Kragujevac J. Math. 25 (2003) 81–90.

AN ALGORITHM FOR CALCULATION OF $(+,\cdot)$ - EXPRESSIONS WITH NATURAL NUMBERS

Verica Milutinović

Faculty of Education, Milana Mijalkovića 14, 35000 Jagodina, Serbia and Montenegro

(Received July 1, 2003)

Abstract. Let $t\,(+,\cdot,0,1,2,\ldots)$ be any expression containing natural numbers $0,1,$ $2,\ldots$ and operation symbols $+$ and $\cdot$. In this paper we describe an algorithm for calculation of such expressions values, on which a program written in C language is made.

1. INTRODUCTION

In this paper we will represent value calculation of $t\,(+,\cdot,0,1,2,\ldots)$, where t is any expression containing natural numbers $0,1,2,\ldots$ and operation symbols $+$ and $\cdot$. For the algorithm, as well as for the program we are intending to expose, it is important to define expressions in syntax, i.e. as specific words. We translate calculation into the characters and words manipulation instead of intuitive numbers.

This paper is organized as follows. In the Section 2 we will define the natural-number expressions (terms) and adopt some substitutions concerning operations of addition and multiplication, and then we shall define the case in which we find the expression evaluated. In the third Section, through several examples, the procedure of such calculation is explained, and the flow of finding term value by algorithm is described in details in Section 4. With this algorithm we could find the value of unlimited length expressions. In Section 5 the program made on the basics of established algorithm is explained. Section 6 brings the conclusion.

2. DEFINITIONS OF EXPRESSIONS, FIRST ALGORITHM

We starts with **the language**:

$L = \{0,', +, \cdot\}$

where $+$ and $\cdot$ are operation symbols of length 2, $'$ operation symbol of length 1 and 0 is a constant symbol.

Terms (expressions) are defined by the following inductive definition:

Definition 2.1.

(i) 0 is term

(ii) If P, Q are terms, than the words P', $(P+Q)$, $(P \cdot Q)$ are terms as well.

Examples of expressions: $0, 0'', (0 + 0''), (0 \cdot 0'''), \ldots$

For $0, 0', 0'', 0''', \ldots$ (that could be called numerals) we shall use the following abbreviations $1, 2, 3, \ldots$ respectively, where $0'$ stands for 1, $0''$ stands for 2, $\ldots$.

So, we can write:

$$2 \longrightarrow 0'' \longrightarrow 1', 3 \longrightarrow 0''' \longrightarrow 1'' \longrightarrow 2', \ldots. \tag{$*$}$$

Substitutions (Peano's) that we are adopting are:

$(P1)$ $(t + 0) \longrightarrow t$

$(P2)$ $(t + u') \longrightarrow (t + u)'$

$(P3)$ $(t \cdot 0) \longrightarrow 0$

$(P4)$ $(t \cdot u') \longrightarrow ((t \cdot u) + t)$

We know that value calculation of an expression (i.e. $((2 + 3) \cdot 6)$) depends on adopted agreement (or definition). To evaluate some expression A means finding another equal expression which is, by definition, evaluated. Let us define the case in which we find the term evaluated.

Definition 2.2. As evaluated we find the terms that do not contain brackets, addition and multiplication symbols (i.e. $0, 0''', 7, 9'' \ldots$) which are basic (resumed) terms .

Let us adopt the rule that value of term we get with applying substitutions $(*)$ as long as there is $'$-s in the term.

Under the calculation of term value we find the flow (algorithm) whose starting point is term in which in every step we apply a substitution of the form:

$$Expression \to value(Expression) \qquad (**)$$

where $Expression$ is some sub-term, and $value(Expression)$ is its value. Value can be reached by application $(*)$ or some of substitutions $(P1)$ to $(P4)$. We are doing so as long as we reach the expression which is by definition evaluated. That is how in a new way, by words (expressions, terms) instead of numbers we could reach the result. Algorithm is described in details in Section 4. The main role in it plays well known *algorithm of the first right bracket*(see [1]).

In the given term we find "pieces" i.e. sub-terms of the form a', $(a + b)$ or $(a \cdot b)$, and then substitute them with appropriate value. Thus step by step we come to the value of the expression.

The question is how to find such sub-terms? In the given expression, realized as word, we "walk" from left to right as long as we reach the first character of right bracket. Then, from that spot we are retracing left to first character of left bracket and those two bracket are enclosing sub-term of desired form. This algorithm is so called algorithm of the first right bracket which belongs to the most important algorithms in the field of computation.

In the next section through several examples the procedure of finding value of different expressions is proposed in details. We will use natural numbers arithmetic and algorithm of the first right bracket.

3. EXAMPLES

Let us now see in practice through several examples how our term calculation looks like. By definition as evaluated we find the terms that do not contain brackets, addition and multiplication characters (i.e. $0, 0''', 7, 9'' \ldots$).

Example 3.1. Find the value of term $(2+2)$.

Solution:

The flow (happening) of calculation is running in the next way:

$$\begin{aligned}(2+2) &\xrightarrow{(*)} (2+0'')\\ &\xrightarrow{(P2)} (2+0')'\\ &\xrightarrow{(P2)} (2+0)''\\ &\xrightarrow{(P1)} 2''\\ &\xrightarrow{(*)} 3'\\ &\xrightarrow{(*)} 4\end{aligned}$$

Example 3.2. Find the value of expression:

1^0 $(5+3)$

2^0 $(5 \cdot 3)$

3^0 $(6+(2\cdot(3+1)))$

Solution:

1^0 In this example we are talking about addition of terms, which means that we will use substitutions of the form $(*)$, $(P1)$ and $(P2)$ in our calculation:

$$\begin{aligned}(5+3) &\xrightarrow{(*)} (5+2') \xrightarrow{(P2)} (5+2)' \xrightarrow{(*)} (5+1')' \xrightarrow{(P2)} (5+1)''\\ &\xrightarrow{(*)} (5+0')'' \xrightarrow{(P2)} (5+0)''' \xrightarrow{(P1)} 5''' \xrightarrow{(*)} 8\end{aligned}$$

2^0 Now we are talking about multiplication of terms so we will use substitutions of the form $(*)$, $(P3)$ and $(P4)$ in calculation, but also $(P1)$ and $(P2)$ because multiplication is reduced to addition. In this example we could see in practice the procedure of term multiplication.

$$\begin{aligned}(5\cdot 3) &\xrightarrow{(*)} (5\cdot 2') \xrightarrow{(P4)} ((5\cdot 2)+5) \xrightarrow{(*)} ((5\cdot 1')+5) \xrightarrow{(P4)} (((5\cdot 1)+5)+5)\\ &\xrightarrow{(*)} (((5\cdot 0')+5)+5) \xrightarrow{(P4)} ((((5\cdot 0)+5)+5)+5)\\ &\xrightarrow{(P3)} (((0+5)+5)+5) \xrightarrow{(P1)} ((5+5)+5)\\ &\xrightarrow{(*)} ((5+4')+5) \xrightarrow{(P2)} ((5+4)'+5) \xrightarrow{(*)} ((5+3')'+5)\\ &\xrightarrow{(P2)} ((5+3)''+5) \xrightarrow{(*)} ((5+2')''+5) \xrightarrow{(P2)} ((5+2)'''+5)\\ &\xrightarrow{(*)} ((5+1')'''+5) \xrightarrow{(P2)} ((5+1)''''+5) \xrightarrow{(*)} ((5+0')''''+5)\end{aligned}$$

$$\xrightarrow{(P2)} ((5+0)''''' + 5) \xrightarrow{(P1)} (5''''' + 5) \xrightarrow{(P2)} (5+5'''')' \xrightarrow{(P2)} (5+5''')''$$
$$\xrightarrow{(P2)} \dots \xrightarrow{(P2)} (5+5)''''' \xrightarrow{(*)} (5+4')''''' \xrightarrow{(P2)} (5+4)'''''' \xrightarrow{(*)} (5+3')''''''$$
$$\xrightarrow{(P2)} (5+3)''''''' \xrightarrow{(*)} (5+2')''''''' \xrightarrow{(P2)} (5+2)'''''''' \xrightarrow{(*)} (5+1')''''''''$$
$$\xrightarrow{(P2)} (5+1)''''''''' \xrightarrow{(*)} (5+0')''''''''' \xrightarrow{(P2)} (5+0)'''''''''' \xrightarrow{(P1)} 5''''''''''$$
$$\xrightarrow{(*)} 15$$

Notice. Let us describe term addition and multiplication procedures which we used by now.

Procedure of addition:

(S): In term A (which equals $(x+y)$ at the moment) we investigate second addend i.e. the forth character y. If y:

- ▷ is a numeral of the form c' – apply $(P2)$ on A. With new term $A = (x+c)'$ while $y = c$ go to (S).
- ▷ is not a numeral – transform it into numeral of the form c' by applying $(*)$. Substitute y with c' in A and go to (S).
- ▷ equals 0 – apply $(P1)$ on A. Ending the procedure with new value.

Procedure of multiplication:

(M) In term A (which equals $(x \cdot y)$ at the moment) from left to right we split the first piece i.e. sub-term t of the form $(a \cdot b)$, $(a+b)$ or a. If term t takes the form of:

- ▷ $(a+b)$ – Apply (S) on t. Substitute sub-term t with new value and with such A go to (M).
- ▷ $(a \cdot b)$ – Investigate next literal (b) behind the literal " $\cdot$ ". If b:
  - ↪ is numeral of the form c' – apply $(P4)$ on t. Substitute sub-term t in A with new term. $((a \cdot c) + a)$ and go to (M).
  - ↪ is not numeral – substitute b in sub-term t of term A with appropriate numeral. Go to (M)
  - ↪ equals 0 – apply $(P3)$ on t.
- ▷ a – End the procedure with message that a is the value of the term.
- ▷ is of some other form – End the procedure with message that word is not term i.e. isn't written in appropriate way.

3^0 Fore-solution. Given term is a "word" composed of characters (in this example it is: $"(","6","+","(","2",\ldots)$. When we are reading character by character of the term from left to right and reach to the first appearance of character ")", then we return to the first character "(" and collect all characters in returning (it ought to be three of them). We check the value of middle character and if it is "+", we apply addition procedure (S), if it is "$\cdot$", we apply multiplication procedure (M), and if it is some other character, the word is not the term and we could not find its value.

Solution. We are reading character by character of the term $(6+(2\cdot(3+1)))$ from left to right. First right bracket is in sub-term $(3+1)$, we call the procedure (S) for addition calculation and substitute sub-term with new value 4.

$(6+(2\cdot(3+1)))$

$\longrightarrow (6+(2\cdot 4))$ We continue to go right to the next right bracket,

We split sub-term $(2\cdot 4)$, call the procedure (M) for multiplication calculation of terms and substitute it with evaluated term 8,

$\longrightarrow (6+8)$ Again we go right to the next right bracket, and split $(6+8)$

We call the procedure (S) for addition calculation and substitute the term with its value 14

$\longrightarrow 14.$

As there are no more operation characters, nor brackets we reached to the end i.e. to the value of starting expression.

Example 3.3. Find the value of term: $(((2+3)\cdot 5)+(4\cdot(3+1)))$

Solution:

Having in mind presented procedure in every step we apply some of the procedures (S) or (M):

$$(((2+3)\cdot 5)+(4\cdot(3+1))) \xrightarrow{(S)} ((5\cdot 5)+(4\cdot(3+1))) \xrightarrow{(M)} (25+(4\cdot(3+1)))$$
$$\xrightarrow{(S)} (25+(4\cdot 4)) \xrightarrow{(M)} (25+16)$$
$$\xrightarrow{(S)} 41$$

4. ALGORITHM FOR CALCULATION OF TERM VALUE

Let us describe the general procedure for calculation of term value denoted by t. In this description we use auxiliary variables *Lis* and *string*.

Step 1 : We read character by character of the term (i.e. word) t and by the way we load a new list Lis in this manner:

(A) From left to right we split character of the term t which can be:

1^0 Type (*). We put it into *string*, and then put *string* on the top of the list Lis and go to (A).

2^0 Some of characters: $" + "," \cdot "$ or "(". Than we put this character into *string*, and put *string* on the top of the list Lis, then go to (A).

3^0 Character ")". Go to *Step*2.

4^0 Symbol for the end of the input. We reached to the end of term i.e. there isn't characters we could read. Then:

- If the list Lis is containing only one member, exactly that member represents the value of the term and algorithm is ending with that value.
- If Lis is containing more than one member algorithm is ending with the message that input data are wrong i.e. expression whose value is supposed to be calculated is not a term.

5^0 Some other cases. Algorithm is ending with message about wrong input term.

Step 2 : From the list Lis we take first three elements, erase them from the list Lis and the forth element of the list Lis substitute with new value Rez attained in this way:

We check the value of the second of three elements we get from the list Lis. If its value is:

1^0 "+", we call the procedure (S)(addition of terms) and the new value is placed into variable Rez.

2^0 "·", we call the procedure (M) (multiplication of terms) and the new value is placed into Rez.

3^0 not "+" nor "·", then algorithm ends with the message about the wrong input data.

When we replace the first four elements of the list Lis with the result Rez we go to *Step* 1, so we can continue to read the term t as long as we reach its end.

Notice. We could adopt additional substitutions (addition and multiplication tables) and use them in the algorithm instead of using procedures (S) or (M):

$(1+1) \longrightarrow 2$, $(2+1) \longrightarrow 3 \ldots$

$(44+21) \longrightarrow 65$

$\ldots$

$(1 \cdot 1) \longrightarrow 1$, $(2 \cdot 1) \longrightarrow 2$

$\ldots$

Algorithm of the first right bracket is up to the values of expressions in the form $(a+b)$ *or* $(a \cdot b)$*, where* a *and* b *are natural numbers (numerals).*

5. ABOUT THE PROGRAM ITSELF

Let us describe the program which implements given procedure (algorithm). To allow more efficient usage of memory we use dynamic structures. Input data are words (terms) whose value we are calculating (i.e. $((4+2) \cdot (3+7))$).

For the purpose of saving characters of the input term we utilize the structure

```
struct drvo{
    char *glava;
    struct drvo *levi, *desni;} ;
```

Since we need to create the list *Lis* in the program the new structure is utilized

```
struct lista{
    struct drvo *glava;
    struct lista *rep;};
```

which is used for saving characters of the term we are dealing with as a list. In natural words, we are making "avenue". The whole algorithm runs using this structure.

After starting the program, the very first thing to do is to enter the term that we are evaluating. End of inputting term is denoted with double $<$ *Enter* $>$. While entering term it is immediately being calculated. Because the algorithm of the first right bracket we are using for term calculation, is up to the values of expressions in the form $(a + b)$ or $(a \cdot b)$, where a and b are natural numbers (numerals) we could work in the tree language as well, and that is how this program is working.

When program "reads" the character it makes *a small tree* out of it which is of type **struct drvo** and whose head is containing the value of that character, while left and right arms are equal to $NULL$ (because of that, we are calling it *a small tree*). So we arrange trees into the avenue Lis by putting each new at the beginning of the list, till we come to the first right bracket. Then calculation is done in a tree algebra i.e. out of first three "trees" from Lis we make a new tree whose head contains the middle tree the left arm contains the first, and the right contains the third tree. The new tree is located in place of the first following tree in Lis containing, in fact, the left bracket.

We continue reading the term till the next right bracket when we return once again. We keep doing it till we reach the end of the term. We could immediately calculate the values of the sub-term within the left and right bracket either using procedures (S) and (M) or by using additional substitutions (addition and multiplication tables), instead of using the trees. However, in such a cases we could not gradually print all mid-results. In the end list Lis (if we feed the data correctly) is reduced to one big tree dr, whose value should be calculated. It is done through function **izracunaj_term**. It helps calculate the value of the term and print mid-results. The function is given by pseudo-code:

```
void izracunaj_term(struct drvo *dr) {
  while (dr is not a small tree ){
  pom=dr;
  while (pom is not a small tree ){
  while (pom->levi is not a small tree )
      pom=pom->levi;
      if (pom->desni is a small tree ) {
        if (pom->glava == '*')
        pom->glava = proizvod (pom->levi, pom->desni);
        pom->levi i pom->desni becomes NULL }
```

```
          if (pom->glava == '+')
          { pom->glava =zbir (pom->levi, pom->desni);
          pom->levi  i pom->desni  becomes NULL }
          print dr;
          }
        else pom=pom->desni;
        }
}
```

Functions *proizvod* and *zbir* calculate the final result of its input argument values based on algorithms (M) and (S).

6. THE CONCLUSION

Natural numbers are natural to us, but computer can only operates with preciously defined syntax not intuitive definitions. Because of that, we adopt syntax definitions and substitutions for natural numbers using terms and develop algorithm for value calculation of such terms. The aim is to show the students that mathematics consists of presently adopted symbols, language and rules that should be studied and learned, but that it is the discipline still developing, with new ideas appearing and creativity expressed but not as final outcomes. By using the explained program students could view the given calculation in a practical way and understand its substance easier.

References

[1] Prešić, S., Milić S., Ognjanović S., Vujić S., *Produbnice (matematičke)* Arhimedes, Beograd, (1999), 33–35.

[2] Prešić, S. , *Raznice, II*, Prosvetni pregled, Beograd, (1998), 9–11.