REAL AUTOMATION IN THE FIELD*

CÉSAR MUÑOZ† AND MICAELA MAYERO‡

Abstract. We provide a package of PVS strategies that supports automation of non-linear arithmetic. In particular, we describe a semi-decision procedure for the field of real numbers and a strategy for cancellation of common terms.

Key words. Non-linear arithmetic, PVS strategies, real number automation

Subject classification. Computer Science

1. Introduction. While conducting research on the formal safety analysis of Air Traffic Management systems at NASA Langley [3, 2, 5], we have found the verification system PVS [6] very well suited for specifying *avionics systems*. Avionics systems are *hybrid systems*, that is, they are digital systems that interact with the continuous physical environment where they are deployed. These two levels of dynamics, discrete and continuous, can be easily modeled using PVS higher-order logic. Despite that, deductive methods, such as theorem proving, are not usually the first approach to model and verify hybrid systems. Most of the literature on verification of hybrid systems concerns variations of algorithmic methods, such as model checking, on restricted sub-classes of the general problem. This can be explained by the fact that although theorem provers based on higher-order logic, e.g., PVS [6], Coq [1], Isabelle-HOL [7], integrate expressive specification languages with powerful theorem provers, they lack of adequate support for continuous mathematics.

PVS, for instance, incorporates a rich type system supporting predicate sub-typing and dependent types. It also comes with a set of decision procedures that solve problems in a wide range of decidable fields. PVS includes by default a large collection of properties for real numbers[1]. However proofs of formulas involving non-linear arithmetic can be quite cumbersome. For example, consider the formula

$$(a+1)/((a+1)/(b+1)) - b = 1 \tag{1.1}$$

where a, b are positive real numbers. An automatic proof via the PVS command `(GRIND :theories "real_props")` fails to discharge this formula. In [8], Rushby and Shankar show how a small augmentation of the PVS prelude library and a deep knowledge of the PVS rewriting mechanism can provide effective automatic arithmetic simplification. We comment on that approach later.

In the case of a proof by hand, it is useful to know the names of the lemmas included in the prelude library. It may help that some names in this library are mnemonics. One lemma we could use to check Formula 1.1 is `div_div1`: $x/(n0y/n0z) = (x * n0z)/n0y$, where $n0y$ and $n0z$ denote real numbers different from zero. After applying this lemma on Formula 1.1, via `(REWRITE "div_div1")`, we get the formula:

$$(1 + b + (a \times b + a))/(1 + a) - b = 1. \tag{1.2}$$

*This work was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the authors were in residence at ICASE, NASA Langley Research Center, Hampton, VA 23681-2199, USA.

†Main and Conctact Author: ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23681-2199, USA, e-mail: `munoz@icase.edu`.

‡INRIA, Domaine de Voluceau - Rocquencourt - B.P. 105, 78153 Le Chesnay Cedex, France, e-mail: `Micaela.Mayero@inria.fr`.

[1]See for example the theory `real_props` in the prelude library.

Before attempting the next rewriting step, we may have to rearrange this expression. This step is usually done with the proof command `CASE`. The complete proof is a few lines long and none of the steps is really difficult. However, they are painful, at best, when checking large formulas. Consider for example Formula 1.3, taken from the verification of a conflict detection and resolution algorithm developed at ICASE - NASA Langley [5]. The first naive attempt to verify this formula was several pages long.

$$\frac{vi_y \times \sqrt{s^2 - D^2} - vi_x \times D}{vo_y \times \sqrt{s^2 - D^2} - vo_x \times D} = \frac{D^2 - s^2}{\frac{(s^2-D^2)\times voy - D \times vo_x \times \sqrt{s^2-D^2}}{s \times (vi_x \times vo_y - vo_x \times vi_y)} \times s \times vo_x} + vi_x/vo_x \tag{1.3}$$

In [9], Di Vito presents a package of strategies and functions for manipulating arithmetic expressions in PVS. Using this package, we have developed a semi-decision procedure for the field of real numbers. The strategy, called `FIELD`, helped us to reduce the proofs of formulas 1.1, 1.2, and 1.3 to just one line. In Section 2, we describe the strategy and give some examples of usage. In addition to `FIELD`, we have developed strategies for cancellation of common terms, and some support strategies for artihmetic manipulation. These strategies are described in Section 3. Finally, in Section 4, we discuss some issues related to real automation in PVS.

Most of the strategies developed in our package act on *relational formulas*, that is, arithmetic formulas having the form $e \mathrel{\bowtie} f$, where $\mathrel{\bowtie} \in \{=, <, \leq, >, \geq\}$. All the strategies were tuned and tested on real examples taken from the verification of avionics systems conducted at ICASE - NASA Langley.

2. Reals on the Field. `FIELD` is a semi-decision procedure for relational formulas on the field of real numbers. It was originally based on the tactic Field of Coq V7 [4], but it has been extensively adapted to cope with PVS idiosyncrasies. The tactic Field of Coq first removes all the inverses appearing in a given equality and then applies a decision procedure for the ring of real numbers. In PVS, we have extended the procedure to deal with inequalities as well, and instead of using a decision procedure on the ring of real numbers, we use PVS decision procedures on arithmetic expressions.

2.1. Algorithm. Given a relational formula $e \mathrel{\bowtie} f$, `FIELD` is described by the algorithm:

1. Collect inverses $x_1, \ldots, x_n$ of e, f. If $n = 0$, return.
2. Assume $p \neq 0$, where $p = x_1 \times \ldots \times x_n$.
3. Let $e' = p \times e$ and $f' = p \times f$.
4. If $\mathrel{\bowtie}$ is the equality operator, replace the equality $e = f$ with the equality $e' = f'$.
5. If $\mathrel{\bowtie}$ is in $\{<, \leq, >, \geq\}$, replace the inequality $e \mathrel{\bowtie} f$ with the two inequalities (1) $e' \mathrel{\bowtie} f'$, assuming $p > 0$, and (2) $e' \mathrel{\bowtie}' f'$, assuming $p < 0$, where $<'$ denotes $>$, $\leq'$ denotes $\geq$, and so on.
6. For each one of the new formulas:
   (a) Cancel inverses in e', f', and simplify with decision procedures.
   (b) Call `FIELD` with the resulting formula.

As explained in [4], the recursive call is needed to handle the case of nested divisions. In order to remove inverses, an explicit control on the application of distributive laws is required. Unfortunately, PVS applies simplification rules, including distributive laws, after performing a rewriting step. Note, for example, that the rewrite command on Formula 1.1 yields a re-organized Formula 1.2 where multiplications have been completely distributed. In particular, the common term $(a + 1)$ is not longer available for cancellation. Later, we describe a strategy that blocks the automatic application of distribute laws by introducing new variable names.

2.2. Use and Examples. Following is the description of the strategy and some examples of usage. The examples refer to the PVS code in Figure 2.1.

```
a,b   : VAR posreal
x,y   : VAR real
nx,ny : VAR nzreal

ex1 : LEMMA (a+1)/((a+1)/(b+1)) - b = 1

ex2 : LEMMA (1+b+(a×b+a))/(1+a) - b = 1

ex3 : LEMMA x > 1 ⇒
  x/((x-1)×(x+1)) - 1 /((x-1)×(x+1)) ≥ 1 /(x+1) - 1

ex4 : LEMMA x × nx/ny - y × nx/ny = 0 ⇒
  x = y

ex5 : LEMMA 4×((a+1)×b) + ((a+1)×6)×(a+1) = -(a+1)×((x+1)×2) ⇒
  2×b + 3×(a+1) + x + 1  = 0

IMPORTING reals@sqrt
t,vix,viy,vox,voy,s : VAR real
D                   : VAR posreal
kb3d : LEMMA
  vox > 0 ∧ s > D ∧ s × vix × voy - s × viy × vox ≠ 0 ∧
  ((s × s - D × D) × voy - D × vox × sqrt(s × s - D × D)) /
   (s × (vix × voy - vox × viy)) × s × vox ≠ 0 ∧
  voy × sqrt(s × s - D × D) - D × vox ≠ 0
  ⇒
    (viy × sqrt(s × s - D × D) - vix × D) /
    (voy × sqrt(s × s - D × D) - vox × D)
     =
    (D × D - s × s) /
     (((s × s - D × D) × voy - D × vox × sqrt(s × s - D × D)) /
      (s × (vix × voy - vox × viy)) × s × vox) +
    vix / vox
```

FIG. 2.1. *PVS code for examples of* `FIELD`

syntax: `(FIELD &OPTIONAL fnum[1])`
effect: Applies a semi-decision procedure for the field of real numbers on the formula `fnum`.
requirements: If `fnum` is a formula in the consequent, it should have the form $\forall(x_1,\ldots,x_n) : e \lessapprox f$. Otherwise, it should have the form $\exists(x_1,\ldots,x_n) : e \lessapprox f$. In both cases n may be zero.

The command `(FIELD)` takes care of lemmas `ex1` and `ex2`, which correspond to formulas 1.1 and 1.2, respectively. `FIELD` also works on inequalities. The sequence of proof commands `(SKOSIMP)(FIELD)` discharges Lemma `ex3`.

Validation of Formula 1.3 requires a little bit more work. First, we import the theory `reals@sqrt`, which makes part of the NASA Langley PVS library `reals`. This library is available at `http://shemesh.larc.nasa.gov/ftp/larc/PVS-library/pvslib.html`. Finally, we write Formula 1.3 as Lemma `kb3d` in PVS and prove it with the sequence of commands `(SKOSIMP)(FIELD 4)`. Only one TCC of `kb3d` has to be proven by hand; it can be easily discharged using the properties in the theory `sqrt`.

As a simplification strategy, `FIELD` can be used to remove inverses. For example, the sequence of proof commands `(SKOLEM 1 ("nx" "ny" "x" "y"))(FLATTEN)(FIELD -1)` on Lemma `ex4` yields

```
{-1}  x × nx - y × nx = 0
  |-------
[1]   x = y
```

To finish this proof, we multiply formula 1 with `nx`, for example using Di Vito's strategy [9] `MULT-BY`, and then apply the proof command `(ASSERT)`. We may also recognize that the term `nx` in formula `{-1}` can be cancelled. We show a strategy for cancellation of common terms in the next section.

3. Cancellation of Common Terms. `FIELD` is very effective removing inverses, however, by doing that, it tends to introduce common terms as in the case of Lemma `ex4` above. `CANCEL-BY` is a simplification strategy for cancellation of common terms appearing in a relational formula. First, it divides both sides by the common term, then it distributes, and finally, it simplifies the expressions using PVS decision procedures.

3.1. Algorithm. Given a term t and a relational formula $e \lessgtr f$, `CANCEL-BY` is described by the algorithm:

1. Assume $t \neq 0$.
2. If $\lessgtr$ is the equality operator, replace the equality $e = f$ with the equality $e/t = f/t$.
3. If $\lessgtr$ is in $\{<, \leq, >, \geq\}$, replace the inequality $e \lessgtr f$ with the two inequalities (1) $e/t \lessgtr f/t$, assuming $t > 0$, and (2) $e/t \lessgtr' f/t$, assuming $t < 0$, where $<'$ denotes $>$, $\leq'$ denotes $\geq$, and so on.
4. For each one of the new formulas:
   (a) Distribute t on expressions e and f. Let e' and f' be the new formulas.
   (b) Remove inverses in e', f', and simplify with decision procedures.

3.2. Use and Examples. Following is the description of the strategy and some examples of usage.

syntax: `(CANCEL-BY fnum expr)`

effect: Cancels the expression `expr` in the formula `fnum`.

The sequent

```
{-1}  x × nx - y × nx = 0
  |-------
[1]   x = y
```

comming from Lemma `ex4`, Figure 2.1, can be discharged with the command `(CANCEL-BY -1 "nx")`.

The commands `(SKOLEM 1 ("a" "b" "x")) (FLATTEN)` on Lemma `ex5`, Figure 2.1, yield the sequent

```
{-1}  4 × ((a + 1) × b) + ((a + 1) × 6) × (a + 1) = -(a + 1) × ((x + 1) × 2)
  |-------
{1}   2 × b + 3 × (a + 1) + x + 1 = 0
```

It can be discharge with the command `(CANCEL-BY -1 "(a+1)×2")`.

4. "Extra-tegies". In addition to `FIELD` and `CANCEL-BY`, we have implemented some general strategies, in a package called `extra-tegies`, to support automation on real numbers and arithmetic manipulations. This package is complementary to the one developed by Di Vito [9].

4.1. Automation support.

syntax: (GRIND-REALS)

effect: Applies PVS command GRIND with the theories real_props and extra_real_props. The latter one makes part of the package in [9]. GRIND-REALS does no expand definitions.

caveat: Just as GRIND, GRIND-REALS may not terminate.

syntax: (REALS-PROPS &OPTIONAL fnums[*])

effect: Applies PVS command AUTO-REWRITE, with the theories real_props and extra_real_props, on the formulas fnums.

tip: If GRIND-REALS does too much, try REAL-PROPS.

syntax: (NAME-DISTRIBS &OPTIONAL name fnums[*])

effect: Introduces new names, based on name, to block the automatic application of distribute laws in formulas fnums. If no name is given, new names are automatically generated.

usage: Given the sequent, which does not intend to have any logic or arithmetic meaning,

```
{-1}  x×(x + 1) = y
{-2}  y×(y + 1) = x
  |-------
{1}   x×(x + 2) = y×(y + 2)
```

the command (NAME-DISTRIBS "A" 1) uses "A" to create new names:

```
{-1,(A1:)}
      (y + 2) = A12__
{-2,(A1:)}
      (x + 2) = A11__
[-3]  x × (x + 1) = y
[-4]  y × (y + 1) = x
  |-------
{1}   x × A11__ = y × A12__
```

In contrast, (NAME-DISTRIBS :name NIL :fnums 1) generates new names each time the command is invoked.

caveat: If name is null, there is not guarantee that the same names are used when re-running the proof. In that case, explicit reference to expressions containing new names should be avoided. If a name is supplied, the names are guarantee to be the same each time the proof is executed.

syntax: (REPLACES &OPTIONAL fnums [*] in[*] from to hide?[T] dir[LR] step[(SKIP)])

effect: Iterates the PVS command REPLACE to rewrite with the formulas in fnums, respecting the order, the formulas in in. The keys dir and hide? are like in REPLACE. Notice that in contrast to REPLACE, true is the default value of hide?. Instead of using fnums, rewriting formulas can be addressed via from and to. The key step specifies the command to be executed after all the replaces have taken place.

usage: Given the sequent

```
{-1}  Delta = sq(B) - 4×A×C
{-2}  Delta ≥ 0
{-3}  x = (sqrt(Delta) - B) / (2×A)
```

```
  |-------
{1}   A×sq(x) + B×x + C = 0
```

the command `(REPLACES :from -1 :to -3 :in 1)` yields the sequent

```
{-1}  Delta = sq(B) - 4×A×C
{-2}  Delta ≥ 0
  |-------
{1}   A×sq((sqrt(Delta) - B) / (2×A)) +
       B×((sqrt(Delta) - B) / (2×A))
       + C
       = 0
```

while the command `(REPLACES :from -3 :to -1 :in 1)` yields the sequent

```
{-1}  Delta ≥ 0
  |-------
{1}   A×sq((sqrt(sq(B) - 4×A×C) - B) / (2×A)) +
       B×((sqrt(sq(B) - 4×A×C) - B) / (2×A))
       + C
       = 0
```

The latter sequent is also produced with the command `(REPLACES)`.

5. Arithmetic and logic manipulations.

syntax: `(NEG-FORMULA fnum)`

effect: Negates both sides of the relational formula `fnum`.

usage: `(NEG-FOMULA -1)` on sequent

```
{-1}  x < y
{-2}  a = b
  |-------
{1}   x ≠ a + b ∧ y > a - b
```

produces

```
{-1}  -x > -y
[-2]  a = b
  |-------
[1]   x ≠ a + b ∧ y > a - b
```

syntax: `(ADD-FORMULAS fnum1 fnum2 &OPTIONAL :hide?[T])`

effect: Adds relational formulas `fnum1` and `fnum2`. Hides them when key `hide?` is true.

usage: `(ADD-FOMULAS -1 -2 :hide?  NIL)` on the last sequent yields

```
{-1}  -x + a > -y + b
[-2]  -x > -y
[-3]  a = b
  |-------
[1]   x ≠ a + b ∧ y > a - b
```

syntax: `(WRAP-FORMULA fnum str)`

effect: Wraps both sides of the relational formula `fnum` with the string `str`.

usage: `(WRAP-FORMULA -3 "sq")` on the last sequent yields

```
{-1}  sq(a) = sq(b)
[-2]  -x + a > -y + b
[-3]  -x > -y
[-4]  a = b
  |-------
[1]   x ≠ a + b ∧ y > a - b
```

When proving properties on large arithmetic propositions, involving logical operators, it happens that side conditions have to be proven and then *kept* as hypothesis. The PVS command `SPLIT` is not very effective decomposing this kind of formulas, since it symmetrically splits the propositions. We have implemented an asymmetric split strategy called `SPLASH`.

syntax: `(SPLASH fnum)`
effect: Asymmetrically splits a conjunction `fnum` in the consequent (or a disjunction in the antecedent).
usage: `(SPLASH 1)` on the last sequent yields the two sequents

```
{-1}  x ≠ a + b
[-2]  sq(a) = sq(b)
[-3]  -x + a > -y + b
[-4]  -x > -y
[-5]  a = b
  |-------
{1}   y > a - b

[-1]  sq(a) = sq(b)
[-2]  -x + a > -y + b
[-3]  -x > -y
[-4]  a = b
  |-------
{1}   x ≠ a + b
```

Note the hypothesis `{-1}` in the first sequent. Now try `(UNDO)`, and compare with `(SPLIT 1)`, where this hypothesis is missing.

REFERENCES

[1] B. Barras, S. Boutin, C. Cornes, J. Courant, J. Filliatre, E. Giménez, H. Herbelin, G. Huet, C. Muñoz, C. Murthy, C. Parent, C. Paulin, A. Saïbi, and B. Werner, *The Coq Proof Assistant Reference Manual – Version V6.1*, Tech. Report 0203, INRIA, August 1997.

[2] R. Butler, V. Carreño, G. Dowek, and C. Muñoz, *Formal verification of conflict detection algorithms*, in Proceedings of the 11th Working Conference on Correct Hardware Design and Verification Methods CHARME 2001, vol. 2144 of LNCS, Livingston, Scotland, UK, 2001, pp. 403–417. A long version appears as report NASA/TM-2001-210864.

[3] V. Carreño and C. Muñoz, *Aircraft trajectory modeling and alerting algorithm verification*, in Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000, J. Harrison and

M. Aagaard, eds., vol. 1869 of Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 90–105. An earlier version appears as report NASA/CR-2000-210097 ICASE No. 2000-16.

[4] D. Delahaye and M. Mayero, *Field: une procdure de dcision pour les nombres rels en coq.* Electronic Proceedings of JFLA'2001. Available at `http://pauillac.inria.fr/jfla/2001/`, January 2001.

[5] G. Dowek, C. Muñoz, and A. Geser, *Tactical conflict detection and resolution in a 3-D airspace*, Tech. Report NASA/CR-2001-210853 ICASE Report No. 2001-7, ICASE-NASA Langley, ICASE Mail Stop 132C, NASA Langley Research Center, Hampton VA 23681-2199, USA, April 2001.

[6] S. Owre, J. M. Rushby, and N. Shankar, *PVS: A prototype verification system*, in 11th International Conference on Automated Deduction (CADE), D. Kapur, ed., vol. 607 of Lecture Notes in Artificial Intelligence, Saratoga, NY, June 1992, Springer-Verlag, pp. 748–752.

[7] L. Paulson, *Isabelle: The next 700 theorem provers*, in Logic and Computer Science, P. Odifreddi, ed., Academic Press, 1990, pp. 361–386.

[8] J. Rushby and N. Shankar, *Arithmetic simplification in PVS.* Final Report for SRI Project 6464, Task 15; NASA Langley contract number NAS1-20334, December 2000.

[9] B. D. Vito, *A pvs prover strategy package for common manipulations.* Manuscript. Available at `http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS2-library/pvslib.html`, 2001.