avoids several steps of the naïve procedure, and the verification is done on-the-fly. Part of our immediate future work will be experimenting with an implementation of our approach to tackle complex, industrial-sized case studies. In addition, the resulting procedure still seems to offer scope for improvement. For example, the construction of the product states of the property automaton in parallel with the construction of the behavior-and-property product automaton could be improved by giving up the requirement of being able to detect a violation of Condition 1 of Corollary 2 on the fly, i.e. immediately during the construction of the product automaton: Checking the second condition first, and only then the first condition would make it possible to reduce the automaton before determinization, but on the cost of losing the on-the-fly effect. Exploring these potential improvements will also be part of our future work.

We believe the IFLTV is tailored to model checking liveness properties under fairness constraints. And as model-checkers do not or only partly offer fairness implementation, we will try to integrate the given procedure into an existing model-checker. Such an implementation would allow us to compare our approach with existing algorithms.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass., first edition, 1974.
2. B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.
3. J. R. Büchi. On a decision method in restricted second order arithmetic. In E. Nagel et al., editors, *Proceedings of the International Congress on Logic, Methodology and Philosophy of Science 1960*, pages 1–11. Stanford University Press, 1962.
4. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.
5. S. Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, New York, 1974.
6. N. Francez. *Fairness*. Springer Verlag, New York, first edition, 1986.
7. R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In P. Dembinski and M. Sredniawa, editors, *Protocol Specification, Testing, and Verification XV '95*, pages 3–18. Chapman & Hall, 1996.
8. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley Longman, 2001.
9. U. Nitsche and P. Wolper. Relative liveness and behavior abstraction (extended abstract). In *Proceedings of the 16th ACM Symposium on Principles of Distributed Computing (PODC'97)*, pages 45–52, Santa Barbara, CA, 1997.
10. S. St James and U. Ultes-Nitsche. Computing property-preserving behaviour abstractions from trace reductions. In *Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing (PODC 2001)*, pages 238–245. ACM Press, August 2001.
11. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, pages 133–191. Elsevier, 1990.
12. U. Ultes-Nitsche and S. St James. Improved verification of linear-time properties within fairness — weakly continuation-closed behaviour abstractions computed from trace reductions. *Software Testing, Verification and Reliability (STVR)*, 13(4):241–255, 2003.

13. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st Symposium on Logic in Computer Science*, Cambridge, June 1986.
14. M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.