Multiset Relations: a Tool for Proving Termination *

J.-L. Ruiz-Reina, J.-A. Alonso, M.-J. Hidalgo and F.-J. Martín
{jruiz,jalonso,mjoseh,fjesus}@cica.es

Departamento de Ciencias de la Computación e Inteligencia Artificial.
Facultad de Informática y Estadística, Universidad de Sevilla
Avda. Reina Mercedes, s/n. 41012 Sevilla, Spain

Abstract. We present in this paper a formalization of multiset relations in ACL2, and we show how multisets can be used to prove non-trivial termination properties in ACL2. Intuitively, multisets are sets that admit multiple occurrences of elements. Every relation on a set A induces a relation on finite multisets over A and it can be shown that the multiset relation induced by a well-founded relation is also well-founded. We proved this property in the ACL2 logic, and used it by functional instantiation in order to provide well-founded relations to the admissibility test of recursive functions. We also developed a macro `defmul`, to define well-founded multiset relations in an easy way. Finally, we present three case studies where multisets are used to prove non-trivial termination properties: a tail-recursive version of Ackermann's function, a definition of McCarthy's 91 function and a proof of Newman's lemma for abstract reduction relations.

Introduction

We present in this paper a formalization of multiset relations in ACL2, and we show how these relations can be used to prove non-trivial termination properties, providing a tool for defining relations on finite multisets and showing that these relations are well-founded. Such well-founded relations are used in the admissibility test for recursive functions, allowing the user to provide a particular multiset measure in order to prove termination of recursively defined functions.

Multisets are usually defined in an informal way as "sets with repeated elements". Dershowitz and Manna [4] proved that every well-founded relation on a set A induces a well-founded relation on the set of finite multisets of elements taken from A. We have formalized this theorem using ACL2, and stated it in an abstract way. This allows to instantiate the theorem to show well-foundedness of concrete multiset relations.

We have also developed a macro `defmul` in order to easily make definitions of induced multiset relations. Besides defining the multiset relation induced by a given relation, this macro proves, by functional instantiation, well-foundedness of the defined multiset relation, provided that the original relation is well-founded.

The first section of this paper presents how we have formalized and proved well-foundedness of multiset relations induced by well-founded relations. The second section presents the macro `defmul` and it is shown how it can be used to define multiset well-founded relations. In the three sections after that, three case studies of increasing complexity are presented, showing how multisets can be used to prove non-trivial termination properties. The first one is a tail-recursive definition of Ackermann's function. The second one shows admissibility of an iterative version of McCarthy's 91 function. The third one is a proof of Newman's lemma about abstract reduction relations: terminating and locally confluent reduction relations are confluent.

* This work has been supported by DGES/MEC: Projects PB96-0098-C04-04 and PB96-1345

We will assume the reader has a working knowledge of ACL2. The best introduction to ACL2 is [5]. Due to the lack of space, we will skip details of the mechanical proofs and we will not mention guards or guards verification. The complete books are available on the web in `http://www-cs.us.es/~jruiz/acl2-mul/`.

1 Formalization of multiset relations in ACL2

1.1 Multisets: definitions and properties

A *multiset* M over a set A is a function from A to the set of natural numbers. This is a formal way to define "sets with repeated elements". Intuitively, $M(x)$ is the number of copies of $x \in A$ in M. This multiset is *finite* if there are finitely many x such that $M(x) > 0$. The set of all finite multisets over A is denoted as $\mathcal{M}(A)$.

We will use standard set notation to represent multisets. For example, if $A = \{a, b, c\}$, an example of multiset over A is $M = \{a, b, b, b\}$, an abbreviation of the function $M(a) = 1$, $M(b) = 3$ and $M(c) = 0$. Thus, $\{a, b, b, b\}$ is identical to the multiset $\{b, b, a, b\}$, but distinct from the multiset $\{a, b, b\}$.

Basic operations on multisets are defined to generalize the same operations on sets, taking into account multiple occurrences of elements: $x \in M$ means $M(x) > 0$, $M \subseteq N$ means $M(x) \leq N(x)$, for all $x \in A$, $M \cup N$ is the function $M + N$ and $M \setminus N$ is the function $M \dot{-} N$ (where $x \dot{-} y$ is $x - y$ if $x \geq y$ and 0 otherwise). For example, $\{a, b, b, a\} \cup \{c, c, a, b\}$ is the multiset $\{a, a, a, b, b, b, c, c\}$ and $\{a, b, b, a\} \setminus \{c, c, a, b\}$ is the multiset $\{a, b\}$.

Any ordering defined on a set A induces an ordering on multisets over A: given a multiset, a smaller multiset can be obtained by removing a non-empty subset X and adding elements which are smaller than some element in X. This construction can be generalized to binary relations in general, not only for partial orderings. This is the formal definition:

DEFINITION 1. Given a relation $<$ on a set A, the *multiset relation* induced by $<$ on $\mathcal{M}(A)$, denoted as $<_{mul}$, is defined as $N <_{mul} M$ iff there exist $X, Y \in \mathcal{M}(A)$ such that $\emptyset \neq X \subseteq M, N = (M \setminus X) \cup Y$ and $\forall y \in Y \exists x \in X, y < x$.

For example, if $A = \{a, b, c, d, e\}$ and $b < a$, $d < c$, then $\{a, b, b, b, b, d, d, d, d, d, e\}$ $<_{mul}$ $\{a, a, b, c, d, e\}$ by replacing $X = \{a, c\}$ by $Y = \{b, b, b, d, d, d, d\}$. It can be easily shown that if $<$ is a strict ordering, then so is $<_{mul}$. In such case we talk about *multiset orderings*.

A relation $<$ on a set A is *terminating* if there is no infinite decreasing[1] sequence $x_0 > x_1 > x_2 \ldots$. An important property of multiset relations on finite multisets is that they are terminating when the original relation is terminating, as stated by the following theorem:

THEOREM 1. Let $<$ be a terminating relation on a set A, and $<_{mul}$ the multiset relation induced by $<$ on $\mathcal{M}(A)$. Then $<_{mul}$ is terminating.

The above theorem provides a tool for showing termination of recursive function definitions, by using multisets: show that some multiset measure decreases in each recursive call comparing multisets with respect to the relation induced by a given terminating relation. In the following subsection, we explain how we formalized theorem 1 in the ACL2 logic.

[1] Although not explicitly, we will suppose that the relations given here represent some kind of "smaller than" relation.

1.2 Formalization of well-founded multiset relations in ACL2

Let us deal with formalization of terminating relations in ACL2. A restricted notion of terminating relations is built into ACL2 based on the following meta-theorem (axiom of choice needed): a relation $<$ on a set A is terminating iff there exists a function $F : A \to Ord$ such that $x < y \Rightarrow F(x) < F(y)$, where Ord is the class of all ordinals. In this case, we also say that the relation is *well-founded*. Note that we are denoting the relation on A and the ordering between ordinals using the same symbol $<$. An arbitrary well-founded relation `rel` defined on a set of objects satisfying a property `mp` can be defined in ACL2 as shown below (dots are used to omit technical details, as in the rest of the paper):

```
(encapsulate
 ((mp (x) booleanp) (rel (x y) booleanp) (fn (x) e0-ordinalp))
 ...
 (defthm rel-well-founded-relation-on-mp
   (and (implies (mp x) (e0-ordinalp (fn x)))
        (implies (and (mp x) (mp y) (rel x y))
                 (e0-ord-< (fn x) (fn y))))
   :rule-classes :well-founded-relation))
```

The predicate `mp` recognizes the kind of objects (called *measures*) that are ordered in a well-founded way by `rel`. The *embedding* function `fn` is an order-preserving function mapping every measure to an ordinal. Once a relation is proved to satisfy these properties and the theorem is stored as a well-founded relation rule, it can be used in the admissibility test for recursive functions. We call the theorem `rel-well--founded-relation-on-mp` above the *well-foundedness theorem* for `rel`, `mp` and `fn`. In ACL2, every particular well-founded relation has to be given by means of three functions (a binary relation, a measure predicate and an embedding function) and the corresponding well-foundedness theorem for such functions. As a particular case, when `mp` is `t` we can omit any reference to `mp` in the statement of the corresponding well-foundedness theorem. See `well-founded-relation` in the ACL2 manual [6].

The above notion of termination is restricted: since only ordinals up to ε_0 are formalized in the ACL2 logic, a limitation is imposed on the maximal order type of well-founded relations that can be formalized. Consequently, our formalization suffers from the same restriction (nevertheless, our proofs do not depend on particular properties of ε_0, except well-foundedness).

Let us now deal with formalization of multisets relations. We represent multisets in ACL2 as true lists. Given a predicate `(mp x)` describing a set A, finite multisets over A are described by the following function:

```
(defun mp-true-listp (l)
  (if (atom l)
      (equal l nil)
      (and (mp (car l)) (mp-true-listp (cdr l)))))
```

Note that this function depends on the particular definition of the predicate `mp`. With this representation, different true lists can represent the same multiset: two true lists represent the same multiset iff one is a permutation of the other. Thus, the order in which the elements appear in a list is not relevant, but the number of occurrences of an element is important. This must be taken into account, for example, when defining multiset difference in ACL2 (the function `remove-one`, omitted here, deletes one occurrence of an element from a list, whenever possible):

```
(defun multiset-diff (m n)
  (if (atom n) m (multiset-diff (remove-one (car n) m) (cdr n))))
```

The definition of $<_{mul}$ given in the preceding subsection is quite intuitive but, due to its many quantifiers, difficult to implement. Instead, we will use a somewhat restricted definition, based on the following theorem:

THEOREM 2. Let $<$ be a strict ordering on a set A, and M, N two finite multisets over A. Then $N <_{mul} M$ iff $M \setminus N \neq \emptyset$ and $\forall n \in N \setminus M, \exists m \in M \setminus N,$ such that $n < m$.

From the computational point of view, the main advantage of this alternative definition is that the we do not have to search the multisets X and Y of the original definition because we can take $M \setminus N$ and $N \setminus M$, respectively. It should be remarked that this equivalence is true only when $<$ is a strict partial ordering. Take for example, the relation S defined on $A = \{a, b, c, d\}$, such that dSc, cSb and bSa. In this case we have $\{b, c, d\}S_{mul}\{a, b, c\}$, taking $X = \{a, b, c\}$ and $Y = \{b, c, d\}$. Nevertheless, $d \in \{b, c, d\} \setminus \{a, b, c\}$, $\{a\} = \{a, b, c\} \setminus \{b, c, d\}$ and it is not true that dSa.

Anyway, this is not a severe restriction. Moreover, well-foundedness of $<_{mul}$ also holds when this restricted definition is used, even if the relation $<$ is not transitive, as we will see. Thus, given a defined (or constrained) binary relation `rel`, we define the induced relation on multisets based on this alternative definition:

```
(defun exists-rel-bigger (x l)
  (cond ((atom l) nil)
        ((rel x (car l)) t)
        (t (exists-rel-bigger x (cdr l)))))

(defun forall-exists-rel-bigger (l m)
  (if (atom l)
      t
      (and (exists-rel-bigger (car l) m)
           (forall-exists-rel-bigger (cdr l) m))))

(defun mul-rel (n m)
  (let ((m-n (multiset-diff m n))
        (n-m (multiset-diff n m)))
    (and (consp m-n) (forall-exists-rel-bigger n-m m-n))))
```

Finally, let us see how we can formalize in the ACL2 logic the theorem 1 above, which states well-foundedness of the relation `mul-rel`. As said before, in order to establish well-foundedness of a relation in ACL2, in addition to the relation (`mul-rel` in this case), we have to give the measure predicate and the embedding function, and then prove the corresponding well-foundedness theorem. Since `mul-rel` is intended to be defined on multisets of elements satisfying `mp`, then `mp-true-listp` is the measure predicate in this case. Let us suppose we have defined a suitable embedding function called `map-fn-e0-ord`. Then theorem 1 is formalized as follows:

```
(defthm multiset-extension-of-rel-well-founded
  (and (implies (mp-true-listp x)
                (e0-ordinalp (map-fn-e0-ord x)))
       (implies (and (mp-true-listp x)
                     (mp-true-listp y)
                     (mul-rel x y))
         (e0-ord-< (map-fn-e0-ord x) (map-fn-e0-ord y))))
  :rule-classes :well-founded-relation)
```

In the next subsection we show a suitable definition of `map-fn-e0-ord` and describe some aspects of the ACL2 proof of this theorem.

1.3 A proof of well-foundedness of the multiset relation

In the literature [1], theorem 1 is usually proved using König's lemma: every infinite and finitely branched tree has an infinite path. Nevertheless, we have to find a different proof in ACL2, defining an order-preserving embedding function `map-fn-e0-ord` from `mp-true-listp` objects to `e0-ordinalp` objects. Thus, our proof is based on the following result from ordinal theory: given an ordinal α, the set $\mathcal{M}(\alpha)$ of finite multisets of elements of α (ordinals less than α), ordered by the multiset relation induced by the order between ordinals, is order-isomorphic to the ordinal ω^{α} and the isomorphism is given by the function H where $H(\{\beta_1, \ldots, \beta_n\}) = \omega^{\beta_1} + \ldots + \omega^{\beta_n}$. This result can be proved using Cantor's normal form of ordinals and its properties.

As a by-product, an interesting property about multiset well-founded relations can be deduced. Since $\alpha \leq \varepsilon_0$ implies $\omega^{\alpha} \leq \omega^{\varepsilon_0} = \varepsilon_0$, this means that one can always prove, in the ACL2 logic, well-foundedness of the multiset relation induced by a given well-founded ACL2 relation (i.e., using embeddings in the ordinal ε_0). This is not the case, for example, of lexicographic products, since the maximal ordinal type of a lexicographic product of two ACL2 well-founded relations may be greater than ε_0.

The isomorphism H above suggests the following definition of the embedding function `map-fn-e0-ord`: given a multiset of elements satisfying `mp`, apply `fn` to every element to obtain a multiset of ordinals. Then apply H to obtain an ordinal less than ε_0. If ordinals are represented in ACL2 notation, then the function H can be easily defined, provided that the function `fn` returns always a non-zero ordinal: the function H simply has to sort the ordinals in the multiset and add 0 as the final `cdr`. These considerations lead us to the following definition of the embedding function `map-fn-e0-ord`. Note that the non-zero restriction on `fn` is easily overcome, defining (the macro) `fn1` equal to `fn` except for integers, where `1` is added. In this way `fn1` returns non-zero ordinals for every measure object and it is order-preserving if and only if `fn` is.

```
(defun insert-e0-ord-< (x l)
  (cond ((atom l) (cons x l))
        ((not (e0-ord-< x (car l)))  (cons x l))
        (t (cons (car l) (insert-e0-ord-< x (cdr l))))))

(defun add1-if-integer (x) (if (integerp x) (1+ x) x))
```

```
(defmacro fn1 (x) `(add1-if-integer (fn ,x)))

(defun map-fn-e0-ord (l)
  (if (consp l)
      (insert-e0-ord-< (fn1 (car l)) (map-fn-e0-ord (cdr l)))
    0))
```

Once `map-fn-e0-ord` has been defined, let us now deal with the ACL2 mechanical proof of the well-foundedness theorem for `mul-rel`, `mp-true-listp` and `map-fn-e0-ord` as stated at the end of subsection 1.2 by `multiset-extension-of--rel-well-founded`. The first part of the theorem, which establishes that `(map-fn--e0-ord x)` is an ordinal when `(mp-true-listp x)`, it is not difficult, and can be proved in ACL2 with minor help form the user. The hard part of the theorem is to show that `map-fn-e0-ord` is order-preserving. Here is an informal proof sketch:

Proof sketch: Let us denote, for simplicity, the functions `fn1` and `map-fn-e0-ord`, as f and f_{mul}, and the relation `rel`, `mul-rel` and `e0-ord-<` as $<_{rel}$, $<_{mul}$ and $<$, respectively. Let M and N be two multisets of `mp` elements such that $N <_{mul} M$. We have to prove that $f_{mul}(N) < f_{mul}(M)$. We can apply induction on the number of elements of N. Note that M can not be empty, and if N is empty the result trivially holds. So let us suppose that M and N are not empty. Let $f(x)$, $f(y)$ be the biggest elements of $f[N]$ and $f[M]$, respectively. Note that $f(x)$ and $f(y)$ are the `car` elements of $f_{mul}(N)$ and $f_{mul}(M)$, respectively. Since $f(x)$ and $f(y)$ are ordinals, three cases may arise:

1. $f(x) < f(y)$. Then, by definition of $<$, we have $f_{mul}(N) < f_{mul}(M)$.
2. $f(x) > f(y)$. This is not possible: in that case x is in $N \setminus M$ and by the multiset relation definition, exists z in $M \setminus N$ such that $x <_{rel} z$. Consequently $f(z) > f(x) > f(y)$. This contradicts the fact that $f(y)$ is the biggest element of $f[M]$.
3. $f(x) = f(y)$. In that case, $x \in M$, since otherwise it would exist $z \in M \setminus N$ such that $x <_{rel} z$ and the same contradiction as in the previous case appears. Let $M' = M \setminus \{x\}$ and $N' = N \setminus \{x\}$. We have $N' <_{mul} M'$ and, in addition, $f_{mul}(N')$ and $f_{mul}(M')$ are the `cdr` of $f_{mul}(N)$ and $f_{mul}(M)$, respectively. Induction hypothesis can be applied here to conclude that $f_{mul}(N') < f_{mul}(M')$ and therefore $f_{mul}(N) < f_{mul}(M)$. $\square$

To lead ACL2 to the above informal proof sketch, an induction scheme must be supplied as hint. This is a function suggesting such induction scheme (the function `max-fn1-list`, omitted here, returns the element of a given list with the greatest value of `fn1`):

```
(defun induction-multiset (n m)
  (declare (xargs :measure (acl2::len n)))
  (cond ((atom n) (if (atom m) 1 2))
        ((atom m) 3)
        (t (let* ((max-m (max-fn1-list m)) (max-n (max-fn1-list n))
                  (fn1-max-m (fn1 max-m)) (fn1-max-n (fn1 max-n)))
```

```
            (cond ((equal fn1-max-m fn1-max-n)
                   (if (member max-n m)
                       (induction-multiset (remove-one max-n n)
                                           (remove-one max-n m))
                       5))
                  ((e0-ord-< fn1-max-n fn1-max-m) 6)
                  ((e0-ord-< fn1-max-m fn1-max-n) 7)
                  (t 8))))))
```

Using this induction scheme we proved the following theorem, which is the hard part of the theorem `multiset-extension-of-rel-well-founded`.

```
(defthm map-fn-e0-ord-order-preserving
  (implies (and (mp-true-listp n) (mp-true-listp m)
                (mul-rel n m))
           (e0-ord-< (map-fn-e0-ord n) (map-fn-e0-ord m)))
   :hints (("Goal" :induct (induction-multiset n m)))))
```

The proof of this result was not easy: lemmas to handle each of the cases generated by the above induction scheme have to be proved, obtaining a mechanical proof very close to the previous proof sketch. See the book `multiset.lisp` in the web page for details about the mechanical proof.

Well-foundedness of `mul-rel` has been proved in an abstract framework, without assuming any particular properties of `rel`, `mp` and `fn`, except those concerning well-foundedness. This allows us to functionally instantiate the theorem in order to establish well-foundedness of the multiset relation induced by any given well-founded ACL2 relation. We developed a macro named `defmul` in order to mechanize this process of functional instantiation. The following section describes the macro.

2 The `defmul` macro and the `multiset` book

We defined a macro `defmul` in order to provide a convenient way to define the multiset relation induced by a well-founded relation, and to declare the corresponding well-founded relation rule. We explain now how `defmul` is used.

Let us suppose we have a previously defined (or constrained) relation *my-rel*, which is known to be well-founded on a set of objects satisfying the measure property *my-mp* and justified by the embedding function *my-fn*. That is to say, the following theorem, using variables x and y, has been proved (and stored as a well-founded relation rule):

```
(defthm theorem-name
   (and (implies (my-mp x) (e0-ordinalp (my-fn x)))
        (implies (and (my-mp x) (my-mp y) (my-rel x y))
                 (e0-ord-< (my-fn x) (my-fn y))))
    :rule-classes :well-founded-relation))
```

In order to define the (well-founded) multiset relation induced by *my-rel*, we write the following macro call:

```
(defmul (my-rel theorem-name my-mp my-fn x y))
```

The expansion of this macro generates a number of ACL2 forms. You may use the ACL2 `trans1` command in order to view the translated form of a `defmul` call. The main non-local events generated by this macro call are:

- the definitions needed for the multiset relation induced by *my-rel*: functions `exists-`*`my-rel`*`-bigger`, `forall-exists-`*`my-rel`*`-bigger`, and `mul-`*`my-rel`* analogous to the functions given in subsection 1.2.
- the definition of the multiset measure property, *`my-mp`*`-true-listp`.
- the definition of `map-`*`my-fn`*`-e0-ord`, the embedding function from multisets to ordinals.
- the well-foundedness theorem for `mul-`*`my-rel`*, *`my-mp`*`-true-listp` and `map-`*`my-fn`*`-e0-ord`. This theorem is proved by functional instantiation from `multiset-extension-of-rel-well-founded` and is named `multiset-extension-of-`*`my-rel`*`-well-founded`

We expect `defmul` to work without assistance from the user. After the above call to `defmul`, the function `mul-`*`my-rel`* is defined as a well-founded relation on multisets of elements satisfying the property *my-mp*, induced by the well-founded relation *my-rel*. From this moment on, `mul-`*`my-rel`* can be used in the admissibility test for recursive functions to show that the recursion terminates.

To know the list of names we need to supply to a `defmul` call, we have developed a tool to extract the information from the ACL2 world and print it. This macro is simply called in this way:

```
(defmul-components my-rel )
```

This is only an informative tool, not a event. This macro looks up the ACL2 world, and returns the list of names that are needed in the `defmul` call for *my-rel*.

We have divided the results and tools about multisets into two books. The book `multiset.lisp` contains the proof of the theorem `multiset-extension-of-rel-well-founded` shown in subsection 1.3. Names in this book are defined in the package `MUL`. The book `defmul.lisp` contains the macro definitions of `defmul` and `defmul-components` and includes the `multiset` book. See the web page for details.

We have also included some non-local rules in `multiset.lisp`, which helped us to prove the three examples presented in this paper, and we think they are general enough to assist in other cases. Two relevant examples of these additional results and tools are:

- We have defined the function `equal-set` as an equivalence relation. This function implements equality for sets, not for multisets, but it turned out useful in our case studies because it can be proved to be a congruence with respect to both arguments of `forall-exists-`*`my-rel`*`-bigger`:

  ```
  (defun equal-set (x y) (and (subsetp x y) (subsetp y x)))

  (defequiv equal-set)

  (defcong equal-set iff forall-exists-my-rel-bigger l m 1)

  (defcong equal-set iff forall-exists-my-rel-bigger l m 2)
  ```

  Since the latter two congruence rules depend on the particular definition of *my-rel*, they are generated by every particular call to `defmul`.

– We also define a meta rule to deal with difference of multisets represented by lists with final common suffix. This rule rewrites expressions of the form

(multiset-diff (list* x_1 $x_2 \ldots x_m$ l) (list* y_1 $y_2 \ldots y_k$ l))

to the following equivalent expression (with respect to `equal-set`):

(multiset-diff (list x_1 $x_2 \ldots x_m$) (list y_1 $y_2 \ldots y_k$))

This meta rule is very useful[2] when proving that a particular multiset measure decreases in every recursive call of a function: it is "usual" that the multiset obtained measuring the arguments of a recursive call is a list with the same final part than the multiset obtained measuring the arguments in the original call.

3 Case studies using multiset relations

In the next subsections, we show three examples where well-founded multiset relations play an important role in the ACL2 proof of non-trivial termination properties. The first example is taken from [4]. We use a multiset relation to show termination of a tail-recursive version of Ackermann's function. In the second example, also taken from [4], we use a multiset relation to admit an iterative version of McCarthy's 91 function. The third example is a proof of Newman's lemma for abstract reduction systems: every terminating and locally confluent reduction relation has the Church-Rosser property. This last example is part of a larger project developed by the authors in order to formalize some aspects of equational reasoning using ACL2 [8, 9].

All the examples show one function whose termination is proved using a well-founded multiset relation and a multiset measure function. When the function is presented for the first time, its code is commented (using semicolons), to emphasize that a suitable measure has still to be given in order to pass the admissibility test.

3.1 A tail-recursive version of Ackermann's function

The following is the standard definition of Ackermann's function in ACL2:

```
(defun ack (m n)
  (declare (xargs :measure (cons (+ (nfix m) 1) (nfix n))))
  (cond ((zp m) (+ n 1))
        ((zp n) (ack (- m 1) 1))
        (t (ack (- m 1) (ack m (- n 1))))))
```

We now try to define the following iterative program to compute Ackerman's function:

```
; (defun ack-it-aux (S z)
;   (if (endp S)
;       z
;     (let ((head (first S))
```

[2] Due to a bug in ACL2 version 2.5, this meta rule fails to be applied. This is not the case in version 2.4, where we originally developed this work. The problem in version 2.5 can be fixed by including a patch that will be included in Version 2.6. Thanks to Matt Kaufmann for the patch.

```
;            (tail (rest S)))
;        (cond ((zp head) (ack-it-aux tail (+ z 1)))
;              ((zp z) (ack-it-aux (cons (- head 1) tail) 1))
;              (t (ack-it-aux (cons head (cons (- head 1) tail))
;                             (- z 1)))))))

; (defun ack-it (m n) (ack-it-aux (list m) n))
```

The intended behavior of the function `ack-it-aux` is that in every iterative step `(ack-it-aux S z) = (ack` s_k `(ack` s_{k-1} `... (ack` s_1 `z)))`, where `S` is a stack with k elements, $(s_1 \ldots s_k)$. Therefore, it can be proved (and we did) that `(ack m n)` is equal to `(ack-it m n)`.

A proof of termination of `ack-it-aux` may be difficult. Note that in the third recursive call the stack increases its number of elements while the second argument decreases. Nevertheless in the first and the second recursive calls, the second argument increases, although the stack does not increase its number of elements.

As shown in [4], a multiset measure can be used to prove termination of `ack--it-aux`. In this case, we use multisets of pairs of natural numbers, where pairs are supposed to be ordered by the lexicographic product of the usual order between naturals. The measure associated to arguments $\mathtt{S} = (s_1 \ldots s_k)$ and `z` is the multiset $\{(s_1, \mathtt{z}), (s_2+1, 0) \ldots, (s_k+1, 0)\}$.

Using `defmul`, we can easily replay in ACL2 the proof given in [4]. First of all, we define the well-founded relation on pairs of natural numbers, called here `rel-ack`. This can be done by the following sequence of events:

```
(defun rel-ack (p1 p2)
  (cond ((< (car p1) (car p2)) t)
        ((= (car p1) (car p2)) (< (cdr p1) (cdr p2)))))

(defun mp-ack (p)
  (and (consp p)
       (integerp (car p)) (>= (car p) 0)
       (integerp (cdr p)) (>= (cdr p) 0)))

(defun fn-ack (p) (cons (+ 1 (car p)) (cdr p)))

(defthm rel-ack-well-founded
  (and (implies (mp-ack x)
                (e0-ordinalp (fn-ack x)))
       (implies (and (mp-ack x) (mp-ack y) (rel-ack x y))
                (e0-ord-< (fn-ack x) (fn-ack y))))
  :rule-classes :well-founded-relation)
```

We define the well-founded multiset relation induced by `rel-ack` on multisets of pairs of natural numbers, using the following `defmul` call:

```
(defmul (rel-ack rel-ack-well-founded mp-ack fn-ack x y))
```

Now we have defined the function `mul-rel-ack` as a well-founded relation with measure property `mp-ack-true-listp` and embedding function `map-fn-ack-e0-ord`.

The relation `mul-rel-ack` can be used as a well-founded relation in the the admissibility test for the function `ack-it-aux`, with a suitable measure function. The function `measure-ack-it-aux` implements the multiset measure sketched above, using the auxiliary function `get-pairs-add1-0`:

```
(defun get-pairs-add1-0 (S)
  (if (endp S)
      nil
    (cons (cons (+ (nfix (car S)) 1) 0) (get-pairs-add1-0 (cdr S)))))

(defun measure-ack-it-aux (S z)
  (if (endp S)
      nil
      (cons (cons (nfix (car S)) (nfix z))
            (get-pairs-add1-0 (cdr s)))))
```

We can now prove termination of `ack-it-aux`, giving `mul-rel-ack` as well-founded relation and `measure-ack-it-aux` as measure function:

```
(defun ack-it-aux (s z)
  (declare (xargs :measure (measure-ack-it-aux s z)
                  :well-founded-relation mul-rel-ack
                  :hints ....))
  (if (endp s)
      z
    (let ((head (first s))
          (tail (rest s)))
      (cond ((zp head) (ack-it-aux tail (+ z 1)))
            ((zp z) (ack-it-aux (cons (- head 1) tail) 1))
            (t (ack-it-aux (cons head (cons (- head 1) tail))
                           (- z 1)))))))
```

Given the measure and the well-founded relation in the definition of `ack-it-aux`, the proof of its termination is not difficult, and only a very few previous lemmas are needed, in order to prove that the multiset measure given decreases in each recursive call. See the book `ackermann.lisp` in the web page for details. Moreover, after the admission of the definition we can define the function `ack-it` as shown above, and finally prove in ACL2 the following theorem:

```
(defthm ack-it-equal-ack
   (equal (ack-it m n) (ack m n)))
```

3.2 McCarthy's 91 function

This example is taken from [4] and shows admissibility of an iterative version of the recursive definition of McCarthy's 91 function. For a detailed treatment (in ACL2) of McCarthy's 91 function and its generalization given by Knuth, we urge the interested reader to consult the work of Cowles [3], where proofs are done over arbitrary archimedian fields. Our intention here is only to show how multisets can help to prove a non-trivial termination property.

The "91 function" is a function acting on integers, originally given by McCarthy by the following recursive scheme:

```
(defun mc (x)
  (declare (xargs :mode :program))
  (cond ((not (integerp x)) x)
        ((> x 100) (- x 10))
        (t (mc (mc (+ x 11))))))
```

See [3] for a description of ACL2's resistance to accept this definition (in logic mode). Instead, we try to define the following iterative version of that recursive scheme, as given by the following functions:

```
; (defun mc-aux (n z)
;   (cond ((or (zp n) (not (integerp z))) z)
;         ((> z 100) (mc-aux (- n 1) (- z 10)))
;         (t (mc-aux (+ n 1) (+ z 11)))))

; (defun mc-it (x) (mc-aux 1 x))
```

As we will show, the recursive algorithm given by `mc-it` and `mc-aux` is a somewhat complicated way to compute the following function:

```
(defun f91 (x)
  (cond ((not (integerp x)) x)
        ((> x 100) (- x 10))
        (t 91)))
```

The intended behavior of the function `mc-aux` is that in every iterative step `(mc-aux n z)`= `(f91 (f91` $.\overset{n}{.}.$`(f91 z)))` and, consequently, `(mc-it x)`=`(f91 x)`. Proving termination of `mc-aux` may be difficult: note the different behavior of the two recursive calls. In [4], a multiset measure is given to justify termination of the function: every recursive call of `(mc-aux n z)` is measured with the following multiset: $\{$`z, (f91 z), (f91 (f91 z))`$, \ldots,$ `(f91 (f91` $\overset{n-1}{\ldots}$`(f91 z)))`$\}$, and multisets are compared with respect to the multiset relation induced by the "greater-than" relation defined for integers equal [3] or less than `111`. In the sequel, we describe how ACL2 is guided to this termination argument.

First, we define the well-founded relation `rel-mc` that will be extended later to a multiset relation. Note that in this case, the measure property is `t`, although only integers under `111` are comparable with respect to `rel-mc`. One could think that `integerp-<=-111` should be the measure property of the well-founded relation, instead of `t`. But there is a subtle difference: the multiset measure we will define can contain elements greater than `111`, although those elements are not comparable w.r.t. `rel-mc`. The following sequence of events defines `rel-mc` and stores it as a well founded relation:

[3] Performing the ACL2 proof, we discovered a minor bug in the proof given in [4]: it is necessary to consider integers equal or less than `111`, and not only strictly less than `111`.

```
(defun integerp-<=-111 (x)
  (and (integerp x) (<= x 111)))

(defun rel-mc (x y)
  (and (integerp-<=-111 x) (integerp-<=-111 y) (< y x)))

(defun fn-mc (x)
  (if (integerp-<=-111 x) (- 111 x) 0))

(defthm rel-mc-well-founded
  (and (e0-ordinalp (fn-mc x))
       (implies (rel-mc x y)
                (e0-ord-< (fn-mc x) (fn-mc y))))
  :rule-classes :well-founded-relation)
```

We define the well-founded multiset relation induced by `rel-mc` on multisets (`true-listp` objects in this case), using the following `defmul` call:

```
(defmul (rel-mc rel-mc-well-founded t fn-mc x y))
```

Through this macro call, we have defined the well-founded relation `mul-rel-mc` (with measure property `true-listp` and embedding function `map-fn-mc-e0-ord`), allowing us to use it in the admissibility test for the function `mc-aux`, with the measure function given above, and implemented by the function `measure-mc-aux`:

```
(defun measure-mc-aux (n z)
  (if (zp n) nil (cons z (measure-mc-aux (- n 1) (f91 z)))))
```

We can now define the function `mc-aux`, giving `mul-rel-mc` and `measure-mc-aux` as the well-founded relation and measure function to be used, respectively:

```
(defun mc-aux (n z)
  (declare (xargs :measure (measure-mc-aux n z)
                  :well-founded-relation mul-rel-mc))
  (cond ((or (zp n) (not (integerp z))) z)
        ((> z 100) (mc-aux (- n 1) (- z 10)))
        (t (mc-aux (+ n 1) (+ z 11)))))
```

The function is admitted with a minor help from the user (surprisingly, only one specific lemma is needed). See the book `mccarthy-91.lisp` in the web page for details. After this definition we can define the function `mc-it` as above, and show that verifies the original recursion scheme given by McCarthy. Moreover, we can even prove very easily that `mc-it` is equal to `f91` (previously proving a suitable generalization, as sketched above):

```
(defthm mc-it-recursive-schema
  (equal (mc-it x)
         (cond ((not (integerp x)) x)
               ((> x 100) (- x 10))
```

```
                (t (mc-it (mc-it (+ x 11)))))))

(defthm mc-it-equal-f91
  (equal (mc-it x) (f91 x)))
```

3.3 Newman's lemma

Abstract reduction systems: Newman's lemma is a result about abstract reduction systems, which plays an important role in the study of decidability of certain equational theories. We give a short introduction to basic concepts and definitions from abstract reductions. See [1] for more details.

Reductions system are simply an abstract formalization of step by step activities, such as the execution of a computation, the gradual transformation of an object until some normal form is reached, or the traversal of some directed graph. The term "reduction" gives the intuition that an element of less complexity is obtained in every step. Formally speaking, an *abstract reduction* is simply a binary relation $\rightarrow$ defined on a set A. We will denote as $\leftarrow$, $\leftrightarrow$, $\overset{*}{\rightarrow}$ and $\overset{*}{\leftrightarrow}$ respectively the inverse relation, the symmetric closure, the reflexive-transitive closure and the equivalence closure. The following concepts are defined with respect to a reduction relation $\rightarrow$. We say that x and y are *equivalent* if $x \overset{*}{\leftrightarrow} y$. We say that x and y are *joinable* (denoted as $x \downarrow y$) if there exists u such that $x \overset{*}{\rightarrow} u \overset{*}{\leftarrow} y$. An element x is in *normal form* (or *irreducible*) if there is no z such that $x \rightarrow z$.

A reduction relation has the *Church-Rosser property* if every two equivalent elements are joinable. An equivalent property is *confluence*: for all x, u, v such that $u \overset{*}{\leftarrow} x \overset{*}{\rightarrow} v$, then $u \downarrow v$. In a reduction relation with the Church-Rosser property, two distinct elements in normal form cannot be equivalent. A reduction relation is *normalizing* if every element has an equivalent normal form (denoted as $x \downarrow$). Obviously, every terminating (as defined in subsection 1.1) reduction is normalizing. Church-Rosser and normalizing reduction relations have a nice property: provided normal forms are computable and identity in A is decidable, then the equivalence relation $\overset{*}{\leftrightarrow}$ is decidable. This is due to the fact that, in that case, $x \overset{*}{\leftrightarrow} y$ iff $x\downarrow = y\downarrow$, for all $x, y \in A$.

Confluence can be localized when the reduction is terminating. In that case, an equivalent property is *local confluence*: for all x, u, v such that $u \leftarrow x \rightarrow v$, then $u \downarrow v$. The following theorem, named Newman's lemma, states this:

THEOREM 3 **(Newman's lemma)**. Let $\rightarrow$ be a terminating and locally confluent reduction relation. Then $\rightarrow$ is confluent.

This result allows to make easier the study of confluence (or equivalently, of the Church-Rosser property) for terminating reduction relations. One has only to deal with joinability of local divergences. This is crucial in the development of completion algorithms for term rewriting systems in order to obtain decision procedures for equational theories [1].

Formalization of Newman's lemma in ACL2: Every reduction relation has two important aspects. On the one hand, a declarative aspect, since every reduction relation describes its equivalence closure. On the other hand, a computational aspect, describing a stepwise activity, a gradual transformation of objects until (eventually) a normal form is reached. Thus, if $x \to y$, the point here is that y is obtained from x by applying some kind of transformation or *abstract operator*. In its most abstract formulation, we can view a reduction as a binary function that, given an element and an operator, returns another element, performing a *one-step reduction*. Of course not any operator can be applied to any element: we need a boolean binary function to test if it is *legal* to apply an operator to an element.

The discussion above leads us to formalize a general abstract reduction relation using two partially defined functions: `reduce-one-step` and `legal`; `(reduce-one-step x op)` represents a one-step reduction applying operator `op` to `x`, and `(legal x op)` represents a test to check if the operator `op` may be applied to `x`[4]. It should be remarked that no predicates are used to recognize neither operators nor elements, thus ensuring abstractness.

These two functions are introduced using `encapsulate`. In order to formalize Newman's lemma, additional properties are included to assume termination and local confluence of the reduction relation, encoding in this way the assumptions of the theorem we want to prove. This is shown in figure 1. In the following, we describe in detail the events appearing in it.

Before describing how we formalized termination and local confluence, we show how we can define the equivalence closure of a reduction relation. In order to define $x \overset{*}{\leftrightarrow} y$, we have to include an argument with a sequence of steps $x = x_0 \leftrightarrow x_1 \leftrightarrow x_2 \ldots \leftrightarrow x_n = y$. An *abstract proof* (or simply, a *proof*) is a sequence of legal steps and each proof step is a structure[5] `r-step` with four fields: `elt1`, `elt2` (the elements connected), `direct` (a boolean value indicating if the step is direct or inverse) and `operator`:

```
(defstructure r-step direct operator elt1 elt2)
```

A proof step is *legal* if one of its elements is obtained by applying the (legal) operator to the other, in the sense indicated. The function `proof-step-p` implements this concept. The function `equiv-p` implements the equivalence closure of our abstract reduction relation: `(equiv-p x y p)` checks if `p` is a proof justifying that `x`$\overset{*}{\leftrightarrow}$`y`. See the definitions of `proof-step-p` and `equiv-p` in item `(b)` of figure 1.

Two proofs justifying the same equivalence will be said to be *equivalent*. We hope it will be clear from the context when we talk about abstract proofs objects and proofs in the ACL2 system.

Let us now see how can we formalize termination. Our formalization is based on the following meta-theorem: a reduction is terminating if and only if it is contained in a well-founded partial ordering (axiom of choice needed). Thus, let `rel`[6] be a given general well-founded partial order, as defined in item `(a)` of figure 1.

This well-founded partial order `rel` will be used to state termination of the general reduction relation defined, by assuming that every legal reduction step returns a smaller object, with respect to `rel`. See item `(b)` in figure 1 for a statement of this assumed property.

The Church-Rosser property and local confluence can be redefined with respect to the form of a proof. We define (omitted here) functions to recognize proofs with

---

[4] In [9] a third function `reducible` is introduced, in order to formalize computation of normal forms. Nevertheless, in the proof of Newman's lemma we don't need to deal with normal forms.

[5] We used the `defstructure` tool developed by Bishop Brock [2].

[6] Conflicts with names used in the `multiset.book` are avoided using packages.

```
;;; (a) A well-founded partial order:
(encapsulate
 ((rel (x y) t) (fn (x) t))
...
 (defthm rel-well-founded-relation
   (and (e0-ordinalp (fn x))
        (implies (rel x y) (e0-ord-< (fn x) (fn y))))
   :rule-classes (:well-founded-relation :rewrite))

 (defthm rel-transitive
   (implies (and (rel x y) (rel y z)) (rel x z))))

;;; (b) A terminating and locally confluent reduction relation:
(encapsulate
 ((legal (x u) boolean) (reduce-one-step (x u) element)
  (reducible (x) boolean) (transform-local-peak (x) proof))
....
 (defun proof-step-p (s)
   (let ((elt1 (elt1 s)) (elt2 (elt2 s))
         (operator (operator s)) (direct (direct s)))
     (and (r-step-p s)
          (implies direct (and (legal elt1  operator)
                               (equal (reduce-one-step elt1 operator)
                                      elt2)))
          (implies (not direct) (and (legal elt2 operator)
                                     (equal (reduce-one-step elt2 operator)
                                            elt1))))))

 (defun equiv-p (x y p)
   (if (endp p)
       (equal x y)
     (and (proof-step-p (car p)) (equal x (elt1 (car p)))
          (equiv-p (elt2 (car p)) y (cdr p)))))

 (defthm terminating
   (implies (legal x op) (rel (reduce-one-step x op) x)))

 (defthm locally-confluent
   (let ((valley (transform-local-peak p)))
     (implies (and (equiv-p x y p) (local-peak-p p))
              (and (steps-valley valley) (equiv-p x y valley))))))
```

Fig. 1. Assumptions of Newman's lemma

particular shapes (*valleys* and *local peaks*): `local-peak-p` recognizes proofs of the form $v \leftarrow x \rightarrow u$ and `steps-valley` recognizes proofs of the form $v \stackrel{*}{\rightarrow} x \stackrel{*}{\leftarrow} u$.

To deal with the assumption of local confluence, note that a reduction is locally confluent iff for every local peak proof there is an equivalent valley proof. Therefore, in order to state local confluence of the general reduction relation defined, we assume the existence of a function `transform-local-peak` which returns a valley proof for every local peak proof. See again item `(b)` in figure 1 for a statement of this assumed property.

Having established the assumptions, in order to prove Newman's lemma we must show confluence of this general reduction relation assumed to be terminating and locally confluent. Instead of confluence, we prove the Church-Rosser property, which is equivalent. Therefore, we must prove that for every proof there exists an equivalent valley proof, i.e., *we have to define* a function `transform-to-valley` and prove that `(transform-to-valley p)` is a valley proof equivalent to `p`. This is the statement of Newman's lemma:

```
(defthm Newman-lemma
  (let ((valley (transform-to-valley p)))
    (implies (equiv-p x y p)
             (and (steps-valley valley) (equiv-p x y valley)))))
```

A suitable definition of `transform-to-valley` and a proof of this theorem in ACL2 is shown in the following subsection. The hard part of the proof is to show termination of `transform-to-valley`. It will be done with the help of a well-founded multiset relation.

An ACL2 proof of Newman's lemma: The proof commonly found in the literature [1], is done by well-founded induction on the terminating reduction relation. Our approach is more constructive and is based on a proof given in [7]. We have to define a function `transform-to-valley` which transforms every proof in a equivalent valley proof. For that purpose, we can use the function `transform-local-peak`, assumed to transform every local peak proof in a equivalent valley proof. Thus, the function we need is defined to iteratively apply `replace-local-peak`, (which replaces the first local peak subproof by the equivalent subproof given by `transform-local-peak`) until there are no local peaks (checked by `exists-local-peak`). The following is the definition of `transform-to-valley` (we omit here the definition of `replace-local-peak` and `exists-local-peak`):

```
;(defun transform-to-valley (p)
;  (if (not (exists-local-peak p))
;      p
;    (transform-to-valley (replace-local-peak p))))
```

This function is not admitted without help from the user. The reason is that when a local peak in a proof is replaced by an equivalent valley subproof, the length of the proof obtained may be larger than the length of the original proof. Nevertheless, the key point here is that every element of the new subproof is smaller (w.r.t. the well-founded relation `rel`) than the greatest element of the local peak. If we measure a proof as the multiset of the elements involved in it, then replacing a local peak subproof by an equivalent valley subproof, we obtain a proof with smaller measure with respect to the well-founded multiset relation induced by `rel`. The function `proof-measure` returns this measure for a given proof: it collects the `elt1` elements of every proof step in a proof.

```
(defun proof-measure (p)
  (if (endp p)
      nil
    (cons (elt1 (car p)) (proof-measure (cdr p)))))
```

Using `defmul`, we define the well-founded relation `mul-rel`, induced by the well-founded relation `rel` introduced in the previous subsection:

```
(defmul (rel rel-well-founded-relation-on-mp t fn x y))
```

The main result we proved states that the proof measure decreases (with respect to the well-founded relation `mul-rel`) if a local-peak is replaced by an equivalent valley subproof:

```
(defthm transform-to-valley-admission
  (implies (exists-local-peak p)
           (mul-rel (proof-measure (replace-local-peak p))
                    (proof-measure p)))
  :rule-classes nil)
```

With this theorem, admission of the function `transform-to-valley` is now possible, giving a suitable hint:

```
(defun transform-to-valley (p)
  (declare (xargs :measure (proof-measure p)
                  :well-founded-relation mul-rel
                  :hints
                    (("Goal" :use
                      (:instance transform-to-valley-admission)))))
  (if (not (exists-local-peak p))
      p
    (transform-to-valley (replace-local-peak p))))
```

Once `transform-to-valley` is admitted (which is the hard part of the theorem), the following two theorems are proved, and this trivially implies Newman's lemma as stated at the end of subsection 3.3.

```
(defthm equiv-p-x-y-transform-to-valley
  (implies (equiv-p x y p)
           (equiv-p x y (transform-to-valley p))))

(defthm valley-transform-to-valley
   (implies (equiv-p x y p)
            (steps-valley (transform-to-valley p))))
```

The mechanical proof of Newman's lemma is the most difficult of the three examples presented here. Lemmas have to be proved to simplify the multiset differences appearing in the conjecture generated by the termination proof of `transform-to--valley`. See the file `newman.lisp` in the web page for details. We also provide books proving decidability of the equivalence relation generated by a terminating and locally confluent reduction relation. To see how this result can be exported to the study of equational theories, see [9].

4 Conclusions

We have presented a formalization of multiset relations in ACL2, showing how they can be used as a tool for proving non-trivial termination properties of recursive functions in ACL2. We have defined the multiset relation induced by a given relation and proved a theorem establishing well-foundedness of the multiset relation induced by a well-founded relation. This theorem is formulated in an abstract way, so that functional instantiation can be used to prove well-foundedness of concrete multiset relations.

We have presented also a macro named `defmul`, implemented to provide a convenient tool to define well-founded multiset relations induced by well-founded relations. This macro allows the definition of these multiset relations in a single step.

Three case studies are presented, to show how this tool can be useful in obtaining proofs of non-trivial termination properties of functions defined in ACL2. The first case study is the definition of a tail-recursive version of Ackermann's function. The second is the admissibility of a definition of McCarthy's 91 function, and a study of its properties. The third is a proof of Newman's lemma for abstract reduction relations.

This work arose as part of a larger project, trying to formalize properties of abstract reduction relations, equational theories and term rewriting systems [8, 9]. In that work, ACL2 is used as a meta-logic to study properties of a formal proof system, namely equational logic. Newman's lemma is a key result needed to prove decidability of equational theories given by complete term rewriting systems [1]. Once formalized multiset relations and used in the proof of Newman's lemma, we decided to make a tool (`defmul`) which allowed to export the results on multisets to other contexts. To test this implementation, we applied it to two examples described in [4]: Ackermann's function and McCarthy's 91 function.

Further work has to be done to provide a good library of lemmas to handle multisets and their operations. We plan also to improve the use of `defmul`, in order to provide only the name of the well-founded relation, avoiding to give the functions, variables and event associated with it. Updated versions of the books will be in the web page.

The examples presented here are all of a theoretical nature. Nevertheless, a remark given at the end of section III in [4], pointing an heuristic procedure for proving termination of loops using multisets, suggests that this kind of orderings could be applied to a wider class of termination problems and that the search for a suitable multiset measure could be mechanized to some extent. Another application of multisets orderings is to provide the basis for some proofs of termination of term rewriting systems [1]. We intend to make further research following these two lines.

References

1. Baader, F., and Nipkow, T. *Term rewriting and all that.* Cambridge University Press, 1998.
2. Brock, B. `defstructure` for ACL2 version 2.0. Technical Report, 1997.
3. Cowles, J. Knuth's generalization of McCarthy's 91 function. In *Computer-Aided Reasoning: ACL2 Case Studies*, M. Kaufmann, P. Manolios, and J S. Moore, Eds. Kluwer Academic Publishers, 2000, ch. 17.
4. Dershowitz, N., and Manna, Z. Proving termination with multiset orderings. In *Annual International Colloquium on Automata, Languages and Programming* (1979), H. Maurer, Ed., no. 71 in LNCS, Springer-Verlag, pp. 188–202.

5. Kaufmann, M., Manolios, P., and Moore, J S. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, 2000.
6. Kaufmann, M., and Moore, J S. ACL2 version 2.5. http://www.cs.utexas.edu/users/moore-/acl2/acl2-doc.html, 2000.
7. Klop, J. Term rewriting systems. *Handbook of Logic in Computer Science* (1992).
8. Ruiz-Reina, J., Alonso, J., Hidalgo, M., and Martín, F. http://www-cs.us.es/~jruiz/acl2-rewr. Formalizing equational reasoning in the ACL2 theorem prover, 2000.
9. Ruiz-Reina, J., Alonso, J., Hidalgo, M., and Martín, F. Formalizing rewriting in the ACL2 theorem prover. In *Proceedings of AISC'2000 (Fifth International Conference Artificial Intelligence and Symbolic Computation)* (to appear), LNCS, Springer Verlag.