EPiC Series in Computing

Volume 39, 2016, Pages 1–13

SCSS 2016. 7th International Symposium on Symbolic Computation in Software Science

Space Analysis of a Predicate Logic Fragment for the Specification of Stream Monitors*

David M. Cerna, Wolfgang Schreiner, and Temur Kutsia

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
{David.Cerna,Wolfgang.Schreiner,Temur.Kutsia}@risc.jku.at

Abstract

We analyze the space complexity of monitoring streams of messages whose expected behavior is specified in a fragment of predicate logic; this fragment is the core of the LogicGuard specification language that has been developed in an industrial context for the runtime monitoring of network traffic. The execution of the monitors is defined by an operational semantics for the step-wise evaluation of formulas, which requires the preservation of formula instances in memory until their truth value can be determined. In the presented work, we analyze the number of instances that have to be preserved over time for a significant fragment of the core language that involves only "future looking quantifiers"; this lays the foundations for the space analysis of the entire core language.

1 Introduction

The goal of *runtime verification* is to check at runtime whether the execution of a system satisfies a formal specification. For this purpose, the specification is translated into an operational representation that monitors the system and stops its execution or reports an error, if a violation of the specification is detected. This approach results in a trade-off between the expressiveness of a specification formalism and the efficiency of monitoring corresponding specifications. Often these formalisms are based on (timed extensions of) linear temporal logic (LTL) [11] that provide effective monitoring strategies, although at the price of limited expressiveness.

In contrast, the LogicGuard project [10] aims at a framework for monitoring event streams (messages flowing through a network) that provides a declarative specification language with great expressiveness (for describing security properties of message streams): in fact, the LogicGuard language [13] is based on predicate logic (where quantifiers bind variables that represent stream positions), set theory (with stream constructions that mimic set builder notation), limited arithmetic (quantifiers to compute the minimum/maximum position respectively the number of positions where a property holds) and general computational facilities (stream folds).

It is challenging to make the operation of monitors arising from such specifications effective and efficient. For this purpose, we have devised a static analysis that determines whether

*The project "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.

a specification gives rise to a monitor that is able to operate with only a finite set of past messages in memory; in [9] we proved the soundness of this analysis and the corresponding "history pruning" optimization. That proof was based on a simplified core of the language for which a formal operational semantics was devised. In this paper, we present first steps towards a complementary analysis that determines the space requirements of the monitor operation itself: every quantified formula of the monitor has to preserve in memory (runtime representations of) those formula instances whose results could not yet be determined because they still depend on future messages. Together with the history buffer, this measure determines the memory requirements of the monitor (and also the time required for processing every message).

The core of the LogicGuard language could be considered as Monadic First-Order Logic (MFO) [12] which is equivalent to LTL and captures the class of star-free languages; however, several aspects of the full language (e.g., general computations) exceed this framework and might be more adequately described by Monadic Second-Order Logic (MSO) [2] which captures the class of omega-regular languages. Most complexity results with respect to MFO/LTL and MSO refer to the size (number of states) of the non-deterministic Büchi automaton that accepts the language of a formula: for MFO/LTL, this size is in the worst case exponential in the formula size [14]. This result is directly applicable to automata-based model checking where this automaton is indeed constructed; the construction of a corresponding deterministic automaton (as is necessary for runtime checking) causes another exponential blow-up. Since for MSO, the size of the accepting automaton is in general even non-elementary in the formula size [6], MSO has been often neglected as a "practical" specification language.

Most investigations on improving the use of these logics for model checking or runtime verification thus focus on specific subsets. The hardware design language PSL [7] which is based on LTL defines a "simple subset" that restricts the general use of disjunction in a specification to avoid exponential blowup. In [8], the class of "locally checkable" properties (a subclass of the "locally testable" properties introduced in [12]) is defined that are satisfied by a word, 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 [5] 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.

Our work differs from this related research in that 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; it is this measure that we are interested in. The reason for this approach is that all quantified phrases are subject to the same evaluation mechanism; some (stream folds) only coordinate computations performed by external code. Any encoding of these evaluations by automata would obscure more than highlight the complexity measures that are relevant for our framework (indeed it is not entirely clear how these measures refer to the automata sizes). It should be also noted that our analysis does not only give rise to asymptotic bounds but to concrete complexity functions, although these functions still suffer from overestimation. Ongoing work investigating optimizations for the complexity functions shows promise in reducing the overestimation and providing more accurate bounds than the proposed asymptotic bounds.

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. Section 3 presents a two-quantifier-nested fragment of this core language; monitors in this language can be abstracted into triples of natural numbers. Section 4 defines on such triples an evaluation relation that

mimics the interesting aspects of the monitor operation in the sense that the relation preserves an upper bound for the memory usage of monitor execution. Based on this relation, Section 5 analyzes the space complexity of triple evaluation. Section 6 extends the results from the two-quantifier-nested fragment to the general core language by the translation of a monitor to a dominating formula whose evaluation provides an upper bound for the memory usage of the original monitor. Section 7 concludes by judging the presented analysis and outlining current work that aims to improve its accuracy.

2 The Core Language and its Operational Semantics

We consider the core language presented in Fig. 1 (l) (a simplified version of the language given in [9]). A specification in this language describes a monitor that processes a stream of Boolean values indicated by $\top$ (true) and $\bot$ (false). The various constructs are to be interpreted as follows: the atomic predicate $@V$ denotes the value in the stream at the position denoted by the variable V, $\neg F$ denotes the negation of the formula F, 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 $[B_1, B_2]$. We assume that a monitor constructed from the core language has no free variables and that we assume each bound variable uses distinct names. Unless otherwise noted x will be the variable bound by the quantifier $\forall_{0\leq x} : F$ and will be referred to as the *stream variable*.

The grammar depicted in Fig. 1 (l) uses the typed variables $M, F, \ldots$ to denote elements of the syntactic domains $\mathbb{M}, \mathbb{F}, \ldots$ Similarly, in Fig. 1 (2), $\mathbf{M}, \mathbf{F}, \ldots$ denote elements of the corresponding runtime domains $\mathcal{M}, \mathcal{F}, \ldots$ to which the syntactic domains are translated. Here we use the domain $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$ over which 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 $\infty \pm n = \infty$. The notions $\mathbb{P}(S)$ and $A \rightarrow^{part.} B$ are to be interpreted as the powerset of S and the set of partial mappings from A to B, respectively. The definition of $\mathbf{C}$ in Fig. 1 (2) assigns both a position to the variable as well as the boolean value at that position. This choice was made to extract the referenced stream value from the stream into the formula such that the history pruning optimization [9] may be safely applied.

A monitor $M \in \mathbb{M}$ is translated by a function $T : \mathbb{M} \to \mathcal{M}$ into its runtime representation $\mathbf{M} = T(M)$ which stores information concerning the current state of the evaluation, in particular the set of instances $\mathbf{I}$ of the body F of M which could not yet be evaluated to $\top$ or $\bot$. The definition of the translation function given in Fig. 2 is based on two auxiliary functions $T^F : \mathbb{F} \to \mathcal{F}$ and $T^B : \mathbb{B} \to \mathcal{B}$ which translate formulas and bounds, respectively. By $c.1$ in Fig. 2 we are referring to accessing the positions in the tuple of Fig. 1(2) for $\mathbf{C}$.

The evaluation of a monitor's runtime representation is formalized by a small-step operational semantics with a 6-ary transition relation $\rightarrow \subseteq \mathcal{M} \times \mathbb{N} \times \{\top, \bot\}^\omega \times \{\top, \bot\} \times \mathbb{P}(\mathbb{N}) \times \mathcal{M}$. In a transition $\mathbf{M} \rightarrow_{p,MS,m,RS} \mathbf{M}'$, p is the index of the next message m arriving on the stream, MS denotes the sequence of messages that have previously arrived (by $MS(t)$ we denote the message at position t) and RS (meaning 'reported set') denotes the set of violations determined by the monitor transition, i.e., those positions that when assigned to the stream variable x make the monitor body F false.

The operational semantics defines a monitor transition $\forall^{IS}_{0\leq x} : f \rightarrow_{p,MS,m,RS} \forall^{IS'}_{0\leq x} : f$ from the current state with instance set IS (initially $\emptyset$) to a state with instance set IS' by the

$$
\begin{array}{lcl}
M & ::= & \forall_{0\leq V} : F. \\
F & ::= & @V \big| \neg F \big| F \ \& \ F \big| \forall_{V\in[B,B]} : F. \\
B & ::= & \infty | 0 | V | B \pm N \ . \\
V & ::= & x | y | z | \ldots . \\
N & ::= & 0 | 1 | 2 | \ldots .
\end{array}
\qquad
\begin{array}{lcl}
\mathbf{M} & ::= & \forall^{\mathbf{I}}_{0\leq V} : \mathbf{F} \\
\mathbf{F} & ::= & \mathbf{d}(\mathbf{Bool}) \mid \mathbf{n}(\mathbf{G}) \\
\mathbf{G} & ::= & @V \mid \neg \mathbf{F} \mid \mathbf{F} \ \& \ \mathbf{F} \mid \forall_{V\in[\mathbf{B},\mathbf{B}]} : \mathbf{F} \mid \\
 & & \forall_{V\in[\mathbb{N}^\infty,\mathbb{N}^\infty]} : \mathbf{F} \mid \forall^{\mathbf{I}}_{V\leq\mathbb{N}^\infty} : \mathbf{F} \\
\mathbf{B} & ::= & \mathbf{C} \to \mathbb{N}^\infty \\
\mathbf{I} & ::= & \mathbb{P}(\mathbb{N} \times \mathbf{F} \times \mathbf{C}) \\
\mathbf{C} & ::= & (V \to^{part.} \mathbb{N}) \times \\
 & & (V \to^{part.} \{\top, \bot\}) \\
\mathbf{Bool} & ::= & \top \mid \bot
\end{array}
$$

Figure 1: The core language (l) and its runtime representation (2).

$$
\begin{array}{rcl}
T(\forall_{0\leq V} : F) & := & \forall^{\emptyset}_{0\leq V} : T^F(F) \\
T^F(@V) & := & \mathbf{n}(@V) \\
T^F(\neg F) & := & \mathbf{n}(\neg T^F(F)) \\
T^F(F_1 \ \& \ F_2) & := & \mathbf{n}(T^F(F_1) \ \& \ T^F(F_2)) \\
T^F(\forall_{V\in[B_1,B_2]} : F) & := & \forall_{V\in[T^B(B_1),T^B(B_2)]} : T^F(F)
\end{array}
\qquad
\begin{array}{rcl}
T^B(0)(c) & := & 0 \\
T^B(\infty)(c) & := & \infty \\
T^B(V)(c) & := & \begin{cases} c.1(V) & V \in dom(c.1) \\ 0 & \text{otherwise} \end{cases} \\
T^B(b \pm n)(c) & := & T^B(b)(c) \pm \bar{n}
\end{array}
$$

Figure 2: The translation functions.

following definitions:

$$
\begin{gathered}
c = ((x,p),(x,m)), \quad IS_0 = IS \cup \{(p,f,c)\}, \\
RS = \{t \in \mathbb{N} \mid \exists g \in \mathcal{F}, k \in \mathcal{C} : (t,g,k) \in IS_0 \wedge \ \vdash g \to_{p,ms,m,k} \mathbf{d}(\bot)\}, \\
IS' = \{(t, \mathbf{n}(h), k) \in \mathcal{I} \mid \exists g \in \mathcal{F} : (t,g,k) \in IS_0 \wedge \ \vdash g \to_{p,ms,m,k} \mathbf{n}(h)\}.
\end{gathered}
$$

Here we first construct a context c which maps variable x to position p and value m; this context is used to create a new monitor instance (p,f,c) which is added to IS yielding the new instance set IS_0. Next we construct RS which contains the positions of monitor instances which have evaluated to $\bot$. Finally, we determine the set IS' of those instances which have not yet evaluated to $\bot$ or $\top$. Here a term $\mathbf{d}(b)$ denotes the result of the evaluation of a formula to a value $b \in \{\top, \bot\}$ ($\mathbf{d}(\cdot)$ stands for 'done') while $\mathbf{n}(h)$ denotes a formula h whose value is still unknown ($\mathbf{n}(\cdot)$, stands for next, indicates what will be considers by the next application of the operational semantics).

This definition of the monitor transition relation is based on a formula transition relation $\to \subseteq \mathcal{F} \times \mathbb{N} \times \{\top, \bot\}^\omega \times \{\top, \bot\} \times \mathbf{C} \times \mathcal{F}$. In a formula transition $f \to_{p,MS,m,c} f'$, the additional arguments p, MS, and m are defined as for the monitor transition and c is the context for the current given monitor instance. When they are clear from the environment, we drop these arguments and simply write $f \to f'$. The formula transition relation is described by the rules depicted in Fig. 3 for which we use the following definitions:

$$
\begin{gathered}
IS^f = \{(p_0, f, c[((y,p_0),(y, MS(p_0 + p - |MS|)))]) \mid p_1 \leq p_0 \leq \min\{p_2 + 1, p\}\}, \\
IS^f_0 = \begin{cases} IS^f & \text{if } p_2 < p \\ IS^f \cup (p, f, c[((y,p),(y,m))]) & \text{otherwise} \end{cases}, \\
IS^f_1 = \left\{(t, \mathbf{n}(h), k) \in \mathcal{I} \mid (t,g,k) \in IS^f_0 \wedge \ \vdash g \to \mathbf{n}(h)\right\}, \\
DF = \exists t \in \mathbb{N}, g \in \mathcal{F}, k \in \mathcal{C} : (t,g,k) \in IS^f_0 \wedge \ \vdash g \to \mathbf{d}(\bot).
\end{gathered}
$$

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

Figure 3: The operational semantics of formula evaluation.

The set IS^f used in rules Q4–7 denotes the set of all instances of a quantified formula which can be created by assigning to the bound variable y positions up to p. By $MS(\cdot)$, we mean access to a given position in the stream of messages. The set IS^f_0 used to define IS^f_1 and DF is essentially the same as IS^f, except that it is updated with a new instance when position p is reached. The set IS^f_1 used in Q6 and Q7 contains all instances which cannot yet be evaluated. Finally the formula DF checks to see if some instance in IS^f_0 evaluates to $\bot$.

We illustrate this operational semantics by an example (for more details, see [9]).

Example 1. *Take monitor* $M_0 = \forall_{0\leq x} : \forall_{y\in[x+1,x+2]} : @x \ \& \ @y$*, which states that the current position of the stream is true as well as the next two future positions. The runtime form of* M_0 *is* $\forall^{\emptyset}_{0\leq x} : F_0$ *where* $F_0 = \forall_{y\in[T^B(x+1),T^B(x+2)]} : \mathbf{n}(\mathbf{n}(@x) \ \& \ \mathbf{n}(@y))$*. Operating on a stream* $\langle \top, \top, \bot, \ldots \rangle$*, the first step of the evaluation is* $\forall^{\emptyset}_{0\leq x} : F_0 \to_{0,\langle\rangle,\top,\emptyset} \forall^{IS^0}_{0\leq x} : F_0$*. Here*

$$IS^0 = \big\{(0, \mathbf{n}(\forall_{y\in[1,2]} : \mathbf{n}(\mathbf{n}(@x) \ \& \ \mathbf{n}(@y))), [((x,0),(x,\top))])\big\}$$

is derived by the rules Q2 and Q3 (which are applicable because the stream position 0 *is less than the lower bound* 1*). The next step is* $\forall^{IS^0}_{0\leq x} : F_0 \to_{1,\langle\top\rangle,\top,\emptyset} \forall^{IS^1}_{0\leq x} : F_0$ *where*

$$IS^1 = \big\{(1, \mathbf{n}(\forall_{y\in[2,3]} : \mathbf{n}(\mathbf{n}(@x) \ \& \ \mathbf{n}(@y))), [((x,1),(x,\top))]),\\ (0, \mathbf{n}(\forall^{\emptyset}_{y\leq 2} : \mathbf{n}(\mathbf{n}(@x) \ \& \ \mathbf{n}(@y))), [((x,0),(x,\top))])\big\}$$

The quantifier from the instance in IS^0 *can now be unrolled, and the matrix is evaluable to* $\top$ *using rules A1, C1, and C3, hence the instance set* $\emptyset$ *in* $\mathbf{n}(\forall^{\emptyset}_{y\leq 2} : \mathbf{n}(\mathbf{n}(@x) \ \& \ \mathbf{n}(@y)))$*. The*

new instance is the same as the instance in IS^0 *but the positions are shifted by* 1. *The next step is* $\forall^{IS^1}_{0\leq x} : F_0 \rightarrow_{2,\langle\top,\top\rangle,\bot,\{0,1\}} \forall^{IS^2}_{0\leq x} : F_0$ *where*

$$IS^2 = \left\{(2, \mathbf{n}(\forall_{y\in[3,4]} : \mathbf{n}(\mathbf{n}(@x) \ \& \ \mathbf{n}(@y))), [((x,2),(x,\bot))])\right\}$$

The quantifier from the first and second instance evaluated to $\bot$ *by rule* $Q5$, *thus* $RS = \{0,1\}$. *Again, there is a shifting of the remaining instances.*

3 The Restricted Fragment and $\mathbb{N}$-Triples

When concerning ourselves with the worst-case space complexity of the operational semantics defined in Sec. 2, we may ignore the actual result of a monitor evaluation but may focus on the general structure of its evaluation. Thus we can greatly simplify the operational semantics to perform the necessary analysis. This simplification also extends to the formula structure, where we can ignore the propositional structure for the most part. However, what cannot be ignored is the construction of the instance set. Essentially, the instance set grows when an instance contains a quantifier whose interval upper bound is ∞ or the quantifier's evaluation requires a message at a future position.

Rather than performing our analysis on the global structure of a given monitor, we will perform the analysis on the local structure around a given quantifier. Once we have a space analysis for the local structure we can then apply the analysis to the global structure recursively over the formula structure. Dealing with constant bounds, negative shifts, and ∞, while doable (see [4]), adds unneeded complexity to our analysis. Also, the space analysis of these cases is rather straightforward. We assume that the bounds of the quantifiers only contain the stream variable x from $\forall_{0\leq x} : F$. This assumption seems quite restrictive, but by assuming that monitors do not have free variables, essentially every variable's interval is defined by the stream variable. This fact is important for our analysis.

In this section, we work with a restricted version of the core language which differs in the definition of the bounds, i.e. $B ::= V|B + N$ is used. We refer to this language as the *variable only fragment* $\mathbb{M}^{vb}$ (the other domains are labelled in a similar way). Formulas of $\mathbb{M}^{vb}$ only contain quantifiers whose bounds are non-negative.

Definition 1 (Restricted Fragment)**.** *Let* $M \in \mathbb{M}^{vb}$, $F \in \mathbb{F}^{vb}$ *be a formula (without quantification) with two holes* $(F[\bullet, \star])$ *, one for a formula* $(\bullet)$ *and one for a variable* $(\star)$, *and* $F' \in \mathbb{F}^{vb}$, *without quantification, be a formula with three holes* $(F'[\bullet, \star_1, \star_2])$, *one for a formula* $(\bullet)$ *and two for variable* $(\star_1, \star_2)$. *Then* $M \in \mathbb{M}^{vb}_{\mathbb{N}}$, *if it can be expressed in the form*

$$M \equiv \forall_{0\leq x} : F[\forall_{y\in[x+a,x+b]} : F'[\forall_{z\in[x+c,x+c]} : @z, x, y], x]$$

where $a, b, c \in \mathbb{N}$, $a \leq b$, *and* $x, y, z \in \mathbb{V}$. *We assume that* F *contains only one occurrence of* $\bullet$.

The idea behind Def. 1 is that the quantifier binding y is at a local position in a more complex formulas f and the quantifier binding z represents the furthest future position which can be assigned to an atom located in the matrix of the quantifier binding y. We don't consider a higher nesting of quantifiers in this work being that we only want to represent the *local environment* around a given quantifier in order to use the results for this local environment as a induction invariant later on in this work. Take for example a monitor $m = \forall_{0\leq x} : \forall_{w\in[x,x+5]} : @x \ \& \ \forall_{y\in[x+1,x+2]} : @y \ \& \ @w$. Consider the local position of the quantifier binding y, we can write it in the form of Def. 1 as $m' = \forall_{0\leq x} : F\left[\forall_{y\in[x+1,x+2]} : F'\left[\forall_{z\in[x+5,x+5]} : @w, x, y\right], x\right]$,

where $F[\bullet, \star] = @\star \ \& \bullet$ and $F[\bullet, \star_1, \star_2] = @\star_2 \ \& \bullet$. Though we provide a particular F and F', the point of this abstraction is to make the propositional context surrounding the quantifier frivolous.

What is fruitful about this abstraction is that m' is really defined by three values. Following the notation of Def. 1, a monitor in $\mathbb{M}_{\mathbb{N}}^{vb}$ can be expressed as $\mathbb{N}$-*triples* $\langle a, b, c\rangle_{\mathbb{N}}$. Triples of the form $\langle a, a, c\rangle_{\mathbb{N}}$ as *singleton triples*. They are essentailly a tuple of three natural numbers. An *instantiation of a triple*, i.e. $\langle a, b, c\rangle_{\mathbb{N}}(n)$, for $n \in \mathbb{N}$, essentially assigns a position to the stream variable. Rather than evaluating monitors as outlined in Sec. 2, we define in Sec. 4 a greatly reduced evaluation procedure over $\mathbb{N}$-triples that specifically highlights the instance set construction.

4 The Evaluation of $\mathbb{N}$-Triples

Unlike the operational semantics outlined in Sec. 2, we mix the evaluation of monitors with the evaluation of formulas. The restricted fragment is simple enough that this simplification of the evaluation procedure does not cause any issues. Essentially, our evaluation procedure is a simplified version of the rules for evaluating quantifiers (Q1-7 in Fig. 3). The idea is that instances of the triples are evaluated over a fragment of the external stream with a start point α and an end point at β. In this setting, consider $\triangleright$ to be a symbol corresponding to one place to the left of α, i.e. the start of the stream.

Definition 2. *An* evaluation structure *is a tuple of the form* $[n, t, \mathbf{I}]$*, where* $n \in \mathbb{N} \cup \{\triangleright\}$*,* t *is an* $\mathbb{N}$*-triple and* $\mathbf{I}$ *is a set of instances of* t*. When* $n = \triangleright$*, then the structure is called a* start evaluation structure *and* $\mathbf{I} \equiv \emptyset$*.*

Now we move on to transitions between two evaluation structures.

Definition 3. *Given an evaluation structure* $[n, t, \mathbf{I}]$*, and an interval* $[\alpha, \beta]$*, where* $n \in [\alpha, \beta) \cup \{\triangleright\}$ *, we have the* evaluation transition $[n, t, \mathbf{I}] \xrightarrow{[\alpha,\beta]} [n+1, t, \mathbf{I}']$ *, where*

$$\mathbf{I}' = ((\mathbf{I}^{n+1} - \mathbf{I}_u^{(n+1)}) \cup L_{n+1} \cup U_{n+1}) - \mathbf{I}_1^{(n+1)}$$

and the various sets are defined as follows: $\mathbf{I}^{n+1} = \mathbf{I} \cup \{t(n+1)\}$ *is the previous instance set plus the new instance of* t*. The set*

$$\mathbf{I}_u^{(n+1)} = \left\{ \ (\mathbf{a}, i) \ \middle| \ \exists a', b, c, \gamma : \mathbf{a} = \langle a', b, c\rangle_{\mathbb{N}}(\gamma) \in \mathbf{I}^{n+1} \ \wedge \ a' \leq i \leq \ b \ \wedge \ i + \gamma = n + 1 \ \right\}$$

contains the instance/position pairs (a, i)*, where the instance* $\mathbf{a}$ *can be partially* unrolled *up to position* i*. The set*

$$L_{n+1} = \bigcup_{(\mathbf{a},i)\in\mathbf{I}_u^{(n+1)}} \{(\langle a', a', c\rangle_{\mathbb{N}}(n+1-i) \mid a' \in L[(\mathbf{a}, i)]\}.$$

contains the singleton triple instances derived from the unrolling of instances of $\mathbf{I}_u^{(n+1)}$*. The set* $L[(\langle a, b, c\rangle_{\mathbb{N}}(\gamma), i)]$ *consist of all* $w \in \mathbb{N}$ *such that* $a \leq w \leq i \leq b$*. The set*

$$U_{n+1} = \bigcup_{(\langle a',b,c\rangle_{\mathbb{N}}(\gamma),i)\in\mathbf{I}_u^{(n+1)}} \{\langle i+1, b, c\rangle_{\mathbb{N}}(\gamma) \mid a' < i+1 < b\}$$

contains the instances in $\mathbf{I}_u^{(n+1)}$ *after every singleton triple instance below position* i *has been removed. Notice that given an* $\mathbb{N}$*-triple instance* $\langle a, b, c\rangle_{\mathbb{N}}(\gamma)$ *which can be unrolled up to* b*, the singleton* $\mathbb{N}$*-triple instance* $\langle b, b, c\rangle_{\mathbb{N}}(\gamma)$ *is in* L_{n+1} *and not in* U_{n+1}*. The set*

$$\mathbf{I}_1^{(n+1)} = \left\{ \mathbf{a} \;\middle|\; \exists a', c, \gamma : \mathbf{a} = \langle a', a', c\rangle_{\mathbb{N}}(\gamma) \in \mathbf{I}_0^{(n+1)} \wedge \max\{a', c\} + \gamma \leq n+1) \right\}$$

contains all $\mathbb{N}$*-triple instances which can be evaluated given the new external stream position. By* $\mathbf{I}_0^{(n+1)}$ *we refer to the set* $(\mathbf{I}^{n+1} - \mathbf{I}_u^{(n+1)}) \cup L_{n+1} \cup U_{n+1}$*.*

The idea behind Def. 3 is to mimic the operation semantic's treatment of quantifiers for triples. Using Def. 3, we can define *evaluation chains.*

Definition 4 (Evaluation Chain)**.** *Given a* $\mathbb{N}$*-triple* t*, an interval* $[\alpha, \beta]$*, a* complete proper evaluation chain *is a sequence of evaluation steps starting at* $\triangleright$ *and ending at* β*.*

$$[\triangleright, t, \emptyset] \xrightarrow{[\alpha,\beta]} \cdots [n, t, \mathbf{I}] \xrightarrow{[\alpha,\beta]} [n+1, t, \mathbf{I}'] \xrightarrow{[\alpha,\beta]} \cdots \xrightarrow{[\alpha,\beta]} [\beta, t, \mathbf{I}'']$$

The chain is considered to be a complete improper evaluation chain *if* $\beta = \infty$*.*

In the work presented here, we are interested in evaluation over complete improper evaluation chains. We want to show that after a finite segment of the interval the size of the instance set is constant.

Example 2. *Let us consider the monitor from Ex. 1,* $M_0 = \forall_{0\leq x} : \forall_{y\in[x+1,x+2]} : @x \;\&\; @y$*, which can be written as an* $\mathbb{N}$*-triple* $t = \langle 1, 2, 2\rangle_{\mathbb{N}}$*. One way of writing this monitor in the form of the restricted fragment is*

$$\forall_{0\leq x} : F\left[\forall_{y\in[x+1,x+2]} : F\left[\forall_{z\in[x+2,x+2]} : @z, x, y\right], x\right]$$

where $F = \bullet$ *and* $F' = @_{\star_1} \;\&\; \bullet$*. We replace the* y *with the quantifier representing the furtherest point in the future being that the furtherest point in the future is from the variable* y*. Consider the following proper evaluation chain*

$$[\triangleright, t, \emptyset] \xrightarrow{[0,2]} [0, t, \mathbf{I}^0] \xrightarrow{[0,2]} [1, t, \mathbf{I}^1] \xrightarrow{[0,2]} [2, t, \mathbf{I}^2]$$

where the instance sets are $\mathbf{I}^0 = \{\langle 1, 2, 2\rangle_{\mathbb{N}}(0)\}$*,* $\mathbf{I}^1 = \{\langle 1, 1, 2\rangle_{\mathbb{N}}(0), \langle 2, 2, 2\rangle_{\mathbb{N}}(0), \langle 1, 2, 2\rangle_{\mathbb{N}}(1)\}$*, and* $\mathbf{I}^2 = \{\langle 1, 1, 2\rangle_{\mathbb{N}}(1), \langle 2, 2, 2\rangle_{\mathbb{N}}(1), \langle 1, 2, 2\rangle_{\mathbb{N}}(2)\}$*.*

Notice how in Ex. 2 there are more instances in memory than in Ex. 1 even though the same formula is being evaluated. The reason for this is that we made the worst case assumption that every instance of a quantifier will be blocked by the furthest in the future position assigned to a variable used in the matrix; however this need not be the case, as Ex. 1 shows. More formally, the relationship between the operational semantics of Sec. 2 and the evaluation of $\mathbb{N}$-triples of Sec. 3 is described by the following theorem.

Theorem 1. *Let* $\forall_{0\leq x} : f \in \mathbb{M}^{vb}$ *be a monitor in which* f *contains at most two quantifiers (when* f *contains two quantifiers, they are nested). Let* t_m *be a* $\mathbb{N}$*-triple corresponding to the structure of* $\forall_{0\leq x} : f$*. Let* $\forall_{0\leq x}^{\emptyset} : f$ *be the runtime form of* $\forall_{0\leq x} : f$*. Then given*

$$\forall_{0\leq x}^{\emptyset} : f \to_{\alpha, MS, m, RM} \cdots \to_{\alpha+n, MS, m, RM} \forall_{0\leq x}^{\mathbf{IS}} : f$$

and

$$[\triangleright, t_m, \emptyset] \xrightarrow{[\alpha,\infty)} \cdots \xrightarrow{[\alpha,\infty)} [\alpha + n, t_m, \mathbf{I}],$$

for $\alpha, n \in \mathbb{N}$*, it is the case that* $|\mathbf{IS} + \mathbf{IS}_q| \leq |\mathbf{I}|$*, where* $\mathbf{IS}_q$ *is the set of instances held in the quantifier instances of the monitor.*

Proof. This trivially holds because the constraints on quantifier evaluation are stronger for evaluation structures than they are for the step transition of the operational semantics. Equality of the cardinality of the instance sets can occur when the monitors match the syntactic constraints of the restricted fragment. □

5 Space Analysis of the Restricted Fragment

In this section we provide a space analysis of restricted fragment by first analyzing three types of $\mathbb{N}$-triples, namely , $\langle 0, b, b\rangle_{\mathbb{N}}$, $\langle 0, b, b + c\rangle_{\mathbb{N}}$, $\langle 0, b, b - c\rangle_{\mathbb{N}}$, and then putting the results together to provide an analysis for all $\mathbb{N}$-triples. We chose these $\mathbb{N}$-triples because $\langle 0, b, b\rangle_{\mathbb{N}}$ is a foundational form, i.e. the evaluation of the quantifier is held up by the upper bound of the quantifier's interval. Shifting by c represents the influence of the global structure on the quantifier's evaluation. Also, we set the first component of the triples to zero because shifting of the quantifier's interval only makes things unnecessarily complex and obfuscates the underlying structure of the instance set.

Theorem 2. *Given an $\mathbb{N}$-triple $t = \langle 0, b, b\rangle_{\mathbb{N}}$, for $b \in \mathbb{N}$, and an interval $[\alpha, \infty)$, there exists a value $x \in [\alpha, \infty)$ such that for all $x \leq \beta$, the evaluation chain*

$$[\triangleright, t, \emptyset] \xrightarrow{[\alpha,\infty)} \cdots \xrightarrow{[\alpha,\infty)} [x-1, t, \mathbf{I}_0] \xrightarrow{[\alpha,\infty)} [x, t, \mathbf{I}_1] \xrightarrow{[\alpha,\infty)} \cdots \xrightarrow{[\alpha,\infty)} [\beta, t, \mathbf{I}_{\beta-x+1}] \xrightarrow{[\alpha,\infty)} \cdots$$

has the property $|\mathbf{I}_0| \neq |\mathbf{I}_1| = \cdots = |\mathbf{I}_{\beta-x+1}|$.

Proof (sketch). The key to proving this theorem is finding the value of x; we can derive $x = \alpha + b - 1$. If one closely observes the construction of the sets in Def. 3, one will notice that $\mathbf{I}_1^{(\gamma)} \equiv \emptyset$ for $\gamma \in [\alpha, \alpha + b - 1]$. This implies that the instance set is at least growing until this point. At $\alpha + b$, the first position such that $\mathbf{I}_1^{(\alpha+b)} \not\equiv \emptyset$, there is the possibility for the instance set to stay constant. One can easily check as a basecase for an induction that $|\mathbf{I}_1| \equiv |\mathbf{I}_2|$ using the notation provided in the evaluation chain of the theorem statement. The argument for the base case can be generalized to show the stepcase of the induction. For a more detailed proof, see Appendix **??**. □

The following corollary to Thm. 2 concerns how large the instance set becomes.

Corollary 1. *Given an $\mathbb{N}$-triple $t = \langle 0, b, b\rangle_{\mathbb{N}}$, for $b \in \mathbb{N}$, evaluated over the evaluation chain*

$$[\triangleright, t, \emptyset] \xrightarrow{[\alpha,\infty)} [\alpha, t, \mathbf{I}_\alpha] \xrightarrow{[\alpha,\infty)} [\alpha + 1, t, \mathbf{I}_{\alpha+1}] \xrightarrow{[\alpha,\infty)} \cdots$$

then,

$$|\mathbf{I}_n| \leq \frac{(b+1) * (b+2)}{2} - 1$$

for all $n \in [\alpha, \infty]$.

Proof (sketch). Count the instances added to the instance set by unrolling prior to position $\alpha + b - 1$. □

Thus we derive a worst case space complexity of $O(b^2)$ for the size of the instance set.

Theorem 3. *Given an* $\mathbb{N}$*-triple* $t = \langle 0, b, b+c\rangle_{\mathbb{N}}$*, for* $b, c \in \mathbb{N}$*, and an interval* $[\alpha, \infty)$*, there exists a value* $x \in [\alpha, \infty)$ *such that for all* $\beta \in \mathbb{N}$*, the evaluation chain*

$$[\triangleright, t, \emptyset] \xrightarrow{[\alpha,\infty)} \cdots \xrightarrow{[\alpha,\infty)} [x-1, t, \mathbf{I}_{x-1}] \xrightarrow{[\alpha,\infty)} [x, t, \mathbf{I}_x] \xrightarrow{[\alpha,\infty)} \cdots \xrightarrow{[\alpha,\beta]} [\beta, t, \mathbf{I}_\beta] \xrightarrow{[\alpha,\infty)} \cdots$$

has the property $|\mathbf{I}_{x-1}| \neq |\mathbf{I}_x| = \cdots = |\mathbf{I}_\beta|$*. Also, for all* $n \in \mathbb{N}$*,*

$$|\mathbf{I}_n| \leq \frac{(b+1) * (b+2)}{2} + c * (b+1) - 1$$

Proof (sketch). In this case we have $x = \alpha + b + c$. We use Thm. 2 as a base case and perform induction on c. The bound is derived from considering the state of the monitor instances introduced at positions $p \in [\alpha + 1, \alpha + c]$. □

Thus, we derive a worst case space complexity of $O(b^2 + bc)$ for the size of the instance set. The next case is the most interesting because the convergence isn't as straightforward. We will not provide the full argument, see [4] for the detailed proof.

Theorem 4. *Given an* $\mathbb{N}$*-triple* $t = \langle 0, b, b-c\rangle_{\mathbb{N}}$*, for* $b, c \in \mathbb{N}$ *and* $0 \leq c \leq b$*, and an interval* $[\alpha, \infty)$*, there exists a value* $x \in [\alpha, \infty)$ *such that for all* $\beta \in \mathbb{N}$*, the evaluation chain*

$$[\triangleright, t, \emptyset] \xrightarrow{[\alpha,\infty)} \cdots \xrightarrow{[\alpha,\infty)} [x-1, t, \mathbf{I}_{x-1}] \xrightarrow{[\alpha,\infty)} [x, t, \mathbf{I}_x] \xrightarrow{[\alpha,\infty)} \cdots \xrightarrow{[\alpha,\beta]} [\beta, t, \mathbf{I}_\beta] \xrightarrow{[\alpha,\infty)} \cdots$$

has the property $|\mathbf{I}_{x-1}| \neq |\mathbf{I}_x| = \cdots = |\mathbf{I}_\beta|$*. Also, for all* $n \in \mathbb{N}$*,*

$$|\mathbf{I}_n| \leq \frac{(b-c) * ((b-c)+1)}{2} + (c-1)$$

Proof (sketch). The value of x is the same as Thm. 2, $x = \alpha + b - 1$, but the way we arrive at this value is different. Given $0 < c$, at some point $\gamma < b$, instances are removed from the instance set, but the quantifier has not fully unrolled. Thus, at this point the instance set increases by 1 as the position is incremented till position $\alpha + b - 1$. At this position the instance set has a constant size. The bound is derived from considering the growth of the instance set in two parts, those which are unrolled before γ is reached and those unrolled afterwards. □

Theorem 5. *Given an* $\mathbb{N}$*-triple* $t = \langle a, b, c\rangle_{\mathbb{N}}$*, for* $a, b, c \in \mathbb{N}$*, and an interval* $[\alpha, \infty)$*, there exists a value* $x \in [\alpha, \infty)$ *such that for all* $\beta \in \mathbb{N}$*, the complete improper evaluation chain*

$$[\triangleright, t, \emptyset] \xrightarrow{[\alpha,\infty)} \cdots \xrightarrow{[\alpha,\infty)} [x-1, t, \mathbf{I}_{x-1}] \xrightarrow{[\alpha,\infty)} [x, t, \mathbf{I}_x] \xrightarrow{[\alpha,\infty)} \cdots \xrightarrow{[\alpha,\beta]} [\beta, t, \mathbf{I}_{\beta+x}] \xrightarrow{[\alpha,\infty)} \cdots$$

has the property $|\mathbf{I}_{x-1}| \neq |\mathbf{I}_x| = \cdots = |\mathbf{I}_{\beta+x}|$*. Also, for all* $n \in \mathbb{N}$*,*

$$|\mathbf{I}_n| \leq f(a, b, c)$$

where $f(a, b, c)$ *is defined as*

$$f(a,b,c) = \left\{ \begin{array}{l|c} a + (I-1) & c \leq a \\ a + \dfrac{(I-d)(I-d+1)}{2} + (d-1) & c = b - d \ ,\ a < c \leq b \\ a + \dfrac{I * (I+1)}{2} + d * I - 1 & c = b + d \end{array} \right.$$

where $I = (b - a) + 1$.

Proof (sketch). Put the results of Thm. 2, 3, & 4 together and add the shift of the interval provided by a. □

Example 3. *Let us consider the evaluation of the* $\mathbb{N}$*-triple* $t = \langle 0,2,2\rangle_{\mathbb{N}}$ *over the interval* $[0,\infty)$.

$$[\triangleright, t, \emptyset] \xrightarrow{[0,\infty)} \left[0, t, \left\{ \begin{array}{l} \langle 1,2,2\rangle_{\mathbb{N}}(0) \\ \langle 0,0,2\rangle_{\mathbb{N}}(0) \end{array} \right\}\right] \xrightarrow{[\alpha,\infty)} \left[1, t, \left\{ \begin{array}{l} \langle 2,2,2\rangle_{\mathbb{N}}(0) \\ \langle 1,1,2\rangle_{\mathbb{N}}(0) \\ \langle 0,0,2\rangle_{\mathbb{N}}(0) \\ \langle 1,2,2\rangle_{\mathbb{N}}(1) \\ \langle 0,0,2\rangle_{\mathbb{N}}(1) \end{array} \right\}\right] \xrightarrow{[\alpha,\infty)}$$

$$\left[2, t, \left\{ \begin{array}{l} \langle 2,2,2\rangle_{\mathbb{N}}(1) \\ \langle 1,1,2\rangle_{\mathbb{N}}(1) \\ \langle 0,0,2\rangle_{\mathbb{N}}(1) \\ \langle 1,2,2\rangle_{\mathbb{N}}(2) \\ \langle 0,0,2\rangle_{\mathbb{N}}(2) \end{array} \right\}\right] \xrightarrow{[\alpha,\infty)} \left[3, t, \left\{ \begin{array}{l} \langle 2,2,2\rangle_{\mathbb{N}}(2) \\ \langle 1,1,2\rangle_{\mathbb{N}}(2) \\ \langle 0,0,2\rangle_{\mathbb{N}}(2) \\ \langle 1,2,2\rangle_{\mathbb{N}}(3) \\ \langle 0,0,2\rangle_{\mathbb{N}}(3) \end{array} \right\}\right] \xrightarrow{[\alpha,\infty)} \cdots$$

One can easily calculate that $f(0,2,2) = 5$.

For Sec. 6 the following theorem concerning the monotonicity of $f(a,b,c)$ is needed

Theorem 6. *Given the function* f *of Thm. 5 and* $a,b,c \in \mathbb{N}$ *then the following properties hold:* $f(a,b,c) \leq f(a,b+1,c)$*,* $f(a+1,b,c) \leq f(a,b,c)$*, and* $f(a,b,c) \leq f(a,b,c+1)$.

Proof (sketch). The properties follow from tedious, but trivial computations. □

Essentially what we derived in this section is a function bounding the space requirements of $\mathbb{N}$-triples in the positive fragment, i.e. Thm. 5. The function can be extended to all $\mathbb{N}$-triples, which is done in [4]. We use this function in the next section as invariant for deriving a function bounding the space requirements of a much larger fragment.

6 Space Analysis of the Variable Only Fragment

Now that we have a space analysis of the individual quantifiers and their local formula structure, we can generalize the results to arbitrary formulas of the variable only fragment. A major problem with the generalization is that we assumed in the analysis of the restricted fragment that quantifier interval bounds are always of the form $x+n$ where $n \in \mathbb{N}$. Though in actuality, the variable used can be any of the bound variable above the quantifier.

To deal with this issue we introduce the concept of a *dominating formula*. The dominating formula f_D of a formula f has the same propositional structure, but the bounds of the quantifier intervals only contain x and the interval is at least as large as the interval in f. To make this concept more formal, we introduce the dominating formula transformation.

Definition 5 (Dominating Formula Transformation)**.** *Given a sentence* $f \in \mathbb{M}^{vb}$ *we construct the* dominating formula f_D of f *using the following transformation*

$$\begin{aligned}
D(\forall_{0\leq x} : f_D, \emptyset, \emptyset) &\Longrightarrow D(\forall_{0\leq x} : D(f, \{x \leftarrow x\}, \{x \leftarrow x\})) \\
D(f_1 \;\&\; f_2, \sigma_l, \sigma_h) &\Longrightarrow D(f_1, \sigma_l, \sigma_h) \;\&\; D(f_2, \sigma_l, \sigma_h) \\
D(\neg f, \sigma_l, \sigma_h) &\Longrightarrow \neg D(f, \sigma_l, \sigma_h) \\
D(\forall_{y\in[b_1,b_2]} : f, \sigma_l, \sigma_h) &\Longrightarrow \forall_{y\in[h_L(b_1),h_H(b_2)]} : D(f, \sigma_l\{y \leftarrow h_L(b_1)\}, \sigma_h\{y \leftarrow h_H(b_2)\}) \\
D(@x, \sigma_l, \sigma_h) &\Longrightarrow @x
\end{aligned}$$

where $h_L(b_1) = \min\{b_1\sigma_l, b_1\sigma_h\}$ *and* $h_H(b_2) = \max\{b_2\sigma_l, b_2\sigma_h\}$.

The substitutions used in the dominating formula transformation implement the inequalities of Thm. 6, essentially maximizing the value of $f(\cdot,\cdot,\cdot)$. The dominating formula transformation allows us to construct a theorem similar to Thm. 1. The problem is that we do not have a transition relation for the variable only fragment to compare to the operational semantics. Instead, being that we assuming the worst case behavior at each quantifier, based on Thm. 1, we know that we have an upper bound for the variable only fragment. The bounding function of Def. 6 essentially implements this idea. Remember that the function $f(\cdot,\cdot,\cdot)$ is the upper bound instance set size discussed in Thm. 1 for the evaluation transition. Thm. 7 below provides concrete bounds for the space complexity of the above method.

Definition 6 (Bounding Function)**.** *Given a sentence* $f \in \mathbb{M}^{vb}$*, let* f_D *be its dominating formula. We construct the bounding function* $b(f_D)$ *as follows:*

$$
\begin{array}{rcl}
b(\forall_{0\leq x} : f) & \Longrightarrow & b(f, \{x \leftarrow 0\}) \\
b(@f \ \& \ @g, \sigma) & \Longrightarrow & b(f,\sigma) + b(g,\sigma) \\
b(\neg f, \sigma) & \Longrightarrow & b(f,\sigma) \\
b(\forall_{y\in[x+a,x+b]} : f, \sigma) & \Longrightarrow & g(a, b, w(f, \sigma\{y \leftarrow (x+b)\sigma\})) * b(f, \sigma\{y \leftarrow (x+b)\sigma\}) \\
b(@y, \sigma) & \Longrightarrow & 1 \\
w(@f \ \& \ @g, \sigma) & \Longrightarrow & \max\{w(f,\sigma), w(G,\sigma)\} \\
w(\neg f, \sigma) & \Longrightarrow & w(f,\sigma) \\
w(\forall_{y\in[x+a,x+b]} : f, \sigma) & \Longrightarrow & w(f, \sigma\{y \leftarrow (x+b)\sigma\}) \\
w(@y, \sigma) & \Longrightarrow & y\sigma
\end{array}
$$

Theorem 7. *Given a sentence* $f \in \mathbb{M}^{vb}$*, the space complexity of evaluating* f *is* $O(b(D(f,\emptyset,\emptyset))) = O((I_{max})^{2n})$*, where* I_{max} *is the largest interval in* $D(f,\emptyset,\emptyset)$ *and* n *is the formula depth.*

Proof(sketch). The dominating formula requires more space to evaluate than f, by construction. By performing induction on the formula complexity of f (i.e., its quantifier structure) we get the exponential bound $((I_{max})^2)^n$ where n is the length of the longest chain of quantifiers. (remember that each $f(\cdot,\cdot,\cdot)$ has a worst case space complexity of $O((I_{max})^2)$). □

Thm. 7 provides a worst case space complexity which we provide in order to connect our results with known results for similar languages, for example weak fragments of predicate logic [1]. In our case, we are speaking about the number of states of the stream which need to be checked. However, this bound represents an overapproximation of the bounding function given in Def. 6. This function can be applied to each monitor individually, i.e., it does not just provide a general space complexity but also gives bounds for the space usage of a given monitor. Nevertheless, since even this bounding function suffers from overestimation, we are currently investigating various optimizations to improve the accuracy of the analysis.

7 Conclusions

In this paper we have provided a method for bounding the memory needed to execute a monitor which has been expressed in a fragment of the core language defined in Sec. 2. The representation of a monitor and its operational semantics are abstracted as outlined in Sec. 3 and Sec. 4 to derive an upper bound for the monitor's memory consumption. For details concerning the extension of this work to the entire core language, see [4].

The work presented here answers our questions concerning an upper bound for the number of instances in memory, as well as an upper bound for individual monitors. The extensions

concerning constants and ∞ provide a method for discerning monitors which use infinite memory and those which do not [4]. However, one open problem we would like to address is the accuracy of the derived upper bound. It turns out that it is easy to design a monitor such that the results of computing the expected memory usage is off by a few orders of magnitude (though the bounds are accurate, if one abides by the structure of the restricted fragment). Some methods for fixing these problems are given in [4], i.e. the method used for abstracting monitors into $\mathbb{N}$-triples seems to play a role in the output of the bounding function. Nonetheless, it seems that the global formula structure has a larger than expected influence on how many instances need to be held in memory. We have already started to investigate this issue in [3]. Our future work will focus on understanding the influence of the global structure on the actual number of instances needed in memory.

References

[1] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer, 1997.

[2] Julius Richard Büchi. Weak Second-Order Arithmetic and Finite Automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

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

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

[5] Bernd Finkbeiner and Lars Kuhtz. Monitor Circuits for LTL with Bounded and Unbounded Future. In *Runtime Verification, 9th International Workshop, RV 2009*, volume 5779 of *Lecture Notes in Computer Science*, pages 60–75, Grenoble, France, June 26–28, 2009. Springer, Berlin.

[6] Markus Frick and Martin Grohe. The Complexity of First-Order and Monadic Second-Order Logic Revisited. *Annals of Pure and Applied Logic*, 130(1–3):3–31, 2004.

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

[8] Orna Kupferman, Yoad Lustig, and Moshe Y. Vardi. On Locally Checkable Properties. In *Logic for Programming, Artificial Intelligence, and Reasoning, 13th International Conference, LPAR 2006*, volume 5779 of *Lecture Notes in Artificial Intelligence*, pages 302–316, Phnom Penh, Cambodia, November 13–17, 2006. Springer, Berlin, Germany.

[9] Temur Kutsia and Wolfgang Schreiner. 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, 2014.

[10] LogicGuard II, November 2015. http://www.risc.jku.at/projects/LogicGuard2/.

[11] Oded Maler, Dejan Nickovic, and Amir Pnueli. Real Time Temporal Logic: Past, Present, Future. In Paul Pettersson and Wang Yi, editors, *Formal Modeling and Analysis of Timed Systems, Third International Conference (FORMATS)*, volume 3829 of *Lecture Notes in Computer Science*, pages 2–16, Uppsala, Sweden, September 26–28, 2005. Springer, Berlin, Germany.

[12] Robert McNaughton and Seymour Papert. *Counter-Free Automata*, volume 65 of *Research Monograph*. MIT Press, Cambridge, MA, USA, 1971.

[13] Wolfgang Schreiner, Temur Kutsia, David Cerna, Michael Krieger, Bashar Ahmad, Helmut Otto, Martin Rummerstorfer, and Thomas Gössl. The LogicGuard Stream Monitor Specification Language (Version 1.01). Tutorial and reference manual, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria, November 2015.

[14] Moshe Y. Vardi and Pierre Wolper. 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*, pages 332–344. IEEE Computer Society, 1986.