Revisiting Polyhedral Analysis for Hybrid Systems

Anna Becchi[1] and Enea Zaffanella[2]

[1] University of Udine, Italy
becchi.anna@spes.uniud.it
[2] University of Parma, Italy
enea.zaffanella@unipr.it

Abstract. Thanks to significant progress in the adopted implementation techniques, the recent years have witnessed a renewed interest in the development of analysis tools based on the domain of convex polyhedra. In this paper we revisit the application of this abstract domain to the case of reachability analysis for hybrid systems, focusing on the lesson learned during the development of the tool PHAVerLite. In particular, we motivate the implementation of specialized versions of several well known abstract operators, as well as the adoption of a heuristic technique (*boxed polyhedra*) for the handling of finite collections of polyhedra, showing their impact on the efficiency of the analysis tool.

1 Introduction

Hybrid automata model dynamic systems exhibiting both discrete and continuous behaviors. Due to the intrinsic complexity of these systems, soon after their introduction several approaches have been put forward to apply formal methods, so as to support the developer when reasoning about their correctness. Most notably, in [21,22] it was shown how abstract interpretation [9] based on the domain of convex polyhedra [12] can be used to compute correct approximations of the reachable states for the class of linear hybrid automata.

During the following years, many tools for the automatic analysis of hybrid systems have been implemented. In particular, PHAVer (Polyhedral Hybrid Automaton Verifier, [13,14]) represented a significant progress with respect to its predecessor HyTech [23]. The applicability of the approach was extended from the piecewise constant to the affine class of automata, by on-the-fly overapproximation of the continuous dynamics tailored by a systematic partitioning of the state space. Building on the PPL (Parma Polyhedra Library [4,5]), PHAVer features a robust and relatively efficient backend for computing on the domain of NNC (not necessarily closed) polyhedra. Moreover, it is characterized by the systematic adoption of heuristic techniques meant to overcome the inherent limitations affecting the implemented analysis: the excessive complexity of operators based on convex polyhedra; the loss of accuracy caused by the convex approximation; and the slow convergence of the fixpoint computation, in particular when relying on partitioning techniques while using exact arithmetic. Since

2011, PHAVer is included as a plug-in (PHAVer/SX) in SpaceEx [17]. Several other verification tools for hybrid systems that are based on different abstract domains and/or approximation algorithms have been developed [31]. Here we only mention another SpaceEx plug-in, the LGG scenario [17,18], based on a domain of template polyhedra, which however sacrifices formal soundness due to the adoption of floating point computations.

In recent years, we are witnessing new momentum in the development of efficient algorithms for the domain of convex polyhedra:

- by revisiting the Cartesian factoring technique proposed in [19,20], it was shown that a static analysis tool based on convex polyhedra [32] is able to obtain impressive speedups when analyzing benchmarks taken from the software verification competition SV-COMP;
- a constraint-only version of the domain of convex polyhedra has been implemented in the VPL (Verimag Polyhedra Library), exploiting Parametric Linear Programming to quickly identify and remove redundancies [26];
- a new conversion algorithm has been proposed in [6] for the domain of NNC polyhedra, improving upon the previous approaches [2,21]; this lead to the development of the PPLite library [7], which is shown to obtain remarkable efficiency improvements on the static analysis of C programs.

This progress motivated new interest in revisiting the application of polyhedral computations in the context of the analysis and verification of hybrid systems. In particular, choosing PHAVer/SX as a starting point, a new plug-in PHAVer-lite/SX [16] has been implemented for the SpaceEx platform, mainly characterized by the replacement of the PPL backend with the newly developed library PPLite [7]. Building on the encouraging efficiency results obtained by PHAVer-lite/SX, in this paper we describe the new tool PHAVerLite [15].

While providing the same formal soundness guarantees, PHAVerLite differs from PHAVer-lite/SX in that it is designed as a *stand-alone* tool, like the original PHAVer. The independence from the SpaceEx platform simplifies the application of more significant changes to the underlying algorithm for reachability analysis, so as to easily experiment with novel computational heuristics, design tradeoffs and specialized operators on the underlying domain of NNC polyhedra. In Figure 1 we summarize the efficiency improvements obtained, with respect to both PHAVer/SX and PHAVer-lite/SX, when analyzing the benchmarks coming from the HPWC (hybrid systems with piecewise constant dynamics) category of the ARCH-COMP friendly competition [15,16]. In the 2019 edition, the HPWC category had a total of 25 tests: 15 'safe' tests (aiming at *proving* a safety property, so that a reachability analysis is permitted to compute overapproximations) and 10 'unsafe' tests (aiming at *disproving* a safety property, meaning that no overapproximation is permitted). For the 13 tests on which PHAVer/SX is able to terminate[3] the average speedup factor obtained by PHAVerLite is ~ 337; moreover, PHAVerLite is able to complete the analysis (successfully proving

[3] The tests on PHAVer-lite/SX and PHAVerLite have been executed on an Intel Core i7-3632QM CPU; the tests on PHAVer/SX were executed on a faster CPU ($\sim 25\%$).

or disproving the corresponding property as required) of all but one of the 25 benchmarks in ~ 224 seconds.

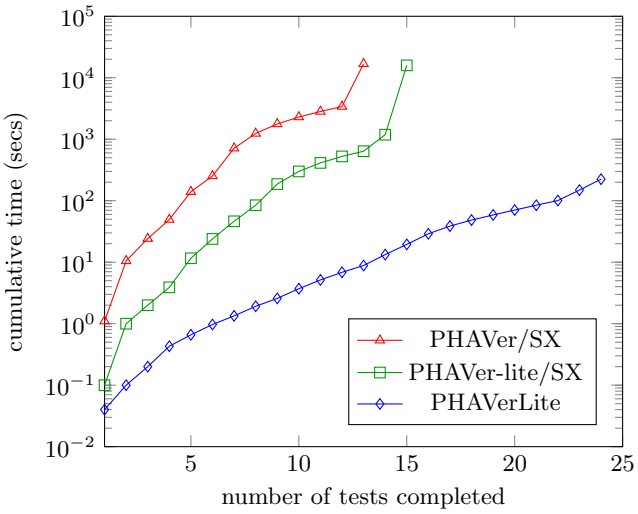

Fig. 1. Comparing the tools PHAVer/SX, PHAVer-lite/SX and PHAVerLite on the ARCH-COMP HPWC benchmarks.

The main contribution of this paper, however, is not the presentation of the tool itself: rather, we describe in detail a few of the specific improvements implemented in PHAVerLite, highlighting their impact on the overall efficiency. In particular,

- we propose specialized implementations for several "common" abstract operators on the domain of convex polyhedra: the computation of affine images (Section 3.1); the approximation of the convex polyhedral hull (Section 3.2); and the splitting of a polyhedron according to a constraint (Section 5);
- we propose a novel heuristic approach (*boxed polyhedra*) for a more efficient handling of finite collections of polyhedra (Section 4.2), which can also be viewed as an instance of an online meta-analysis [11].

In summary, our investigation shows that, when adopting an abstract domain sacrificing some performance to favor precision, a good portion of the inefficiencies can often be eliminated by the identification of suitable heuristic techniques.

The paper is structured as follows. Section 2 briefly recalls some preliminary concepts. Section 3 reconsiders the implementation of affine images for convex polyhedra and discusses ways to overapproximate the expensive convex polyhedral hull. Section 4 tackles the problem of the efficient handling of finite collections of polyhedra, showing the effectiveness of a new heuristics. Section 5

proposes a new operator for polyhedra libraries based on the Double Description framework, motivated by the usage of location partitioning techniques. We conclude in Section 6.

2 Preliminaries

A non-trivial, non-strict linear inequality constraint β defines a closed half-space $\mathrm{con}(\{\beta\})$ of the vector space $\mathbb{R}^n$; we write $\neg\beta$ to denote the complement of β, i.e., the open half-space $\mathrm{con}(\{\neg\beta\}) = \mathbb{R}^n \setminus \mathrm{con}(\{\beta\})$. A not necessarily closed (NNC) convex polyhedron $\phi = \mathrm{con}(\mathcal{C}) \subseteq \mathbb{R}^n$ is defined as the set of solutions of a finite system $\mathcal{C}$ of (strict or non-strict) linear inequality constraints; equivalently, $\phi = \mathrm{gen}(\mathcal{G})$ can be defined as the set obtained by suitably combining the elements (lines, rays, points and closure points) of a generator system $\mathcal{G}$. The Double Description framework [28] exploits both representations; we write $\phi \equiv (\mathcal{C}, \mathcal{G})$ to denote that $\phi = \mathrm{con}(\mathcal{C}) = \mathrm{gen}(\mathcal{G})$. The set $\mathbb{P}_n$ of all NNC polyhedra on $\mathbb{R}^n$, partially ordered by set inclusion, is a lattice $\langle \mathbb{P}_n, \subseteq, \emptyset, \mathbb{R}^n, \cap, \uplus \rangle$, where the emptyset and $\mathbb{R}^n$ are the bottom and top elements, the binary meet operator is set intersection and the binary join operator '$\uplus$' is the convex polyhedral hull.

The use of the domain of convex polyhedra for static analyses based on abstract interpretation has been introduced in [12]. The semantics of the analyzed system is modeled by suitably combining the lattice operators mentioned above with other operators that approximate the concrete behavior of the system. For instance, the effect of a conditional guard described by linear constraints can be modeled by the meet of the lattice, whereas the convex polyhedral hull can be used to approximate the merging of control flow paths. The effect of affine assignments on state variables can be modeled by computing the image of a domain element under an affine transformation; the addition of k new state variables is modeled by operator $\mathrm{add\_dims}_k \colon \mathbb{P}_n \to \mathbb{P}_{n+k}$, embedding the input polyhedron in a higher dimension space, where the newly added dimensions are unconstrained; similarly, the removal of a set V of state variables, where $|V| = k$, can be modeled by a projection operator $\mathrm{rem\_dims}_V \colon \mathbb{P}_{n+k} \to \mathbb{P}_n$.

The set $\mathbb{CP}_n$ of closed polyhedra on the vector space $\mathbb{R}^n$ is a sublattice of $\mathbb{P}_n$; $\mathbb{CB}_n$ denotes the set of closed boxes on $\mathbb{R}^n$, i.e., those polyhedra that can be defined by inequality constraints having the form $\pm x_i \leq k$. Note that $\langle \mathbb{CB}_n, \subseteq, \emptyset, \mathbb{R}^n, \cap \rangle$ is a meet-sublattice of $\mathbb{CP}_n$.

For a set S, $\wp(S)$ denotes the powerset of S; we will write $\wp_{\mathrm{f}}(S)$ to denote the *finite* powerset of S, i.e., the set of all the finite subsets of S. The cardinality of S is denoted by $|S|$. The finite powerset construction [3] is a domain refinement similar to disjunctive completion [10]. It can be used to lift a base-level abstract domain to model disjunctions by explicit (hence, finite) collections of base-level elements. In the following, we instantiate the finite powerset construction by fixing $\mathbb{P}_n$ as the base-level abstract domain, thereby trading some generality for concreteness and readability. The reader interested in obtaining more details and some links to the relevant literature is referred to [3]. For efficiency, it is important that these finite collections of elements do not contain redundancies.

A set $S \in \wp_{\mathrm{f}}(\mathbb{P}_n)$ is *non-redundant* (with respect to the base-level partial order $\subseteq$) if and only if $\emptyset \notin S$ and $\forall \phi_1, \phi_2 \in S : \phi_1 \subseteq \phi_2 \implies \phi_1 = \phi_2$. The set of finite non-redundant subsets of $\mathbb{P}_n$ is denoted $\wp_{\mathrm{fn}}(\mathbb{P}_n)$. The reduction function $\Omega\colon \wp_{\mathrm{f}}(\mathbb{P}_n) \to \wp_{\mathrm{fn}}(\mathbb{P}_n)$ is defined, for each $S \in \wp_{\mathrm{f}}(\mathbb{P}_n)$ by

$$\Omega(S) \stackrel{\text{def}}{=} S \setminus \{\, \phi_1 \in S \mid \phi_1 = \emptyset \vee \exists \phi_2 \in S \,.\, \phi_1 \subset \phi_2 \,\}.$$

Definition 1 (Finite powerset over $\mathbb{P}_n$.). *The finite powerset domain over $\mathbb{P}_n$ is the join-semilattice $\langle \wp_{\mathrm{fn}}(\mathbb{P}_n), \vdash, \bot, \oplus_\Omega \rangle$, where the bottom element is $\bot = \emptyset$ and the binary join operator is defined by $S_1 \oplus_\Omega S_2 \stackrel{\text{def}}{=} \Omega(S_1 \cup S_2)$.*

Note that $S_1 \vdash S_2$ if and only if $\forall \phi_1 \in S_1 : \exists \phi_2 \in S_2 \,.\, \phi_1 \subseteq \phi_2$ (i.e., '$\vdash$' is the Hoare powerdomain partial order).

We adopt a tailored definition for hybrid automata. In particular, we assume that: initial states and state invariants are modeled by NNC polyhedra having a fixed space dimension n; the discrete post operator is modeled by a linear relation on pre-state and post-state variables (hence, having space dimension $2n$); and the continuous flow operator is modeled by linear constraints on the first-order derivatives of the variables (i.e., piecewise constant dynamics).

Definition 2. *Let Loc, Lab and Var be finite sets of locations, synchronization labels and state variables, respectively, where $n = |Var|$. A hybrid automaton $H = \langle Loc, Lab, Var, \mathrm{init}, \mathrm{inv}, \longrightarrow, \mathrm{flow} \rangle$ is defined by:*

- *initial states* $\mathrm{init}\colon Loc \to \mathbb{P}_n$ *and invariant states* $\mathrm{inv}\colon Loc \to \mathbb{P}_n$*, satisfying* $\mathrm{init}(\ell) \subseteq \mathrm{inv}(\ell)$*;*
- *a finite set* $\longrightarrow \subseteq (Loc \times Lab \times \mathbb{P}_{2n} \times Loc)$ *of discrete transitions between locations; we write* $\ell_1 \xrightarrow{a,\mu} \ell_2$ *to denote that* $(\ell_1, a, \mu, \ell_2) \in \longrightarrow$*;*
- *a continuous flow relation* $\mathrm{flow}\colon Loc \to \mathbb{P}_n$ *specifying the constraints on the first order derivatives of the state variables.*

Note that we consider the case where all initial states and invariants are convex. Finite disjunctions can still be modeled by splitting locations; as an alternative, one may explicitly choose $\wp_{\mathrm{f}}(\mathbb{P}_n)$ as the codomain of 'init' and/or 'inv'.

The goal of reachability analysis is to compute or overapproximate the reachable set of configurations for the automaton. The reachable set is defined as the fixpoint of a system of semantic equations, one for each location of the automaton, having the following form [21]:

$$\mathrm{reach}(\ell) \stackrel{\text{def}}{=} \left(\left(\mathrm{init}(\ell) \cup \bigcup_{\ell' \xrightarrow{a,\mu} \ell} \mathrm{dpost}\big(\mathrm{reach}(\ell'), \mu, \mathrm{inv}(\ell)\big) \right) \nearrow \mathrm{flow}(\ell) \right) \cap \mathrm{inv}(\ell)$$

Informally, this equation means that the reachable state at location ℓ satisfies its invariant predicate $\mathrm{inv}(\ell)$ and is obtained by letting the state evolve according to the continuous relation $\mathrm{flow}(\ell)$ (using the *time-elapse* operator $\nearrow$), starting from either an initial state in $\mathrm{init}(\ell)$ or from a state that can reach ℓ through any incoming transition, via the discrete post operator dpost.

Depending on the desired precision/efficiency tradeoff, on the domain of NNC polyhedra the set reach(ℓ) can be modeled by using either a single polyhedron or a finite set of polyhedra: analysis tools such as PHAVer let the user choose the approach. Convergence may be enforced by using widening operators [1,3,12,21], but it is often the case that no widening is applied (e.g., when overapproximations are not permitted), thereby obtaining a potentially non-terminating analysis.

3 Improving polyhedra operators

An approximation of the reachable set of a hybrid automaton can be computed by iterating a suitable composition of well known operators on the domain of NNC polyhedra [21,22]. In this section we focus our attention on the efficient (exact or approximated) implementation of some of these operators.

3.1 Computing the discrete post operator

The discrete post operator models the effect of a transition $\ell_1 \xrightarrow{a,\mu} \ell_2$ mapping the automaton state from the source location ℓ_1 to the target location ℓ_2. Namely, if $S_1 = \{\phi_1, \ldots, \phi_{m_1}\}$ is the current reachable state at ℓ_1, each disjunctive component $\phi_i \in \mathbb{P}_n$ is mapped by $\mu \in \mathbb{P}_{2n}$ to a disjunctive component $\psi_j \in \mathbb{P}_n$, contributing to the formation of the reachable state $S_2 = \{\psi_1, \ldots, \psi_{m_2}\}$ of the target location ℓ_2.[4]

Focusing now on a single disjunctive component $\phi \in \mathbb{P}_n$, the *relational approach* to compute the corresponding target component $\psi \in \mathbb{P}_n$ is by a straightforward application of the relational constraints in μ, which amounts to the following abstract domain operations:

- state $\phi \in \mathbb{P}_n$ is embedded in space $\mathbb{P}_{2n}$, by adding n unconstrained primed variables V', yielding $\mu_1 = \text{add\_dims}_n(\phi)$;
- the constraints in μ are added to $\mu_1 \in \mathbb{P}_{2n}$, obtaining $\mu_2 = \mu_1 \cap \mu$;
- μ_2 is brought back to $\mathbb{P}_n$, by projecting away the n unprimed variables V, obtaining $\psi_2 = \text{rem\_dims}_V(\mu_2)$;
- finally, ψ_2 is intersected with the target invariant, yielding $\psi = \psi_2 \cap \text{inv}(\ell_2)$.

While generally applicable, this relational approach may incur a high computational overhead, due to the temporary doubling of the number of variables. As a consequence, most analysis tools provide optimized implementations for those special cases when the relational constraints in μ happen to encode a rather simple relation between the pre- and post- values of state variables. A common approach is to classify the constraints in μ as follows:

- *guard constraints*: these are constraints that mention unprimed (i.e., source-state) variables only; they are meant to filter the source state, possibly disabling the transition altogether;

[4] The synchronization label $a \in \mathit{Lab}$ only plays a role when a hybrid automaton is defined as the parallel composition of several smaller automata.

- *identity relations*: these are constraints having the form $x'_i = x_i$, modeling the fact that state variable x_i is not affected by the transition;
- *simple resets* and *increments*: these have the form $x'_i = k$ and $x'_i = x_i \pm k$, respectively.

If all the constraints defining μ have one of the forms above, then the discrete transition can be implemented rather efficiently, without the addition of new variables. However, there are cases escaping from the given classification.

Example 1. The Dutch Railway Network benchmark (DRNW) is one of the test in the HPWC category of the ARCH-COMP competition [15]. The automaton specified in this benchmark is rather peculiar: being derived from a MPL (max-plus-linear) system specified using difference-bound constraints, it happens to be a purely discrete automaton (i.e., it has a trivial continuous flow dynamics). The automaton tracks the value of 14 variables $Var = \{x_1, \ldots, x_{14}\}$, each one representing the departure time of trains from given railway stations. It has a single location, featuring 12 self-loop discrete transitions, each one corresponding to a different "region". An example of linear relation $\mu \in \mathbb{P}_{28}$ (modeling the discrete transition for Region 1) is described by the following constraint system where x_i and x'_i denote the pre- and post- values of state variable x_i, respectively:

$$
\begin{array}{ll}
40 + x_1 \geq 72 + x_6, & 55 + x_7 \geq 54 + x_8, \\
55 + x_7 \geq 37 + x_5, & 90 + x_{11} \geq 93 + x_{12}, \\
x'_1 = 38 + x_6, & x'_2 = 40 + x_1, \\
x'_3 = 50 + x_2, & x'_4 = 41 + x_3, \\
x'_5 = 41 + x_4, & x'_6 = 53 + x_5, \\
x'_7 = 38 + x_{14}, & x'_8 = 36 + x_{14}, \\
x'_9 = 55 + x_7, & x'_{10} = 35 + x_9, \\
x'_{11} = 54 + x_{10}, & x'_{12} = 58 + x_{10}, \\
x'_{13} = 90 + x_{11}, & x'_{14} = 16 + x_{13}.
\end{array}
$$

The first 4 constraints describing μ, mentioning unprimed variables only, form the transition guard. The remaining 14 constraints of μ bind a distinct primed variable to a linear expression on unprimed variables only; these can be seen as implementing a (non-simple) reset of all the state variables. Note that these resets are meant to be computed simultaneously: in particular, due to the presence of circular dependencies (e.g., $x_1 \to x_6 \to x_5 \to x_4 \to x_3 \to x_2 \to x_1$), the semantics of the overall parallel reset operation is not equivalent to a sequential composition of the individual resets.

Due to the problem with variable dependencies, tools such as PHAVer/SX implement the discrete transition $\ell_1 \xrightarrow{a,\mu} \ell_2$ following the relational approach.

An alternative, *parallel approach*, which can be adopted whenever μ is a combination of linear guard and reset constraints as in Example 1, relies on polyhedra libraries implementing the *parallel affine image* operator. In this case, after intersecting the source state ϕ with the guard constraints, this specific operator is applied, avoiding the intermediate changes of space dimension. While

being more efficient than the relational approach, even in this case the computational overhead may be significant; this is due to the fact that, in libraries based on the Double Description method, the parallel affine image operator is often implemented by rewriting the generator system: in order to obtain the constraint description, a *non-incremental* application of the conversion algorithm by Chernikova is required.[5]

To avoid the problem above, we propose the *compiled parallel approach*, based on an alternative implementation of the parallel affine image operator (available in PPLite 0.4), only requiring *incremental* applications of the Chernikova algorithm. To this end, we "compile" the set of parallel bindings into a carefully chosen sequence of calls to the non-parallel affine image operator (whose incremental computation is simple). We initially build a dependency graph where each arc $x_i \to x_j$ means that the new value of x_i depends on x_j (hence, a binding resetting x_j can be processed only after having processed the binding for x_i). Using the graph, we process those bindings having no dependencies, keeping the graph up-to-date. When identifying a circular dependency (i.e., a cycle in the graph), we break it by introducing a minimal number of primed variables, so as to allow continuing with the sequential processing of the bindings. The unprimed versions of these additional variables are later projected away. By following this technique, all of the bindings can be processed by adding only a few variables, thereby better exploiting the incremental nature of Chernikova algorithm.

Example 2. Considering Example 1, the new approach to compute the discrete post operator starts, as before, by adding to ϕ the guard constraints. Then we compute the dependency graph for the reset constraints, identifying cycles

$$x_1 \to x_6 \to x_5 \to x_4 \to x_3 \to x_2 \to x_1,$$
$$x_{14} \to x_{13} \to x_{11} \to x_{10} \to x_9 \to x_7 \to x_{14},$$

as well as the non-circular dependencies $x_8 \to x_{14}$ and $x_{12} \to x_{14}$.

Since x_8 and x_{12} have no entering arcs, the corresponding bindings $x'_8 = 36 + x_{14}$ and $x'_{12} = 58 + x_{10}$ can be processed (in any order). After that no other binding can be processed, since we are left with the two cycles. Considering the first cycle, we add a new space dimension for variable x'_1; as a consequence, we can process the sequence of bindings $x'_1 = 38 + x_6$, $x'_6 = 53 + x_5$, $x'_5 = 41 + x_4$, $x'_4 = 41 + x_3$, $x'_3 = 50 + x_2$, $x'_2 = 40 + x_1$ (in this order) and then project away the unprimed variable x_1. The bindings forming the other cycle are handled similarly, for instance adding (and then projecting away) a single space dimension for x'_{14}.

In Table 1 we show the time spent in PHAVerLite when computing the reachable states for the specific benchmark DRNW-BDR01, when adopting the classical relational and the compiled parallel approaches. As said before, since this benchmark has a trivial continuous dynamics, almost all of the analysis time is actually spent in the 96 calls to the discrete post operator. By exploiting incrementality, the new approach is able to obtain a significant speedup factor

[5] This is the case for the Apron library [24] and for PPLite up to version 0.3.

(column 'ratio'). It is worth stressing that, even though the relational constraints $\mu \in \mathbb{P}_{28}$ of Example 1 are all difference-bound constraints, the approach we are proposing is more general, as it can handle all kinds of affine constraints.

	discrete post		overall analysis	
implementation	calls	time	time	ratio
relational	96	129.28	129.60	11.86
compiled parallel	96	10.64	10.93	1.00

Table 1. Efficiency of the discrete post operator for the DRNW-BDR01 benchmark.

3.2 Approximating the convex polyhedral hull

As briefly recalled in Section 2, in principle the computation of reach(ℓ) requires to compute the *set union* of the initial states and the contributes of incoming transitions. Since this may incur high computational costs, the classical approach [21,22] maintains a single polyhedron per location and systematically overapproximates set unions using the convex polyhedral hull '$\uplus$'.

As a matter of fact, there are cases when even the computation of the convex polyhedral hull can be regarded as an overkill, so that more aggressive approximations are applied. One approach is to replace the abstract domain of NNC polyhedra with some further abstraction (such as octagons [27] or even boxes). Another possibility is to keep computing on the domain of NNC polyhedra, but use an approximate version of the convex polyhedral hull operator. Letting $\phi_1 = \mathrm{con}(\mathcal{C}_1)$ and $\phi_2 = \mathrm{con}(\mathcal{C}_2)$, there are several options:

- the *envelope* $\phi_1 \sqcup_{\mathrm{env}} \phi_2$, proposed in [8], is defined by keeping only those constraints $\beta \in \mathcal{C}_1 \cup \mathcal{C}_2$ that are valid for both ϕ_1 and ϕ_2;
- the *weak join* $\phi_1 \sqcup_{\mathrm{w}} \phi_2$, formalized in [30], is the smallest polyhedron containing $\phi_1 \cup \phi_2$ which is defined by constraints sharing the same slope with the ones occurring in $\mathcal{C}_1 \cup \mathcal{C}_2$;
- the *inversion join* $\phi_1 \sqcup_{\mathrm{inv}} \phi_2$ [30] further improves on the weak join by also inferring some constraint slopes not occurring in $\mathcal{C}_1 \cup \mathcal{C}_2$.

Note that $(\phi_1 \cup \phi_2) \subseteq (\phi_1 \uplus \phi_2) \subseteq (\phi_1 \sqcup_{\mathrm{inv}} \phi_2) \subseteq (\phi_1 \sqcup_{\mathrm{w}} \phi_2) \subseteq (\phi_1 \sqcup_{\mathrm{env}} \phi_2)$.

In the following we consider the operator adopted in the original PHAVer, named *constraint hull*, which happens to be equivalent to the weak join of [30]. Given a constraint $\beta_1 \in \mathcal{C}_1$, the problem of finding the tightest constraint β_2 having the same slope of β_1 and satisfying $\phi_2 \subseteq \mathrm{con}(\beta_2)$ can be addressed either as a Linear Programming problem or, if the chosen representation allows it, by enumerating the generators defining ϕ_2. In both cases, we can obtain a significant efficiency improvement with respect to the computation of the convex polyhedral hull, which may either require a high number of iterations of the Chernikova

conversion algorithm or a high number of redundancy checks when adopting a constraint-only approach.

Example 3. The Fischer protocol benchmark (FISC) is one of the tests from the HPWC category of the ARCH-COMP competition [15]; it models a time based protocol for mutual exclusion between processes described in [25]. For the instance FISCS06, whose composed automaton has 6 variables and 28672 locations, the verification goal is to prove that, during the interaction of 6 processes, no two processes can be in the critical section at the same time. In order to prove this property it is sufficient to keep a single polyhedron for each location. In Table 2 we compare the overall analysis time obtained when using different implementations to compute (exact or approximated) unions of NNC polyhedra. Column 'iters' reports the number of iterations of the fixpoint computation; column 'poly' reports the number of polyhedra in the reachable set (which equals the number of reachable locations when using '$\uplus$' or '$\sqcup_{\mathrm{w}}$'). The analysis exceeds a 20 minutes timeout threshold when adopting exact unions (i.e., when computing on the finite powerset domain of NNC polyhedra); the "constraint hull" approach performs significantly better than the convex polyhedral hull, also because it causes the analysis to converge after fewer iterations. Note that the one reported is the *total time* spent by PHAVerLite, including the parsing phase and the generation of the automaton by parallel composition of its components; together, these consume almost 40% of the 11.42 seconds spent on FISCS06.

implementation	iters	poly	time	time ratio
$\cup$	> 184040	> 137072	> 1200.00	> 100.0
$\uplus$	27289	2378	261.00	22.9
$\sqcup_{\mathrm{w}}$	8738	2378	11.42	1.0

Table 2. Comparing exact and approximated unions for the FISCS06 benchmark.

The results above have been obtained after replacing the original constraint hull implementation in PHAVer with a *specialized operator* (based on the enumeration of generators) made available in version 0.4 of the PPLite library. While this change has a negligible effect on the FISCS06 benchmark itself, we have observed impressive speedups on those benchmarks characterized by a higher number of state variables. For example, the time to compute the 917 applications of the constraint hull operator for the DISC04 benchmark (having 17 state variables) dropped from 675.50 to 2.14 seconds.

When using PHAVerLite on the experiments of Figure 1, 3 of the 15 'safe' tests require the full precision of the set union operator; for 11 tests the overapproximation provided by the constraint hull operator is precise enough to prove the property of interest; for the remaining test, a timeout is obtained no matter the considered approach.

4 Handling sets of polyhedra

As discussed in the previous section, when the analyzed system can be verified by approximating disjunctions using a single polyhedron (either by applying the convex polyhedral hull or by more aggressive forms of approximation), then this is usually the most efficient approach. There are however cases (e.g., when disproving the safety property in an 'unsafe' test) when in order to complete successfully the verification task at hand the analysis needs to (explicitly or implicitly) maintain a collection of elements of the chosen abstract domain. Therefore, in this section we consider an analysis that models disjunctions using explicit, finite collections of polyhedra in $\mathbb{P}_n$.

In this context, it can be seen that most of the operators defining the semantics of the system happens to be *additive*, so that they can be modeled by an element-wise application of the corresponding approximation operator defined on the base-level domain $\mathbb{P}_n$. For instance, the meet (i.e., set intersection) operator on $\mathbb{P}_n$ can be lifted on finite sets S_1, S_2 of polyhedra as follows:

$$S_1 \sqcap S_2 \stackrel{\text{def}}{=} \{\, \phi_1 \cap \phi_2 \mid \phi_1 \in S_1, \phi_2 \in S_2 \,\}. \tag{1}$$

If $n_1 = |S_1|$ and $n_2 = |S_2|$, this approach requires $n_1 \cdot n_2$ applications of the base-level meet operator. In order to keep efficiency under control, it is therefore important that these finite collections of elements do not encode redundant information.

4.1 On redundancy removal

The intuitive notion of "redundancy" needs some clarification. In the context of reachability analysis, the concrete semantics of a finite set $S \in \wp_{\mathrm{f}}(\mathbb{P}_n)$ is defined by the set union operator $[\![S]\!] = \bigcup S$. Hence, strictly speaking, S may be encoding redundant information in several, distinct ways:

1. an element $\phi \in S$ can be said to be redundant in S when it can be simply dropped without affecting the semantics, so that $[\![S]\!] = [\![S \setminus \{\phi\}]\!]$;
2. if there exists $\phi_1, \phi_2 \in S$ such that $\phi_1 \cap \phi_2 \neq \emptyset$, then ϕ_1 and/or ϕ_2 could be *partitioned* in sets S'_1 and S'_2 of smaller, pairwise disjoint polyhedra such that $[\![S]\!] = [\![S \setminus \{\phi_1, \phi_2\} \cup S'_1 \cup S'_2]\!]$; after partitioning, some of the elements in $S'_1 \cup S'_2$ may become redundant according to 1 and hence removed;
3. a subset $S' \subseteq S$, where $|S'| > 1$, could be *merged* into a single polyhedron $\phi = \biguplus S'$ such that $[\![S]\!] = [\![S \setminus S' \cup \{\phi\}]\!]$, decreasing the cardinality of the finite collection.

Note that the first form of redundancy listed above corresponds to the one used when introducing the finite powerset construction (see Section 2). Since maintaining non-redundancy has its own computational cost, most analysis tools usually choose this lighter definition.

As a matter of fact, the original code in PHAVer was sometimes adopting an even weaker form of redundancy removal when joining two finite sets of

polyhedra; namely, $S_1 \oplus_{\mathrm{w}} S_2 \stackrel{\text{def}}{=} S_1 \cup \{\, \phi_2 \in S_2 \mid \nexists \phi_1 \in S_1 \,.\, \phi_2 \subseteq \phi_1 \,\}$. Note that '$\oplus_{\mathrm{w}}$' is not symmetric: thus, it may fail to remove some elements in S_1 that are made redundant by elements coming from S_2. On the other hand, the application of the symmetric operator '$\oplus_\Omega$' requires a higher number of inclusion tests: in the worst case, which is always attained when ϕ is not redundant in S, $S \oplus_{\mathrm{w}} \{\phi\}$ requires $|S|$ inclusion tests, whereas $S \oplus_\Omega \{\phi\}$ requires $2 \cdot |S|$ tests.

Example 4. The Distributed Controller (DISC) is one of the tests coming from the HPWC category of the ARCH-COMP competition [15]; it models the distributed controller for a robot that reads and processes data from a number of sensors having different priorities. In Table 3 we show some statistics collected during the analysis of instance DISC03 of the benchmark (i.e., using 3 sensors, so that the automaton is defined on 11 variables and 258 locations), where we have prevented PHAVerLite from computing the poly hull approximation and rather maintain a finite set of polyhedra for each location.[6] Distinguishing between those calls that actually add ϕ to S and those calls that detect ϕ to be redundant, we report the total number of calls to the semantic operator $S \oplus \{\phi\}$, the resulting total number of inclusion tests performed, as well as their average number and the average size of S. By detecting and removing redundant elements in S, operator '$\oplus_\Omega$' is able to significantly reduce the average size of S; moreover, since some of the removed elements were in the "waiting list", they no longer need to be processed by the reachability algorithm, resulting in a significant decrease of the number of calls to $S \oplus_\Omega \{\phi\}$. As a result, the total number of inclusion tests is reduced by a factor of more than 20.

	ϕ not redundant (added)			ϕ redundant (not added)		
	$\oplus_{\mathrm{w}}$	$\oplus_\Omega$	ratio	$\oplus_{\mathrm{w}}$	$\oplus_\Omega$	ratio
calls to $S \oplus \{\phi\}$	63738	15131	4.2	109827	14945	7.3
total $\subseteq$ tests	79312223	4112746	19.3	11613424	331650	35.0
avg $\lvert S \rvert$	1244.3	135.9	9.2	792.0	113.1	7.0
avg $\subseteq$ tests	1244.3	271.8	4.6	105.7	22.2	4.8
total rem from S	0	9692	—	0	0	—

Table 3. The effectiveness of operators '$\oplus_{\mathrm{w}}$' and '$\oplus_\Omega$' on the DISC03 benchmark.

4.2 Improving efficiency of the inclusion tests

As seen in the previous section, Ω-reduction can significantly decrease the number of inclusion tests that need to be performed. In our quest for efficiency, the next step is to try and improve the efficiency of the inclusion test itself.

[6] This was done for exposition purposes, since this specific benchmark can be successfully verified, more efficiently, by using a single polyhedron for each location.

Assuming that we are computing on a polyhedra library based on the Double Description approach, the inclusion test $\phi_1 \subseteq \phi_2$ on polyhedra $\phi_1, \phi_2 \in \mathbb{P}_n$ is usually implemented by checking that all the generators of $\phi_1 = \mathrm{gen}(\mathcal{G}_1)$ satisfy all the constraints of $\phi_2 = \mathrm{con}(\mathcal{C}_2)$. In the worst case, this amounts to the computation of $|\mathcal{G}_1| \cdot |\mathcal{C}_2|$ scalar products, each one requiring $n+1$ multiplications and n additions of (arbitrary precision) integral coefficients, where n is the dimension of the vector space.

By observing again the data in Table 3, we can see that most of the inclusion tests are *failing*: for instance, in order to detect that ϕ is redundant in S (last but one column in the table), we perform 331650 inclusion tests, among which the successful ones are 14945, i.e., only 4.5%; things are even worse when ϕ is not redundant (3rd column of the table), since in this case we perform 4112746 inclusion tests, among which the successful ones are 9692, i.e., only 0.24%. Therefore, in order to improve efficiency, we look for heuristic procedures that allow to quickly identify cases when the polyhedra inclusion test will necessarily fail. To this end, we associate further abstractions to our polyhedra.

Definition 3. *The* bounding box *function* $\mathrm{bbox}\colon \mathbb{P}_n \to \mathbb{CB}_n$ *is defined, for each polyhedron* $\phi \in \mathbb{P}_n$*, as follows:*

$$\mathrm{bbox}(\phi) = \bigcap \{ B \in \mathbb{CB}_n \mid \phi \subseteq B \}.$$

Note that the bounding box is required to be *tight*, i.e., it is the most precise box in $\mathbb{CB}_n$ containing ϕ.

Lemma 1. *Let* $\phi_1, \phi_2 \in \mathbb{P}_n$*. If* $\mathrm{bbox}(\phi_1) \not\subseteq \mathrm{bbox}(\phi_2)$*, then* $\phi_1 \not\subseteq \phi_2$*.*

Thus, a correct (but incomplete) test for non-inclusion on $\mathbb{P}_n$ can be obtained by checking non-inclusion of the bounding boxes. Since non-inclusion on boxes can be checked by performing at most $2 \cdot n$ (arbitrary precision) extended rational comparisons, the efficiency gain with respect to the test on $\mathbb{P}_n$ may be significant.

The same approach can be iterated by further abstracting the bounding box information into an even lighter approximation.

Definition 4. *For each 1-dimensional box* $B \in \mathbb{CB}_1$*, the* pseudo volume *and the* number of rays *of* B *are defined as*

$$\mathrm{pvol}(B) \stackrel{\mathrm{def}}{=} \begin{cases} 0, & \text{if } B \text{ is empty;} \\ 1 + (ub - lb), & \text{if } B = [lb, ub] \neq \emptyset \text{ is bounded;} \\ +\infty, & \text{otherwise;} \end{cases}$$

$$\mathrm{nrays}(B) \stackrel{\mathrm{def}}{=} \begin{cases} 0, & \text{if } B = [lb, ub] \text{ is bounded;} \\ 2, & \text{if } B = \mathbb{R}; \\ 1, & \text{otherwise.} \end{cases}$$

These are extended to n*-dimensional boxes* $B \in \mathbb{CB}_n$ *as follows:*

$$\mathrm{pvol}(B) \stackrel{\mathrm{def}}{=} \prod_{i=1}^{n} \mathrm{pvol}(\pi_i(B)), \qquad \mathrm{nrays}(B) \stackrel{\mathrm{def}}{=} \sum_{i=1}^{n} \mathrm{nrays}(\pi_i(B)),$$

where $\pi_i(B)$ is the projection of box B on the i-th coordinate of the vector space.

Note that, in the definition of 'pvol', the systematic addition of 1 to the length of each 1-dimensional box is meant to force the computation of a positive pseudo volume even for those boxes having some projections of length zero.

Lemma 2. *Let $B_1, B_2 \in \mathbb{CB}_n$. If $\mathrm{pvol}(B_1) > \mathrm{pvol}(B_2)$ or $\mathrm{nrays}(B_1) > \mathrm{nrays}(B_2)$, then $B_1 \not\subseteq B_2$.*

Hence, a correct (but incomplete) test for non-inclusion on $\mathbb{CB}_n$ can be obtained by performing a constant number of comparisons.

This is the main idea behind the *boxed polyhedra* domain, where each polyhedron $\phi \in \mathbb{P}_n$ in the finite collection is matched by information on the corresponding bounding box. Note that the approach we are proposing can also be viewed as the application of a dynamic meta-analysis [11].

Definition 5. *A* boxed polyhedron *is a tuple $\langle v, r, B, \phi \rangle$ such that $\phi \in \mathbb{P}_n$, $B = \mathrm{bbox}(\phi)$, $r = \mathrm{nrays}(B)$ and $v = \mathrm{pvol}(B)$. The set of boxed polyhedra is partially ordered by the lexicographic composition of the orders defined on its components.*

In Table 4 we evaluate the effectiveness of Lemmas 1 and 2 in reducing the number of polyhedra inclusion tests for the DISC03 benchmark. Note that we are using the '$\oplus_\Omega$' operator, so that the total number of inclusion tests has already been reduced from 90.9M to 4.4M, as reported in Table 3. It can be seen that, for the considered benchmark, the semi-decision procedures are effective on more than 95% of the inclusion tests performed. Also note that the non-inclusion tests based on the number of rays never succeeds: this is due to the fact that, in test DISC03, all polyhedra happen to be polytopes.

	Lemma 2		Lemma 1	
	$v_1 > v_2$	$r_1 > r_2$	$B_1 \not\subseteq B_2$	$\phi_1 \not\subseteq \phi_2$
num tests	4444396	2274428	2274428	212357
$\subseteq$ decided	2169968	0	2062071	212357
%	48.82	0.00	46.40	4.78

Table 4. The effectiveness of the semi-decision procedures for inclusion tests on boxed polyhedra $\langle v_i, r_i, B_i, \phi_i \rangle$ for the DISC03 benchmark.

When implementing the inclusion test on boxed polyhedra, an optimization can be obtained even when Lemma 1 fails to apply.[7] In fact, by exploiting the knowledge that $\mathrm{bbox}(\phi_1) \subseteq \mathrm{bbox}(\phi_2)$, we can replace the full inclusion test $\phi_1 \subseteq \phi_2$ with a lighter one, where we avoid to check the interval constraints of ϕ_2 against the generators of ϕ_1. The effect of this heuristics can be significant: for

[7] Note that this implies that neither Lemma 2 applies.

instance, when computing the inclusion tests $\phi_1 \subseteq \phi_2$ for the DISC03 benchmark, about 70% of the constraints are interval constraints.

A further improvement can be obtained if the finite collections are sorted according to a suitable variant of the partial order relation defined on boxed polyhedra: this enables the application of binary search (rather than linear search) to quickly detect the subrange of boxed polyhedra that actually need to be checked for inclusion. In our experiments, we sorted these lists in increasing order of the v_i component (the bounding box pseudo-volume). For the DISC03 benchmark, this reduces the total number of tests for applicability of Lemma 2 by more than 90% (from 4444396 to 314722).

reduction	num iter	boxing	time	ratio
$\oplus_{\mathrm{w}}$	63805	unboxed	1492.79	141.77
		boxed	108.76	10.33
$\oplus_{\Omega}$	9625	unboxed	93.28	8.86
		boxed	10.53	1.00

Table 5. Comparing efficiency for the DISCS03 benchmark.

In Table 5 we report the timings obtained for the DISC03 benchmark when varying the reduction strategy and the choice of the powerset element (boxed or unboxed polyhedra). The improvements provided by the two techniques carry over to computation times; moreover, the two techniques provide almost orthogonal efficiency improvements.

In our implementation of boxed polyhedra, the bounding box information is computed on-demand and cached. Some care has to be taken to invalidate these caches after applying semantic operators that change the polyhedra.

4.3 Improving other operators

When adopting the finite powerset of boxed polyhedra, we can improve the efficiency of other semantic operators. For instance, consider the implementation of the lattice meet: as discussed before (see Equation 1), after the element-wise application of the polyhedra intersections, the resulting powerset needs to be checked for redundancies; in particular, some of the computed intersections $\phi_1 \cap \phi_2$ may be empty. In principle, the generation of these empty elements could be avoided by checking if the two arguments ϕ_1 and ϕ_2 are disjoint, but on the domain of polyhedra $\mathbb{P}_n$ this check happens to be as expensive as the computation of the intersection itself. With boxed polyhedra, the following result applies.

Lemma 3. *Let* $\phi_1, \phi_2 \in \mathbb{P}_n$. *If* $\mathrm{bbox}(\phi_1) \cap \mathrm{bbox}(\phi_2) = \emptyset$, *then* $\phi_1 \cap \phi_2 = \emptyset$.

Lemma 3 can be used, for instance, to quickly detect that $\phi_1 \in \mathrm{reach}(\ell_1)$ is disjoint from the guard component of an outgoing transition $\ell_1 \xrightarrow{a,\mu} \ell_2$ (i.e., the

transition is disabled). For the DISC03 benchmark this happens in about 34% of cases (15374 times on a total of 45514 checks). However, the efficiency gain obtained is negligible.

5 Splitting polyhedra

In Sections 3.2 and 4 we revisited different ways to model, in the abstract semantic construction, the merging of different execution paths. The efficient handling of this merge operator is quite often one of the main concerns when designing a static analysis tool. In this section, we turn our attention to the "dual" semantic operator, which intuitively *splits* an execution path in two branches.

In a classical (forward semantics) program static analysis, splitting typically occurs when approximating the effect of conditional branching: for instance, the analysis of an `if-then-else` statement splits the current approximation into a `then`-component (satisfying the conditional guard) and an `else`-component (satisfying its complement). In the context of the verification of hybrid systems, a similar semantic operator may be needed when a location state is partitioned according to some constraints, so as to better approximate a continuous flow relation which is not piecewise constant. Similarly, the split operator can be used in the *abstract solving* of a geometric CSP [29], where the current search space is partitioned into subdomains, refining the following propagation steps. Splits are also relevant for powerset domains: for instance, given two sets of polyhedra $S_1, S_2 \in \wp_{\mathrm{f}}(\mathbb{P}_n)$, the algorithm checking whether S_1 is *geometrically covered* by S_2, i.e., $(\cup\, S_1) \subseteq (\cup\, S_2)$, typically requires the splitting of those polyhedra in S_1 that are not included in a polyhedron in S_2.

Depending on the application and the underlying abstract domain, different variants of this operator may be defined.

Definition 6. *Let β be a non-strict linear inequality constraint on $\mathbb{R}^n$ and β' be the non-strict version of its (strict) complement $\neg\beta$. The* strict *and the* non-strict *split operators are defined, for each $\phi \in \mathbb{P}_n$, as* $\mathrm{split}^{\mathrm{s}}_{\beta}(\phi) = (\phi_1, \phi_2)$ *and* $\mathrm{split}^{\mathrm{ns}}_{\beta}(\phi) = (\phi_1, \phi'_2)$*, where* $\phi_1 = \phi \cap \mathrm{con}(\{\beta\})$*,* $\phi_2 = \phi \cap \mathrm{con}(\{\neg\beta\})$ *and* $\phi'_2 = \phi \cap \mathrm{con}(\{\beta'\})$.

Note that $\phi = \phi_1 \cup \phi_2 = \phi_1 \cup \phi'_2$; also, $\phi_1 \cap \phi_2 = \emptyset$, while ϕ_1 and ϕ'_2 may overlap. The non-strict operator '$\mathrm{split}^{\mathrm{ns}}_{\beta}$' can also be defined on the domain of topologically closed polyhedra $\mathbb{CP}_n$.

Available polyhedra libraries do not provide a direct implementation for the split operator: it is typically implemented by the user, by first cloning the input polyhedron and then separately adding the constraint β and its (strict or non-strict) complement to the constraint systems of the two polyhedra. Such an approach, however, easily results in a *duplication* of the computational work.

To see this, consider an implementation based on the Double Description method[8] and, for ease of exposition, consider the *non-strict* split operator applied

[8] To some extent, the reasoning should also apply to constraint-only representations, if the implementation attempts to identify and remove redundant constraints.

to a *closed* polyhedron $\phi \equiv \langle \mathcal{C}, \mathcal{G} \rangle \in \mathbb{CP}_n$. The addition of β to constraint system $\mathcal{C}$ requires a call to the incremental Chernikova conversion algorithm. The core of this procedure partitions the generator system $\mathcal{G}$ into $\mathcal{G}^+$, $\mathcal{G}^0$ and $\mathcal{G}^-$, according to the sign of the scalar product of each generator with β, and then linearly combines $\mathcal{G}^+$ and $\mathcal{G}^-$ to produce $\mathcal{G}^\star$; the resulting polyhedron $\phi_1 \equiv \langle \mathcal{C} \cup \{\beta\}, \mathcal{G}_1 \rangle$ is defined by the new generator system $\mathcal{G}_1 = \mathcal{G}^+ \cup \mathcal{G}^0 \cup \mathcal{G}^\star$. Since β and β' only differ in the sign of their coefficients, when adding β' to $\mathcal{C}$ (so as to obtain ϕ'_2) we end up recomputing the same partition of $\mathcal{G}$, modulo exchanging the roles of $\mathcal{G}^+$ and $\mathcal{G}^-$; also, the previously computed set $\mathcal{G}^\star$ can be reused as is, since the linear combination procedure is symmetric. Hence, $\phi'_2 \equiv \langle \mathcal{C} \cup \{\beta'\}, \mathcal{G}'_2 \rangle$ is easily obtained by reusing the computation done before, letting $\mathcal{G}'_2 = \mathcal{G}^- \cup \mathcal{G}^0 \cup \mathcal{G}^\star$, with no additional scalar products or linear combinations.

When encoding NNC polyhedra using the *direct* representation proposed in [6], the implementation of the strict operator '$\text{split}^{\text{s}}_\beta$' is more complicated,but it essentially preserves all of the computational savings mentioned above. This is not the case when the NNC polyhedra are encoded by using an additional slack variable [2,21], which is the classical approach implemented in Apron and PPL. In such a case, the NNC polyhedron $\phi \in \mathbb{P}_n$ would be encoded by a closed representation $\psi \in \mathbb{CP}_{n+1}$, violating a basic assumption underlying our optimization. The following example describes the problem in more detail.

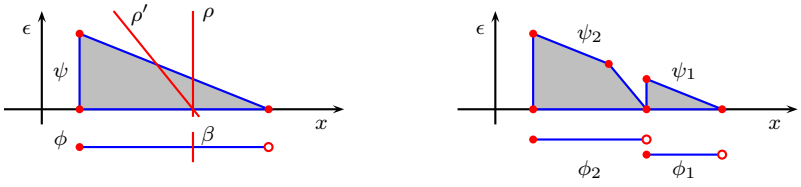

Fig. 2. When using the ϵ-representation approach, the complementary constraints β and $\neg\beta$ are encoded by ρ and ρ', which are not complementary in $\mathbb{R}^2$.

Example 5. On the upper left portion of Figure 2 we shown the topologically closed ϵ-representation $\psi \in \mathbb{CP}^2$ for the 1-dimensional, half-open interval $\phi = \text{con}(\{1 \leq x < 6\}) \in \mathbb{P}_1$, which is depicted below ψ. The (closure) points of the polyhedra are denoted by (unfilled) circles. Consider the constraint $\beta \equiv (x \geq 4)$, so that $\neg\beta \equiv (x < 4)$ and $\text{split}^{\text{s}}_\beta(\mathcal{P}) = (\phi_1, \phi_2)$, where $\phi_1 = \text{con}(\{4 \leq x < 6\})$ and $\phi_2 = \text{con}(\{1 \leq x < 4\})$, represented on the lower right portion of the figure. Working on the ϵ-representations though, the two (non-strict and strict) inequalities β and $\neg\beta$ are respectively encoded by $\rho \equiv (4 \leq x + 0 \cdot \epsilon)$ and $\rho' \equiv (x + \epsilon \leq 4)$, which are both non-strict and not complementary on $\mathbb{R}^2$. Hence, a proper computation of the split operator on the ϵ-representation ψ (shown on the upper right portion of Figure 2) requires two distinct calls to the incremental conversion procedure to obtain ψ_1 and ψ_2.

In Table 6 we show the efficiency of different implementations of the split operation: we compare the 'standard', user-defined implementation (on both the PPL and the PPLite libraries) with the newly defined abstract operator (only available in PPLite). The polyhedron chosen for the test is a half-open hypercube $\mathcal{H} \in \mathbb{P}_{12}$, defined by constraints of the form $-1 < x_i \leq 1$; for the tests on $\mathbb{CP}_{12}$ we use its topological closure $\mathrm{cl}(\mathcal{H})$; we also perform a test on $\mathcal{H}' \in \mathbb{P}_{12}$, which is obtained from $\mathcal{H}$ by adding three *non-skeleton* points [6] (that is, $\mathcal{H}'$ also contains the relative interior of three of the facets that are disjoint from $\mathcal{H}$). In all cases, the polyhedron is split by constraint $\beta \equiv (x_0 + 2x_1 - 2x_{10} - x_{11} \geq 0)$; when splitting $\mathcal{H}$, we test both the strict and non-strict variant of the split operation.

		$\mathrm{split}^{\mathrm{ns}}_{\beta}(\mathcal{H})$			$\mathrm{split}^{\mathrm{s}}_{\beta}(\mathcal{H})$			$\mathrm{split}^{\mathrm{ns}}_{\beta}(\mathrm{cl}(\mathcal{H}))$			$\mathrm{split}^{\mathrm{s}}_{\beta}(\mathcal{H}')$		
library	impl	time	vec	sat	time	vec	sat	time	vec	sat	time	vec	sat
PPL	standard	0.450	244	43.65	0.457	245	43.66	0.167	10	9.45	7.823	283	1742.13
PPLite	standard	0.068	10	4.73	0.069	10	4.73	0.066	10	4.73	0.070	10	4.81
	split	0.035	5	2.36	0.036	5	2.37	0.035	5	2.36	0.038	5	2.44

Table 6. Splitting $\mathcal{H}$, $\mathrm{cl}(\mathcal{H})$ and $\mathcal{H}'$ using β. Units: time (s), vec (K), sat (M).

In columns 'vec' and 'sat' we report the number of operations performed on vectors (scalar products and linear combinations) and saturation rows (population counts, unions and tests for inclusion on bit-vectors): it can be seen that the newly implemented operator systematically halves the values of these counters. Note that, since this test is characterized by low magnitude coefficients, the efficiency gain on vector operations is probably underestimated. The comparison with the PPL implementation confirms that libraries based on the ϵ-dimension approach are significantly less efficient, in particular when the input polyhedron contains non-skeleton constraints/generators.

6 Conclusion

Starting from PHAVer, we have developed a new tool PHAVerLite for the analysis of hybrid systems characterized by piecewise constant continuous dynamics. While revisiting the application of the domain of NNC polyhedra to the problem of computing or overapproximating the reachable states, we focused our attention on several well known abstract operators, showing that remarkable efficiency improvements can be obtained by providing implementations that are specialized for the considered context. For a more efficient handling of sets of polyhedra, we have proposed a new heuristic approach, where we couple each polyhedron in the set with information corresponding to further approximations (bounding box and pseudo volume). As future work, we plan to extend our investigation to other semantic operators, including those that are needed when extending the analysis to more general classes of hybrid systems.

Acknowledgment

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

References

1. Bagnara, R., Hill, P.M., Ricci, E., Zaffanella, E.: Precise widening operators for convex polyhedra. Science of Computer Programming **58**(1–2), 28–56 (2005)
2. Bagnara, R., Hill, P.M., Zaffanella, E.: Not necessarily closed convex polyhedra and the double description method. Formal Aspects of Computing **17**(2), 222–257 (2005)
3. Bagnara, R., Hill, P.M., Zaffanella, E.: Widening operators for powerset domains. Software Tools for Technology Transfer **8**(4/5), 449–466 (2006)
4. Bagnara, R., Hill, P.M., Zaffanella, E.: 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**(1–2), 3–21 (2008)
5. Bagnara, R., Hill, P.M., Zaffanella, E.: Applications of polyhedral computations to the analysis and verification of hardware and software systems. Theoretical Computer Science **410**(46), 4672–4691 (2009)
6. Becchi, A., Zaffanella, E.: A direct encoding for NNC polyhedra. In: Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. pp. 230–248 (2018)
7. Becchi, A., Zaffanella, E.: An efficient abstract domain for not necessarily closed polyhedra. In: Static Analysis - 25th International Symposium, SAS 2018, Freiburg, Germany, August 29-31, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11002, pp. 146–165. Springer (2018)
8. Bemporad, A., Fukuda, K., Torrisi, F.D.: Convexity recognition of the union of polyhedra. Computational Geometry: Theory and Applications **18**(3), 141–154 (2001)
9. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the Fourth Annual ACM Symposium on Principles of Programming Languages. pp. 238–252. ACM Press, Los Angeles, CA, USA (1977)
10. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Proceedings of the Sixth Annual ACM Symposium on Principles of Programming Languages. pp. 269–282. ACM Press, San Antonio, TX, USA (1979)
11. Cousot, P., Giacobazzi, R., Ranzato, F.: A^2I: Abstract^2 Interpretation. PACMPL **3**(POPL), 42:1–42:31 (2019)
12. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages. pp. 84–96. ACM Press, Tucson, Arizona (1978)
13. Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech. In: Hybrid Systems: Computation and Control: Proceedings of the 8th International Workshop (HSCC 2005). Lecture Notes in Computer Science, vol. 3414, pp. 258–273. Springer-Verlag, Berlin, Zürich, Switzerland (2005)
14. Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech. Software Tools for Technology Transfer **10**(3), 263–279 (2008)

15. Frehse, G., Abate, A., Adzkiya, D., Becchi, A., Bu, L., Cimatti, A., Giacobbe, M., Griggio, A., Mover, S., Mufid, M., Riouak, I., Tonetta, S., Zaffanella, E.: ARCH-COMP19 category report: Hybrid systems with piecewise constant dynamics. In: ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 61, pp. 1–13. EasyChair (2019)
16. Frehse, G., Abate, A., Adzkiya, D., Bu, L., Giacobbe, M., Mufid, M.S., Zaffanella, E.: ARCH-COMP18 category report: Hybrid systems with piecewise constant dynamics. In: ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 54, pp. 1–13. EasyChair (2018)
17. Frehse, G., Guernic, C.L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In: Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA. Proceedings. pp. 379–395 (2011)
18. Guernic, C.L., Girard, A.: Reachability analysis of hybrid systems using support functions. In: Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France. Proceedings. pp. 540–554 (2009)
19. Halbwachs, N., Merchat, D., Gonnord, L.: Some ways to reduce the space dimension in polyhedra computations. Formal Methods in System Design **29**(1), 79–95 (2006)
20. Halbwachs, N., Merchat, D., Parent-Vigouroux, C.: Cartesian factoring of polyhedra in linear relation analysis. In: Static Analysis: Proceedings of the 10th International Symposium. Lecture Notes in Computer Science, vol. 2694, pp. 355–365. Springer-Verlag, Berlin, San Diego, California, USA (2003)
21. Halbwachs, N., Proy, Y.E., Raymond, P.: Verification of linear hybrid systems by means of convex approximations. In: Static Analysis: Proceedings of the 1st International Symposium. Lecture Notes in Computer Science, vol. 864, pp. 223–237. Springer-Verlag, Berlin, Namur, Belgium (1994),
22. Halbwachs, N., Proy, Y.E., Roumanoff, P.: Verification of real-time systems using linear relation analysis. Formal Methods in System Design **11**(2), 157–185 (1997)
23. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: HyTech: A model checker for hybrid systems. Software Tools for Technology Transfer **1**(1+2), 110–122 (1997)
24. Jeannet, B., Miné, A.: Apron: A library of numerical abstract domains for static analysis. In: Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France. Proceedings. pp. 661–667 (2009)
25. Lamport, L.: A fast mutual exclusion algorithm. ACM Trans. Comput. Syst. **5**(1), 1–11 (1987)
26. Maréchal, A., Monniaux, D., Périn, M.: Scalable minimizing-operators on polyhedra via parametric linear programming. In: Static Analysis - 24th International Symposium, SAS 2017, New York, NY, USA, Proceedings. pp. 212–231 (2017)
27. Miné, A.: The octagon abstract domain. Higher-Order and Symbolic Computation **19**(1), 31–100 (2006)
28. Motzkin, T.S., Raiffa, H., Thompson, G.L., Thrall, R.M.: The double description method. In: Contributions to the Theory of Games – Volume II, pp. 51–73. No. 28 in Annals of Mathematics Studies, Princeton University Press, Princeton, New Jersey (1953)
29. Pelleau, M., Miné, A., Truchet, C., Benhamou, F.: A constraint solver based on abstract domains. In: Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy. Proceedings. pp. 434–454 (2013)

30. Sankaranarayanan, S., Colón, M., Sipma, H.B., Manna, Z.: Efficient strongly relational polyhedral analysis. In: Verification, Model Checking and Abstract Interpretation: Proceedings of the 7th International Conference (VMCAI 2006). Lecture Notes in Computer Science, vol. 3855, pp. 111–125. Springer-Verlag, Berlin, Charleston, SC, USA (2006)
31. Schupp, S., Ábrahám, E., Chen, X., Makhlouf, I.B., Frehse, G., Sankaranarayanan, S., Kowalewski, S.: Current challenges in the verification of hybrid systems. In: Cyber Physical Systems. Design, Modeling, and Evaluation - 5th International Workshop, CyPhy 2015, Proceedings. Lecture Notes in Computer Science, vol. 9361, pp. 8–24. Springer (2015)
32. Singh, G., Püschel, M., Vechev, M.T.: Fast polyhedra abstract domain. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017. pp. 46–59 (2017)