

A Coq Tactic for Equality Learning in Linear Arithmetic Sylvain Boulmé, Alexandre Maréchal

▶ To cite this version:

Sylvain Boulmé, Alexandre Maréchal. A Coq Tactic for Equality Learning in Linear Arithmetic. 2017. hal-01505598v1

HAL Id: hal-01505598 https://hal.science/hal-01505598v1

Preprint submitted on 11 Apr 2017 (v1), last revised 20 Jul 2017 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Coq Tactic for Equality Learning in Linear Arithmetic *

Sylvain Boulmé and Alexandre Maréchal

Univ. Grenoble-Alpes, VERIMAG, F-38000 Grenoble, France {sylvain.boulme,alex.marechal}@univ-grenoble-alpes.fr

Abstract. COQ provides linear arithmetic tactics like omega or lia. Currently, these tactics let the current goal unchanged when they can not prove it. We propose to improve this behavior: when the goal is not provable in linear arithmetic, we inject in hypotheses new equalities discovered from the linear inequalities. These equalities may help other COQ tactics to discharge the goal. The paper describes how we have implemented this idea in a new COQ tactic, dealing with linear arithmetic over rationals. It also illustrates that equality learning allows our tactic to interact with other COQ tactics.

Keywords: Linear Programming, Clause Learning, Skeptical Approach

1 Introduction

Several CoQ tactics prove goals containing linear (in)equalities: omega and lia on integers; fourier or lra on reals and rationals [7,1]. This paper provides yet another tactic for proving such goals. This tactic – called vpl¹ – is currently limited to rationals. It is built on the top of the Verified Polyhedra Library (VPL), a CoQ-certified abstract domain of convex polyhedra [4]. Its main feature appears when it can not prove the goal. In this case, whereas above tactics let the goal unchanged (they 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 can not 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 \rightarrow | Qc): | x \leq 1 \rightarrow | (f | x)<(f | 1) $ ightarrow$ | 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".

¹ Coq plugin available on 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.

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 new algorithm which learns these equalities from conflicts between strict inequalities detected by a linear programming solver. This algorithm can be viewed as a special but optimized case of "conflict driven clause learning" – at the heart of modern DPLL procedures [6]. On most cases, it is strictly more efficient than the naive equality learning algorithm previously implemented in the VPL [4]. 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 are computed in CoQ by reflection from these witnesses.

Specification of the Oracle. Let us now introduce the specification of the vpl tactic. As mentioned above, the algorithmic core of the tactic is performed by an oracle programmed in OCAML, and called reduce. This oracles 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 equals to that of P.

Definition 1 (Convex Polyhedron). A (convex) polyhedron² 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} , where x_i are variables ranging over \mathbb{Q} , and where \bowtie represents a binary relation on \mathbb{Q} among $\geq_i > or =$.

A polyhedron may be suboptimally written. In particular, one of its constraints may be implied by the others: it is thus *redundant* and can be discarded. Moreover, a set of inequalities can imply *implicit* equalities, such as x = 0 that can be deduced from $x \ge 0 \land -x \ge 0$. 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. Formally, E is complete iff – with t_1 and t_2 ranging over linear terms –

for all
$$t_1 t_2$$
, $(E \land I \Rightarrow t_1 = t_2)$ implies $(E \Rightarrow t_1 = t_2)$ (1)

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

² Dealing only with convex polyhedra on \mathbb{Q} , we often omit the adjective "convex".

- If P is unsatisfiable, then P is a single constant constraint like 0 > 0 or $0 \ge 1$. In other words, its unsatisfiability is checked by one comparison on \mathbb{Q} .
- Otherwise, P contains no redundant constraint and it 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. We shall show further (in Lemma 1) that this also happens for a larger class of equalities.

Specification 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, the goal is rewritten equivalently as $\Gamma', \llbracket P \rrbracket(m) \vdash T'$ where P is a polyhedron and m an assignment of P variables. For example, the **ex1** goal is rewritten as $\llbracket P_1 \rrbracket(m_1) \vdash \texttt{False}$, where

$$P_1 := x_1 \le 1 \land x_2 < x_3 \land x_1 \ge 1$$
$$m_1 := \{ x_1 \mapsto \mathbf{x}; x_2 \mapsto (\mathbf{f} \ \mathbf{x}); x_3 \mapsto (\mathbf{f} \ \mathbf{1}) \}$$

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 $\llbracket P \rrbracket(m)$ and T' is proposition False.³ This step is traditionally called *reification* in Coq tactics.

2. Second, the goal is rewritten equivalently as $\Gamma', \llbracket P' \rrbracket (m) \vdash T'$ where P' is the *reduced polyhedron* computed from P by our **reduce** oracle. For instance, polyhedron P_1 found in the **ex1** goal is reduced into

$$P'_1 := x_1 = 1 \land x_2 < x_3$$

3. At last, if P' is unsatisfiable, then so is $\llbracket P' \rrbracket (m)$, and the goal is finally discharged. Otherwise, given E the complete set of equalities in P', equalities of $\llbracket E \rrbracket (m)$ 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

$$x=1$$
, (f 1)<(f 1) \vdash 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.

³ Here, $T \Leftrightarrow (\neg T \Rightarrow \texttt{False})$ because comparisons on \mathbb{Q} are decidable.

4

The Coq Proof Built by the Tactic. The tactic mainly proves the two following implications which are verified by the Coq kernel:

$$\Gamma', \llbracket P \rrbracket (m) \vdash T' \implies \Gamma \vdash T \tag{2}$$

$$\forall m, \llbracket P \rrbracket(m) \Rightarrow \llbracket P' \rrbracket(m) \tag{3}$$

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 is syntactically equals 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 in 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.

Overview of the Paper. Section 2 illustrates our tactic on a non-trivial example. It also explains how it collaborates with other CoQ tactics through equality learning. Section 3 details the certificate format produced by our oracle, and how it is applied in our CoQ tactic. At last, Section 4 details the algorithm we developed to produce such certificates.

2 Applications of the vpl Tactic

This section illustrates the applications of the tactic, in particular when it can not prove the goal. Section 2.1 details the following completeness result – which is a consequence of the Lemma 1 given below.

Our tactic removes the need to consider inequalities on goals of the form " $P \Rightarrow t_1 = t_2$ " when P is a polyhedron and t_1, t_2 are first-order terms of an equational extension of linear rational arithmetic (see Definition 5).

Section 2.2 provides a detailed example which illustrates such equality proofs and how they may appear inside a larger proof.

2.1 Application to equality proofs

Definition 4. We note $\mathbb{Q}_{1,\geq,>}[\vec{X}]$ the first-order theory (with congruence) of linear arithmetic over \mathbb{Q} . This theory syntactically distinguishes \mathbb{Q} constants from other \mathbb{Q} terms: each \mathbb{Q} constant is considered as a distinct function symbol. On \mathbb{Q} terms, this theory axiomatizes addition, linear multiplication, and orders using axioms of totally ordered vector spaces – where vectors are \mathbb{Q} terms and scalars are \mathbb{Q} constants.

Definition 5. We call equational extension of linear rational arithmetic a firstorder theory that extends $\mathbb{Q}_{1,\geq,>}[\vec{X}]$ with axioms that are only equational Horn clauses. Such an axiom is of the following form (where t_i and t'_i are first-order terms, and \vec{x} ranges over all the variables of the axiom):

$$\forall \vec{x}, \ (t_1 = t'_1 \land \ldots \land t_n = t'_n) \ \Rightarrow \ t_{n+1} = t'_{n+1}$$

For example, equational extensions of linear rational arithmetic are closed under extensions with uninterpreted function symbols. Indeed, for these new symbols, we only add congruence axioms, which are equational Horn clauses.

As another example, consider an extension of $\mathbb{Q}_{1,\geq,>}[\vec{X}]$ embedding the ring theory of $\mathbb{Q}[\vec{X}]$, and that provides an axiom – for any rational constant c – linking linear multiplication (noted below \cdot) to unrestricted multiplication (noted below \times) given by

$$\forall x, \ c \times x = c \cdot x$$

This theory is an equational extension of linear rational arithmetic.

As a counter-example, field theory $\mathbb{Q}(\vec{X})$ can not be encoded with only equational Horn clauses. Intuitively, a field must provide an axiom for inverse like $x \neq 0 \Rightarrow x \times x^{-1} = 1$, which is not an equational Horn clause.

Lemma 1 (Extended Completeness). We assume E and I two polyhedra satisfying property (1) when t_1 and t_2 are linear terms (i.e. range over $\mathbb{Q}_1[\vec{X}]$). Property (1) is also satisfied when t_1 and t_2 range over first-order terms of any equational extension of linear rational arithmetic.

Proof. By induction on a cut-free proof of $t_1 = t_2$. There are two cases. Either we have applied an axiom corresponding to an equational Horn clause, and we conclude by induction hypotheses. Otherwise, we are on a pure equality of $\mathbb{Q}_1[\vec{X}]$ which can be deduced from E.

For example, property (1) ensures that we have a complete tactic for solving goals of the form " $P \Rightarrow t_1 = t_2$ " – where P is a polyhedron and where t_1 and t_2 are polynomials in $\mathbb{Q}[X]$ – by first applying our tactic and then applying the **ring** tactic. We can extend this completeness result when t_1 and t_2 contain uninterpreted symbols by also applying congruence closure.

When t_1 and t_2 are fractions of $\mathbb{Q}(\vec{X})$, we can not have such a completeness result. For example, $x = y \land y \ge 1 \Rightarrow x \times y^{-1} = 1$ is a tautology whereas $x = y \Rightarrow x \times y^{-1} = 1$ is false when x = y = 0. However, if denominators appearing in t_1 and t_2 are themselves linear, then we can still discharge goals of the form " $P \Rightarrow t_1 = t_2$ " by combining our tactic with the **field** tactic. This is illustrated in the following example.

2.2 A Detailed Example

This section illustrates the main aspects of the tactic on a single example. 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 " $g(..) \Leftrightarrow g(13)$ " – 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 \rightarrow Qc) (g:Qc \rightarrow A) (v1 v2 v3 v4: Qc):
6*v1 - v2 - 10*v3 + 7*(f(g v1) + 1) \leq -1
\rightarrow 3*(f(g v1) - 2*v3) + 4 \geq v2 - 4*v1
\rightarrow 8*v1 - 3*v2 - 4*v3 - f(g v1) \leq 2
\rightarrow 11*v1 - 4*v2 > 3
\rightarrow v3 > -1
\rightarrow v4 \geq 0
\rightarrow g((11 - v2 + 13*v4) / (v3+v4)) <> g(13)
\rightarrow 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 \rightarrow False
vpl : v1 = 4 - 4 * v3
vpl0 : v2 = 11 - 13 * v3
vpl1 : f (g (4 - 4 * v3)) = -3 + 3 * v3
------(1/1)
0 \leq v4 \rightarrow (3#8) \leq v3 \rightarrow False
```

Here, three equations vpl, vpl0 and vpl1 have been learned from the goal. Two non-redundant inequalities remain in the hypotheses of the conclusion – where (3#8) is the CoQ notation for $\frac{3}{8}$. The bound "v3 > -1" had 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 called vpl_post which automatically proves this goal. This tactic combines equational reasoning on Qc field with a bit of congruence.⁴ Let us detail how it works on this example. First, in backward reasoning, H5 is applied to eliminate False from the conclusion. We get the following conclusion (where previous hypotheses have been omitted).

 $\frac{(1/1)}{g((11 - (11 - 13*v3) + 13*v4) / (v3+v4))} = g 13$

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
```

 $^{^4}$ It is currently implemented on the top of ${\tt auto}$ with a dedicated basis of lemma.

Indeed, the field tactic mainly applies ring rewriting 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.

$$v4 \ge 0 \land v3 > \frac{3}{8} \land v3 + v4 = 0$$

Let us remark that since lemma ex2 is provable on \mathbb{Q} , it is also valid when the codomain of f and types of variables v1 ... v4 are restricted to \mathbb{Z} . However, both omega and lia fail on this goal without providing any help to the user.

3 The Witness Format and its Interpreter in the Tactic

Section 3.3 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 introduction. In preliminaries, Section 3.1 recalls the Farkas operations of the VPL, at the basis of our proof witness format, itself illustrated in Section 3.2.

3.1 Certified Farkas Operations on Linear Constraints

The tactic uses the linear constraints defined in the VPL [3], that we recall here. Type var is the type of variables in polyhedra. Actually, it is simply defined as type positive, the unbounded binary positive integers of CoQ. Module Cstr provides an efficient representation for linear constraints on Qc, the CoQ type for 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), (sat c m) expressing that model m: var \rightarrow Qc satisfies constraint c. Module Cstr provides also the following operations:

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

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 c1 and c2 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 also 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.

| | Script | Computations |
|--|--|-------------------------------|
| $x_1 + x_2 \ge x_3$ | 1 BindHyp H_0 | $x_1 + x_2 - x_3 \ge 0$ |
| $x_1 \ge -10$ | 2 SkipHyp | |
| $P := \left\{ 3x_1 \ge x_2 \right.$ | 3 BindHyp H_1 | $3x_1 - x_2 \ge 0$ |
| $2x_3 \ge x_2$ | 4 BindHyp H_2 | $2x_3 - x_2 \ge 0$ |
| $\int -\frac{1}{2}x_2 \ge x_1$ | 5 BindHyp H_3 | $-x_1 - \frac{1}{2}x_2 \ge 0$ |
| re | 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 $X \leftarrow (\frac{1}{2} \cdot H_2 + H_3) \& H_0$ | $-x_1 - x_2 + x_3 = 0$ |
| lCe | 8 Bind $H_5 \leftarrow 	ilde{X} + H_4$ | $-\frac{1}{2}x_2 + x_3 = 0$ |
| Ru | 9 Bind $H_6 \leftarrow H_5 + H_4$ | $x_1 + x_3 = 0$ |
| ₽↓ | 10 Return { | |
| $\int x_1 = -x_3$ | 11 $[x_1]$ H_6 | $x_1 + x_3 = 0$ |
| $P' := \begin{cases} x_2 = 2x_3 \end{cases}$ | 12 $[x_2] - 2 \cdot H_5$ | $x_2 - 2x_3 = 0$ |
| $x_3 \leq 0$ | 13 $\frac{1}{5} \cdot (H_1 + -3 \cdot H_6 + -2 \cdot H_5)$ | $-x_3 \ge 0$ |
| | 14 } | |

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

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

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

3.2 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 part. 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, $x_1 \ge -10$ 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 4, each equation defines one variable in terms of 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".

3.3 The HOAS of Proof Witnesses and its Interpreter

Our reduceRun function and its correctness are defined on Figure 2. In this Coq code, the input polyhedron of reduceRun is given as a list of constraints 1 of

```
Definition pedra := list Cstr.t.
Definition [[1]] m := List.Forall (fun c ⇒ Cstr.sat c m) l.
Definition answ (o: option pedra) m
        := match o with Some l ⇒ [[1]] m | None ⇒ False end.
Definition reduceRun (l:pedra)(p:∀ v, script v):option pedra
        := scriptEval (s:=[[1]]) (p _) l (* ... *).
Lemma reduceRun_correct l m p: [[1]] m →answ (reduceRun l p) m.
```

| Fig. | 2 . | Definition | of | reduceRun | and | its | Correctness |
|------|------------|------------|----|-----------|-----|-----|-------------|
|------|------------|------------|----|-----------|-----|-----|-------------|

```
Inductive fexp (v: Type): Type :=

| Var: v \rightarrow fexp v (* name bound to [Bind] or [BindHyp] *)

| Add: fexp v \rightarrow fexp v \rightarrow fexp v

| Mul: Qc \rightarrow fexp v \rightarrow fexp v

| Merge: fexp v \rightarrow fexp v

| Merge: fexp v \rightarrow fexp v.

Fixpoint fexpEval {s} (c: fexp (wcstr s)): (wcstr s) :=

match c with

| Var c \Rightarrow c

| Add c1 c2 \Rightarrow (fexpEval c1) + (fexpEval c2)

| Mul n c \Rightarrow n·(fexpEval c)

| Merge c1 c2 \Rightarrow (fexpEval c1) & (fexpEval c2)

end.
```

Fig. 3. Farkas Expressions and their Interpreter

type pedra. Its output is given as type (option pedra) where a None value corresponds to the case where 1 is unsatisfiable.

Given a value 1: pedra, its semantics – still noted $[\![1]\!]$ – is a predicate of type $(var \rightarrow Qc) \rightarrow Prop$ which is defined from Cstr.sat. This semantics is extended to type (option pedra) by the predicate answ. Lemma reduceRun_correct thus formalizes property (3) of page 4 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 $\forall v$, script v. Here, script – defined at Figure 5 – is the type of a Higher-Order Abstract Syntax (HOAS) parameterized by the type v of variables. 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. The universal quantification over v avoids exposing the representation of v – used by reducedRun – in the p proof witness.

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

```
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 1 ⇒ Some (ret 1 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 \rightarrow script v

| BindHyp: (v \rightarrow script v) \rightarrow script v

| Run: (pexp v) \rightarrow script v.

Fixpoint scriptEval {s} (p:script(wcstr s)) (l: pedra):

(\forall m, s m \rightarrow [[1] m) \rightarrow option pedra := (* ... *)

Lemma scriptEval_correct s (p:script(wcstr s)) m: \forall l:pedra,

(\forall m, s m \rightarrow [[1] m) \rightarrows m \rightarrowansw (scriptEval p l) m.
```

Fig. 5. Script Expressions and their Interpreter

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 $H \Rightarrow p$)) is a higher-order binder 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 1) 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 1: pedra) and then runs a polyhedral expression in this naming context. This semantics is given by scriptEval specified at Figure 5. On a script (SkipHyp p'), function scriptEval simply skips the first constraint by running recursively (scriptEval p' (List.tl 1)). Similarly, on a script (BindHyp (fun $H \Rightarrow p$ ')), scriptEval pops the first constraint of 1 in variable H and then run itself on p'. Technically, scriptEval assumes the following precondition on polyhedron 1: it must satisfies all models m characterized by s. As shown on Figure 2, (reduceRun 1 p) is a simple instance of (scriptEval (p (wcstr s)) 1) where s:=[1]. Hence, this precondition is trivially satisfied.

4 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 4.4 and 4.5 describe our implementation. In preliminaries, Section 4.1 gives a sufficient condition, through Lemma 4, for a polyhedron to be reduced. This condition leads to learn equalities from conflicts between strict inequalities as detailed in Sections 4.2 and 4.3. 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 " $t_1 \bowtie t_2$ ".

4.1 A Refined Specification of the Reduction

Definition 6 (Echelon Polyhedron). An echelon polyhedron is written as a conjunction $E \wedge I$ where polyhedron I contains only inequalities and where E is written " $\bigwedge_{i \in \{1,...,k\}} x_i - t_i = 0$ " such that each x_i is a variable and each t_i is a linear term, and such that the two following conditions are satisfied. First, no variable x_i appears in polyhedron I. Second, for all integers $i, j \in \{1,...,k\}$ with i < 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[2] which reduces the unsatisfiability of a polyhedron to the one of a constant constraint, like 0 > 0 or $-1 \ge 0$. The unsatisfiability of such a constraint is checked by a simple comparison on \mathbb{Q} .

Lemma 2 (Farkas). Let I be a polyhedron containing only inequalities. If I is unsatisfiable then there is an unsatisfiable constraint " $-\lambda \bowtie 0$ ", computable from a non-negative linear combination of constraints of I (i.e. using operators "+" and "·" defined at Section 3.1), and such that $\bowtie \in \{\geq, >\}$ and $\lambda \in \mathbb{Q}^+$.

Proof. This standard lemma is proved by induction on the number of variables in I. In the inductive case, one variable is eliminated using Fourier-Moztkin's elimination (i.e. by combining all pairs of inequalities in which this variable appears with an opposite sign).

From Farkas lemma, we derive the following auxiliary lemma which reduces the verification of an implication $I \Rightarrow t \ge 0$ to the verification of a syntactic equality between linear terms. **Lemma 3 (Implication Witness).** Let t be a linear term and let I be a satisfiable polyhedron written $\bigwedge_{j \in \{1,...,k\}} t_j \bowtie_j 0$ with $\bowtie_j \in \{\geq,>\}$.

If $I \Rightarrow t \ge 0$ then there are k+1 non-negative rationals $(\lambda_j)_{j \in \{0,...,k\}}$ such that $t = \lambda_0 + \sum_{j \in \{1,...,k\}} \lambda_j t_j$.

Proof. Let us assume that $I \wedge -t > 0$ is unsatisfiable.

By Farkas lemma, there is an unsatisfiable constant constraint $-\lambda_0 \bowtie 0$ such that $-\lambda_0 = (\Sigma_{j \in \{1,...,k\}} \lambda_j t_j) + \lambda_{k+1} \cdot (-t)$ with all λ_j being non-negative rationals. Actually, $\lambda_{k+1} > 0$. Otherwise, $-\lambda_0 \bowtie 0$ would be a proof that I is unsatisfiable. Thus, for all $j \in \{0, ..., k\}$, $\frac{\lambda_j}{\lambda_{k+1}} \ge 0$ and $t = \frac{\lambda_0}{\lambda_{k+1}} + \Sigma_{j \in \{1,...,k\}} \frac{\lambda_j}{\lambda_{k+1}} t_j$.

Definition 7 (Strict Version of Inequalities). Let I be a polyhedron with only inequalities. We note $I^{>}$ the polyhedron obtained from I by replacing each large inequality " $t \ge 0$ " by its strict version "t > 0". Strict inequalities of I remain unchanged in $I^{>}$.

Lemma 4 (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. Obviously, because $I^{>}$ is satisfiable, then I is satisfiable and $E \wedge I$ also. Now, let us prove property (1) of Definition 3. Let t_1 and t_2 be two linear terms such that $E \wedge I \Rightarrow t_1 = t_2$. We aim to prove $E \Rightarrow t_1 = t_2$.

First, we rewrite equations of E into terms of the form $t_1 - t_2$. This gives a term t which does not contain variables x_i defined in E. Because variables defined in E do not appear in I and t, we deduce $I \Rightarrow t = 0$ from $E \land I \Rightarrow t_1 = t_2$.

By Lemma 3, because $I \Rightarrow t \ge 0$, we have $t = \lambda_0 + \sum_{j \in \{1,...,k\}} \lambda_j t_j$ where $\lambda_j \ge 0$ and $I = \bigwedge_{j \in \{1,...,k\}} t_j \bowtie_j 0$. Similarly, since $I \Rightarrow -t \ge 0$, we have $-t = \lambda'_0 + \sum_{j \in \{1,...,k\}} \lambda'_j t_j$ where $\lambda'_j \ge 0$.

Let us define $t' \triangleq (\lambda_0 + \sum_{j \in J} \lambda_j t_j) + (\lambda'_0 + \sum_{j \in J'} \lambda'_j t_j)$. Let m be an assignment of I variables such that $\llbracket I^{>} \rrbracket m$. By definition, $\forall j \in \{1, \ldots, k\}, \llbracket t_j \rrbracket m > 0$. If there exists $j \in \{0, \ldots, k\}$ such that $\lambda_j \neq 0$, then we have $0 < \llbracket t' \rrbracket (m) = \llbracket t - t \rrbracket (m) = 0$. This is impossible. Therefore, t is syntactically the constant 0. This actually proves $E \Rightarrow t_1 = t_2$, because $E \Rightarrow t_1 - t_2 = t = 0$.

In this proof, the model m of $I^{>}$ is a *witness* allowing to prove that E is complete. However, as noted in introduction, for our tactic, we do not need to formally prove in CoQ that E is complete: we do not need to output m from our oracle.

Lemma 4 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 on an echelon polyhedron $E \wedge I$ where $I^{>}$ is unsatisfiable, we can learn new equalities from a minimal subset of $I^{>}$ inequalities that is unsatisfiable. The inequalities in such a minimal subset are said "in conflict".

4.2 Conflict Driven Equality Learning

Conflict Driven Clause Learning (CDCL) is a standard framework of modern DPLL SAT-solving [6]. Given a set of large inequalities I, we reformulate the satisfiability of I into this framework by considering each large constraint $t \ge 0$ of I as a clause $(t > 0) \lor (t = 0)$. Hence, our literals are either strict inequalities or equalities.

Let us run a CDCL SAT-solver on such a set of clauses I. First, let us imagine that the SAT-solver assumes all literals of $I^>$. Then, an oracle decides whether $I^>$ is satisfiable. If so, then we are done. Otherwise, by Farkas' lemma, the oracle returns an unsatisfiable constant constraint $-\lambda \bowtie 0$ such that $\lambda \ge 0$ and $-\lambda$ is written $\sum_{j \in J} \lambda_j t_j$ where for all $j \in J$, $\lambda_j > 0$ and $(t_j > 0) \in I^>$. The CDCL solver learns the new clause $\bigvee_{j \in J} t_j = 0$ equivalent to $\neg I^>$ under hypothesis I.

In fact, a simple arithmetic argument improves this naive CDCL algorithm by learning directly the conjunction of literals $\bigwedge_{j \in J} t_j = 0$ instead of the clause $\bigvee_{j \in J} t_j = 0$. Indeed, because $I \Rightarrow \bigwedge_{j \in J} t_j \ge 0$, we have – for all $j' \in J$ –

$$(I \wedge t_{j'} \neq 0) \Rightarrow 0 < \Sigma_{j \in J} \lambda_j t_j = -\lambda$$

Therefore, since $\lambda \ge 0$, we have $I \Rightarrow t_{j'} = 0$ for all $j' \in J$.

In the following, we learn equalities from conflicts between strict inequalities in an approach inspired from this naive CDCL algorithm. Whereas the number of oracle calls for learning n equalities in the naive CDCL algorithm is $\Omega(n)$, our additional arithmetic argument limits it to $\mathcal{O}(1)$ in the best cases.

4.3 Building Equality Witnesses from Conflicts

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 $n \ge 1$) that implies a contradiction:

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

We know that this witness represents a contradictory constraint 0 > 0 and that each inequality I_i is large: otherwise, we would have a proof that I is unsatisfiable. Each inequality I_i is turned into an equality written $I_i^=$ – proved by

$$I_i \& \frac{1}{\lambda_i} \cdot \sum_{\substack{j \in \{1...n+1\}\\ j \neq i}} \lambda_j \cdot I_j$$

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 $(E_i)_{i \in \{1,...,n\}}$, where – for $i \in \{1,...,n\}$ – each E_i corresponds to the unsatisfiability witness where the *i*-th "+" has been replaced by a "&":

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

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

because $(I_i^{=})_{i \in \{1,...,n+1\}}$ contains one more equality than $(E_i)_{i \in \{1,...,n\}}$.

 $I_1^{=} = \frac{1}{\lambda_1} \cdot E_1$ and $I_{n+1}^{=} = -\frac{1}{\lambda_n} \cdot E_n$ and for $i \in \{2, \ldots, n\}$, $I_i^{=} = \frac{1}{\lambda_i} \cdot (E_i - E_{i-1})$ In order to use a linear number of combinations, we build $(E_i)_{i \in \{1,\ldots,n\}}$ thanks to two lists of intermediate constraints $(A_i)_{i \in \{1,\ldots,n\}}$ and $(B_i)_{i \in \{2,\ldots,n+1\}}$ defined by

 $A_1 := \lambda_1 \cdot I_1$ and for i from 2 up to $n, A_i := A_{i-1} + \lambda_i \cdot I_i$

 $B_{n+1} := \lambda_{n+1} \cdot I_{n+1}$ and for *i* from *n* down to 2, $B_i := B_{i+1} + \lambda_i \cdot I_i$

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

4.4 Illustration on the Running Example

Let us detail how to compute the reduced form of polyhedron P from Figure 1. $P := \{I_1 : x_1 + x_2 \ge x_3, I_2 : x_1 \ge -10, I_3 : 3x_1 \ge x_2, I_4 : 2x_3 \ge x_2, I_5 : -\frac{1}{2}x_2 \ge x_1\}$ 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 to 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

$$E_{1}: x_{1} + x_{2} = x_{3} \text{ proved by} \underbrace{(x_{1} + x_{2} \ge x_{3})}_{A_{1}: I_{1}} \& \underbrace{(x_{3} \ge x_{1} + x_{2})}_{B_{2}: \frac{1}{2} \cdot I_{4} + I_{5}} \\ E_{2}: x_{1} = -\frac{1}{2}x_{2} \text{ proved by} \underbrace{(x_{1} \ge -\frac{1}{2}x_{2})}_{A_{2}: I_{1} + \frac{1}{2} \cdot I_{4}} \& \underbrace{(-\frac{1}{2}x_{2} \ge x_{1})}_{B_{3}: I_{5}} \\ \end{bmatrix}$$

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

$$E := \left\{ E_1 : x_1 + x_2 = x_3, \ E_2 : x_1 = -\frac{1}{2}x_2 \right\},\$$

$$I := \left\{ I_2 : x_1 \ge 10, \ I_3 : 3x_1 \ge x_2 \right\}$$

To be reduced, the polyhedron must be in echelon form, as explained in Definition 6. 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 . To be in the form t = 0, E_1 is rewritten 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, our oracle goes one step further by rewriting x_1 (using its definition in E'_2) into E_1 in order to get a reduced echelon system E' of equalities:

$$E' := \{E'_1 : x_2 - 2 \cdot x_3 = 0, \ E'_2 : x_1 + x_3 = 0\}$$

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

$$I' := \{I'_2 : -x_3 \ge -10, \ I'_3 : -x_3 \ge 0\}$$

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.

For simplicity, construction of proof witness is omitted on the pseudo-code. In other words, the result of **reduce** is

• either "Contrad(c)" where c is a contradictory constraint

• or "Reduced(P')" where P' is a satisfiable reduced polyhedron.

```
function reduce (E \land I) =

(E, I) \leftarrow echelon (E, I)

match is_sat(I) with

| Unsat(\lambda) \rightarrow return Contrad(\lambda^{T} \cdot I)

| Sat(_) \rightarrow

loop

match is_sat(I^{>}) with

| Unsat(\lambda) \rightarrow

(E', I') \leftarrow learn(I, \lambda)

(E, I) \leftarrow echelon(E \land E', I')

| Sat(m) \rightarrow

I \leftarrow rm_redundancies(I, m)

return Reduced(E \land I)
```

Fig. 6. Pseudo-code of the reduce oracle

4.5 Description of the Algorithm

The pseudo-code of Figure 6 describes the reduce algorithm. 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 and inserted as a contradictory constraint $-1 \geq 0$ in I). Second, the satisfiability of I is tested by function is_sat. If is_sat returns "Unsat (λ)", then λ is a Farkas witness allowing to return a contradictory constant constraint written $\lambda^{\mathrm{T}} \cdot I$. 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 (λ)", then a new set E' of equalities is learned from λ 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 function rm_redundancies to detect and remove redundant constraints of I, by a ray-tracing method described in [5]. At last, reduce returns $E \wedge I$, which is a satisfiable reduced polyhedron because of Lemma 4.

5 Conclusion and Related Works

This paper describes a CoQ tactic that learns equalities from a set of linear inequalities. It uses a simple algorithm – implemented in the new VPL – 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 non-negativity witnesses. Hence, it seems generalizable to arbitrary polynomials. We may also hope to generalize it to totally ordered rings like \mathbb{Z} .

The initial implementation of the VPL [4] also reduces polyhedra as defined in Definition 3. It implements equality learning in a more naive way: for each inequality $t \ge 0$ of the current (satisfiable) inequalities I, the algorithm checks whether $I \land t > 0$ 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, the new algorithm performs only one satisfiability test, whereas the previous version checks all inequalities.

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 [1], except that it provides a bind operator which avoids duplications 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*⁵. 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 [3] would also allow to perform such proofs by reflection. Here, we believe than the HOAS approach followed in Section 3.3 is simpler and more efficient than this previous implementation.

References

- 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)
- Farkas, J.: Theorie der einfachen Ungleichungen. Journal f
 ür die Reine und Angewandte Mathematik 124 (1902)
- 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)
- 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)
- 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)
- 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)
- 7. The Coq Development Team: The Coq proof assistant reference manual version 8.6. INRIA (2016)

⁵ See our paper *Certification for Free!* submitted to ICFP'2017.