Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 307 (2014) 3–15

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Efficient Constraint/Generator Removal from Double Description of Polyhedra

Gianluca Amato[1] Francesca Scozzari[2]

Department of Economic Studies, University of Chieti-Pescara, Italy

Enea Zaffanella[3]

Department of Mathematics and Computer Science, University of Parma, Italy

Abstract

We present an algorithm for the removal of constraints (resp., generators) from a convex polyhedron represented in the Double Description framework. Instead of recomputing the dual representation from scratch, the new algorithm tries to better exploit the information available in the Double Description pair, so as to capitalize on the computational work already done. A preliminary experimental evaluation shows that significant efficiency improvements can be obtained. In particular, a combined algorithm can be defined that dynamically selects whether or not to apply the new algorithm, based on suitable profitability heuristics.

Keywords: convex polyhedra, Double Description, incremental computation

1 Introduction

The domain of convex polyhedra [11] has been widely adopted in applications for the static analysis and verification of hardware/software systems [7] leading to the specification of many operators that are meant to compute (or approximate in a safe way) the effects of a semantic operation affecting the state of the system. When considering the Double Description (DD) framework, these operators can be implemented in most cases (intersection, convex polyhedral hull, time elapse, projection, ...) by adding some new constraints or some new generators to the available descriptions, thereby directly exploiting the incremental nature of Chernikova's conversion procedure [8,9,10]. In other cases (e.g., invertible affine images and preimages) it is

[1] Email: gamato@unich.it

[2] Email: fscozzari@unich.it

[3] Email: enea.zaffanella@unipr.it

http://dx.doi.org/10.1016/j.entcs.2014.08.002

even possible to directly and efficiently modify both the constraint and the generator representations, so as to fully preserve the computational work already done. There are cases, however, when an operator on polyhedra is defined by removing some constraints or generators, so that the information content of the other representation is no longer up-to-date and not easily recoverable. The usual approach is to simply discard the other representation and, if later needed, recompute it from scratch. In this paper we propose and experimentally evaluate a new algorithm for removing constraints (or generators) that is meant to better exploit the information already available in the input DD pair.

The paper is structured as follows: Section 2, besides recalling a few concepts and notations, briefly presents the DD method; Section 3 introduces the constraint removal operation, defines the new algorithm and shows its equivalence with respect to the executable specification; Section 4 provides an experimental evaluation and proposes an integration of the new algorithm with the old one based on profitability heuristics; we conclude in Section 5 by discussing potential applications of constraint/generator removal and the extension of the removal operation to the case of NNC polyhedra.

2 Preliminaries

The scalar product of two vectors $\boldsymbol{a}_1, \boldsymbol{a}_2 \in \mathbb{R}^n$ is denoted by $\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{a}_2$. For each vector $\boldsymbol{a} \in \mathbb{R}^n$ and scalar $b \in \mathbb{R}$, where $\boldsymbol{a} \neq \boldsymbol{0}$, the linear non-strict inequality constraint $c = (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} \geq b)$ defines a topologically closed affine half-space of $\mathbb{R}^n$. A topologically closed, convex polyhedron (for short, polyhedron) is defined by a finite system of linear non-strict inequality constraints. If a polyhedron $\mathcal{P}$ is contained in both half-spaces $c = (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} \geq b)$ and $c^- = (-\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} \geq -b)$ then we say that c is a *singular* constraint for $\mathcal{P}$ and write $c \in \mathrm{eq}(\mathcal{P})$. We write $\mathrm{con}(\mathcal{C})$ to denote the polyhedron $\mathcal{P} \subseteq \mathbb{R}^n$ described by the finite *constraint system* $\mathcal{C}$. Formally, we define

$$\mathcal{P} = \mathrm{con}(\mathcal{C}) := \left\{ \boldsymbol{p} \in \mathbb{R}^n \;\middle|\; \forall c = (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} \geq b) \in \mathcal{C} : \boldsymbol{a}^{\mathrm{T}}\boldsymbol{p} \geq b \right\}.$$

A vector $\boldsymbol{r} \in \mathbb{R}^n$ such that $\boldsymbol{r} \neq \boldsymbol{0}$ is a *ray* of a non-empty polyhedron $\mathcal{P} \subseteq \mathbb{R}^n$ if, for every point $\boldsymbol{p} \in \mathcal{P}$ and every non-negative scalar $\rho \in \mathbb{R}_+$, it holds $\boldsymbol{p} + \rho\boldsymbol{r} \in \mathcal{P}$. The empty polyhedron has no rays. If both $\boldsymbol{r}$ and $-\boldsymbol{r}$ are rays of $\mathcal{P}$, then we say that $\boldsymbol{r}$ is a *singular* ray (or line) of $\mathcal{P}$ and write $\boldsymbol{r} \in \mathrm{lines}(\mathcal{P})$. By Minkowski and Weyl theorems [18], the set $\mathcal{P} \subseteq \mathbb{R}^n$ is a polyhedron if and only if there exist finite sets $R, P \subseteq \mathbb{R}^n$ of cardinality r and p, respectively, such that $\boldsymbol{0} \notin R$ and

$$\mathcal{P} = \mathrm{gen}\big((R, P)\big) := \left\{ R\boldsymbol{\rho} + P\boldsymbol{\pi} \in \mathbb{R}^n \;\middle|\; \boldsymbol{\rho} \in \mathbb{R}_+^r, \boldsymbol{\pi} \in \mathbb{R}_+^p, \sum_{i=1}^{p} \pi_i = 1 \right\}.$$

When $\mathcal{P} \neq \varnothing$, we say that $\mathcal{P}$ is described by the *generator system* $\mathcal{G} = (R, P)$: the vectors of R and P are rays and points of $\mathcal{P}$, respectively.

The Double Description method due to Motzkin et al. [17], by exploiting the duality principle, allows for a combination of the two approaches outlined above:

a *conversion* procedure computes each representation starting from the other one. If $\mathcal{P} = \mathrm{con}(\mathcal{C}) = \mathrm{gen}(\mathcal{G})$, then we say that $(\mathcal{C}, \mathcal{G})$ is a *DD pair* for polyhedron $\mathcal{P}$. A DD pair is in minimal form if both $\mathcal{C}$ and $\mathcal{G}$ are minimal, i.e., they contain no redundant element. [4] In a DD pair $(\mathcal{C}, \mathcal{G})$ in minimal form for polyhedron $\mathcal{P}$, each non-singular constraint $c = (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} \geq b) \in \mathcal{C}$ defines a facet of $\mathcal{P}$ given by $\mathcal{F} := \{\, \boldsymbol{p} \in \mathcal{P} \mid \boldsymbol{a}^{\mathrm{T}}\boldsymbol{p} = b \,\}$. We say that the non-singular constraints $c, c' \in \mathcal{C}$ are *adjacent* in $\mathcal{P}$, denoted $\mathrm{adjacent}_{\mathcal{P}}(c, c')$, if the corresponding facets are adjacent. Adjacency between faces is defined in [3].

The conversion procedure, denoted $(\mathcal{C}_{out}, \mathcal{G}_{out}) \leftarrow \mathrm{conversion}(\mathcal{C}_{in})$, maps an input constraint system $\mathcal{C}_{in}$ into an output DD pair $(\mathcal{C}_{out}, \mathcal{G}_{out})$ in minimal form: starting from an initial DD pair $(\mathcal{C}_{univ}, \mathcal{G}_{univ})$ representing the whole vector space, the procedure incrementally adds each of the constraints in $\mathcal{C}_{in}$ as described above. We will write $(\mathcal{C}_{min}, \mathcal{G}_{min}) \leftarrow \mathrm{simplify}(\mathcal{C}, \mathcal{G})$ to denote the simplification step, which enforces the minimal form by removing redundancies from the input DD pair $(\mathcal{C}, \mathcal{G})$. The algorithm for incremental constraint addition (and the conversion procedure) can be adapted to handle the dual case, when adding a generator to a DD pair.

2.1 Low Level Encoding of Polyhedra

In the DD framework, polyhedra in $\mathbb{R}^n$ are generally mapped to polyhedral cones in $\mathbb{R}^{n+1}$ via homogenization: the known term of constraints is associated to an extra space dimension ξ and the *positivity* constraint $pos = (\xi \geq 0)$ is added. Homogenization allows for a more uniform handling of constraints and generators (for instance, all points of the polyhedron become rays of the cone). The basic step of conversion procedures based on Chernikova's algorithm [8,9,10] is the *incremental addition* of a new homogeneous constraint $c = (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} \geq 0)$ to a DD pair $(\mathcal{C}, \mathcal{G})$ describing a polyhedral cone $\mathcal{P}$. The set of rays $\mathcal{G}$ is partitioned into three components $\mathcal{G}^+$, $\mathcal{G}^0$, $\mathcal{G}^-$, based on the sign of the scalar product of the rays with constraint c (those in $\mathcal{G}^0$ are the *saturators* of constraint c); the new generator system is computed as $\mathcal{G}' := \mathcal{G}^+ \cup \mathcal{G}^0 \cup \mathcal{G}^\star$, where

$$\mathcal{G}^\star := \left\{\, (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{r}^+)\boldsymbol{r}^- - (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{r}^-)\boldsymbol{r}^+ \;\middle|\; \boldsymbol{r}^+ \in \mathcal{G}^+, \boldsymbol{r}^- \in \mathcal{G}^-, \mathrm{adjacent}_{\mathcal{P}}(\boldsymbol{r}^+, \boldsymbol{r}^-) \,\right\}.$$

The definition of adjacency for rays is obtained from that for constraints, by exploiting duality. Implementations adopt different strategies for the computation of the adjacency relation and for detecting redundancies: a common approach is to systematically maintain *saturation* information [13,15,19].

In the following, to simplify exposition, we will specify and describe the algorithms in terms of polyhedra, so that the mapping to polyhedral cones and the special handling of the positivity constraint will be transparent.

[4] Actual implementations are usually based on a stronger minimality concept, taking into special account singular constraints and generators (i.e., equalities and lines).

3 Constraint/Generator Removal for the DD Method

In this section we propose a new algorithm for removing a set of constraints from a DD pair in minimal form. Note that the same algorithm can be easily adapted, by exploiting duality properties of polyhedra representations, to the case of the removal of a set of generators.

A first observation that should be made is that there is no well defined notion of removing a singular constraint from a polyhedron, as shown by the following example.

Example 3.1 Consider the polyhedron $\mathcal{P} = \{\mathbf{0}\} \subseteq \mathbb{R}^2$ (the origin of the two dimensional vector space). This polyhedron can be described by the constraint systems $\mathcal{C}_1 = \{x = 0, y = 0\}$ and $\mathcal{C}_2 = \{x = 0, x + y = 0\}$, which are both in minimal form. Depending on the chosen syntactic representation, the removal of constraint $x = 0$ leads to the computation of two different polyhedra.

To avoid the problem above, in the following we will only consider the removal of non-singular constraints, i.e., we will assume that all the equality constraints in the input DD pair in minimal form are left untouched. Note that, in many practical contexts, such an assumption is plainly justified; for instance, for the case of the widening operation on polyhedra, variants of the standard widening have been proposed where the more precise polyhedral convex hull is used whenever there is a change in the affine dimension of the polyhedron [4].

The straightforward approach (see Algorithm 1) to implement constraint removal requires the computation of a generator system from the set $\mathcal{C}_{kept}$ of constraints that have not been removed, using the conversion procedure by Chernikova. While being based on what was meant to be an incremental algorithm, this approach recomputes the new generator system from scratch, completely disregarding the generator system component of the input DD pair. In the following we will refer to Algorithm 1 as the *naive algorithm*.

Algorithm 1 Naive removal of a set of constraints

Require: a DD pair $(\mathcal{C}_{in}, \mathcal{G}_{in})$ in minimal form defining $\mathcal{P}_{in} \neq \varnothing$;
Require: a set $\mathcal{C}_{rem} \subseteq \mathcal{C}_{in}$ of constraints such that $\mathcal{C}_{rem} \cap \mathrm{eq}(\mathcal{P}_{in}) = \varnothing$.
Ensure: a DD pair $(\mathcal{C}_{out}, \mathcal{G}_{out})$ in minimal form defining $\mathcal{P}_{out} = \mathrm{con}(\mathcal{C}_{in} \setminus \mathcal{C}_{rem})$.
Begin
$\mathcal{C}_{kept} \leftarrow \mathcal{C}_{in} \setminus \mathcal{C}_{rem}$
$(\mathcal{C}_{out}, \mathcal{G}_{out}) \leftarrow \mathrm{conversion}(\mathcal{C}_{kept})$
End

The goal of the new algorithm is to exploit, as far as possible, the information encoded in the input DD pair, so as to capitalize on the computational work already done. For this reason, in the following we will refer to it as the *incremental algorithm*. To get an idea on how this incremental algorithm should work, let us consider as simple examples the two polyhedra in the left hand side of Figure 1. First consider polyhedron $\mathcal{P}_1$, which is a trapezium whose DD pair is composed by

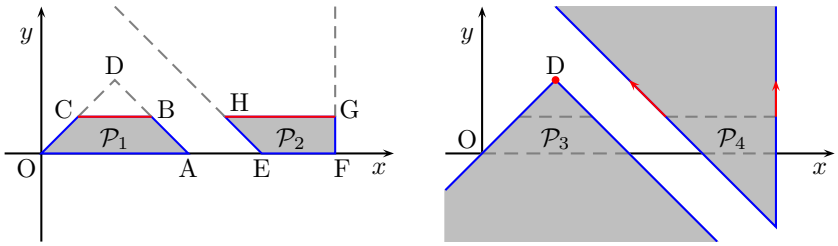

Fig. 1. Examples of constraint removal

four constraints and four vertices. Suppose that we need to remove the constraint corresponding to facet BC, so as to obtain the triangle OAD. When reasoning in terms of generators, the removal of BC corresponds to the addition of vertex D (which also causes vertices B and C to become redundant). In particular, vertex D is a generator obtained by combining the constraints that define facets OC and AB, which are those constraints that are adjacent to the one being removed; also, the generator D violates the constraint being removed. Now consider trapezium $\mathcal{P}_2$ and suppose we need to remove the constraint corresponding to facet GH; by doing this, we obtain an unbounded polyhedron described by two vertices (E and F) and two rays (along the directions EH and FG). When reasoning in terms of generators, we need to add these two rays (which also causes vertices H and G to become redundant). As before, the two rays can be seen as originating from the constraints adjacent to the one being removed (facets EH and FG); also, they were violating the removed constraint.

The observations made when discussing the examples in Figure 1 lead to the specification of Algorithm 2. Here we first select in $\mathcal{C}_{adj}$ those constraints that are adjacent to at least one of the constraints being removed: these constraints are added to the singular ones to form constraint system $\mathcal{C}_{conv}$, which is fed to the Chernikova's conversion procedure, obtaining the generator system $\mathcal{G}_{conv}$. Then, we select into $\mathcal{G}_{add}$ the generators that violate a constraint being removed:[5] these are added to the input generator representation $\mathcal{G}_{in}$ to obtain a generator representation for the output polyhedron. Finally, the DD pair is put in minimal form by procedure 'simplify'.

The following result states the equivalence of the two algorithms.

Theorem 3.2 (Algorithm 2 is correct) *The DD pairs computed by Algorithm 1 and Algorithm 2 represent the same polyhedral cone.*

Compared to Algorithm 1, Algorithm 2 requires an application of the conversion procedure too, but this is applied to a potentially smaller description ($\mathcal{C}_{conv}$): depending on the input, this could result in significant efficiency gains, although there are corner cases which result in a loss of efficiency. Note that the cost of the call to 'simplify' is usually dominated by the computation of $\mathcal{G}_{conv}$.

[5] In Figure 1, the polyhedron described by $(\mathcal{C}_{conv}, \mathcal{G}_{conv})$ for $\mathcal{P}_1$ (resp., $\mathcal{P}_2$) is shown on the right hand side as $\mathcal{P}_3$ (resp., $\mathcal{P}_4$); the generators in $\mathcal{G}_{add}$ are highlighted.

Algorithm 2 Incremental removal of a set of constraints

Require: same as for Algorithm 1.
Ensure: same as for Algorithm 1.
Begin
$\mathcal{C}_{eq} \leftarrow \mathcal{C}_{in} \cap \mathrm{eq}(\mathcal{P}_{in})$
$\mathcal{C}_{kept} \leftarrow \mathcal{C}_{in} \setminus \mathcal{C}_{rem}$
$\mathcal{C}_{adj} \leftarrow \{\, c \in \mathcal{C}_{kept} \mid \exists c' \in \mathcal{C}_{rem} \,.\, \mathrm{adjacent}_{\mathcal{P}_{in}}(c, c') \,\}$
$\mathcal{C}_{conv} \leftarrow \mathcal{C}_{eq} \cup \mathcal{C}_{adj}$
$(\mathcal{C}_{conv}, \mathcal{G}_{conv}) \leftarrow \mathrm{conversion}(\mathcal{C}_{conv})$
$\mathcal{G}_{add} \leftarrow \{\, g \in \mathcal{G}_{conv} \mid \exists c \in \mathcal{C}_{rem} \,.\, \; g \text{ violates } c \,\}$
$(\mathcal{C}_{out}, \mathcal{G}_{out}) \leftarrow \mathrm{simplify}(\mathcal{C}_{kept}, \mathcal{G}_{in} \cup \mathcal{G}_{add})$
End

It is worth stressing that, when describing the new algorithm, our main goal is to provide an executable specification of a procedure for removing constraints that makes better use of information already available in the input DD pair. Probably, such a specification can be further optimized for speed.

For instance, the final call to procedure 'simplify' checks for redundant generators in the whole system $\mathcal{G}_{in} \cup \mathcal{G}_{add}$. On the contrary, a specialized implementation may distinguish between the generators in $\mathcal{G}_{in}$, since those that saturate none of the removed constraints are known to be not redundant.

Another opportunity for further optimization, which is currently under investigation, is based on the following observation. When removing many constraints using Algorithm 2, all of the adjacent ones are merged into a single system of constraints $\mathcal{C}_{conv}$ and converted from scratch to obtain $\mathcal{G}_{conv}$. A fully incremental approach would rather compute a separate subsystem of adjacent constraints for each removed constraint and perform many conversions. A priori, it is unclear which one of the two options above could be more efficient: on the one hand, the fully incremental approach deals with smaller subsystems; on the other hand, since some adjacent constraints will appear in more than a single subsystem, some computation will be uselessly repeated many times and the overall conversion cost could be higher. An interesting tradeoff is to partition the set of constraints to be removed into smaller subsets having only a few adjacent constraints in common.

4 Experimental Evaluation

In Tables 1 and 2 we report part of the results of a preliminary experimental evaluation aimed at comparing the efficiency of the incremental algorithms with respect to the naive ones.[6] We considered some of the examples of the `ppl_lcdd` tool, which is part of the Parma Polyhedra Library [6]. The original `ppl_lcdd` tool takes as input a constraint (resp., generator) representation of a convex polyhedron and produces as output the dual generator (resp., constraint) representation, thereby

[6] The tests have been performed on a laptop with an Intel Core i7-3632QM CPU, 16 GB of RAM and running GNU/Linux.

Removing constraints			naive comp			incremental comp				
test	size	rem	sat	sp	time	conv	add	sat	sp	time
ccc6.ext	211	1	*2G*	*145K*	*t/o*	16	1	5K	468	0.000
		5	*2G*	*149K*	*t/o*	55	5	8M	28K	0.100
		10	*2G*	*150K*	*t/o*	77	10	42M	81K	0.652
cut32-16.ext	368	1	*2G*	*126K*	*t/o*	16	1	7K	639	0.000
		5	*2G*	*88K*	*t/o*	49	5	53M	54K	0.840
		10	*2G*	*100K*	*t/o*	71	10	2G	263K	25.076
cyclic14-8.ext	240	1	22M	72K	0.628	8	1	2K	311	0.000
		5	23M	75K	0.628	23	15	63K	6K	0.004
		10	26M	85K	0.760	39	54	980K	26K	0.040
reg600-5-m.ext	2360	1	24M	3M	2.900	6	2	374K	5K	0.052
		5	24M	3M	2.892	11	4	388K	10K	0.048
		10	25M	3M	2.932	19	15	423K	37K	0.060
cyclic17-8.ine	17	1	285K	2K	0.008	17	165	720K	5K	0.012
		5	17K	275	0.000	13	112	50K	3K	0.004
		10	56	72	0.004	8	9	184	157	0.000
kkd38-6.ine	38	1	28M	48K	0.612	8	2	60K	137	0.000
		5	13M	29K	0.280	9	4	46K	255	0.000
		10	4M	14K	0.096	9	4	30K	285	0.000
mit31-20.ine	31	1	131M	22K	1.260	31	3651	234M	141K	1.592
		5	767K	4K	0.024	27	1231	3M	38K	0.068
		10	7K	806	0.004	22	131	26K	4K	0.036
sampleh8.ine	66	1	266M	263K	6.724	26	266	190M	34K	1.320
		5	190M	212K	5.052	46	1940	258M	278K	3.636
		10	116M	156K	3.032	49	3361	190M	378K	2.832
trunc10.ine	112	1	67M	193K	1.668	21	11	764K	6K	0.012
		5	67M	191K	1.656	57	98	15M	56K	0.292
		10	63M	188K	1.596	102	1242	62M	313K	1.580

Table 1
Removing constraints: naive vs incremental

solving the *facet/vertex enumeration problem*. For our experiments, we modified the tool so that, after having computed the DD pair, it removes a few constraints (resp., generators) using the algorithms under evaluation: in Table 1, we consider the case where 1, 5 and 10 constraints are removed. For space reasons, we omitted all of the smaller tests as well as several tests which turn out to be minor variations (often, the dual) of other tests; we also omitted most of the bigger tests, since their computational cost is well beyond the chosen timeout threshold.

The first two columns ('test' and 'size') report the name of the benchmark and the size of the input representation. Hence, the test having name 'ccc6.ext' is for an input polyhedron described by 211 constraints (when counting constraints at the implementation level, we include the positivity constraint). For each test we have three rows, corresponding to different numbers of constraints removed, which are reported in column 'rem'. The next three columns are measures taken on the naive algorithm: besides timings (in seconds), we also report the number of saturation inclusion tests, in column 'sat', and the number of scalar products, in column 'sp'. Suffixes K, M and G stand for 10^3, 10^6 and 10^9; when using these scaling suffixes, numbers are rounded upwards (i.e., we provide upper bounds). For each invocation

Removing generators			naive comp			incremental comp				
test	size	rem	sat	sp	time	conv	add	sat	sp	time
ccc6.ext	32	1	438K	3K	0.008	31	405	770K	16K	0.016
		5	67K	2K	0.004	27	221	169K	9K	0.004
		10	8K	690	0.000	22	50	20K	3K	0.000
cut32-16.ext	32	1	773K	4K	0.012	31	415	2M	18K	0.024
		5	126K	2K	0.004	27	293	321K	11K	0.008
		10	13K	839	0.000	22	69	35K	4K	0.000
cyclic14-8.ext	14	1	21K	314	0.000	13	84	49K	2K	0.004
		5	168	72	0.000	9	10	584	222	0.000
		10	12	32	0.000	4	6	270	82	0.000
reg600-5-m.ext	600	1	15M	713K	1.016	17	16	6M	10K	0.072
		5	15M	704K	1.008	54	69	6M	46K	0.088
		10	15M	692K	0.972	89	119	6M	81K	0.096
cyclic17-8.ine	935	1	*741M*	*930K*	*t/o*	8	3	8K	3K	0.000
		5	*730M*	*909K*	*t/o*	39	5	3M	24K	0.080
		10	*730M*	*915K*	*t/o*	69	27	55M	178K	1.644
kkd38-6.ine	252	1	629K	32K	0.072	6	4	3K	2K	0.000
		5	622K	31K	0.068	13	11	7K	3K	0.004
		10	614K	31K	0.068	16	10	8K	3K	0.004
mit31-20.ine	18553	1	*3G*	*118K*	*t/o*	19	3	354K	57K	0.032
		5	*3G*	*118K*	*t/o*	71	3	41M	120K	0.756
		10	*3G*	*118K*	*t/o*	302	0	*1G*	*242K*	*t/o*
sampleh8.ine	13865	1	*915M*	*1M*	*t/o*	9	1	130K	14K	0.020
		5	*913M*	*1M*	*t/o*	27	15	447K	210K	0.080
		10	*917M*	*1M*	*t/o*	53	51	29M	801K	1.248
trunc10.ine	290	1	2G	713K	18.000	10	1	16K	411	0.000
		5	390M	220K	2.508	19	34	39K	11K	0.004
		10	394M	270K	2.724	19	17	182K	11K	0.008

Table 2
Removing generators: naive vs incremental

of the removal algorithms, a timeout is set on 30 seconds of CPU time; if the timeout expires, we report 't/o' in the 'time' column and columns 'sat' and 'sp' will contain lower bounds. The next five columns report measures taken on the incremental algorithm. Besides the columns 'sat', 'sp' and 'time' described above, we also report the cardinality of $\mathcal{C}_{conv}$ in column 'conv' and the cardinality of $\mathcal{G}_{add}$ in column 'add'. Significant time improvements (above 0.2 seconds) are highlighted using underlining.

Table 2 shows the results obtained for the same tests when removing 1, 5 and 10 generators. The reader is warned that, in this case, some of the columns have to be interpreted dually; namely, 'size' is the cardinality of $\mathcal{G}_{in}$, 'rem' is the cardinality of $\mathcal{G}_{rem}$, 'conv' is the cardinality of $\mathcal{G}_{conv}$ and 'add' is the cardinality of $\mathcal{C}_{add}$.

The measurements reported allow for a few observations:

- the naive removal algorithm, while performing reasonably well on many tests, sometimes suffers high computational costs, leading to 6 timeouts in Table 1 and 9 timeouts in Table 2;
- since we are removing relatively few constraints or generators, the incremental algorithm usually performs much better: the timeout threshold is reached only

once, in Table 2;

- occasionally, there are cases where the incremental algorithm turns out to be slower than the naive one (see, for instance, the rows for tests 'cyclic17-8.ine' and 'mit31-20.ine' in Table 1); this happens because the constraints removed from the input polyhedra are adjacent to almost all of the constraints that are kept, as can be seen by comparing the value in column 'conv' (i.e., the cardinality of $\mathcal{C}_{conv}$) with the difference of 'size' and 'rem' (i.e., the cardinality of $\mathcal{C}_{kept}$).

Obviously, the observations above hold for the considered tests, which are not meant to be fully representative of the typical pattern for removing constraints (or generators) that can be found in other specific applications. In particular, most of the considered tests are characterized by the fact that one of the two representations is much smaller than the other: this usually causes one of the two conversions to require a significantly higher computation time.

4.1 Dynamic selection of the computational strategy

The observations made above suggest that an interesting balance in efficiency could be obtained by combining the naive and the incremental algorithms into a third one, where the choice of the computational strategy is taken dynamically. The combined algorithm, for the case of constraints removal, is shown as Algorithm 3: it uses helper function '*profitable*' to perform a heuristic guess about the profitability of using the incremental rather than the naive algorithm. The profitability test intuitively compares the sizes of constraint systems $\mathcal{C}_{conv}$ and $\mathcal{C}_{kept}$, falling back to the naive computation if $\mathcal{C}_{conv}$ is not small enough. Since the profitability check is based on both the input descriptions as well as intermediate results, we will refer to Algorithm 3 as the *introspective algorithm* for constraint removal.

Algorithm 3 Introspective removal of a set of constraints

Require: same as for Algorithm 1.
Ensure: same as for Algorithm 1.

Begin
$\mathcal{C}_{eq} \leftarrow \mathcal{C}_{in} \cap \mathrm{eq}(\mathcal{P}_{in})$
$\mathcal{C}_{kept} \leftarrow \mathcal{C}_{in} \setminus \mathcal{C}_{rem}$
$\mathcal{C}_{adj} \leftarrow \{\, c \in \mathcal{C}_{kept} \mid \exists c' \in \mathcal{C}_{rem} \,.\, \mathrm{adjacent}_{\mathcal{P}_{in}}(c, c') \,\}$
$\mathcal{C}_{conv} \leftarrow \mathcal{C}_{eq} \cup \mathcal{C}_{adj}$
if *profitable*$(\mathcal{C}_{conv}, \mathcal{C}_{kept})$ **then**
$(\mathcal{C}_{conv}, \mathcal{G}_{conv}) \leftarrow \mathrm{conversion}(\mathcal{C}_{conv})$
$\mathcal{G}_{add} \leftarrow \{\, g \in \mathcal{G}_{conv} \mid \exists c \in \mathcal{C}_{rem} \,.\, \; g \text{ violates } c \,\}$
$(\mathcal{C}_{out}, \mathcal{G}_{out}) \leftarrow \mathrm{simplify}(\mathcal{C}_{kept}, \mathcal{G}_{in} \cup \mathcal{G}_{add})$
else
//Fallback to naive computation
$(\mathcal{C}_{out}, \mathcal{G}_{out}) \leftarrow \mathrm{conversion}(\mathcal{C}_{kept})$
end if
End

Removing constraints			naive comp			introspective comp				
test	size	rem	sat	sp	time	conv	add	sat	sp	time
ccc6.ext	211	1	*2G*	*145K*	*t/o*	16	1	5K	468	0.000
		5	*2G*	*149K*	*t/o*	55	5	8M	28K	0.100
		10	*2G*	*150K*	*t/o*	77	10	42M	81K	0.656
cut32-16.ext	368	1	*2G*	*126K*	*t/o*	16	1	7K	639	0.000
		5	*2G*	*88K*	*t/o*	49	5	53M	54K	0.832
		10	*2G*	*100K*	*t/o*	71	10	2G	263K	24.968
cyclic14-8.ext	240	1	22M	72K	0.632	8	1	2K	311	0.000
		5	23M	75K	0.664	23	15	63K	6K	0.004
		10	26M	85K	0.784	39	54	980K	26K	0.040
reg600-5-m.ext	2360	1	24M	3M	2.944	6	2	374K	5K	0.048
		5	24M	3M	2.888	11	4	388K	10K	0.048
		10	25M	3M	3.008	19	15	423K	37K	0.060
cyclic17-8.ine	17	1	285K	2K	0.008	**17**	—	286K	2K	0.012
		5	17K	275	0.000	**13**	—	17K	275	0.004
		10	56	72	0.000	**8**	—	161	72	0.000
kkd38-6.ine	38	1	28M	48K	0.624	8	2	60K	137	0.000
		5	13M	29K	0.280	9	4	46K	255	0.000
		10	4M	14K	0.092	9	4	30K	285	0.000
mit31-20.ine	31	1	131M	22K	1.404	**31**	—	131M	22K	0.952
		5	767K	4K	0.020	**27**	—	767K	4K	0.036
		10	7K	806	0.004	**22**	—	8K	806	0.020
sampleh8.ine	66	1	266M	263K	6.936	26	266	190M	34K	1.408
		5	190M	212K	5.248	**46**	—	190M	212K	4.816
		10	116M	156K	3.052	**49**	—	116M	156K	2.744
trunc10.ine	112	1	67M	193K	1.652	21	11	764K	6K	0.016
		5	67M	191K	1.680	**57**	—	67M	191K	1.668
		10	63M	188K	1.608	**102**	—	63M	188K	1.588

Table 3
Naive vs introspective: removal of constraints

In Table 3 we show the results obtained by the introspective algorithm for removing constraints for the same tests of Table 1 with the following, tentative heuristics:

$$profitable(\mathcal{C}_{conv}, \mathcal{C}_{kept}) := \left(\#\mathcal{C}_{conv} \leq \frac{1}{2} \#\mathcal{C}_{kept} \right)$$

Since the profitability check can be implemented efficiently, the introspective algorithm incurs very little overhead when falling back to the naive algorithm (the fallbacks are reported by showing the values of column 'conv' in boldface and no value at all for column 'add'): hence, the introspective algorithm is able to exploit almost all the significant efficiency gains of the incremental algorithm, while avoiding most of the cases where the incremental algorithm incurs a slowdown. Note that in a few cases (e.g., when removing more than a single constraint in tests 'sampleh8.ine' and 'trunc10.ine') the heuristics causes a fallback that was not really needed, possibly preventing more significant efficiency gains. It is therefore clear that the implementation of the profitability heuristics should be tailored to the specific application at hand and, in general, can not ensure a decrease of the computation time.

An alternative approach for combining the naive and the incremental algorithms is to exploit the availability of multiple processing units and run both algorithms in different threads, stopping as soon as any of the two terminates.

5 Discussion

The standard widening [11,14] is likely to be the most well known operation on the domain of convex polyhedra whose implementation is based on the removal of constraints. Using widening '∇', a post-fixpoint of the monotonic operator '$\mathcal{F}^\sharp$' (the abstract semantics function) can be obtained as the limit of an increasing iteration sequence computed as follows:

$$\mathcal{P}_{i+1} := \mathcal{P}_i \nabla \left(\mathcal{P}_i \uplus \mathcal{F}^\sharp(\mathcal{P}_i)\right).$$

Widening is thus a good candidate for the application of algorithms for constraint removal that *preserve* the DD pair, such as the one proposed in this paper, because both representations of $\mathcal{P}_i$ are used when computing the next iterate $\mathcal{P}_{i+1}$: the generators are used when computing the convex polyhedral hull '$\uplus$' in the second argument; the constraints are used when computing the widening itself. These benefits are even more relevant when using a framework such as [2], where each program point is potentially a widening point, and [1], where widening is intertwined with narrowing and each loop may be analyzed several times.

Besides the standard widening, several less well known uses of constraint or generator removal can be found by inspecting the available literature. Each of these could be considered as a potential application of the algorithms proposed in this paper and may be the subject of further investigation to assess the profitability of the new algorithm for the considered context.

Miné [16] considers the problem of inferring sufficient conditions for a program property to hold. The proposed approach is modeled as a polyhedral analysis computing an under-approximation of the backward semantics of the program. In this setting, constraints removal is used to implement a safe under-approximation of backward affine tests. Similarly, in order to ensure the convergence of the under-approximation, [16] defines a *lower widening* which is similar to the standard widening used when computing over-approximations, but discards unstable generators rather than unstable constraints. By exploiting the duality of polyhedra representation, this new widening can be easily implemented by removing some of the generators from the polyhedron and hence could potentially benefit from the existence of more efficient algorithms for generator removal.

Fhrese [12] illustrates two techniques that are meant to manage the complexity of constraint descriptions in polyhedral computations. The first technique aims at limiting the number of bits used in the arbitrary precision integer coefficients used to represent the constraints; to this end, the constraints having huge coefficients are first identified and then *replaced* in the polyhedral description by other constraints having smaller coefficient, but still preserving the soundness of the approximation. This replacement step could be implemented by combining the decremental con-

straint removal algorithm proposed in this paper with the usual incremental constraint addition, thereby obtaining a polyhedron having both representations up to date. The second technique is a more direct one, in that it limits the number of constraints in the polyhedral representation by removing the less significant ones. Several variations for constraint selection are proposed (volumetric, slack, angle), which are then applied in two alternative procedures: a *construction* procedure, where the most significant constraints are added to the universe polyhedron; and a *deconstruction* procedure, where the least significant constraints are removed from the starting polyhedron. Depending on the final number of constraints obtained, a decremental constraint removal algorithm might become competitive with respect to incremental constraint addition.

In this paper we only considered topologically closed polyhedra. Not Necessarily Closed (NNC) polyhedra can be specified by allowing for strict inequalities in the constraint description (resp., closure points in the generator description [5]). Some care should be taken when trying to properly define the constraint or generator removal operators on the domain of NNC polyhedra. To start with, the DD pair has to be *fully* minimized as proposed in [5], so as to remove all kinds of redundancies. However, full minimization is not enough, as shown by the following example.

Example 5.1 Consider the NNC polyhedron $\mathcal{P} \subseteq \mathbb{R}^2$ defined by the constraint system $\mathcal{C} = \{0 \leq x \leq 1, 0 \leq y \leq 1, x + y < 2\}$. Note that $\mathcal{P}$ can also be defined by the constraint system $\mathcal{C}' = \{0 \leq x \leq 1, 0 \leq y \leq 1, x + 2y < 3\}$, and both $\mathcal{C}$ and $\mathcal{C}'$ are in minimal form. Depending on the chosen syntactic representation, the removal of constraint $x \leq 1$ leads to the computation of different NNC polyhedra.

In order to avoid the problem above, one possibility is to specify that the removal of a non-strict constraint from an NNC polyhedron is obtained by first tightening the constraint to be strict and then removing the strict constraint. In the example above, we first add strict constraint $x < 1$ (so that constraints $x + y < 2$ and $x + 2y < 3$ become redundant and are discarded from the input DD pairs) and then remove it, thereby obtaining NNC polyhedron $\mathcal{P}' = \mathrm{con}\big(\{0 \leq x, 0 \leq y \leq 1\}\big)$. A dual example can be devised for generator removal: in this case, the workaround is to transform closure points into points before removing them. We plan to extend the algorithms presented in this paper to the case of NNC polyhedra, based on these amended specifications.

Acknowledgment

The work of Enea Zaffanella has been supported by *Gruppo Nazionale per il Calcolo Scientifico* of *Istituto Nazionale di Alta Matematica*.

References

[1] Amato, G. and F. Scozzari, *Localizing widening and narrowing*, in: F. Logozzo and M. Fahndrich, editors, *Static Analysis, 20th International Symposium, SAS 2013*, Lecture Notes in Computer Science **7936** (2013), pp. 25–42.

[2] Apinis, K., H. Seidl and V. Vojdani, *How to combine widening and narrowing for non-monotonic systems of equations*, in: H.-J. Bohem and C. Flanagan, editors, *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (2013), pp. 377–386.

[3] Bachem, A. and M. Groötschel, *Characterization of adjacency of faces of polyhedra*, Mathematical Programming Study **14** (1981), pp. 1–22.

[4] Bagnara, R., P. M. Hill, E. Ricci and E. Zaffanella, *Precise widening operators for convex polyhedra*, Science of Computer Programming **58** (2005), pp. 28–56.

[5] Bagnara, R., P. M. Hill and E. Zaffanella, *Not necessarily closed convex polyhedra and the double description method*, Formal Aspects of Computing **17** (2005), pp. 222–257.

[6] Bagnara, R., P. M. Hill and E. Zaffanella, *The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems*, Science of Computer Programming **72** (2008), pp. 3–21.

[7] Bagnara, R., P. M. Hill and E. Zaffanella, *Applications of polyhedral computations to the analysis and verification of hardware and software systems*, Theoretical Computer Science **410** (2009), pp. 4672–4691.

[8] Chernikova, N. V., *Algorithm for finding a general formula for the non-negative solutions of system of linear equations*, U.S.S.R. Computational Mathematics and Mathematical Physics **4** (1964), pp. 151–158.

[9] Chernikova, N. V., *Algorithm for finding a general formula for the non-negative solutions of system of linear inequalities*, U.S.S.R. Computational Mathematics and Mathematical Physics **5** (1965), pp. 228–233.

[10] Chernikova, N. V., *Algorithm for discovering the set of all solutions of a linear programming problem*, U.S.S.R. Computational Mathematics and Mathematical Physics **8** (1968), pp. 282–293.

[11] Cousot, P. and N. Halbwachs, *Automatic discovery of linear restraints among variables of a program*, in: *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages* (1978), pp. 84–96.

[12] Frehse, G., *PHAVer: Algorithmic verification of hybrid systems past HyTech*, in: M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control: Proceedings of the 8th International Workshop (HSCC 2005)*, Lecture Notes in Computer Science **3414** (2005), pp. 258–273.

[13] Fukuda, K. and A. Prodon, *Double description method revisited*, in: M. Deza, R. Euler and Y. Manoussakis, editors, *Combinatorics and Computer Science, 8th Franco-Japanese and 4th Franco-Chinese Conference, Brest, France, July 3-5, 1995, Selected Papers*, Lecture Notes in Computer Science **1120** (1996), pp. 91–111.

[14] Halbwachs, N., "Détermination Automatique de Relations Linéaires Vérifiées par les Variables d'un Programme," Thèse de 3ème cycle d'informatique, Université scientifique et médicale de Grenoble, Grenoble, France (1979).

[15] Le Verge, H., *A note on Chernikova's algorithm*, *Publication interne* 635, IRISA, Campus de Beaulieu, Rennes, France (1992).

[16] Miné, A., *Inferring sufficient conditions with backward polyhedral under-approximations*, in: *Proceedings of the 4th International Workshop on Numerical and Symbolic Abstract Domains (NSAD'12)*, Electronic Notes in Theoretical Computer Science (ENTCS) **287** (2012), pp. 89–100.

[17] Motzkin, T. S., H. Raiffa, G. L. Thompson and R. M. Thrall, *The double description method*, in: H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games – Volume II*, number 28 in Annals of Mathematics Studies, Princeton University Press, Princeton, New Jersey, 1953 pp. 51–73.

[18] Stoer, J. and C. Witzgall, "Convexity and Optimization in Finite Dimensions I," Springer-Verlag, Berlin, 1970.

[19] Zolotykh, N. Y., *New modification of the double description method for constructing the skeleton of a polyhedral cone*, Computational Mathematics and Mathematical Physics **52** (2012), pp. 146–156.