

A Formal Proof of Dickson's Lemma in ACL2*

F.J. Martín-Mateos, J.A. Alonso, M.J. Hidalgo, and J.L. Ruiz-Reina

Computational Logic Group, Dept. of Computer Science and Artificial Intelligence,
University of Seville, E.T.S.I. Informática,
Avda. Reina Mercedes, s/n. 41012 Sevilla, Spain,
[http://www.cs.us.es/{~fmartin,~jalonso,~mjoseh,~jruiuz}](http://www.cs.us.es/~fmartin,~jalonso,~mjoseh,~jruiuz)

Abstract. Dickson's Lemma is the main result needed to prove the termination of Buchberger's algorithm for computing Gröbner basis of polynomial ideals. In this case study, we present a formal proof of Dickson's Lemma using the ACL2 system. Due to the limited expressiveness of the ACL2 logic, the classical non-constructive proof of this result cannot be done in ACL2. Instead, we formalize a proof where the termination argument is justified by the multiset extension of a well-founded relation.

1 Introduction

Dickson's Lemma is the main result needed to prove the termination of Buchberger's algorithm [2] for computing Gröbner basis of polynomial ideals. Thus, a formal proof of this result is needed by any formal termination proof of this algorithm. In particular, if we use the ACL2 system [7] to define and verify Buchberger's algorithm, a formal proof of Dickson's Lemma is essential, in order to reason about it in ACL2. This is our motivation for doing the formal proof of Dickson's Lemma. Since ACL2 consists of a programming language (an extension of an applicative subset of Common Lisp), a logic describing the programming language and a theorem prover supporting deduction in the logic, a formally verified Buchberger's algorithm in ACL2 would allow an environment in which proving and computing would be intermixed.

The ACL2 logic is a subset of first-order logic, without quantifiers and with a principle of proof by induction. Due to this limited expressiveness, it is not possible to reproduce the classical non-constructive proof of Dickson's Lemma as it is usually presented in the literature. The proof we present here is constructive and it is mainly based on a multiset extension of a well-founded relation. In the mechanization of this proof, we use a tool for defining multiset well-founded relations in ACL2 in an automated way, a tool that we used previously in other formalizations [13] and that can now be reused.

Dickson's Lemma is usually stated as follows:

Theorem 1 (Dickson's Lemma). *Let $n \in \mathbb{N}$ and $\{m_k : k \in \mathbb{N}\}$ be an infinite sequence of monomials in the variables $\{X_1, \dots, X_n\}$. Then, there exist indices $i < j$ such that m_i divides m_j .*

* This work has been supported by project TIC2000-1368-C03-02 (Ministry of Science and Technology, Spain) and FEDER funds.

Given a fixed set of variables $V = \{X_1, \dots, X_n\}$, we can naturally identify the set of n -variate monomials (with variables in V) with the set \mathbb{N}^n of n -tuples of natural numbers: a monomial $X_1^{e_1} X_2^{e_2} \dots X_n^{e_n}$ can be seen as the n -tuple $\langle e_1, \dots, e_n \rangle$. The divisibility relation between monomials is then identified with the relation \leq^n on \mathbb{N}^n , defined as $\langle k_1, \dots, k_n \rangle \leq^n \langle l_1, \dots, l_n \rangle$ if and only if $k_i \leq l_i$ for all $1 \leq i \leq n$. In the sequel, we will identify tuples and monomials in this sense. Thus, Dickson's Lemma can be reformulated stating that for every infinite sequence $\{f_k : k \in \mathbb{N}\}$ of n -tuples of natural numbers there exist indices $i < j$ such that $f_i \leq^n f_j$.

As we said above, the classical proof of Dickson's Lemma is non-constructive (see [1], for example), and thus it is not suitable for being formalized in the ACL2 logic. The proof we describe in the following is based on the same ideas as some constructive proofs already present in the literature [10, 14], and it essentially shows a well-founded measure that can be associated to the initial segments of the sequence of tuples and that decreases whenever a tuple in the sequence is not divided by any of the previous tuples.

2 Formalizing the Proof in ACL2

The ACL2 logic is a quantifier-free, first-order logic with equality, describing an applicative subset of Common Lisp. The syntax of terms is that of Common Lisp and the logic includes axioms for propositional logic and for a number of Lisp functions and data types. Rules of inference of the logic include those for propositional calculus, equality and instantiation. One important rule of inference is the *principle of induction*, that permits proofs by well-founded induction on the ordinal ε_0 . The theory has a constructive definition of the ordinals up to ε_0 , in terms of lists and natural numbers, given by the predicate `e0-ordinalp` and the order `e0-ord-<`. Although this is the only built-in well-founded relation, the user may define new well-founded relations from that, by previously providing an order-preserving ordinal function.

By the *principle of definition*, new function definitions are admitted as axioms only if there exists a measure in which the arguments of each recursive call decrease with respect to a well-founded relation; in this way, it is ensured that no inconsistencies are introduced by new definitions. Usually, the system can prove automatically this property using a predefined ordinal measure on Lisp objects and the relation `e0-ord-<`. Nevertheless, if the termination proof is not trivial, the user has to explicitly provide a measure on the arguments and a well-founded relation ensuring termination.

The ACL2 theorem prover mechanizes the logic, being particularly well suited for obtaining automated proofs based on simplification and induction. For a detailed description of ACL2, we refer the reader to the ACL2 book [6].

For the sake of readability, the ACL2 expressions in this paper are presented using a notation closer to the usual mathematical notation than its original Common Lisp syntax. Some of the functions are also used in infix notation. The complete proof can be found in <http://www.cs.us.es/~fmartin/acl2/dickson/>.

2.1 Formulation of Dickson's Lemma

To formalize Dickson's Lemma in the ACL2 logic, we consider a constant N (that is, a 0-ary function) representing the number of variables, and a unary function f , representing the infinite sequence of monomials given as N -tuples of natural numbers. These functions are abstractly defined by means of the **encapsulate** mechanism, which allows the user to introduce new function symbols by axioms constraining them to have certain properties. To ensure consistency, local witness functions having the same properties have to be exhibited. Inside an **encapsulate** construct, the properties stated need to be proved for the local witnesses, and outside, they work as assumed axioms. In this case, the assumed properties about N and f are the following¹:

ASSUMPTION: **N-is-natural->-0**

$N \in \mathbb{N} \wedge 0 < N$

ASSUMPTION: **f-sequence-of-N-tuples**

$i \in \mathbb{N} \rightarrow [\text{len}(f(i)) = N \wedge \text{natural-listp}(f(i))]$

where **natural-listp** checks if its argument is a list of natural numbers (we use lists of length n to represent n -tuples).

Here, the **encapsulate** mechanism behaves like an universal quantifier over the functions abstractly defined with it. So, any theorem proved about these functions is true for any functions with the same properties as the ones assumed in the **encapsulate** construct, by means of functional instantiation (see [6] for details). This is the case for the ACL2 formalization of Dickson's Lemma: as the infinite sequence of monomials is abstractly defined via **encapsulate**, the proved properties about it are valid for any infinite sequence of monomials.

Let us now define the functions needed to state Dickson's Lemma. First, the function **tuple-<=** implements the divisibility relation (that is, the relation \leq^n):

DEFINITION:

```

T1 tuple-<= T2 ⇔
  if endp(T1) then endp(T2)
  elseif endp(T2) then endp(T1)
  elseif car(T1) ∈ ℕ ∧ car(T2) ∈ ℕ
    then car(T1) ≤ car(T2) ∧ cdr(T1) tuple-<= cdr(T2)
  else nil

```

The following function **get-tuple-<=-f** has two arguments, a natural number j and a N -tuple T , and it returns the largest index i such that $i < j$ and $f(i)$ **tuple-<=** T whenever such index exists (**nil** otherwise):

DEFINITION:

```

get-tuple-<=-f(j,T) =
  if j ∈ ℕ then if j = 0 then nil
    elseif f(j - 1) tuple-<= T then j - 1
    else get-tuple-<=-f(j - 1,T)
  else nil

```

¹ The local witnesses are irrelevant to our description of the proof.

Finally, the following function `dickson-indices` receives as input an index k and uses `get-tuple-<=f` to recursively search a pair of indices $i < j$ such that $j \geq k$ and $f(i) \text{ tuple-}\leq f(j)$:

DEFINITION:

```

dickson-indices(k) =
  if k ∈ ℕ then let i be get-tuple-<=f(k,f(k))
    in if i ≠ nil then ⟨i, k⟩
      else dickson-indices(k + 1)
  else nil

```

Let us assume for the moment that we have proved that the function `dickson-indices` terminates and that this definition has been admitted by the system. Then the following property is easily proved as direct consequence of the definitions of the functions involved:

THEOREM: `dickson-lemma`

$$[k \in \mathbb{N} \wedge \text{dickson-indices}(k) = \langle i, j \rangle] \rightarrow [i < j \wedge f(i) \text{ tuple-}\leq f(j)]$$

This theorem ensures that for any infinite sequence of monomials $\{f_k : k \in \mathbb{N}\}$, there exists $i < j$ such that f_i divides f_j (and the function `dickson-indices` explicitly provides these values). Thus, it is a formal statement of Dickson's Lemma in ACL2.

The hard part is the termination proof of the function `dickson-indices`. For that purpose, we have to explicitly provide to the system a measure on the input argument and prove that the measure decreases with respect to a given well-founded relation in every recursive call. We present the details in the next subsections.

2.2 A Well-Founded Measure

Before giving a formal definition of the termination measure, we give some intuition by means of an example. Let $\{f_k : k \in \mathbb{N}\}$ be an infinite sequence of pairs of natural numbers. Let us assume that $f_0 = \langle 3, 2 \rangle$, $f_1 = \langle 1, 5 \rangle$ and $f_2 = \langle 2, 1 \rangle$. In figure 1, we sequentially represent (by the shaded regions) the set of tuples that are divisible by some element of the sequence.

Thus, in each step, the non-shaded region represents the set of tuples that can be the next in the sequence without being divisible by the previous tuples. The main idea is that for every tuple of the sequence that is not divisible by any of the previous tuples, this "free space" decreases with respect to a well-founded relation.

Let us precise this intuitive idea. We can have a compact representation of the non-shaded regions by means of *patterns*. A **pattern** is an element of $(\mathbb{N} \cup \{*\})^n$, representing the set of tuples obtained replacing every occurrence of $*$ in the pattern by a natural number (occurrences of $*$ in a pattern will be called *freedom*s). Thus, the non-shaded regions may be represented by a multiset of patterns. For example, the non-shaded region of figure 1-b) is represented by

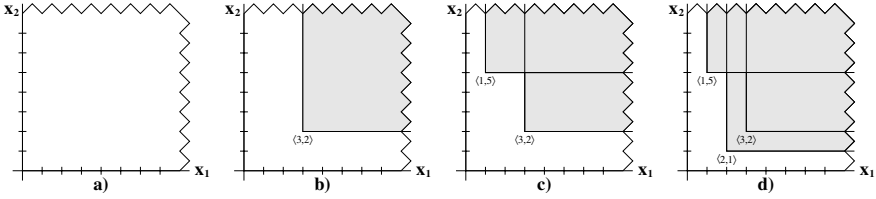


Fig. 1. Graphical idea of the measure.

$\{\langle 0, * \rangle, \langle 1, * \rangle, \langle 2, * \rangle, \langle *, 0 \rangle, \langle *, 1 \rangle\}$. We denote as $\mathcal{S}(\Pi)$ and $\mathcal{S}(\mathcal{P})$, the set of tuples represented by a pattern Π and by a multiset of patterns \mathcal{P} , respectively.

In every step, the new region is obtained from the previous one, by replacing some patterns by others. Given a new tuple T in the sequence, a pattern Π has to be replaced if there is some $T' \in \mathcal{S}(\Pi)$ divisible by T (we say in that case that Π is **reducible** by T). These reducible patterns are replaced by a new collection of patterns representing the new region obtained excluding the divisible tuples (we call these new patterns the **reductions** of Π with respect to T). Note that Π' is a reduction of Π with respect to T , if Π' is equal to Π except that one of the occurrences of $*$ in Π has been replaced by a natural number less than the number that appears in the same position in T . In the following table we present the patterns computed, for $k = 0, 1, 2, 3$, when f is the sequence of the example of figure 1. We also indicate the reducible patterns in each step.

k	Non-shaded regions	f_k	Reducible patterns
0	$\{\langle *, * \rangle\}$	$\langle 3, 2 \rangle$	$\langle *, * \rangle$
1	$\{\langle 0, * \rangle, \langle 1, * \rangle, \langle 2, * \rangle, \langle *, 0 \rangle, \langle *, 1 \rangle\}$	$\langle 1, 5 \rangle$	$\langle 1, * \rangle, \langle 2, * \rangle$
2	$\{\langle 0, * \rangle, \langle 1, 0 \rangle, \langle 1, 1 \rangle, \langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 1, 4 \rangle, \langle 2, 0 \rangle, \langle 2, 1 \rangle, \langle 2, 2 \rangle, \langle 2, 3 \rangle, \langle 2, 4 \rangle, \langle *, 0 \rangle, \langle *, 1 \rangle\}$	$\langle 2, 1 \rangle$	$\langle 2, 1 \rangle, \langle 2, 2 \rangle, \langle 2, 3 \rangle, \langle 2, 4 \rangle, \langle *, 1 \rangle$
3	$\{\langle 0, * \rangle, \langle 1, 0 \rangle, \langle 1, 1 \rangle, \langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 1, 4 \rangle, \langle 2, 0 \rangle, \langle *, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 1 \rangle\}$		

If we define the **dimension** of a pattern as its number of freedoms, then it is clear that every reducible pattern is replaced by a finite number of patterns with dimension strictly smaller. For example, the multiset of dimensions of the patterns representing the region of figure 1-b) is $\{\{1, 1, 1, 1, 1\}\}$ and the corresponding multiset for the region of figure 1-c) is $\{\{1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1\}\}$. In other words, every time the region is reduced, the multiset of the dimensions of the patterns representing the region decreases with respect to the multiset relation induced by the usual order between natural numbers. This multiset relation is known to be well-founded, and thus it is not possible to reduce the regions infinitely often, justifying Dickson’s Lemma.

We now explain how we formalize these constructions in ACL2. A pattern will be represented as a list with the same length as the tuples. The function `member-tuple` implements the property $T \in \mathcal{S}(\Pi)$ (note that any occurrence in the pattern of an object that it is not a natural number stands for “*”):

DEFINITION:

```

member-tuple( $T, II$ ) =
  if endp( $T$ ) then endp( $II$ )
  elseif endp( $II$ ) then endp( $T$ )
  elseif  $\text{car}(II) \in \mathbb{N}$  then  $\text{car}(T) = \text{car}(II) \wedge \text{member-tuple}(\text{cdr}(T), \text{cdr}(II))$ 
  else member-tuple( $\text{cdr}(T), \text{cdr}(II)$ )

```

Next, we present the definition of the function **reductions**(II, T) that computes the multiset² of reductions of a pattern II with respect to a tuple T (assuming that II is reducible by T). Let us recall that every reduction is obtained by replacing a freedom of II by a natural number less than the one that appears in the same position in T :

DEFINITION:

```

reductions( $II, T$ ) =
  if endp( $II$ ) then nil
  elseif  $\text{car}(II) \in \mathbb{N}$ 
    then cons-list-cdr( $\text{car}(II), \text{reductions}(\text{cdr}(II), \text{cdr}(T))$ )
  else cons-list-car(natural-<-list( $\text{car}(T), \text{cdr}(II)$ ) @
    cons-list-cdr( $\text{car}(II), \text{reductions}(\text{cdr}(II), \text{cdr}(T))$ )

```

where the symbol @ is the “append” operation between lists, the function **natural-<-list** returns the list of natural numbers less than its argument (that is, (**natural-<-list** n) = '(0 1 ... $n-1$)) and the functions **cons-list-car** and **cons-list-cdr** behave schematically in the following way:

$$(\text{cons-list-car } '(x_1 \dots x_n) 'l) = '((x_1 . l) \dots (x_n . l))$$

$$(\text{cons-list-cdr } 'x '(l_1 \dots l_n)) = '((x . l_1) \dots (x . l_n))$$

Given a multiset of patterns \mathcal{P} and a tuple T , the function **reduction-list** describes how the multiset of patterns \mathcal{P} is reduced to a new multiset by the tuple T :

DEFINITION:

```

reductions-list( $\mathcal{P}, T$ ) =
  if endp( $\mathcal{P}$ ) then  $\mathcal{P}$ 
  elseif member-tuple( $T, \text{car}(\mathcal{P})$ ) then reductions( $\text{car}(\mathcal{P}), T$ ) @ cdr( $\mathcal{P}$ )
  else cons( $\text{car}(\mathcal{P}), \text{reductions-list}(\text{cdr}(\mathcal{P}), T)$ )

```

Note that the above function differs from the intuitive construction outlined above in two aspects. First, *only the first reducible pattern* is replaced by its reductions. Second, instead of looking for a pattern II reducible by T , we check the stronger condition $T \in \mathcal{S}(II)$. As we will see, both simplifications are sound³.

² We will represent multisets as lists. Although this representation is not unique (the same multiset may be represented by different lists), it is adequate for our purposes.

³ It is interesting to note that the soundness of both simplifications (which are not intuitive, especially the first one) makes for simpler proofs and were discovered from the interaction with the prover.

The function `reductions-tuple-list` iterates the reduction process over a finite sequence of tuples. It must be noticed that the list of tuples is provided in the reverse order:

DEFINITION:

```

reductions-tuple-list( $\mathcal{P}, T\text{-lst}$ ) =
  if endp( $T\text{-lst}$ ) then  $\mathcal{P}$ 
  else reductions-list(reductions-tuple-list( $\mathcal{P}, \text{cdr}(T\text{-lst})$ ), car( $T\text{-lst}$ ))

```

The function `pattern-list-measure` computes the multiset of dimensions of a multiset of patterns (we omit here the definition of the function `dimension` which computes the number of freedoms in a pattern):

DEFINITION:

```

pattern-list-measure( $\mathcal{P}$ ) =
  if endp( $\mathcal{P}$ ) then nil
  else cons(dimension(car( $\mathcal{P}$ )), pattern-list-measure(cdr( $\mathcal{P}$ )))

```

And finally, following the intuitive idea sketched above, we can associate a measure (a multiset of natural numbers) to every index k :

DEFINITION:

```

dickson-indices-measure( $k$ ) =
  pattern-list-measure(
    reductions-tuple-list(list(initial-pattern( $N$ ))),
    initial-segment-f( $k - 1$ ))

```

where the function `initial-pattern(N)` builds the initial pattern $\langle *, \dots, * \rangle$ and the function `initial-segment-f(k)` builds the list of tuples $(f_k \dots f_1 f_0)$.

2.3 Termination Proof of dickson-indices

The last step in this formal proof is to define a well-founded relation and prove that the given measure decreases with respect to it in every recursive call of the function `dickson-indices`. We will define it as the relation induced by a well-founded relation on finite multisets of natural numbers. Intuitively, this relation is defined such that a smaller multiset can be obtained by removing a non-empty subset of elements, and adding elements which are smaller than some element removed. In [5], Dershowitz and Manna show that if the base relation is well-founded, then the relation induced on finite multisets is also well-founded.

As we said above, the only predefined well-founded relation in ACL2 is `e0-ord-<`, implementing the usual order between ordinals less than ε_0 . The function `e0-ordinalp` recognizes those ACL2 objects representing such ordinals. If we want to define a new well-founded relation in ACL2, we have to explicitly provide a monotone ordinal function, and prove the corresponding order-preserving theorem (see [6] for details). Fortunately, we do not have to do this: we use the `defmul` tool. This tool, previously implemented and used by the authors in [13], automatically generates the definitions and prove the theorems needed to introduce in ACL2 the multiset relation induced by a given well-founded relation. In our case, we only need the following `defmul` call:

```
(defmul (e0-ord-< nil e0-ordinalp e0-ord-<-fn nil nil))
```

This automatically generates the definition of a function `mul-e0-ord-<`, implementing the multiset relation on finite multisets (lists) of ordinals induced by the relation `e0-ord-<`. And it also automatically proves the theorems needed to introduce this relation as a well-founded relation in ACL2. See details about the `defmul` syntax in [13]. For simplicity, in the following we denote `mul-e0-ord-<` as $<_{\varepsilon_0, \mathcal{M}}$.

We finally prove that the measure decreases with respect to $<_{\varepsilon_0, \mathcal{M}}$ in the recursive call of the function `dickson-indices`, hence justifying its termination. We now explain the main lemmas needed to show this result.

Note that if $T \in \mathcal{S}(\mathcal{P})$, then the multiset measure of `reduction-list`(\mathcal{P}, T) is smaller than the measure of \mathcal{P} with respect to $<_{\varepsilon_0, \mathcal{M}}$. This is established by the following theorem, where the property $T \in \mathcal{S}(\mathcal{P})$ is defined by the function `exists-pattern`, omitted here:

```
LEMMA: reductions-list-reduces-pattern-list-measure
exists-pattern( $\mathcal{P}, T$ )
  → pattern-list-measure(reductions-list( $\mathcal{P}, T$ ))
     <_{\varepsilon_0, \mathcal{M}} pattern-list-measure( $\mathcal{P}$ )
```

This lemma is an easy consequence of the definition of $<_{\varepsilon_0, \mathcal{M}}$ and the fact that the replaced pattern has a bigger dimension than its reductions:

```
LEMMA: reductions-property
 $\Pi_1 \in \text{reductions}(\Pi_2, T) \rightarrow \text{dimension}(\Pi_1) < \text{dimension}(\Pi_2)$ 
```

The following lemma establishes the main property of the function `reductions-tuple-list`. If a tuple T is in the set of tuples represented by a pattern multiset \mathcal{P} , then this tuple is still in the pattern multiset obtained after applying a sequence of reductions corresponding to a given sequence of tuples $T\text{-lst}$, provided that T is not divisible by any of the tuples of $T\text{-lst}$ (this divisibility condition is checked by the function `divisible-tuple`, omitted here):

```
LEMMA: exists-pattern-reductions-tuple-list
( natural-listp( $T$ ) ∧ natural-list-listp( $T\text{-lst}$ )
  ∧ exists-pattern( $\mathcal{P}, T$ ) ∧ ¬divisible-tuple( $T\text{-lst}, T$ ) )
  → exists-pattern(reductions-tuple-list( $\mathcal{P}, T\text{-lst}, T$ ))
```

In addition, every tuple is in the initial multiset pattern:

```
LEMMA: initial-pattern-exists-pattern
len( $T$ ) =  $n \rightarrow \text{exists-pattern}(\text{list}(\text{initial-pattern}(n)), T)$ 
```

As a consequence of the above two lemmas, if f_k is not divisible by any of $f_0 \dots f_{k-1}$ (that is, the recursive case in the definition of `dickson-indices`), then there exists a pattern Π in the multiset of patterns generated in the k -th step such that $f_k \in \mathcal{S}(\Pi)$. So now we can use the lemma `reductions-list-reduces-pattern-list-measure` to conclude that the measure of the argument in the recursive call in `dickson-indices` decreases with respect to $<_{\varepsilon_0, \mathcal{M}}$. That is, we have the following theorem:

THEOREM: dickson-indices-termination-property
 $k \in \mathbb{N} \wedge \neg \text{get-tuple-}<=f(k, f(k))$
 $\rightarrow \text{dickson-indices-measure}(k + 1) <_{\mathcal{M}} \text{dickson-indices-measure}(k)$

This is exactly the proof obligation generated to show the termination of the function `dickson-indices`. Thus, its definition is admitted in the logic and then the theorem `dickson-lemma` presented in subsection 2.1 is easily proved.

3 Conclusions and Related Work

We have presented a formalization and proof of Dickson’s Lemma in the ACL2 system. This is an essential preliminary step to obtain a formal termination proof of a Common Lisp implementation of Buchberger’s algorithm [9]. We think that this is a good example of how a non-trivial result can be formalized in the first-order, quantifier-free logic of ACL2 (overriding its apparent lack of expressiveness). In fact, the automation of the proof is very simple: the hard part was to preconceive a proof of the result in the restricted ACL2 logic. It is worth pointing that after obtaining it, we realized that we had rediscovered a proof with similar arguments to some constructive proofs of Dickson’s Lemma already present in the literature [10, 14].

There are several contributions related to the formalization of Dickson’s Lemma using proof checkers. In [16] a formalization of Buchberger’s Algorithm is presented in COQ, using a non-constructive proof of Dickson’s Lemma developed in [12]. There is also a non-constructive development in Mizar [8] based on the book [2]. In [3] a particular case of Dickson’s Lemma ($n = 2$) is constructively formalized in the system MINLOG. Another constructive approach is [4], in which a constructive proof of Dickson’s Lemma is mechanized using open induction in the system AGDA. This proof is used to get a fully constructive proof of the existence of Gröbner bases in COQ [11]. A comparison with our work is difficult since the proof we formalize is substantially different and, more important, the ACL2 logic is less expressive than the logics of those systems. At the time of this writing, a new proof of Dickson’s Lemma [15] was carried out in ACL2. In this proof, instead of using multisets, an explicit ordinal mapping is assigned to finite sequences of monomials, and proved to be strictly decreasing if no monomial divides a subsequent monomial.

To quantify the proof effort, it should be noted that only 20 definitions and 32 lemmas are needed in the proof, which gives an idea of the degree of automation of the proof and its simplicity. Of course, part of its simplicity comes from the use of the `multiset` book, which provides a proof of well-foundedness of the multiset relation induced by a well-founded relation. It is worth pointing the reuse of the `defmul` tool for generating multiset well-founded relations in ACL2: although it was originally developed to prove Newman’s Lemma about abstract reductions [13], it was designed in a very general way such that it has turned out to be useful in other formalization tasks, being Dickson’s Lemma a relevant example of this.

References

1. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
2. T. Becker and V. Weispfenning. *Gröbner Bases: A Computational Approach to Commutative Algebra*. Springer–Verlag, 1998.
3. U. Berger, H. Schwichtenberg and M. Seisenberger. The Warshall Algorithm and Dickson’s Lemma: Two Examples of Realistic Program Extraction. *Journal of Automated Reasoning* 26: 205–221, 2001.
4. T. Coquand and H. Persson. Gröbner Bases in Type Theory. In *Types for Proofs and Programs: Selected papers of TYPES’98*, LNCS 1657, pages 33–46. Springer–Verlag, 1999.
5. N. Dershowitz and Z. Manna. Proving Termination with Multiset Orderings. *Communications of the ACM* 22(8):465–476, 1979.
6. M. Kaufmann, P. Manolios, and J.S. Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, 2000.
7. M. Kaufmann and J.S. Moore. *ACL2 Version 2.7*, 2001. Homepage: <http://www.cs.utexas.edu/users/moore/ac12/>
8. G. Lee and P. Rudnicki. Dickson’s Lemma. *Journal of Formalized Mathematics* 14, 2002.
9. I. Medina–Bulo, J.A. Alonso, F. Palomo. Polynomial algorithms in ACL2 (an approach to Buchberger algorithm). In *I Taller Iberoamericano sobre Deducción Automática e Inteligencia Artificial, IDEIA 2002* (in spanish), 2002. Available at <http://www.cs.us.es/ideia>
10. H. Perdry. Strong noetherianity: a new constructive proof of Hilbert’s basis theorem. Available at <http://perdry.free.fr/StrongNoetherianity.ps>
11. H. Persson. *An Integrated Development of Buchberger’s Algorithm in Coq*. Rapport de recherche de l’INRIA, n 4271, 2001.
12. L. Pottier. *Dixon’s lemma*, 1996. Available at <ftp://ftp-sop.inria.fr/lemme/Loic.Pottier/MON/>
13. J.L. Ruiz–Reina, J.A. Alonso, M.J. Hidalgo, and F.J. Martín. Multiset Relations: a Tool for Proving Termination. In *Second ACL2 Workshop*, Technical Report TR-00-29, Computer Science Department, University of Texas, 2000. Available at <http://www.cs.utexas.edu/users/moore/ac12/workshop-2000/>
14. S.G. Simpson. Ordinal numbers and the Hilbert basis theorem. *Journal of Symbolic Logic* 53(3): 961–974, 1988.
15. M. Sustyk. Proof of Dickson’s Lemma Using the ACL2 Theorem Prover via an Explicit Ordinal Mapping. In *Fourth ACL2 Workshop*, 2003. Available at <http://www.cs.utexas.edu/users/moore/ac12/workshop-2003/>
16. L. Théry. A Machine-Checked Implementation of Buchberger’s Algorithm. *Journal of Automated Reasoning* 26(2): 107–137, 2001.