A first course to D7 with examples

Laboratoire d'Arithmétique, de Calcul formel et d'Optimisation
ESA - CNRS 6090

A first course to D_7 with examples

Stéphane Dellière

Rapport de recherche n° 2000-17

A first course to D_7 with examples

S.Dellière
delliere@yahoo.fr

Introduction

In [GD94], T.Gomez-Diaz gives a very simple example in order to introduce dynamic evaluation [DDDD85,DD89]: to solve the equation with one unknown (x): $a\,x + b = 0$ where a, b are parameters. This equation has two different solutions sets S depending on the values of the parameters a, b:

- if $a = 0$ and $b = 0$ then $S = \{ \textit{any } x\};$
- if $a \neq 0$ and *any b* then $S = \{ -b/a\}$.

Dynamic evaluation is the computational process that allows to get automatically both sets of solutions. In the previous example it is obviously supposed that the parameters a and b take values in a field. Note that the formulation of the problem and its solution do not depend of the field considered. This also gives an idea about the genericity of the problem we are taking into account. One of the first applications of dynamic evaluation was computations with algebraic numbers, that is the dynamic algebraic closure program [DDD85,DD89,Duv95a]. Its principle is simple. An algebraic number is considered as a parameter c submitted to an algebraic condition:

$$P(c) = 0$$

where P is an univariate polynomial. One can consider here several parameters that can, for example, be used to produce polynomials to introduce new algebraic parameters. As polynomials in the algebraic conditions aren't irreducible, computations involving these algebraic elements lead quickly to the concept of *split*, as in the above example.
This work has been continued in [GD94] and it leads to the dynamic constructible closure programs implanted in Axiom [JS92] by T.Gomez-Diaz (1992-1996). These programs extend D.Duval programs for the dynamic algebraic closure in Axiom (1988-1992). These programs offer the possibility to make calculus with parameters in a very large way. Indeed, a parameter d can be not only submitted to algebraic constraints but also to inequalities:

$$Q_1(d) \neq 0, \ldots, Q_r(d) \neq 0$$

where $Q_1, \ldots, Q_r$ are univariate polynomials.
The range of applications of the dynamic constructible closure programs is very large. One can mention the computation of Jordan forms with parameters [GD97], automatic demonstration of theorems in elementary geometry [GD93,GD95b], solving constructible systems [GD93,Del99a], i.e. polynomial systems with inequalities ($\neq 0$)…

The D_7 programs are in fact based on an old version of the dynamic constructible closure realized in 1994 by T.Gomez-Diaz in Aldor. This version was produced in order to study the capabilities of the compiler to improve the part of the programs dealing with the splits [Watt et al.]. There is one main difference with the 1994's version. The author adds the square free property of D.Lazard triangular sets [Laz91]. This strategy (developed and justified in [Del99a,Del99b]) gives dramatic improvements on the number of splittings and time complexity. We would like to recall to the D_7 user about the experimental state of the whole set of dynamic evaluation programs. There are several versions (written in Axiom, Aldor) that correspond to the study of different aspects (splits, dealing with denominators, subresultants, the squarefree condition…) unless they provide very little differences from the user's point of view. In particular, the Aldor version mainly corresponds to the state of Aldor back in 1994, which make us dreaming about the real possibilities of an up-to-date, efficient implementation.

This paper presents a brief introduction to D_7 with several examples. We think indeed that learning by example is the best way to understand these programs. Many comments come directly from the previous work of D.Duval and T.Gomez-Diaz (especially [Duv90,GD93,GD94,GD95a]). These are good references for the reader interested also to the Axiom version of the dynamic constructible closure programs. For an introduction to dynamic evaluation in terms of *sketches*, one can see [DR94]. Finally, we refer to [Del99] for an *algebraic* approach.

At last, the author would like to thank T.Gomez-Diaz for her endless encouragements and for her critical lecture of the paper. The author would like to thank D.Duval for all her advices. The author is also grateful to C.Dicrescenzo. Without her, there will be no package D_7. Finally the author would like to thank Cathode team in Grenoble for their reception.

1. A bit of vocabulary

The dynamic constructible closure is an Aldor constructor which provides parameters. More precisely, given a ground field K, one builds *the dynamic constructible closure* of K in the following way:

CL:= DynamicConstructibleClosure(K)

Note that K can be *any* field. This provides a function to introduce *parameters*:

parameter: Symbol → CL

It is the new version of the previous function called **newElement** in [GD94]. In addition, there are two functions to impose or to forbid values for the parameters, i.e. imposing constraints over the parameters:

mustBeEqual: (CL,CL) → Boolean
mustBeDifferent: (CL,CL) → Boolean

It is the new version of the two previous functions called **areEqual** and **areDifferent** in [GD94]. The Boolean result of these operators says whether a new constraint is compatible with the previous ones.
The construction of CL is recursive. If there is no call to the **parameter** function, CL is K. Otherwise, let n be the number of calls of **parameter** and let $a_1,\dots,a_n$ be the generated parameters. Now the elements of CL that we can access are constructed with the constants from K and $a_1,\dots,a_n$ by using field operations (see later). So CL is $K(a_1,\dots,a_n)$. By recursion we can consider that we know how to build the field $K(a_1,\dots,a_{n-1})$ and so the question is reduced to compute in $K(a)$ where K is a field and "a" a parameter introduced with the function **parameter**.
At every step of a calculus in $K(a)$, the parameter a is submitted to a *constraint* (or *law*). Constraints provide any parameter with a description of the set of the possible values; they can be of three possible kinds:

- *anyElement*: there is no constraint on a (this means that a can take any value);
- *algebraic*: there is a constraint of type $P(a) = 0$ with P a monic univariate polynomial of positive degree with coefficients in K (this means that a can take as value any zero of P);
- *exception*: there is a constraint of type $P_1(a) \neq 0$ and ... and $P_r(a) \neq 0$ with $P_1,\dots, P_r$ monic univariate polynomials of positive degree with coefficients in K and pairwise coprime (this means that a can take any value different from any zero of any of the polynomials $P_1,\dots, P_r$).

In addition, when the characteristic of the ground field K is zero, we can suppose that the polynomials $P, P_1,\dots, P_r$ above are squarefree[1].

First note that a parameter a with an algebraic constraint, say $P(a)=0$, place computations in an algebraic extension of the field K. This is the situation considered by the dynamic algebraic

[1] As mentioned in introduction, the reader must note that we have implemented in D_7 the squarefree condition of D.Lazard triangular sets. See [Del99a,Del99b] for more details.

closure programs of D.Duval [DD89,Duv95a]. In this case elements of *K(a)* are represented by polynomials $Q(a) = c_n a^n + \ldots + c_0,\ c_i \in K$, and so there is no problem to perform arithmetic operations. Division needs equality test, which is explained later.
Now if *a* is of exception type, say $P_1(a) \neq 0$ and ... and $P_r(a) \neq 0$, the elements of *K(a)* are represented by functions like:

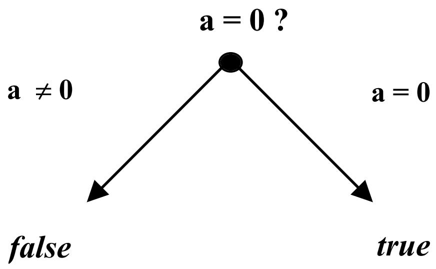
$$\frac{Q(a)}{P_1^{n_1}(a) \ldots P_r^{n_r}(a)}$$

where *Q* is an univariate polynomial with coefficients in *K* and $n_1,\ldots,n_r$ are non-negative integers. Again, there is no problem into performing arithmetic operations and divisions by polynomials which aren't in the multiplicative set generated by $P_1,\ldots,P_r$ lead to equality test.
Finally if *a* is of the "anyElement" type, the elements of *K(a)* are represented by polynomials *Q(a)* with *Q(X)* in *K[X]*.[2]

So the main point now is that parameters can take different values, so that it is in general impossible to answer true or false to an equality test over parameters. When both answers are possible, it is essential to distinguish the values of the parameters corresponding to the true answer from the values corresponding to the false one. This is called a *split*. For example, given a parameter *a* (i.e. an element of *CL*) submitted to an *anyElement* law, suppose that we want to ask if *a* is equal to *0*:

a = 0 ?

By definition of an *anyElement* law, the parameter *a* can take any value so that this question leads to two possible answers:



In fact, this little picture above gives rise to the concept of *splitting tree* introduced by D.Duval in [Duv90], see a complete definition in [BGDW95,GD94]. This is a key tool to see how dynamic evaluation acts. Every computation is associated to a splitting tree from dynamic evaluation point of view (see the references above for more details and examples). In few words, it is a binary tree where a node has two edges if and only if it is related to an equality test for which the two answers true and false are possible (as in the example above), i.e. if we have a split (this is called a *splitting point* in [GD95a]). The exploration of the tree is done from the root (node) to the leaves and from the left to the right. This last is very important because, at a splitting point, the left-edge is always associated with the answer false and the right-edge with the answer true.

[2] One can see [Del99a] for another approach of this problem (the representation of the elements of *CL*) in terms of commutative algebra.

It is important to understand the difference between the equality test = and the two functions **mustBeEqual**, **mustBeDifferent**[3]. Let *a,b* be two parameters. Then:

- **mustBeEqual(a,b)** means that the computation can be continued only in case $a = b$. In the other case, the computation stops without returning any output;
- **mustBeDifferent(a,b)** means that the computation can be continued only in case $a \neq b$. In the other case, the computation stops without returning any output;

Thus, one can consider that **mustBeEqual(a,b)** (resp. **mustBeDifferent(a,b)**) provides the same split that the test **a=b** but we only focus on the branch associated with the true answer (resp. with the false answer) [Duv90].
Otherwise, these three functions use only gcd computations, implemented with subresultants [Duv90,Dell99a].

In fact, there exists another equality, non dynamic this time. It is the function **rawEqual?**. It is the new version of the previous function called **roughEqual** in [GD94]. Suppose that we have introduced a parameter (*a*). *CL* is then equal to *K(a)*. Every element of *K(a)* can be represented by a fraction in *K(X),* where *X* is an indeterminate. Then the goal of the function **rawEqual?** is to detect, without any splitting, if two different fractions represent the same element. For example if the constraint over *a* is $a^2+1=0$ then the element $a+1$ can be represented by $f(X)=X+1$ as well as by $g(X)=(X^3+X^2)/X^2$. Then in this case **rawEqual?(f,g)** returns true. Note that if two elements are "rawEqual" they are equal, but the converse can be false [GD95a]. Finally we refer again to [Del99a] for an approach in terms of commutative algebra of this function.

2. How can we write our own examples ?

The general diagram of an example file is the following:

```
++ First Part ++
K = => ++ ground field;
CL = => DynamicConstructibleClosure(K);
CC = => ConstructibleClause(K,Type);
CP = => ConstructibleControlPackage(K,Type);

++ Second Part ++
f():Type ==
        ++ operations to be performed in CL
        ++ in particular:
        ++ introduction of the parameters

++ Third part ++
finalResult(func: () -> Type): == {
        tim:= cpuTime();
        ll:List CC:= allCases(func)$CP;
        tim:= cpuTime – tim;
        for ele in ll repeat print << ele << newline;
        print << "Time: " << tim << "(mls) " << newlline;
```

[3] It will be the goal of the example 3.1.1.

```
}

++ Last part ++
finalResult(f);
refresh()$CL;
```

In the first part, we introduce the ground field *K* and we build the dynamic constructible closure **CL** of *K*. The package Aldor denoted by **CP** provides one of the key functions of dynamic evaluation programs called **allCases**. This function allows dynamic evaluation programs to give the *complete* answer of a calculus. In other words, it manages the splitting tree corresponding to a computation (see [Section 3.1.2, GD95a]).
A result of a calculus with D_7 is a finite list of *clauses.* More precisely, a clause is an element of the domain **ConstructibleClause** (**CC**) and is made of:

- a value v;
- a tower[4] T.

This means, with our notations above, that the function **f** takes as value v when the values given to the parameters satisfy the constraints in the tower T. Thus, in the example called *consis.as* (p.14), the result of the computation is:

[value is true in case y^2 + (-1/8) = 0 and x = (1/2)]

In this case, it means that the function **consSystem** of this example (**f** with our notations) takes the value true if the values of the parameters *x* and *y* verify the system:

$$\begin{cases} y^2 - 1/8 = 0 \\ x - 1/2 = 0 \end{cases}$$

In most of the examples, the first part is the following:

```
I = => Integer;
RI = => Ratio Integer;
SI = => SingleInteger;
CL = => DynamicConstructibleClosure(RI);
CC = => ConstructibleClause(RI,Boolean);
CP = => ConstructibleControlPackage(RI,Boolean);

import from I,SI,RI,CL,CC,CP;
```

In the second part, the function **f** (which returns a result of type **Type**) defines the calculus to be performed in the dynamic constructible closure. It is important to remark that you can only use a function *without argument*. Furthermore, one must be careful with the way you introduce the parameters. For example, if you write:

```
a:= parameter("a");
b:= parameter("b");
```

[4] Briefly, a tower is a list of laws where each law is of one of the three types mentioned before. See [GD94,Del99] for more details.

then it implies that $a < b$. On the opposite, writing:

```
b:= parameter("b");
a:= parameter("a");
```

means that $b < a$.
In the third part, the function **finalResult** returns (with **allCases**) the complete answer of the computation of a "general" function **func** of type **Type** without argument. Finally, the fourth part calculates the complete answer of the computation of **f**. The last line refreshes the laws operating to the parameters: all of them are then submitted to *anyElement* laws. From a more computational point of view, this cleans some internal variables after any dynamic computation, which can be important if more computations are performed with the same parameters.

Finally, we refer the D_7 user to the file *Readme* for useful informations (installation of D7, environment variables,…) created by C.Dicrescenzo and to the file *about-D7* which contains the list of the Aldor files for dynamic evaluation.

3. Some examples

This section is devoted to the examples given in the package D_7. In each example, the ground field is the field *Q* of the rationals. From now on, we denote by *CL(Q)* the dynamic constructible closure of *Q*.
Each example is made of two files: an input file (.as) and its corresponding output file (.outng). In all this section, we concentrate ourselves on the main computation to operate, i.e. with our previous notations, we focus on the function **f**. Thus, we do not mention anymore that we use the function **allCases,** for example. Finally, for each example, we explain how the problem is treated *in terms of dynamic evaluation*.

3.1. First (easy) examples

3.1.1. File ecgrado1.as

```
DynReso():Boolean == {
        a:CL:= parameter("a");
        b:CL:= parameter("b");
        x:CL:= parameter("x");
        mustBeEqual(a*x+b,0);
}

DynReso():Boolean == {
        a:CL:= parameter("a");
        b:CL:= parameter("b");
        x:CL:= parameter("x");
        (a*x+b=0)$CL;
}
```

This file illustrates the difference between the functions **mustBeEqual** and the dynamic equality "=" of D_7. For this purpose, we introduce three parameters *a*,*b* and *x* in

CL(Q) with the function **parameter**. Note that $a < b < x$. Then in the first function **dynReso** we have:

mustBeEqual(a*x+b,0);

and in the second function **dynReso** we have on the opposite:

(a*x + b = 0)$CL;

Remark. Note that the symbol **$** means in Aldor that the equality "=" above is the one implemented in the domain **CL**.

The difference between these two operators (**mustBeEqual** and "=") is that:

- **mustBeEqual** *imposes* values for the parameters. In this case, we want x to be the zero of the polynomial $aX+b$. Note that x is submitted to an anyElement constraint. So we can summarize the computation by the following weaker form of splitting tree:

$$\begin{tikzcd}
 & \textbf{a:= parameter(“a”)} \arrow[d] & \\
 & \textbf{b:= parameter(“b”)} \arrow[d] & \\
 & \textbf{x:= parameter(“x”)} \arrow[d] & \\
 & \textbf{mustBeEqual(a*x+b,0)} \arrow[d] & \\
 & \mathbf{a = 0\ ?} \arrow[dl, "\textit{false}"'] \arrow[dr, "\textit{true}"] & \\
\textbf{mustBeEqual(x+b/a,0)} \arrow[d, "\textit{true}"'] & & \textbf{mustBeEqual(b,0)} \arrow[d, "\textit{true}"] \\
\left\{\begin{array}{l} \mathbf{x = -\ b/a} \\ \textbf{any b} \\ \mathbf{a \neq 0} \end{array}\right. & & \left\{\begin{array}{l} \textbf{any x} \\ \mathbf{b = 0} \\ \mathbf{a = 0} \end{array}\right.
\end{tikzcd}$$

First note that there is a splitting point in the computation associated to the question (**a = 0 ?**). This is because we want to make monic the polynomial *aX+b* (which represents the element *ax+b).* So we have to consider the two cases *a=0* and $a \neq 0$. In the last case we carry on the computation under the hypothesis $a \neq 0$ and then impose the constraint $x + b/a = 0$. The parameters *b* and *x* are submitted to anyElement laws. Therefore the new set of constraints is obviously: $\{a \neq 0$, *any b* and $x + b/a = 0\}$.
On the opposite, in case $a = 0$, the polynomial *aX+b* is then reduced to *b*. So we are led to impose the constraint *b=0*. Then the new set of constraints is obviously: *{a = 0, b= 0* and *any x}* (we have not impose an additional constraint on *x*).
This explains that the result (see the file ecgrado1.outng) is:[5]

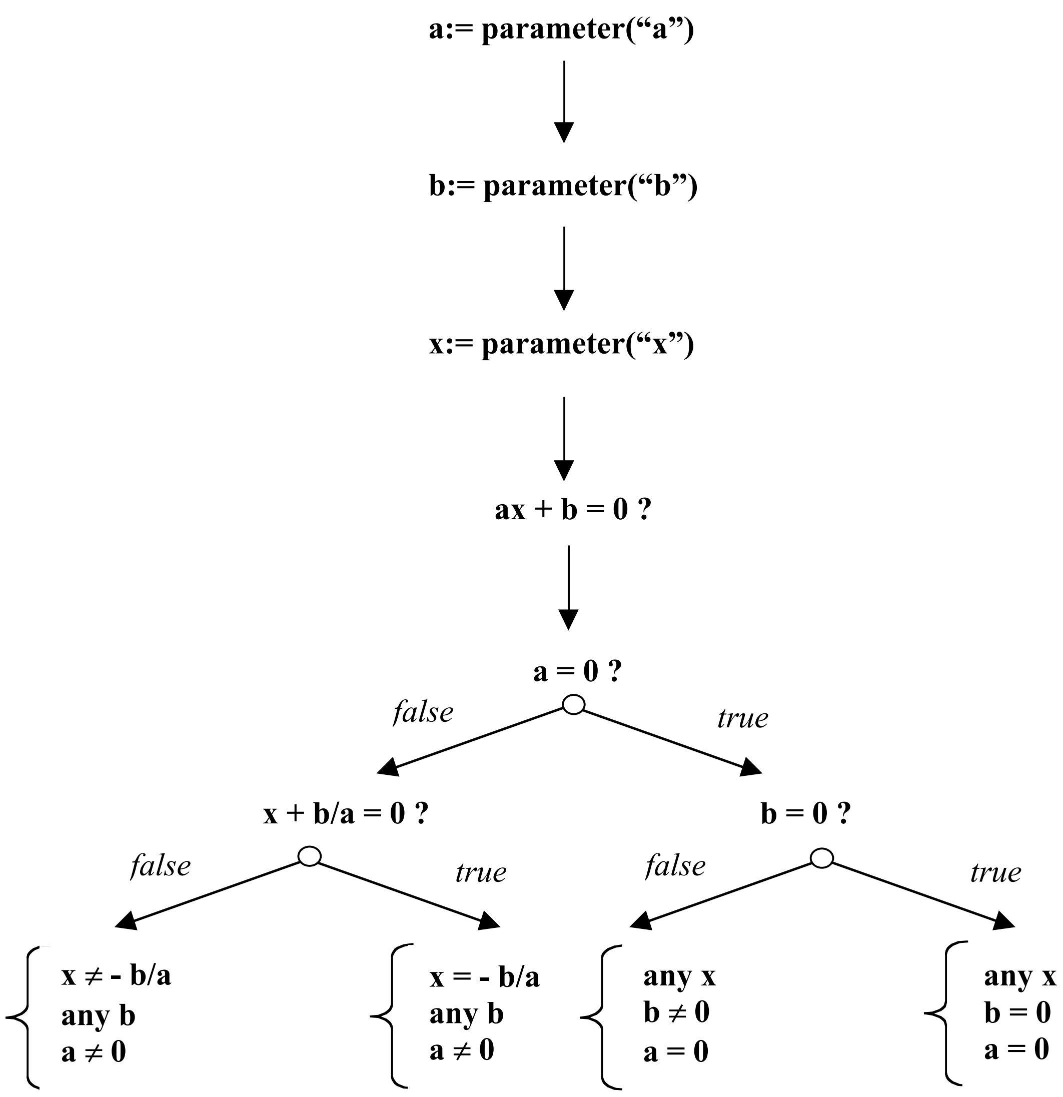
```
[value is true in case any x and b = (0/1) and a = (0/1)]
[value is true in case x = ((-1/1)/a))*/b and any b and a /= (0/1)]
```

This means that we can impose the constraint *ax+b=0* on the parameters *a,b,x* if and only if they satisfy the following sets of constraints:

$\{a=0$ and $b=0$ and *any x}*, $\{a \neq 0$ and *any b* and $x= -b/a\}$.

- The goal of the operator "=" is quite different[6]. It is not used to impose values on parameters but rather to ask whether parameters can take specific values or not. Thus, the operation "**(a*x + b = 0)$CL**" means that you ask the question: "are there values of the parameters *a,b,x* such that the expression *ax+b* is zero ?". One can then construct the following weaker form of splitting tree associating with the computation:

[5] One can see the poor writing of the outputs here and in the results of the next computation.
[6] We refer to [Section 4.2, Del99a] for an algebraic approach of this dynamic equality.



As in previous example, the goal of the first split is to make monic the polynomial *aX+b*. Moreover, there are two splitting points for the questions: **x + b/a = 0 ?** and **b = 0 ?**. This is because we consider this time the values of the parameters *a,b,x* for which the answer of each question is true *and* the values of the parameters *a,b,x* for which the answer is false. We do not focus only on the true answer as previously when we were using the function **mustBeEqual**.
That's why the result of this computation is (see file ecgrado1.outng):

```
[value is true in case any x and b = (0/1) and a = (0/1)]
[value is false in case any x and b /= (0/1) and a = (0/1)]
[value is true in case x = ((-1/1)/a))*/b and any b and a /= (0/1)]
[value is false in case x /= ((-1/1)/a))*/b and any b and a /= (0/1)]
```

This means that the expression *ax+b* is zero if and only if the parameters *a,b,x* verify the sets of constraints:

$$\{a=0 \text{ and } b=0 \text{ and } any\ x\},\ \{a \neq 0 \text{ and } any\ b \text{ and } x= -b/a\}$$

and the expression $ax+b$ is not equal to zero if and only if the parameters a,b,x verify the sets of constraints:

$$\{a=0 \text{ and } b \neq 0 \text{ and } any\ x\},\ \{a \neq 0 \text{ and } any\ b \text{ and } x \neq -b/a\}.$$

3.1.2. File ecgrado2.as [GD94,Del99a]

We can use D_7 to solve the classic equation:

$$ay^2+by+c=0$$

This is quite simple. After the introduction of the parameters a,b,c and y, we just have to call the function **mustBeEqual**. We obtain then the little program:

```
DynReso():Boolean == {
        a:CL:= parameter("a");
        b:CL:= parameter("b");
        c:CL:= parameter("c");
        y:CL:= parameter("y");
        mustBeEqual(a*y^2+b*y+c,0) ;
}
```

with result (see ecgrado.outng):

```
[value is true in case any y and c = (0/1) and b = (0/1) and a = (0/1)]
[value is true in case y = ((-1/1)/(b))*c and any c and b /= (0/1) and a = (0/1)]
[value is true in case y = ((-1/2)/(a))*b and c = ((1/4)/(a))*b^2 and any b and a /= (0/1)]
[value is true in case y^2 + (((1/1)/a))*b)*y + ((1/1)/(a))*c = 0 and c /= ((1/4)/(a))*b^2 and
any b and a /= (0/1)]
```

If we denote by S the set of solutions of the equation, this means that:

- if $a = b = c = 0$ then $S = CL(Q)$;
- if $a = 0$ and $b \neq 0$ then $S = \{-c/b\}$;
- if $a \neq 0$ and $c = b^2/(4a)$ then $S = \{-b/(2a)\}$;
- if $a \neq 0$ and $c \neq b^2/(4a)$ then $S = \{y \in CL(Q);\ y^2+(b/a)y+(c/a)=0\}$.

3.1.3. File ecdeno.as [GD94]

In this example, we also want to solve an equation (with the unknown y and a parameter a). But this time, there are non trivial denominators:

$$\frac{y + 2a}{2a - y} + \frac{y - 2a}{2a + y} = \frac{4a^2}{4a^2 - y^2}$$

Of course, this is equivalent to:

$$4a^2 - y^2 \neq 0 \text{ and } (y + 2a)^2 - (y - 2a)^2 = 4a^2$$

where there is no more denominator but an inequation instead. In fact, the goal of this example is to show that D_7 allows to solve *directly* equations with non trivial denominators

To solve the equation with D_7 is not difficult. After introducing the parameters *a* and *y* with **parameter**, we need to impose the conditions:

$$2a - y \neq 0,\ 2a + y \neq 0 \text{ and } 4a^2 - y^2 \neq 0.$$

with **mustBeDifferent** in order to be able to construct the denominators. Then we just have to translate the equation into D_7 terms with **mustBeEqual**. The only subtlety is to solve the equation under the hypothesis that the two parameters satisfy the three inequalities. We construct then the following program:

```
dynReso():Boolean == {
        a:CL:= parameter("a");
        y:CL:= parameter("y");
        tt :Boolean := {
                mustBeDifferent((2@I)*a-y,0);
                mustBeDifferent((2@I)*a+y,0);
                mustBeDifferent((4@I)*a^2-y^2,0);
        }
        if tt then {
                e1:CL:= (y+(2@I)*a)/((2@I)*a-y) + (y-(2@I)*a)/((2@I)*a+y);
                mustBeEqual(e1,((4@I)*a^2)/((4@I)*a^2-y^2));
        }
        else tt;
}
```

where we denote by I the Aldor domain for the integer numbers. The reader not familiar with Aldor have to know that when we write for example:

4@I

it means that we take the element "**4**" of the domain **I**.

We obtain then the result (see the file ecdeno.outng):

```
[value is true in case y /= (0/1) and a = (0/1)]
[value is true in case y /= (1/2)*a and a /= (0/1)]
```

If we denote by *S* the set of solutions of our equation, it means that:

- if $a = 0$ then $S = \{y \in CL(Q); y \neq 0\};$
- if $a \neq 0$ then $S = \{a/2\}.$

3.2. Solving polynomial and constructible systems

A polynomial system or a constructible system (i.e. a polynomial system with inequalities $\neq 0$) with coefficients in *K* can be "solved", in the sense of triangularized, by computations in the constructible closure of *K*. The key-point is to consider each indeterminate as a parameter introduced by **parameter** and respectively each equation and

each inequation as a constraint introduced by **mustBeEqual** or **mustBeDifferent**. In particular, this means that indeterminates, equations and inequations are ordered.
Suppose that we want to triangularize an algebraic system T (polynomial or constructible) with coefficients in a field K. We get as output a finite union of *constructible triangular systems* $T_1,\dots,T_r$ with the Boolean true and *constructible triangular systems* $T_{r+1},\dots,T_s$ with the Boolean false. This means that (see [GD94,DGD95,Del99a] for more details):

$$Zero(T) = Zero(T_1) \uplus Zero(T_2) \uplus \dots \uplus Zero(T_r)$$

where:

- given two sets E and F, the expression $E \uplus F$ is the disjoint union of E and F;
- given a constructible triangular system S, we denote by *Zero(S)* the set of zeros of S in CL(K).

Note that the systems $T_{r+1},\dots,T_s$ are useless in this context as they are associated with the Boolean false.

One must note that these systems $T_1,\dots,T_s$ are not "polynomial" in the usual sense. Indeed, if $a_1,\dots,a_n$ denote the parameters (introduced in this order), each constructible triangular system T_i has the form:

$$\begin{cases} expr_n(a_n) \\ \dots \\ \dots \\ expr_2(a_2) \\ expr_1(a_1) \end{cases}$$

where $exp_i(a_i)$ is a description of a constraint for a_i over $K(a_1,a_2,\dots,a_{i-1})$. Thus $exp_i(a_i)$ is either:

- *any* a_i for the *anyElement* constraint;
- $P_{i,1}(a_i) \neq 0, \dots, P_{i\,k(i)}(a_i) \neq 0$ for an *exception* constraint;
- $P_i(a_i) = 0$ for an *algebraic* constraint;

with $P_{i,1}(X_i), \dots, P_{i,k(i)}(X_i), P_i(X_i)$ in $K(a_1,\dots, a_{i-1})[X_i]$, monic, of positive degree in a_i, and square free (in the sense of D.Lazard, see footnote p.4 [Del99a,Del99b]) if K has characteristic 0. In addition, the polynomials $P_{i,1}(X_i), \dots, P_{i,k(i)}(X_i)$ are coprime. The non-trivial denominators which appear in coefficients of these polynomials involve only the parameters a_j of *exception* type (with $j < i$).
In fact, we can establish an algebraic model for these systems in the classic commutative algebra. This is one of the main goals of the work done in [Del99a]. The reader interested by these questions is also refered to [Del99b,Del99c] where we point out connections between these kind of systems and respectively D.Lazard triangular sets [Laz91] and D.M.Wang simple systems [Wan98].

3.2.1. File var4.as [GD93,GD95c]

In this example, we want to "solve" in the sense of triangularization the following polynomial system (see for example [Laz92]):

$$\begin{cases} a + b + c + d = 0 \\ ab + bc + cd + da = 0 \\ abc + bcd + cda + dab = 0 \\ \mathrm{abcd} - 1 = 0 \end{cases}$$

with $a < b < c < d$. It is the system of "cyclic 4-roots". In order to solve this system with D_7, we have first to introduce the parameters *a,b,c,d* which represent the variables of the system. This is done in the dynamic constructible closure in the following way:

```
a:CL:= parameter("a");
b:CL:= parameter("b");
c:CL:= parameter("c");
d:CL:= parameter("d");
```

Then we only have to interpret every equation as a call of **mustBeEqual** that is to say[7]:

```
mustBeEqual(a + b + c +d,0)
and mustBeEqual(ab + bc + cd + da,0)
and mustBeEqual(abc + bcd + cda + dab,0)
and mustBeEqual(abcd – 1,0);
```

We deduce then easily a little program which solve "cyclic 4-roots" and obtain the following result:

```
[value is false in case d  = - b and c = (0/1) and any b and a = (0/1)]
[value is true in case d = - b and c = - a and b^2 + (-1/1)/(a)^2 = 0 and a /= (0/1)]
```

This means that the solutions of "cyclic 4-roots" can be also expressed as the solutions of the constructible triangular system:

$$\begin{cases} d + b = 0 \\ c + a = 0 \\ b^2 - 1/a^2 = 0 \\ a \neq 0 \end{cases}$$

Of course, one can use other decomposition methods to "solve" the system "cyclic 4-roots" (see for example [MM97]).

3.2.2. File romin.as [GLR93,Del99a,Del99b]

In this file, we are interested in an example coming from robotic called the ROMIN inverse geometric model [GLR93]. This problem leads to the following constructible system:

[7] You should not forget the "and" when you translate polynomial or constructible systems into dynamic evaluation terms !

$$\begin{cases} l_2 \neq 0 \\ l_3 \neq 0 \\ -ds_1-a=0 \\ dc_1-b=0 \\ l_2c_2+l_3c_3-d=0 \\ l_2s_2+l_3s_3-c=0 \\ s_1^2+c_1^2-1=0 \\ s_2^2+c_2^2-1=0 \\ s_3^2+c_3^2-1=0 \end{cases}$$

with $d < c < b < a < l_3 < l_2 < s_1 < c_1 < s_2 < c_2 < s_3 < c_3$.

It is easy to translate this system in the dynamic constructible closure using the three functions **parameter**, **mustBeEqual** and **mustBeDifferent**:

```
DynamicReso():Boolean == {
        d:CL:= parameter("d");
        c:CL:= parameter("c");
        b:CL:= parameter("b");
        a:CL:= parameter("a");
        l3:CL:= parameter("l3");
        l2:CL:= parameter("l2");
        s1:CL:= parameter("s1");
        c1:CL:= parameter("c1");
        s2:CL:= parameter("s2");
        c2:CL:= parameter("c2");
        s3:CL:= parameter("s3");
        c3:CL:= parameter("c3");
        mustBeDifferent(l2,0)
        and mustBeDifferent(l3,0)
        and mustBeEqual(-d*s1-a,0)
        and mustBeEqual(d*c1-b,0)
        and mustBeEqual(l2*c2+l3*c3-d,0)
        and mustBeEqual(l2*s2+l3*s3-c,0)
        and mustBeEqual(s1^2+c1^2-1,0)
        and mustBeEqual(s2^2+c2^2-1,0)
        and mustBeEqual(s3^2+c3^2-1,0);
}
```

After the computation (see the file romin.outng), we obtain four constructible triangular systems with the Boolean true. We refer to [section 7.4 p.211, Del99] for a detailed analysis of the result.
One can note that this system have been also studied in the polynomial case i.e. without the inequalities (see [AMM99,Aub99,MM97]).

3.2.3. File consis.as [GD94]

It is a second example of *constructible* system solving. More precisely, we want to "solve" the following constructible system:

$$\begin{cases} x \neq 0 \\ y^2-x^3 =0 \\ 2y^2-x^2 = 0 \\ (8x-3)^2+8y^2-9 = 0 \end{cases}$$

with $x < y$. Nothing difficult now, it leads to the function:

```
DynReso():Boolean == {
        x:CL:= parameter("x");
        y:CL:= parameter("y");
        mustBeDifferent(x,0)
        and mustBeEqual(y^2-x^3,0)
        and mustBeEqual((2@I)*y^2-x^2,0)
        and mustBeEqual(((8@I)*x-(3@I::CL))^2+((8@I)*y)^2,9@I::CL);
}
```

with result:

[value is true in case y^2-1/8=0 and x=1/2]

This means that the set *S* of solutions of the constructible system is:

$$S = \{(1/2,y) \text{ with } y^2=1/8\}.$$

3.3. Automatic theorem proving in elementary geometry

Let us first explain what we call automatic theorem proving in elementary geometry and how we can use D_7 to study this kind of problems. Given a field *K* of characteristic zero, we denote by $h_1,\ldots,h_r$ and *t* r+1 polynomials in $K[X_1,\ldots,X_n]$ such that $H=\{h_1,\ldots,h_r\}$ and *t* are respectively the set of hypotheses and the conclusion of a theorem *T* in elementary geometry:

$$T: (H=0) \implies (t=0).$$

The problem is to study the validity of the theorem *T*.

Definition [definition 1.3 p.5, RV99]. The theorem ($(H=0) \implies (t=0)$) is *geometrically true* if the variety of the hypotheses *V(H)* is included in the variety of the conclusion *V(t)*.

We can use D_7 to decide whether the theorem *T* is geometrically true or not. For this purpose, we write a program in the following form:

```
if (mustBeEqual(h1,0)
        and mustBeEqual(h2,0)
        …
        …
        and mustBeEqual(hr,0))
then [(t = 0)$CL];
else [failed];
```

By construction, we obtain three kinds of systems in the output (i.e. three kinds of clauses):

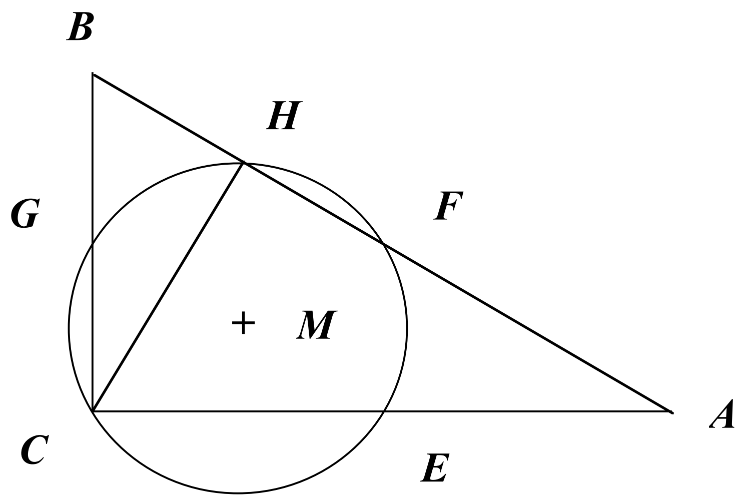
- constructible systems with the Boolean true: they describe the variety V(H) ∩V(t);
- constructible systems with the Boolean false: they describe the set V(H) – V(t);
- constructible systems with "failed": they describe subsets of $CL(Q)^n - V(H)$.

One can then easily prove that the theorem *T* is geometrically true if and only if there is no constructible system associated with the Boolean false.

Finally, in this kind of problems, one can note that the operator **mustBeDifferent** is a good tool to avoid degenerate cases. For example we can impose that two points are distinct or that three points are not on a line [GD93]. Let's see an example.

3.3.1. File dccgeo.as[8] [Duv90,GD93,GD94,GD95c]

In this file, we want to prove the following theorem. Given a triangle *ABC*, right-angled in *C*, the circle *Γ* through the midpoints *E*, *F* and *G* of the sides *CA*, *AB* and *BC* (*Γ* is the Euler's circle of triangle *ABC*), and the foot *H* of the altitude from *C*. Then *Γ* passes through *H*.



We now state this problem as a computation in the constructible closure of the field *Q*. Let *M* denote the center of the circle *Γ*. We choose a coordinate system adapted to the problem and we give a name to the coordinates of every point in the figure. For example:

$$C=(0,0)\ A=(a_1,0)\ B=(0,a_2)\ E=(a_3,0)\ F=(a_4,a_5)\ G=(0,a_6)\ M=(a_7,a_8)\ H=(a_9,a_{10}).$$

So we need ten parameters in the constructible closure: $a_1,a_2,\dots,a_{10}$. To avoid degenerate triangles we impose:

$$a_1 \neq 0,\ a_2 \neq 0.$$

[8] The beginning of this section (i.e. the translation of this problem into dynamic evaluation context) comes directly from [GD93].

The choice of the coordinate system already reflects that the triangle is right-angled. Next we write that *E*, *F* and *G* are the midpoints of *CA*, *AB* and *BC* respectively:

$$a_3=(1/2)a_1,\ a_4=(1/2)a_1,\ a_5=(1/2)a_2,\ a_6=(1/2)a_2.$$

The center *M* of the circle Γ is given by the conditions *EM=FM* and *EM=GM*, hence:

$$(a_7 - a_3)^2 + a_8^2 = (a_7 - a_4)^2 + (a_8 - a_5)^2$$
$$(a_7 - a_3)^2 + a_8^2 = a_7^2 + (a_8 - a_6)^2$$

The point *H* is the intersection of two perpendicular lines *AB* and *CH*. The equation of *AB* is:

$$(1/a_1)x + (1/ a_2)y = 1.$$

(recall that a_1 and a_2 are non-zero) hence:

$$(1/ a_1)\ a_1 + (1/ a_1) = 1$$
$$- a_1 a_9 + a_2 a_{10} = 0.$$

Finally, the conclusion of the theorem is *EM = HM*, that is:

$$(a_7 - a_3)^2 + a_8^2 = (a_7 - a_9)^2 + (a_8 - a_{10})^2.$$

The corresponding program is:

```
Geo():Union(bo:Boolean, fa:Enumeration(failed)) == {
        a1:CL:= parameter("a1");
        a2:CL:= parameter("a2");
        a3:CL:= parameter("a3");
        a4:CL:= parameter("a4");
        a5:CL:= parameter("a5");
        a6:CL:= parameter("a6");
        a7:CL:= parameter("a7");
        a8:CL:= parameter("a8");
        a9:CL:= parameter("a9");
        a10:CL:= parameter("a10");
        e1:CL:= (a7-a3)^2 + a8^2 - (a7-a4)^2 – (a8-a5)^;
        e2:CL:= (a7-a3)^2 + a8^2 –a7^2 – (a8-a6)^2;
        e3:CL:= - a1*a9 + a2*a10;
        e4:CL:= (a7-a3)^2 + a8^2 – (a7-a9)^2 – (a8-a10)^2;
        if (mustBeDifferent(a1,0)
                and mustBeDifferent(a2,0)
                and mustBeEqual(a3,(1/2)*a1)
                and mustBeEqual(a4,(1/2)*a1)
                and mustBeEqual(a5,(1/2)*a2)
                and mustBeEqual(a6,(1/2)*a2)
                and mustBeEqual(e1,0)
                and mustBeEqual(e2,0)
                and mustBeEqual((1/a1)*a9 + (1/a2)*a10 – 1,0)
                and mustBeEqual(e3,0))
```

```
            then [(e4 = 0)$CL);
            else [failed];
        }
```

with result:[9]

```
[value is U[2, ??@Enumeration(??)] in case a10 = (((-1/1)/(a1))*a2)*a9 + a2 and
any a9 and a8= ((1/4))*a2 and a7 = ((1/4))*a1 and a6 = ((1/2))*a2 and a5 =
((1/2))*a2 and a4 = ((1/2))*a1 and a3 = ((1/2))*a1 and a2^2 + a1^2 = 0 and a1 /=
(0/1)],
[value is U[1, true@Boolean] in case a10 = (a1^2)*a2/(a2^2 + a1 ^2) and a9 =
(a1)*a2^2/(a2^2 +a1^2) and a8 = ((1/4))*a2 and a7 = ((1/4))*a1 and a6 =
((1/2))*a2 and a5 = ((1/2))*a2 and a4 = ((1/2))*a1 and a3 = ((1/2))*a1 and a2 /=
(0/1), a2^2 + a1^2 /=0 and a1 /= (0/1)]
```

This means that:

- in the first case, the answer is "failed" which means that it is impossible to build the figure. Note that this case cannot happen if a_1 and a_2 are real because they verify:

$$\begin{cases} a_2^2 + a_1^2 = 0 \\ a_1 \neq 0 \end{cases}$$

- In the second case, the answer is true. Then the theorem is true in this case, i.e. if the parameters $a_1, a_2, \ldots, a_{10}$ verify the constraints of this clause:

$$\begin{cases} a_{10} = a_1^2 a_2/(a_2^2+a_1^2) \\ a_9 = a_1 a_2^2/(a_2^2+a_1^2) \\ a_8 = (1/4)a_2 \\ a_7 = (1/4)a_1 \\ a_6 = (1/2)a_2 \\ a_5 = (1/2)a_2 \\ a_4 = (1/2)a_1 \\ a_3 = (1/2)a_1 \\ a_2 \neq 0,\ a_2^2 + a_1^2 \neq 0 \\ a_1 \neq 0 \end{cases}$$

The key point here is that there is no answer with Boolean value false. Therefore the theorem is geometrically true and so it is proved.

3.4. Gcd of two univariate polynomials with parameters

One can use D_7 to compute gcd of two univariate polynomials with parameters in their coefficients. This method is similar to the one developed by S.A.Abramov and K.Yu.Kvashenko in [AK93]. In fact, there are deep connections between these two methods [Del00].

[9] The reader can see again the poor writing of the outputs. In particular, by **??@Enumeration(??)** one must understand "failed".

3.4.1. File ko.as [GD95c]

In this file, we want to compute the gcd of the two univariate polynomials (with coefficients in the field *Q* of the rational numbers):

$$q_1 = X^2+aX-3a$$
$$q_2 = X^2+bX-3b$$

where *a, b* are parameters. For this purpose, we have to consider the domain **DynamicUnivariatePolynomial(CL)** (denoted DUP) of the dynamic univariate polynomials with coefficients in **CL**. We recall that (in practice) **CL** is here the Aldor symbol which represents the dynamic constructible closure of the field *Q*.
The polynomials q_1 and q_2 belong to the domain **DUP**. We use a function gcd implemented in this domain. The resulting program is the following:[10]

```
x:DUP:= monomial(1@CL,1$SI)$DUP;

DynamicGcd():DUP == {
        a:CL:= parameter("a");
        b:CL:= parameter("b");
        p1:DUP := x^2+(1$SI)*(1$DUP) ;
        p2:DUP := x^2+(2@SI)*(1$DUP) ;
        f :DUP := x – (3@I::CL)*(1$DUP);
        gcd(p1+a*f,p2+b*f)$DUP ;
}
```

The first line introduces the polynomial *X* of the Aldor domain **DUP** with the function **monomial**. The reader not familiar with Aldor needs to know that the symbol "**$**" in **(1$SI)** means that **1** is a function implemented in the **SingleInteger** domain (**SI**). Note that the symbol @ in "**1@CL**" (for example) means that we consider the element **1** in the domain **CL**. Moreover the expression **(3@I::CL)** means that the element **3** of the domain **I** of integer numbers is coerced to an element of the constructible closure domain **CL**.

The polynomials **p1** and **p2** are respectively X^2+1 and X^2+2. Their definition follows from the structure of "**SI** left-module" of **DUP**. More precisely, the instruction:

1$DUP

denotes the constant polynomial of **DUP** equal to *1*. Then, there is a left multiplication with elements of the domain **SI**. So if we want to define the constant polynomial of **DUP** equal to *2* we write (note that there are multiple ways to write it):

(2@SI)*(1$DUP).

We finally call the function gcd of the domain DUP with the two univariate polynomials **p1+a*f** and **p2+b*f** which represent respectively the polynomials $q_1 = X^2+aX-3a$ and $q_2 = X^2+bX-3b$.

[10] Note that we can also define the polynomials **p1** and **f** as **p1:DUP:= x^2+(1$DUP) ;** and **f:DUP:= x – (3@SI)*(1$DUP)**.

Using the function **allCases**, we obtain the result[11]:

[value is ? + ((-3/1))*a + (2/1)/(a + (6/1)) in case b = ((21/20))*a + (3/10) and a^2 + ((12/1))*a + (-4/1) = 0]
[value is (1/1) in case b /= ((21/20))*a + (3/10) and a2^+ ((12/1))*a + (-4/1) = 0]
[value is ? + ((-3/1))*b + ((3/1))*a +(1/1)/(b + - a) in case b^2 + (((-21/10))*a + (-3/5))*b + ((11/10))*a^2 + ((3/5))*a + (1/10) = 0 and a^2 + ((12/1))*a + (-4/1) /= 0]
[value is (1/1) in case b^2 + (((-21/10))*a + (-3/5))*b + ((11/10))*a^2 + ((3/5))*a + (1/10) /= 0 and a^2 + ((12/1))*a + (-4/1) /= 0]

This means that:

- if $a^2 + 12a - 4 = 0$ and $b = (21/20)a + (3/10)$ then $gcd(q_1,q_2) = X - 3a + 2/(a+6)$;
- if $a^2 + 12a - 4 = 0$ and b $\neq (21/20)a + (3/10)$ then $gcd(q_1,q_2) = 1$;
- if $a^2 + 12a - 4 \neq 0$ and $b^2 + ((-21/10)a-3/5)b + (11/10)a^2 + (3/5)a + (1/10) = 0$ then $gcd(q_1,q_2) = X - 3b + 3a + 1/(b-a)$;
- if $a^2 + 12a - 4 \neq 0$ and $b^2 + ((-21/10)a-3/5)b + (11/10)a^2 + (3/5)a + (1/10) \neq 0$ then $gcd(q_1,q_2) = 1$.

3.5. Linear algebra with parameters

3.5.1. File mat.as [GD94,BGDW95]

In this file, we ask for the rank of the two matrices:

$$M_1 = \begin{pmatrix} 1 & 1 \\ 1 & a \end{pmatrix} \qquad M_2 = \begin{pmatrix} a & 1 \\ 1 & 1 \end{pmatrix}$$

according all possible values of the parameter a. The reader must note that there is a mistake in the file mat.as which deals with a matrix M_2 equal to:

$$\begin{pmatrix} 1 & a \\ 1 & 1 \end{pmatrix}$$

and then leads to a second example of rank computing not very interesting…
To compute the rank of these matrixes, we only need to introduce the domain of the matrices with coefficients in **CL** (noted **MM**) and call a standard rank function (*implemented without any "parameter" notion* in a domain called **MiMatrixOps**). Therefore, if we denote for example **dynRang1** the function Aldor which calculate the rank of the matrix M_1, one can implement **dynRang1** as follows:

```
dynRang1():SI == {
        a:CL:= parameter("a");
        m:MM:= matrix[[1,1],[1,a]];
        rank(m)$MiMatrixOps(CL) ;
}
```

Recall that **SI** is the abbreviation for the domain **SingleInteger**. We obtain the result:

[11] In output, the symbol "?" denotes the indeterminate X. This is another example of the poor writing of the outputs, you should not forget that these programs are in an experimental state.

[value is 1 in case a = (1/1), value is 2 in case a /= (1/1)]

This means that:

- if *a=1* then the rank of M_1 is *1*;
- in the other cases ($a\neq 1$) then the rank of M_1 is *2*.

In order to compute the rank of the matrix M_2, we implement the following program:

```
DynRang2():SI == {
        a:CL:= parameter("a");
        m:MM:= matrix[[a,1],[1,1]];
        rank(m)$MiMatrixOps(CL) ;
}
```

with result:

**[value is 2 in case a = (0/1), value is 1 in case a = (1/1),
value is 2 in case a /= (0/1), a /= (1/1)]**

This means that:

- if *a=0* then the rank of M_2 is *2*;
- if *a=1* then the rank of M_2 is *1*;
- in the other cases ($a\neq 0,1$) then the rank of M_2 is *2*.

One can note the difference with the computation of the rank of M_1. It is because the parameter *a* is the first entry of the matrix M_2. So one has to distinguish the case $a = 0$ and $a \neq 0$ when applying a classical rank algorithm.

3.5.2. File linsis.as [GD94]

In this file, we want to solve the following linear system with one parameter *a* and two unknowns $\{x,y\}$:

$$\begin{pmatrix} a & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

It is quite simple with D_7, you just have to introduce *a, x* and *y* in the dynamic constructible closure with **parameter** and then translate the two equations as two calls of **mustBeEqual**[12]. This leads to the implementation of the following little function:

```
DynamicReso():Boolean == {
        a:CL:= parameter("a");
        x:CL:= parameter("x");
        y:CL:= parameter("y");
        mustBeEqual(a*x+y, 2@I:CL)
```

[12] In fact, we just have to consider every linear system as a particular case of polynomial system. If there are additional constraints on the parameters, one has to consider the polynomial or constructible system made of the linear system and these additional equations and inequations.

```
            and mustBeEqual(x + y, 1);
    }
```

with result:

```
    [value is false in case y = -x + (2/1) and any x and a = (1/1)]
    [value is true in case y = a +(-2/1)/(a + (-1/1)) and x = (1/1)/(a + (-1/1)) and a /=
    (1/1)]
```

This means that the set of solutions *S* of the linear system is:

- if $a \neq 1$ then $S = \left\{ \begin{pmatrix} 1/(a-1) \\ \\ a - 2/(a-1) \end{pmatrix} ; a \in CL(Q) \right\}$;
- $S = \varnothing$ otherwise.

3.5.3. File rob.as

This file contains three other examples of solving linear systems. More precisely, the functions **lin1**, **lin2** and **lin3** are implemented to solve respectively the three following linear systems:

$$\begin{pmatrix} 1 & -2 & 3 \\ 2 & a & 6 \\ -1 & 3 & a-3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 6 \\ a \end{pmatrix}$$

$$\begin{pmatrix} 1 & -2 & 3 \\ 2 & a & 6 \\ -1 & 3 & (a/b)- \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} b \\ 6b \\ a \end{pmatrix}$$

$$\begin{pmatrix} -2 & 3 \\ a & 6 \\ 3 & a-3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Note that in the second one (see the function **lin2**), we impose two additional conditions on the parameters *a, b*:

$$\begin{cases} a^2+b^2 = 1 \\ b \neq 0 \end{cases}$$

The treatment of these three problems is exactly the one presented in the file **linsis.as**.

References.

[AK93] S.A. Abramov, K.Yu. Kvashenko. - *On the greatest common divisor of polynomials which depend on a parameter.* Proceedings ISSAC'93, p.152-156 (1993).

[AMM99] P.Aubry, M.Moreno Maza. *Triangular sets for solving polynomial systems: A comparative implementation of four methods.* Journal of Symbolic Computation 28 (1-2) p.125-154 (1999).

[Aub99] P. Aubry. - *Ensembles triangulaires de polynômes et résolution de systèmes algébriques. Implantation en Axiom.* PhD thesis, Université de Paris 6 (1999).

[BGDW95] P.A. Broadbery, T. Gomez-Diaz, S.M. Watt. - *On the implementation of dynamic evaluation.* Proceedings ISSAC'95, p.77-84 (1995).

[DDDD85] J. Della Dora, C. Dicrescenzo, D. Duval. - *About a new method for computing in algebraic number fields.* Eurocal'85, vol. 2, Springer Lecture Notes in Computer Science 204, ed. G. Goos, J. Hartmanis, p.289-290 (1985).

[Del99a] S. Dellière. - *Triangularisation de systèmes constructibles. Application à l'évaluation dynamique.* PhD thesis, Université de Limoges, http://www.unilim.fr/laco/theses/index.html (1999).

[Del99b] S. Dellière. - O*n the links between triangular sets and dynamic constructible closure.* To appear in the Journal of Pure and Applied Algebra.

[Del99c] S. Dellière. - *Simple systems and dynamic constructible closure.* Preprint.

[Del00] S. Dellière. - *Pgcd de deux polynômes à paramètres : approche par la clôture constructible dynamique et généralisation de la méthode de S.A.Abramov et de K.Yu.Kvashenko.* Rapport de recherche I.N.R.I.A. RR-3882, http://www.inria.fr/RRRT/RR-3882.html (2000).

[Duv89] D.Duval. – *Computations in fields of arbitrary characteristic.* Computers and Mathematics, Springer, ed. E.Kaltofen, S.M.Watt, p.321-326 (1989).

[DD89] C. Dicrescenzo, D. Duval. - *Algebraic extensions and algebraic closure in Scratchpad.* Symbolic and algebraic computation, Springer Lecture Notes in Computer Science 358, ed. P. Gianni, p.440-446 (1989).

[Duv90] D.Duval. – *Calculs avec discussion automatique : description et applications.* Publications du Département de Mathématiques de Limoges (1990).

[DR94] D. Duval, J.C. Reynaud. - *Sketches and computation. Part I: Basic Definitions and Static Evaluation. Part II: Dynamic Evaluation and Applications.* Mathematical Structures for Computer Science 4, p.185-238 and 239-271. Cambride University Press (1994).

[Duv95a] D.Duval. - *Evaluation dynamique et clôture algébrique en Axiom.* Journal of Pure and Applied Algebra 99, p.267-295 (1995).

[Duv95b] D. Duval. - *Symbolic or algebraic computation ?* Publications du Département de Mathématiques de Limoges (1995).

[DGD95] D. Duval, T. Gomez-Diaz. - *A lazy method for triangularizing polynomial systems.* Publications du Département de Mathématiques de Limoges (1995).

[GD93] T. Gomez-Diaz. - *Examples of using dynamic constructible closure.* Proceedings of International Imacs Symposium on Symbolic Computation, ed. G. Jacob, N.E. Oussous, S. Steiberg. International Association for Mathematics and Computers in Simulation, p.17-21 (1993). Also in mathematics and Computers in Simulation 42, p.375-383 (1996).

[GD94] T. Gomez-Diaz. - *Quelques applications de l'évaluation dynamique.* PhD thesis, Université de Limoges (1994).

[GD95a] T. Gomez-Diaz. - *Dynamic constructible closure.* Proceedings of the PoSSo Workshop on software, ed. Jean-Charles Faugère, Joël Marchand, Renaud Rioboo p.73-79, Paris (1995).

[GD95b] T. Gomez-Diaz. - *Let T be a triangle, is it isosceles ?* Actas del primer encuentro de algebra computacional y aplicaciones – EACA'95 (Santander) p.67-73 (1995).

[GD95c] T. Gomez-Diaz. - *Dynamic constructible closure programs.* Directory *dyn.tar* available at ftp://medicis.polytechnique.fr/pub/src/dynamic_evaluation (1995).

[GD97] T.Gomez-Diaz. - *On the computation of Jordan forms with parameters.* Preprint (1997).

[GLR93] M.J. Gonzalez-Lopez, T. Recio. - *The ROMIN inverse geometric model and the dynamic evaluation method.* The Scafi papers draft, p.117-141 (1991). See also A.M. Cohen, J.H. Davenport, A.J.P. Heck. - An overview of computer Algebra. Computer algebra in industry. Problem solving in practice. Proceedings of the 1991 SCAFI Seminar at CWI, Amsterdam, ed. A.M. Cohen, p.1-52 (1993).

[GVD93] L. Gonzalez Vega, D. Duval. - *Dynamic evaluation and real closure.* Proceedings of International Imacs Symposium on Symbolic Computation, ed. G. Jacob, N.E. Oussous, S. Steiberg. International Association for Mathematics and Computers in Simulation, p.17-21 (1993).

[JS92] R.D. Jenks, R.S. Sutor. - *Axiom, The Scientific Computation System.* NAG, Springer Verlag (1992).

[Laz91] D. Lazard. - *A new method for solving algebraic systems of positive dimension.* Discrete Applied Mathematics 33, p.147-160 (1991).

[Laz92] D. Lazard. - *Solving zero dimensional algebraic systems.* Journal of Symbolic Computation 15, p.117-132 (1992).

[Lec96] G. Lecerf. - *Dynamic evaluation and real closure implementation in Axiom.* Mémoire de D.E.A. (1996).

[MM97] M. Moreno Maza. - *Calculs de pgcd au dessus des tours d'extensions simples et resolution des systèmes d'équations algébriques.* PhD thesis, Université de Paris 6 (1997).

[RV99] T. Recio, M.P. Velez. - *Automatic discovery of theorems in elementary geometry.* Journal of Automated Reasoning 23, p.63-82 (1999).

[Wan98] D.M. Wang. - *Decomposing polynomial systems into simple systems.* Journal of Symbolic Computation 25, p.295-314 (1998).

[Watt et al.] S.M. Watt, P.A. Broadbery, S.S. Dooley, P. Iglio, S.C. Morrison, J.M. Steinbach, R.S. Sutor. - *Axiom, Library Compiler. User guide.* NAG (1994).