A

Temporal Specifications with Accumulative Values

UDI BOKER, Hebrew University and IST Austria
KRISHNENDU CHATTERJEE, IST Austria
THOMAS A. HENZINGER, IST Austria
ORNA KUPFERMAN, Hebrew University

Recently there has been an effort to add quantitative objectives to formal verification and synthesis. We introduce and investigate the extension of temporal logics with quantitative atomic assertions.

At the heart of quantitative objectives lies the accumulation of values along a computation. It is often the accumulated sum, as with energy objectives, or the accumulated average, as with mean-payoff objectives. We investigate the extension of temporal logics with the *prefix-accumulation assertions* $\mathsf{Sum}(v) \geq c$ and $\mathsf{Avg}(v) \geq c$, where v is a numeric (or Boolean) variable of the system, c is a constant rational number, and $\mathsf{Sum}(v)$ and $\mathsf{Avg}(v)$ denote the accumulated sum and average of the values of v from the beginning of the computation up to the current point in time. We also allow the *path-accumulation assertions* $\mathsf{LimInfAvg}(v) \geq c$ and $\mathsf{LimSupAvg}(v) \geq c$, referring to the average value along an entire infinite computation.

We study the border of decidability for such quantitative extensions of various temporal logics. In particular, we show that extending the fragment of CTL that has only the EX, EF, AX, and AG temporal modalities with both prefix-accumulation assertions, or extending LTL with both path-accumulation assertions, result in temporal logics whose model-checking problem is decidable. Moreover, the prefix-accumulation assertions may be generalized with "controlled accumulation," allowing, for example, to specify constraints on the average waiting time between a request and a grant. On the negative side, we show that this branching-time logic is, in a sense, the maximal logic with one or both of the prefix-accumulation assertions that permits a decidable model-checking procedure. Extending a temporal logic that has the EG or EU modalities, such as CTL or LTL, makes the problem undecidable.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Automata*; F.1.2 [**Computation by Abstract Devices**]: Models of Computation—*Alternation and nondeterminism*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Temporal logic*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages

General Terms: Verification, Theory, Algorithms

Additional Key Words and Phrases: Formal verification, Model checking, Nondeterminism, Temporal Logic, Specification, Accumulation.

1. INTRODUCTION

Traditionally, formal verification has focused on Boolean properties of systems, such as "every request is eventually granted". Temporal logics such as LTL and CTL, as well as automata over infinite objects, have been studied as specification formalisms to express such Boolean properties.

The research was supported by the FWF NFN Grant No S11407-N23 (RiSE), the EU STREP Grant COMBEST, the ERC Advanced Grant QUAREM, the EU NOE Grant ArtistDesign, and a Microsoft Faculty Fellowship.
Authors' address: School of Engineering and Computer Science, Hebrew University, Jerusalem 91904, Israel; IST Austria, Klosterneuburg, Austria.
The present article extends [Boker et al. 2011].

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

In the last years we experience a growing need to extend specification formalisms with quantitative aspects that can express properties such as "the average success-rate is eventually above half", "the total energy of a system is always positive", or "the long-run average of the costs is below 5". Such quantitative aspects of specifications are essential for systems that work in a resource-constrained environment (as an embedded system).[1]

There has recently been a significant effort to study such quantitative-oriented specifications. The approach that has been mostly followed is to consider specific objectives, as mean-payoff or energy-level, by means of weighted automata [Chatterjee et al. 2008; Droste et al. 2009; Alur et al. 2009; Bloem et al. 2009]. No attention, however, has been put in extending temporal logics to provide a general framework for quantitative-oriented specifications. In this work, we introduce and investigate this direction.

When considering quantitative-objectives, one should distinguish between two different aspects. The first is extending the verified system to have numeric variables rather than Boolean ones. The second is extending the specification language to handle accumulative values of variables along a computation.

To understand the difference between the two issues, consider, for example, a Kripke structure with a numeric variable "consumption" that gets a rational value rather than a Boolean one. This alone is of no real interest, as numeric variables over a bounded domain can be encoded by Boolean variables. Hence, one can easily express properties like "the consumption in each state is at most 10" with standard temporal logic.

The main challenge in the quantitative setting is the second issue, namely the accumulation of values. Here, one may wish to specify, for example, that the total consumption, from the beginning of the computation up to the current point in time, is always positive. Note that accumulation is interesting also for systems with only Boolean variables. For example, if the Boolean variable "active" holds exactly when a communication channel is active, one may wish to specify that the activity rate, namely the rate of states in which active is valid, is always above half. It is not hard to see that properties that involve accumulation cannot be specified using standard temporal logics. Indeed, accumulation yields languages that are not ω-regular.

The basic accumulation operators are summation and average. One may formalize them by adding to temporal logics atomic assertions of the form $\gamma \geq \gamma'$, where γ and γ' are arithmetic expressions that use atoms like $\mathsf{Sum}(v)$, $\mathsf{Avg}(v)$, and c, where v is a numeric variable of the system, c is a constant rational number, and $\mathsf{Sum}(v)$ and $\mathsf{Avg}(v)$ denote the accumulated sum and average of the values of v from the beginning of the computation up to the current point in time. Example to basic atomic assertions are $\mathsf{Sum}(v) \geq c$ and $\mathsf{Avg}(v) \geq c$, and one can also have expressions like $\mathsf{Sum}(v) \geq 2 \cdot \mathsf{Sum}(u) + 5$. A natural question that arises is which temporal logics, if at all, can be extended, and with which type of arithmetic expressions, while still allowing for a decidable model-checking problem.

On the positive side, we show that the EF logic (also known as UB^-) [Manna and Pnueli 1979], which is the fragment of CTL with the EF, AG, EX, and AX temporal operators, can be extended with a rich class of arithmetic expressions (we would formally define it below), retaining decidability. We denote the extended logic by EF^Σ. A simple example of an EF^Σ specification is given below.

[1]Different classes of formalisms with quantitative aspects are *real-time* logic and automata [Alur and Henzinger 1994], as well as logics that support *probabilistic* reasoning [Courcoubetis et al. 1991]. The contributions made in these areas are orthogonal to the quantitative aspects that are the subject of this work. Yet, discrete real-time logics that count the number of steps turn out to be special cases of this work, as counting steps can be done by controlled-accumulation. (For details, see Section 3.3.)

Reliable system with energy constraint. Consider a system with a Boolean variable p that is true when the system produces a correct output, and is false when the output is erroneous. The system is reliable if in every computation, the average of correct outputs is always at least 0.95. The system also has a numeric variable v that denotes the energy level, and it must not reach a negative value. The required properties can be specified in EF^{Σ} by: $\text{AG}(\mathsf{Avg}(p) \geq 0.95 \wedge \mathsf{Sum}(v) \geq 0)$.

Moreover, we show that EF^{Σ} can include a rich family of arithmetic expressions: in the atomic assertions $\gamma \geq \gamma'$, both sides can be linear combinations over $\mathsf{Sum}(v)$, $\mathsf{Avg}(v)$, and c, as long as there is no comparison between summation and average. For example, we can have $\mathsf{Sum}(u) - \mathsf{Sum}(v) > 3 \wedge \mathsf{Avg}(u) \geq 2 \cdot \mathsf{Avg}(v)$, but cannot have $\mathsf{Sum}(v) \geq \mathsf{Avg}(u)$. Moreover, the atomic assertions can have *controlled accumulation*, allowing to control when and how the accumulation is done by means of regular expressions. This enables, for example, to specify constraints on the average waiting time between a request and a grant, by accumulating the time-ticks of definable transactions.

The decidability of the logic EF^{Σ} has been a nice surprise for us. Due to the value accumulation, the logic EF^{Σ} has "memoryful semantics": When we unwind the Kripke structure to an infinite tree, the accumulation of values depends on the path taken from the beginning of the computation (the root of the tree) and the current state. Accordingly, different occurrences of the same state may not agree on the set of atomic assertions they satisfy, and hence may also disagree on the satisfaction of formulas. Standard temporal logics have a memoryless semantics, and model-checking algorithms for them heavily depends on this fact. Handling of memoryful logics is much more challenging. For the non-accumulative setting, model checking of memoryful logics is possible thanks to the fact that different histories can be partitioned into finitely many regular languages [Kupferman and Vardi 2006]. In our accumulative setting, there is no bound on the accumulative values and no finite partition is possible.

For that reason, the model-checking procedure is very different from standard model-checking procedures, and is based on a reduction to the validity problem of a Presburger Arithmetic (PA) sentence. That is, given an EF^{Σ} formula φ and a Kripke structure $\mathcal{K}$ with numeric values, we generate a PA sentence θ, such that $\mathcal{K}$ satisfies φ if and only if θ is true. For coping with infinitely many computation paths, we characterize the possible *segments* of the Kripke structure, where a segment consists of a starting-state, an ending state, and a set of edges connecting between them. We show that there are finitely many segments and that it suffices to formulate with PA a "proper computation path over a segment".

On the negative side, we show that EF^{Σ} is, in a sense, the maximal extendable logic. Extending a temporal logic that has either of the temporal operators EG, EU, ER or EW results in a logic whose model-checking problem is undecidable. In particular, CTL and LTL cannot be extended. The undecidability result applies already to an extension with the atomic assertion $\mathsf{Sum}(v) \geq 0$ or $\mathsf{Avg}(v) \geq 0$, and holds even when restricting attention to systems with only Boolean variables. The proof proceeds by a reduction from the halting problem of counter machines. An open problem is whether a logic with the, less standard, operators EFG and EGF (standing for "exists a computation such that eventually-always and always-eventually") can be extended, while retaining decidability.

The logic EF^{Σ} considers *prefix accumulation*, accumulating a value from the beginning of the computation up to the current point of time. It significantly enriches the currently known energy-objectives and opens new directions for specifications with average values and timed-transactions. For *path-accumulation* assertions, in which the accumulation is done along the entire, infinite, computation, referring to the summation is usually useless, as it need not converge. Researchers have thus consider *discounted accumulation* [de Alfaro et al. 2005], or refer to the limit-average of the

accumulated values. We therefore also study the extension of temporal logics with the path-accumulation assertions $\mathsf{LimInfAvg}(v) \geq c$ and $\mathsf{LimSupAvg}(v) \geq c$, for a numeric variable v and a constant number c, referring to the (infimum/supremum of the) long-run average of v along an entire computation. We do not know of other extensions of LTL that capture limit-average (mean-payoff) objectives.

As additional good news we show that LTL can be extended with the path-accumulation assertions $\mathsf{LimInfAvg}(v) \geq c$ and $\mathsf{LimSupAvg}(v) \geq c$, denoted LTL$^{lim\Sigma}$, while allowing for a decidable model checking. This is indeed a nice surprise, as a small fragment of LTL extended with the prefix-accumulation assertion $\mathsf{Avg}(v) \geq c$ is undecidable. The extended logic LTL$^{lim\Sigma}$ significantly enriches the currently known mean-payoff objectives. An example for a specification in LTL$^{lim\Sigma}$ is given below.

Long run happiness. Consider a system with Boolean variables $Wish$ and $ComesTrue$, and numeric variables $Income$ and $Pleasure$. A system is said to be *happy* if every wish eventually comes true or the long run average of both the income and the pleasure are positive. The required properties can be specified by the LTL$^{lim\Sigma}$ formula: $G(Wish \rightarrow F(ComesTrue)) \vee \mathsf{LimInfAvg}(Income) > 0 \wedge \mathsf{LimInfAvg}(Pleasure) > 0$.

Related work. Weighted automata over semirings (i.e., finite automata in which transitions are associated with weights taken from a semiring) have been used to define cost functions, called formal power series for finite words [Schützenberger 1961; Kuich and Salomaa 1986] and ω-series for infinite words [Culik II and Karhumäki 1994; Droste and Kuske 2003; Ésik and Kuich 2004]. In [Chatterjee et al. 2008], new classes of cost functions were studied using operations over rational numbers that do not form a semiring. In [Alur et al. 2009], deterministic weighted automata with mean-payoff objectives were further studied, providing closure under Boolean operations. Several other works have considered quantitative generalizations of languages, over finite words [Droste and Gastin 2007], over trees [Droste et al. 2008], or using finite lattices [Gurfinkel and Chechik 2003; Kupferman and Lustig 2007]. The work of [Droste and Meinecke 2010] gives an extension of MSO to capture weighted mean-payoff automata. All these works consider weighted automata and their expressive power for quantitative specification languages. The extension of temporal logic with accumulation assertions to express quantitative properties of systems has not been considered before.

The model of turn-based games with mean-payoff and energy objectives have been deeply studied in the literature [Zwick and Paterson 1996; Bjorklund et al. 2004; Chakrabarti et al. 2003; Chatterjee et al. 2010]. These works focus on the extension of energy and mean-payoff objectives from the Kripke structure models to game models. Our work, on the other hand, remains with a (quantitative) Kripke structure, while extending the objective by means of temporal logic.

2. THE SETTINGS

In this section we define quantitative Kripke structures – our model for systems with numeric variables, and introduce temporal logics that can specify quantitative aspects of quantitative Kripke structures. Assertions that relate to the current value of a numeric variable, as $v > 7$, are of no interest as they can be expressed in standard, Boolean, temporal logic, by referring to the binary representation of v. We are interested, instead, in assertions like $\mathsf{Sum}(v) > 7$, which refer to the accumulated value of v from the beginning of the computation up to the current time position. Such assertions are no longer ω-regular.

Quantitative Kripke structure. In a Boolean Kripke structure, the variables (atomic propositions) are assigned a Boolean value. Quantitative Kripke structures have both

Boolean and numeric variables, where the latter are assigned rational numbers. Formally, a *quantitative Kripke Structure* is a tuple $\mathcal{K} = \langle P, V, S, s_{in}, R, L\rangle$, with a finite set of Boolean variables P, a finite set of numeric variables V, a finite set of states S, an initial state $s_{in} \in S$, a total transition relation $R \subseteq S \times S$ and a labeling function $L : S \to 2^P \times \mathbb{Q}^V$.

A *computation* of $\mathcal{K}$ is an infinite sequence of states $\pi = s_0, s_1, \ldots$ such that $s_0 = s_{in}$ and $\langle s_i, s_{i+1}\rangle \in R$ for every $i \geq 0$. We denote by $\inf(\pi)$ the set of states that π visits infinitely often, that is $\inf(\pi) = \{s \in S \mid \text{ for infinitely many } i \in \mathbb{N}, \text{ we have that } \pi_i = s\}$.

A quantitative Kripke structure may also have a *fairness condition* α, added as the last element in its definition tuple. A Büchi (unconditional) fairness condition is a set $\alpha \subseteq S$, and a computation π is fair if $\inf(\pi) \cap \alpha \neq \emptyset$.

We denote the labeling (value) of a Boolean variable p and of a numeric variable v in a state s by $[\![p]\!]_s \in \{\mathtt{T}, \mathtt{F}\}$ and $[\![v]\!]_s \in \mathbb{Q}$, respectively. We often talk about Kripke structures, meaning quantitative ones.

Extended temporal logics. We consider two kinds of assertions on accumulative values, for which the accumulation is done either along a prefix of a computation or on the entire, infinite, computation. Let V be a set of numeric variables.

— A *prefix-accumulation assertion over* V is of the form $\gamma \geq \gamma'$, where γ and γ' are linear arithmetic expressions defined over the atoms $c \in \mathbb{Q}$, and $\mathsf{Sum}(v)$ or $\mathsf{Avg}(v)$ for $v \in V$. For example, $\mathsf{Sum}(v) \geq 4$, $\mathsf{Avg}(v) \geq 2\frac{1}{2}$, and $\mathsf{Sum}(v) \geq 2 \cdot \mathsf{Sum}(u) + 5$. A single atomic assertion cannot have both $\mathsf{Sum}()$ and $\mathsf{Avg}()$ (while different atomic-assertions in the same formula can).
— A *path-accumulation assertion over* V is of the form $\mathsf{LimInfAvg}(v) \geq c$ or $\mathsf{LimSupAvg}(v) \geq c$, for $v \in V$ and $c \in \mathbb{Q}$.

Note that prefix-accumulation assertions allow to compare between two different variables, while path-accumulation assertions do not.

We shall investigate the extension of both linear-time and branching-time logics with prefix-accumulation assertions, and the extension of LTL with path-accumulation assertions. For example, the logic CTL extended with prefix-accumulation assertions is denoted CTL$^\Sigma$ and has the following syntax. Let P and V be finite sets of Boolean variables (atomic propositions) and numeric variables, respectively.

— A CTL$^\Sigma$ formula is $p \in P$, a prefix-accumulation assertion over V, $\neg\varphi$, $\varphi_1 \wedge \varphi_2$, $EX\varphi$, $EF\varphi$, $EG\varphi$, or $\varphi_1 EU \varphi_2$, for CTL$^\Sigma$ formulas φ, φ_1, and φ_2.

Of special interest would be the fragment of CTL with the EF and EX temporal operators, in addition to the $\neg$ and $\wedge$ Boolean operators, known in the literature as the EF or UB$^-$ logic [Manna and Pnueli 1979]. We shall denote its extension with prefix-accumulation assertions by EF$^\Sigma$.

The logic LTL extended with path-accumulation assertions is denoted LTL$^{lim\Sigma}$, and has the following syntax, again with respect to sets P and V.

— An LTL$^{lim\Sigma}$ formula is $p \in P$, a path-accumulation assertion over V, $\neg\varphi$, $\varphi_1 \wedge \varphi_2$, $X\varphi$, $F\varphi$, $G\varphi$ or $\varphi_1 U \varphi_2$, for LTL$^{lim\Sigma}$ formulas φ, φ_1 and φ_2.

The semantics of the extended logics is defined with respect to the computation tree of a quantitative Kripke structure. Note that, due to the value accumulation, the extended logics have "memoryful semantics", as opposed to the memoryless semantics of standard CTL and LTL. This is why we define the semantic with respect to the computation tree and not directly with respect to the Kripke structure. We thus start with the definition of trees and computation trees.

Given a finite set D of *directions*, a *D-tree* is a set $T \subseteq D^*$ such that if $x \cdot d \in T$, where $x \in D^*$ and $d \in D$, then also $x \in T$. The elements of T are called *nodes*, and the empty word ε is the *root* of T. The prefix relation induces a partial order $\leq$ between nodes of T. Thus, for two nodes x and y, we say that $x \leq y$ iff there is some $z \in D^*$ such that $y = x \cdot z$. For every $x \in T$, the nodes $x \cdot d$, for $d \in D$, are the *successors* of x. A node is a *leaf* if it has no successors. A *path* of T is a minimal set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $y \in \pi$, either y is a leaf or there exists a unique $d \in D$ such that $y \cdot d \in \pi$. A *path starting in a node* x is a path of the subtree of T whose root is x. For a set Z, a *Z-labeled D-tree* is a pair $\langle T, \tau \rangle$ where T is a D-tree and $\tau : T \to Z$ maps each node of T to an element in Z.

A Kripke structure $\mathcal{K}$ induces a *computation tree* $\langle T_{\mathcal{K}}, \tau_{\mathcal{K}} \rangle$ that corresponds to the computations of $\mathcal{K}$. (See an example in Fig. 1.) Formally, for a Kripke structure $\mathcal{K} = \langle P, V, S, s_{in}, R, L \rangle$, we have that $\langle T_{\mathcal{K}}, \tau_{\mathcal{K}} \rangle$ is a $(2^P \times \mathbb{Q}^V)$-labeled S-tree, where $\mathsf{state}(x)$ denotes the rightmost state in a node x of $T_{\mathcal{K}}$ and $\tau_{\mathcal{K}}(x) = L(\mathsf{state}(x))$.

As has been the case with states in $\mathcal{K}$, we denote the labeling (value) of a Boolean variable p and of a numeric variable v in a node x by $[\![p]\!]_x \in \{\mathtt{T}, \mathtt{F}\}$ and $[\![v]\!]_x \in \mathbb{Q}$, respectively.

For the path quantifiers and the temporal operators, the semantics is as in standard temporal logic. Thus, E stands for "exists a computation", A for "all computations", X for "next", F for "eventually", G for "always", U for "until", R for "release", and W for "weak until". For example, in CTL$^\Sigma$, $[\![E\varphi U\psi]\!]_x = \mathtt{T}$ in a node x of the computation tree, iff there exists a path $\pi = x_1, x_2, \ldots$, with $x_1 = x$, and an index $i \geq 1$ such that $[\![\psi]\!]_{x_i} = \mathtt{T}$ and for every $1 \leq j < i$, we have $[\![\varphi]\!]_{x_j} = \mathtt{T}$.

We define the prefix-accumulation values of a numeric variable v at a node x of the computation tree as follows.

$$[\![\mathsf{Sum}(v)]\!]_x = \sum_{y \leq x} [\![v]\!]_y$$

$$[\![\mathsf{Avg}(v)]\!]_x = \frac{[\![\mathsf{Sum}(v)]\!]_x}{|x| + 1}$$

The Sum and Avg functions can also be defined for a Boolean variable, by viewing it as a numeric variable with $\mathtt{F} = 0$ and $\mathtt{T} = 1$.

The limit-average value along an infinite computation path is intuitively the limit of the average values of its prefixes. However, these average values need not converge, thus a standard solution is to consider their infimum and supremum. We define the path-accumulation values of a numeric variable v along a path $\pi = x_1, x_2, \ldots$ of the computation tree as follows.

— $[\![\mathsf{LimInfAvg}(v)]\!]_\pi = \lim_{n \to \infty} \inf\{[\![\mathsf{Avg}(v)]\!]_{x_i} \mid i \geq n\}$
— $[\![\mathsf{LimSupAvg}(v)]\!]_\pi = \lim_{n \to \infty} \sup\{[\![\mathsf{Avg}(v)]\!]_{x_i} \mid i \geq n\}$

For example, for the computation $\pi = (s_1 s_2)^\omega$ of the Kripke structure in Fig. 1 we have that $[\![\mathsf{LimInfAvg}(v)]\!]_\pi$ is the limit of $\inf\{\frac{3}{1}, \frac{-2}{2}, \frac{1}{3}, \frac{-4}{4}, \frac{-1}{5}, \frac{-6}{6}, \frac{-3}{7}, \frac{-8}{8}, \ldots\} = -1$, which is also $[\![\mathsf{LimSupAvg}(v)]\!]_\pi$. Note that the values of path-accumulation assertions are indifferent to finite prefixes of π. Thus, for all suffixes π' of π, we have that $[\![\mathsf{LimInfAvg}(v)]\!]_\pi = [\![\mathsf{LimInfAvg}(v)]\!]_{\pi'}$, and similarly for LimSupAvg. Accordingly, allowing path-accumulation assertions within the scope of temporal operators does not add to the expressive power of LTL$^{lim\Sigma}$. Yet, we do allow it for convenience; one may prefer to use, for example, the formula $(p \vee \mathsf{LimInfAvg}(v) \geq c) U (q \vee \mathsf{LimSupAvg}(u) \geq c)$, rather than the equivalent formula $(\mathsf{LimSupAvg}(u) \geq c) \vee ((\mathsf{LimInfAvg}(v) \geq c) \wedge F(q)) \vee pUq$.

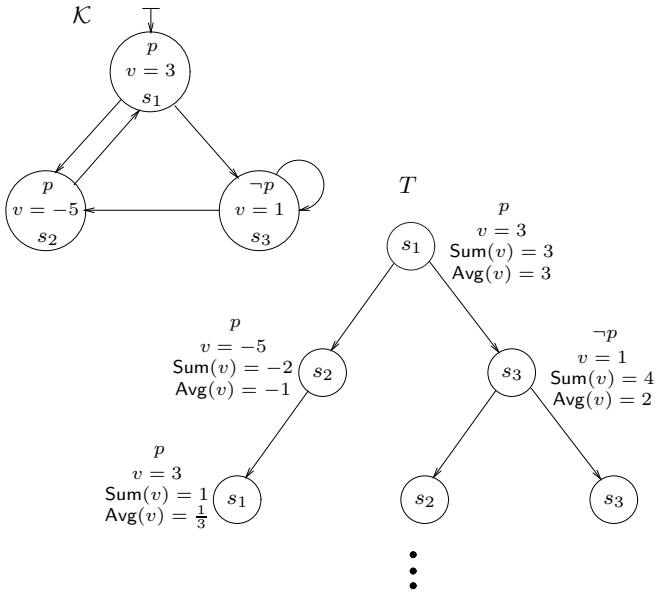


Fig. 1. A quantitative Kripke structure $\mathcal{K}$ and its computation-tree T.

3. TEMPORAL LOGICS WITH PREFIX ACCUMULATION

In this section we consider temporal logics extended by prefix-accumulation assertions. The central question is which of the standard temporal logics, if at all, can be extended while still allowing for a decidable model checking.

One may notice that prefix accumulation takes us from the "comfort zone" of finite state systems into the "hazardous" zone of infinite state systems. Indeed, it is closely related to counter machines and makes our paradigm especially close to model checking Petri-nets. Yet, while model checking is undecidable for Petri-nets and all relevant temporal logics [Esparza 1996], we show that it is decidable for quantitative Kripke structures and specifications in the logic EF^{Σ}. It also turns out that, in a sense, the logic EF is the maximal one that can be extended with prefix accumulation.

In Section 3.1, we show the decidability of the model-checking problem for the logic EF^{Σ}. In Section 3.3, we further extend EF^{Σ} with assertions on controlled accumulation, while retaining the above decidability. These assertions allow, for example, to specify constraints on the average waiting time between a request and a grant. On the other hand, we show in Section 3.4 that adding prefix-accumulation assertions to a temporal logic with any of the other standard temporal operators (that is, EG, EU, ER, or EW) makes the model-checking problem undecidable. In particular, extending CTL and LTL makes them undecidable.

One may first observe that all the prefix-accumulation assertions can be expressed by the $\mathsf{Sum}(v) \geq c$ assertion [2]:

LEMMA 3.1. *Consider a Kripke structure $\mathcal{K}$ and a specification φ in a temporal logic with prefix-accumulation assertions. It is possible to obtain from $\mathcal{K}$ and φ a structure $\mathcal{K}'$ and a specification φ' such that $\mathcal{K}'$ differs from $\mathcal{K}$ only in new numeric variables, φ' differs from φ only in some of the prefix-accumulation assertions, all the prefix-accumulation assertions in φ' are of the form $\mathsf{Sum}(v) \geq c$, and $\mathcal{K} \models \varphi$ iff $\mathcal{K}' \models \varphi'$.*

[2]The $\mathsf{Sum} \geq c$ assertion can be switched to an $\mathsf{Avg} \geq 0$ assertion, by setting an initial value of c to v.

PROOF. Let u and v be numeric variables and c a rational constant. We obtain $\mathcal{K}'$ and φ' as follows.

— For an expression $\mathsf{Sum}(v) \pm \mathsf{Sum}(u)$, we add a new variable v' to $\mathcal{K}'$ that is assigned the value $[\![v']\!]_s = ([\![v]\!]_s \pm [\![u]\!]_s)$ in each state s of the Kripke structure. We then replace $\mathsf{Sum}(v) \pm \mathsf{Sum}(u)$ by $\mathsf{Sum}(v')$. An analogous treatment is given to $\mathsf{Avg}(v) \pm \mathsf{Avg}(u)$.
— We replace an $\mathsf{Avg}(v) \geq \mathsf{Avg}(u)$ assertion by $\mathsf{Sum}(v) \geq \mathsf{Sum}(u)$.
— For a $\mathsf{Sum}(v) \geq \mathsf{Sum}(u)$ assertion, we add a new variable v' to $\mathcal{K}'$ that is assigned the value $[\![v']\!]_s = ([\![v]\!]_s - [\![u]\!]_s)$ in each state s of the Kripke structure. We then replace $\mathsf{Sum}(v) \geq \mathsf{Sum}(u)$ by $\mathsf{Sum}(v') \geq 0$.
— For an $\mathsf{Avg}(v) \geq c$ assertion, we add a new variable v' to $\mathcal{K}'$ that is assigned the value $[\![v']\!]_s = ([\![v]\!]_s - c)$ in each state s of the Kripke structure. We then replace $\mathsf{Avg}(v) \geq c$ by $\mathsf{Sum}(v') \geq 0$.

It is easy to see that, in all nodes of the computation-tree, the original assertions are valid iff the new ones are. Moreover, since the computation-trees of $\mathcal{K}$ and $\mathcal{K}'$ are identical, up to the new variables, the assertion-equivalence extends to formula-equivalence in all temporal logics. □

3.1. Decidability

We show the decidability of the model-checking problem for the logic EF^{Σ}. Given a Kripke structure and a specification, we shall formulate their model-checking problem by a Presburger arithmetic (PA) sentence, such that the sentence is true iff the Kripke structure satisfies the specification.

Presburger Arithmetic. In 1929, Mojżesz Presburger formalized the first order theory of the natural numbers with addition, and showed that it is consistent, complete and decidable [Presburger 1929].

A PA formula is a first order formula with the constants 0 and 1 and the binary function $+$. The PA theory has the following axioms:

— $\forall x.\ \neg(0 = x + 1)$
— $\forall x.\ (x + 1 = y + 1) \rightarrow x = y$
— $\forall x.\ x + 0 = x$
— $\forall x, y.\ (x + y) + 1 = x + (y + 1)$

In addition, the PA theory has the induction scheme: for every PA formula $\theta(x)$, we have that if $\theta(0) \wedge \forall x(\theta(x) \rightarrow \theta(x + 1))$, then $\forall y.\theta(y)$.

The syntax of PA formulas can be extended to contain inequality notions $(\leq, \geq, <, >)$ and rational coefficients. For example, having the statement $\exists x \forall y\ \frac{3}{4}x - 2y < \frac{1}{2}$. The latter can be translated to the sentence $\exists x \forall y \exists z\ \neg(z = 0) \wedge 3x + z = 8y + 2$, maintaining the original truth value.

The PA formulation, in a glance. For convenience, we shall view the Kripke structure $\mathcal{K}$ as having the numeric values on the edges (transitions), rather than in the states. The edges are named $e_1, e_2, \ldots, e_n$, and the value of a variable v on an edge e_i is denoted v_i.

We use the PA variables $x_1, x_2, \ldots, x_n$ in correlation with the edges $e_1, e_2, \ldots, e_n$. Intuitively, a finite path π of $\mathcal{K}$ induces an assignment to the PA variables, describing the number of times that each edge is repeated in π. Using these variables, we can translate, for example, the EF^{Σ} formula $EF(\mathsf{Sum}(v) \geq 3)$ to the PA formula $\exists x_1, x_2, \ldots, x_n.\ \sum_{i=1}^{n} v_i x_i \geq 3$. This follows the approach of [Kosaraju and Sullivan 1988], where linear programming is used rather than Presburger arithmetic.

For handling nested quantifications, there would be a new set of PA variables for every temporal quantifier, while the PA variables of the upper levels are added to the

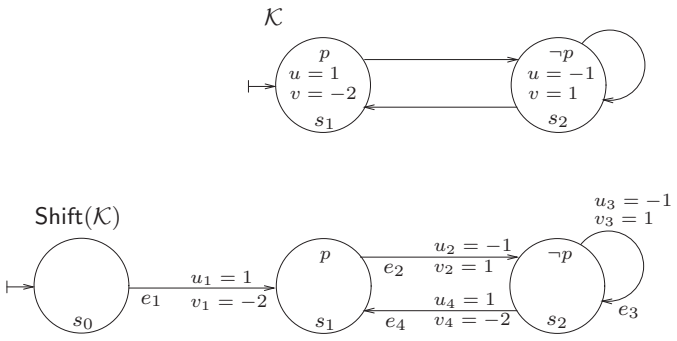


Fig. 2. The Kripke structure $\mathcal{K}$ and its equivalent structure $\mathsf{Shift}(\mathcal{K})$, having the numeric values on the edges.

summation. For example, $EF(\mathsf{Sum}(v) \geq 3 \wedge \neg EF(\mathsf{Sum}(u) = 0))$ would be translated to the PA formula $\exists x_1, x_2, \ldots, x_n.\ \sum_{i=1}^{n} v_i x_i \geq 3 \wedge \neg(\exists y_1, y_2, \ldots, y_n.\ \sum_{i=1}^{n} u_i(x_i + y_i) = 0)$.

The problem is that a valid assignment of the PA variables does not guarantee a valid computation of the Kripke structure – the edge repetition need not match a connected path.

For handling path-connectivity, we define a "segment" of the Kripke structure to be a triple, of a starting-state, an ending state, and a set of edges connecting between them. For every segment κ of $\mathcal{K}$, we formalize in PA the assertion that "the edge repetition corresponds to a connected path over the segment κ". Namely, we assert that all the edges of the segment are used, and no edge but them, as well as that the number of times a state is entered is equal to the number of times it is left, with the exception of the starting and ending states. The latter assertion is an adjustment of Kirchhoff's circuit laws.

We then change, top to bottom, every EF or EX subformula into a disjunction of identical subformulas, each in conjunction with a specific segment. The starting state of the segments in an inner formula is taken to be the ending state of the segment in the upper-level formula.

In the rest of this section, we formalize this PA formulation and prove its correctness.

Moving the numeric values to the edges. It is a common practice to switch between the values of the states and the edges, for example in the process of translating a Kripke structure to an automaton. For convenience, we move the numeric variables to the edges, while keeping the Boolean variables in the states.

The translation (see Fig. 2) adds a new state, s_0, as the new initial state, and a transition from s_0 to the original initial state. Every numeric variable v in a state s is moved to all the incoming edges of s. The edges are named $e_1, e_2, \ldots, e_n$, and the value of a variable v on an edge e_i is denoted v_i.

Given a Kripke structure $\mathcal{K}$ and a specification φ, we translate $\mathcal{K}$ to $\mathsf{Shift}(\mathcal{K})$ as above, and change the specification φ to $\mathsf{Shift}(\varphi)$, referring to the next state. In the case of a linear-time specification, we define $\mathsf{Shift}(\varphi)$ as $X\varphi$ and with a branching-time specification, we define $\mathsf{Shift}(\varphi)$ to be $AX\varphi$ or $EX\varphi$ (since s_0 has a single successor, path quantification is not important).

PROPOSITION 3.2. *Consider a Kripke structure $\mathcal{K}$ and a temporal logic specification φ. Then $\mathcal{K} \models \varphi$ iff $\mathsf{Shift}(\mathcal{K}) \models \mathsf{Shift}(\varphi)$.*

Segments. Recall that in our PA formulation of the model-checking problem, the PA variables denote the number of times that each edge of the Kripke structure is re-

peated in a satisfying path. Yet, an arbitrary edge-repetition need not correspond to a connected path. For formalizing this constraint in PA, we define the "segments" of a Kripke structure. A segment is a triple, of a starting-state, ending state, and a set of edges connecting between them.

Formally, for a path p (not necessarily simple) in a directed graph, we denote by $\mathsf{Edges}(p)$ the set of edges that appear in p.

Definition 3.3. Given a Kripke structure $\mathcal{K}$ with states S and edges E, we define a *segment* of $\mathcal{K}$ to be a triple $\langle a, b, C\rangle$ with a starting state $a \in S$, an ending state $b \in S$, and a set of edges $C \subseteq E$, such that there is a path p from a to b with $\mathsf{Edges}(p) = C$. Note that C may be the empty set.

Since every edge appears at most once in every segment, a Kripke structure has finitely many segments. For example, the structure $\mathsf{Shift}(\mathcal{K})$ of Fig. 2 has the following segments:

$$
\begin{array}{lll}
\kappa_1 = \langle s_0, s_0, \emptyset\rangle & & \\
\kappa_2 = \langle s_0, s_1, \{e_1\}\rangle & \kappa_3 = \langle s_0, s_1, \{e_1, e_2, e_4\}\rangle & \kappa_4 = \langle s_0, s_1, \{e_1, e_2, e_3, e_4\}\rangle \\
\kappa_5 = \langle s_0, s_2, \{e_1, e_2\}\rangle & \kappa_6 = \langle s_0, s_2, \{e_1, e_2, e_3\}\rangle & \kappa_7 = \langle s_0, s_2, \{e_1, e_2, e_3, e_4\}\rangle \\
\kappa_8 = \langle s_1, s_1, \emptyset\rangle & \kappa_9 = \langle s_1, s_1, \{e_2, e_4\}\rangle & \kappa_{10} = \langle s_1, s_1, \{e_2, e_3, e_4\}\rangle \\
\kappa_{11} = \langle s_1, s_2, \{e_2\}\rangle & \kappa_{12} = \langle s_1, s_2, \{e_2, e_3\}\rangle & \kappa_{13} = \langle s_1, s_2, \{e_2, e_3, e_4\}\rangle \\
\kappa_{14} = \langle s_2, s_1, \{e_4\}\rangle & \kappa_{15} = \langle s_2, s_1, \{e_3, e_4\}\rangle & \kappa_{16} = \langle s_2, s_1, \{e_2, e_3, e_4\}\rangle \\
\kappa_{17} = \langle s_2, s_2, \emptyset\rangle & \kappa_{18} = \langle s_2, s_2, \{e_3\}\rangle & \kappa_{19} = \langle s_2, s_2, \{e_2, e_3, e_4\}\rangle
\end{array}
$$

PA formulation of a connected path. Equipped with the notion of a segment, we may formalize in PA the assertion that "an edge repetition-set corresponds to a connected path" by a disjunction of the assertions "an edge repetition-set corresponds to a connected path on a segment κ" over all relevant segments. For each segment, the corresponding PA formula will be an adjustment of Kirchhoff's circuit laws.

Definition 3.4 (*PA formulation of a segment*). Consider a Kripke structure $\mathcal{K}$ with states S and edges $E = \{e_1, e_2, \ldots, e_n\}$. We denote the set of indices of the incoming edges to a state $s \in S$ by $\mathsf{In}(s)$ and of the outgoing edges by $\mathsf{Out}(s)$. For a segment $\kappa = \langle a, b, C\rangle$ of $\mathcal{K}$, we define its PA formula, ψ_κ, to be the conjunction of the following formulas, over the PA variables $x_1, x_2, \ldots, x_n$:

— For every i such that $e_i \in C$, the formula $x_i \geq 1$.
— For every j such that $e_j \in E \setminus C$, the formula $x_j = 0$.
— If $a = b$ (i.e., a cycle), then:
  — For every state $s \in S$, the formula $\sum_{i \in \mathsf{In}(s)} x_i = \sum_{j \in \mathsf{Out}(s)} x_j$.
— If $a \neq b$ (i.e., not a cycle), then:
  — For every $s \in S \setminus \{a, b\}$, the formula $\sum_{i \in \mathsf{In}(s)} x_i = \sum_{j \in \mathsf{Out}(s)} x_j$.
  — The formula $\sum_{i \in \mathsf{Out}(a)} x_i = (\sum_{j \in \mathsf{In}(a)} x_j) + 1$.
  — The formula $\sum_{i \in \mathsf{In}(b)} x_i = (\sum_{j \in \mathsf{Out}(b)} x_j) + 1$.

For example, the PA formula of the segment $\kappa = \langle s_0, s_2, \{e_1, e_2, e_3\}\rangle$ of the structure $\mathsf{Shift}(\mathcal{K})$ of Fig. 2 is $\psi_\kappa =$

$$
\begin{array}{ll}
x_1 \geq 1 \ \wedge \ x_2 \geq 1 \ \wedge \ x_3 \geq 1 \ \wedge \ x_4 = 0 & \text{(edges)} \\
\wedge \ x_1 + x_4 = x_2 & \text{(internal states)} \\
\wedge \ x_1 = 0 + 1 \ \wedge \ x_3 + x_4 = x_2 + x_3 - 1 & \text{(start and end states)}
\end{array}
$$

It is easy to see that the edge repetition-set, $x_1, x_2, \ldots, x_n$, of a connected path over a segment κ satisfies the PA formula $\exists x_1, x_2, \ldots, x_n.\ \psi_\kappa$. Furthermore, the opposite is also true, as shown below. The reason is that Kirchhoff's circuit laws guarantee a set of proper cycles, while the requirement to visit all the segment-edges guarantees that these cycles can be connected.

LEMMA 3.5. *Consider a segment $\kappa = \langle a, b, C\rangle$ of a Kripke structure $\mathcal{K}$ with states S and edges $E = \{e_1, e_2, \ldots, e_n\}$. There is a path p from a to b with $\mathsf{Edges}(p) = C$ iff the PA formula, $\exists x_1, x_2, \ldots, x_n.\ \psi_\kappa$, as defined above, is valid. Moreover, every assignment $x_1, x_2, \ldots, x_n$ with which ψ_κ holds corresponds to the number of times that each edge e_i is repeated in a path p, and vice versa.*

PROOF. Given a path p from a to b over C, it is easy to see that the edge repetitions of p provide a solution to the PA formula.

As for the other direction, we will iteratively generate a path p from the formula solution $x_1, x_2, \ldots, x_n$. We call the PA variable x_i the "counter of the edge e_i", and decrease it by 1 once we use e_i.

— Step I - the skeleton path.
  (1) Start from the state a.
  (2) Arbitrarily choose an edge e_i from the current state, whose counter x_i is not 0. Decrease x_i by one.
  (3) Continue with step (2) above with respect to the ending state of e_i, until reaching a state for which all the outgoing edges have zeroed counters.
— Step II - the added cycles.
  If there are still positive counters:
  (1) Choose a state s in p that has an outgoing edge with a positive counter.
  (2) Continue from s, as in step I.2.
  (3) The zeroed-counter state, which we stop on, must be s. Add this cycle as a loop in the first occurrence of s in p.
  (4) Repeat step II until all edge-counters are zeroed.

We should prove the following claims:

— Step I ends in b.
— Step II.1 is always possible when there are positive counters.
— Step II always produces cycles.

The correctness of the first and third claims follows from the In-Out edge counting. As for the second claim, let s' be the source state of an edge with a positive counter. Since s' is reachable from a along edges in the segment-edges C, there is some corresponding path $p' = a \to s'_1 \to s'_2 \to \ldots s'$, all of whose edges are in C. Let e be the first edge in p' with a positive counter. We will choose s to be the source-state of e.

It is left to show that $s \in p$. If $s = a$ we are done. Otherwise, since all the edges of p' must be used at least once, and the edge before s has a zeroed counter, we know that it has been used, implying that s belongs to the generated path p. □

Translating temporal logic into Presburger arithmetic. We can now describe the formulation of the model-checking problem for $\mathcal{K}$ and φ by means of a PA formula. We do so by defining a recursive procedure, $\mathsf{Trans}(\xi, s, Y)$, that gets as input an EF^Σ formula ξ, a state s of $\mathsf{Shift}(\mathcal{K})$, and a finite set Y of n-tuples of PA variables, and returns a PA formula that is valid iff the state s of $\mathsf{Shift}(\mathcal{K})$ satisfies ξ under the assumption that s has been reached along a path described by Y (we formalize this below). Accordingly, model checking of φ in $\mathcal{K}$ is reduced to checking the validity of $\mathsf{Trans}(\mathsf{Shift}(\varphi), s_0, \emptyset)$.

Consider a set Y of n-tuples of PA variables, say $Y = \{\langle x_1^1, \ldots, x_n^1\rangle, \ldots, \langle x_1^k, \ldots, x_n^k\rangle\}$. We write $\sum Y_i$ as a shortcut for $\sum_{j=1}^{k} x_i^j$. In the procedure, we use the symbol κ to denote a segment of $\mathsf{Shift}(\mathcal{K})$, and ψ_κ to denote its PA formulation, as in Definition 3.4. All the PA quantifications use new PA variables.

The formula $\mathsf{Trans}(\xi, s, Y)$ is defined according to the structure of ξ as follows.

— $\mathsf{Trans}(\neg\xi, s, Y) = \neg\mathsf{Trans}(\xi, s, Y)$.
— $\mathsf{Trans}(\xi_1 \wedge \xi_2, s, Y) = \mathsf{Trans}(\xi_1, s, Y) \wedge \mathsf{Trans}(\xi_2, s, Y)$.
— $\mathsf{Trans}(p, s, Y) = [\![p]\!]_s$, for an atomic proposition p.
— $\mathsf{Trans}(\mathrm{EF}\xi, s, Y) = \exists x_1, \ldots, x_n. \bigvee_{\kappa=\langle s,b,C\rangle} \psi_\kappa \wedge \mathsf{Trans}(\xi, b, Y \cup \{\langle x_1, \ldots, x_n\rangle\})$.
— $\mathsf{Trans}(\mathrm{EX}\xi, s, Y) = \mathsf{Trans}(\mathrm{EF}\xi, s, Y) \wedge \sum_{i=1}^{n} x_i = 1$. [3]
— $\mathsf{Trans}(\mathsf{Sum}(v) \geq c, s, Y) = \sum_{i=1}^{n}(v_i \sum Y_i) \geq c$, where v_i is the value of the Kripke-variable v on the edge e_i.

We can now use Trans for the decidability of the model-checking problem.

THEOREM 3.6. *Given a quantitative Kripke structure $\mathcal{K}$ and a specification φ in EF^Σ, it is decidable to check whether $\mathcal{K}$ satisfies φ.*

PROOF. We prove that the PA formula $\mathsf{Trans}(\mathsf{Shift}(\varphi), s_0, \emptyset)$ is valid iff $\mathsf{Shift}(\mathcal{K}) \models \mathsf{Shift}(\varphi)$. By Proposition 3.2, the latter holds iff $\mathcal{K} \models \varphi$. The proof proceeds by induction on the nesting level of temporal operators in φ.

The base of the induction is a formula with a single temporal operator. For a single segment, the translation correctness follows from Lemma 3.5. By the disjunction on all the segments that start in the designated state, we get the correctness with respect to the whole Kripke structure.

As for the induction step, it directly corresponds to the recursive step in the PA-formulation procedure: setting the starting state of the inner segment to be the ending state of the upper level ensures a correct path, and the addition of the PA variables of the upper level to the summation in the inner level ensures a proper calculation of the accumulated variable values. □

Note that model checking an EF^Σ formula is also decidable with respect to a quantitative Kripke structure with a fairness condition. The reason is that a fairness condition only depends on computation suffixes, while an EF formula only depends on computation prefixes. Indeed, consider a Kripke structure $\mathcal{K}$ with states S and a fairness condition α. Let $D \subseteq S$ be the "dead-end states" of $\mathcal{K}$, from which no computation of $\mathcal{K}$ satisfies α. Consider the unfair Kripke structure $\mathcal{K}'$ over the restriction of $\mathcal{K}$ to $S \setminus D$. Then, for an EF^Σ formula of the form $EF\xi$ (or $EX\xi$), one can see that $\mathcal{K}$ has a fair computation that satisfies $EF\xi$ iff $\mathcal{K}'$ has a computation that satisfies $EF\xi$.

3.2. Complexity

The decidability proof presented in Section 3.1 reduces the problem of model checking an EF^Σ formula to the validity problem of a Presburger arithmetic formula. In this section, we study the complexity of the model-checking problem in detail, showing that the two problems are roughly equivalent, complexity wise. We therefore have a triply-exponential deterministic upper bound [Oppen 1978; Reddy and Loveland 1978], and a doubly-exponential nondeterministic lower bound [Fischer and Rabin 1974]. Moreover, the lower bound of [Fischer and Rabin 1974] shows hardness in 2NEXPTIME, thus, by the wide belief that non-deterministic algorithms can be exponentially faster

[3] The disjunction in the formula $\mathsf{Trans}(\mathrm{EF}\xi, s, Y)$ may be restricted to segments with a single edge, or alternatively be replaced with a straightforward disjunction on the outgoing edges of s.

than deterministic ones, the triply-exponential upper bound cannot be significantly improved.

Upper bound. The upper bound follows from the construction of Theorem 3.6 together with the algorithm of [Reddy and Loveland 1978] for checking the validity of a PA formula. Formally, we have the following.

THEOREM 3.7. *Model checking a Kripke structure with n states and an EF$^\Sigma$ formula of length m and nesting level d of EF operators, can be done in a deterministic time complexity of $2^{2^{O(2^{n\times d}+m)^{O(d)}}}$. Thus, assuming $d \leq m \leq n$, it is in $2^{2^{2^{O(n^3)}}}$ (i.e., in deterministic triply-exponential time).*

PROOF. Given a Kripke structure with n states and an EF$^\Sigma$ formula of length m and nesting level d of EF operators, the algorithm of Theorem 3.6 constructs a PA formula of length in $O(2^{n\times d}+m)$ with quantifier nesting level d. Upfront, since the PA formula is exponentially longer than the input to the model-checking problem, and checking the validity of a PA formula is triply exponential [Oppen 1978], we get a four-exponent bound. However, a closer analysis of the quantifier-elimination technique, used for checking the validity of a PA formula, provides a $2^{2^{O(a)^{O(b)}}}$ upper bound for a PA formula of length a with nesting level b [Reddy and Loveland 1978]. Noting that the construction in Theorem 3.6 exponentially widens the formula, but does not make it any deeper, we remain with a triply-exponential upper bound. More precisely, the bound is $2^{2^{O(2^{n\times d}+m)^{O(d)}}}$, and assuming $d \leq m \leq n$, it is in $2^{2^{2^{O(n^3)}}}$. □

Lower bound. We now proceed to the lower bound. We show a polynomial reduction from the validity problem of a PA formula. That is, given a PA formula η, we construct in polynomial time an EF$^\Sigma$ formula φ and a Kripke structure $\mathcal{K}$, such that φ is valid iff $\mathcal{K}$ satisfies φ.

Intuitively, φ is the same as η, except for replacing the logic existential quantifier $\exists$ with the temporal existential quantifier EF. The Kripke structure $\mathcal{K}$ is constructed in a way that follows the nesting hierarchy of the quantifiers in η: For every PA variable x_i, there is a state q_i in $\mathcal{K}$ with a self loop, and a state q_i has a transition to a state q_j iff the PA variable x_j is in the scope of the PA variable x_i. Instantiating η with a specific value m to x_i is then equivalent to taking a computation tree of $\mathcal{K}$ that loops m times in q_i. For ensuring that the computation tree properly alternates between the quantifiers, all transitions from a state q_i goes through a state q'_i, having a Boolean variable b_i, to which φ refers.

The reduction is formally defined below and is illustrated in Fig. 3. It is done in three steps:

— Given a PA formula η, we first translate it to a normal form, such that i) the variables get unique names, $x_1, x_2, \ldots, x_k$, and ii) every atomic proposition (which is originally in the form of $t = s$, for PA terms s and t), gets the form $\sum_{i=1}^{k} a_i x_i = c$, where a_i and c are integers. For simplicity, we refer to the normal-formed formula as η, as well.
— The Kripke structure $\mathcal{K}$ is constructed as follows. For every variable x_i of η, $\mathcal{K}$ has the states q_i and q'_i, as well as the numeric variable v_i and the Boolean variable b_i. The value of v_i is 1 in q_i and 0 everywhere else, and the value of b_i is `T` in q'_i and `F` everywhere else. For every $1 \leq i \leq k$, there is a transition from q_i to itself and to q'_i. If a variable x_i is in the topmost level (not in the scope of any other variable) then q_i and q'_i are initial states of $\mathcal{K}$. If a variable x_j is in the immediate scope of a variable x_i (meaning that it is in the scope of x_i and there is no variable x_h, for $h \notin \{i, j\}$, such

that x_j is in the scope of x_h and x_h is in the scope of x_i) then there is a transition from the state q'_i to both q_j and q'_j.

— The EF^Σ formula φ is constructed with the following recursive procedure tr that translates a PA formula into an EF^Σ formula.
  — $\mathsf{tr}(\exists x_i.\eta') = EF(b_i \wedge \mathsf{tr}(\eta'))$;
  — $\mathsf{tr}(\sum_{i=1}^{k} a_i x_i = c) = \sum_{i=1}^{k} a_i \mathsf{Sum}(v_i) = c$;
  — $\mathsf{tr}(\neg\eta_1) = \neg\mathsf{tr}(\eta_1)$; and
  — $\mathsf{tr}(\eta_1 \wedge \eta_2) = \mathsf{tr}(\eta_1) \wedge \mathsf{tr}(\eta_2)$.

The PA formula η:

$$\exists x.(\neg\exists y.\ (x + x = 2y + y + 1 + 1 \wedge \exists z.\ z = y + y + 3) \wedge \exists y.\ 2y = x)$$

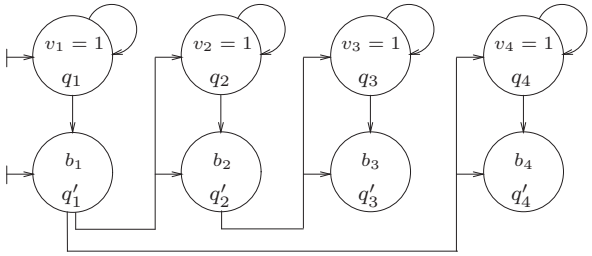
The PA formula η in normal form:

$$\exists x_1.(\neg\exists x_2.\ (2x_1 - 3x_2 = 2 \wedge \exists x_3.\ -2x_2 + x_3 = 3) \wedge \exists x_4.\ -x_1 + 2x_4 = 0)$$

The EF^Σ formula φ:

$$EF(b_1 \wedge \neg EF(b_2 \wedge 2\mathsf{Sum}(v_1) - 3\mathsf{Sum}(v_2) = 2 \wedge EF(b_3 \wedge -2\mathsf{Sum}(v_2) + \mathsf{Sum}(v_3) = 3)) \wedge \\ \wedge EF(b_4 \wedge -2\mathsf{Sum}(v_1) + \mathsf{Sum}(v_2) = 0))$$

The Kripke structure $\mathcal{K}$:



The equivalence: η is valid if and only if $\mathcal{K}$ satisfies φ.

Fig. 3. A reduction from the validity problem of a Presburger-arithmetic formula to the model-checking problem of a Kripke structure and an EF^Σ formula.

The correctness of the above construction is proved in the following lemma.

LEMMA 3.8. *Consider a PA formula η of length n. There is a polynomial-time construction of a quantitative Kripke structure $\mathcal{K}$ with $O(n)$ states, and an EF^Σ formula φ of length in $O(n)$, such that η is valid iff $\mathcal{K}$ satisfies φ.*

PROOF. We prove the correctness of the above construction by induction on the structure of η and its corresponding Kripke structure $\mathcal{K}$.

Note that a subformula η' of η may have some variables that are bound to quantifiers, for example x_3 and x_4, and others that are unbound, for example x_1 and x_2. We shall express it by writing $\eta'(x_1, x_2)$, meaning that x_1 and x_2 are the parameters of η'.

Analogously to parameterized formulas, we introduce a "parameterized Kripke structure" to be a quantitative Kripke structure with a single initial state, where some of the numeric variables have no value in this state. The initial values of these variables are given as parameters. For example, a parameterized Kripke structure $\mathcal{K}'$ in which the numeric variables v_1 and v_2 are parameters is written $\mathcal{K}'(v_1, v_2)$. Then, we instantiate $\mathcal{K}'$ to a standard (i.e., non-parametrized) quantitative Kripke structure by

providing specific values to v_1 and v_2. That is, $\mathcal{K}'(m_1, m_2)$, for some specific values m_1 and m_2, is a quantitative Kripke structure, where $v_1 = m_1$ and $v_2 = m_2$ in the initial state.

Having the notion of a parameterized Kripke structure, we can define the steps of the inductive proof: For a subformula η' of η, the corresponding parameterized Kripke structure $\mathcal{K}'$ is a substructure of $\mathcal{K}$, having only states q_i and q'_i such that x_i is bound in η', as well as a new initial state. A variable v_i has the value 0 in the initial state of $\mathcal{K}'$ if x_i is bound in η', and it is a parameter of $\mathcal{K}'$ otherwise. The new initial state is the only initial state of $\mathcal{K}'$, and it has transitions to all other top-level states of $\mathcal{K}'$ (except for itself), meaning to states with no other incoming transitions. We then show that for every tuple of values M, where the size of M is the number of unbound variables in η', it holds that $\eta'(M)$ is valid iff $\mathcal{K}'(M)$ satisfies φ'.

We now proceed to the induction proof itself. The base case is an atomic proposition $\eta'(x_1, \ldots, x_k)$ of η, in the form of $\sum_{i=1}^{k} a_i x_i = c$. The corresponding EF^{Σ} formula φ' is $\sum_{i=1}^{k} a_i \mathsf{Sum}(v_i) = c$, and the corresponding Kripke structure $\mathcal{K}'$ has a single state (the new initial state), in which all the numeric variables are parameters. For every specific values $m_1, \ldots, m_k$ it holds that $\eta'(m_1, \ldots, m_k)$ is valid iff $\mathcal{K}'(m_1, \ldots, m_k)$ satisfies φ', since $\mathcal{K}'$ has no transitions and for every $1 \leq i \leq k$, $\mathsf{Sum}(v_i) = m_i$.

In the induction step, we should handle the negation, the conjunction, and the existential quantifier.

The correctness with respect to negation is straightforward, since the Kripke structure does not change, and the semantics of negation is the same in Presburger arithmetic and in EF^{Σ}. That is, a PA formula $\neg\eta'$ is valid iff η' is not valid and a Kripke structure $\mathcal{K}'$ satisfies an EF^{Σ} formula $\neg\varphi'$ iff $\mathcal{K}$ does not satisfy φ'. (Note that this is true for a branching temporal logic, such as EF^{Σ}, but not for a linear temporal logic.)

The case of a conjunction is more involved, since the Kripke structure might change. Consider the formula $\eta'(X_1 \cup X_2) = \eta_1(X_1) \wedge \eta_2(X_2)$, where X_1 and X_2 are tuples of unbound variables. Let φ_1 and φ_2 be the EF^{Σ} formulas for η_1 and η_2, respectively, and let $\mathcal{K}_1(V_1)$ and $\mathcal{K}_2(V_2)$ be the Kripke structures for η_1 and η_2, respectively, where V_1 and V_2 are tuples of numeric variables. Then, the Kripke structure for η', $\mathcal{K}'(V_1 \cup V_2)$, has all the states of $\mathcal{K}_1(V_1)$ and $\mathcal{K}_2(V_2)$, except for their initial states, and its initial state has the parameterized numeric variables of both of them. For showing the correctness of the construction, we should show that for every instantiations M_1 and M_2 of V_1 and V_2, respectively, it holds that φ_1 is satisfied by $\mathcal{K}_1(M_1)$ iff it is satisfied by $\mathcal{K}'(M_1 \cup M_2)$, and analogously for φ_2 with respect to $\mathcal{K}_2(M_2)$ and $\mathcal{K}'(M_1 \cup M_2)$. Indeed, this can be shown by induction on the structure of φ_1 (and analogously φ_2). The base case of the induction, as well as the induction step with respect to negation and to conjunction is straightforward. As for the induction step with respect to the existential quantifier, let $\varphi_1 = EF(b_i \wedge \psi)$, for some $i \in \{1, \ldots, k\}$. If $\mathcal{K}_1$ satisfies φ_1 then so does $\mathcal{K}'$, since its computation tree includes $\mathcal{K}_1$'s computation tree. If $\mathcal{K}_1$ does not satisfy φ_1 then $\mathcal{K}'$ also cannot satisfy it, since all the states in its computation tree that are not a part of $\mathcal{K}_1$'s computation tree assign `F` to b_i.

It is left to show the correctness with respect to the existential quantifier. Consider the subformula $\eta'(X) = \exists x_i.\eta''(X, x_i)$, where $i \in 1, \ldots, k$ and X is a tuple of unbound variables. Let φ'' and $\mathcal{K}''(V, v_i)$ be the EF^{Σ} formula and the Kripke structure, respectively, for η'', where V is a tuple of numeric variables. The EF^{Σ} formula φ', for η', is defined by the above construction to be $EF(b_i \wedge \varphi'')$. The Kripke structure $\mathcal{K}'$, for η', is the same as $\mathcal{K}''$, except for replacing the initial state of $\mathcal{K}''$ with the following three states: a new initial state, having the numeric variables of V as parameters, from which there are transitions to q_i and to q'_i. The outgoing transitions of $\mathcal{K}''$'s initial state are the outgoing transitions from q'_i in $\mathcal{K}'$. Consider an instantiation of X and of V

with a tuple M of specific values. Now, if $\eta'(M)$ is valid, then there exists a value m such that $\eta''(M,m)$ is valid. By the induction assumption, $\mathcal{K}''(M,m)$ satisfies φ''. Thus, the following path p in $\mathcal{K}'(M)$ satisfies $b_i \wedge \varphi''$: if $m = 0$ then p goes directly from the initial state to q'_i, otherwise it goes to q_i, makes $m-1$ self loops and then goes to q'_i. The computation tree starting in q'_i is the same as the computation tree of $\mathcal{K}''$, thus satisfying φ''. Therefore, $\mathcal{K}'(M)$ satisfies φ'. As for the other direction, assume that $\mathcal{K}'(M)$ satisfies φ'. Then there exists a path p that satisfies $(b_i \wedge \varphi'')$. Since b_i is true only in q'_i, it follows that p ends in q'_i, after making some $m \geq 0$ self loops in q_i. Hence, $\mathcal{K}''(M,m)$ satisfies φ'', and by the induction assumption $\eta''(M,m)$ is valid. Hence, $\eta'(M)$ is valid.

□

COROLLARY 3.9. *Model checking EF$^{\Sigma}$ is 2NEXPTIME-hard.*

PROOF. Directly follows from Lemma 3.8 and the known lower bound for the validity problem of a Presburger arithmetic formula [Fischer and Rabin 1974]. □

3.3. Controlled Accumulation

One may wish to have some control on when and how the accumulation is done, in order, for example, to make assertions on the average waiting time between a request and a grant. For the latter, we need the accumulative-sum of the time-ticks between the requests and their corresponding grants, divided by the number of such request-grant transactions.

Viewing the period between a request and a grant as a "transaction", one may wish to further generalize the accumulation with respect to transactions. For example, handling discontinuous transactions, speaking about their average cost, and setting different importance values to their different occurrences.

All that, and more, can be done by adding the following *controlled accumulation* atomic-assertion to the logic: $\mathsf{cAvg}(u, r_1, v, r_2) \geq c$, for a numeric variable u, a positive numeric variable v, regular expressions r_1 and r_2 over 2^P, and a constant c. The value of a controlled average at a node x of the computation tree is defined as follows (we use $r(y)$ to indicate that the prefix y is a member in the language of the regular expression r).

$$[\![\mathsf{cAvg}(u, r_1, v, r_2)]\!]_x = \frac{\sum_{(y \leq x \mid r_1(y))} [\![u]\!]_y}{\sum_{(y \leq x \mid r_2(y))} [\![v]\!]_y}.$$

Intuitively, r_1 indicates whether the current point in time is relevant to the transaction, according to which we sum-up the costs u, while r_2 indicates a new transaction occurrence. The value of v indicates the importance of the transaction-occurrence, denoting its influence on the averaging.

Note that the controlled average is undefined before the first true valuation of r_2. Indeed, there is no meaning to a transaction average before the first transaction occurrence.

Controlled-average can obviously express standard summation and averaging. Indeed, for all nodes x, we have that

$$\begin{aligned} [\![\mathsf{Sum}(u)]\!]_x &= [\![\mathsf{cAvg}(u, \mathtt{T}, 1, \text{“First computation step”})]\!]_x \\ [\![\mathsf{Avg}(u)]\!]_x &= [\![\mathsf{cAvg}(u, \mathtt{T}, 1, \mathtt{T})]\!]_x \end{aligned}$$

For example, the average waiting time between a request (denoted p) and a grant (denoted q) over an alphabet Σ can be defined by: $\mathsf{cAvg}(1, r_1, 1, r_2)$, where $r_1 = \Sigma^* p (\Sigma \setminus q)^*$ describes all prefixes with a request that is not yet granted, and $r_2 = (\varepsilon + \Sigma^* q)(\Sigma \setminus p)^* p)$ describes all prefixes in which a request that needs a grant has been issued.

Thus, $\mathsf{cAvg}(1, r_1, 1, r_2)$ is the sum of the waiting durations divided by the number of requests. Another interesting special case is when $r_1 = r_2$, providing the ratio between the summations of two variables.

Decidability. We show that adding controlled-average assertions to the logic EF^{Σ} preserves the decidability of the model-checking problem.

We first reduce the problem to model checking assertions of the form $\mathsf{cAvg}(u, p_1, v, p_2) \geq c$, for Boolean variables p_1 and p_2. The semantics is the expected one: the values of u and v are taken into an account only in states in which p_1 and p_2 are valid, respectively. In order to talk about p_1 and p_2 rather than r_1 and r_2, we refer to the product $\mathcal{K} \times \mathcal{A}_1 \times \mathcal{A}_2$ of the Kripke structure $\mathcal{K}$ and the deterministic finite automata $\mathcal{A}_1$ and $\mathcal{A}_2$ for r_1 and r_2, in which p_1 and p_2 are true in the accepting states of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. (A product of a Kripke structure and an automaton is formally described in Section 4.) Note that since $\mathcal{A}_1$ and $\mathcal{A}_2$ are deterministic, then for every node x in the computation tree of $\mathcal{K}$, there are unique states of $\mathcal{A}_1$ and $\mathcal{A}_2$ that correspond to x, which we denote by $\mathcal{A}_1(x)$ and $\mathcal{A}_2(x)$, respectively. It is easy to see that $[\![\mathsf{cAvg}(u, r_1, v, r_2)]\!]_x$, for a node x in the computation tree of $\mathcal{K}$ is equal to $[\![cAvg(u, p_1, v, p_2)]\!]_{\langle x, \mathcal{A}_1(x), \mathcal{A}_2(x)\rangle}$ in the computation tree of $\mathcal{K} \times \mathcal{A}_1 \times \mathcal{A}_2$. Accordingly, it is enough to show the decidability of controlled-accumulation assertions that use Boolean variables instead of regular expressions.

Now, a controlled-average assertion with Boolean variables p and q, instead of regular expressions, can be reduced to an assertion of the form $\mathsf{Sum}(v) \geq 0$, as follows. Consider an assertion $\mathsf{cAvg}(u, p, v, q) \geq c$. We define a new numeric variable v' with the following value (for all states s):

$$[\![v']\!]_s = \begin{cases} 0 & \text{if } [\![p]\!]_s = \mathtt{F} \text{ and } [\![q]\!]_s = \mathtt{F} \\ -cv & \text{if } [\![p]\!]_s = \mathtt{F} \text{ and } [\![q]\!]_s = \mathtt{T} \\ u & \text{if } [\![p]\!]_s = \mathtt{T} \text{ and } [\![q]\!]_s = \mathtt{F} \\ u - cv & \text{if } [\![p]\!]_s = \mathtt{T} \text{ and } [\![q]\!]_s = \mathtt{T} \end{cases}$$

PROPOSITION 3.10. *Consider a Kripke structure $\mathcal{K}$ with a numeric variable u, a positive numeric variable v and Boolean variables p and q. Let $\mathcal{K}'$ be a Kripke structure identical to $\mathcal{K}$, up to having a new numeric variable v', defined as above, for a constant number c. Then, for every node x of the computation tree of $\mathcal{K}'$, we have that* $\mathsf{cAvg}(u, p, v, q) \geq c$ *iff* $\mathsf{Sum}(v') \geq 0$.

PROOF. We have that:

$$\begin{array}{ll} [\![\mathsf{cAvg}(u, r_1, v, r_2)]\!]_x \geq c & \text{iff} \\ \frac{\sum_{(y \leq x \mid [\![p]\!]_y)} [\![u]\!]_y}{\sum_{(y \leq x \mid [\![q]\!]_y)} [\![v]\!]_y} \geq c & \text{iff} \\ \sum_{(y \leq x \mid [\![p]\!]_y)} [\![u]\!]_y \geq c(\sum_{(y \leq x \mid [\![q]\!]_y)} [\![v]\!]_y) & \text{iff} \\ \sum_{(y \leq x \mid [\![p]\!]_y)} [\![u]\!]_y - \sum_{(y \leq x \mid [\![q]\!]_y)} c[\![v]\!]_y \geq 0 & \text{iff} \\ \sum_{y \leq x} v' \geq 0 & \text{iff} \\ [\![\mathsf{Sum}(v')]\!]_x \geq 0. & \end{array}$$

□

3.4. Undecidability

We show that the model-checking problem for extended logics that have the temporal operators EG or EU (or their duals, AF or AR) is undecidable. This implies the unde-

cidability of the extension of all temporal logics that include or can be translated to these operators. In particular, the model-checking problems for the extensions of CTL* [Emerson and Halpern 1983], LTL [Pnueli 1977], RTL [Sistla and Zuck 1993], CTL [Emerson and Clarke 1982], STL [Alur and Henzinger 1999], UB [Ben-Ari et al. 1983], and EG [Ben-Ari et al. 1983] are all undecidable.

The proof is by a reduction from the halting problem of counter machines. Given a counter machine $\mathcal{M}$, we construct a Kripke structure $\mathcal{K}$ and a specification φ such that $\mathcal{K}$ satisfies φ iff $\mathcal{M}$ halts. The proof goes along similar lines to those used for proving the undecidability of model-checking Petri nets [Esparza 1996].

The intuitive explanation. A quantitative Kripke structure has the flavor of a counter machine, in the sense that the states correspond to the counter machine command-lines and the accumulated values to the counters. With two numeric variables, it is possible to mimic two counters. The crucial difference is that a counter machine has a conditional-jump command, in which it can check the counter values and branch accordingly. In contrast, the transitions of a Kripke structure are not guarded by the accumulated values.

Equipped with a suitable specification language, we can address this difference as follows. The Kripke structure uses its nondeterminism and has two transitions from each state associated with a conditional jump. These transitions can be taken regardless of the accumulated values. The specification, however, would limit attention to computations of the Kripke structure in which transitions are taken properly. As we show, this can be done using the G or U temporal operators. Below we describe the reduction in detail.

Counter machines. An n-counter machine is a sequence of uniquely-labeled commands, involving n counters. The counters are initialized to non-negative integers, or equivalently, all are initialized to zero and their desired initial value is set by the first machine commands. There are five command types, as demonstrated in Example 3.11.

Example 3.11. A machine with two counters, x and y. The machine adds the value of x to y and nullifies x.

l_1. `if x = 0 then goto l5 else goto l2`
l_2. `x := x − 1`
l_3. `y := y + 1`
l_4. `goto l1`
l_5. `halt`

We refer to commands of the form `if x = 0 then goto l5 else goto l2` as x-jumps. We assume that the machine never reaches a line of the form `x := x − 1` when the counter x is zero. Since we can add a guarding x-jump before reducing the value of x, the assumption does not lose generality.

The reduction. Given a two-counter machine M, we construct a Kripke structure $\mathcal{K}$ and a specification φ, such that $\mathcal{K}$ satisfies φ iff M does not halt. The values of the Kripke structure variables are from $\{0, 1, -1\}$ and the specification only uses the EG modality. The specification may either relate to the accumulative sum or to the accumulative average of $\mathcal{K}$'s variables. An illustration of the reduction is given in Fig. 4, with respect to the counter machine of Example 3.11.

For a two-counter machine $\mathcal{M}$ with n lines and the counters x and y, we define the Kripke structure $\mathcal{K} = \langle P, V, S, s_{in}, R, L\rangle$ as follows.

— $P = \{halt, x_z, x_p, y_z, y_p\}$. The latter variables are used for denoting whether a counter, for example x, should be zero (x_z), or positive (x_p), in a proper computation.
— $V = \{u, v\}$, corresponding to the x and y counters of $\mathcal{M}$, respectively.

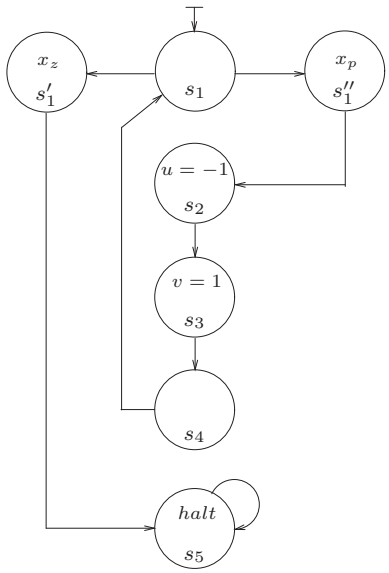


Fig. 4. The Kripke structure corresponding to the counter machine of Example 3.11.

— $S = \{s_i \mid l_i \in M\} \cup \{s'_i, s''_i \mid l_i \text{ is a conditional jump}\}$.
— $s_{in} = s_1$.
— $R = \{\langle s_i, s'_i\rangle, \langle s_i, s''_i\rangle, \langle s'_i, s_j\rangle, \langle s''_i, s_m\rangle \mid$
$l_i =$ `if x = 0 then goto l_j else goto l_m`$\}$
$\cup\ \{\langle s_i, s_{i+1}\rangle \mid l_i \in \{$`x := x + 1, x := x − 1,`
`y := y + 1, y := y − 1`$\}\}$
$\cup\ \{\langle s_i, s_j\rangle \mid l_i =$ `goto l_j`$\}$
$\cup\ \{\langle s_i, s_i\rangle \mid l_i =$ `halt`$\}$.
Thus, the transitions follow the control of $\mathcal{M}$, where each of the jumps in a conditional jump command l_i is partitioned into two transitions, visiting the intermediate states s'_i (in case the jump is according to the case $x = 0$) or s''_i (in case the jump is according to the case $x \neq 0$).
— L: All values are `F` or 0, except for the following values, defined for all $1 \leq i \leq n$:

$$\begin{array}{rcll}
[\![u]\!]_{s_i} & = & 1 & \text{if } l_i = \texttt{x := x + 1};\\
[\![u]\!]_{s_i} & = & -1 & \text{if } l_i = \texttt{x := x − 1};\\
[\![v]\!]_{s_i} & = & 1 & \text{if } l_i = \texttt{y := y + 1};\\
[\![v]\!]_{s_i} & = & -1 & \text{if } l_i = \texttt{y := y − 1};\\
[\![x_z]\!]_{s'_i} & = & \texttt{T} & \text{if } l_i \text{ is an } x \text{ jump};\\
[\![x_p]\!]_{s''_i} & = & \texttt{T} & \text{if } l_i \text{ is an } x \text{ jump};\\
[\![y_z]\!]_{s'_i} & = & \texttt{T} & \text{if } l_i \text{ is a } y \text{ jump};\\
[\![y_p]\!]_{s''_i} & = & \texttt{T} & \text{if } l_i \text{ is a } y \text{ jump};\\
[\![halt]\!]_{s_i} & = & \texttt{T} & \text{if } l_i = \texttt{halt}.
\end{array}$$

Consider the following formulas.

$$\begin{aligned}
\psi_{Proper} &= (x_z \rightarrow \mathsf{Sum}(u) = 0) \wedge (x_p \rightarrow \mathsf{Sum}(u) \neq 0) \wedge\\
&\quad\ (y_z \rightarrow \mathsf{Sum}(v) = 0) \wedge (y_p \rightarrow \mathsf{Sum}(v) \neq 0).\\
\varphi &= EG(\psi_{Proper} \wedge \neg halt).\\
\varphi' &= \psi_{Proper}\ EU\ halt.
\end{aligned}$$

Note that the specification can be equivalently defined using $\mathsf{Avg}()$ instead of $\mathsf{Sum}()$.

LEMMA 3.12. *Given a counter machine $\mathcal{M}$, let $\mathcal{K}$, φ, and φ' be as defined above. Then, $\mathcal{M}$ does not halt iff $\mathcal{K} \models \varphi$ iff $\mathcal{K} \not\models \varphi'$.*

PROOF. The counter machine $\mathcal{M}$ is deterministic, having a single run. A computation of $\mathcal{K}$ simply follows the run of $\mathcal{M}$, except for the conditional jumps, in which it has nondeterminism. It may either follow the run of $\mathcal{M}$ (that is, in states s_i of an x jump, branch to s'_i or s''_i according to the value of x) or violate it (that is, branch not according to the value of x). Note that all the computations of $\mathcal{K}$ violate the run of $\mathcal{M}$, except for exactly one computation r that follows it. Hence, all computations of $\mathcal{K}$, except for r, do not satisfy φ, while r satisfies φ iff $\mathcal{M}$ does not halt. Also, r satisfies φ' iff $\mathcal{M}$ halts. □

Since the operator G can be expressed by the operator W (Weak Until), and similarly for U and R (Release), Lemma 3.12 implies undecidability also for the EW and ER modalities. Using negation, we get undecidability also for the extension of logics with the AF, AR, AR, and AW modalities. It follows that the decidability result we have seen in Section 3.1 for a logic with the modalities EF and EX is maximal. We conclude that extending all the standard temporal logics with accumulative values, except for the EF logic, makes the model-checking problem undecidable.

COROLLARY 3.13. *The model-checking problem is undecidable for the temporal logics CTL*, LTL , RTL, CTL, STL, UB, and EG, extended by the atomic assertion* $\mathsf{Sum}(v) \geq c$.

3.5. Boolean Kripke Structures

Our setting considers a quantitative Kripke structure and a specification over its accumulated values. One may consider a possibly simpler setting, where the Kripke structure is Boolean and the specification refers to the average of truth values. For an atomic proposition p, let $\mathsf{Avg}(p)$ denote the average of truth values of p up to the current point in time. We can then have specifications with new atomic assertions, like $\mathsf{Avg}(p) \geq \frac{1}{2}$.

Is there a temporal logic whose extension with such atomic assertions is decidable, while its extension with respect to quantitative Kripke structures is undecidable? No. The undecidability proofs of Section 3.4 also apply to the case of Boolean Kripke structures, by a small adaptation: Instead of having numerical variables u and v with values $\{1, 0, -1\}$, we can have atomic propositions p and q, and represent the numeric values by $-1 = \mathtt{FF}$, $0 = \mathtt{TF}$, and $1 = \mathtt{TT}$. For example, whenever there is a state in the quantitative Kripke structure with $u = 0$ and $v = 1$, we produce two consecutive states in the Boolean Kripke structure, one with $p = \mathtt{T}$ and $q = \mathtt{T}$ and the other with $p = \mathtt{F}$ and $q = \mathtt{T}$, corresponding to $\mathtt{TF}$ for $u = 0$ and $\mathtt{TT}$ for $v = 1$. (The Boolean Kripke structure that corresponds to the machine in Example 3.11 is shown in Fig. 5. Note that the original states s_1, s'_1 and s''_1 were not doubled, unlike the other original states, due to a local optimization.) The formula ψ_{Proper} is adjusted accordingly as follows.

$$\begin{aligned}\psi_{Proper} \;=\; & (x_z \rightarrow \mathsf{Avg}(p) = \frac{1}{2}) \wedge (x_p \rightarrow \mathsf{Avg}(p) \neq \frac{1}{2}) \wedge \\ & (y_z \rightarrow \mathsf{Avg}(q) = \frac{1}{2}) \wedge (y_p \rightarrow \mathsf{Avg}(q) \neq \frac{1}{2}).\end{aligned}$$

4. LTL WITH PATH ACCUMULATIONS

In this section, we show the decidability of model checking a quantitative Kripke structure and a specification given by an LTL$^{lim\Sigma}$ formula (an LTL formula extended by path-accumulation assertions, as defined in Section 2). An example of such an extended

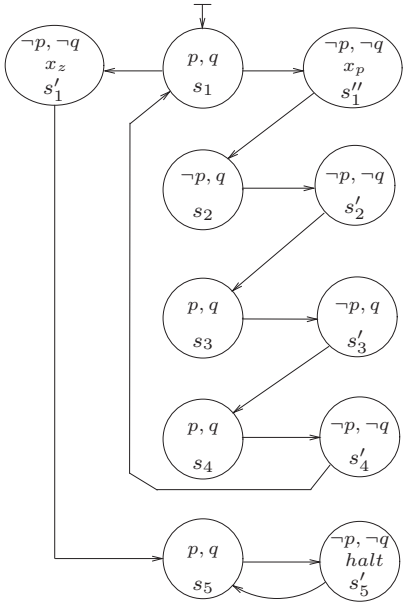


Fig. 5. The Boolean Kripke structure corresponding to the counter machine of Example 3.11.

formula is:

$$FG(q) \rightarrow ((\mathsf{LimSupAvg}(u) = 5) \vee Gp \wedge (\mathsf{LimInfAvg}(v) > 4)).$$

Given an LTL$^{lim\Sigma}$ formula ψ, we shall consider its negation $\varphi = \neg\psi$, and check whether the given Kripke structure $\mathcal{K}$ has a computation that satisfies φ. We do it as follows:

— Translating φ to $\varphi' = \varphi_1 \vee \varphi_2 \vee \ldots \vee \varphi_n$, such that each φ_i is of the form $\chi \wedge \xi$, where χ is a Boolean combination of limit-average assertions and ξ is a standard LTL formula.
— For each disjunct $\chi \wedge \xi$, checking whether $\mathcal{K}$ has a computation that satisfies $\xi \wedge \chi$. We do this by translating ξ to a nondeterministic Büchi automaton (NBW) $\mathcal{A}$ [Vardi and Wolper 1986] and checking whether the product $\mathcal{K} \times \mathcal{A}$, which is a quantitative Kripke structure with a fairness condition, has a fair computation that satisfies the limit-average formula χ.

Below we describe the model-checking procedure in detail and prove its correctness.

Detaching limit-average assertions. Consider an LTL$^{lim\Sigma}$ formula φ with n limit-average assertions, $\theta_1, \ldots, \theta_n$. For $b_i \in \{\mathtt{T}, \mathtt{F}\}$, we use $\varphi(b_1, \ldots, b_n)$ to denote the LTL formula obtained form φ by replacing all occurrences of the assertion θ_i by the truth value b_i. Recall that path-accumulation assertions are interpreted with respect to entire paths and their value is the same in all the suffixes of a path. Therefore, for an LTL$^{lim\Sigma}$ formula φ with n limit-average assertions, $\theta_1, \ldots, \theta_n$, the LTL$^{lim\Sigma}$ formula φ' defined below is equivalent to φ.

$$\begin{array}{rcl} \varphi' = & [\quad \theta_1 \wedge \theta_2 \wedge \ldots \wedge \theta_n \quad \wedge \quad \varphi(\mathtt{T},\mathtt{T},\ldots,\mathtt{T}) \quad] \\ \vee & [\quad \neg\theta_1 \wedge \theta_2 \wedge \ldots \wedge \theta_n \quad \wedge \quad \varphi(\mathtt{F},\mathtt{T},\ldots,\mathtt{T}) \quad] \\ \vee & \qquad\qquad \vdots \\ \vee & [\quad \neg\theta_1 \wedge \neg\theta_2 \wedge \ldots \wedge \neg\theta_n \quad \wedge \quad \varphi(\mathtt{F},\mathtt{F},\ldots,\mathtt{F}) \quad] \end{array}$$

Note that in the formula φ', each disjunct is a conjunction of a standard LTL formula and a Boolean combination of limit-average assertions. We refer to the latter as a *limit-average formula*.

Now, since we check for the existence of a computation that satisfies φ, each disjunct of φ' can be checked separately. Therefore, we should only solve the problem of deciding whether there is a computation satisfying $\chi \wedge \xi$ for a limit-average formula χ and a standard LTL formula ξ. Before describing the solution, we recall the relevant theory of automata on infinite words.

A nondeterministic Büchi automaton (NBW, for short) is $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha \rangle$, where Σ is the input alphabet, Q is a finite set of states, $q_{in} \in Q$ is an initial states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, and $\alpha \subseteq Q$ is a set of accepting states. A run $r = r_0, r_1, \cdots$ of $\mathcal{A}$ on a word $w = w_1 \cdot w_2 \cdots \in \Sigma^\omega$ is an infinite sequence of states such that $r_0 = q_{in}$, and for every $i \geq 0$, we have that $r_{i+1} \in \delta(r_i, w_{i+1})$. The run r is *accepting* iff $\inf(r) \cap \alpha \neq \emptyset$. An automaton accepts a word if it has an accepting run on it. The language of an automaton $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. Given an LTL formula ξ over a set P of atomic propositions, it is possible to translate ξ to an NBW $\mathcal{A}_\xi$ over the alphabet 2^P. For every word $w \in (2^P)^\omega$, the NBW $\mathcal{A}_\xi$ has an accepting run on w iff a computation that is labeled w satisfies φ [Vardi and Wolper 1986].

Consider a Kripke structure $\mathcal{K} = \langle P, V, S, s_{in}, R, L \rangle$ and the NBW $\mathcal{A}_\xi = \langle 2^P, Q, q_{in}, \delta, \alpha \rangle$. We define their product $\mathcal{B} = \mathcal{K} \times \mathcal{A}_\xi$ as the fair Kripke structure $\mathcal{B} = \langle \emptyset, V, S \times Q, \langle s_{in}, q_{in} \rangle, R', L', S \times \alpha \rangle$, where $R'(\langle s, q \rangle, \langle s', q' \rangle)$ iff $R(s, s')$ and $q' \in \delta(q, [\![P]\!]_s)$, and L' is such that for every $v \in V, s \in S$, and $q \in Q$, we have $[\![v]\!]_{\langle s,q \rangle} = [\![v]\!]_s$. Note that $\mathcal{A}$'s transitions correspond to the labeling of $\mathcal{K}$'s Boolean variables, ensuring that a word over which $\mathcal{A}$ runs is a labeling of $\mathcal{K}$'s computation.

Checking for a fair computation with limit-average properties. Given a limit-average formula χ and quantitative Kripke structure $\mathcal{K}$ with a Büchi fairness condition, we check whether $\mathcal{K}$ has a fair computation that satisfies χ. The problem for Kripke structures without fairness was solved in [Alur et al. 2009]. (The paradigm in [Alur et al. 2009] is different from ours, as the limit-average formula there constitutes the acceptance conditions for the automata.) For extending the technique there to Kripke structures with fairness, we first need the following lemma. It intuitively shows that inserting infinitely, but negligibly, many constant values to a computation does not change its limit-average values.

LEMMA 4.1. *Consider an infinite computation $\pi = x_1, x_2, \ldots$ and a finite computation $\mu = y_1, y_2, \ldots, y_k$ with a numeric variable v absolutely bounded by a constant c (that is, $|x_i| \leq c$ and $|y_j| \leq c$ for all $i \geq 1$ and $1 \leq j \leq k$). Let π' be the infinite computation obtained from π by inserting μ at positions $\{2^i \mid i \in \mathbb{N}\}$. Then, $[\![\mathsf{LimInfAvg}(v)]\!]_\pi = [\![\mathsf{LimInfAvg}(v)]\!]_{\pi'}$ and $[\![\mathsf{LimSupAvg}(v)]\!]_\pi = [\![\mathsf{LimSupAvg}(v)]\!]_{\pi'}$.*

PROOF. Let $\pi' = z_1, z_2, z_3, \ldots$. For showing that π and π' have the same limit-average values, we define a surjective mapping ρ between the positions of π' and π,

such that $\rho(j_1) \geq \rho(j_2)$ iff $j_1 \geq j_2$, and show that $\lim_{j\to\infty} |\frac{\sum_0^j [\![v]\!]_{z_j}}{j} - \frac{\sum_0^{\rho(j)} [\![v]\!]_{x_{\rho(j)}}}{\rho(j)}|$ converges to 0.

We denote the range of a function f by $\mathsf{range}(f)$ and define the functions $\mathsf{Move} : \mathbb{N} \to \mathbb{N}$, $\mathsf{Next} : \mathbb{N} \to \mathbb{N}$, and $\rho : \mathbb{N} \to \mathbb{N}$ as follows.

$$\begin{aligned} \mathsf{Move}(j) &= j + k \cdot |\{2^i \mid i \in \mathbb{N} \text{ and } 2^i \leq j\}|; \\ \mathsf{Next}(j) &= \min\{i \mid i \in \mathsf{range}(\mathsf{Move}) \text{ and } i \geq j\}; \text{ and} \\ \rho(j) &= \mathsf{Move}^{-1}(\mathsf{Next}(j)). \end{aligned}$$

Intuitively, every position of π' that originated in π is mapped by ρ to its original position in π, while a position of π' that originated in μ is treated as the next position of π' that originated in π.

Now, for every $j \in \mathbb{N}$, we have that $|\frac{\sum_0^j [\![v]\!]_{z_j}}{j} - \frac{\sum_0^{\rho(j)} [\![v]\!]_{x_{\rho(j)}}}{\rho(j)}| \leq |\frac{c(j-\rho(j))}{\rho(j)}|$. Hence, $\lim_{j\to\infty} |\frac{\sum_0^j [\![v]\!]_{z_j}}{j} - \frac{\sum_0^{\rho(j)} [\![v]\!]_{x_{\rho(j)}}}{\rho(j)}| \leq \lim_{j\to\infty} |\frac{c(j-\rho(j))}{\rho(j)}| = 0$, as required. □

We can now show how to adjust the emptiness algorithm of [Alur et al. 2009] for handling the Büchi fairness condition.

LEMMA 4.2. *Consider a quantitative Kripke structure $\mathcal{B}$ with a Büchi fairness condition α. There is an algorithm to check whether $\mathcal{B}$ has a fair computation that satisfies a limit-average formula χ.*

PROOF. In [Alur et al. 2009], the authors describe an algorithm to check whether a Kripke structure $\mathcal{K}$ (without fairness) has a computation that satisfies a limit-average formula χ. The algorithm is based on a procedure $\mathsf{ComponentCheck}(M, \chi)$, which is called in over every reachable maximally strongly connected component M of $\mathcal{K}$. It is shown that $\mathsf{ComponentCheck}(M, \chi) = \mathtt{T}$ iff there is a computation of M that satisfies χ. Since $[\![\mathsf{LimInfAvg}(v)]\!]_\pi$ and $[\![\mathsf{LimSupAvg}(v)]\!]_\pi$, are indifferent to any finite prefix of π, it follows that $\mathcal{K}$ has a computation that satisfies χ iff some component M of $\mathcal{K}$ has such a computation [Alur et al. 2009].

We claim that $\mathcal{B}$ has a fair computation satisfying χ iff $\mathcal{B}$ has a maximally strongly component M such that $M \cap \alpha \neq \emptyset$ and $\mathsf{ComponentCheck}(M, \chi) = \mathtt{T}$.

Obviously, if $\mathcal{B}$ has no such component, then no computation of $\mathcal{B}$ can satisfy both α and χ. As for the other direction, assume that there is a component M with a state $s \in M \cap \alpha$, such that $\mathsf{ComponentCheck}(M, \chi) = \mathtt{T}$. Let π be a computation of $\mathcal{B}$, such that $\inf(\pi) \subseteq M$ and π satisfies χ. If $s \in \inf(r)$ then we are done. Otherwise, let s' be a state in $\inf(r)$, and let μ be a finite cycle in M that visits both s and s'.

Consider the computation π' of $\mathcal{B}$ that is derived from π by inserting μ at the positions $\{2^i \mid i \in \mathbb{N}\}$. We have that π satisfies the Büchi condition α, as it visits $s \in \alpha$ infinitely often. In addition, by Lemma 4.1, the limit-average values of π' are the same as those of π, thus π' also satisfies the limit-average formula χ, and we are done. □

We can thus conclude:

THEOREM 4.3. *The model-checking problem for LTL$^{lim\Sigma}$ is decidable.*

Note that model checking an LTL$^{lim\Sigma}$ formula is also decidable with respect to a quantitative Kripke structure with a fairness condition. The reason is that the algorithm already handles a Büchi condition, derived from the LTL formula, which can be combined with the fairness condition of the Kripke structure. Also, since the model-checking procedure anyway translates the temporal-logic component to an NBW, we

can easily extend it to handle LTL$^{lim\Sigma}$ with a regular layer – one in which the path formulas may also contain regular expressions.

Complexity. The algorithm in the proof of Lemma 4.2 requires time and space that are exponential in the Kripke structure and doubly exponential in the LTL formula. Yet, replacing this algorithm with techniques presented in [Kosaraju and Sullivan 1988; Velner and Rabinovich 2011; Velner 2011] allows for improved complexities, as described below.

Consider a Kripke structure $\mathcal{K}$ with n states and an LTL$^{lim\Sigma}$ formula of length m. Recall that we translate the LTL$^{lim\Sigma}$ formula to a disjunction of up to 2^m subformulas, each of which is a conjunction of a standard LTL formula ξ and a limit-average formula χ. Moreover, χ is a conjunction of up to m limit-average assertions. Our model-checking procedure iterates over the disjuncts, checking each of them separately. This is done by translating ξ to an NBW $\mathcal{A}$ with up to 2^m states, constructing the product of $\mathcal{A}$ and $\mathcal{K}$, which is a quantitative Kripke structure $\mathcal{K}'$ with a Büchi fairness condition, and checking whether $\mathcal{K}'$ has a fair computation that satisfies χ. By Lemma 4.1, the problem reduces to checking whether $\mathcal{K}'$ has some computation (not necessarily fair) satisfying χ. As with model checking of standard LTL formulas, the NBW $\mathcal{A}$ and the product Kripke structure $\mathcal{K}'$ can be constructed on-the-fly, using space that is polynomial in the LTL formula.

In [Velner 2011], Velner studied the problem of deciding whether a Kripke structure $\mathcal{K}'$ has a computation that satisfies a conjunction of limit-average assertions, all of which are of the form $\mathsf{LimSupAvg}(v) \geq 0$ or $\mathsf{LimInfAvg}(v) \geq 0$, for quantitative variables v. It is shown in [Velner 2011] that the problem can be solved deterministically in time that is polynomial in the number of states in $\mathcal{K}'$, as well as non-deterministically in space that is logarithmic in $\mathcal{K}'$ and polynomial in the assertions. Our limit-average formula χ is richer in three aspects: i) it may have strict inequalities; ii) it allows for $\leq$; and iii) it may have arbitrary constants, not only 0. The technique in [Velner 2011] can be extended to strict inequalities, as done in [Velner and Rabinovich 2011], which resolves aspect (i). For handling aspect (ii), one can add new quantitative variables with corresponding weights. For example, $\mathsf{LimSupAvg}(v) \geq 5$ is changed to $\mathsf{LimSupAvg}(v') \geq 0$, where v' is a new variable, taking the value $(v - 5)$ in all edges. Finally, aspect (iii) can be solved by adding new variables that equal to the original ones, but with an opposite sign, and replacing between LimSupAvg and LimInfAvg. For example, $\mathsf{LimSupAvg}(v) \leq 0$ is changed to $\mathsf{LimInfAvg}(v') \geq 0$, where v' is a new variable, taking the value $(-v)$ in all edges.

We can thus conclude with the following. Since LTL model checking is PSPACE-complete, the complexities are tight.

PROPOSITION 4.4. *Model-checking a Kripke structure with n states and an LTL$^{lim\Sigma}$ formula of length m can be solved deterministically in time complexity that is polynomial in n and exponential in m, as well as non-deterministically in space complexity that is logarithmic in n and polynomial in m.*

Acknowledgments

We thank Dejan Nickovic for stimulating discussions on controlled accumulation and Yaron Velner for discussions that have led to improving the complexity of LTL$^{lim\Sigma}$ model checking.

REFERENCES

ALUR, R., DEGORRE, A., MALER, O., AND WEISS, G. 2009. On omega-languages defined by mean-payoff conditions. In *FOSSACS*, L. de Alfaro, Ed. LNCS Series, vol. 5504. Springer, 333–347.

ALUR, R. AND HENZINGER, T. 1994. A really temporal logic. *Journal of the ACM 41,* 1, 181–204.

ALUR, R. AND HENZINGER, T. A. 1999. Computer-aided verification: An introduction to model building and model checking for concurrent systems. *Book in preparation*.

BEN-ARI, M., PNUELI, A., AND MANNA, Z. 1983. The temporal logic of branching time. *Acta Inf. 20*, 207–226.

BJORKLUND, H., SANDBERG, S., AND VOROBYOV, S. 2004. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. In *MFCS'04*. 673–685.

BLOEM, R., CHATTERJEE, K., HENZINGER, T. A., AND JOBSTMANN, B. 2009. Better quality in synthesis through quantitative objectives. In *CAV*, A. Bouajjani and O. Maler, Eds. LNCS Series, vol. 5643. Springer, 140–156.

BOKER, U., CHATTERJEE, K., HENZINGER, T. A., AND KUPFERMAN, O. 2011. Temporal specifications with accumulative values. In *Proc. 26th IEEE Symp. on Logic in Computer Science*. 43–52.

CHAKRABARTI, A., DE ALFARO, L., HENZINGER, T. A., AND STOELINGA, M. 2003. Resource interfaces. In *Proc. of EMSOFT: Embedded Software*. LNCS 2855. Springer, 117–133.

CHATTERJEE, K., DOYEN, L., AND HENZINGER, T. A. 2008. Quantitative languages. In *Proc. of CSL*. LNCS 5213. Springer, 385–400.

CHATTERJEE, K., DOYEN, L., HENZINGER, T. A., AND RASKIN, J.-F. 2010. Generalized mean-payoff and energy games. In *FSTTCS*. LIPIcs Series, vol. 8. 505–516.

COURCOUBETIS, C., ALUR, R., AND DILL, D. 1991. Model-checking for probabilistic real-time system. In *Proc. 18th Int. Colloq. on Automata, Languages, and Programming*. LNCS. Springer.

CULIK II, K. AND KARHUMÄKI, J. 1994. Finite automata computing real functions. *SIAM J. Comput. 23,* 4, 789–814.

DE ALFARO, L., FAELLA, M., HENZINGER, T., MAJUMDAR, R., AND STOELINGA, M. 2005. Model checking discounted temporal properties. *Theoretical Computer Science 345,* 1, 139–170.

DROSTE, M. AND GASTIN, P. 2007. Weighted automata and weighted logics. *Theoretical Computer Science 380*, 69–86.

DROSTE, M., KUICH, W., AND RAHONIS, G. 2008. Multi-valued MSO logics over words and trees. *Fundamenta Informaticae 84*, 305–327.

DROSTE, M., KUICH, W., AND VOGLER, H. 2009. Monograph in theoretical computer science: Eatcs series. In *Handbook of Weighted Automata*. Springer.

DROSTE, M. AND KUSKE, D. 2003. Skew and infinitary formal power series. In *Proc. of ICALP*. LNCS 2719. Springer, 426–438.

DROSTE, M. AND MEINECKE, I. 2010. Describing average- and longtime-behavior by weighted MSO logics. In *MFCS*. 537–548.

EMERSON, E. A. AND CLARKE, E. M. 1982. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program. 2,* 3, 241–266.

EMERSON, E. A. AND HALPERN, J. Y. 1983. "sometimes" and "not never" revisited: On branching versus linear time. In *POPL*. 127–140.

ÉSIK, Z. AND KUICH, W. 2004. An algebraic generalization of omega-regular languages. In *Proc. of MFCS*. LNCS 3153. 648–659.

ESPARZA, J. 1996. Decidability and complexity of Petri net problems - an introduction. In *Petri Nets*, W. Reisig and G. Rozenberg, Eds. LNCS Series, vol. 1491. Springer, 374–428.

FISCHER, M. AND RABIN, M. O. 1974. Super-exponential complexity of presburger arithmetic. In *Proceedings of the SIAM-AMS Symposium in Applied Mathematics*. Vol. 7. 27–41.

GURFINKEL, A. AND CHECHIK, M. 2003. Multi-valued model checking via classical model checking. In *Proc. of CONCUR*. LNCS 2761. Springer, 263–277.

KOSARAJU, S. R. AND SULLIVAN, G. F. 1988. Detecting cycles in dynamic graphs in polynomial time (preliminary version). In *STOC*. ACM, 398–406.

KUICH, W. AND SALOMAA, A. 1986. *Semirings, Automata, Languages*. Monographs in Theoretical Computer Science. Series, vol. 5. Springer.

KUPFERMAN, O. AND LUSTIG, Y. 2007. Lattice automata. In *Proc. of VMCAI*. LNCS 4349. Springer, 199–213.

KUPFERMAN, O. AND VARDI, M. 2006. Memoryful branching-time logics. In *Proc. 21st IEEE Symp. on Logic in Computer Science*. 265–274.

MANNA, Z. AND PNUELI, A. 1979. The modal logic of programs. In *ICALP*, H. A. Maurer, Ed. LNCS Series, vol. 71. Springer, 385–409.

OPPEN, D. C. 1978. A $2^{2^{2^{pn}}}$ upper bound on the complexity of presburger arithmetic. *J. Comput. Syst. Sci. 16,* 3, 323–332.

PNUELI, A. 1977. The temporal logic of programs. In *FOCS*. IEEE, 46–57.

PRESBURGER, M. 1929. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *Comptes Rendus du I Congrès de Mathématiciens des Pays Slaves*, 92–101.

REDDY, C. R. AND LOVELAND, D. W. 1978. Presburger arithmetic with bounded quantifier alternation. In *Proceedings of the tenth annual ACM symposium on Theory of computing*. STOC '78. ACM, 320–325.

SCHÜTZENBERGER, M. P. 1961. On the definition of a family of automata. *Information and Control 4*, 245–270.

SISTLA, A. P. AND ZUCK, L. D. 1993. Reasoning in a restricted temporal logic. *Inf. Comput. 102,* 2, 167–195.

VARDI, M. Y. AND WOLPER, P. 1986. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*. IEEE Computer Society, 332–344.

VELNER, Y. 2011. The complexity of mean-payoff automaton expressions. *CoRR*.

VELNER, Y. AND RABINOVICH, A. 2011. Church synthesis problem for noisy input. In *FOSSACS*. Lecture Notes in Computer Science Series, vol. 6604. Springer, 275–289.

ZWICK, U. AND PATERSON, M. 1996. The complexity of mean payoff games on graphs. *Theoretical Computer Science 158*, 343–359.

Received Month Year; revised Month Year; accepted Month Year