

Logahedra: a New Weakly Relational Domain

Jacob M. Howe¹ and Andy King^{2,3}

¹ Department of Computing, City University London, EC1V 0HB, UK

² Portcullis Computer Security Limited, Pinner, HA5 2EX, UK

³ Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK

Abstract. Weakly relational numeric domains express restricted classes of linear inequalities that strike a balance between what can be described and what can be efficiently computed. Popular weakly relational domains such as bounded differences and octagons have found application in model checking and abstract interpretation. This paper introduces logahedra, which are more expressive than octagons, but less expressive than arbitrary systems of two variable per inequality constraints. Logahedra allow coefficients of inequalities to be powers of two whilst retaining many of the desirable algorithmic properties of octagons.

1 Introduction

Polyhedra are used in abstract interpretation [4] and model checking real-time [9] and hybrid systems [7]. The domain operations of general polyhedra can be prohibitively expensive, thus there has been much recent interest in so-called weakly relational domains that seek to balance expressivity and cost by imposing restrictions on the class of inequalities that can be represented. For example, octagons [11] restrict polyhedra [4] to inequalities of at most two variables where the coefficients are -1, 0 or 1 and thereby obtain (at worst) cubic domain operations. Other weakly relational domains whose operations reside in low complexity classes are pentagons [10], two variable per inequality (TVPI) constraints [17] and bounded differences [9]. Domains that do not impose the two variable per inequality restriction include octahedra [3] and template constraints [14].

This paper introduces a new class of weakly relational domain called logahedra. A logahedron is a system of implicitly conjoined two variable inequalities where the coefficients are constrained to be powers of two (or zero). Such coefficients naturally arise because the size of primitive types. For instance, suppose an array of 32-bit integers was dynamically allocated with, say, `malloc(n)` where *n* is the size of the memory block in bytes. Then an array index *i* is in range iff the logahedral inequalities $0 \leq i$ and $4i + 4 \leq n$ are satisfied. Logahedra are proposed as a solution to two problems arising in program analysis. The first problem is that octagons, whilst having good computational properties, are restricted in what they can describe. The second problem is that when the coefficients of inequalities are not constrained (as they are for octagons), for example in general polyhedra or TVPI, the coefficients can easily become very large, requiring

multiple precision libraries for their storage and manipulation. This can be prohibitively costly [16]. Logahedra address the first problem by allowing a greater variety of constraints to be expressible than octagons, whilst retaining octagons' good computational properties (indeed, logahedra are a true generalisation of octagons; logahedra are strictly more expressive, with octagons being a special case). They address the second problem by restricting the possible coefficients of inequalities; further, since the allowable coefficients are powers of two, they can be represented by their exponents rather than by the number itself, allowing very large coefficients to be represented using machine integers.

Logahedra are themselves a strict subset of TVPI constraints and inherit many of their domain operations. Yet the most important domain operation, (full) completion, has the same complexity as for octagons hence is more efficient than for TVPI, being (truly) cubic. The most complicated domain operation, as with TVPI, is incremental completion. This is the operation of adding a single constraint to an already complete system to give an updated complete system. Incrementally adding constraints is more in tune with the needs of analysis than full completion, therefore incremental completion is arguably the key operation. This operation is also the most complicated and is synthesised from the way new inequalities can be derived in the act of completing a system. This result is applicable to arbitrary two variable systems, not just logahedra. The paper advances the theory of weakly relation domains by making the following contributions:

- The class of logahedral constraints is introduced and it is argued that they have representational advantages over TVPI constraints, whilst being more expressive than octagons.
- A parameterised subclass of logahedra, bounded logahedra, is defined that is a generalisation of octagons in that octagons are a special case of bounded logahedra. Bounded logahedra are more expressive than octagons, whilst retaining their asymptotic complexity.
- Domain operations for both logahedral and bounded logahedral constraints are defined and algorithms for the operations are presented. In part, these build on TVPI operations and include original approaches to completion and abstraction that are applicable to other weakly relational domains.
- Preliminary experiments (and an example) indicate that logahedra have the potential to significantly increase the power of analysis.

2 Logahedral Constraints

Logahedra fall between octagons [11] and TVPI [17] in that octagonal constraints can be expressed as logahedral constraints which, in turn, can be expressed as TVPI constraints, that are themselves two variable restrictions of polyhedral constraints. These classes are defined over a set of (indexed) variables X :

Definition 1. $\text{Oct} = \{ax + by \leq d \mid x, y \in X \wedge a, b \in \{-1, 0, 1\} \wedge d \in \mathbb{Q}\}$

Definition 2. $\text{Log} = \{ax + by \leq d \mid x, y \in X \wedge a, b \in \{-2^n, 0, 2^n \mid n \in \mathbb{Z}\} \wedge d \in \mathbb{Q}\}$

Definition 3. $\text{TVPI} = \{ax + by \leq d \mid x, y \in X \wedge a, b, d \in \mathbb{Q}\}$

Definition 4. $\text{Poly} = \{\sum_{i=1}^{|X|} a_i x_i \leq d \mid x_i \in X \wedge a_i, d \in \mathbb{Q}\}$

Both **Oct** and **TVPI**, like **Poly**, are closed under variable elimination, that is, if $y \in X$ and $S \subseteq \text{Oct}$ (respectively $S \subseteq \text{TVPI}$) then $\exists y.S \in \text{Oct}$ (respectively $\exists y.S \in \text{TVPI}$). For instance, if $S = \{x - 2y \leq 5, 3y + z \leq 7, 5y - u \leq 0\} \subseteq \text{TVPI}$ then $\exists y.S$ can be derived by combing pairs of inequality with opposing signs for y to obtain $\exists y.S = \{3x + 2z \leq 29, -2u + 5x \leq 25\}$ which is indeed in **TVPI**. Variable elimination (projection) is an important operation in program analysis and an abstract domain should ideally be closed under it so as to minimise the loss of information; **Log** possesses this property. Furthermore, as well as being more expressive than **Oct**, **Log** has representational advantages over **TVPI**. This makes **Log** a natural object for study.

2.1 Representation of Coefficients

The representational advantages of logahedra are hinted at by their name. Since the absolute value of the coefficients are powers of two, it suffices to represent the logarithm of the value, rather than the value itself. This allows coefficients to be presented by their exponents in machine words, thereby avoiding the computational burden of large coefficients.

As with **TVPI** inequalities, a logahedral inequality can be binary, that is, involve two variables, when $a \neq 0$ and $b \neq 0$; or be unary, when either $a = 0$ or $b = 0$ (but not both); or be constant when $a = b = 0$. Constant constraints are written as either *true* or *false*. A dense representation for the binary case can be achieved by observing that $ax + by \leq d$ can be expressed as $x + (b/a)y \leq d/a$ if $a > 0$ and $-x - (b/a)y \leq -d/a$ otherwise. Thus to represent $ax + by \leq d$ it is sufficient to distinguish the variables x and y , represent the signs of a and b (as two bits) and then either represent $\lg |b/a|$ and d/a or $\lg |b/a|$ and $-d/a$. A unary inequality such as $ax \leq d$ can be expressed as $x \leq d/a$ if $a > 0$ and $-x \leq -d/a$ otherwise, therefore it is not even necessary to represent a logarithm. Henceforth, without loss of generality, all logahedra will be represented with first coefficient of -1, 0 or 1.

2.2 Representation of Constants

Unlike **TVPI**, that can alternatively be defined with $a, b, d \in \mathbb{Z}$, the constant d is required to be rational, even without a restriction on the first coefficient. Consider, for example, $4x - 2y \leq 2/3$. This has no equivalent logahedral representation with an integer constant. Furthermore, rational constants are required for closure under variable elimination. Consider eliminating y from a system such as $\{x - 2y \leq 3, x + 4y \leq -1\}$ which yields the single inequality $3x \leq 5$. This, in itself, is not logahedral but the constraint can be equivalently expressed as $x \leq 5/3$ which is logahedral.

2.3 Bounded Logahedra and their Representation

The **Log** class contains an unbounded number of inequalities for each two variable pair. The **Oct** class, however, is bounded since the size of the coefficients is bounded. Therefore it is worth considering restricting the coefficients of logahedra to within a bound:

Definition 5. $\text{Log}_k = \{ax + by \leq d \mid x, y \in X \wedge a \in \{-1, 0, 1\} \wedge b \in C_k \wedge d \in \mathbb{Q}\}$ where $C_k = \{-2^n, 0, 2^n \mid n \in \mathbb{Z} \wedge -k \leq n \leq k\}$, $k \in \mathbb{N}$.

Notice that $\text{Oct} = \text{Log}_0$. An alternative definition would restrict the first coefficient to be a power of two rather than a unit. However, it is curious to observe that the class Log_3 is not expressible with the alternative definition.

The case for Log_k is further motivated by considering the inequalities required to describe relationships between values stored in machine integers. The following proposition states that inside a bounded box (induced by the size of the type) the set of integer points described by a **Log** constraint can also be described by a Log_k constraint where k is determined by the size of the bounding box.

Proposition 1. Suppose $ax + by \leq d \in \text{Log}$ and $\text{Box}_k = ([-2^k, 2^k - 1] \cap \mathbb{Z})^2$. Then there exists $ax + b'y \leq d' \in \text{Log}_{k+1}$ such that $\{\langle x, y \rangle \in \text{Box}_k \mid ax + by \leq d\} = \{\langle x, y \rangle \in \text{Box}_k \mid ax + b'y \leq d'\}$

Proof. Wlog suppose $c \equiv ax + by \leq d \in \text{Log}$ where $a \in \{-1, 1\}$ and $b \notin C_{k+1}$.

Find $\langle x^*, y^* \rangle \in \mathbb{Z}^2$ that maximises $ax + by$ subject to $ax + by \leq d$, $-2^k \leq x \leq 2^k - 1$ and $-2^k \leq y \leq 2^k - 1$. (This can be achieved in $O(\lg|b|)$ time [5].) The **Log** constraint $ax + by \leq d'$, where $d' = ax^* + by^*$, describes the same set of integer points in Box_k as c . Put $s = \text{sign}(b)$, $l = \lg(|b|)$, $k' = \text{sign}(l)(k + 1)$ and $b' = s2^{k'}$. Then $c' := (ax + b'y \leq d') \in \text{Log}_{k+1}$.

If $b' = s2^{k+1}$ then c' adds no new integer solutions since Box_k has height $2^{k+1} - 1$ and c' passes through $\langle x^*, y^* \rangle$. Likewise, if $b' = s2^{-(k+1)}$ then again c' adds no new solutions since Box_k has width $2^{k+1} - 1$ and c' passes through $\langle x^*, y^* \rangle$. The result follows.

The Log_k class has $4(2k + 1)$ binary and 4 unary inequalities for each pair of variables, therefore, for fixed k , the domain is bounded. The force of the proposition is that for an signed integer representation of, say, 32 bits, it is sufficient to restrict attention to Log_{33} . For unsigned integers analogous results hold. Importantly, observe that for any given Box_k , the domain Log_{k+1} retains closure under variable elimination. This is because, by proposition 1, any inequality (obtained by combining a pair of inequalities) that falls outside Log_{k+1} can be replaced with another drawn from Log_{k+1} without loss of information. A final observation that is potentially exploitable is that for a given Box_k and a pair of coefficients, there is a maximum value for the constant beyond which the inequality does not restrict the box. For example, $x + y \leq d$ is vacuous if $2^{k+1} - 2 \leq d$.

3 Worked example

This section contains an example to demonstrate the use of logahedra in value range analysis. It serves to illustrate the domain operations required, for which definitions and algorithms will be given in the next section. In addition, the example illustrates the expressivity of logahedra versus octagons.

In the following C program, `read_value()` reads a value from a file. The objective of the analysis is to verify the safety of the array access in line 5, no matter what values are read. To this end, the set of $\langle x, y \rangle$ values that can arise immediately after executing lines (1) ... (6) are over-approximated by the logahedra P_1, \dots, P_6 . Each of these logahedra are defined by a separate equation. The set P'_2 over-approximates the set of $\langle x, y \rangle$ values occurring immediately before line 2. P'_2 differs from P_2 in that P_2 assumes that the loop condition holds. The updates at lines (3), (4) and (6) are modelled as translations. Since the value `read_value()` is not known at analysis time, the values of $\langle x, y \rangle$ at line (5) can be either that at line (3) or (4). P_5 is thus defined as the join of P_3 and P_4 . P'_2 is also formulated as the join of P_1 and P_6 , but also applies widening, ∇ .

(1)	<code>int x = 0, y = 0, array[8];</code>	$P_1 = \{\langle 0, 0 \rangle\}$
(2)	<code>while (x < 4)</code>	$P'_2 = P'_2 \nabla (P_1 \sqcup P_6) \quad P_2 = P'_2 \sqcap \{\langle x, y \rangle \mid x < 4\}$
	<code>{ if (read_value() == 0)</code>	
(3)	<code>y = y+2;</code>	$P_3 = \{\langle x, y + 2 \rangle \mid \langle x, y \rangle \in P_2\}$
	<code>else</code>	
(4)	<code>y = y+1;</code>	$P_4 = \{\langle x, y + 1 \rangle \mid \langle x, y \rangle \in P_2\}$
(5)	<code>array[y] = y;</code>	$P_5 = P_3 \sqcup P_4$
(6)	<code>x = x+1; }</code>	$P_6 = \{\langle x + 1, y \rangle \mid \langle x, y \rangle \in P_5\}$

Solutions to the equations, or at least upper-approximations to them, can be found by repeatedly applying the equations until a fixpoint is reached. As in other polyhedral analyses [4], widening is introduced to enforce convergence since P_1, P'_2, \dots, P_6 grow as the equations are reapplied. To obtain convergence, it is sufficient to put $Q_1 \nabla Q_2 = Q_1 \sqcup Q_2$ if Q_1 and Q_2 differ in dimension; otherwise $Q_1 \nabla Q_2$ is defined as the (non-redundant) inequalities of Q_1 that hold for Q_2 . This removes unstable bounds from Q_1 whilst ensuring $Q_1 \sqcup Q_2 \subseteq Q_1 \nabla Q_2$ [6].

The diagrams in Fig. 1 show how P_1, P'_2, \dots, P_6 develop during the fixpoint calculation from their initial values of \emptyset . Diagram (a) shows how P_1 is changed by the first equation; thereafter P_1 is stable. Initially $P_6 = \emptyset$ so that $P_1 \sqcup P_6 = P_1$. Since $P'_2 = \emptyset$ differs in dimension from P_1 , the second equation assigns P_1 to P'_2 . Diagram (c) illustrates P_5 , which is a line segment, that is the join of P_3 and P_4 which are themselves translations of P_2 . P_6 in diagram (d) is a translation of P_5 . When reapplied the second equation, P'_2 and $P_1 \sqcup P_6$ again differ in dimension so that P'_2 is updated to the solid triangle depicted in diagram (e). Diagrams (f) and (g) illustrate the effect of translations and a join, P_5 , and another translation, P_6 . On the third application of the second equation, P'_2 and $P_1 \sqcup P_6$ are 2 dimensional, hence P'_2 is updated to just retain the two stable inequalities, as illustrated in (i). Diagram (j) shows how the loop condition is

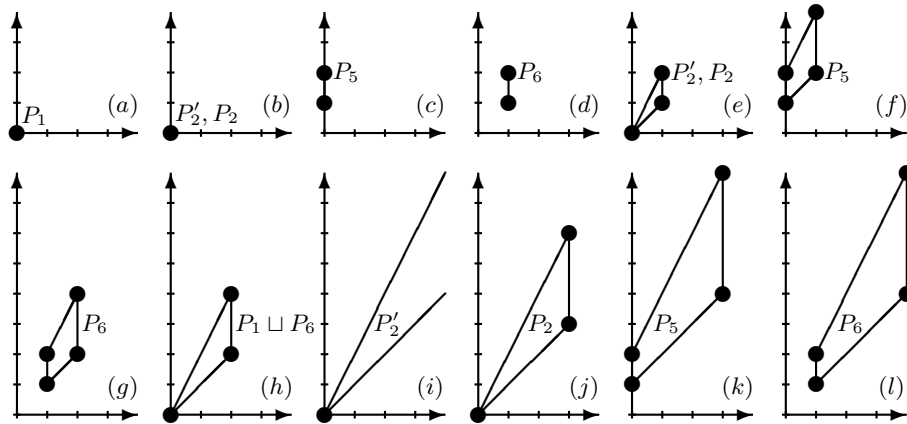


Fig. 1. Logahedra P_1, P'_2, P_2, P_5 and P_6 (P_3 and P_4 are omitted)

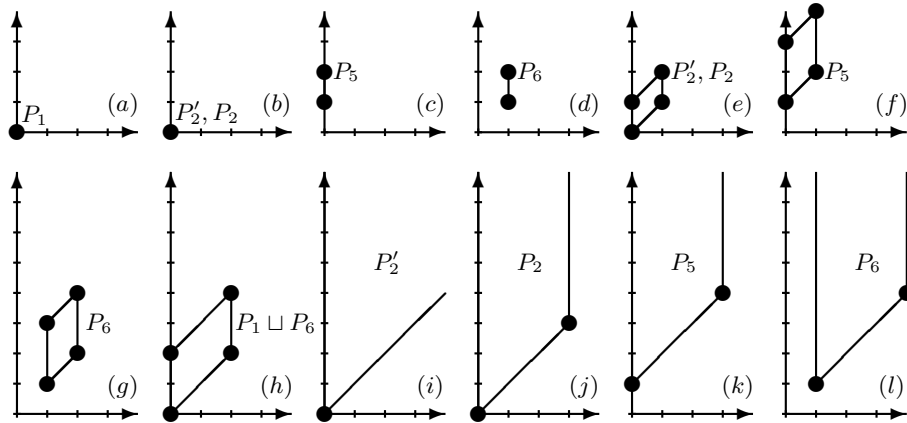


Fig. 2. Octagons P_1, P'_2, P_2, P_5 and P_6 (P_3 and P_4 are omitted)

reinserted. P'_2 remains unchanged when its equation is applied a fourth time. The $\langle x, y \rangle$ values summarised by P_5 in diagram (k) show that y can possibly take a value of 8, indicating a possibly erroneous array access. The analysis shows that enlarging the array by one element alleviates the problem. For this example, the analysis would terminate without widening, though this is not always so.

Fig. 2 repeats the analysis with octagons. Diagrams (e) and (h) show that the upper bound on y is unstable since the domain cannot express $y \leq 2x$. Hence, in diagram (i), P'_2 loses crucial information on the maximal values of y . Moreover, if widening was not applied, then the upper bound of y would grow indefinitely. Thus the loss of information cannot be remedied by more sophisticated widening. Curiously, the example was not manufactured to show the benefits of logahedra, but rather to illustrate polyhedral analysis to a lay audience.

4 Logahedral Domain Operations

This section details the domain operations previously introduced. These operate on finite sets of inequalities, such as $\wp^f(\mathbf{Log})$ and $\wp^f(\mathbf{Log}_k)$, where $\wp^f(S)$ denote the set of finite subsets of a given set S . The *entailment* ordering on $\wp^f(\mathbf{Poly})$, and its subdomains $\wp^f(\mathbf{TVPI})$, $\wp^f(\mathbf{Log})$ and $\wp^f(\mathbf{Log}_k)$, is given by $I_1 \models I_2$ iff any assignment that satisfies each inequality $c_1 \in I_1$ also satisfies each $c_2 \in I_2$. *Equivalence* is defined as $I \equiv I'$ iff $I \models I'$ and $I' \models I$.

Example 1. If $I = \{x - 2y \leq 7, y \leq 2\}$ and $I' = \{x \leq 12\}$ then $I \models I'$ since every assignment to x and y satisfying I also satisfies I' . But $I' \not\models I$. If $c := 3x \leq 2$ and $c' := x \leq 2/3$ then $\{c\} \equiv \{c'\}$ and indeed c and c' are multiples of one another.

4.1 Completion

The algorithms for many operations on logahedra (and other weakly relational domains) require implied inequalities to be made explicit. The process of inferring all implied constraints is called completion and is the dominating computational expense in the operations of which it forms a part. Therefore a clear understanding of how completion is applied, along with efficient algorithms, is essential. Completion is formalised in terms of syntactic projection:

Definition 6. If $Y \subseteq X$ then *syntactic projection* onto Y is defined $\pi_Y(S) = \{c \in S \mid \text{vars}(c) \subseteq Y\}$, where $\text{vars}(c)$ is the set of variables occurring in c .

Definition 7. The set of logahedral inequalities $I \subseteq \mathbf{Log}$ is *complete* iff for all $c \in \mathbf{Log}$ it holds that if $I \models c$ then $\pi_{\text{vars}(c)}(I) \models c$.

Example 2. Let $I_0 = \{x - y \leq 0, 2x + y \leq 1\}$. I_0 is not complete because $I_0 \models x \leq 1/3$ but $\pi_{\{x\}}(I_0) = \emptyset \not\models x \leq 1/3$. Put $I_1 = I_0 \cup \{x \leq 1/3\}$. The constant constraint *true* does not compromise completion since $\pi_{\emptyset}(I_1) = \emptyset \models \text{true}$.

Example 3. Suppose $I_0 = \{x - y \leq -1, y - z \leq -1, z - x \leq -1\}$. I_0 is not complete since $\pi_{\{x,z\}}(I_0) \not\models x - z \leq -2$ and $\pi_{\{x,y\}}(I_0) \not\models y - x \leq -2$. Put $I_1 = I_0 \cup \{x - z \leq -2, y - x \leq -2\}$. I_1 is still not complete since $\pi_{\emptyset}(I_1) \models \text{false}$ (I_0 is unsatisfiable). Put $I_2 = I_1 \cup \{\text{false}\}$. Then I_2 is complete.

The action of deriving implied inequalities, or computing resultants to use the terminology of Nelson [12], is formalised below:

Definition 8. If $c = ax + by \leq d$, $c' = a'x + b'z \leq d'$ and $a.a' < 0$ then $\text{result}(c, c', x) = |a'|by + |a|b'z \leq |a'|d + |a|d'$ otherwise $\text{result}(c, c', x) = \perp$.

The operation $\text{result}(c, c', x)$ is analogously defined when $c := ax \leq d$ or $c' := a'x \leq d'$. Note that it is necessary to stipulate which variable is eliminated because a single pair of inequalities may possess two resultants, as is illustrated by the pair $c := x + y \leq 1$, $c' := -2x - 3y \leq 1$ for which $\text{result}(c, c', x) = -y \leq 3$ and $\text{result}(c, c', y) = x \leq 4$. The resultant operator lifts to sets of inequalities:

Definition 9. If $I_1, I_2 \subseteq \mathbf{TVPI}$ then

$$\text{result}(I_1, I_2) = \{c \mid c_i \in I_i \wedge x \in \text{vars}(c_1) \cap \text{vars}(c_2) \wedge c = \text{result}(c_1, c_2, x) \neq \perp\}$$

Full completion Completing a set of inequalities I [17] amounts to repeatedly augmenting I with $\text{result}(I, I)$ until an I' is obtained such that no further (non-redundant) inequalities can be added to $\pi_Y(I')$ for any $Y \subseteq X$. During completion, the computation of resultants is interleaved with the removal of redundant inequalities. An inequality c is considered to be redundant in I iff $\pi_{\text{vars}(c)}(I) \setminus \{c\} \models c$, that is, c is redundant in its syntactic projection $\pi_{\text{vars}(c)}(I)$. To remove such constraints from I , the existence of an operator $\text{filter}_Y(I) = I'$ is assumed for each $Y \subseteq X$ such that $|Y| \leq 2$. The operator is assumed to satisfy the three conditions that $I' \subseteq I$, $I' \equiv I$ and $I'' \not\equiv I$ for all $I'' \subset I'$. Such an operator can be constructed straightforwardly, and resides in $O(|I|)$ when I is ordered [8, section 2]. With filter_Y in place, it is possible to filter an entire system $I \subseteq \text{Log}$ by computing $\text{filter}(I) = \cup \{\text{filter}_Y(\pi_Y(I)) \mid Y \subseteq X \wedge |Y| = 2\}$.

Definition 10. The (full) completion operator $\text{complete} : \wp(\text{Log}) \rightarrow \wp(\text{Log})$ is defined: $\text{complete}(I) = \cup_{i \geq 0} I_i$ where $I_0 = I$ and $I_{i+1} = \text{filter}(I_i \cup \text{result}(I_i, I_i))$.

Nelson [12], working over TVPI, used a divide and conquer argument to bound the number of iterations that need be computed before stability is achieved:

Lemma 1. $\text{complete}(I) = I_m$ where $m = \lceil \lg(|X|) \rceil$ and I_m is defined as above.

This result becomes more intriguing when the domain is Log_k . Completion can be calculated in a semi-naive fashion by defining $I_0 = I$ and $\delta_0 = I$ and computing $I_{i+1} = \text{filter}(I_i \cup \text{result}(\delta_i, I_i))$ and $\delta_{i+1} = I_{i+1} \setminus I_i$ for $i \in [0, m-1]$. Since $|\cup_{i=0}^m \delta_i|$ is in $O(|X|^2)$ it follows that the cumulative running time of $\text{result}(\delta_i, I_i)$ is $O(|X|^3)$. Since each invocation of filter resides in $O(|X|^2)$ and it is called $m-1$ times, it follows that the running time of completion is $O(|X|^3)$. Hence, like Oct , but unlike TVPI, Log_k comes with a (full) completion operation that resides in $O(|X|^3)$. It is conceivable that this result extends to other subclasses of TVPI that also retain closure under variable elimination.

Incremental completion During analysis inequalities are encountered one by one. Thus an important addition to full completion is *incremental completion* that takes a complete system, augments it with an additional inequality and returns the completion of the augmented system. Such an algorithm has been proposed for TVPI [15, Algorithm 7], together with a sketched correctness proof. Given the importance of completion, the following proposition, whose proof is given in [8], provides a rigorous foundation for an incremental algorithm.

Proposition 2. If $c' \in \text{Log}$, $I \subseteq \text{Log}$ is complete and $c \in \text{complete}(I \cup \{c'\})$ then one of the following holds:

- $c \in I \cup \{c'\}$
- $c \in \text{result}(c', c_0)$ where $c_0 \in I$
- $c \in \text{result}(\text{result}(c', c_0), \{c_1\})$ where $c_0, c_1 \in I$

$I \cup \{c'\}$ can thus be completed by computing $I_2 = \text{filter}(I_1 \cup \text{result}(I_1 \setminus \{c'\}, I_1 \setminus I))$ where $I_1 = \text{filter}(I \cup \{c'\} \cup \text{result}(I, \{c'\}))$. Nelson showed that if $J_1, J_2 \subseteq \text{TVPI}$ where $\text{vars}(J_1) = \{x, y\}$ and $\text{vars}(J_2) = \{y, z\}$ then $|\text{result}(J_1, J_2)| \leq 2|J_1| + 2|J_2|$ [12, section 3]. It follows that $|I_1| \leq 3|I| + 3$ and $|I_2| \leq 13|I| + 13$, thus although computing I_2 for **Log** takes $O(|I|^2)$ time it requires $O(|I|)$ space overall. For Log_k with fixed k , computing I_1 and I_2 both reside in $O(|X|^2)$. This squares with incremental closure for octagons which is also in $O(|X|^2)$.

4.2 Entailment

The value of completeness is that it simplifies other operations. To illustrate, consider the problem of detecting if a fixpoint has been reached, that is, deciding whether $I_1 \models I_2$ for $I_1, I_2 \in \wp^f(\text{Log})$. Suppose I_1 is complete. If $\text{false} \in I_1$ then it follows $I_1 \models I_2$. Otherwise $I_1 \models I_2$ iff $\pi_Y(I_1) \models \pi_Y(I_2)$ for all $Y \subseteq X$ and $|Y| = 2$. Moreover, recall that inequalities $ax + by \leq d$ are maintained in the form $a \in \{-1, 0, 1\}$ and suppose $b \in \{-1, 1\}$ if $a = 0$. Then the planar entailment check $\pi_Y(I_1) \models \pi_Y(I_2)$ can be decided by testing $\text{filter}_Y(\pi_Y(I_1) \cup \pi_Y(I_2)) = \pi_Y(I_1)$.

4.3 Variable Elimination

Variable elimination (projection) is required to remove out of scope variables, and all information pertaining to them, from a logahedral abstraction. Fourier-Motzkin can be applied to eliminate a variable x from $I \in \wp^f(\text{Log})$ which amounts to computing $\exists x.I = \cup\{c \mid c = \text{result}(c_1, c_2, x) \wedge c_1, c_2 \in I \wedge c \neq \perp\}$. However if I is complete then $\exists x.I = \cup\{\pi_Y(I) \mid Y \subseteq X \setminus \{x\} \wedge |Y| \leq 2\}$. If I were incomplete then $\exists x.I$ may lose some information as is witnessed by $I = \{w - x \leq 0, x - y \leq 0\}$. Then $\cup\{\pi_Y(I) \mid Y \subseteq X \setminus \{x\} \wedge |Y| \leq 2\} = \emptyset$ yet $\exists x.I \models w - y \leq 0$.

Projection also provides a way to realise translations induced by assignments of the form $x = x + c$ where $c \in \mathbb{Z}$. If I describes the state prior to the assignment, then the state after is described by $\exists x'.(\{x = x'\} \cup \exists x.(I \cup \{x' = x + c\}))$.

4.4 Abstraction

This section explains how to approximate a finite set of arbitrary Poly constraints by a finite set of **Log** constraints. Abstraction is employed as a component of join but is also used to translate program statements, for example, loop conditions, into logahedral inequalities. Approximation has two stages: projection onto planar sets of TVPI constraints and relaxation of these to logahedral constraints. Consider the latter step first, and suppose $I \subseteq \text{TVPI}$ is finite and that $\text{vars}(I) \subseteq Y$ where $|Y| = 2$. Suppose $I = \{c_0, \dots, c_{n-1}\}$ is non-redundant and ordered by orientation. This can be achieved in $O(|I| \lg |I|)$ time. Wlog each c_i is assumed to take the form $a_i x + b_i y \leq d_i$ where $a_i \in \{-1, 0, 1\}$. Let $c_1 \angle c_2$ be a predicate that holds when $a_1 b_2 - a_2 b_1 < 0$, that is, that c_1 is oriented before c_2 in a clockwise relative order by angle. Suppose also that $c_{-1} = c_{n-1}$ and $c_n = c_0$. If c_i and c_{i+1}

do not intersect at a vertex let $p_i = \perp$, otherwise let $p_i = (\psi_i, \phi_i)$ be this vertex, which can be calculated in constant time. If $b_i = 0$ then put $l_i = u_i = 0$, otherwise define $l_i = \text{sign}(b_i)2^{\lceil \lg(|b_i|) \rceil}$ and $u_i = \text{sign}(b_i)2^{\lfloor \lg(|b_i|) \rfloor}$. For each $i \in [0, n-1]$ put $c'_i = \perp$ if $p_{i-1} = \perp$ otherwise define

$$c'_i = \begin{cases} a_i x + u_i y \leq a_i \psi_{i-1} + u_i \phi_{i-1} & \text{if } a_i = \text{sign}(b_i) \\ a_i x + l_i y \leq a_i \psi_{i-1} + l_i \phi_{i-1} & \text{if } a_i \neq \text{sign}(b_i) \end{cases}$$

Likewise if $p_i = \perp$ put $c''_i = \perp$ otherwise define

$$c''_i = \begin{cases} a_i x + l_i y \leq a_i \psi_i + l_i \phi_i & \text{if } a_i = \text{sign}(b_i) \\ a_i x + u_i y \leq a_i \psi_i + u_i \phi_i & \text{if } a_i \neq \text{sign}(b_i) \end{cases}$$

The $a_i = \text{sign}(b_i)$ test determines whether the inequalities resulting from the upper and lower approximations of $|b_i|$ support p_{i-1} or p_i . Some of the logahedral constraints c'_i, c''_i may be too strong in that $I \not\models c'_i$ or $I \not\models c''_i$ and these, with the \perp constraints, are filtered out to abstract I as follows:

Definition 11. Let $\alpha_Y(I) = \{c'_i \mid c'_i \neq \perp \wedge c_{i-1} \angle c'_i\} \cup \{c''_i \mid c''_i \neq \perp \wedge c''_i \angle c_{i+1}\}$ where c'_i, c''_i are defined as above for finite $I \subseteq \text{TVPI}$, $\text{vars}(I) \subseteq Y$ and $|Y| = 2$.

Note that the filtering test amounts to an $O(1)$ orientation check. The angular test guarantees that the retained inequalities are supporting.

Example 4. The example illustrates the approximation of two constraints from a larger system I . Suppose $c_{i-1} := -x \leq 0$, $c_i := -x + (3/2)y \leq 0$, $c_{i+1} := -x + (5/2)y \leq 2$ and $p_{i+1} = \perp$. The logahedral approximation is illustrated in Fig. 3. Observe $p_{i-1} = (0, 0)$ and $p_i = (3, 2)$. Since $l_i = 2^{\lceil \lg(3/2) \rceil} = 1$, $u_i = 2^{\lfloor \lg(3/2) \rfloor} = 2$, $a_i \neq \text{sign}(b_i)$, $p_{i-1} \neq \perp$ and $p_i \neq \perp$ it can be seen that $c'_i := -x + y \leq 0$ and $c''_i := -x + 2y \leq 1$. Moreover, since $l_{i+1} = 2^{\lceil \lg(5/2) \rceil} = 2$, $u_{i+1} = 2^{\lfloor \lg(5/2) \rfloor} = 4$, $a_{i+1} \neq \text{sign}(b_{i+1})$, $p_i \neq \perp$ and $p_{i+1} = \perp$ it follows $c'_{i+1} := -x + 2y \leq 1$ and $c''_{i+1} := \perp$.

Now suppose instead, $c_{i+1} := -x + (7/4)y \leq 1/2$, which preserves $p_i = (3, 2)$. Then $I \not\models c''_i$ and c''_i is not an approximating constraint. But $c''_i \angle c_{i+1}$ does not hold since $(-1.7/4) - (-1.2) > 0$, hence $c''_i \notin \alpha_Y(I)$. Likewise $c'_{i+1} \notin \alpha_Y(I)$.

Although α_Y is partial, it can be used to abstract arbitrary TVPI systems:

Definition 12. The abstraction map $\alpha : \wp^f(\text{TVPI}) \rightarrow \wp^f(\text{Log})$ is given by $\alpha(I) = \cup\{\alpha_Y(\exists Y.I) \mid Y \subseteq X \wedge |Y| = 2\}$

In the above $\exists Y.I$ denotes project I onto Y , that is, the repeated application of Fourier-Motzkin to elimination of all variables $x \in X \setminus Y$ from I . However, if I is complete then $\alpha(I) = \cup\{\pi_Y(I) \mid Y \subseteq X \wedge |Y| = 2\}$. Interestingly, α can be immediately lifted to $\alpha : \wp^f(\text{Poly}) \rightarrow \wp^f(\text{Log})$ since if I is a finite set then $\exists Y.I$ is a finite set. The symbol α hints at the existence of a Galois connection between $\langle \wp^f(\text{Poly}), \models \rangle$ and $\langle \wp^f(\text{Log}), \models \rangle$, and indeed α is monotonic. But such a structure can only be obtained by quotienting $\wp^f(\text{Poly})$ and $\wp^f(\text{Log})$ by \equiv to obtain posets. The upper adjoint of α is the identity.

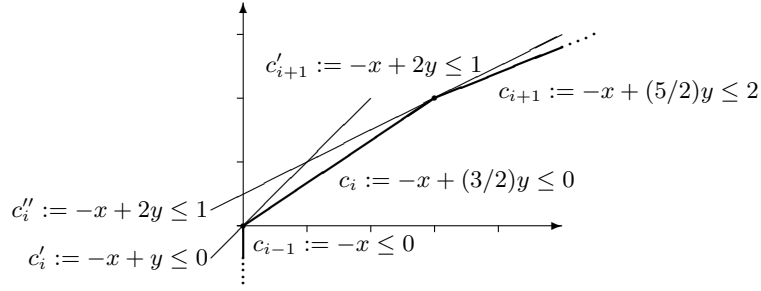


Fig. 3. Approximation with logahedral constraints

To abstract the inequalities I or $I \cup \text{Box}_k$ to Log_k put $c'_i := \text{sign}(b_i)y \leq \text{sign}(b_i)\phi_{i-1}$ and $c''_i := a_i x + \text{sign}(b_i)2^k y \leq a_i \psi_i + \text{sign}(b_i)2^k \phi_i$ if $a_i = \text{sign}(b_i)$ and $k < \lceil \lg(|b_i|) \rceil$. Likewise, if $\lfloor \lg(|b_i|) \rfloor < -k$ then put $c'_i := a_i x + \text{sign}(b_i)2^{-k} y \leq a_i \psi_{i-1} + \text{sign}(b_i)2^{-k} \phi_{i-1}$ and $c''_i := a_i x \leq a_i \psi_i$. Similarly for $a_i \neq \text{sign}(b_i)$.

4.5 Meet and Abstraction

Meet over $\langle \wp^f(\text{Log}), \models \rangle$ can be defined by $[I_1]_{\equiv} \sqcap [I_2]_{\equiv} = [I_1 \cup I_2]_{\equiv}$. Thus, meet reduces to set union when a class $[I]_{\equiv}$ is represented by a set I . Henceforth quotienting will be omitted, for brevity, and to reflect implementation.

A less trivial problem is that of computing $\alpha(I_1 \cup I_2)$ where $I_1 \in \wp^f(\text{Log})$ and $I_2 \in \wp^f(\text{Poly})$. This arises as a special case when a statement is encountered, such as a loop condition $x \leq y+z$ or an assignment $x = y+z$, that cannot be expressed logahedrally. If these statements are described respectively by $I_2 = \{x \leq y+z\}$ and $I_2 = \{y+z \leq x, x \leq y+z\}$, and I_1 is a logahedral description of the program state, then the subsequent state is described by $\alpha(I_1 \cup I_2)$. Thus, in an analysis, $\alpha(I_1 \cup I_2)$ performs the role of meet when I_2 is not logahedral.

This problem has been tackled using linear programming [15]. To illustrate, suppose $I_2 = \{c'\}$ where $c' := \sum_{k=1}^n a'_k x_k \leq d'$. Let $i, j \in [1, n]$ where $i \neq j$ and minimise $\sum_{k \neq i, k \neq j} a'_k x_k$ subject to I_1 and c' . If a minimum $d_{i,j}$ exists then it follows $I_1 \cup \{c'\} \models a'_i x_i + a'_j x_j \leq d' - d_{i,j}$. TVPI inequalities found this way are added to I_1 . An alternative and potentially more precise approach, which is applicable to both TVPI and logahedra, is based on extending the resultant operator to polyhedral inequalities in the following fashion:

Definition 13. If $c := a_i x_i + a_j x_j \leq d \in \text{TVPI}$, $c' := \sum_{k=1}^n a'_k x_k \leq d' \in \text{Poly}$, $a'_j \neq 0$ and $a_i a'_i < 0$ then $\text{result}(c, c', x_i) = (|a'_i| a_j + |a_i| a'_j) x_j + |a_i| \sum_{k \notin \{i,j\}} a'_k x_k \leq |a'_i| d + |a_i| d'$ otherwise $\text{result}(c, c', x_i) = \perp$.

The $a'_j \neq 0$ condition ensures $|\text{vars}(\text{result}(c, c', x_i))| < |\text{vars}(c')|$. Thus if c' is ternary then $\text{result}(c, c', x_i)$ is either binary or undefined. The partial result map can be lifted to a total map $\text{result}(I_1, I_2)$ for $I_1 \subseteq \text{TVPI}$ and $I_2 \subseteq \text{Poly}$ by analogy with definition 9. With this in place, an operator $\text{extend}(I_1, I_2)$ is introduced, designed so that $\alpha(I_1 \cup I_2) \models \text{extend}(I_1, I_2)$.

Definition 14. The map $\text{extend} : \wp^f(\text{TVPI}) \times \wp^f(\text{Poly}) \rightarrow \wp^f(\text{TVPI})$ is defined $\text{extend}(I, \delta) = \cup_{i \geq 0} I_i$ where $I_0 = \text{complete}(I \cup (\delta \cap \text{TVPI}))$, $\delta_0 = \delta \setminus \text{TVPI}$, $R_{i+1} = \text{result}(I_i, \delta_i)$, $I_{i+1} = \text{complete}(I_i \cup (R_{i+1} \cap \text{TVPI}))$ and $\delta_{i+1} = R_{i+1} \setminus I_{i+1}$.

If δ is comprised of ternary constraints, which is the dominating case [15], then result is applied once. If desired, incremental closure can be used to compute $\text{complete}(I_i \cup (R_{i+1} \cap \text{TVPI}))$, since proposition 2 extends to TVPI [8]. If $\text{extend}(I, \delta)$ is not logahedral, then α must subsequently be applied.

Example 5. Consider augmenting $I = \{x - y \leq 0, -x + y \leq 0\}$ with $c' := x - 2y + z \leq 0$. Then $I_0 = I$ and $\delta_0 = \{c'\}$. Thus $R_1 = \text{result}(I_0, \delta_0) = \{y + z \leq 0\}$ whence $\delta_1 = \emptyset$ and $I_1 = \text{complete}(I_0 \cup R_1) = I \cup \{y + z \leq 0, x + z \leq 0\}$. These inequalities cannot be inferred with the linear programming technique of [15].

Example 6. Let $I = \{w - x \leq 0, y - z \leq 0\}$ and $c' := w + x + y + z \leq 1$. Then $I_0 = \text{complete}(I) = I$ and $\delta_0 = \{c'\}$. Thus $R_1 = \text{result}(I_0, \delta_0) = \{2w + y + z \leq 1, x + w + 2y \leq 1\}$, $I_1 = I_0$ and $\delta_1 = R_1$. Next $R_2 = \text{result}(I_1, \delta_1) = \{2w + 2y \leq 1\}$, $I_2 = \text{complete}(I_1 \cup R_2) = I_1 \cup R_2 = I \cup \{w + y \leq 1/2\}$ and $\delta_2 = \emptyset$.

4.6 Join

Join is required to merge abstractions on different paths, as is shown with P_5 in section 3. If quotienting is omitted, then join over $\wp^f(\text{Log})$ can be defined $I_1 \sqcup I_2 = \cup \{\exists Y. I_1 \sqcup_Y I_2 \mid Y \subseteq X \wedge |Y| = 2\}$ where $J_1 \sqcup_Y J_2 = \alpha_Y(J_1 \vee J_2)$ and \vee is join (planar convex hull) for TVPI [15]. Join also benefits from completeness since if I_i is complete then $\exists Y. I_i = \pi_Y(I_i)$. The TVPI operation \vee is $O(n \lg n)$ where $n = |J_1| + |J_2|$, hence the overall cost of \sqcup is dominated by completion.

With \sqcap and \sqcup thus defined, $(\wp^f(\text{Log}), \models, \sqcap, \sqcup)$ forms a lattice. Furthermore, Log_k has additional structure since C_k is finite (see definition 5). In particular, if $I \subseteq \text{Log}_k$ then there exists a finite $K \subseteq I$ such that $I \equiv K$. As a consequence $(\wp^f(\text{Log}_k), \models, \sqcap, \sqcup)$ is a complete lattice.

Widening [6] is often applied with join in order to enforce termination. As has been explained elsewhere [11, 15], care must be taken not to reintroduce inequalities through completion that have been deliberately discarded in widening.

5 Logahedra versus Octagons and TVPI

Logahedra are theoretically well motivated and their representational advantages, as a generalisation of octagons, are of interest. To provide preliminary data on the power of bounded logahedra two sets of experiments were performed.

In the first experiment sets of integer points, $\{(x, y) \mid x, y \in [-32, 32]\}$ were randomly selected. For each set, between 1 and 63 points were generated. The best $\text{Oct} = \text{Log}_0$ and Log_k (for $k \in [1, 5]$) abstractions were computed and compared with the best TVPI abstraction. The comparison is based on the number of integer points in the abstractions. For example, one set of 23 points had 7 extreme points. The Oct , the five Log and the TVPI descriptions were satisfied

<i>vertices</i>	<i>sets</i>	Oct	Log_k				
			k=1	k=2	k=3	k=4	k=5
1	2044	0.000	0.000	0.000	0.000	0.000	0.000
2	2958	82.640	43.941	30.675	26.396	25.024	24.481
3	5923	1.874	0.998	0.700	0.611	0.584	0.576
4	10423	0.557	0.294	0.195	0.163	0.153	0.149
5	14217	0.352	0.192	0.125	0.100	0.092	0.089
6	13550	0.276	0.152	0.097	0.075	0.067	0.064
7	9058	0.234	0.131	0.081	0.061	0.054	0.051
8	4345	0.205	0.115	0.071	0.053	0.046	0.043
9	1508	0.188	0.105	0.064	0.047	0.040	0.038
10	398	0.171	0.096	0.058	0.042	0.035	0.033
11	64	0.165	0.095	0.054	0.037	0.031	0.029
12	6	0.179	0.107	0.06	0.045	0.038	0.036

Table 1. Precision comparison of Oct and Log_k against TVPI on random data

by 3221, 3027, 2881, 2843, 2835, 2819, 2701 integer points. Thus the precision loss incurred by Oct over TVPI is $(3321 - 2701)/2701 = 0.230$. Likewise the losses for Log relative to TVPI are 0.121, 0.067, 0.053, 0.050 and 0.044. This was done for 64K sets and the results are summarised in Table 1. The table presents the mean precision loss where the sets are grouped by their number of vertices.

The data reveals that describing exactly two points is by far the most inaccurate scenario for Oct and Log_k ; if the angle between the points is suitably acute then the TVPI constraints are satisfied by just the two points whereas the Oct and Log_k constraints can be satisfied by a band of integral points. Precision loss decreases beyond this two point case. Enlarging the sample space does not noticeably effect the relative precision loss other than accentuating the two point case. Observe that the relative precision loss declines as k increases, suggesting that logahedra can offer a significant precision improvement over octagons.

The second set of experiments compares the abstractions of two variable projections of the results of a polyhedral analysis tightened to integer points for a series of benchmarks [1, 2, 13]. As in the first experiment, comparisons are made between the number of points in the TVPI, Oct and Log_k abstractions. Table 2 details benchmarks, their number of variables, the pair of variables in the projection, the number of integer points in the TVPI abstraction and the number of additional points for Oct and Log_k abstractions. Benchmarks and projections where Oct and TVPI give the same abstraction are omitted from the table. This was the case for the additional 12 programs in the benchmark suite.

The data illustrates the power of octagons, with the majority of two variable projections being octagonal (indeed, many of these were intervals). It also illustrates that there are cases when non-octagonal constraints occur, with (as in the first experiment) Log_k precision improving as k increases.

Together, the results motivate further study. The random data demonstrates that logahedra have the potential to deliver precision gains over octagons, how-

<i>fixpoint vars project</i>	Oct	Log_k					TVPI	
		k=1	k=2	k=3	k=4	k=5		
<i>cars.inv</i>	5 (4, 5)	312	234	78	54	54	54	3640
<i>efm1.inv</i>	6 (3, 6)	1	0	0	0	0	0	128
<i>heap.inv</i>	5 (1, 2)	1056	0	0	0	0	0	33
<i>heap.inv</i>	5 (3, 4)	465	0	0	0	0	0	1055
<i>heap.inv</i>	5 (3, 5)	465	0	0	0	0	0	1055
<i>robot.inv</i>	3 (1, 2)	528	0	0	0	0	0	1089
<i>scheduler-2p.invl1</i>	7 (3, 6)	135	120	90	30	18	18	180
<i>scheduler-2p.invl1</i>	7 (4, 6)	115	100	70	20	12	12	260
<i>scheduler-2p.invl1</i>	7 (6, 7)	135	120	90	30	18	18	180
<i>scheduler-2p.invl2</i>	7 (4, 6)	189	168	126	42	26	26	245
<i>scheduler-2p.invl2</i>	7 (6, 7)	90	80	60	20	12	12	215
<i>scheduler-3p.invl1</i>	10 (4, 9)	264	198	66	45	45	45	390
<i>scheduler-3p.invl1</i>	10 (9, 10)	264	198	66	45	45	45	390
<i>scheduler-3p.invl3</i>	10 (4, 8)	312	234	78	54	54	54	455
<i>scheduler-3p.invl3</i>	10 (8, 9)	144	108	36	24	24	24	405
<i>scheduler-3p.invl3</i>	10 (9, 10)	534	128	128	128	128	128	1725
<i>see-saw.inv</i>	2 (1, 2)	990	231	110	110	110	110	2454
<i>swim-pool-1.inv</i>	9 (4, 6)	62	61	59	55	47	31	4162

Table 2. Precision comparison of Oct and Log_k against TVPI on analysis data

ever, the analysis data adds a note of caution. It is not surprising that many program invariants can be described by intervals, nor that many of the remainder are octagonal. The data (and the example in section 3) shows that the descriptive power of logahedra can improve analysis, leaving open the question of whether the additional cost (a higher constant in the complexity) pays for itself with improved accuracy. The answer to this question is in part application specific.

6 Conclusion

This paper has introduced logahedra, a new weakly relational abstract domain. A variant of logahedra, bounded logahedra Log_k , where k is the maximum exponent is also introduced. Logahedra lie strictly between octagons and TVPI in terms of expressive power, with octagons forming a special case, $\text{Oct} = \text{Log}_0$. Bounded logahedra retain the good computational properties of octagons, whilst being less restrictive. The theory for the abstract domain has been given, along with algorithms for each domain operation. Logahedra have the further advantage that their variable coefficients can be represented by their exponents, mitigating the problem of large coefficients that arise when using polyhedra or TVPI. A preliminary investigation into the expressive power of logahedra, plus a worked example, suggests that they can lead to more accurate analysis, but cautions that many invariants can be described using intervals and octagons. Future work will further investigate the application of logahedra in verification.

Acknowledgements The authors thank Phil Charles, Tim Hopkins, Stefan Kahrs and Axel Simon for useful discussions. The work was supported by EPSRC projects EP/E033105/1 and EP/E034519/1. The authors would like to thank the University of St Andrews and the Royal Society for their generous support.

References

1. S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: Fast Acceleration of Symbolic Transition Systems. In *Computer Aided Verification*, volume 2725 of *LNCS*, pages 118–121. Springer, 2003.
2. P. J. Charles, J. M. Howe, and A. King. Integer Polyhedra for Program Analysis. In *AAIM*, volume 5564 of *LNCS*, pages 85–99. Springer, 2009.
3. R. Clarisó and J. Cortadella. The Octahedron Abstract Domain. *Science of Computer Programming*, 64:115–139, 2007.
4. P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints among Variables of a Program. In *POPL*, pages 84–97. ACM Press, 1978.
5. F. Eisenbrand and S. Laue. A Linear Algorithm for Integer Programming in the Plane. *Mathematical Programming*, 102(2):249–259, 2005.
6. N. Halbwachs. *Détermination Automatique de Relations Linéaires Vérifiées par les Variables d'un Programme*. PhD thesis, Université Scientifique et Médicale de Grenoble, 1979.
7. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A Model Checker for Hybrid Systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.
8. J. M. Howe and A. King. Closure Algorithms for Domains with Two Variables per Inequality. Technical Report TR/2009/DOC/01, School of Informatics, City University London, 2009. http://www soi.city.ac.uk/organisation/doc/research/tech_reports/.
9. K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient Verification of Real-time Systems: Compact Data Structure and State-space Reduction. In *IEEE Real-Time Systems Symposium*, pages 14–24. IEEE Computer Society, 1997.
10. F. Logozzo and M. Fähndrich. Pentagons: a Weakly Relational Abstract Domain for the Efficient Validation of Array Accesses. In *ACM Symposium on Applied Computing*, pages 184–188. ACM Press, 2008.
11. A. Miné. The Octagon Abstract Domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
12. C. G. Nelson. An $n^{lg n}$ Algorithm for the Two-Variable-Per-Constraint Linear Programming Satisfiability Problem. Technical Report STAN-CS-78-689, Stanford University, Computer Science Department, 1978.
13. S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Constraint Based Linear Relations Analysis. In *SAS*, volume 3148 of *LNCS*, pages 53–68. Springer, 2004.
14. S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Scalable Analysis of Linear Systems Using Mathematical Programming. In *Verification, Model Checking, and Abstract Interpretation*, volume 3385 of *LNCS*, pages 25–41. Springer, 2005.
15. A. Simon. *Value-Range Analysis of C Programs: Towards Proving the Absence of Buffer Overflow Vulnerabilities*. Springer, 2008.
16. A. Simon and A. King. Exploiting Sparsity in Polyhedral Analysis. In C. Hankin, editor, *SAS*, volume 3672 of *LNCS*, pages 336–351. Springer, 2005.
17. A. Simon, A. King, and J. M. Howe. Two Variables per Linear Inequality as an Abstract Domain. In M. Leuschel, editor, *Logic Based Program Development and Transformation*, volume 2664 of *LNCS*, pages 71–89. Springer, 2002.