The Dynamic Real Closure implemented in Axiom

Grégoire Lecerf
GAGE, Centre de mathématiques (CNRS URA 169)
École Polytechnique, F-91128 Palaiseau Cedex France.
lecerf@ariana.polytechnique.fr

Abstract

Let R be a real field, we are interested in computing in the real algebraic closure of R. We show in this paper a new implementation in Axiom [1]computer algebra system. No hypothesis on R is required. Let P be a polynomial in $R[X]$, and a a real root of P. Computations with a are handled in a generic way so that the user does not need to specify which real root a of P is considered. Cases are discussed automatically by the program through the method of dynamic evaluation.

Key words: real parameter; dynamic evaluation; real closure; Axiom; computer algebra.

1 Introduction

Let R be a real field (see [11] definition 1.1.1), we are interested in computing in its real algebraic closure $\overline{R}$([11] theorem 1.3.2).

1.1 Motivations

Consider the following elementary problem:
Let u, v, w be real numbers, what is the value of the following expression:

$$\lim_{w \to +\infty} uw^3 + vw$$

Here is Maple's answer:

```
>limit(u*w^3+v*w,w=+infinity);

                              signum(u)(+infinity)
```

This answer is true only if $u \neq 0$. Other cases are forgotten. The right answer is: 0 in case $u = 0$ and $v = 0$, $+\infty$ in case $u > 0$ or $u = 0$ and $v > 0$, and $-\infty$ in case $u < 0$ or $u = 0$ and $v < 0$.

So far no computer algebra system is able to return the right answer to this problem. We present in this article a new package for Axiom computer algebra system, allowing to do this in a particular situation: u and v are real roots of given polynomials. For instance see our result with $u^3 - u = 0$ and $v^3 - v = 0$ in 4.3.1.

[1]Axiom, computer algebra system, is a trademark of NAG

1.2 History

In the above example u and v are called **parameters**. A parameter describes any element of a given set of values. It is said to be algebraic when it satisfies a polynomial equation (detailed in 2).

The present work is the continuation of:

- The Dynamic Algebraic Closure(DAC): (in [12], [5], [1], [3], [2], [4]). Let K be a field, the DAC allows to compute with algebraic parameters of one algebraic closure (see [18] chap V,§2 proposition 2.3) of K.
- The Dynamic Constructible Closure (DCC): (in [21], [20], [15], [19]). Let K be a field, the DCC allows to compute with parameters in the algebraic closure of K, but the parameters are not necessarily algebraic.
- A theoretical work about dynamic evaluation: description of the above closures in term of sketches, see [6], [7].

Our package, the Dynamic Real Closure (DRC), is the analog of DAC, but in the situation of real numbers. It has been described for the first time in [8]. We show here an implementation in Axiom. The code of the DRC relies mainly on the one of the DCC.

It is not the first time that a real closure has been implemented. The implementation of R. Rioboo, described in [17] and [22] (code available at [16]), is based on the interval root coding. The user introduces a real algebraic number u by specifying a polynomial P such that $P(u) = 0$ and an integer i such that u is the ith real root of P (the real root is ordered by the order of the real closure). Some comparisons between this implementation and the DRC have been done in [10].

In the following section we shall introduce sets with parameters. In the third section, the DRC is constructed by induction on the number of parameters introduced. The last section is devoted to the presentation of the source and to some examples using it.

2 Computation with parameters

In the following S represents a set. From computational point of view S has two kinds of basic operations : **constructional** operations and **relational** operations. The signature of the first ones is $S^k \longrightarrow S$, where k is the arity of the operation. The signature of the later is $S^k \longrightarrow \{0, 1\}$. The constants of S are constructional operations of arity 0.

2.1 Definition

We define sets with parameters according to computational point of view. A set S is said to be computational when its elements have a representation and operations on them are given algorithmically.

Definition 1 *Let $U = \{u_i,\ i \in I\}$ be a set of symbols indexed by I, an effective and infinite well ordered set, and S a computational set. We denote by $S\langle U\rangle$ the set S complemented with the parameters U. $S\langle U\rangle$ is defined by the following rules:*

- *The elements of $S\langle U\rangle$ are the **terms** build with the elements of S, the parameter and the constructional operations of S.*
- *$S\langle U\rangle$ has a function called* `newParameter()`*. It returns a new parameter of U. The latter is the element u_n where n is the smallest element i of I such that u_i has not be introduced yet.*
- *$S\langle U\rangle$ has got the same constructional operations as S. If f is one of them of arity k and $x_1, \ldots, x_k$ are elements of $S\langle U\rangle$, $f(x_1, \ldots, x_k)$ is the element of $S\langle U\rangle$ represented by the term $f(x_1, \ldots, x_k)$.*
- *$S\langle U\rangle$ is linked to a **case** $C(U)$. $C(U)$ is a first order formula on S without quantifier. Its atomic formulae are of type $g(x_1, \cdots, x_n)$ where the x_i are in $S\langle U\rangle$, and g a relational operation of S.*
- *$S\langle U\rangle$ has got a function* `currentCase()` *which returns $C(U)$ and, a function* `setCase(C)` *to set $C(U)$ to C.*
- *$S\langle U\rangle$ has the same relational operations as S. Let g be one of them of arity k, and $x_1, \ldots, x_n$ elements of $S\langle U\rangle$. Evaluation of $g(x_1, \cdots, x_n)$ creates a **split**, that is $S\langle U\rangle$ splits into two $S\langle U\rangle$: the first linked to the case $C(U) \wedge g(x_1, \ldots, x_k)$ and the second to $C(U) \wedge \neg g(x_1, \ldots, x_k)$. In the first $S\langle U\rangle$ computation goes on with the answer **true** and the second one with the answer **false**.*

Example: $I = \mathbb{N}$, $S = \mathbb{Q}$. S has the constructional operations $+, -, \times, /$ and the relational operations $=, <$. Let $U = \{u_1, \ldots, u_n, \ldots\}$ and consider the following function:

```
foo : () -> IN
  u1:= newParameter()
  u2:= newParameter()
  a:= u1 + u2
  b:= u1 * u2
  if a = b then return(0) else return(1)
```

The result of `foo()` is `0` in case `u1+u2=u1*u2` and `1` in case $\neg$`u1+u2=u1*u2`.

Remarks: If S is a ring the $/$ function is not defined on the whole S but only on the unity of S. But the signature is $S^2 \longrightarrow S$. An error is raised when the second argument is not invertible.
The terms are purely syntactical objects but we do not impose their representation.

Practically, the above definition requires more features. An effective implementation must specify:

1. A representation of the terms. It must be thought with great care to control term growing.
2. A representation of the case. The case should be in a natural, organised and readable form.
3. An algorithm to decide when a case is empty. This avoids to carry on a computation in an empty case.

Further in this paper we describe our answers for the DRC.

2.2 Using parameters to construct a real closure

Let R be a real field. R is supposed to be computational with the constructional operations $0, 1, +, -, \times, /$ and the relational operations $=, <$. Let us now consider $\overline{R}$, the real algebraic closure of R. We are interested in computing in $R\langle U\rangle$: the Dynamic Semi-Algebraic Closure (DSAC)[2] of R.

A case of $R\langle U\rangle$ describes a semi-algebraic set ([11] chapter 2). We restricted ourselves to the special situation when the parameters are introduced as being a real root of a polynomial.

Definition 2 *A parameter u of a real closed field k is said to be (real) algebraic if it is a (real) root of a specified polynomial of $k[X]$.*

We define and denote by $\overline{R}\langle U\rangle_a$ the DRC of R as $R\langle U\rangle$ where the function `newParameter()` has been replaced by the function `realRootOf :` $\overline{R}\langle U\rangle_a[X] \longrightarrow \overline{R}\langle U\rangle_a \cup$ `failed`. `realRootOf(P)` returns `failed` if P has no real root, else a new parameter u_n satisfying $P(u) = 0$.

The DRC has all the operations to be a real closure. In the next section we describe effective computation in DRC.

3 The Dynamic Real Closure

When computing with the DRC, parameters are introduced one by one by the user. In what follows we suppose that $I = \mathbb{N}$ and the parameters are introduced in the order $u_1, u_2, \ldots, u_n$. In this section we show a method to compute effectively in the DRC. This method is inductive on the number n of parameters introduced. First we explain the situation when only one parameter is introduced. Then we generalise.

3.1 Computing with one algebraic real parameter

On $\overline{R}\langle U\rangle_a$ `realRootOf` is supposed to be called only once, it introduces the parameter u. u satisfies $P(u) = 0$, where $P \in \overline{R}[X]$. With this hypothesis we are ready to answer the three questions stated at the end of 2.1.

Proposition 1 *The elements of $\overline{R}\langle U\rangle_a$ can be represented by a polynomial r of $\overline{R}[X]$, such that $deg(r) < deg(P)$. Such a representation is called polynomial **reduced**.*

Proposition 2 *The case can be represented by a conjunction of the form:*

$$P(u) = 0 \wedge sign(Q_1(u)) = \sigma_1 \wedge \ldots \wedge sign(Q_n(u)) = \sigma_n \quad (1)$$

Where $\sigma_i \in \{+, -\}$. We have $deg(P) > 0$, Q_i prime with P and $deg(Q_i) < deg(P)$ for all i. All these polynomials are monic and square free, n is bounded by the number of real roots of P.

[2] DSAC is the analog of DCC in the situation of real numbers

One can write it shorter (1):

$$P(u) = 0 \wedge sign(\vec{Q}(u)) = \vec{\sigma} \tag{2}$$

We call P the **algebraic part** of the case and the rest the **semi-algebraic part**. The decision of the emptiness of a case is ensured by:

PROPOSITION 3 *The **Simultaneous Inequality algorithm (SI)**. Let R be a real closed field.*
Input: `(P,lQ)`. `P` *is a polynomial in* $R[X]$, `lQ` *is a list of polynomials in* $R[X]$.
Output: `[(n1,s1),(n2,s2),...,(nl,sl)]`. *The* `ni` *are strictly positive integers and the* `si` *are lists of signs* `+,-,0` *of the same size as* `lQ`. *The number of real roots of* `P` *satisfying sign*(`lQ`) =`si` *is* `ni`. $\sum_{i=1}^{l}$`ni` *is exactly the number of real roots of* `P`.

This algorithm is described in [14]. It relies on **Sylvester-Habicht** polynomial sequences, and the use of a structure called **Tarski data type**.

Remark: The bound on the size of the semi-algebraic part of a case given in proposition 2 is a consequence of this algorithm.

We are now ready to show the algorithm used to compute in the DRC. We use Axiom's terminology (see [13]): `%` refers to $\overline{R}\langle U\rangle_a$and `Rep(x)` is the polynomial representing `x`.

Algorithm DRC:

```
reduce(x) : % ⟶ % ==
  Let P be the algebraic part of currentCase()
  Compute r, the remainder of the Euclidean
    division of Rep(x) with P.
  return(r(u))

x +( resp -,*) y : %×% ⟶ % ==
  r:= reduce (Rep(u) + (resp -,*) Rep y)
  return(r(u))

x / y : %×% ⟶ % ==
  if y=0
  then raise error(``Division by zero'')
  else
    compute a Bezoutian equality of Rep(y) and P:
      1 = Rep(y)*r + P*s
    return(reduce(x*r(u)))

x = y : %×% ⟶ {true,false}
  z:= Rep(x-y)
  Let P be the algebraic part of currentCase()
    and sign(lQ)=ls be the semi algebraic part.
  pt:= gcd(z,P)
  pf:= P/pt
  lqt:= [ remainder(q,pt) for q in lQ ]
  lqf:= [ remainder(q,pf) for q in lQ ]
  lqt:= [ realSquareFree(q) for q in lqt ]
  lqf:= [ realSquareFree(q) for q in lqf ]
  ct:= (pt=0) and sign(lqt)=ls
  cf:= (pf=0) and sign(lqf)=ls
  decide using SI(pt,lqt) and SI(pf,lqf)
    whether ct and cf are empty.
    (at this step one can remove
    useless polynomials in lqt and lqf)
  if ct is empty then setCase(cf) and return(false)
  if cf is empty then setCase(ct) and return(true)
  split the computation:
    return true in case ct
    return false in case cf

x < y : %×% ⟶ {true,false}
  z:= x-y
  if z=0 then return(false)
  c:= currentCase()
  r:= realSquareFree(Rep(z))
  ct:= c and r<0
  cf:= c and r>0
  with SI decide the emptiness of ct and cf
  if ct is empty then return false
  if cf is empty then return true
  split the computation:
    return true in case ct
    return false in case cf

realRootOf(p) : %[X] ⟶ % ∪ failed
  p1:= squareFree(p)
  c:= p=0
  with SI decide if c is empty
  if c is not empty then setCase(c) return(u)
  else return(failed)
```

`squareFree(p)` returns the square free part of `p`. `realSquareFree(p)` removes the factors of even power in `p` and returns the resulting square free part. It computes a square free decomposition with the algorithm of [9].

Let us now prove propositions 1 and 2. Proposition 1 is ensured by the `reduce` function. In `=` we note that `pt` and `pf` are monic and square free since `P` is. `ct` and `cf` satisfies the required conditions. In the function `<`, `r` is monic, squarefree, has a degree strictly less then `P` and is prime with `P`(since r(u) is not zero).

Remark: `=` is not implemented from `<` because it is always interesting to split algebrically the case. It leads to faster computations and shorter outputs.

3.2 The problem of computing with several parameters

The DRC $\overline{R}\langle U\rangle_a$of R in the situation of one parameter u presented above inherits the operations of R. It can be regarded as being $R(u)$, the algebraic extension of R with the real algebraic number u. We can iterate the construction, adding one new parameter v, and obtain $R(u)(v) = R(u, v)$. This method leads to an effective way of computing in the DRC. We have implemented it in [10].

This is the method used in the DCC. The drawback of it is that it leads to redundant computations and excessive outputs. The reason is that the algorithm deciding the emptiness of a case requires some tests on the ground field. Let us show it on an example:

Let u be a real root of the polynomial $P_1(u) = u^3 - 3u - 1$ and v a real root of the polynomial $P_2(v) = v^3 - (u+1)v^2 + (u^2 - 1)v - u^2 - 2u$. Here is what we obtain just after introducing u and v.

```
value is 1 in case
   3            2      2            2
  v  + (- u - 1)v  + (u  - 1)v - u  - 2u =0 and

   3                  2  7      2          2
  u  - 3u - 1 =0  ,  u  + - u + - <0  ,  u  - u - 2 <0
                         3      3
```

```
,
value is 1 in case
   3            2     2          2
  v  + (- u - 1)v  + (u  - 1)v - u  - 2u =0 and

   3                2   7     2          2
  u  - 3u - 1 =0  ,  u  + - u + - <0  ,  u  - u - 2 >0
                          3     3
,
value is 1 in case
   3            2     2          2
  v  + (- u - 1)v  + (u  - 1)v - u  - 2u =0 and

   3                2   7     2
  u  - 3u - 1 =0  ,  u  + - u + - >0
                          3     3
```

When SI computes the number of real roots of P_2 it needs to know the sign of some elements of $R(u)$, and so splitting the case. As SI performs many tests it involves many splits, producing redundant computations and excessive outputs. The expected value for this example is:

```
value is 1 in case
   3            2     2          2
  v  + (- u - 1)v  + (u  - 1)v - u  - 2u = 0 and

   3
  u  - 3u - 1 = 0
```

We obtain this result with the method presented in the next section. The solution has been proposed in [8]: data structures are extended in order to compute in several case at once. This is the solution implemented. We have also introduced new features with respect to the DCC.

3.3 Hiding splits

In the previous section we have seen that deciding the emptiness of a case implies many splits. In this section we propose a solution: hiding unwanted splits.

The main idea is to allow disjunctions in the cases. Disjunctions appear when we do not want a case to split. For this purpose we have introduced the function `internalTest`. We present it in a general context.

Let us come back to the notations of 2.1. S is a computational set, and U is a set of parameters of S indexed by $I = \mathbb{N}^*$. We call i the **level** of the parameter u_i. The level of an element x in $S\langle U\rangle$is the maximum of the levels of the parameters in the term representing x. The level of an element of S is 0. The level of a set of elements of $S\langle U\rangle$is the maximum level of its elements. Suppose we have an algorithm Π satisfying the following conditions:

i Π is composed of two parts Π_1 and Π_2.

ii `[af1,...,afk]` = Π_1`(af)`. `af` and the `afi` are atomic formulae of $S\langle U\rangle$. The level of the `afi` is strictly less than the level of `af`. If the level of `af` is zero then Π_1 returns an empty list. More over Π_1 does not perform any test on expressions of level greater or equal to the level of `af`.

iii `a=`Π_2`[b1,...,bk]` is an element of `Record( caseTrue:` $\{$`empty, notempty`$\}$`, caseFalse:` $\{$`empty, notempty`$\}$`)`. The `bi` are boolean values. If `c` is a non empty conjunctive case appeared after the end of the call of Π_1`(af)` and containing the atomic formulae `afi` for all `i` such that `bi` is true and `not afi` such that `bi` is false then `a.caseTrue` tells whether `c and af` is empty or not and `a.caseFalse` tells it for `c and not af`.

The case $C(U)$ can be written: $\vee_i c_i(U)$ and each $c_i(U)$ is $\wedge_j af_{i,j}$. This is disjunctive conjunctive form. We call the $c_i(U)$ the **subcases** of $C(U)$. In DRC each $c_i(U)$ is a case verifying described in proposition 2.

Remarks: Practically Π_2 requires extra arguments, but for the sake of presentation we omit them. Also it is not judicious to represent the case is it disjunctive conjunctive form. We explain in [10] the tree representation we have chosen.

Let us now introduce the new functions:

<u>`internalTest`</u>
Input: `af`, an atomic formula of $S\langle U\rangle$.
Output: Nothing.
Code:

```
internalTest(af) ==
  if level of af is 0 then return()
  rl:= Π1(af)
  for each af' in rl do
    internalTest(af')
  Cold:= currentCase()
  Cnew:= ∨∅
  for each c subcase of Cold do
    bl:=getValue(c,rl)
    r:=Π2(bl)
    if r.caseTrue=empty then
      Cnew:= Cnew∨(c∧ notaf)
      next
    if r.caseFalse=empty then
      Cnew:= Cnew∨(c∧af)
      next
   Cnew:= Cnew∨(c∧af)∨(c∧ notaf)
  setCase(Cnew)$S<U>
  return()
```

<u>`getValue`</u>
Input:`(c,rl)`. `c` is a conjunctive case, `rl` is a list of atomic formulae.
Output:`bl`, a list of boolean of the sale size of `rl`. The ith element of `bl` is $true$(resp $false$) is `c` contains `rl[i]`(resp `not rl[i]`. It raises an error when neither case is possible.

<u>`externalTest`</u>
Input: `af`, an atomic formula of $S\langle U\rangle$.
Output: the boolean value of `af`.
Code:

```
externalTest(af) ==
  if level(af) is 0 then compute its values on S
  internalTest(af)
  C:= currentCase()
  Ct:= ∨ c, c subcase of C containing af
  Cf:= ∨ c, c subcase of C containing not af
  if Ct has no subcase then return(false)
  if Cf has no subcase then return(true)
  split the answer
    return(false) in case Cf
    return(true) in case Ct
```

Suppose that we know whenever an atomic formula in the case come from an `internalTest` or an `externalTest`. We call an **exterior**(resp **interior**) atomic formula when it has been inserted in the case by the `externalTest`(resp `internalTest`).

PROPOSITION 4 *1. The function* **internalTest** *stops.*

2. *Subcases of a case are never empty.*

3. *Subcases of a case are pairwise disjoint.*

4. *A case is never empty.*

5. *After* **internalTest(af)** *every subcase contains either* **af** *or* **not af**.

6. *The subcases of a case have the same exterior atomic formulae.*

7. *The case is set equivalent to the conjunction of its exterior atomic formulae.*

Proof : We prove it by induction on the level of the set of parameters introduced.

If level of `af` is 0 then: `internalTest(af)` does nothing,`externalTest(af)` just evaluates `af` on S.

Suppose that level of `af` is $n > 0$. Examine: by hypothesis (ii) on Π_1 `rl` has level at most $n-1$ and tests appear on level strictly less the $n-1$. Thus after line 2, by induction the proposition is still true. The same in line 5. Note that (5) ensures that `getValue` in line 8 does not raise any error. Thus `internalTest(af)` stops. We need to show that `Cnew` satisfies the properties of the proposition. By hypothesis (iii) on Π_2 we conclude that all the subcases of Cnew are not empty. They are clearly pairwise disjoint, and since `Cold` is not empty so is `Cnew`. The exterior atomic formulae of `Cold` and `Cnew` are the same, and `Cnew` is set equivalent to `Cold`. We conclude that `Cnew` is set equivalent to the conjunction of its exterior atomic formulae. Now let us examine `internalTest(af)`: we need just to prove points 6 and 7. We will prove it for the case `Ct`, case `Cf` is similar. Because of 4 each subcase of `C` contains either `af` or `not af`, so `Ct` is set equivalent to `C`$\wedge$`af`. And the subcases of `Ct` have the same exterior atomic formulae of `C` plus `af`. `C` is the conjunction of its atomic formulae $\wedge_i$`eafi`, and so `Ct` is set equivalent to $\wedge_i$`eafi`$\wedge$`af`, which is precisely the conjunction of its exterior atomic formulae. □

The last point we explain is the construction of the decision algorithm Π.

3.4 The decision algorithm

Deciding the emptiness of a case is a crucial question. With the methods presented here cases are tested at each step of modification. So the decision algorithm must be incremental on the number of parameters introduced and the number of atomic formulae in the case.

We explain, briefly, how to build the algorithm Π used in the previous section. Suppose that we have an algorithm π to decide if a case with one parameter(see 3.1) is empty or not and satisfying the conditions of 3.3. At the level n, Π is π running with the parameter u_n, the other parameters being considered as constants.

This concludes the presentation of the methods used to compute in the DRC. The next section is devoted to the presentation of the structure of the code and some examples using it.

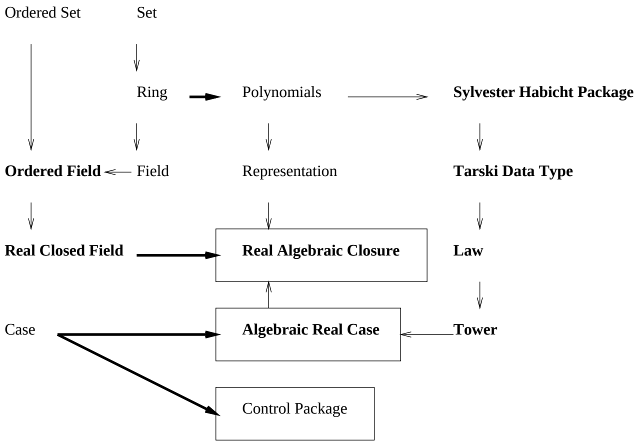

Figure 1: *Overview of the program*

4 Implementation in Axiom, user guide

For Axiom's terminology the reader can refer to [13]. First we describe the structure of the code. More details can be found in [10] and consulting the code via Axiom's hyperDoc. Then we show various examples of using the DRC.

4.1 Description of the source

The structure of the code of the dynamic real closure is similar the one of the DCC. The figure 1 shows a summary overview of the complete program. Thick arrows represent the hierarchy between categories and domain, other arrows show the logical dependency between the objects. The domains in frame are the only ones to be known by the user. The objects in bold are the ones specific to the DRC, the rest comes from the DAC and the DCC.Note that the whole code of the DRC is more than $140Kb$ of pure axiom code.

Sylvester Habicht Package and Tarski Data Type contains the SI algorithm. Law and Tower are domains for representation of the case level by level (see [12]).

Different implementations have been already discussed in [15]. One of the main questions is: how to implement the splits ? The natural tools seem to be a $fork()$ like in C language. It is not directly available through Axiom (the old compiler). The solution is to use mutable variables, which are a kind of global variables, but safer. More details can be found in [21].

4.2 User guide

We describe here, through some simple examples[3] the way of computing in the dynamic real algebraic closure. The main functions to know are:

- `allCases(f)`: evaluates dynamically the function `f` given in argument.
- `refresh()`: removes the constraints on the parameters. It is necessary to use it before using `allCases`.

1. `K` is a real field;

[3]The times printed with the result have been obtained on a SUN Sparcstation 1000M with 4 40Mhz sparc processors and 196 Mb of ram.

2. I is the dynamic real algebraic closure of K;
3. CP is the package for computing all branches;
4. C is the domain dealing with the cases of the dynamic closure.

4.2.1 Example: Introduction of parameters

We introduce the parameters u, v, w in order to find if the following sytem has real solutions or not.

$$\begin{cases} (u^3-3u-1)(u^2-1)=0 \\ (v^3-(u+1)v^2+(u^2-1)v-u^3-u^2+u+1) \\ \times(v^2-u^2)=0 \\ (w^2+v^2+u^2+u-1)uv=0 \end{cases}$$

```
 system() : INT ==
  x:=indeterminate()$I
  uf:=realRootOf('u,(x^3-3*x-1)*(x^2-1))
  uf case failed => 1
  u:=uf.parameter
  vf:=realRootOf('v,
        (x^3-(u+1)*x^2+(u^2-1)*x-u^3-u^2+u+1)*(x^2-u^2))
  vf case failed => 2
  v:=vf.parameter
  wf:=realRootOf('w,(x^2+v^2+u^2+u-1)*u*v)
  wf case failed => 3
  0

refresh()$C
allCases(system)$CP

[value is 3 in case v = 0 and u + 1 = 0,
                                       2
 value is 0 in case w = 0 and v  - 1 = 0 and u + 1 = 0,
 value is 3 in case v = 0 and u - 1 = 0,
 value is 3 in case w "failed" and v "failed" and u
   "failed",
                           2    2    2
 value is 0 in case w  + v  + u  + u - 1 = 0 and
     5           4    3          2         2
    v  + (- u - 1)v  - v  + (u + 1)v  + (- 2u  - u)v
         2
     + 3u  + 7u + 2 = 0

 and
     3
    u  - 3u - 1 = 0]

Type: List DynamicRealClause(Fraction Integer,Integer)
Time: 0.29 (IN) + 8.43 (EV) + 0.45 (OT) + 9.11 (GC)
   = 18.28 sec
```

4.3 Some advanced examples

4.3.1 Advanced Example 1 : using the limit function

We have written another category adding more properties to the real closure in order to use the Axiom's domain *Expression* [4] on it.

Let us now look at the problem introduced in the beginning :

$$\lim_{w \to +\infty} uw^3 + vw$$

we can now solve it with algebraic parameters. For instance u and v such that $u^3-2u=0$ and $v^3-2v=0$.

```
K:=FRAC INT
EI:= EDRCLOS K
EXP:= Expression EI
FRat:= FRAC POLY EI
ANS:= Union(OrderedCompletion FRat,"failed")
CP:=DynamicRealControlPackage(K,ANS)
C:=DynamicRealCase(K)

limitExample() : ANS ==
  x:= indeterminate()$EI
  uf:= realRootOf('u,x^3-2*x)
  u:= uf.parameter
  vf:= realRootOf('v,x^3-2*x)
  v:= vf.parameter
  w: FRat := coerce('w)
  f:= u*w^3+v*w
  limit(f,'w=%plusInfinity)

refresh()$C

allCases(limitExample)$CP1
[value is 0 in case v = 0 and u = 0,
 value is - infinity in case
     2
    v  - 2 = 0 , v < 0 and u = 0,
 value is  + infinity in case
     2
    v  - 2 = 0 , v > 0 and u = 0,
 value is - infinity in case
                 2
    v = 0 and u  - 2 = 0 , u < 0,
 value is  + infinity in case
             2
 v = 0 and u  - 2 = 0 , u > 0,
 value is - infinity in case
  2             2
 v  - 2 = 0 and u  - 2 = 0 , u < 0,
 value is  + infinity in case
  2             2
 v  - 2 = 0 and u  - 2 = 0 , u > 0]
Type: List DynamicRealClause(Fraction Integer,
Union(OrderedCompletion Fraction Polynomial
ExtendedDynamicRealClosure Fraction Integer,"failed"))
Time: 8.45 (IN) + 8.20 (EV) + 5.87 (OT) + 1.42 (GC)
   = 23.94 sec
```

Note that this answer is now quite a natural one. Note that we use Axiom's function to compute the limit without any modification.

4.3.2 Advanced Example 2 : Integration

Let us now look at the computation of:

$$\int \frac{dx}{x^2+u}$$

where u is an algebraic parameter satisfying $u^3-2u=0$.

```
K:=FRAC INT
EI:= EDRCLOS K
EXP:= Expression EI
ANS:= Union(EXP,LIST EXP)
CP:=DynamicRealControlPackage(K,ANS)
C:=DynamicRealCase(K)

refresh()$C
```

[4] *Expression*(R) where R is an ordered set is a domain involving symbolic functions such as *sqrt*, *log*, *cos* ...

```
integrateExample() : ANS ==
  x:= indeterminate()$EI
  uf:= realRootOf('u,x^3-2*x)
  u:= uf.parameter
  w: FRAC POLY EI:= coerce('w)
  f:= 1/(w^2+u)
  integrate(f,'w)

allCases(integrateExample)$CP

            1
[value is - - in case u = 0,
            w
                             +-----+
                             |   1       2
             +-----+    2u w |- - u  + w  - u
          1  |  1           \|   2
 value is -  |- - u log(----------------------)
          2 \|  2                2
                                w  + u
               2
      in case u  - 2 = 0 , u < 0,
                         +---+
                         |1
             +---+     u |- u
             |1         \|2
 value is -  |- u atan(-------) in case
            \|2               w
    2
   u  - 2 = 0 , u > 0]

Type: List DynamicRealClause(Fraction Integer,
                             Union(Expression
 ExtendedDynamicRealClosure Fraction Integer,
 List Expression
 ExtendedDynamicRealClosure Fraction Integer))
Time: 0.16 (IN) + 53.23 (EV) + 35.43 (OT) + 14.08 (GC)
   = 102.90 sec
```

Note that the result is exactly what we expected.

5 Conclusion

We have presented a new implementation in Axiom to compute with real algebraic parameters while using all the Axiom's functions without any modification. We have been able to adapt the code of the Dynamic Constructible Closure for our purposes.

This implementation is a new step showing how it is possible to compute with real algebraic parameters and a first step towards computing with general real parameters.

ftp :

The DRC is available on `ftp: medicis.polytechnique.fr` in `/pub/src/dynamic_evaluation/`, the file is `drc.tar.gz`.

Acknowledgments :

I thank D.Duval for having given me the opportunity to work in this interesting field, J.Davenport for his welcome towards me at the University of Bath, and I especially thank Teresa Gómez-Díaz, whose help has been very important to me.

References

[1] C. Dicrescenzo, and D. Duval. Symbolic and algebraic computation. *Lecture note in Computer science, Springer Verlag 358* (1989), 440–446.

[2] D. Duval. *Diverses questions relatives au calcul formel avec des nombres algebriques.* PhD thesis, Université de Grenoble 1, 1987.

[3] D. Duval. Simultaneous compoutation in fields of arbitrary characteristic. In *Computer and mathematics 89* (1989), Springer-Verlag, pp. 321–326.

[4] D. Duval. Algebraic numbers : an example of dynamic evaluation. 429–445.

[5] D. Duval. Evaluation dynamic et cloture algebrique en axiom. *J. of pure and Applied Algebra 99* (1995), 267–295. ftp://posso.ibp.fr/pub/softwares/Axiom/General-Closures/.

[6] D. Duval, and J.-C. Reynaud. Sketches and computation - i: basic definition and static evaluation. In *Math.Struct. in Com-. Science*, vol. 4. Cambridge University Press, 1994.

[7] D. Duval, and J.-C. Reynaud. Sketches and computation - ii: dynamic evaluation and applications. In *Math.Struct. in Computer Science*, vol. 4. Cambridge University Press, 1994.

[8] D. Duval, and L. González-Vega. Dynamic evaluation and real closure. *IMACS Symposium SC-1993* (1993).

[9] D. Y. Y. Yun. On square-free decomposition algorithms. In *ACM Symposium on Symbolic and Algebraic Computation* (1976).

[10] G. Lecerf. Dynamic evaluation and real closure, implementation in axiom. École normale Supérieure de Paris, 1996. Mémoire de DEA, ftp://medicis.polytechnique.fr/pub/src/dynamic_evaluation/.

[11] J. Bochnak, M. Coste, and M.-F. Roy. *Géometrie algébrique réelle.* Springer-Verlag, 1987.

[12] J. Della Dora, C. Dicrescenzo, and D. Duval. About a new method for computing in algebraic number field. In *EuroCal'85, LNCS 204* (1985), pp. 289–290.

[13] Jenks, and Sutor. *Axiom The Scientific Computation System.* Springer-Verlag, 1992.

[14] M.-F. Roy. Basic algorithm in real algebraic geometry and their complexity : from sturm theorem to the existential theory of reals. Tech. rep., Universit de Rennes, 1996.

[15] P.A. Broadbery, T. Gómez-Díaz, and S.M. Watt. On the implementation of dynamic evaluation. In *ISSAC'95* (1995), pp. 77–84.

[16] R. Rioboo. Real closure : Axiom sources. ftp://posso.ibp.fr/pub/softwares/Axiom/RealClosures.

[17] R. Rioboo. Real algebraic closure of an ordered field: Implementation in axiom. In *ISSAC'92* (1992), pp. 206–215.

[18] S. Lang. *Algebra*. Addison Wesley, 1965.

[19] T. Gómez Díaz. Let t be a triangle, is it isocele ? *11th European workshop on computationl geometry CG'95* (1995).

[20] Teresa Gómez-Díaz. Examples of using dynamic constructible closure. In *IMACS Symposium SC-1993* (1993).

[21] Teresa Gómez-Díaz. *Quelques application de l'évaluation dynamique*. PhD thesis, Université de Limoges, 1994. ftp://medicis.polytechnique.fr/pub/src/dynamic_evaluation/.

[22] Z. Legatsikas, R. Rioboo, and M.-F. Roy. Generic computation of the real closure of an ordered field. In *IMACS Symposium SC-1993* (1993).