Examples of using Dynamic Constructible Closure

IMACS Symposium SC–1993

Teresa Gómez-Díaz*

Laboratoire d'arithmétique, calcul formel et optimisation (URA D1586)

Université de Limoges (FRANCE)

Abstract: We present here some examples of using the "*Dynamic Constructible Closure*" program, which performs automatic case distinction in computations involving parameters over a base field **K**. This program is an application of "*Dynamic Evaluation*" principle, which generalizes traditional evaluation and was first used to deal with algebraic numbers.

Keywords: dynamic evaluation, parameters, computer algebra, polynomial systems, automatic theorem proving

Introduction

The aim of this document is to present some examples of using the "*Dynamic Constructible Closure*" program that we have implemented in the computer algebra system Axiom [JS]. This program does automatic case distinction in computations involving parameters over a base field **K**.

This paper only concentrates in the study of some examples (computed by hand in [Du90]) and does not describe the algorithm itself. A complete description of the algorithm will be given in a forthcoming paper [Go].

The program of Dynamic Constructible Closure raised as an application of the "*Dynamic Evaluation*" principle: it generalizes traditional evaluation and was first used to deal with algebraic numbers [DDD]. This principle can be interpreted in sketch theory [DR].

Dynamic evaluation has been used to implement the *Dynamic Algebraic Closure of a Field* [DDD, DD] or the *Prime Field of Arbitrary Characteristic* [Du89].

For example, if you ask for the rank of the matrix:

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

according to all possible values of the complex parameter a, our program answers automatically:

```
[ value is 2 in case a = 0,
  value is 1 in case a = 1,
  value is 2 in case a /= 0, a /= 1 ]
```

(where `/=` means $\neq$). To get this result, we call essentially:

- the `Dynamic Constructible Closure` program that provides a field with as many parameters as you like.
- the `allCases` function, it is some "inference engine" which ensures that you get the complete answer.
- a standard `rank` function (implemented without any "parameter" notion).

This simple example already shows the importance of equality tests.

It is the first time that such a method to deal with parameters is implemented in a general way.

This paper is organised as follows: in next section we sketch some main points of the Dynamic Constructible Closure program. In the second section, we study polynomial system solving using this program and we apply this to

*This work is partially supported by CICyT PB/0379/C02/01 and POSSO Esprit/Bra 6846 compute singular points of plane curves. In the third section we study un example of automatic theorem proving in elementary geometry.

1 About dynamic constructible closure

1.1 Equality test

The main point 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 `true` from the values corresponding to `false`. This is called a "*splitting*".

1.2 Constraints

The possible values for parameters are expressed as "*constraints*". Next theorem shows the constraints manipulated in the program.

Theorem 1 *Let a be a parameter and* $\mathbf{K}$ *a base field. Constraints over a always are of one of these kinds:*

- `anyElement`*: there is not constraint over a (this means that a can take any value)*
- `exception`*: there is a constraint of type* $P_1(a) \neq 0, \ldots, P_k(a) \neq 0$ *with* $P_1, \ldots, P_k$ *monic univariate polynomials of positive degree with coefficients in* $\mathbf{K}$ *and coprime (this means that a can take any value that is not a zero of polynomials* $P_1, \ldots, P_k$*)*
- `algebraic`*: there is a constraint of type* $P(a) = 0$ *with* P *monic univariate polynomial of positive degree with coefficients in* $\mathbf{K}$ *(this means that a can take as value any zero of* P*).*

In addition, when the characteristic of $\mathbf{K}$ *is* 0*, we can suppose that polynomials* $P, P_1, \ldots, P_k$ *above are squarefree.*

1.3 Dynamic constructible closure

Let $\mathbf{K}$ be a base field. The *constructible closure* of $\mathbf{K}$ can be defined as a field $\hat{\mathbf{K}}$ which contains all the finitely generated extensions of $\mathbf{K}$ and such that is minimal for this property.
More constructively, $\hat{\mathbf{K}}$ (noted `CL` bellow) is a field that contains $\mathbf{K}$:

```
CL:= DynamicConstructibleClosure(K)
```

and is provided with a "generator of constants":

```
newElement: Symbol -> CL
```

More precisely, each call to this generator:

```
a:= newElement('a)
```

provides a new parameter a of $\hat{\mathbf{K}}$ whith a symbol `a` for its representation. In addition, $\hat{\mathbf{K}}$ has two operators:

```
areEqual: (CL,CL) -> Boolean
areDifferent: (CL,CL) -> Boolean
```

Their job is to impose or to forbid values for the parameters provided by `newElement`. In other words, they *impose constraints* over the parameters. The boolean result of these operators says whether a new constraint is (or not) compatible with the old ones.

1.4 Computations

A computation in $\hat{\mathbf{K}}$ is a combination of operations `+`, `-`, `*` and `inv`, from the constants `0`, `1` and the parameters (added with `newElement`), together with equality tests `=` and instructions `areEqual` and `areDifferent`, and usual control structures as "`if...then...else...`" or "`while...repeat...`". For example:

```
a:= newElement('a)
if areDifferent(a**4,1) then
  b:= a**2 + a
  if b = 2 then return 0
  else return 1/(b-2)
else return 0
```

With result:

```
[ value is 0 in case a = - 2,
                  1
  value is --------------- in case
           (a - 1) (a + 2)
  3    2
 a  + a  + a + 1/= 0, a/= 1, a/= -2 ]
```

Obviously, we do not get the case $a^4 - 1 = 0$.

2 Solving polynomial systems

A system of polynomial equations with coefficients in $\mathbf{K}$ can be "solved" (in the sense of triangularized, similary to [La]) by computations in the constructible closure of $\mathbf{K}$. Consider each indeterminate as a parameter introduced by `newElement` (which means that indeterminates must be ordered) and each equation as a constraint introduced by `areEqual` (which means that equations must be ordered).
For example, the system of "cyclic 4-roots" [BF]:

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

is expressed in the closure as:

```
a:= newElement('a)
b:= newElement('b)
c:= newElement('c)
d:= newElement('d)

areEqual(a+b+c+d,0) and
areEqual(a*b+b*c+c*d+d*a,0) and
areEqual(a*b*c+b*c*d+c*d*a+d*a*b,0)
and areEqual(a*b*c*d, 1)
```

While equations are introduced, the system automatically updates the constraints on parameters in order to keep them in the required form. In this way we get a union of *triangular* systems $S_1, \ldots, S_m$, but these systems are not polynomial in the usual sense. Indeed, if $a_1, a_2, \ldots, a_n$ denote the parameters (introduced in this order), each system S_j has the form:

$$\begin{cases} exp_1(a_1) \\ exp_2(a_2) \\ \ldots \\ exp_n(a_n) \end{cases}$$

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

- *any* a_i for the `anyElement` constraint,
- $P_{i,1}(a_i) \neq 0, \ldots, P_{i,k_i}(a_i) \neq 0$ for an `exception` constraint,
- $P(a_i) = 0$ for an `algebraic` constraint,

with $P_{i,1}(a_i), \ldots, P_{i,k_i}(a_i), P(a_i)$ in $\mathbf{K}(a_1, \ldots, a_{i-1})[a_i]$, monic, of positive degree in a_i, and squarefree if $\mathbf{K}$ has characteristic 0. In addition, the polynomials $P_{i,1}(a_i), \ldots, P_{i,k_i}(a_i)$ are coprime. The non-trivial denominators which appear in coefficients of these polynomials involve only the parameters a_j of `exception` type (with $j < i$).
We get the solutions of the given system by collecting the solutions of the systems $S_1, \ldots, S_m$. There is no redundancy: there is no solution common to two of the systems $S_1, \ldots, S_m$. We remark that the `exception` constraints appear during computations since no `areDifferent` is used in the formulation of the problem.
In this simple example the solution is:

```
[ value is false in case d = - b and
c = 0 and any b and a = 0,

  value is true in case d = - b and
             2   1
c = - a and b  - -- = 0 and a /= 0 ]
                  2
                 a
```

This means that there is no solution if $a = 0$, and for each non-zero value of a there are exactly two solutions:

$$(a\ ,\ b = \pm\frac{1}{a}\ ,\ c = -a\ ,\ d = -b)$$

This method can be compared to a "complex" version of cylindrical algebraic decomposition [ACC], or to the characteristic sets method of Ritt-Wu [Ch, BCK]. But it is fairly different from the Gröbner basis method [Bu].

Singular points of a curve

To compute the singular points of a curve is an application of solving polynomial systems. Given a plane curve defined by a polynomial $P(x,y) \in \mathbf{K}[x,y]$ with derivatives $Px(x,y) = \frac{\partial P(x,y)}{\partial x}$ and $Py(x,y) = \frac{\partial P(x,y)}{\partial y}$, let us compute in the constructible closure:

```
x:= newElement('x)
y:= newElement('y)

areEqual(P(x,y),0) and
areEqual(Px(x,y),0) and
areEqual(Py(x,y),0)
```

We can go further and consider a family of curves which depends on one or more parameters and compute the singular points of these curves according to values of parameters, as in the next example. We consider the family of curves

$$P(a,b,x,y) = y^2 - x(x-a)(x-b(a+1))$$

which depends on parameters a and b. The program in the closure is:

```
a:= newElement('a)
b:= newElement('b)
x:= newElement('x)
y:= newElement('y)

P(a,b,x,y):= y**2-x*(x-a)*(x-b*(a+1))
Px(a,b,x,y):= -3*x**2+((2*a+2)*b+2*a)
  *x+(-a**2-a)*b
Py(a,b,x,y):= 2*y

areEqual(Py(a,b,x,y),0) and
areEqual(Px(a,b,x,y),0) and
areEqual(P(a,b,x,y),0)
```

The description of singular points is:

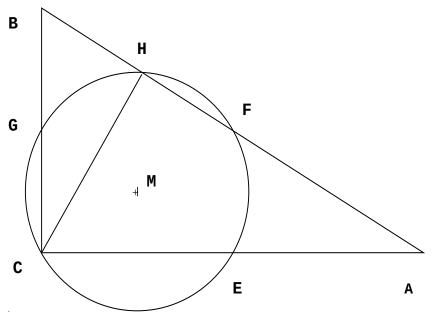
```
[ y=0 and x=0 and any b and a=-1,
  y=0 and x=0 and b=0 and a=0,
  y=0 and x=0 and b/=0 and a=0,
                               2    a
  y=0 and x=(a+1)b and b - ----- b=0
                                  (a+1)
  and a/=-1, a/=0 ]
```

Note that when $a = 0$, the cases $b = 0$ and $b \neq 0$ are considered separately. It has a geometric sense: the behaviour of the curve at the singular point is not the same. If $a = 0$ and $b \neq 0$, the point $(0,0)$ is an ordinary double point, and if $a = 0$ and $b = 0$, the point $(0,0)$ is a cusp.

3 Automatic proving in elementary geometry

In this kind of problems, `exception` constraints are 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. Then our method is similar to the Ritt-Wu method [Ch, BCK] but more powerfull because it is possible to avoid degenerate cases.

For example, consider the next 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 $\mathbf{Q}$. Let M denote the center of Γ. We choose a coordinate system adapted to the problem and we give a name to coordinates of every point in this coordinate system. For example:

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

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

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

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:

$$a_3 = \frac{1}{2}a_1\,,\ a_4 = \frac{1}{2}a_1\,,\ a_5 = \frac{1}{2}a_2\,,\ a_6 = \frac{1}{2}a_2$$

The center M of the circle Γ is given by the conditions $\overline{EM} = \overline{FM}$ and $\overline{EM} = \overline{GM}$, 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:

$$\frac{1}{a_1}x + \frac{1}{a_2}y = 1$$

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

$$\frac{1}{a_1}a_9 + \frac{1}{a_2}a_{10} = 1\,,\ -a_1a_9 + a_2a_{10} = 0$$

Finally, the conclusion of the theorem is $\overline{EM} = \overline{HM}$, that is

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

The corresponding program is:

```
 a1:= newElement('a1)
 a2:= newElement('a2)
 a3:= newElement('a3)
 a4:= newElement('a4)
 a5:= newElement('a5)
 a6:= newElement('a6)
 a7:= newElement('a7)
 a8:= newElement('a8)
 a9:= newElement('a9)
 a10:= newElement('a10)

 e0:=(a7-a3)**2
 e1:=e0+a8**2-(a7-a4)**2-(a8-a5)**2
 e2:=e0+a8**2-a7**2-(a8-a6)**2
 e3:=-a1*a9 + a2*a10
 e4:=e0+a8**2-(a7-a9)**2-(a8-a10)**2

 if areDifferent(a1,0) and
   areDifferent(a2,0) and
   areEqual(a3,(1/2)*a1) and
   areEqual(a4,(1/2)*a1) and
   areEqual(a5,(1/2)*a2) and
   areEqual(a6,(1/2)*a2) and
   areEqual(e1,0) and
   areEqual(e2,0) and
   areEqual((1/a1)*a9+(1/a2)*a10-1,0)
   and areEqual(e3,0)
 then return (e4 = 0)
 else return "failed"
```

and the result:

```
 [ value is "failed" in case
       1
a10=- ----a2 a9+a2 and any a9 and
      (a1)
    1           1           1
a8= -a2 and a7= -a1 and a6= -a2 and
    4           4           2
    1           1           1
a5= -a2 and a4= -a1 and a3= -a1 and
    2           2           2
  2   2
a2 +a1 = 0 and a1/= 0,

   value is true in case
         2                 2
       a1 a2             a1 a2
a10= --------- and a9= --------- and
        2   2             2   2
     (a2 +a1 )         (a2 +a1 )
    1           1           1
a8= -a2 and a7= -a1 and a6= -a2 and
    4           4           2
    1           1           1
a5= -a2 and a4= -a1 and a3= -a1 and
    2           2           2
          2   2
a2/= 0, a2 +a1 /= 0 and a1/= 0 ]
```

In the result, there are two different cases. They appear when the system tries to impose the constraint `areEqual(e3,0)`. At this moment the program asks the question "$a_2^2+a_1^2 = 0$?". There are two possibilities for the answer.

- If the answer is `true`, the system updates constraints over a_2 and a_1, and gets $a_2^2 + a_1^2 = 0$ and $a_1 \neq 0$. To impose `areEqual(e3,0)` becomes to impose `areEqual(a1*a2**2,0)`, but it is not compatible with current constraints. Thus `areEqual(e3,0)` returns `false` and the system returns `"failed"`. This means that it is impossible to build the figure in this case. However, this cannot happen if a_2 and a_1 are real.
- If the answer is `false`, the system updates constraints over a_2 and a_1, and gets $a_2 \neq 0$, $a_2^2 + a_1^2 \neq 0$ and $a_1 \neq 0$. To impose `areEqual(e3,0)` becomes to impose the value $\frac{a_1a_2^2}{a_2^2+a_1^2}$ for a_9, and this is compatible with the `anyElement` constraint over a_9. Now the figure is built. Then the system asks whether "`e4 = 0`" and answers `true`, whithout any additional spitting. The theorem is proved.

Conclusion

We have presented here the program "*Dynamic Constructible Closure*". Thanks to it:

Given a computation involving parameters we are able to get every solution according to the values of the parameters, by means of dynamic evaluation techniques.

Therefore it gives a new general and automatic way to deal with parameters (but closed to a "by hand" manipulation). It can be used in a simple and natural way, as can be seen from our examples.
We have shown several applications of our program. Other applications are planned; for example robotics, where some problems were already studied from this point of view, but "by hand", in [GR].
Acknowledgements. The original idea of this work is due to D. Duval and I gratefully acknowledge the help she has given me in understanding it and extending it.

References

[ACC] Arnon D.S., Collins G. E. and McCallum S. – Cylindrical algebraic decomposition I: the basic algorith, and II: an adjacency algorithm for the plane, in: *SIAM J. of Computing 13*, (1984).

[BCK] Buchberger B., Collins G. E. and Kutzler B. – Algebraic Methods for Geometry Reasoning, in: *Ann. Rev. Comput. Sci. 3*, (1988).

[BF] Björck G. and Fröberg R. – A Faster Way to Count the Solutions of Inhomogeneus Systems of Algebraic Equations, with Applications to Cyclic n-roots, in: *J. Symbolic Computation 12, N.3* (1991).

[Bu] Buchberger B. – Gröbner Bases: an Algorithmic Method in Polynomial Ideal Theory, in: *Recent Trends in Multidimensional System Theory*, Bose (ed.), (Reidel, 1985).

[Ch] Chou S. C. – *Mechanical Geometry Theorem Proving* (Reidel Publishing Company, 1988).

[DDD] Della Dora J., Dicrescenzo C. and Duval D. – About a New Method for Computing in Algebraic Number Fields, in: *Eurocal'85, vol.2*, Goos, G. and Hartmanis, J. (eds.), (Springer Lecture Notes in Computer Science 204, 1985) pp. 289-290.

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

[Du89] Duval D. – Computations in fields of arbitrary characteristic, in: *Computers and Mathematics*, Kaltofen E. and Watt S.M. (eds.), (Springer, 1989) pp. 321-326.

[Du90] Duval D. – Calculs avec discussion automatique : Description et applications, in: *Publications du Département de Mathématiques de Limoges*, (1990) 20 pp.

[DR] Duval D. and Reynaud J. C. – Sketches and Computation, Rapport de Recherche RR871-I-IMAG-123 LIFIA, (1991) 71 pp.

[Go] Gómez-Díaz T. – Calculs avec discussion automatique : Clôture Constructible Dynamique, in preparation.

[GR] González-López M. J. and Recio T. – The ROMIN inverse geometric model and the dynamical evaluation method, in: *Computer Algebra in Industry*, Cohen A. M. (ed.), (John Wiley & Sons Ldt., 1993) pp. 117-141.

[JS] Jenks R. D. and Sutor R. D. – *AXIOM, The Scientific Computation System*, (Springer-Verlag, 1992).

[La] Lazard D. – A new method for solving algebraic systems of positive dimension, Rapport LITP 87-77, (1989) 17 pp.