Model-checking Inherently Fair Linear-time Properties*

Thierry Nicola, Frank Nießner and Ulrich Ultes-Nitsche

telecommunications, *networks* & *security* Research Group
Department of Computer Science, University of Fribourg
Chemin du Musée 3, 1700 Fribourg, Switzerland

Abstract. The concept of linear-time verification with an inherent fairness condition has been studied under the names *approximate satisfaction*, *satisfaction up to liveness*, and *satisfaction within fairness* in several publications. Even though proving the general applicability of the approach, reasonably efficient algorithms for *inherently fair linear-time verification* (IFLTV) are lacking. This paper bridges the gap between the theoretical foundation of IFLTV and its practical application, presenting a model-checking algorithm based on a structural analysis of the synchronous product of the system and property (Büchi) automata.

1 Introduction

To be able to verify liveness properties of a system [1], it is almost always necessary to include a fairness hypothesis in the system description [3]. 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 [6]? 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.

Such a characterization has been given in previous years, leading to the exploring linear-time verification with an inherent fairness condition. *Inherently fair linear-time verification* (IFLTV) has been studied under the name *approximate satisfaction* [10], *satisfaction up to liveness* [11], and *satisfaction within fairness* [12, 14, 15]. All mentioned papers deal with the general concept of IFLTV [11] as well as the relation of IFTLV to abstraction [10, 11] and partial-order methods [4, 16], and the combination of the two state-space reduction techniques [15]. What has not yet been considered is the actual implementation of IFLTV by means of a reasonably efficient model-checking algorithm. Here, "reasonably efficient" refers to algorithms behaving not too badly on

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

Nicola T., Nießner F. and Ultes-Nitsche U. (2005).
Model-checking Inherently Fair Linear-time Properties.
In *Proceedings of the 3rd International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*, pages 3-8
DOI: 10.5220/0002574300030008

practical examples, since the general model-checking problem is PSPACE-complete [11].

In this paper, we present a model-checking algorithm for IFLTV based on a structural analysis of the synchronous product of the two Büchi automata representing system and property respectively. The system will always be represented by a labeled transition system (deterministic Büchi automaton in which all states are accepting) where the property, in the most general case, will require a *non-deterministic* Büchi automaton to represent it. We will start with the case in which the property automaton is *deterministic*, and develop the IFLTV model-checking algorithm for this case. Deterministic Büchi automata cover already all safety and many liveness properties [1, 7]. We will then discuss how to extend the result for the deterministic case to inherently non-deterministic properties. Finally we will comment on the additional effort of IFLTV of inherently non-deterministic properties.

2 Motivation

The motivation for IFLTV is twofold: first, IFLTV possesses an inherent *fairness* condition, and second, IFLTV related to *observable* differences in system behavior.

2.1 Inherent Fairness

Consider the following "telecommunications" system: two users of the system may call one another; if the called user is not busy, the call will reach her/him; otherwise the call is rejected. A calling user has no control of whether the called user is engaged in another call (busy) or not. Such a system is normally modeled by a nondeterministic choice: whenever a user attempts to call another user, the system model decides nondeterministically whether the called user is busy or not. In such a scenario, there exists the extreme execution in which, whenever a user is called, the user is busy. Such executions are normally ignored by using an *explicit fairness assumption* [3] restricting the allowed executions of the system. Applying IFLTV frees one from the need of finding an explicit fairness restriction on the system model by having a fairness assumption *inherent* in its definition.

2.2 Observability

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 decide nondeterministically to operate forever. An outside observer will never be able to distinguish the two systems: if a system has stopped, it may be either system; if it has not stopped, it may again be either system. Only *infinite* observations could distinguish the two systems which is apparently practically impossible. So, system one is as good as system two from that point of view. Linear-time verification, however, distinguishes the two systems as one system does not satisfy the property "performing only finitely many operations" where the other one does. IFLTV is as powerful as linear-time verification, but insensitive to differences requiring infinite observations. We therefore consider IFLTV the more practical verification technique.

3 Preliminaries

The *behavior* of a distributed system is a set of infinitely long sequences of *actions* from a finite set Σ of actions. Thus behaviors are ω-languages on Σ. Since each (infinite) behavior is the infinite continuation of finite behaviors of the distributed system, and since each prefix of a finite behavior is itself a finite behavior of the system, the set of behaviors of a distributed system is the *Eilenberg-limit* [2] of a *prefix-closed* language.[1]

Let Σ^* be the set of all finitely long sequences on Σ, let Σ^ω be the set of all infinitely long sequences, and let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. Let $L \subseteq \Sigma^*$.

- $pre(M) = \{v \in \Sigma^* \mid \exists x \in \Sigma^\infty : vx \in M\}$ is the set of all finite prefixes of $M \subseteq \Sigma^\infty$. Then $pre(x) = pre(\{x\})$ is that of $x \in \Sigma^\infty$.
- L is prefix-closed if and only if $pre(L) = L$.
- $lim(L) = \{x \in \Sigma^\omega \mid \exists^\infty w \in pre(x) : w \in L\}$ is the Eilenberg-limit of language L [2, 13].[2]
- A property P on Σ is a subset of Σ^ω. Behaviour $lim(L)$ satisfies property P (written: "$lim(L) \vDash P$") if and only if $lim(L) \subseteq P$ [1].
- $cont(w, M) = \{v \in \Sigma^\infty \mid wv \in M\}$ is the leftquotient of $M \subseteq \Sigma^\infty$ by $w \in \Sigma^*$.

To introduce an implicit fairness assumption into the satisfaction relation, *relative liveness properties* [5, 11] are defined as a satisfaction relation of properties [10, 11]. This satisfaction relation is called *inherently fair linear-time verification* (IFLTV) relation in this paper. There are three different ways of defining IFLTV. Two are important regarding this paper and are presented subsequently:

1. $lim(L)$ satisfies inherently fair $P \subseteq \Sigma^\omega$ (written: "$lim(L) \Vdash P$") if and only if $\forall w \in pre(lim(L)) : \exists x \in cont(w, lim(L)) : wx \in P$.
2. $lim(L) \Vdash P$ if and only if $pre(lim(L)) = pre(lim(L) \cap P)$.

From the second definition it follows that we can check the IFLTV relation by examining the automaton representing $lim(L) \cap P$. Since $pre(lim(L)) \supseteq pre(lim(L) \cap P$ is always true, we only have to find a condition ensuring $pre(lim(L)) \subseteq pre(lim(L) \cap P$. Will we examine this condition subsequently for the case in which P is represented by a *deterministic* Büchi automaton. We use B as a shorthand for behavior $lim(L)$, which is always deterministic.

We construct subsequently the automaton $\mathscr{A}_{B \cap P}$ representing the intersection of behavior and property (the so-called synchronous product automaton) for the case of deterministic P. This yields an IFLTV model-checking algorithm for the deterministic case.

[1] It is important to note that dealing only with languages is not a restriction since finite automata can *completely* (including state information) be encoded by their local languages [2] (the languages over transition triples $(state, event, successorstate)$).

[2] Read "$\exists^\infty \ldots : \ldots$" as "there exist infinitely many different ... such that ...".

4 Construction of $\mathscr{A}_{B \cap P}$

It must be guaranteed during construction of the product automaton that $B \Vdash P$ remains valid. It is necessary to modify the classic algorithm of the synchronous product construction. The additional feature ensures that the result automaton does not violate $B \Vdash P$, or if it does that it is detected. Let $\mathscr{A}_B = (Q_B, \Sigma, q_0, F_B, \Delta_B)$ the automaton representing the behaviour and $\mathscr{A}_P = (Q_P, \Sigma, p_0, F_P, \Delta_P)$ the one of the properties. The construction creates first the new initial state (q_0, p_0) of the product automaton $\mathscr{A}_{B \cap P}$. Then for every transition $(q_0, a, q_i) \in \Delta_B$, where q_0 is the initial state, $a \in \Sigma$ and $q_i \in Q_B$, there must be a transition $(p_0, a, p_j) \in \Delta_P$, where p_0 the initial state of $\mathscr{A}_P$, $a \in \Sigma$ and $p_j \in Q_P$. If that does not hold, we abort, because $B \nVdash P$. Otherwise we add the state (q_i, p_j) and the transition $((q_0, p_0), a, (q_i, p_j))$ to $\mathscr{A}_{B \cap P}$.

We continue that process of adding new states and transitions until the product automaton $\mathscr{A}_{B \cap P}$ is complete (no more states and transistions can be added), or we have found that for a state (q, p) in $\mathscr{A}_{B \cap P}$, there is a transition $(q, a, q') \in \Delta_B$ without a matching transition $(p, a, p') \in \Delta_P$. The accepting states (q, p) in $\mathscr{A}_{B \cap P}$ are those where q is an accepting state of $\mathscr{A}_B$ and p is an accepting state of $\mathscr{A}_P$.

Only if the above construction could be completed, $B \nVdash P$ potentially holds true and we have to continue exploring the graph structure of $\mathscr{A}_{B \cap P}$.

5 Model Checking IFLTV by Exploring Strongly Connected Components of $\mathscr{A}_{B \cap P}$

Our algorithm is based on a structural analysis of the graph representing $\mathscr{A}_{B \cap P}$. We partition the graph into its maximal *Strongly Connected Components* and *Strongly Connected Bottom Components*:

- A *strongly connected component* (SCC) is a set of nodes of a graph such that for any two nodes v_1 and v_2 in the SCC are paths from v_1 to v_2 and vice versa.
- An SCC is maximal if and only if by adding any addition node to the SCC, the resulting set of nodes is not an SCC anymore.
- A *strongly connected bottom component* (SCBC) is an SCC such that no node outside the SCBC can reached from nodes within the SCBC. Note that SCBC are always maximal.

The model-checking algorithm that we aiming at can now be stated by the following theorem:

Theorem 1. *$B \Vdash P$ if and only if $\mathscr{A}_{B \cap P}$ can be constructed as described in the previous section and all SCBC of $\mathscr{A}_{B \cap P}$ contain at least one accepting state.*

Proof. '$\Rightarrow$': We assume that if $\mathscr{A}_{B \cap P}$ cannot be constructed as defined in the previous chapter or it contains at least one SCBC without any accepting state, then $B \nVdash P$:

Let (q, p) be a state produced during the construction of $\mathscr{A}_{B \cap P}$ which causes the construction to stop. Then there is a transition (q, a, q') in $\mathscr{A}_B$ without a matching transition (p, a, p') in $\mathscr{A}_P$. Let w be a string along a path from (q_0, p_0) to (q, p) in

the partially constructed $\mathscr{A}_{B\cap P}$. Then wa is in $pre(B)$ but not in $pre(B \cap P)$. Hence $B \not\Vdash P$.

If the construction of $\mathscr{A}_{B\cap P}$ completed, but it contains an SCBC without accepting states, then let w be a string along a path in $\mathscr{A}_{B\cap P}$ leading into that SCBC. Then w is in $pre(B)$ but not in $pre(B \cap P)$. Hence $B \not\Vdash P$.

'$\Leftarrow$': We assume $B \not\Vdash P$ and show that either $\mathscr{A}_{B\cap P}$ cannot be constructed as defined in the previous chapter or it contains at least one SCBC without any accepting state:

If $B \not\Vdash P$ then there exists a string w which is in $pre(B)$ but not in $pre(B \cap P)$. Hence w either does not exist along a path from the initial state in $\mathscr{A}_{B\cap P}$ at all — then the construction of $\mathscr{A}_{B\cap P}$ did not complete — or w cannot be continued within $\mathscr{A}_{B\cap P}$ to reach infinitely often an accepting state — which implies that there is an SCBC without accepting states in which continuing w is trapped. □

6 The Non-deterministic Case

As in general, there exist properties requiring non-deterministic Büchi automata to represent them, the model checking algorithm resulting from the previous section does not cover all cases. It seems to be likely that the loss of information when determinising $\mathscr{A}_{B\cap P}$ is insignificant with respect to IFLTV, i.e. we probably can determinise $\mathscr{A}_{B\cap P}$ and then decide $B \not\Vdash P$ as presented in the previous section. This is, however, only a conjecture that we have not proved yet.

7 Conclusion

We presented a model-checking procedure for inherently fair linear-time verification (IFLTV) based on an analysis of strongly connected components in the synchronous product automaton for the behavior and property. We could show that model checking with repsect to IFLTV can, in the case in which the property can be represented by a deterministic Büchi automaton, be reduced to checking that constructing the synchronous product automaton does not ignore any transitions present in the automaton of the behavior, and that the product automaton does not contain strongly connected bottom components without accepting states.

For the case of non-deterministic Büchi properties we conjectured that the construction should be similar — however, a proof of this conjecture is part of future work. Additional future work will be experiments with an implementation of the discussed algorithm.

References

1. B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.
2. S. Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, New York, 1974.

3. N. Francez. *Fairness*. Springer Verlag, New York, first edition, 1986.
4. P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. *Formal Methods in System Design*, 2(2):149–164, April 1993.
5. T. A. Henzinger. Sooner is safer than later. *Information Processing Letters*, 43:135–141, 1992.
6. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems—Specification*. Springer Verlag, New York, first edition, 1992.
7. F. Nießner, U. Nitsche, and P. Ochsenschläger. Deterministic ω-regular liveness properties. In S. Bozapalidis, editor, *Proceedings of the 3rd International Conference on Developments in Language Theory (DLT'97)*, pages 237–247, Thessaloniki, Greece, 1998.
8. U. Nitsche. Application of formal verification and behaviour abstraction to the service interaction problem in intelligent networks. *Journal of Systems and Software*, 40(3):227–248, March 1998.
9. U. Nitsche. *Verification of Co-Operating Systems and Behaviour Abstraction*, volume 7 of *GMD Research Series*. GMD, Sankt Augustin, Germany, 1998. Publication of PhD thesis. ISBN: 3-88457-331-4.
10. U. Nitsche and P. Ochsenschläger. Approximately satisfied properties of systems and simple language homomorphisms. *Information Processing Letters*, 60:201–206, 1996.
11. 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.
12. 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.
13. 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.
14. U. Ultes-Nitsche and S. St James. Testing liveness properties: Approximating liveness properties by safety properties. In M. Kim, B. Chin, S. Kang, and D. Lee, editors, *Formal Techniques for Networked and Distributed Systems, FORTE 2001, IFIP TC6/WG6.1 - 21st International Conference on Formal Techniques for Networked and Distributed Systems, August 28-31, 2001, Cheju Island, Korea*, volume 197 of *IFIP Conference Proceedings*, pages 369–376. Kluwer, 2001.
15. 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)*, pages 241–255, 2003.
16. P. Wolper and P. Godefroid. Partial-order methods for temporal verification. In E. Best, editor, *CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 233–246. Springer Verlag, 1993.