

# Quantifier Elimination for Queues

Christian Straßer

February 11, 2006

## Abstract

In this paper the author extends one-sorted first-order languages and structures by a second sort queues. The central result is a quantifier elimination algorithm for queue-quantifiers for an arbitrary given element-theory  $T$ . Also full quantifier elimination, the existence of an quantifier elimination procedure, completeness and/or decidability of  $T$  extend to the corresponding theory of queues. As queues are a widely used data structure in programming languages the framework is set for applications in manifold areas, where it is important to prove and decide statements about domains containing queues, like for example in informational flow control for software security.

*Key Words:* Quantifier Elimination, queues, Verification

AMS Subj. Class.: 03C10

## 1 Introduction

The main motivation throughout the literature on quantifier elimination in first-order languages extended with queues comes from software and hardware verification. Although stacks can be easily interpreted in many-sorted versions of term algebras, a lot of investigation has been made for the case with queues, for which this is not possible. After Bjørner [1999] presented a decision procedure for the existential theory of queues, Rybina and Voronkov [2001] came up with a quantifier elimination and a proof for the decidability for term algebras with queues. Both, as well as the present approach, take extensively advantage of the periodicity of queues. My investigations were born out of the wish to generalize Rybina and Voronkov's results for arbitrary one sorted first order languages and structures in order to show, that quantifier elimination, decidability as well as completeness are preserved by extension with queues, and to demonstrate a generic quantifier elimination procedure for them. Zhang et al. [2005] gave a quantifier elimination procedure for the extension of the theory of queues with length functions, which results in a combination of this theory with Presburger arithmetic.

Especially in the context of query languages and their applications, data bases, a lot of research has been taken place. Benedikt et al. [2003] investigated in different structures  $(\Sigma^*, \Omega)$ , where the alphabet  $\Sigma$  is finite, and  $\Omega$  consists of functions like for example  $l_a$ ,  $a \in \Sigma$ , where  $l_a(x) = x \cdot a$ , binary predicates  $el(x, y)$  stating that  $x$  and  $y$  have the same

length and  $x \preceq y$  stating that  $x$  is a prefix of  $y$ . As central results they present quantifier elimination for a structure  $\mathbf{S}^+$ , which deals with an expansion of the signature for the structure  $\mathbf{S}_{reg}$ , which itself has as definable subsets of  $\Sigma^*$  exactly the regular languages. Analogous they deal with the structure  $\mathbf{S}_{left}$ , which's definable subsets are precisely the star-free languages.

Unlike in the present investigation, the underlying element theories in Zhang et al. [2005] and Benedikt et al. [2003] are empty. This paper is structured in the following way: The central results will be presented in chapter two. In chapter three we extend languages by queues and show that in order to achieve quantifier elimination, a signature with only ladd, radd is not sufficient. Chapter four is the heart of the paper presenting quantifier elimination for quantifiers binding variables of basic type, as well as quantifiers binding variables of queue type. In chapter five it is shown that properties like decidability, completeness, substructure completeness and model completeness are preserved by queue extensions. Chapter six provides some applications as well as some outlook for possibilities of further investigations in this area.

Throughout the paper the sometimes very technical proofs will be skipped, or we will provide only insight in major steps of our argumentation. The interested reader is referred to Straßer [2006].

## 2 Central Results

Here we want to state the central results. In the corresponding later chapters we will look at them more in detail.

**Theorem 2.1.** *If an arbitrary one-sorted first-order structure  $\mathbf{M}$  allows quantifier elimination, then there exists also a quantifier elimination for basic type variables in the extension of  $\mathbf{M}$  by the sort queue. In case there is an algorithmic quantifier elimination, then on this basis a quantifier elimination procedure for basic quantifiers can be constructed effectively for the queue extension structure.*

**Theorem 2.2.** *For an extension of an arbitrary one-sorted first-order structure  $\mathbf{M}$  by the sort queue there exists a quantifier elimination algorithm for queue-quantifiers.*

**Theorem 2.3.** *For a one-sorted first-order theory  $\mathbf{T}$  the properties completeness and/or decidability extend to the corresponding theory of queues.*

## 3 Extending Languages by Queues

We are interested in extending arbitrary one-sorted first-order languages by queues. In the following we will generalize the extension of term algebras with queues presented in Rybina and Voronkov [2001] to cases with arbitrary one-sorted logics. Therefore let  $\mathcal{L} = \{F, R, \mu\}$  be a language, for which  $F = \{f_i \mid i \in I_F\}$  describes a set of functions and  $R = \{\tau_i \mid i \in I_R\}$  describes a set of relations, so that  $I_F, I_R \subseteq \mathbb{N}$ . Beside the basic sort  $\alpha$  we want to introduce

a new sort "queue". We also extend our given set of function and constant symbols with the constant  $\varepsilon$  and the following functions operating on queues:

$$\text{ladd} : \alpha \times \text{queue} \rightarrow \text{queue}, \quad \text{radd} : \alpha \times \text{queue} \rightarrow \text{queue}.$$

This defines the language  $\mathcal{L}_q$ .

For a given  $\mathcal{L}$ -Structure  $\mathbf{M}$  with universe  $U(\mathbf{M})$  the universe  $U(\mathbf{M}')$  of the extended structure  $\mathbf{M}'$  is defined in the following way:

1. For each  $m \in U(\mathbf{M})$  :  $m$  is a element of sort  $\alpha$ .
2. For all  $m_1, \dots, m_n \in U(\mathbf{M})$ , with  $n \geq 1$ , the sequence  $m_1 \dots m_n$  is an element of the sort queue. The empty sequence will be called the *empty queue* and denoted with  $\varepsilon$

The semantics of our functions is given in the following way:

$$\text{ladd}(m, q) = mq, \quad \text{radd}(m, q) = qm$$

We can easily see that we have already the tools in hands to express the operations, which are typically used to describe the datatype queue in informatics: *push*, *top* and *pop*:

$$\begin{aligned} \text{push}(x, y) = z &\leftrightarrow \text{radd}(x, y) = z \\ \text{top}(x) = y &\leftrightarrow \exists z \text{ladd}(y, z) = x \\ \text{pop}(x) = y &\leftrightarrow \exists z \text{ladd}(z, y) = x. \end{aligned}$$

Vice versa we can define the functions of our extended language with the datatype operations in the following way:

$$\begin{aligned} \text{ladd}(x, y) = z &\leftrightarrow z \neq \varepsilon \wedge \text{top}(z) = x \wedge \text{pop}(z) = y \\ \text{radd}(x, y) = z &\leftrightarrow \text{push}(x, y) = z \end{aligned}$$

Unfortunately we find that the language  $\mathcal{L}_q$  doesn't allow quantifier elimination.

**Theorem 3.1.** *A  $\mathcal{L}_q$ -structure  $\mathbf{M}$  doesn't admit quantifier elimination.*

*Proof.* As a counterexample we take a look at the following formula  $\varphi \stackrel{\text{def}}{=} \exists x(\text{radd}(x, q) = \text{ladd}(x, q))$ . If there is an equivalent quantifier free formula, then there is also a quantifier free formula  $\psi$  for which  $\mathcal{V}(\psi) \subseteq \mathcal{V}_f(\varphi) = \{q\}$ . Such a formula would be a boolean combination of  $\perp, \top, q = \varepsilon$  and  $q \neq \varepsilon$ . It is obvious that in this way we cannot express the satisfaction set  $\{(c_1 \dots c_n) \mid \forall i, j \in \mathbb{N}_n : c_i = c_j, c_i, c_j \in M, n \in \mathbb{N}_2\} \cup \{\varepsilon\} \cup \{(c) \mid c \in M\}$ .  $\square$

Therefore we will extend our queue language in the following way. We introduce the following unary functions on queues:

$$\begin{aligned} \text{lhead} : \text{queue} &\rightarrow \alpha, \quad \text{rhead} : \text{queue} \rightarrow \alpha \\ \text{ltail} : \text{queue} &\rightarrow \text{queue}, \quad \text{rtail} : \text{queue} \rightarrow \text{queue}. \end{aligned}$$

In order to avoid partial functions we also introduce constant  $\eta$  of type  $\alpha$ , which is defined as the head of the empty queue:  $\text{head}(\varepsilon) = \eta$ , with  $\text{head} \in \{\text{lhead}, \text{rhead}\}$ . We define our new functions like follows:

$$\begin{aligned}\text{lhead}(q) &= y \leftrightarrow \exists w ((\text{ladd}(y, w) = q) \vee (q = \varepsilon \wedge y = \eta)) \\ \text{ltail}(q) &= w \leftrightarrow \exists y ((\text{ladd}(y, w) = q) \vee (y = \varepsilon \wedge w = \varepsilon))\end{aligned}$$

The functions  $\text{rhead}$  and  $\text{rtail}$  are defined analogously. Vice versa we have:

$$\begin{aligned}\text{ladd}(x, y) &= (z \leftrightarrow z \neq \varepsilon \wedge \text{lhead}(z) = x \wedge \text{rtail}(z) = y \wedge x \neq \eta) \vee (x = \eta \wedge y = z) \\ \text{radd}(x, y) &= (z \leftrightarrow z \neq \varepsilon \wedge \text{rhead}(z) = x \wedge \text{ltail}(z) = y \wedge x \neq \eta) \vee (x = \eta \wedge y = z)\end{aligned}$$

We further define functions on  $\eta$  in the following way:

$$\begin{aligned}\text{ladd}(\eta, q) &\stackrel{\text{def}}{=} q, \quad \text{radd}(\eta, q) \stackrel{\text{def}}{=} q \\ \text{for all } n \in \mathbb{N} : f \in F \text{ with } \mu(f) = n : f'(\eta, x_2, \dots, x_n) &\stackrel{\text{def}}{=} \\ f'(x_1, \eta, \dots, x_n) &\stackrel{\text{def}}{=} f'(x_1, \dots, \eta) \stackrel{\text{def}}{=} \eta\end{aligned}$$

**Theorem 3.2.** *For each formula  $\varphi$  we can construct effectively a equivalent formula  $\varphi'$  without occurrences of the function symbols  $\text{ladd}, \text{radd}$ . Formula  $\varphi'$  has a length of  $O(2^a)$  where  $a$  is the number of occurrences of  $\text{ladd}, \text{radd}$  in  $\varphi$ .*

## 4 Quantifier Elimination

### 4.1 Quantifiers binding Variables of Basic Type

#### 4.1.1 General Procedure

In this chapter  $\mathcal{L}$  is a first order one sorted language and  $\mathcal{L}_q$  is it's queue extension.

**Definition 4.1.** Let  $\varphi = \varphi_1 \vee \dots \vee \varphi_n$  be the DNF of a formula  $\phi$  without occurrences of  $\text{ladd}, \text{radd}$ . In each conjunction  $\varphi_i(q_{i,1}, \dots, q_{i,m})$  we want for  $M_j \in \text{Prefix}_{\text{head}}$  to replace terms of the form  $M_j q_{i,j}$  with  $a_{i,j}$ . Denote the corresponding index set with  $I$ . Terms with occurrences of  $\eta$  and  $M\varepsilon$  with  $M \in \text{Prefix}_{\text{head}}$  are replaced by  $\eta$ . Let the resulting conjunction be  $\varphi'_i$ . Define  $\psi_i \stackrel{\text{def}}{=} \varphi'_i \wedge \bigwedge_{j \in I} (M_j q_{i,j} = a_{i,j})$ . Then the formula  $\psi_1 \vee \dots \vee \psi_n$  is called the *basic type normal form (BTNF)* of  $\phi$ .

**Theorem 4.2.** *Let  $\varphi(a_1, \dots, a_n)$  be an extended  $\mathcal{L}$ -formula. For an arbitrary  $\mathcal{L}$ -structure  $\mathbf{M}$  and it's  $\mathcal{L}_q$ -extension  $\mathbf{M}'$  we have*

$$\begin{aligned}\text{Sat}_{\mathbf{M}}(\varphi) &= \text{Sat}'_{\mathbf{M}'} \left( \varphi \wedge \bigwedge_{i=1}^n a_i \neq \eta \right), \text{ where} \\ \text{Sat}_{\mathbf{M}}(\varphi) &\stackrel{\text{def}}{=} \{(a_1, \dots, a_n) \in U(\mathbf{M})^n \mid \mathbf{M} \models \varphi(a_1, \dots, a_n)\} \text{ and} \\ \text{Sat}'_{\mathbf{M}'} &\stackrel{\text{def}}{=} \{(a_1, \dots, a_n) \in U(\mathbf{M}')^n \mid \mathbf{M}' \models \varphi(a_1, \dots, a_n)\}.\end{aligned}$$

**Lemma 4.3.** *For the quantifier elimination we need a given formula  $\varphi$  in a special form. Let  $\varphi$  be an arbitrary conjunction of atomic formulas in BTNF*

$$\bigwedge_{i=1}^{m_1} ((t_{1,i} \tau_i t_{2,i}) \wedge \mu_i) \wedge \bigwedge_{i=m_1+1}^{m_2} (\neg(t_{1,i} \tau_i t_{2,i}) \wedge \mu_i),$$

where  $\tau_i, \rho_j \in R \cup \{=\}$  with  $i \in \mathbb{N}_{m_1}, j \in \mathbb{N}_{m_2}$  and  $\mu_i$  are atomic formulas of the form  $a = Mq$  (compare the definition of BTNF). We will denote  $[\neg](t_{1,i} \tau_i t_{2,i}) \wedge \mu_i$  with  $K_i$ .

It is true that  $\varphi$  is equivalent to  $\varphi' \stackrel{\text{def}}{=} \bigwedge_{i=1}^{m_2} K'_i$ , so that the following is valid for  $\varphi'$ :

- $\eta$  only occurs in atomic formulas  $[\neg]a = \eta$ , where  $a$  is a variable
- each atomic formula  $t_1 \tau t_2$  without occurrences of  $\eta$  is only occurring in conjunction with  $\bigwedge_{a \in \mathcal{V}(t_1) \cup \mathcal{V}(t_2)} a \neq \eta$  (we define  $\bigwedge_{a \in \emptyset} \tau(a, \mathbf{x})$  with  $\top$ ).

$K'_i$ , for  $i \in \mathbb{N}_{m_2}$  is constructed in the following way:

- Replace each  $K_i$  with basic type atomic formula of the form  $t_1 = t_2$ ,  $\mathcal{V}(t_1), \mathcal{V}(t_2) \neq \emptyset$ , with

$$K'_i \stackrel{\text{def}}{=} \left( t_1 = t_2 \wedge \bigwedge_{a \in \mathcal{V}(t_1) \cup \mathcal{V}(t_2)} (a \neq \eta) \wedge \mu_i \right) \vee \left( \bigvee_{a \in \mathcal{V}(t_1)} (a = \eta) \wedge \bigvee_{a \in \mathcal{V}(t_2)} (a = \eta) \wedge \mu_i \right).$$

- Replace each  $K_i$  with a negated basic type atomic formula  $t_1 \neq t_2$ ,  $\mathcal{V}(t_1), \mathcal{V}(t_2) \neq \emptyset$ , with

$$K'_i \stackrel{\text{def}}{=} \left( t_1 \neq t_2 \wedge \bigwedge_{a \in \mathcal{V}(t_1) \cup \mathcal{V}(t_2)} (a \neq \eta) \wedge \mu_i \right) \vee \left( \bigvee_{a \in \mathcal{V}(t_1)} (a = \eta) \wedge \bigwedge_{a \in \mathcal{V}(t_2)} (a \neq \eta) \wedge \mu_i \right) \vee \left( \bigvee_{a \in \mathcal{V}(t_2)} (a = \eta) \wedge \bigwedge_{a \in \mathcal{V}(t_1)} (a \neq \eta) \wedge \mu_i \right)$$

- Replace each  $K_i$  with occurrence of an equation  $[\neg](t_1 = \eta)$ ,  $\mathcal{V}(t_1) \neq \emptyset$ , with

$$K'_i \stackrel{\text{def}}{=} \bigvee_{a \in \mathcal{V}(t_1)} [\neg](a = \eta) \wedge \mu_i.$$

Treat the symmetrical case analogously.

- For other cases the constructions are analogous.

**Example 4.4.** Let  $\varphi$  be :

$$x_1 \neq \eta \wedge M_1 p_1 + x_1 \geq 5 \wedge x_1 + x_2 = \eta \wedge M_2 p_2 + x_2 > x_1$$

then we have for the basic type normal form

$$x_1 \neq \eta \wedge a_1 + x_1 \geq 5 \wedge a_1 = M_1 p_1 \wedge x_1 + x_2 = \eta \wedge a_2 + x_2 > x_1 \wedge a_2 = M_2 p_2$$

and for  $\varphi'$  from our lemma

$$x_1 \neq \eta \wedge (a_1 + x_1 \geq 5 \wedge a_1 \neq \eta \wedge x_1 \neq \eta) \wedge a_1 = M_1 p_1 \wedge (x_1 = \eta \vee x_2 = \eta) \wedge \\ ((a_2 + x_2 > x_1 \wedge a_2 \neq \eta \wedge x_2 \neq \eta \wedge x_1 \neq \eta) \vee ((a_2 = \eta \vee x_2 = \eta) \wedge x_1 = \eta)) \wedge a_2 = M_2 p_2$$

*Proof of Theorem 2.1.* Let there be a  $\mathcal{L}_q$ -formula  $\exists x \varphi(x, m_1, \dots, m_n)$  with  $\varphi$  of form (1). We know with our lemma that  $\varphi$  is equivalent to  $\varphi'$ . We transform  $\varphi'$  in disjunctive normal form  $\bigvee_{i=1}^k L_i \vee \varphi_2$ , where, using notations of the lemma, the conjunctions  $L_i, i \in \mathbb{N}_k$  unlike conjunctions in  $\varphi_2$  contain a  $[\neg](t_{1,j} \tau_i t_{2,j}) \wedge \mu_j$  with  $\mathcal{V}(t_{1,j}), \mathcal{V}(t_{2,j}) \neq \emptyset, j \in \mathbb{N}_{m_2}$  \*. This allows us to iteratively eliminate quantifiers in the conjuncts and then to build the disjunction of our quantifier free results.

A  $L_j$  with  $j \in \mathbb{N}_k$  has the following form:

$$\bigwedge_{i \in M_1^j} (t_{1,i} \tau_i t_{2,i}) \wedge \mu_i \wedge \bigwedge_{i \in M_2^j} \neg(t_{1,i} \rho_i t_{2,i}) \wedge \mu_i \wedge \bigwedge_{k \in \{1,2\}} \bigwedge_{i \in M_k^j} \left( \bigwedge_{a \in \mathcal{V}(t_{1,i}) \cup \mathcal{V}(t_{2,i})} a \neq \eta \right) \wedge \xi_j, \quad (1)$$

where  $M_1^j \subseteq \mathbb{N}_{m_1}, M_2^j \subseteq \mathbb{N}_{m_2} \setminus \mathbb{N}_{m_1}$ , and  $\xi_j$  is a conjunction of formulas of the form  $[\neg]a = \eta$ , with  $a$  being a variable, as well as  $[\neg]a_{k,l} = \eta \wedge a_{k,l} = M_{k,l} p_{k,l}$ . We replace now in  $\varphi'$  all occurrences of  $x \neq \eta$  with  $\top$ , and conjuncts  $L_i$  with occurrences of the atomic formula  $x = \eta$  in  $\xi_j$  with  $\perp$ . After that we get a DNF  $L'_1 \vee \dots \vee L'_k \vee \varphi_2^\dagger$ . We know with the quantifier free formula  $\psi_j$ , which we win as a result of the quantifier elimination of the  $\mathcal{L}$ -structure for the subformula

$$\exists x \left( \bigwedge_{i \in M_1^j} (t_{1,i} \tau_i t_{2,i}) \wedge \bigwedge_{i \in M_2^j} \neg(t_{1,i} \rho_i t_{2,i}) \right), \quad (2)$$

we have the following:

$$M_j \stackrel{\text{def}}{=} \psi_j \wedge \bigwedge_{i \in M_1^j} \left( \bigwedge_{a \in \mathcal{V}(t_{1,i}) \cup \mathcal{V}(t_{2,i})} a \neq \eta \wedge \mu_i \right) \wedge \bigwedge_{i \in M_2^j} \left( \bigwedge_{a \in \mathcal{V}(s_{1,i}) \cup \mathcal{V}(s_{2,i})} a \neq \eta \wedge \mu_i \right) \wedge \xi_j'$$

---

\*Note at this point, that  $x$  occurs in the atomic formulas of  $\varphi_2$  only in the forms  $x = \eta$  and  $x \neq \eta$ .

†Note that  $x$  occurs now only in  $L'_i$  with  $i \in \mathbb{N}_{k'}$ .

is with the help of our lemma equivalent to  $\exists x L'_j$ .

It is now easy to show, that the quantifier free formula

$$M_1(\mathbf{m}) \vee \cdots \vee M_k(\mathbf{m}) \vee \varphi'_2 \vee \varphi(\eta, \mathbf{m}) \quad (3)$$

is equivalent to  $\exists x L_1 \vee \cdots \vee L_k \vee \varphi_2$  and therefore to (1).  $\square$

For specific structures there are more efficient quantifier eliminations available, like shown in Straßer [2006].

## 4.2 Quantifiers binding Variables of Queue Type

### 4.2.1 Important Concepts

Let there be a formula  $\exists p \varphi$  without occurrences of ladd, radd, where  $\varphi$  is of the following form:

$$\bigwedge_{i \in I_1} q = z_i \wedge \bigwedge_{i \in I_2} q = L_i l_i \wedge \bigwedge_{i \in I_3} M_i q = m_i \wedge \bigwedge_{i \in I_4} N_i q = O_i q \wedge \bigwedge_{i \in I_5} \neg(q = x_i) \wedge \bigwedge_{i \in I_6} \neg(P_i q = y_i) \wedge \bigwedge_{i \in I_7} \neg(Q_i q = R_i q) \wedge \bigwedge_{\tau \in R} \left( \bigwedge_{i \in I_\tau} t_{1,i} \tau t_{2,i} \wedge \bigwedge_{i \in I_\tau^-} \neg(t_{1,i} \tau t_{2,i}) \right) \wedge \chi, \quad (4)$$

where for all  $\tau \in R$  and  $i \in I_\tau \cup I_\tau^-$  it is valid that:  $q \in \text{Var}(t_{1,i}) \cup \text{Var}(t_{2,i})$  and  $\chi$  represents a conjunction of atomic formulas without occurrences of variable  $p$ . For  $i \in \{3, 4, 5, 6, 7\}$  the index set  $I_i^q$  represents all indexes in  $I_i$  which represent queue type atomic formulas. With  $At_J$  we denote the set of all atomic formulas represented with indexes from the index set  $J$ . In the term  $Mq$  we call  $M = \text{head}^h \text{ltail}^l \text{rtail}^r$ ,  $\text{head} \in \{\text{lhead}, \text{rhead}\}$  the *prefix* of  $p$ . Further we define the following concepts:

**Definition 4.5.** Let  $\text{Prefix}_\varphi$  be the set of all prefixes of a formula  $\varphi$  of the form (4). If it is obvious with what formula we deal, we will often skip the index  $\varphi$ . We define the following subsets of prefixes:

$$\text{Prefix}_{\text{head}} \stackrel{\text{def}}{=} \{K \in \text{Prefix} \mid K = \text{head}(\text{ltail})^l (\text{rtail})^r, l, r \in \mathbb{N}_0\}, \quad \text{head} \in \{\text{lhead}, \text{rhead}\}$$

Let  $K$  be the prefix  $(\text{head})^h (\text{ltail})^l (\text{rtail})^r, h \in \{0, 1\}, l, r \in \mathbb{N}_0$ . We need the following concepts:

$$\text{lefts}(K) = \begin{cases} r + 1 & K \in \text{Prefix}_{\text{lhead}} \\ r & \text{else} \end{cases}, \quad \text{rights}(K) = \begin{cases} l + 1 & K \in \text{Prefix}_{\text{rhead}} \\ l & \text{else} \end{cases}$$

The length of a prefix is simply the sum of the above defined notions:

$$|\cdot| : \text{Prefix} \rightarrow \mathbb{N} \text{ mit } |M| = \text{lefts}(M) + \text{rights}(M)$$

The *left area* [*right area*] of a formula  $\varphi$  of the form (4) is defined in the following way:

$$LA(\varphi) [RA(\varphi)] = \begin{cases} \infty & I_1 \cup I_2 \cup I_{3_2} \neq \emptyset \\ \max(\{0\} \cup \{\text{lefts}(K) [\text{rights}(K)] : K \in \text{Prefix}_\varphi\}) & \text{else} \end{cases}$$

Atomic formulas represented by indexes in  $I_{pat} \stackrel{\text{def}}{=} \{i \in I_4^p \cup I_7^p \mid |N_i| = |O_i| \text{ resp. } |Q_i| = |R_i|\}$ , describe what we will call *pattern*. We will define *pattern* as a set of positions depending on a given length  $l$ :

$$pat(i)_l \stackrel{\text{def}}{=} \begin{cases} (\min(\{\text{lefts}(N_i), \text{lefts}(O_i)\}), \dots, \max(\{\text{lefts}(N_i), \text{lefts}(O_i)\})) & , \text{ if } \psi_1 \\ (\text{lefts}(N_i), \dots, l - \text{rights}(N_i)) & , \text{ if } \psi_2 \\ \lambda & \text{else} \end{cases}$$

$$\text{where } \psi_1 = l > |N_i| \wedge \max_{\text{lefts}} \leq \min_{\text{rights}} \text{ and } \psi_2 = l > |N_i| \wedge \max_{\text{lefts}} > \min_{\text{rights}}$$

$$\max_{\text{lefts}} \stackrel{\text{def}}{=} \max(\{\text{lefts}(N_i), \text{lefts}(O_i)\}) \text{ and } \min_{\text{rights}} \stackrel{\text{def}}{=} \min(\{\text{rights}(N_i), \text{rights}(O_i)\}).$$

## 4.2.2 Investigations on the Length of satisfying Queues

**Theorem 4.6.** *Let  $\varphi$  be of the form (4) with  $I_1 \cup I_2 \cup I_3^q \cup I_5 \cup I_6^q = \emptyset$ . If there is no  $p \in \text{Sat}(\varphi)$  with  $\text{length}(p) = LA(\varphi) + RA(\varphi) + t$ , with  $t \in \{0, \dots, \tau - 1\}$ , where  $\tau \stackrel{\text{def}}{=} \text{lcm}(\{l_i \mid i \in I_{pat}\})^*$ , then there is also no  $q \in \text{Sat}(\varphi)$  with  $\text{length}(q) \in \{\text{length}(p) + m \cdot \tau\} \mid m \in \mathbb{N}\}$ , where  $l_i \stackrel{\text{def}}{=} |\text{lefts}(N_i) - \text{lefts}(O_i)|$ , for  $i \in I_{pat}$ . That means:*

$$\text{Sat}_{\leq LA(\varphi) + RA(\varphi) + \tau - 1}(\varphi) \cap \text{Sat}_{\geq LA(\varphi) + RA(\varphi)}(\varphi) = \emptyset \text{ then } \text{Sat}_{\geq LA(\varphi) + RA(\varphi) + \tau}(\varphi) = \emptyset.$$

**Theorem 4.7.** *Let  $\varphi$  be of the form (4) with  $I_1 \cup I_2 \cup I_3^q = \emptyset$ . We have for the vector  $\mathbf{m}$ :*

$$\begin{aligned} \exists p (\varphi(p, \mathbf{m})) &\Leftrightarrow \exists p (\varphi_1(p, \mathbf{m}) \vee \varphi_2(p, \mathbf{m})), \text{ with} \\ \varphi_1(p, \mathbf{m}) &\stackrel{\text{def}}{=} \varphi(p, \mathbf{m}) \wedge \text{ltail}^{LA(\varphi) + RA(\varphi) + \tau} p = \varepsilon \\ \varphi_2(p, \mathbf{m}) &\stackrel{\text{def}}{=} \psi(p, \mathbf{m}) \wedge \text{ltail}^{LA(\varphi) + RA(\varphi) + \tau} p \neq \varepsilon \end{aligned}$$

where  $\tau \stackrel{\text{def}}{=} \text{kgV}(\{\text{lefts}(M) - \text{lefts}(N) \mid Mp = Np \in \text{At}_{I_{pat}}\})$ . The formula  $\psi$  denotes the formula  $\varphi$  without conjuncts over the index sets  $I_5 \cup I_6^q$ .

## 4.2.3 Quantifier Elimination

**Theorem 4.8.** *Let  $\varphi$  be of the form (4) with  $I_1 \cup I_2 \cup I_3^q \neq \emptyset$ . There is a quantifier elimination procedure for  $\exists p \varphi(p)$ , with  $p$  being variable of type queue.*

---

\*We define  $\text{lcm}(\emptyset) = 1$

*Proof.* Case  $I_1 \neq \emptyset$ : The formula  $\exists p \varphi(p)$  is equivalent to  $\varphi(z_1/p)^*$ . Let in the following  $I_1$  be equal to  $\emptyset$ . Analogously we show case  $I_2 \neq \emptyset$ .

Case  $I_3^q \neq \emptyset$ : Let  $j \in \{i \in I_3^q \mid |M_i| = \min(\{|M_j| \mid j \in I_3^q\})\}$ . With  $l = \text{lefts}(M_j), r = \text{rights}(M_j)$  let

$$\psi \stackrel{\text{def}}{=} \exists x_1 \dots \exists x_l \exists y_1 \dots \exists y_r \varphi(|x_1 \dots x_l p_j y_1 \dots y_r|/p), \quad \rho_k \stackrel{\text{def}}{=} \exists x_1 \dots \exists x_k \varphi(|x_1 \dots x_k|/p).$$

Then  $\exists p \varphi(p)$  is equivalent to  $\bigvee_{k=1}^{|M_j|-1} \rho_k \vee \psi \vee \varphi(\varepsilon/p)$ . □

**Theorem 4.9.** *Let  $\varphi$  be of the form (4) with  $I_5 \cup I_6^q = \emptyset$ . There is a quantifier elimination procedure for  $\exists p \varphi(p)$ , with  $p$  being a variable of type queue.*

*Proof.* Let  $\text{Satlengths} \stackrel{\text{def}}{=} \{0, \dots, \text{LA}(\varphi) + \text{RA}(\varphi) + \tau - 1\}$ , where  $\tau$  is defined like in theorem 4.6. Let  $E_i$  be a sequence of quantifiers in such a way that  $E_i \stackrel{\text{def}}{=} \exists p_1 \dots \exists p_i$ . Our quantifier (of queue type) free formula has the form  $\bigvee_{i \in \text{Satlengths}} E_i \varphi(|p_1 \dots p_i|/p)$ . □

**Corollary 4.10.** *With the notation of proof of (4.9), we have the following quantifier elimination procedure with help of lemma 4.7:*

$$\exists p \varphi(p) \Leftrightarrow \bigvee_{i \in \text{Satlengths}(\varphi_1)} E_i \varphi_1(|p_1 \dots p_i|/p) \vee \bigvee_{i \in \text{Satlengths}(\psi)} E_i \varphi_2(|p_1 \dots p_i|/p)$$

Together with our quantifier elimination procedure for quantifiers of basic type full quantifier elimination is reached for queue extensions of structures which permit a quantifier elimination procedure.

## 5 Preservation of Some Important Properties under Extension by Queues

Throughout this chapter let  $\Sigma$  be a class of structures for a given first order language  $\mathcal{L}$ . With  $\Sigma'$  we denote the corresponding class of structures for the queue extension  $\mathcal{L}_q$ .

**Lemma 5.1.** *Let  $\varphi$  be an arbitrary  $\mathcal{L}_q$ -sentence. There is an equivalent sentence  $\varphi'$ , where each atomic subformula is of basic type.*

*Proof.* After the quantifier elimination of quantifiers of type queue we can replace each occurrence of a queue type atomic formula  $[\neg]p = q$ , which has to have the form  $|p_1 \dots p_n| = |q_1 \dots q_m|$ , with  $[\neg] \bigwedge_{i=1}^n p_i = q_i$ , if  $n = m$ , else with  $\perp$ . We apply a similar procedure to queue type atomic formulas with prefixes  $[\neg]Lp = Mq$ . □

---

\*We denote with  $a/b$  the substitution of  $b$  with  $a$ .

**Theorem 5.2.** *If a class of structures  $\Sigma$  is complete in  $\mathcal{L}$ , then also  $\Sigma'$  is complete in  $\mathcal{L}_q$ .*

*Proof.* Let  $\varphi$  be  $\mathcal{L}_q$ -sentence in the form of lemma (5.1). We need to show our statement only for conjunctions of atomic formulas. We know  $\varphi$  is equivalent to  $\varphi'$  from lemma (4.3). Let  $\varphi''$  be the DNF of  $\varphi'$ . Therefore a clause  $\psi$  in  $\varphi''$  has the form  $\psi = \psi_1 \wedge \psi_2 \wedge \psi_3$ , where  $\psi_1$  contains only atomic formulas without occurrences of  $\eta$ ,  $\psi_2$  consists only of  $a \neq \eta$  for all variables  $a$  occurring in  $\psi_1$ , and  $\psi_3$  contains atomic formulas of the type  $a = \eta$  and  $a \neq \eta$  for variables  $a$  not occurring in  $\psi_1$ . Therefore  $\psi_1$  is also a  $\mathcal{L}$ -formula and we know with lemma (4.2):  $\Sigma \models \psi_1$  if and only if  $\Sigma' \models \psi_1 \wedge \psi_2$ . For formula  $\psi_3$  it is obvious that  $\Sigma' \models \psi_3$  or  $\Sigma' \models \neg\psi_3$ . As we also know that  $\mathcal{V}(\psi_3) \cap \mathcal{V}(\psi_1 \wedge \psi_2) = \emptyset$  we proved our statement.  $\square$

**Theorem 5.3.** *If a class of structures  $\Sigma$  is substructure [model] complete in  $\mathcal{L}$ , then also  $\Sigma'$  is substructure [model] complete in  $\mathcal{L}_q$ .*

*Proof.* Shown in a similar way.  $\square$

**Theorem 5.4.** *If a class of structures  $\Sigma$  is decidable in  $\mathcal{L}$ , then  $\Sigma'$  is decidable in  $\mathcal{L}_q$ .*

## 6 Outlook and Applications

The efficiency of the generic quantifier elimination procedure for quantifiers binding variables of queue type presented above can be improved drastically by taking into account minimal and maximal lengths of queue variables. This is done under consideration of terms like  $Mq = \varepsilon$  for upper bounds and  $Mq \neq \varepsilon$  for lower bounds, as well as  $Mq = Lp$ , where  $q$  or  $p$  also appear at least in one of the two atomic formula types from above. A quantifier elimination with usage of test terms is developed in Straßer [2006].

Quantifier eliminations for various first order theories based upon the results of this paper are currently implemented for the REDLOG system (compare Dolzmann and Sturm [1997]). Straßer [2006] demonstrates an automata accepting queue languages, which are defined as  $Sat(\varphi)$  with  $\mathcal{V}(\varphi) = p$  and  $p$  of type queue.

For further investigations it would be interesting to generalize the presented results for multi-sorted languages, as well as to extend the signature with other functions on queues, like for example the length function, and so to combine my results with the ones presented in [Zhang et al., 2005].

As queues are a widely used data structure in programming languages the framework is set for applications in manifold areas, where it is important to prove and decide statements about domains containing queues. In the context of information flow control (IFC) Snelting [2005] points out, that in the process of solving path conditions for dependency graphs it would be desirable to *apply quantifier elimination not just to arithmetic structures*. My results are answering his *”Challenge to the quantifier community”*, which says: *”We need quantifier elimination for combinations of arithmetic with free algebras or with equationally specified algebras.”* For the sake of software safety and security the *Common Criteria* Com [1999] requires inter alia IFC as a security technique, the goal of which is to guarantee that confidential data cannot leak to public variables and that critical computations cannot be

manipulated from outside. For program analysis IFC uses a combination of *dependency graphs* (DG) and *constraint solving* (compare G. Snelting [1996], G. Snelting [2006]). The nodes of a DG are program statements and expressions, the edges represent data and control dependencies. A data dependency edge  $x \rightarrow y$  means that statement  $x$  assigns a variable which is used in statement  $y$ . A control dependency edge  $x \rightarrow y$  means that the mere execution of  $y$  depends on the value of the expression  $x$ . Paths  $x \rightarrow^* y$  represent possible information flows from  $x$  to  $y$ . Further, *path conditions* are introduced, which represent necessary conditions for information flows. In a very simplified way the following definition answers our purposes:

$$PC(x, y) \stackrel{\text{def}}{=} \bigvee_{P \text{ path } x \rightarrow^* y} \bigwedge_{u \text{ node in } P} E(u),$$

where  $E(u)$  is a necessary condition for the execution of  $u$ , and is defined in a similar way. Quantifier elimination comes into play, as variables in path conditions are existentially quantified. If we take for example the following program fragment

$$\begin{aligned} (1) \quad & \mathbf{a[i+1]} = \mathbf{x}; \\ (2) \quad & \mathbf{if (i>10)} \quad \quad \quad PC(1, 3) \equiv \exists i, j (i > 10) \wedge (i + 3 = 2j - 42) \\ (3) \quad & \quad \mathbf{y} = \mathbf{a[2*j-42]}; \end{aligned}$$

Eliminating e.g.  $i$  yields the condition  $2j > 55$ .

The results of the present investigations are of great help in this context, as arithmetic conditions are often combined with data structures, as for example in

$$\begin{aligned} (1) \quad & \mathbf{a[i+3]} = \mathbf{x}; \\ (2) \quad & \mathbf{s} = \mathbf{push(s, a[2*j-42])}; \\ (3) \quad & \mathbf{if (i>10)} \\ (4) \quad & \quad \mathbf{y} = \mathbf{top(s)}; \end{aligned}$$

Here we need to perform quantifier elimination on

$$PC(1, 4) \equiv \exists i, j, s ((i > 10) \wedge (top(push(s, a[2j - 42])) = a[i + 3])).$$

As another example we take a look at a scheduling problem. We have  $m$  input queues  $q_i$ ,  $i \in \mathbb{N}_m$  of tasks. Let  $\varphi_i$ ,  $i \in \mathbb{N}_m$  denote our pre-knowledge of the task queues (e.g. periodicity). Tasks are modeled by pairs  $(a, n)$  where  $a \in M$  and  $n \in \mathbb{N}$  models the running time. That means the universe of our element structure is given with  $M \times \mathbb{N}$ . We introduce a relation  $\tau$  between tasks, where  $a\tau b$  means, that task  $b$  is not allowed to directly follow task  $a$ . Also we want two tasks in a row not to take more time then  $z$ . Therefore we introduce the binary addition function  $\oplus$  with  $(a, n_a) \oplus (b, n_b) = (a, n_a + n_b)$  and the total order  $\leq$  with  $(a, n_a) \leq (b, n_b)$  if and only if  $n_a \leq n_b$ . We formulate the following requirement

$$\begin{aligned} \psi(q, z) \stackrel{\text{def}}{=} q = \varepsilon \vee (\text{ltail}(q) = \varepsilon \wedge \text{rhead}(q) \leq z) \vee \\ (\text{rhead}(q) \oplus \text{rhead}(\text{ltail}(q)) \leq z \wedge \neg(\text{rhead}(q) \tau \text{rhead}(\text{ltail}(q)))). \end{aligned}$$

In our scheduling problem we are allowed to take any of the *tops* of the input queues  $q_i$ ,  $i \in N_m$  and *push* it on our output queue  $q_o$ . This continues until all input queues are empty. At each point in the procedure  $\psi(q_o, z)$  has to be valid. The question is: can we treat all possible input queues and what is the lower bound for the time threshold given by  $z$ . We give a solution to this problem with quantifier elimination in the following way:

$$\psi_i(q_o, q_1, \dots, q_m) \stackrel{\text{def}}{=} \left( \bigwedge_{j \in N_m} q_j = \varepsilon \wedge \psi_0(p_o, z) \right) \vee \left( \bigvee_{j \in N_m} \text{lhead}(q_o) = \text{rhead}(q_j) \wedge q_j \neq \varepsilon \wedge \psi_{i-1}(\text{rtail}(q_o), q_1, \dots, \text{ltail}(q_j), \dots, q_m) \wedge \psi_0(q_o, z) \right)$$

We get a solution to our problem dependent on the time  $\pi_2(z)$  of two tasks in a row with the quantifier elimination of

$$\exists q_o \forall q_1 \dots \forall q_m \left( \psi_{\text{maxlength}}(q_o, q_1, \dots, q_m, z) \wedge \bigwedge_{i \in N_m} \varphi_i(q_i) \right),$$

where *maxlength* is a maximal value for the sum of the lengths of the input queues.

## References

- Common criteria project sponsoring organizations. common criteria for information technology evaluation. ISO/IEC 15408, 1999.
- Michael Benedikt, Leonid Libkin, Thomas Schwentick, and Luc Segoufin. Definable relations and first-order query languages over strings. *J. ACM*, 50(5):694–751, 2003. ISSN 0004-5411.
- Nikolaj Skallerud Bjørner. *Integrating decision procedures for temporal verification*. PhD thesis, 1999. Adviser-Zohar Manna.
- Andreas Dolzmann and Thomas Sturm. REDLOG: Computer algebra meets computer logic. *SIGSAM Bulletin (ACM Special Interest Group on Symbolic and Algebraic Manipulation)*, 31(2):2–9, 1997. URL [citeseer.ist.psu.edu/dolzmann96redlog.html](http://citeseer.ist.psu.edu/dolzmann96redlog.html).
- J. Krinke G. Snelting, T. Robschink. Efficient path conditions in dependene graphs for software safety analysis. To appear in *ACM Transactions on Software Engineering and Methodology*.
- Tatiana Rybina and Andrei Voronkov. A decision procedure for term algebras with queues. *ACM Trans. Comput. Logic*, 2(2):155–181, 2001. ISSN 1529-3785.

- Gregor Snelting. Combining slicing and constraint solving for validation of measurement of software. *Proc. Static Analysis Symposium*, 1145 of LNCS:332–348, 1996.
- Gregor Snelting. Quantifier elimination and information flow control for software security. In *Algorithmic Algebra and Logic 2005*, 2005.
- Christian Straßer. Quantifier elimination for queues, technical report, unpublished. 2006.
- T. Zhang, H. Sipma, and Z. Manna. Decision procedures for queues with integer constraints, 2005. URL <http://theory.stanford.edu/~tingz/papers/fsttcs05.html>.