L R I

RAPPORT DE RECHERCHE

A CONSTRUCTION OF SEVERAL DEFINITIONS RECURSIVE OVER THE VARIABLE UNDER THE EXPONENT FOR THE EXPONENT FUNCTION

FRANOVA M

Unité Mixte de Recherche 8623
CNRS-Université Paris Sud – LRI

06/2009

Rapport de Recherche N° 1519

CNRS – Université de Paris Sud
Centre d'Orsay
LABORATOIRE DE RECHERCHE EN INFORMATIQUE
Bâtiment 490
91405 ORSAY Cedex (France)

A Construction of Several Definitions Recursive over the Variable under the Exponent for the Exponent function

Marta Fraňová

mf@lri.fr, http://www.lri.fr/~mf/index.html

UMR 8623 CNRS - Université Paris Sud
LRI, Bât. 490, 91405 Orsay Cedex, France

1 Introduction

This paper presents a construction of several definitions recursive over the variable under the Exponent for the Exponent function. The focus is on guiding a constructive proof of two specification theorems. This guidance is provided by our *Constructive Matching Methodology*. The main technique is the *CM-formula construction* the most complete presentation of which can be found in *Constructive Matching methodology and automatic plan-construction revisited*; Rapport de Recherche No.874, L.R.I., Univ. de Paris-Sud, Orsay, France, November, 1993.

For user's convenience we recall here some basic knowledge used as well as some strategic questions inherent to the inductive proof's construction.

1.1 Some basic knowledge

1. Skolem's function corresponding to a specification theorem.
Let X be the vector of input variables, and Z the vector of output variables of the relation R(X,Z). Suppose we prove that $\forall$X $\exists$Z R(X,Z). Then, the function that realizes the computation of Z is called the Skolem's function of X, Sf(X), and is defined by: Sf(X) = Z and $\forall$X R(X,Sf(X)). In the following, we shall stress the computational quality of the Skolem's function by naming it "aux-i" since it defines an auxiliary function needed to solve our problem.

Performing a proof by recursion consists in the analysis of what are called "base case" and "general case" and to apply to the general case what is called the "induction hypothesis." Since we restrict ourselves here to the natural numbers, the base case is x = 1 since the simplest formula we start from includes no conditions on x. The general case is x = n+1 where '+1' is the successor function. The induction hypothesis ("if the formula is assumed to be true for n, then we can prove that it is true for n+1") writes $\exists$e R(n,e) $\Rightarrow$ $\exists$z R(n+1,z) or else, using a Skolem function, R(n,Sf(n)) $\Rightarrow$ R(n+1,Sf(n+1)).

2. Trivial recurrence
In order to simplify notations, suppose that we study a binary function f(n,y) where 'n' is the induction variable. Suppose further that we find out that f(1,y) = y+a and f(n+1,y) = f(n,y), where a is a constant. Then recurrence is removed since we know that $\forall$n f(n,y) = y + a. If f(n+1,y) = f(n,y)+1, then $\forall$n f(n,y) = n+(y+a).

1.2 Jump operator

Suppose we are in a situation where, in some way, a generalized 'less than', $\angle$, operator can be defined. If we know that x$\angle$ y, in Natural Numbers it is possible to state that $\exists$z, x + z = y, and the proof on this existential theorem will provide us the value of z. Here are three trivial but important cases where we apply this 'jump operator.

Case 1. Since the exponential function is increasing, we know the $\forall x\ \forall y$ exp1(x,y) $\angle$ exp1(x,y+1). Using the jump operator, we know that $\forall x\ \forall y\ \exists z$ exp1(x,y+1) = exp1(x,y) + z is true and that proving this theorem will enable us to build a suitable 'z'.

Case 2. Another important, if not obvious, case of application of the 'jump operator' occurs when we observe that applying the induction hypothesis is not possible in a formula that is an equation. Our solution is to generate an intermediary lemma. Let us write the induction hypothesis as $h_l(n,X) = h_r(n,X')$, where X and X' are vector variables. The general case is given by $c_l(n+1,X) = c_r(n+1,X')$. When the induction hypothesis cannot be directly applied to the generated case, we generate the intermediary lemma $\forall n\ \forall X\ \exists z\ c_l(n+1,X) = h_l(n,X) + z$.

1.3 What and when is it necessary to evaluate?

Our methodology is designed to avoid a systematic use of all the properties of the functions implied by the defining axioms (such as, for instance, the commutative law for the addition function). The reason of this restraint is that such a use may impair the possibility of generalizations for completing the proof or may complicate *ad absurdum* the application of the induction hypothesis. This is why we do apply systemically the defining axioms and the induction hypothesis. For the last, we even go up to 'inventing' an intermediary lemma to transform considered expressions so that to the new equivalent instances the induction hypothesis may be applied.

Thus, we can always introduce intermediary lemmas that will, in a sense, prove again the property needed exactly when it is needed. The price to pay for the application of this choice is somewhat a cumbersome accumulation of intermediary lemmas. However, the strategy of a systematic use of a library of properties leads to the complexity of handling the search in such a library as well as the necessity of handling several situations in parallel, when several properties can be used. For instance, using the well-known properties of the addition, the expression a+(b+c) can be transformed into (b+c)+a, but also to (a+b)+c, (a+c)+b, and so on. Our focusing on the defining axioms avoids such situations.

1.4 Instances of failure analysis

The failure analysis is a whole domain of research. We shall here give a simple example of failure analysis, namely, spotting the variables that can or cannot be used as induction variables.

In section 1.5, our motivation for attempting a generalization is that no variable is suitable to become the induction variable. This is quite obvious when we observe a recursive definition. For instance, if we define the addition by

A3: 0 + y = y

A4: (n+1) + y = (n + y)+1

then the variable occurring as the first argument, here 'n' is the variable suitable to become the induction variable, not y. This is slightly less obvious when we consider

function embedded into another one.
For example, in the term

$$(n + y) + (m + z),$$

'n' is indeed suitable to become the induction variable. Inversely, if we attempt to use 'm' as the induction variable, the general case will include the study of m = (m'+1). Thus the considered term will become

$$(n + y) + ((m'+1) + z),$$

and by evaluation, using A4, (n + y) + ((m'+z) + 1), and we gain nothing more that could help us to apply the induction hypothesis that shall contain the expression (n + y) + (m' + z). The only solution we will have is to generate a new intermediary lemma enabling us to apply the induction hypothesis. Such a lemma, using the jump constructor + (see section 1.2), would be of the form

$$\forall n\ \forall y\ \forall m'\ \forall z\ \ \exists w\ (n + y) + ((m'+z) + 1) = ((n + y) + (m' + z)) + w$$

Thus, when we consider the term (n + y) + (m + z), m is not suitable to become the induction variable.

When no variable is suitable for becoming the induction variable, we try to generalize the formula under study (see section 1.5). It may happen that no generalization is obvious or, as it is often the case, the generalization also shows that no variable is suitable for becoming the induction variable. Since we have no other choice, we accept to choose one induction variable, knowing ahead of time that this leads us to introduce yet other intermediary lemmas. Such lemmas will, in general, hide an effort to obtain the new defining axioms for the given functions.

1.5 Generalization

The problem of finding a proper generalization is also met, for instance, when none of the variables of the formula to prove is a good candidate to become induction variable. We shall illustrate with more details in section 3 the way we react to this situation. The position we adopt here is that this situation is a hint to suggest that a generalization is necessary for the proof to go on.

We must at first acknowledge that spotting possible generalizations (namely, terms that are repeated within the formula to prove) is quite trivial. The difficult problem is the one spotting a non absurd one. This section will describe rapidly the method we suggest in order to provide an informal proof of the validity of a generalization. This problem is extremely difficult and has been worked upon some 30 year ago by (Kodratoff, 1979), (Arsac et al., 1982). The solution proposed can hold in this simple sentence: "Try to obtain recurrence relations among the variables and terms of by matching each pair of items in the sequence, until you obtain constant substitutions of the form 't_i is substituted by t_{i+1} and $t_i = t_{i+1}$', that is, all substitutions between the i-th and the (i+1)-th items are constant." We shall explain, by using the following example, that we do not try to solve this problem but a much simpler one, namely: "When observing an infinite sequence of formulas, what are the variables and terms that are already constant when comparing the 1st and the 2nd terms of the sequence?"
We shall see in section 3 that we need to prove the following intermediary lemma:

L2: $\forall$m $\forall$x $\forall$e $\exists$z, x *(exp1(m,x) + e) + (exp1(m,x) + e) = x *exp1(m,x) + z.

Motivation for undertaking a generalization:
We are motivated to perform a generalization because we observe that our definition of the function '+', given in the recursive case as A4: (n+1) + y = (n + y)+1, does not allow us a recurrence on the second variable (called 'y' in A4). Thus, choosing the induction variable 'in position of y' in L2, underlined above, will not be useful for the remaining of the proof. However, evaluating repetitively the recursive definition of exp1, namely A2: exp1(n+1,a) = a* exp1(n,a), will provide an infinite sequence of lemmas to prove. As we said, evaluating the limit of this infinite sequence is very difficult. We simply want to find the terms that undergo a constant substitution between the 1^{st} and 2^{nd} items of this sequence. These terms are good candidates for being generalized by the same variable. Obviously, this does not prove that we can generalize them, it simply eliminates the false generalizations that would lead to a lemma that would lead to a failure.

For the sake of convenience, let us write the first item of the sequence at hand as: t_1 = x *(exp1(m,x) + e) +1*(exp1(m,x) + e) = x *exp1(m,x) + z. Applying axiom A4 for m := m+1 gives the second term: t_2 = x*(x*exp1(m,x) + e) + (x*exp1(m,x) + e) = x*x*exp1(m,x) + z. Matching them provides an obvious set of substitutions: x $\leftarrow$ x*x, exp1(m,x) $\leftarrow$ exp1(m,x), e $\leftarrow$ e, 1 $\leftarrow$ x, exp1(m,x) $\leftarrow$ exp1(m,x), e $\leftarrow$ e, x $\leftarrow$ x*x, exp1(m,x) $\leftarrow$ exp1(m,x), z $\leftarrow$ z.
It follows that the general form of t_n will be
t_n = f1(x)*(exp1(m,x) + e) + (f2(x)*exp1(m,x) + e) = f1(x)*exp1(m,x) + z.
as can be checked by observing the other items of the sequence. Note also that using the equation 1*y = y is suggested by the need to have constant substitutions starting from the first item.

We observe that the term exp1(m,x) undergoes a constant substitution, it can thus be generalized to one variable w. In the contrary case, we would have to generalize it to several different variables.

2 Specifying the problem

Classical definition of the exponential is: $a^1 = a$, $a^{n+1} = a*a^n$.
Equivalent recursive definition, exp1(n, a), recursive on the exponent:

A1: exp1(1,a) = a (Base case)
A2: exp1(n+1,a) = a* exp1(n,a) (Recursive case)

In order to invent a definition recurring on the variable under the exponent, we shall use the following set of axioms relative to + and *. Note that we do not introduce the classical properties of commutative and distributive laws.

A3: 0 + y = y
A4: (n+1) + y = (n + y)+1
A5: 0*y = 0
A6: (n+1)*y = (n*y) + y
A7: 1*y = y (Note that A7 is a consequence of A5 and A6):

Now, we use the fact that exp1 is increasing on both variables. This is why we can use the jump operator as introduced in section 3.1. There must be a variable z taking such values so that the two following formulas are true:

I. $\forall n\ \exists z$ exp1(n,1) = z (base case)

II. $\forall n\ \forall x\ \exists z$ exp1(n,x+1) = exp1(n,x) + z (general case)

As we explained already, proving these formulas will provide us the values of 'z' that insure the formulas are true, and, as a side effect, will give us a recursive definition of exp2(n,y) where the recursion is on 'y' instead of 'n'.

3 Solution

The problem of the construction of the definition for exp1 recursive with respect to the second argument is formulated through the following two problems :

I. $\forall x\ \exists z$ exp1(x,1) = z

II. $\forall x\ \forall y\ \exists z$ exp1(x,y+1) = exp1(x,y) + z

A successful proof for I. will provide a recursive definition of an auxiliary function; we shall call it aux1. A successful proof for II. will provide a recursive definition of an auxiliary function; we shall call it aux2. Let us call exp2 the function exp1 when it is defined with respect to the second argument. The solution for I. and II. Will provide the definition for exp2 as follows:

exp2(x,1) = aux1(x)

exp2(x,y+1) = exp2(x,y) + aux2(x,y)

Let us consider the problem

I. $\forall x\ \exists z$ exp1(x,1) = z

I.A. In the basis case of an inductive proof, x = 1 and the *CM*-formula construction proceeds as follows:

The evaluation of exp1(1,1) gives exp1(1,1) = 1, thus exp1(1,1) = ξ is constructed. This gives, $C_\xi = \{ \xi \mid \xi = 1 \}$. Does z belong to C_ξ ? Yes, if z = 1. z is existentially quantified and thus the condition on z is admissible. We thus have succeeded to construct $\exists z$ exp1(1,1) = z, where z = 1. We thus have aux1(1) = 1.

I.B. In the general case, x = n+1 and the induction hypothesis is $\exists e$ exp1(n,1) = e. This means that aux1(n) = e. The *CM*-formula construction proceeds as follows:

We have the evaluation of exp1(n+1,1) = 1*exp1(n,1). Thus exp1(n+1,1) = ξ and C_ξ = { ξ | ξ = 1*exp1(n,1) }. C_ξ is made more explicit by applying the induction hypothesis. This yields C_ξ = { ξ | ξ = 1*e} = { ξ | ξ = e}. Does z belong to C_ξ ? Yes, provided that z = e. z is existentially quantified and thus the condition on z is admissible. We thus have succeeded to construct $\exists z$ exp1(n+1,1) = z, where z = e. From this we have aux1(n+1) = aux1(n). Putting together aux1(1) and aux1(n+1) we have aux1(x) = 1 for all x. In other words, exp2(x,1) = 1.

Let us consider II. ∀x ∀y ∃z exp1(x,y+1) = exp1(x,y) + z
We look for z = aux2(x,y).

II.A. In the basis case, x = 1, y is a parameter and we have to construct the formula ∃z exp1(1,y+1) = exp1(1,y) + z.

The evaluation of exp1(1,y+1) gives y+1 ; the evaluation of exp1(1,y) gives y.
exp1(1,y+1) = ξ is constructed. Thus, C_ξ = { ξ | ξ = y + 1 }.
Does exp1(1,y) + z, i.e., y + z belong to C_ξ ? Yes, provided that z = 1. z is existentially quantified and thus the condition on z is admissible. We thus have successfully constructed ∃z exp1(1,y+1) = y + z, where z = 1. In other words, aux2(1,y) = 1.

II.B. In the general case, x = n+1 and the induction hypothesis is
∃e exp1(n,y+1) = exp1(n,y) + e. This means aux2(n,y) = e. We look for aux2(n+1,y). The *CM*-formula construction proceeds as follows:

the evaluation of exp1(n+1,y+1) = (y+1)*exp1(n,y+1)

the evaluation of exp1(n+1,y) = y*exp1(n,y)

exp1(n+1,y+1) = ξ is constructed. Thus, C_ξ = { ξ | ξ = (y+1)*exp1(n,y+1) } ; C_ξ is made more concrete by applying the induction hypothesis. This gives C_ξ = { ξ | ξ = (y+1)* (exp1(n,y) + e)} = { ξ | ξ = y*(exp1(n,y) + e) + (exp1(n,y) + e)}.
Does y*exp1(n,y) + z belong to C_ξ ? Yes, provided there is z such that
y*(exp1(n,y) + e) + (exp1(n,y) + e) = y*exp1(n,y) + z. Thus, the lemma L2. is generated :

∀n ∀y ∀e ∃z y*(exp1(n,y) + e) + (exp1(n,y) + e) = y*exp1(n,y) + z.

This lemma is generalized to L3.∀w ∀y ∀e ∃z y*(w + e) + (w + e) = y*w + z. Proving this lemma by our method we obtain a recursive definition of aux3 such that

aux2(n+1,y) = aux3(exp1(n,y),y,e) = aux3(exp1(n,y),y,aux2(n,y))

Proof for L3: ∀w ∀y ∀e ∃z y*(w + e) + (w + e) = y*w + z
y is chosen as the induction variable. L3 shall give aux3(w,y,e).

The basis case for L3: y = 1. One seeks for aux3(w,0,e).
The evaluation of the left hand side of the equation gives (w + e) + (w + e). The evaluation of the right hand side of the equation gives w + z. (w + e) + (w + e) = ξ is constructed. Thus, C_ξ = { ξ | ξ = (w + e) + (w + e) }.

Does w + z belongs to C_ξ ? Yes, provided that there is z such that (w + e) + (w + e) = w + z. A new lemma is generated:

L3a. ∀e ∀w ∃z (w + e) + (w + e) = w + z.

Proof for L3a. ∀e ∀w ∃z (w + e) + (w + e) = w + z. The proof of this lemma shall provide the function aux3a. We shall have aux3(w,1,e) = aux3a(e,w).
w is chosen as the induction variable.

The basis case for L3a.
w = 0. The evaluation of (w + e) + (w + e) for w = 0 gives e + e ; the evaluation of w + z for w = 0 gives z. We construct e + e = ξ. Thus, C_ξ = { ξ | ξ = e + e }. Does z belong to C_ξ ? Yes provided that z = e + e. This yields aux3a(e,0) = e + e.

The general case for L3a :
w = m+1. We have the induction hypothesis ∃g (m + e) + (m + e) = m + g. g = aux3a(e,m). the evaluation of ((m+1) + e) + ((m+1) + e) = {(m + e) + ((m + e)+1)}+1.
The evaluation of (m+1) + z = (m + z) + 1. z = aux3a(e,m+1). We thus simplify to :
(m + e) + ((m + e)+1) = ? m + z

(m + e) + ((m + e)+1) = ξ is constructed. Thus, C_ξ = { ξ | ξ = (m + e) + ((m + e)+1) }. We would like to make C_ξ more concrete by applying the induction hypothesis. We fail and thus we generate the lemma

L4 : ∀m ∀e ∃z' (m + e) + ((m + e)+1) = ((m + e) + (m + e)) + z'

This lemma will provide aux4(m,e) such that

C_ξ = { ξ | ξ = ((m + e) + (m + e)) + aux4(m,e) }.

We shall then be able to apply the induction hypothesis obtaining

(m + g) + aux4(m,e) = ξ.

We generalize L4 to

L4a ∀w ∃z' w + (w+1) = (w + w) + z'.

We shall have aux4(m,e) = aux4a(m+e).

Proof by the induction for L4a. The variable w becomes the induction variable.

The base case for L4a : w = 0. The evaluation of w + (w+1) gives 1. The evaluation of (w + w) + z' = z'. We construct 1 = ξ. Thus, C_ξ = { ξ | ξ = 1) }. Does z' belong to C_ξ. Yes, provided that z' = 1.

The general case for L4a. w = n+1. The induction hypothesis : ∃e' n + (n+1) = e'. e' = aux4a(n). We want to prove that there is z' such that

((n+1) + ((n+1)+1)) = ((n+1) + (n+1)) + z'.

We evaluate :

(n+1) + ((n+1)+1) = (n + ((n+1)+1)) + 1

((n+1) + (n+1)) + z' = ((n + (n+1)) + z') + 1.

The problem is simplified. We have to prove that there is z' such that

n + ((n+1)+1) = (n + (n+1)) + z').

This is generalized to

L4b : ∀n ∀k ∃z' n + (k+1) = (n + k) + z'.

We shall have aux4a(n+1) = aux4b(n,n+1).

Proof for L4b.
By the induction on n.
The base case:

n = 0 ; 0 + (k+1) = k +1

(n + k) + z'= k + z'.

C_ξ = { ξ | ξ = k+1) }. Does k + z' belong to C_ξ ? Yes, provided that z' = 1. We thus have aux4b(0,k) = 1.

The general case for L4b.
n = h + 1. The induction hypothesis : ∃e4b h + (k+1) = (h + k) + e4b.
We evaluate

(h+1) + (k+1) = (h + (k+1))+1

((h+1) + k) + z'= ((h + k) + z')+1.

Thus (h + (k+1))+1 = ((h + k) + z')+1 is simplified to

h + (k+1) = (h + k) + z'.

C_ξ = { ξ | ξ = h + (k+1)) }. We make C_ξ more concrete by applying the induction hypothesis. This yields C_ξ = { ξ | ξ = (h + k) + e4b }. Does (h + k) + z' belong to C_ξ ? Yes, provided that z' = e4b. Thus, aux4b(h+1,k) = aux4(h,k). In other words, aux4b(x,y) = 1.

Let us return to the general case for L4a. We have aux4a(n+1) = aux4b(n,n+1) = 1. Since aux4a(0) = 1, we have aux4a(x) = 1 for all x.

We had aux4(m,e) = aux4a(m+e) = 1. In the general case for L3a we thus have
C_ξ = { ξ | ξ = ((m + e) + (m + e)) + aux4(m,e) } = { ξ | ξ = ((m + e) + (m + e)) + 1 }.
We apply the induction hypothesis. This yields C_ξ = { ξ | ξ = (m + g) + 1 }. Does m + z belong to C_ξ ? Yes, provided that z' = g + 1. We obtain this solution by proving the lemma ∀m ∀g ∃z (m + g) + 1 = m + z'.
Thus, aux3a(e,m+1) = aux3a(e,m) + 1.

We had aux3a(e,0) = e + e

Thus aux3a(x,y) = y + (x + x).

We return now to the base case for L3. We have

aux3(w,0,e) = aux3a(e,w) = w + (e + e).

Therefore, **aux3(w,0,e) = w + (e + e)**

The general case for L3:
y = n+1. We have the induction hypothesis ∃e' n*(w + e) + (w + e) = n*w + e'.
e' = aux3(w,n,e). We want to prove ∃z (n+1)*(w + e) + (w + e) = (n+1)*w + z.
The evaluations give:

(n+1)*(w + e) + (w + e) = (n*(w + e) + (w + e)) + (w + e)

(n+1)*w + z = (n*w + w) + z

(n*(w + e) + (w + e)) + (w + e) = ξ is constructed. Thus
C_ξ = { ξ | ξ = (n*(w + e) + (w + e)) + (w + e) }. We kake C_ξ more concrete by applying the induction hypothesis. This yields C_ξ = { ξ | ξ = (n*w + e') + (w + e) }.
Does n*w + w + z belong to C_ξ ? Yes, provided there is z such that

(n*w + e') + (w + e) = (n*w + w) + z.

We generate the lemma

L3b : ∀n ∀e ∀w ∀e' ∃z' (n*w + e') + (w + e) = (n*w + w) + z'.

This lemma shall give the function aux3b(n,e,w,e'). We shall have

aux3(w,n+1,e) = aux3b(n,e,w,e') = aux3b(n,e,w,aux3(w,n,e)).

We generalize L3b to L3c : ∀m ∀e ∀w ∀e' ∃z' (m + e') + (w + e) = (m + w) + z'. This lemma constructs aux3c(m,e,w,e') and we shall have

aux3b(n,e,w,e') = aux3c(n*w,e,w,e').

Proof for L3c. ∀m ∀e ∀w ∀e' ∃z' (m + e') + (w + e) = (m + w) + z'
By the induction on m.
For the basis case, m = 0, the evaluation of (m + e') + (w + e) is e' + (w + e), the evaluation of (m + w) + z' gives w + z. e' + (w + e) = ξ is constructed.
Thus, C_ξ = { ξ | ξ = e' + (w + e) }. Does w + z belongs to C_ξ ? Yes, provided there is z such that e' + (w + e) = w + z. We generate the lemma

L3d : ∀e ∀w ∀e' ∃z'' e' + (w + e) = w + z''.
This lemma shall provide the function aux3d(e,w,e'). We thus shall have aux3c(0,e,w,e') = aux3d(e,w,e').

Proof for L3d : ∀e ∀w ∀e' ∃z'' e' + (w + e) = w + z''
By the induction on e'.

The basis case. e' = 0. The evaluation of e' + (w + e) gives w + e. The evaluation of w + z'' gives w + z''. (w + e) = ξ is constructed. Thus, C_ξ = { ξ | ξ = (w + e) }.
Does w + z'' belong to C_ξ ? Yes, provided that z'' = e. We thus have aux3d(e,w,0) = e.

The general case. e' = q+1.
We have the induction hypothesis ∃e'' q + (w + e) = w + e'', i.e., aux3d(e,w,q) = e''.
The evaluation of (q+1) + (w + e) gives (q + (w + e)) + 1. The evaluation of w + z'' gives w + z''. (w + e) = ξ is constructed. Thus, C_ξ = { ξ | ξ = (q + (w + e)) + 1 }. We make C_ξ more concrete by applying the induction hypothesis. This yields
C_ξ = { ξ | ξ = (w + e'') + 1 }. Does w + z'' belong to C_ξ? Yes, provided there is z'' such that (w + e'') + 1 = w + z''.
We generate the lemma

L3e : ∀w ∀e'' ∃z''' (w + e'') + 1 = w + z'''.

This lemma shall provide the function aux3e(w,e'') and we shall have

aux3d(e,w,q+1) = aux3e(w,aux3d(e,w,q)).

Proof for L3e. ∀w ∀e'' ∃z''' (w + e'') + 1 = w + z'''.
By induction on w.

The basis case : w = 0. The evaluation of (w + e'') + 1 gives e'' + 1. The evaluation of w + z''' gives z'''. e'' + 1 = ξ is constructed. Thus, C_ξ = { ξ | ξ = e'' + 1 }. Does z''' belong to C_ξ ? Yes, provided that z''' = e'' + 1. We thus have aux3e(0,e'') = e'' + 1.

The general case: w = r + 1. The induction hypothesis is ∃e''' (r + e'') + 1 = r + e''', i.e., aux3e(r,e'') = e'''.
The evaluation of (w + e'') + 1 gives ((r+1)+ e'') + 1 = ((r + e'') + 1) + 1 . The evaluation of w + z''' gives (r + z''') + 1. ((r + e'') + 1) + 1 = ξ is constructed. Thus, C_ξ = { ξ | ξ = ((r + e'') + 1) + 1 }. We make C_ξ more concreete by applying the

induction hypothesis. This yields C_ξ = { ξ | ξ = (r + e’’’) + 1 }. Does (r + z’’’) + 1 belong to C_ξ? Yes, provided that (r + e’’’) + 1 = (r + z’’’) + 1. We thus have z’’’ = e’’’ ; i.e., aux3e(r,e’’) = aux3e(r,e’’).
In consequence, by simplifying the base and the recursive partial definitions for aux3e, aux3e(w,e’’) = e’’ + 1.

Since aux3d(e,w,q+1) = aux3e(w,aux3d(e,w,q)), we thus have

aux3d(e,w,q+1) = aux3d(e,w,q) + 1 ;
aux3d(e,w,0) = e.

In consequence, aux3d(e,w,e’) = e’ + e.

Thus aux3c(0,e,w,e’) = e’ + e.

The general case for L3c: ∀m ∀e ∀w ∀e’ ∃z’ (m + e’) + (w + e) = (m + w) + z’
m = n+1. The induction hypothesis is ∃f’ (n + e’) + (w + e) = (n + w) + f’, thus, aux3c(n,e,w,e’) = f’.

the *CM*-formula construction proceeds as follows:

The evaluation of ((n+1) + e’) + (w + e) = ((n + e’) + (w + e)) + 1; the evaluation of ((n+1) + w) + z’ = ((n + w) + z’) + 1. ((n + e’) + (w + e)) + 1 = ξ is constructed. Thus, C_ξ = { ξ | ξ = ((n + e’) + (w + e)) + 1 }. We make C_ξ more concrete by applying the induction hypothesis. This yields C_ξ = { ξ | ξ = ((n + w) + f’) + 1}.
Does((n + w) + z’) + 1 belong to C_ξ ? Yes, provided that z’ = f’. We thus have

z’ = aux3c(n+1,e,w,e’) = aux3c(n,e,w,e’).

Since aux3c(0,e,w,e’) = e’ + e, by simplifying the base and the recursive partial definitions for aux3c, we have aux3c(m,e,w,e’) = e’ + e.

Since aux3b(n,e,w,e’) = aux3c(n*w,e,w,e’) we have aux3b(n,e,w,e’) = e + e’. We have also aux3(w,n+1,e) = aux3b(n,e,w,e’) = aux3b(n,e,w,aux3(w,n,e)) = e + aux3(w,n,e).

We can thus summarize the extracted definitions as follows:

aux3(w,0,e) = w + (e + e)
aux3(w,n+1,e) = e + aux3(w,n,e)

aux2(1,y) = 1
aux2(n+1,y) = aux3(exp1(n,y),y,aux2(n,y))

aux1(x) = 1

We reformulate for exp2. This yields

exp2(x,1) = 1
exp2(x,y+1) = exp2(x,y) + aux2(x,y)

aux2(1,y) = 1

aux2(n+1,y) = aux3(exp2(n,y),y,aux2(n,y))

aux3(w,0,e) = w + (e + e)
aux3(w,n+1,e) = e + aux3(w,n,e)

****

```
(defun exp2 (x y)
      (if (= y 1) 1 (+ (aux2 x ( - y 1)) (exp2 x (- y 1)))))

(defun aux2 (x y)
            (if (= x 1) 1 (aux3 (exp2 (- x 1) y) y (aux2 (- x 1) y))))

(defun aux3 (w x e)
      (if (= x 1) (+ e (+ e w)) (+ (aux3 w (- x 1) e) e)))
```

Below are given two programs obtained by human creativity instead of our *CMM* :

****

```
(defun exp2 (x y)
      (if (= y 1) 1 (+ (sf x ( - y 1)) (exp2 x (- y 1)))))

(defun sf (x y)
      (if (= x 1) 1 (+ (* (+ y 1) (sf (- x 1) y)) (exp2 (- x 1) y))))
```

exp2(x,1) = 1
exp2(x,y+1) = exp2(x,y) + sf(x,y)

sf(1,y) = 1
sf(x+1,y) = (y + 1)*sf(x,y) + exp2(x,y)

****

```
(defun exp2 (x y)
      (if (= y 1) 1 (+ (sf x y) (exp2 x (- y 1)))))

(defun sf (x y)
      (if   (= x 1) 1
            (+ (* y (sf (- x 1) (- y 1))) (exp2 (- x 1) (- y 1)))))
```

exp2(x,1) = 1
exp2(x,y+1) = exp2(x,y) + sf(x,y+1)

sf(1,y) = 1
aux2(x+1,y) = y*sf(x,y-1) + exp2(x,y-1)

****

In the proof for L1 we have generated a lemma L2.
L2. ∀n ∀y ∀e ∃z y*(exp1(n,y) + e) + (exp1(n,y) + e) = y*exp1(n,y) + z
Let us show why this lemma is generalized:
A careful analysis of recursive arguments in the given definitions for +, * and exp1 shows that no one variable is perfectly suited, since all of them occur in non-recursive positions. The best approximate seems to be n.

We thus go on with n and let us consider the general case for
n := n1+1
y*(exp1(n1+1,y) + e) + (exp1(n1+1,y) + e) = y*exp1(n1+1,y) + z gives
y*(y*exp1(n1,y) + e) + (y*exp1(n1,y) + e) = y*y*exp1(n1,y) + z

Let us generate L2'.
∀n1 ∀y ∀e ∃z y*(y*exp1(n1,y) + e) + (y*exp1(n1,y) + e) = y*y*exp1(n1,y) + z
No variable is recursive in all the occurrences. Let us go on with n1 and let us consider the general case forL2':
n1 := n2+1
y*(exp1(n2+1,y) + e) + (exp1(n2+1,y) + e) = y*exp1(n2+1,y) + z gives
y*(y*y*exp1(n2,y) + e) + (y*y*exp1(n2,y) + e) = y*y*y*exp1(n2,y) + z

Let us generate L2''
∀n1 ∀y ∀e ∃z y*(y*exp1(n1,y) + e) + (y*exp1(n1,y) + e) = y*y*exp1(n1,y) + z
No variable is recursive in all the occurrences. Let us go on with n2 and let us consider the general case for L2''
n2 := n3+1
y*(exp1(n3+1,y) + e) + (exp1(n3+1,y) + e) = y*exp1(n3+1,y) + z gives
y*(y*y*y*exp1(n2,y) + e) + (y*y*y*exp1(n2,y) + e) = y*y*y*y*exp1(n2,y) + z

etc.

The sequence
y*(exp1(n,y) + e) + (exp1(n,y) + e) = y*exp1(n,y) + z
y*(y*exp1(n1,y) + e) + (y*exp1(n1,y) + e) = y*y*exp1(n1,y) + z
y*(y*y*exp1(n2,y) + e) + (y*y*exp1(n2,y) + e) = y*y*y*exp1(n2,y) + z
y*(y*y*y*exp1(n3,y) + e) + (y*y*y*exp1(n3,y) + e) = y*y*y*y*exp1(n3,y) + z

suggests the generalization
exp1(n,y) = w

We thus shall have L3 ∀w ∀y ∀e ∃z y*(w + e) + (w + e) = y*w + z.

References

Arsac J., Kodratoff Y. (1982). *Some Techniques for Recursion Removal from Recursive Functions*; ACM Transactions on Programming Languages and

Systems, Vol. 4, No. 2, April, 1982, 295-322.
Kodratoff Y. (1979). *A class of functions synthesized from a finite number of examples and a LISP program* scheme, International J. of Computational and Information Science 8, 489-521.
Peter R. (1967). *Recursive Functions*; Academic Press, New York.