Cooperating Languages - Spreadsheet Example

Hesam Samimi

This material is based upon work supported in part by the National Science Foundation under Grant No. 0639876. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

VPRI Memo M-2012-007

Viewpoints Research Institute, 1209 Grand Central Avenue, Glendale, CA 91201 t: (818) 332-3001 f: (818) 244-9761

Cooperating Languages
Spreadsheet Example

Hesam Samimi

September 13, 2012

Our current strategy is to look at more examples to examine the opportunities for cooperative work (between languages) and determine the limits of the current cooperating languages framework. To this end, we look at a super simplistic spreadsheet.

Spreadsheet Demo

We start by a table where cells can contain expressions involving real constants and/or references to other cells, as shown in Fig. 1.

	A	B	C
1	5	*A1 + 1*	0
2	*C1*	*C2 - B1*	1

Fig. 1. Spreadsheet table

As usual we start by defining any class / relations, a sketch of which is shown in Fig. 2.

Task #1: Table Layout

Columns and rows can be resized subject to a maximum window size. As usual *OCassowary* is a suitable language for this purpose (see Fig. 3).

Task #2: Which Cells To Recompute When Updating A Cell

Once the user makes a change to a cell, we need to figure out what other cells need to change value due to a dependency upon the updated cell. This is essentially a reachability operation and so *Datalog* is used as shown in Fig. 4.

```
class Expr : Thing
    Attributes
        dependedUponCells : Set(Cell)
        vars : Set(Symbol)
        value : Real
class CellSequence : Thing
    Attributes
        index : Int
        cells : List(Cell)
class Column : CellSequence
    Attributes
        width : Real
class Row : Thing
    Attributes
        height : Real
class Cell : Thing
    Attributes
        row : Row
        columm : Column
        expr : Expr
class Table : Thing
    Attributes
        rows : List(Row)
        columns : List(Column)
        cells : List(Cell)
        maxTableWidth, maxTableHeigth : Real
        minColumnWidth, minRowHeight : Real
```

Fig. 2. Sketch of spreadsheet model.

```
sum(columns.width) <= maxTableWidth
sum(rows.height) <= maxTableHeight
all c : Column | c.width >= minColumnWidth
all r : Row | r.height >= minRowHeight
all c : Column | c.width stay
all c : Row | r.height stay
```

Fig. 3. Enforcing layout with Cassowary.

```
relation UpdatedCell(a: Cell)
relation UpdatableCells(a: Cell) <--
    UpdatedCell(a) ||
    exists b : Cell | (Expr_dependedUponCells(b.expr, a) && UpdatableCells(b))
```

Fig. 4. Determining which cells should be recomputed using Datalog.

Task #3: Updating Cells

Now that we have determined the set of cells that should be recomputed, we would like to determine the new values. A constraint solver is suitable for this task, because it allows us to have two-way dependencies.

For the sample table of Fig. 1, we have the following constraints as shown in Fig. 5, where we employ Z3 for the task.

```
env hostProgram:
    A1 = 5.0
    B1 = A1 + 1.0
    C1 = 0.0
    A2 = C1
    B2 = C2 - B1
    C2 = 1.0
in:
    #Z3
modifiableVariables:
    ModifiableVars tuples
```

Fig. 5. Constraints for table in Fig. 1.

Task #4: Frame Conditions for Constraint Solving

And finally we use Datalog again to determine which are the modifiable variables need updating and which should not change when solving the constraints. See Fig. 6 below.

```
relation ModifiableVars(v: Symbol) <--
    exists b : Cell | (UpdatableCells(b) && !UpdatedCell(b) && Expr_vars(b.expr, v))
```

Fig. 6. Determining set of modifiable variables for the purpose of constraint solving with Z3.