Implementation in ACL2 of Well-Founded Polynomial Orderings

I. Medina-Bulo[*], F. Palomo-Lozano[*], and J. A. Alonso-Jiménez[†]

[*]Dept. de Lenguajes y Sistemas Informáticos (Univ. de Cádiz)
Escuela Superior de Ingeniería de Cádiz. C/ Chile, s/n. 11003 Cádiz. Spain.
{inmaculada.medina , francisco.palomo }@uca.es
[†]Dept. de Ciencias de la Computación e Inteligencia Artificial (Univ. de Sevilla)
Facultad de Matemáticas. Avda. Reina Mercedes, s/n. 41012 Sevilla. Spain.
jose-antonio.alonso@cs.us.es

Abstract

This paper presents how the development of a polynomial ordering and the verification of its properties can be fit in the framework of ACL2. The key result is the well-foundedness of a polynomial ordering, which is proved by a proper ordinal embedding. Normalized polynomials have been formalized to achieve this. The motivation for this work is to serve as a basis for proving the termination of certain reduction relations on polynomials in ACL2.

1 Motivation

The abstract notion of reduction can be modeled by using a unary function, *red*, which tries to simplify its argument with respect to a given strict partial order $<$. If the element cannot be simplified it is said to be *irreducible* or "in normal form" and *red* returns it unchanged. Otherwise, the element is *reducible*. The characteristic property of such a function is:

$$\mathit{red}(p) \neq p \implies \mathit{red}(p) < p.$$

In this context, *red* is called a *reduction function*. This framework can be translated into ACL2 [4, 5] by encapsulating three functions: the reduction function, the order and an ordinal measure.

The (unbounded) exponentiation of the reduction function *red* is recursively defined by the following function, *red**.

$$\mathit{red}^*(p) = \begin{cases} p, & \mathit{red}(p) = p \\ \mathit{red}^*(\mathit{red}(p)), & \mathit{red}(p) \neq p. \end{cases}$$

Unfortunately, this function is not guaranteed to terminate. However, termination can be assured whenever $<$ is well-founded. This is what would precisely enable ACL2 to admit *red** under its definitional principle.

Therefore we are mainly interested in reduction functions related to well-founded orders. In fact, our main concern is the formalization of certain reduction functions on polynomials arising in Buchberger's algorithm for Gröbner bases computation. Thus, we must develop a well-order $<$ on polynomials. The goal of this work is to explore different alternatives to achieve this.

In Sect. 2 the process of obtaining a normalized representation for polynomials from an unnormalized representation is explained. The reason for this is twofold. First, a previous work by the authors on polynomials in which an unnormalized representation was employed [6] can be reused. Finally, and more importantly, a normalized representation makes the rest of the work considerably easier.

As we have explained above, in order to accommodate polynomials to our notion of reduction, the monomial ordering must be extended to polynomials. This is done in Sect. 3. We will see that the underlying monomial embedding into ϵ_0-ordinals must be taken with great care.

2 Normalized Polynomials on Top of an Unnormalized Layer

In a normalized representation for polynomials [3, 10], a unique representation is associated with each polynomial. A polynomial is said to be in normal form if it satisfies the following conditions:

1. Its monomials are in strictly decreasing order.

2. It does not contain a null monomial.

Using this approach, which predates Computer Algebra Systems, very efficient algorithms can be obtained. These algorithms operate on normalized polynomials and return normalized polynomials.

The main advantage of this method from the verification point of view stems from the fact that the semantic equivalence between polynomials becomes syntactic equality. It is not difficult to implement operations using this representation. However, problems appear when trying to prove such "elemental" properties like associativity of addition.

This led us to use another representation, less efficient from an algorithmic perspective, but more appropriate to verify the properties. We think that this is a paradigm of the trade-off between algorithmic efficiency and verification simplicity: it is clear that in an unnormalized setting, addition is just concatenation of the monomial lists, whose associativity is trivial to prove in ACL2.

On the other hand, an unnormalized representation presents some drawbacks. The major disadvantage from the verification standpoint is that equality becomes semantic, that is, it has to operate with the equivalence classes induced by the normalization process. This problem can be overcome in ACL2 by using equivalences and congruences.

Surprisingly, normalized and unnormalized representations have something in common: both of them use the notion of "monomial ordering" (a strict total order on terms lifted to monomials). This order is used in different places depending on the representation. In a normalized representation the order is taken into account in each operation. However, an unnormalized representation will use the order in the semantic equivalence predicate.

This observation allows us to build normalized polynomials (package `NPOL`) on top of a layer of unnormalized polynomials (package `UPOL`) presented by the authors in [6]. Let `nfp` stands for the predicate that checks whether an object is a list of non-null monomials in strictly decreasing order and `nf` the normalization function:

```
(in-package "NPOL")

(defun polynomialp (p)
  (and (UPOL::polynomialp p) (UPOL::nfp p)))

(defun + (p q)
  (UPOL::nf (UPOL::+ p q)))

(defun * (p q)
  (UPOL::nf (UPOL::* p q)))

(defun - (p)
  (UPOL::nf (UPOL::- p)))
```

```
(defun null ()
  (UPOL::null))

(defun identity ()
  (UPOL::identity))
```

Notice that the polynomials returned by the functions `UPOL::null` and `UPOL::identity` are already in normal form. The core ring properties for these normalizing operations follows:

```
(defthm |p + q = q + p|
  (equal (+ p q) (+ q p)))

(defthm |(p + q) + r = p + (q + r)|
  (equal (+ (+ p q) r) (+ p (+ q r))))

(defthm |p * q = q * p|
  (equal (* p q) (* q p)))

(defthm |(p * q) * r = p * (q * r)|
  (equal (* (* p q) r) (* p (* q r))))

(defthm |p * (q + r) = (p * q) + (p * r)|
  (equal (* p (+ q r)) (+ (* p q) (* p r))))

(defthm |p + (- p) = 0|
  (equal (+ p (- p)) (null)))

(defthm |0 + p = p|
  (implies (polynomialp p)
           (equal (+ (null) p) p)))

(defthm |1 * p = p|
  (implies (polynomialp p)
           (equal (* (identity) p) p)))
```

Most of these `NPOL` theorems are proved by disabling `UPOL` operations and using the `UPOL` counterpart of the theorem and the following properties:

```
(in-package "UPOL")

(defthm |nfp(p) <=> nf(p) = p|
  (iff (nfp p) (equal (nf p) p)))

(defcong = equal (nf p) 1)
```

The last form is a congruence theorem where `=` represents the semantic equivalence between unnormalized polynomials. It states that if the first argument of `nf` is replaced by an equivalent element, with respect to `=`, then the result is syntactically the same, i.e. `equal`. In fact, it produces the following form when expanded:

```
(defthm =-implies-equal-nf-1
  (implies (= p p-equiv)
           (equal (nf p) (nf p-equiv)))
  :rule-classes :congruence)
```

Nevertheless, the proof of some theorems such as associativity and distributivity is much more difficult to obtain. We have found that they require the following extra properties and congruences:

```
(defthm |p + nf(q) = p + q|
  (= (+ p (nf q)) (+ p q)))

(defthm |p * nf(q) = p * q|
  (= (* p (nf q)) (* p q)))

(defcong = = (+ p q) 2)
(defcong = = (* p q) 2)
```

The theorem `|p * nf(q) = p * q|` and the congruence of `*` with respect to `=` were very hard to prove. They were not present in our previous work on unnormalized polynomials, but they are required to extend them to a normalized representation.

In addition, the last two congruences were proved by using reversed versions of `|p + nf(q) = p + q|` and `|p * nf(q) = p * q|`. Unfortunately, reversing these rules produces infinite rewriting:

```
(+ p q) ⟶ (+ p (nf q)) ⟶ (+ p (nf (nf q))) ⟶ ···
(* p q) ⟶ (* p (nf q)) ⟶ (* p (nf (nf q))) ⟶ ···
```

In order to avoid this, we introduced syntactic restrictions to the applicability of the reversed rules. A term `(+ p q)` or `(* p q)` will only be rewritten by these rules if the term `q` is not of the form `(nf` x`)`. Syntactic restrictions can be imposed to a rule by means of `syntaxp`. A macro can help to improve the readability of the restricted rules.

```
(defmacro not-nf-syntaxp (p)
  `(syntaxp (not (and (consp ,p) (eq (first ,p) 'nf)))))
```

```
(defthm |p + q = p + nf(q)|
  (implies (not-nf-syntaxp q)
           (= (+ p q) (+ p (nf q)))))

(defthm |p * q = p * nf(q)|
  (implies (not-nf-syntaxp q)
           (= (* p q) (* p (nf q)))))
```

Once this was done, congruences with respect to the first argument were established by using commutativity explicitly.

Henceforth we will assume `NPOL` as the current package and we will use normalized polynomials. This package also imports common symbols from the `ACL2` and `COMMON-LISP` packages.

3 Lifting the Order from Monomials to Polynomials

Polynomials are constructed from monomials. Each monomial has a coefficient and a term. Let $X = \{x_1, \ldots, x_n\}$ be a finite set of variables with an ordering relation $<_X = \{(x_i, x_j) : 1 \leq i < j \leq n\}$. A term on X is a finite power product of the form $x_1^{e_1} \cdots x_n^{e_n}$, $e_i \in \mathbb{N}$. Once variables have been indexed, terms can be represented by lists of natural numbers, i.e. $x_1^{e_1} \cdots x_n^{e_n}$ is represented by $\langle e_1, \ldots, e_n \rangle$.

Taking into account exponent lists, the obvious choice is to set up a *lexicographical ordering* on these sequences of natural numbers:

$$\langle a_1, \ldots, a_n \rangle < \langle b_1, \ldots, b_n \rangle \equiv \exists i \; (a_i < b_i \wedge \forall j < i \; a_j = b_j).$$

In [6] this relation is shown to be a strict total order on terms in ACL2. In addition, this order was proved to be well-founded and admissible. The "monomial ordering" is just the translation of the term ordering into monomials where monomials are compared according to their terms. Therefore the monomial ordering inherits its properties from the term ordering.

3.1 Induced Polynomial Ordering

Monomial orderings can be lifted to normalized polynomials in a straightforward way. The following predicate compares two polynomials as lists of monomials by using `MON::=T` (the equality between the underlying terms of two monomials) and `MON::<` (the monomial ordering).

```
(defun < (p q)
  (cond ((or (endp p) (endp q))
         (not (endp q))
        ((MON::=T (first p) (first q))
         (< (rest p) (rest q)))
        (t
         (MON::< (first p) (first q))))))
```

It is proved that this relation satisfies the properties of a strict partial order (irreflexivity and transitivity).

```
(defthm |p < p|
  (not (< p p)))

(defthm |p < q & q < r => p < r|
  (implies (and (< p q) (< q r)) (< p r)))
```

3.2 Ordinal Type

Terms are embedded into the ordinals with the function `term->e0-ordinal`.

```
(defun term->e0-ordinal (a)
  (if (endp a)
      0
    (cons (cons (len a) (first a))
          (term->e0-ordinal (rest a)))))
```

This function maps every list of natural numbers to its corresponding ϵ_0-ordinal. It is proved that this is an order-preserving morphism which follows the pattern below.

$$\langle e_1, \ldots, e_n \rangle \longmapsto \omega^{\omega^n + e_1} + \cdots + \omega^{\omega + e_n} = \sum_{i=1}^{n} \omega^{\omega^{n-i+1} + e_i}.$$

There are other possible embeddings. However, we have found that this embedding presents the advantage of providing a small ordinal type, making this representation easier to handle.

The translation of monomials into ϵ_0-ordinals is accomplished by the function `monomial->e0-ordinal` which simply applies `term->e0-ordinal` to the underlying term. Now, we can proceed to embed polynomials in ϵ_0-ordinals by using the following function.

```
(defun polynomial->e0-ordinal (p)
  (if (endp p)
      0
    (cons (monomial->e0-ordinal (first p))
          (polynomial->e0-ordinal (rest p)))))
```

Let $p = \langle m_1, \ldots, m_k \rangle$ a polynomial, and $m_i = \langle e_{i1}, \ldots, e_{in} \rangle$, $1 \leq i \leq k$ its monomials. If p is given in a normalized representation then:

$$m_k <_M \cdots <_M m_1 \implies M_{\epsilon_0}(m_k) <_{\epsilon_0} \cdots <_{\epsilon_0} M_{\epsilon_0}(m_1),$$

where $<_M =$ `MON::<`, $M_{\epsilon_0} =$ `monomial->e0-ordinal` and $<_{\epsilon_0} =$ `e0-ord-<`. This follows from the theorem stating that $<_M$ is a well-founded relation. Thus, the function `polynomial->e0-ordinal` implements this mapping:

$$\langle m_1, \ldots, m_k \rangle \longmapsto \sum_{i=1}^{k} \omega^{\sum_{j=1}^{n} \omega^{\omega^{n-j+1}+e_{ij}}}.$$

3.3 Well-Foundedness

The first step to prove the well-foundedness of the polynomial ordering is proving that the polynomial ordinal embedding is correct.

```
(defthm correctness-of-polynomial->e0-ordinal
  (implies (polynomialp p)
           (e0-ordinalp (polynomial->e0-ordinal p))))
```

We were amazed when our first attempt failed. In fact, this is not a theorem in the current context: if `monomial->e0-ordinalp` happens to return 0, `polynomial->e0-ordinalp` does not construct a proper ϵ_0-ordinal in Cantor's Normal Form. One possible solution is to increment the ordinals associated to each term by 1:

```
(defun term->e0-ordinal (a)
  (if (endp a)
      1                                  ; formerly 0
    (cons (cons (len a) (first a))
          (term->e0-ordinal (rest a)))))
```

Obviously, this shift does not change the well-foundedness of the monomial ordering. With this slight modification, ordinals are assigned to polynomials according to:

$$\langle m_1, \ldots, m_k \rangle \longmapsto \sum_{i=1}^{k} \omega^{\sum_{j=1}^{n} \omega^{\omega^{n-j+1}+e_{ij}}+1}.$$

It is interesting to remark that the proof attempt fails again. However, the conjecture is not false any more. Inspection of the proof script reveals that ACL2 should use the antisymmetric property of `e0-ord-<`:

```
(defthm |a <e0 b => ~(b <e0 a)|
  (implies (e0-ord-< a b)
           (not (e0-ord-< b a))))
```

Once this property is proven, the correctness theorem is automatically proved. Then, the well-foundedness of the polynomial ordering can be established.

```
(defthm well-foundedness-of-<
  (and (implies (polynomialp p)
                (e0-ordinalp (polynomial->e0-ordinal p)))
       (implies (and (polynomialp p) (polynomialp q)
                     (< p q))
                (e0-ord-< (polynomial->e0-ordinal p)
                          (polynomial->e0-ordinal q))))
  :rule-classes :well-founded-relation)
```

Alternatively, we might prefer to leave `term->e0-ordinal` unchanged and modify `polynomial->e0-ordinal` accordingly. This latter possibility has the advantage of separating two concerns: the development of monomial orderings and the development of induced polynomial orderings.

```
(defun polynomial->e0-ordinal (p)
  (if (endp p)
      0
    (cons (ordinal-increment (monomial->e0-ordinal (first p)))
          (polynomial->e0-ordinal (rest p)))))

(defun ordinal-increment (a)
  (if (consp a)
      (if (and (atom (rest a)) (integerp (rest a)))
          (cons (first a) (+ (rest a) 1))
        (cons (first a) (ordinal-increment (rest a))))
    (if (and (integerp a) (<= 0 a))
        (+ a 1)
      a)))
```

As we can see, the ordinal produced by `monomial->e0-ordinal` is now incremented by the function `ordinal-increment`. It is proved that the increment of an ordinal is also an ordinal and that it cannot produce 0.

```
(defthm |e0-ordinalp(a) => e0-ordinalp(ordinal-increment(a))|
  (implies (e0-ordinalp a)
           (e0-ordinalp (ordinal-increment a))))

(defthm |~(ordinal-increment(a) = 0)|
  (not (equal (ordinal-increment a) 0)))
```

The key property about `ordinal-increment` states that if an ordinal is less than another, their relative order does not change when they are incremented. Although the proof of this theorem is rather complex, ACL2 can prove it on its own.

```
(defthm |a <e0 b => a + 1 <e0 b + 1|
  (implies (and (e0-ordinalp a) (e0-ordinalp b)
                (e0-ord-< a b))
           (e0-ord-< (ordinal-increment a)
                     (ordinal-increment b))))
```

Once these theorems are proved we proceed as before: first, the correctness of the polynomial ordinal embedding must be stated; then the well-foundedness of the polynomial ordering follows.

A final remark about `ordinal-increment`. It can be shown that there are no ordinals between an ordinal and its increment.

```
(defthm |~(a <e0 b & b <e0 a + 1)|
  (implies (and (e0-ordinalp a) (e0-ordinalp b))
           (not (and (e0-ord-< a b)
                     (e0-ord-< b (ordinal-increment a)))))
  :rule-classes nil)
```

4 Conclusions and Future Work

A polynomial ordering has been developed and its well-foundedness has been proved. This has been carried out by the only way allowed by ACL2: a proper embedding of polynomials into the ϵ_0-ordinals. The development is generic, in the sense that the construction of an induced polynomial ordering can be made independent of the particular underlying monomial ordering. This also proves that if the ordinal type of a monomial ordering is less than ϵ_0, then the induced polynomial ordering is also less than ϵ_0.

We adapted our previous development on polynomials to suit the needs of our current work. Several new events have been added and others have been modified. An example is the theorem `|p * nf(q) = p * q|`. Another case is the embedding of terms into the ϵ_0-ordinals which was slightly modified before being lifted to polynomials.

This particular term ordering is, basically, a lexicographical ordering on the set of sequences of natural numbers. The term ordering is translated to monomials in a straightforward manner by comparing monomials according to their terms (coefficients are not taken into account).

The motivation for this work is to build up a solid basis for proving the termination of reduction relations on polynomials in ACL2.

Equational reasoning and rewriting in a general setting [1] is covered in J. L. Ruiz Reina's PhD Thesis [7, 8]. With respect to this point, our approach is more ad hoc: we are interested in very special reductions. Therefore we plan to introduce polynomial reduction relations as subsets of the polynomial ordering. This is akin to the approach taken by L. Théry [9] in COQ [2] and by the Computer Algebra community [3, 10].

The main application of these reductions on polynomials is the formalization of the notions related to Buchberger's algorithm for Gröbner bases computation.

References

[1] Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998)

[2] Dowek, G., Felty, A., Herbelin, H., Huet, G., Murty, C., Parent, C., Paulin-Mohring, C., Werner, B.: The COQ Proof Assistant Reference Manual. INRIA. Technical Report **0203** (1999)

[3] Geddes, K. O., Czapor, S. R., Labahn, G.: Algorithms for Computer Algebra. Kluwer Academic Publishers (1992)

[4] Kaufmann, M., Manolios, P., Moore, J S.: Computer-Aided Reasoning: An Approach. Kluwer Academic Publishers (2000)

[5] Kaufmann, M., Manolios, P., Moore, J S.: Computer-Aided Reasoning: ACL2 Case Studies. Kluwer Academic Publishers (2000)

[6] Medina-Bulo, I., Alonso-Jiménez, J. A., Palomo-Lozano, F.: Automatic Verification of Polynomial Rings Fundamental Properties in ACL2. The University of Texas at Austin, Department of Computer Sciences. Technical Report **0029** (2000)

[7] Ruiz-Reina, J. L., Alonso-Jiménez, J. A., Hidalgo-Doblado, M. J., Martín-Mateos, F. J.: Formalizing Rewriting in the ACL2 Theorem Prover. LNCS **1930** (2000)

[8] Ruiz-Reina, J. L.: Una Teoría Computacional acerca de la Lógica Ecuacional. Formalización en ACL2 de la Lógica Ecuacional y Demostración Automática de sus Propiedades. University of Sevilla. PhD Thesis. (2001)

[9] Théry, L.: A Machine-Checked Implementation of Buchberger's Algorithm. J. Automated Reasoning **26** (2001)

[10] Winkler, F.: Polynomial Algorithms in Computer Algebra. Springer-Verlag (1996)