Quantifier Elimination in Ordered Domains *

Douglas R. Smith

Kestrel Institute
3260 Hillview Avenue
Palo Alto, CA 94304 USA
smith@kestrel.edu
30 April 2019

Abstract. Quantifier elimination is the process of transforming a quantified formula into an equivalent quantifier-free formula. Quantifier elimination rules serve to optimize expressions that naturally arise during the specification, design, and analysis of programs. We present general rules for eliminating quantifiers over ordered domains and illustrate them with a variety of examples. A motivating insight is that continuity conditions provide a unifying abstraction that subsumes quantifier elimination rules over ordered domains.

Keywords: quantifier elimination · program derivation · program verification · partial order · Scott-continuous function

1 Introduction

Given an unquantified formula ϕ over theory T and a variable y, the classical task of Quantifier Elimination (QE) is to transform the quantified formula $\exists y.\phi$ into an equivalent unquantified formula ϕ'. The equivalence $\exists y.\phi \equiv \phi'$ means that both sides have the same set of solutions over the remaining variables. Eliminating a universal quantifier is treated by reducing it to the existential case.

Historically, quantifier elimination was developed to prove the decidability of theories, such as the first-order theory of real closed fields, or the first-order theory of the natural numbers with addition (Pressburger Arithmetic). The general strategy is to show that each quantified expression in the theory is equivalent to an unquantified expression, then to show that the unquantified expressions are decidable. Our purpose is different: to define high-level general QE rules that are suitable for implementation in a deductive framework. In particular, our QE rules are intended to help automatically simplify expressions that arise during software design and analysis.

The concept of Scott-continuous functions [9] provides a general setting for quantifier elimination over ordered domains. In this setting the notion of existential/universal quantification is generalized to maximization/minimization in a partial order or semilattice. QE rules then produce max/min-free expressions for a given extremized expression.

The main results of this paper are:

1. A general statement of quantifier elimination rules in an order-theoretic setting, together with special cases of the general rules that are easier to apply.
2. Rules for transforming a quantified expression into weaker and stronger expressions.
3. A range of worked examples to show applicability of the QE rules.

* This work has been sponsored in part by ONR, NSF, DARPA, and by the US Dept. of Defense.

2 Motivating Examples

Quantified expressions play a natural role in specifying many design and analysis problems. Quantifier elimination then plays a key role in formal reasoning about those problems. This section lists a range of examples to illustrate the techniques of this paper. Section 5 works through the examples using the QE rules that we introduce.

Eliminating Variables in Linear Systems: To eliminate variable x from a set of linear inequalities L, we partition L into those that are monotone in x and those that are antitone in x. Simplifying somewhat for the sake of focus, the result is a reformulation of the solvability of L as a quantified expression of the form

$$\exists x.\ lb \leq x\ \wedge\ x \leq ub$$

for unquantified expressions lb and ub that contain no occurrences of x. Eliminating the quantifier in this case results in the equivalent expression

$$lb \leq ub$$

which has the same set of solutions in the remaining variables as L.

Predicate Transformers: Dijkstra [2] popularized the use of predicate transformers to formalize the semantics of programming constructs and developed their use for program construction. The weakest precondition of a statement S is a predicate transformer mapping any postcondition R to the weakest precondition P such that if S is executed in a state st satisfying P, then the resulting state will satisfy R:

$$wp(st, S, R) \equiv \exists(st'{:}State)(Post_S(st, st') \wedge R(st'))$$

where $Post(st, S, st')$ gives the valid state transitions of S and R is a state predicate. The definition of wp for particular classes of statements can be calculated using quantifier elimination on the right-hand side (RHS).

In system construction settings, a game-like mechanism is needed to construct a feasible strategy. Suppose that the system (protagonist) wants to achieve a next state st' satisfying $R(st')$ by making an action $\alpha \in SysAct$, but the uncontrolled environment (antagonist) will simultaneously take some action $\beta \in EnvAct$. The semantics of the joint action τ is $Post_\tau(st, \alpha, \beta, st')$ which relates the pre-state st, actions α and β, and the post-state st'. The weakest precondition in this game-like setting is

$$wp(st, \tau, R)\ \equiv\ \exists(\alpha)(\alpha \in SysAct \wedge \forall(\beta)(\beta \in EnvAct \Longrightarrow \exists(st'{:}State)(Post_\tau(st, \alpha, \beta, st') \wedge R(st'))))$$

which requires three QE steps. The result defines the initial states from which the system can perform action α assured that no matter which action β the environment might take, the resulting state will satisfy R.

Algorithm Design: In the theory of backtrack algorithms, called global search theory [11, 12], pruning mechanisms are specified by a quantified formula:

$$\exists(z)(z \sqsubseteq \hat{r} \wedge O(x, z))$$

where $\hat{r}$ is a partial solution and z is a refinement of it to a complete solution, $O(x, z)$ asserts that z is a feasible solution with respect to input x. Elimination of the quantifier gives a test as to whether the current partial solution $\hat{r}$ can be extended to a complete feasible solution – if not, then it can be pruned from further consideration.

Suppose we are designing a global search algorithm for cost minimization. We would like to know a lower bound on the cost of any feasible solution that extends the current partial solution $\hat{r}$. This too can be specified by a quantified expression:

$$\min_{z \sqsubseteq \hat{r}\ \wedge\ O(x,z)} cost(x, z)$$

where $\hat{r}$ is a partial solution, z is a refinement of $\hat{r}$, and $O(x, z)$ asserts that z is a feasible solution with cost $cost(x, z)$. Elimination of the minimization quantifier gives a test as to whether the current partial assignment $\hat{r}$ can be extended to a complete minimal-cost solution – if not, then it can be pruned from further consideration. Note that this example illustrates the use of a non-Boolean quantifier.

Temporal Propagation in Scheduling: Scheduling problems are often treated as constraint satisfaction/optimization problems where a given set of tasks must be assigned start times and resources for their accomplishment subject to some constraints. A key step in calculating a scheduling program is deriving a constraint propagation algorithm, which propagates local decisions about start times and resources to other parts of a partial schedule to further constrain the allocation of time and resources to related tasks. In [13] we show how constraint propagation rules can be systematically generated from concrete constraints, and QE laws play a central role. The start time st of a task is often represented abstractly by a time window: $\langle earliestStartTime, latestStartTime\rangle$, abbreviated $\langle est, lst\rangle$. That is, if st_i is a variable representing the (concrete, unknown) start time of task i, then $est_i \leq st_i \leq lst_i$ represent bounds on feasible values for st_i.

A typical scheduling constraint expresses that task i must finish before task $i + 1$ can start:

$$st_i + duration_i \leq st_{i+1}$$

over tasks executing on a unit-capacity resource, such as a printer. This binary constraint gives rise to arc-consistency constraints on the abstract representation, such as

$$\forall st_1(est_1 \leq st_1 \leq lst_1 \implies \exists st_2(est_2 \leq st_2 \leq lst_2 \ \wedge \ st_1 + duration \leq st_2))$$

When QE is applied, the result is unquantified definite constraints [8], which can be propagated very efficiently via fixpoint iteration.

3 Basic Concepts

A partially ordered set (poset) $P = \langle P, \preceq\rangle$ is an *upward-directed set* if P is nonempty and every pair of elements of P has an upper bound. P is a *downward-directed set* if it is nonempty and every pair of elements has a lower bound. The term *directed set* may refer to either case. Directed sets are generalizations of sequences, linearly-ordered sets, and semilattices in that every finite subset Q of an upward-directed set P has a unique maximum element in Q (and dually, every finite subset Q of a downward-directed set P has a unique minimum element in Q). They have been an important concept in generalizing the notion of the limit of a convergent sequence from analysis.

The following proposition provides a common sufficient condition that a poset is directed.

Proposition 1. *Let $P = \langle P, \preceq\rangle$ be a poset and $D \subseteq P$. If D has a maximal element $\hat{d} \in D$, then D is an upward-directed set. Dually, if D has a minimal element $\check{d} \in D$, then D is a downward-directed set.*

Proof. Let $\hat{d} \in D$ be a maximal element of D. D is nonempty since $\hat{d} \in D$, and for any pair d_1, $d_2 \in D$ we have $\hat{d}$ as an upper bound, so D is an upward-directed set. Similar reasoning handles the dual case. □

A poset $P = \langle P, \leq\rangle$ is a *join semilattice* $\langle P, \sqcup, \leq\rangle$ if it has a binary least upper bound operator $\sqcup$, called *join*. P is a *meet semilattice* $\langle P, \sqcap, \leq\rangle$ if it has a binary greatest lower bound operator $\sqcap$, called *meet*. P is a *lattice* $\langle P, \sqcap, \sqcup, \leq\rangle$ if it has both a meet and join operator.

A *(semi)lattice quantifier* is the meet or join of an expression over the elements of a partially ordered set. We use the notation $\bigsqcup_{a\in A} F(a)$ and dually $\bigsqcap_{a\in A} F(a)$ to quantify the expression $F(a)$ over the

set A. Although F is treated here as a unary function, the intent is that it is an unquantified expression over variable a plus other variables. The usual logical quantifiers $\exists$ and $\forall$ are special cases of a lattice quantifier over an expression from the usual Boolean lattice $\langle Boolean, \wedge, \vee, \Rightarrow \rangle$:

$$\exists a{:}A.\ F(a) \ \equiv\ \bigvee_{a\in A}\ F(a)$$

and

$$\forall a{:}A.\ F(a) \ \equiv\ \bigwedge_{a\in A}\ F(a).$$

Meet quantifiers are monotone in their domain and join quantifiers are antitone in their domain, as summarized in the following Quantifier Change (QC) rules:

$D \subseteq E$ and $f{:}E \to \langle L, \sqcup, \sqcap, \leq \rangle$			
QC1	$\bigsqcup_{a\in D}\ f(a) \leq \bigsqcup_{a\in E}\ f(a)$	QC2	$\bigsqcap_{a\in D}\ f(a) \geq \bigsqcap_{a\in E}\ f(a)$

Let $P = \langle P, \preceq \rangle$ and $Q = \langle Q, \leq \rangle$ be posets and let $F{:}P \to Q$. F is *monotone* if $x \preceq y \implies F(x) \leq F(y)$. F is *antitone* if $x \preceq y \implies F(y) \leq F(x)$. F is *Scott-continuous* [9], or simply *continuous*, if for every upward-directed subset $D \subseteq P$ that has a supremum $\hat{d} \in P$, the set $\{F(d) \mid d \in D\}$ has a supremum that is $F(\hat{d})$, i.e.

$$\bigsqcup_{d\in D}\ F(d) \ =\ F(\bigsqcup_{d\in D}\ d) \ =\ F(\hat{d}).$$

Dually, F is continuous if for every downward-directed subset $D \subseteq P$ that has an infimum $\check{d} \in P$, then

$$\bigsqcap_{d\in D}\ F(d) \ =\ F(\bigsqcap_{d\in D}\ d) \ =\ F(\check{d}).$$

4 Quantifier Elimination Laws

Quantifier elimination rules transform a quantified expression into an equivalent quantifier-free expression. Lattice quantifiers are useful in formally specifying programs and their properties, and laws to eliminate them are a powerful tool during program design and analysis.

General Cases: We focus on the problem of eliminating a single quantifier over an unquantified expression F. Multiple quantifiers can be handled iteratively. We treat F as a function of a single variable $F{:}P \to Q$, treating the other variables as constants for the purposes of quantifier elimination. Each of our QE rules quantifies over a directed subset $D \subseteq P$. A basic question is whether D has a supremum (resp. infimum) at all, and, if it does, then whether it is in D or P. In general, if P is a poset, then D does not necessarily have a supremum (resp. infimum). If P is a cpo, then by definition a unique supremum (resp. infimum) exists and $sup(D) \in P$ (resp. $\inf(D) \in P$).

Continuity conditions provide the general form of quantifier elimination rules over ordered domains. Let $P = \langle P, \preceq \rangle$ and $Q = \langle Q, \leq \rangle$ be posets.

$F{:}P \to Q$ is monotone continuous $D \subseteq P$ an upward-directed set with supremum $\hat{d}$		$F{:}P \to Q$ is monotone continuous $D \subseteq P$ a downward-directed set with infimum $\check{d}$	
QE 1.1	$\bigsqcup_{a\in D}\ F(a) \ =\ F(\bigsqcup_{a\in D}\ a) \ =\ F(\hat{d})$	QE 1.2	$\bigsqcap_{a\in D}\ F(a) \ =\ F(\bigsqcap_{a\in D}\ a) \ =\ F(\check{d})$

$F{:}P \to Q$ is antitone continuous $D \subseteq P$ a downward-directed set with infimum $\check{d}$		$F{:}P \to Q$ is antitone continuous $D \subseteq P$ an upward-directed set with supremum $\hat{d}$	
QE -1.1	$\bigsqcup_{a\in D}\ F(a) \ =\ F(\bigsqcap_{a\in D}\ a) \ =\ F(\check{d})$	QE -1.2	$\bigsqcap_{a\in D}\ F(a) \ =\ F(\bigsqcup_{a\in D}\ a) \ =\ F(\hat{d})$

Applying one of these rules left-to-right eliminates the quantification of F over its P-valued parameter, replacing the parameter with a constant. The four rules QE $\pm$1.1, $\pm$1.2 simply restate the definition of a continuous function between posets, so they require no proof.

The applicability of these rules can be broadened by combining them with the quantifier change rules QC1 and QC2 in cases where the quantification is over a subset or superset of a directed subset. This can arise when an extra constraint is placed on the elements of a directed set, or when the quantification domain has a directed subset. For example, if an extra constraint c is placed on elements of a directed set D, i.e. such that $D' = \{d \mid d \in D \wedge c(d)\} \subseteq D$. By applying QC1 then QE 1.1, we get

$$\bigsqcup_{a \in D'} F(a) \leq \bigsqcup_{a \in D} F(a) = F(\hat{d})$$

To apply one of the QE rules requires checking several applicability conditions. For example, applying rule QE 1.1 requires verifying

1. F is continuous: Meeting this condition may seem circular since it is tantamount to verifying the very rule that we are trying to apply. However, the development of domain theory in denotational semantics has shown that most programming constructs are Scott-continuous, as too are many data structure and math operators [6]. Moreover, continuity is compositional. Recursive analysis of the syntax of expression $F(a)$ together with tables of known continuous functions and language constructs allows for efficient determination of whether it is continuous.
2. Determine the polarity of $F(a)$ relative to a: Any continuous function is either monotone or antitone. The polarity of a in $F(a)$ may be calculated via recursive analysis of the (term) structure of F [7].
3. D is an upward-directed set: This requires showing that every pair of elements in D has an upper bound.
4. $\bigsqcup_{a \in D} a = \hat{d}$ exists.
5. $\bigsqcup_{a \in D} F(a)$ exists.
6. $\bigsqcup_{a \in D} F(a) = F(\hat{d})$.

The generality of QE rules based on Scott-continuity stems from the need to handle the supremum/infimum of infinite directed sets. By formulating special cases of these rules, we can simplify the applicability conditions. An important special case occurs when poset D has a maximal or minimal element (i.e. Proposition 1 holds).

$F{:}P \to Q$ is monotone $D \subseteq P$ with maximum $\hat{d} \in D$		$F{:}P \to Q$ is monotone $D \subseteq P$ with minimum $\check{d} \in D$	
QE 2.1	$\bigsqcup_{a \in D} F(a) = F(\bigsqcup_{a \in D} a) = F(\hat{d})$	QE 2.2	$\bigsqcap_{a \in D} F(a) = F(\bigsqcap_{a \in D} a) = F(\check{d})$

$F{:}P \to Q$ is antitone $D \subseteq P$ with minimum $\check{d} \in D$		$F{:}P \to Q$ is antitone $D \subseteq P$ with maximum $\hat{d} \in D$	
QE -2.1	$\bigsqcup_{a \in D} F(a) = F(\bigsqcap_{a \in D} a) = F(\check{d})$	QE -2.2	$\bigsqcap_{a \in D} F(a) = F(\bigsqcup_{a \in D} a) = F(\hat{d})$

Under these conditions we have $\sup(D) = max(D)$ and checking the applicability conditions becomes simpler. For example, the applicability of rule QE 2.1 requires

1. Determining the polarity of $F(a)$ relative to a: again, this requires analyzing an expression for the polarity of a variable (or subexpression). The polarity of parameters to common language constructs, data structure operations, and math functions are well-known and can be tabulated. Moreover, polarities are compositional, allowing efficient analysis of polarities of an expression [7].
2. Showing $max(D) = \hat{d}$ exists: This can be checked by analyzing the expression that defines D.

To prove rule QE 2.1, assume that $F(a)$ is monotone in a, and D is a subset of P with maximal element $\hat{d} \in D$. $F(\hat{d})$ is an upper bound on $F(D)$, since by assumption $d \preceq \hat{d}$ for each $d \in D$, so $F(d) \leq F(\hat{d})$

by monotonicity. $F(\hat{d})$ is the least upper bound on $F(D)$, since for any upper bound $d' \in P$ on D, we have $\hat{d} \leq d'$, hence $F(\hat{d}) \leq F(d')$. Proofs of other rules are similar.

Specialization to the Discrete Order: Equality is a special case of a partial order, and the usual substitution-of-equals-for-equals rule (Leibniz) falls out as the special case of quantifying over a singleton domain.

Specialization to Predicates: When F is a predicate, the QE2 laws can be restated in more familiar syntax. Let $P = \langle P, \preceq \rangle$ be a poset and let $F{:}P \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$.

$F{:}P \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$ is monotone, $D \subseteq P$ with maximum $\hat{d} \in D$		$F{:}P \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$ is monotone, $D \subseteq P$ with minimum $\check{d} \in D$	
QE 3.1	$\exists(a{:}P)(a \in D \wedge F(a)) \equiv F(\hat{d})$	QE 3.2	$\forall(a{:}P)(a \in D \Longrightarrow F(a)) \equiv F(\check{d})$

$F{:}P \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$ is antitone, $D \subseteq P$ with minimum $\check{d} \in D$		$F{:}P \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$ is antitone, $D \subseteq P$ with maximum $\hat{d} \in D$	
QE -3.1	$\exists(a{:}P)(a \in D \wedge F(a)) \equiv F(\check{d})$	QE -3.2	$\forall(a{:}P)(a \in D \Longrightarrow F(a)) \equiv F(\hat{d})$

Specialization to Propositional Expressions: The specialization of quantifier elimination to propositional expressions (e.g. as in SAT and QBF problems) is as follows.

$F : \langle Boolean, \Rightarrow \rangle \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$ is monotone			
QE 4.1	$\exists(a{:}Boolean)\, F(a) \equiv F(true)$	QE 4.2	$\forall(a{:}Boolean)\, F(a) \equiv F(false)$

$F : \langle Boolean, \Rightarrow \rangle \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$ is antitone			
QE -4.1	$\exists(a{:}Boolean)\, F(a) \equiv F(false)$	QE -4.2	$\forall(a{:}Boolean)\, F(a) \equiv F(true)$

QE $\pm$4.1 and $\pm$4.2 are the basis for Pure Literal Rules in SAT and QBF solving. They also provide strong simplication rules in logic circuit design and verification. To prove them, note that we are quantifying over the finite set $\{true, false\}$ which has a maximum element *true* and a minimum element *false*.

QE over Boolean expressions with mixed polarity: Resolution rules in logic provide an example of quantifier elimination rules for expressions that contain subexpressions of differing polarities. The resolvant of expression $F[a] \wedge G[a]$ is the logical consequence $\exists(a)F[a] \wedge G[a]$. The following laws build on earlier QE laws.

monotone $F : \langle Boolean, \Rightarrow \rangle \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$, antitone $G : \langle Boolean, \Rightarrow \rangle \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$	
QE 5.1	$\exists(a{:}Boolean)(F(a) \wedge G(a)) \Rightarrow (F(false) \vee G(true))$
QE 5.2	$\exists(a{:}Boolean)(F(a) \vee G(a)) \equiv (F(true) \vee G(false))$
QE 5.3	$\forall(a{:}Boolean)(F(a) \wedge G(a)) \equiv (F(false) \wedge G(true))$
QE 5.4	$\forall(a{:}Boolean)(F(a) \vee G(a)) \Leftarrow (F(true) \wedge G(false))$

To prove QE 5.1, we assume F is monotone and G is antitone, and first prove a simple consequence

$\exists(a{:}Boolean)(F(a) \wedge G(a))$
$\Rightarrow \quad \exists(a{:}Boolean)F(a) \wedge \exists(a{:}Boolean)G(a)$ distributing the quantification
$\equiv \quad F(true) \wedge G(false).$ applying QE 4.1 and QE -4.1

Then from the top:

$\exists(a{:}Boolean)(F(a) \wedge G(a))$ LHS of QE5.1
$\equiv$ $(F(false) \wedge G(false)) \ \vee \ (F(true) \wedge G(true))$ unfolding the quantification
$\Rightarrow$ $(F(false) \wedge true) \ \vee \ (true \wedge G(true))$ using $F(true) \ \wedge \ G(false)$
$\equiv$ $F(false) \ \vee \ G(true)$. simplifying

To prove QE 5.2:

$\exists(a{:}Boolean)(F(a) \vee G(a))$ LHS of QE5.2
$\equiv$ $\exists(a{:}Boolean)F(a) \ \vee \ \exists(a{:}Boolean)G(a))$ distributing the quantification
$\equiv$ $F(true) \ \vee \ G(false)$. applying QE 4.1 and -4.1

The other laws are proved similarly. The Cut Rule from Propositional Logic, Generalized Resolution [7], and inference rules in other logics can be derived using these rules [13].

QE over real linear expressions with mixed polarity: Linear constraints over the reals provide another class of quantifier elimination rules that apply to expressions containing mixed polarities.

monotone linear $F : \langle \mathbb{R}, \leq \rangle \rightarrow \langle \mathbb{R}, \leq \rangle$, $x_\ell \leq x_u$	
QE 6.1	$\exists(x{:}\mathbb{R})(x \in [x_\ell, x_u] \wedge a \leq F(x) \leq b) \ \equiv \ a \leq F(x_u) \wedge F(x_\ell) \leq b \ \wedge \ a \leq b$
QE 6.2	$\forall(x{:}\mathbb{R})(x \in [x_\ell, x_u] \Longrightarrow a \leq F(x) \leq b) \ \equiv \ a \leq F(x_\ell) \wedge F(x_u) \leq b$

antitone linear $F : \langle \mathbb{R}, \leq \rangle \rightarrow \langle \mathbb{R}, \leq \rangle$, $x_\ell \leq x_u$	
QE -6.1	$\exists(x{:}\mathbb{R})(x \in [x_\ell, x_u] \wedge a \leq F(x) \leq b) \ \equiv \ a \leq F(x_\ell) \wedge F(x_u) \leq b \ \wedge \ a \leq b$
QE -6.2	$\forall(x{:}\mathbb{R})(x \in [x_\ell, x_u] \Longrightarrow a \leq F(x) \leq b) \ \equiv \ a \leq F(x_u) \wedge F(x_\ell) \leq b$

To prove QE 6.1, we assume F is linear and monotone, hence it has the form $F(x) = \alpha x + \beta$ for some α, β. If $\alpha = 0$, then F is constant and both sides of QE 6.1 simplify to true. So we assume $\alpha > 0$ and decompose the equivalence into two implications and proceed by case analysis.
Case ($\Rightarrow$): Let x' be a value such that $x_\ell \leq x' \leq x_u \ \wedge \ a \leq F(x') \leq b$ (implying $a \leq b$). By monotonicity, we have $F(x_\ell) \leq F(x') \leq b$ and $a \leq F(x') \leq F(x_u)$.
Case ($\Leftarrow$): Assume $a \leq F(x_u) \wedge F(x_\ell) \leq b \ \wedge \ a \leq b$. We proceed by case analysis on the value of $F(x_\ell)$. First, if $a \leq F(x_\ell)$, then let $x' = x_\ell$ and we have $x_\ell \leq x' \leq x_u$ (since $x_\ell \leq x_u$) and $a \leq F(x') \leq b$, so x' is a witness to the existential. Second, if $F(x_\ell) < a$, then consider an increment to x_ℓ to increase F's value to a: we have $F(x') = a$ for $x' = x_\ell + (a - \alpha x_\ell - \beta)/\alpha$. Since $F(x') = a \leq F(x_u)$, it follows that $x' \leq x_u$ since F is strictly monotone ($\alpha > 0$), hence $x_\ell \leq x' \leq x_u$. We also have $a = F(x') \leq b$ since $a \leq b$, hence x' is a witness to the existential. $\square$

To prove QE 6.2, we assume F is linear and monotone, hence it has the form $F(x) = \alpha x + \beta$ for some α, β. If $\alpha = 0$, then F is constant and both sides of QE 6.2 simplify to true. So we assume $\alpha > 0$ and decompose the equivalence into two implications and proceed by case analysis.
Case ($\Rightarrow$): Assume that for an arbitrary x' such that $x_\ell \leq x' \leq x_u$ we have $a \leq F(x') \leq b$. For $x' = x_\ell$, we have $a \leq F(x_\ell$ and for $x' = x_u$, we have $F(x_u) \leq b$.
Case ($\Leftarrow$): Suppose $a \leq F(x_\ell)$ and $F(x_u) \leq b$, and let x' be an arbitrary value such that $x_\ell \leq x' \leq x_u$. Using our assumptions and monotonicity of F we have $a \leq F(x_\ell) \leq F(x') \leq F(x_u) \leq b$. $\square$

The proofs of the other rules are similar.

5 Worked Examples

Eliminating Variables in Linear Systems: The Fourier-Motzkin method for eliminating variables in systems of linear equations over ordered fields (e.g. the reals) was discovered independently by Fourier (1836), Dines (1918), and Motzkin (1936). Given unquantified expressions lb and ub that contain no occurrences of x, Fourier-Motzkin essentially works as follows on the quantified expression

$\exists x.\ lb \leq x\ \wedge\ x \leq ub$

$\equiv\ \inf(\{x \mid lb \leq x\}) \leq ub$ — apply QE -3.1 where $F \mapsto \lambda x.\ x \leq ub$ and $D = \{x \mid lb \leq x\}$

$\equiv\ lb \leq ub.$ — simplifying

The same result can be obtained by applying QE 3.1 appropriately.

Predicate Transformers: The definition of wp for particular classes of statements can be calculated using quantifier elimination. wp is used to calculate with stateful operators. We treat state as an environment map from variables to values, and use a map update function $m \ll \{v = e\}$ to denote the new map m' that is the same as m except that $m'(v) = e$. The value of expression e in state st is written $st[e]$. The weakest precondition of an assignment statement is calculated as follows.

$wp(st, x{:=}\ e, R)$

$\equiv\ \exists(st'{:}State)(Post(st, x{:=}\ e, st') \wedge st'[R])$ — definition

$\equiv\ \exists(st'{:}State)(st' = st \ll \{x = e\} \wedge st'[R])$ — definition of $Post$

$\equiv\ st \ll \{x = e\}[R].$ — applying QE 6.1

Other predicate transformer expressions are calculated in a similar way.

A more complex example arises from controller synthesis design problems [10]. Suppose that we are modeling a system with two real-valued state variables x_1 and x_2 and a linear transition function

$$F(x_1, x_2, u, d)\ =\ \langle x_1 + d - x_2, x_2 + u\rangle$$

where $u \in [-1, 1]$ is a control input and $d \in [0, 4]$ is an uncontrollable disturbance. The goal is to enforce that the result of a transition satisfies a safety constraint $\phi\ \equiv\ 0 \leq x_1 \leq 20\ \wedge\ 0 \leq x_2 \leq 4$, giving rise to a game-like weakest precondition formula $wp(\langle x_1, x_2\rangle, F, \phi)$:

$$\exists(u)(u \in [-1,1] \wedge \forall(d)(d \in [0,4]\ \Longrightarrow\ \exists(x_1', x_2')(\langle x_1', x_2'\rangle = F(x_1, x_2, u, d) \wedge 0 \leq x_1' \leq 20 \wedge 0 \leq x_2' \leq 4))).$$

and we calculate as follows:

$wp(\langle x_1, x_2\rangle, F, \phi)$

$\equiv$ { definition }

$\exists(u)(u \in [-1,1] \wedge \forall(d)(d \in [0,4] \Longrightarrow \exists(x_1', x_2')(\langle x_1', x_2'\rangle = F(x_1, x_2, u, d) \wedge 0 \leq x_1' \leq 20\ \wedge\ 0 \leq x_2' \leq 4)))$

$\equiv$ { applying QE 6.1 on the innermost quantification, singleton domain }

$\exists(u)(u \in [-1,1] \wedge \forall(d)(d \in [0,4] \Longrightarrow 0 \leq F(x_1, x_2, u, d).1 \leq 20\ \wedge\ 0 \leq F(x_1, x_2, u, d).2 \leq 4))$

$\equiv$ { unfolding the definition of F }

$$\exists(u)(u \in [-1,1] \wedge \forall(d)(d \in [0,4] \Longrightarrow 0 \leq x_1 + d - x_2 \leq 20 \ \wedge \ 0 \leq x_2 + u \leq 4))$$

$\equiv$ { rearranging }

$$\exists(u)(u \in [-1,1] \wedge 0 \leq x_2 + u \leq 4 \ \wedge \ \forall(d)(d \in [0,4] \Longrightarrow 0 \leq x_1 + d - x_2 \leq 20))$$

$\equiv$ { applying QE 6.2 on the quantification of d }

$$\exists(u)(u \in [-1,1] \wedge 0 \leq x_2 + u \leq 4 \ \wedge \ 0 \leq x_1 + 0 - x_2 \ \wedge \ x_1 + 4 - x_2 \leq 20)$$

$\equiv$ { rearranging }

$$0 \leq x_1 - x_2 \leq 16 \ \wedge \ \exists(u)(u \in [-1,1] \wedge 0 \leq x_2 + u \leq 4)$$

$\equiv$ { applying QE 6.1 on the quantification of u }

$$0 \leq x_1 - x_2 \leq 16 \ \wedge \ 0 \leq x_2 + 1 \ \wedge \ x_2 - 1 \leq 4$$

$\equiv$ { rearranging}

$$0 \leq x_1 - x_2 \leq 16 \ \wedge \ -1 \leq x_2 \leq 5.$$

The result is the set of states from which there is a control input that lands the system in a safe state after one transition, no matter what the disturbance is.

Algorithm Design: In global search theory [11, 12], pruning mechanisms are specified by a quantified expression:

$$\exists z.\, O(x,z) \ \wedge \ z \ \sqsubseteq \ \hat{r}$$

where O asserts that z is a feasible solution and the other subexpression asserts that z is an instance of the abstract domain element $\hat{r}$.

For example, problems such as the k-queens or the Traveling Salesman Problem seek a permutation of a set with certain properties. A global search algorithm searching over permutations might interpret the abstract domain element as a (partial solution) sequence ps with the requirement that a complete solution must be an injection to a set, say, S, which we express as the constraint $injective(z, S)$. Interpreting feasibility condition O as $injective(z, S)$, which is antitone, and the refinement relation $z \ \sqsubseteq \ \hat{r}$ as the function $extends(z, ps)$ (i.e. the sequence z is an extension of ps or ps is a suffix of z), we instantiate the pruning specification as

$$\exists z.\, injective(z, S) \ \wedge \ extends(z, ps).$$

With this much structure, we can calculate

$\exists z.\, injective(z, S) \ \wedge \ extends(z, ps)$

$\equiv$ $injective(min(\{z \mid extends(z, ps)\}), S)$ — apply QE -3.1 where F is $\lambda z.\, injective(z, S)$ and directed subset D is $\{z \mid extends(z, ps)\}$

$\equiv$ $injective(ps, S)$. — simplifying

We use the result as a pruning test: if $injective(ps, S)$ is *false* for some partial solution ps during search then we know there do not exist any feasible solutions that extend ps so we can eliminate it from further consideration/search. In the above calculation, we focused on just one constraint, but typically a feasible solution satisfies a conjunction of constraints and we select just those conjuncts that are monotone to derive a pruning test.

A similar derivation can be given for the optimization pruning mechanism given earlier. It is typically the case that $cost(x, z)$ is the linear function $c \cdot z$ which is monotone in the candidate solution sequence z. Instantiating the cost minimization specification, we get

$$\min_{extends(z,ps) \wedge injective(ps,S)} c \cdot z$$

$$\begin{array}{lll} \geq & \min_{extends(z,ps)} c \cdot z & \text{apply QC2} \\ = & c \cdot min(\{z \mid extends(z, ps)\}) & \text{apply QE 2.2 where monotone } F \text{ is } \lambda z.\ c \cdot z \\ & & \text{and } D \text{ is } \{z \mid extends(z, ps)\} \\ = & c \cdot ps. & \text{simplifying} \end{array}$$

This expression provides a lower-bound function on the cost of feasible solutions that refine the partial solution ps.

Temporal Propagation in Scheduling: We can apply QE to derive constraint propagation rules from the formula that relates the concrete start times of tasks to their abstract time windows:

$$\begin{array}{lll} \forall st_1(est_1 \leq st_1 \leq lst_1 \implies \exists st_2(est_2 \leq st_2 \leq lst_2 \ \wedge \ st_1 + duration \leq st_2)) & & \\ \equiv & \forall st_1(est_1 \leq st_1 \leq lst_1 \implies st_1 + duration \leq lst_2) & \text{applying QE 3.1} \\ \equiv & lst_1 + duration \leq lst_2. & \text{applying QE -3.2} \end{array}$$

A similar calculation applying QE -3.1 then QE 3.2 yields

$$\forall st_2(est_2 \leq st_2 \leq lst_2 \implies \exists st_1(est_1 \leq st_1 \leq lst_1 \ \wedge \ st_1 + duration \leq st_2)) \equiv est_1 + duration \leq est_2.$$

The results are definite constraints that can be used to propagate the effect of decisions on time windows, ensuring that the current partial valuation is temporally consistent.

6 Related Work

Many problems can be formulated in terms of quantifier elimination in some theory T, but cannot be solved simply based on the order properties of T. Let ϕ be a constraint over the variables $V = \{v_1, v_2, \cdots, v_n\}$ and the vocabulary of T. *Constraint solving* seeks to determine if there exists a structure for $T \bigcup V$ that satisfies ϕ; i.e. $T \models \exists(V)\phi$. Such a solution structure can be expressed as quantifier-free formula of the form $v_1 = c_1 \wedge v_2 = c_2 \wedge \cdots \wedge v_n = c_n$ where each c_i, $i = 1, .., n$ is a constant. It is a sufficient condition of $\exists(V)\phi$. *Theorem proving (validity checking)* is dual to constraint-solving in that the task is to show that all structures over $T \bigcup V$ satisfy the given constraint ϕ modulo T. Typically this involves mostly universal quantification. As a QE problem, the task is to show that the universally quantified formula is equivalent to the unquantified constant *true*.

Cylindrical Algebra Decomposition is a quantifier elimination algorithm that is specialized to closed real fields, allowing quantifier elimination over systems of polynomials [1].

Projection techniques (that reduce dimensionality of set of constraints) are a form of quantifier elimination and are known to provide a unifying view of logical inference and optimization [4].

Gulwani and Musuvathi [3] define a notion of eliminating existential quantifiers by finding their least upper bound when the contextual theory cannot express an equivalent unquantified expression. A bound is often sufficient in program analysis tasks.

7 Conclusion

We have presented general rules for eliminating quantifiers over ordered domains and illustrated them with a variety of examples. These rules provide a component of a deductive inference capability that is useful in program calculation and program verification. They generalize and simplify rules that we have used in several program synthesis systems [5, 12].

Acknowledgments: Thanks to Christoph Kreitz and Julia Leighton for comments on this paper.

References

1. Collins, G. E. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In *Automata Theory and Formal Languages 2nd GI Conference* (1975), Springer, pp. 134–183.
2. Dijkstra, E. W. *A Discipline of Programming.* Prentice-Hall, Englewood Cliffs, NJ, 1976.
3. Gulwani, S., and Musuvathi, M. Cover algorithms and their combination. In *Programming Languages and Systems* (2008), Springer, pp. 193–207.
4. Hooker, J. N. Projection, consistency, and George Boole. *Constraints 21*, 1 (2016), 59–76.
5. Kestrel Institute. *Specware System and documentation*, 2003. http://www.specware.org/.
6. Manna, Z. *Mathematical Theory of Computation.* New York, McGraw-Hill, 1974.
7. Manna, Z., and Waldinger, R. Special relations in automated deduction. *Journal of the ACM 33*, 1 (January 1986), 1–59.
8. Rehof, J., and Mogenson, T. Tractable constraints in finite semilattices. *Science of Computer Programming 35* (1999), 191–221.
9. Scott, D. Continuous lattices. In *Toposes, Algebraic Geometry and Logic* (1972), F. W. Lawvere, Ed., Springer, pp. 97–136.
10. Slanina, M., Sankaranarayanan, S., Sipma, H., and Manna, Z. Controller synthesis of discrete linear plants using polyhedra. Tech. rep., Technical Report REACT-TR-2007-01, Stanford University, 2007.
11. Smith, D. R. Structure and design of global search algorithms. Tech. Rep. KES.U.87.12, Kestrel Institute, November 1987. ftp://ftp.kestrel.edu/pub/papers/smith/gs.pdf.
12. Smith, D. R. KIDS – a semi-automatic program development system. *IEEE Transactions on Software Engineering Special Issue on Formal Methods in Software Engineering 16*, 9 (1990), 1024–1043.
13. Smith, D. R., and Westfold, S. Toward the Synthesis of Constraint Solvers. Tech. rep., Kestrel Institute, 2013. http://www.kestrel.edu/home/people/smith/pub/CW-report.pdf.

Quantifier Elimination and Quantifier Change Rules

$F{:}P \to Q$ is monotone continuous $D \subseteq P$ an up-directed set with supremum $\hat{d}$		$F{:}P \to Q$ is monotone continuous $D \subseteq P$ a down-directed set with infimum $\check{d}$	
QE 1.1	$\bigsqcup_{a \in D} F(a) \;=\; F(\hat{d})$	QE 1.2	$\bigsqcap_{a \in D} F(a) \;=\; F(\check{d})$
$F{:}P \to Q$ is antitone continuous $D \subseteq P$ a down-directed set with infimum $\check{d}$		$F{:}P \to Q$ is antitone continuous $D \subseteq P$ an up-directed set with supremum $\hat{d}$	
QE -1.1	$\bigsqcup_{a \in D} F(a) \;=\; F(\check{d})$	QE -1.2	$\bigsqcap_{a \in D} F(a) \;=\; F(\hat{d})$

$F{:}P \to Q$ is monotone $D \subseteq P$ with maximum $\hat{d}$		$F{:}P \to Q$ is monotone $D \subseteq P$ with minimum $\check{d}$	
QE 2.1	$\bigsqcup_{a \in D} F(a) \;=\; F(\hat{d})$	QE 2.2	$\bigsqcap_{a \in D} F(a) \;=\; F(\check{d})$
$F{:}P \to Q$ is antitone $D \subseteq P$ with minimum $\check{d}$		$F{:}P \to Q$ is antitone $D \subseteq P$ with maximum $\hat{d}$	
QE -2.1	$\bigsqcup_{a \in D} F(a) \;=\; F(\check{d})$	QE -2.2	$\bigsqcap_{a \in D} F(a) \;=\; F(\hat{d})$

$F{:}P \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$ is monotone, $D \subseteq P$ with maximum $\hat{d}$		$F{:}P \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$ is monotone, $D \subseteq P$ with minimum $\check{d}$	
QE 3.1	$\exists(a{:}P)(a \in D \wedge F(a)) \;\equiv\; F(\hat{d})$	QE 3.2	$\forall(a{:}P)(a \in D \Longrightarrow F(a)) \;\equiv\; F(\check{d})$
$F{:}P \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$ is antitone, $D \subseteq P$ with minimum $\check{d}$		$F{:}P \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$ is antitone, $D \subseteq P$ with maximum $\hat{d}$	
QE -3.1	$\exists(a{:}P)(a \in D \wedge F(a)) \;\equiv\; F(\check{d})$	QE -3.2	$\forall(a{:}P)(a \in D \Longrightarrow F(a)) \;\equiv\; F(\hat{d})$

$F : \langle Boolean, \Rightarrow \rangle \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$ is monotone			
QE 4.1	$\exists(a{:}Boolean)\, F(a) \;\equiv\; F(true)$	QE 4.2	$\forall(a{:}Boolean)\, F(a) \;\equiv\; F(false)$
$F : \langle Boolean, \Rightarrow \rangle \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$ is antitone			
QE -4.1	$\exists(a{:}Boolean)\, F(a) \;\equiv\; F(false)$	QE -4.2	$\forall(a{:}Boolean)\, F(a) \;\equiv\; F(true)$

monotone $F : \langle Boolean, \Rightarrow \rangle \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$, antitone $G : \langle Boolean, \Rightarrow \rangle \to \langle Boolean, \wedge, \vee, \Rightarrow \rangle$	
QE 5.1	$\exists(a{:}Boolean)(F(a) \wedge G(a)) \Rightarrow (F(false) \vee G(true))$
QE 5.2	$\exists(a{:}Boolean)(F(a) \vee G(a)) \;\equiv\; (F(true) \vee G(false))$
QE 5.3	$\forall(a{:}Boolean)(F(a) \wedge G(a)) \;\equiv\; (F(false) \wedge G(true))$
QE 5.4	$\forall(a{:}Boolean)(F(a) \vee G(a)) \Leftarrow (F(true) \wedge G(false))$

monotone linear $F : \langle \mathbb{R}, \leq \rangle \to \langle \mathbb{R}, \leq \rangle$, $x_\ell \leq x_u$	
QE 6.1	$\exists(x{:}\mathbb{R})(x \in [x_\ell, x_u] \wedge a \leq F(x) \leq b) \;\equiv\; a \leq F(x_u) \wedge F(x_\ell) \leq b \;\wedge\; a \leq b$
QE 6.2	$\forall(x{:}\mathbb{R})(x \in [x_\ell, x_u] \Longrightarrow a \leq F(x) \leq b) \;\equiv\; a \leq F(x_\ell) \wedge F(x_u) \leq b$
antitone linear $F : \langle \mathbb{R}, \leq \rangle \to \langle \mathbb{R}, \leq \rangle$, $x_\ell \leq x_u$	
QE -6.1	$\exists(x{:}\mathbb{R})(x \in [x_\ell, x_u] \wedge a \leq F(x) \leq b) \;\equiv\; a \leq F(x_\ell) \wedge F(x_u) \leq b \;\wedge\; a \leq b$
QE -6.2	$\forall(x{:}\mathbb{R})(x \in [x_\ell, x_u] \Longrightarrow a \leq F(x) \leq b) \;\equiv\; a \leq F(x_u) \wedge F(x_\ell) \leq b$

$D \subseteq E$ and $f{:}E \to \langle L, \sqcup, \sqcap, \leq \rangle$			
QC1	$\bigsqcup_{a \in D} f(a) \leq \bigsqcup_{a \in E} f(a)$	QC2	$\bigsqcap_{a \in D} f(a) \geq \bigsqcap_{a \in E} f(a)$