A Normal Form for Function Rings Of Piecewise Functions

Martin von Mohrenschildt
Symbolic Computation Group
University of Waterloo, Waterloo, Canada
mmohrenschildt@daisy.uwaterloo.ca

March 7, 1996

Abstract

Computer algebra systems often have to deal with piecewise defined functions, for example, the absolute value function. We present a new approach to this kind of problem. This paper provides a normal form for function rings containing piecewise functions. We give a compiled rule system to compute the normal form of a function. With a normal form, we can decide equality in our function ring. In this ring, we can define supremum and infimum of two functions. In fact we show that the function ring is a compiled lattice. We present a method to solve equalities and inequalities in this function ring.

1 Introduction

In this paper, we consider function rings with one real variable extended with a "piecewise" construct, meaning that according to a decision depending on the variable, a certain piece of a function is chosen. Such functions are, for example, the absolute value function $|x|$ or the signum function, and also include functions such as the supremum or infimum of two functions like $\sup(x^2 - 2, x)$.

The main property of our approach is the possibility of defining a normal form.

$$f_0(x) + \sum_{n=1}^{N} f_i(x)\, step(x - a_i) + h_i(x)\, step(-x + b_i)$$

Here, $f_i(x)$ and $h_i(x)$ are from the ground function ring, a_i, b_i are numbers and it holds that if $a_i = b_j$ then $i = j$ same for h_i (collected) and $h_i(b_i) \neq 0$. For this normal from we give a compiled rule system. This means rules transform any term into its normal form after a finite number if steps. Our approach can be extended to define closed form solutions of differential equations with piecewise coefficients [Moh94]. An approach for normal forms of composition in function fields can be found in [Aber90].

The theory takes place in a function ring extended with the function *step*. $step(x) = \begin{cases} 0 & x \leq 0 \\ 1 & \text{otherwise} \end{cases}$. We give an algorithm to convert a function in piecewise notation to a *step* expression and one to convert a *step*-expression in normal form back into the piecewise notation e.g. $piecewise(x > 0, f(x), g(x)) = g(x) + step(x)(g(x) - g(x))$. $abs(x) = -x + 2x\, step(x)$.

We can express supremum and infimum in terms of *step* functions. It turns out that the function ring is a lattice.

Using the above properties it is possible to solve equalities and inequalities in the function ring. To solve inequalities, we profit from the lattice structure in the sense that the solution of $f(x) > 0$ is translated into $step(f(x)) = 1$. We only have to compute the normal form to get the solution.

2 The Normal Form

The ground ring of our function ring is a ring over a ordered real algebraically closed field of constants. We use the polynomials over the real-part of the algebraic closure of the rational numbers, meaning that the polynomials can be split into linear factors with real roots contained in the ground field. This ring is then extended with the function symbol *step*. An expression of this type is called an *step*-expression. For example, $step(x) - step(2-x)(x-5)$ is a *step*-expression and so are $step(step(x)x - x(x-3))$ and $step(x-2)\, step(x) - step(x-3)^2 + x$.

Note that the function field extended with *step* is not a field but a ring since we have zero-divisors like in every conditional function ring. With this

construction, it is possible to express piecewise functions in one real variable. This is not possible with the recursive real numbers [Rice54], but for example in the set of the algebraic real numbers.

To discuss normal forms, let us give the definition of a normal form:

Definition 2.1 *A normal form is given by a operator mapping functions into functions with the properties:*

$$normal(normal(f)) = normal(f)$$

$$f = g \ \textit{iff}\ normal(f) \equiv normal(g)$$

The functions in our function ring are sometime given by a piecewise-constructor. We give an algorithm to convert a conditional expression into its *step* representation, section 5. For example,

$$piecewise(x < a, f_1(x), x < b, f_2(x), x > c, f_3(x))$$

converts to $step(x-a)\ f_1(x) + step(x-b)f_2(x) + step(x-c)f_3(x)$.

The rule system

Let $f(x), g(x)$ denote (*step*-free) functions of the ground ring and a, b be numbers. Our rule system is:

$$step(ax-b) \rightarrow step\left(\frac{ax-b}{|a|}\right) \quad (1)$$
$$\text{if } a \neq -1, 1, 0$$

$$f(x)\, step(-x+a) \rightarrow f(x)(1 - step(x-a)) \text{if} f(a) = 0 \quad (2)$$

$$step(x-a)\, step(x-b) \rightarrow step(x-a) \text{ if } a \geq b \quad (3)$$

$$step(-x+a)\, step(x-b) \rightarrow step(x-a) - 1 + step(-x+b) \quad (4)$$
$$\text{if } a \geq b \text{ else } 0$$

$$step(-x+a)\, step(-x+b) \rightarrow step(-x+b) \text{ if } b \leq a \quad (5)$$

$$step(step(\pm x \mp a)f(x) + g(x)) \rightarrow step(\pm x \mp a)\, step(f(x) + g(x)) \quad (6)$$
$$+(1 - step(\pm x \mp a))\, step(g(x))$$

$$step(p(x)) \rightarrow \sum_{i=1}^{N} a_i\, step(d_i(x - b_i)) \quad (7)$$
$$d_i = \pm 1 \text{ by algorithm } [1]$$

$$step(a) \rightarrow a \text{ not depending on x is}$$
$$1 \text{ if } a > 0 \text{ else } 0$$

Definition 2.2 *For piecewise expressions we define the normal form:*

$$f_0(x) + \sum_{n=1}^{N} f_i(x)\, step(x - a_i) + h_i(x)\, step(-x + b_i)$$

where $f_i(x)$ and $h_i(x)$ are from the ground function ring, a_i, b_i are numbers and it holds that if $a_i = b_j$ then $i = j$ same for h_i (collected) and $h_i(b_i) \neq 0$

We give some examples of normal forms using the absolute[2] value function:

Example 2.3

$$normal(abs(x)) = -x + 2x\, step(x)$$

$$normal(abs(2 - abs(x)) = -2 - x + (4 + 2x)\, step(2 + x) -$$

$$2x\, step(x) + (-4 + 2x)\, step(x - 2)$$

$$normal(abs(x)^2) = (-x + 2x\, step(x))^2 =$$

$$= x^2 - 4x^2\, step(x) + 4x^2\, step(x) = x^2$$

Theorem 2.4 *The above rule system is a complete rule system and provides the defined normal form.*

Proof We prove this theorem in two steps.

1) We will define a measure Q of the distance of an expression to its normal form and show that applying a rule reduces this measure. We will show that if the term is in normal form, then no more rules can be applied and if any rule can be applied, the term is not in normal form. For expressions of the ground ring, polynomials, we use some canonical normal form.

2) We will show that it does not matter in which order we apply the rules. (Confluence.) Any application order of the rules will yield the same end-result.

[1] The algorithm is given at the end of the next proof

[2] We show later how to convert the abs function into a *step* expression

Proof of (1). Let us define $Q(\tau) =< n, m+k+l >$ as a measure of the distance of an expression τ to its normal form. If the expression does not contain a *step*, then $Q(\tau) =< 0, 1 >$.

Otherwise, n, m, k, l are defined as follows.

Integer n denotes the product of the number of *step*'s appearing as inner argument of *step*'s times the depth, defined as follows:

$$\begin{aligned} depth(\tau) &:= \quad 0, \text{ for } \tau = x \text{ or } \tau = \text{number} \\ depth(\tau\mu) &:= \quad \max(depth(\tau), depth(\mu)) \\ & \qquad \text{similarly, for } + \text{ and } - \\ depth(step(\tau)) &:= \qquad depth(\tau) + 1 \end{aligned}$$

For example, the depth of $step(x)$ is 1 and that of $step(1 + step(x)) + x$ is 2. Then n for $step(step(x)) + step(step(x) + step(x-2))$ is $2 * 2 * 2 = 12$.

Integer m denotes the product of the number of multiplications of *step*'s with *step*'s in every product of the expression. For example for $step(x-2)\, step(x)\, step(x-8) + step(x)\, step(x-2)$, the m is $3 * 2 = 6$.

Integer k is the product of all the numbers of x appearing inside a *step*, times the absolute values of it's coefficient. (For $step(-2x^2 + x) + step(x^2)$ this is $(3 * 2) * 2 = 12$. We write x^3 for xxx.) If the expression contains no *step*, then $k = 1$.

Integer l is the number of subexpressions of the form $f(x)\, step(-x+a)$ where $f(a) = 0$.

We show that for an expression in normal form, n is 0, m is 0, l is 0, and k is 1. We define an ordering of Q using the lexicographical ordering of the pairs $< a, b >$.

$$< a, b > \prec < a', b' > \leftrightarrow \begin{cases} a < a' & \text{or} \\ a = a' & b < b' \ . \end{cases}$$

Let $g(x)$ be an expression with *step* in normal form.

$$g(x) = f_0(x) + \sum_{n=1}^{N} f_i(x)\, step(x - a_i) + h_i(x)\, step(-x + b_i)$$

We show that $Q(g(x))$ is $< 0, 1 >$.
Rule 1 can not be applied since no x inside a *step* has a coefficient other than 1 or -1. Hence, k is 1. Rule 2 can not be applied since in the normal form we have $h_i(b_i) \neq 0$. Hence, $l = 0$. Rules 3-5 can not be applied since in the normal form an expression has no product of a *step* with a *step*. Hence, m is 0. Rule 6 can not be applied since an expression in normal form has no *step* inside the arguments of a *step*. Hence, n is 0. Finally, rule 7 can not be used since all arguments of *step* are linear in x. It follows that $Q =< 0, 1 >$. We also showed that if $g(x)$ is in normal form, none of the rules can be applied.

Now we have to show that for every rule in the system:

$$Q(\text{left-hand side}) \succ Q(\text{right-hand side})$$

Applying rule 6 reduces n and therefore Q:

$$step(f(x)\, step(\pm x \mp a) + g(x)) \rightarrow$$
$$step(g(x))(1 - step(\pm x \mp a)) + step(f(x) + g(x))\, step(\pm x \mp a).$$

The left-hand side has one $step(.)$ as an argument of a *step* and some x's. Hence $n > 1$. Looking at the right-hand side we see that all *step*'s contain fewer arguments than before and the depth is also reduced. Hence

$$n(\text{left-hand side}) \succ n(\text{right-hand side})$$

for all inner rules. Even if m or k get bigger, Q gets smaller (lexicographical ordering).

The application of the multiplicative rules 3,4,5 reduce n.
We show this in an example using rule 3:

$$step(x - a)\, step(x - b) \rightarrow$$

$$\text{if } a < b \text{ then } step(x - b) \text{ else } step(x - a).$$

The left-hand side m is 1. Looking at the right-hand side, we see that m is reduced to 0. This holds for all multiplicative rules. Hence,

$$m(\text{left-hand side}) \succ m(\text{right-hand side})$$

for all multiplicative rules. Since n is not changed, Q is getting smaller.

Rule 7 reduces the number of x's occurring inside a *step*, which reduces k and therefore Q. Rule 1 normalizes the coefficient of x to 1 or -1, so also reduces k.

Finally, rule 2 reduces l and leaves n, m, k alone. Hence, Q gets smaller.

We showed that if Q of an expression τ is $< 0, 1 >$, then τ is in normal form. Applying the rules reduces Q and, therefore, ends after a finite number of steps.

Proof of (2) We have to show confluence, since some rules can have overlapping usage. The first rule can only be applied once to every $step(ax - b)$ with $a \neq 1,\ a \neq -1$. The second rule can only be applied once to every $step(-x + a)f(x)$. Hence, there is no overlap here. For the rules 3,4,5, we observe that an expression $step(x - a)\, step(x - b)\, step(-x + c)\, step(-x + d)$ is reduced to its unique normal form and that expressions of the form $step(step(x - a)f(x) + g(x))\ g(x)$ possible continuing a *step* can only be reduced to a single normal form. (The actual cases analysis for these two last rules are not detailed here).

We have to examine expressions of the form $step(f(x)\, step(x-a)+g(x)\, step(x-b) + h(x))$. There are two rules that overlap on this expression. Calculating its normal form in the two different ways always gives (assuming $a < b$): $step(h(x)) - s(x - a)\, step(h(x)) + step(x - a)\, step(f(x) + h(x)) - step(f(x) + h(x))\, step(x - b) - step(x - b)\, step(f(x) + g(x) + h(x))$.

Last we have to show that $step(p(x))$, where $p(x)$ is a polynomial, can be reduced to a sum of *step*'s.

We give an algorithm to do this:
Let $a_1, a_2, \cdots, a_n$ be the list of the real roots of $p(x)$ in ascending order $a_1 \leq a_2 < \cdots < a_n$ such that if a_i has even multiplicity then a_i appears twice in the list, otherwise a_i appears once in the list. Complex roots are

ignored. If the sign of the highest coefficient of the polynomial p is positive and the degree is odd or the highest coefficient is negative and the degree is even, then the polynomial approaches the x-axis from below, otherwise if approaches the x-axis from above. Initially, we set $out := 0; i := 1$.
Now, if we are above the x-axis, then $out := out + step(-x + a_1)$ and $i := i+1$, otherwise we start with the main loop below.
Loop:
While there are still two roots do
If $a_i = a_{i+1}$ (root with even multiplicity), do nothing, otherwise $out := out + step(x - a_i) - step(-x + a_{i+1})$ and $i := i + 2$. End of the loop.
If there is still one more a_i, it is a_n and $out := out + step(x - a_n)$.
This *out* is the desired sum of *step*'s. ∎

3 Lattice structure

With *step*, we can express the supremum and the infimum of two functions. We show that our function ring is a lattice, which means supremum and infimum of two functions of the ring belong to the ring. Hence, supremum and infimum represent maximum and minimum. For example, we can compute $\sup(x, -x)$ which is the piecewise function $-x$ for $x \leq 0$ and x for $x > 0$, in other words, the absolute value function.

Definition 3.1 *We define the* sup *and the* inf *of two functions* f, g*:*

$$\sup(f,g) = f\,step(f-g) + g\,step(g-f) + \frac{f+g}{2}(1 - step(f-g) - step(g-f))$$

$$\inf(f,g) = b\,step(f-g) + f\,step(g-f) + \frac{f+g}{2}(1 - step(f-g) - step(g-f))$$

Theorem 3.2 *With the* sup *and* inf *defined above, the function field becomes a lattice.*

Proof We have to show that $\sup(f(x), f(x)) = f(x)$, $\sup(f(x), g(x)) = \sup(g(x), f(x))$, and $\sup(f(x), \sup(g(x), h(x))) = \sup(\sup(f(x), g(x)), h(x)))$ for any $f(x), g(x), h(x)$.

$\sup(f(x), f(x)) = f(x)\, step(f(x)-f(x)) + f(x)\, step(f(x)-f(x)) + \frac{f(x)+f(x)}{2}(1 - step(f(x)-f(x)) - step(f(x)-f(x))) = f(x)$.
$\sup(f(x), g(x)) = f(x)\, step(f(x)-g(x)) + g(x)\, step(g(x)-f(x)) + \frac{f(x)+g(x)}{2}(1 - step(f(x)-g(x)) - step(g(x)-f(x))) = sup(g(x), f(x))$.
To show transitivity, we have to work harder. We need to use identities like $step(f)\, step(-f) = 0$ for any f. Because of the technical nature of this computation, it is not presented here. $\sup(f(x), \sup(g(x), h(x))) = \cdots = \sup(\sup(f(x), g(x)), h(x))$
The supremum and infimum are compatible with the ordering relation of the ground ring. That is, if $a < b$ then $sup(a, b) = a$. $sup(a, b) = a\, step(a-b) + b\, step(b-a) + \frac{a+b}{2}(1 - step(a-b) - step(b-a)) = a$ ∎

Example 3.3

$$\sup(-x, x) = 2x\, step(x) - x$$

$$sup(x^2-2, x) = \left(x - x^2 + 2\right) step(x+1) + \left(x - x^2 + 2\right) step(x-2) - x + 2\, x^2 - 4$$

4 Computing Zeros and Solving Inequalities

If the zeros of the underlying function ring are computable, then the zeros in the ring extended by *step* are computable. Since we use polynomials over the real algebraic closed numbers in this paper, the zeros of these polynomials are computable.

Let us give two examples. First, $x^2 - 4\, step(x) = 0$. This equation has the solution $x = 2, x = 0$. $x = -2$ is not a solution since $step(-2) = 0$. Second, $step(2 + x) + step(x - 2) = 0$ represents, already, it's own solution. Converting this function to it's piecewise representation, the range where this function is 0 is the range for x for which the function is 0.

Theorem 4.1 *Let D be a function ring, the constants of D are real algebraic closed, and the solutions of the equations $f \in D = 0$ be computable. Then the zeros in the with function ring extended by step are computable.*

Theorem 4.2 *Let D be a function ring, the constants of D are real algebraic closed, and the solutions of the equations $f \in D = 0$ be computable. Then the solutions of inequalities in the function ring extended by step are computable.*

First, we prove theorem 4.2.

Proof Let us examine the the equation $\tau > 0$. This equation can be written as $step(\tau) = 1$ and we use theorem 4.1. The inequality $\tau >= 0$ can be written as $step(-\tau) = 0$. Now we use theorem 4.1 to solve these equations. ∎

Next, we prove theorem 4.1.

Proof Let τ be the equation to be solved. We examine the normal form of

$$\tau = f_0(x) + \sum_{n=1}^{N} f_i(x)\, step(x - a_i) + h_i(x)\, step(-x + b_i)$$

We distinguish two cases:

If in the normal form f_i and h_i are constant, then we are done. The solutions is the union of ranges where this piecewise constant function is 0. This can be done by our *step* to piecewise conversion algorithm 5.

Otherwise, let L be the list of all zeros of τ which can be found by substituting every *step* of τ by 0 or 1. This means that we have to solve (in the worst case) 2^N equations which are *step* free. Having all these zeros we check if they are zeros of τ by substituting them into τ. We find all the zeros this way. ∎

Example 4.3 *Let us solve* $x^2 - 4\, step(x) = 0$. *We have one step hence we first have to solve the equations* $x^2 - 4 = 0$ *and* $x^2 = 0$. *These give us* $2, -2, 0$. *Now we check these solutions and find that* 2 *and* 0 *are the solutions.*

Now we solve $x^2 - step(x)4 > 0$. *According to 4.2 we have to solve* $step(x^2 - step(x)4) = 1$. *Compute the normal form:* $step(-x) + step(x - 2)$. *This represents the solution,* $step(-x) + step(x - 2)$ *is 1 for* $x < 0$ *or* $x > 2$. *Hence the solution is* $x \in (-\infty, 0) \cup (2, \infty)$

5 Converting piecewise expressions

In this section, we describe the process of converting a piecewise-expression into an *step*-expression and vice versa.

Let us start with a definition for piecewise$(c_1, f_1, c_2, f_2, ...)$. This piecewise should be read like an if .. elif .. else .. fi statement. If the first condition

is true, then the piecewise expression evaluates to the first function f_1, otherwise, if the second condition is true, then f_2, and so on. It is possible to give a last argument without a condition for the else case. Note that if no condition is true and there is no else case, then the result is 0 by definition.

To convert a piecewise expression into it's *step* representation is straight forward. For every condition in the piecewise-expression we construct a *step* expression which is 1 if and only if the condition is satisfied. We can describe the algorithm by a conversion function S.

The conversion function S is given by:

$$\begin{aligned}
S(x > a) &\rightarrow step(x-2) \\
S(x \leq a) &\rightarrow 1 - step(x-2) \\
S(x \geq a) &\rightarrow step(-x+a) \\
S(x < a) &\rightarrow 1 - step(-x+a) \\
S(\text{not} E) &\rightarrow 1 - step(S(E)) \\
S(E \text{ and } B) &\rightarrow step(S(E))\, step(S(b)) \\
S(E \text{ or } B) &\rightarrow step(S(E) + S(B)) \\
S(piecewise(E, f_1, B, f_2, \cdots) &\rightarrow S(E)(1 - S(B))f_1 + \\
&\quad + S(B)(1 - S(E))f_2 + \cdots
\end{aligned}$$

Let us apply S to the absolute value function, for example. $piecewise(x > 0, x, x \leq 0, -x) = S(x > 0)(1-S(x \leq 0))x + (1-S(x > 0))S(x \leq 0)(-x) = = step(x)(1-step(-x))x - (1-step(x))\, step(-x))x = step(-x)x = 2x\, step(x) - x$.

To convert a *step* expression into a piecewise expression, we compute the normal form of the *step* expression. Sort the a_i and b_i in increasing order. For every a_i or b_i, create a new condition. If $a_i = b_i$, we get a condition of the form $x = a_i$ in the piecewise-expression. Looking at the normal form, we have three different possibilities for a discontinuity at a point a contained in the normal form:

1. $f(x)\, step(x-a)$ corresponds to $x \leq a, f_1(x) x > a, f_2(x)$
2. $f(x)\, step(x-a) + g(x) step(-x+a)$ corresponds to $x < a\ f_1(x), x = a\ f_2(x), x > a\ f_3(x)$
3. $f(x)\, step(-x+a)$ corresponds to $x < a, f_1(x) x \geq a, f_2(x)$

Having all these conditions we have to determine the functions f_i for those pieces.

Example 5.1 *Let us convert* $step(-x+2) + x\,step(x) - step(x-3)$ *to a piecewise expression: the "critical points" are* $-2, 0, 3$. *Hence, we get* $piecewise(x \leq -2, f_1(x), x \leq 0, f_2(x), x \leq 3, f_3(x), x > 4 f_4(x))$ *Then we have to find* $f_i(x)$. *We get* $piecewise(x \leq -2, 1, x \leq 0, 0, x \leq 3, x, x > 2, x-1)$.

Let us convert $step(x) + 2step(-x) + step(x-2)$ *to a piecewise. Since the normal from has* $step(x)$ *and* $step(-x)$ *we are in the second case for the* 0 *and in the first case for the* $step(x-2)$. *Hence, we get the conditions* $x < 0, x = 0, x \leq 2, x > 2$. *This corresponds to the piecewise function* $piecewise(x < 0, 2, x = 0, 0, x \leq 2, 1, x > 2, 2)$.

6 Conclusion

The idea of converting piecewise functions into *step*-expressions has some advantages. We were able to define a normal form and with this we could decide equality. Also, we could define a lattice structure for piecewise conditional expressions.

References

[Abe90] K. Aberer (1990). Normal Forms in Function Fields. *Proceedings ISSAC '90*, 1-7.

[Kan83] R. P. Kanval (1983) Generalized functions: Theory and Technique. *Academic Press.*

[Rice54] H. G. Rice (1954) Recursive Real Numbers: Proc. Amer. Math. Soc. 5,784-791.

[Moh94] M. von Mohrenschildt (1994) Discontinuous Ordinary Differential Equations. *PhD-Thesis, Dept. of Math., ETH Zurich, Nr. 10768.*