WEB BASED SPREAD SHEET APPLICATION USING JAVASCRIPT

Implementation Details

Name : **Arun K** ***Roll No:*** **MT2009054**

Following are the different stages in the implementation of the Spread Sheet,

1. Parsing
2. Normalizing
3. Data Flow Analysis
4. Evaluating and Computing Dependencies
5. Handling Insertion of Rows/columns
6. Deletion of rows/columns and highlighting invalid formulas.
7. Processing of Range expressions like SUM and PROD
8. Adjusting the range expressions after deletion or insertion of rows/cols.
9. Persist the spread sheet and reload it back.
10. Automated testing using Javascript.

A detailed explanation of these stages is given below.

1. **Parsing:**

   This is done using the jsparse functions, which allows to build the ast for any user defined grammar. The grammar used for this spread sheet are,

   Number :: = range(0-9)

   Operators::=+, *, -, /,

   Alpha ::= range(A-Z)|range(a-z)

   Address :: = sequence(repeat1(Alpha),Number)

Expression ::= wsequence(“(”,Operators, Address|Number|Expression, Address|Number|Expression, “)”)

Keywords ::= “SUM”|”PROD”

Functions ::= wsequence(“(“,Keywords, Address, Address, “)”)

Formula ::= Number|Address|Expression|Functions

For example, an expression could be (+ 2 3), which evaluates to 5. It could also be a nested expression like (* 3 (+ 4 5)) which evaluates to 27. Cell addresses are specified as <col num><row name >, for example, A1 represents cell at column A and row 1. Hence an expression that uses cell address could be (+ A1 4).

2. **Normalizing**

   This is the process of converting the index expressions to cell-ref expressions. Because an index expression like Z43 is valid, but it is a valid cell-ref expression only if there is a cell with address Z43 in the current spread sheet.

3. **Data Flow Analysis**

   In this stage, we find the Parents of a given formula, and see if there are any circular dependencies. For eg: if A1 = B1 and B1 = C1, then entering A1 = C1 will result in circular dependency.

4. **Evaluating and Computing Dependencies**

   Here we evaluate the formula and ripple the value to its children. For eg: if A1 = B1 + C1, and Say D1 = A1. Now if B1 is changed, then both A1 and D1 have to be recomputed. Note, the evaluation is done automatically by the recompute handler, once the cell’s formula changes.

5. **Handling Insertion of Rows/columns**

   The advantage of normalizing is that insertion of rows and columns will automatically show the correct formula after insertion. Hence this stage, only requires us to be able to re-draw the table after a new row/ col has been inserted.

6. **Deletion of rows/columns and highlighting invalid formulas**

   Here the updation of the view is similar to stage-5, but an additional step that needs to be done here is to highlight the dependent cells as error. This is done by converting a cell-ref expression to a Garb-ref expression, once when a cell’s

parent is deleted. The basic idea is that for all cells in the deleted row/column, update all the children's corresponding cell-ref expression with garb-ref expression.

7. Processing of Range expressions like SUM and PROD

This stage, involves designing a new type of formula which will handle range expressions. The parsing logic is trivial, but the real challenge is in normalizing. Error checking is also simple, where we only need to check for co-axial range. In normalizing we store all the cell references between the lower and upper range in an array. For eg: (SUM B1 F1), then the normalized expression will contain an array of all the cell objects between b1 and f1, including the both. This eases the data flow analysis and updating dependencies. Also another parameter called Direction is stored in the normalized expression object. Eg: dir=0 ➔ Co-axial in row, and dir=1 ➔ Co-axial in column. This will help in the following stages.

8. Adjusting the range expressions after deletion or insertion of rows/cols.

One of the most difficult feature to implement is to adjust the range expression on deletion and insertion or rows/cols. Thankfully, the design decision that we chose in the stage-7, helps us in handling this with relatively less complexity. What we need to do here, is that when a row is say inserted between two rows, then we need to check all the cells of the affected and see if any of its children is a range expression and that this cell is co-axial with that. In that case we re-normalize the range expressions, which will now make the range including the added row. A similar principle can be applied to deletion as well.

9. Persist the spread sheet and reload it back

Here the cell formulas are converted to a two-dimensional array of strings and then converted to a JSON string. Once this is done, it is stored in the POW server using AJAX. A library called JQUERY is used for this purpose, which is a wrapper to Ajax. The api's are extremely simple and provides callbacks to handle any error conditions from the server.

10. Automated testing using Javascript

This is done using Fire-Unit. The major portion of the testing was manual, except for parsing and persistence, which were automated through fire-unit test cases.