Foundations of Boolean Stream Runtime Verification[⋆]

Laura Bozzelli[1] and César Sánchez[2,3]

[1] Technical University of Madrid (UPM), Madrid, Spain
[2] IMDEA Software Institute, Madrid, Spain
[3] Institute for Information Security, CSIC, Spain

Abstract. Stream runtime verification (SRV), pioneered by the tool LOLA, is a declarative approach to specify synchronous monitors. In SRV, monitors are described by specifying dependencies between output streams of values and input streams of values. The declarative nature of SRV enables a separation between (1) the evaluation algorithms, and (2) the monitor storage and its individual updates. This separation allows SRV to be lifted from conventional failure monitors into richer domains to collect statistics of traces. Moreover, SRV allows to easily identify specifications that can be efficiently monitored online, and to generate efficient schedules for offline monitors.

In spite of these attractive features, many important theoretical problems about SRV are still open. In this paper, we address complexity, expressiveness, succinctness, and closure issues for the subclass of Boolean SRV (BSRV) specifications. Additionally, we show that for this subclass, offline monitoring can be performed with only two passes (one forward and one backward) over the input trace in spite of the alternation of past and future references in the BSRV specification.

1 Introduction

Runtime verification (RV) has emerged in the last decades as an applied formal technique for software reliability. In RV, a specification, expressing correctness requirements, is automatically translated into a *monitor*. Such a monitor is then used to check either the *current* execution of a running system, or a finite set of *recorded* executions with respect to the given specification. The former scenario is called *online* monitoring, while the latter one is called *offline* monitoring. Online monitoring is used to detect and possibly handle (e.g., by the execution of additional repair code) violations of the specification when the system is in operation. On the other hand, offline monitoring is used in post-mortem analysis and it is convenient for testing large systems before deployment. Unlike static verification (such as model-checking) which formally checks that all the (infinite) executions or traces of a system satisfy the specification, RV only considers a single finite trace. Thus, this methodology sacrifices completeness guarantees to obtain an immediately applicable and formal extension of testing. See [17,14] for modern surveys on runtime verification.

[⋆] This work was funded in part by Spanish MINECO Project "TIN2012-39391-C04-01 STRONGSOFT" and by Spanish MINECO Project "TIN2012-38137-C02 VIVAC".

Stream Runtime Verification. The first specification formalisms proposed for runtime verification were based on specification languages for static verification, typically LTL [18] or past LTL adapted for finite paths [15,9,5]. Other formalisms for expressing monitors include regular expressions [23], rule based specifications as proposed in the logic Eagle [1], or rewriting [22]. Stream runtime verification (SRV), first proposed in the tool LOLA [8], is an alternative to define monitors for synchronous systems. In SRV, specifications declare explicitly the dependencies between *input* streams of values (representing the observable behavior of the system) and *output* streams of values (describing error reports and diagnosis information). These dependencies can relate the current value of an output stream with the values of the same or other streams in the present moment, in past instants (like in past temporal formulas) or in future instants. A similar approach to describe temporal relations as streams was later introduced as temporal testers [21].

Stream runtime verification offers two advantages to the description of monitors. First, SRV separates the algorithmic aspects of the runtime evaluation (by explicitly declaring the data dependencies) from the specific individual operations performed at each step (which depend on the type of data being observed, manipulated and stored). In this manner, well-known evaluation algorithms for monitoring Boolean observations – for example those from temporal logics – can be generalized to richer data domains, producing monitors that collect statistics about traces. Similarly to the Boolean case, the first approaches for collecting statistics from running traces were based on extensions of LTL [10]. SRV can be viewed as a generalization of these approaches to streams. Other modern approaches to the runtime verification for statistic collection extend first-order LTL [4,2,3]. Moreover, the declarative nature of SRV allows to identify specifications that are amenable for efficient online monitoring, essentially those specifications whose values can be resolved by past and present observations. Additionally, the analysis of dependencies also allows to generate efficient offline monitors by scheduling passes over the dumped traces, where the number of passes (back and forth) depends on the number of alternations between past and future references in the specification.

SRV can be seen as a variation of synchronous languages [7] – like Esterel [6], Lustre [13] or Signal [11] – specifically designed for observing traces of systems, removing the causality assumption. In synchronous languages, stream values can only depend on past or present values, while in SRV a dependency on future values is additionally allowed to describe future temporal observations. In recent years, SRV has also been extended to real-time systems [20,12].

When used for synthesizing monitors, SRV specifications need to be *well-defined*: for every input there is a unique corresponding output stream. However, as with many synchronous languages, the declarative style of SRV allows specifications that are not well-defined: for some observations, either there is no possible output (*over-definedness*) or there is more than one output (*under-definedness*). This anomaly is caused by circular dependencies, and in [8], a syntactical constraint called *well-formedness* is introduced in order to ensure the absence of circular dependencies, and guarantee well-definedness.

Our Contribution. In spite of its applicability, several foundational theoretical problems of SRV have not been studied so far. In this paper, we address complexity, expressiveness, succinctness, and closure properties for Boolean SRV (BSRV). Our results can be summarized as follows:

- we establish the complexity of checking whether a specification is under-defined, over-defined or well-defined. Apart from the theoretical significance of these results, many important practical properties of specifications (like semantic equivalence, implication and redundancy) can be reduced to the decision problems above.
- BSRV specifications can be naturally interpreted as language recognizers, where one selects the inputs for which the specification admits some output. We prove that in this setting, BSRV captures precisely the class of regular languages. We also show efficient closure constructions for many language operations. Additionally, BSRV specifications can be exponentially more succinct than nondeterministic finite-state automata (NFA).
- Finally, based on the construction of the NFA associated with a *well-defined* BSRV specification, we show how to schedule an offline algorithm with only two passes, one forward and one backward. This gives a partial answer (for the Boolean case) to the open problem of reducing the number of passes in offline monitoring for *well-formed* SRV specifications [8].

The rest of the paper is structured as follows. Section 2 revisits SRV. In Section 3 we establish expressiveness, succinctness, and closure results for BSRV specifications when interpreted as language recognizers. In Section 4, we describe the two-pass offline monitoring algorithm. Section 5 is devoted to the decision problems for BSRV specifications. Finally, Section 6 concludes. Due to lack of space, some proofs are omitted and are included in the longer version of this document[1].

2 Stream Runtime Verification (SRV)

In this Section, we recall the SRV framework [8]. We focus on SRV specifications over stream variables of the same type (with emphasis on the Boolean type).

A type T is a tuple $T = \langle D, F \rangle$ consisting of a countable value domain D and a finite collection F of interpreted function symbols f, where f denotes a computable function from D^k to D and $k \geq 0$ is the specific arity of f. Note that 0-ary function symbols (constants) are associated with individual values. In particular, we consider the *Boolean type*, where $D = \{0, 1\}$ and F consists of the Boolean operators $\wedge$ and $\vee$ and $\neg$. A *stream of type* T is a non-empty *finite* word w over the domain D of T. Given such a stream w, $|w|$ is the length of w and for all $1 \leq i \leq |w|$, $w(i)$ is the ith letter of w (the value of the stream at time step i). The stream w is *uniform* if there is $d \in D$ such that w is in d^*.

For a finite set Z of (stream) variables, a *stream valuation of type T over Z* is a mapping σ assigning to each variable $z \in Z$, a stream $\sigma(z)$ of type T such that

[1] The longer version can be obtained at `http://software.imdea.org/~cesar/`

the streams associated with the different variables in Z have the same length N for some $N \geq 1$. We also say that N is the length of σ, which is denoted by $|\sigma|$.

Remark 1. Note that for the Boolean type, a stream valuation σ over Z can be identified with the non-empty word over 2^Z of length $|\sigma|$ whose ith symbol, written $\sigma(i)$, is the set of variables $z \in Z$ such that $\sigma(z)(i) = 1$.

Stream Expressions. Given a finite set Z of variables, the set of *stream expressions* E *of type* T *over* Z is inductively defined by the following syntax:

$$\mathsf{E} := \tau \;\big|\; \tau[\ell|c] \;\big|\; f(\mathsf{E_1}, \ldots, \mathsf{E_k})$$

where τ is either a constant of type T or a variable in Z, ℓ is a non-null integer, c is a constant of type T, and $f \in F$ is a function of type T and arity $k > 0$. Informally, $\tau[\ell|c]$ refers to the value of τ offset ℓ positions from the current position, and the constant c is the *default* value of type T assigned to positions from which the offset is after the end or before the beginning of the stream. Stream expressions E of type T over Z are interpreted over stream valuations σ of type T over Z. The *valuation* of E with respect to σ, written $[\![\mathsf{E}, \sigma]\!]$, is the stream of type T and length $|\sigma|$ inductively defined as follows for all $1 \leq i \leq |\sigma|$:

- $[\![c, \sigma]\!](i) = c$ and $[\![z, \sigma]\!](i) = \sigma(z)(i)$ for all $z \in Z$
- $[\![\tau[\ell|c], \sigma]\!](i) = \begin{cases} [\![\tau, \sigma]\!](i+\ell) & \text{if } 1 \leq i+\ell \leq |\sigma| \\ c & \text{otherwise} \end{cases}$
- $[\![f(\mathsf{E_1}, \ldots, \mathsf{E_k}), \sigma]\!](i) = f([\![\mathsf{E_1}, \sigma]\!](i), \ldots, [\![\mathsf{E_k}, \sigma]\!](i))$

For the *Boolean type*, we use some shortcuts: $\mathsf{E_1} \rightarrow \mathsf{E_2}$ stands for $\neg\mathsf{E_1} \vee \mathsf{E_2}$, $\mathsf{E_1} \leftrightarrow \mathsf{E_2}$ stands for $(\mathsf{E_1} \rightarrow \mathsf{E_2}) \wedge (\mathsf{E_2} \rightarrow \mathsf{E_1})$, and *if* E *then* $\mathsf{E_1}$ *else* $\mathsf{E_2}$ stands for $(\mathsf{E} \wedge \mathsf{E_1}) \vee (\neg\mathsf{E} \wedge \mathsf{E_2})$. Additionally, we use first and last for the Boolean stream expressions $0[-1|1]$ and $0[+1|1]$, respectively. Note that for a Boolean stream, first is 1 precisely at the first position, and last is 1 precisely at the last position.

Example 1. Consider the following *Boolean* stream expression E over $Z = \{x\}$:

$$\mathsf{E} := \textit{if } x \textit{ then } x \textit{ else } x[1|0]$$

For every *Boolean* stream valuation σ over Z such that $\sigma(Z) \in (01)^+$, the valuation of E with respect to σ is the uniform Boolean stream $1^{|\sigma|}$.

Stream Runtime Verification Specification Language (SRV). Given a finite set X of *input* variables and a set $Y = \{y_1, \ldots, y_n\}$ of *output* variables with $X \cap Y = \emptyset$, an SRV φ of type T over X and Y is a set of equations

$$\varphi := \{y_1 = \mathsf{E_1}, \ldots, y_n = \mathsf{E_n}\}$$

where $\mathsf{E_1}, \ldots, \mathsf{E_n}$ are stream expressions of type T over $X \cup Y$. Note that there is exactly one equation for each output variable. A stream valuation of φ is a stream valuation of type T over $X \cup Y$, while an *input* (resp., *output*) of φ is a stream valuation of type T over X (resp., Y). Given an input σ_X of φ and an output σ_Y of φ such that σ_X and σ_Y have the same length, $\sigma_X \cup \sigma_Y$ denotes the stream valuation of φ defined in the obvious way. The SRV φ describes a relation, written $[\![\varphi]\!]$, between inputs σ_X of φ and outputs σ_Y of φ, defined as follows: $(\sigma_X, \sigma_Y) \in [\![\varphi]\!]$ *iff* $|\sigma_X| = |\sigma_Y|$ and for each equation $y_j = \mathsf{E_j}$ of φ,

$$[\![y_j, \sigma]\!] = [\![\mathsf{E_j}, \sigma]\!] \qquad \text{where } \sigma = \sigma_X \cup \sigma_Y$$

If $(\sigma_X, \sigma_Y) \in [\![\varphi]\!]$, we say that the stream valuation $\sigma_X \cup \sigma_Y$ is a *valuation model of* φ (associated with the input σ_X). Note that in general, for a given input σ_X, there may be zero, one, or multiple valuation models associated with σ_X. This leads to the following notions for an SRV φ:

– *Under-definedness:* for some input σ_X, there are at least two distinct valuation models of φ associated with σ_X.
– *Over-definedness:* for some input σ_X, there is no valuation model of φ associated with σ_X.
– *Well-definedness:* for each input σ_X, there is exactly one valuation model of φ associated with σ_X.

Note that an SRV φ may be both under-defined and over-defined, and φ is well-defined iff it is neither under-defined nor over-defined. For runtime verification, SRV serves as a query language on program behaviors (input streams) from which one computes a unique answer (the output streams). In this context, a specification is useful only if it is well-defined. However, in practice, it is convenient to distinguish *intermediate* output variables from *observable* output variables separating output streams that are of interest to the user from those that are used only to facilitate the computation of other streams. This leads to a more general notion of well-definedness. Given a subset $Z \subseteq Y$ of output variables, an SRV φ is *well-defined with respect to* Z if for each input σ_X, there is exactly one stream valuation σ_Z over Z having the same length as σ_X such that $\sigma_X \cup \sigma_Z$ can be extended to some valuation model of φ (*uniqueness of the output streams over* Z).

Analogously, we consider a notion of semantic equivalence between SRV of the same type and having the same input variables, which is parameterized by a set of output variables. Formally, given an SRV φ of type T over X and Y, an SRV φ' of type T over X and Y', and $Z \subseteq Y \cap Y'$, we say that φ and φ' are equivalent with respect to Z if for each valuation model σ of φ, there is a valuation model σ' of φ' such that σ and σ' coincide on $X \cup Z$, and vice versa. Moreover, if $Y' \supseteq Y$, then we say that φ' is *φ-equivalent* if φ and φ' are equivalent with respect to Y.

Remark 2. In the rest of the paper, we focus on Boolean SRV (BSRV for short). Thus, in the following, we omit the reference to the type T in the various definitions. We assume that the offsets ℓ in the subexpressions $\tau[\ell|c]$ of a BSRV are encoded in unary. For a Boolean stream expression E, we denote by $||\mathsf{E}||$ the offset ℓ if E is a stream expression of the form $\tau[\ell|c]$; otherwise, $||\mathsf{E}||$ is 1. The size $|\varphi|$ of a BSRV φ is defined as $|\varphi| := \sum_{\mathsf{E} \in SE(\varphi)} ||\mathsf{E}||$, where $SE(\varphi)$ is the set of stream subexpressions of φ.

Example 2. Consider the following BSRV over $X = \{x\}$ and $Y = \{y\}$:

$$\varphi_1 := \{y = x \wedge y\} \quad \varphi_2 := \{y = x \wedge \neg y\} \quad \varphi_3 := \{y = \mathit{if}\ x\ \mathit{then}\ x[2|0]\ \mathit{else}\ x[-2|0]\}$$

The specification φ_1 is under-defined since $(1^N, 0^N)$ and $(1^N, 1^N)$ are two valuation models for each $N \geq 1$. On the other hand, the specification φ_2 is over-defined since for each $N \geq 1$, there is no valuation model associated with the input 1^N. Finally, the specification φ_3 is well-defined.

3 BSRV as Language Recognizers

BSRV can be interpreted as a simple declarative formalism to specify languages of non-empty finite words. Formally, we associate to a BSRV φ over X and Y, the language $\mathcal{L}(\varphi)$ of non-empty finite words over 2^X (or, equivalently, input stream valuations) for which the specification φ admits a valuation model, i.e.,

$$\mathcal{L}(\varphi) := \{\sigma_X \mid (\sigma_X, \sigma_Y) \in [\![\varphi]\!] \text{ for some } \sigma_Y\}$$

Example 3. Let $X = \{x\}$, $Y = \{y\}$, and $\varphi = \{y = \mathit{if}\ \mathsf{E}\ \mathit{then}\ y\ \mathit{else}\ \neg y\}$, where

$$\mathsf{E} := \big(\mathsf{first} \rightarrow (x \wedge y)\big) \wedge \big(y \rightarrow \neg y[+1|0]\big) \wedge \big(\neg y \rightarrow (x[+1|1] \wedge y[+1|1])\big)$$

A pair (σ_X, σ_Y) is a valuation model of φ iff the valuation of the stream expression E w.r.t. $\sigma_X \cup \sigma_Y$ is in 1^+ iff $\sigma_X(x)(i) = 1$ for all odd positions i. Hence, $\mathcal{L}(\varphi)$ is the set of Boolean streams which assume the value 1 at the odd positions.

In the following, we show that BSRV, as language recognizers, are effectively equivalent to nondeterministic finite automata (NFA) on finite words. While the translation from NFA to BSRV can be done in polynomial time, the converse translation involves an *unavoidable* singly exponential blowup. Moreover, BSRV turn out to be effectively and *efficiently* closed under many language operations.

In order to present our results, we shortly recall the class of NFA on finite words. An NFA $\mathcal{A}$ over a finite input alphabet I is a tuple $\mathcal{A} = \langle Q, q_0, \delta, F\rangle$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times I \rightarrow 2^Q$ is the transition function, and $F \subseteq Q$ is a set of accepting states. Given an input word $w \in I^*$, a run π of $\mathcal{A}$ over w is a sequence of states $\pi = q_1, \ldots, q_{|w|+1}$ such that q_1 is the initial state and for all $1 \leq i \leq |w|$, $q_{i+1} \in \delta(q_i, w(i))$. The run π is accepting if it leads to an accepting state (i.e, $q_{|w|+1} \in F$). The language $\mathcal{L}(\mathcal{A})$ accepted by $\mathcal{A}$ is the set of non-empty finite words w over I such that there is an accepting run of $\mathcal{A}$ over w. $\mathcal{A}$ is *universal* if $\mathcal{L}(\mathcal{A}) = I^+$. A language over non-empty finite words is *regular* if it is accepted by some NFA. An NFA is *unambiguous* if for each input word w, there is at most one accepting run on w.

Fix a BSRV φ on X and Y. In order to build an NFA accepting $\mathcal{L}(\varphi)$, we define an encoding of the valuation models of φ. For this, we associate to φ two parameters, the *back reference distance* $b(\varphi)$ and the *forward reference distance* $f(\varphi)$:

$$b(\varphi) := \mathit{max}(0, \{\ell \mid \ell > 0 \text{ and } \varphi \text{ contains a subexpression of the form } z[-\ell, c]\})$$
$$f(\varphi) := \mathit{max}(0, \{\ell \mid \ell > 0 \text{ and } \varphi \text{ contains a subexpression of the form } z[\ell, c]\})$$

For a stream valuation σ of φ and an expression E of φ, the value of E w.r.t. σ at a time step i is completely specified by the values of σ at time steps j such that $i - b(\varphi) \leq j \leq i + f(\varphi)$. We define the following alphabets:

$$A := 2^{X \cup Y} \qquad A_\perp := A \cup \{\perp\} \qquad P_\varphi := (A_\perp)^{b(\varphi)} \times A \times (A_\perp)^{f(\varphi)}$$

where $\perp$ is a special symbol. Note that a stream valuation of φ corresponds to a non-empty finite word over the alphabet A, and the cardinality of P_φ is singly exponential in the size of φ. For an element $p = (a_{-b(\varphi)}, \ldots, a_{-1}, a_0, a_1, \ldots, a_{f(\varphi)})$ of P_φ, the component a_0, called the *main value of* p, intuitively represents the value of some stream valuation σ at some time step i, while $a_{-b(\varphi)}, \ldots, a_{-1}$ (resp., $a_1, \ldots, a_{f(\varphi)}$) represent the values of σ at the previous $b(\varphi)$ (resp., next $f(\varphi)$) time steps, if any (the symbol $\perp$ is used to denote the absence of a previous or next time step). Let τ be either a Boolean constant or a variable in $X \cup Y$, and $a \in A$. Then, the Boolean *value* of τ in a is τ if τ is a constant, otherwise the value is 1 iff $\tau \in a$. For a Boolean stream expression E over $X \cup Y$ and an element $p = (a_{-b(\varphi)}, \ldots, a_{-1}, a_0, a_1, \ldots, a_{f(\varphi)})$ of P_φ, the *value* $[\![\mathsf{E}, p]\!]$ *of* E *with respect to* p is the computable Boolean value inductively defined as follows:

- $[\![c, p]\!] = c$ and $[\![z, p]\!] =$ the value of z in a_0
- $[\![\tau[\ell|c], p]\!] = \begin{cases} \text{the value of } \tau \text{ in } a_\ell & \text{if } -b(\varphi) \leq \ell \leq f(\varphi) \text{ and } a_\ell \neq \perp \\ c & \text{otherwise} \end{cases}$
- $[\![f(\mathsf{E}_1, \ldots, \mathsf{E}_k), p]\!] = f([\![\mathsf{E}_1, p]\!], \ldots, [\![\mathsf{E}_k, p]\!])$

We denote by Q_φ the subset of P_φ consisting of the elements p of P_φ such that for each equation $y = \mathsf{E}$ of φ, the value of y with respect to p coincides with the value of E with respect to p. Let $\#$ be an additional special symbol (which will be used as initial state of the NFA associated with φ). An *expanded valuation model* of φ is a word of the form $\# \cdot w$ such that w is a non-empty finite word w over the alphabet Q_φ satisfying the following:

- $w(1)$ is of the form $(\perp, \ldots, \perp, a_0, a_1, \ldots, a_{f(\varphi)})$;
- $w(|w|)$ is of the form $(a_{-b(\varphi)}, \ldots, a_{-1}, a_0, \perp, \ldots, \perp)$;
- if $1 \leq i < |w|$ and $w(i) = (a_{-b(\varphi)}, \ldots, a_{-1}, a_0, a_1, \ldots, a_{f(\varphi)})$, then there is $d \in A_\perp$ such that $w(i+1)$ is of the form $(a_{-b(\varphi)+1}, \ldots, a_{-1}, a_0, a_1, \ldots, a_{f(\varphi)}, d)$.

For an expanded valuation model $\# \cdot w$ of φ, the *associated stream valuation* $\sigma(w)$ is the stream valuation of φ of length $|w|$ whose ith element is the main value of the ith element of w. By construction, we easily obtain that $\sigma(w)$ is a valuation model of φ and, more precisely, the following lemma holds.

Lemma 1. *The mapping assigning to each expanded valuation model* $\# \cdot w$ *of* φ *the associated stream valuation* $\sigma(w)$ *is a bijection between the set of expanded valuation models of* φ *and the set of valuation models of* φ.

By the above characterization of the set of valuations models of a BSRV φ, we easily obtain the following result.

Theorem 1 (From BSRV to NFA). *Given a BSRV φ over X and Y, one can construct in singly exponential time an NFA $\mathcal{A}_\varphi$ over the alphabet 2^X accepting $\mathcal{L}(\varphi)$ whose set of states is $Q_\varphi \cup \{\#\}$. Moreover, for each input σ_X, the set of accepting runs of $\mathcal{A}_\varphi$ over σ_X is the set of expanded valuation models of φ encoding the valuation models of φ associated with the input σ_X.*

Proof. The NFA $\mathcal{A}_\varphi$ is defined as $\mathcal{A}_\varphi = \langle Q_\varphi \cup \{\#\}, \#, \delta_\varphi, F_\varphi \rangle$, where F_φ is the set of elements of Q_φ of the form $(a_{-b(\varphi)}, \ldots, a_{-1}, a_0, \bot, \ldots, \bot)$, and $\delta(p, \iota)$ is defined as follows for all states p and input symbol $\iota \in 2^X$:

– if $p = \#$, then $\delta_\varphi(p, \iota)$ is the set of states of the form $(\bot, \ldots, \bot, a_0, a_1, \ldots, a_{f(\varphi)})$ such that $a_0 \cap X = \iota$;
– if $p = (a_{-b(\varphi)}, \ldots, a_{-1}, a_0, a_1, \ldots, a_{f(\varphi)}) \in Q_\varphi$, then $\delta_\varphi(p, \iota)$ is the set of states of the form $(a_{-b(\varphi)+1}, \ldots, a_{-1}, a_0, a_1, \ldots, a_{f(\varphi)}, d)$ for some $d \in A_\bot$ whose main value a satisfies $a \cap X = \iota$.

By construction, for each input σ_X, the set of accepting runs of $\mathcal{A}_\varphi$ over σ_X coincides with the set of expanded valuation models $\# \cdot w$ of φ such that the stream valuation $\sigma(w)$ is associated with the input σ_X. Thus, by Lemma 1, the result follows. $\square$

For the converse translation from NFA to BSRV, we show the following.

Theorem 2 (From NFA to BSRV). *Given an NFA $\mathcal{A}$ over the input alphabet 2^X, one can construct in polynomial time a BSRV $\varphi_\mathcal{A}$ with set of input variables X such that $\mathcal{L}(\varphi_\mathcal{A}) = \mathcal{L}(\mathcal{A})$.*

Proof. Let $\mathcal{A} = \langle Q, q_0, \delta, F \rangle$. We construct a BSRV $\varphi_\mathcal{A}$ over the set of input variables X as follows. First, for each input symbol ι, we use a Boolean expression E_ι over X, encoding the input symbol ι, defined as $\mathsf{E}_\iota := (\bigwedge_{x \in \iota} x) \wedge (\bigwedge_{x \in X \setminus \iota} \neg x)$. The set Y of output variables of $\varphi_\mathcal{A}$ is defined as follows:

$$Y = \bigcup_{q \in Q} \{\mathsf{q}\} \cup \{\mathsf{control}\}$$

Thus, we associate to each state $q \in Q$, an output variable q, whose associated equation is the trivial one given by $\mathsf{q} = \mathsf{q}$. The equation for the output variable $\mathsf{control}$ is given by

$$\mathsf{control} = \textit{if } \mathsf{E}_{\mathsf{ev}} \textit{ then } \mathsf{control} \textit{ else } \neg\mathsf{control}$$

where the boolean stream expression E_{ev} describes accepting runs of the NFA $\mathcal{A}$ and is defined as follows:

$$\mathsf{E}_{\mathsf{ev}} = \underbrace{\bigvee_{q \in Q} (\mathsf{q} \wedge \bigwedge_{p \in Q \setminus \{q\}} \neg\mathsf{p})}_{\text{at each step, } \mathcal{A} \text{ is exactly in one state}} \wedge \underbrace{(\mathsf{first} \longrightarrow \mathsf{q}_0)}_{\text{a run of } \mathcal{A} \text{ starts at the initial state}} \wedge$$

$$\underbrace{\bigwedge_{q \in Q} \bigwedge_{\iota \in I} \big((\mathsf{q} \wedge \mathsf{E}_\iota) \longrightarrow \bigvee_{p \in \delta(q,\iota)} \mathsf{p}[+1|1]\big)}_{\text{the evolution of } \mathcal{A} \text{ is } \delta\text{-consistent}} \wedge \underbrace{\big(\mathsf{last} \longrightarrow \bigvee_{(q,\iota) \in \{(q,\iota) \mid \delta(q,\iota) \cap F \neq \emptyset\}} (\mathsf{q} \wedge \mathsf{E}_\iota)\big)}_{\text{the run of } \mathcal{A} \text{ is accepting}}$$

By construction, it easily follows that given an input stream valuation σ_X, there is a valuation model of $\varphi_{\mathcal{A}}$ associated with the input σ_X *if and only if* there is a stream valuation σ associated with the input σ_X such that the valuation of E_{ev} with respect to σ is a uniform stream in 1^+ *if and only if* there is an accepting run of $\mathcal{A}$ over the input σ_X. Hence, the result follows. □

Corollary 1. *BSRV, when interpreted as language recognizers, capture the class of regular languages over non-empty finite words.*

Succinctness Issues. It turns out that the singly exponential blow-up in Theorem 1 cannot be avoided. To prove this we first show a linear time translation from standard linear temporal logic LTL with past over finite words (which captures a subclass of regular languages) into BSRV. Recall that formulas ψ of LTL with past over a finite set AP of atomic propositions are defined as follows:

$$\psi := p \mid \neg\psi \mid \psi \vee \psi \mid \bigcirc\psi \mid \ominus\psi \mid \psi\,\mathcal{U}\,\psi \mid \psi\,\mathcal{S}\,\psi$$

where $p \in AP$ and $\bigcirc$, $\ominus$, $\mathcal{U}$, and $\mathcal{S}$ are the 'next', 'previous', 'until', and 'since' temporal modalities. For a finite word w over 2^{AP} and a position $1 \leq i \leq |w|$, the satisfaction relation $(w,i) \models \psi$ is defined as follows (we omit the rules for the boolean connectives and the atomic propositions, which are standard):

$$\begin{array}{lll}
(w,i) \models & \bigcirc\psi & \Leftrightarrow\ i+1 \leq |w| \text{ and } (w,i+1) \models \psi \\
(w,i) \models & \ominus\psi & \Leftrightarrow\ i > 1 \text{ and } (w,i-1) \models \psi \\
(w,i) \models & \psi_1\,\mathcal{U}\,\psi_2 & \Leftrightarrow \exists\, i \leq j \leq |w|,\ (w,j) \models \psi_2 \text{ and } \forall\, i \leq h < j,\ (w,h) \models \psi_1 \\
(w,i) \models & \psi_1\,\mathcal{S}\,\psi_2 & \Leftrightarrow \exists\, 1 \leq j \leq i,\ (w,j) \models \psi_2 \text{ and } \forall\, j < h \leq i,\ (w,h) \models \psi_1
\end{array}$$

The language $\mathcal{L}(\psi)$ of a LTL formula ψ is the set of non-empty finite words w over 2^{AP} such that $(w,1) \models \psi$.

Proposition 1. *LTL with past can be translated in linear time into BSRV.*

Proof. Let ψ be a formula of LTL with past over a finite set AP of atomic propositions. We construct in linear time a BSRV specification φ over the set of input variables $X = AP$ such that $\mathcal{L}(\varphi) = \mathcal{L}(\psi)$. Let $SF(\psi)$ be the set of subformulas of ψ. Then, the set of output variables Y of φ is defined as follows.

$$Y = \bigcup_{\theta \in SF(\psi)} \{y_\theta\} \cup \{\mathsf{init}\}$$

Thus, we associate to each subformula θ of ψ, an output variable y_θ. The intended meaning is that for an input valuation σ_X (corresponding to a non-empty finite word over 2^{AP}) and a valuation model σ associated with σ_X, at each time step i, the value of variable y_θ is 1 iff θ holds at position i along σ_X. The equations for the output variables are defined as follows, where $p \in AP = X$.

$$\begin{array}{ll}
\mathsf{init} = \mathsf{first} \to (y_\psi \vee \neg\mathsf{init}) & y_p = p \\
y_{\neg\theta} = \neg\, y_\theta & y_{\theta_1 \vee \theta_2} = y_{\theta_1} \vee y_{\theta_2} \\
y_{\bigcirc\theta} = y_\theta[+1|0] & y_{\ominus\theta} = y_\theta[-1|0] \\
y_{\theta_1 \mathcal{U} \theta_2} = y_{\theta_2} \vee (\neg\mathsf{last} \wedge y_{\theta_1} \wedge y_{\theta_1 \mathcal{U} \theta_2}[+1|1]) & \\
y_{\theta_1 \mathcal{S} \theta_2} = y_{\theta_2} \vee (\neg\mathsf{first} \wedge y_{\theta_1} \wedge y_{\theta_1 \mathcal{S} \theta_2}[-1|1]) &
\end{array}$$

One can easily show that the construction is correct, i.e., $\mathcal{L}(\varphi) = \mathcal{L}(\psi)$. □

It is well-known that there is a singly exponential succinctness gap between LTL with past and NFA [16]. Consequently, we obtain the following result.

Theorem 3. *BSRV are singly exponentially more succinct than NFA, that is, there is a finite set X of input variables and a family $(\varphi_n)_{n \geq 1}$ of BSRV such that for all $n \geq 1$, φ_n has input variables in X and size polynomial in n, and every NFA accepting $\mathcal{L}(\varphi_n)$ has at least $2^{\Omega(n)}$ states.*

Effective Closure under Language Operations. An interesting feature of the class of BSRV is that, when interpreted as language recognizers, BSRV are effectively and *efficiently* closed under many language operations. For two languages $\mathcal{L}$ and $\mathcal{L}'$ of finite words, $\mathcal{L}^R$ denotes the reversal of $\mathcal{L}$, $\mathcal{L} \cdot \mathcal{L}'$ denotes the concatenation of $\mathcal{L}$ and $\mathcal{L}'$, and $\mathcal{L}^+$ denotes the positive Kleene closure of $\mathcal{L}$.

For a BSRV φ, we say that an output variable y of φ is *uniform* if for each valuation model of φ, the stream for y is uniform.

Theorem 4. *BSRV are effectively closed under the following language operations: intersection, union, reversal, positive Kleene closure, and concatenation. Additionally, the constructions for these operations can be done in* linear time.

Proof. We illustrate the constructions for the considered language operations.
Intersection, Union, and Reversal. The constructions are illustrated in Fig. 1. For the intersection, assuming w.l.o.g. that the BSRV φ and φ' have no output variable in common, the BSRV recognizing $\mathcal{L}(\varphi) \cap \mathcal{L}(\varphi')$ is simply the joint set of the equations of φ and φ'. For the union, we use two new output variables check and main. Intuitively, check is a uniform output variable used to guess whether the input has to be considered an input for φ or for φ'. The equation for check ensures that the streams for check range over all the uniform Boolean streams. Depending on the uniform value of check (if it is in 0^+ or 1^+), the equation for the output variable main ensures that the input is recognized iff either the equations of φ are fulfilled or the equations of φ' are fulfilled. For the reversal, the BSRV recognizing $\mathcal{L}(\varphi)^R$ is obtained from φ by replacing each subexpression $\tau[k|d]$ (resp., $\tau[-k|d]$) with $k > 0$ with the subexpression $\tau[-k|d]$ (resp., $\tau[k|d]$).
Positive Kleene closure. The construction is given in Fig. 2.

The BSRV recognizing $[\mathcal{L}(\varphi)]^+$ uses two new output variables: wbegin and wend. Intuitively, wbegin and wend are used for guessing a decomposition in the given input σ_X of the form $\sigma_X = \sigma_{X,1} \cdot \ldots \cdot \sigma_{X,N}$ for some $N \geq 1$ in such a way that each component $\sigma_{X,i}$ is in $\mathcal{L}(\varphi)$. In particular, the output variable wbegin (resp., wend) is used to mark the first (resp., the last) positions of the components $\sigma_{X,i}$. Moreover, the equations for the output variables of φ are modified to allow checking for an offset k of φ and a position j inside a component $\sigma_{X,i}$ in the guessed decomposition of the input σ_X, whether $k + j$ is still a position inside $\sigma_{X,i}$.

Concatenation. The construction is given in Fig. 3. We assume w.l.o.g. that the BSRV φ and φ' have no output variables in common. The BSRV recognizing $\mathcal{L}(\varphi) \cdot \mathcal{L}(\varphi')$ uses a new output variable: wmark. This variable is used for guessing

$$\varphi = \{y_1 = \mathsf{E}_1, \ldots, y_k = \mathsf{E}_k\} \quad \varphi' = \{y'_1 = \mathsf{E}'_1, \ldots, y'_h = \mathsf{E}'_h\}$$

Intersection: $\varphi \cap \varphi' = \{y_1 = \mathsf{E}_1, \ldots, y_k = \mathsf{E}_k, y'_1 = \mathsf{E}'_1, \ldots, y'_h = \mathsf{E}'_h\}$ where $\{y_1, \ldots, y_k\} \cap \{y'_1, \ldots, y'_h\} = \emptyset$.

Union: $\varphi \cup \varphi' = \{y_1 = y_1, \ldots, y'_h = y'_h, \mathsf{check} = \mathsf{E}_{\mathsf{check}}, \mathsf{main} = \mathsf{E}_{\mathsf{main}}\}$

$$\mathsf{E}_{\mathsf{check}} = \mathit{if}\ \neg\mathsf{last} \to (\mathsf{check} \leftrightarrow \mathsf{check}[+1|1])\ \mathit{then}\ \mathsf{check}\ \mathit{else}\ \neg\mathsf{check}$$

$$\mathsf{E}_{\mathsf{main}} = \mathit{if}\ \Big((\mathsf{check} \to \bigwedge_{i=1}^{i=k} y_i \leftrightarrow \mathsf{E}_i) \wedge (\neg\mathsf{check} \to \bigwedge_{i=1}^{i=h} y'_i \leftrightarrow \mathsf{E}'_i)\Big)\ \mathit{then}\ \mathsf{main}\ \mathit{else}\ \neg\mathsf{main}$$

Reversal: $\varphi^R = \{y_1 = \mathsf{E}_1^R, \ldots, y_k = \mathsf{E}_k^R\}$

E_i^R is obtained from E_i by converting each offset k in its opposite $-k$.

Fig. 1. Constructions for intersection, union, and reversal

Positive Kleene closure for $\varphi = \{y_1 = \mathsf{E}_1, \ldots, y_k = \mathsf{E}_k\}$

$$\varphi^+ = \{y_1 = \mathsf{E}_1^+, \ldots, y_k = \mathsf{E}_k^+, \mathsf{wbegin} = \mathsf{E}_{\mathsf{wbegin}}, \mathsf{wend} = \mathsf{E}_{\mathsf{wend}}\}$$

$$\mathsf{E}_{\mathsf{wbegin}} = \mathit{if}\ (\mathsf{first} \to \mathsf{wbegin}) \wedge (\mathsf{wbegin} \to \mathsf{wend}[-1|1])\ \mathit{then}\ \mathsf{wbegin}\ \mathit{else}\ \neg\mathsf{wbegin}$$
$$\mathsf{E}_{\mathsf{wend}} = \mathit{if}\ (\mathsf{last} \to \mathsf{wend}) \wedge (\mathsf{wend} \to \mathsf{wbegin}[+1|1])\ \mathit{then}\ \mathsf{wend}\ \mathit{else}\ \neg\mathsf{wend}$$

and E_i^+ is obtained from E_i by replacing each stream subexpression $\tau[k|d]$ with $\mathsf{E}_{\tau,k,d}$:

$$\mathsf{E}_{\tau,k,d} = \begin{cases} \mathit{if}\ \bigvee_{j=1}^{j=k} \mathsf{wbegin}[j|1]\ \mathit{then}\ d\ \mathit{else}\ \tau[k|d] & \text{if } k > 0 \\ \mathit{if}\ \bigvee_{j=1}^{j=-k} \mathsf{wend}[-j|1]\ \mathit{then}\ d\ \mathit{else}\ \tau[k|d] & \text{if } k < 0 \end{cases}$$

Fig. 2. Construction for positive Kleene closure

a decomposition in the given input of the form $\sigma_X \cdot \sigma'_X$ in such a way that $\sigma_X \in \mathcal{L}(\varphi)$ and $\sigma'_X \in \mathcal{L}(\varphi')$. In particular, the output variable wmark assumes the value 1 along all and only the positions of σ_X (the equation for wmark ensures that a Boolean stream for wmark is always in 1^+0^+). Moreover, the equations for the output variables of φ are modified in order to allow to check for a positive offset $k > 0$ of φ and a position j inside σ_X in the guessed decomposition $\sigma_X \cdot \sigma'_X$ of the input, whether $k+j$ is still a position inside σ_X. Analogously, the equations for the output variables of φ' are modified to allow checking for a negative offset $k < 0$ of φ' and a position j inside σ'_X in the guessed decomposition $\sigma_X \cdot \sigma'_X$ of the input, whether $k + j$ is still a position inside σ'_X. $\square$

4 Offline Monitoring for Well-Defined BSRV

In this section, we propose an offline monitoring algorithm for well-defined BSRV based on Theorem 1. The algorithm runs in time linear in the length of the input

$$\varphi = \{y_1 = \mathsf{E}_1, \ldots, y_k = \mathsf{E}_k\} \quad \varphi' = \{y'_1 = \mathsf{E}'_1, \ldots, y'_h = \mathsf{E}'_h\}$$

Concatenation: $\{y_1, \ldots, y_k\} \cap \{y'_1, \ldots, y'_h\} = \emptyset$

$$\varphi \cdot \varphi' = \{y_1 = \mathit{if}\ \mathsf{wmark}\ \mathit{then}\ \widetilde{\mathsf{E}}_1\ \mathit{else}\ y_1, \ldots, y_k = \mathit{if}\ \mathsf{wmark}\ \mathit{then}\ \widetilde{\mathsf{E}}_k\ \mathit{else}\ y_k,$$
$$y'_1 = \mathit{if}\ \neg\mathsf{wmark}\ \mathit{then}\ \widetilde{\mathsf{E}}'_1\ \mathit{else}\ y'_1, \ldots, y'_h = \mathit{if}\ \neg\mathsf{wmark}\ \mathit{then}\ \widetilde{\mathsf{E}}'_h\ \mathit{else}\ y'_h, \mathsf{wmark} = \mathsf{E}_{\mathsf{wmark}}\}$$

$$\mathsf{E}_{\mathsf{wmark}} = \mathit{if}\ (\mathsf{first} \to \mathsf{wmark}) \wedge (\mathsf{last} \to \neg\mathsf{wmark}) \wedge (\mathsf{wmark} \to \mathsf{wmark}[-1|1]) \wedge$$
$$(\neg\mathsf{wmark} \to \neg\mathsf{wmark}[+1|0])\ \mathit{then}\ \mathsf{wmark}\ \mathit{else}\ \neg\mathsf{wmark}$$

$\widetilde{\mathsf{E}}_i$ is obtained from E_i by replacing each stream subexpression $\tau[k|d]$ s.t. $k > 0$ with:

$$\mathit{if}\ \bigvee_{j=1}^{j=k} \neg\mathsf{wmark}[j|0]\ \mathit{then}\ d\ \mathit{else}\ \tau[k|d]$$

$\widetilde{\mathsf{E}}'_i$ is obtained from E'_i by replacing each stream subexpression $\tau[k|d]$ s.t. $k < 0$ with:

$$\mathit{if}\ \bigvee_{j=1}^{j=-k} \mathsf{wmark}[-j|1]\ \mathit{then}\ d\ \mathit{else}\ \tau[k|d]$$

Fig. 3. Construction for concatenation

```
Monitoring($\varphi, \sigma_X$)        /** $\varphi$ is a well-defined BSRV and $\mathcal{A}_\varphi = \langle Q, q_0, \delta, F\rangle$ **/
  $\Lambda \leftarrow \{q_0\}$
  for $i = 1$ upto $|\sigma_X|$ do
    update $\Lambda \leftarrow \{q \in Q \mid q \in \delta(p, \sigma_X(i))$ for some $p \in \Lambda\}$
    store $\Lambda$ at position $i$ on the tape
  for $i = |\sigma_X|$ downto 1 do
    let $\Lambda$ be the set of states stored at position $i$ on the tape
    if $i = |\sigma_X|$ then $p \leftarrow$ the unique accepting state in $\Lambda$
    else let $q$ be the unique state in $\Lambda$ such that $p \in \delta(q, \sigma_X(i+1))$; update $p \leftarrow q$
    output at position $i$ the main value of $p$
```

Fig. 4. Offline monitoring algorithm for well-defined BSRV

trace (input streams) and singly exponential in the size of the specification. Additionally, we partially solve a question left open in [8] for the case of BSRV.

Let φ be a BSRV over X and Y, and $\mathcal{A}_\varphi = \langle Q, q_0, \delta, F\rangle$ be the NFA over 2^X accepting $\mathcal{L}(\varphi)$ of Theorem 1. Recall that $Q \setminus \{q_0\}$ is contained in $(A_\perp)^{b(\varphi)} \times A \times (A_\perp)^{f(\varphi)}$, where $A = 2^{X \cup Y}$ and $A_\perp := A \cup \{\perp\}$, and an expanded valuation model of φ is of the form $\pi = q_0, q_1, \ldots, q_k$, where $q_i \in Q \setminus \{q_0\}$ for all $1 \leq i \leq k$. Moreover, the valuation model of φ encoded by π is the sequence of the main values of the states q_i visited by π. By Theorem 1, the set of accepting runs of $\mathcal{A}_\varphi$ over an input σ_X is the set of expanded valuation models of φ encoding the valuation models of φ associated with the input σ_X. Hence, the following holds.

Proposition 2. *A BSRV φ is well-defined if and only if the NFA $\mathcal{A}_\varphi$ is universal and unambiguous.*

The offline monitoring algorithm for well-defined BSRV is given in Fig. 4, where we assume that the input trace σ_X is available on a tape. The algorithm operates in two phases. In the first phase, a forward traversing of the input trace is performed, and the algorithm simulates the unique run over the input σ_X of the deterministic finite state automaton (DFA) that would result from $\mathcal{A}_\varphi$ by the classical powerset construction. Let $\{q_0\}, \Lambda(1), \ldots, \Lambda(|\sigma_X|)$ be the run of this DFA over σ_X. Then, at each step i, the state $\Lambda(i)$ of the run resulting from reading the input symbol $\sigma_X(i)$ is stored in the ith position of the tape. In the second phase, a backward traversing of the input trace is performed, and the algorithm outputs a stream valuation of φ. Since φ is well-defined, by using Proposition 2, we easily deduce that the uniqueness conditions in the second phase of the algorithm are satisfied. Moreover, the sequence of states computed by the algorithm in the second phase is the unique accepting run π of $\mathcal{A}_\varphi$ over σ_X. Therefore, the algorithm outputs the valuation model of φ encoded by π, which is the unique valuation model of φ associated with the input σ_X. Thus, since the size of the NFA $\mathcal{A}_\varphi$ is singly exponential in the size of φ, we obtain the following result.

Theorem 5. *One can construct an offline monitoring algorithm for well-defined BSRV running in time linear in the length of the input trace and singly exponential in the size of the specification. Additionally, the algorithm processes a position of the input trace exactly twice.*

In [8], a syntactical condition for general SRV, called *well-formedness*, is introduced, which can be checked in polynomial time and implies well-definedness. Well-formedness ensures the absence of circular definitions by requiring that a dependency graph of the output variables have not zero-weight cycles. As illustrated in [8], for the restricted class of well-formed SRV, it is possible to construct an offline monitoring algorithm which runs in time linear in the length of the input trace and the size of the specification. Moreover, one can associate to a well-formed SRV φ a parameter $ad(\varphi)$, called *alternation depth* [8], such that the monitoring algorithm processes each position of the input trace exactly $ad(\varphi)+1$ times. An important question left open in [8] is whether for a well-formed SRV φ, it is possible to construct a φ-equivalent SRV whose alternation depth is minimal. Here, we settle partially this question for the class of BSRV. By using the same ideas for constructing the algorithm of Fig. 4, we show that for the class of BSRV, the semantic notion of well-definedness coincides with the syntactical notion of well-formedness (modulo BSRV-equivalence), and the hierarchy of well-formed BSRV induced by the alternation depth collapses to the level 1. In particular, we establish the following result.

Theorem 6. *Given a well-defined BSRV φ, one can build in doubly exponential time a φ-equivalent BSRV which is well-formed and has alternation depth* 1.

5 Decision Problems

We investigate complexity issues for some relevant decision problems on BSRV. In particular, we establish that while checking well-definedness is in EXPTIME, checking for a given BSRV φ and a given subset Z of output variables, whether φ

is well-defined with respect to Z (*generalized well-definedness problem*) is instead EXPSPACE-complete. Our results can be summarized as follows.

Theorem 7. *For BSRV:*

1. *The under-definedness problem is PSPACE-complete, the well-definedness problem is in EXPTIME and at least PSPACE-hard, while the over-definedness problem and the* generalized *well-definedness problem are both EXPSPACE-complete.*
2. *Checking semantic equivalence is EXPSPACE-complete.*
3. *When interpreted as language recognizers, language emptiness is PSPACE-complete, while language universality, language inclusion, and language equivalence are EXPSPACE-complete.*

Here, we illustrate the upper bounds of Theorem 7(1). We need a preliminary result (Proposition 3). For an NFA $\mathcal{A} = \langle Q, q_0, \delta, F\rangle$, a *state projection of* $\mathcal{A}$ is a mapping $\Upsilon : Q \to P$ for some finite set P such that for all $q \in Q$, $\Upsilon(q)$ is computable in logarithmic space (in the size of Q). The mapping Υ can be extended to sequences of states in the obvious way. We say that the NFA $\mathcal{A}$ is *unambiguous with respect to* Υ if for all $w \in \mathcal{L}(\mathcal{A})$ and accepting runs π and π' of $\mathcal{A}$ over w, their projections $\Upsilon(\pi)$ and $\Upsilon(\pi')$ coincide.

Proposition 3. *Given an NFA* $\mathcal{A}$ *and a state projection* Υ *of* $\mathcal{A}$*, checking whether* $\mathcal{A}$ *is* not *unambiguous with respect to* Υ *can be done in NLOGSPACE.*

Upper Bounds of Theorem 7(1). Let φ be a BSRV over X and Y, and $\mathcal{A}_\varphi$ be the NFA of Theorem 1 accepting $\mathcal{L}(\varphi)$ and whose size is *singly exponential* in the size of φ.

Under-definedness: by Theorem 1 and Lemma 1, φ is under-defined iff $\mathcal{A}_\varphi$ is *not* unambiguous. Thus, since $\mathcal{A}_\varphi$ can be constructed on the fly and PSPACE = NPSPACE, by Proposition 3 (with Υ being the identity map), it follows that the under-definedness problem is in PSPACE.

Over-definedness: since $\mathcal{A}_\varphi$ accepts $\mathcal{L}(\varphi)$, φ is over-defined iff $\mathcal{A}_\varphi$ is not universal. Thus, since checking universality for NFA is a well-known PSPACE-complete problem [19], membership in EXPSPACE for checking over-definedness follows.

Well-definedness: it is well-known that checking universality of unambiguous NFA can be done in polynomial time [24]. By Proposition 2, φ is well-defined iff $\mathcal{A}_\varphi$ is universal and unambiguous. Thus, since checking that $\mathcal{A}_\varphi$ is unambiguous can be done in PSPACE (in the size of φ), membership in EXPTIME for checking well-definedness follows.

Generalized Well-definedness: let $Z \subseteq Y$. Recall that the set of non-initial states of $\mathcal{A}_\varphi$ is contained in $(A_\perp)^{b(\varphi)} \times A \times (A_\perp)^{f(\varphi)}$, where $A = 2^{X \cup Y}$ and $A_\perp := A \cup \{\perp\}$. Let Υ_Z be the state projection of $\mathcal{A}_\varphi$ assigning to the initial state q_0 of $\mathcal{A}_\varphi$ q_0 itself, and assigning to each non-initial state $(a_{-b(\varphi)}, \ldots, a_{-1}, a_0, a_1, \ldots, a_{f(\varphi)})$ of $\mathcal{A}_\varphi$ the tuple $(d_{-b(\varphi)}, \ldots, d_{-1}, d_0, d_1, \ldots, d_{f(\varphi)})$, where for all $b(\varphi) \leq i \leq f(\varphi)$, $d_i = a_i$ if $a_i = \perp$, and $d_i = a_i \cap Z$ otherwise. Now, let σ and σ' be two valuation models of φ associated with an input σ_X, and π and π' be the *expanded* valuation models encoding σ and σ', respectively. By construction, it follows that $\Upsilon_Z(\pi) = \Upsilon_Z(\pi')$ iff the restrictions of σ and σ' to Z coincide. By Theorem 1, we obtain that φ is well-defined with respect to Z iff $\mathcal{A}_\varphi$ is unambiguous with respect to Υ_Z and $\mathcal{A}_\varphi$ is universal. Thus, since checking universality for NFA is PSPACE-complete, by Proposition 3, membership in EXPSPACE for checking generalized well-definedness follows.

6 Conclusion

In this paper, we have studied some theoretical problems for the class of Boolean SRV. We have also presented an offline monitoring algorithm for well-defined BSRV that only requires two passes over the dumped trace. An open question is the exact complexity of checking well-definedness for BSRV: it lies somewhere between PSPACE and EXPTIME. Future work includes the theoretical investigation and the development of monitoring algorithms for SRV over richer data types, such as counters and stacks. In particular, the emerging field of symbolic automata and transducers [25]—that extend the classical notions from discrete alphabets to theories handled by solvers—seems very promising to study in the context of SRV, which in turn can extend automata from states and transitions to stream dependencies. The combination of these two extensions has the potential to provide a rich but *tractable* foundation for the runtime verification of values from rich types. Additionally, we are studying the extension to the monitoring of visibly pushdown systems, where SRV is extended to deal with traces containing calls and returns.

Finally, we plan to study the monitorability of well-definedness of specifications. If one cannot determine well-definedness statically, a plausible alternative would be to use a monitor that *assumes* well-definednees in tandem with a monitor that *detects* non-well-definedness (and hence, the incorrectness of the first monitor).

References

1. Barringer, H., Goldberg, A., Havelund, K., Sen, K.: Rule-based runtime verification. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 44–57. Springer, Heidelberg (2004)
2. Basin, D., Harvan, M., Klaedtke, F., Zălinescu, E.: MONPOLY: Monitoring usage-control policies. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 360–364. Springer, Heidelberg (2012)
3. Basin, D., Klaedtke, F., Müller, S.: Policy monitoring in first-order temporal logic. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 1–18. Springer, Heidelberg (2010)
4. Bauer, A., Goré, R., Tiu, A.: A first-order policy language for history-based transaction monitoring. In: Leucker, M., Morgan, C. (eds.) ICTAC 2009. LNCS, vol. 5684, pp. 96–111. Springer, Heidelberg (2009)
5. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology 20(4), 14 (2011)

6. Berry, G.: The foundations of Esterel. In: Proof, Language, and Interaction: Essays in Honour of Robin Milner, pp. 425–454. MIT Press (2000)
7. Caspi, P., Pouzet, M.: Synchronous Kahn Networks. In: Proc. of ICFP 1996, pp. 226–238. ACM Press (1996)
8. D'Angelo, B., Sankaranarayanan, S., Sánchez, C., Robinson, W., Finkbeiner, B., Sipma, H.B., Mehrotra, S., Manna, Z.: LOLA: Runtime monitoring of synchronous systems. In: Proc. of TIME 2005, pp. 166–174. IEEE CS Press (2005)
9. Eisner, C., Fisman, D., Havlicek, J., Lustig, Y., McIsaac, A., Van Campenhout, D.: Reasoning with temporal logic on truncated paths. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 27–39. Springer, Heidelberg (2003)
10. Finkbeiner, B., Sankaranarayanan, S., Sipma, H.B.: Collecting statistics over runtime executions. ENTCS 70(4), 36–54 (2002)
11. Gautier, T., Le Guernic, P., Besnard, L.: SIGNAL: A declarative language for synchronous programming of real-time systems. In: Kahn, G. (ed.) FPCA 1987. LNCS, vol. 274, pp. 257–277. Springer, Heidelberg (1987)
12. Goodloe, A.E., Pike, L.: Monitoring distributed real-time systems: A survey and future directions. Technical report, NASA Langley Research Center (2010)
13. Halbwachs, N., Caspi, P., Pilaud, D., Plaice, J.: Lustre: a declarative language for programming synchronous systems. In: Proc. of POPL 1987, pp. 178–188. ACM Press (1987)
14. Havelund, K., Goldberg, A.: Verify your runs. In: Meyer, B., Woodcock, J. (eds.) Verified Software. LNCS, vol. 4171, pp. 374–383. Springer, Heidelberg (2008)
15. Havelund, K., Roşu, G.: Synthesizing monitors for safety properties. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 342–356. Springer, Heidelberg (2002)
16. Laroussinie, F., Markey, N., Schnoebelen, P.: Temporal logic with forgettable past. In: Proc. of LICS 2002, pp. 383–392. IEEE CS Press (2002)
17. Leucker, M., Schallhart, C.: A brief account of runtime verification. The Journal of Logic and Algebraic Programming 78(5), 293–303 (2009)
18. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety. Springer, New York (1995)
19. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: Proc. of FOCS 1972, pp. 125–129. IEEE CS Press (1972)
20. Pike, L., Goodloe, A., Morisset, R., Niller, S.: Copilot: A hard real-time runtime monitor. In: Barringer, H., et al. (eds.) RV 2010. LNCS, vol. 6418, pp. 345–359. Springer, Heidelberg (2010)
21. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 573–586. Springer, Heidelberg (2006)
22. Roşu, G., Havelund, K.: Rewriting-based techniques for runtime verification. Autom. Softw. Eng. 12(2), 151–197 (2005)
23. Sen, K., Roşu, G.: Generating optimal monitors for extended regular expressions. ENTCS 89(2), 226–245 (2003)
24. Stearns, R.E., Hunt, H.B.: On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. SIAM J. Comput. 14(3), 598–611 (1985)
25. Veanes, M., Hooimeijer, P., Livshits, B., Molnar, D., Bjrner, N.: Symbolic finite state transducers: algorithms and applications. In: Proc. of POPL 2012, pp. 137–150. ACM (2012)