Quantifier elimination over finite strings with monotone functions

Definitions: A string is a finite structure over the domain \{1, ..., n\} with constants \(n \geq 1\), totally ordered by a binary relation <, together with a fixed number of monadic predicates. The initial signature will be augmented by an infinite family of additional monotone (increasing) functions, i.e. those satisfying \(x_1 < x_2 \Rightarrow f(x_1) \leq f(x_2)\).

Some are defined by using minimum and maximum operators applied to the parameterized set \(\varphi(y) = \{x : \varphi(x, y)\}\) defined by a first-order formula \(\varphi\), where whenever we take the minimum or maximum of the empty set, we return the maximum or minimum element of the domain respectively, i.e. \(\min \{} = n\) and \(\max \{} = 1\). These both have first-order definitions.

\[
\begin{align*}
\min \{x : \varphi(x, y)\} & = z & \Leftrightarrow & \forall x [\varphi(x, y) \rightarrow x \geq z] \land \exists x \varphi(x, y) \rightarrow \varphi(z, y) \land \forall x \varphi(x, y) \rightarrow z = n \\
\max \{x : \varphi(x, y)\} & = z & \Leftrightarrow & \forall x [\varphi(x, y) \rightarrow x \leq z] \land \exists x \varphi(x, y) \rightarrow \varphi(z, y) \land \forall x \varphi(x, y) \rightarrow z = 1
\end{align*}
\]

E.g. from these, it is easy to define the successor and predecessor functions \(s\) and \(p\) respectively, by:

\[
\begin{align*}
s(y) & = \min \{x : x > y\} & \text{That is, } s(y) = y + 1 & \text{for } y < n \ (n \text{ otherwise}). \\
p(y) & = \max \{x : x < y\} & \text{That is, } p(y) = y - 1 & \text{for } y > 1 \ (1 \text{ otherwise}).
\end{align*}
\]

Notice that both are monotone, because successor tops out at \(n\) and predecessor bottoms out at \(1\).

Other monotone increasing functions will be derived by taking any function \(F\) and applying:

\[
\begin{align*}
F_{\max}(y) & = \max \{x : F(x) \leq y\} \\
F_{\min}(y) & = \min \{x : F(x) \geq y\}
\end{align*}
\]

E.g. \(s_{\min}(y) = p(y)\) and \(p_{\max}(y) = s(y)\), \(s_{\max}(y) = p(y)\) for \(y < n \ (n \text{ o.t.})\) and \(p_{\min}(y) = s(y)\) for \(y > 1 \ (1 \text{ o.t.})\).

Another application is the Skolem function returning the nearest element satisfying a formula \(\varphi(x)\). Let \(\gamma_\varphi\) and \(\lambda_\varphi\) be the monotone (and idempotent) upward and downward Skolem functions that return respectively the greatest element \(\leq y\) (1 otherwise) or least element \(\geq y\) (\(n\) otherwise) satisfying \(\varphi\). I.e.

\[
\begin{align*}
\gamma_\varphi & = F_{\max} \quad \text{(except for } n) \quad \text{where } F(x) = x \text{ if } \varphi(x) \ (n \text{ otherwise}) \\
\lambda_\varphi & = F_{\min} \quad \text{(except for } 1) \quad \text{where } F(x) = x \text{ if } \varphi(x) \ (1 \text{ otherwise})
\end{align*}
\]

We will see that the incorrect values for \(\gamma_\varphi(n)\) or \(\lambda_\varphi(1)\) are never used in our construction.

Theorem: Every first-order formula over strings is equivalent to a Boolean combination of quantifier-free formulas in a vocabulary expanded by linear-time computable function symbols.

Proof: Proceed by a quantifier elimination induction on the structure of formulas, where the only nontrivial case is application of an existential quantifier \(\exists x \theta(x, \bar{y})\) to a formula in which negations have been pushed all the way down, replacing instances of \(<, \geq, \neq\) by the corresponding disjunctions over \(<, \geq\), and =. The key observation is that every term involving \(x\) (or any other variable) must be of the form \(F(x)\) where \(F\) is monotone, because all the functions are monadic and monotonicity is preserved under composition. In this case, \(F_{\max}\) and \(F_{\min}\) serve as quasi-inverses, allowing us to solve for \(x\) in any equality or inequality involving \(F(x)\) by moving \(F\) to the other side,
denoted by any term \( t \) (not containing \( x \)):

\[
F(x) \leq t \quad \iff \quad x \leq F_{\text{max}}(t) \land F(1) \leq t \\
F(x) \geq t \quad \iff \quad x \geq F_{\text{min}}(t) \land F(n) \geq t
\]

Use the transformations: \( F(x) < t \iff F(x) \geq t; F(x) > t \iff F(x) \leq t; \) and \( F(x) = t \iff t \leq F(x) \leq t. \)

Next, put \( \theta \) in disjunctive normal form and distribute the existential over disjunctions, factoring out any atomic formulas that don’t involve \( x \) from within the scope of the quantifier to obtain \( \exists y \psi(x, y) \) where \( \psi \) is a conjunction of atomic formulas, each containing \( x \). Collect all atomic formulas which depend only on \( x \) (including any equations which involve \( x \) on both sides), and call their conjunction \( \varphi(x) \). If there is an equality remaining it must be of the form \( x = \tau(y) \) for \( y \) in \( \bar{y} \), and consequently \( \exists x \psi \) is equivalent to \( \psi[x \leftarrow \tau(y)] \) in which all occurrences of \( x \) have been replaced by the term \( \tau(y) \). Otherwise, all remaining formulas are inequalities of the form \( x > \alpha(y) \) for \( 1 \leq i \leq k \) and \( x < \beta(y) \) for \( 1 \leq j \leq l \) where each of the terms \( \alpha_i \) and \( \beta_j \) involve some variable \( y \) among \( \bar{y} \) (not necessarily the same one). Rewrite these as \( \alpha_1, \ldots, \alpha_k < x < \beta_1, \ldots, \beta_l \) in order to see that they are equivalent to the disjunction over all \( i \) and \( j \) of the conjunction over all \( i' \) and \( j' \) of \( \alpha_i \leq \alpha_i < x < \beta_j \leq \beta_j \). So \( x \) is sandwiched between two terms \( \alpha_i = \alpha < x < \beta = \beta_j \) (let \( \alpha = 1 \) if \( k = 0 \) and \( \beta = \) \( n \) if \( l = 0 \)). To express \( \exists x \psi \) we simply assert \( \alpha < \gamma_\varphi(p(\beta)) \) or equivalently \( \lambda_\varphi(s(\alpha)) < \beta \) (this is where we need successor and predecessor, and note that \( \gamma_\varphi(n) \) or \( \lambda_\varphi(1) \) are never used).

To see each newly defined monotone function is computable in linear-time, it suffices to show by induction that \( F_{\text{max}} \) and \( F_{\text{min}} \) can be computed in linear-time, since clearly the values for predecessor and successor can be tabulated outright in linear-time. By induction hypothesis assume that a monotone \( F \) has been tabulated in linear-time. Starting from the beginning for \( F_{\text{max}} \) and the end for \( F_{\text{min}} \) respectively, it is a simple matter to assign all the values for them in a loop:

\[
F_{\text{max}}: \quad \text{For } y = 1, \text{ let } F_{\text{max}}(1) = \max \{x : F(x) = 1\}, \text{ starting the search from } x = 1 \text{ and ending when } F(x) > 1. \text{ For } y = 2 \text{ upto } n \text{ compute } F_{\text{max}}(y) = \max \{x : F(x) \leq y\} \text{ starting from } x = F_{\text{max}}(y - 1) \text{ and ending when } F(x) > y.
\]

\[
F_{\text{min}}: \quad \text{For } y = n, \text{ let } F_{\text{min}}(n) = \min \{x : F(x) = n\}, \text{ starting the search from } x = n \text{ and ending when } F(x) < n. \text{ For } y = n - 1 \text{ downto } 1 \text{ compute } F_{\text{min}}(y) = \min \{x : F(x) \geq y\} \text{ starting from } x = F_{\text{min}}(y + 1) \text{ and ending when } F(x) < y.
\]

The total number of \( x \) values searched over all \( y \) values is linear because of monotonicity. [Therefore a minor modification might need to be made in computing the Skolem functions.]

**Corollary:** After a linear-time preprocessing stage; every first-order formula can be evaluated in constant-time given any particular assignment of its free variables.

**Proof:** Convert to a quantifier-free formula \( \psi(\bar{y}) \), and after computing tables for all the functions needed in linear-time, simply plug in values for the free variables and evaluate it in constant-time.

P.S. Extending this to modular counting quantifiers is even easier, since we only need to add monadic truth predicates.