Specification of the Tabulated Equations Problem

Prabhaker Mateti

Consider the informal description of a program, given in the next section. Give a carefully written specification of it, using well-chosen notations, and words. Pretend that the requirements analysis is already done: That is, wherever relevant in the specs, explain in English the appropriate parts of requirements.

1 Informal Statement of the Problem

Our mathematician friend wants a program that can "tabulate" a sequence of symbolic equations found in typical linear algebra into a matrix form.

For example, given the input shown in Figure 1,

```
3x + 5y + 714z = 12g + 53b,
76489x +3z = 2g+8b+99a + 1024,
           x+ 49 = 7g,
1234567y+ 2z = a + 999b + 6x + 911,
 + 3x + 5y =  53b + 12g.
```

Figure 1: The Input

your program should produce the result shown in Figure 2.

```
    3x +       5y + 714z      = 12g +  53b,
76489x +              3z      =  2g +   8b + 99a +     1024,
     x +                   49 =  7g,
         1234567y +   2z      =        999b +   a + 6x +  911,
    3x +       5y             = 12g +  53b.
```

Figure 2: Output Corresponding to Input of Figure 1

Our friend tells us that her equations do not contain minus signs, nor floating point numbers, but that sometimes the coefficients are larger than can fit in 64-bits, and the equations become so long that she has to write them on a long scroll of paper sideways. We impressed her by remarking that we will imagine the paper width to be infinity.

2 A Pretty Good Spec

This note contains a "good" answer, and expects you to produce a "really" precise answer to possible exam question in a Software Engineering class. The occasional commentary by this instructor is set in the type size of this paragraph.

There are essentially three things we must consider.

1. What are the preconditions?
2. Spec that output is equivalent to the given input.
3. Spec that output looks "good".

This solution is deliberately written in an informal way to emphasize what needs to be specified versus how to specify it. So, find all the ambiguities – but, do not just nit pick!

(Requirements Analysis: Since nothing particular is said about how IO ought to be, we assume that the required `tabulateeqn` program takes its textual input from `stdin` and outputs to `stdout`.)

2.1 Syntax of Equations

The input must be well formed "equations."

Other than using some kind of grammar, I know of no way to describe this precondition. Syntax diagrams etc. are, of course, other ways of specifying context-free grammars.

The input must be a sentence derived from the non-terminal ***eqns*** of grammar G given below.

(RA: Note that terms of the same variable may occur multiply either in the ***rhs*** or in the ***lhs*** or both.)

```
eqns ::= eqn {',' eqn } '.'
eqn ::= lhs '=' rhs
lhs ::= exp
rhs ::= exp
exp ::= term | term '+' exp
term ::= number | coeff varid
coeff ::= empty | ['+'] number
varid ::= letter
```

(Requirements Analysis: It is not clear if long equations are presented on multiple lines in the input. We will de facto allow it as our grammar/parser of input takes care of it. If our client insists that that should not be permitted, our task becomes more complex. The required grammar is no longer context-free.)

Why should we let ***coeff*** begin with an optional plus? Is the grammar "unambiguous" ? What does unambiguous mean in this context? The same as in an "unambiguous specification" ? If duplicate-var-id terms are not to be permitted in an equation, why does the required grammar become more complex? For that matter, are Pascal, C, Ada, ... context-free?

2.2 Output is Equivalent to the Input

(Requirements Analysis: Since we impressed our customer by remarking that we will imagine the paper width to be infinity, we choose to output each equation as one line, regardless of its length.)

The output is also a sentence derivable from ***eqns***.

Let n be the number of lines in the output.

The output consists of equations printed one per line. The order of the equations is the same as that of input.

The i-th equation of the input is *semantically equivalent* to the i-th equation of output. Two equations $e_1 = e_2$ and $e_3 = e_4$ are equivalent if e_1 is equivalent to e_3, and e_2 is equivalent to e_4. An expression e is equivalent to f if both e and f have the same terms, same number times in each. Note that permutations of terms is allowed.

(RA: Simplification of terms is not even attempted.)

2.3 Output is Well-Formatted

To describe the formatting of the output, imagine dividing the 2-d output into columns of n rows as follows. Draw vertical straight-lines, from the topmost line to the bottom-most spanning the n lines of output, immediately surrounding each variable id letter, each `" + "` and the `" = "`. Similarly draw a line to the immediate right of each number, and one straight line at the leftmost edge of the output.

The first (i.e., the leftmost) column must be a number-column. Each row in a column of numbers must be either a string of blanks, or a right-justified number.

Each row of the equals-column must contain `" = "`. Each row of each plus-column must contain either a `" + "` or three blanks. All the rows of a variable id column must be either one blank or contain some letter, and all these letters must be the same.

Should we also say "no column is all blanks". I think not; what say you? What about "The first (i.e., the leftmost) column is a number-column" ?

3 A Precise Solution

The above is not precise enough. Redo so that it is. Enumerate all the ambiguities of the above.

4 Discussion

The following are some imagined questions from students.

If we have some 40 minutes to spend on this problem, do you expect us to finish it this well?

Well, if you want an A, yes! More seriously, an answer along the lines of Section 2 is certainly expected. Section 3 could be home work that you should be able to do a good draft in a few hours, say 4. Such a draft probably contains errors. It is best to let others read it. If that is not possible, read it yourself, but after a day or two.

Is it worth spending this much effort in specifying instead of producing several hundreds of lines of code?

I think so. There are no statistics that can validate or invalidate this belief. Assuming that the program being developed is not a throw-away program, but has a long lifetime, the careful translation of requirements to specs and then paraphrasing them back to the user/client is an effective technique to prevent expensive design and coding errors.