The development of the Verified Polyhedra Library

made of untrusted parts mixing Ocaml, C++, threads, ...

and just a little bit of Coq for certification

joint work, started in 2012, with

Sylvain Boulmé, Alexis Fouilhé, Alexandre Maréchal,
David Monniaux, Michaël Périn, Hang Yu

Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, France
The Verimag Verified Polyhedra Library (Features)

- OCAML implementation of standard **polyhedral operators** of a **relational abstract domain** (⊆, ⊇, ∩, elimination of variables, ...)

Explanation:

Verimag Verified Polyhedra Library

Introduction

The VPL

Certification

Why ? How ?

VPL correctness

Farkas Lemma

The certified checker

Certification in COQ

Usage in VPL

Computation

Representation of Polyhedra

New algorithms

Experiments

Ongoing Work

Conclusion

Related work
The Verimag Verified Polyhedra Library (Features)

- OCAML implementation of standard polyhedral operators of a relational abstract domain (\(\subseteq, \sqcup, \sqcap\), elimination of variables, ...)

- certifying library by a posteriori verification of each computation
  1. OCAML operators generate witnesses
  2. witness are checked by a simple COQ checker
The Verimag Verified Polyhedra Library

**Features**

- OCAML implementation of standard polyhedral operators of a relational abstract domain (\(\sqsubseteq, \sqcup, \sqcap\), elimination of variables, ...)

- Certifying library by a posteriori verification of each computation
  1. OCAML operators generate witnesses
  2. Witness are checked by a simple COQ checker

- Developed for the COQ-certified static analyzer VERASCO [Jourdan+, POPL’2015], a companion tool for COMPCERT C compiler [Leroy, JAR 2009]
The Verimag Verified Polyhedra Library (Features)

- OCAML implementation of standard **polyhedral operators** of a **relational abstract domain** ($\subseteq$, $\sqcup$, $\sqcap$, elimination of variables, ...)

- certifying library by a **posteriori verification** of each computation
  
  1. OCAML operators generate witnesses
  2. witness are checked by a simple COQ checker

- **developed for the COQ-certified static analyzer** VERASCO [Jourdan+, POPL’2015], a companion tool for **CompCert C compiler** [Leroy, JAR 2009]

- can be used as a **standalone OCAML library**, e.g. in the **FramaC** static analyzer [Buhler+VMCAI’2017]
The Verimag Verified Polyhedra Library (Features)

- OCAML implementation of standard polyhedral operators of a relational abstract domain (⊆, ⊔, ⊓, elimination of variables, ...)
- Certifying library by a posteriori verification of each computation
  1. OCAML operators generate witnesses
  2. Witness are checked by a simple COQ checker

- Developed for the COQ-certified static analyzer VERASCO [Jourdan+, POPL’2015], a companion tool for COMPCERT C compiler [Leroy, JAR 2009]

- Can be used as a standalone OCAML library, e.g. in the FRAMAC static analyzer [Buhler+VMCAI’2017]

- Used in a new COQ-tactic for simplifying affine expressions (S.Boulmé, A.Maréchal) (submitted)
The Verimag Verified Polyhedra Library

- polyhedra library in **constraint-only** representation
- new algorithms
  - precise polyhedral approximation of polynomial guards
  - minimization by raytracing
  - projection via Parametric Linear Programming
- novel certification approach using factories
  - correct by construction (by A. Maréchal, S. Boulmé)
- efficiency issues: parallelization, floating-point computations, external libraries (GLPK, GMP, EIGEN, FLINT), reconstruction of the exact solution (on \( \mathbb{Q} \))
- available at [github.com/VERIMAG-Polyhedra](https://github.com/VERIMAG-Polyhedra)
- state of the art **Parametric Linear Programming Solver**
Topics

- Polyhedra: basics
- All you need to understand the field of Polyhedra
- Farkas combinations and Linear Programming
- Why certifying software verification tools?
- Certification by result verification
- ... with as little CoQ as possible
- Why another polyhedra library?
- Why are polyhedra expensive?
- Revisiting the algorithmic
- Experimental results
- Will Polyhedra be usable?
Convex Polyhedra

capture **affine relations between program variables** such as

**inequalities:** \( x_1 - 2x_2 \geq 3x_3 \leadsto x_1 - 2x_2 - 3x_3 \geq 0 \)

**boundaries:** \( 2 \leq x_1 \leq 3 \leadsto x_1 - 2 \geq 0 \land -x_1 - 3 \geq 0 \)

**equalities:** \( x_1 = x_2 + 2 \leadsto \{ \begin{array}{l} x_1 - x_2 - 2 \geq 0 \\ x_2 - x_1 + 2 \geq 0 \end{array} \} \)

**affine form = linear form + constant**

**Definition**

A **convex polyhedron** is a set of vectors \((x_1, ..., x_n) \in \mathbb{Q}^n\) satisfying

**a system of affine inequalities between variables** \(x_1, ..., x_n\)

**Remark (It is convex)**

if two points are in the set, the segment also is.
A 3D polyhedron ...

... as system of constraints

\[
\begin{align*}
C_1 : & \quad x_1 + 2x_2 - 2x_3 & \leq & \quad 7, \\
C_2 : & \quad x_1 - 2x_2 & \leq & \quad -1, \\
C_3 : & \quad -3x_1 + x_2 & \leq & \quad 0, \\
C_4 : & \quad x_3 & \leq & \quad 10, \\
C_5 : & \quad -x_1 - x_2 - x_3 & \leq & \quad -5
\end{align*}
\]
Uses of Polyhedra

- **Linear Programming**
  - optimization of a cost function under affine inequalities,
  - decide the existence of a solution fulfilling affine inequalities
Uses of Polyhedra

- **Linear Programming**
  - optimization of a cost function under affine inequalities,
  - decide the existence of a solution fulfilling affine inequalities

- **Loop Optimization**
  “The Polyhedron Model” [Feautrier and Lengauer, 2011]
  1. approximate by a polyhedron the cells of a $n$-dimensions array to be updated by a loop
  2. compute vectors that exactly describe that space of cells
  3. generate the optimized loop
Uses of Polyhedra

- **Linear Programming**
  - optimization of a cost function under affine inequalities,
  - decide the existence of a solution fulfilling affine inequalities

- **Loop Optimization**
  “The Polyhedron Model” [Feautrier and Lengauer, 2011]
  1. approximate by a polyhedron the cells of a $n$-dimensions array to be updated by a loop
  2. compute vectors that exactly describe that space of cells
  3. generate the optimized loop

- **Static Analysis of Programs**
  POPL’78 [Cousot and Halbwachs, 1978]
  - capture affine relation between variables
  - discover implicit equalities
  - more precise than interval analysis but costlier
Why certifying results of Software Verification Tools?

An old greek syllogism

Programs contain bugs.
Software Verification Tools are programs.
Thus, ___ s contain ___ s
Why certifying results of Software Verification Tools?

An old greek syllogism

Programs contain bugs.
Software Verification Tools are programs.
Thus, SVTs contain BUGs
Why certifying results of Software Verification Tools?

An old greek syllogism

Programs contain bugs. 
Software Verification Tools are programs. 
Thus, SVTs contain BUGs

... more than other programs

- mostly prototypes developed by several students
- complex underlying theory
- less users, less tested
Three ways to gain confidence in SVT

1. **No tool is trustable ... but if they agree on the result**
   - Running each tool with (same inputs $\rightarrow$ same answer) increases the confidence
   - **Quantifiable?** How many **tools** to reach P.Failure=$10^{-9}$?
Three ways to gain confidence in SVT

1. **No tool is trustable ... but if they agree on the result**
   - Running each tool with (same inputs $\Rightarrow$ same answer) increases the confidence
   - **Quantifiable?** How many tools to reach P.Failure=$10^{-9}$?

2. **Only trust the proof-checker ... which becomes critical**
   - extend SVT to generate certificates
     - SAT, UNSAT, $\Rightarrow_{Theory}$, $[\text{Program}] \models \text{Properties}$
     - $[...]$ is a formal semantics of the programming language
       - (e.g. CompCert C semantics in Coq)
     - How long can a bug stay silent in the proof-checker?
       - (the Coq-engine is not as simple as it used to be)
Three ways to gain confidence in SVT

1. **No tool is trustable ... but if they agree on the result**
   - Running each tool with (same inputs ⇨ same answer) increases the confidence
   - **Quantifiable?** How many tools to reach P.Failure=10^{-9}?

2. **Only trust the proof-checker ... which becomes critical**
   - extend SVT to **generate certificates**
     SAT, UNSAT, \( \rightarrow_{Theory} \) \([\text{Program}] \models \text{Properties}\)
   - [...] is a formal semantics of the programming language (e.g. **CompCert** C semantics in **COQ**)
   - How long **can a bug stay silent** in the proof-checker? (the COQ-engine is not as simple as it used to be)

3. **Correct by extraction using** **COQ** (e.g. **CompCert**)
   \[\text{proof(algo } \models \text{spec) \underbrace{\text{extraction}}_{\text{in COQ}}} \rightarrow \text{OCAML program}\]
Certification versus Result Verification

**Full Coq-certified development (CompCert, Versasco S.A)**
- time consuming (refactoring the code means adapting the proofs)
- requires proof skills
- the algorithms must be designed to be easy to prove
- correctness of all results **guaranteed** by a Coq-proof

**Result verification (CompCert, B.S.A [Besson et al., 2010], VPL)**
- external libraries (untrusted code)
- correctness of each result is checked by Coq
  1. external code generates witness of correctness
  2. verification by a simple Coq-certified checker
What must be proved for over-approximations?

- **join/union** operator \((P' := P_1 \cup P_2)\) is sound if \(P_1 \subseteq P'\) and \(P_2 \subseteq P'\).

- **meet/intersection** operator \((P' := P_1 \cap P_2)\) is sound if \(P' \subseteq P_1\) and \(P' \subseteq P_2\).

- **minimization** operator \((P' := \text{min}\ (P))\) is sound if \(P \subseteq P'\) and **precise** if \(P' \subseteq P\).

- **elimination/projection** \((P' := \text{elim}\ \{x\}\ P)\) is sound if \(P \subseteq P'\) and \(x\) is unbounded in \(P'\).

Soundness boils down to inclusion [Besson et al., 2010]

\[P_1 \subseteq P_2\] can be proved by Farkas combinations.

Each operator must provide **Farkas combinations** to prove the required inclusions.
Farkas combinations I

Example

\[ P = \left\{ \begin{array}{l} C_1 : x_1 - x_2 - 1 \geq 0, \\ C_2 : x_1 + 2x_2 + 1 \geq 0 \end{array} \right\} \]

\[ P' = \{ C' : 3x_1 - 1 \geq 0 \} \]

The Farkas Combination \( 2 \times C_1 + 1 \times C_2 \) ... is \( C' \)

\[ \begin{align*}
2x_1 - 2x_2 - 2 & \geq 0 \\
1x_1 + 2x_2 + 1 & \geq 0
\end{align*} \]

\[ \equiv \begin{align*}
3x_1 - 1 & \geq 0
\end{align*} \]

It shows that \( \{ C_1, C_2 \} \sqsubseteq \{ C' \} \)
Farkas combinations II

Farkas combination of constraints (linear version)

\( x \in P := \{ C_1, \ldots, C_k \} \) means

\( x \) satisfies \( C_1(x) \geq 0 \land \ldots \land C_k(x) \geq 0 \) then,

for any non-negative scalars \( \lambda_1, \ldots, \lambda_k \in \mathbb{Q} \)

\[
\lambda_1 \times C_1(x) + \ldots + \lambda_k \times C_k(x) \geq 0
\]

\( \geq 0 \) \qquad \( \geq 0 \)
Farkas combinations II

Farkas combination of constraints (linear version)

\( x \in P := \{ C_1, \ldots, C_k \} \) means
\( x \) satisfies \( C_1(x) \geq 0 \land \ldots \land C_k(x) \geq 0 \) then, for any non-negative scalars \( \lambda_1, \ldots, \lambda_k \in \mathbb{Q} \)

\[ \underbrace{\lambda_1 \times C_1(x)}_{\geq 0} + \ldots + \underbrace{\lambda_k \times C_k(x)}_{\geq 0} \geq 0 \]

Now, if \( C' = \lambda_1 \times C_1 + \ldots + \lambda_k \times C_k \) then

\[ \forall x, \; C_1(x) \geq 0 \land \ldots \land C_k(x) \geq 0 \implies C'(x) \geq 0 \]

which is the definition of the geometric inclusion

\[ \{ C_1, \ldots, C_k \} \subseteq \{ C' \} \]
Farkas combinations III

How to find the Farkas coefficient $\lambda$’s

\[
P = \begin{cases} 
C_1 : & 1x_1 -1x_2 -1 \geq 0, \\
C_2 : & 1x_1 +2x_2 +1 \geq 0, \\
C_3 : & -1x_1 +1x_2 +1 \geq 0 
\end{cases}
\]

\[
P' = \begin{cases} 
C' : & 3x_1 +0x_2 -1 \geq 0 
\end{cases}
\]

\[
\exists \lambda_i \geq 0 \quad \lambda_1 \times C_1 + \lambda_2 \times C_2 + \lambda_3 \times C_3 = C'
\]
Farkas combinations III

How to find the Farkas coefficient $\lambda$’s

\[
P = \left\{ \begin{array}{l}
C_1 : \quad 1x_1 - 1x_2 - 1 \geq 0, \\
C_2 : \quad 1x_1 + 2x_2 + 1 \geq 0, \\
C_3 : \quad -1x_1 + 1x_2 + 1 \geq 0
\end{array} \right.
\]

\[
P' = \left\{ \begin{array}{l}
C' : \quad 3x_1 + 0x_2 - 1 \geq 0
\end{array} \right.
\]

\[
\exists \lambda_i \geq 0 \quad \lambda_1 \times C_1 + \lambda_2 \times C_2 + \lambda_3 \times C_3 = C'
\]

\[
(x_1) \quad \lambda_1 \times \mathbf{1} + \lambda_2 \times \mathbf{1} + \lambda_3 \times -\mathbf{1} = 3
\]
Farkas combinations III

How to find the Farkas coefficient \( \lambda \)'s

\[
P = \begin{cases} 
  C_1 : & 1 \ x_1 - 1 \ x_2 - 1 \geq 0, \\
  C_2 : & 1 \ x_1 + 2 \ x_2 + 1 \geq 0, \\
  C_3 : & -1 \ x_1 + 1 \ x_2 + 1 \geq 0 
\end{cases}
\]

\[
P' = \begin{cases} 
  C' : & 3 \ x_1 + 0 \ x_2 - 1 \geq 0 
\end{cases}
\]

\[
\exists \lambda_i \geq 0 \quad \lambda_1 \times C_1 + \lambda_2 \times C_2 + \lambda_3 \times C_3 = C'
\]

\[
\begin{align*}
(x_1) & : \quad \lambda_1 \times 1 \quad + \quad \lambda_2 \times 1 \quad + \quad \lambda_3 \times -1 \quad = \quad 3 \\
(x_2) & : \quad \lambda_1 \times -1 \quad + \quad \lambda_2 \times 2 \quad + \quad \lambda_3 \times 1 \quad = \quad 0
\end{align*}
\]
Farkas combinations III

How to find the Farkas coefficient $\lambda$’s

\[ P = \begin{cases} 
C_1 : & 1 \ x_1 - 1 \ x_2 - 1 \geq 0, \\
C_2 : & 1 \ x_1 + 2 \ x_2 + 1 \geq 0, \\
C_3 : & -1 \ x_1 + 1 \ x_2 + 1 \geq 0 
\end{cases} \]

\[ P' = \{ C' : 3 \ x_1 + 0 \ x_2 - 1 \geq 0 \} \]

\[ \exists \lambda_i \geq 0 \quad \lambda_1 \times C_1 + \lambda_2 \times C_2 + \lambda_3 \times C_3 = C' \]

\[
\begin{align*}
(x_1) & \quad \lambda_1 \times 1 + \lambda_2 \times 1 + \lambda_3 \times -1 = 3 \\
(x_2) & \quad \lambda_1 \times -1 + \lambda_2 \times 2 + \lambda_3 \times 1 = 0 \\
(cst) & \quad \lambda_1 \times -1 + \lambda_2 \times 1 + \lambda_3 \times 1 = -1
\end{align*}
\]
Linear Algebra / Linear Programming

Gauss Resolution (Linear Algebra)

It gives solutions of the systems of equalities

\[
\begin{align*}
(x_1) & \quad \lambda_3 = -3 + \lambda_1 + \lambda_2 \\
(x_2) & \quad \lambda_2 = 1 \\
(cst) & \quad 0 \times \lambda_1 = 0 \quad \lambda_1 \text{ is free, thus choose } \lambda_1 \geq 0
\end{align*}
\]

But some are not Farkas Combinations i.e. satisfying \( \lambda_i \geq 0 \)

\[
(\lambda_1, \lambda_2, \lambda_3) = \{(0, 1, -2), (1, 1, -1), (2, 1, 0), (3, 1, 1), \ldots \}
\]

\[\exists \lambda_i \geq 0 \ldots \text{ is not in the realm of Linear Algebra but that of Linear Programming}\]
The Simplex Algorithm (Linear Programming)

It’s a way to choose pivots in Gauss elimination

\[
\begin{align*}
(x_1) \quad & \lambda_1 = 3 - \lambda_2 + \lambda_3 \\
(x_2) \quad & \lambda_2 = 1 \\
cst \quad & 0 \times \lambda_3 = 0 \quad \lambda_3 \text{ is free, thus choose } \lambda_3 \geq 0
\end{align*}
\]

It focuses on Farkas Combinations i.e. satisfying \(\lambda_i \geq 0\)

\[
(\lambda_1, \lambda_2, \lambda_3) = \{ (2, 1, 0), (3, 1, 1), (4, 1, 2), (5, 1, 3), \ldots \}
\]
Efficient floating-point simplex algorithms (such as GLPK) do not provide exact solution (due to rounding)

\[(\lambda_1, \lambda_2, \lambda_3) = (1.99..., 1.0, 0.0...1)\]

But they are trustable on variables that must be null (e.g. \(\lambda_3 = 0\)) from which we can use fast linear algorithm over the rationals (FLINT) to solve the simplified system and obtain an exact solution \((\lambda_1, \lambda_2, \lambda_3) = (2, 1, 0)\)
Polyhedra inclusion checker in COQ

1. **Original work in 2010** [Besson et al., 2010]
   - results checking of Bytecode Static Analyzer
   - operations are performed by NewPolka
     [Jeannet and Miné, 2009]
   - witnesses are computed afterwards by solving Linear Programming problems

2. **VPL, started in 2012**
   - produces witnesses on-the-fly (no duplicate computation)
   - constraint-only representation
The inclusion checker (Coq code) (Definitions & Lemma)

**Definition** Polyhedra := list (cstr Q).

**Definition** sat (x : Vec) (p : Polyhedra) : Prop := List.Forall p (fun (c : csrt Q) → c(x) ≥ 0).

**Definition** (infix ⊑) (p₁ p₂ : Polyhedra) : Prop := ∀ x : Vec, sat x p₁ ⇒ sat x p₂.

**Lemma** Farkas : ∀ (Λ : list (list Q)) (p₁ p₂ : Polyhedra) (∀ λ ∈ Λ, λ ≥ 0) ∧ (combine Λ p₁) = p₂ → p₁ ⊑ p₂
The inclusion checker \((\texttt{COQ code})\) (Program & Extraction)

**Definition** *inclusion_checker*

\[
(\Lambda : \text{list (list } \mathbb{Q}) \rightarrow (p_1 \ p_2 : \text{Polyhedra}) : \text{option } (p_1 \subseteq p_2) :=
\]

\[
\text{let } \text{nn} := (\text{non_negative } \Lambda) \text{ in}
\]

\[
\text{let } eq := (\text{equal } (\text{combine } \Lambda p_1) p_2) \text{ in}
\]

\[
\text{match } (\text{nn},eq) \text{ with}
\]

\[
\mid (\text{Some } \text{proof\_nn}, \text{Some } \text{proof\_eq}) \rightarrow \text{Some } (\text{Farkas } \Lambda p_1 p_2 \text{ proof\_nn proof\_eq})
\]

\[
\mid (_,_) \rightarrow \text{None}
\]

end

\[\text{COQ inclusion\_checker} : (\Lambda, p_1, p_2) \rightarrow \begin{cases} 
\text{Some } (p_1 \subseteq p_2) \\
\text{None}
\end{cases}\]

\[\downarrow \text{extraction}\]

\[\text{OCAML inclusion\_checker} : (\Lambda, p_1, p_2) \rightarrow \text{bool}\]
Using the COQ checker in OCAML code

Illustration on the convex-hull operator of the VPL

The convex-hull operator $P := P_1 \sqcup P_2$ is sound if

$$P_1 \subseteq P \text{ and } P_2 \subseteq P$$

which is proved using two Farkas inclusion witnesses $\Lambda_1$ and $\Lambda_2$ using

$$\text{inclusion_checker} (\Lambda_1, P_1, P) = \text{Some}(P_1 \subseteq P)$$

$$\text{inclusion_checker} (\Lambda_2, P_2, P) = \text{Some}(P_2 \subseteq P)$$
The convex-hull (OCAML code)

```ocaml
type polyhedra = { ocaml: (rat cstr) list ; coq: (Q cstr) list };

let convex_hull (p1:polyhedra) (p2:polyhedra) : polyhedra =
  let (f1, f2, pOcaml) =
    untrusted_convex_hull p1.ocaml p2.ocaml
  in let Λ1 = rat_to_Q f1
      Λ2 = rat_to_Q f2
      pCoq = rat_to_Q pOcaml
  in if (inclusion_checker Λ1 p1.coq pCoq)
    && (inclusion_checker Λ2 p2.coq pCoq)
  then { ocaml = pOcaml ; coq = pCoq }
  else error "convex_hull"
```
What is guaranteed by result verification?

- The checker is extracted in OCaml but still uses Coq representations (trustable but inefficient)
  
  ```coq
  type nat = O | S of nat
  type positive = B1 of positive | B0 of positive | BH
  e.g. 5 ≃ S(S(S(S(O)))) ≃ B1(B0(B1(BH)))
  ```

- **12% overhead when the Coq checker is activated**
  (conversion into Coq representation then computations)

- **The Coq checker can be de/activated.**

- **The equality (p.ocaml = p.coq) cannot be guaranteed**

- **Guaranty:** the Coq side mimics the computations of the untrusted side and the Coq side checks soundness

- S.Boulmé noticed that "the Coq type of polyhedra can even be an opaque abstract data type or a generic type 'α" leading to new certification means using factories."
The Double Description of Polyhedra

used in most Polyhedra Libraries (NewPolka, PPL, Cudd, ...)

\[
\begin{pmatrix}
\text{as constraints} \\
\{ C_1, C_2, C_3, C_4 \} \\
\text{as generators} \\
\{ v_1, v_2, v_3 \} + \{ R_1 \}
\end{pmatrix}
\]

vertices + rays
Why are polyhedra expensive? 

Polyhedra as generators: \( G \cap \{ C' \} \)

The intersection with one constraint can **double the number of generators**.

Example (A sliced tube unbounded in one direction)

\[
G = \{ v_1, \ldots, v_n \} + \{ r_1 \} \\
G \cap \{ C' \} = \{ v_1, \ldots, v_n \} + \{ v'_1, \ldots, v'_n \}
\]
Why are polyhedra expensive? II

Polyhedra as constraints: \( \text{elim} (x_3, \mathcal{C}) = \text{projection on } (x_1, x_2) \)

The elimination of one variable can double the number of constraints

Example (An orange segment)

\[
\mathcal{C} = \{ C', C'', C_1, \ldots, C_n \} \\
\text{elim} (x_3, \mathcal{C}) = \{ C'_1, \ldots, C'_n, C''_1, \ldots, C''_n \} \\
\text{elim} (\{ x_3, x_2 \}, \mathcal{C}) = \{ b \leq x_1, x_1 \leq b' \}
\]
Choosing the good representation $C$? $G$?

- The polyhedra representation can double on basic operations ($\sqcap$, elim)
- Sequential eliminations of variables is exponential on constraints $\text{elim} \ [x_1 ; \ldots ; x_n] \ C$
  based on Fourier-Motzkin’s elimination of one variable
- Sequential intersections is exponential on generators
  $$G \sqcap [C_1 ; \ldots ; C_k]$$
  based on Chernikova’s intersection with one cutting plane
- $C$ is needed for intersection and widening, used for inclusion and minimization
- DD can choose the best algorithm or an even better algorithm using $(C, G)$
Why a polyhedra library in constraint-only?

- no polynomial algorithm to check equivalence \((\mathcal{C}, \mathcal{G})\)
  (it probably does not exist)
- DD would have meant
  - implementing a naive version of Chernikova’s conversion algorithm in COQ,
  - proving it correct then
  - extracting to OCAML to get a correct but inefficient algorithm
- out of curiosity: no conversion, less memory space, can it be as efficient as DD?
- could we do better than Fourier-Motzkin one-variable elimination algorithm?  
  [Howe and King, 2012]: Parametric Linear Programming can eliminate several variables simultaneously
- could we improve minimization?
Minimization by Raytracing

Launch a ray orthogonally to each constraint

the first constraint hit by the ray is irredundant
Minimization by Raytracing

Launch a ray orthogonally to each constraint

the first constraint hit by the ray is irredundant
Minimization by Raytracing

Launch a ray orthogonally to each constraint

the first constraint hit by the ray is irredundant
Minimization by Raytracing

Launch a ray orthogonally to each constraint

the first constraint hit by the ray is irredundant
Minimization by Raytracing

Launch a ray orthogonally to each constraint

the first constraint hit by the ray is irredundant
Minimization by Raytracing

Launch a ray orthogonally to each constraint

the first constraint hit by the ray is irredundant
Minimization by Raytracing

Launch a ray orthogonally to each constraint

the first constraint hit by the ray is irredundant
When raytracing fails

We use the Standard Minimization Algorithm

\[ \begin{align*}
\{ C_3, C_5 \} \subseteq \{ C_4 \}? & \quad \text{yes} \\
\{ C_1, C_2 \} \subseteq \{ C_6 \}? & \quad \text{no}
\end{align*} \]

Finally,
\[ C_1, C_2, C_3, C_5 \] were determined without solving LP problems, it only costs a matrix-matrix product:

\[ \text{matrix of constraints} \times \text{matrix of rays} \]
Experiments

We compared three algorithms:

1. **The standard algorithm (SMA)**
   Detecting redundancies by finding Farkas combinations. Requires one Linear Programming for each constraint. Each Linear Programming contains all the constraints.

2. **Raytracing with rationals (RRT)**
   using a rational simplex for finding Farkas combination

3. **Raytracing with floating points (FRT)**
   using a floating simplex (GLPK) then reconstruction
Experiments on random polyhedra

We generated random polyhedra to study the sensitivity of algorithms to

- the number of variables
- the number of constraints
- the number of generators
- the redundancy rate
- the density

Example (6 variables, density 50%)

\[0 \times x_1 + 21x_2 + 0 \times x_3 + 26x_4 + 0 \times x_5 - 13x_6 \leq 20\]
Experimental results

Time (ms) when varying the number of constraints, with 10 variables, a redundancy rate of 50%, and a density of 50%.
Comparison with Chernikova’s conversion algorithm

Time (ms) in log scale
Experimentation is not easy

- **Comparing libraries: How to be fair?**
  - DD delay some operations (conversion, minimization)
  - start by a conversion to build \((C, G)\),
  - \(n\)-dimensional hypercubes \((2^n \text{ generators})\) kill them

- **Using which analyzer?**
  static analyzers have been designed for intervals, not for polyhedra (Frama-C, VERASCO)
  - too much (useless) variables involved
  - duplication of computations
  - \((P \sqcup P') = P'\) instead of \(P \subseteq P'\)
An experimental setup is under development

**Goal:** Profile each operator on random polyhedra to study sensitivity to the number of variables, of constraints, of generators, of redundancies, and the density (number of nonzero coefficients)

- record the call to polyhedral operators during analysis of realistic programs with a polyhedra-aware static analyzer
- extract significant sequences of operations, *e.g.*

\[
DD \ ; \ timer-start \ ; (\subseteq \ ; \cap ; \ := \ ; \sqcup)^* \ ; \ min \ ; \ timer-stop
\]

- run the sequence on each library with random inputs
Parallelization, floating-point computations then reconstruction

- minimization by raytracing
  independant determination of each constraint
- the Solver of Parametric Linear Problems
  \((C, C++, GLPK, FLINT, EIGEN, COQ, OCAMLM)\)
- new algorithm for inclusion
Will someday Polyhedra be usable?

May be with

- a polyhedra-aware static analyzer
- polyhedra used in a second phase where intervals failed
- dynamic packing of variables, removal of useless variable
- algorithms in constraint-only can benefit libraries in DD: the costly Chernikova’s conversion can be delayed
Will someday Polyhedra be usable?

The field of polyhedra still makes progress

Nice result by [Singh et al., 2017] based on [Halbwachs et al., 2006] to cope with the hypercube phenomenon:

\[ \mathcal{H} = \text{DD}(\bigwedge_{i=1}^{n} -1 \leq x_i \leq 1) \]
\[ = (2 \times n \text{ constraints, } 2^n \text{ generators}) \]

- The Fast Polyhedra Abstract Domain automatically splits polyhedra into a cartesian product during the analysis.
- Fast polyhedra almost behave like intervals when variables are not related.

Cartesian product of polyhedra [Halbwachs+1996, Singh+2015]

\[ \mathcal{H} = P_1 \times \ldots \times P_n \text{ where } P_i = (\{ -1 \leq x_i \leq 1 \}, \{-1, 1\}) \]
References of inspiring work I


References of inspiring work II

APRON: A library of numerical abstract domains for static analysis.
In Computer Aided Verification (CAV), volume 5643 of LNCS, pages 661–667.

On polyhedral projections and parametric programming.

Symbolic methods to enhance the precision of numerical abstract domains.
In Verification, Model Checking, and Abstract Interpretation (VMCAI), volume 3855 of LNCS, pages 348–363. Springer.

Fast polyhedra abstract domain.

Theorems for free!
## Publications related to the VPL

### on the algorithmic side of the VPL

<table>
<thead>
<tr>
<th>Conference</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS’2013</td>
<td>Efficient generation of correctness certificates for the abstract domain of polyhedra</td>
</tr>
<tr>
<td>VMCAI’2016</td>
<td>Polyhedral Approximation of Multivariate Polynomials using Handelman’s Theorem</td>
</tr>
<tr>
<td>VMCAI’2017</td>
<td>Efficient Elimination of Redundancies in Polyhedra using Raytracing</td>
</tr>
<tr>
<td>SAS’2017</td>
<td>Scalable Minimizing-Operators on Polyhedra via Parametric Linear Programming</td>
</tr>
</tbody>
</table>

### on the COQ side of the VPL

<table>
<thead>
<tr>
<th>Conference</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITP’2015</td>
<td>Refinement to Certify Abstract Interpretations, Illustrated on Linearization for Polyhedra</td>
</tr>
<tr>
<td>VSTTE’2014</td>
<td>A Certifying Frontend for (Sub)polyhedral Abstract Domains</td>
</tr>
</tbody>
</table>