Integrating Simplex with Tableaux

Guillaume Bury, David Delahaye

Integrating Simplex with Tableaux*

Guillaume Bury and David Delahaye

Cedric/Cnam/Inria, Paris, France
Guillaume.Bury@inria.fr
David.Delahaye@cnam.fr

Abstract. We propose an extension of a tableau-based calculus to deal with linear arithmetic. This extension consists of a smooth integration of arithmetic deductive rules to the basic tableau rules, so that there is a natural interleaving between arithmetic and regular analytic rules. The arithmetic rules rely on the general simplex algorithm to compute solutions for systems over rationals, as well as on the branch and bound method to deal with integer systems. We also describe our implementation in the framework of Zenon, an automated theorem prover that is able to deal with first order logic with equality. This implementation has been provided with a backend verifier that relies on the Coq proof assistant, and which can verify the validity of the generated arithmetic proofs. Finally, we present some experimental results over the arithmetic category of the TPTP library, and problems of program verification coming from the benchmark provided by the BWare project.

Keywords: Tableaux, Linear Arithmetic, General Simplex Algorithm, Branch and Bound Method, Proof Checking, Zenon, Coq.

1 Introduction

Program analysis and verification often involve to verify arithmetic constraints. For example, this is the case when verifying loop invariants over imperative programs. In general, these arithmetic constraints are not very complex, but they lie in more complex first order formulas where pure first order logic and arithmetic are actually mixed. To deal with this kind of formulas, we propose, in this paper, to integrate arithmetic reasoning with a tableau-based proof search method. Regarding the nature of arithmetic reasoning, we consider the linear fragment, also known as Presburger arithmetic (where we only consider the addition operation and equality), which is weaker than Peano arithmetic but has the advantage to be decidable. There exist several decision procedures for linear arithmetic, and we chose to rely on the general simplex algorithm [6] (i.e. the usual simplex algorithm but without considering the optimization problem) to compute solutions for problems over rationals, as well as on the branch and bound method to deal with integer problems.

* This work is supported by the BWare project [9, 17] (ANR-12-INSE-0010) funded by the INS programme of the French National Research Agency (ANR).

This work is motivated by needs that arose in the framework of the BWare project [9, 17]. This project aims to provide a mechanized framework to support the automated verification of proof obligations coming from the development of industrial applications using the B method [1] and requiring high integrity. The methodology used in this project consists in building a generic platform of verification relying on different automated theorem provers, such as first order provers and SMT (Satisfiability Modulo Theories) solvers. Among the considered provers, there is Zenon [5], which is an automated theorem prover for classical first order logic with equality, and which is based on the tableau method. As Zenon is not able to deal with arithmetic, which may be required when verifying some proof obligations (involving mainly integer arithmetic in the benchmark of BWare), this is why we proposed to develop an extension of this tool to arithmetic, which will be described in this paper. This extension modifies not only the proof search rules of the prover, but also the backend as Zenon is able to produce proofs checkable by external tools, such as Coq [18] for instance. This backend is part of the objectives of the BWare project, which requires the verification tools to produce proof objects that are to be checked independently.

To extend Zenon to arithmetic, the idea is to add new specific rules, which are completely orthogonal to the other usual analytic rules of tableaux, and which use the computations performed by the simplex procedure as oracles. These rules are intended to deal with arithmetic formulas that are universally quantified. As for arithmetic formulas that are existentially quantified, no new rule is needed, but the instantiation mechanism has to be modified in order to call the simplex procedure to find instantiations. However, these instantiations must help us close all the branches of the proof search tree and not only a part of them (this is not unsound but does not help us find a proof). To do so, we introduce a notion of arithmetic constraint tree, which is a tree labeled with sets of arithmetic formulas, and which is built from the proof search tree. From this arithmetic constraint tree, a set of formulas is selected in order to cover the tree, i.e. it is sufficient to find a solution for this set of formulas (its negation to be more precise) to get a solution that closes the arithmetic constraint tree and therefore the proof search tree. It should be noted that our extension is able to deal with pure universal or existential arithmetic formulas, i.e. we do not consider alternation of universal and existential quantifiers and the variables occurring in an arithmetic formula must be of the same nature (either Skolem symbols or free variables). It should also be noted that as Zenon deals with equality, the arithmetic reasoning can be naturally combined with the equational reasoning involving both uninterpreted functions and predicates.

This paper is organized as follows: in Secs. 2 and 3, we first introduce respectively the proof search method of Zenon and the general simplex algorithm, as well as the branch and bound method; we then present, in Secs. 4 and 5, the arithmetic rules for Zenon and describe how the instantiation has to be modified to handle arithmetic; finally, in Sec. 6, we provide an overview of our implementation and the experimental results obtained on the benchmarks provided by the TPTP library and the BWare project, and propose some related work in Sec. 7.

2 The Zenon Automated Theorem Prover

The Zenon automated theorem prover relies on a tableau-based proof search method for classical first order logic with equality. The proof search rules of Zenon are described in detail in [5] and summarized in Fig. 1 (for the sake of simplification, we have omitted the unfolding and extension rules), where ϵ is Hilbert's operator ($\epsilon(x).P(x)$ means some x that satisfies $P(x)$, and is considered as a term), capital letters are used for metavariables, and R_r, R_s, R_t, and R_{ts} are respectively reflexive, symmetric, transitive, and transitive-symmetric relations (the corresponding rules also apply to the equality in particular). As hinted by the use of Hilbert's operator, the δ-rules are handled by means of ϵ-terms rather than using Skolemization. What we call here metavariables are often named free variables in the tableau-related literature; they are not used as variables as they are never substituted. The proof search rules are applied with the normal tableau method: starting from the negation of the goal, apply the rules in a top-down fashion to build a tree. When all branches are closed (i.e. end with an application of a closure rule), the tree is closed, and this closed tree is a proof of the goal. Note that this algorithm is applied in strict depth-first order: we close the current branch before starting work on another branch. Moreover, we work in a non-destructive way: working on one branch will never change the formulas of any other branch. We divide these rules into five distinct classes to be used for a more efficient proof search. This extends the usual sets of rules dealing with $\alpha, \beta, \delta, \gamma$-formulas and closure ($\odot$) with the specific rules of Zenon. We list below the five sets of rules and their elements:

α	$\alpha_{\neg\vee}$, $\alpha_{\wedge}$, $\alpha_{\neg\Rightarrow}$, $\alpha_{\neg\neg}$, $\neg_{\text{refl}}$
β	$\beta_{\vee}$, $\beta_{\neg\wedge}$, $\beta_{\Rightarrow}$, $\beta_{\Leftrightarrow}$, $\beta_{\neg\Leftrightarrow}$, pred, fun, sym, trans$*$
δ	$\delta_{\exists}$, $\delta_{\neg\forall}$
γ	$\gamma_{\forall M}$, $\gamma_{\neg\exists M}$, $\gamma_{\forall\text{inst}}$, $\gamma_{\neg\exists\text{inst}}$
$\odot$	$\odot_{\top}$, $\odot_{\bot}$, $\odot$, $\odot_r$, $\odot_s$

where "trans$*$" gathers all the transitivity rules.

To deal with arithmetic formulas, we use the ability of Zenon to perform typed proof search, which relies on a polymorphic type system. To simplify, we do not consider types in our presentation of Zenon and its extension to arithmetic, as they tend to make the presentation uselessly heavy since, in our case, types are actually just used to distinguish arithmetic formulas from the other ones.

3 The Simplex and the Branch and Bound Methods

We define linear arithmetic expressions as expressions built using addition and multiplication by numeric constants, while subtraction is seen as syntactic sugar for addition with multiplication by a negative constant. An arithmetic formula is a comparison of two linear arithmetic expressions, for example $2x + 1 < 7 - \frac{1}{2}y$. We consider 5 comparison operators, i.e. $=$, $<$, $>$, $\leq$, and $\geq$[1]. An arbitrary

[1] We use the notation $e \neq e'$ as syntactic sugar for $\neg(e = e')$.

Closure and Cut Rules

$$\frac{\bot}{\odot}\ \odot_{\bot} \qquad \frac{\neg\top}{\odot}\ \odot_{\neg\top} \qquad \frac{}{P \mid \neg P}\ \text{cut}$$

$$\frac{\neg R_r(t,t)}{\odot}\ \odot_r \qquad \frac{P \qquad \neg P}{\odot}\ \odot \qquad \frac{R_s(a,b) \qquad \neg R_s(b,a)}{\odot}\ \odot_s$$

Analytic Rules

$$\frac{\neg\neg P}{P}\ \alpha_{\neg\neg} \qquad \frac{P \Leftrightarrow Q}{\neg P, \neg Q \mid P, Q}\ \beta_{\Leftrightarrow} \qquad \frac{\neg(P \Leftrightarrow Q)}{\neg P, Q \mid P, \neg Q}\ \beta_{\neg\Leftrightarrow}$$

$$\frac{P \wedge Q}{P, Q}\ \alpha_{\wedge} \qquad \frac{\neg(P \vee Q)}{\neg P, \neg Q}\ \alpha_{\neg\vee} \qquad \frac{\neg(P \Rightarrow Q)}{P, \neg Q}\ \alpha_{\neg\Rightarrow}$$

$$\frac{P \vee Q}{P \mid Q}\ \beta_{\vee} \qquad \frac{\neg(P \wedge Q)}{\neg P \mid \neg Q}\ \beta_{\neg\wedge} \qquad \frac{P \Rightarrow Q}{\neg P \mid Q}\ \beta_{\Rightarrow}$$

$$\frac{\exists x.P(x)}{P(\epsilon(x).P(x))}\ \delta_{\exists} \qquad \frac{\neg\forall x.P(x)}{\neg P(\epsilon(x).\neg P(x))}\ \delta_{\neg\forall}$$

γ-Rules

$$\frac{\forall x.P(x)}{P(X)}\ \gamma_{\forall M} \qquad \frac{\neg\exists x.P(x)}{\neg P(X)}\ \gamma_{\neg\exists M}$$

$$\frac{\forall x.P(x)}{P(t)}\ \gamma_{\forall\text{inst}} \qquad \frac{\neg\exists x.P(x)}{\neg P(t)}\ \gamma_{\neg\exists\text{inst}}$$

Relational Rules

$$\frac{P(t_1,\ldots,t_n) \qquad \neg P(s_1,\ldots,s_n)}{t_1 \neq s_1 \mid \ldots \mid t_n \neq s_n}\ \text{pred} \qquad \frac{f(t_1,\ldots,t_n) \neq f(s_1,\ldots,s_n)}{t_1 \neq s_1 \mid \ldots \mid t_n \neq s_n}\ \text{fun}$$

$$\frac{R_s(s,t) \qquad \neg R_s(u,v)}{t \neq u \mid s \neq v}\ \text{sym} \qquad \frac{\neg R_r(s,t)}{s \neq t}\ \neg_{\text{refl}}$$

$$\frac{R_t(s,t) \qquad \neg R_t(u,v)}{u \neq s, \neg R_t(u,s) \mid t \neq v, \neg R_t(t,v)}\ \text{trans}$$

$$\frac{R_{ts}(s,t) \qquad \neg R_{ts}(u,v)}{v \neq s, \neg R_{ts}(v,s) \mid t \neq u, \neg R_{ts}(t,u)}\ \text{transsym}$$

$$\frac{s = t \qquad \neg R_t(u,v)}{u \neq s, \neg R_t(u,s) \mid \neg R_t(u,s), \neg R_t(t,v) \mid t \neq v, \neg R_t(t,v)}\ \text{transeq}$$

$$\frac{s = t \qquad \neg R_{ts}(u,v)}{v \neq s, \neg R_{ts}(v,s) \mid \neg R_{ts}(v,s), \neg R_{ts}(t,u) \mid t \neq u, \neg R_{ts}(t,u)}\ \text{transeqsym}$$

Fig. 1. Proof Search Rules of Zenon

comparison operator distinct from the equality may be noted $\bowtie$, and its negation $\overline{\bowtie}$, with the following correspondence: $\overline{<} \equiv \geq$, $\overline{\leq} \equiv >$, $\overline{>} \equiv \leq$, and $\overline{\geq} \equiv <$.

In the following, arithmetic expressions can involve integers, rationals, or reals[2]. However, we do not consider mixed problems, and assume that a given problem only involves one numeric type (either integers, rationals, or reals).

As a solving method, we consider the general simplex, as described in [13], which is a variant of the simplex algorithm, designed to solve the satisfiability problem on linear systems, rather than the optimization of a given objective function under a system of constraints.

The general simplex accepts only two forms of constraints: equations of the form $v = \sum_i a_i x_i$, with $a_i \in \mathbb{Q}$, and bounds on variables $l_i \leq v \leq u_i$, with $l_i, u_i \in \mathbb{Q} \cup \{-\infty, +\infty\}$. A system that contains only formulas of either form is said to be in general form. This representation does not restrict expressivity, given that any linear system can be translated into this representation. To do so, two transformations are required:

1. Any equality $e = e'$, where neither e nor e' is a variable, is replaced by $e \leq e' \wedge e' \leq e$;
2. Any comparison $e \bowtie e'$ is rewritten as $f \bowtie k$, where f is a non-empty sum of variables with coefficients[3], and k a numeric constant, s.t. $e - e' = f - k$. The comparison can then be replaced by $x = f \wedge x \bowtie k$, with x a fresh variable.

This transformation into general forms allows us to get an interesting property of the system: all variables on the left-hand side of equalities do not appear on the right-hand side of equalities. In the following, we suppose that all general forms satisfy this property, i.e we only consider general forms that come from the application of the process described above to a linear system.

The simplex method performs a series of pivot operations over the system[4], and stores the result in an internal state. This internal state allows the algorithm to be incremental, i.e we can easily add new equalities and bounds to this state, and get a new state that we can try to solve. When the given system is satisfiable, the simplex algorithm returns its new state together with a solution, and it is straightforward to check that it is a correct solution of the linear system. When the algorithm meets an unsatisfiable system S, it returns an equality $x = \sum_i a_i y_i$, which is implied by the equalities in S, s.t. the following properties hold[5]:

- There exists l (resp. u) s.t. $x \geq l \in S$ (resp. $x \leq u \in S$);
- There exist numeric constants l_i, u_i s.t. for all i, if $a_i > 0$, then $y_i \leq u_i \in S$ (resp. $y_i \geq l_i \in S$), and if $a_i < 0$, then $y \geq l_i \in S$ (resp. $y_i \leq u_i \in S$);

[2] For reals, numeric constants are represented as arbitrary precision rationals.

[3] If f is the empty sum, then the comparison is either trivially false, in which case the system if unsatisfiable, or a tautology, in which case it is useless.

[4] The termination of the simplex method is ensured using Bland's rule (see [11]).

[5] The new state of the simplex is of no use in this case, as if a system is unsatisfiable, then adding new equalities or bounds will not change anything to its satisfiability.

– $\sum_{a_i>0} a_i u_i + \sum_{a_i<0} a_i l_i < l$ (resp. $u < \sum_{a_i>0} a_i l_i + \sum_{a_i<0} a_i u_i$), resulting in a contradiction, since $l \leq x = \sum_i a_i y_i \leq \sum_{a_i>0} a_i u_i + \sum_{a_i<0} a_i l_i$ (resp. $\sum_{a_i>0} a_i l_i + \sum_{a_i<0} a_i u_i \leq \sum_i a_i y_i = x \leq u$) should hold according to S.

In order to deal with integer systems, we adopt a branch and bound strategy. An integer linear system can be seen as a rational system where all variables are required to have an integer value. For this reason, we can accept rational coefficients in the system: given a constraint with rational coefficients, we multiply it by the least common multiple of the denominators of the coefficients in order to get an equivalent constraint with only integer coefficients. Given an integer system S, we call relaxed system of S, noted relaxed(S), the system S without the condition that the variables must have an integer assignment.

Given a system S, the branch and bound algorithm works as follows:

– If relaxed(S) is unsatisfiable (as a rational system), then return false;
– If the system has a rational solution then:
  • If all the variables have an integer assignment, then return true;
  • If a non-integer value v is assigned to an integer variable x, then call the branch and bound twice with the two systems $S \cup \{x \leq \lfloor v \rfloor\}$ and $S \cup \{x \geq \lfloor v \rfloor + 1\}$, and return the disjunction of the two returned values.

Unfortunately, this algorithm is not complete: if we consider the system $1 \leq 3x + 3y \leq 2$, the branch and bound algorithm will loop. More generally, the branch and bound will not terminate on unsatisfiable integer systems with unbounded rational solutions (but no integer solution). However, if the system is satisfiable, then a solution will be found by the algorithm, provided that we use a breadth-first search.

To ensure termination, we can use global bounds, as found in [14]. Given an $m \times n$ rational matrix $A = (a_i)$, a vector $b \in \mathbb{Q}^m$, and the set of rational solutions $P = \{x \in \mathbb{Q} \mid Ax \leq b\}$, if the set of integer solutions $S = P \cap \mathbb{Z}^n$ is non-empty, then there exists an integer solution $x \in S$ s.t. $|x_j| \leq \omega_{A,b}$ for all $1 \leq j \leq n$, with $\omega_{A,b} = (2n'^2\theta)^{n'}$, where $n' = \max(n, m)$ and $\theta = \max_{ij}(|a_{ij}|)$.

4 Arithmetic Proof Search Rules

In this section, we present the arithmetic proof search rules for Zenon, which rely on the simplex and branch and bound methods, and discuss the soundness and completeness of these rules.

4.1 Rules

The arithmetic proof search rules for Zenon are summarized in Fig. 2. These rules do not use global bounds because in practice, these bounds grow too quickly to be useful on non-trivial systems. Among these rules, there are two rules, i.e. Branch and Simplex-Lin, which need parameters and therefore require the proof

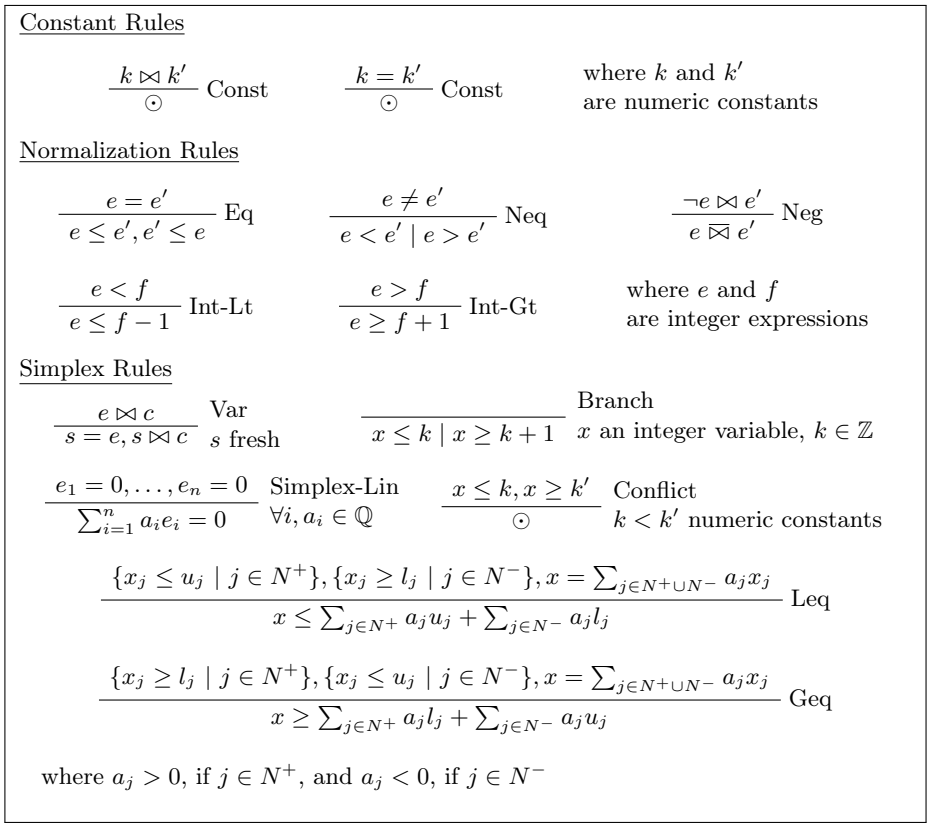

Fig. 2. Proof Search Rules for Arithmetic

search method to choose these parameters. In the following, we describe how Zenon relies on the branch and bound method to make use of these rules.

Each time Zenon processes a new formula, there are two cases. Either the formula is a bound on a variable, in which case it is simply added to the current simplex state. Or it is not, and the rule Var can be applied so that a new variable is generated, and the resulting constraints (which are in general form) can be added to the current simplex state. For every addition to the simplex state, Zenon tries to solve the system of the simplex state. If this yields an unsatisfiable statement, then the explanation is translated into proof search rules and introduced in the proof search tree, effectively closing the current branch. Thanks to the fact that the simplex is incremental, we have a persistent simplex state, which allows us to keep all the work previously done up to a point when the proof search tree branches.

When the simplex algorithm returns an unsatisfiability explanation of the form $x = \sum_i a_i y_i$, we use three proof nodes to close the current branch. First, we use the Simplex-Lin rule to introduce the formula $x - \sum_i a_i y_i = 0$. We can

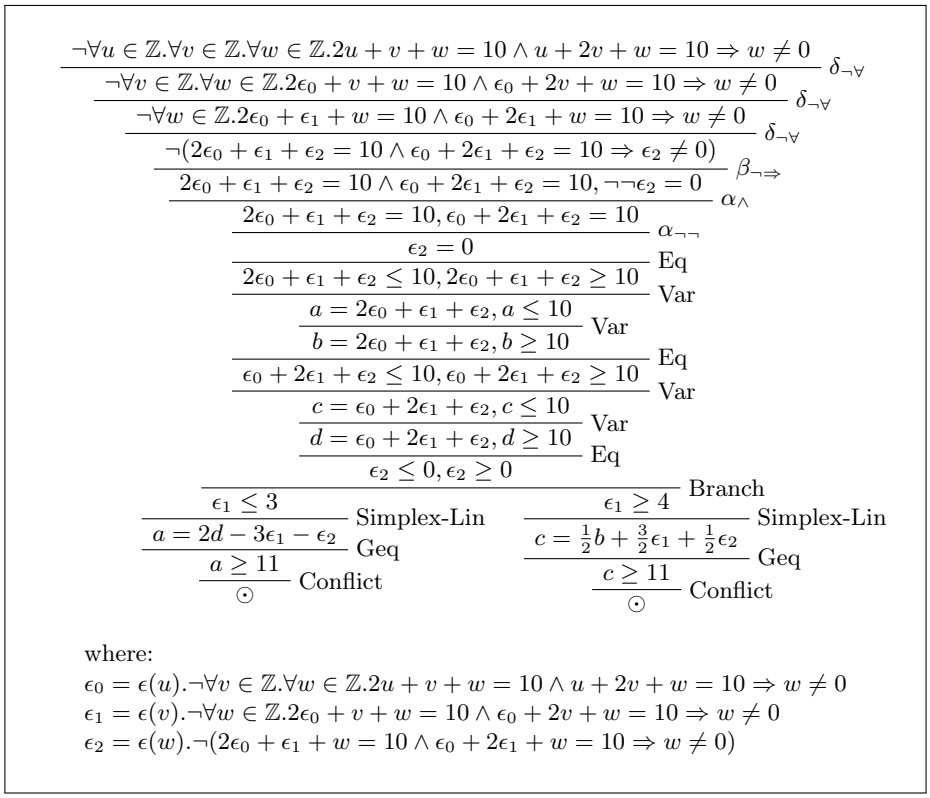

Fig. 3. Proof of Problem ARI178=1

then use either the Leq or Geq rules to deduce a new bound on x using the equality $x = \sum_i a_i y_i$ (which is equivalent to $x - \sum_i a_i y_i = 0$ using some simple rewrite rules; see the next paragraph). Finally, we use the Conflict rule to close the tree, since the simplex guarantees that the newly deduced bound will be in direct conflict with a pre-existing bound of x. For integer systems, the branch and bound strategy returns a tree where nodes are split cases on integer variables, i.e. choices of the form $x \leq k \vee x \geq k + 1$ (see Sec. 3), and leaves are usual simplex explanations. This structure can be easily translated into proof search rules using the Branch rule and the explanation in three steps for the simplex.

It should be noted that the equalities involved in the premises of the arithmetic proof search rules are either of the form $e = 0$, where e is an expression (see the Simplex-Lin rule), or of the form $x = e$, where x is a variable and e an expression (see the Leq and Geq rules). This means that the equalities of the premises must be seen modulo rewriting over a ring structure, which mainly consists in factorizing multiplicative coefficients of expressions (usually variables), computing the result of constant expressions, and moving an expression from one side of a comparison to the other side. In order to keep a compact proof

search tree, these steps of rewriting are not included in the tree (but should be considered when producing a proof object to be checked by an external tool).

As an example of proof with these rules, let us consider the following formula, which comes from the ARI178=1 problem of the ARI category (which is the arithmetic category) of the TPTP library [16]:

$$\forall u \in \mathbb{Z}.\forall v \in \mathbb{Z}.\forall w \in \mathbb{Z}.2u + v + w = 10 \wedge u + 2v + w = 10 \Rightarrow w \neq 0$$

Using the proof search rules of Fig. 2, we can obtain the proof of Fig. 3, where the parameters for the Branch rule come from the application of the branch and bound algorithm (see Sec. 3).

4.2 Soundness

Provided their side conditions are met, the rules presented in Fig. 2 are sound for integer, rational, and real arithmetic. The proof is trivial for all the rules, except for the following rules, which perform more complicated operations:

- Simplex-Lin, which introduces a linear combination of equalities;
- Conflict, which uses transitivity of orders, e.g. $a \leq x \wedge x < b \Rightarrow a < b$, and applies the comparison of ground numeric constants;
- Leq and Geq, for which we need the compatibility of addition and multiplication with orders, e.g. $x \leq a \wedge y \geq b \Rightarrow 2x - y \leq 2a - b$.

4.3 Completeness

The set of rules of Fig. 2 is complete for the satisfiability of rational and real linear arithmetic systems, but not for integer systems since we do not use global bounds (which ensure the termination of the branch and bound algorithm).

Completeness for rational arithmetic comes from the termination and soundness of the simplex algorithm. For real arithmetic, since most of the input languages for automated theorem provers only allow real numeric constants to be rationals[6], we can consider the restriction of a real system S to the rationals and try to solve it with the simplex method, which appears to be complete in this particular case. In fact, there are two possibilities: either the simplex finds a rational solution, then it is also a real solution; or the simplex finds the real system to be unsatisfiable, then we can build a proof search tree using our rules (which are sound for real arithmetic), which proves that the real system is unsatisfiable.

Using simplex optimizations such as Gomory's cuts [12], which reduce the search space, would require to complicate the explanation procedure, i.e the procedure that translates an unsatisfiable result from the simplex into a proof search tree. It might also require to add new arithmetic rules in order to keep the proof search tree as simple as possible. This is why no optimization of the simplex has been considered yet.

[6] We therefore represent real numeric constants using arbitrary precision rationals.

5 Arithmetic Instantiation Mechanism

In order to find instantiations leading to contradictions, we try to find instantiations that satisfy a selected set of formulas. To do so, we introduce arithmetic constraint trees and the notion of counter-example for these trees.

5.1 Arithmetic Constraint Trees

Arithmetic constraint trees (referred to as trees in the following) and the notion of counter-example for these trees are defined as follows:

Definition 1 (Arithmetic Constraint Trees). *An arithmetic constraint tree is a tree whose nodes and leaves are labeled with sets of arithmetic formulas.*

Definition 2 (Cover of Nodes of a Tree). *Given a tree $\mathcal{T}$, and a set of formula $\mathcal{E}$, the set of nodes of $\mathcal{T}$ covered by $\mathcal{E}$ is the least set of nodes $\mathcal{N}$ s.t. for all $n \in \mathcal{N}$, either $\mathsf{label}(n) \cap \mathcal{E} \neq \emptyset$, where $\mathsf{label}(n)$ is the set of formulas labeling the node n, or all children of $\mathcal{N}$ are covered by $\mathcal{E}$.*

Definition 3 (Cover of a Tree). *A set of formulas $\mathcal{E}$ is said to cover a tree $\mathcal{T}$ iff the root of $\mathcal{T}$ belongs to the set of nodes covered by $\mathcal{E}$.*

Definition 4 (Counter-Example of a Tree). *A counter-example of a tree $\mathcal{T}$ is an assignment of the metavariables of $\mathcal{T}$ s.t. there exists a set of formulas $\mathcal{E}$ that covers $\mathcal{T}$ and s.t. all the negation of the formulas of $\mathcal{E}$ are satisfied.*

In order to find a counter-example of an arithmetic constraint tree $\mathcal{T}$, we simply need to solve the negation of a system (a set of formulas) that covers $\mathcal{T}$. To do so, we enumerate a sufficient set of systems that covers $\mathcal{T}$ and try to solve each of them until we find a counter-example. We can enumerate a sufficient set of covering sets with the following formula:

$$\mathsf{cover}(\mathcal{T}) = \{\{f\} \mid f \in \mathsf{label}(\mathcal{T})\} \cup \{ \bigcup_{1 \leq i \leq n} s_i \mid s_i \in \mathsf{cover}(\mathcal{T}[i])\}$$

where $\mathsf{label}(\mathcal{T})$ is the label of the root of $\mathcal{T}$, and $\mathcal{T}[i]$ the i-th children of the root of $\mathcal{T}$. This set is sufficient in the sense that any cover set of a tree $\mathcal{T}$ must be a superset of at least one set in $\mathsf{cover}(\mathcal{T})$.

5.2 Interleaving with Zenon

In Zenon, a proof search tree can be seen as a tree labeled with sets of formulas. To use this tree to find instantiations, we first have to allow Zenon to return a tree with open branches in the case where it did not find any contradiction. We then filter all the formulas in the tree, and keep only the arithmetic constraints to build an arithmetic constraint tree. Finally, we try to find a counter-example of this tree, and once found, we can prove the initial formula by using the $\gamma_{\forall \text{inst}}$

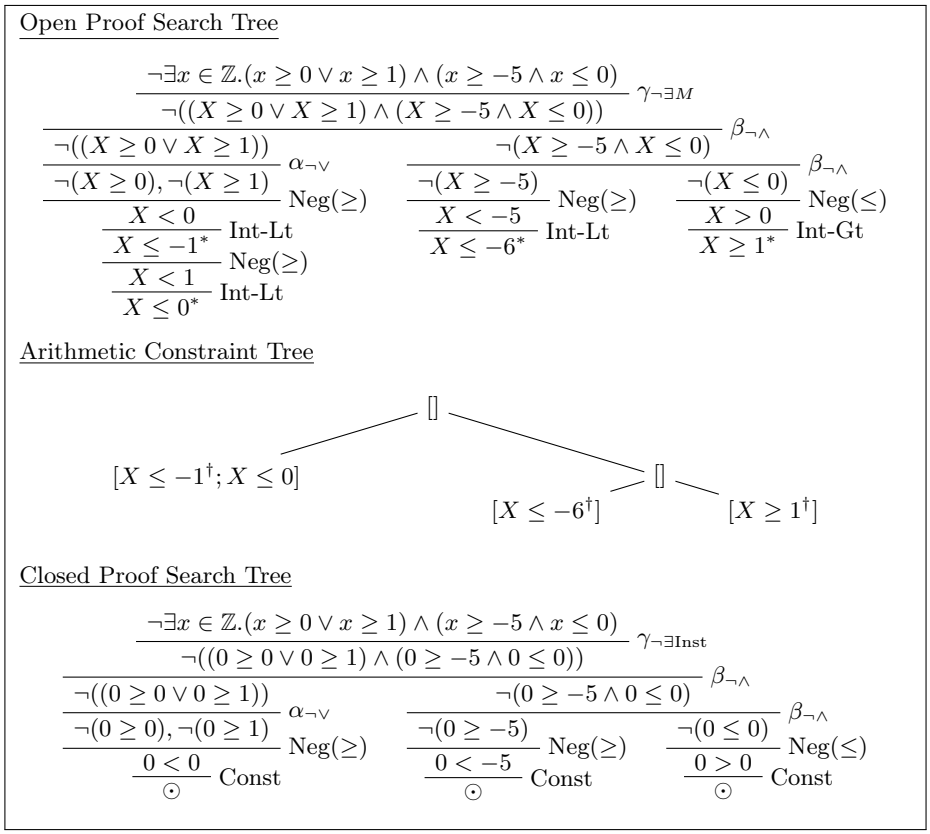

Fig. 4. Proof of Formula 1

and $\gamma_{\neg\exists\text{inst}}$ rules (see Fig. 1) to instantiate the metavariables with the values of the counter-example. Let us illustrate this mechanism with an example:

$$\exists x \in \mathbb{Z}.(x \geq 0 \vee x \geq 1) \wedge (x \geq -5 \wedge x \leq 0) \tag{1}$$

To prove this formula, we first consider its negation, then decompose it using the proof search rules to obtain the open proof search tree of Fig. 4. From this tree, we can get the arithmetic constraint tree of Fig. 4 by keeping the formulas of the open proof search tree that are labeled with "*", and by collapsing the empty nodes. During the enumeration of covering sets, we reach the set S that contains the formulas that are labeled with "†" in the arithmetic constraint tree. We can then solve the system that is formed by the negation of the formulas in S, and which yields the counter-example $X \mapsto 0$. Finally, we can produce the closed proof search tree of Fig. 4 by instantiating X by 0.

With this mechanism, Zenon alternates between regular proof search with the usual proof search rules, and arithmetic solving over open proof search trees to get counter-examples that provide instantiations to close the proof search trees.

This approach is sound as instantiations cannot introduce inconsistencies. This approach is also complete for the validity of purely existential arithmetic formulas, where all the variables are existentially quantified[7]. Given such a formula, if the negation of this formula is unsatisfiable, then there exists a substitution σ of the variables s.t. the resulting ground formula is unsatisfiable. Since the formula is ground, it means that after applying the propositional proof search rules, we have a tree $\mathcal{T}$ s.t. there is an unsatisfiable comparison of numeric constants in each branch of the proof search tree. The substitution σ is a counter-example of $\mathcal{T}$, with a covering set $\mathcal{E}$ s.t. each comparison $e \bowtie f \in \mathcal{E}$ is absurd after substitution by σ. Our enumeration of potential covering sets is s.t. there exists $\mathcal{E}' \in \mathsf{cover}(\mathcal{T})$ s.t. $\mathcal{E}' \subseteq \mathcal{E}$. This means that there is a counter-example σ' of $\mathcal{T}$, with the covering set $\mathcal{E}'$ s.t. it also closes all branches after substitution, and σ' will be found during the proof search since the branch and bound always terminates when there exists a solution.

5.3 Limitations of the Simplex Method

The main limitation of the simplex method is that it is not able to perform abstract computations, i.e. it is only able to handle numeric constants. For instance, if we want to prove the formula $\exists x \in \mathbb{Q}.x \leq a$, where a is a rational constant, we cannot feed the simplex with the formula $X \leq a$, where X is the metavariable corresponding to the existential variable x, because in this context, X and a are fundamentally different: we cannot change the value of a, while we can choose the value of X, but the simplex is not able to make this difference. By extension, this prevents us from dealing with formulas containing both metavariables and ϵ-terms, and therefore with formulas involving alternations of quantifiers.

6 Experimental Results and Proof Certification

We have implemented our extension of Zenon to arithmetic according to what is described in Secs. 4 and 5, using arbitrary precision rationals through the Zarith OCaml library[8], and we have performed some tests using problems from the ARI category (i.e. the arithmetic category) of the TPTP library [16]. We consider 500 problems of this category that only involve linear arithmetic. These tests have been run on an Intel Xeon E5-2660 v2 2.20GHz computer, with a timeout of 60 s and a memory limit of 2 GiB. The results are summarized in Tab. 1, where Zenon extended to arithmetic[9] is compared to two first order automated theorem provers able to deal with arithmetic and the TPTP input formats, i.e. Princess casc-2014-07-04 [15] and Beagle 0.9 (2/7/2014) [4]. The execution time for a prover is the sum of the user and system times taken by the prover, i.e the total CPU time used by the process and its children. It may differ from the real

[7] This approach also handles formulas that are negations of purely universal arithmetic formulas, where all the variables are universally quantified.

[8] See: `https://forge.ocamlcore.org/projects/zarith/`.

[9] Available at: `https://www.rocq.inria.fr/deducteam/ZenonArith/`.

Prover	Proofs	Rate	Total (1) Time (s)	Average (1) Time (s)	Total (2) Time (s)	Average (2) Time (s)
Zenon (arith.)	459	92%	23.33	0.05	23.08	0.05
Princess	491	98%	2129.20	4.34	2048.20	4.52
Beagle	495	99%	678.62	1.37	596.53	1.32

Table 1. Experimental Results over the ARI Category of TPTP

time for provers that use more than one thread, which is the case of Princess and Beagle. The total and average times labeled with "(1)" are computed w.r.t. all the problems, i.e. 500 problems. The total time for a given prover only considers the set of problems that the tool succeeds in proving, i.e. the problems over which the prover reaches the timeout are not included in the total time. The average time is computed as the total time divided by the number of proved problems. The total and average times labeled with "(2)" are computed w.r.t. the problems that are proved by all the tools, i.e. 453 problems.

As can be observed, Zenon is able to prove less problems than Princess and Beagle, but it is noticeably faster over the problems that it succeeds in proving, while proving a reasonable amount of problems. This trend can be seen in Fig. 5, which presents the cumulative times of the provers according to the numbers of proved problems. To obtain the curve for each prover, we consider its run times over all the problems that it proves, sort these times in increasing order, and then plot the cumulative sum of these times. The speed of Zenon is confirmed by the times computed w.r.t. the problems that are proved by all the tools (labeled with "(2)" in Tab. 1). These times show that the time difference between Zenon and the other provers is not due to the other provers taking more time over the problems that are not proved by Zenon, but rather because Zenon is typically faster over the problems that are proved by all the tools.

If we exclude the problems involving alternation of quantifiers, the typical problems not proved by Zenon actually make use of uninterpreted functions. This is due to the lack of exchange of information between the arithmetic extension and the rest of the proof search rules (the equality rules in particular). In fact, the non-trivial arithmetic proof search rules are only applied when the simplex detects an unsatisfiable system, which prevents the propagation of potentially relevant information to other parts of the proof search algorithm.

In the framework of the BWare project [9, 17], this extension of Zenon to arithmetic has been integrated to another extension of Zenon, called Zenon Modulo [8], which extends Zenon to deduction modulo [10]. Zenon Modulo extended to arithmetic has been benchmarked over a set of 12,876 proof obligations coming from the development of industrial applications using the B method [1] and requiring high integrity. It has allowed us to go from 10,340 proved problems (without the extension to arithmetic) to 12,281 proved problems (with the extension to arithmetic), and obtain an increase of almost 20%. This shows that our implementation is scalable, and effective for program verification in particular.

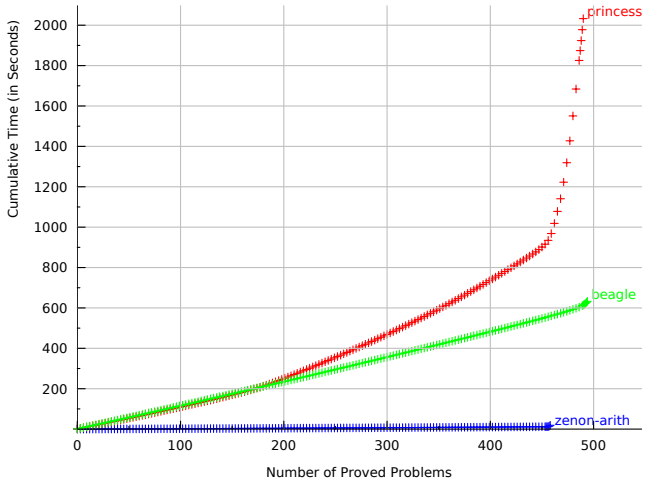

Fig. 5. Cumulative Times according to the Numbers of Proved Problems

Zenon is a certifying automated theorem prover in the sense that it is able to produce proofs checkable by external tools, such as Coq [18] for example. This Coq backend has been extended to support the addition of the arithmetic proof search rules. The main challenge was to translate the implicit rewriting steps that are performed over the arithmetic formulas in the proof search rules. Using this extended backend, all the proofs found by Zenon for the problems of the ARI category have been successfully produced by Zenon and checked by Coq.

7 Related Work

The closest work from our approach is probably the one of the Princess automated theorem prover, which integrates the Omega test [15] with a tableau-based proof search method. Compared to our work, Princess offers a complete procedure for integer problems involving purely universal and purely existential formulas. In our case, we do not ensure this property in the case of purely universal formulas, as we do not use global bounds, which appear to be ineffective in practice. Moreover, Princess proposes a better integration of arithmetic with the other proof search rules, as it is able to deal with uninterpreted predicates (and also with uninterpreted functions by extension). However, compared to Princess, we provide a more efficient implementation (as pointed out by the experimental results of Sec. 6), which is partly due to the fact that the simplex method is more efficient than the Omega test in practice. But the main difference is that our approach is proof producing along the lines of what is proposed in [2], which allows us to increase the level of confidence in our implementation.

Arithmetic is also the area of expertise of SMT solvers. A large part of them, such as CVC4 [3] or Z3 [7], proposes linear (and also non-linear for some of them) arithmetic as a built-in theory, as well as very efficient implementations. Compared to SMT solvers, we offer a better support for the first order layer of arithmetic problems, as SMT solvers relies on pattern-matching (controlled by a system of triggers) rather than unification to deal with instantiation, which is not complete in general. In addition, our experimental results (see Sec. 6) let us hope that our implementation could compete, in terms of time, with some of the most efficient SMT solvers, even though no experiment has been realized yet. Finally, as mentioned previously, our implementation is able to produce proofs, which is not the case of most SMT solvers.

8 Conclusion

In this paper, we have proposed an extension of the Zenon tableau-based first order automated theorem prover to linear arithmetic. This extension relies on the general simplex algorithm to deal with rational systems, as well as on the branch and bound method to deal with integer systems. This extension has been implemented, and this implementation appears to be quite efficient compared to similar first order automated theorem provers, as pointed out by the experimental results over arithmetic problems coming from the TPTP library, and even though it is able to prove less problems than these other provers. As shown by the tests over the benchmark of the BWare project, this implementation also appears to be scalable. In addition, this implementation includes an extension of the Coq backend of Zenon as well, which allows us to produce Coq proofs from arithmetic automated proofs.

As future work, we plan to investigate the introduction of Gomory's cuts [12], which reduce the search space, and which appear to be very effective in combination with the branch and bound method (called the branch and cut method in this case), even though we think that it would require to complicate the explanation procedure. We also aim to realize a better integration of arithmetic with the other parts of the proof search method, in particular to deal with arithmetic formulas involving uninterpreted functions and predicates, even though our implementation is already able to prove difficult problems in this domain (see the problem ARI619=2 with a TPTP ranking of 0.78[10] for example). Finally, we would like to consider mixed problems (involving expressions of distinct arithmetic types) and non-linear arithmetic, which would allow us to deal with all the problems of the arithmetic category of TPTP (i.e. 557 problems).

References

1. J.-R. Abrial. *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, Cambridge (UK), 1996. ISBN 0521496195.

[10] It means that at least 78% of the tested automated theorem provers fail in proving the considered problem.

2. H. Barendregt and E. Barendsen. Autarkic Computations in Formal Proofs. *Journal of Automated Reasoning (JAR)*, 28(3):321–336, Apr. 2002.
3. C. W. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Computer Aided Verification (CAV)*, volume 6806 of *LNCS*, pages 171–177, Snowbird (UT, USA), July 2011. Springer.
4. P. Baumgartner and U. Waldmann. Hierarchic Superposition with Weak Abstraction. In *Conference on Automated Deduction (CADE)*, volume 7898 of *LNCS*, pages 39–57, Lake Placid (NY, USA), June 2013. Springer.
5. R. Bonichon, D. Delahaye, and D. Doligez. Zenon: An Extensible Automated Theorem Prover Producing Checkable Proofs. In *Logic for Programming Artificial Intelligence and Reasoning (LPAR)*, volume 4790 of *LNCS/LNAI*, pages 151–165, Yerevan (Armenia), Oct. 2007. Springer.
6. V. Chvátal. *Linear Programming*. Series of Books in the Mathematical Sciences. W. H. Freeman and Company, New York (NY, USA), 1983. ISBN 0716715872.
7. L. M. De Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4963 of *LNCS*, pages 337–340, Budapest (Hungary), Apr. 2008. Springer.
8. D. Delahaye, D. Doligez, F. Gilbert, P. Halmagrand, and O. Hermant. Zenon Modulo: When Achilles Outruns the Tortoise using Deduction Modulo. In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 8312 of *LNCS/ARCoSS*, pages 274–290, Stellenbosch (South Africa), Dec. 2013. Springer.
9. D. Delahaye, C. Dubois, C. Marché, and D. Mentré. The BWare Project: Building a Proof Platform for the Automated Verification of B Proof Obligations. In *Abstract State Machines, Alloy, B, VDM, and Z (ABZ)*, Lecture Notes in Computer Science (LNCS), pages 126–127, Toulouse (France), June 2014. Springer.
10. G. Dowek, T. Hardin, and C. Kirchner. Theorem Proving Modulo. *Journal of Automated Reasoning (JAR)*, 31(1):33–72, Sept. 2003.
11. B. Dutertre and L. M. De Moura. Integrating Simplex with DPLL(T). Technical Report SRI-CSL-06-01, SRI International, May 2006.
12. R. E. Gomory. An Algorithm for Integer Solutions to Linear Problems. In R. L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, New York (NY, USA), 1963.
13. D. Kroening and O. Strichman. *Decision Procedures: An Algorithmic Point of View*. Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin Heidelberg (Germany), 2008. ISBN 9783540741046.
14. G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., New York (NY, USA), 1999. ISBN 9780471359432.
15. P. Rümmer. A Constraint Sequent Calculus for First-Order Logic with Linear Integer Arithmetic. In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 5330 of *LNCS*, pages 274–289, Doha (Qatar), Nov. 2008. Springer.
16. G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning (JAR)*, 43(4):337–362, Dec. 2009.
17. The BWare Project, 2012. `http://bware.lri.fr/`.
18. The Coq Development Team. *Coq, version 8.4pl6*. Inria, Apr. 2015. `http://coq.inria.fr/`.