Temporal Specifications with Accumulative Values

Udi Boker*†, Krishnendu Chatterjee†, Thomas A. Henzinger†, and Orna Kupferman*
*Hebrew University, Israel
†IST Austria

***Abstract*—There is recently a significant effort to add quantitative objectives to formal verification and synthesis. We introduce and investigate the extension of temporal logics with quantitative atomic assertions, aiming for a general and flexible framework for quantitative-oriented specifications.**

In the heart of quantitative objectives lies the accumulation of values along a computation. It is either the accumulated summation, as with the energy objectives, or the accumulated average, as with the 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 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 of 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 computation.

We study the border of decidability for 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 by prefix-accumulation assertions and extending LTL with path-accumulation assertions, result in temporal logics whose model-checking problem is decidable. The extended logics allow to significantly extend the currently known energy and mean-payoff objectives. Moreover, the prefix-accumulation assertions may be refined 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 the fragment we point to is, in a sense, the maximal logic whose extension with prefix-accumulation assertions permits a decidable model-checking procedure. Extending a temporal logic that has the EG or EU modalities, and in particular CTL and LTL, makes the problem undecidable.

I. 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.

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 specification 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 specification. The approach that has been mostly followed is to consider specific objectives, as mean-payoff or energy-level, by means of weighted automata [3], [4], [5], [6]. 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, which is the core issue, 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 of 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 activeness-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 no longer 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 $\mathsf{Sum}(v)$, $\mathsf{Avg}(v)$,

[1] Different classes of formalisms with quantitative aspects are *real-time* logic and automata [1], as well as logics that support *probabilistic* reasoning [2]. 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 III-B.)

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 of time. For example, basic atomic assertions are $\mathsf{Sum}(v) \geq c$ and $\mathsf{Avg}(v) \geq c$, and one can also have expressions like $\mathsf{Sum}(v) \geq 2\mathsf{Sum}(u) + 5$. 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^-) [7], which is the fragment of CTL with the EF, AG, EX, and AX temporal operators, can indeed be extended, with a rich class of arithmetic expressions (we would formally define it below). We denote the extended logic by EF^Σ. A simple example of an EF^Σ specification is given below.

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 output 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: $\mathsf{AG}(\mathsf{Avg}(p) \geq 0.95 \wedge \mathsf{Sum}(v) \geq 0)$.

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 $\mathsf{Sum}(u) - \mathsf{Sum}(v) > 3 \wedge \mathsf{Avg}(u) \geq 2\mathsf{Avg}(v)$, 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 extension is of special interest, as it allows to accumulate the time-ticks of definable transactions. For example, one may specify constraints on the average waiting time between a request and a grant.

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 [8]. 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. 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.

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* [9], or refer to the limit-average of the accumulated values. We do not know of a simple way to express the limit-average by prefix-accumulation, and, at any rate, extending LTL with prefix accumulation results in a logic whose model-checking problem is undecidable. Other known extensions of LTL also cannot capture limit-average (mean-payoff) objectives. We therefore study also 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 long-run average of (the infimum/suprimum of) v along an entire computation.

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 $\mathrm{LTL}^{\mathrm{lim}}$, 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 $\mathrm{LTL}^{\mathrm{lim}}$ significantly enriches the currently known mean-payoff objectives. An example for a specification in $\mathrm{LTL}^{\mathrm{lim}}$ is given below.

Long run happiness. Consider a system with Boolean variables Wish and $\mathit{ComesTrue}$, and numeric variables Income and $\mathit{Pleasure}$. 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$^{\lim}$ formula: $G(\mathit{Wish} \rightarrow F(\mathit{ComesTrue})) \vee \mathsf{LimInfAvg}(\mathit{Income}) > 0 \wedge \mathsf{LimInfAvg}(\mathit{Pleasure}) > 0$.

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 [10], [11] and ω-series for infinite words [12], [13], [14]. In [4], new classes of cost functions were studied using operations over rational numbers that do not form a semiring. In [5], 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 [15], over trees [16], or using finite lattices [17], [18]. The work of [19] 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 literature [20], [21], [22], [23]. 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.

II. 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 standard, 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 $L : S \rightarrow 2^P \times \mathbb{Q}^V$.

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 of states that the 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. For a Büchi fairness condition, we have that $\alpha \subseteq S$, 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 $\mathsf{Sum}(v) \geq 2\mathsf{Sum}(u)+5$. 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 [7]. We shall denote its extension with prefix-accumulation assertions by EF$^{\Sigma}$.

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

- An LTL$^{\lim}$ formula is $p \in P$, a path-accumulation assertion over V, $\neg\varphi$, $\varphi_1 \wedge \varphi_2$, $X\varphi$, $F\varphi$, $G\varphi$ and $\varphi_1 U \varphi_2$, for 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. 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", and U for "until". Other standard temporal operators that will be mentioned in the sequel are R for "release" and W for "weak until". For the accumulation assertions, the semantics is defined below. 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 $y \cdot d \in \pi$. 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 $\mathcal{K}$. Formally (see an example in Figure 1), 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 $\tau_{\mathcal{K}}(x) = L(\mathsf{state}(x))$.

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.

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 Figure 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 $\mathsf{LimInfAvg}$. Accordingly, the nesting of path-accumulation assertions in temporal operators does not add to the expressive power of LTL$^{\text{lim}}$. We still allow this nesting, as it enables more succinct formulas.

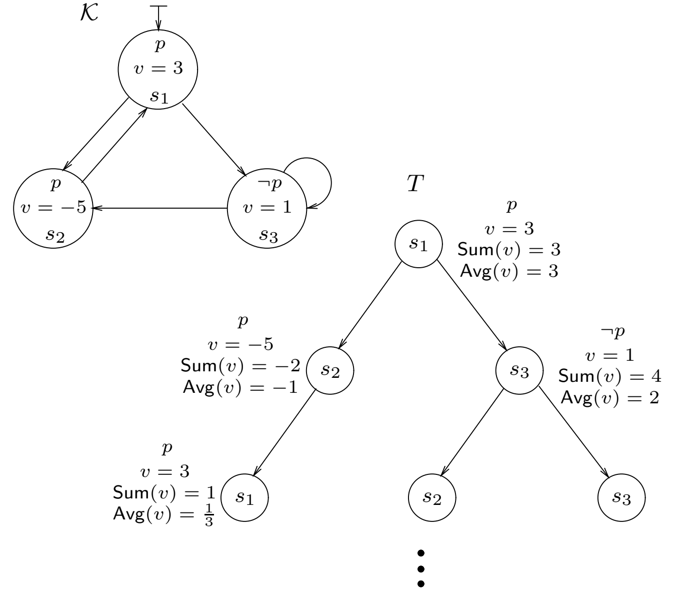

Fig. 1. A quantitative Kripke structure $\mathcal{K}$ and its computation-tree T.

III. 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 Petri-nets is undecidable for all relevant temporal logics [24], we show that it is decidable for a quantitative Kripke structure and a specification in the logic EF$^\Sigma$. It also turns out that, in a sense, the logic EF is the maximal one that can be extended with prefix-accumulation.

In Section III-A, we show the decidability of the model-checking problem for the logic EF$^\Sigma$. In Section III-B, we further extend EF$^\Sigma$ with assertions on controlled accumulation, while keeping 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 III-C 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 1. *Consider a Kripke structure $\mathcal{K}$ and a specification φ in a temporal logic with prefix-accumulation assertions. It is*

[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.

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 $\mathcal{K}' \models \varphi'$.

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. ■

A. 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 [25].

A Presburger arithmetic (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 [26], 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 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, 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 Figure 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, $\mathsf{Shift}(\varphi) = X\varphi$ and with a branching-time

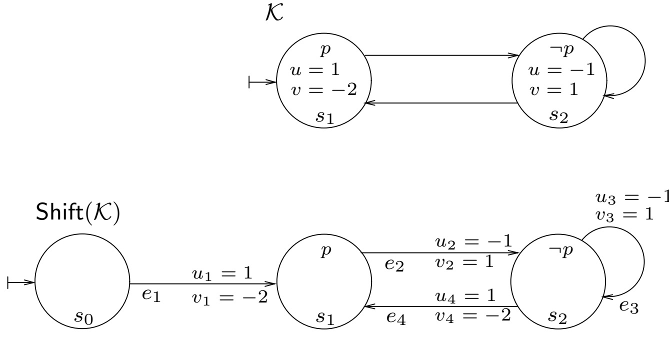

Fig. 2. The Kripke structure $\mathcal{K}$ and its equivalent structure $\mathsf{Shift}(\mathcal{K})$, having the numeric values on the edges.

specification $\mathsf{Shift}(\varphi)$ may be $AX\varphi$ or $EX\varphi$ (since s_0 has a single successor, path quantification is not important).

Proposition 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: 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 repeated 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. *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 Figure 2 has the following segments:

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

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 4 (PA-formulation of a path). *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 $s \in S$: $\sum_{i\in\mathsf{In}(s)} x_i = \sum_{j\in\mathsf{Out}(s)} x_j$.*
- *If $a \neq b$ (i.e. not a cycle) then:*
  - *For every $s \in S \setminus \{a, b\}$: $\sum_{i\in\mathsf{In}(s)} x_i = \sum_{j\in\mathsf{Out}(s)} x_j$.*
  - *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 Figure 2 is $\psi_\kappa =$

$x_1 \geq 1 \ \wedge\ x_2 \geq 1 \ \wedge\ x_3 \geq 1 \ \wedge\ x_4 = 0$ (edges)
$\wedge\ x_1 + x_4 = x_2$ (internal states)
$\wedge\ x_1 = 0 + 1 \ \wedge\ x_3 + x_4 = x_2 + x_3 - 1$ (start and end)

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 5. *Consider a segment $\kappa = \langle a, b, C\rangle$ of a Kripke structure $\mathcal{K}$ with states S and edges $E = \{e_1, e_2, \ldots, e_n\}$. Then, 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 solution $x_1, x_2, \ldots, x_n$ of the formula 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 $p' = a \rightarrow s'_1 \rightarrow s'_2 \rightarrow \ldots s'$, 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$^{\Sigma}$ 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 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.

[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.

Theorem 6. *Given a quantitative Kripke structure $\mathcal{K}$ and a specification φ in EF$^{\Sigma}$, 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 2, the latter holds iff $\mathcal{K} \models \varphi$. The proof proceeds by an induction on the nesting level of $\mathsf{Shift}(\varphi)$.

For a single temporal operator and a single segment, the translation correctness follows from Lemma 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, setting the starting state of the inner segment to be the ending state of the upper level ensures a correct path, while 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$^{\Sigma}$ formula is also decidable with respect to a quantitative Kripke structure with a fairness condition. The reason is that a fairness condition only relates to computation suffixes, while an EF formula only relates to computation prefixes. The single intersection-point between the two is the liveness-property of the prefix states. 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$^{\Sigma}$ 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 $EF\xi$.

Complexity: The complexity of the construction is roughly quad-exponential in the size of the Kripke structure. The best known algorithm for solving a PA formula is triple-exponential, while our PA formula might be exponential in the size of the Kripke structure. The length of the PA formula is $O(2^{n \times d} + m)$ for a Kripke structure with n states and an EF$^{\Sigma}$ formula of length m and nesting-level d of EF operators.

A lower bound for the required complexity is an open problem. Specifically, one may seek an algorithm that uses a weak version of Presburger arithmetic, as integer or linear programming, and try to avoid the brute-force segmentation of the Kripke structure.

B. 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).

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

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

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

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.

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. 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. Now, 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):

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

Proposition 7. *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:

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

■

C. 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 undecidability 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* [27], LTL [28], RTL [29], CTL [30], STL [31], UB [32], and EG [32] 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 [24].

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 (Always) or U (Until) 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 8.

Example 8. *A machine with two counters, x and y. The machine adds the value of x to y and nullifies x.*

l_1. $\texttt{if x} = 0 \texttt{ then goto } l_5 \texttt{ else goto } l_2$
l_2. $\texttt{x} := \texttt{x} - 1$
l_3. $\texttt{y} := \texttt{y} + 1$
l_4. $\texttt{goto } l_1$
l_5. $\texttt{halt}$

We refer to commands of the form $\texttt{if x} = 0 \texttt{ then goto } l_5 \texttt{ else goto } l_2$ as x-jumps. We assume that the machine never reaches a line of the form $\texttt{x} := \texttt{x} - 1$ 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 Figure 3, with respect to the counter machine of Example 8.

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.
- $S = \{s_i \mid l_i \in M\} \cup \{s'_i, s''_i \mid l_i \text{ is a conditional jump}\}$.
- $s_{in} = s_1$.
- $R = \{\langle s_i, s'_i\rangle, \langle s_i, s''_i\rangle, \langle s'_i, s_j\rangle, \langle s''_i, s_m\rangle \mid l_i = \texttt{ if x} = 0 \texttt{ then goto } l_j \texttt{ else goto } l_m\}$
  $\cup \ \{\langle s_i, s_{i+1}\rangle \mid l_i \in \{\texttt{x} := \texttt{x}+1, \texttt{x} := \texttt{x}-1, \texttt{y} := \texttt{y}+1, \texttt{y} := \texttt{y}-1\}\}$
  $\cup \ \{\langle s_i, s_j\rangle \mid l_i = \texttt{goto } l_j\}$
  $\cup \ \{\langle s_i, s_i\rangle \mid l_i = \texttt{halt}\}$.
  Thus, the transitions follow the control of $\mathcal{M}$, where each of the jumps in a conditional jump command l_i is divided 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 $\texttt{F}$ or 0, except for every $1 \leq i \leq n$:

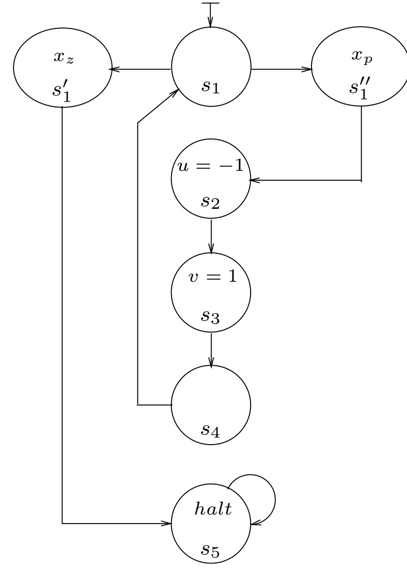

Fig. 3. The Kripke structure corresponding to the counter machine of Example 8.

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

Consider the following formulas.

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

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

Lemma 9. *Given a counter machine $\mathcal{M}$, let $\mathcal{K}$, φ, and φ' 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 G operator can be expressed by the W (Weak Until) operator, and similarly for U and R (Release), Lemma 9

Fig. 4. The Boolean Kripke structure corresponding to the counter machine of Example 8.

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 III-A for a logic with the modalities EF and EX is maximal. We conclude that extending all the standard temporal logics, except for the EF logic, makes the model-checking problem undecidable.

Corollary 10. *The model-checking problem is undecidable for the temporal logics CTL***, LTL , RTL, CTL, STL, UB, and EG, extended by the atomic assertion* $\mathsf{Sum}(v) \geq c$.

Boolean Kripke structure: Our setting considers a quantitative Kripke structure and a specification over its accumulated values. One may consider a possibly simpler question, concerning a Boolean Kripke structure and a specification over 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 of time. We can then have specifications with new atomic assertions, like $\mathsf{Avg}(p) \geq \frac{1}{2}$.

Is the model checking of such a specification decidable? No. It is undecidable by a simple reduction from our setting. Instead of using the numerical variables u and v with values $\{1, 0, -1\}$, we can use the atomic propositions p and q, and represent the numeric values by $-1 = \mathtt{FF}$, $0 = \mathtt{TF}$ and $1 = \mathtt{TT}$. The Boolean Kripke structure that corresponds to the machine in Example 8 is shown in Figure 4, and ψ_{Proper} is adjusted as follows.

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

IV. 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$^{\text{lim}}$ formula (an LTL formula extended by path-accumulation assertions, as defined in Section II). An example of such an extended formula is:

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

Given an LTL$^{\text{lim}}$ 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) $\mathcal{A}$ [33] 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 the limit-average assertions: Consider an LTL$^{\text{lim}}$ 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$^{\text{lim}}$ formula φ with n limit-average assertions, $\theta_1, \ldots, \theta_n$, the LTL$^{\text{lim}}$ formula φ' defined below is equivalent to φ.

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

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 denote 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 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, $\delta : Q \times \Sigma \rightarrow 2^Q$ 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 [33].

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 $[\![v]\!]_{\langle s,q\rangle} = [\![v]\!]_s$.

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 [5][4] 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 11. *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 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 $\pi' = z_i, z_2, z_3, \ldots$. 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, and show that $[\![v]\!]_{z_i} - [\![v]\!]_{x_{\rho(i)}}$ 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.

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

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.

For showing that π and π' have the same limit-average values of v, we need to show that the $\lim_{j\to\infty} \frac{\sum_0^j [\![v]\!]_{z_j}}{j} - \frac{\sum_0^{\rho(j)} [\![v]\!]_{x_{\rho(j)}}}{\rho(j)} = 0$. Indeed, for every $j \in \mathbb{N}$ we have that $\frac{\sum_0^j [\![v]\!]_{z_j}}{j} - \frac{\sum_0^{\rho(j)} [\![v]\!]_{x_{\rho(j)}}}{\rho(j)} \leq \frac{c(j-\rho(j))}{\rho(j)}$, which converges to 0. ∎

We can now show how to adjust the emptiness algorithm of [5] for handling the Büchi fairness condition.

Lemma 12. *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 [5], 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 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 [5].

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 11, 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 13. *The model-checking problem for LTL$^{\lim}$ is decidable.*

Note that model checking an LTL$^{\lim}$ 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$^{\lim}$ with a regular layer – one in which the path formulas may also contain regular expressions.

Complexity: The complexity of the construction is roughly exponential in the size of the Kripke structure, doubly exponential in the LTL formula, and triply exponential in the number of numeric variables. More formally, for a Kripke structure with n states and an LTL formula of length m with k

[4]The paradigm in [5] is different from ours, as the limit-average formula there constitutes the acceptance conditions for the automata.

numeric variables, the construction-complexity is bounded by $O((2^{n\times 2^m})^k)$. It conveys the following complexities: 2^m for translating an LTL formula to a Büchi automaton, $n \times 2^m$ for the product of the Kripke structure and the Büchi automaton, $2^{n\times 2^m}$ for the number of simple cycles in the product, and $(2^{n\times 2^m})^k$ for solving the convex-hull intersection questions, involving $2^{n\times 2^m}$ points of dimension k.

A lower bound for the required complexity is an open problem. Specifically, one may seek a construction that does not rely on the simple cycles of the Kripke structure, for removing the exponential dependency in the Kripke structure.

Acknowledgment. We thank Dejan Nickovic for stimulating discussions on controlled accumulation.

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.

REFERENCES

[1] R. Alur and T. Henzinger, "A really temporal logic," *Journal of the ACM*, vol. 41, no. 1, pp. 181–204, 1994.

[2] C. Courcoubetis, R. Alur, and D. Dill, "Model-checking for probabilistic real-time system," in *Proc. of ICALP 91*, ser. LNCS. Springer, 1991.

[3] M. Droste, W. Kuich, and H. Vogler, "Monograph in theoretical computer science: Eatcs series," in *Handbook of Weighted Automata*. Springer, 2009.

[4] K. Chatterjee, L. Doyen, and T. A. Henzinger, "Quantitative languages," in *Proc. of CSL*, ser. LNCS 5213. Springer, 2008, pp. 385–400.

[5] R. Alur, A. Degorre, O. Maler, and G. Weiss, "On omega-languages defined by mean-payoff conditions," in *FOSSACS*, ser. LNCS, vol. 5504, 2009, pp. 333–347.

[6] R. Bloem, K. Chatterjee, T. A. Henzinger, and B. Jobstmann, "Better quality in synthesis through quantitative objectives," in *CAV*, ser. LNCS, vol. 5643, 2009, pp. 140–156.

[7] Z. Manna and A. Pnueli, "The modal logic of programs," in *ICALP*, ser. LNCS, vol. 71, 1979, pp. 385–409.

[8] O. Kupferman and M. Vardi, "Memoryful branching-time logics," in *Proc. 21st LICS*, 2006, pp. 265–274.

[9] L. de Alfaro, M. Faella, T. Henzinger, R. Majumdar, and M. Stoelinga, "Model checking discounted temporal properties," *Theoretical Computer Science*, vol. 345, no. 1, pp. 139–170, 2005.

[10] M. P. Schützenberger, "On the definition of a family of automata," *Information and Control*, vol. 4, pp. 245–270, 1961.

[11] W. Kuich and A. Salomaa, *Semirings, Automata, Languages*, ser. Monographs in Theoretical Computer Science. Springer, 1986, vol. 5.

[12] K. Culik II and J. Karhumäki, "Finite automata computing real functions," *SIAM J. Comput.*, vol. 23, no. 4, pp. 789–814, 1994.

[13] M. Droste and D. Kuske, "Skew and infinitary formal power series," in *Proc. of ICALP*, ser. LNCS 2719. Springer, 2003, pp. 426–438.

[14] Z. Ésik and W. Kuich, "An algebraic generalization of omega-regular languages," in *Proc. of MFCS*, ser. LNCS 3153, 2004, pp. 648–659.

[15] M. Droste and P. Gastin, "Weighted automata and weighted logics," *Theoretical Computer Science*, vol. 380, pp. 69–86, 2007.

[16] M. Droste, W. Kuich, and G. Rahonis, "Multi-valued MSO logics over words and trees," *Fundamenta Informaticae*, vol. 84, pp. 305–327, 2008.

[17] A. Gurfinkel and M. Chechik, "Multi-valued model checking via classical model checking," in *Proc. of CONCUR*, ser. LNCS 2761. Springer, 2003, pp. 263–277.

[18] O. Kupferman and Y. Lustig, "Lattice automata," in *Proc. of VMCAI*, ser. LNCS 4349. Springer, 2007, pp. 199–213.

[19] M. Droste and I. Meinecke, "Describing average- and longtime-behavior by weighted MSO logics," in *MFCS*, 2010, pp. 537–548.

[20] U. Zwick and M. Paterson, "The complexity of mean payoff games on graphs," *Theoretical Computer Science*, vol. 158, pp. 343–359, 1996.

[21] H. Bjorklund, S. Sandberg, and S. Vorobyov, "A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games," in *MFCS'04*, 2004, pp. 673–685.

[22] A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga, "Resource interfaces," in *Proc. of EMSOFT: Embedded Software*, ser. LNCS 2855. Springer, 2003, pp. 117–133.

[23] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin, "Generalized mean-payoff and energy games," in *FSTTCS*, ser. LIPIcs, vol. 8, 2010, pp. 505–516.

[24] J. Esparza, "Decidability and complexity of Petri net problems - an introduction," in *Petri Nets*, ser. LNCS, vol. 1491, 1996, pp. 374–428.

[25] M. Presburger, "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*, pp. 92–101, 1929.

[26] S. R. Kosaraju and G. F. Sullivan, "Detecting cycles in dynamic graphs in polynomial time (preliminary version)," in *STOC*, 1988, pp. 398–406.

[27] E. A. Emerson and J. Y. Halpern, ""sometimes" and "not never" revisited: On branching versus linear time," in *POPL*, 1983, pp. 127–140.

[28] A. Pnueli, "The temporal logic of programs," in *FOCS*, 1977, pp. 46–57.

[29] A. P. Sistla and L. D. Zuck, "Reasoning in a restricted temporal logic," *Inf. Comput.*, vol. 102, no. 2, pp. 167–195, 1993.

[30] E. A. Emerson and E. M. Clarke, "Using branching time temporal logic to synthesize synchronization skeletons," *Sci. Comput. Program.*, vol. 2, no. 3, pp. 241–266, 1982.

[31] R. Alur and T. A. Henzinger, "Computer-aided verification: An introduction to model building and model checking for concurrent systems," *Book in preparation*, 1999.

[32] M. Ben-Ari, A. Pnueli, and Z. Manna, "The temporal logic of branching time," *Acta Inf.*, vol. 20, pp. 207–226, 1983.

[33] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification (preliminary report)," in *LICS*, 1986, pp. 332–344.