Dynamic Evaluation and Real Closure Implementation in Axiom [1]

Grégoire Lecerf

June 29, 1996

[1] Axiom is a trademark of NAG

Abstract

The following document relates the work I completed at the University of Bath, in the context of my DEA[1] from 19th March to 26th June. I was supervised by J.H.Davenport. The original idea was proposed by D.Duval. The aim of this work was to implement a dynamic real closure in Axiom. This report describes the way I achieved it and shows some examples of using it.

[1] 5th year of the university in France

Contents

History

The first use of the dynamic evaluation goes back to 1985 with the D5 package implemented in Lisp for Reduce [2]an algebraic closure ([15], [6]). Simultaneous computations were performed in parallel without any user assistance. Few years later the program was translated into Axiom (the old [3]) [5]. In the meanwhile another application of dynamic evaluation was implemented in Axiom : a field with an arbitrary characteristic ([7]). Then T.Gómez-Díaz in her thesis implemented the constructible closure in Axiom [3] to compute with general parameters. After D.Duval and L.L.González-Vega tried to extend the techniques to the real numbers, in [11], an attempt was done to implement it.

The applications are definitely numerous : polynomial system solving, automatic geometry theorem proving ([3], [2] and [4]) even some industrial problems can be solved in an efficient way, in robotics for instance see [18].

The dynamic evaluation is not something very easy to explain even if it seems to deal with very natural problems. D.Duval and J.-C.Reynaud propose a presentation in term of sketchs, in [9] and in [10]. Some other explanations can be found in [8].

The position of this present work was to complete a dynamic real algebraic closure in Axiom, avoiding as much as possible redundancy with T.Gómez-Díaz's constructible closure. The main structure of the code was already written. The main work was to understand it and reuse it to add real tools for the $<$ test. To my knowledge such attempts to make a computer algebra system working with parameters is unique, and is a totally new concept which tends to be quite efficient and useful. Some small other points of view have already come out (in [14]). However none of them has led to interesting efficient and concrete results.

The *ftp* addresses for all the above programs can be found at the end, with the references the main addresses are :

$$ftp://posso.ibp.fr/pub/softwares/Axiom/GeneralClosures/$$

and

$$ftp://medicis.polytechnique.fr/pub/src/dynamic\_evaluation/$$

[2] Reduce is a computer algebra system

[3] Scratchpad was developed in the IBM's reseach center in Yorktown

1 Introduction

First to understand the main idea of this work, we present some concrete easy problems. Then we introduce some definitions.

1.1 Motivations

We present some problems that no computer algebra system can solve correctly. u denotes any real number. The problems are elementary, we tried to run them on Maple [4]and Axiom [5]. Here is what we have obtained.

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

   Maple session :

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

   Axiom session : after having spent 10 min to find a package for this problem, I encountered a bug.
   The result we would like to obtain is the following :

   - 0 in case $u = 0$ and $v = 0$.
   - $+\infty$ in case $u = 0$ and $v > 0$.
   - $-\infty$ in case $u = 0$ and $v < 0$.
   - $+\infty$ in case $u > 0$.
   - $-\infty$ in case $u < 0$.

2. $$A = \begin{pmatrix} 1 & u \\ u & 1 \end{pmatrix}$$

   We want to compute the signature of A. First we must notice that neither Maple, nor Axiom propose any functions to compute the signature of a symmetric matrix. If we had such a function, the result we would like to have is :

   - $(1, 0)$ in case $u^2 - 1 = 0$.
   - $(1, 1)$ in case $u^2 - 1 > 0$.
   - $(2, 0)$ in case $u^2 + 1 < 0$.

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

   The Maple session :

[4]Maple is a trademark of Waterloo Maple Software
[5]Axiom is a trademark of NAG

```
> int(1/(x^2+u),x);

                                           x
                                   arctan(----)
                                           1/2
                                          u
                                   ------------
                                        1/2
                                       u
```

Axiom tends to do better :

```
 frame0 (83) ->integrate(1/(x^2+u),'x)

                    2      +---+
          1     (x   - u)\|- u  + 2u x         +-+
          - log(----------------------)      x\|u
          2             2                atan(-----)
                       x  + u                  u
   (83)  [----------------------------,-----------]
                     +---+                +-+
                   \|- u                \|u
 Type: Union(List Expression Fraction Integer,...)
```

Nevertheless, this result omitss the case $u = 0$ and we do not know the conditions corresponding to each cases.

1.2 What is a parameter ?

Our aim is not to write again some new functions for integration or limit computation and so on. We would like to make the computer understand by itself what a parameter is. Our goal is to dispose of a real field where parameters can be manipulated very easily. The notion of parameter in mathematics is very often used but never properly described. In the common sense a parameter u of a set S represents any element of a set. Besides one can need to restrict the set of value of u when cases need to be distinguish. Here is what one can found in a small dictionary of mathematics [13].

Parameter : a constant or variable that distinguishes special cases of a general mathematical expression. For example, the general form of the equation for a line, y=mx+c contains parameters m and c, representing the gradient and y-intercept of any specific line.

Constant : 1. A fixed quantity or numerical value 2. A symbol that is assigned a specific fixed entity under an interpretation. Constants contrast with variables, which range over a set of entities.

Variable : a mathematical entity that can stand for any of the members of a given set. The members of the set constitute 'values' of the variable, and the set itself defines the variable's 'range' (i.e. the possible values that it may make).

A parameter for us has first a symbol to represent itself, then a type to say what kind of operations can be applied to it, and last a set of value, which describe all the possible values it can be.

For example, with the set of integers one can use the parameter p to represent the set of the prime number. Its symbol is p, its type is integer and its set of value is the set of the prime number.

1.3 Real closed field and Real closure

We recall some basic definitions. More details can be found in [12] or in [16], for instance.

Definition 1 *A real field K is a field where -1 is not a sum of squares.*

Definition 2 *A field R is said to be real closed if it is real, and if any algebraic extension of R which is real must be equal to R. In other words R is maximal with respect to the property of reality in an algebraic closure.*

For example, the field of real numbers is a real closed field.

By real closure of K we mean a real closed field R which is algebraic over K and extends the ordering of K.

Theorem 1 *Let K a real field. Then there exists a real closure of K. If R is real closed, then R has a unique ordering. The positive elements are the squares of R. Every positive element is a square, and every irreducible polynomial of odd degree in $R[X]$ has a root in R. We have $\overline{R} = R(\sqrt{-1})$. ($\overline{R}$ is the algebraic closure of R).*

The real closure is unique up to an isomorphism of real fields. In particular the number of real roots of a polynomial is independent of the closure.

These definitions are not suitable for a computational point of view. We describe the real closure R of the field K as a real closed field supplying the usual operations and $realRootOf(P)$ where P is an univariate polynomial over R. $realRootOf(P)$ returns the set of the real roots of P.

2 Computation with parameters

Dynamic evaluation is a very general method which generalises the usual evaluation methods. In [9] and [10] a general framework is presented to explain, in terms of sketches, the basis of dynamic evaluation. Our goal is not to present this work, but just a particular case. We are just interested in computations with parameters. We describe the way such computation have been expressed so far in Axiom.

Note that we introduce the notion of parameters in a general way. The next example will not be solved by our program, since it is only able to deal with algebraic parameters.

2.1 Sets with parameters

In the following S is a set in the mathematical point of view. S has two kinds of operations : constructional operations and relational operations. The constructional operations are of type $S^k \longrightarrow S$. Usually $k = 2$, for instance $+$ or $*$ or $/$. And k may be one for the $-$ operation. The relational operations are of type $S^k \longrightarrow \{0, 1\}$. Let $\mathcal{R}$ be such a relation, of arity k. 1 corresponds to the interpretation true, and 0 to false.

Definition 3 *$U = \{u_1, \ldots, u_n\}$ a set of parameters of S. We denote by $S < U >$ the domain built over S and U in the following way :*

- *$S < U >$ has the same mathematical properties as S. Thus the same operations with the same signatures. The elements are represented by a tree. The nodes are the constructional operations, and the leaves are elements of $S \cup U$.*
- *$S < U >$ is described by C a case on U. C is a condition describing the set of the values of the parameters. C is a first order formula on S without quantifiers, the free variables are the u_i and the atomic formulae are of type $\mathcal{R}(a_1, \ldots, a_k)$ where a_i are elements of $S < U >$ and $\mathcal{R}$ is a relational operation.*
- *Deciding the result of a relational operation $\mathcal{R}(a_1, \ldots, a_k)$ can cause a split : the computation is divided in two computations. One with the case $C(U) \wedge \mathcal{R}(a_1, \ldots, a_k)$ and true for answer to the relation test, and the other $C(U) \wedge \neg\mathcal{R}(a_1, \ldots, a_k)$ with false answer.*
- *we define $val(C(U)) = \{(v_1, \ldots, v_n) \in S^n | C(v_1, \ldots, v_n) \text{ is true}\}$. The set of values for U in S.*

Proposition 1 *Every function from $S < U >$ may cause splits. Let*

$$f : S < U >^k \times D \longrightarrow A$$

$f(x_1, \ldots, n_k, y)$ returns not only one result but may return several. Each result is linked to a case. We can express it $((r_1, C_1(U)), \ldots, (r_l, C_l(U))$, where $r_i \in A$. Moreover $val(C_i(U)) \cap val(C_j(U)) = \emptyset$ for all $i \neq j$. $(r_i, C_i(U))$ is called a clause.

Proof : By induction on the construction of a function.

The reason to call this kind of set dynamic is that its intrinsic properties change during the computation, while with usual evaluation, called static, the properties of the set does not change.

2.2 An example of a computation with parameters

We consider R, a real closed field as the ground field, we introduce two parameters u_1 and u_2. The constructional operations are $*, +, -, /$, and the relational operations are $=, <$. At the beginning the case $C(U)$ is $true$, so that $val(C(U)) = R^2$. Let consider the following problem :

$$M = \begin{pmatrix} u_1 & 1 \\ 1 & u_2 \end{pmatrix} \in \mathcal{M}(R < U >)(2,2)$$

We want to determine the signature of this symmetric matrix.

We write a function **signature**.

$$signature : Symmetric(\mathcal{M}(R < U >)(2,2)) \longrightarrow N \times N$$

Let us note

$$M = \begin{pmatrix} a & b \\ b & c \end{pmatrix} \in \mathcal{M}(R < U >)(2,2)$$

Here is one possible algorithm :

1. if $ac - b^2 = 0$
   (a) if $a + c = 0$ return$((0,0))$
   (b) if $a + c < 0$ return$((0,1))$
   (c) return$((1,0))$
2. if $ac - b^2 < 0$ return$((1,1))$
3. if $a + c < 0$ return$((0,2))$
4. return$((2,0))$

Let's now describe this algorithm on the matrix previously introduced : the tree figure 1 describe the execution of the algorithm.

The result is :

$$(0,0) \ in \ case \ u_1u_2 - 1 = 0 \wedge u_1 + u_2 = 0$$

$$(0,1) \ in \ case \ u_1u_2 - 1 = 0 \wedge u_1 + u_2 \neq 0 \wedge u_1 + u_2 < 0$$

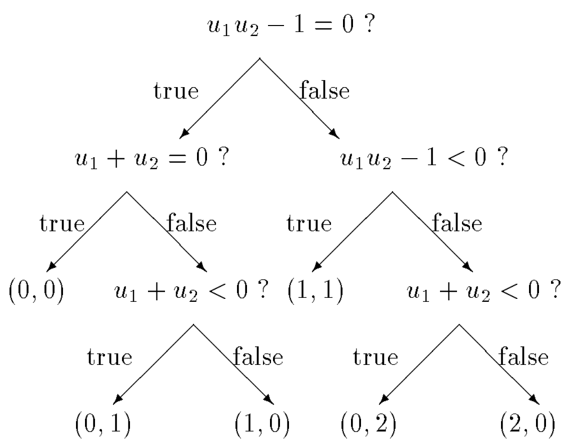
$$(1,0) \ in \ case \ u_1u_2 - 1 = 0 \wedge u_1 + u_2 \neq 0 \wedge u_1 + u_2 \not< 0$$

$$(1,1) \ in \ case \ u_1u_2 - 1 \neq 0 \wedge u_1u_2 - 1 < 0$$

$$(0,2) \ in \ case \ u_1u_2 - 1 \neq 0 \wedge u_1u_2 - 1 \not< 0 \wedge u_1 + u_2 < 0$$

$$(2,0) \ in \ case \ u_1u_2 - 1 \neq 0 \wedge u_1u_2 - 1 \not< 0 \wedge u_1 + u_2 \not< 0$$

Figure 1: A splitting tree



2.3 Dynamic closures

Definition 4 Algebraic parameter *If S is at least a Ring, we can call algebraic a parameter when the case implies the parameter to be a root of a polynomial.*

For example $S = Z$ as ground ring. $Z < u, v >$, $C(u) = u^2 - 1 \wedge v > 0 \wedge u > 0$, u is algebraic and not v.

We give now the definitions of some different dynamic closures.

- **The dynamic algebraic closure (DAC)** : it concerns any field K. $DAC(K)$, the DAC of K allows to compute in the algebraic closure of k with algebraic parameters. It provides the following operations :
  - $+ - */$, the usual field operations.
  - $=$ equality test, which can cause the splits.
  - $rootOf(P)$ returns either an algebraic parameter u satisfying $P(u) = 0$ or "failed" in case that P is constant.
- **The dynamic constructible closure (DCC)** : it concerns any field K. $DCC(K)$, the DCC of K allows to compute in any field containing the algebraic closure of K, with any kind of parameter. It provides the following operations :
  - $+ - */$, the usual field operations.
  - $=$ equality test, which can cause the splits.
  - $newParameter()$ returns a new parameters without any initially condition.
- **The dynamic real closure (DRC)** : it concerns any real field K. $DRC(K)$ allows to compute in the real closure of a K with algebraic parameters. It provides the following operations :

- $+ - * /$, the usual filed operations.
- $=$ the equality test, which can cause some splits.
- $<$ the sign test, which can also cause some splits.
- $realRootOf(P)$ returns a parameter which has for constraint to be a real root of the polynomial P, if possible, and "failed" otherwise.

- **The dynamic semi-algebraic closure (DSAC)** : it concerns any real field K. $DSAC(K)$ allows to compute in every field containing the real closure of a K with general parameters. It provides the following operations :
  - $+ - * /$, the usual filed operations.
  - $=$ the equality test, which can cause some splits.
  - $<$ the sign test, which can also cause some splits.
  - $newParameter()$ returns a new parameter without any initial constraint.

The aim of this report is to show how we have implemented in Axiom a real closure with algebraic parameters. We recall (see the history of the beginning) that the DAC is described in [5], the DCC is described mainly in [3] and the DRC in [11]. The two first closures have already been implemented. Hence the code of a dynamic closure has already been very well structured.

The examples introduced so far are relevant to the dynamic semi-algebraic closure, thus do not concern the implementation we did.

2.4 More specifications

From the previous example 2.2 two important points must call our attention :

1. Can we ensure that a case is empty or not ?

2. Is there a way the specify that a case in a minimal canonical form ?

In general, both answers are no. The first question is an undecidable problem, and the second is an ill-formulated problem. The interested reader can found some algorithms for simplification in [1]. These questions are sharply difficult. For instance, the integer ring with the power operation, leads to undecidable problems : the Diophantian equations. So we can answer these questions only in particular cases. We will see later the solutions chosen for the real closure. In the algebraic closure, both answers are quite easy. Nevertheless, in general we can follow this strategy : if we can decide whether a case is empty or not, when a relational operation occurs, we split only in the case it is necessary. That is to say if $C(U)$ is the case and $R(x,y)$ the relational test, we study whether $C(U) \wedge R(x,y)$ or $C(U) \wedge \neg R(x,y)$ is empty. If one of them is empty, then there is no split to perform and case is unchanged.

3 Real tool box

We give here a precise description of the algorithms used to determine the emptiness of our cases. For the proofs and the questions of complexity see [17].

The ground field is R a real closed field. And we consider here $R < U >$ the dynamic corresponding field with the set U of algebraic parameters. We will see in the next section that the emptiness of a case relies on the emptiness of the system :

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

where P and Q are in $R[X]$, and u is the parameter, $\sigma_i \in \{+, -\}$. Note that P will always be supposed non zero and no hypothesis are needed for the Q_i. We will call also :

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

We will introduce the Tarski data type structure. It allows us to answer in an efficient way the question :

$$val(C(u)) = \emptyset \ ?$$

The following sections describes this structure. It is divided in two independent parts : one performing computations in the dynamic field and another one just combining the results in performing only linear algebra computations on the rational field.

$\#$ denotes the cardinal function, its result is in $N \cup \{+\infty\}$.

3.1 Sylvester-Habicht's sequences

In this section we describe a well-known algorithm to compute

$$\#val(P(u) = 0 \wedge Q(u) > 0) - \#val(P(u) = 0 \wedge Q(u) < 0)$$

We will call $d(P, Q)$ this quantity. This computation is called a *Sturm Query*.

Before, we need some definitions.

Definition 5 *We define the function δ.*

$$\delta_k = (-1)^{k(k-1)/2}$$

We call $cf_i(P)$ the ith coefficient of the polynomial P.

Definition 6 Pseudo-remainder
We call $rem(P, Q)$ the usual remainder of Q with respect to P. We define prem.

$$prem(P, Q) = cf_q(Q)^{p-q+1} rem(P, Q)$$

where $p = sup(deg(P), deg(Q) + 1)$ and $q = deg(Q)$.

Definition 7 Sylvester-Habicht sequence *Let P and Q be two polynomials. Let's call $Sres(P, Q)$ the subresultants sequence. We define the Sylvester-Habicht $SyHa(P, Q)$ sequence :*

$$SyHa_j(P, Q) = \delta_{p-j}(P, Q)$$

where $p = sup(deg(P), deg(Q) + 1)$.

The Sylvester-Habicht sequence is just a generalised polynomial pseudo remainder sequence. For general results about generalised polynomial pseudo remainder sequence one can read [20]. So the following algorithm just compute the modified subresultant sequence.

Algorithm 1 Sylvester-Habicht sequence
Input : *the polynomials* P *and* Q.
Output : *the Sylvester-Habicht sequence* $SyHa(P,Q)$

$$H_j = SyHa_j(P,Q)\ ,\ 0 \leq j \leq sup(deg(P), deg(Q)+1)$$

Initialisation :

$$p := sup(deg(P), deg(Q)+1),\ H_p := P,\ h_p := 1,\ H_{p-1} := Q,\ j := p$$

Next Step : H_j, h_j, *and* $H_j - 1$ *are already known with* H_j *and* h_j *non zero and* $k = deg(H_{j-1})$.

1. *if* $k < j-1$ *then* $H_k := \delta_{j-k} \frac{cf_k(H_{j-1})^{j-1-k}}{h_j^{j-1-k}} H_{j-1}$
2. *if* $k < j-2$ *and* $k < l < j-1$, $H_l := 0$
3. $H_{k-1} := \delta_{j-k+2} \frac{prem(H_j, H_{j-1})}{h_j^{j-k+1}}$
4. $h_k := cf_k(H_k)$ *and* $j := k$ *goto Next Step.*

End : *When* H_0 *is computed, that is when* $j \leq 1$.

It is important to note that the divisions are exact and thus the sequence can be computed over a ring.

Definition 8 Sign counting function
Let $\mathcal{P} = [P_0, \ldots, P_n]$ *be a sequence of polynomials and* a *be an element of* $R \cup \{-\infty, +\infty\}$. *We define* $W(\mathcal{P}, a)$ *the number of changes of signs of* $\mathcal{P}$ *at* a :

1. *Remove from* $\mathcal{P}$ *the polynomials which are zero, to obtain a new sequence of polynomials* $[Q_0, \ldots, Q_s]$.
2. *Define* $W(\mathcal{P}, a)$ *as the number of changes of signs in the sequence* $[Q_0(a),$-$\ldots, Q_s(a)]$ *as follows* :
   - *count 1 variation of signs for the groups (whatever the number of zeros in the middle)* :
     $[+, 0, \ldots, 0, -]$ *and* $[-, 0, \ldots, 0, +]$.
   - *count 2 variations of sign for the groups* :
     $[+, 0, 0, +]$ *and* $[-, 0, 0-]$.
   - *count 0 variations of sign in the other cases.*

Definition 9

$$W_{SyHa}(P,Q) = W(SyHa(P,Q), -\infty) - W(SyHa(P,Q), +\infty)$$

where P *and* Q *are in* $R[X]$.

Proposition 2 **Sturm query**

$$d(P,Q) = W_{SyHa}(P, P'Q)$$

This proposition give an algorithm to compute $d(P,Q)$.

Let's give an example. Consider $P = x^3 - 3x - 1$ and $Q = x^2 - 1$. Then $P'Q = 3 * x^4 - 6x^2 + 3$. It it obvious to note that $d(P, p'Q) = d(P, Q_1)$ where Q_1 is the remainder of $P'Q$ with respect to P. Thus $Q_1 = 3x^2 + 3x + 3$.

$$SyHa_0(P, Q_1) = -243$$

$$SyHa_1(P, Q_1) = 27x$$

$$SyHa_2(P, Q_1) = 3x^2 + 3x + 3$$

$$SyHa_3(P, Q_1) = x^3 - 3x - 1$$

So we note $\mathcal{P} = [-243, 27x, 3x^2 + 3x + 3, x^3 - 3x - 1]$, there is no polynomial to remove in this case. When we evaluate this list at $-\infty$ we get $[-, -, +, -]$ which has 2 change of sign. In $+\infty$ we get $[-, +, +, +]$ which has 1 change of sign. Thus $W(\mathcal{P}, -\infty) = 2$ and $W(\mathcal{P}, +\infty) = 1$. Thus $d(P,Q) = 1$.

3.2 Tarski Data Type

This section shows a well known algorithm to compute the set of sign of a sequence of polynomials $[Q_1, \ldots, Q_n]$ over the real roots of a polynomial P. We only need some linear algebra tools and the Sturm query function. We describe it with the hypothesis that each Q_i is prime with P, so that P has no common root with any Q_i. The general algorithms can be found in [17].

3.2.1 Definition

A Tarski data type (TDT) has the following structure.

$$< P, [Q_1, \ldots, Q_n], \mathcal{B}, \mathcal{H}, \mathcal{A}, \mathcal{C}, \mathcal{V} >$$

where :

$$\mathcal{B} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_t \end{pmatrix} \quad \mathcal{H} = \begin{pmatrix} H_1 \\ \vdots \\ H_t \end{pmatrix}$$

$$\mathcal{A} = \begin{pmatrix} a_{1,1} & \ldots & a_{1,t} \\ \vdots & & \vdots \\ a_{t,1} & \ldots & a_{t,t} \end{pmatrix} \quad \mathcal{C} = \begin{pmatrix} c_1 \\ \vdots \\ c_t \end{pmatrix} \quad \mathcal{V} = \begin{pmatrix} v_1 \\ \vdots \\ v_t \end{pmatrix}$$

- t is a positive integer, called the size of the TDT.
- $\beta_i = [\beta_{i,1}, \ldots, \beta_{i,n}]$ is a list of signs of size n, $\beta_{i,j} \in \{+, -\}$.
- $H_i = \Pi_{j=1}^{n} Q_j^{\epsilon_{i,j}}$ for some $\epsilon_{i,j} \in \{0, 1\}$.
- $a_{i,j}$ is in $\{-1, 0, 1\}$.
- c_i is a non negative integer.

- v_i is an integer.

Properties of this structure :

1. $\#val(P = 0 \wedge sign(\vec{Q}) = \beta_j) = c_j$
2. $v_j = d(P, H_j)$
3. $\mathcal{V} = \mathcal{A} \times \mathcal{C}$ and $\mathcal{A}$ is an invertible matrix.

3.2.2 Algorithm

We describe an incremental algorithm.

Initialisation : it corresponds to the case $n = 0$, that is $\vec{Q}$ is empty. The TDT is :

$$< P, [\,], ([\,]), (1), (1), (r), (r) >$$

where r is $d(P, 1)$, the number of real roots of P.

Incrementation phase : we suppose that we know a TDT for P and $[Q_1, \ldots, Q_n]$. The following steps allows us to calculate a TDT for P and $[Q_1, \ldots, Q_{n+1}]$.

Algorithm 2 TDT incremental construction

1. $\mathcal{B} = \begin{pmatrix} (\beta_{1,1}, \ldots, \beta_{1,n}, +) \\ \vdots \\ (\beta_{t,1}, \ldots, \beta_{t,n}, +) \\ (\beta_{1,1}, \ldots, \beta_{1,n}, -) \\ \vdots \\ (\beta_{t,1}, \ldots, \beta_{t,n}, -) \end{pmatrix}$

2. $\mathcal{H} = \begin{pmatrix} H_1 \\ \vdots \\ H_t \\ H_1 \times Q_{n+1} \\ \vdots \\ H_t \times Q_{n+1} \end{pmatrix}$

3. $\mathcal{A} = \mathcal{A} \otimes \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ *where* $\otimes$ *is the tensor product.*

4. $\mathcal{V} = \begin{pmatrix} v_1 \\ \vdots \\ v_t \\ d(P, H_{t+1}) \\ \vdots \\ d(P, H_{2t}) \end{pmatrix}$

5. $t = 2t$

6. $\mathcal{C} = \mathcal{A}^{-1} \times \mathcal{V}$

Note that the $c_i = 0$ of the TDT are useless. Here is an algorithm to remove them. To avoid this useless growth we execute the following algorithm until all the c_i are non zero.

Algorithm 3 TDT simplification

1. *Note i_0 the position of a c_i such that $c_{i_0} = 0$.*
2. *Remove in $\mathcal{B}$ and $\mathcal{C}$ the i_0th line.*
3. *Remove in $\mathcal{A}$ the i_0th column. Chose a line j_0 such that $\mathcal{A}$ without this line is invertible. Remove this line.*
4. *Remove the j_0th line in $\mathcal{H}$ and in $\mathcal{V}$.*

Thus to determine if $\#val(C(u)) = 0$ we just have to search $\vec{\sigma}$ in $\mathcal{B}$ of the simplified corresponding TDT.

The TDT allows to compute the whole set of signs of a list of a list of polynomial over the real roots of another one. It is exactly what we need for the dynamic real algebraic closure. Its incremental construction is also very well fitted. Moreover the complexities of these algorithm are very well understood ([17]). And it requires no other hypothesis on the basic field, on the contrary to the interval root encoding.

3.3 Example

We describe here very precisely an example of using the above algorithms. First we start with a TDT for $P(u) = u^3 - 3u - 1$ and with no Q_i. Thus the TDT is :

$$< u^3 - 3u - 1, [\,], ([\,]), (1), (1), (3), (3) >$$

It means that P has 3 real roots.

$Q_1(u) = u^-1$ is added, the TDT becomes :

$$< u^3 - 3u - 1, \left[u^2 - 1\right], \begin{pmatrix} [+] \\ [-] \end{pmatrix}, \begin{pmatrix} 1 \\ u^2 - 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ 1 \end{pmatrix} >$$

Thus, one can read on it that there are 2 real roots of P for which Q_1 is positive and 1 for which Q_1 is negative.

$Q_2(u) = u$ is added. The TDT becomes :

$$< u^3 - 3u - 1, \left[1, u^2 - 1, u\right], \begin{pmatrix} [+,+] \\ [+,-] \\ [-,-] \end{pmatrix}, \begin{pmatrix} 1 \\ u^2 - 1 \\ u \end{pmatrix},$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & -1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ 1 \\ -1 \end{pmatrix} >$$

It says that : there is one root of P such that $Q_1 > 0$ and $Q_2 > 0$, there is one other such that $Q_1 > 0$ and $Q_2 < 0$ and over the last one $Q_1 < 0$ and $Q_2 < 0$.

4 Dynamic Real Algebraic Closure (1)

Indeed I have completed two implementations of the dynamic real algebraic closure. The first one is presented in this section and the second one in the next section. The first version is a natural step to the second one. I did it to understand better the problems; trying to implement directly the second version would have failed. Thus, it is presented here for two reasons : first to present how I have thought this implementation, and secondly because I think it is a necessary step before presenting more complicated things. This version is actually not very inefficient. The second one is really better.

We recall that R is the real closure of the ordered field K. The dynamic set we consider here is $R < U >$ where U is a set of parameters. We suppose here that the parameters are algebraic, in the sense that they are solution of a polynomial equation. More precisely, we note $U = \{u_1, \ldots, u_n\}$, the set of parameters introduced in this order. This implies that :

$$P_1(u_1) = 0,\ P_1 \in K[X]$$

$$P_2(u_2) = 0,\ P_2 \in K[u_1][X]$$

$$\ldots$$

$$P_n(u_n) = 0,\ P_n \in K[u_1, \ldots, u_{n-1}][X]$$

4.1 Shape of a case

This proposition describes the structure of a case.

Proposition 3 *$C(U)$ has the following form :*

$$P_1(u_1) = 0 \wedge sign(\vec{Q}_1(u_1)) = \vec{\sigma}_1$$

$$\wedge P_2(u_2) = 0 \wedge sign(\vec{Q}_2(u_2)) = \vec{\sigma}_2$$

$$\wedge \ldots$$

$$\wedge P_n(u_n) = 0 \wedge sign(\vec{Q}_n(u_n)) = \vec{\sigma}_n$$

Where $P_i,\ Q_{i,j} \in K[u_1, \ldots, u_{i-1}][X]$

We define the **level** of a parameter u_i as i. A **law** for a parameter is the part of the case concerning especially this parameter. We call the law of level i the law of the parameter u_i. We note it L_i. A sequence laws is called a **tower of laws**.

More precisely we define a law as a triplet : a polynomial P, a list of polynomials $\vec{Q} = [Q_1, \ldots, Q_n]$ and a list of signs $\vec{\sigma} = [\sigma_1, \ldots, \sigma_n]$. Let's note it :

$$L = \ll P\ ,\ [Q_1, \ldots, Q_n]\ ,\ [\sigma_1, \ldots, \sigma_n] \gg$$

We define also the corresponding system :

$$S(L)(u) = (P(u) = 0) \wedge \bigwedge_{i=1}^{n} sign(Q_i(u)) = \sigma_i$$

We require also some properties of this object. $\mathcal{P}(L)$ represents the following set of properties for the law L :

1. P and each Q_i, for i from 1 to n, are monic and square free, their degree is strictly positive.
2. $degQ_i < deg(P)$, for all i from 1 to n.
3. Q_i is prime with P for all i.

This conditions just mean that the case is described by a kind of triangular system of the parameters. Note that we are not internal reasons to impose such conditions to the cases. We did it only to have a result more easy to read, to interpret and to use.

Proposition 4 *The emptiness of a case is decidable.*

Proof : The proof is an induction on the structure of the program. We present in 4.3 the detail of each operations. The proof relies on the use of the TDT.

4.2 Representation of the elements

With this special type of case we have special properties about the expressions of the elements.

Proposition 5
- *Each element of $R < U >$ can be expressed only with the $+$, $-$ and $*$.*
- *Moreover x an element of the closure can be reduced in the sense that it has an expression of the form :*
  - *$T(u_k)$ where T is a polynomial of degree at least one, its coefficients are expressed in a reduced form and depending only of $u_1, \ldots, u_{k-1}$. And the coefficients of degree superior or equal of the one of P_k are 0. k is called the* **level** *of x.*
  - *Or x can be expressed without parameters, its level is 0.*

Proof : The first part of the proposition is easy to proof. We need to show how to reduce the / operation. Let x a non zero element of $R < U >$. We make an induction on the number of / in the expression of x. If x has no / operation, it is trivial. If x has one / let call y its denominator. y has no /. We want to reduce $1/y$. To prove that we can reduce $1/y$ we make an induction on the level of y. y can be written has a polynomial in u_k, $T(u_k)$, where the coefficients are in $R < u_1, \ldots, u_{k-1} >$ and $deg(T) > 0$. We compute a Bezout's relation between the polynomials P_k and T on the ring $R < u_1, \ldots, u_{k-1} >$. It does not involve any split. We get two polynomials R and S such that $RP_k + ST = a$ where a belongs to $R < u_1, \ldots, u_{k-1} >$ and is non zero. Thus, by the hypothesis of the induction $1/a$ can be expressed without /. Thus we have $R_1P_k + S_1T = 1$ and $R_1(u_k)P_k(u_k) + S_1(u_k)T(u_k) = 1$. $S_1(u_k)$ is an expression of $1/y$ without /. This completes the first part.

Existence of the reduced form : we prove it by induction on the level of x. Let x an element of $R < U >$. We can express it with a polynomial expression $T(u_k)$ where the coefficients are already in reduced form, and $deg(T) > 0$. We perform the Euclidean division $T = SP_k + R$ (which involves no test). Then we reduce the coefficients of R. $R(u_k)$ is the required form.

Note that we can not ensure the uniqueness of the reduced form. For example, $P(u) = (u-1)(u+1)$, $Q(u) = u$, $C(u) = P(u) = 0 \wedge Q(u) > 0$. In this case u and 1 are two valid reduced forms. But the uniqueness is true if the expression are all reduced by the algorithm of the proof. That is to say, we ignore the semi algebraic part of the case and then we are in a pure algebraic case.

4.3 Computation in the real closure

We show here how to compute in the real closure so that we can ensure all the properties quoted above. Take great care while reading, think that this algorithms are dynamic. There are not really easy to explain to see more details the best thing is to have a look in the code. We did not want to put here all the code. This is just a summary to give an idea of how it works.

- $reduce \;:\; R < U > \longrightarrow R < U >$ is the operation already introduced in the last proof. It involves no split. Here is an algorithmic presentation to compute $reduce(x)$:
  - Let $T(u_k)$ the representative polynomial of x.
  - Compute the remainder T_1 of T with respect to P_k. As P_k is monic, it involves no division. Thus no split.
  - Apply *reduce* on each coefficient of T_1, to obtain T_2.
  - Return $T_2(u_k)$
- $\times, +, -$: we perform the operation with the polynomials expressions and then we reduce the result.
- / is done like in the proof above. It raises an error if the divisor is zero.
- $realRootOf \;:\; R < U > \longrightarrow R < U > \cup "failed"$. $u_{n+1} = realRootOf(P_{n+1})$, P_{n+1} is a polynomial which can be expressed in $K[u_1, \ldots, u_n][X]$. It introduces a new parameter. The result is failed in case the polynomial has no real root and returns a new parameter otherwise. This is the algorithm we use :
  - Replace P by its square free part.
  - Replace P by its monic part.
  - Initialise a new TDT for this new level $n+1$ with this new P.
  - In the case P has no real roots return "failed".
  - Create the law of level $n+1$:

    $$<< P, [\,], [\,] >>$$

    Add it to the case.
  - Return a new parameter u_{n+1}.

  Note that, this computation involves some splits concerning the levels $< n+1$.
- $=$. We need only to consider the test $x = 0$. x is supposed to be in its reduced form. Here is the algorithm to decide the equality test :

- If the level of x is zero then perform the test in K
- $x = T(u_k)$ and T is a polynomial in $K[u_1, \ldots, u_{k-1}][X]$.
- Replace T by its monic part. This operation can involve some splits.
- If T is constant then return $T(u_k) = 0$. That is the equality is decided in a lower level.
- T is now a monic polynomial of degree at least 1.
- Replace T by its square free part.
- Compute $P_{k,1}$ the gcd between P_k and T, and $P_{k,2}$ such that $P_k = P_{k,1}P_{k,2}$. These two polynomials are monic and square free.
- If $P_{k,1}$ has degree 0 then do not change the case and return false.
- If $P_{k,2}$ has degree 0 then do not change the case and return true.
- Rebuild a new TDT of level k with respect to each $P_{k,i}$: td1 and td2. This can involve some splits as well.
- Now create two new cases :
  1. replace in the old case the law of level k by a new law of level k similar to the previous one except that we replace P_k by $P_{k,1}$ and take the remainder of the $Q_{k,i}$ with respect to $P_{k,1}$.
  2. Similar construction as just above but with $P_{k,2}$.
- The TDT can answer to the emptiness of these case. If one of them is empty them keep the old case and return true is case case 2 is empty and false otherwise. Else return true in the first case and false in the second. (This creates an effective split)

Note that $\mathcal{P}(L)$ is conserved for all the laws of the cases. And that we have decided the emptiness of the cases.

- $<$. We just need to consider the case $x = 0$.
  - If $x = 0$ then return false.
  - If the level of x is zero then perform the test in K
  - $x = T(u_k)$ and T is a polynomial in $K[u_1, \ldots, u_{k-1}][X]$.
  - Evaluate the sign of the leading coefficient of T : $\sigma \in \{+, -\}$. It involves some splits.
  - Replace T by its remainder with respect to P_k.
  - Replace T by its monic part. This operation can involve some splits.
  - If T is constant then return $T(u_k) < 0$. That is the equality is decided in a lower level.
  - T is now a monic polynomial of degree at least 1.
  - Remove in T the factor of even degree and the replace T by its square free part. This is performed by a square free decomposition.
  - Add T in the TDT of level k.
  - Read in this TDT the possible answers for the sign of T with respect to the current case.
  - Make a split if necessary and return the sign of T times σ.

  Note that the conditions of the laws of the case are conserved and that the current case is never empty.

5 Dynamic Real Closure (2)

5.1 Motivations

We show first a result of the program of the previous section :

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

To understand this code the reader must refer to the next section. The result is :

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

This is not surprising at all since we have mentioned that the determination of the number of roots of a polynomial involves splits occurring during the Sturm queries. We have two reasons to avoid this phenomenon. First the results becomes too huge and thus are more difficult to interpret. The second reason relies on the fact that all the constructional operations behave the same for two cases having the same algebraic part (differing only by the $<$ conditions). When a split occurs because of a $<$ test the two branches go on with exactly the same calculi. So if we reduce the number of split caused by $<$ we decrease sharply the execution time. This is the aim of this section. The original idea can be found in [11]. In this article no precise implementation is given and no code has ever been completely achieved.

The result we get with this version is :

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

And the computation times in nearly 50 times faster. The result is more natural.

5.2 Shape of a case

Let's now describe the structure of the new cases. $u_1, \ldots, u_n$ are the parameters already introduced in this order. K is the ground field, R its real closure. The parameters u_i are still algebraic numbers is R.

$$P_1(u_1) = 0, \ P_1 \in K[X]$$

$$P_2(u_2) = 0, \ P_2 \in K[u_1][X]$$

$$\ldots$$

$$P_n(u_n) = 0, \ P_n \in K[u_1, \ldots, u_{n-1}][X]$$

We still call the law of level i an object of type :

$$L_i = \ll P_i \ , \ \vec{Q_i} \ , \ \vec{\sigma_i} \gg$$

$$\vec{\sigma_i} = [\sigma_{i,1}, \ldots, \sigma_{i,m_i}]$$

$$\vec{Q_i} = [Q_{i,1}, \ldots, Q_{i,m_i}]$$

Where $Q_{i,j}, \ P_i \in K[u_1, \ldots, u_{i-1}][X]$. The difference is that we allow $\sigma_{i,j}$ to be either $+$, or $-$ or ?. $sign(Q) =?$ means $(sign(Q) = +) \vee (sign(Q) = -)$. This difference will allow us to avoid many splits.

The case is still a sequence of laws as in the previous section. But now we would like to make a sign test without split, at least we ensure that there are no split at the level concerned by the split. To achieve this we introduce a tree where a node of level i is a law of level i, without ? in its sign vector. This tree T represents a first order logical formula denoted by $S(T)$. and defined by induction : T has $T_1, \ldots, T_k$ as subtree, and L as root :

$$S(T) = S(L) \wedge (S(T_1) \vee \ldots \vee S(T_k))$$

In fact each level of this tree represent a new conjunction and the nodes of a same level a disjunction. For example the tree figure 2 the logic formula it represents is $S(L_0) \wedge ((S(L_1^1) \wedge (S(L_2^1) \vee S(L_2^2))) \vee (S(L_1^2)) \wedge (S(L_2^3) \vee S(L_2^4)))$

Before going further we need to precise some vocabulary.

Definition 10 Tree terminology
$root(T)$ is the root of T.
$father(L)$ is the father of L in the tree.
$children(L)$ is the set of children of L in the tree.
$subtree(L)$ denotes the subtree of which L is the root.
$path(L)$ is the path from the root of the tree to L.
$branch(L)$ is the set of $path(C)$ where C is a leaf of $subtree(L)$.

For example if T is the one of the figure 2 we have : $root(T) = L_0$
$father(L_2^2) = L_1^1$
$children(L_1^2) = \{L_2^3, L_2^4\}$
$subtree(L_1^1) = L_1^1(L_2^1, L_2^2)$

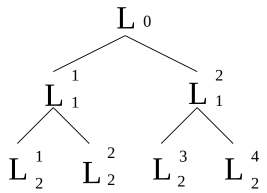


Figure 2: *Tree of the case*

$path(L_2^2) = [L_0, L_1^1, L_2^2]$
$branch(L_0) = \{[L_0, L_1^1, L_2^1], [L_0, L_1^1, L_2^2], [L_0, L_1^2, L_2^3], [L_0, L_1^2, L_2^4]\}$

Figure 2 shows such a tree. For each node L_i^j we need to note :

$$\vec{\sigma_i^j} = [\sigma_{i,1}^j, \ldots, \sigma_{i,n_i}^j]$$

$$L_i^j = \ll P_i \ , \ \vec{Q_i} \ , \ \vec{\sigma_i^j} \gg$$

Definition 11 *If p is any path in the tree, $p = [L_1, \ldots, L_k]$, we define $S(p)$ as $\wedge_{i=1}^k S(L_i)$.*

We say that a list of signs σ_1 **matches** an other list of signs σ_1 if they have the same length and if $\sigma_{1,i} \leq \sigma_{2,i}$, for all i. Where $<$ denotes the partial order over $\{+, -, ?\}$ defined by $+ <?$ and $- <?$. For example $[+, -, +]$ and $[+, -, -]$ match $[+, ?, ?]$.

Properties of this tree. Let $\mathcal{P}(T)$ the following set of properties :

1. The laws of the same level differ only by their list of signs.
2. For all $b \in branch(root(T))$, $val(S(b))$ is not empty.
3. For all i, for each node L_i^j of level i the children are all the possible laws such that (1) $\sigma_{i,j}$ matches $\vec{\sigma_i}$ and (2) the previous conditions must be satisfied.

This proposition is the link between the version of the closure we have described in the previous section and this version : it says that the logical formula represented by a tree is the disjunctions of the logical formula of its branches. Thus it represents several cases(of the previous implementation) in one in the new implementation.

Proposition 6

$$S(T) \Longleftrightarrow \bigvee_{b \in branch(root(T))} S(b)$$

Proof : Induction on the depth of the tree : for depth one it result is obvious. Let now L_0 be the root of T and $T_1, \ldots, T_K$ then children subtrees. $S(T) = S(L_0) \wedge \bigvee_{i=1}^k S(T_i)$. We apply the proposition to each T_i, and we get the expected result on T.

The next proposition says that the logical formula represented by the tree is equivalent to the formula represented by the tower of laws. Thus we have two representation of the case : one expanded and one more compact. In other words we are able to represent several cases of the previous implementation in one.

Proposition 7 *If the tree of root L_0 verifies $\mathcal{P}(T)$ then the following holds :*

$$S(T) \Longleftrightarrow S([L_1, \ldots, L_n])$$

Proof : First we complete T with all the empty branches which are necessary to ensure the property : at each node the children are all the different laws matching σ_i where i is the level of the considered node without more hypothesis. This new tree is strictly equivalent to the old one. Now we make an induction on n, the depth of a tree verifying this new property .If the tree has only one level, $S(T) = S(L_0)$. . Hence the proposition is obvious. Let now suppose that we have the proposition for the level $n-1$. We consider a tree of depth n. If we truncate the tree T of its bottom level, we obtain a tree T_1 verifying the new property. And so the proposition, by the hypothesis of the induction.

$$S(T_1) \Longleftrightarrow \bigvee_{b \in branch(root(T_1))} S(b)$$

Using the previous proposition. We have also :

$$S(T) \Longleftrightarrow \bigvee_j (S(path(L^j_{n-1})) \wedge \bigvee_{L^i_n \in children(L^j_{n-1})} S(L^i_n))$$

Or

$$\bigvee_{L^i_n \in children(L^j_{n-1})} S(L^i_n) \Longleftrightarrow S(L_n)$$

for all j, since the tree has been completed.

$$S(T) \Longleftrightarrow S(T_1) \wedge S(L_n)$$

Thus

$$S(T) \Longleftrightarrow S(L_1) \wedge \ldots \wedge S(L_{n-1}) \wedge S(L_n)$$

This completes the proof.

5.3 InternalInequalityTest function

We now going to explain the use of this tree. When performing a sign test, we do not want the case to split at the level concerned by the test. So we define a new operation of the dynamic real field. It takes an element of the field, adds the representative polynomial of it into the tower, add eventually new branches in tree and returns informations useful to compute the sign of it in each branch. DropSign function takes as argument a result given by the previous function and returns its sign in each branch of the tree. A sign question has now several answer, depending of the branch of the tree. Here is more precisely the inequalityInequalityTest function :

$$inequalityInequalityTest \; : \; R < U > \longrightarrow SignMemorisation$$

SignMemorisation is a structure which codes the position of the monic part of a representative polynomial in the tree and the sign of the leading coefficient. The new polynomial introduced in a law has no sign, so its sign is ? in the list of the signs. And the tree changes in order to verify the required conditions.

$$dropSign \; : \; SignMemorisation \times N \longrightarrow \{+, -\}^*$$

dropSign(sm, i) returns the list of the signs of sm in each branch of the tree, at the level i.

5.4 Review of the real tools

Each branch of the tree correspond to a case of the previous implementation. Thus we would need one TDT per branches per level. However, as the laws of a same level differ only by their list of sign one can think to avoid to much redundancy. So we regroup in one all the TDT of the same level. So that we can still speak about the TDT corresponding to the law of level i. It is quite the same as previously except for $\mathcal{C}$ and $\mathcal{V}$. We need to duplicate as many as the number of the nodes of level $i-1$.

$$\mathcal{C} = \begin{pmatrix} c_{1,1} & \dots & c_{r,1} \\ \vdots & & \vdots \\ c_{1,t} & \dots & c_{r,t} \end{pmatrix} \quad \mathcal{V} = \begin{pmatrix} v_{1,1} & \dots & v_{r,1} \\ \vdots & & \vdots \\ v_{1,t} & \dots & v_{r,t} \end{pmatrix}$$

Let's note $[b_1, \dots, b_r]$ the sequence of $S(path(L))$ when L describes the nodes of level $i-1$. The interpretation of this extended TDT is :

1. $\#val(P_i = 0 \wedge sign(\vec{Q}_i) = \beta_j) = c_{i,j}$ in the case the set of parameters U verifies $S(p_i)$.
2. $v_{i,j} = d(P, H_j)$ in case the set of parameters U verifies $S(p_i)$.
3. $\mathcal{V} = \mathcal{A} \times \mathcal{C}$ and $\mathcal{A}$ is an invertible matrix.

So each column corresponds to a branch in the tree. And the interpretation of the value in one column is done with respect to the branch it is related to.

Then the algorithms are mostly the same except some little differences due to the fact we compute on different cases in one.

Algorithm 4 *The algorithm 2 must be modified a bit. The step are all the same except the computation of $\mathcal{V}$. Just the step 4 becomes.*

1. *Compute all the $d(P, H_{t+i})$ for all t from 1 to t. But use only internalInequalityTest instead of $<$ and freeze the computation until all are done.*
2. *Ask the sign of all the frozen expressions of the Sturm queries and create as many columns as the number of nodes of the considered level.*

The simplification phase (algorithm 3) removes the lines of $\mathcal{C}$ when all the values are 0 but its works exactly the same.

Let see a complete example of the all structure of a case with the TDT. This example shows the structures explained so far. It is a rough result to display the case, it represent its internal structure. We have introduced two parameters u and v. We note $P_1(u) = u^3 - 3u - 1$, $P_2(v) = v^3 - (u+1)v^2 + (u^2-1)v - u^2 - 2u$, $Q_{1,1}(u) = u^2 + \frac{9}{5}u + \frac{1}{2}$, $Q_{1,2}(u) = y^2 - u - 2$. Here is the laws :

$$L_1 =<< u^3 - 3\ u - 1, \left[u^2 + \frac{9}{5}u + \frac{1}{2}, u^2 - u - 2\right] [?, ?] >>$$

$$L_2 =<< v^3 + (-u-1)\ v^2 + \left(u^2 - 1\right)\ v - u^2 - 2\ u, [\,], [\,] >>$$

Here is the tree :

$$[level = 1,\ position = 1,\ slist = [],\ sq = +]$$

$$[level = 2,\ position = 3,\ slist = [+,\ +],\ sq = +]$$

$$[level = 2,\ position = 2,\ slist = [+,\ -],\ sq = +]$$

$$[level = 2,\ position = 1,\ slist = [-,\ -],\ sq = +]$$

The tree has two levels. The field *position* corresponds to the column is the TDT of the concerned level. Note that we only need to store the list of sign ($slist$) for each node.

And these are the TDT. Note that we do not need to store either the polynomial P_i either the polynomials $Q_{i,j}$, which are in the laws.

For v :

$$[[]][1][1]\begin{bmatrix} 3 & 1 & 1 \end{bmatrix}\begin{bmatrix} 3 & 1 & 1 \end{bmatrix}$$

For u :

$$\begin{bmatrix} [+,\ +] \\ [+,\ -] \\ [-,\ -] \end{bmatrix}\begin{bmatrix} 1 \\ u^2 + \frac{9}{5}u + \frac{1}{2} \\ u^2 - u - 2 \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\begin{bmatrix} 3 \\ 1 \\ -1 \end{bmatrix}$$

So one can read on it that :

- There is one real root u of P_1 satisfying $Q_{1,1} > 0$ and $Q_{1,2} > 0$, in this case P_2 has 1 real root.
- There is one real root u of P_1 satisfying $Q_{1,1} > 0$ and $Q_{1,2} < 0$, in this case P_2 has 1 real root.
- There is one real root u of P_1 satisfying $Q_{1,1} < 0$ and $Q_{1,2} < 0$, in this case P_2 has 3 real roots.

5.5 Computations

We describe the algorithm to compute in this dynamic field, so that we have this proposition. Note that our aim is not to rewrite the whole algorithm in precise details. The interested reader will refer to the code and all the comments spread over it. We just want to give the general idea of how its works.

Proposition 8 *The properties of $\mathcal{P}(T)$ are preserved at each step of the computation in the field.*

Proof : The proof is done by induction on the structure of the program. If the field has no parameters the proposition is obvious.

1. $reduce, +, -, *, /$ are exactly the same as in the previous section.
2. $u := realRootOf(P(x))$ is like as in the previous section, except that we use the new TDT algorithms. After a new parameter is introduced, the part of the branches of the tree which lead to cases where P has no real roots are removed from the tree.

3. $internalInequalityTest(x)$ is the same as the $<$ in the previous section except that we keep in one case the different cases thanks to the structure of the tree. Note that we suppose x to be non zero.
4. $x < 0$.
   - If $x = 0$ then return false.
   - Call $sm = internalInequalityTest(x)$. Then it is easy to check if two answer are possible, if not return the only case. If there is a split to do one just have to cut the tree in two trees and specify into the law the sign imposed by the answer.
5. $=$ is the same as previously. But we must update the tree after the reconstruction of the TDT, cutting the part of branches which are empty.

The reader will refer to the appendix to see a full session of the dynamic real algebraic closure with a full internal objects printing.

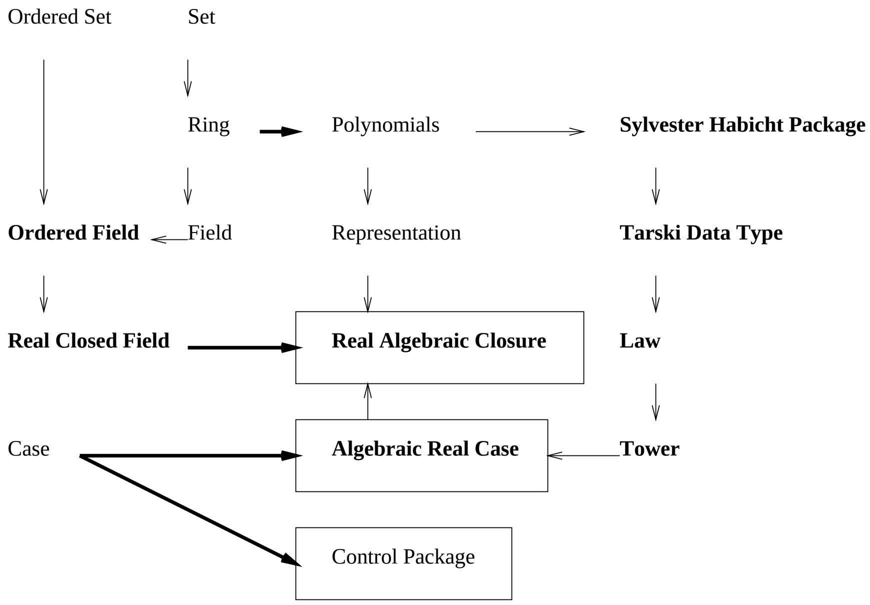


Figure 3: *Overview of the program*

6 Implementation in Axiom

For Axiom's terminology the reader can refer to [23]. First we describe the structure of the code. More details can be found consulting the code through the Axiom's hyperdoc. Then we show the using with some various examples. And we make some comparisons with the real closure (non dynamic) written by R.Rioboo [21].

6.1 Description of the source

The code of the dynamic real closure is very closed to the one of the Constructible Algebraic Closure ([3], [2]). The main structure has not been changed. The figure 3 shows a summary overview of the complete program. The thick arrows represent the hierarchy between categories and domain, the other arrows show just the logical dependency between the objects. The domain in frame are the only one to be known by the user. Here is a small description of them. And the objects in bold are what I have specially written, the rest comes straight from *D.Duval* and T.Gómez Díaz (cf [3]).

Different implementations has been already discussed in [19]. The main question is : how to implement the splits ? The natural tools seems to be a $fork()$ like in C language. This 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 [3]

We present now the implementation of the dynamic real closure program, in Axiom. Here is the main interesting operations of the dynamic real closed field category.

DynamicRealClosedFieldCategory: Category == Exports <u>where</u>

Pol ==> DynamicUnivariatePolynomial($)

```
Exports == DynamicOrderedFieldCategory with

  -- no splitting

  indeterminate: () -> Pol
    ++ indeterminate() returns an the indeterminate of type
    ++ \spadtype{DynamicUnivariatePolynomial(%)}.
    ++ it must be only used to introduce new real algebraic paramteters
    ++ with the function \spadfun{realRootOf}.

  -- internal&external splittings

  realRootOf: (Symbol,Pol) -> Union(parameter:$,failed:"failed")
    ++ realRootOf(sy,p) declares sy as a new parameter being real root of
    ++ the polynomial p. returns failed iff there is no root in the current
    ++ case
  mustBeZero: $ -> Boolean
    ++ mustBeZero(x) returns true in case x=0 can have been added to the
    ++ constraint. Otherwise false and the constraint is unchanged.
  mustBePositive: $ -> Boolean
    ++ mustBePositive(x) returns true in case x>0 can have been added to
    ++ the constraint. Otherwise false and the constraint is unchanged.
  mustBeNegative: $ -> Boolean
    ++ mustBeNegative(x) returns true in case x<0 can have been added to
    ++ the constraint. Otherwise false and the constraint is unchanged.
```

This code shows the most important operations of the dynamic real closure category.

```
DynamicRealClosureCategory(Ko): Category == Exports where

 Ko: Field

 Law ==> DynamicRealLaw($)

 Exports == Join(DynamicRealClosedFieldCategory, Algebra(Ko)) with

  -- no reduction, no splitting

  constructibleKernel: PositiveInteger -> $
    ++ constructibleKernel(pi) returns the constructible kernel of
    ++ level pi in $
  law: PositiveInteger -> Law
    ++ law(pi) returns the current law of level pi
  level: $ -> NonNegativeInteger
    ++ level(x) returns the level of x
```

Here is a brief description of the dynamic real case category.

```
DynamicCaseCategory(Ko:Field): Category == DynamicSetCategory with

  basic: () -> $
    ++ basic() returns the basic case
  current: () -> $
    ++ current returns the current case
  next: () -> List $
    ++ next returns the list of the next cases
  setCurrent : $ -> Void
    ++ setCurrent(cas) sets the current case to cas
    ++ and does not change the list of next cases
  setNext : List $ -> Void
    ++ setNext(lcas) sets the list of next cases to lcas
```

```
  ++ and does not change the current case
 refreshCurrent : () -> Void
  ++ refreshCurrent() sets the current case to the basic case
 refreshNext : () -> Void
  ++ refreshNext() sets the list of next cases to the empty list
 refresh : () -> Void
  ++ refresh() sets the current case to the basic case
  ++ and the list of next cases to the empty list
 addNext : $ -> Void
  ++ addNext(cas) adds the case cas to the list of next cases
  ++ If the case cas is a tower, its level is reset to 0.
```

Here is the Axiom domain for the clauses.

```
DynamicRealClause(Ko,D): Exports == Implementation where

 Ko: Field
 D: SetCategory

 Dfk ==> DynamicBuildOrderedField(Ko)
 Ick ==> InternalDynamicRealClosure(Dfk)
 Law ==> DynamicRealLaw(Ick)
 Cas ==> DynamicRealTower(Ick,Law)

 Exports == DynamicSetCategory with

  valueIs: $ -> D
   ++ valueIs(x) returns the "result" part of a clause
  inCase: $ -> Cas
   ++ inCase(x) returns the "case" part of a clause
```

And the control package in the following :

```
DynamicRealControlPackage(Ko,D): Exports == Implementation where

 Ko: Field
 D: SetCategory

 Ccl ==> DynamicRealClause(Ko,D)

 Exports == with

  allCases: (()->D) -> List Ccl
   ++ allCases(f) returns the value of f() in every case
   ++ together with a description of the corresponding case
```

6.2 User guide and examples

We describe here, through some simple examples[6] the way of computing in the dynamic real algebraic closure (second version). Between two calls of the *allCases* functions do not forget to call $refresh()$ function to refresh the old case of the dynamic set (indeed the mutable variables).

1. `K` is a real field.
2. `I` is the dynamic real algebraic closure of K
3. `CP` is the package for computing all branches

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

4. C is the domain dealing with the cases of the dynamic closure.

6.2.1 Example 1 : Introduction of parameters

We introduce two parameters u and v such that $u^2 - 1 = 0$ and $v^2 - u = 0$. Then ask the sign of $u^2 - 4v$.

```
K:=FRAC INT
I:=DynamicRealClosure(K)
CP:=DynamicRealControlPackage(K,INT)
C:=DynamicRealCase(K)

example1(): INT ==
  x:=indeterminate()$I
  uf:=realRootOf('u,x^2-1)
  uf case failed => 3
  u:=uf.parameter
  vf:=realRootOf('v,x^2-u)
  vf case failed => 3
  v:=vf.parameter
  if u^2-4*v>0 then 1 else 2

refresh()$C
allCases(example1)$CP

                       2               1          2
   [value is 1 in case v  - u = 0 , v - - < 0 and u  - 1 = 0,
                                       4
                       2               1          2
    value is 2 in case v  - u = 0 , v - - > 0 and u  - 1 = 0]
                                       4
                      Type: List DynamicRealClause(Fraction Integer,Integer)
              Time: 0.62 (IN) + 0.51 (EV) + 0.33 (OT) + 1.73 (GC) = 3.19 sec
```

6.2.2 Example 2 : Splits caused by normalisation of polynomials

We can use the program to know if a triangular polynomial system has real solution or not. The system is :

$$\left\{\begin{array}{l} (u^3 - 3*u - 1)*(u^2 - 1) = 0 \\ (v^3 - (u+1)*v^2 + (u^2-1)*v - u^3 - u^2 + u + 1)*(v^2 - u^2) = 0 \\ (w^2 + v^2 + u^2 + u - 1)*u*v = 0 \end{array}\right.$$

```
 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          2
       v  + (- u - 1)v  - v  + (u + 1)v  + (- 2u  - u)v + 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
```

Note the shape of the answer, the cases contain only monic polynomials.

6.2.3 Example 3 : Signature

Signature of the matrix M. $M = \begin{pmatrix} u & 1 \\ 1 & u \end{pmatrix}$ Where u verifies $3u^5 - 3u^4 - 2u^3 + 3u^2 - 2u + 1 = 0$.

```
signature() : INT ==
  x:=indeterminate()$I
  uf:=realRootOf('u,3*x^5-3*x^4-2*x^3+3*x^2-2*x+1)
  u:=uf.parameter
  sylvesterHabichtQuery(x^2-2*x+1-u^2,x)


allCases(signature)$CP

   [value is 1 in case u - 1 = 0,
                      4   2  2   1     1       2
    value is 0 in case u   - - u  + - u - - = 0 , u  + u - 1 < 0,
                           3      3     3
                      4   2  2   1     1       2
    value is 2 in case u   - - u  + - u - - = 0 , u  + u - 1 > 0]
                           3      3     3
                   Type: List DynamicRealClause(Fraction Integer,Integer)
            Time: 0.13 (IN) + 0.71 (EV) + 0.25 (OT) + 0.82 (GC) = 1.91 sec
```

6.2.4 Advanced Example 1 : using the limit function

We have written another category adding some more properties to the real closure in order to use the Axiom's domain *Expression* [7] on it. It does not actually work with the current libraries. But I hope the modifications will be done soon. For the moment these tests have been performed just with a very light modification of the file sign.spad of the Axiom's library.

We are coming back to the problem :

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

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)
```

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

```
  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,
                                2
    value is - infinity in case v  - 2 = 0 , v < 0 and u = 0,
                                 2
    value is  + infinity in case v  - 2 = 0 , v > 0 and u = 0,
                                          2
    value is - infinity in case v = 0 and u  - 2 = 0 , u < 0,
                                           2
    value is  + infinity in case v = 0 and u  - 2 = 0 , u > 0,
                                2              2
    value is - infinity in case v  - 2 = 0 and u  - 2 = 0 , u < 0,
                                 2              2
    value is  + infinity in case 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 quite a natural one.

6.2.5 Advanced Example 2 : Integration

With the same precautions as previously. We are coming back also on :

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

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

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

```
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 expect and that the time is not so bad.

6.3 Other implementations

The first real closure has been done by R.Rioboo ([24], [22] and [21]) in Axiom. It uses intervals as root coding and is not dynamic at all. We have done some comparison of times.

6.3.1 Using a usual real closure to simulate all the cases

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

With the real closure of R.Rioboo :

```
Ran := RECLOS(FRAC INT)
l: List List Ran:= []
for u in allRootsOf(x^3-3*x-1)$Ran repeat
  for v in allRootsOf(x^3-(u+1)*x^2+(u^2-1)*x-u^3-u^2+u+1)$Ran repeat
    for w in allRootsOf(x^2+v^2+u^2+u-1)$Ran repeat
      l:=cons([u,v,w],l)

                                                                    Type: Void
           Time: 3.57 (IN) + 20.12 (EV) + 1.09 (OT) + 10.95 (GC) = 35.73 sec
                                   Type: List RealClosure Fraction Integer
l
   [[%R27,%R32,%R38], [%R27,%R32,%R37], [%R27,%R31,%R36], [%R27,%R31,%R35],
    [%R27,%R30,%R34], [%R27,%R30,%R33]]
```

That is we are obliged to isolate all the root of all the polynomials over each root of its lower polynomial. Not that the result is not very beautiful.

The same program with the dynamic real closure is :

```
rKo:=FRAC INT
I:=DynamicRealClosure(rKo)
CP:=DynamicRealControlPackage(rKo,INT)
C:=DynamicRealCase(rKo)

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

   [
                         2    2    2
     value is 0 in case w  + v  + u  + u - 1 = 0 and
        3             2     2           2               3
       v  + (- u - 1)v  + (u  - 1)v - u  - 2u = 0 and u  - 3u - 1 = 0
     ]
                     Type: List DynamicRealClause(Fraction Integer,Integer)
             Time: 0.35 (IN) + 9.75 (EV) + 1.84 (OT) + 1.65 (GC) = 13.59 sec
```

The dynamic real closure answers that the system has solutions, without giving more informations about it. It alows to work with it without any distinction. While the usual real closure comptutes a precise caracterisation of all solutions. On such an example the dynamic real closure is more efficient. We can think about it as a lazy evaluation system.

6.3.2 Using the dynamic closure to compute only one case

The folowing problem is to proove the equality :

$$\sqrt{\sqrt{10}+3}\sqrt{\sqrt{5}+2}-\sqrt{\sqrt{10}-3}\sqrt{\sqrt{5}-2}=\sqrt{10\sqrt{2}+10}$$

With the real closure of R.Rioboo :

```
Ran := RECLOS(FRAC INT)

a := sqrt(2)$Ran
b := sqrt(5)$Ran
c := sqrt(10)$Ran
d := sqrt(c+3)$Ran
e := sqrt(b+2)$Ran
f := sqrt(c-3)$Ran
g := sqrt(b-2)$Ran
h := sqrt(10*a+10)$Ran

d*e-f*g = h
%::Boolean

            +---------+ +--------+    +---------+ +--------+   +-----------+
            | +--+      | +-+         | +--+      | +-+        |   +-+
   (29)  - \|\|10  - 3 \|\|5  - 2  + \|\|10  + 3 \|\|5  + 2 = \|10\|2  + 10
                                  Type: Equation RealClosure Fraction Integer
                           Time: 0.27 (IN) + 0.02 (EV) + 0.22 (OT) = 0.51 sec
%::Boolean


   (30)  true
                                                                Type: Boolean
                         Time: 14.74 (IN) + 0.01 (OT) + 6.56 (GC) = 21.31 sec
```

Then with the dynamic real closure :

```
rKo:=FRAC INT
I:=DynamicRealClosure(rKo)
CP:=DynamicRealControlPackage(rKo,INT)
C:=DynamicRealCase(rKo)

refresh()$C

sqrtTest(): INT ==

  x:=indeterminate()$I

  af:=realRootOf('a,x^2-2)
  af case failed => 0
  a:=af.parameter

  bf:=realRootOf('b,x^2-5)
  bf case failed => 0
  b:=bf.parameter

  cf:=realRootOf('c,x^2-10)
  cf case failed => 0
  c:=cf.parameter

  df:=realRootOf('d,x^2-(c+3))
  df case failed => 0
  d:=df.parameter
```

```
  ef:=realRootOf('e,x^2-(b+2))
  ef case failed => 0
  e:=ef.parameter

  ff:=realRootOf('f,x^2-(c-3))
  ff case failed => 0
  f:=ff.parameter

  gf:=realRootOf('g,x^2-(b-2))
  gf case failed => 0
  g:=gf.parameter

  hf:=realRootOf('h,x^2-(10*a+10))
  hf case failed => 0
  h:=hf.parameter

  mustBePositive(d)
  mustBePositive(e)
  mustBePositive(f)
  mustBePositive(g)
  mustBePositive(h)

  h=d*e-f*g => 1
  0

allCases(sqrtTest)$CP

   [
     value is 1 in case h + ((- 2a - 3)b + 5a + 5)d e = 0 and
                                 2                           2
       g + (- b + 2)d e f = 0 and f  - c + 3 = 0 , f > 0 and e  - b - 2 = 0 ,
                  2                                         2
       e > 0 and d  - c - 3 = 0 , d > 0 and c - a b = 0 and b  - 5 = 0 and
        2
       a  - 2 = 0
     ]
                       Type: List DynamicRealClause(Fraction Integer,Integer)
         Time: 0.54 (IN) + 247.61 (EV) + 50.71 (OT) + 71.80 (GC) = 370.66 sec
```

The dynamic closure is much slower. But their are further information available.

The dynamic real closure is well fitted for working on real roots parameters without any distinction. The TDT is well suited to do that since it computes all the cases.

A Axiom session : Dynamic Real Algebraic Closure (2) with internal representation printing

Here is the submitted program. The result is print the whole internal objects. For a description of them the reader must consult the comments about this object through the hyperdoc and the sources.

```
rKo:=FRAC INT
I:=DynamicRealClosure(rKo)
CP:=DynamicRealControlPackage(rKo,INT)
C:=DynamicRealCase(rKo)
UIf:=Union(parameter:I,failed:"failed")

test() : 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)*u*v)
  wf case failed => 3
  0

allCases(test)$CP
```

Here is the result, with printing the internal object roughly.

```
[
value is 3 in case
  [level= 1,position= 1,slist= [],sq= "+"]
     [level= 2,position= 1,slist= [],sq= "+"]
  v [] [[]] [1] [1] [1] [1] and u + 1 [] [[]] [1] [1] [1] [1]
,

 value is 0 in case
  [level= 1,position= 1,slist= [],sq= "+"]
     [level= 2,position= 1,slist= [],sq= "+"]
        [level= 3,position= 1,slist= ["+"],sq= "+"]
   2                               2
  w  - v [] [[]] [1] [1] [2] [2] and v  - 1 v ["?"] [["+"],["-"]] [1,?]
  +1   1 + +1+ +2+
  |      | | | | | and u + 1 [] [[]] [1] [1] [1] [1]
  +1  - 1+ +1+ +0+
,

value is 3 in case
  [level= 1,position= 1,slist= [],sq= "+"]
     [level= 2,position= 1,slist= [],sq= "+"]
  v [] [[]] [1] [1] [1] [1] and u - 1 [] [[]] [1] [1] [1] [1]
,

value is 0 in case
  [level= 1,position= 1,slist= [],sq= "+"]
     [level= 2,position= 1,slist= [],sq= "+"]
        [level= 3,position= 1,slist= ["+"],sq= "+"]
   2                               3     2
  w  - v [] [[]] [1] [1] [2] [2] and v  - 2v  - v + 2 v ["?"]
                      +1   1 + +2+ +3+
```

```
  [["+"],["-"]] [1,?] |      | | | | | and u - 1 [] [[]] [1] [1] [1] [1]
                      +1   - 1+ +1+ +1+
,

value is 0 in case
  [level= 1,position= 1,slist= [],sq= "+"]
     [level= 2, position= 3, slist= ["+","+","+","+","+","+","+","+","+"],
      sq= "+"]
        [level= 3,position= 3,slist= ["+"],sq= "+"]
 ,
     [level= 2, position= 2, slist= ["+","+","+","+","-","+","-","+","-"],
      sq= "+"]
        [level= 3,position= 1,slist= ["+"],sq= "+"]
 ,
     [level= 2, position= 1, slist= ["-","-","+","+","+","+","+","+","+"],
      sq= "+"]
        [level= 3,position= 2,slist= ["+"],sq= "+"]
   2
  w  - v [] [[]] [1] [1] [2  2  2] [2  2  2] and
   5            4    3            2         2           2
  v  + (- u - 1)v  - v  + (u + 1)v  + (- 2u  - u)v + 3u  + 7u + 2 v ["?"]
                      +1   1 + +3  1  2+ +5   3   3+      3
  [["+"],["-"]] [1,?] |      | |       | |         | and u  - 3u - 1
                      +1   - 1+ +2  2  1+ +1   - 1  1+
   2   661264     187186  2   16574     4653  2   114     32  2        7
  u  + ------ u + ------ u  + ----- u + ---- u  + --- u + -- u  + 2u + -
      352101     352101      9145      9145      77      77            2
   2   17034630     4822871  2   86110     24331  2   5     3  2
  u  + -------- u + ------- u  + ----- u + ----- u  + - u + - u  + 2u + 2
      9063517      9063517      46219     46219      2     4
  u + 1 ["?","?","?","?","?","?","?","?","?"]
  [["+","+","+","+","+","+","+","+","+"],
   ["-","-","+","+","+","+","+","+","+"],
   ["+","+","+","+","-","+","-","+","-"]]
                                                            +1   1    1 + +1+
     2   661264     187186  2   17034630     4822871  |           | | |
  [1,?  + ------ ? + ------,?  + -------- ? + -------] |1  - 1   1 | |1|
         352101     352101      9063517      9063517  |           | | |
                                                            +1   1   - 1+ +1+
  +3+
  | |
  |1|
  | |
  +1+
]
                Type: List DynamicRealClause(Fraction Integer,Integer)
        Time: 0.31 (IN) + 7.80 (EV) + 1.03 (OT) + 3.55 (GC) = 12.69 sec
```

Conclusion :

We have presented here how the dynamic evaluation method can be used to deal with the real closure of an ordered field. The implementation in Axiom is now achieved, and can be used in a natural way. A general theoretical description of the dynamic evaluation can be found in [9] and [10]. One can expect now :

1. An upgrading of all the dynamic closures in AxiomXL [8]. This would lead to more efficient program. The split could be performed with a $fork()$ and launched eventually on different machines in parallel. The $fork()$ function has already been implemented and a dynamic constructible closure has already been done in AxiomXL (see [19]). The main problem is that AxiomXL is still under development and not wide-spread.

2. An implementation of the dynamic semi-algebraic closure. This could be extended to an efficient quantifier elimination program.

3. To extend the actual computer algebra systems so that they could be able to deal with dynamic evaluation internally.

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.

[8] AxiomXL is a trademark of NAG

References

[1] Thomas Sturm Andreas Dolzman. Simplification of quantifier-free formulas over ordered fields. Technical report, Fakultät für Mathematik und Informatik, Universität Passau, Germany, 1995.

[2] Teresa Gómez-Díaz. Examples of using dynamic constructible closure. In *IMACS Symposium SC-1993*, 1993.

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

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

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

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

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

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

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

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

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

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

[13] R.D.Nelson J.Daintith. *Dictionary of Mathematics*. Penguin, 1989.

[14] C.Faure J.Davenport. The unknown in computer algebra. Technical report, School of Mathematical Sciences, University of Bath, England, 1993.

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

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

[17] M.-F.Roy. Basic algorithm in real algebraic geometry and thier complexity : from sturm theorem to the existential theory of reals. Technical report, Université de Rennes, 1996.

[18] T.Recio M.J.Gonzalez Lopez. The romin inverse geometric model and the dynamic evaluation method. In *Studies on Computer Algebra for Industry*, pages p.117–142. A.Cohen, John Wiley and sons, 1993.

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

[20] R.Loos. Generalized polynomial remainder sequences. In R.Loos B.Burch-Berger, G.E.Collins, editor, *Computer Algebra, Symbolic and Algebraic Computation*, pages 115–137. Springer-Verlag, 1983.

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

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

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

[24] Marie Françoise Roy Zenon Legatsikas, Renaud Rioboo. Generic computation of the real closure of an ordered field. In *IMACS Symposium SC-1993*, 1993.