Better Space Bounds for Future-Looking Stream Monitors

David M. Cerna

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria

Wolfgang Schreiner

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria

Temur Kutsia

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria

Abstract

The LogicGuard specification language for the runtime monitoring of event steams represents monitors as predicate logic formulas of a certain kind. In this paper we derive upper bounds for the *runtime representation size* of a LogicGuard monitor specification under the assumption that in these formulas quantifiers can be arbitrarily nested, but in the body of each quantifier there is at most one quantifier. Such monitors can be described by *quantifier vectors*. We provide an analytic expression for the runtime representation size of an important class of the monitors which we represent by *standard evaluation vectors*. Based on this optimal solution, we provide a method for computing the runtime representation size of a monitor, which proceeds by manipulation of the intervals of quantifier bounds. We show that this method gives a good approximation of the optimal solution for the general class and always outperforms results for the runtime representation size presented earlier.

Key words: Space complexity, Predicate logic, Runtime verification

⋆ LogicGuard II: The Optimized Checking of Time-Quantified Logic Formulas with Applications in Computer Security is sponsored by the FFG BRIDGE program, project No. 846003.

Email addresses: David.Cerna@risc.jku.at (David M. Cerna), Wolfgang.Schreiner@risc.jku.at (Wolfgang Schreiner), Temur.Kutsia@risc.jku.at (Temur Kutsia).

URLs: http://www.risc.jku.at/home/dcerna (David M. Cerna),

1. Introduction

The overall goal of runtime verification is to check at runtime whether the execution of a system satisfies a formal specification. This specification must take an operational form, which at runtime monitors the system and, in case of an error, reports a violation of the specification. The execution of the system may be abstracted to a stream of events (potentially infinite), such that the specification describes properties of the stream which the operational form monitors. We describe the efficiency of a stream specification in terms of the memory used during execution. We refer to this measure as *runtime representation size*.

In particular, we investigate the space complexity of the LogicGuard stream monitor specification language (LogicGuard II, 2015; Schreiner et al., 2015), which was developed in an industrial collaboration for the runtime monitoring of networks for security violations. LogicGuard represents an alternative to the commonly used language of linear temporal logic (LTL) (Maler et al., 2005; Armoni et al., 2002; Banieqbal and Barringer, 1987; Rosu and Bensalem, 2006), from which efficient stream monitors can be generated but in which, due to its limited expressiveness, it can be difficult to formulate more complex properties of interest.

The expressive power of the LogicGuard language is the result of the language being based on a larger fragment of predicate logic than LTL, i.e. it encompasses limited set construction, arithmetic and computation. In particular, the LogicGuard language allows the construction and monitoring of new streams from external streams in order to raise the level of abstraction of the specification.

Though, a downside to allowing very expressive elements into the language is that efficiency is lost for monitoring of a large portion of the possible specifications. Thus, it is an important issue to find efficient ways to describe properties of interest within the LogicGuard specification. In an effort to find efficient fragments of the language, a static analysis was developed to determine whether a specification gives rise to a monitor that is able to operate with only a finite set of past messages in memory. This static analysis was shown to be sound (Kutsia and Schreiner, 2014) and in the end resulted in a "history pruning" optimization. The proof of soundness was carried out for the same simplified core language we will discuss in this paper. This choice was made because a formal operational semantics could be easily devised for this core language.

In (Cerna, 2015b; Cerna et al., 2016b), a complementary analysis was carried out to determine the space requirements of the "future" looking parts of the operational form of the monitors. Essentially, in that work a method was developed to analyse the number of quantified formula instances which have to be preserved in memory during runtime, because, from the observations made so far, their truth values cannot be determined. Together with the the number of past messages preserved in memory, these measures determine the runtime representation size of the monitor as well as the time required for processing the monitor's operational form. In (Cerna, 2015b), we analysed both the cases when the number of instances kept in memory is infinite and finite. In (Cerna et al., 2016b), we have given upper bounds for the number of instances in the case of finite memory usage. In this work, we aim to more precisely capture this bound. For a few restricted, though important, formula classes we where able to derive precise bounds.

`http://www.risc.jku.at/home/schreine` (Wolfgang Schreiner),
`http://www.risc.jku.at/home/tkutsia` (Temur Kutsia).

Theses precise bounds can be used to upper bound a larger class of formula. These bounds are a non-trivial improvements over the bounds presented in (Cerna et al., 2016b) as it will be shown in Section 4. Essentially, the results presented here, though based on (Cerna et al., 2016b), are novel.

Comparatively, the core language of the LogicGuard framework has much in common with Monadic First-Order Logic (MFO), see (McNaughton and Papert, 1971). It is well known that LTL (Schnoebelen, 2003b) formulas are translatable into MFO formulas, which captures the class of star-free languages. The full language, on the other hand is closely related to Monadic Second-Order Logic (MSO) (Büchi, 1960), which captures the class of omega-regular languages, and is thus more expressive than LTL. Most space complexity results with respect to MFO and MSO use as a measure the size of the non-deterministic Büchi automaton that accepts the language of a formula. For MFO, the automaton size is in the worst case exponential with respect to the formula size (Vardi and Wolper, 1986). When a runtime verification method relies on automata-based model checking where the automaton is indeed constructed, this complexity result is applicable. Moreover, the previous result considers non-deterministic automata and in practice determinisation must be performed resulting in a second exponential blow-up. Complexity-wise, MSO fairs far worse in that the size of the accepting automaton is in general non-elementary with respect to the formula size (Frick and Grohe, 2004). This makes the use of MSO as a specification language seem quite impractical.

Even in the case of MFO, a doubly exponential sized automaton is still impractical for runtime verification, such that other subclasses of these logics are considered. The hardware design language PSL (IEEE, 2007) which is based on LTL defines a "simple subclass" that restricts the general use of disjunction in a specification to avoid exponential blowup. In (Kupferman et al., 2006), the class of "locally checkable" properties (a subclass of the "locally testable" properties introduced in (McNaughton and Papert, 1971)) is defined, where a word satisfies a property, if every k-length subword of the word (for some $k \in \mathbb{N}$) satisfies this property; such properties can be recognized by deterministic automata whose number of states is exponential in k but independent of the formula size. In (Finkbeiner and Kuhtz, 2009) a procedure for synthesizing monitor circuits from LTL specifications is defined that restricts the exponential blow-up to those parts of a formula that involve unbounded-future operators.

Another related set of works, but only in a dual sense, is the study of the computational complexity of *Interval Temporal Logic* (ITL) (Allen, 1983; Halpern and Shoham, 1991). Most of the analysis of ITL is performed with a focus on the difficulty of model checking formulas and the size of the Kripke structures corresponding to the given fragment of ITL, thus again, complexity is defined by automata size. However, for some severely restricted classes, space complexity results have been devised in terms of the smallest satisfying path in the Kripke Structure (Bozzelli et al., 2016), as well as results concerning the overall memory use for model checking certain fragments (Molinari et al., 2015). For some of these restricted classes it has been shown that there is a relatively small, hard kernel (the size of which is logarithmic in terms of input size) which makes the model checking procedure NP-hard (Gottlob, 1995; Schnoebelen, 2003a). We do not delve into the relationship of these results to our specification in this paper, however, we do borrow some of the assumptions from these restricted classes. For example, there is an interval temporal modality which enforces a specific starting time. This modality is integral in

separating the hard classes from the simple ones. In this work all quantifiers will have the same lower bound. This greatly simplifies the analysis and has led to very useful results.

Unlike these investigations, we do not consider the translation of formulas to automata and do not use automata sizes as the space complexity measure. Rather we base our investigations directly on an operational semantics of formula evaluation which exhibits the formula instances that are kept in memory during each evaluation step. In (Cerna et al., 2016b) we have investigated the number of instances kept in memory for a fragment of the core language by abstracting the operational semantics into a simple rewriting system and then applying complexity results for this rewrite system recursively to the formula structure. This method allows for complexity results in terms of concrete complexity functions, not just asymptotic bounds. However, in that work the resulting functions suffered from overestimation (Cerna et al., 2016b). In this paper, at least for a certain class of formulas, the results are precise. For more details concerning the work presented here (in particular proofs), see (Cerna, 2015a).

The rest of this paper is structured as follows: in Section 2, we present the core of the LogicGuard specification language and its operational semantics. In Section 3, we introduce the concept of evaluation vectors, as well as an abstraction of the operational semantics to evaluate them. In Section 4, we provide an optimal analytic solution for the space requirements of both uniform and standard evaluation vectors; we also introduce a naive solution which is more accurate than previous work and is computationally efficient. In Section 5, we conclude by judging the presented analysis and outlining a few open problems which we would like to address in future work.

2. Core Language

First we give a short introduction to the LogicGuard specification language and then introduce the core language and its operation semantics.

2.1. LogicGuard Specification Language

The LogicGuard language (Schreiner et al., 2015) for monitoring event streams allows, for example, the derivation of a higher level stream (representing e.g. a sequence of messages transmitted by the datagrams) from a lower level input stream (representing e.g. a sequence of TCP/IP datagrams). Such a stream is processed by a monitor for a particular property (e.g. that every message is within a certain time bound followed by another message whose value is related in a particular way to the value of the first one). A specification of this kind has the following form:

```
type tcp; type message; ...
stream<tcp> IP;
stream<message> S = stream<IP> x satisfying start(@x) :
  value[seq,@x,combine]<IP> y
      with x < _ satisfying same(@x,@y) until end(@y) :
    @y ;
monitor<S> M = monitor<S> x satisfying trigger(@x) :
  exists<S> y with x < _ <=# x+T:
    match(@x,@y);
```

After the declaration of types `tcp` and `message` and external functions and predicates operating on objects of these types, a stream `IP` of TCP/IP datagrams is declared that is connected by the runtime system to the network interface. From this stream, a "virtual" stream `S` of "messages" is derived; each message is created by sequentially combining every datagram at position `x` on `IP` (whose value is denoted by `@x`) that satisfies a predicate `start` by application of a function `combine` with every subsequent datagram at position `y` that is related to the first one by a predicate `same` until a termination condition `end` is satisfied. For this purpose the construct value $[\text{seq}, f, b]\, x$ with $c\ t$. It computes the value $f(f(\cdots f(b, x_1), x_2) \cdots, x_n)$, where $x_1, \cdots x_n$ are values of t when the variable x is assigned all values determined by constraint c. The stream `S` is monitored by a monitor `M` that checks whether for every message on `S` that satisfies a `trigger` predicate within `T` time, a partner message appears that fits with the first message according to some `match` predicate.

To support a formal analysis, in (Kutsia and Schreiner, 2014) a core version of the LogicGuard language (see Figure 1) was defined and given a formal operational semantics. This core language has been subsequently used to analyse the complexity of monitoring and to derive the results presented in this paper. The analysis was also implemented in the LogicGuard system by translating specifications from the full language to the core language such that the analysis of the translated specification also predicts the complexity of monitoring the original specification (the translation is not semantics-preserving but generates a specification for which monitoring is at least as complex as the monitoring of the initial specification).

The syntax of this core language is depicted on the lefthand side of Figure 1 where the typed variables $M, F, \ldots$ denote elements of the syntactic domains $\mathbb{M}, \mathbb{F}, \ldots$ of monitors, formulas, etc. A monitor M has the form $\forall_{0 \le V} : F$ for some variable V and formula F; it processes an infinite stream of truth values $\top$ (true) or $\bot$ (false) by evaluating F for $V = 0, V = 1, \ldots$ The predicate $@V$ denotes the value in the stream at position V, $\neg F$ denotes the negation of F, $F_1 \ \wedge \ F_2$ denotes parallel conjunction (both F_1 and F_2 are evaluated simultaneously), F_1 & F_2 denotes sequential conjunction (the evaluation of F_2 is delayed until the value of F_1 becomes available), $\forall_{V \in [B_1, B_2]} : F$ denotes universal quantification over the interval $[B_1, B_2]$.

A monitor $M \in \mathbb{M}$ is translated by the function $T : \mathbb{M} \to \mathcal{M}$ defined at the bottom of Figure 1 into its runtime representation $m = T(M) \in \mathcal{M}$ whose structure is depicted on the right-hand side; here the typed variables $m, f, \ldots$ denote elements of the runtime domains $\mathcal{M}, \mathcal{F}, \ldots$, i.e., the runtime representations of $M, F, \ldots$ Over the domain $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$ arithmetic operations are interpreted in the usual way, i.e., the operator $-$ is interpreted as truncated subtraction and for every $n \in \mathbb{N}$ we have $\infty \pm n = \infty$ (we assume $0 \in \mathbb{N}$). The notions $\mathbb{P}(S)$ and $A \to^{\text{part.}} B$ denote the powerset of S and the set of partial mappings from A to B, respectively. A context c consists of a pair of partial functions that assign to every variable its position and the truth value that the stream holds at that position, respectively. Though mapping variable to the the position seems redundant, this mapping is necessary for soundness of the history pruning optimization introduced in (Kutsia and Schreiner, 2014).

During the execution of a monitor M, the set I in its runtime representation $T(M) = \forall^I_{0 \le V} : f$ contains those instances of its body F which could not yet be evaluated to $\top$ or $\bot$; each such instance is represented by a tuple (p, f, c) where p is the position assigned to V, f is the (current) runtime representation of F, and c represents the context to be

$$
\begin{array}{lll}
M ::= & \forall_{0\le V} : F. & m ::= \quad \forall^{\mathbb{P}(\mathbb{N}\times f\times c)}_{0\le V} : f \\
F ::= & @V \mid \neg F \mid F \ \wedge \ F \mid F \ \& \ F & f ::= \quad \mathsf{d}(\top) \mid \mathsf{d}(\bot) \mid \mathsf{n}(g) \\
\mid & \forall_{V\in[B,B]} : F. & g ::= \quad @V \mid \neg f \mid f \ \wedge \ f \mid f \ \& \ f \\
B ::= & 0 \mid \infty \mid V \mid B \pm N. & \mid \quad \forall_{V\in[b,b]} : f \mid \forall_{V\in[\mathbb{N}^\infty,\mathbb{N}^\infty]} : f \\
V ::= & x \mid y \mid z \mid \ldots & \mid \quad \forall^{\mathbb{P}(\mathbb{N}\times f\times c)}_{V\le\mathbb{N}^\infty} : f \\
N ::= & 0 \mid 1 \mid 2 \mid \ldots & b ::= \quad c \to \mathbb{N}^\infty \\
 & & c ::= \quad (V \to^{\text{part.}} \mathbb{N}) \times (V \to^{\text{part.}} \{\top,\bot\})
\end{array}
$$

$$
\begin{aligned}
T(\forall_{0\le V} : F) &:= \forall^{\emptyset}_{0\le V} : T^{\mathrm{F}}(F) \\
T^{\mathrm{F}}(@V) &:= \mathsf{n}(@V) \\
T^{\mathrm{F}}(\neg F) &:= \mathsf{n}(\neg T^{\mathrm{F}}(F)) \\
T^{\mathrm{F}}(F_1 \ \wedge \ F_2) &:= \mathsf{n}(T^{\mathrm{F}}(F_1) \wedge T^{\mathrm{F}}(F_2)) \\
T^{\mathrm{F}}(F_1 \ \& \ F_2) &:= \mathsf{n}(T^{\mathrm{F}}(F_1) \ \& \ T^{\mathrm{F}}(F_2)) \\
T^{\mathrm{F}}(\forall_{V\in[B_1,B_2]} : F) &:= \forall_{V\in[T^B(B_1),T^B(B_2)]} : T^{\mathrm{F}}(F)
\end{aligned}
$$

$$
\begin{aligned}
T^{\mathrm{B}}(0) &:= \lambda c.\ 0 \\
T^{\mathrm{B}}(\infty) &:= \lambda c.\ \infty \\
T^{\mathrm{B}}(V) &:= \lambda c.\ c.1(V) \\
T^{\mathrm{B}}(B \pm N) &:= \lambda c.\ T^{\mathrm{B}}(B)(c) \pm N
\end{aligned}
$$

Fig. 1. The core language: syntax, runtime representation, translation.

used for the evaluation of f. A runtime representation f can be a tagged value $\mathsf{n}(g)$ where g represents the runtime representation of the formula to be evaluated in the **n**ext step; when the evaluation has completed, its value becomes $\mathsf{d}(t)$ when the evaluation is **d**one, the truth value t represents the result.

2.2. Operational Semantics

The evaluation of a monitor's runtime representation is formally defined by a small-step operational semantics with a 6-ary transition relation

$$\to \subseteq \mathcal{M} \times \mathbb{N} \times \{\top,\bot\}^\omega \times \{\top,\bot\} \times \mathbb{P}(\mathbb{N}) \times \mathcal{M}.$$

A transition is of the form $m \to_{p,s,v,R} m'$ where m is the runtime representation of the monitor prior to the transition, m' is its representation after the transition, p is the stream position of the next message value v to be processed, s denotes the sequence of p messages that have previously been processed, and R denotes the set of those positions which are reported by the transition relation to make the monitor body false. The monitor thus processes a stream $\langle v_0, v_1, \ldots\rangle$ by a sequence of transitions

$$\left(\forall^{I_0}_{0\le x} : f\right) \to_{0,s_0,v_0,R_0} \left(\forall^{I_1}_{0\le x} : f\right) \to_{1,s_1,v_1,R_1} \left(\forall^{I_2}_{0\le x} : f\right) \to \cdots$$

where $s_p = \langle v_0, \ldots, v_{p-1}\rangle$. Each set I_p contains those instances of the monitor which, by the p messages processed so far, could not be evaluated to a truth value yet and each

Atomic Formulas		
#	Transition	Constraints
A1	$\mathsf{n}(@y) \rightarrow \mathsf{d}(c.2(y)))$	$y \in dom(c.2)$
A2	$\mathsf{n}(@y) \rightarrow \mathsf{d}(\bot)$	$y \notin dom(c.2)$
Negation		
N1	$\mathsf{n}(\neg f) \rightarrow \mathsf{n}(\neg \mathsf{n}(f'))$	$f \rightarrow \mathsf{n}(f')$
N2	$\mathsf{n}(\neg f) \rightarrow \mathsf{d}(\bot)$	$f \rightarrow \mathsf{d}(\top)$
N3	$\mathsf{n}(\neg f) \rightarrow \mathsf{d}(\top)$	$f \rightarrow \mathsf{d}(\bot)$
Sequential conjunction		
C1	$\mathsf{n}(f_1 \ \& \ f_2) \rightarrow \mathsf{n}(\mathsf{n}(f_1') \ \& \ f_2)$	$f_1 \rightarrow \mathsf{n}(f_1')$
C2	$\mathsf{n}(f_1 \ \& \ f_2) \rightarrow \mathsf{d}(\bot)$	$f_1 \rightarrow \mathsf{d}(\bot)$
C3	$\mathsf{n}(f_1 \ \& \ f_2) \rightarrow \mathsf{n}(f_2')$	$f_1 \rightarrow \mathsf{d}(\top),\ f_2 \rightarrow \mathsf{n}(f_2')$
Parallel conjunction		
P1	$\mathsf{n}(f_1 \wedge f_2) \rightarrow \mathsf{n}(\mathsf{n}(f_1') \wedge \mathsf{n}(f_2'))$	$f_1 \rightarrow \mathsf{n}(f_1'),\ \ f_2 \rightarrow \mathsf{n}(f_2')$
P2	$\mathsf{n}(f_1 \wedge f_2) \rightarrow \mathsf{n}(f_1')$	$f_1 \rightarrow \mathsf{n}(f_1'),\ \ f_2 \rightarrow \mathsf{d}(\top)$
P3	$\mathsf{n}(f_1 \wedge f_2) \rightarrow \mathsf{n}(f_2')$	$f_2 \rightarrow \mathsf{n}(f_2'),\ \ f_1 \rightarrow \mathsf{d}(\top)$
P4	$\mathsf{n}(f_1 \wedge f_2) \rightarrow \mathsf{d}(\bot)$	$f_2 \rightarrow \mathsf{n}(f_2'),\ \ f_1 \rightarrow \mathsf{d}(\bot)$
P5	$\mathsf{n}(f_1 \wedge f_2) \rightarrow \mathsf{d}(\bot)$	$f_1 \rightarrow \mathsf{n}(f_1'),\ \ f_2 \rightarrow \mathsf{d}(\bot)$
Quantification		
Q1	$\forall_{y \in [b_1, b_2]} : f \rightarrow \mathsf{d}(\top)$	$p_1 = b_1(c)\ ,\ p_2 = b_2(c)\ ,\ p_1 > p_2 \vee p_1 = \infty$
Q2	$\forall_{y \in [b_1, b_2]} : f \rightarrow F'$	$p_1 = b_1(c)\ ,\ p_2 = b_2(c), p_1 \neq \infty\ ,\ p_1 \leq p_2,$ $\mathsf{n}(\forall_{y \in [p_1, p_2]} : f) \rightarrow F'$
Q3	$\mathsf{n}(\forall_{y \in [p_1, p_2]} : f) \rightarrow \mathsf{n}(\forall_{y \in [p_1, p_2]} : f)$	$p < p_1$
Q4	$\mathsf{n}(\forall_{y \in [p_1, p_2]} : f) \rightarrow F'$	$p_1 \leq p,\ \mathsf{n}(\forall^{I_0}_{y \leq p_2} : f) \rightarrow F'$
Q5	$\mathsf{n}(\forall^{I}_{y \leq p_2} : f) \rightarrow \mathsf{d}(\bot)$	DF
Q6	$\mathsf{n}(\forall^{I}_{y \leq p_2} : f) \rightarrow \mathsf{d}(\top)$	$\neg DF, I'' = \emptyset, p_2 < p$
Q7	$\mathsf{n}(\forall^{I}_{y \leq p_2} : f) \rightarrow \mathsf{n}(\forall^{I''}_{y \leq p_2} : f)$	$\neg DF, (I'' \neq \emptyset \vee p \leq p_2)$

Fig. 2. The operational semantics of formula evaluation.

set R_p contains the positions of those instances that were reported to become false by transition p. In particular, we have

$$I_{p+1} = \{(t, \mathsf{n}(g), c) \in \mathcal{I} \mid \exists f \in \mathcal{F} : (t, f, c) \in I' \wedge \vdash f \rightarrow_{p,s_p,v_p,c} \mathsf{n}(g)\}$$
$$R_{p+1} = \{t \in \mathbb{N} \mid \exists f \in \mathcal{F}, c \in \mathcal{C} : (t, f, c) \in I' \wedge \vdash f \rightarrow_{p,s_p,v_p,c} \mathsf{d}(\bot)\}$$

where $I' = I_p \cup \{(p, f, ((V, p), (V, v_p)))\}$. The transition relation on monitors depends on a corresponding transition relation $f \rightarrow_{p,s,v,c} f'$ on formulas where c represents the context for the evaluation of f. In each step p of the monitor transition, a new instance of the monitor body F is added to set I_p, and all instances in that set are evaluated according to the formula transition relation. Note that each formula instance in that set contains the runtime representation of a quantified formula (otherwise, it could have been immediately evaluated) which in turn contains its own instance set; thus instance sets are nested up to a depth that corresponds to the quantification depth of the monitor.

Figure 2 shows an excerpt of the operational semantics of formula evaluation (a more thorough discussion can be found in (Kutsia and Schreiner, 2014)) where the transition arrow $\rightarrow$ is to be read as $\rightarrow_{p,s,v,c}$ and rules Q4–Q7 are based on the following definitions:

$$I_0 = \{(i, f, (c.1[V \mapsto i], c.2[V \mapsto s(i + p - |s|)])) \mid p_1 \leq i \leq \min\{p_2 + 1, p\}\}$$
$$I' = \begin{cases} I & \text{if } p_2 < p \\ I \cup (p, f, (c.1[V \mapsto p], c.2[V \mapsto v])) & \text{otherwise} \end{cases}$$
$$I'' = \{(t, \mathsf{n}(g), c) \in \mathcal{I}' \mid (t, f, c) \in I' \wedge \vdash f \rightarrow \mathsf{n}(g)\}$$
$$DF \equiv \exists t \in \mathbb{N}, f \in \mathcal{F}, c \in \mathcal{C} : (t, f, c) \in I' \wedge \vdash f \rightarrow \mathsf{d}(\bot)$$

Essentially, rules, Q1, Q2, and Q3 deal with quantifiers of which are either malformed, have an uninstantiated stream variable, or The lower end of the interval is greater than the current stream position. If the quantifier is correctly formed and the lower end of the interval is greater than or equal the current stream position we can move on to the application of Q4 which adds an instance set to the runtime representation of the Quantifier. The rules Q5 and Q6 check if any instance from the instance set evaluated to false or if every instance in the instance set evaluated to true and the stream position is beyond the upper bound of the quantifier. When neither Q5 or Q6 can be applied we apply Q7 upon transition to the next position in the stream. New instances are added to the instance set of the quantifier. We provide an example adapted from (Cerna et al., 2016b) concerning the application of these rules.

Example 1. Consider the monitor $M = \forall_{0 \leq x} : \forall_{y \in [x+1, x+2]} : @x \ \& \ @y$, which states that the current position of the stream is true as well as the next two future positions. We determine its runtime representation $m = T(M)$ as $m = \forall^{\emptyset}_{0 \leq x} : f$ with $f = \forall_{y \in [b_1, b_2]} : g$ for some b_1 and b_2 and $g = @x \ \& \ @y$. We evaluate m over the stream $\langle \top, \top, \bot, \ldots \rangle$. First consider the transition $(\forall^{\emptyset}_{0 \leq x} : f) \rightarrow_{0, \langle\rangle, \top, \emptyset} (\forall^{I_0}_{0 \leq x} : f)$. which generates the instance set

$$I^0 = \left\{(0, \mathsf{n}(\forall_{y \in [1,2]} : g), (\{(x, 0)\}, \{(x, \top)\}))\right\}.$$

Performing another step $(\forall^{I_0}_{0 \leq x} : f) \rightarrow_{1, \langle\top\rangle, \top, \emptyset} (\forall^{I_1}_{0 \leq x} : f)$ we get

$$I^1 = \begin{array}{l} \big\{(1, \mathsf{n}(\forall_{y \in [2,3]} : g), (\{(x, 1)\}, \{(x, \top)\})), \\ \quad (0, \mathsf{n}(\forall^{\emptyset}_{y \leq 2} : g), (\{(x, 0)\}, \{(x, \top)\}))\big\} \end{array}.$$

The instance set $\emptyset$ in the runtime representation of the formula is empty, because the body of the quantified formula is propositional and evaluates instantly. Notice that the new instance is the same as the instance in I^0 but the positions are shifted by 1. The next step is $(\forall^{I_1}_{0\leq x} : f) \rightarrow_{2,\langle\top,\top\rangle,\bot,\{0,1\}} (\forall^{I_2}_{0\leq x} : f)$ where

$$I^2 = \left\{(2, \mathsf{n}(\forall_{y\in[3,4]} : g), (\{(x, 2)\}, \{(x, \bot)\}))\right\}.$$

The first two instances evaluate at this point and both violate the specification, thus yielding the set $\{0, 1\}$ of violating positions of the monitor. Again, the remaining instance is shifted by one position.

Our goal is to determine for arbitrary sequences

$$\left(\forall^{I_0}_{0\leq x} : f\right) \rightarrow_{0,s_0,v_0,R_0} \left(\forall^{I_1}_{0\leq x} : f\right) \rightarrow_{1,s_1,v_1,R_1} \left(\forall^{I_2}_{0\leq x} : f\right) \rightarrow \cdots$$

arising for some monitor $M = \forall_{0\leq x} : f$ the maximum size of I_p plus for each $q = \mathsf{n}(\forall^{I}_{y\leq p_2} : f) \in IS^{f_i}$ the size of I. We formalize this problem in the following subsection.

2.3. Runtime Representation Size

Now we address the central problem of this work. Our goal is to determine the maximum size of the runtime representation of a monitor during its execution. For doing this we have to define the size of the runtime representation of monitors, formulas and formula instances.

Definition 1. We define the functions $c_{\mathrm{m}} : \mathcal{M} \rightarrow \mathbb{N}$, $c_{\mathrm{f}} : \mathcal{F} \rightarrow^{\text{part.}} \mathbb{N}$, $c_{\mathrm{g}} : \mathcal{G} \rightarrow \mathbb{N}$, and $c_{\mathrm{i}} : \mathcal{I} \rightarrow \mathbb{N}$ which denote the size of the runtime representation of a monitor respectively unevaluated formula (with and without tag) respectively formula instance:

$$\begin{aligned}
c_{\mathrm{m}}(\forall^{I}_{0\leq V} : f) &= \textstyle\sum_{g\in I} c_{\mathrm{i}}(g) & c_{\mathrm{f}}(\mathsf{n}(g)) &= c_{\mathrm{g}}(g)\\
c_{\mathrm{g}}(@V) &= 0 & c_{\mathrm{g}}(f_1 \ \wedge \ f_2) &= c_{\mathrm{f}}(f_1) + c_{\mathrm{f}}(f_2)\\
c_{\mathrm{g}}(\neg f) &= c_{\mathrm{f}}(f) & c_{\mathrm{g}}(f_1 \ \& \ f_2) &= c_{\mathrm{f}}(f_1) + c_{\mathrm{f}}(f_2)\\
c_{\mathrm{g}}(\forall_{V\in[b_1,b_2]} : f) &= 1 & c_{\mathrm{g}}(\forall^{I}_{V\leq p} : f) &= 1 + \textstyle\sum_{g\in I} c_{\mathrm{i}}(g)\\
c_{\mathrm{g}}(\forall_{V\in[p_1,p_2]} : f) &= 1 & c_{\mathrm{i}}((n, f, c)) &= c_{\mathrm{f}}(f)
\end{aligned}$$

Now we can define a relation which determines the maximum size of the runtime representation of a monitor encountered during its execution.

Definition 2. We define the relation $\multimap \subseteq \mathcal{M} \times \mathbb{N} \times \{\top, \bot\}^* \times \mathbb{N} \times \mathbb{N}$ inductively as follows:

$$\begin{aligned}
&M \multimap_{p,s,0} S' \leftrightarrow S' = c_{\mathrm{m}}(M)\\
&M \multimap_{p,s,(n+1)} S' \leftrightarrow\\
&\quad \left(\exists R.\ (M \rightarrow_{p,s,s(p),R} M') \wedge (M' \multimap_{p+1,s,n} S) \wedge S' = \max\left\{c_{\mathrm{m}}(M), S\right\}\right)
\end{aligned}$$

Since $\rightarrow$ is deterministic $M \multimap_{p,s,n} S$ uniquely determines S from M, p, s, and n. Essentially, $M \multimap_{p,s,n} S$ states that S is the maximum size of the representation of

monitor m during the execution of n transitions over the stream s starting at position p. Our goal is to compute/bound the value of S by a static analysis, i.e., without having to actually perform the transitions. Towards this goal, we introduce in Section 3 an abstraction of the operational semantics for our abstraction of monitor specifications, which we call *evaluation vectors*. We then formalize the connection between our analysis and the above relation in Theorem 44. The rules of our abstracted operational semantics found in Sec. 3.2 (Definition 29) correspond directly to the rules of Figure 2. The relationship is not one to one, but all semantic properties of quantifiers are preserved. Non-quantifier rules are not important for memory analysis, so they have been dropped. Instance introduction is handled by a separate mechanism which exploits the translational symmetry of different stream position.

2.4. Dominating Formula Transformation

A concept introduced in (Cerna et al., 2016b), *the Dominating Monitor/Formula*, allows us to restrict our analysis to quantified formulas whose variable intervals only depend on the outermost monitor variable, i.e. the size of every interval is the same for every value of the monitor variable.

Definition 3 (Dominating Monitor/Formula Transformation)**.** Let $\mathbb{A} = \mathbb{V} \rightarrow^{\text{part.}} \mathbb{N}$ be the domain of assignments that map variables to natural numbers. Then the *dominating monitor transformation* $D : \mathbb{M} \rightarrow \mathbb{M}$ respectively *formula transformation* $D' : \mathbb{F} \times \mathbb{A} \times \mathbb{A} \rightarrow \mathbb{F}$ are defined as follows:

$$\begin{aligned}
D(\forall_{0\leq V} : F) &= \forall_{0\leq V} : D'(F, [V \mapsto 0], [V \mapsto 0]) \\
D'(@V, a_l, a_h) &= @V \\
D'(\neg F, a_l, a_h) &= \neg D'(F, a_l, a_h) \\
D'(F_1 \ \& \ F_2, a_l, a_h) &= D'(F_1, a_l, a_h) \ \& \ D'(F_2, a_l, a_h) \\
D'(F_1 \ \wedge \ F_2, a_l, a_h) &= D'(F_1, a_l, a_h) \ \wedge \ D'(F_2, a_l, a_h) \\
D'(\forall_{V\in[B_1,B_2]} : F, a_l, a_h) &= \forall_{V\in[h_L(B_1),h_H(B_2)]} : D'(F, a'_l, a'_h)
\end{aligned}$$

In the last equation we have $a'_l = a_l[V \mapsto h_L(B_1)]$, $a'_h = a_h[V \mapsto h_H(B_2)]$, $h_L(B_1) = \min\{[\![B_1]\!]^{a_l}, [\![B_1]\!]^{a_h}\}$, $h_H(B_2) = \max\{[\![B_2]\!]^{a_l}, [\![B_2]\!]^{a_h}\}$ where $[\![B]\!]^a$ denotes the result n of the evaluation of bound expression B for assignment a; actually, if B contains the monitor variable x, the result shall be the expression $x + n$ (we omit the formal details, see example below).

Dominating monitors are used in the construction of *evaluation vectors* (see Definition 10).

Example 2. Consider the following monitor M:

$$\begin{aligned}
\forall_{0\leq x} : \forall_{y\in[x+1,x+5]} : ((\forall_{z\in[y,x+3]} : \neg@z \ \& \ @z) \ \& \ G(x, y)) \\
G(x, y) = \forall_{w\in[x+2,y+2]} : (\neg@y \ \& \ (\forall_{m\in[y,w]} : \neg@x \ \& \ @m))
\end{aligned}$$

The dominating form $D(M)$ of M is the following:

$$\begin{aligned}
\forall_{0\leq x} : \forall_{y\in[x+1,x+5]} : ((\forall_{z\in[x+1,x+3]} : \neg@z \ \& \ @z) \ \& \ G(x, y)) \\
G(x, y) = \forall_{w\in[x+2,x+7]} : (\neg@y \ \& \ (\forall_{m\in[x+1,x+7]} : \neg@x \ \& \ @m))
\end{aligned}$$

Notice that additional instances are needed for the evaluation of $D(M)$.

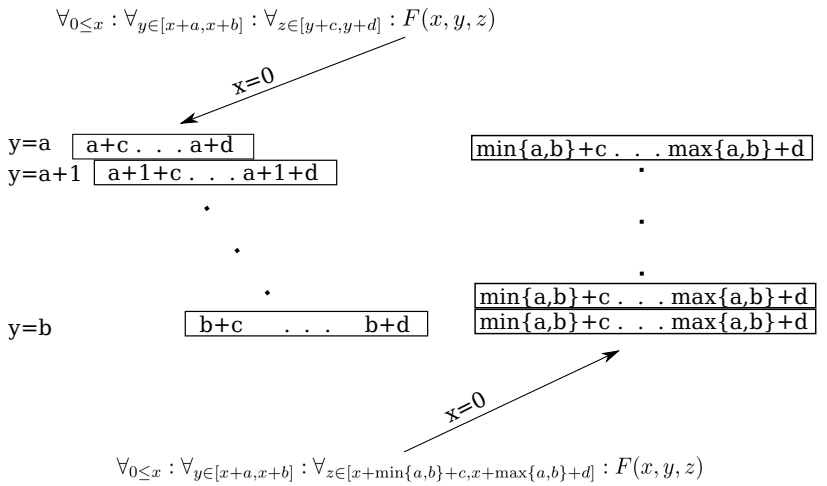

Fig. 3. Illustration of the dominating monitor transformation.

See Figure 3 for an illustration of the dominating monitor transformation. The left side of Figure 3 shows the intervals of the inner quantifier for each value of y in the outer quantifier interval. Notice that the values increase with respect to the value of y. The right-hand side of Figure 3 represents the interval structure of the dominating monitor. The values of the inner quantifier's interval are constant, invariant of the value of y. Essentially the future most position is fixed in the dominating monitor, where as, in the original monitor, the future most position is dependent on y.

The relationship, in terms of the maximum size of instance sets, between a monitor M and its dominating form $D(M)$ is summarized in the following theorem.

Theorem 4. *Let $M \in \mathbb{M}$. Then for all $p, n, S, S' \in \mathbb{N}$ and $s \in \{\top, \bot\}^\omega$ such that $T(M) \multimap_{p,s,n} S$ and $T(D(M)) \multimap_{p,s,n} S'$, we have $S \leq S'$.*

The correctness of this theorem follows from Definition 2 and 3. Clearly, if $M = D(M)$, i.e., the monitor is already in its dominating form, then we have $S = S'$.

3. Quantifier Vectors and their Evaluation

In (Cerna et al., 2016b) the logical structure of the monitor specifications was considerably abstracted. However, when dealing with quantifiers which are nested more than two levels deep, that abstraction becomes quite cumbersome. In this paper we thus introduce *quantifier trees*, as an alternative. In the new abstraction, quantifier free parts of the monitor specification are dropped entirely and only the structures which influence space complexity to a significant extent (future looking quantifiers) are preserved. For deriving analytic expressions for the most restricted monitors, we focus on a set of quantifier trees, namely *quantifier vectors*, and more specifically *evaluation vectors*, which only preserve a small part of the formula structure.

3.1. Quantifier and Evaluation Vectors

Definition 5 (Quantifier Trees)**.** A *quantifier tree* is inductively defined to be either $\emptyset$ or a tuple of the form (y, b_1, b_2, Q) where $y \in V$, $b_1, b_2 \in B$ and Q is a set of quantifier trees. Let $\mathbb{QT}$ be the set of all quantifier trees.

Definition 6 (Quantifier Tree Transformation)**.** We define the *quantifier tree transformation* $QT : \mathbb{M} \to \mathbb{QT}$, respectively $QT : \mathbb{F} \to \mathbb{QT}$, recursively as follows:

$$\begin{array}{llll} QT(\forall_{0\leq V} : F) & = (V, 0, 0, QT(F)) & QT(F\&G) & = QT(F) \cup QT(G) \\ QT(F \wedge G) & = QT(F) \cup QT(G) & QT(\neg F) & = QT(F) \\ QT(\forall_{V\in[B_1,B_2]} : F) & = (V, B_1, B_2, QT(F)) & QT(@V) & = \emptyset \end{array}$$

Notice that quantifier trees drop the distinction between sequential and parallel conjunction.

Definition 7 (Quantifier Tree Node)**.** Let $q = (y, b_1, b_2, Q) \in \mathbb{QT}$. Then n is a *quantifier tree node of* q if for some $b \in \{0, 1\}$, $n = [(y, b_1, b_2, b)]$.

Definition 8 (Quantifier Tree Path)**.** Let $q = (y_1, b_1^1, b_2^1, Q) \in \mathbb{QT}$. Then we define inductively p as a *quantifier tree path of* q, if $p = []$ or

$$p = \left[(y_1, b_1^1, b_2^1, b_1), (y_2, b_1^2, b_2^2, b_2), \cdots, (y_n, b_1^n, b_2^n, b_n)\right]$$

and for some $q' \in Q$, $p' = \left[(y_2, b_1^2, b_2^2, b_2), \cdots, (y_n, b_1^n, b_2^n, b_n)\right]$ is a quantifier tree path of q'. Let $\mathbb{QP}(q)$ denote the set of quantifier tree paths of a quantifier tree q.

Note, that if a quantifier tree path p has length 1 then it is also a quantifier tree node. The fourth position of a quantifier tree node is used during evaluation as defined in Section 3.2. We will work with a degenerate case of quantifier trees, namely, *quantifier vectors*.

Definition 9 (Quantifier Vectors)**.** A *quantifier vector* is inductively defined to be either $\emptyset$ or a tuple of the form (y, b_1, b_2, Q) where $y \in V$, $b_1, b_2 \in B$ and Q is a set of quantifier vectors s.t. $|Q| \leq 1$. Let $\mathbb{QV}$ be the set of all quantifier trees. We define *the length* $|v|$ *of a quantifier vector* v, as $|v| = 0$ if $v = \emptyset$, $|v| = 1$ if $v = (y, b_1, b_2, \emptyset)$, or $|v| = 1 + |q|$ if $v = (y, b_1, b_2, Q)$ and $q \in Q$.

From now on we will use the quantifier tree path representation of a quantifier vector being that the structure is more explicit. The following concept considers only monitors whose bounds are expressed in terms of the stream variable, i.e. dominating monitors (Definition 3). The subsequently derived space complexity results are optimal for specific dominating monitors and represent upper bounds for pre-transformation monitors.

Definition 10 (Evaluation Vector)**.** Let $M \in \mathbb{M}$ such that $QT(M) \in \mathbb{QV}$. Then an *evaluation vector* of M is a quantifier tree path $e \in \mathbb{QP}(QT(D(M)))$. Being that quantifier variables no longer play a role in interval evaluation we use the short hand $\left[(c_1^1, c_2^1, b_1), \cdots, (c_1^n, c_2^n, b_n)\right]$ for $\left[(y_2, x + c_1^1, x + b_2^1, b_1), \cdots, (y_n, x + c_1^n, x + c_2^n, b_n)\right]$. Let $\mathbb{EV}$ be the set of all evaluation vectors.

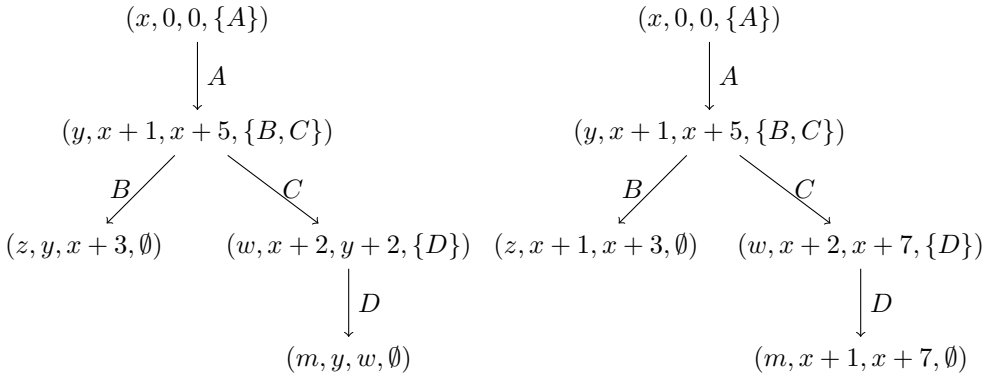

Fig. 4. Quantifier trees for the monitor of Example 2.

Definition 11. Let $v = \left[(c_1^1, c_2^1, b_1), \cdots, (c_1^n, c_2^n, b_n)\right] \in \mathbb{EV}$. We define $v(i) = (c_1^i, c_2^i, b_i)$, $v(i,1) = c_1^i$, $v(i,2) = c_2^i$, and $v(i,3) = b_i$. When $v(i,3) = 0$ for all $1 \leq i \leq |v|$ we call v *proper*.

Definition 12. Given $v = \left[(c_1^1, c_2^1, b_1), \cdots, (c_1^n, c_2^n, b_n)\right] \in \mathbb{EV}$, the *state string of* v, $ss(v)$, is the binary word $b_1 b_2 \cdots b_n$. By $ss(v,i)$ for $i \in \{0,1\}$ we denote the number of positions in state strings of v which have value i.

Example 3. Let us consider the monitor specification M from Example 2. The $QT(M)$ and $QT(D(M))$ of M can be found in Figure 4:
The quantifier tree $QT(D(M))$ consists of two quantifier vectors invariant of the fourth position $[(y, x+1, x+5, b_1), (z,\ \ x+1, x+3, b_2)]$ and

$$[(y, x+1, x+5, b_1), (w, x+2, x+7, b_2), (m, x+1, x+7, b_3)]\,.$$

Possible evaluation vectors for the former are $[(1,5,0),(1,3,0)]$ or $[(1,5,1),(1,3,0)]$.

Definition 13. Let $v, w \in \mathbb{EV}$. The *concatenation of* v *and* w, $v \otimes w$, is defined as the vector $c \in \mathbb{EV}$ such that, for $1 \leq i \leq |v|$, $v(i) = c(i)$ and for $1 \leq j \leq |w|$, $w(|v| + j) = c(|v| + j)$.

Definition 14. Let $v, w_0, w, w_1 \in \mathbb{EV}$ s.t. $v = w_0 \otimes w \otimes w_1$. We refer to w as a *sub-evaluation vector of* v. If $0 < |w_0| + |w_1|$ then w is called *proper*. The set of sub-evaluation vectors of v is $\mathbb{EV}(v)$.

Example 4. Let us consider the evaluation vector $v = [(1,5,0),(2,7,0),(1,7,0)]$. We can write it using concatenation as $v = [(1,5,0)] \otimes [(2,7,0)] \otimes [(1,7,0)]$. A proper sub-evaluation vector of v would be $[(2,7,0)]$ where $w_0 = [(1,5,0)]$ and $w_1 = [(1,7,0)]$.

Definition 15. Let $v \in \mathbb{EV}$. We refer to v as *uniform standard* if there exist $a, b \in \mathbb{Z}$, where $a \leq b$, and for all $w \in \mathbb{EV}(v)$, $w(1,1) = a$ and $w(1,2) = b$. Let $\mathbb{US}$ be the set of uniform standard evaluation vectors and $\mathbb{US}(v)$ be the uniform standard evaluation vectors of a vector v.

Using concatenation and $\mathbb{US}$, we can decompose evaluation vectors into *sections*.

Definition 16. Let $v \in \mathbb{EV}$ and $s \in \mathbb{U}\$(v)$. s is a *section* of v if for every non-empty $w \in \mathbb{EV}(v)$, we have $s \otimes w \notin \mathbb{U}\$(v)$ and $w \otimes s \notin \mathbb{U}\$(v)$. Let $\$(v)$ be the multi-set of sections of a vector v.

Definition 17. Let $v \in \mathbb{EV}$. We define $<_v$ as a total ordering on $\$(v)$ such that for $s, s' \in \$(v)$, $s <_v s'$ iff there exist $v_0, v_1, v_2 \in \mathbb{EV}(v)$ (not necessarily non-empty) such that $v = v_0 \otimes s \otimes v_1 \otimes s' \otimes v_2$. For any vector $v \in \mathbb{EV}$ [] is $(v,0)-$*perfect*. The minimum element of $<_v$ is $(v,1)-$*perfect*. For $w \in \mathbb{EV}(v)$ and $m \in \mathbb{N}$, w is $(v,m+1)-$*perfect* if $w = h \otimes s$, for some $h \in \mathbb{EV}(v)$, such that h is $(v,m)-$*perfect*, $s \in \$(v)$, and $v = w \otimes v_2$, for some $v_2 \in \mathbb{EV}(v)$.

Definition 18. Let $v \in \mathbb{EV}$. We refer to v as *top-free* if there exists $a \in \mathbb{Z}$ such that for all $w \in \mathbb{EV}(v)$ we have $w(1,1) = a$ and $a \leq w(1,2)$. Let $\mathbb{TF}$ be the set of top-free vectors and $\mathbb{TF}(v)$ the set of top-free vectors of a vector v.

Example 5. Let us consider the vector

$$v = [(1,5,0),(1,7,0)(1,7,0),(1,8,0),(1,3,0),(1,3,0),(1,8,0)]$$

This vector is a member of $\mathbb{TF}$ and has the following sections contained in the multi-set $\$(v)$:

$$\$(v) = \left\{ \begin{array}{ll} [(1,5,0)] & [(1,7,0)(1,7,0)] \\ [(1,8,0)] & [(1,3,0),(1,3,0)] \\ [(1,8,0)] & \end{array} \right\}$$

The ordering of these sections is as follows:

$$[(1,5,0)] <_v [(1,7,0)(1,7,0)] <_v [(1,8,0)] <_v [(1,3,0),(1,3,0)] <_v [(1,8,0)]$$

From the order we can derive the perfect vectors of v. The $(v,1)-$*perfect* vector is $[(1,5,0)]$ and the $(v,2)-$*perfect* vector $[(1,5,0),(2,7,0)(2,7,0)]$. The $(v,5)-$*perfect* vector is v itself. Every member of $\$(v)$ is also a member of $\mathbb{U}\$$.

The perfect vectors of top-free vectors allow us to precisely define two important subsets of evaluation vectors.

Definition 19. Let $v \in \mathbb{EV}$ be top-free and $1 \leq i < |\$(v)|$ and $w, h \in \mathbb{EV}(v)$ be $(v,i)-$*perfect* and $(v,i+1)-$*perfect*, respectively. We refer to v as *standard* if $w(|w|,2) \leq h(|h|,2)$. We refer to v as *inverse standard* if $w(|w|,2) \geq h(|h|,2)$. Let $\mathbb{S}\mathbb{E}$ be the set of standard vectors and $\mathbb{I}\$$ be the set of inverse standard vectors. Let $\mathbb{S}\mathbb{E}(v)$ be the set of standard vectors of a vector v and $\mathbb{I}\$(v)$ be the set of inverse standard vectors of v.

Example 6. Let us consider the vector

$$v = [(1,5,0),(1,7,0)(1,7,0),(1,8,0),(1,3,0),(1,3,0),(1,8,0)]$$

which is a member of $\mathbb{TF}$. The following two vectors are constructable from $\$(v)$:

$$v_s = [(1,3,0),(1,3,0),(1,5,0),(1,7,0)(1,7,0),(1,8,0),(1,8,0)] =$$

$$[(1,3,0),(1,3,0)] \otimes [(1,5,0)] \otimes [(1,7,0)(1,7,0)] \otimes [(1,8,0)] \otimes [(1,8,0)]$$

$$v_i = [(1,8,0),(1,8,0),(1,7,0)(1,7,0),(1,5,0),(1,3,0),(1,3,0)] =$$
$$[(1,8,0)] \otimes [(1,8,0)] \otimes [(1,7,0)(1,7,0)] \otimes [(1,5,0)] \otimes [(1,3,0),(1,3,0)]$$

Obviously be Definition 19, $v_s \in \mathbb{SE}$ and $v_i \in \mathbb{IS}$. This property is formalized in the following lemma.

Lemma 20. *Let $v \in \mathbb{TF}$. Then there exist vectors $v', v'' \in \mathbb{TF}$ such that $\mathbb{S}(v) = \mathbb{S}(v') = \mathbb{S}(v'')$ and $v' \in \mathbb{SE}$ and $v'' \in \mathbb{IS}$.*

Proof. We choose v' to be the vector with the ordering relation $s, s' \in \mathbb{S}(v')$, $s <_{v'} s'$ iff $s(1,2) < s'(1,2)$ and v'' with the ordering relation $s, s' \in \mathbb{S}(v')$, $s <_{v''} s'$ iff $s(1,2) > s'(1,2)$. □

In the next subsection we discussed the purpose of the third component of an evaluation vector and why zeroing these components is considered proper. As one might have guessed, these components aid the evaluation of evaluation vectors, essentially simulating rules $Q1-7$ from Figure 2. We reduced these rules to a splitting operation (splitting a vector into its evaluated and to be evaluated parts). Splitting is then generalized to define evaluation.

3.2. Evaluation

In this section we develop a method to evaluate evaluation vectors in a similar way as the operational semantics evaluates monitor specifications.

Definition 21. Let $v = h \otimes [(a,b,0)] \otimes w \in \mathbb{EV}$, where $ss(h,1) = |h|$ and $ss(w,0) = |w|$. A *split $spt(v)$ of v*, denotes the following set of evaluation vectors:

- If $ss(v,1) < |v|$ then
  - · if $w \neq []$ then
    - – if $a < b$, then $spt(v) = \{h \otimes [(a,b,1)] \otimes w, h \otimes [(a+1,b,0)] \otimes w\}$
    - – if $a = b$, then $spt(v) = \{h \otimes [(a,b,1)] \otimes w\}$
    - – if $a \geq b$, then $spt(v) = \{[]\}$.
  - · if $w = []$ then
    - – if $a < b$ then $spt(v) = \{h \otimes [(a+1,b,0)]\}$
    - – if $a = b$ then $spt(v) = \{h \otimes [(a,b,1)]\}$
    - – if $a \geq b$, then $spt(v) = \{[]\}$.
- If $ss(v,1) = |v|$ then $spt(v) = \{v\}$.

Definition 22. Let $v \in \mathbb{EV}$ be proper and $n \in \mathbb{N}$. We define the *n-iterated split $SI(n,v)$* of v inductively as follows:

$$SI(n+1,v) = \left(\bigcup_{w \in SI(n,v)} spt(w)\right) \cup SI(n,v)$$
$$SI(0,v) = \{v\}$$

Example 7. Figure 5 shows the iterated split operation applied up to the Third level to the vector $[(0,3,0),(0,3,0)]$. The iterated split $SI(3,v)$ is the union of all levels of the given tree. The gray nodes of the tree are evaluation vectors of which the split operator cannot be applied to, i.e. vectors v such that $ss(v,1) = |v|$.

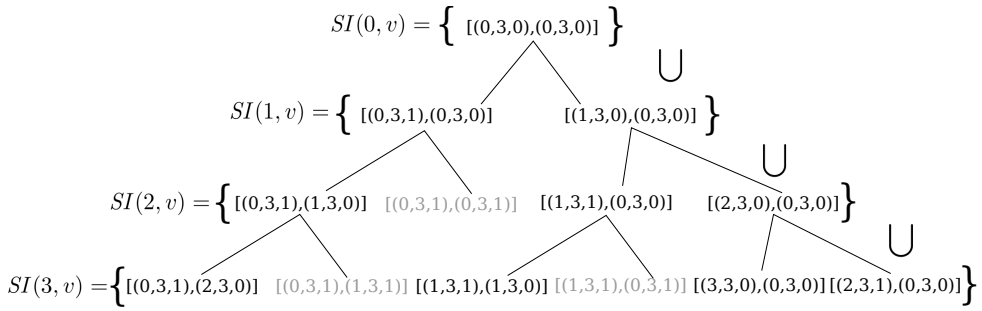

Fig. 5. Example of iterated splitting applied to a uniform evaluation vector.

Lemma 23. *There exists $n \in \mathbb{N}$ such that $SI(n+1, v) = SI(n, v)$.*

Proof (sketch). Trivial from the definition of split. □

Definition 24. Let $v \in \mathbb{EV}$ be proper. We define the *split closure* $SC_v = SI(n, v)$ of v, where n is the smallest $n \in \mathbb{N}$ such that $SI(n+1, v) = SI(n, v)$.

Lemma 25. *Let $v \in \mathbb{EV}$ be proper. Then for every vector $w \in SC_v$, there exists $k, m \in \mathbb{N}$ such that $m + k = |v|$ and $ss(w) = \overbrace{1 \cdots 1}^{m}\overbrace{0 \cdots 0}^{k}$.*

Definition 26. Let $v \in \mathbb{EV}$ be proper. The *degenerate vector set* DV_v of v is defined as follows:

$$DV_v = \{w \in SC_v | spt(w) = \{[]\}\}$$

Definition 27. Let $v \in \mathbb{EV}$ be proper. The set of *derived vectors* D_v of v is $D_v = (SC_v \setminus DV_v)$.

Example 8. Let us consider $v = [(0,1,0),(0,2,0)]$. Then

$$SI(5, v) =$$

$$\left\{ \begin{array}{cccc} [(0,1,0),(0,2,0)] & [(1,1,0),(0,2,0)] & [(0,1,1),(0,2,0)] & [(1,1,1),(0,2,0)] \\ [(0,1,1),(0,2,1)] & [(0,1,1),(1,2,0)] & [(1,1,1),(0,2,1)] & [(1,1,1),(1,2,0)] \\ [(0,1,1),(1,2,1)] & [(0,1,1),(2,2,0)] & [(1,1,1),(2,2,0)] & [(1,1,1),(1,2,1)] \\ [(0,1,1),(2,2,1)] & [(1,1,1),(2,2,1)] & [] & \end{array} \right\}$$

$$DV_v = \left\{ \begin{array}{cccc} [(0,1,1),(0,2,1)] & [(1,1,1),(0,2,1)] & [(0,1,1),(1,2,1)] & [(1,1,1),(1,2,1)] \\ [(0,1,1),(2,2,1)] & [(1,1,1),(2,2,1)] & [] & \end{array} \right\}$$

$$D_v = \left\{ \begin{array}{cccc} [(0,1,0),(0,2,0)] & [(1,1,0),(0,2,0)] & [(0,1,1),(0,2,0)] & [(1,1,1),(0,2,0)] \\ [(0,1,1),(1,2,0)] & [(1,1,1),(1,2,0)] & [(0,1,1),(2,2,0)] & [(1,1,1),(2,2,0)] \end{array} \right\}$$

The idea behind the derived set D_v is that these vectors represent the monitor instances which possibly need to be kept in memory during the evaluation of a monitor specification. However, to count the number of instances in memory, knowing the size of the derived set is not enough, because not all of these instances need to be in memory at the same time. For instance, $[(1,1,0),(0,2,0)]$ and $[(1,1,1),(1,2,0)]$ cannot be in memory at the same time because $(1,1,1)$ implies that position 1 has been reached and $(1,1,0)$ implies the opposite. The derived set is constructed specifically for defining the domain of our evaluation function.

Note that splitting divides the vector into two parts, a part with one in the third component and a part with zeroes in the third component, see Lemma 25. The following lemma addresses this and simplifies the definition of evaluation.

Lemma 28. *Let $v \in \mathbb{EV}$ be non-empty and proper. For every vector $w \in D_v$, there exist $\mathbf{1}_w, \mathbf{h}_w, \mathbf{0}_w \in \mathbb{EV}(v)$, such that $w = \mathbf{1}_w \otimes \mathbf{h}_w \otimes \mathbf{0}_w$, where $\mathbf{1}_w(i,3) = 1$, $1 \leq i \leq |\mathbf{1}_w|$, $\mathbf{h}_w(1,3) = 0$ and $|\mathbf{h}_w| = 1$, and $\mathbf{0}_w(i,3) = 0$, $1 \leq i \leq |\mathbf{0}_w|$. Both $\mathbf{0}_w$ and $\mathbf{1}_w$ can be the empty vector, but $\mathbf{h}_w$ is always non-empty.*

Proof. Follows, almost directly, from Lemma 25. □

Definition 29. Let $v \in \mathbb{EV}$ be non-empty and proper, $s \in \mathbb{N}$, and $w \in D_v$. We define the sets $e(s,w,v)$ and $\overline{e}(s,w,v)$ as follows:

E1) If $s < \mathbf{h}_w(1,1)$, then

$$e(s,w,v) = \{w\}$$
$$\overline{e}(s,w,v) = \emptyset$$

E2) If $\mathbf{h}_w(1,1) \leq s < \mathbf{h}_w(1,2)$ and $\mathbf{0}_w \neq []$, then

$$e(s,w,v) = \left(\bigcup_{i=\mathbf{h}_w(1,1)}^{s} e(s, \mathbf{1}_w \otimes v_i \otimes \mathbf{0}_w, v)\right) \cup \{\mathbf{1}_w \otimes w_{s+1} \otimes \mathbf{0}_w\}$$
$$\overline{e}(s,w,v) = \left(\bigcup_{i=\mathbf{h}_w(1,1)}^{s} \overline{e}(s, \mathbf{1}_w \otimes v_i \otimes \mathbf{0}_w, v)\right)$$

where $v_i = [(i, \mathbf{h}_w(1,2), 1)]$ and $w_{s+1} = [(s+1, \mathbf{h}_w(1,2), 0)]$.

E3) If $\mathbf{h}_w(1,2) \leq s$ and $\mathbf{0}_w \neq []$,then

$$e(s,w,v) = \left(\bigcup_{i=\mathbf{h}_w(1,1)}^{\mathbf{h}_w(1,2)} e(s, \mathbf{1}_w \otimes v_i \otimes \mathbf{0}_w, v)\right)$$
$$\overline{e}(s,w,v) = \left(\bigcup_{i=\mathbf{h}_w(1,1)}^{\mathbf{h}_w(1,2)} \overline{e}(s, \mathbf{1}_w \otimes v_i \otimes \mathbf{0}_w, v)\right)$$

where $v_i = [(i, \mathbf{h}_w(1,2), 1)]$.

E4) If $\mathbf{h}_w(1,1) \leq s < \mathbf{h}_w(1,2)$ and $\mathbf{0}_w = []$, then

$$e(s,w,v) = \{\mathbf{1}_w \otimes w_{s+1} \otimes \mathbf{0}_w\}$$
$$\overline{e}(s,w,v) = \bigcup_{i=\mathbf{h}_w(1,1)}^{s} \{\mathbf{1}_w \otimes v_i \otimes \mathbf{0}_w\}$$

where $w_{s+1} = [(s+1, \mathbf{h}_w(1,2), 0)]$, and $v_i = [(i, \mathbf{h}_w(1,2), 1)]$.

E5) If $\mathbf{h}_w(1,2) \leq s$ and $\mathbf{0}_w = []$, then

$$e(s,w,v) = \emptyset$$
$$\overline{e}(s,w,v) = \bigcup_{i=\mathbf{h}_w(1,1)}^{\mathbf{h}_w(1,2)} \{\mathbf{1}_w \otimes v_i \otimes \mathbf{0}_w\}$$

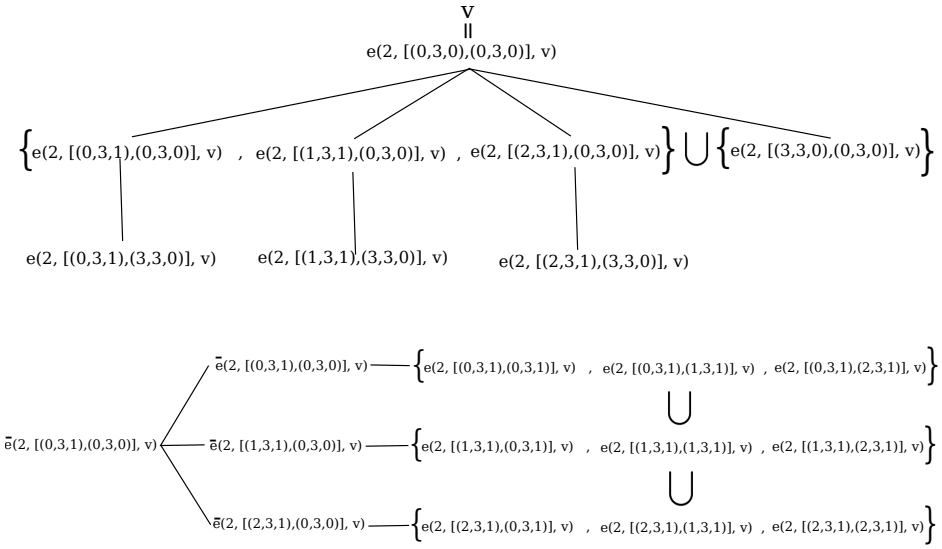

Fig. 6. Application of Definition 29 to the evaluation vector $[(0,3,0),(0,3,0)]$

Example 9. In Figure 6 we provide the derivation tree of both the function e and the function $\overline{e}$ for the value 2 applied to the vector $[(0,3,0),(0,3,0)]$. For

$$e(2,[(0,3,0),(0,3,0)],[(0,3,0),(0,3,0)])$$

we did not completely expand the set being that it is too big. However, we show the result of the first two steps. For $\overline{e}(2,[(0,3,0),(0,3,0)],[(0,3,0),(0,3,0)])$, the union of the leaves represents the content of the set.

In Section 2.3, we referred to the relationship between Definition 29 and Figure 2, especially to the function $e(\cdot,\cdot,\cdot)$. Now that both concepts have been introduced we can discuss the relationship further. Let us start with the rule **E1** and $Q3$. The rule states that evaluation of an evaluation vector w at a position s such that $s < \mathbf{h}_w(1,1)$ is not possible. The rule $Q3$ similarly states that a formula transition does nothing when the current position is lower than the lower bound of the quantifier. The relationship between **E2** and $Q4$ is more interesting. The rule **E2** states that evaluation of an evaluation vector w at a position s, such that $\mathbf{h}_w(1,1) < s < \mathbf{h}_w(1,2)$, constructs $s - \mathbf{h}_w(1,1) + 1$ new instances and evaluates them individually. Also the lower bound of $\mathbf{h}_w$ is replaced by $s+1$ to represent the partial evaluation. The rule $Q4$ similarly states that an instance set ought to be constructed for the formula being evaluated by instantiating the quantifier's variable with the values between p_1 and p. replace the lower bound of the quantifier with the next position. From these examples the relationship ought to be clear, we leave the rest of the comparisons to the reader.

Lemma 30. *Let $v \in \mathbb{EV}$ be non-empty and proper, $w \in D_v$ and $0 \leq m$. Then $E(m,w,v) \equiv E(m, \mathbf{h}_w \otimes \mathbf{0}_w, v)$.*

Proof. Follows from Definition 29. □

Lemma 31. *Let $v \in \mathbb{EV}$ be non-empty and proper, $w \in D_v$ and $0 \leq m$, then $e(m,w,v) \subseteq D_v$ and $\overline{e}(m,w,v) \subseteq DV_v$.*

Proof. $e(m,w,v)$ is multiple applications of the split operator. The split operator results in vectors which are either in D_v or DV_v. By Definition 29, we know that for every $v' \in e(m,w,v)$, $ss(v',1) \leq |v'|$ and thus $v' \notin DV_v$. Also, for every $v' \in \overline{e}(m,w,v)$, $ss(v',1) = |v'|$ and thus $v' \in DV_v$. And thus, from these facts the lemma follows. □

We can also consider the application of Definition 29 to sets of vectors.

Definition 32. Let $v \in \mathbb{EV}$ be non-empty and proper, V a set of vectors, such that $V \subseteq D_v$, and $0 \leq m$. Then we define $e(m,V,v) \equiv \bigcup_{w \in V} e(m,w,v)$.

Definition 33. Let $v \in \mathbb{EV}$ be non-empty and proper, $w \in D_v$, and $0 \leq m$. Then we define

$$E(m,w,v) = \begin{cases} e(m, E(m-1,w,v), v) & v(1,1) < m \\ e(v(1,1), m, v) & \text{otherwise} \end{cases}$$
$$\overline{E}(m,w,v) = \bigcup_{i=v(1,1)}^{m} \overline{e}(i,w,v)$$

Lemma 34. *Let $v \in \mathbb{EV}$ be non-empty and proper, $w \in D_v$, and $0 \leq m$. Then $E(m,w,v) \subseteq D_v$ and $\overline{E}(m,w,v) \subseteq DV_v$.*

Proof. Similar to the proof of Lemma 31. □

Within a set of vectors $V \subseteq D_v$ for some non-empty and proper $v \in \mathbb{EV}$, it is of importance (especially for the proofs of theorems in the next section) to know how many vectors $w \in V$ exists such that $w(i) = \mathbf{h}_w$.

Definition 35. Let $v \in \mathbb{EV}$ be non-empty and proper, and $V \subseteq D_v$ be a set of vectors. We define $\{V\}^i$ for all $1 \leq i \leq |v|$ as $\{V\}^i = \{w \in V \mid \mathbf{h}_w = w(i)\}$. The size $|V|^i$ of $\{V\}^i$ is defined as $|V|^i = |\{V\}^i|$.

Example 10. Let us consider the evaluation of the vector $v = [(0,5,0),(0,6,0),(0,7,0)]$.

$$E(0,v,v) = \left\{ \begin{array}{ll} [(1,5,0),(0,6,0),(0,7,0)] & [(0,5,1),(1,6,0),(0,7,0)] \\ [(0,5,1),(0,6,1),(1,7,0)] & \end{array} \right\}$$

$$E(1,v,v) = \left\{ \begin{array}{ll} [(2,5,0),(0,6,0),(0,7,0)] & [(1,5,1),(2,6,0),(0,7,0)] \\ [(1,5,1),(0,6,1),(2,7,0)] & [(1,5,1),(1,6,1),(2,7,0)] \\ [(0,5,1),(2,6,0),(0,7,0)] & [(0,5,1),(1,6,1),(2,7,0)] \\ [(0,5,1),(0,6,1),(2,7,0)] & \end{array} \right\}$$

$$E(2,v,v) = \left\{ \begin{array}{ll} [(3,5,0),(0,6,0),(0,7,0)] & [(2,5,1),(3,6,0),(0,7,0)] \\ [(2,5,1),(0,6,1),(3,7,0)] & [(2,5,1),(1,6,1),(3,7,0)] \\ [(2,5,1),(2,6,1),(3,7,0)] & [(1,5,1),(3,6,0),(0,7,0)] \\ [(1,5,1),(2,6,1),(3,7,0)] & [(1,5,1),(0,6,1),(3,7,0)] \\ [(1,5,1),(1,6,1),(3,7,0)] & [(0,5,1),(3,6,0),(0,7,0)] \\ [(0,5,1),(2,6,1),(3,7,0)] & [(0,5,1),(1,6,1),(3,7,0)] \\ [(0,5,1),(0,6,1),(3,7,0)] & \end{array} \right\}$$

Take note of the growth of $|E(i,v,v)|^{j+1}$, for $j \in \{0,1,2\}$ and $0 \leq i$:

		i						
		0	1	2	3	4	5	6
	0	1	1	1	1	1	0	0
j	1	1	2	3	4	5	6	0
	2	1	4	9	16	25	36	42

Essentially, $|E(i,v,v)|^1 = O(i)$ and $|E(i,v,v)|^2 = O(i^2)$. This observation is an essential part of the results in the next section.

Lemma 36. *Let $v \in \mathbb{EV}$ be non-empty and proper, $l \in \mathbb{N}$, where $1 \leq l < |\mathbb{S}(v)|$, and $w,h \in \mathbb{EV}(v)$ where w is $(v,l+1)-$perfect and h is $(v,l)-$perfect. Then for $m,q \in \mathbb{N}$, such that $m \leq w(|w|,2)$ and $|h| \leq q < |w|$, $|E(m,v,v)|^q = |E(m,w,w)|^q$ holds.*

Proof. Follows from Definition 29, Definition 35 and Lemma 25. Any vector $r \in E(m,v,v)$ such that $ss(r,1) \geq |w|$ will not be counted by $|E(m,v,v)|^q$. Thus we can ignore them. This allows us to only consider the vector w itself. □

Lemma 37. *Let $v \in \mathbb{EV}$ be non-empty and proper, $l \in \mathbb{N}$, where $1 \leq l < |\mathbb{S}(v)|$, $s \in \mathbb{S}(v)$, and $w,h \in \mathbb{EV}(v)$ where w is $(v,l+1)-$perfect, h is $(v,l)-$perfect, and $w = h \otimes s$. Then for $m,q \in \mathbb{N}$, such that $m \leq w(|w|,2)$ and $|h| \leq q < |w|$. Then*

$$|E(m,v,v)|^q = |\overline{E}(m,h,h)| \cdot |E(m,s,s)|^{|s|}$$

Proof. For every $g \in \overline{E}(m,h,h)$, $ss(w,1) = |h|$. If we were to extend each of these evaluation vectors with every evaluation vector in $E(m,s,s)$, then we would get $E(m,w,w)$. By Lemma 36 we get $|E(m,v,v)|^q = |E(m,w,w)|^q$. Putting everything together we get the lemma. □

Corollary 38. *Let $v \in \mathbb{EV}$ be non-empty and proper and $l \in \mathbb{N}$, where $0 \leq l < |\mathbb{S}(v)|$. Let $w,h \in \mathbb{EV}(v)$, where w is $(v,l)-$perfect, and $v = w \otimes h$. Let $q,m \in \mathbb{N}$, where $|w| < q \leq |v|$ and $m \leq h(1,2)$. Then*

$$|E(m,v,v)|^q = |\overline{E}(m,w,w)| \cdot |E(m,h,h)|^{q'}$$

where $q' = q - |w|$.

Proof. Apply Lemma 36 in the inverse direction. □

Corollary 38 can easily be applied within a single section of an evaluation, which we will use in the following sections.

Lemma 39. *Let $v \in \mathbb{EV}$ be non-empty and proper, V be a set of vectors, s.t. $V \subseteq D_v$,and $m \geq 0$. Then the following holds:*

$$|E(m,v,v)| = \sum_{i=1}^{|v|} |E(m,v,v)|^i \tag{1a}$$

$$E(m,V,v) = \bigcup_{i=1}^{|v|} E(m,\{V\}_1^i,v) = \bigcup_{i=1}^{|v|} \bigcup_{w\in\{V\}^i} E(m,w,v) \tag{1b}$$

$$|E(m,V,v)| = \sum_{i=1}^{|v|} |E(m,\{V\}^i,v)| = \sum_{i=1}^{|v|} \sum_{w\in\{V\}^i} |E(m,w,v)| \tag{1c}$$

$$|E(m,V,v)|^i = \sum_{j=1}^{|v|} |E(m,\{V\}^j,v)|^i = \sum_{j=1}^{|v|} \sum_{w\in\{V\}^j} |E(m,w,v)|^i \tag{1d}$$

$$|E(m,V,v)|^i = |\bigcup_{j=1}^{i} E(m,\{V\}^j,v)|^i \tag{1e}$$

Lemma 39 will be used in the proof of Theorem 47 which heavily relies on rewriting of the evaluation function. Each of the statements in Lemma 39 is easily proven from the above definitions concerning the evaluation function. Though, Definition 33 is close to our meaning of runtime representation size, it does not take into account the addition of new instances of the vector v.

Definition 40. Let $s = [(a,b,0)] \in \mathbb{EV}$ and $n \in \mathbb{N}$. The *n-step-pair $s:n$ of s* is defined as $s:n = [(a+n, b+n, 0)]$.

Definition 41. Let $v, v' \in \mathbb{EV}$ be non-empty and proper, $s \in \$(v)$, s.t. $v = v' \otimes s$, and $n \in \mathbb{N}$. We define the *step-pair generation function* $st(n,v)$ inductively as follows:

$$st(n, v' \otimes s) = st(n,v') \otimes s:n$$
$$st(n,[]) = [].$$

We write $st(n,v)$ as $v:n$.

Note that in Definition 41, the induction is not over n, but rather over the total order $<_v$.

Lemma 42. *Let $v \in \mathbb{EV}$ be non-empty and proper and $m,n \in \mathbb{N}$. Then $|E(m,v,v)| = |E(n+m, v:n, v:n)|$.*

Proof. Essentially, adding one to every position in a vector v is the same as evaluating the vector at a position one step in the future. □

Definition 43. Let $v \in \mathbb{EV}$ be non-empty and proper and $c, m \in \mathbb{N}$, such that $0 \leq c \leq \max\{0, v(1,1)\}$ and $m = \max_{s \in \mathbb{S}(v)}\{s(1,2)\} - 1$. The *space requirements* $SR(c,r)$ of v starting at c is

$$SR(c,v) = \left| \bigcup_{i=c}^{m} E(m, v:(i-c), v:(i-c)) \right| = \sum_{i=c}^{m} |E(i,v,v)|\,.$$

The idea behind Definition 43 is that we find the furthest position in the future used in the vector and look at the evaluation of every n-step pair prior to that position. The n-step pairs represent the new monitor instance generated when a position is reached. This definition is essentially a formalized version of the concept described at the end of Section 2.3, however, our formalization corresponds to the above definition of evaluation rather than to the operational semantics of Figure 2. We can more clearly reach the results of the next section by working with this alternative formalization.

Now using Definition 43 we are able to connect the operational semantics of Section 2 with the abstraction found in this section.

Theorem 44. *Let $M \in \mathbb{M}$ such that $qt = QT(D(M)) \in \mathbb{QV}$ and $v = \mathbb{QP}(qt)$. Then for all $n, p, s, s' \in \mathbb{N}$, $ms \in \{\top, \bot\}^\omega$, such that $T(M) \multimap_{p,ms,n} s$, we have $s \leq SR(0,v)$.*

Proof(Sketch). Clearly, Theorem 4 lets us ignore non-dominating monitors being that if $T(M) \multimap_{p,ms,n} s$, $T(D(M)) \multimap_{p,ms,n} s'$, and $s' \leq SR(0,v)$, it is obviously the case that $s \leq SR(0,v)$. Now let us consider for our basecase the simplest monitor $M = \forall_{0 \leq x} : \forall_{y \in [x+a, x+b]} : \varphi(x,y)\ \&\ \psi(x)$ where ψ and φ are propositional, $a, b \in \mathbb{Z}$ and $a \leq b$. For such a monitor the runtime representation size is dependent on how future looking the interval is. For example, if M is past looking, then the runtime representation size is always zero. For $SR(0,v)$ we would get the following:

$$SR(0,v) = \sum_{i=0}^{b-1} |E(i,v,v)| = 0$$

The reason $SR(0,v) = 0$ is that $b \leq 0$ and **E5** returns the empty set. If M is future looking, i.e. $0 \leq a \leq b$, we can refer to the work of (Cerna et al., 2016b) which tells use that $T(M) \multimap_{p,ms,n} b$ for $b - 1 \leq n$. Now let us consider $SR(0,v)$:

$$SR(0,v) = \sum_{i=0}^{b-1} |E(i,v,v)| = \sum_{i=0}^{a-1} |E(i,v,v)| + \sum_{i=a}^{b-1} |E(i,v,v)|.$$

Let us consider the first sum on the right side. Being that $i < a$ in $E(i,v,v)$ we evaluate using **E1** and thus, $|E(i,v,v)| = 1$ and we get the following

$$\sum_{i=0}^{a-1} |E(i,v,v)| + \sum_{i=a}^{b-1} |E(i,v,v)| = a + \sum_{i=a}^{b-1} |E(i,v,v)|.$$

For the second summation we use **E4** and again we get that $|E(i,v,v)| = 1$ and the following:

$$|a| + \sum_{i=a}^{b-1} |E(i,v,v)| = |a| + |b-1-a| + 1 = |b|.$$

The final case is when M is both future and past looking. However, this case is simple because the future looking part and past looking part are treated separately and the result is again the same as the upper bound b. This concludes the base case.

For the step case, let us assume the theorem holds for all monitors M such that $|\mathbb{QP}(QT(D(M)))| = n$ and show that the theorem holds for monitors M' such that $|\mathbb{QP}(QT(D(M')))| = n+1$. This time the upper bound is

$$m' = \max_{s \in \mathbb{S}(\mathbb{QP}(QT(D(M'))))} \{s(1,2)\} - 1.$$

From now on we refer to $\mathbb{QP}(QT(D(M')))$ as v' and assume that $v' = [(a,b,0)] \otimes v$, where $a, b \in \mathbb{Z}$, $a \leq b$ and there exists a monitor M such that $v = \mathbb{QP}(QT(D(M)))$ and $|v| = n$. We assume that both v' and v are proper. Note that

$$SR(0,v') = \sum_{i=0}^{m'} |E(i,v,v)| = \sum_{i=0}^{a-1} |E(i,v',v')| + \sum_{i=a}^{b-1} |E(i,v',v')| + \sum_{i=b}^{m'} |E(i,v',v')|$$

We start with the right most summation which is evaluated using **E3**:

$$\sum_{i=b}^{m'} |E(i,v',v')| = \sum_{i=b}^{m'} \left| \bigcup_{j=a}^{b} E(i,[(j,b,1)] \otimes v, v') \right| = \sum_{i=b}^{m'} \sum_{j=a}^{b} |E(i,[(j,b,1)] \otimes v, v')|$$

This statement can be further reduced as follows:

$$\sum_{i=b}^{m'} (|b-a|+1) \cdot |E(i,v,v)| = (|b-a|+1) \cdot \sum_{i=b}^{m'} |E(i,v,v)|$$

Now let us consider the next summation which evaluates by **E2**:

$$\sum_{i=a}^{b-1} |E(i,v',v')| = \sum_{i=a}^{b-1} \left| \bigcup_{j=a}^{i} E(i,v',v') \cup \{[(i+1,b,0)] \otimes v\} \right| =$$

$$\sum_{i=a}^{b-1} \left(\sum_{j=a}^{i} |E(i,[(j,b,1)] \otimes v, v')| \right) + 1 = \left(\sum_{i=a}^{b-1} (|i-a|+1) \cdot |E(i,v,v)| \right) + (b-a)$$

We do the same for the last summation, this time using **E1**:

$$\sum_{i=0}^{a-1} |E(i,v',v')| = \sum_{i=0}^{a-1} |\{v'\}| = a$$

When we put all the parts together we get the following:

$$a + \left(\sum_{i=a}^{b-1} (|i-a+1|) \cdot |E(i,v,v)| \right) + |b-a| + (|b-a|+1) \cdot \sum_{i=b}^{m'} |E(i,v,v)|$$

We haven't explicitly constructed it, but $SR(0, v)$ is hidden in the equation, of which we know the theorem holds for by the induction hypothesis. Also, by the base case we know that $|b - a|$ and a are derived from the evaluation of a single quantifier. Of course, when $b \leq 0$ we remove these terms. There is no other way to unroll a single quantifier, thus the summations produced represent the maximum runtime representation size which can be produced. This is obvious from the unrolling of the outer most quantifier and matches the rules of Table 2. Thus, be induction we have proven the theorem. □

Example 11. By Lemma 42 and the table of Example 10, we can calculate $SR(0, v)$, where v is the vector from Example 10, as

$$SR(0, v) = \left| \bigcup_{i=0}^{6} E(6, v : i, v : i) \right| = \sum_{i=0}^{6} |E(i, v, v)| = \sum_{i=0}^{6} \sum_{j=0}^{2} |E(i, v, v)|^{j+1}$$

where v is the vector of Example 10. $SR(0, v)$ is the sum of all the columns of the matrix which is 159. Previous work (Cerna et al., 2016b) determines 1088 as the approximate bound for the space requirements of the vector. Thus, our new results provide a much tighter bound. In the next section we construct an analytic expression for both uniform and standard evaluation vectors.

4. Analysis of Uniform, Standard, and inverse-standard Evaluation Vectors

In this section, we provide precise expressions for the space requirements of uniform and standard evaluation vectors. The table at the end of Example 10 hints to the results found in Section 4.1. However, when dealing with standard evaluation vectors a new problem arises, that one must count not only what stays memory at a given position, but also what leaves memory. This occurs because the intervals are not the same size. We deal with this issue in Section 4.2.

4.1. Uniform Standard Evaluation Vectors

In this section we focus on the space complexity of uniform standard evaluation vectors. A precise bound is provided.

Theorem 45. *Let $v \in \mathbb{U}\$$ be a non-empty and proper. Then $|e(v(1,1), v, v)| = |v|$ and $|\overline{e}(v(1,1), v, v)| = 1$.*

Proof. We will refer to $v(i, 1)$ as a and $v(i, 2)$ as b for $1 \leq i \leq |v|$. We prove the theorem by induction on $|v|$. Let us consider the case when $|v| = 1$, then we know that $v = [] \otimes \mathbf{h}_v \otimes []$. Based on our assumptions, when evaluating $e(a, v, v)$ using Definition 29, we apply **E4**. We arrive at the following derivation:

$$e(a, v, v) = \{[(a + 1, b, 0)]\} .$$

and

$$\overline{e}(a, v, v) = \{[(a, a, 1)]\} .$$

The resulting set has size one and thus the theorem holds for the base case. Let us now assume the theorem holds for vectors of length up to n. We show that the theorem holds for vectors of length $n + 1$. We know that $v = [] \otimes [(a, b, 0)] \otimes \mathbf{0}_v$.

Also we know, by the induction hypothesis that $\overline{e}(a,[(a,b,0)],[(a,b,0)]) = \{[(a,a,1)]\}$, $e(a,[(a,b,0)],[(a,b,0)]) = \{[(a+1,b,0)]\}$, and $|e(a,\mathbf{0}_v,\mathbf{0}_v)| = |v|-1$. From these sets we can derive the following result:

$$|e(a,[(a,b,0)],[(a,b,0)])|+|\overline{e}(a,[(a,b,0)],[(a,b,0)])|*|e(a,\mathbf{0}_v,\mathbf{0}_v)| = 1+1*(|v|-1) = |v|.$$

As for $|\overline{e}(a,v,v)|$ we just need to compute:

$$|\overline{e}(a,[(a,b,0)],[(a,b,0)])| * |e(a,\mathbf{0}_v,\mathbf{0}_v)| = 1*1 = 1$$

□

Corollary 46. *Let $v \in \mathbb{U}\$$ be non-empty and proper. Then for $1 \leq i \leq |v|$,*

$$|e(v(1,1),v,v)|^i = 1.$$

Theorem 47. *Let $v \in \mathbb{U}\$$ be a non-empty and proper, and $i,m \in \mathbb{N}$, such that $1 \leq i \leq |v|$ and $v(1,1) \leq m < v(1,2)$. Then $|E(m,v,v)|^i = ((m-v(1,1))+1)^{i-1}$.*

Proof. From now on in this proof we will refer to $v(1,1)$ as a and $v(1,2)$ as b and $|v|$ as n. To prove this theorem we perform an induction over the lexicographical ordering of the ordered pairs (m,i) for $a \leq m < b$ and $1 \leq i \leq n$. In the ordered pair (m,i), m refers to $E(m,v,v)$ and $i = ss(w,1)+1$ for $w \in D_v$.

Fixed Recursion Depth Induction From Theorem 45 and Corollary 46 we know that

$$|E(a,v,v)|^i = 1^{i-1}.$$

We refer to $|E(a,v,v)|^1 = 1$ as $(a,1)$-BC (BaseCase). One induction hypothesis (IH) based on $(a,1)$-BC assumes that $|E(a,v,v)|^j = 1$ for $1 \leq j \leq i < n$, and we show that the theorem also holds for $i+1$ (Theorem 45 and Corollary 46). We refer to this IH as $(a,\overrightarrow{\omega})$-IH and the theorem that it proves as (a,ω)-BC.

Fixed Evaluation Vector Position Induction Now we construct the other IH. We need to use $(a,1)$-BC to construct our induction hypothesis $(\overrightarrow{\omega},1)$-IH. we assume that $|E(r,v,v)|^1 = 1$ holds when $a \leq r \leq m < b-1$, and we show that it also holds for $m+1$. Essentially we show that $|E(m+1,v,v)|^1 = (((m+1)-a)+1)^0 = 1$. First, by Definition 33, $E(m+1,v,v) = e(m+1,E(m,v,v),v)$. By $(\overrightarrow{\omega},1)$-IH, $|E(m,v,v)|^1 = 1$. By Definition 32:

$$e(m+1,E(m,v,v),v) = \bigcup_{w \in E(m,v,v)} e(m+1,w,v) \tag{2a}$$

$$e(m+1,w,v) = \bigcup_{i=a}^{m+1} e(m+1,w'',v) \cup \{w'\} \tag{2b}$$

By Definition 29, $\left|\bigcup_{i=a}^{m+1} e(m+1,w'',v)\right|^1 = 0$, and thus,$|e(m+1,w,v)|^1 = |\{w'\}|^1 = 1$.

Induction on Recursion Depth and Position in v Using both (a,ω)-BC and $(\omega,1)$-BC We now construct our final induction hypothesis. We assume that the theorem holds for $|E(m,v,v)|^j$, $|E(r,v,v)|^{i+1}$, and $|E(r,v,v)|^1$, where $a \leq r \leq m < b-1$ and $1 \leq j \leq i < n$. We refer to the first assumption, $(m+1,j)$, as IH_1 and the two other assumptions, $(r,i+1)$ and $(r,1)$, as IH_2. Given these assumptions we now show that the theorem holds for $|E(m+1,v,v)|^{i+1}$.

We start, as in $(\omega,1)$-BC, by deconstructing $E(m+1,v,v)$ into parts justified by either IH_1 or IH_2.

$$E(m+1,v,v) = e(m+1,E(m,v,v),v) \tag{3}$$

Using Lemma 39 parts 1d & 1e we can justify the following equality:

$$|E(m+1,v,v)|^{i+1} = \tag{4}$$

$$|e(m+1,E(m,v,v),v)|^{i+1} = \tag{5}$$

$$|e(m+1,\bigcup_{k=1}^{i}\{E(m,v,v)\}^k)|^{i+1} + \sum_{w\in\{E(m,v,v)\}^{i+1}} |e(m+1,w,v)|^{i+1} \tag{6}$$

We now concern ourselves with the summation in Equation 4. By IH_2, we know that $|E(m,v,v)|^{i+1} = ((m-a)+1)^{i+1}$, thus, if the evaluation of $|e(m+1,w_1,v)|^{i+1} = |e(m+1,w_2,v)|^{i+1}$ for any $w_1,w_2 \in \{E(m,v,v)\}^{i+1}$, the summation can be rewritten as $((m-a)+1)^{i+1}\cdot|e(m+1,w',v)|^{i+1}$ for some arbitrary $w' \in \{E(m,v,v)\}^{i+1}$.

There are two ways $e(m+1,w,v)$ can evaluate for $w = \mathbf{1}_w \otimes \mathbf{h}_w \otimes \mathbf{0}_w$ when $\mathbf{h}_w = [(m+1,b,0)]$. Note, $\mathbf{h}_w(1,1) = m+1$ because every member in the set $\{E_v(m,v,v)\}^{i+1}$ was derived from prior evaluations. The following two equations are the possible evaluations (we leave out $\mathbf{1}_w$, justified by Lemma 30):

$$e(m+1,w,v) = e(m+1,[(a,b,0)]\otimes\mathbf{0}'_w,v)\cup\{[(m+2,b,0)]\otimes\mathbf{0}_w\} \tag{7a}$$

$$e(m+1,w,v) = \{[(m+2,b,0)]\} \tag{7b}$$

when $\mathbf{0}_w = []$, case Equation 7b, then $|e(m+1,w,v)|^{i+1} = 1$. In the case of Equation 7a we just need to consider the same argument from Section 4.1, that is that $|e(m+1,[(a,b,0)]\otimes w'_0,v)|^{i+1} = 0$. Thus, $|e(m+1,w,v)|^{i+1} = 1$ for any $w \in \{E(m,v,v)\}^{i+1}$, and

$$\sum_{w\in\{E(m,v,v)\}^{i+1}} |e(m+1,w,v)|^{i+1} = ((m-a)+1)^{i+1}\cdot|e(m+1,w',v)|^{i+1} \tag{8}$$

Notice that this argument also implies $|e(m+1,w',v)|^{i+1} = 1$ and thus, we get $((m-a)+1)^{i+1}\cdot|e(m+1,w',v)|^{i+1} = ((m-a)+1)^{i+1}$.

Now we consider the other part of Equation 4:

$$|e(m+1,\textstyle\bigcup_{k=1}^{i}\{E(m,v,v)\}^k,v)|^{i+1} \tag{9}$$

We need to consider vectors of the form $w = \mathbf{1}_w\otimes[(m+1,b,0)]\otimes\mathbf{0}_w$, where $|\mathbf{1}_w| = i-1$ and how they evaluate to vectors $w' = \mathbf{1}'_w\otimes[(m+2,b,0)]\otimes\mathbf{0}'_w$, where $|\mathbf{1}'_w| = i$. By IH_1

and Definition 29 we know the following:

$$|e(m+1, \bigcup_{k=1}^{i} \{E(m,v,v)\}^k, v)|^i = (((m+1)-a)+1)^{i-1}. \tag{10}$$

By IH_2 and Definition 29 it can also be derived that,

$$|\bigcup_{k=1}^{i} \{E(m,v,v)\}^k|^i = ((m-a)+1)^{i-1}. \tag{11}$$

We are specifically interested in the behaviour of $|e(m+1, D, v)|^i$, were

$$D = \bigcup_{k=1}^{i} \{E(m,v,v)\}^k.$$

Using Lemma 39 we can derive that $|e(m+1, D, v)|^i = \sum_{w \in D} |e(m+1, w, v)|^i$. Each w is of the form $w = \mathbf{1}_w \otimes [(m+1, b, 0)] \otimes \mathbf{0}_w$, where $|\mathbf{1}_w| = i < n$. Now consider the evaluation $e(m+1, w, v) = e(m+1, w', v) \cup \{w''\}$ As in Equation 7a, $\{e(m+1, w', v)\}^i$ is empty, thus, $|e(m+1, w, v)|^i = 1$. This results in the following derivation

$$|e(m+1, D, v)|^i = \sum_{w \in D} |e(m+1, w, v)|^i = \sum_{w \in D} 1 = ((m-a)+1)^{i-1}. \tag{12}$$

Getting back to Equation 9, we can now compute $|e(m+1, w, v)|^{i+1}$. It is $|e(m+1, w, v)|^{i+1} = 1$ as well. Let us consider $e(m+1, w', v)$, where $w' = \mathbf{1}'_{w'} \otimes [(a, b, 0)] \otimes \mathbf{0}'_{w'}$, $\mathbf{h}'_{w'} = [(a, b, 0)]$ and $|\mathbf{1}_{w'}| = i$. This time using brackets for $i+1$, i.e. $\{e(m+1, w', v)\}^{i+1}$. Now consider the application of Definition 29 to w' for $m+1$, there are two cases:

$$e(m+1, w', v) = \bigcup_{k=a}^{m+1} e(m+1, w''(k), v) \cup \{w'''\} \tag{13}$$

$$e(m+1, w', v) = \{\mathbf{1}'_{w'} \otimes \mathbf{h}'_{w'} \otimes []\} \tag{14}$$

Obviously in the case of Equation 14, $|e(m+1, w', v)|^{i+1} = 1$. Like before $|e(m+1, w''(k), v)|^{i+1} = 0$.Thus, $|e(m+1, D, v)|^{i+1} = \sum_{w \in D} |e(m+1, w, v)|^{i+1} = \sum_{w \in D} 1 = ((m-a)+1)^{i-1}$. Now we can get back to Equation 10. Using our derivations and Lemma 39 the following holds:

$$|e(m+1, \bigcup_{k=1}^{i} \{E(m,v,v)\}^k, v)|^i = |e(m+1, \bigcup_{k=1}^{i-1} \{E(m,v,v)\}^k, v)|^i + ((m-a)+1)^{i-1} \tag{15}$$

We know the value of the left side Equation 15, and thus

$$|e(m+1, \bigcup_{k=1}^{i-1} \{E(m,v,v)\}^k, v)|^i = \sum_{j=1}^{i-1} \binom{i-1}{j} \cdot ((m-a)+1)^{(i-1)-j}.$$

The last step is computing the size of from $|e(m+1, \bigcup_{k=1}^{i-1} \{E(m,v,v)\}^k, v)|^{i+1}$ from $|e(m+1, \bigcup_{k=1}^{i-1} \{E(m,v,v)\}^k, v)|^i$.

important to this derivation is that $|\bigcup_{k=1}^{i-1}\{E(m,v,v)\}^k|^i = 0$. Essentially every $w \in \left\{e(m+1, \bigcup_{k=1}^{i-1}\{E(m,v,v)\}^k)\right\}^{i+1}$ was constructed from $m+1$ applications of Definition 29. Every vector used to construct the ws, w', had a head position $\mathbf{h}_{w'} = [(a,b,0)]$. Thus, $w' = \mathbf{1}_w \otimes [(a,b,0)] \otimes \mathbf{0}_w$, where $|\mathbf{1}_w| = i-1$. The evaluation is $e(m+1, w', v) = \left(\bigcup_{j=a}^{m+1} e(m+1, w''(j), v)\right) \cup \{w''' \otimes \mathbf{0}_w\}$. Again we see that $|e(m+1, w, v)|^i = 1$, but, $\left|\left(\bigcup_{j=a}^{m+1} e(m+1, w''(j) \otimes \mathbf{0}'_w, v)\right)\right|^{i+1} = ((m+1)-a)+1$. Now we put the results together to get the value of $|E(m+1,v,v)|^{i+1}$.

$$\left|\left(\bigcup_{j=a}^{m+1} e(m+1, \mathbf{1}'_w(j) \otimes [(a,b,0)] \otimes \mathbf{0}'_w, v)\right)\right|^{i+1} \cdot \left|e(m+1, \bigcup_{k=1}^{i-1}\{E(m,v,v)\}^k, v)\right|^i +$$

$$\left|e(m+1, \left\{\bigcup_{k=1}^{i}\{E(m,v,v)\}^k\right\}^i, v)\right|^{i+1} + |e(m+1, \{E(m,v,v)\}^{i+1}, v)|^{i+1} =$$

$$(((m+1)-a)+1) \cdot \left(\sum_{j=1}^{i-1} \binom{i-1}{j} \cdot ((m-a)+1)^{(i-1)-j}\right) + ((m-a)+1)^{i-1} + ((m-a)+1)^i$$

Distribution over the summation and placing $((m-a)+1)^{i-1}$ in the right most summation results in the following:

$$\left(\sum_{j=1}^{i-1} \binom{i-1}{j} \cdot ((m-a)+1)^{i-j}\right) + \left(\sum_{j=0}^{i-1} \binom{i-1}{j} \cdot ((m-a)+1)^{(i-1)-j}\right) + ((m-a)+1)^i \tag{16}$$

We now use the property $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ and a few simple manipulations we can derive:

$$\sum_{j=0}^{i} \binom{i}{j} \cdot ((m-a)+1)^{i-j} = (((m-a)+1)+1)^{i+1} = |E(m+1,v,v)|^{i+1} \tag{17}$$

and thus by induction the theorem is true. □

Theorem 47 can be used to derive the following space requirements for uniform standard evaluation vectors.

Corollary 48. *Let $v \in \mathbb{U}\$$ be a non-empty and proper, and $c \in \mathbb{N}$, such that $0 \leq c \leq v(1,1)$. Then*

$$SR(c,v) = (v(1,1)-c) + \sum_{i=1}^{m'}\sum_{j=0}^{|v|-1} i^j = ((v(1,1)+|v|)-c) + \left(\sum_{i=2}^{m'} \frac{(1-i^{|v|})}{1-i}\right)$$

where $m' = v(1,2) - v(1,1)$.

Proof. First we start by applying Lemma 39a to Definition 43:

$$SR(c,v) = \sum_{i=c}^{v(1,2)-1} |E(i,v,v)| = \sum_{i=c}^{v(1,2)-1}\sum_{j=1}^{|v|} |E(i,v,v)|^j.$$

By Definition 29, we know that for all k such that $c \leq k < v(1,1)$, that $|E(k,v,v)| = 1$. Thus, we have

$$(v(1,1)-c) + \sum_{i=v(1,1)}^{v(1,2)-1} \sum_{j=1}^{|v|} |E(i,v,v)|^j =$$

$$(v(1,1)-c) + \sum_{i=0}^{v(1,2)-v(1,1)-1} \sum_{j=1}^{|v|} |E(i+v(1,1),v,v)|^j.$$

By Theorem 47, we derive the following:

$$(v(1,1)-c) + \sum_{i=0}^{v(1,2)-v(1,1)-1} \sum_{j=1}^{|v|} |E(i+v(1,1),v,v)|^j =$$

$$(v(1,1)-c) + \sum_{i=0}^{v(1,2)-v(1,1)-1} \sum_{j=1}^{|v|} (i+1)^{j-1} = (v(1,1)-c) + \sum_{i=1}^{v(1,2)-v(1,1)} \sum_{j=0}^{|v|-1} i^j.$$

□

A direct consequence of Corollary 48 is that a partial evaluation function of a vector $v \in \mathbb{U}\$$ can be defined. This will be useful for the next section.

Definition 49. Let $v \in \$\mathbb{E}$ be a non-empty and proper, and $a, b \in \mathbb{N}$ such that $v(1,1) \leq a \leq b < v(1,2)$. We define the *partial evaluation of* v $P(a,b,v)$ as follows:

$$P(a,b,v) = \sum_{i=a}^{b} \sum_{j=0}^{|v|-1} ((i - v(1,1)) + 1)^j$$

Essentially Definition 49 allows us to choose the interval over which we evaluate the evaluation vector. Notice that we use a vector in $\$\mathbb{E}$ rather than $\mathbb{U}\$$.

4.2. Standard Evaluation Vectors

The following lemma highlights a major difference between standard and uniform evaluation vectors:

Lemma 50. *Let $v, h, g \in \$\mathbb{E}$ be non-empty and proper and $l \in \mathbb{N}$ such that $0 \leq l \leq |\$(v)|$, h is $(v,l)-$perfect, and $v = h \otimes g$. Then for all $i, k \in \mathbb{N}$ such that $g(1,2) \leq k$ and $1 \leq i \leq |h|$,*

$$|E(k,v,v)|^i = |E(k,h,h)|^i = 0.$$

Essentially Lemma 50 states that, for every vector $w \in E(k,v,v)$, $i \leq ss(w,1)$, i.e. $\mathbf{h}_w = v(k)$ such that $i < k$. However, more importantly the perfect vectors of v evaluate like v. A consequence of Lemma 50 is that the functions $\overline{e}(\cdot,\cdot,\cdot)$ and $\overline{E}(\cdot,\cdot,\cdot)$ need to be considered for the analysis of standard evaluation vectors, see Corollary 38.

Lemma 51. *Let $v, h \in \$\mathbb{E}$ be non-empty and proper, and $l \in \mathbb{N}$ such that $0 \leq l \leq |\$(v)|$ and h is $(v,l)-$perfect. Then $|\overline{E}(h(|h|,2),h,h)| = ((h(|h|,2) - h(1,1)) + 1)^{|h|}$.*

Proof. Essentially, $\overline{E}(h(|h|,2),h,h)$ is the entire degenerate set of h. The size of the degenerate set can be found as follows: consider h as the set of all words of length $|h|$ with an alphabet of size $((h(|h|,2)-v(1,1))+1)$. Members of the set are vectors $w \in SC_h$ such that $ss(w,1)=|h|$, subsets are vectors $w \in SC_h$ such that $ss(w,1) \leq |h|$. □

Inductively applying this result to the sections of a standard evaluation vector, we get the following lemma:

Lemma 52. *Let $v,h,g \in \mathbb{SE}$ be non-empty and proper, and $l \in \mathbb{N}$, such that $0 \leq l \leq |\mathbb{S}(v)|$, h is $(v,l)-$perfect, and $v = h \otimes g$. Then $|\overline{E}(v(|v|,2),v,v)| = |\overline{E}(h(|h|,2),h,h)| \cdot |\overline{E}(v(|v|,2),g,g)|$.*

Proof. Essentially, this recursive property results from the construction of the degenerate set. □

This entails the following result.

Corollary 53. *Let $v \in \mathbb{SE}$ be non-empty and proper. Then*

$$|\overline{E}(v(|v|,2),v,v)| = \prod_{h \in \mathbb{S}(v)} ((h(1,2)-h(1,1))+1)^{|h|}.$$

Inductively applying the result for uniform standard evaluation vectors, Corollary 48, to the sections of standard vectors provides us with the following space bounds for standard evaluation vectors.

Theorem 54. *Let $v,h,g,w \in \mathbb{SE}$ be non-empty and proper, $l \in \mathbb{N}$, $s \in \mathbb{S}(v)$, such that $0 \leq l < |\mathbb{S}(v)|$, h is $(v,l)-$perfect, w is $(v,l)-$perfect, $v = h \otimes g$, and $w = h \otimes s$. Then for all $k,q \in \mathbb{N}$, such that $h(|h|,2) \leq k < w(|w|,2)$, $|h| < q < |v|$ the following holds:*

$$|E(k,v,v)|^q = |\overline{E}(h(1,2),h,h)| \cdot |E(k,g,g)|^{q-|l|} = \prod_{r \in \mathbb{S}(h)} ((r(1,2)-r(1,1))+1)^{|r|} \cdot ((k-s_1(1,1))+1)^{q-(|l|+1)}$$

Proof. We prove this theorem by complete induction over l and $|v|$. The first basecase considers the case when $|v|=1$ and $l=0$. This implies that h is $(v,0)-$*perfect* and g is $(v,1)-$*perfect*. We know that $(v,0)-$*perfect* vectors are empty and $g=v$ because $|\mathbb{S}(s)|=1$. The fact that $|\mathbb{S}(s)|=1$ also implies that $v \in \mathbb{US}$. Thus, using Theorem 47, the statement reduces to the following:

$$|E(k,v,v)|^q = |E(k,g,g)|^q = ((k-g(1,1))+1)^{q-1}$$

When $|v|>1$ the same argument works because we choose k such that it is less than $g(1,2)$ and thus g behaves as if it is uniform standard even though it is standard. The case when $|v|=1$ and $l>0$ is not admitted by the theorem and thus is not considered. The induction hypothesis concerning $|v|$ is trivial in that the size of v does not directly influence the theorem statement. We needed to consider $|v|$ to construct a proper base case. Let us now assume that the theorem holds for all $0 \leq m \leq l < |\mathbb{S}(v)|-1$ and show

that it holds for $l+1$. Let us assume that h is $(v, l+1)-$*perfect* in the following statement and that k, q, g,and w are picked accordingly,

$$|E(k,v,v)|^q = |\overline{E}(h(1,2),h,h)| \cdot |E(k,g,g)|^{q-|l|}$$

We know, by Lemma 52, that $|\overline{E}(h(1,2),h,h)| = |\overline{E}(h'(1,2),h',h')| \cdot |\overline{E}(h(1,2),s',s')|$ where h' is $(v,l)-$*perfect* and $s' \in \mathbb{S}(h)$, such that $h = h' \otimes s'$. We can also derive, with the help of Lemma 39, that

$$E(k,g,g) = \bigcup_{g' \in E(k',g,g)} e(k, e(k-1, \cdots e(k'+1, g', g) \cdots, g), g)$$

where $h'(|h'|,2) \leq k' \leq h(|h|,2)$. If we are evaluating over k', $0 < |E(k',s',s')|$, Thus, We need to consider a vector $r = s' \otimes g$ instead of g. for the union,

$$E(k,g,g) = \bigcup_{g' \in E(k',g,g)} e(k, e(k-1, \cdots e(k'+1, g', g) \cdots, g), g)$$

and thus, we have

$$E(k,r,r) = \bigcup_{g' \in E(k',r,r)} e(k, e(k-1, \cdots e(k'+1, g', r) \cdots, r), r)$$

Also, we do not need the evaluation of positions greater than k' being that they don't correspond to the theorem statement,

$$E(k',r,r) = \bigcup_{g' \in E(k',r,r)} g'$$

Putting everything together we get $E(k',r,r)^q = |\overline{E}(h'(1,2),h',h')| \cdot |E(k',r,r)|^{q-|h'|}$ which is exactly the theorem statement for l and thus, by induction, we have proven the theorem. □

Theorem 55. *Let $v, h_1 \cdots, h_{|\mathbb{S}(v)|}, g_1 \cdots, g_{|\mathbb{S}(v)|} \in \mathbb{SE}$ be non-empty and proper, $c, i \in \mathbb{N}$, such that $0 \leq c \leq v(1,1)$, $1 \leq i \leq |\mathbb{S}(v)|$, h_i is $(v,i)-$perfect, and $v = h_i \otimes g_i$. Then*

$$SR(c,v) = (v(1,1) - c) + P(v(1,1), v(1,2) - 1, v) + \sum_{i=1}^{|\mathbb{S}(v)|-1} \overline{E}(h_i(i,2), h_i, h_i) \cdot P(h_i(1,2), g_i(1,2) - 1, g_i)$$

Proof. This is a result of adding up all the possibilities of Theorem 54. □

Example 12. Using the standard evaluation vector of Example 10,

$$v = [(0,5,0),(0,6,0),(0,7,0)]$$

we get the following value for the runtime representation size when computing $SR(0,v)$.

$$SR(0,v) = P(0,4,v) + \sum_{i=1}^{2} \overline{E}(h_i(i,2), h_i, h_i) \cdot P(h_i(1,2), g_i(1,2) - 1, g_i) =$$

$$\sum_{i=0}^{4} \sum_{j=0}^{2} (i+1)^j + \overline{E}(5, [(0,5,0)], [(0,5,0)]) \cdot P(5, 5, [(0,6,0),(0,7,0)]) +$$

$$\overline{E}(6,[(0,5,0),(0,6,0)],[(0,5,0),(0,6,0)]) \cdot P(6,6,[(0,7,0)]) =$$

$$3+7+13+21+31+6*\left(\sum_{i=5}^{5}\sum_{j=0}^{1}(i+1)^j\right)+42*\left(\sum_{i=6}^{6}\sum_{j=0}^{0}(i+1)^j\right) =$$
$$75+42+42=159$$

An interesting point to make about the innermost sum of Theorem 55 is that it behaves the same as the equation for uniform standard evaluation vectors even though it contains multiple sections. This occurs because until the point when an interval ends, it behaves exactly the same as larger intervals. An unexpected observation concerning Theorem 56 is that a naive computation is a good approximation of the space requirements of standard evaluation vectors, even outperforming the more complex analysis found in (Cerna et al., 2016b).

Theorem 56. *Let $v \in \mathbb{SE}$ be non-empty and proper, and $c \in \mathbb{N}$ such that $0 \leq c \leq v(1,1)$. Then*

$$SR(c,v) \leq \prod_{i=1}^{j} I_i^{|s_i|} = O\left(I_j^{|v|}\right)$$

Proof. This is essentially replacing the partial evaluation with the degenerate set, obviously a worst case scenario. □

This result improves the $O\left(I_j^{2\cdot|v|}\right)$ space complexity bound presented in (Cerna et al., 2016b).

The results presented in this section lead to an interesting question: does Theorem 55 provide an upper bound for the runtime representation size of proper top-free evaluation vectors? We chose to study proper standard evaluation vector specifically because they seemed to be a worst case scenario, but this does not necessarily have to be the case. This issue is to be addressed in future work.

5. Conclusion

We have provided a new and more precise abstraction of the operational semantics of the LogicGuard core language (LogicGuard II, 2015) than what has been presented in previous work. From this abstraction we where able to derive a precise expression for the space requirements of an important set of monitor specifications, which can be represented by standard evaluation vectors. In the end, we are able to show that naively multiplying the size of the intervals provides a bound on the space requirements of this set of monitor specifications which is more accurate than the bound previously provided in (Cerna et al., 2016b), an unexpected result. As for future work, we have yet to analyse inverse standard evaluation vectors and find a relationship between top-free, inverse standard and standard evaluation vectors. We conjecture that the space requirements for inverse standard and standard evaluation vectors represent lower and upper bounds, respectively, for the space requirements of top-free evaluation vectors. If this is the case, the naive method provides a computationally efficient approximation of the space requirements while the work of Section 4.2 provides a more accurate alternative. Another direction for future work is to

weaken the constraint that the quantifier lower bounds must be equal for all quantifiers in the formula. In a soon to be published paper (Cerna et al., 2016a), we weakened the constraint on the lower bound, and in doing so, we were able to derived an algorithm for computing the runtime representation size for a larger set of monitor specification. It is still open whether or not there is a precise expression which works for any monitor specification expressible in the LogicGuard core language. We also expect that a deeper understanding of the operational semantics will lead to optimizations which reduce the memory footprint of a monitor specification at runtime as the history pruning analysis reduced the size of the stream buffer.

References

Allen, J. F., Nov. 1983. Maintaining Knowledge About Temporal Intervals. Commun. ACM 26 (11), 832–843.

Armoni, R., Fix, L., Flaisher, A., Gerth, R., Ginsburg, B., Kanza, T., Landver, A., Mador-Haim, S., Singerman, E., Tiemeyer, A., Vardi, M. Y., Zbar, Y., 2002. The ForSpec Temporal Logic: A New Temporal Property-Specification Language. In: TACAS '02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Vol. 2280 of Lecture Notes in Computer Science. Springer, Berlin, Germany, Grenoble, France, April 6–14, pp. 296–211.

Banieqbal, B., Barringer, H., 1987. Temporal Logic with Fixed Points. In: Banieqbal, B., Barringer, H., Pnueli, A. (Eds.), Temporal Logic in Specification. Vol. 398 of Lecture Notes in Computer Science. Springer, Altrincham, UK, April 8-10, pp. 62–74.

Bozzelli, L., Molinari, A., Montanari, A., Peron, A., Sala, P., 2016. Interval Temporal Logic Model Checking: The Border Between Good and Bad HS Fragments. In: Automated Reasoning: 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 – July 2, 2016, Proceedings. Springer International Publishing, pp. 389–405.

Büchi, J. R., 1960. Weak Second-Order Arithmetic and Finite Automata. Zeitschrift für mathematische Logik und Grundlagen der Mathematik 6, 66–92.

Cerna, D., October 2015a. Space Complexity of LogicGuard Revisited. Technical report, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria.

Cerna, D., May 2015b. Space Complexity of Operational Semantics for the LogicGuard Core Language. Technical report, Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz.

Cerna, D. M., Schreiner, W., Kutsia, T., 2016a. Predicting Space Requirements for a Stream Monitor Specification Language. In: Runtime Verification: 16th International Conference, RV 2016, Madrid, Spain, September 28-30, 2016. Proceedings. Springer International Publishing.

Cerna, D. M., Schreiner, W., Kutsia, T., 2016b. Space Analysis of a Predicate Logic Fragment for the Specification of Stream Monitors. In: Davenport, J. H., (ed.), F. G. (Eds.), Proceedings of The 7th International Symposium on Symbolic Computation in Software Science. Vol. 39 of EPiC Series in Computing. pp. 29–41.

Finkbeiner, B., Kuhtz, L., 2009. Monitor Circuits for LTL with Bounded and Unbounded Future. In: Runtime Verification, 9th International Workshop, RV 2009. Vol. 5779 of Lecture Notes in Computer Science. Springer, Berlin, Grenoble, France, June 26–28, pp. 60–75.

Frick, M., Grohe, M., 2004. The Complexity of First-Order and Monadic Second-Order Logic Revisited. Annals of Pure and Applied Logic 130 (1–3), 3–31.

Gottlob, G., Mar. 1995. Np trees and carnap's modal logic. J. ACM 42 (2), 421–457.

Halpern, J. Y., Shoham, Y., Oct. 1991. A Propositional Modal Logic of Time Intervals. Journal of the ACM (JACM) 38 (4), 935–962.

IEEE, 2007. IEEE Std 1850-2007: Standard for Property Specification Language (PSL).

Kupferman, O., Lustig, Y., Vardi, M. Y., 2006. On Locally Checkable Properties. In: Logic for Programming, Artificial Intelligence, and Reasoning, 13th International Conference, LPAR 2006. Vol. 5779 of Lecture Notes in Artificial Intelligence. Springer, Berlin, Germany, Phnom Penh, Cambodia, November 13–17, pp. 302–316.

Kutsia, T., Schreiner, W., 2014. Verifying the Soundness of Resource Analysis for LogicGuard Monitors (Revised Version). Technical Report 14-08, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria.

LogicGuard II, November 2015. LogicGuard II. `http://www.risc.jku.at/projects/LogicGuard2/`.

Maler, O., Nickovic, D., Pnueli, A., 2005. Real Time Temporal Logic: Past, Present, Future. In: Pettersson, P., Yi, W. (Eds.), Formal Modeling and Analysis of Timed Systems, Third International Conference (FORMATS). Vol. 3829 of Lecture Notes in Computer Science. Springer, Berlin, Germany, Uppsala, Sweden, September 26–28, pp. 2–16.

McNaughton, R., Papert, S., 1971. Counter-Free Automata. Vol. 65 of Research Monograph. MIT Press, Cambridge, MA, USA.

Molinari, A., Montanari, A., Peron, A., 2015. A Model Checking Procedure for Interval Temporal Logics based on Track Representatives. In: Kreutzer, S. (Ed.), 24th EACSL Annual Conference on Computer Science Logic (CSL 2015). Vol. 41 of Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 193–210.

Rosu, G., Bensalem, S., 2006. Allen Linear (Interval) Temporal Logic - Translation to LTL and Monitor Synthesis. In: Ball, T., Jones, R. B. (Eds.), Computer Aided Verification, 18th International Conference, (CAV). Vol. 4144 of Lecture Notes in Computer Science. Springer, Berlin, Germany, Seattle, WA, USA, August 17–20, pp. 263–277.

Schnoebelen, P., 2003a. Oracle circuits for branching-time model checking. In: Baeten, J. C. M., Lenstra, J. K., Parrow, J., Woeginger, G. J. (Eds.), Automata, Languages and Programming: 30th International Colloquium, ICALP 2003 Eindhoven, The Netherlands, June 30 – July 4, 2003 Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 790–801.

Schnoebelen, Ph., 2003b. The Complexity of Temporal Logic Model Checking. In: Balbiani, Ph., Suzuki, N.-Y., Wolter, F., Zakharyaschev, M. (Eds.), Selected Papers from the 4th Workshop on Advances in Modal Logics (AiML'02). King's College Publication, Toulouse, France, pp. 393–436, invited paper.

Schreiner, W., Kutsia, T., Cerna, D., Krieger, M., Ahmad, B., Otto, H., Rummerstorfer, M., Gössl, T., November 2015. The LogicGuard Stream Monitor Specification Language (Version 1.01). Tutorial and reference manual, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria.

Vardi, M. Y., Wolper, P., 1986. An Automata-Theoretic Approach to Automatic Program Verification (Preliminary Report). In: Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18. IEEE Computer Society, pp. 332–344.