Relative Liveness and Behavior Abstraction (Extended Abstract)

Ulrich Nitsche*
University of Zürich, Department of Computer Science
Winterthurerstr. 190, CH-8057 Zürich, Switzerland
Email: nitsche@ifi.unizh.ch

Pierre Wolper
University of Liège, Institute Montefiore, B28
B-4000 Liège Sart Tilman, Belgium
Email: pw@montefiore.ulg.ac.be

Abstract

This paper is motivated by the fact that verifying liveness properties under a fairness condition is often problematic, especially when abstraction is used. It shows that using a more abstract notion than truth under fairness, specifically the concept of relative liveness property can lead to interesting possibilities. Technically, it is first established that deciding relative liveness is a PSPACE-complete problem and it is shown that relative liveness properties can always be satisfied by some fair implementation. Thereafter, the interaction between behavior abstraction and relative liveness properties is studied and it is proved that relative liveness properties can be verified on behavior abstractions, if the abstracting homomorphism is *simple* in the sense of Ochsenschläger.

Keywords. Verification, Relative Liveness Properties, Behavior Abstraction, Simple Homomorphisms.

1 Introduction

To be able to verify liveness properties of a system [3], it is almost always necessary to include a fairness hypothesis in the system description [9]. Indeed, introducing a fairness hypothesis makes it possible to ignore behaviors that correspond to extreme execution scenarios and that, in any case, would not occur in any reasonable implementation. Even though this intuition is clear, making fairness precise is somewhat more complicated: should one be "weakly" or "strongly" fair, "transition" or "process" fair, or isn't "justice" or even "compassion" what fairness should really be [15]? Of course, there is a rational way of choosing which fairness notion is adequate for a given problem by considering the nature of the model being used and making reasonable assumptions about how it might be implemented, but it remains that this choice is crucial and delicate.

Furthermore, introducing a fairness hypothesis often makes the verification process somewhat more problematic. This is especially true when abstraction is used. Indeed, since after moving to the abstract level one deals with a reduced set of observables, it can become impossible to express correctly the fairness hypothesis under which the system is correct. This makes one wish for a more general and abstract notion of truth under fairness that would contribute to simplifying verification, especially in the context of abstraction. Intuitively, the notion to be formalized is that of a property being true provided one is given "some control" over the choices made during infinite executions. In other words, one wants to characterize the properties that can be made true by "some fair implementation" of the system.

In this paper, we show that the concept of being a *relative liveness property* is a suitable abstraction of truth under fairness that lends itself easily to verification in the context of abstraction by using the techniques of [16, 20, 21, 22]. Relative liveness properties are liveness properties within the universe of behaviors of the system. Their definition is a relativized version of the definition of liveness: every prefix of a behavior of the system can be extended to an infinite behavior that satisfies the property. This concept and the dual notion of relative safety property were introduced in [12] as a means of clarifying the shift from liveness to safety when timing constraints are introduced in a system. It can also be traced to the notion of machine-closed property [1, 2, 4].

Here we make a different use of the concept and view it as an abstraction of a liveness property being true under fairness. In fact, we interpret relative liveness as a satisfaction relation for properties represented by temporal logic formulas [8, 23]. Notice that for a property to be a relative liveness property of a system does correspond, in the desired abstract sense, to the property being satisfied under fairness. Indeed, in crude terms, the system almost satisfies its relative liveness properties: it just needs the "help of some fairness" (remember that every prefix of a behavior of the system can be extended to an infinite behavior that satisfies the relative liveness property). Furthermore, we show that for ω-regular systems and properties, deciding relative liveness and relative safety are PSPACE-complete problems. This and the fact that, in a reasonable sense, relative liveness properties can be satisfied by some fair implementation are first indications of the usefulness of the abstraction ap-

*Ulrich Nitsche was supported by a DAAD-fellowship HSP II/AUFE to visit the University of Liège. Parts of this work have been done during his visit. He is supported by the Swiss National Science Foundation (SNSF).

proach for this concept for verification.

This usefulness is even more apparent when considering abstraction. Indeed, relative liveness enables us to circumvent the fact that truth under fairness is usually not preserved by abstraction mappings. Precisely, we consider abstractions defined by language homomorphisms in the context of systems described by ω-languages. We prove that whether a property is a relative liveness property can be reliably checked on the abstract system, provided that the homomorphism is *simple* in the sense of [21]. Simplicity of the homomorphism essentially means that it is faithful with respect to the continuation of a word within a language, i.e. the image of the continuation is the continuation of the image of the word and the image of the language.

2 Introductory Examples

To motivate the definitions we present later on, we start with a small example of a concurrent reactive system. Consider the system described as a Petri net in Figure 1.

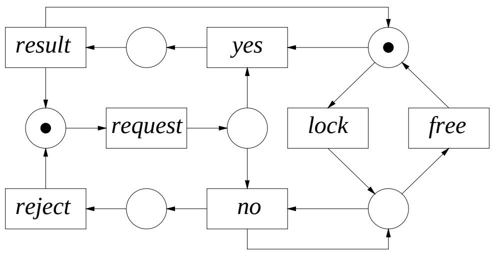

Figure 1: A small system

It is a server that, after having received a $request$, can send a $result$ or a $reject$ion to its client, depending on whether the resource it manages has been $free$ed or $lock$ed. The possible behaviors of the system are represented by the finite-state system shown in Figure 2 (the reachability graph of the Petri net). The initial state is shaded grey, a convention we will also use in subsequent state diagrams.

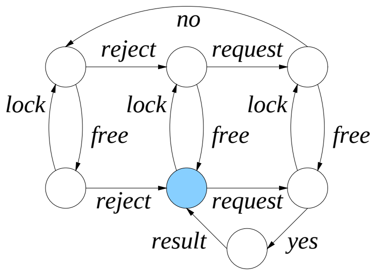

Figure 2: The behaviors of the small system

From Figure 2, it is easy to see that our system does not satisfy the propositional linear time temporal logic [8, 23] property $\Box\Diamond(result)$. Indeed, $lock\cdot(request\cdot no\cdot reject)^\omega$ is a computation of the system that does not satisfy $\Box\Diamond(result)$. Nevertheless, it is clear that what is missing for the property $\Box\Diamond(result)$ to be true is a fairness hypothesis on the system executions. The notion of relative liveness property captures this: $\Box\Diamond(result)$ is a relative liveness property of the set of behaviors described by Figure 2.

Figure 3 gives a finite-state diagram describing the behaviors of a system similar to the one of Figure 1 but containing an error: in Figure 3, if the resource is locked, there is no possibility to free it again. There is also another difference, namely that in Figure 3 a request can also be rejected when the resource is available, but the motivation for this is linked to our subsequent discussion of abstraction. The point to notice now, is that no notion of fairness can make $\Box\Diamond(result)$ true of the new system and that the notion of relative liveness property captures this again: $\Box\Diamond(result)$ is not a relative liveness property of the set of behaviors described in Figure 3.

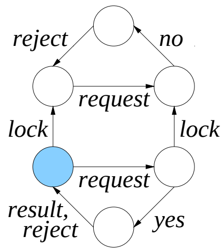

Figure 3: The behaviors of the small system with an error

Let us now consider abstraction. Imagine we are only interested in the actions $request$, $result$, and $reject$. We thus consider an abstraction homomorphism that maps all other actions to the empty word. If we apply this homomorphism to the transition system of Figure 2, we obtain after reduction the transition diagram of Figure 4. The property $\Box\Diamond(result)$ is a relative liveness property of the behaviors described in Figure 4. Can we conclude from there that it is

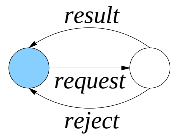

Figure 4: An abstract version of the small system.

also a relative liveness property of the behaviors described by Figure 2? Not without caution since Figure 4 is also obtained by abstracting from Figure 3. What distinguishes the two abstractions is the nature of the homomorphism. In the case of Figure 2 the homomorphism preserves relative liveness properties, whereas it does not do so in the case of Figure 3. In Section 8 we will elaborate on this and show that, if the homomorphism is simple in the sense of [21], one can conclude that relative liveness properties that hold on the abstract system also hold on the concrete system.

3 Preliminaries

For defining our concepts, we need several notions from language theory [5, 7, 11, 24]. Let $L \subseteq \Sigma^*$ be a language and let $L_\omega \subseteq \Sigma^\omega$ be an ω-language.

Definition 3.1 *The* left quotient *of L by a word $w \in \Sigma^*$ is defined by $cont(w, L) = \{v \in \Sigma^* \mid wv \in L\}$. The left quotient of L_ω by $w \in \Sigma^*$ is similarly defined by $cont(w, L_\omega) = \{x \in \Sigma^\omega \mid wx \in L_\omega\}$.*

The left quotient describes the possible continuations of a word in a language. When considering system behaviors, it describes "what can happen after w has happened". Therefore we denote the left quotient of L by w by $cont(w, L)$, "the set of continuations of w in L", instead of the notation $w^{-1}(L)$ common in language theory.

The notation $pre(L)$ designates the set of prefixes of words in L. A language L is called *prefix-closed* if and only if $L = pre(L)$. For an ω-word x, $pre(x)$ designates the set of all finite prefixes of x and, for an ω-language L_ω, $pre(L_\omega)$ designates the set of all finite prefixes of ω-words in L_ω. The limit of a language L is the set $lim(L) = \{x \in \Sigma^\omega \mid \exists^\infty w \in pre(x) : w \in L\}$. Here, "$\exists^\infty$..." abbreviates: "there exist infinitely many different ...". The notation $x_{(n\ldots)}$, $n \in I\!N$, represents the suffix $x_n x_{n+1} x_{n+2} \ldots$ of an ω-word $x \in \Sigma^\omega$ starting with the n^{th} letter of x.

To describe properties, we use *propositional linear temporal logic* (PLTL) [8, 23]. PLTL-formulas are defined with respect to a set AP of atomic propositions. All atomic propositions and the proposition $true$ are PLTL-formulas. If ξ and ζ are PLTL-formulas, then so are $\neg(\xi)$, $(\xi) \wedge (\zeta)$, $\bigcirc(\xi)$ and $(\xi)\,\mathcal{U}\,(\zeta)$. There exist additional operators that are abbreviations of particular operator combinations:

$$
\begin{aligned}
&(\xi) \vee (\zeta) = \neg((\neg(\xi)) \wedge (\neg(\zeta))),\\
&(\xi) \Rightarrow (\zeta) = (\neg(\xi)) \vee (\zeta),\\
&(\xi) \Leftrightarrow (\zeta) = ((\xi) \Rightarrow (\zeta)) \wedge ((\zeta) \Rightarrow (\xi)),\\
&\Diamond(\xi) = (true)\,\mathcal{U}\,(\xi),\\
&\Box(\xi) = \neg(\Diamond(\neg(\xi))),\\
&(\xi)\,\mathcal{B}\,(\zeta) = \neg((\neg(\xi))\,\mathcal{U}\,(\zeta)).
\end{aligned}
$$

PLTL-formulas are interpreted over infinite sequences of truth values for the atomic propositions, i.e. over functions of the type $I\!N \to 2^{AP}$ or, equivalently over ω-words defined on the alphabet 2^{AP}. For convenience, we will also interpret PLTL formulas over infinite words defined on an arbitrary alphabet Σ with the help of a labeling function $\lambda : \Sigma \to 2^{AP}$. The semantics of a PLTL formula with respect to an infinite word $x \in \Sigma^\omega$ and a labeling function $\lambda : \Sigma \to 2^{AP}$ is then the following. (Read "$\models$" as "satisfies.")

- $x, \lambda \models true$.
- If η is an atomic proposition, then $x, \lambda \models \eta$ if and only if $\eta \in \lambda(x_0)$.
- If $\eta = \neg(\xi)$, then $x, \lambda \models \eta$ if and only if it is not the case that $x, \lambda \models \xi$.
- If $\eta = (\xi) \wedge (\zeta)$, then $x, \lambda \models \eta$ if and only if $x, \lambda \models \xi$ and $x, \lambda \models \zeta$.
- If $\eta = \bigcirc(\xi)$, then $x, \lambda \models \eta$ if and only if $x_{(1\ldots)}, \lambda \models \xi$.
- If $\eta = (\xi)\,\mathcal{U}\,(\zeta)$, then $x, \lambda \models \eta$ if and only if there exists $i \in I\!N$ such that $x_{(i\ldots)}, \lambda \models \zeta$ and, for all $j < i$, $x_{(j\ldots)}, \lambda \models \xi$.

The meaning of the other operators can be derived from their definition. We will write $L_\omega, \lambda \models \eta$ if and only if, for all $x \in L_\omega$, $x, \lambda \models \eta$.

Definition 3.2 *A property $\mathcal{P}$ over an alphabet Σ is a subset of Σ^ω. An ω-language $L_\omega \subseteq \Sigma^\omega$ satisfies $\mathcal{P}$ if and only if $L_\omega \subseteq \mathcal{P}$. For an alphabet Σ and a labeling function $\lambda : \Sigma \to 2^{AP}$, the property represented by a PLTL-formula η over AP is the set $L_\eta = \{x \in \Sigma^\omega \mid x, \lambda \models \eta\}$.*

4 Relative Liveness and Safety

In this section, we review the definition of relative liveness properties of an ω-language, as well as their counterpart relative safety properties. Let $L_\omega \subseteq \Sigma^\omega$ be an ω-language representing the behavior of a system and let $\mathcal{P} \subseteq \Sigma^\omega$ be a property.

Definition 4.1 *A property $\mathcal{P}$ is a* relative liveness property *of L_ω (written $L_\omega \models_{RL} \mathcal{P}$) if and only if $\forall w \in pre(L_\omega)$, $\exists x \in cont(w, L_\omega)$ such that $wx \in \mathcal{P}$.*

Definition 4.2 *A property $\mathcal{P}$ is a* relative safety property *of L_ω if and only if $\forall x \in L_\omega$, if $x \notin \mathcal{P}$, then $\exists w \in pre(x) : \forall z \in cont(w, L_\omega) : wz \notin \mathcal{P}$.*

Remark 1 *If $L_\omega = \Sigma^\omega$, then the definitions of relative liveness and relative safety become exactly the definitions of liveness and safety given in [3].*

To prove the decidability of relative liveness and safety for regular ω-languages, we use the following characterizations of these properties.

Lemma 4.3 *$\mathcal{P}$ is a relative liveness property of L_ω if and only if*

$$pre(L_\omega) = pre(L_\omega \cap \mathcal{P}).$$

Lemma 4.4 *$\mathcal{P}$ is a relative safety property of L_ω if and only if*

$$L_\omega \cap lim(pre(L_\omega \cap \mathcal{P})) \subseteq \mathcal{P}.$$

Theorem 4.5 *Given an ω-regular ω-language L_ω and an ω-regular property $\mathcal{P}$ given by nondeterministic Büchi automata or PLTL formulas, determining if $\mathcal{P}$ is a relative liveness or safety property is decidable and is a PSPACE-complete problem.*

Proof The characterizations given by Lemma 4.3 and Lemma 4.4 reduce the problem to questions decidable in PSPACE [24, 10] (notice that for PLTL formulas one can build in PSPACE an automaton for the formula and for its complement [28]). Hardness can be established by a reduction from regular language inclusion [10]. $\Box$

Note that Lemma 4.3 provides the link between relative liveness and machine closure. Indeed, recall the following definition [1, 2, 4].

Definition 4.6 *Let $\Lambda \subseteq L_\omega \subseteq \Sigma^\omega$, for an alphabet Σ. (L_ω, Λ) is called a machine closed live structure if and only if $pre(L_\omega) \subseteq pre(\Lambda)$.*

We thus have that $P \subseteq \Sigma^\omega$ is a relative liveness property of L_ω if and only if $(L_\omega, P \cap L_\omega)$ is a machine closed live structure (see Lemma 4.3).

General properties can always be represented as the intersection of a liveness and a safety property [3]. As given precisely below, the relativized version of this result is that a property holds for an ω-language if it is both a relative liveness and a relative safety property of the language.

Theorem 4.7 *An ω-language L_ω satisfies a property $\mathcal{P}$ ($L_\omega \subseteq \mathcal{P}$) if and only if $\mathcal{P}$ is a relative safety and a relative liveness property of L_ω.*

Proof If $L_\omega \subseteq \mathcal{P}$, then, trivially, $\mathcal{P}$ is a relative safety and a relative liveness property of L_ω.

If $\mathcal{P}$ is a relative safety property of L_ω, then $L_\omega \cap lim(pre(L_\omega \cap \mathcal{P})) \subseteq \mathcal{P}$ (Lemma 4.4). If, additionally, $\mathcal{P}$ is a relative liveness property of L_ω, then, by Lemma 4.3, $pre(L_\omega) = pre(L_\omega \cap \mathcal{P})$. Therefore, we can replace $pre(L_\omega \cap \mathcal{P})$ by $pre(L_\omega)$ in the safety condition and obtain $L_\omega \cap lim(pre(L_\omega)) \subseteq \mathcal{P}$. Because $L_\omega \cap lim(pre(L_\omega)) = L_\omega$, we finally obtain $L_\omega \subseteq \mathcal{P}$. $\Box$

As shown in [12], relative liveness and safety properties also have an elegant definition within the Cantor topology, i.e. the topological space over Σ^ω compatible with the following metric [7]. (For topological notions see [14].)

Definition 4.8 *Let $common(x, y)$ designate the longest common prefix of two ω-words x and y in Σ^ω. We define the metric $d(x, y)$ by*

$$\forall x, y \in \Sigma^\omega, x \neq y : d(x, y) = \frac{1}{|common(x, y)| + 1}$$
$$\forall x \in \Sigma^\omega : d(x, x) = 0.$$

Lemma 4.9 *A property $\mathcal{P}$ is a relative liveness property of an ω-language L_ω if and only if $L_\omega \cap \mathcal{P}$ is a dense set in L_ω.*

Lemma 4.10 *A property $\mathcal{P}$ is a relative safety property of an ω-language L_ω if and only if $L_\omega \cap \mathcal{P}$ is a closed set in L_ω.*

5 Implementing Systems that Satisfy Relative Liveness Properties

If a property is a relative liveness property of a set of behaviors, our expectation is that a fair implementation of this set of behaviors will satisfy the property in the classical sense. Unfortunately, this is not true for every implementation, even if one assumes strong fairness. As an example, consider the set of behaviors $\{a, b\}^\omega$. It is not sufficient to impose strong fairness on the minimal automaton representing $\{a, b\}^\omega$ in order to satisfy all relative liveness properties of $\{a, b\}^\omega$. For instance, $\Diamond(a \wedge (\bigcirc a))$ would not be satisfied, even though it is a relative liveness property of $\{a, b\}^\omega$. The reason for this is that, even if fairness is used, more state information needs to be kept in order to be able to satisfy the property $\Diamond(a \wedge (\bigcirc a))$. However, it is always possible to add sufficient state information to a system in order to turn relative liveness properties into properties that are satisfied in the classical sense under fairness. The following Theorem makes this precise.

Theorem 5.1 *Let L_ω be a limit closed finite-state set of behaviors (one accepted by a finite state automaton without acceptance conditions) and let $\mathcal{P}$ be an ω-regular property. Then, if $\mathcal{P}$ is a relative liveness property of L_ω, there exists a finite-state system $\mathcal{A}$ such that the ω-language accepted by $\mathcal{A}$ is L_ω and all strongly fair computations in $\mathcal{A}$ satisfy $\mathcal{P}$.*

Proof Since $\mathcal{P}$ is a relative liveness property of L_ω, by Lemma 4.3 we have that $pre(L_\omega) = pre(L_\omega \cap \mathcal{P})$. Furthermore, since L_ω is limit closed we have that $L_\omega = lim(pre(L_\omega))$ and hence

$$L_\omega = lim(pre(L_\omega \cap \mathcal{P})). \tag{1}$$

Consider thus a reduced Büchi automaton A accepting $L_\omega \cap \mathcal{P}$ (by reduced we mean that states from which no ω-word can be accepted have been eliminated). The finite-state system $\mathcal{A}$ we are trying to construct is A with its acceptance condition removed. Indeed, by equation (1) $\mathcal{A}$ accepts L_ω. Furthermore, all strongly fair infinite computations of $\mathcal{A}$ will go infinitely often through an accepting state of A and thus will satisfy $\mathcal{P}$. □

The Theorem we have just proved gives an interesting insight into relative liveness properties. They are the properties that fairness makes true of the system, but possibly at the cost of adding state information to the system implementation in a noninterfering way, i.e. without altering the set of limit-closed behaviors of the system.

6 Behavior Abstractions

We now turn to the problem of verifying a system using abstraction. We consider finite-state transition systems without acceptance conditions. Hence the finite-word languages accepted by the systems we consider are the prefix-closed regular languages, and the ω-languages they accept are the limits of prefix-closed regular languages.

We consider abstractions that hide or rename the actions of our systems. Precisely, we consider *abstracting homomorphisms* that are extensions of alphabetic language homomorphisms to mappings on finite and infinite words as defined below.

Definition 6.1 *Let $h : \Sigma \to (\Sigma' \cup \{\varepsilon\})$ be a total function (ε designates the empty word) and let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. Then, the* abstracting homomorphism *generated by h is the extension of h to a mapping $h : \Sigma^\infty \to \Sigma'^\infty$ defined as follows. For all words $w = w_1 w_2 w_3 \ldots w_n \in \Sigma^*$, $n \in I\!N$, we define $h(w) = h(w_1)h(w_2)h(w_3)\ldots h(w_n)$. For all ω-words $x = x_1 x_2 x_3 \ldots \in \Sigma^\omega$, we define $h(x) = h(x_1)h(x_2)h(x_3)\ldots$, if $lim(h(pre(x))) \neq \emptyset$. Otherwise, if $lim(h(pre(x))) = \emptyset$, then $h(x)$ is undefined.*

This leads naturally to the following definition of the abstract behavior of a system under an abstracting homomorphism.

Definition 6.2 *Let $\mathcal{S}$ be a system whose behaviors are the limit $lim(L)$ of a prefix-closed regular language L. Then, the* abstract behavior *of $\mathcal{S}$ with respect to the abstracting homomorphism h is $lim(h(L))$.*

Our goal is to prove properties of the behaviors $lim(L)$ of a system $\mathcal{S}$ by only considering the abstract behaviors $lim(h(L))$ for some abstracting homomorphisms h. More specifically, we are interested in the preservation of relative liveness properties under the abstraction homomorphism.

Essential information about the relative liveness properties of $lim(L)$ is contained in the sets $cont(w, L)$, for $w \in L$. At the abstract level, we obviously have access to $cont(h(w), h(L))$, but we really need $h(cont(w, L))$ in order to be able to establish relative liveness properties that will also hold at the concrete system level. Thus, we need investigate the relation between the sets $h(cont(w, L))$ and $cont(h(w), h(L))$ and find conditions under which $cont(h(w), h(L))$ can be used instead of $h(cont(w, L))$.

In general, $h(cont(w, L))$ is a proper subset of $cont(h(w), h(L))$. In order to obtain sufficient information about $h(cont(w, L))$ from $cont(h(w), h(L))$, one would be tempted to require equality of the two sets. However, this is stronger than needed. Indeed, since we are dealing with relative liveness properties, it is sufficient that the behaviors in $cont(h(w), h(L))$ "eventually" become behaviors in $h(cont(w, L))$. This condition is the one called *simplicity* of an abstraction homomorphism in [21]. Its exact definition is the following.

Definition 6.3 *An abstracting homomorphism $h : \Sigma^\infty \to \Sigma'^\infty$ is* simple *for a language $L \subseteq \Sigma^*$ and a word $w \in L$ if and only if there exists $u \in cont(h(w), h(L))$ such that $cont(u, cont(h(w), h(L))) = cont(u, h(cont(w, L)))$. The homomorphism h is simple for L if and only if it is simple for L and all words $w \in L$.*

Theorem 8.2 will show that this definition indeed meets all the requirements we have informally described above. More details about simple homomorphisms can be found in [21].

7 Preservation of Linear Properties

Before turning to the preservation of relative liveness properties under simple homomorphisms, we need some general results about abstraction homomorphisms and properties. The problem we address is that the properties true of the abstracted system and of the concrete system can rarely be identical. Indeed, one needs to take into account the fact that the abstraction can rename or hide symbols. Our goal here is to define a transformation on properties that mirrors this.

We consider properties defined by PLTL formulas (see Section 3). In order to make the definition of property transformations easier and to make the interpretation of formulas over words more direct (remember that we are dealing with systems represented by sets of infinite words), we define some normal forms for PLTL formulas.

A first restriction is to consider only positive normal form formulas.

Definition 7.1 *A PLTL-formula η is in* positive normal form *if and only if the scope of all negations is a single atomic proposition.*

Now we turn to the problem of interpreting formulas over words. Our generic way of doing this (see Section 3) is to use a mapping $\lambda : \Sigma \to 2^{AP}$ from the alphabet Σ of the word to the subsets of the atomic propositions AP of the formula. However, in this context, it is quite natural to consider the elements of Σ directly as atomic propositions, which implies that one is using a mapping λ_Σ such that $\forall a \in \Sigma : \lambda_\Sigma(a) = \{a\}$. We define a normal from that corresponds to this.

Definition 7.2 *Let Σ be an alphabet. We say that a PLTL formula η is in* Σ-normal form *if and only if η is in positive normal form and all its atomic propositions are in Σ (i.e. $AP \subseteq \Sigma$).*

For an alphabet Σ, the canonical Σ-labeling function $\lambda_\Sigma : \Sigma \to 2^\Sigma$ *is the one such that such that $\forall a \in \Sigma : \lambda_\Sigma(a) = \{a\}$.*

We will assume by default that, on a word x over the alphabet Σ, a formula in Σ-normal form is interpreted with the canonical Σ-labeling function. We will write $x, \lambda_\Sigma \models \eta$ to express that the word x satisfies the Σ-normal form formula η under the canonical Σ-labeling function λ_Σ.

Note that using Σ-normal for formulas is not really restrictive. Indeed, for any PLTL-formulas η over a set AP of atomic proposition and any labeling function $\lambda : \Sigma \to 2^{AP}$, there exists a PLTL-formula η' in Σ-normal form such that, for all $x \in \Sigma^\omega$, $x, \lambda \models \eta$ if and only if $x, \lambda_\Sigma \models \eta'$.

We now turn to the interaction between properties and abstraction homomorphisms. Consider an abstraction homomorphism $h : \Sigma^\infty \to \Sigma'^\infty$ and assume we have established a (Σ'-normal form) property η of the abstract version L'_ω of a system obtained under this homomorphism. Of what system can we say that the property is true on the concrete level? One would expect $h^{-1}(L'_\omega)$. However, this is a language on Σ on which we cannot directly interpret η. One could modify η to take this into account, but it is simpler to modify the labeling function.

Definition 7.3 *For alphabets Σ and Σ', and for an abstraction homomorphism $h : \Sigma^\infty \to \Sigma'^\infty$, the* canonical $h_{\Sigma\Sigma'}$-labeling function $\lambda_{h_{\Sigma\Sigma'}} : \Sigma \to 2^{\Sigma' \cup \{\varepsilon\}}$ *is the one such that such that $\forall a \in \Sigma : \lambda_{h_{\Sigma\Sigma'}}(a) = \{h(a)\}$.*

Notice that this labeling function maps some letters to the proposition ε which stands for the empty word. So, we can't expect a formula η true of the abstract system L'_ω to be true of $h^{-1}(L'_\omega)$, even using the mapping $\lambda_{h_{\Sigma\Sigma'}}$. Indeed, this mapping takes care of the fact that letters are renamed, but does not take care of the fact that ε is the empty word. What is needed is to ignore the empty word in the evaluation of the formula. This is handled by transforming the formula η from Σ'-normal form to $\Sigma' \cup \varepsilon$-normal form as follows.

Definition 7.4 *Let η be a PLTL-formula in Σ'-normal form. We define recursively a mapping $T(\eta)$ that yields a formula in $\Sigma' \cup \varepsilon$-normal form (see Figure 5; $\hat{b}$ designates binary boolean operators: $\hat{b} \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$).*

$$T(\eta) = \begin{cases} true, & \text{if } \eta = true, \\ \neg(true), & \text{if } \eta = \neg(true), \\ a, & \text{if } \eta = a \in \Sigma', \\ (\neg(a)) \wedge (\neg(\varepsilon)), & \text{if } \eta = \neg(a) \text{ and } a \in \Sigma', \\ (T(\xi))\, \hat{b}\, (T(\zeta)), & \text{if } \eta = (\xi)\, \hat{b}\, (\zeta), \\ ((\varepsilon) \vee (T(\xi)))\ \mathcal{U}\ (T(\zeta)), & \text{if } \eta = (\xi)\, \mathcal{U}\, (\zeta), \\ (T(\xi))\ \mathcal{B}\ (T(\zeta)), & \text{if } \eta = (\xi)\, \mathcal{B}\, (\zeta), \\ \Diamond(T(\xi)), & \text{if } \eta = \Diamond(\xi), \\ (\Box((\varepsilon) \vee (T(\xi)))) \wedge (\Box(\Diamond(T(\xi)))), & \text{if } \eta = \Box(\xi), \\ (\varepsilon)\ \mathcal{U}\ ((\neg(\varepsilon)) \wedge (\bigcirc((\varepsilon)\ \mathcal{U}\ (T(\xi))))), & \text{if } \eta = \bigcirc(\xi). \end{cases}$$

Figure 5: The syntactical transformation of PLTL.

As defined, the mapping T does not modify pure Boolean formulas (not including any temporal operator). However, a pure Boolean formula η should be mapped to $(\varepsilon)\,\mathcal{U}\,(\eta)$. We thus extend T into a mapping $\bar{R}$ such that $\bar{R}(\eta)$ is $T(\eta)$ with all maximal pure Boolean subformulas ξ_b replaced by $(\varepsilon)\,\mathcal{U}\,(\xi_b)$.

We can now give a statement relating a property true on an abstraction of a system to a property true at the concrete level [17, 16]

Lemma 7.5 *Let $L_\omega \subseteq \Sigma'^\omega$, let η be a PLTL-formula in Σ'-normal form, and let $h : \Sigma^\infty \to \Sigma'^\infty$ be an abstracting homomorphism. Then*

$$L'_\omega, \lambda_{\Sigma'} \models \eta \iff h^{-1}(L'_\omega), \lambda_{h_{\Sigma\Sigma'}} \models \bar{R}(\eta).$$

8 Preservation of Relative Liveness Properties

Let $L \subseteq \Sigma^*$ be a prefix-closed language, let $h : \Sigma^\infty \to \Sigma'^\infty$ be an abstracting homomorphism, and let η be a PLTL-formula in Σ'-normal form. Assume that η is a relative liveness property of $lim(h(L))$; in our notation $lim(h(L)), \lambda_{\Sigma'} \models_{RL} \eta$. We will prove that, if the homomorphisms h is simple, then η is also a relative liveness property of $lim(L)$ in the sense that $lim(L), \lambda_{h_{\Sigma\Sigma'}} \models_{RL} \bar{R}(\eta)$.

To establish this result we need a condition that allows to commute limit application of the homomorphism.

Lemma 8.1 *If $L \subseteq \Sigma^*$ is a prefix-closed regular language and $h : \Sigma^\infty \to \Sigma'^\infty$ is an abstracting homomorphism, then $lim(h(L)) = h(lim(L))$.*

Proof (outline) The case $h(lim(L)) \subseteq lim(h(L))$ is straightforward. To show $lim(h(L)) \subseteq h(lim(L))$, we use König's Lemma ([13], Lemma 3.3.). We can show that, to each ω-word $x' \in \lim(h(L))$, there exists a sequence $(u_n)_{n\in\mathbb{N}}$ of words in L such that $(u_n)_{n\in\mathbb{N}}$ generates an ω-word x in $lim(L)$ and $h(x) = x'$ [16]. $\square$

Using Lemma 8.1, we can now prove our result relating a relative liveness property of $lim(h(L))$ to a relative liveness property of $lim(L)$.

Theorem 8.2 *Let $L \subseteq \Sigma^*$ be a prefix-closed regular language, let $h : \Sigma^\infty \to \Sigma'^\infty$ be an abstracting homomorphism such that h is simple on L and $h(L)$ does not contain maximal words (words not being a proper prefix of another word in $h(L)$), and let η be a PLTL-formula in Σ'-normal form. We have that*

$$lim(h(L)), \lambda_{\Sigma'} \models_{RL} \eta \quad \textit{implies} \quad lim(L), \lambda_{h_{\Sigma\Sigma'}} \models_{RL} \bar{R}(\eta).$$

Proof We assume that $lim(h(L)), \lambda_{\Sigma'} \models_{RL} \eta$ and derive $lim(L), \lambda_{h_{\Sigma\Sigma'}} \models_{RL} \bar{R}(\eta)$. By definition $lim(L), \lambda_{h_{\Sigma\Sigma'}} \models_{RL} \bar{R}(\eta)$ if for all $u \in L$, there exists some $x \in cont(u, lim(L))$ such that $ux, \lambda_{h_{\Sigma\Sigma'}} \models \bar{R}(\eta)$. Consider thus an arbitrary $u \in L$. Because h is simple on L, there exists $v \in cont(h(u), h(L))$ such that

$$\begin{gathered} cont(v, h(cont(u, L))) = \\ cont(v, cont(h(u), h(L))) = \\ cont(h(u)v, h(L)). \end{gathered} \tag{1}$$

By our hypothesis that $lim(h(L)), \lambda_{\Sigma'} \models_{RL} \eta$, we have that $\forall r \in pre(lim(h(L))) : \exists s \in cont(r, lim(h(L))) : rs, \lambda_{\Sigma'} \models \eta$, and in particular, substituting $h(u)v$ for r that there exists some $y \in cont(h(u)v, lim(h(L))) = lim(cont(h(u)v, h(L)))$ such that

$$h(u)vy, \lambda_{\Sigma'} \models \eta. \tag{2}$$

Given equation (1) this is equivalent to

$$y \in lim(cont(v, h(cont(u, L)))) =$$

$$cont(v, lim(h(cont(u, L)))).$$

Thus we know that vy is in $lim(h(cont(u, L)))$, which, in view of Lemma 8.1, is equivalent to

$$vy \in h(lim(cont(u, L))).$$

So, there exists $x \in lim(cont(u, L))$ such that

$$h(x) = vy. \tag{3}$$

Viewing vy as a single word z, we have thus shown that for all $u \in L$, there exists $x \in lim(cont(u, L))$ and $z \in cont(h(u), lim(h(L)))$ such that $h(x) = z$ (equation (3)) and $h(u)z, \lambda_{\Sigma'} \models \eta$ (equation (2)).

Consider now the language $\tilde{L} = pre(ux)$ of prefixes of ux. Clearly, $lim(\tilde{L}) = \{ux\}$ and $lim(h(\tilde{L})) = \{h(u)z\}$.

Because $h(u)z, \lambda_{\Sigma'} \models \eta$, $lim(h(\tilde{L})), \lambda_{\Sigma'} \models \eta$. Using Lemma 7.5 and given that $lim(\tilde{L}) \subseteq h^{-1}(lim(h(\tilde{L})))$, we obtain $lim(\tilde{L}), \lambda_{h_{\Sigma\Sigma'}} \models \bar{R}(\eta)$, or $ux, \lambda_{h_{\Sigma\Sigma'}} \models \bar{R}(\eta)$. We have this shown that for all $u \in L$, there exists $x \in cont(u, lim(L))$, such that $ux, \lambda_{h_{\Sigma\Sigma'}} \models \bar{R}(\eta)$. Thus we have shown that $lim(L), \lambda_{h_{\Sigma\Sigma'}} \models_{RL} \bar{R}(\eta)$ $\square$

As shown in Section 2 with an example, Theorem 8.2 does not hold, if we do not require the abstracting homomorphism to be simple.

Theorem 8.3 *Let $L \subseteq \Sigma^*$ be a prefix-closed regular language. Let $h : \Sigma^\infty \to \Sigma'^\infty$ be an abstracting homomorphism such that $h(L)$ does not contain maximal words. Let η be a PLTL-formula in Σ'-normalform. Then*

$$lim(L), \lambda_{h_{\Sigma\Sigma'}} \models_{RL} \bar{R}(\eta) \quad \textit{implies} \quad lim(h(L)), \lambda_{\Sigma'} \models_{RL} \eta.$$

Proof We assume that $lim(L), \lambda_{h_{\Sigma\Sigma'}} \models_{RL} \bar{R}(\eta)$ and show $lim(h(L)), \lambda_{\Sigma'} \models_{RL} \eta$. Let $w' \in pre(lim(h(L)))$, let $w \in pre(lim(L)) \cap h^{-1}(w')$, and let $x \in cont(w, lim(L))$ such that $wx, \lambda_{h_{\Sigma\Sigma'}} \models \bar{R}(\eta)$.

If $h(wx)$ is defined, then, by Lemma 7.5, $h(wx), \lambda_{\Sigma'} \models \eta$. Therefore, there exists an $x' = h(x) \in cont(w', lim(h(L)))$ such that $w'x', \lambda_{\Sigma'} \models \eta$.

If $h(wx)$ is undefined, then there is a prefix v of wx such that $h(cont(v, pre(wx))) = \{\varepsilon\}$. (In fact, there are infinitely many of these prefixes v.) Then, by definition of $\bar{R}$ and $\lambda_{h_{\Sigma\Sigma'}}$, we have, for all $y \in \Sigma^\omega$, that $vy, \lambda_{h_{\Sigma\Sigma'}} \models \bar{R}(\eta)$.

If there exists a $y \in \Sigma^\omega$ such that $h(y) \in cont(h(v), lim(h(L)))$, then let x' be the only ω-word in $cont(w', \{h(vy)\})$. x' is in $cont(w', lim(h(L)))$. According to Lemma 7.5, $w'x', \lambda_{\Sigma'} \models \eta$.

If there exists no $y \in \Sigma^\omega$ such that $h(y) \in cont(h(v), lim(h(L)))$, then $h(L)$ contains maximal words, which contradicts the theorem's preconditions.

So, for all $w' \in pre(lim(h(L)))$, there exists an $x' \in cont(w', lim(h(L)))$ such that $w'x', \lambda_{\Sigma'} \models \eta$. Thus $lim(h(L)), \lambda_{\Sigma'} \models_{RL} \eta$. $\square$

Corollary 8.4 *Let $L \subseteq \Sigma^*$ be a prefix-closed regular language, let $h : \Sigma^\infty \to \Sigma'^\infty$ be an abstracting homomorphism such that h is simple on L and $h(L)$ does not contain maximal words, and let η be a PLTL-formula in Σ'-normalform. Then*

$$lim(h(L)), \lambda_{\Sigma'} \models_{RL} \eta \quad \textit{if and only if} \quad lim(L), \lambda_{h_{\Sigma\Sigma'}} \models_{RL} \bar{R}(\eta).$$

Proof This corollary is a summary of Theorem 8.2 and Theorem 8.3. $\square$

If $h(L)$ contains maximal words, we have to extend them by $\{\#\}^*$ as presented in [20], to keep maximal words "visible" when considering $lim(h(L))$.

9 Conclusion

We have introduced relative liveness properties as an abstraction of properties true under fairness. One major advantage of relative liveness properties is that they can be verified using behavior abstraction under simple homomorphisms as is shown in Theorem 8.2. A related result for the $\forall\Box\exists\Diamond$-fragment of CTL* appears in [18, 19]. Additionally, a recent result [20] shows that simplicity of homomorphisms is also a necessary condition for the preservation of relative liveness properties under abstraction.

For practical purposes, it is essential to be able to obtain a representation of the abstract behavior of a system without an exhaustive construction of the finite-state system generating the original behavior. A compositional analysis technique [22] makes it possible to compute the finite-state representation of the abstract behavior by a partial state-space exploration. Therefore, regarding the results of this paper, we can check relative liveness properties of specifications without an exhaustive construction of their state-spaces. An application of abstraction techniques to the detection of undesired feature interactions in intelligent networks can be found in [6].

Relative liveness properties reveal a satisfaction relation for properties that informally says: "almost all computations satisfy the property." In this sense, they appear to be close to properties that are probabilistically true [26, 27]. It would be an interesting topic for further study to investigate the exact link between relative liveness and probabilistic verification.

References

[1] ABADI, M., AND LAMPORT, L. The existence of refinement mappings. SRC Report 29, DEC System Research Center, July 1988.

[2] ABADI, M., AND LAMPORT, L. Composing specifications. SRC Report 66, DEC System Research Center, October 1990.

[3] ALPERN, B., AND SCHNEIDER, F. B. Defining liveness. *Information Processing Letters 21*, 4 (October 1985), 181–185.

[4] ALUR, R., AND HENZINGER, T. A. Local liveness for compositional modeling of fair reactive systems. In *Computer Aided Verification (CAV) '95* (1995), P. Wolper, Ed., vol. 939 of *Lecture Notes in Computer Scinece*, Springer, pp. 166–179.

[5] BERSTEL, J. *Transductions and Context-Free Languages*, first ed. Studienbücher Informatik. Teubner Verlag, Stuttgart, 1979.

[6] CAPELLMANN, C., DEMANT, R., FATAHI-VANANI, F., GALVEZ-ESTRADA, R., NITSCHE, U., AND OCHSENSCHLÄGER, P. Verification by behavior abstraction: A case study of service interaction detection in intelligent telephone networks. In *Computer Aided Verification (CAV) '96* (New Brunswick, 1996), vol. 1102 of *Lecture Notes in Computer Science*, pp. 466–469.

[7] EILENBERG, S. *Automata, Languages and Machines*, vol. A. Academic Press, New York, 1974.

[8] EMERSON, E. A. Temporal and modal logic. In van Leeuwen [25], pp. 995–1072.

[9] FRANCEZ, N. *Fairness*, first ed. Springer Verlag, New York, 1986.

[10] GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., New York, 1979.

[11] HARRISON, M. A. *Introduction to Formal Language Theory*, first ed. Addison-Wesley, Reading, Mass., 1978.

[12] HENZINGER, T. A. Sooner is safer than later. *Information Processing Letters 43* (1992), 135–141.

[13] HOOGEBOOM, H., AND ROZENBERG, G. Infinitary languages: Basic theory and applications to concurrent systems. In *Current Trends in Concurrency* (1986), J. de Bakker, W.-P. de Roever, and G. Rozenberg, Eds., vol. 224 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 266–342.

[14] KELLEY, J. L. *General Topology*. Van Nostrand, Princeton, 1955.

[15] MANNA, Z., AND PNUELI, A. *The Temporal Logic of Reactive and Concurrent Systems—Specification*, first ed. Springer Verlag, New York, 1992.

[16] NITSCHE, U. *Verification of Co-Operating Systems and Behaviour Abstraction*. PhD thesis, University of Frankfurt, Germany. handed in 1996.

[17] NITSCHE, U. Propositional linear temporal logic and language homomorphisms. In *Logical Foundations of Computer Science '94, St. Petersburg* (1994), A. Nerode and Y. V. Matiyasevich, Eds., vol. 813 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 265–277.

[18] NITSCHE, U. A verification method based on homomorphic model abstraction. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing* (Los Angeles, 1994), ACM Press, p. 393.

[19] NITSCHE, U. Verification and behavior abstraction – towards a tractable verification technique for large distributed systems. *Journal of Systems and Software 33*, 3 (June 1996), 273–285.

[20] NITSCHE, U., AND OCHSENSCHLÄGER, P. Approximately satisfied properties of systems and simple language homomorphisms. *Information Processing Letters 60* (1996), 201–206.

[21] OCHSENSCHLÄGER, P. Verification of cooperating systems by simple homomorphisms using the product net machine. In *Workshop: Algorithmen und Werkzeuge für Petrinetze* (1994), J. Desel, A. Oberweis, and W. Reisig, Eds., Humboldt Universität Berlin, pp. 48–53.

[22] OCHSENSCHLÄGER, P. Compositional verification of cooperating systems using simple homomorphisms. In *Workshop: Algorithmen und Werkzeuge für Petrinetze* (1995), J. Desel, H. Fleischhack, A. Oberweis, and M. Sonnenschein, Eds., Universität Oldenburg, pp. 8–13.

[23] PNUELI, A. The temporal logic of programs. In *Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science* (1977), pp. 46–57.

[24] THOMAS, W. Automata on infinite objects. In van Leeuwen [25], pp. 133–191.

[25] VAN LEEUWEN, J., Ed. *Formal Models and Semantics* (1990), vol. B of *Handbook of Theoretical Computer Science*, Elsevier.

[26] VARDI, M. Y. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science* (Portland, October 1985), pp. 327–338.

[27] VARDI, M. Y., AND WOLPER, P. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st Symposium on Logic in Computer Science* (Cambridge, June 1986).

[28] VARDI, M. Y., AND WOLPER, P. Reasoning about infinite computations. *Information and Computation 115*, 1 (November 1994), 1–37.