Data Cleaning DSL
MileStone 2

Hao Wang wanghao@oregonstate.edu
Jingyuan Xu xujing@oregonstate.edu
Changshuang Kou kouc@oregonstate.edu

Jun 1, 2016

Description

In our project, we are going to design a DSL language in data cleaning. This DSL language in data cleaning try to operate the one data set spreadsheet or more than one, such as delete data (if repeat, or error), change data (for some specific requirements), filter and so on. Our objects are basically string dataset, which are the inputs from and output to files. We will implement them as a string spreadsheet type Dset = [[String]], and each list has the same number of elements. In order to do some kinds of mathematical operations, we define a value type Val, which contains the basic types of values, such as integers, float number, strings and so on. We assume each column describes a feature of data, which has the same value type, thus the data set (Dset) and the value set ([[Val]]) can be transformed to each other by a header, which contains the information of every column's type. What's more, we defined 2 kinds of value set: Vset is the vertical set, means that inner lists are columns; Hset is the horizontal set, means that inner lists are rows. Dset, Vset and Hset can be transformed to each other, thus we can implement our functions using any of these types by convenience.

We have implemented most of the functional semantics of our DSL. These are:

synSetReplace: input a synonym dictionary and a Dset, then output a Dset that change all synonyms to the prescribed words.

dupForAll: remove the duplicate data and empty data from the input data set.

dupForKey: combine the non-empty value together for all data that has the same key value.

joinForAll: merge multiple Dset to one Dset and maintain the column number. In other words, empty element should be output as "".

joinForKey: same as joinForAll but combine the key column together. Key column should be indicated by column number from users.

deleteColumn, deleteRow, deleteValue: delete a specific column/row/value from an input data set.

changeColumnType, changeRowType, changeValueType: change type of a specific column/row/value from an input data set.

changeValueByPos: It takes a Dset, multiple positions (row,column), and multiple target strings as input, and output a Dset that change the specific strings to the target strings for the target positions.

changeValueBySearch: It takes a Dset, multiple original strings, and multiple target strings as input, and output a Dset that change the original strings to the corresponding target strings.

How to Run

To run the DSL, your ghci must contains the library module "Text.CSV". If your OS is MacOS or Linux, just type "cabal install csv" in the terminal.

Put all .hs files in a directory, then simply type:

```
ghci Tests.hs
```

We have already put multiple Dset, headers, synonym dictionary and operations in Tests.hs. You can simply type their names and check what they are. What's more, you can define your own stuff according to our syntax. All data types are defined in Syntax.hs.

You can also type:

```
ghci Samples.hs
```

This is the real world sample. It takes the input from file "w1.csv" and "w2.csv" and do some operations. If users type:

```
sample
```

the system will output csv files to show these operations separately.

Our DSL is usable, which means users can write their own script for the real world use. Please see the Samples.hs as a reference. To input a csv file, use the following sentences:

```
parameterName <- readFile "inputFileName.csv"
let dsetName = input "inputFileName.csv" parameterName
```

To output a csv file, use the following sentences:

```
output "outputFileName.csv" dsetName
```

To do operations, you can write your own stuff according to our syntax and semantics. All data types are defined in Syntax.hs.

Problems and Future Work

To make our DSL more usable, it seems that we would better abandon the header part. But we do not know how to adjust the type system of Haskell by just reading the input csv file as a string. We got the idea of header by the hackage library "Data.Csv"[1], and we do not know how to do without header.

In the future, we are going to work on this DSL at the following points:

1. Solve the header problem mentioned before;
2. Add more semantical functions to make the DSL stronger;
3. Make our DSL more semantically clear;
4. Add more comments on the code to make our DSL more readable;
5. Provide a user manual for the convenient use of users.

Reference

[1]. https://hackage.haskell.org/package/cassava-0.4.5.0/docs/Data-Csv.html