Foundations of Boolean Stream Runtime Verification☆

Laura Bozzelli[a], César Sánchez[b]

[a] *Technical University of Madrid (UPM), Madrid, Spain*
[b] *IMDEA Software Institute, Madrid, Spain*

Abstract

Stream runtime verification (SRV), pioneered by the tool LOLA, is a declarative formalism 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 the evaluation algorithms, and 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.

Keywords: Complexity and expressiveness, Succinctness, Efficiency in closure operations, Offline monitoring

1. Introduction

Runtime verification (RV) has emerged in the last decades as an applied formal technique for software reliability. In RV, a specification expresses correctness requirements and is automatically translated into a *monitor*. Such a monitor is then used

☆This paper is an extended version of the results published in the Proceedings of the 5th International Conference on Runtime Verification (RV'14), LNCS 8734, pages 64–79, Springer, 2014.

Email addresses: `laura.bozzelli@fi.upm.es` (Laura Bozzelli), `cesar.sanchez@imdea.org` (César Sánchez)

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 violations of the specification when the system is in operation, for example, by the execution of additional repair code. On the other hand, offline monitoring is used in post-mortem analysis and it is convenient for testing large systems before deployment, or to inspect system logs. Unlike static verification techniques like 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 [1, 2] for modern surveys on runtime verification.

Stream runtime verification and related work. The first specification formalisms proposed for runtime verification were based on specification languages for static verification, typically LTL [3] or past LTL adapted for finite paths [4, 5, 6].

Other formalisms for expressing monitors include regular expressions [7], rule based specifications as proposed in the system Eagle [8], or rewriting [9]. Stream runtime verification (SRV), first proposed in the tool LOLA [10], 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 [11]. More modernly, the semantics of some temporal logics for continuous signals, like STL (see e.g. [12]) are defined in terms of the relation between the signals defined for an STL formula and the signals assumed for the subexpressions, in a similar manner as for SRV.

Stream runtime verification offers two advantages for 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 in these evaluation algorithms—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 adapted 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 or automata [13]. 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 [14, 15, 16]. 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 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 [17]—like Esterel [18], Lustre [19] or Signal [20]—but specifically designed for observing traces of systems, by 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 [21, 22] in the system Copilot, developed by Galois and NASA.

When used for synthesizing monitors, SRV specifications need to be *well-defined*: for every input there must be a unique corresponding output stream. However, as with many synchronous languages, the declarative style of SRV permits to write syntactically correct 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. In [10], a syntactical constraint called *well-formedness* was introduced in order to ensure the absence of circular dependencies, and guarantee well-definedness. Natural specifications, written by engineers or translated from specifications in temporal logic and similar formalisms, are usually well-formed and hence well-defined. However, many practical questions that specification engineers ask—like whether a specification is consistent, universal or satisfiable, or whether two specifications are equivalent—can be reduced to the decision problems that we study in this paper, including checking well-definedness.

Symbolic transducers [23, 24] have been recently introduced as an extension to finite state automata and transducers that annotate transitions with logical formulae on the input values, to model sets of concrete transitions. Like SRV, symbolic transducers allow to model complex data in the input and to produce complex data as output. The main difference is that symbolic transducers do not allow to relate or compare inputs produced at different instants, as the only information that the transducer is allowed to store is its finite state. On the other hand, SRV allows to manipulate and store intermediate streams and relate stream values at different instants. The price to pay is, of course, undecidability when manipulating complex data. Extending symbolic transducers with the ability to relate inputs at different positions leads to undecidable decision problems [24]. Similarly, decision problems for general SRV are also undecidable when one allows rich enough data to be manipulated and stored in the streams. In this paper, we limit the data to Boolean values, but (unlike symbolic transducers) we allow to relate streams at different instants, and study the complexity of the corresponding problems.

Another related line of work introduced recently is regular string transformations [25] and the language DReX, which allows to define in a controlled manner a subset of string transformations. SRV (on characters as input data) can also be used to define string transformations, but the synchronous model of computation of SRV restricts the output to have the same length as the input. However, general SRV is not restricted to characters and allows to define richer (synchronous) specifications. Moreover, the evaluation algorithms of DReX require space depending on the size of the input string, while for SRV it is known how to schedule the offline evaluation of a given specification using an amount of memory that is independent on the size of the input, and is only constant on the size of the spec. More generally, extending stream runtime verification to define string transformations, and in particular non-synchronous string transformations, is an unexplored area of research.

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. 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 can be reduced to the decision problems above. For example, our results provide algorithms to check whether two specifications are equivalent, or whether a part of a specification is redundant because it is subsumed by another part of the specification.

- BSRV specifications can be naturally interpreted as language recognizers, where one defines a language by selecting 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 [10].

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 for well-defined BSRV specifications. Section 5 is devoted to the decision problems for BSRV specifications. Finally, Section 6 concludes.

2. Stream Runtime Verification (SRV)

In this Section, we recall the SRV framework [10]. We focus on SRV specifications over stream variables of the same type (with emphasis on the Boolean type).

Informally, an SRV specification describes a relation between input and output streams, where a stream is a finite sequence of values from a given type T (the i^{th} value in the sequence represents the value of the stream at time step i). In the specification, each stream is referred by a variable (denoting the value of the stream at the current time step), and the relation between input and output streams is described by making use of stream expressions, whose building blocks are:

- variables and constants;
- function application;
- offset expressions for referring to the value of a stream at a future/past time with a specified offset from the current time.

Now, we proceed with the formal definition of the SRV specification language.

For all natural numbers i and j with $i \leq j$, we denote by $[i, j]$ the set of natural numbers h such that $i \leq h \leq j$.

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 $\wedge$, $\vee$ 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. For all positions $1 \leq i \leq |w|$, $w(i)$ is the i^{th} 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 $\mathsf{z} \in Z$, a stream $\sigma(\mathsf{z})$ of type T such that 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 i^{th} symbol, written $\sigma(i)$, is the set of variables $\mathsf{z} \in Z$ such that $\sigma(\mathsf{z})(i) = 1$.

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, $\tau[\ell|c]$ is an offset expression which refers to the value of τ offset ℓ positions from the current position, and the constant c is the *default* value of type T, which is assigned to positions from which the offset falls 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 $1 \leq i \leq |\sigma|$:

- **Constants:** $[\![c, \sigma]\!](i) = c$
- **Variables:** $[\![\mathsf{z}, \sigma]\!](i) = \sigma(\mathsf{z})(i)$ for all $\mathsf{z} \in Z$
- **Offsets:** $[\![\tau[\ell|c], \sigma]\!](i) = \begin{cases} [\![\tau, \sigma]\!](i+\ell) & \text{if } 1 \leq i+\ell \leq |\sigma| \\ c & \text{otherwise} \end{cases}$
- **Expressions:** $[\![f(\mathsf{E}_1, \ldots, \mathsf{E}_k), \sigma]\!](i) = f([\![\mathsf{E}_1, \sigma]\!](i), \ldots, [\![\mathsf{E}_k, \sigma]\!](i))$

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* E_1 *else* E_2 stands for $(\mathsf{E} \wedge \mathsf{E}_1) \vee (\neg\mathsf{E} \wedge \mathsf{E}_2)$. Additionally, we use first for the Boolean stream expressions $0[-1|1]$ and we use last for $0[+1|1]$. Note that for a Boolean stream, first is 1 precisely at the first position (and 0 elsewhere), and last is 1 precisely at the last position (and 0 elsewhere).

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

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

Consider a Boolean stream valuation σ over $\{\mathsf{x}\}$ such that $\sigma(\mathsf{x}) \in (01)^+$. 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 $Y = \{\mathsf{y}_1, \ldots, \mathsf{y}_n\}$ of *output* variables with $X \cap Y = \emptyset$, an SRV specification φ of type T over X and Y is a set of *defining* equations

$$\varphi : \{\mathsf{y}_1 := \mathsf{E}_1, \ldots, \mathsf{y}_n := \mathsf{E}_n\}$$

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, and that each defining equation can use both input and output variables. The intended meaning of a defining equation is to provide the value of the corresponding output stream at every given position. Before formally defining the semantics of SRV, we give some examples as motivation.

Example 2. Consider the specification "*every request must be eventually followed by a grant before the trace ends.*" This specification can be expressed by the following SRV specification with Boolean input streams request and grant, output streams reqgrant and evgrant, and the following defining equations:

$$\begin{array}{lcl} \mathsf{reqgrant} & := & \mathit{if}\ \mathsf{request}\ \mathit{then}\ \mathsf{evgrant}\ \mathit{else}\ 0 \\ \mathsf{evgrant} & := & \mathsf{grant} \vee \mathsf{evgrant}[+1|0] \end{array}$$

Essentially, the stream reqgrant holds at a given position whenever either there is not a request, or there is a grant in a future position, as stated by the stream evgrant. This specification corresponds to the LTL expression $\square(\mathsf{request} \rightarrow \Diamond\mathsf{grant})$. For a given input trace, the stream reqgrant is true exactly at those positions where the LTL formula holds. □

Example 3. The specification in the previous example contains a single positive offset. It is possible, as shown in [10], to write an equivalent specification that only uses negative offsets:

$$\begin{array}{lcl} \mathsf{waitgrant} & := & \neg\mathsf{grant} \wedge \big(\mathsf{request} \vee \mathsf{waitgrant}[-1|0]\big) \\ \mathsf{ok} & := & \mathsf{last} \rightarrow \neg\mathsf{waitgrant} \end{array}$$

The output stream waitgrant captures whether no grant has been produced since the last request. By evaluating waitgrant at the end of the trace, one can obtain whether there is a pending request. □

Example 4. Let us revisit the specification from the previous examples. One criticism is that a single grant suffices to match several previous requests. A refined version of the intended specification states that "*every request has a response, and every response happens to satisfy a request*". One way to express this specification is by using integers as a type for intermediate streams countreq and countgrant, with defining equations:

$$\begin{array}{lcl} \mathsf{countreq} & := & \mathsf{countreq}[-1|0] + \big(\mathit{if}\ \mathsf{request}\ \mathit{then}\ 1\ \mathit{else}\ 0\big) \\ \mathsf{countgrant} & := & \mathsf{countgrant}[-1|0] + \big(\mathit{if}\ \mathsf{grant}\ \mathit{then}\ 1\ \mathit{else}\ 0\big) \\ \mathsf{ok} & := & (\mathsf{countgrant} \leq \mathsf{countreq}) \wedge (\mathsf{last} \rightarrow (\mathsf{countreq} - \mathsf{countgrant}) = 0) \end{array}$$

Essentially, the stream countreq with type integer counts the number of requests that have been seen in the past, and similarly countgrant counts the number of grants. The Boolean stream ok states that at any point, the number of grants cannot be higher than the number of requests and at the end of the trace requests and grants must match. □

Example 5. The previous specification can be refined to store the pending requests, that are removed when responses with the same identifier are received. Let reqid and $\mathsf{grantid}$ be input streams of identifiers for actual requests and grants. We define an output stream $\mathsf{pending}$ whose type is the range of finite sets of identifiers, and a Boolean output stream ok with the following defining equations:

$$\begin{array}{lcll} \mathsf{pending} & := & \mathsf{pending}[-1|\emptyset] & \cup\big(\mathit{if}\ \mathsf{request}\ \mathit{then}\ \{\mathsf{reqid}\}\ \mathit{else}\ \emptyset\big) \\ & & & \setminus\big(\mathit{if}\ \mathsf{grant}\ \mathit{then}\ \{\mathsf{grantid}\}\ \mathit{else}\ \emptyset\big) \\ \mathsf{ok} & := & \mathsf{last} \rightarrow (\mathsf{pending} = \emptyset) & \end{array}$$

Essentially, the stream $\mathsf{pending}$ stores at each position those requests that have not been granted yet. Hence, one requires that, at the end of the trace, the set of pending requests is empty. In this manner, this specification computes the set of pending requests. Additionally, one can ask that grants only arrive on pending requests using the following output stream $\mathsf{goodgrant}$:

$$\begin{array}{lcl} \mathsf{goodgrant} & := & (\neg\mathsf{grant}) \vee \begin{pmatrix} \mathsf{grantid} = \mathsf{reqid} \\ \wedge \\ \mathsf{grantid} \notin \mathsf{pending} \end{pmatrix} \vee \begin{pmatrix} \mathsf{grantid} \neq \mathsf{reqid} \\ \wedge \\ \mathsf{grantid} \in \mathsf{pending} \end{pmatrix} \\ \mathsf{ok}_2 & := & \mathsf{ok} \wedge \mathsf{goodgrant} \end{array}$$

The stream $\mathsf{goodgrant}$ guarantees that every grant satisfies a unique request, by checking that the grant satisfies either an immediate request or a past request (but not both). □

Example 6. (From [26]) We consider a simple latch, as described in [27] with a single Boolean input and a single Boolean output. Whenever the input is true the output is reversed with respect to the previous state. This can be accomplished with the following specification with input x, output y:

$$\mathsf{y} \ := \ \mathit{if}\ \mathsf{x}\ \mathit{then}\ \neg\mathsf{y}[-1|0]\ \mathit{else}\ \mathsf{y}[-1|0]$$

□

Example 7. (Resettable counter, From [26]) Consider a resettable counter with two Boolean inputs, inc and reset. The input inc increments the counter and the input reset resets the counter. The counter is modeled by a stream cnt of type integer that is initially set to zero. The defining equation for cnt is:

$$\mathsf{cnt} \ := \ \mathit{if}\ \mathsf{reset}\ \mathit{then}\ 0\ \mathit{else}\ \big(\mathsf{cnt}[-1|0] + \mathit{if}\ \mathsf{inc}\ \mathit{then}\ 1\ \mathit{else}\ 0\big)$$

□

Semantics of SRV. A stream valuation of a specification $\varphi : \{\mathsf{y}_1 = \mathsf{E}_1, \dots, \mathsf{y}_n = \mathsf{E}_\mathsf{n}\}$ is a stream valuation of type T over $X \cup Y$, while an *input* of φ is a stream valuation of type T over X and an *output* of φ is a stream valuation of type T over 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 specification 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 $\mathsf{y}_j = \mathsf{E}_\mathsf{j}$ of φ,

$$[\![\mathsf{y}_j, \sigma]\!] = [\![\mathsf{E}_\mathsf{j}, \sigma]\!] \qquad \text{where } \sigma = \sigma_X \cup \sigma_Y$$

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 specification φ:

- *Under-definedness*: for some input σ_X, there are at least two distinct valuation models of φ associated with σ_X. In this case we also say that φ is under-defined for σ_X.
- *Over-definedness*: for some input σ_X, there is no valuation model of φ associated with σ_X. In this case we also say that φ is over-defined for 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 specification φ may be both under-defined and over-defined (for different inputs), 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 specification φ is *well-defined with respect to* Z if for each input σ_X, there is exactly one stream valuation σ_Z over Z of 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 specifications 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 specification 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. For the complexity analysis, 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 absolute value of offset ℓ if E is a stream expression of the form $\tau[\ell|c]$, and we let $\|\mathsf{E}\|$ be defined as 1 otherwise. 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. Essentially the size of an expression φ is the size of its encoding when offsets are written in unary.

Example 8. Consider the following BSRV specifications over $X = \{\mathsf{x}\}$ and $Y = \{\mathsf{y}\}$:

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

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. □

Example 9. Consider a scenario where we have two input variables start and end and an observable output variable y. We intend that for every input σ_X, where $X = \{\mathsf{start}, \mathsf{end}\}$, the output Boolean stream σ_{y} for y gets the value 1 exactly at those positions i such that i belongs to a *session* of the input, that is an interval of positions $I = [h, k]$ satisfying the following:

- within I, the Boolean stream for start gets the value 1 exactly at position h. That is, $\sigma_X(\mathsf{start})(h) = 1$ and $\sigma_X(\mathsf{start})(\ell) = 0$ for all $h < \ell \leq k$;
- within I, the Boolean stream for end gets the value 1 exactly at position k. That is, $\sigma_X(\mathsf{end})(k) = 1$ and $\sigma_X(\mathsf{end})(\ell) = 0$ for all $h \leq \ell < k$.

The above requirement is accomplished by a BSRV specification φ which uses two additional intermediate output variables $\mathsf{havestarted}$ and $\mathsf{willend}$, whose associated equations are:

$$\begin{array}{lcl} \mathsf{havestarted} & := & (\neg\mathsf{start} \wedge \neg\mathsf{end}) \wedge \big(\mathsf{havestarted}[-1|0] \vee (\mathsf{start}[-1|0] \wedge \neg\mathsf{end}[-1|1])\big) \\ \mathsf{willend} & := & (\neg\mathsf{start} \wedge \neg\mathsf{end}) \wedge \big(\mathsf{willend}[+1|0] \vee (\mathsf{end}[+1|0] \wedge \neg\mathsf{start}[+1|1])\big) \end{array}$$

The Boolean stream for the output variable $\mathsf{havestarted}$ assumes the value 1 exactly at the *inner* positions i (i.e., positions which are neither *start* nor *end* positions) such that the greatest *start* position j which precedes i exists, and j is also a *non-end* position. Analogously, the Boolean stream for the output variable $\mathsf{willend}$ assumes the value 1 exactly at the *inner* positions i such that the smallest *end* position j which follows i exists, and j is also a *non-start* position. Finally, the equation for the output variable y (capturing the positions inside the sessions of the input) is defined as follows:

$$\mathsf{y} := (\mathsf{start} \wedge \mathsf{end}) \vee (\mathsf{start} \wedge \mathsf{willend}[+1|0]) \vee (\mathsf{end} \wedge \mathsf{havestarted}[-1|0]) \vee (\mathsf{willend} \wedge \mathsf{havestarted})$$

$\square$

3. BSRV as Language Recognizers

BSRV can be interpreted as a simple declarative formalism to specify languages of non-empty finite words. We associate to a BSRV specification 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. Formally,

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

Example 10. Let $X = \{\mathsf{x}\}$, $Y = \{\mathsf{y}\}$, and $\varphi : \{\mathsf{y} := \textit{if } \mathsf{E} \textit{ then } \mathsf{y} \textit{ else } \neg\mathsf{y}\}$, where

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

A pair (σ_X, σ_Y) is a valuation model of φ precisely whenever the valuation of the stream expression E w.r.t. $\sigma_X \cup \sigma_Y$ is in 1^+. This happens when $\sigma_X(x)(i) = 1$ for all odd positions i. Hence, $\mathcal{L}(\varphi)$ is the set of Boolean streams which assume the value 1 at the odd positions. Note that this is the finite version of the language that Wolper used to exhibit the limitation in expressive power of LTL [28]. $\square$

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. The NFA $\mathcal{A}$ is *deterministic* if for all $(q, \iota) \in Q \times I$, $\delta(q, \iota)$ is either empty or a singleton. 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.

Let us fix a BSRV specification 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* $f(\varphi)$, which are defined as follows:

$$\begin{aligned} b(\varphi) = &\quad max(0, \{\ell \mid \ell > 0 \text{ and } \varphi \text{ contains a subexpression of the form } \mathsf{z}[-\ell, c]\}) \\ f(\varphi) = &\quad max(0, \{\ell \mid \ell > 0 \text{ and } \varphi \text{ contains a subexpression of the form } \mathsf{z}[\ell, c]\}) \end{aligned}$$

The meaning of $b(\varphi)$ and $f(\varphi)$ is to capture 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 within $i - b(\varphi) \leq j \leq i + f(\varphi)$. 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 such that $\perp \notin A$. 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. Given 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, which intuitively represents the value of some stream valuation σ at some time step. The prefix $a_{-b(\varphi)}, \ldots, a_{-1}$ represent the values of σ at the previous $b(\varphi)$ time steps. The suffix $a_1, \ldots, a_{f(\varphi)}$ denotes the values of σ in the following $f(\varphi)$ steps. The symbol $\perp$ is used to denote the absence of a previous or future 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 precisely when $\tau \in a$ (that is, when the variable τ is present 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:

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

We denote by Q_φ the subset of P_φ consisting of the elements p of P_φ such that for each equation $\mathsf{y} = \mathsf{E}$ 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 $(\bot, \ldots, \bot, a_0, a_1, \ldots, a_{f(\varphi)})$;
- $w(|w|)$ is of the form $(a_{-b(\varphi)}, \ldots, a_{-1}, a_0, \bot, \ldots, \bot)$;
- 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 $d \in A_\bot$ 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 i-th element is the main value of the i-th element of w.

Example 11. Consider the BSRV φ over $X = \{\mathsf{x}\}$ and $Y = \{\mathsf{y}\}$ of Example 10. We have that $b(\varphi) = 0$ and $f(\varphi) = 1$. The following word

$$\#, \left(\boxed{\begin{matrix} \mathsf{x}:1 \\ \mathsf{y}:1 \end{matrix}}, \begin{matrix} \mathsf{x}:0 \\ \mathsf{y}:0 \end{matrix} \right), \left(\boxed{\begin{matrix} \mathsf{x}:0 \\ \mathsf{y}:0 \end{matrix}}, \begin{matrix} \mathsf{x}:1 \\ \mathsf{y}:1 \end{matrix} \right), \left(\boxed{\begin{matrix} \mathsf{x}:1 \\ \mathsf{y}:1 \end{matrix}}, \begin{matrix} \mathsf{x}:1 \\ \mathsf{y}:0 \end{matrix} \right), \left(\boxed{\begin{matrix} \mathsf{x}:1 \\ \mathsf{y}:0 \end{matrix}}, \bot \right)$$

(where at each position the boxed element represents the central value) is an expanded valuation model of φ, whose associated stream valuation is given by

$$\left(\begin{matrix} \mathsf{x}:1 \\ \mathsf{y}:1 \end{matrix} \right), \left(\begin{matrix} \mathsf{x}:0 \\ \mathsf{y}:0 \end{matrix} \right), \left(\begin{matrix} \mathsf{x}:1 \\ \mathsf{y}:1 \end{matrix} \right), \left(\begin{matrix} \mathsf{x}:1 \\ \mathsf{y}:0 \end{matrix} \right)$$

□

By construction, we easily obtain that for an expanded valuation model $\# \cdot w$ of φ, $\sigma(w)$ is a valuation model of φ. More precisely, the following lemma holds.

Lemma 1. The map that assigns 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. Recall that Q_φ is the subset P_φ consisting of consistent valuations (all output valuations receive the same value as their defining equations), which constrain the set of values of future and past elements of the tuples.

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 $(a_{-b(\varphi)}, \ldots, a_{-1}, a_0, \perp, \ldots, \perp)$, 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 $(\perp, \ldots, \perp, a_0, a_1, \ldots, a_{f(\varphi)})$ 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 $d \in A_\perp$ whose main value a_1 satisfies $a_1 \cap X = \iota$.

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. □

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 specification $\varphi_\mathcal{A}$ over the set of input variables X as follows. First, for each input symbol ι, we introduce a Boolean expression E_ι over X, encoding the input symbol ι, defined as $\mathsf{E}_\iota := (\bigwedge_{\mathsf{x} \in \iota} \mathsf{x}) \wedge (\bigwedge_{\mathsf{x} \in X \setminus \iota} \neg \mathsf{x})$. The set Y of output variables of $\varphi_\mathcal{A}$ is defined as follows:

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

We associate to each state $q \in Q$, an output variable q, whose defining 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 $\mathsf{E_{ev}}$ captures precisely the accepting runs of the NFA $\mathcal{A}$ and is defined as follows:

$$\mathsf{E_{ev}} = \underbrace{\bigvee_{q\in Q}(\mathsf{q} \wedge \bigwedge_{p\in Q\setminus\{q\}} \neg\mathsf{p})}_{\text{at each step, } \mathcal{A} \text{ is exactly in one state}} \quad\wedge\quad \underbrace{(\mathsf{first} \longrightarrow \mathsf{q_0})}_{\text{a run of } \mathcal{A} \text{ starts at the initial state}} \quad\wedge$$

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

By construction, 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 $\mathsf{E_{ev}}$ with respect to σ is a uniform stream in 1^+. In turn, the valuation of $\mathsf{E_{ev}}$ 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. □

As a corollary of Theorems 1 and 2, we obtain the following result.

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

Succinctness. It turns out that the singly exponential blow-up in Theorem 1 cannot be avoided. To prove this succinctness result we first show a linear time translation from 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:

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

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:

$$\begin{array}{lll}
(w,i)\models p & \Leftrightarrow & p\in w[i]\\
(w,i)\models \neg\psi & \Leftrightarrow & (w,i)\not\models \psi\\
(w,i)\models \psi_1\vee\psi_2 & \Leftrightarrow & (w,i)\models \psi_1 \text{ or } (w,i)\models \psi_2\\
(w,i)\models \bigcirc\psi & \Leftrightarrow & i+1\leq|w| \text{ and } (w,i+1)\models\psi\\
(w,i)\models \ominus\psi & \Leftrightarrow & i>1 \text{ and } (w,i-1)\models\psi\\
(w,i)\models \psi_1\,\mathcal{U}\,\psi_2 & \Leftrightarrow & \text{for some } j \text{ with } i\leq j\leq|w|,\ (w,j)\models\psi_2 \text{ and}\\
 & & \quad\text{for all } h \text{ with } i\leq h<j,\ (w,h)\models\psi_1\\
(w,i)\models \psi_1\,\mathcal{S}\,\psi_2 & \Leftrightarrow & \text{for some } j \text{ with } 1\leq j\leq i,\ (w,j)\models\psi_2 \text{ and}\\
 & & \quad\text{for all } h \text{ with } j<h\leq i,\ (w,h)\models\psi_1
\end{array}$$

The language $\mathcal{L}(\psi)$ of an 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. We introduce the following set of output variables Y in φ:

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

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

$$\begin{aligned}
\mathsf{init} &:= \mathsf{first} \rightarrow (\mathsf{y}_\psi \vee \neg \mathsf{init}) \\
\mathsf{y}_p &:= p \\
\mathsf{y}_{\neg\theta} &:= \neg\, \mathsf{y}_\theta \\
\mathsf{y}_{\theta_1 \vee \theta_2} &:= \mathsf{y}_{\theta_1} \vee \mathsf{y}_{\theta_2} \\
\mathsf{y}_{\bigcirc\theta} &:= \mathsf{y}_\theta[+1|0] \\
\mathsf{y}_{\ominus\theta} &:= \mathsf{y}_\theta[-1|0] \\
\mathsf{y}_{\theta_1 \mathcal{U} \theta_2} &:= \mathsf{y}_{\theta_2} \vee (\neg \mathsf{last} \wedge \mathsf{y}_{\theta_1} \wedge \mathsf{y}_{\theta_1 \mathcal{U} \theta_2}[+1|1]) \\
\mathsf{y}_{\theta_1 \mathcal{S} \theta_2} &:= \mathsf{y}_{\theta_2} \vee (\neg \mathsf{first} \wedge \mathsf{y}_{\theta_1} \wedge \mathsf{y}_{\theta_1 \mathcal{S} \theta_2}[-1|1])
\end{aligned}$$

We show now that the construction is correct by showing $\mathcal{L}(\varphi) = \mathcal{L}(\psi)$. For the inclusion $\mathcal{L}(\varphi) \subseteq \mathcal{L}(\psi)$, let $\sigma_X \in \mathcal{L}(\varphi)$. Hence, there is a valuation model σ of φ associated with the input σ_X. We need to show that $(\sigma_X, 1) \models \psi$. One can easily show by construction and structural induction that for all $\theta \in SF(\psi)$ and positions i along σ_X, $(\sigma_X, i) \models \theta$ if and only if $\sigma(\mathsf{y}_\theta)(i) = 1$. Moreover, the equation for the output variable init ensures that $\sigma(\mathsf{y}_\psi)(1) = 1$. Hence, $(\sigma_X, 1) \models \psi$. Consequently, $\mathcal{L}(\varphi) \subseteq \mathcal{L}(\psi)$.

For the converse inclusion $\mathcal{L}(\psi) \subseteq \mathcal{L}(\varphi)$, let $\sigma_X \in \mathcal{L}(\psi)$ and consequently $(\sigma_X, 1) \models \psi$. We define a stream valuation σ associated with the input σ_X as follows: $\sigma(\mathsf{init}) = 1^{|\sigma|}$ and for all $\theta \in SF(\psi)$ and positions i along σ_X, $\sigma(\mathsf{y}_\theta)(i) = 1$ if $(\sigma_X, i) \models \theta$, and $\sigma(\mathsf{y}_\theta)(i) = 0$ otherwise. Since $(\sigma_X, 1) \models \psi$, by construction, it easily follows that σ is a valuation model of φ associated with the input σ_X. Hence, $\sigma_X \in \mathcal{L}(\varphi)$ and consequently $\mathcal{L}(\psi) \subseteq \mathcal{L}(\varphi)$. $\square$

It is well-known [29] that there is a singly exponential succinctness gap between LTL with past and NFA. Consequently, by Proposition 1, we obtain the following result.

Theorem 3. BSRV specification are singly exponentially more succinct than NFA. In particular, there is a finite set X of input variables and a family $(\varphi_n)_{n\geq 1}$ of BSRV specifications with input variables X such that for all $n \geq 1$, φ_n has 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 performed in *linear time*.

Proof. Let $\varphi : \{\mathsf{y}_1 := \mathsf{E}_1, \ldots, \mathsf{y}_k := \mathsf{E}_k\}$ and $\varphi' : \{\mathsf{y}'_1 := \mathsf{E}'_1, \ldots, \mathsf{y}'_h := \mathsf{E}'_h\}$ be two BSRV specifications over the same set X of input variables. Let $Y = \{\mathsf{y}_1, \ldots, \mathsf{y}_k\}$ be the set of output variables of φ and $Y' = \{\mathsf{y}'_1, \ldots, \mathsf{y}'_h\}$ the set of output variables of φ'. We assume without loss of generality that the BSRV specifications φ and φ' have no output variable in common (i.e. $Y \cap Y' = \emptyset$). We describe each construction individually.

Intersection. The construction for intersection is illustrated in Fig. 1. The BSRV specification recognizing $\mathcal{L}(\varphi) \cap \mathcal{L}(\varphi')$ is simply the joint set of the equations of φ and φ'. Correctness of the construction immediately follows.

Union. The construction of the BSRV $\varphi \cup \varphi'$ recognizing $\mathcal{L}(\varphi) \cup \mathcal{L}(\varphi')$ is given in Fig. 1. Note that we use two new additional output variables: switch and main. These variables use a common gadget, built by assigning to an output variable y an equation of the form:

$$\mathsf{y} := \textit{if }\ \alpha \textit{ then } \mathsf{y} \textit{ else } \neg\mathsf{y}$$

This gadget forces the sub-expression α to be true at all positions in every valuation model. In the case of the construction for *union*, switch is forced by the sub-expression to be a uniform output variable, which is then used to guess whether the input has to be considered an input for φ or for φ'. Depending on the uniform value of switch (if it is in 0^+ or 1^+), the equation for the output variable main ensures that the input is recognized precisely when either the equations of φ are fulfilled or the equations of φ' are fulfilled.

Now, we show that the construction is correct by showing that $\mathcal{L}(\varphi \cup \varphi') = \mathcal{L}(\varphi) \cup \mathcal{L}(\varphi')$. For the inclusion $\mathcal{L}(\varphi \cup \varphi') \subseteq \mathcal{L}(\varphi) \cup \mathcal{L}(\varphi')$, let $\sigma_X \in \mathcal{L}(\varphi \cup \varphi')$. Hence, there is a valuation model σ'' of $\varphi \cup \varphi'$ associated with the input σ_X. By construction, $\sigma''(\mathsf{switch}) \in 0^+ \cup 1^+$. Assume that $\sigma''(\mathsf{switch}) \in 1^+$ (the other case is analogous). By definition of the equation for main, for every $1 \leq i \leq k$, $[\![\mathsf{y}_i, \sigma'']\!] = [\![\mathsf{E}_i, \sigma'']\!]$. Thus, the restriction of σ'' to $X \cup Y$ is a valuation model of φ. Hence, $\sigma_X \in \mathcal{L}(\varphi)$ and the result follows. For the converse inclusion $\mathcal{L}(\varphi) \cup \mathcal{L}(\varphi') \subseteq \mathcal{L}(\varphi \cup \varphi')$, let $\sigma_X \in \mathcal{L}(\varphi) \cup \mathcal{L}(\varphi')$. Assume that $\sigma_X \in \mathcal{L}(\varphi')$ (the other case being similar). Hence, there is a valuation model σ' of φ' associated with the input σ_X. Let σ'' be the stream valuation of $\varphi \cup \varphi'$ associated with the input σ_X defined as follows: for all variables z of $\varphi \cup \varphi'$, $\sigma''(\mathsf{z}) = \sigma'(\mathsf{z})$ if $\mathsf{z} \in X \cup Y'$, and $\sigma''(\mathsf{z}) \in 0^+$ otherwise. By construction, it follows that σ'' is a valuation model of $\varphi \cup \varphi'$. Hence $\sigma_X \in \mathcal{L}(\varphi \cup \varphi')$.

Reversal. The construction for *reversal* is very simple and appears in Fig. 1. The BSRV φ^R recognizing $\mathcal{L}(\varphi)^R$ is obtained from φ by replacing each equation $\mathsf{y} = \mathsf{E}$ with the equation $\mathsf{y} = \mathsf{E}^R$, where E^R denotes the stream expression obtained from E by replacing each sub-expression $\tau[k|d]$ with $k > 0$ for $\tau[-k|d]$, and replacing each sub-expression $\tau[-k|d]$ with $k > 0$ for $\tau[k|d]$.

We prove now that the construction is correct. For a finite word w and a position $1 \leq i \leq |w|$, we denote by w^R the reverse of w, and by $R(i)$ the position of w given by $|w| - i + 1$. Note that the suffix of w^R from position $R(i)$ is the reverse of the prefix of w

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

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

Union: $\varphi \cup \varphi' : \{\mathsf{y}_1 := \mathsf{y}_1, \ldots, \mathsf{y}'_h := \mathsf{y}'_h, \mathsf{switch} := \mathsf{E}_{\mathsf{switch}}, \mathsf{main} := \mathsf{E}_{\mathsf{main}}\}$

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

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

Reversal: $\varphi^R : \{\mathsf{y}_1 := \mathsf{E}_1^R, \ldots, \mathsf{y}_k := \mathsf{E}_k^R\}$
E_i^R is obtained from E_i by converting each offset k in its opposite $-k$.

Figure 1: Constructions for intersection, union, and reversal.

leading to position i. It follows by structural induction that for all stream expressions E, stream valuations σ over the variables of E, and $1 \leq i \leq |\sigma|$:

$$[\![\mathsf{E}, \sigma]\!](i) = [\![\mathsf{E}^R, \sigma^R]\!](R(i))$$

Then for all stream valuations σ over $X \cup Y$, σ is a valuation model of φ if and only if σ^R is a valuation model of φ^R. Hence, $\mathcal{L}(\varphi^R) = [\mathcal{L}(\varphi)]^R$.

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

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

$$\begin{aligned} \mathsf{E}_{\mathsf{wbegin}} &= \textit{if } (\mathsf{first} \rightarrow \mathsf{wbegin}) \wedge (\mathsf{wbegin} \rightarrow \mathsf{wend}[-1|1]) \textit{ then } \mathsf{wbegin} \textit{ else } \neg\mathsf{wbegin} \\ \mathsf{E}_{\mathsf{wend}} &= \textit{if } (\mathsf{last} \rightarrow \mathsf{wend}) \wedge (\mathsf{wend} \rightarrow \mathsf{wbegin}[+1|1]) \textit{ then } \mathsf{wend} \textit{ else } \neg\mathsf{wend} \end{aligned}$$

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

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

Figure 2: Construction for positive Kleene closure

Positive Kleene closure. The construction is given in Fig. 2. The BSRV specification φ^+ that recognizes $[\mathcal{L}(\varphi)]^+$ uses two new additional output variables: wbegin and wend, again defined using the gadget described above. 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 is used to mark the first positions of the components $\sigma_{X,i}$, and wend is used to mark the last position. 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 $\sigma_{X,i}$.

Now, we show that the construction is correct by proving that $\mathcal{L}(\varphi^+) = [\mathcal{L}(\varphi)]^+$. For the inclusion $\mathcal{L}(\varphi^+) \subseteq [\mathcal{L}(\varphi)]^+$, let $\sigma_X \in \mathcal{L}(\varphi^+)$. Hence, there is a valuation model σ of φ^+ associated with the input σ_X. By the equations for the output variables wbegin and wend, there is $N \geq 1$ such that σ can be written in the form $\sigma = \sigma_1 \cdot \ldots \cdot \sigma_N$ and for all $1 \leq \ell \leq N$, the Boolean stream $\sigma_\ell(\mathsf{wbegin})$ is in 10^* and the Boolean stream $\sigma_\ell(\mathsf{wend})$ is in 0^*1. Fix $1 \leq \ell \leq N$. We show that the restriction $(\sigma_\ell)_{X \cup Y}$ of σ_ℓ to the set of variables $X \cup Y$ is a valuation model of φ. Hence, membership of σ_X in $[\mathcal{L}(\varphi)]^+$ follows. For all positions $1 \leq i \leq |\sigma_\ell|$ along the stream valuation σ_ℓ, we denote by $p(i)$ the corresponding position along σ. In order to prove that σ_ℓ is a valuation model of φ, by hypothesis, it suffices to show that for each equation $\mathsf{y} = \mathsf{E}$ of φ and for each position $1 \leq i \leq |\sigma_\ell|$, the following holds, where $\mathsf{y} = \mathsf{E}^+$ is the equation of φ^+ associated to the output variable y:

$$[\![\mathsf{E}, (\sigma_\ell)_{X \cup Y}]\!](i) = [\![\mathsf{E}^+, \sigma]\!](p(i))$$

We just need to prove that for each subexpression $\tau[k|d]$ of φ, $[\![\tau[k|d], (\sigma_\ell)_{X \cup Y}]\!](i) = [\![\mathsf{E}_{\tau,k,d}, \sigma]\!](p(i))$, where the stream expression $\mathsf{E}_{\tau,k,d}$ is as in Fig. 2. There are two cases:

- $k > 0$: first, assume that $i+k \leq |\sigma_\ell|$. Since $\sigma_\ell(\mathsf{wbegin})$ is in 10^*, we obtain that for all $1 \leq j \leq k$, $\sigma(\mathsf{wbegin})(p(i)+j) = 0$. Hence, by definition of $\mathsf{E}_{\tau,k,d}$, it follows that $[\![\mathsf{E}_{\tau,k,d}, \sigma]\!](p(i)) = [\![\tau[k|d], \sigma]\!](p(i)) = [\![\tau[k|d], (\sigma_\ell)_{X \cup Y}]\!](i)$, and the result follows in this case. Now, assume that $i + k > |\sigma_\ell|$. Hence, $[\![\tau[k|d], (\sigma_\ell)_{X \cup Y}]\!](i) = d$. Then, either $p(i) + k > |\sigma|$, or there is $1 \leq j \leq k$ such that $\sigma(\mathsf{wbegin})(p(i) + j) = 1$. By definition of $\mathsf{E}_{\tau,k,d}$, it follows that $[\![\mathsf{E}_{\tau,k,d}, \sigma]\!](p(i)) = d$, and the result follows in this case as well.

- $k < 0$: first, assume that $i - k \geq 1$. Since $\sigma_\ell(\mathsf{wend})$ is in 0^*1, we obtain that for all $1 \leq j \leq -k$, $\sigma(\mathsf{wend})(p(i) - j) = 0$. Hence, by definition of $\mathsf{E}_{\tau,k,d}$, it follows that $[\![\mathsf{E}_{\tau,k,d}, \sigma]\!](p(i)) = [\![\tau[k|d], \sigma]\!](p(i)) = [\![\tau[k|d], (\sigma_\ell)_{X \cup Y}]\!](i)$, and the result follows in this case. Now, assume that $i - k < 1$. Hence, $[\![\tau[k|d], (\sigma_\ell)_{X \cup Y}]\!](i) = d$. Then, either $p(i) - k < 1$, or there is $1 \leq j \leq -k$ such that $\sigma(\mathsf{wend})(p(i) - j) = 1$. By definition of $\mathsf{E}_{\tau,k,d}$, it follows that $[\![\mathsf{E}_{\tau,k,d}, \sigma]\!](p(i)) = d$, and the result follows in this case as well.

For the converse inclusion $[\mathcal{L}(\varphi)]^+ \subseteq \mathcal{L}(\varphi^+)$, let $\sigma_X \in [\mathcal{L}(\varphi)]^+$. Hence, there is a stream valuation σ of φ associated with the input σ_X such that σ is of the form $\sigma = \sigma_1 \cdot \dots \cdot \sigma_N$ for some $N \geq 1$, and σ_ℓ is a valuation model of φ for all $1 \leq \ell \leq N$. Let $\sigma'' = \sigma''_1 \cdot \dots \cdot \sigma''_N$ be the extension of σ over $X \cup Y \cup \{\mathsf{wbegin}, \mathsf{wend}\}$, where the Boolean streams $\sigma''(\mathsf{wbegin})$ and $\sigma''(\mathsf{wend})$ are defined as follows: for all $1 \leq \ell \leq N$, $\sigma''_\ell(\mathsf{wbegin})$ is in 10^* and the Boolean stream $\sigma''_\ell(\mathsf{wend})$ is in 0^*1. We show that σ'' is a valuation model of φ^+, hence, membership of σ_X in $\mathcal{L}(\varphi^+)$ follows. By construction, the equations for the output variables wbegin and wend are satisfied with respect to the stream valuation σ''. Now, let us consider an equation $\mathsf{y} = \mathsf{E}$ of φ^+ associated with an output variable $\mathsf{y} \in Y$. By construction, in order to show that $[\![\mathsf{y}, \sigma'']\!] = [\![\mathsf{E}, \sigma'']\!]$, it suffices to prove that for all $1 \leq \ell \leq N$, $1 \leq i \leq |\sigma_\ell|$, and subexpression $\tau[k|d]$ of φ, $[\![\tau[k|d], \sigma_\ell]\!](i) = [\![\mathsf{E}_{\tau,k,d}, \sigma'']\!](p(i))$, where for $1 \leq i \leq |\sigma_\ell|$, $p(i)$ denotes the corresponding position along σ. This can be shown as for the proof of the inclusion $\mathcal{L}(\varphi^+) \subseteq [\mathcal{L}(\varphi)]^+$.

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

Concatenation: $\{\mathsf{y}_1, \ldots, \mathsf{y}_k\} \cap \{\mathsf{y}'_1, \ldots, \mathsf{y}'_h\} = \emptyset$

$$\begin{aligned}
\varphi \cdot \varphi' : \{\mathsf{y}_1 \ &:= \mathit{if}\ \mathsf{wmark}\ \mathit{then}\ \widetilde{\mathsf{E}}_1\ \mathit{else}\ \mathsf{y}_1, \\
&\ldots, \\
\mathsf{y}_k \ &:= \mathit{if}\ \mathsf{wmark}\ \mathit{then}\ \widetilde{\mathsf{E}}_k\ \mathit{else}\ \mathsf{y}_k, \\
\mathsf{y}'_1 \ &:= \mathit{if}\ \neg\mathsf{wmark}\ \mathit{then}\ \widetilde{\mathsf{E}}'_1\ \mathit{else}\ \mathsf{y}'_1, \\
&\ldots, \\
\mathsf{y}'_h \ &:= \mathit{if}\ \neg\mathsf{wmark}\ \mathit{then}\ \widetilde{\mathsf{E}}'_h\ \mathit{else}\ \mathsf{y}'_h, \\
\mathsf{wmark} \ &:= \mathsf{E}_{\mathsf{wmark}}\}
\end{aligned}$$

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

$\widetilde{\mathsf{E}}_i$ 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:

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

Figure 3: Construction for concatenation

Concatenation. The construction is given in Fig. 3. The BSRV specification $\varphi \cdot \varphi'$ recognizing $\mathcal{L}(\varphi) \cdot \mathcal{L}(\varphi')$ uses a new additional output variable, wmark, once again based on the same gadget. This variable is used for guessing 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 stream for 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.

Now, we show that the construction is correct: $\mathcal{L}(\varphi \cdot \varphi') = \mathcal{L}(\varphi) \cdot \mathcal{L}(\varphi')$. For the inclusion $\mathcal{L}(\varphi \cdot \varphi') \subseteq \mathcal{L}(\varphi) \cdot \mathcal{L}(\varphi')$, let $\sigma_X \in \mathcal{L}(\varphi \cdot \varphi')$. Hence, there is a valuation model σ'' of $\varphi \cdot \varphi'$ associated with the input σ_X. By the equation for the output variable wmark, σ'' can be written in the form $\sigma'' = \sigma \cdot \sigma'$ such that $\sigma(\mathsf{wmark})$ is in 1^+ and $\sigma'(\mathsf{wmark})$ is in 0^+. We show that the restriction $(\sigma)_{X \cup Y}$ of σ to the set of variables $X \cup Y$ is a valuation model of φ, and the restriction $(\sigma')_{X \cup Y'}$ of σ' to the set of variables $X \cup Y'$ is a valuation model of φ'. Hence, membership of σ_X in $\mathcal{L}(\varphi) \cdot \mathcal{L}(\varphi')$ follows. We consider the stream valuation $(\sigma)_{X \cup Y}$ (the proof for $(\sigma')_{X \cup Y'}$ is similar). In order to prove that $(\sigma)_{X \cup Y}$ is a valuation model of φ, by hypothesis and construction, it suffices to show that for each equation $\mathsf{y} = \mathsf{E}$ of φ and for each position $1 \leq i \leq |\sigma|$, the following holds, where $\mathsf{y} = \textit{if } \mathsf{wmark} \textit{ then } \mathsf{E}'' \textit{ else } \mathsf{y}$ is the equation of $\varphi \cdot \varphi'$ associated to the variable y:

$$[\![\mathsf{E}, (\sigma)_{X \cup Y}]\!](i) = [\![\mathsf{E}'', \sigma'']\!](i)$$

By construction, we just need to prove that for each subexpression $\tau[k|d]$ of φ such that $k > 0$, $[\![\tau[k|d], (\sigma)_{X \cup Y}]\!](i) = [\![\mathsf{E}_{\tau,k,d}, \sigma'']\!](i)$, where the stream expression $\mathsf{E}_{\tau,k,d}$ is as in Fig. 3. This is analogous to to the proof for the positive Kleene closure.

For the converse inclusion $\mathcal{L}(\varphi) \cdot \mathcal{L}(\varphi') \subseteq \mathcal{L}(\varphi \cdot \varphi')$, let $\sigma''_X \in \mathcal{L}(\varphi) \cdot \mathcal{L}(\varphi')$. Hence, $\sigma''_X = \sigma_X \cdot \sigma'_X$ and there are a valuation model σ_m of φ associated with the input σ_X and a valuation model σ'_m of φ' associated with the input σ'_X. Let σ'' be the stream valuation over $X \cup Y \cup Y' \cup \{\mathsf{wmark}\}$ associated with the input σ''_X defined as follows: $\sigma'' = \sigma \cdot \sigma'$ with $|\sigma| = |\sigma_X|$, where (i) $\sigma(\mathsf{z}) = \sigma_m(\mathsf{z})$ if $\mathsf{z} \in X \cup Y$, and $\sigma(\mathsf{z}) = 1$ otherwise, and (ii) $\sigma'(\mathsf{z}) = \sigma'_m(\mathsf{z})$ if $\mathsf{z} \in X \cup Y'$, and $\sigma'(\mathsf{z}) = 0$ otherwise. By construction σ'' is a valuation model of $\varphi \cdot \varphi'$. Hence, membership of σ''_X in $\mathcal{L}(\varphi \cdot \varphi')$ follows.

This concludes the proof of Theorem 4. $\square$

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 trace (input streams) and singly exponential in the size of the specification.

Our algorithm also solves the following question which is left open in [10]. In offline monitoring, one can afford to traverse the input stream several times, in the backward and in the forward directions. However, efficient algorithms must perform each traversal using only a bounded amount of memory that does not depend on the length of the trace. The algorithm in [10] for offline monitoring starts by analyzing the specification, calculating the offset dependencies between output streams by looking at their defining equations. Then, the algorithm performs a forward traversal to resolve (in a memory-less manner) the values of an output stream variable based on the values of other output variables that have been already resolved and which appear with past offsets. Similarly, backward traversals are performed to resolve expressions with future

```
Monitoring(φ, σ_X)        /** φ is a well-defined BSRV and A_φ = ⟨Q, q_0, δ, F⟩ **/
  Λ ← {q_0}
  for i = 1 upto |σ_X| do
    update Λ ← {q ∈ Q | q ∈ δ(p, σ_X(i)) for some p ∈ Λ}
    store Λ at position i on the tape
  for i = |σ_X| downto 1 do
    let Λ be the set of states stored at position i on the tape
    if i = |σ_X| then p ← the unique accepting state in Λ
    else let q be the unique state in Λ such that p ∈ δ(q, σ_X(i + 1));
         update p ← q
    output at position i the main value of p
```

Figure 4: Offline monitoring algorithm for well-defined BSRV

offsets. In summary, the algorithm in [10] performs a number of passes proportional to the number of backward and forward references in the defining equations. Intuitively speaking, this number of passes corresponds to the number of alternations between future and past operators in a temporal logic specification. The open question is whether a specification can be modified into an equivalent specification that only requires a constant number of forward and backward passes, each of which uses an amount of memory that does not depend on the length of the trace. We show here that only two passes (one forward and one backward) are required for 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 $1 \leq i \leq k$. 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, for every 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. Hence, we obtain the following.

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

Proof. The specification φ is well-defined iff for every input σ_X, φ admits one and only one (expanded) valuation model iff for every input σ_X, there is one and only one accepting run of $\mathcal{A}_\varphi$ over σ_X iff $\mathcal{A}_\varphi$ is universal and unambiguous. $\square$

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 an input 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. 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 traversal of the input trace is performed, and the algorithm outputs a stream valuation of φ.

We claim that the uniqueness conditions in the second phase of the algorithm are satisfied, and the output is the unique valuation model of the *well-defined* BSRV specification φ associated with the input σ_X. By Proposition 2, $\mathcal{A}_\varphi$ is universal. Thus, by the classical construction of the DFA associated with $\mathcal{A}_\varphi$, it holds that for all $1 \leq i \leq |\sigma_X|$,

$$\begin{aligned} &\Lambda(i) \neq \emptyset \text{ and for each state } q \text{ of } \mathcal{A}_\varphi,\\ &\qquad q \in \Lambda(i) \text{ if and only if}\\ &\qquad \text{there is a run of } \mathcal{A}_\varphi \text{ over } \sigma_X(1), \ldots, \sigma_X(i) \text{ leading to state } q. \end{aligned} \tag{1}$$

We assume that the uniqueness conditions are not satisfied, and derive a contradiction. Let i be the greatest position along σ_X such that the uniqueness condition at step i is not satisfied. Assume that $i < |\sigma_X|$ (the other case being simpler). For each $q \in Q$, let $\mathcal{A}_\varphi^q$ be the NFA obtained from $\mathcal{A}_\varphi$ by replacing the initial state q_0 with q. By Condition (1) above, $\Lambda(i) \neq \emptyset$. Hence, by hypothesis, there are a state $p \in \Lambda(i+1)$ and two distinct states $q, q' \in \Lambda(i)$ such that $p \in \delta(q, \sigma_X(i+1))$ and $p \in \delta(q', \sigma_X(i+1))$. Moreover, by construction of the algorithm, there is a run of $\mathcal{A}_\varphi^p$ over $\sigma_X(i+2), \ldots, \sigma_X(|\sigma_X|)$ leading to an accepting state q_{acc}. By Condition (1) above, we deduce that there are two distinct accepting runs of $\mathcal{A}_\varphi$ over σ_X. This is a contradiction because by Proposition 2, $\mathcal{A}_\varphi$ is unambiguous. Hence, the uniqueness conditions in the second phase of the algorithm are satisfied. Moreover, by construction, it follows that 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 a well-defined BSRV specification, that runs 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 [10], a syntactical condition for general SRV, called *well-formedness*, is introduced, which can be checked in polynomial time and guarantees that the semantic condition given by well-definedness is met. 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 [10], 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* [10], such that the monitoring algorithm processes each position of the input trace exactly $ad(\varphi)+1$ times. An important question left open in [10] is whether for a well-formed SRV specification φ, it is possible to construct a φ-equivalent SRV specification 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 1.

We give now the formal details. First, we revisit the notion of well-formedness [10]. Given a general SRV specification φ over X and Y, the *dependency graph* G_φ of φ is the finite weighted directed graph whose set of vertices is Y and whose set of weighted edges is defined as follows. There is an edge $\mathsf{y} \xrightarrow{k} \mathsf{z}$ in the graph whenever the equation $\mathsf{y} = \mathsf{E}$ of φ associated with y, either $k = 0$ and z occurs in E, or $k \neq 0$ and $\mathsf{z}[k|d]$ occurs in E for some d. The weight of a finite path of G_φ is the sum of the weights of its edges. An SRV specification φ is *well-formed* if its dependency graph G_φ has no cycle with weight zero. As shown in [10], for a well-formed SRV specification φ, a strongly connected component (*SCC*) of G_φ can be classified as positive or negative, where an *SCC* is positive if every cycle in the *SCC* has weight strictly positive, or negative if every cycle is strictly negative. Clearly, every SCC is positive or negative because otherwise one can build a zero weight path by traversing a negative and a positive cycle a sufficient number of times to cancel each other. The alternation depth $ad(\varphi)$ of a well-formed SRV is then defined as the maximum over the number of alternations between positive and negative vertices along a path of G_φ, where a vertex is positive if it belongs to a positive *SCC* and negative if it belongs to a negative *SCC*.

We establish the following result for the class of BSRV.

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

Proof. Let φ be a well-defined BSRV specification over X and Y, and $\mathcal{A}_\varphi = \langle Q, q_0, \delta, F\rangle$ be the NFA over 2^X accepting $\mathcal{L}(\varphi)$ built in the proof of Theorem 1. We denote by $\mathcal{D}_\varphi$ the DFA accepting $\mathcal{L}(\varphi)$ resulting by determinizing $\mathcal{A}_\varphi$ using the classical power set construction. Recall that $\mathcal{D}_\varphi = \langle 2^Q, \{q_0\}, \delta_\mathcal{D}, F_\mathcal{D}\rangle$, where $F_\mathcal{D} = \{P \in 2^Q \mid P \cap F \neq \emptyset\}$ and for all $P \in 2^Q$ and $\iota \in 2^X$, $\delta_\mathcal{D}(P, \iota) = \{q \in Q \mid q \in \delta(p, \iota)$ for some $p \in P\}$. For each output variable $\mathsf{y} \in Y$, we denote by $Q(\mathsf{y})$ the set of $\mathcal{A}_\varphi$-states $q =$

$(a_{-b(\varphi)}, \dots, a_{-1}, a_0, a_1, \dots, a_{f(\varphi)}) \in Q \setminus \{q_0\}$ such that y is in the main value a_0 of q. In the following, we construct a BSRV specification φ' over X and $Y' \supseteq Y$ satisfying the statement of the theorem, where the set of output variables Y' is defined as follows

$$Y' = Y \cup \bigcup_{q \in Q \setminus \{q_0\}} \{\mathsf{q}\} \cup \bigcup_{P \in 2^Q} \{\mathsf{P}\}$$

Thus, we associate to each non-initial $\mathcal{A}_\varphi$-state q, an output variable q, and to each $\mathcal{D}_\varphi$-state $P \in 2^Q$, an output variable P.

For each $\iota \in 2^X$, let E_ι be the Boolean expression over X encoding precisely the input symbol ι given by $\mathsf{E}_\iota := (\bigwedge_{\mathsf{x} \in \iota} \mathsf{x}) \wedge (\bigwedge_{\mathsf{x} \in X \setminus \iota} \neg \mathsf{x})$. Additionally, for all $q \in Q$ and $P \in 2^Q$, let Acc_q and $\mathsf{Init}(P, \iota)$ be the Boolean constants defined as follows:

$$\mathsf{Acc}_q = \begin{cases} 1 & \text{if } q \in F \\ 0 & \text{otherwise} \end{cases} \qquad \mathsf{Init}(P, \iota) = \begin{cases} 1 & \text{if } P = \delta_{\mathcal{D}}(\{q_0\}, \iota) \\ 0 & \text{otherwise} \end{cases}$$

Then, for all $P \in 2^Q$, $q \in Q \setminus \{q_0\}$, and $\mathsf{y} \in Y$, the equations of the BSRV φ' for the output variables P, q, and y are defined as follows.

$$\mathsf{P} := \mathit{if}\ \mathsf{first}\ \mathit{then} \bigwedge_{\iota \in 2^X} (\mathsf{E}_\iota \rightarrow \mathsf{Init}(P, \iota))\ \mathit{else} \bigvee_{(P', \iota) \in \{(P', \iota) \mid P = \delta_{\mathcal{D}}(P', \iota)\}} \mathsf{E}_\iota \wedge \mathsf{P}'[-1|0]$$

$$\mathsf{q} := \mathit{if} \bigvee_{P \in \{P \in 2^Q \mid q \in P\}} \mathsf{P}\ \mathit{then}\ \mathsf{E}_q\ \mathit{else}\ 0$$

$$\text{where } \mathsf{E}_q := \mathit{if}\ \mathsf{last}\ \mathit{then}\ \mathsf{Acc}_q\ \mathit{else} \bigvee_{\iota \in 2^X} \bigvee_{q' \in \delta(q, \iota)} (\mathsf{E}_\iota[+1|0] \wedge \mathsf{q}'[+1|0])$$

$$\mathsf{y} := \bigvee_{q \in Q(y)} \mathsf{q}$$

By construction, the specification BSRV φ' is well-formed—and consequently well-defined—and the alternation depth of φ' is exactly 1. We still need to show that φ' is φ-equivalent. Let σ_X be an input stream valuation, and σ and σ' be the unique valuation models of φ and φ', respectively, associated with the input σ_X. We need to prove that the restrictions of σ and σ' to Y coincide. Since φ is well-defined, by Proposition 2, $\mathcal{A}_\varphi$ is universal and unambiguous. Let $\pi = q_0, q_1, \dots, q_{|\sigma_X|}$ be the unique accepting run of $\mathcal{A}_\varphi$ over σ_X (which encodes σ). Then, by the equations of φ' associated with the output variables $\mathsf{y} \in Y$, it suffices to prove the following condition:

$$\begin{array}{l}\text{for each } 1 \leq i \leq |\sigma_X|, \text{ there is exactly one state } p \in Q \setminus \{q_0\} \text{ such that} \\ \qquad \mathsf{p} \in \sigma'(i) \text{ and } p = q_i \end{array} \tag{2}$$

Let $\pi_{\mathcal{D}} = \{q_0\}, P_1, \dots, P_{|\sigma_X|}$ be the run of $\mathcal{D}_\varphi$ over σ_X. First, we observe that the equations for the output variables P ensure that for each $1 \leq i \leq |\sigma_X|$, there is exactly one $\mathcal{D}_\varphi$-state $P \in 2^Q$ such that $\mathsf{P} \in \sigma'(i)$, and $P = P_i$. By using this observation and the fact that $\mathcal{A}_\varphi$ is universal and unambiguous, and proceeding as in the proof of correctness of the algorithm of Figure 4, Condition (2) easily follows, which concludes the proof. $\square$

5. Decision Problems

In this section, we show complexity results for some relevant decision problems related to BSRV specifications. 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 the class of BSRV, the following hold:

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. When BSRV are interpreted as language recognizers, language emptiness is PSPACE-complete, while language universality, language inclusion, and language equivalence are EXPSPACE-complete.
3. Checking semantic equivalence is EXPSPACE-complete.

In the following Subsections 5.1 and 5.2, we establish the upper bounds and the lower bounds of Theorem 7.

5.1. Upper bounds of Theorem 7

We first need a preliminary result (Proposition 3 below). 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.

Proof. The following non-deterministic algorithm solves the problem, given the input $(\mathcal{A}, \Upsilon)$: at each step, the algorithm guesses two runs π and π' of $\mathcal{A}$ over the same input. The algorithm keeps in memory only the pair of states (q, q'), where q is the last state of π and q' is the last state of π', and a flag f which is 1 whenever the projections

$\Upsilon(\pi)$ and $\Upsilon(\pi')$ of the two runs π and π' guessed so far are distinct. Initially, q and q' coincide with the initial state, and $f = 0$. If $f = 1$, and q and q' are both accepting (and then π and π' are two accepting runs over the same input and $\Upsilon(\pi)$ and $\Upsilon(\pi')$ are distinct), the algorithm terminates with success. Otherwise, the algorithm guesses two transitions of $\mathcal{A}$ from q and q' reading the same input symbol, leading to states p and p', respectively, re-writes the memory by replacing the pair (q, q') with the new pair (p, p'), and the flag f with the new flag f', where f' is 1 precisely whenever either $f = 1$ or $\Upsilon(p)$ and $\Upsilon(p')$ are distinct, and the whole procedure is repeated. $\square$

Now, we provide the upper bounds of Theorem 7. Fix a BSRV φ over X and Y, and let $\mathcal{A}_\varphi$ be the NFA of Theorem 1 accepting $\mathcal{L}(\varphi)$ and whose size is *singly exponential* in the size of φ.

Membership in PSPACE for under-definedness of BSRV. By Theorem 1 and Lemma 1, for every input σ_X, there is a bijection between the set of accepting runs of $\mathcal{A}_\varphi$ over σ_X and the set of valuation models of φ associated with σ_X. Hence, φ is under-defined if and only if $\mathcal{A}_\varphi$ is *not* unambiguous. Since $\mathcal{A}_\varphi$ can be constructed on the fly and PSPACE = NPSPACE, by Proposition 3 (with Υ as the identity map), it follows that the under-definedness problem is in PSPACE.

Membership in EXPTIME for well-definedness of BSRV. Checking universality of unambiguous NFA can be done in polynomial time [30]. By Proposition 2, φ is well-defined if and only if $\mathcal{A}_\varphi$ is universal and unambiguous. By Proposition 3, checking that $\mathcal{A}_\varphi$ is unambiguous can be done in PSPACE (in the size of φ). Thus, since the size of $\mathcal{A}_\varphi$ is singly exponential in the size of φ, we obtain that checking well-definedness for BSRV is in EXPTIME.

Membership in EXPSPACE for over-definedness and generalized well-definedness of BSRV. 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 PSPACE-complete [31], membership in EXPSPACE for checking over-definedness follows. Now, let us consider the generalized well-definedness problem. Let $Z \subseteq Y$ be a subset of the output variables. 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. By Theorem 1, given an input σ_X, the accepting runs of $\mathcal{A}_\varphi$ over σ_X are the expanded valuation models of φ encoding the valuation models of φ associated with the input σ_X. 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 $\Upsilon_Z(\pi) = \Upsilon_Z(\pi')$ if and only if the restrictions of σ and σ' to Z coincide. By Theorem 1, we obtain that φ is well-defined with respect to Z if and only if $\mathcal{A}_\varphi$ 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.

Membership in PSPACE for language emptiness. Since checking language emptiness for NFA is NLOGSPACE-complete, by Theorem 1, the result follows.

Membership in EXPSPACE for language universality, language inclusion, and language equivalence of BSRV. Recall that for NFA, universality, inclusion, and equivalence are PSPACE-complete [31]. Hence, by Theorem 1, the result follows.

Membership in EXPSPACE for semantic equivalence of BSRV. Let φ be a BSRV over X and Y, φ' be a BSRV over X and Y', and $Z \subseteq Y \cap Y'$. By a straightforward adaptation of Theorem 1, one can construct in singly exponential time two NFA $\mathcal{A}_{\varphi,Z}$ and $\mathcal{A}_{\varphi',Z}$ over $2^{X \cup Z}$ such that $\mathcal{L}(\mathcal{A}_{\varphi,Z})$ is the set of stream valuations over $X \cup Z$ which can be extended to valuation models of φ, and $\mathcal{L}(\mathcal{A}_{\varphi',Z})$ is the set of stream valuations over $X \cup Z$ which can be extended to valuation models of φ'. It follows that φ and φ' are equivalent with respect to Z if and only if $\mathcal{L}(\mathcal{A}_{\varphi,Z}) = \mathcal{L}(\mathcal{A}_{\varphi',Z})$. Thus, since language equivalence for NFA is PSPACE-complete, membership in EXPSPACE for the considered problem follows.

5.2. Lower bounds of Theorem 7

First, we consider the EXPSPACE-hardness results of Theorem 7.

EXPSPACE-hardness for over-definedness of BSRV. The result is obtained by a polynomial-time reduction from a domino-tiling problem for grids with rows of singly exponential length [32]. An instance $\mathcal{I}$ of this problem is a tuple $\mathcal{I} = \langle C, \Delta, m, d_{init}, d_{final} \rangle$, where C is a finite set of colors, $\Delta \subseteq C^4$ is a set of tuples $\langle c_{down}, c_{left}, c_{up}, c_{right} \rangle$ of four colors, called *domino-types*, $m > 0$ is a natural number (written in unary), and $d_{init}, d_{final} \in \Delta$ are two domino-types. For $n > 0$, a *tiling of $\mathcal{I}$ for the $n \times 2^m$-grid* is a mapping $f : [0, n-1] \times [0, 2^m - 1] \rightarrow \Delta$ satisfying the following:

- *row requirement:* two adjacent cells in a row have the same color on the shared edge: for all $(i, j) \in [0, n-1] \times [0, 2^m - 1]$ with $j < 2^m - 1$, $[f(i, j)]_{right} = [f(i, j+1)]_{left}$;
- *column requirement:* two adjacent cells in a column have the same color on the shared edge: for all $(i, j) \in [0, n-1] \times [0, 2^m - 1]$ with $i < n-1$, $[f(i, j)]_{up} = [f(i+1, j)]_{down}$;

- $f(0,0) = d_{init}$ and $f(n-1, 2^m-1) = d_{final}$.

A tiling of $\mathcal{I}$ is a tiling of $\mathcal{I}$ for the $n \times 2^m$-grid for some $n > 0$.

Checking the existence of a tiling for $\mathcal{I}$ is $\mathsf{EXPSPACE}$-complete [32]. We also use the notion of *pseudo-tiling* for $\mathcal{I}$, which is similar to the notion of tiling but the column requirement is relaxed. Note that the column requirement is the crucial feature which makes the considered domino tiling problem $\mathsf{EXPSPACE}$-hard.[1] In the following, we construct in polynomial-time a BSRV specification φ such that there exists a tiling for $\mathcal{I}$ if and only if φ is over-defined. The set X of input variables of φ is given by

$$X = \{\mathsf{d} \mid d \in \Delta\} \cup \{\mathsf{b}_1^+, \ldots, \mathsf{b}_m^+, \mathsf{b}_1^-, \ldots, \mathsf{b}_m^-\}$$

We associate to each domino-type $d \in \Delta$ an input variable d. The additional input variables $\mathsf{b}_1^+, \ldots, \mathsf{b}_m^+, \mathsf{b}_1^-, \ldots, \mathsf{b}_m^-$ are used to encode the value of an m-bit counter numbering the cells of one row of the grid (b_1^+ is 1 whenever the i-th bit is 1, and b_i^- is 1 whenever the i-th bit is 0). Thus, a cell is encoded as a finite word over 2^X of length $m+1$, the first m positions giving the binary encoding of the column number and the last position giving the associated domino-type.[2] More precisely, a cell with content $d \in \Delta$ and column number $j \in [0, 2^m - 1]$ is encoded by the word $\{\mathsf{d}\}\{b_1\}\ldots\{b_m\}$, where $b_k \in \{\mathsf{b}_k^+, \mathsf{b}_k^-\}$ for all $k \in [1, m]$, and $b_k = \mathsf{b}_k^+$ iff the k^{th} bit in the binary encoding of the column number j is 1. A tiling is then encoded as a sequence of rows, starting from the first row, where a row lists the encodings of cells from left to right.

In the following, for a stream variable y and an integer k, we use $\mathsf{y}[k]$ for the stream expression $\mathsf{y}[k|1]$ if $k \neq 0$, and for the stream expression y if $k = 0$.

We illustrate now the construction of φ that ensures that the unique inputs for which there is no output stream valuation are those encoding tilings of $\mathcal{I}$.

The preliminary step in the construction of φ is to enforce a designated output variable PTU to be uniform with the uniform value 1 characterizing the inputs encoding pseudo-tilings. For this, we need two additional output variables PT and test_1.

The equation for the output variable PT ensures that PT assumes the value 1

[1] One can easily show that checking the existence of a *pseudo-tiling* is just PSPACE-complete.

[2] We assume that the first bit is the least significant one.

everywhere *iff* the input streams encode a pseudo-tiling:

$$\mathsf{PT} := \underbrace{\bigvee_{\mathsf{x}\in X} (\mathsf{x} \wedge \bigwedge_{\mathsf{x}'\in X\setminus\{\mathsf{x}\}} \neg\mathsf{x}')}_{\text{exactly one input variable has value 1}} \quad \wedge$$

$$\underbrace{\bigwedge_{i=1}^{m-1} \big((\mathsf{b}_\mathsf{i}^+ \vee \mathsf{b}_\mathsf{i}^-) \rightarrow (\neg\mathsf{last} \wedge (\mathsf{b}_\mathsf{i+1}^+[+1] \vee \mathsf{b}_\mathsf{i+1}^-[+1]))\big) \wedge \big((\mathsf{b}_\mathsf{m}^+ \vee \mathsf{b}_\mathsf{m}^-) \rightarrow (\neg\mathsf{last} \wedge \bigvee_{d\in\Delta} \mathsf{d}[+1])\big)}_{\text{the input is a list of numbered cells}} \wedge$$

$$\underbrace{\big(\mathsf{first} \rightarrow (\bigwedge_{i=1}^{m} \mathsf{b}_\mathsf{i}^-[i-1] \wedge \mathsf{d}_{init}[+m])\big) \wedge}_{\text{the input starts with a } d_{init}\text{-cell numbered } 0} \quad \underbrace{\big(\mathsf{last} \rightarrow (\bigwedge_{i=1}^{m} \mathsf{b}_\mathsf{i}^+[i-m-1] \wedge \mathsf{d}_{\mathit{final}})\big)}_{\text{the input ends with a } d_{\mathit{final}}\text{-cell numbered } 2^m-1} \quad \wedge$$

$$\underbrace{\begin{aligned}&\Big\{(\neg\mathsf{last} \wedge \bigvee_{d\in\Delta} \mathsf{d}) \;\longrightarrow\; \Big((\mathsf{b}_\mathsf{1}^+[+1] \vee \mathsf{b}_\mathsf{1}^-[+1]) \wedge \\ &(\mathsf{b}_\mathsf{1}^+[-m] \leftrightarrow \mathsf{b}_\mathsf{1}^-[+1]) \wedge \bigwedge_{i=1}^{m-1} (\mathsf{b}_\mathsf{i+1}^+[-m+i] \leftrightarrow \mathsf{b}_\mathsf{i+1}^-[i+1]) \leftrightarrow (\mathsf{b}_\mathsf{i}^+[-m+i-1] \wedge \mathsf{b}_\mathsf{i}^-[i])\Big)\Big\}\end{aligned}}_{\text{the cells are listed in increasing order modulo } 2^m} \wedge$$

$$\underbrace{\bigwedge_{d\in\Delta} \mathsf{d} \rightarrow (\mathsf{last} \vee \bigwedge_{i=1}^{i=m} \mathsf{b}_\mathsf{i}^-[i] \vee \bigvee_{d'\in\Delta:(d')_{\mathit{left}}=(d)_{\mathit{right}}} \mathsf{d}'[m+1])}_{\text{two adjacent cells in a row have the same color on the shared edge}}$$

The equation for the output variable PTU is given by

$$\mathsf{PTU} \;\; := \;\; \mathit{if}\;\; (\neg\mathsf{first} \rightarrow (\mathsf{PTU}[-1] \leftrightarrow \mathsf{PTU})) \wedge (\neg\mathsf{PT} \rightarrow \neg\mathsf{PTU}) \;\mathit{then}\; \mathsf{PTU} \;\mathit{else}\; \neg\mathsf{PTU}$$

Hence, PTU is a uniform output variable, and the uniform value of PTU is 0 whenever PT assumes the value 0 at some position. We exploit the additional variable $\mathsf{test_1}$ in order to avoid situations where PT is everywhere 1 and the uniform value of PTU is 0. The equation for $\mathsf{test_1}$ is as follows:

$$\begin{aligned}\mathsf{test_1} \;\; := \;\; &\mathit{if}\;\; \{((\mathsf{first} \wedge \mathsf{PT} \wedge \neg\mathsf{PTU}) \rightarrow \neg\mathsf{test_1}) \wedge (\mathsf{last} \rightarrow \mathsf{test_1}) \wedge \\ &\quad ((\neg\mathsf{first} \wedge \neg\mathsf{test_1}[-1] \wedge \mathsf{PT} \wedge \neg\mathsf{PTU}) \rightarrow \neg\mathsf{test_1})\} \;\mathit{then}\; \mathsf{test_1} \;\mathit{else}\; \neg\mathsf{test_1}\end{aligned}$$

Thus, the uniform value of PTU is 1 iff the input encodes a pseudo-tiling (note that there is an output valuation of PT, PTU, and $\mathsf{test_1}$ for each input).

Now, we describe the crucial step of the construction of φ. Assume that the input encodes a pseudo-tiling, i.e. the uniform value of PTU is 1. Then, the input is *not* a tiling of $\mathcal{I}$ if and only if the followign condition holds:

> there are two adjacent cells (i, j) and $(i+1, j)$ in some column which have *different* color on the shared edge. (3)

In order to check Condition (3), we use $O(m + |\Delta|)$ additional output variables: the equations for these variables ensure that for each input σ_X, there is a valuation model over σ_X if and only if whenever σ_X encodes a pseudo-tiling, the above Condition (3) is satisfied. Hence, the unique inputs for which there is no stream valuation are those encoding tilings of $\mathcal{I}$.

The fulfillment of Condition (3) for an input which is a pseudo-tiling is ensured in three steps.

- *First step*: we use two output variables, $\mathsf{Bl_1}$ and $\mathsf{Bl_2}$, for 'marking' two cells c_1 and c_2 of the pseudo-tiling.
- *Second step*: we use three additional output variables, namely, $\mathsf{m_1}$, $\mathsf{m_2}$, and $\mathsf{test_2}$ in order to check that the two cells c_1 and c_2 marked by $\mathsf{Bl_1}$ and $\mathsf{Bl_2}$, respectively, belong to two adjacent rows (with the c_2's row following the c_1's row).
- *Third step*: finally, we use the output variables $\mathsf{ob_1}, \ldots, \mathsf{ob_m}$, $\mathsf{test_3}$, and od for each $d \in \Delta$ to guarantee that the cells marked by $\mathsf{Bl_1}$ and $\mathsf{Bl_2}$ have the same column number but different color on the shared edge.

Now, we proceed with the formal definition of the equations for the output variables of Steps 1–3.

Equations for Step 1. For each $h = 1, 2$, the equation for the output variable $\mathsf{Bl_h}$ requires that whenever the uniform value of PTU is 1, then the stream for $\mathsf{Bl_h}$ is in 0^*1^+ and the suffix in 1^+ starts with a cell ("the cell marked by $\mathsf{Bl_h}$").

$$\begin{aligned}\mathsf{Bl_h} \;\; := \;\; & \textit{if } \mathsf{PTU} \rightarrow \{(\mathsf{last} \rightarrow \mathsf{Bl_h}) \wedge ((\mathsf{Bl_h} \wedge \neg \mathsf{Bl_h}[-1|0]) \rightarrow (\mathsf{b_1^+} \vee \mathsf{b_1^-})) \wedge \\ & (\neg \mathsf{Bl_h} \rightarrow \neg \mathsf{Bl_h}[-1|0]) \wedge (\mathsf{Bl_h} \rightarrow \mathsf{Bl_h}[+1])\} \textit{ then } \mathsf{Bl_h} \textit{ else } \neg \mathsf{Bl_h}\end{aligned}$$

Equations for Step 2. The output variable $\mathsf{m_1}$ is used to mark the first cell of the row following the one containing the cell marked by $\mathsf{Bl_1}$. Formally, the equation for $\mathsf{m_1}$ ensures that whenever the input is a pseudo-tiling, then the stream for $\mathsf{m_1}$ is in 0^+1^+ and the suffix in 1^+ starts with a cell ("the cell marked by $\mathsf{m_1}$") which is the first cell numbered 0 following the cell marked by $\mathsf{Bl_1}$.

$$\begin{aligned}\mathsf{m_1} \;\; := \;\; & \textit{if } \mathsf{PTU} \rightarrow \{(\mathsf{last} \rightarrow \mathsf{m_1}) \wedge (\mathsf{first} \rightarrow \neg \mathsf{m_1}) \wedge (\neg \mathsf{m_1} \rightarrow \neg \mathsf{m_1}[-1|0]) \wedge \\ & (\mathsf{m_1} \rightarrow \mathsf{m_1}[+1]) \wedge ((\mathsf{Bl_1} \wedge \bigwedge_{i=1}^{m} \mathsf{b_i^-}[i]) \rightarrow \mathsf{m_1}[1]) \wedge \\ & ((\neg \mathsf{m_1} \wedge \mathsf{m_1}[1]) \rightarrow (\mathsf{Bl_1} \wedge \bigwedge_{i=1}^{m} \mathsf{b_i^-}[i]))\} \textit{ then } \mathsf{m_1} \textit{ else } \neg \mathsf{m_1}\end{aligned}$$

The output variable $\mathsf{m_2}$ is used to mark the first cell of the row containing the cell marked by $\mathsf{Bl_2}$. Formally, the equation for $\mathsf{m_2}$ ensures that whenever the input is a pseudo-tiling, then the stream for $\mathsf{m_2}$ is in 0^+1^+ and the prefix in 0^+ ends with the first bit of a cell ("the cell marked by $\mathsf{m_2}$") which is the last cell numbered 0 preceding the cell marked by $\mathsf{Bl_2}$ ($\mathsf{Bl_2}$ included).

$$\begin{aligned}\mathsf{m_2} \;:=\; & \textit{if } \mathsf{PTU} \rightarrow \{(\mathsf{last} \rightarrow \mathsf{m_2}) \wedge (\mathsf{first} \rightarrow \neg\mathsf{m_2}) \wedge (\neg\mathsf{m_2} \rightarrow \neg\mathsf{m_2}[-1|0]) \wedge \\ & \quad (\mathsf{m_2} \rightarrow \mathsf{m_2}[+1]) \wedge ((\neg\mathsf{Bl_2} \wedge \bigwedge_{i=1}^{m} \mathsf{b_i^-}[i]) \rightarrow \neg\mathsf{m_2}[+1|0]) \wedge \\ & \quad ((\neg\mathsf{m_2}[+1|1] \wedge \mathsf{m_2}[+2|0]) \rightarrow (\neg\mathsf{Bl_2} \wedge \bigwedge_{i=1}^{m} \mathsf{b_i^-}[i]))\} \textit{ then } \mathsf{m_2} \textit{ else } \neg\mathsf{m_2}\end{aligned}$$

Thus, the cells marked by $\mathsf{Bl_1}$ and $\mathsf{Bl_2}$ belong to two adjacent rows (with the $\mathsf{Bl_2}$'s row following the $\mathsf{Bl_1}$'s row) if and only if $\mathsf{m_1}$ and $\mathsf{m_2}$ mark the same cell if and only if the first 1-value position of $\mathsf{m_1}$ corresponds to the last 0-value position of $\mathsf{m_2}$. The equation for the output variable $\mathsf{test_2}$ ensures this last condition.

$$\begin{aligned}\mathsf{test_2} \;:=\; & \textit{if } \mathsf{PTU} \rightarrow \{(\neg\mathsf{m_2} \wedge \mathsf{m_2}[+1]) \leftrightarrow (\mathsf{m_1} \wedge \neg\mathsf{m_1}[-1|0])\} \\ & \quad \textit{then } \mathsf{test_2} \textit{ else } \neg\mathsf{test_2}\end{aligned}$$

Equations for Step 3. Finally, we define the equations for the output variables $\mathsf{ob_1}, \ldots, \mathsf{ob_m}$, $\mathsf{test_3}$, and od ($d \in \Delta$) ensuring that the cells marked by $\mathsf{Bl_1}$ and $\mathsf{Bl_2}$ have the same column number but different color on the shared edge. The equation for $\mathsf{ob_i}$ guarantees that there is a *uniform* stream for $\mathsf{ob_i}$ if and only if the i-th bits of the cells marked by $\mathsf{Bl_1}$ and $\mathsf{Bl_2}$ have the same value.

$$\begin{aligned}\mathsf{ob_i} \;:=\; & \textit{if } \mathsf{PTU} \rightarrow \bigwedge_{h=1}^{2} \{(\mathsf{Bl_h} \wedge \neg\mathsf{Bl_h}[-1|0]) \longrightarrow \\ & \quad \big((\mathsf{b_i^+}[i-1] \rightarrow \mathsf{ob_i}[i-1]) \wedge (\mathsf{b_i^-}[i-1] \rightarrow \neg\mathsf{ob_i}[i-1])\big)\} \textit{ then } \mathsf{ob_i} \textit{ else } \neg\mathsf{ob_i}\end{aligned}$$

For each $d \in \Delta$, the equation for od ensures that there is a uniform stream for od if and only if whenever the domino-type of the cell marked by $\mathsf{Bl_1}$ is d, then the cells marked by $\mathsf{Bl_1}$ and $\mathsf{Bl_2}$ have different color on the shared edge.

$$\begin{aligned}\mathsf{od} \;:=\; & \textit{if } \mathsf{PTU} \rightarrow \{((\mathsf{d}[m] \wedge \mathsf{Bl_1} \wedge \neg\mathsf{Bl_1}[-1|0]) \rightarrow \mathsf{od}[m]) \wedge \\ & \quad ((\bigvee_{d' \in \Delta : (d')_{down} = (d)_{up}} \mathsf{d}'[m] \wedge \mathsf{Bl_2} \wedge \neg\mathsf{Bl_2}[-1]) \rightarrow \neg\mathsf{od}[m])\} \textit{ then } \mathsf{od} \textit{ else } \neg\mathsf{od}\end{aligned}$$

Thus, the equation for variable $\mathsf{test_3}$ enforces the output streams for $\mathsf{ob_1}, \ldots, \mathsf{ob_m}, \mathsf{od}$ ($d \in \Delta$) to be uniform.

$$\begin{aligned}\mathsf{test_3} \;:=\; & \textit{if } \mathsf{PTU} \rightarrow \neg\mathsf{first} \rightarrow \big(\bigwedge_{i=1}^{m} (\mathsf{ob_i}[-1] \leftrightarrow \mathsf{ob_i}) \wedge \bigwedge_{d \in \Delta} (\mathsf{od}[-1] \leftrightarrow \mathsf{od})\big) \\ & \quad \textit{then } \mathsf{test_3} \textit{ else } \neg\mathsf{test_3}\end{aligned}$$

Hence, the unique inputs for which the constructed BSRV φ has no output stream valuation are those encoding tilings of $\mathcal{I}$. In other words, φ is over-defined precisely when there is a tiling of $\mathcal{I}$. Note that the size of φ is quadratic in the size of $\mathcal{I}$. This concludes the proof of EXPSPACE-hardness of checking over-definedness for BSRV.

EXPSPACE-hardness for generalized well-definedness of BSRV. First, we observe that the complement of the over-definedness problem can be reduced in linear time to the generalized well-definedness problem. Indeed, let φ be a BSRV specification and φ' be the BSRV specification obtained from φ by adding the equation $z = 0$, where z is a fresh output variable. Clearly, φ is not over-defined if and only if φ' is well-defined with respect to $\{z\}$. Hence, by the obtained EXPSPACE-completeness result for over-definedness of BSRV, EXPSPACE-hardness of checking generalized well-definedness of BSRV follows.

EXPSPACE-hardness for language universality, language inclusion, and language equivalence of BSRV. The results directly follow from EXPSPACE-completeness of checking over-definedness for BSRV and the facts that language universality for BSRV is the complement of the over-definedness problem, and language inclusion and language equivalence can be reduced in linear time to language universality.

EXPSPACE-hardness for semantic equivalence of BSRV. Let φ be a BSRV specification over X and Y, φ' be a BSRV specification over X and Y', and $Z \subseteq Y \cap Y'$. Note that if $Z = \emptyset$, then φ and φ' are equivalent with respect to Z if and only if $\mathcal{L}(\varphi) = \mathcal{L}(\varphi')$. Hence, by EXPSPACE-completeness of checking language equivalence for BSRV, the result follows.

The only missing parts to complete the proof of Theorem 7 are the PSPACE-hardness of checking under-definedness, well-definedness, and language emptiness for BSRV.

PSPACE-hardness for under-definedness and well-definedness of BSRV. We proceed by a polynomial-time reduction from a domino-tiling problem for grids with rows of polynomial length [32]. An instance $\mathcal{I} = \langle C, \Delta, m, d_{init}, d_{final} \rangle$ of this problem is defined as in the proof of EXPSPACE-hardness for over-definedness of BSRV. However, in this case a tiling of $\mathcal{I}$ is a tiling of $\mathcal{I}$ for the $n \times m$-grid for some $n > 0$ (i.e., the length of any row is m). The existence of a tiling for $\mathcal{I}$ is an PSPACE-complete problem [32]. We construct in polynomial time a BSRV φ such that the following holds:

- there exists a tiling of $\mathcal{I}$ if and only if φ is under-defined;
- φ is *not* under-defined if and only if φ is well-defined.

Hence, the result follows. Now, we illustrate the construction of φ. The set X of input variables of the specification φ is given by

$$X = \{\mathsf{d} \mid d \in \Delta\} \cup \{\mathsf{mk}\}$$

We associate to each domino-type $d \in \Delta$ an input variable d. The additional input variable mk is used as a separator between two adjacent rows. Thus, a tiling is encoded as a sequence of rows separated by the special marker, starting from the first row. Additionally, the first row is preceded by the special marker, and the last row is followed by the special marker. The specification φ has two output variables: PT and PTU. The output variable PT is used to check that the input encodes a tiling of $\mathcal{I}$: in particular, PT assumes the value 1 everywhere if and only if the input streams encode a tiling. Formally, the equation for variable PT is as follows.

$$\begin{aligned}
\mathsf{PT} := \quad & \underbrace{\bigvee_{\mathsf{x}\in X} (\mathsf{x} \wedge \bigwedge_{\mathsf{x}'\in X\setminus\{\mathsf{x}\}} \neg\mathsf{x}')}_{\text{exactly one input variable has value 1}} \quad \wedge \\
& \underbrace{\big(\mathsf{first} \rightarrow (\mathsf{mk} \wedge \mathsf{d}_{init}[+1|0])\big)}_{\text{the input starts with a } d_{init}\text{-cell}} \;\wedge\; \underbrace{\big(\mathsf{last} \rightarrow (\mathsf{mk} \wedge \mathsf{d}_{final}[-1|0])\big)}_{\text{the input ends with a } d_{final}\text{-cell}} \;\wedge \\
& \underbrace{\Big((\mathsf{mk} \wedge \neg\mathsf{last}) \longrightarrow (\mathsf{mk}[m+1|0] \wedge \bigwedge_{i=1}^{m} \neg\mathsf{mk}[i|1])\Big)}_{\text{adjacent rows are separated by the marker}} \;\wedge \\
& \underbrace{\bigwedge_{d\in\Delta} \Big(\mathsf{d} \rightarrow (\mathsf{mk}[+1|0] \vee \bigvee_{d'\in\Delta:(d')_{left}=(d)_{right}} \mathsf{d}'[+1|0])\Big)}_{\text{two adjacent cells in a row have the same color on the shared edge}} \;\wedge \\
& \underbrace{\bigwedge_{d\in\Delta} \Big(\mathsf{d} \rightarrow \bigvee_{d'\in\Delta:(d')_{down}=(d)_{up}} \mathsf{d}'[m+1|1]\Big)}_{\text{two adjacent cells in a column have the same color on the shared edge}}
\end{aligned}$$

Finally, the equation for the *uniform* output variable PTU is as follows.

$$\mathsf{PTU} \;:=\; \textit{if } (\neg\mathsf{first} \rightarrow (\mathsf{PTU}[-1|1] \leftrightarrow \mathsf{PTU})) \wedge (\neg\mathsf{PT} \rightarrow \neg\mathsf{PTU}) \textit{ then } \mathsf{PTU} \textit{ else } \neg\mathsf{PTU}$$

Note that if the input does *not* encode a tiling (i.e., for some position, PT assumes the value 0), then the uniform value of PTU is 0. Otherwise, the uniform value of PTU may be 0 or 1. Since for each input, the stream valuation for the other output variable PT is uniquely determined, it follows that there is a tiling for $\mathcal{I}$ iff φ is under-defined. Moreover, since for each input, there is some output stream valuation, it follows that φ is *not* under-defined if and only if φ is well-defined. Note that the size of φ is quadratic in the size of $\mathcal{I}$. Hence, the result follows.

PSPACE-hardness for language emptiness of BSRV. We modify the polynomial-time reduction given in the proof of PSPACE-hardness of checking under-definedness for BSRV as follows: the equation for the output variable PTU of the BSRV φ is updated as follows:

$$\mathsf{PTU} \quad := \quad \textit{if } (1 \leftrightarrow \mathsf{PTU}) \wedge (\neg\mathsf{PT} \rightarrow \neg\mathsf{PTU}) \textit{ then } \mathsf{PTU} \textit{ else } \neg\mathsf{PTU}$$

Hence, PTU is a uniform output variable whose uniform value is always 1. Moreover, the output stream for PTU is defined iff the output stream for PT is in 1^+. Additionally, the construction in the proof of PSPACE-hardness of checking under-definedness for BSRV ensures that for each input, the output stream for the output variable PT is uniquely determined, and PT assumes the value 1 everywhere if and only if the input streams encode a tiling. Hence, the updated construction is a polynomial-time reduction from a PSPACE-complete problem to language emptiness for BSRV.

6. Conclusion

In this paper, we have studied some fundamental 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. We only show here that this problems 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 [23]—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-definedness in tandem with a monitor that *detects* non-well-definedness (and hence, the incorrectness of the first monitor).

References

[1] M. Leucker, C. Schallhart, A brief account of runtime verification, The Journal of Logic and Algebraic Programming 78 (5) (2009) 293–303.

[2] K. Havelund, A. Goldberg, Verify your runs, in: Proc. of VSTTE'05, LNCS 4171, Springer, 2005, pp. 374–383.

[3] Z. Manna, A. Pnueli, Temporal Verification of Reactive Systems: Safety, Springer, New York, 1995.

[4] K. Havelund, G. Roşu, Synthesizing monitors for safety properties, in: Proc. of TACAS'02, LNCS 2280, Springer, 2002, pp. 342–356.

[5] C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, D. V. Campenhout, Reasoning with temporal logic on truncated paths, in: Proc. of CAV'03, Vol. 2725 of LNCS 2725, Springer, 2003, pp. 27–39.

[6] A. Bauer, M. Leucker, C. Schallhart, Runtime verification for LTL and TLTL, ACM Transactions on Software Engineering and Methodology 20 (4) (2011) 14.

[7] K. Sen, G. Roşu, Generating optimal monitors for extended regular expressions, ENTCS 89 (2) (2003) 226–245.

[8] H. Barringer, A. Goldberg, K. Havelund, K. Sen, Rule-based runtime verification, in: Proc. of VMCAI'04, LNCS 2937, Springer, 2004, pp. 44–57.

[9] G. Roşu, K. Havelund, Rewriting-based techniques for runtime verification., Automated Software Engineering 12 (2) (2005) 151–197.

[10] B. D'Angelo, S. Sankaranarayanan, C. Sánchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, Z. Manna, LOLA: Runtime monitoring of synchronous systems, in: Proc. of TIME'05, IEEE CS Press, 2005, pp. 166–174.

[11] A. Pnueli, A. Zaks, PSL model checking and run-time verification via testers, in: Proc. of FM'06, LNCS 4085, Springer, 2006, pp. 573–586.

[12] A. Donzé, O. Maler, E. Bartocci, D. Nickovic, R. Grosu, S. A. Smolka, On temporal logic and signal processing, in: Proc. of ATVA'12, Vol. 7561 of LNCS, Springer, 2012, pp. 92–106.

[13] B. Finkbeiner, S. Sankaranarayanan, H. B. Sipma, Collecting statistics over runtime executions, ENTCS 70 (4) (2002) 36–54.

[14] A. Bauer, R. Gore, A. Tiu, A first-order policy language for history-based transaction monitoring, in: Proc. of ICTAC'09, LNCS 5684, Springer, 2009, pp. 96–111.

[15] D. Basin, M. Harvan, F. Klaedtke, E. Zălinescu, MONPOLY: Monitoring usage-control policies, in: Proc. of RV'12, LNCS 7687, Springer, 2012.

[16] D. Basin, F. Klaedtke, S. Müller, Policy monitoring in first-order temporal logic, in: Proc. of CAV'10, LNCS 6174, Springer, 2010, pp. 1–18.

[17] P. Caspi, M. Pouzet, Synchronous Kahn Networks, in: Proc. of ICFP'96, ACM Press, 1996, pp. 226–238.

[18] G. Berry, Proof, language, and interaction: essays in honour of Robin Milner, MIT Press, 2000, Ch. The foundations of Esterel, pp. 425–454.

[19] N. Halbwachs, P. Caspi, D. Pilaud, J. Plaice, Lustre: a declarative language for programming synchronous systems, in: Proc. of POPL'87, ACM Press, 1987, pp. 178–188.

[20] T. Gautier, P. Le Guernic, L. Besnard, SIGNAL: A declarative language for synchronous programming of real-time systems, in: Proc. of FPCA'87, LNCS 274, Springer, 1987, pp. 257–277.

[21] L. Pike, A. Goodloe, R. Morisset, S. Niller, Copilot: A hard real-time runtime monitor, in: Proc. of RV'10, LNCS 6418, Springer, 2010.

[22] A. E. Goodloe, L. Pike, Monitoring distributed real-time systems: A survey and future directions, Tech. rep., NASA Langley Research Center (2010).

[23] M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, N. Bjørner, Symbolic finite state transducers: algorithms and applications., in: Proc. of POPL'12, ACM, 2012, pp. 137–150.

[24] L. DAntoni, M. Veanes, Extended symbolic finite automata and transducers, Formal Methods in System Design 47 (1) (2015) 93–119.

[25] R. Alur, L. D'Antoni, M. Raghothaman, DReX: A declarative language for efficiently evaluating regular string transformations, in: Proc. of POPL'15, ACM, 2015, pp. 125–137.

[26] L. Pike, N. Wegmann, S. Niller, A. Goodloe, Copilot: Monitoring embedded systems, Tech. rep., NASA, NASA/CR2012-217329 (2012).

[27] Digital design and computer organization, CRC Press, 2004.

[28] P. Wolper, Temporal logic can be more expressive, Information and Control 56 (1983) 72–99.

[29] F. Laroussinie, N. Markey, P. Schnoebelen, Temporal logic with forgettable past, in: Proc. of LICS'02, IEEE CS Press, 2002, pp. 383–392.

[30] R. E. Stearns, H. B. Hunt, On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata, SIAM J. Comput. 14 (3) (1985) 598–611.

[31] A. R. Meyer, L. J. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, in: Proc. of FOCS'72, IEEE CS Press, 1972, pp. 125–129.

[32] D. Harel, The spirit of Computing, 2nd Edition, Addison-Wesley, 1992.