Writing Time- and Space- Efficient LogicGuard Specifications*

David M. Cerna

March 27, 2017

Abstract

Based on the analysis carried out as part of the LogicGuard II project we purpose various methods for reducing the time and space complexity of LogicGuard specifications. The key results of these investigations are that space complexity is most effected by the order of the intervals based on length and time complexity is most effected by the overlap of the intervals. We provide examples illustrating these findings as well as analysis of the theoretical results.

1 Introduction

In previous work, we have presented both a time and space complexity analysis of the core language [7] of the LogicGuard specification language [9]. The analysis has resulted in algorithmic procedures computing accurate approximations of the upper bounds of processing time and memory usage of a given specification during evaluation [1, 5]. For specific classes of monitor specifications we were able to derive more precise bounds and additional results concerning the relationships between "similar" monitor specifications [3, 4]. The majority of these results where achieved by abstracting away parts of the specification which do not directly concern time and space complexity. Also, we ignored variable nesting (when the interval of one quantified variable is defined using another quantified variable) being that it complicates the analysis without providing a significant boost in terms of accuracy. Furthermore, nested variables can actually reduce the space and time complexity [2]. Removing them simplifies the process of constructing upper bounds.

Being that much of our earlier work relied on the removal of variable nesting we decided to investigate, in a precise way, what is loss, in terms of accuracy of the upper bounds, when variable nestings are removed. In the process we discovered that specifications can be divided into two categories based on the resulting monitor after variable nesting is removed [2]. In this technical report we will present the above mentioned results with application to specification writing, that is how the results effect the end-user writing monitors in the specification language. For small specifications, the effect on memory and processor

*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.

usage when running the specification is insignificant and thus, can be ignored all together; the LogicGuard framework itself has a large processing and storage footprint. However, for large specifications, the storage costs (in terms of memory usage) of specification instances can be a few times larger than the LogicGuard framework's footprint and cannot be ignored, same holds for processing requirements. Thus, for large and complex monitor specifications developing strategies for these metrics is of great importance.

When too high, the number of instances which need to evaluated upon arrival of a new message can cause significant slow down of the system. If messages arrive to fast, the time to process a message might exceed the time till the next message; imagine a DDOS attack. In such cases the messages are skipped and the resulting evaluation does not accurately match the state of the stream. Unfortunately, it turns out that reducing the complexity of the evaluation per message increases the number of instances in memory while decreasing the number of instance in memory increases the evaluation per message.

Resolving this inverse relationship is not easy and there is no easy solution to the general problem. The best solution would be, during specification construction, considering both the stream the monitor is specified over and the property being monitored in order to construct a near optimal monitor specification with respect to the time and space complexity. Though, based on the work presented in [3], it is possible to get a baseline specification which has average time and space complexity with respect to similar[1] monitor specification. In [3], we provide a method for placing any given monitor within a "spectrum" where on one side is the most efficient "similar" monitor in terms of space complexity and on the other side the most efficient "similar" monitor in terms time complexity. Sadly, the most efficient monitor in terms of space is the worst in terms of time and vice versa.

Though this spectrum sounds appealing for finding balanced monitors there is the problem, that is such balanced monitors are not easy to find because a monotonic ordering of the monitors in the spectrum is not known. This is an open problem where optimization algorithms might play an important role in finding near optimal balanced monitors.

2 Efficient Space: Quality Dependent on Interval Order

As presented in recent work [2], removing variable nesting can be used to distinguish between good and bad specifications; in particular for space complexity. The method of variable nesting removal used for these results is the *dominating monitor transformation* [6]. It was developed to simplify specifications so that an upper bound can be easily computed. Initially when the transformation was introduced the upper bounds where rough approximations rather than the precise space complexity of a worst case monitor specification [6]. This early work was superseded after further analysis was performed.

The transformation simply produces a specification where every quantified variable is defined over an interval whose terms only contain the outermost

[1]We are using this word a lot without giving it a precise meaning, however, the colloquial meaning, at least for now, is enough to understand our description.

quantified variable, that is the *monitor variable*[2]. This transformation, while simplifying the specification, results in a specification which requires at least as much memory for evaluation as the original, though in most cases, requires more memory. In [3, 4] we studied the class of all dominating monitor specifications and showed that "special" monitors exists which upper and lower bounding a particular set of monitors based on their structutre. Essentially, dominating monitor specifications are an important subclass of monitor specifications which we plan to study further in future work.

The work so far discussed has not been carried out on the full specification language but rather a core language which has simplified quantification and discrete time. The monitor specifications used in practice are defined over a continuous time interval, however, studying the space complexity of the language using continuous time can be quite difficult. We chose discrete time to simplify analysis of the language. It is quite easy to convert the analysis to continuous time by computing the average number of message within a given time interval and adjusting accordingly. The core language is defined as follows [5]:

$$
\begin{array}{rl}
M ::= & \forall_{0 \le V} : F. \\
F ::= & @V \mid \neg F \mid F \ \wedge \ F \mid F \ \& \ F \\
\mid & \forall_{V \in [B,B]} : F. \\
B ::= & 0 \mid \infty \mid V \mid B \pm N. \\
V ::= & x \mid y \mid z \mid \ \ldots \\
N ::= & 0 \mid 1 \mid 2 \mid \ \ldots
\end{array}
$$

Figure 1: The core language syntax

For further details see [8]. Its syntax is depicted in Fig. 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 (evaluation of F_2 is delayed until the value of F_1 becomes available), $\forall_{V \in [B_1, B_2]} : F$ denotes quantification over the interval $[B_1, B_2]$.

we defined *the dominating monitor/formula transformation* over the core language as follows:

Definition 1 (Dominating Monitor/Formula Transformation) *Let* $\mathbb{A} = \mathbb{V} \rightarrow^{part.} \mathbb{N}$ *be the domain of assignments that map variables to natural numbers. Then the* dominating monitor transformation $D : \mathbb{M} \rightarrow \mathbb{M}$ *respectively* formula

[2]The variable instantiated with values from the external stream.

transformation $D' : \mathbb{F} \times \mathbb{A} \times \mathbb{A} \to \mathbb{F}$ *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 is of the form* $x + n$ *(we omit the formal details, see the example below).*

The following example illustrates how the dominating monitor transformation works.

Example 1 *Consider the following monitor* M*:*

$$\forall_{0 \leq x} : \forall_{y \in [x+1, x+5]} : ((\forall_{z \in [y, x+3]} : \neg @z \ \& \ @z) \ \& \ G(x, y))$$

$$G(x, y) = \forall_{w \in [x+2, y+2]} : (\neg @y \ \& \ (\forall_{m \in [y, w]} : \neg @x \ \& \ @m))$$

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

$$\forall_{0 \leq x} : \forall_{y \in [x+1, x+5]} : ((\forall_{z \in [x+1, x+3]} : \neg @z \ \& \ @z) \ \& \ G(x, y))$$

$$G(x, y) = \forall_{w \in [x+2, x+7]} : (\neg @y \ \& \ (\forall_{m \in [x+1, x+7]} : \neg @x \ \& \ @m))$$

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

While this transformation was essential for early work in the subject [5], in [2] we showed that it is not needed for the computation of time complexity. Instead nested sums and variable substitutions where used. Based on the developments in [2] we realized a similar algorithm can be developed for space complexity. However, even in light of these results, the dominating monitor transformation is not obsolete, while as mentioned before, it can be used to distinguish between memory efficient and memory inefficient monitors. Though, the whole point of the space complexity analysis is to make this distinction, the algorithm of [5] and the results of [3, 4] provide a number which the users are meant to interpret subjectively. The analysis presented in [2] instead separates the monitors into two classes based on the interaction between quantified variables. Let us consider the following monitors:

$$m_1 = \forall_{0 \leq x} : \forall_{y \in [x, x+80]} : \forall_{z \in [x, y+1]} : @z \tag{1a}$$

$$m_2 = \forall_{0 \leq x} : \forall_{y \in [x, x+40]} : \forall_{z \in [x, y+40]} : @z \tag{1b}$$

Notice that, for monitor m_1, the instance that is generate at position 0, i.e. $x = 0$ will produce 81 instances of $\forall_{z \in [x, y+1]} : @z$, one for each position in the interval $[0, 80]$. These instances are as follows:

$$\begin{gathered}\forall_{z\in[0,1]} : @z \\ \forall_{z\in[0,2]} : @z \\ \forall_{z\in[0,3]} : @z \\ \vdots \\ \forall_{z\in[0,80]} : @z \\ \forall_{z\in[0,81]} : @z\end{gathered}$$

For monitor m_2 We end up with 41 instances of $\forall_{z\in[x,y+40]} : @z$ when $x = 0$, one for each position in the interval $[0, 80]$:

$$\begin{gathered}\forall_{z\in[0,40]} : @z \\ \forall_{z\in[0,41]} : @z \\ \forall_{z\in[0,42]} : @z \\ \vdots \\ \forall_{z\in[0,80]} : @z \\ \forall_{z\in[0,81]} : @z\end{gathered}$$

Though, m_1 constructs more instance during the interval $[0, 80]$ each instance remains in memory for a much shorter amount of time. For example, instance $\forall_{z\in[0,2]} : @z$ which was constructed at position 1 will be removed at position 2 while instance $\forall_{z\in[0,41]} : @z$ which was constructed at position 1 will be removed at position 41 While m_1 will only have two instances in memory at any moment, m_2 can have up to 40 instances at any moment. Note that in terms of time complexity they are exactly the same, 2 evaluations per step. What is interesting is the comparison between the monitors evaluation and the evaluation of their dominating monitors:

$$D(m_1) = \forall_{0\leq x} : \forall_{y\in[x,x+80]} : \forall_{z\in[x,x+81]} : @z \tag{2a}$$

$$D(m_2) = \forall_{0\leq x} : \forall_{y\in[x,x+40]} : \forall_{z\in[x,x+80]} : @z \tag{2b}$$

First notice that concerning the dominating forms, the two monitors are quite similar. Moreover $D(m_1)$ appears much worst then $D(m_2)$. Computing exactly how many instances are in memory at any time during evaluation we get the following results:

$$\begin{aligned}&Eval(m_1) = 161\\ &Eval(D(m_1)) = 3401\\ &Eval(m_2) = 1680\\ &Eval(D(m_2)) = 2500\end{aligned}$$

What becomes instantly clear from the evaluation is that the ratio between m_1 and its dominating form is much larger than m_2, $1 : 21$ vesus $1 : 1.5$. This implies that m_1 in its original form is much more memory efficient than m_2. The question now is whether this can be quantified and be used to distinguish between good and bad specifications or more specifically good and bad uses of quantifiers.

As it was proven in [2], the following two theorems provide generic monitor structures which distinguish between monitors with high memory requirements compared to the dominating form and those with low memory requirements:

Theorem 1 *Let $m \in \mathbb{M}$, $a_1, b_1, a_2, b_2 \in \mathbb{Z}$, and $x, y, z \in V$ such that*

$$m = \forall_{0 \leq x} : \forall_{y \in [x+a_1, x+b_1]} : \forall_{z \in [x+a_2, y+b_2]} : @z.$$

Then for $0 < b_1$,$a_1 \leq b_1$, $d = min_{i \in [a_1, b_1]} \{0 < b_2 + i\}$, and $a_2 \leq b_2 + d$. If $b_1 = k + b_2 + d$ for $k \geq 0$, then $O(k) \cdot |m|_{sc} \geq |D(m)|_{sc}$.

Theorem 2 *Let $m \in \mathbb{M}$, $a_1, b_1, a_2, b_2 \in \mathbb{Z}$, and $x, y, z \in V$ such that*

$$m = \forall_{0 \leq x} : \forall_{y \in [x+a_1, x+b_1]} : \forall_{z \in [x+a_2, y+b_2]} : @z.$$

Then for $d = min_{i \in [a_1, b_1]} \{0 < b_2 + i\}$. $b_1 < b_2 + d$, then $2 \cdot |m|_{sc} \geq |D(m)|_{sc}$.

We can translate these theorems into a more readable form. Let us consider Theorem 1. If we restrict ourselves to $\mathbb{N}$ instead of $\mathbb{Z}$. This simplification results in the following theorem:

Theorem 3 *Let $m \in \mathbb{M}$, $a_1, b_1, a_2, b_2, k, j \in \mathbb{N}$, and $x, y, z \in V$ such that*

$$m = \forall_{0 \leq x} : \forall_{y \in [x+a_1, x+k+b_1+a_1]} : \forall_{z \in [x+b_1+a_1-(j+1), y+b_1]} : @z$$

and

$$D(m) = \forall_{0 \leq x} : \forall_{y \in [x+a_1, x+k+b_1+a_1]} : \forall_{z \in [x+b_1+a_1-(j+1), x+k+2 \cdot b_1+a_1]} : @z$$

then $O(k) \cdot |m|_{sc} \geq |D(m)|_{sc}$.

For example, let us consider the case when $a_1 = 0, b_1 = 1, j = 0$ and $k = 79$. This produces m_1 which we have mentioned as having the following relationship with its dominating monitor $21 \cdot Eval(m_1) \approx Eval(D(m_1))$. The same can be done for Theorem 2.

Something that we should make note of here is that these theorems can be generalized quite easily to an arbitrary number of quantifiers with arbitrary variable nesting; this was done in [2]. Also, the theorems hold for quantifier intervals of the following form

$$m = \forall_{0 \leq x} : \forall_{y \in [x+a_1, x+b_1]} : \forall_{z \in [y+a_2, y+b_2]} : @z$$

as well, but this was left out being that it complicates the proofs without changing the ratios.

Essentially, when writing monitor specifications one should always try to write monitors whose quantifier intervals obey the constraints of Theorem 1. If one looks closely at the results of [3], the clause of monitor specifications defined by Theorem 1 is closely related to the *inverse standard quantifier vectors.* To be more specific, any monitor specification whose dominating form is an *inverse standard quantifier vectors* is part of the class defined by Theorem 1. This is only a sufficient condition being that monitor $D(m_1)$ is not an inverse standard quantifier vectors but it is none the less part of the good class. When the dominating transformation of a monitor is a *standard quantifier vector* it is in all likelihood the case that the monitor is in the class defined by Theorem 2. As it was shown in [3], these standard quantifier vectors represent the worst case memory requirements.

Interestingly, this does not say much about time complexity. Inverse standard and standard quantifier vectors are defined by the ordering of the quantifier interval upper bounds assume the lower bound of all quantifier intervals are the same. Essentially this orders the intervals by size. For inverse standard the interval size is decreasing and for standard the interval size is increasing. In any other case we refer to the specification as *top free*. It was shown in [3] that if we look at all dominating monitor specifications constructable from n quantifier intervals the inverse standard and standard quantifier vectors using those intervals will lower and upper bound any top free quantifier vector built from those same intervals. Essentially the takes-aways from this analysis are as follows:

1. Construct specifications which fit the constraints of Theorem 1.
2. When possible order the quantifier intervals such that the largest interval is the outermost and the smallest is the innermost.

Interestingly, these constraints do not have a direct influence on our measure of time complexity. Without any additional constraints time complex will increase or decrease with the space complexity. However, there are other properties of the specification which influence time complexity in much the same way as space complexity is influenced by the above.

3 Efficient Time: Quality Dependent on Interval overlap

In all analysis carried out concerning space complexity, one property of the quantifier intervals was completely ignored, that is the interval overlap. In [3], not only did we ignore it, we maximized it! Ironically enough, minimizing interval overlap happens to be one of the best ways to reduce time complexity. So far, we have not proven any results concerning this conjecture, but experimental analysis of a large number of monitors has provide substantial evidence supporting the claim.

The experimental set up is as follows: we will consider the following two monitor specifications

$$m_S(a,i,u,w_1,w_2) = \forall_{0\leq x} : \forall_{y\in[x,x+a*u]} : \forall_{z\in[w_1+i,w_2+u]} : @z$$

$$m_I(a,i,u,w_1,w_2) = \forall_{0\leq x} : \forall_{y\in[w_1+i,w_2+u]} : \forall_{z\in[x,x+a*u]} : @z$$

where $a \in [0,1]$, $u \in \mathbb{N}$, $w_1, w_2 \in \{x, y\}$, and $i \in [0,u]$. The idea being that we adjust i and see what the result of the algorithm [1] is for monitors $m_S(a,i,u,w_1,w_2)$ and $m_I(a,i,u,w_1,w_2)$. We also adjust a to see how the relative interval sizes affect time. For example, in Figure 2 we took the monitor schema $m_S(a,i,500,y,x)$ and evaluated it for 6 values of a and for all $i \in [0,500]$. The interesting result is that for each value of a the lower bound for time complexity is not necessarily setting i is the maximum, but some where in between the maximum and the minimum. As a decreases the lower bound for time complexity is shifted further and further to the maximum possible value of i. These results tell us minimizing the time complexity is not a simple task, but notice

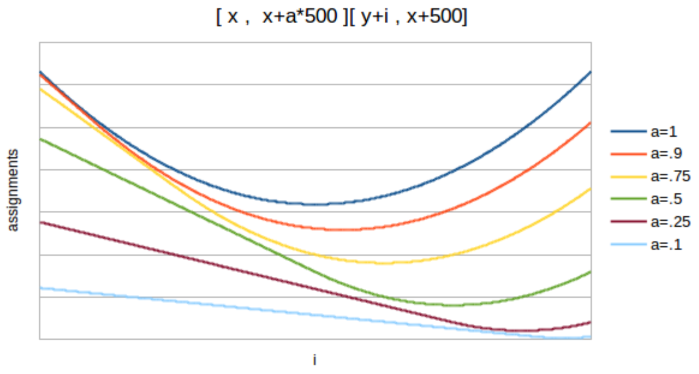

Figure 2: Number of assignments computed for monitors of the form $m_S(a, i, 500, y, x)$.

that the overlap between the two intervals decreases as we increase i, but if it decreases too much we end up with an increase in time complexity again. Let use now consider the monitor schema $m_S(a, i, 500, x, y)$ whose chart is to be found in Figure 3. Unlike, $m_S(a, i, 500, y, x)$, $m_S(a, i, 500, x, y)$ follow quite a simple pattern. This may be easily explained by the fact that i is the only mechanism present which decreases the overlap, while in Figure 2 both the choice of i and the variable nesting influence the overlap of the intervals. Figure 3 provides much better evidence that overlap is one of the kep factors influencing the time complexity.

Continuing our experiments we can look at a monitor schema which has only one free variable, that is we fix everything but i and compare the various charts. Let us start with $m_I(1, i, 500, \cdot, \cdot)$. Notice that Figure 4 seems similar to Figure 3 except that the lines are slightly concave and all three arrangements of variables end at the same point. This concavity is reminiscence of Figure 2. However, looking at Figure 5 one can see that the relationship between a and i is even more complex in this case.

4 Conclusions

The messages we want to send with these charts are as follows:

1. Construct specifications such that the overlap between the quantifier intervals is minimized.
2. If the overlap has been minimized, order the quantifier intervals, if possible such that the larger intervals are nested within the smaller intervals.

The following monitor specification is an example of one with good time complexity

$$\forall_{0\leq x} : \forall_{y\in[x,x+200]} : \forall_{z\in[x+200,y+200]} : @z$$

while the following would have poor time complexity

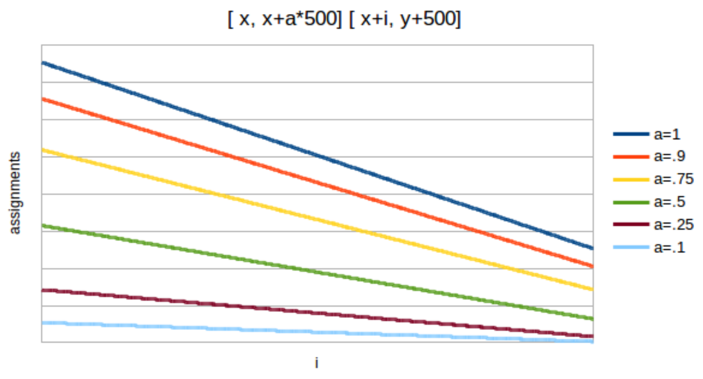

Figure 3: Number of assignments computed for monitors of the form $m_S(a, i, 500, x, y)$.

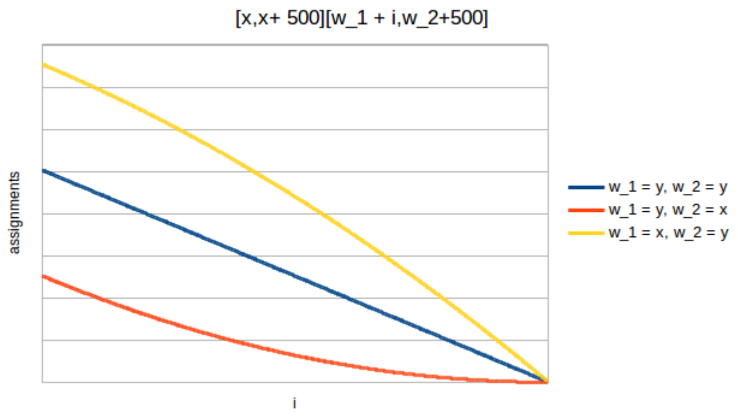

Figure 4: Number of assignments computed for monitors of the form $m_I(1, i, 500, \cdot, \cdot)$.

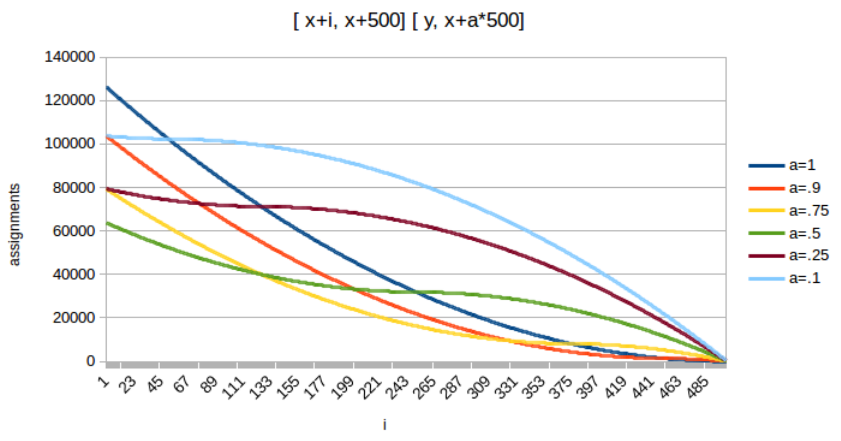

Figure 5: Number of assignments computed for monitors of the form $m_I(1, i, 500, \cdot, \cdot)$.

$$\forall_{0 \leq x} : \forall_{y \in [x, x+400]} : \forall_{z \in [x+100, y+100]} : @z.$$

Nonetheless, there is still further work to be done in this area as there are multiple factors influencing the time complexity.

References

[1] David M. Cerna and Wolfgang Schreiner. An Algorithmic Approach to Bounding the Time Complexity of Stream Specifications. Technical report, Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, Austria, January 2017.

[2] David M. Cerna and Wolfgang Schreiner. Measuring the Gap: Algorithmic Approximation Bounds for the Space Complexity of Stream Specifications. In Mohamed Mosbah and Michaël Rusinowitch, editors, *SCSS 2017: 8th International Symposium on Symbolic Computation in Software Science, Gammarth, Tunisia, April 6-9, 2017*, EPiC Series in Computing, April 2017. To appear.

[3] David M. Cerna, Wolfgang Schreiner, and Temur Kutsia. Analytic and Algorithmic Methods for Computing the Space Requirements of Stream Monitor Specifications. Technical report, Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, Austria, October 2016. Submitted for publication.

[4] David M. Cerna, Wolfgang Schreiner, and Temur Kutsia. Better Space Bounds for Future-Looking Stream Monitors. Technical report, Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, Austria, July 2016. Submitted for publication.

[5] David M. Cerna, Wolfgang Schreiner, and Temur Kutsia. Predicting Space Requirements for a Stream Monitor Specification Language. In Yliès Falcone and César Sánchez, editors, *Runtime Verification — 16th International Conference, RV 2016, Madrid, Spain, September 23–30, 2016*, volume 2012 of *Lecture Notes in Computer Science*, pages 135–151. Springer, September 2016.

[6] David M. Cerna, Wolfgang Schreiner, and Temur Kutsia. Space Analysis of a Predicate Logic Fragment for the Specification of Stream Monitors. In James H. Davenport and Fadoua Ghourabi, editors, *SCSS 2016 — 7th International Symposium on Symbolic Computation in Software Science, Tokyo, Japan, March 28 - 31, 2016*, volume 39 of *EPiC Series in Computing*, pages 29–41, March 2016.

[7] Temur Kutsia and Wolfgang Schreiner. A Resource Analysis for LogicGuard Monitors. Technical report, Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, Austria, December 2013.

[8] Temur Kutsia and Wolfgang Schreiner. Verifying the Soundness of Resource Analysis for LogicGuard Monitors, Part 1. Technical report, Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, Austria, December 2013.

[9] Wolfgang Schreiner, Temur Kutsia, Davic Cerna, Michael Krieger, Bashar Ahmad, Helmut Otto, Martin Rummerstorfer, and Thomas Gössl. The LogicGuard Stream Monitor Specification Language Tutorial and Reference Manual. Technical report, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria, October 2015.