A Coq Tactic for Equality Learning in Linear Arithmetic [⋆]

Sylvain Boulmé[13] and Alexandre Maréchal[24]

[1] Univ. Grenoble-Alpes, CNRS, Grenoble INP, VERIMAG, 38000 Grenoble, France
[2] Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, F-75005 Paris, France
[3] `sylvain.boulme@univ-grenoble-alpes.fr`
[4] `alexandre.marechal@lip6.fr`

Abstract. Coq provides linear arithmetic tactics such as `omega` or `lia`. Currently, these tactics either fully prove the current goal in progress, or fail. We propose to improve this behavior: when the goal is not provable in linear arithmetic, we strengthen the hypotheses with new equalities discovered from the linear inequalities. These equalities may help other Coq tactics to discharge the goal. In other words, we apply – in interactive proofs – a seminal idea of SMT-solving: combining tactics by exchanging equalities. The paper describes how we have implemented equality learning in a new Coq tactic, dealing with linear arithmetic over rationals. It also illustrates how this tactic interacts with other Coq tactics.

1 Introduction

Several Coq tactics solve goals containing linear inequalities: `omega` and `lia` on integers; `fourier` or `lra` on reals and rationals [22,4]. This paper provides yet another tactic for proving such goals on *rationals*. This tactic – called `vpl`[5] – is built on the top of the *Verified Polyhedra Library* (VPL), a Coq-certified abstract domain of convex polyhedra [14,15]. Its main feature appears when it *cannot prove* the goal. In this case, whereas above tactics fail, our tactic "simplifies" the goal. In particular, it injects as hypotheses a *complete* set of linear equalities that are deduced from the linear inequalities in the context. Then, many Coq tactics – like `congruence`, `field` or even `auto` – can exploit these equalities, even if they cannot deduce them from the initial context by themselves. By simplifying the goal, our tactic both improves the user experience and proof automation.

Let us illustrate this feature on the following – almost trivial – Coq goal, where `Qc` is the type of rationals on which our tactic applies.

```
Lemma ex1 (x:Qc) (f:Qc → Qc):  x≤1 → (f x)<(f 1) → x<1.
```

[⋆] This work was partially supported by the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement nr. 306595 "STATOR".

[5] Available at http://github.com/VERIMAG-Polyhedra/VplTactic.

This goal is valid on `Qc` and `Z`, but both `omega` and `lia` fail on the `Z` instance without providing any help to the user. Indeed, since this goal contains an uninterpreted function `f`, it does not fit into the pure linear arithmetic fragment. On the contrary, this goal is proved by two successive calls to the `vpl` tactic. As detailed below, equality learning plays a crucial role in this proof: the rewriting of a learned equality inside a non-linear term (because under symbol `f`) is interleaved between deduction steps in linear arithmetic. Of course, such a goal is also provable in `Z` by SMT-solving tactics: the `verit` tactic of SMTCoq [2], the `hammer` tactic of CoqHammer [11], or the one of Besson et al. [5]. However, such SMT-tactics are also "*prove-or-fail*": they do not simplify the goal when they cannot prove it. On the contrary, our tactic may help users in their interactive proofs, by simplifying goals that do not fully fit into the scope of existing SMT-solving procedures. Note that our tactic does not intend to compete in speed and power with SMT-based procedures. It mainly aims to ease *interactive proofs* which involve linear arithmetic.

In short, this paper provides three contributions. First, we provide a Coq tactic with equality learning, which seems a new idea in the Coq community. Second, we provide a simple and efficient algorithm which learns these equalities from conflicts between strict inequalities detected by a linear programming solver. On most cases, it is strictly more efficient than the naive equality learning algorithm previously implemented in the VPL [14]. In particular, our algorithm is cheap when there is no equality to learn. At last, we have implemented this algorithm in an OCaml oracle, able to produce *proof witnesses* for these equalities. The paper partially details this process, and in particular, how the *proof* of the learned equalities is computed in Coq by reflection from these witnesses. Actually, we believe that our tactic could be easily adapted to other interactive provers, and, in particular, our oracle could be directly reused.

The paper follows a "top-down" presentation. Section 2 describes the specification of the `vpl` tactic. It also introduces a high-level specification of its underlying oracle. Section 3 illustrates our tactic on a non-trivial example and in particular how it collaborates with other tactics through equality learning. Section 4 details the certificate format produced by our oracle, and how it is applied in our Coq tactic. At last, Section 5 details the algorithm we developed to produce such certificates.

2 Specification of the VPL Tactic

Let us now introduce the specification of the `vpl` tactic. As mentioned above, the core of the tactic is performed by an oracle programmed in OCaml, and called `reduce`. This oracle takes as input a *convex polyhedron* P and outputs a *reduced polyhedron* P' such that $P' \Leftrightarrow P$ and such that the *number of constraints* in P' is lower or equal to that of P.

Definition 1 (Polyhedron). *A* (convex) polyhedron[6] *on* $\mathbb{Q}$ *is a conjunction of linear (in)equalities of the form* $\sum_i a_i x_i \bowtie b$ *where* a_i, b *are constants in* $\mathbb{Q}$*,*

[6] Dealing only with convex polyhedra on $\mathbb{Q}$, we often omit the adjective "convex".

where x_i are variables ranging over $\mathbb{Q}$, and where $\bowtie$ represents a binary relation on $\mathbb{Q}$ among $\geq$, $>$ or $=$.

A polyhedron may be suboptimally written. In particular, one of its constraints may be implied by the others: it is said *redundant* and can be discarded. Moreover, a set of inequalities can imply *implicit* equalities, such as $x = 0$ that can be deduced from $x \geq 0 \wedge -x \geq 0$. This notion of implicit equalities is standard and defined for instance in [19]. Definition 2 characterizes polyhedra without *implicit* equalities.

Definition 2 (Complete set of linear equalities). *Let E be a set of linear equalities and I be a set of linear inequalities. E is said* complete *w.r.t. I if any linear equality deduced from the conjunction $E \wedge I$ can also be deduced from E alone, meaning that I contains no equality, neither implicit nor explicit. Formally, E is complete iff for all linear terms t_1 t_2,*

$$(E \wedge I \Rightarrow t_1 = t_2) \textit{ implies } (E \Rightarrow t_1 = t_2) \tag{1}$$

Definition 3 (Reduced Polyhedron). *A polyhedron P is* reduced *iff it satisfies the following conditions.*

- *If P is unsatisfiable, then P is a* single constant constraint *like $0 > 0$ or $0 \geq 1$. In other words, its unsatisfiability is checked by one comparison on $\mathbb{Q}$.*
- *Otherwise, P contains no* redundant *constraint and is syntactically given as a conjunction $E \wedge I$ where polyhedron I contains only inequalities and where polyhedron E is a* complete *set of equalities w.r.t. I.*

Having a reduced polyhedron ensures that any provable linear equality admits a pure equational proof which ignores the remaining inequalities.

2.1 The Three Steps of the Tactic

Roughly speaking, a Coq goal corresponds to a sequent $\Gamma \vdash T$ where context Γ represents a conjunction of hypotheses and T a conclusion. In other words, this goal is logically interpreted as the meta-implication $\Gamma \Rightarrow T$. The tactic transforms the current goal $\Gamma \vdash T$ through three successive steps.

1. First, constraints are retrieved from the goal: it is equivalently rewritten into $\Gamma', \llbracket P \rrbracket (m) \vdash T'$ where P is a polyhedron and m an assignment of P variables. For example, the `ex1` goal is rewritten as $\llbracket P_1 \rrbracket (m_1) \vdash$ `False`, where
$$P_1 := \ x_1 \leq 1 \ \wedge \ x_2 < x_3 \ \wedge \ x_1 \geq 1$$
$$m_1 := \{ \ x_1 \mapsto \texttt{x}; \ x_2 \mapsto \texttt{(f x)}; \ x_3 \mapsto \texttt{(f 1)} \ \}$$
Here, constraint $x_1 \geq 1$ in P_1 comes from the negation of the initial `ex1` goal `x<1`. Hence, $\llbracket P \rrbracket (m)$ corresponds to a conjunction of inequalities on $\mathbb{Q}$ that *are not necessarily* linear, because m may assign variables of P to arbitrary Coq terms on $\mathbb{Q}$. Actually, $\llbracket P \rrbracket (m)$ contains at least all (in)equalities on $\mathbb{Q}$ that appear as hypotheses of Γ. Moreover, if T is an inequality on $\mathbb{Q}$, then an

inequality equivalent to $\neg T$ appears in $[\![P]\!](m)$ and T' is proposition `False`.[7] This step is traditionally called *reification* in Coq tactics.

2. Second, polyhedron P is reduced. In other words, the goal is equivalently rewritten into $\Gamma', [\![P']\!](m) \vdash T'$ where P' is the *reduced polyhedron* computed from P by our `reduce` oracle. For instance, polyhedron P_1 found above is reduced into
$$P'_1 := \; x_1 = 1 \wedge x_2 < x_3$$
3. At last, if P' is unsatisfiable, then so is $[\![P']\!](m)$, and the goal is finally discharged. Otherwise, given E the complete set of equalities in P', equalities of $[\![E]\!](m)$ are rewritten in the goal. For example, on the `ex1` goal, our tactic rewrites the learned equality "`x`=1" into the remaining hypothesis. In summary, a first call to the `vpl` tactic transforms the `ex1` goal into
$$\texttt{x=1},\ \ \texttt{(f 1)<(f 1)} \vdash \texttt{False}$$
A second call to `vpl` detects that hypothesis `(f 1)<(f 1)` is unsatisfiable and finally proves the goal.

In the description above, we claim that our transformations on the goals are equivalences. This provides a guarantee to the user: the tactic can always be applied on the goal, without loss of information. However, in order to make the Coq proof checker accept our transformations, we only need to prove implications, as detailed in the next paragraph.

2.2 The Proof Built by the Tactic

The tactic mainly proves the following two implications which are verified by the Coq kernel:

$$\Gamma', [\![P]\!](m) \vdash T' \;\Rightarrow\; \Gamma \vdash T \tag{2}$$
$$\forall m, [\![P]\!](m) \;\Rightarrow\; [\![P']\!](m) \tag{3}$$

Semantics of polyhedron $[\![.]\!]$ is encoded as a Coq function, using binary integers to encode variables of polyhedra. After simple propositional rewritings in the initial goal $\Gamma \vdash T$, an OCaml oracle provides m and P to the Coq kernel, which simply computes $[\![P]\!](m)$ and checks that it is syntactically equal to the expected part of the context. Hence, verifying implication (2) is mainly syntactical.

For implication (3), our `reduce` oracle actually produces a Coq AST, that represents a *proof witness* allowing to build each constraint of P' as a nonnegative linear combination of P constraints. Indeed, such a combination is necessarily a logical consequence of P. In practice, this proof witness is a value of a Coq inductive type. A Coq function called `reduceRun` takes as input a polyhedron P and its associated witness, and computes P'. A Coq theorem ensures that any result of `reduceRun` satisfies implication (3). Thus, this implication is ensured by construction, while – for the last step of the tactic described above – the Coq kernel computes P' by applying `reduceRun`.

[7] Here, $T \Leftrightarrow (\neg T \Rightarrow \texttt{False})$ because comparisons on $\mathbb{Q}$ are decidable.

3 Using the vpl Tactic

Combining solvers by exchanging equalities is one of the basis of modern SMT-solving, as pioneered by approaches of Nelson-Oppen [17,18] and Shostak [20]. This section illustrates how equality learning in an interactive prover mimics such equality exchange, in order to combine independent tactics. While much less automatic than standard SMT-solving, our approach provides opportunities for the user to compensate by "hand" for the weaknesses of a given tactic.

The main aspects of the `vpl` tactic are illustrated on the following single goal. This goal contains two uninterpreted functions `f` and `g` such that `f` domain and `g` codomain are the same uninterpreted type `A`. As we will see below, in order to prove this goal, we need to use its last hypothesis – of the form "`g(...) <> g(13)`" – by combining equational reasoning on `g` and on `Qc` field. Of course, we also need linear arithmetic on `Qc` order.

```
Lemma ex2 (A:Type) (f:A → Qc) (g:Qc → A) (v1 v2 v3 v4:Qc) :
  6*v1 - v2 - 10*v3 + 7*(f(g v1)+1) ≤ -1
  → 3*(f(g v1)-2*v3)+4 ≥ v2-4*v1
  → 8*v1 - 3*v2 - 4*v3 - f(g v1) ≤ 2
  → 11*v1 - 4*v2 > 3
  → v3 > -1
  → v4 ≥ 0
  → g((11-v2+13*v4)/(v3+v4)) <> g(13)
  → 3 + 4*v2 + 5*v3 + f(g v1) > 11*v1.
```

The `vpl` tactic reduces this goal to the equivalent one given below (where typing of variables is omitted).

```
H5: g((11-(11-13*v3)+13*v4)/(v3+v4))=(g 13) → False
vpl: v1 = 4-4*v3
vpl0: v2 = 11-13*v3
vpl1: f(g(4-4*v3)) = -3+3*v3
______________________________________(1/1)
0 ≤ v4 → (3#8) < v3 → False
```

Here, three equations `vpl`, `vpl0` and `vpl1` have been learned from the goal. Two irredundant inequalities remain in the hypotheses of the conclusion – where (3#8) is the Coq notation for $\frac{3}{8}$. The bound `v3` > -1 has disappeared because it is implied by (3#8) $<$ `v3`. By taking `v3` $= 1$, we can build a model satisfying all the hypotheses of the goal – including (3#8) $<$ `v3` – except `H5`. Thus, using `H5` is necessary to prove `False`.

Actually, we provide another tactic which automatically proves the remaining goal. This tactic (called `vpl_post`) combines equational reasoning on `Qc` field with a bit of congruence.[8] Let us detail how it works on this example. First, in backward reasoning, hypothesis `H5` is applied to eliminate `False` from the conclusion. We get the following conclusion (where previous hypotheses have been omitted).

[8] It is currently implemented on the top of `auto` with a dedicated basis of lemmas.

```
_______________________________________(1/1)
g((11-(11-13*v3)+13*v4)/(v3+v4))=(g 13)
```

Here, backward congruence reasoning reduces this conclusion to

```
_______________________________________(1/1)
(11-(11-13*v3)+13*v4)/(v3+v4)=13
```

Now, the `field` tactic reduces the conclusion to

```
_______________________________________(1/1)
v3+v4 <> 0
```

Indeed, the `field` tactic mainly applies ring rewritings on `Qc` while generating subgoals for checking that denominators are not zero. Here, because we have a linear denominator, we discharge the remaining goal using the `vpl` tactic again. Indeed, it gets the following polyhedron in hypotheses – which is unsatisfiable.

$$\texttt{v4} \geq 0 \quad \wedge \quad \texttt{v3} > \frac{3}{8} \quad \wedge \quad \texttt{v3} + \texttt{v4} = 0$$

Let us remark that lemma `ex2` is also valid when the codomain of `f` and types of variables `v1` ... `v4` are restricted to $\mathbb{Z}$ and operator "/" means the Euclidean division. However, both `omega` and `lia` fail on this goal without providing any help to the user. This is also the case of the `verit` tactic of SMTCoq because it deals with "/" as a non-interpreted symbol and can only deal with uninterpreted types `A` providing a decidable equality. By assuming a decidable equality on type `A` and by turning the hypothesis involving "/" into "`g((11-v2+13*v4)) <> g(13*(v3+v4))`", we get a slightly weaker version of `ex2` goal which is proved by `verit`. CoqHammer is currently not designed to solve such a complex arithmetic goal [11].

This illustrates that our approach is complementary to SMT-solving: it provides less automation than SMT-solving, but it may still help to progress in an interactive proof when SMT-solvers fail.

4 The Witness Format in the Tactic

Section 4.3 below presents our proof witness format in Coq to build a reduced polyhedron P' as a logical consequence of P. It also details the implementation of `reduceRun` and its correctness property, formalizing property (3) given in Section 2.2. In preliminaries, Section 4.1 recalls the Farkas operations of the VPL, at the basis of our proof witness format, itself illustrated in Section 4.2.

4.1 Certified Farkas Operations

The tactic uses the linear constraints defined in the VPL [13], that we recall here. Type `var` is the type of variables in polyhedra. Actually, it is simply defined as type `positive`, the positive integers of Coq. Module `Cstr` provides an efficient representation for linear constraints on `Qc`, the Coq type for $\mathbb{Q}$. Type `Cstr.t` handles constraints of the form "$t \bowtie 0$" where t is a linear term and $\bowtie \in \{=, \geq, >\}$. Hence, each input constraint "$t_1 \bowtie t_2$" will be encoded as "$t_1 - t_2 \bowtie 0$". Linear terms are themselves encoded as radix trees over `positive` with values in `Qc`.

The semantics of `Cstr.t` constraints is given by predicate `(Cstr.sat c m)`, expressing that model `m` of type `var`$\rightarrow$`Qc` satisfies constraint `c`. Module `Cstr` provides also the following operations

Add: $(t_1 \bowtie_1 0) + (t_2 \bowtie_2 0) \triangleq (t_1 + t_2) \bowtie 0$ where $\bowtie \triangleq \max(\bowtie_1, \bowtie_2)$ for the total increasing order induced by the sequence $=, \geq, >$;
Mul: $n \cdot (t \bowtie 0) \triangleq (n \cdot t) \bowtie 0$ assuming $n \in \mathbb{Q}$ and, if $\bowtie \in \{\geq, >\}$ then $n \geq 0$;
Merge: $(t \geq 0)$ & $(-t \geq 0) \triangleq t = 0$.

It is easy to prove that each of these operations returns a constraint that is satisfied by the models of its inputs. For example, given constraints `c1` and `c2` and a model `m` such that `(sat c1 m)` and `(sat c2 m)`, then `(sat (c1+c2) m)` holds. When invoked on a wrong precondition, these operations actually return "$0 = 0$" which is satisfied by any model. Still, this precondition violation only appears if there is a bug in the `reduce` oracle. These operations are called *Farkas operations*, in reference to Farkas' lemma recalled on page 11.

In the following, we actually handle each constraint with a proof that it satisfies a given set `s` of models (encoded here by its characteristic function). The type of such a constraint is `(wcstr s)`, as defined below.

```
Record wcstr (s: (var → Qc) → Prop) := {
  rep: Cstr.t;
  rep_sat: ∀ m, s m → Cstr.sat rep m
}.
```

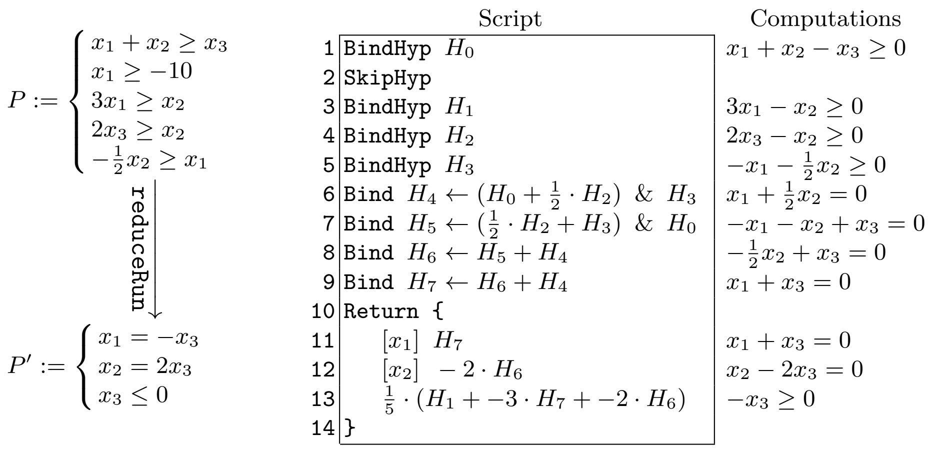
Hence, all the Farkas operations are actually lifted to type `(wcstr s)`, for all `s`.

4.2 Example of Proof Witness

We introduce our syntax for proof witnesses on Figure 1. Our oracle detects that P is satisfiable, and thus returns the "proof script" of Figure 1. This script instructs `reduceRun` to produce P' from P. By construction, we have $P \Rightarrow P'$.

This script has three parts. In the first part – from line 1 to 5 – the script considers each constraint of P and binds it to a name, or skips it. For instance, $x_1 \geq -10$ is skipped because it is redundant: it is implied by P' and thus not necessary to build P' from P. In the second part – from line 6 to 9 – the script builds intermediate constraints: their value is detailed on the right hand-side of the figure. Each of these constraints is bound to a name. Hence, when a constraint – like H_4 – is used several times, we avoid a duplication of its computation.

In the last part – from line 10 to 14 – the script returns the constraints of P'. As further detailed in Section 5, each equation defines one variable in terms of



$$P := \begin{cases} x_1 + x_2 \geq x_3 \\ x_1 \geq -10 \\ 3x_1 \geq x_2 \\ 2x_3 \geq x_2 \\ -\frac{1}{2}x_2 \geq x_1 \end{cases} \quad \xrightarrow{\texttt{reduceRun}} \quad P' := \begin{cases} x_1 = -x_3 \\ x_2 = 2x_3 \\ x_3 \leq 0 \end{cases}$$

	Script	Computations
1	`BindHyp` H_0	$x_1 + x_2 - x_3 \geq 0$
2	`SkipHyp`	
3	`BindHyp` H_1	$3x_1 - x_2 \geq 0$
4	`BindHyp` H_2	$2x_3 - x_2 \geq 0$
5	`BindHyp` H_3	$-x_1 - \frac{1}{2}x_2 \geq 0$
6	`Bind` $H_4 \leftarrow (H_0 + \frac{1}{2} \cdot H_2)\ \&\ H_3$	$x_1 + \frac{1}{2}x_2 = 0$
7	`Bind` $H_5 \leftarrow (\frac{1}{2} \cdot H_2 + H_3)\ \&\ H_0$	$-x_1 - x_2 + x_3 = 0$
8	`Bind` $H_6 \leftarrow H_5 + H_4$	$-\frac{1}{2}x_2 + x_3 = 0$
9	`Bind` $H_7 \leftarrow H_6 + H_4$	$x_1 + x_3 = 0$
10	`Return {`	
11	$[x_1]\ H_7$	$x_1 + x_3 = 0$
12	$[x_2]\ -2 \cdot H_6$	$x_2 - 2x_3 = 0$
13	$\frac{1}{5} \cdot (H_1 + -3 \cdot H_7 + -2 \cdot H_6)$	$-x_3 \geq 0$
14	`}`	

Fig. 1. Example of a Proof Script and its Interpretation by `reduceRun`

```
Definition pedra := list Cstr.t.
Definition ⟦l⟧ m := List.Forall (fun c ⇒Cstr.sat c m) l.
Definition answ (o: option pedra) m :=
  match o with
  | Some l ⇒ ⟦l⟧ m
  | None ⇒ False
  end.

Definition reduceRun(l:pedra)(p:∀ v,script v): option pedra
  := scriptEval (s:=⟦l⟧) (p _) l (* ... *).
Lemma reduceRun_correct l m p:⟦l⟧ m → answ (reduceRun l p) m.
```

Fig. 2. Definition of `reduceRun` and its Correctness

the others. For each equation, this variable is explicitly given between brackets "[.]" in the script of Figure 1, such as x_1 at line 11 and x_2 at line 12. This instructs `reduceRun` to rewrite equations in the form "$x = t$".

4.3 The HOAS of Proof Witnesses

Our `reduceRun` function and its correctness are defined in Coq, as shown on Figure 2. The input polyhedron is given as a list of constraints `l` of type `pedra`. The output is given as type (`option pedra`) where a `None` value corresponds to the case where `l` is unsatisfiable.

Given a value `l: pedra`, its semantics – noted ⟦`l`⟧ – is a predicate of type (`var`→`Qc`)→`Prop` which is defined from `Cstr.sat`. This semantics is extended to type (`option pedra`) by the predicate `answ`. Property (3) of page 4 is hence

```
Inductive fexp (v: Type): Type :=
 | Var: v → fexp v (* name bound to [Bind] or [BindHyp] *)
 | Add: fexp v → fexp v → fexp v
 | Mul: Qc → fexp v → fexp v
 | Merge: fexp v → fexp v → fexp v.

Fixpoint fexpEval {s} (c:fexp (wcstr s)): wcstr s :=
 match c with
 | Var c ⇒ c
 | Add c1 c2 ⇒ (fexpEval c1)+(fexpEval c2)
 | Mul n c ⇒ n·(fexpEval c)
 | Merge c1 c2 ⇒ (fexpEval c1)&(fexpEval c2)
 end.
```

Fig. 3. Farkas Expressions and their Interpreter

formalized by lemma `reduceRun_correct` with a minor improvement: when the input polyhedron is unsatisfiable, a proof of `False` is directly generated.

The proof witness in input of `reduceRun` is a value of type `∀ v, script v`. Here, `script` – defined at Figure 5 page 10 – is the type of a Higher-Order Abstract Syntax (HOAS) parameterized by the type `v` of variables [9]. A HOAS avoids the need to handle explicit variable substitutions when interpreting binders: those are encoded as functions, and variable substitution is delegated to the Coq engine.[9] The universal quantification over `v` avoids exposing the representation of `v` – used by `reduceRun` – in the proof witness `p`.

The bottom level of our HOAS syntax is given by type `fexp` defined at Figure 3 and representing "Farkas expressions". Each constructor in this type corresponds to a Farkas operation, except constructor `Var` that represents a constraint name which is bound to a `Bind` or a `BindHyp` binder (see Figure 1). The function `fexpEval` computes any such Farkas expression `c` into a constraint of type `(wcstr s)` – for some given `s` – where type `v` is itself identified with type `(wcstr s)`.

Farkas expressions are combined in order to compute polyhedra. This is expressed through "polyhedral expressions" of type `pexp` on Figure 4 which are computed by `pexpEval` into `(option pedra)` values. Type `pexp` has 3 constructors. First, constructor `(Bind c (fun H ⇒ p))` is a higher-order binder

[9] For a prototype like our tactic, such a HOAS has mainly the advantage of simplicity: it avoids formalizing in Coq the use of a substitution mechanism. The impact on the efficiency at runtime remains unclear. On one side, typechecking a higher-order term is more expensive than typechecking a first-order term. On the other side, implementing an efficient substitution mechanism in Coq is currently not straightforward: purely functional data-structures induce a non-negligible logarithmic factor over imperative ones. Imperative arrays with a purely functional API have precisely been introduced by [3] in an experimental version of Coq with this motivation. But this extension is not yet integrated into the stable release of Coq.

```
Inductive pexp (v: Type): Type :=
 | Bind: fexp v → (v → pexp v) → pexp v
 | Contrad: (fexp v) → pexp v
 | Return: list ((option var)*(fexp v)) → pexp v.

Fixpoint pexpEval {s} (p:pexp (wcstr s)): option pedra :=
 match p with
 | Bind c bp ⇒ pexpEval (bp (fexpEval c))
 | Contrad c ⇒ contrad c
 | Return l ⇒ Some (ret l nil)
 end.

Lemma pexpEval_correct s (p:pexp (wcstr s)) m:
  s m → answ (pexpEval p) m.
```

Fig. 4. Polyhedral Computations and their Interpreter

```
Inductive script (v: Type): Type :=
 | SkipHyp: script v → script v
 | BindHyp: (v → script v) → script v
 | Run: (pexp v) → script v.

Fixpoint scriptEval {s} (p: script(wcstr s)) (l: pedra):
 (∀ m, s m → ⟦l⟧ m) →option pedra := (*...*)

Lemma scriptEval_correct s (p:script(wcstr s)) l m:
  (∀ m, s m → ⟦l⟧ m) → s m →  answ (scriptEval p l) m.
```

Fig. 5. Script Expressions and their Interpreter

of our HOAS: it computes an intermediate Farkas expression `c` and stores the result in a variable `H` bound in the polyhedral expression `p`. Second, constructor `(Contrad c)` returns an *a priori* unsatisfiable constant constraint, which is verified by function `contrad` in `pexpEval`. At last, constructor `(Return l)` returns an *a priori* satisfiable reduced polyhedron, which is encoded as a list of Farkas expressions associated to an optional variable of type `var` (indicating a variable defined by an equation, see example of Figure 1).

Finally, a witness of type `script` first starts by naming useful constraints of the input (given as a value `l: pedra`) and then runs a polyhedral expression in this naming context. This semantics is given by function `scriptEval` specified at Figure 5. On a script `(SkipHyp p')`, interpreter `scriptEval` simply skips the first constraint by running recursively `(scriptEval p' (List.tl l))`. Similarly, on a script `(BindHyp (fun H ⇒ p'))`, it pops the first constraint of `l` in variable `H` and then runs itself on `p'`. Technically, function `scriptEval` assumes the following precondition on polyhedron `l`: it must satisfy all models `m` characterized by `s`. As shown on Figure 2, `(reduceRun l p)` is a simple instance of `(scriptEval (p (wcstr s)) l)` where `s`$:=$`⟦l⟧`. Hence, this precondition is trivially satisfied.

5 The Reduction Algorithm

The specification of the `reduce` oracle is given in introduction of the paper: it transforms a polyhedron P into a reduced polyhedron P' with a smaller number of constraints and such that $P' \Leftrightarrow P$. Sections 5.3 and 5.4 describe our implementation. In preliminaries, Section 5.1 gives a sufficient condition, through Lemma 2, for a polyhedron to be reduced. This condition lets us learn equalities from conflicts between strict inequalities as detailed in Section 5.2. In our proofs and algorithms, we only handle linear constraints in the restricted form "$t \bowtie 0$". But, for readability, our examples use the arbitrary form "$t_1 \bowtie t_2$".

5.1 A Refined Specification of the Reduction

Definition 4 (Echelon Polyhedron). *An echelon polyhedron is written as a conjunction $E \wedge I$ where polyhedron I contains only inequalities and where polyhedron E is written "$\bigwedge_{i \in \{1,\dots,k\}} x_i - t_i = 0$" such that each x_i is a variable and each t_i is a linear term, and such that the following two conditions are satisfied. First, no variable x_i appears in polyhedron I. Second, for all integers $i, j \in \{1,\dots,k\}$ with $i \leq j$ then x_i does not appear in t_j.*

Intuitively, in such a polyhedron, each equation "$x_i - t_i = 0$" actually *defines* variable x_i as t_i. As a consequence, $E \wedge I$ is satisfiable iff I is satisfiable.

We recall below the Farkas' lemma [12,10] which reduces the unsatisfiability of a polyhedron to the one of a constant constraint, like $0 > 0$. The unsatisfiability of such a constraint is checked by a simple comparison on $\mathbb{Q}$.

Lemma 1 (Farkas). *Let I be a polyhedron containing only inequalities. I is unsatisfiable if and only if there is an unsatisfiable constraint $-\lambda \bowtie 0$, computable from a nonnegative linear combination of constraints of I (i.e. using operators "$+$" and "$\cdot$" defined in Section 4.1), and such that $\bowtie \in \{\geq, >\}$ and $\lambda \in \mathbb{Q}^+$.*

From Farkas' lemma, we derive the following standard corollary which reduces the verification of an implication $I \Rightarrow t \geq 0$ to the verification of a syntactic equality between linear terms.

Corollary 1 (Implication Witness). *Let t be a linear term and let I be a* satisfiable *polyhedron written $\bigwedge_{j \in \{1,\dots,k\}} t_j \bowtie_j 0$ with $\bowtie_j \in \{\geq, >\}$.*

If $I \Rightarrow t \geq 0$ then there are $k+1$ nonnegative rationals $(\lambda_j)_{j \in \{0,\dots,k\}}$ such that $t = \lambda_0 + \Sigma_{j \in \{1,\dots,k\}} \lambda_j t_j$.

In the following, we say that the nonnegative coefficients $(\lambda_j)_{j \in \{0,\dots,k\}}$ define a "*Farkas combination of t in terms of I*".

Definition 5 (Strict Version of Inequalities). *Let I be a polyhedron with only inequalities. We note $I^>$ the polyhedron obtained from I by replacing each non-strict inequality "$t \geq 0$" by its strict version "$t > 0$". Strict inequalities of I remain unchanged in $I^>$.*

Geometrically, polyhedron $I^>$ is the interior of polyhedron I. Hence if $I^>$ is satisfiable (i.e. the interior of I is non empty), then polyhedron I does not fit inside a hyperplane. Lemma 2 formalizes this geometrical intuition as a consequence of Farkas' lemma. Its proof invokes the following corollary of Farkas' lemma, which is really at the basis of our equality learning algorithm.

Corollary 2 (Witness of Empty Interior). *Let us consider a* satisfiable *polyhedron I written $\bigwedge_{j\in\{1,\dots,k\}} t_j \bowtie_j 0$ with $\bowtie_j \in \{\geq, >\}$. Then, $I^>$ is unsatisfiable if and only if there exists k nonnegative rationals $(\lambda_j)_{j\in\{1,\dots,k\}} \in \mathbb{Q}^+$ such that $\Sigma_{j\in\{1,\dots,k\}}\lambda_j t_j = 0$ and $\exists j \in \{1,\dots,k\}, \lambda_j > 0$.*

Proof.
$\Leftarrow$: Suppose k nonnegative rationals $(\lambda_j)_{j\in\{1,\dots,k\}}$ such that $\Sigma_{j\in\{1,\dots,k\}}\lambda_j t_j = 0$ and some index j such that $\lambda_j > 0$. It means that there is a Farkas combination of $0 > 0$ in terms of $I^>$. Thus by Farkas' lemma, $I^>$ is unsatisfiable.
$\Rightarrow$: Let us assume that $I^>$ is unsatisfiable. By Farkas' lemma, there exists an unsatisfiable constant constraint $-\lambda \bowtie 0$, where $-\lambda = \Sigma_{j\in\{1,\dots,k\}}\lambda_j t_j$, with all $\lambda_j \in \mathbb{Q}^+$, and such that there exists some j with $\lambda_j > 0$. Let m be an assignment of I variables such that $[\![I]\!]\, m$. By definition, we have $[\![\Sigma_{j\in\{1,\dots,k\}}\lambda_j t_j]\!]\, m = \lambda'$ with $\lambda' \in \mathbb{Q}^+$. Thus, $-\lambda = \lambda' = 0$.

Lemma 2 (Completeness from Strict Satisfiability). *Let us assume an echelon polyhedron $E \wedge I$ without redundant constraints, and such that $I^>$ is satisfiable. Then, $E \wedge I$ is a reduced polyhedron.*

Proof. Let us prove property (1) of Definition 2, i.e. that E is complete w.r.t. I. Because $t_1 = t_2 \Leftrightarrow t_1 - t_2 = 0$, without loss of generality, we only prove property (1) in the case where $t_2 = 0$ and t_1 is an arbitrary linear term t. Let t be a linear term such that $E \wedge I \Rightarrow t = 0$.

In particular, $E \wedge I \Rightarrow t \geq 0$. By Corollary 1, there are $k' + 1$ nonnegative rationals $(\lambda_j)_{j\in\{0,\dots,k'\}}$ such that $t = \lambda_0 + \Sigma_{j\in\{1,\dots,k'\}}\lambda_j t_j$ where E is written $\bigwedge_{j\in\{1,\dots,k\}} t_j = 0$ and I is written $\bigwedge_{j\in\{k+1,\dots,k'\}} t_j \bowtie_j 0$. Suppose that there exists $j \in \{k+1,\dots,k'\}$, such that $\lambda_j \neq 0$. Since $I^>$ is satisfiable, by Corollary 2, we deduce that $\Sigma_{j\in\{k+1,\dots,k'\}}\lambda_j t_j \neq 0$. Thus, we have $E \wedge I^> \Rightarrow t > 0$ with $E \wedge I^>$ satisfiable. This contradicts the initial hypothesis $E \wedge I \Rightarrow t = 0$. Thus, $t = \lambda_0 + \Sigma_{j\in\{1,\dots,k\}}\lambda_j t_j$ which proves $E \Rightarrow t \geq 0$.

A similar reasoning from $E \wedge I \Rightarrow -t \geq 0$ finishes the proof that $E \Rightarrow t = 0$.

Lemma 2 gives a strategy to implement the `reduce` oracle. If the input polyhedron P is satisfiable, then try to rewrite P as an echelon polyhedron $E \wedge I$ where $I^>$ is satisfiable. The next step is to see that from an echelon polyhedron $E \wedge I$ where $I^>$ is unsatisfiable, we can *learn* new equalities from a subset of $I^>$

inequalities that is unsatisfiable. The inequalities in such a subset are said "*in conflict*". The Farkas witness proving the conflict is used to deduce new equalities from I. This principle can be viewed as an instance of "*conflict driven clause learning*" – at the heart of modern DPLL procedures [21].

5.2 Building Equality Witnesses from Conflicts

Consider a satisfiable set of inequalities I, from which we wish to extract implicit equalities. First, let us build $I^{>}$ the strict version of I as described in Definition 5. Then, an oracle runs the simplex algorithm to decide whether $I^{>}$ is satisfiable. If so, then we are done: there is no implicit equality to find in I. Otherwise, by Corollary 2, the oracle finds that the unsatisfiable constraint $0 > 0$ can be written $\Sigma_{j\in J}\lambda_j t_j > 0$ where for all $j \in J$, $\lambda_j > 0$ and $(t_j > 0) \in I^{>}$. Since $\bigwedge_{j\in J} t_j > 0$ is unsatisfiable, we can learn that $\bigwedge_{j\in J} t_j = 0$. Indeed, since $\Sigma_{j\in J}\lambda_j t_j = 0$ (by Corollary 2) and $\forall j \in J,\ \lambda_j > 0$, then each term t_j of this sum must be 0. Thus, $\forall j \in J,\ t_j = 0$.

Let us now detail our algorithm to compute equality witnesses. Let I be a satisfiable inequality set such that $I^{>}$ is unsatisfiable. The oracle returns a witness combining $n+1$ constraints of $I^{>}$ (for $n \geq 1$) that implies a contradiction:

$$\textstyle\sum_{i=1}^{n+1} \lambda_i \cdot I_i^{>} \quad \text{where} \quad \lambda_i > 0$$

By Corollary 2, this witness represents a contradictory constraint $0 > 0$. Moreover, each inequality I_i is non-strict (otherwise, I would be unsatisfiable). We can thus turn each inequality I_i into an equality written $I_i^{=}$ – proved by

$$I_i \ \& \ \tfrac{1}{\lambda_i} \cdot \textstyle\sum_{\substack{j\in\{1\dots n+1\}\\ j\neq i}} \lambda_j \cdot I_j$$

Hence, each equality $I_i^{=}$ is proved by combining $n+1$ constraints. Proving $(I_i^{=})_{i\in\{1,\dots,n+1\}}$ in this naive approach combines $\Theta(n^2)$ constraints.

We rather propose a more symmetric way to build equality witnesses which leads to a simple linear algorithm. Actually, we build a system of n equalities noted $(E_i)_{i\in\{1,\dots,n\}}$, where – for $i \in \{1,\dots,n\}$ – each E_i corresponds to the unsatisfiability witness where the i-th "+" has been replaced by a "&":

$$\left(\textstyle\sum_{j=1}^{i} \lambda_j \cdot I_j\right) \ \& \ \left(\textstyle\sum_{j=i+1}^{n+1} \lambda_j \cdot I_j\right)$$

This system of equations is proved equivalent to system $(I_i^{=})_{i\in\{1,\dots,n+1\}}$ thanks to the following correspondence.

$$\begin{cases} I_1^{=} = \frac{1}{\lambda_1} \cdot E_1 \\ I_{n+1}^{=} = -\frac{1}{\lambda_n} \cdot E_n \\ \text{for } i \in \{2,\dots,n\}\,, I_i^{=} = \frac{1}{\lambda_i} \cdot (E_i - E_{i-1}) \end{cases}$$

This also shows that one equality $I_i^{=}$ is redundant, because $(I_i^{=})_{i\in\{1,\dots,n+1\}}$ contains one more equality than $(E_i)_{i\in\{1,\dots,n\}}$.

In order to use a linear number of combinations, we build $(E_i)_{i\in\{1,\dots,n\}}$ thanks to two lists of intermediate constraints $(A_i)_{i\in\{1,\dots,n\}}$ and $(B_i)_{i\in\{2,\dots,n+1\}}$ defined by

$$\begin{cases} A_1 := \lambda_1 \cdot I_1 & \text{for } i \text{ from 2 up to } n,\ A_i := A_{i-1} + \lambda_i \cdot I_i \\ B_{n+1} := \lambda_{n+1} \cdot I_{n+1} & \text{for } i \text{ from } n \text{ down to 2},\ B_i := B_{i+1} + \lambda_i \cdot I_i \end{cases}$$

Then, we build $E_i := A_i \ \&\ B_{i+1}$ for $i \in \{1, \dots, n\}$.

5.3 Illustration on the Running Example

Let us detail how to compute the reduced form of polyhedron P from Figure 1.

$$P := \left\{ I_1 \colon x_1 + x_2 \geq x_3,\ I_2 \colon x_1 \geq -10,\ I_3 \colon 3x_1 \geq x_2,\ I_4 \colon 2x_3 \geq x_2,\ I_5 \colon -\frac{1}{2}x_2 \geq x_1 \right\}$$

P is a satisfiable set of inequalities. Thus, we first extract a complete set of equalities E from constraints of P by applying the previous ideas. We ask a Linear Programming (LP) solver for a point satisfying $P^{>}$, the strict version of P. Because there is no such point, the solver returns the unsatisfiability witness $I_1^{>} + \frac{1}{2} \cdot I_4^{>} + I_5^{>}$ (which reduces to $0 > 0$). By building the two sequences (A_i) and (B_i) defined previously, we obtain the two equalities

$$E_1 \colon x_1 + x_2 = x_3 \quad \textbf{proved by} \quad \underbrace{(x_1 + x_2 \geq x_3)}_{A_1\colon\ I_1} \& \underbrace{(x_3 \geq x_1 + x_2)}_{B_2\colon\ \frac{1}{2}\cdot I_4 + I_5}$$

$$E_2 \colon x_1 = -\tfrac{1}{2}x_2 \quad \textbf{proved by} \quad \underbrace{(x_1 \geq -\tfrac{1}{2}x_2)}_{A_2\colon\ I_1 + \frac{1}{2}\cdot I_4} \& \underbrace{(-\tfrac{1}{2}x_2 \geq x_1)}_{B_3\colon\ I_5}$$

Thus, P is rewritten into $E \wedge I$ with

$$E := \left\{ E_1 \colon x_1 + x_2 = x_3,\ E_2 \colon x_1 = -\frac{1}{2}x_2 \right\}, I := \left\{ I_2 \colon x_1 \geq 10,\ I_3 \colon 3x_1 \geq x_2 \right\}$$

To be reduced, the polyhedron must be in echelon form, as explained in Definition 4. This implies that each equality of E must have the form $x_i - t_i = 0$, and each such x_i must not appear in I. Here, let us consider that E_1 defines x_2: we rewrite E_1 into $x_2 - (x_3 - x_1) = 0$. Then, x_2 is eliminated from E_2, leading to $E_2' : x_1 + x_3 = 0$. In practice, we go one step further by rewriting x_1 (using its definition in E_2') into E_1 to get a *reduced* echelon system E' of equalities:

$$E' := \{E_1' \colon x_2 - 2 \cdot x_3 = 0,\ E_2' \colon x_1 + x_3 = 0\}$$

Moreover, the variables defined in E' (i.e. x_1 and x_2) are eliminated from I, which is rewritten into

$$I' := \{I_2' \colon -x_3 \geq -10,\ I_3' \colon -x_3 \geq 0\}$$

The last step is to detect that I_2' is redundant w.r.t. I_3' with a process which is indicated in the next section.

```
function reduce($E \wedge I$) =
   ($E$,$I$) $\leftarrow$ echelon($E$,$I$)
   match is_sat($I$) with
   | Unsat($\boldsymbol{\lambda}$) -> return Contrad($\boldsymbol{\lambda}^{\mathrm{T}} \cdot I$)
   | Sat(_) ->
       loop
          match is_sat($I^{>}$) with
          | Unsat($\boldsymbol{\lambda}$) ->
              ($E'$,$I'$) $\leftarrow$ learn($I$,$\boldsymbol{\lambda}$)
              ($E$,$I$) $\leftarrow$ echelon($E \wedge E'$,$I'$)
          | Sat($m$) ->
              $I$ $\leftarrow$ rm_redundancies($I$,$m$)
              return Reduced($E \wedge I$)
```

Fig. 6. Pseudo-code of the `reduce` oracle

5.4 Description of the Algorithm

The pseudo-code of Figure 6 describes the `reduce` algorithm. For simplicity, the construction of proof witnesses is omitted from the pseudo-code. To summarize, the result of `reduce` is either "`Contrad`(c)" where c is a contradictory constraint or "`Reduced`(P')" where P' is a satisfiable reduced polyhedron. The input polyhedron is assumed to be given in the form $E \wedge I$, where E contains only equalities and I contains only inequalities. First, polyhedron $E \wedge I$ is echeloned: function `echelon` returns a new system $E \wedge I$ where E is an echelon system of equalities without redundancies (they have been detected as $0 = 0$ during echeloning and removed) and without contradiction (they have been detected as $1 = 0$ during echeloning). Second, the satisfiability of I is tested by function `is_sat`. If `is_sat` returns "`Unsat` $(\boldsymbol{\lambda})$", then $\boldsymbol{\lambda}$ is a Farkas witness allowing to return a contradictory constant constraint written $\boldsymbol{\lambda}^{\mathrm{T}} \cdot I$. Otherwise, I is satisfiable and `reduce` enters into a loop to learn all implicit equalities.

At each step of the loop, the satisfiability of $I^{>}$ is tested. If `is_sat` returns "`Unsat` $(\boldsymbol{\lambda})$", then a new set E' of equalities is learned from $\boldsymbol{\lambda}$ and I' contains the inequalities of I that do not appear in the conflict. After echeloning the new system, the loop continues.

Otherwise, `is_sat` returns "`Sat`(m)" where m is a model of $I^{>}$. Geometrically, m is a point in the interior of polyhedron I. Point m helps `rm_redundancies` to detect and remove redundant constraints of I, by a ray-tracing method described in [16]. At last, `reduce` returns $E \wedge I$, which is a satisfiable reduced polyhedron because of Lemma 2.

Variant. In a variant of this algorithm, we avoid to test the satisfiability of I before entering the loop (i.e. the first step of the algorithm). Indeed, the satisfiability of I can be directly deduced from the witness returned by is_sat$(I^{>})$. If the combination of the linear terms induced by the witness gives a negative number instead of 0, it means that I is unsatisfiable. However, we could make several loop executions before finding that I is unsatisfiable: polyhedron I may contain several implicit equalities which do not imply the unsatisfiability of I and which may be discovered first. We do not know which version is the most efficient one. It probably differs according to applications.

6 Conclusion and Related Works

This paper describes a Coq tactic that learns equalities from a set of linear rational inequalities. It is less powerful than Coq SMT tactics [2,5,11] and than the famous `sledgehammer` of Isabelle [7,6]. But, it may help users to progress on goals that do not exactly fit into the scope of existing SMT-solving procedures.

This tactic uses a simple algorithm – implemented in the new VPL [15] – that follows a kind of conflict driven clause learning. This equality learning algorithm only relies on an efficient SAT-solver on inequalities able to generate nonnegativity witnesses. Hence, we may hope to generalize it to polyhedra on $\mathbb{Z}$.

The initial implementation of the VPL [14] also reduces polyhedra as defined in Definition 3. Its equality learning is more naive: for each inequality $t \geq 0$ of the current (satisfiable) inequalities I, the algorithm checks whether $I \wedge t > 0$ is satisfiable. If not, equality $t = 0$ is learned. In other words, each learned equality derives from one satisfiability test. Our new algorithm is more efficient, since it may learn several equalities from a single satisfiability test. Moreover, when there is no equality to learn, this algorithm performs only one satisfiability test.

We have implemented this algorithm in an Ocaml oracle, able to produce *proof witnesses* for these equalities. The format of these witnesses is very similar to the one of micromega [4], except that it provides a bind operator which avoids duplication of computations (induced by rewriting of learned equalities). In the core of our oracle, the production of these witnesses follows a lightweight, safe and evolutive design, called *polymorphic* LCF *style* [8]. This style makes the implementation of VPL oracles much simpler than in the previous VPL implementation. Our implementation thus illustrates how to instantiate "polymorphic witnesses" of polymorphic LCF style in order to generate Coq abstract syntax trees, and thus to prove the equalities in Coq by computational reflection.

The previous Coq frontend of the VPL [13] would also allow to perform such proofs by reflection. Here, we believe that the HOAS approach followed in Section 4.3 is much simpler and more efficient than this previous implementation (where substitutions were very inefficiently encoded with lists of constraints).

Our tactic is still a prototype. Additional works are required to make it robust in interactive proofs. For example, the user may need to stop the tactic before that the rewritings of the learned equalities are performed, for instance when some rewriting interferes with dependent types. Currently, the user can invoke instead a subtactic `vpl_reduce`, and apply these rewritings by "hand". The maintainability of such user scripts thus depends on the stability of the generated equalities and their order w.r.t. small changes in the input goal. However, we have not yet investigated these stability issues. A first step toward stability would be to make our tactic idempotent by keeping the goal unchanged on a already reduced polyhedron.

Another library, called Coq-Polyhedra [1], now formalizes a large part of the convex polyhedra theory without depending on external oracles. Our work is based on the VPL, because it wraps efficient external solvers [14]. In particular, computations in VPL oracles mix floating-points and GMP numbers, which are far more efficient than Coq numbers. However, the usability of the VPL would probably increase by being linked to such a general library.

Acknowledgements. We thank anonymous referees for their useful feedback on a preliminary version of this paper.

References

1. Allamigeon, X., Katz, R.D.: A formalization of convex polyhedra based on the simplex method. In: Interactive Theorem Proving (ITP). LNCS, vol. 10499, pp. 28–45. Springer (2017)
2. Armand, M., Faure, G., Grégoire, B., Keller, C., Théry, L., Werner, B.: A modular integration of SAT/SMT solvers to coq through proof witnesses. In: Certified Programs and Proofs (CPP). LNCS, vol. 7086, pp. 135–150. Springer (2011)
3. Armand, M., Grégoire, B., Spiwack, A., Théry, L.: Extending coq with imperative features and its application to SAT verification. In: Interactive Theorem Proving (ITP). LNCS, vol. 6172, pp. 83–98. Springer (2010)
4. Besson, F.: Fast reflexive arithmetic tactics the linear case and beyond. In: Types for Proofs and Programs (TYPES). LNCS, vol. 4502, pp. 48–62. Springer (2006)
5. Besson, F., Cornilleau, P., Pichardie, D.: Modular SMT proofs for fast reflexive checking inside coq. In: Certified Programs and Proofs (CPP). LNCS, vol. 7086, pp. 151–166. Springer (2011)
6. Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending sledgehammer with SMT solvers. Journal of Automated Reasoning **51**(1), 109–128 (2013). https://-doi.org/10.1007/s10817-013-9278-5
7. Böhme, S., Nipkow, T.: Sledgehammer: Judgement day. In: International Joint Conference on Automated Reasoning (IJCAR). LNCS, vol. 6173, pp. 107–121. Springer (2010)
8. Boulmé, S., Maréchal, A.: Toward Certification for Free! (July 2017), https://hal.archives-ouvertes.fr/hal-01558252, preprint
9. Chlipala, A.: Parametric higher-order abstract syntax for mechanized semantics. In: International Conference on Functional programming (ICFP). ACM Press (2008)
10. Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Schrijver, A.: Combinatorial Optimization. John Wiley & Sons, Inc., New York, NY, USA (1998)
11. Czajka, L., Kaliszyk, C.: Goal Translation for a Hammer for Coq. In: Proceedings First International Workshop on Hammers for Type Theories, HaTT@IJCAR 2016. EPTCS, vol. 210, pp. 13–20 (2016)
12. Farkas, J.: Theorie der einfachen Ungleichungen. Journal für die Reine und Angewandte Mathematik **124** (1902)
13. Fouilhé, A., Boulmé, S.: A certifying frontend for (sub)polyhedral abstract domains. In: Verified Software: Theories, Tools, Experiments (VSTTE). LNCS, vol. 8471, pp. 200–215. Springer (2014)

14. Fouilhé, A., Monniaux, D., Périn, M.: Efficient Generation of Correctness Certificates for the Abstract Domain of Polyhedra. In: Static Analysis Symposium (SAS). LNCS, vol. 7935, pp. 345–365. Springer (2013)
15. Maréchal, A.: New Algorithmics for Polyhedral Calculus via Parametric Linear Programming. Theses, UGA - Université Grenoble Alpes (December 2017), https://hal.archives-ouvertes.fr/tel-01695086
16. Maréchal, A., Périn, M.: Efficient elimination of redundancies in polyhedra by raytracing. In: Verification, Model Checking, and Abstract Interpretation (VMCAI). pp. 367–385. LNCS, Springer (2017)
17. Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Trans. Program. Lang. Syst. **1**(2), 245–257 (1979)
18. Oppen, D.C.: Complexity, convexity and combinations of theories. Theor. Comput. Sci. **12**, 291–302 (1980)
19. Schrijver, A.: Theory of Linear and Integer Programming. John Wiley & Sons, Chichester, New York, NY, USA (1986)
20. Shostak, R.E.: Deciding combinations of theories. J. ACM **31**(1), 1–12 (1984)
21. Silva, J.P.M., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 131–153. IOS Press (2009)
22. The Coq Development Team: The Coq proof assistant reference manual – version 8.7. INRIA (2017)