Measuring the Gap: Algorithmic Approximation Bounds for the Space Complexity of Stream Specifications*

David M. Cerna and Wolfgang Schreiner

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

Abstract

In previous work we presented an algorithmic procedure for analysing the space complexity of monitor specifications written in a fragment of predicate logic. These monitor specifications were developed for runtime monitoring of event streams. Our procedure provides accurate results for a large fragment of the possible specifications, but overestimates the space complexity of precisely those specifications which are more likely to be found in real world applications. Experiments hinted at a relationship between the extent our procedure over-approximates the space requirements of a specification and the quantifier structure of the specification. In this paper we provide a formalization of this relationship as approximation ratios, and are able to pinpoint "good" constructions, that is specifications using less memory. These results are first steps towards categorizing specifications based on memory efficiency.

1 Introduction

This work is part of an ongoing project to analyse the space complexity [1, 2, 3, 4] of the LogicGuard stream monitor specification language [13], which is based on a fragment of predicate logic. In this language, specifications are formulas that quantify over positions in a stream of events, that is the stream being monitored. LogicGuard was developed in an industrial collaboration for the runtime monitoring of networks for security violations [10]. At the heart of our previous analysis were a few core assumptions which allowed the development of a simplified operational semantics, but which also introduced, in some cases, significant over-approximation. Though, previously published experimental results [1, 3] point to the existence of specifications which are reasonably over-approximated, within a factor of two of the true value.

We continue our analysis by categorizing the types of quantifiers which are well approximated by our method and those which are poorly approximated. Both of these quantifier configurations are concerned with the relationship between *nested variables* and *dominating monitor transformation*, a transformation introduced in [4] which results in a monitor bounding the space complexity of the pre-transformation monitor. This transformation has been an essential component of many of our previous results [1, 2, 3, 4].

*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.

In this paper, we formally define the relationship between the pre- and post-transformation monitors, that is what changes occur to the quantifier intervals during the transformation. These results can be extended to specifications with more than two nested quantifiers, and ultimately to any specification. We show that our method provides a worst case $O(k)$-approximation where parameter k is determined by a relationship between quantifier intervals, and by $O(k)$-approximation we denote the ratio between the computed space complexity and actual space complexity. In practice more than three nested quantifiers are rare; thus, for the most commonly occuring cases a precise definition of k is possible. Interestingly, the approximation ratio is inversely related to the monitoring efficiency, thus "poorly" approximated specifications give rise to efficient monitors.

Finding memory efficient classes of specifications has also been an important goal for many prominent monitor specification languages, because the systems running the monitors will usually have very limited resources. For example, LOLA specifications can be represented as weighted directed graphs which are known to be memory efficient when the graph does not have positive cycles [5]. Problematically, positive cycles are related to interdependency of expressions, a property which can be useful for complex scenarios. It is known that there are memory efficient LOLA expressions which have positive cycles [5] thus begging for deeper analysis. We are not aware of any work classifying such expressions. There is a relationship between the LogicGuard quantifiers addressed in this work and positive cycles which is due to be addressed in future work. Other results concerning efficient fragments focus on restricting the fragment of LTL (the most commonly used specification language [11]) being considered. For instance, the hardware design language PSL [7] based on LTL restricts the use of disjunction to avoid exponential blow up. In [8], the class of "locally checkable" properties (a subclass of the "locally testable" properties introduced in [12]) is defined. In [6] 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. Unlike their results, which define space efficiency by the size of non-deterministic automaton, we base our analysis on the size of the runtime representation of the formulas during the execution of the monitors.

The rest of this paper is as follows: In Section 2 we introduce the background material and previous results. In Section 3 we present the approximation ratios and state their correctness. In Section 4 we discuss our results and future work. Appendices A & B contain the proofs of the main theorems found in Section 3.

2 Background & Previous Results

In this section we discuss the specification language, our abstraction of the language, and previous results. We leave out details concerning the operational semantics of the specification language and conversion of monitor specifications to their runtime representation, all of which has been discussed elsewhere [9, 13]. Instead, we present an abstraction of the operational semantics which is a slight generalization of the abstraction introduced in [2, 3]. This generalization is necessary, because we need to compare the evaluation of dominating and non-dominating monitor specifications. Previous work only considered dominating monitor specifications and thus we developed an abstraction which took only those specifications into account [2, 3].

2.1 The LogicGuard Specification Language

The LogicGuard specification language [13] is used for defining monitors over event streams. An important feature of the language is the construction of a higher level stream from a lower level

input stream. These streams can be configured to monitor the input for a particular property. The results of this paper mainly concern the core specification language, but to give an idea of the expressive power of the full specification language we present the following example of a specification in the full language:

```
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);
```

In this specification we declare the abstract types `tcp` and `message` as well as external functions and predicates operating on objects of these types. The stream `IP` of TCP/IP datagrams is declared that is connected by the runtime system to the network interface. This stream is what we refer to as an external stream. 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. 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 units a second message appears satisfying the `match` predicate on input `S`.

The core specification language is a substantially simplified version of this language that is used to analyse the complexity of monitoring with respect to the number of past messages and the number of only partially evaluated formula instances that have to be preserved in memory for the monitoring of the specification. To use our space complexity work within the LogicGuard system [13] we implemented a translator from the full language to the core language (the translation is not semantics-preserving but generates a specification for which monitoring is at least as complex as the monitoring of the original one). If some elements cannot be directly translated user settings provide guidance to direct the translation process.

2.2 The Core Specification Language

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

Figure 1: The core language syntax

The core specification language is described in detail in [4]. 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 $[B_1, B_2]$.

2.3 Dominating Formula Transformation

A concept introduced in [4], *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 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 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 is of the form* $x + n$ *(we omit the formal details, see the example below).*

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 $D(M)$*.*

Fig. 2 illustrates the dominating monitor transformation: The left side 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 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.

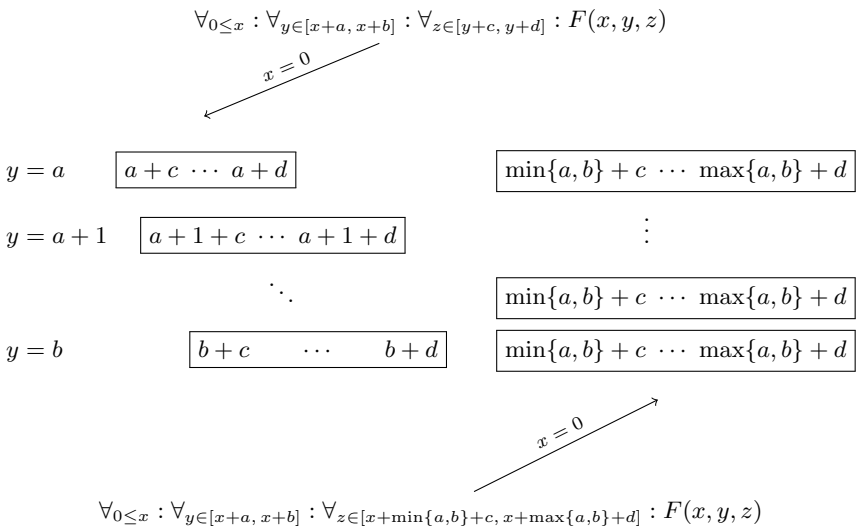

Figure 2: Illustration of the dominating monitor transformation.

Theorem 1 ([1, 3])**.** *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 $S \leq S'$.*

By $T(M) \multimap_{p,s,n} S$ we mean that S is the number of instances created by the runtime representation of M, $T(M)$, during the first n steps of the evaluation on a stream s starting at stream position p. Essentially, Thm. 1 states that the number of instances constructed by a monitor specification during evaluation is bounded from above by the dominating form of the monitor specification. More details can be found in [1, 2, 3].

2.4 Quantifier Trees & Vectors

The algorithm introduced in [3] and the analytic expressions of [1, 2] are based on a translation of monitor formulas into what we refer to as *quantifier trees*, and in the case of [2], the more restricted *quantifier vectors*. Not all details concerning these abstract objects are needed for the results of this paper. We will focus on the important concepts which directly apply to this paper and we refer the reader to [1, 2] for further details.

Definition 2 (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. We define $(y, b_1, b_2, Q)_1 = y$,$(y, b_1, b_2, Q)_2 = b_1$, $(y, b_1, b_2, Q)_3 = b_2$, and $(y, b_1, b_2, Q)_4 = Q$.*

Definition 3 (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:*

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

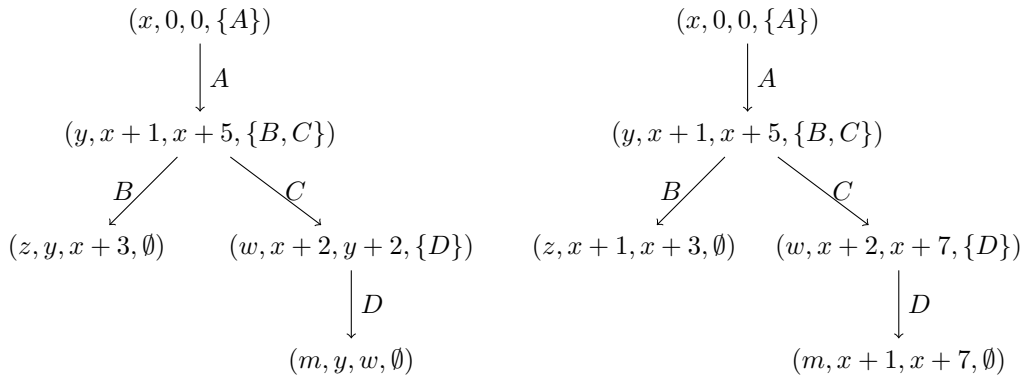

Figure 3: Quantifier trees for the monitor of Ex. 1.

We do not distinguish between sequential and parallel conjunction in quantifier trees because their effect on memory is dependent on the stream they are evaluated over, thus, we just take the worst case scenario of parallel conjunction. Fig. 3 depicts the quantifier tree $QT(M)$ on the left side and $QT(D(M))$ on the right side, where M is the monitor from Ex. 1. Note that $QT(D(M))$ consists of two quantifier vectors: $(y, x+1, x+5, \{(z, x+1, x+3, \emptyset)\})$ and $(y, x+1, x+5, \{(w, x+2, x+7, \{(m, x+1, x+7, \emptyset)\})\})$.

Definition 4 (Sub-quantifier Tree)**.** *Let $q, q' \in \mathbb{QT}$. We call q' a* sub-quantifier tree *of q if $q = (y, b_1, b_2, Q)$ and one of the following holds: $q' = q$, or $q' \in Q$, or there exists $q'' \in Q$ such that q' is a* sub-quantifier tree *of q''. We call a sub-quantifier tree q' of q* proper *if $q' \neq q$. We will use the notation $q.\nu$ to denote a sub-quantifier tree ν of q.*

Definition 5 (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 vectors. We define* the length $|v|$ of a quantifier vector v *as $|v| = 0$ if $v = \emptyset$, $|v| = 1$ if $v = (y, b_1, b_2, \emptyset)$, and $|v| = 1 + |q|$ if $v = (y, b_1, b_2, Q)$ and $q \in Q$.*

In previous work we ignored *nested variables* [1, 2]. By nested variables we mean a bound variable whose value is dependent on the value of another free or bound variable in the specification. Ignoring such cases allowed us to ignore every variable in the specification but the stream variable. However, to deal with non-dominating monitors we need an evaluation procedure which can handle nested variables. We solve this issue by adding term substitution to the evaluation procedure.

Definition 6. *A term t is an element of $B \setminus \{\infty\}$ (see Fig. 2). We define $T = B \setminus \{\infty\}$ as the set of all terms.*

Sometimes we will write a term as $t(y)$, meaning that y may be free in t.

Definition 7 (Quantifier Tree substitution)**.** *Let $q \in \mathbb{QT}$, $a, c_1, c_2 \in \mathbb{Z}$, $x_1, \cdots x_n, y_1, y_2, y \in \mathbb{V}$, and $b_1, b_2, t_1, \cdots t_n \in T$, where $q = (y, b_1(y_1), b_2(y_2), Q)$. We define $q\sigma$, where $\sigma = \{x_1 \leftarrow t_1, \cdots, x_n \leftarrow t_n\}$, as follows: $(y, b_1(y_1), b_2(y_2), Q)\sigma = (y, b_1(y_1)\sigma, b_2(y_2)\sigma, \bigcup_{q \in Q'} q\sigma)$, where for $j = 1, 2$ we have:*

$$b_j(y_j)\sigma = \begin{cases} b_j(t_i) & x_i = y_j, \text{for some } 1 \leq i \leq n \\ b_j(y_j) & \textit{otherwise} \end{cases}$$

Essentially, we are using a standard variable substitution method. We assume all quantifier trees we will be dealing with are *proper*, that is have a root node of the following form $Q = (x, 0, 0, Q')$ and for every $Q.\nu$ and $Q.\mu$, such that $Q.\nu \neq Q.\mu$, $(Q.\nu)_1 \neq (Q.\mu)_1$. This assumption allows us to define an initial set for our evaluation method.

Definition 8. *Let $Q \in \mathbb{QT}$ be proper and let $a \in \mathbb{N} \cup \{0\}$. We define the* initial instance set *$I(a, Q)$ of Q at a, as $I(a, Q) = \bigcup_{q \in Q'} q\,\{x \leftarrow a\}$.*

Essentially the instance set is a forest of quantifier trees. Now we can construct the instance set at a particular point for any quantifier tree.

Definition 9 (Evaluation of Quantifier Trees)**.** *Let $Q \in \mathbb{QT}$, $a, b \in \mathbb{N} \cup \{0\}$, such that $b \leq a$ and I is the initial instance set of Q at b. We define the* evaluation $E(a, I)$ at I *as follows:*

$$E(a, I) = \bigcup_{\substack{q \in I \\ q = (y, b_1, b_2, Q')}} (ACC(a, q) \cup REM(a, q))$$

$$ACC(a, q) = \begin{cases} \bigcup_{i=b_1}^{\min\{a, b_2\}} E(a, \bigcup_{q' \in Q'} q'\,\{y \leftarrow i\}) & b_1 \leq a \ \wedge \ b_1 \leq b_2 \\ \emptyset & \text{otherwise} \end{cases}$$

$$REM(a, q) = \begin{cases} \{(y, a+1, b_2, Q')\} & b_1 \leq a \leq b_2 \\ \{(y, b_1, b_2, Q')\} & a \leq b_1 \\ \emptyset & \text{otherwise} \end{cases}$$

In Definition 9 we break the process of evaluating a set of quantifier trees into two parts. The first part is denoted by $ACC(a, q)$, which is the ACCumulation of instances. This step generates a new evaluation function for each instance generated from unrolling the quantifier. The second part is denoted by $REM(a, q)$, which is the REMaining part of the quantifier interval which could not be evaluated at a. As we have done in previous work [1, 2], we can use our definition of evaluation to define the size of the *runtime representation*, or in other words, the number of instances kept in memory. Though, in previous work computing the maximum upper bound of a quantifier interval in a monitor specification was easy to define due to the dominating formula transformation. To define the maximum upper bound for the above evaluation procedure, we need to find the first value i such that $E(i, I(0, Q)) = \emptyset$, i.e. the point when everything evaluates. Obviously, if $E(i, I(0, Q)) = \emptyset$, then $E(i + 1, I(0, Q)) = \emptyset$ as well.

Definition 10 (Storage Complexity)**.** *Let $m \in \mathbb{M}$, and $a \in \mathbb{N} \cup \{0\}$, such that $a = (\min_{0 \leq i} E(i, I(0, QT(m))) = \emptyset)$. Then the* storage complexity *of m, i.e. $|m|_{sc}$ is defined as follows:*

$$|m|_{sc} = \sum_{i=0}^{a-1} |E(i, I(0, QT(m)))|$$

3 Approximation Bounds

In this section we first introduce approximation ratios for monitor specifications which can be represented as proper quantifier vectors of length three, and then we generalize these results to arbitrary quantifier trees. Note that we only consider variable nestings for the upper bound of the innermost quantifier. We ignore the lower bounds, because variable lower bounds do little to influence how far into the future a monitor needs to look, rather they effect how long one has to wait till evaluation can take place. However, when considering quantifier vectors of length greater than three, the number of instances only depends on the complexity of the sub-trees, but is independent of the quantifier itself.

3.1 Approximations for Length Three Quantifier Vectors

We derive our first approximation ratios by comparing the number of instances generated by the quantifier vector with nested variables and the resulting quantifier vector of the dominating formula transformation. Note that we need to deal with an abstract representation of the quantifier vector's dominating form because we do not know the precise values of the upper and lower bounds.

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*

$$QT(m) = (x, 0, 0, (y, x + a_1, x + b_1, (z, x + a_2, y + b_2, \emptyset))),$$

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

Proof. The full proof can be found in Appendix A. Essentially, we proceed by considering all possible numeric orderings of a_1, b_1, a_2 and b_2. We evaluate m and $D(m)$ for each ordering using Definition 9. Then we compare the results and show that $2 \cdot |m|_{sc} \geq |D(m)|_{sc}$. □

The bound provided by Theorem 2 is quite coarse and experimental results point to a tighter bound of $\frac{3}{2} \cdot |m|_{sc} \geq |D(m)|_{sc}$ in the general case. As the difference between b_1 and b_2 grows, experimental results show that $|m|_{sc} \approx |D(m)|_{sc}$. Proving these points is technical and provides little gain. What is important is that this class of monitor specifications has a constant approximation ratio and thus shows the runtime representation size of the dominating formula is a good approximation of the runtime representation size of the original formula.

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

$$QT(m) = (x, 0, 0, (y, x + a_1, x + b_1, (z, x + a_2, y + b_2, \emptyset)))$$

and

$$D(QT(m)) = (x, 0, 0, (y, x + a_1, x + b_1, (z, x + a_2, x + \max\{a_1, b_1\} + b_2, \emptyset))).$$

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

Proof. The full proof can be found in Appendix B and is similar to the proof of Theorem 2. □

The previous two theorems give us a very good idea of how well or how poorly a monitor specification will behave based on the relationship between b_1 and b_2, the quantifier upper bounds. In the next subsection we generalize these results for any quantifier tree. To make this generalization we need a clear definition of *an $O(r)$-approximation* of the storage complexity.

Definition 11. *Let $m \in \mathbb{M}$ and $f : \mathbb{N} \to \mathbb{N}$. We say that $|QT(D(m))|_{sc}$ is an $O(f(k))$-approximation of $|QT(m)|_{sc}$ if $O(f(k)) \cdot |QT(m)|_{sc} \geq |QT(D(m))|_{sc}$.*

For Theorem 2, the statement $2 \cdot |m|_{sc} \geq |D(m)|_{sc}$ can be thought of as a special case of $O(1) \cdot |m|_{sc} \geq |D(m)|_{sc}$.

To get a better idea of what Theorems 2 & 3 are stating let us consider the following two pairs of formula which where used for the experimental results of [3].

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

$$D(m_1) = \forall_{0 \leq x} : \forall_{y \in [x, x+80]} : \forall_{z \in [x, x+81]} : @z \tag{1b}$$

$$m_2 = \forall_{0 \leq x} : \forall_{y \in [x, x+40]} : \forall_{z \in [x, y+40]} : @z \tag{2a}$$

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

Notice that (1b) and (2b) are the dominating formulas of (1a) and (2a). Also the bounds of (1a) fall into the class of Theorem 3 and the bounds of (2a) fall into the class of Theorems 2. Now using the theorems we can make the following calculations:

$$O(n) = c \cdot 80 \geq \frac{|D(m_1)|_{sc}}{|m_1|_{sc}} = \frac{3401}{161} = 21.12 \qquad 2 \geq \frac{|D(m_2)|_{sc}}{|m_2|_{sc}} = \frac{2500}{1680} \approx 1.488$$

Thus, we see that $c \geq \frac{1}{4} + \epsilon$. We show the convergence of Theorem 2 (left) and divergence of Theorem 3 (right) in Fig. 4 using the following two specification schema:

$$\forall_{0 \leq x} : \forall_{y \in [x, x+100]} : \forall_{z \in [x, y+100+10 \cdot n]} : @z \tag{3a}$$

$$\forall_{0 \leq x} : \forall_{y \in [x, x+100+10 \cdot n]} : \forall_{z \in [x, y+100]} : @z \tag{3b}$$

where $n \in [0, 200]$. The two lines in Fig. 4 (left) represent the cases when $\frac{3}{2} = \frac{|D(m)|_{sc}}{|m|_{sc}}$ and $1 = \frac{|D(m)|_{sc}}{|m|_{sc}}$. We know that $\frac{|D(m)|_{sc}}{|m|_{sc}}$ can never be less than 1 nor greater than 2. However, experimentally, it has never shown to be more than $\frac{3}{2}$. The curve of Fig. 4 (left) illustrates what happens as the distance between b_1 and b_2 increases whilst b_1 remains smaller than b_2, that is the ratio $\frac{|D(m)|_{sc}}{|m|_{sc}}$ converges to 1. In the case of Fig. 4 (right), the opposite happens, as the distance between b_1 and b_2 increases and $b_2 \leq b_1$, there is a divergence and the ratio $\frac{|D(m)|_{sc}}{|m|_{sc}}$ goes to infinity. This divergence occurs as a linear growth in terms of the distance between b_1 and b_2. Importantly, this divergence does not imply that the space requirements of the monitor gets worse for large distances between b_1 and b_2, but rather that $D(m)$ provides less and less information concerning the space requirements of m. The convergence of Fig. 4 (left) implies that $D(m)$ provides more information concerning the space requirements of m.

3.2 Aproximations for Quantifier Trees

In this section we use the previous section's results to derive approximations for quantifier trees.

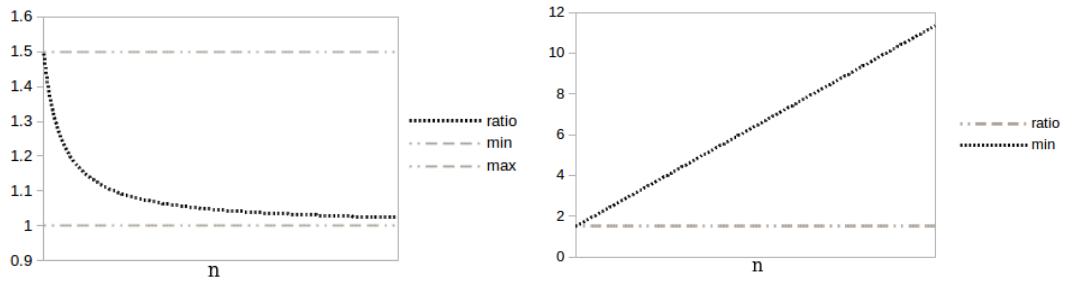

Figure 4: Approximation ratios for the specification schema of Equation (3a) & (3b).

Lemma 1. *Let* $m \in \mathbb{M}$, $a_1, b_1, a_2, b_2 \in \mathbb{Z}$, $x, y, z \in V$, *and* $Q, Q' \subset \mathbb{QT}$ *such that* $q \in Q$ *iff* $D(q) \in Q'$ *and* $|Q'|_{sc}$ *is a* $O(k_1^r)$*-approximation, where* $r \in \{0, 1\}$. *If*

$$D(QT(m)) = (x, 0, 0, (y, x + a_1, x + b_1, (z, x + a_2, x + \max\{a_1, b_1\} + b_2, \emptyset)))$$

is an $O(k_2^{r'})$*-approximation, where* $r' \in \{0, 1\}$, *of* $QT(m) = (x, 0, 0, (y, x + a_1, x + b_1, (z, x + a_2, y + b_2, \emptyset)))$, *then*

$$D(QT(m)) = (x, 0, 0, (y, x + a_1, x + b_1, (z, x + a_2, x + \max\{a_1, b_1\} + b_2, Q')))$$

is at most an $O(\max\{k_1, k_2\}^{\max\{r, r'\}})$*-approximation of* $QT(m) = (x, 0, 0, (y, x + a_1, x + b_1, (z, x + a_2, y + b_2, Q)))$.

Proof. This lemma is complex to state and a bit unintuitive but the proof is pretty straight forward. One just needs to consider the fact that the variable substitutions provide a linear change in the size of a quantifier's interval. Thus, a sequence of nested quantifier can be reduced to two nested quantifiers with a linear shift in one of the two quantifiers upper bounds. Thus, the linear approximation might be worse, but it stays a linear approximation. □

We can illustrate Lemma 1 using the following three specification schema for $n \in [0, 200]$:

$$\forall_{0 \leq x} : \forall_{y \in [x, x+1]} : \forall_{z \in [x, y+1+n]} : \forall_{w \in [x, z+1+2 \cdot n]} : @w \tag{4a}$$

$$\forall_{0 \leq x} : \forall_{y \in [x, x+1+n]} : \forall_{z \in [x, y+1+2 \cdot n]} : \forall_{w \in [x, z+1]} : @w \tag{4b}$$

$$\forall_{0 \leq x} : \forall_{y \in [x, x+1+2 \cdot n]} : \forall_{z \in [x, y+1+n]} : \forall_{w \in [x, z+1]} : @w \tag{4c}$$

Equation (4a) is $O(1)$-approximated by the dominating formula while both Equation (4b) and Equation (4c) are $O(n)$-approximated by the dominating formula, but by different linear functions. The relationship between the upper bounds of the inner most quantifiers of Equation (4b) is defined by Theorem 2 and the relationship between the upper bounds of the outer quantifiers is defined by Theorem 3. However for Equation (4c) the relationship of both the inner and outer bounds is defined by Theorem 3. For (4a) the relationship of both the inner and outer bounds is defined by Theorem 2. In Fig. 5, we provide a graphical representation of the relationship between the three schemata and the value n.

First, we need to formalize a relationship between the sub-trees of a quantifier tree.

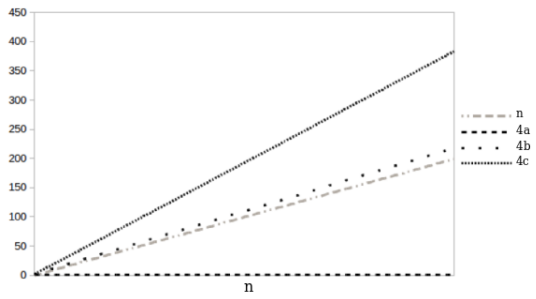

Figure 5: Approximation ratios for the specification schema of Equations (4a), (4b), & (4c).

Definition 12. *Let* $q \in \mathbb{QT}$. *We say that* $q.\nu$ is related to $q.\nu.\mu$, *where* $q.\nu.\mu \neq q.\nu$, *if* $(q.\nu.\mu)_3 = y + b_2$, *where* $y \in V$ *and* $b_2 \in \mathbb{Z}$, *and* $(q.\nu)_1 = y$. *We define the set of pairs* $\mathbf{SR}_q$ *as follows:* $(q', q'') \in \mathbf{SR}_q$ *if* $\exists\ q.\nu = q'$ *and* $q'.\mu = q''$ *such that* $q' \neq q'.\mu$ *and* q' is related to q''.

Definition 13. *Let* $q \in \mathbb{QT}$. *We say that a set* $S \subseteq \mathbf{SR}_q$ *is a* relation chain of q *if the undirected graph* (V, S), *where* $V = \{q \mid \exists r \in \mathbb{QT}((q, r) \in S \vee (r, q) \in S)\}$, *is a connected path.*

Definition 14. *Let* $q \in \mathbb{QT}$. *We define the* the class index of $q.\nu$ and $q.\nu.\mu$, $C_i(q.\nu, q.\nu.mu)$, *where* $q.\nu.\mu \neq q.\nu$, $(q.\nu)_3 = y + b_1$, *and* $(q.\nu.\mu)_3 = y + b_2$, *as* $C_i(q.\nu, q.\nu.mu) = \max\{b_1 - b_2, 2\}$. *We define the* class index of q, $C_i(q)$, *as* $C_i(q) = \max_{(q',q'') \in \mathbf{SR}_q} \{C_i(q', q'')\}$.

Now we can describe the worst-case approximation of a quantifier tree by its dominating formula.

Theorem 4. *Let* $m \in \mathbb{M}$ *and* S *the longest relation chain of* $QT(m)$. *Then* $|QT(D(m))|_{sc}$ *is at most an* $O(n)$*-approximation of* $|QT(m)|_{sc}$, *where* n *is dependent on* $C_i(QT(m))$.

Proof. The characterisation index $C_i(QT(m))$ tells us if we are in the class of Theorem 2 or Theorem 3. More precisely, the longest relation chain of $QT(m)$, S tells us the number of times we need to apply Lemma 1, and thus, whether we have a linear or constant approximation of the runtime representation. The result follows from these statements. □

The surprising thing about Theorem 4 is that a poorer approximation implies a smaller runtime representation. Even more so, the better the dominating formula is at approximating the space requirements, the worse a monitor will behave in terms of space. Thus, the best monitors in terms of space are the ones with maximal characterisation index and maximizal relation chain size. These results can be used by those writing monitor specifications in the LogicGuard language as a guideline towards an efficient construction.

4 Conclusion

We analysed the effects of the assumptions made in previous work [1, 2, 3, 4]. The goal was to study how much our assumptions influence the resulting bounds [1, 2] and how close to the true space requirements our resulting bounds are. We performed this analysis by defining a more precise evaluation method than the ones introduced in [2]. This allowed us to compare the evaluation of a coarse abstraction of a monitor specification to the evaluation of an abstraction

closely resembling the original specification. This analysis resulted in approximation ratios dependent on the relationship between quantifiers in the monitor specification and provides us with a method to classify efficient and inefficient monitor specifications. Thus, we can provide guidelines for efficiently writing monitor specifications using the LogicGuard Language. It is an open problem as to whether these are the only guidelines for writing memory efficient specifications or if other properties influence memory as well. Also, we would like to investigate possible optimizations based on this work which can be applied to existing specifications.

References

[1] David M. Cerna, Wolfgang Schreiner, and Temur Kutsia. Analytic and Algorithmic Methods for Computing the Space Requirements of Stream Monitor Specifications. 2016. Submitted for publication.

[2] David M. Cerna, Wolfgang Schreiner, and Temur Kutsia. Better Space Bounds for Future-Looking Stream Monitors. 2016. Submitted for publication.

[3] David M. Cerna, Wolfgang Schreiner, and Temur Kutsia. Predicting Space Requirements for a Stream Monitor Specification Language. In *Runtime Verification: 16th International Conference, RV 2016, Madrid, Spain, September 28-30, 2016. Proceedings*, pages 135–151. Springer International Publishing, 2016.

[4] 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 (ed.), editors, *Proceedings of The 7th International Symposium on Symbolic Computation in Software Science*, volume 39 of *EPiC Series in Computing*, pages 29–41, 2016.

[5] B. D'Angelo, S. Sankaranarayanan, C. Sanchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, and Z. Manna. LOLA: Runtime Monitoring of Synchronous Systems. In *12th International Symposium on Temporal Representation and Reasoning (TIME'05)*, pages 166–174, June 2005.

[6] 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.

[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.

A Proof of Thm. 2

First let us consider the case when $b_1 < a_1$. If $a_1 < 0$ then $|m|_{sc} = |D(m)|_{sc} = 0$ and the inequality obviously holds. By Def. 9, we violate the condition of the second union in LOW and $HIGH$ and Rem evaluates to empty. If $a_1 > 0$, we have to wait till we reach the future position a_1 to evaluate the formula. Thus, $|m|_{sc} = |D(m)|_{sc} = a_1$ and it is obvious that the inequality holds. By Def. 9, we violate the condition of the second union in LOW and $HIGH$, but Rem cannot be evaluated till a_1 is reached. Thus, we see the theorem holds in the case of $b_1 < a_1$.

When $a_1 < b_1$ the next case is when $b_2 < a_2$. If it is also the that $b_2 + b_1 < a_2$ then we are back to the previous case and it is trivial to show that the inequality holds. Essentially, without going into details, the difference between m and $D(m)$ does not show up in this case, because a_2 is the largest position in the second interval.

Otherwise, there exists some $c \in [a_1, b_1]$ such that $a_2 \leq c + b_2$. It is possible that $c = a_1$. We get the following two quantifier trees:

$$Q'_f = (x, 0, 0, (y, x + c, x + b_1, (z, x + a_2, y + b_2, \emptyset)))$$

$$Q'_{D(f)} = (x, 0, 0, (y, x + c, x + b_1, (z, x + a_2, x + b_1 + b_2, \emptyset))).$$

Now let us consider the case can $b_1 + b_2 \leq 0$. Again, everything will be evaluated right away and the inequality trivially holds. Simply, This implies the interval is negative. Next let us consider when $b_1 \leq 0$, but $0 < b_1 + b_2$. To remove trivial cases again, let us consider the first position $d \in [c, b_1]$ such that $0 < d + b_2$. Again it is possible that $d = a_1$. It is quite obvious that $E(b_1 + b_2, I(0, QT(m))) = \emptyset$. Using this fact we can construct the storage complexities as follows:

$$|m|_{sc} = \sum_{i=0}^{a_2-1} E_i(QT(m)) + \sum_{i=a_2}^{b_2+d-1} E_i(QT(m)) + \sum_{i=b_2+d}^{b_1+b_2-1} E_i(QT(m))$$

$$|D(m)|_{sc} = \sum_{i=0}^{a_2-1} E_i(QT(D(m))) + \sum_{i=a_2}^{b_2+d-1} E_i(QT(D(m))) + \sum_{i=b_2+d}^{b_1+b_2-1} E_i(QT(D(m)))$$

where $E_i(qt) = |E(i, I(0, qt))|$. Essentially, prior to $b_2 + d$ the two formulas evaluate identically. Thus, the first two summations can be replaced by a constant c. Thus we derive,

$$|m|_{sc} = c + \sum_{i=d}^{b_1+b_2-1} E_i(QT(m)) \qquad |D(m)|_{sc} = c + \sum_{i=d}^{b_1+b_2-1} E_i(QT(D(m)))$$

Now we have to deal with the interval $[b_2 + d, b_1 + b_2 - 1]$. It is easy to see that for $i \in [b_2 + d, b_1 + b_2 - 1]$,

$$E_{i-1}(QT(m)) = E_i(QT(D(m))) + 1 \qquad E_{i-1}(QT(D(m))) = E_i(QT(D(m))).$$

We know that $|E(|d-1|, I(0, QT(m)))| = |E(|d-1|, I(0, QT(D(m))))| = b_1 - d$ because $b_1 + b_2 - 1 - (b_2 + d) + 1 = (b_1 - d)$. These equations result in the sums

$$\sum_{i=1}^{b_1-d} i = \frac{(b_1 - d) \cdot (b_1 - d + 1)}{2} \qquad \sum_{i=1}^{|b_1-d|} |b_1 - d| = (b_1 - d)^2$$

respectively. Plugging everything into the inequality from the theorem statement we get

$$2 \cdot \left(c + \frac{(b_1 - d) \cdot (b_1 - d + 1)}{2} \right) \geq c + (b_1 - d)^2$$

which reduces to $\frac{c}{(b_1 - d)} + 1 \geq 0$ implying that the inequality holds.

If $b_1 > 0$ there are two cases to consider but we only consider the case $b_1 < d + b_2$. Prior to position $d + b_2$ the evaluation is the same for m and $D(m)$. Thus, the storage complexity of the the two formulas can be written as follows:

$$|m|_{sc} = c + \sum_{i=b_2+d}^{b_1+b_2-1} E_i(QT(m)) \qquad |D(m)|_{sc} = c + \sum_{i=b_2+d}^{b_1+b_2-1} E_i(QT(D(m)))$$

as in the previous case. It is easy to see that this case behaves exactly like the previous case because again for $i \in [b_2 + d, b_1 + b_2 - 1]$,

$$E_{i-1}(QT(m)) = E_i(QT(m)) + 1 \qquad E_{i-1}(QT(D(m))) = E_i(QT(D(m))).$$

Thus, we can use the same argument from the previous case to show that the inequality holds.

B Proof of Thm. 3

Again, everything that happens prior to position $d + b_2$, as shown in Theorem 2 is pretty much irrelevant and can be summed up as a constant c. We know by our assumption that $b_1 >= d+b_2$ that for $i \in [d + b_2, b_1 - 1]$, $|E(i-1, I(0, QT(m)))| = |E(i, I(0, QT(m)))|$, However, $|E(i - 1, I(0, QT(D(m))))| + 1 = |E(i, I(0, QT(D(m))))|$. Also we know that $|E(d + b_2 - 1, I(0, QT(m)))| = |E(d + b_2 - 1, I(0, QT(D(m))))| = b_2$ because the first unrolling happens at d. This results in the following two sums (we assume $b_1 = (k + b_2 + d)$):

$$\sum_{i=d+b_2}^{b_1-1} b_2 = \sum_{i=d+b_2}^{b_1-1} b_2 = b_2 \cdot (b_1 - b_2 - d) = k \cdot b_2$$

$$\sum_{i=d+b_2}^{b_1-1} b_2 + i = \sum_{i=0}^{b_1-1-d-b_2} i =$$

$$b_2 \cdot (b_1 - b_2 - d) + \frac{(b_1 - b_2 - d - 1) \cdot (b_1 - b_2 - d)}{2} = k \cdot b_2 + \frac{(k-1) \cdot k}{2}$$

For $i \in [b_1, b_1 + b_2 - 1]$, we $|E(i - 1, I(0, QT(m)))| = |E(i, I(0, QT(m)))| + 1$ and $|E(i - 1, I(0, QT(D(m))))| = |E(i, I(0, QT(D(m))))$. This results in the following two sums (we assume $b_1 = (k + b_2 + d)$):

$$\sum_{i=b_1}^{b_1+b_2-1} |E(i, I(0, QT(m)))| = \sum_{i=0}^{b_2} i = \frac{(b_2 + 1) \cdot b_2}{2}$$

$$\sum_{i=b_1}^{b_1+b_2-1} |E(i, I(0, QT(D(m))))| = \sum_{i=b_1}^{b_1+b_2-1} (b_1 - d) + 1 = b_2 \cdot (k + b_2 + 1)$$

The following derivation shows that the theorem holds:

$$O(k)\left(c + k \cdot b_2 + \frac{(b_2+1)\cdot b_2}{2}\right) \geq c + k \cdot b_2 + \frac{(k-1)\cdot k}{2} + b_2 \cdot (k + b_2 + 1)$$

$$O(k)c + O(k)k \cdot b_2 + O(k)((b_2+1)\cdot b_2) \geq c + k \cdot b_2 + \frac{(k-1)\cdot k}{2} + b_2 \cdot (k + b_2 + 1)$$

$$O(k)c + O(k)k \cdot b_2 + O(k)((b_2+1)\cdot b_2) \geq \frac{(k-1)\cdot k}{2}$$

$$O(k^2)b_2 \geq \frac{(k-1)\cdot k}{2} \implies \frac{O(k^2)b_2}{\frac{(k-1)\cdot k}{2}} \geq 1 \implies O(1)b_2 \geq 1$$

Note that c, b_1, and b_2 are linear in k.