Language Specification for SUGAR

Health Informatics 3

June 26, 2015

1 Introduction

This document will describe the language used to describe analyses in our program. The language should be simple and expressive. This will ensure the best usability and improve the user experience.

We've opted to name her SUGAR, which stands for Simple Usable Great Analysis Reasoner.

2 Basic Idea

We like the BASH approach to piping input from one process to another, so we'll be borrowing the pipe | command. Don't worry, we'll give it back. Parentheses are great to use for grouping things like parameters and the ; is great for closing things.

3 Example

Let's start with something like:

$$def\ idCheck : Constraint = statSensor.id = 170;$$
$$from(statSensor)|constraints(idCheck)|is(person1)$$

Here we run into a couple of things, first of all we have our processes (from, constraints etc.). These are straight forward and should pose no issues. They always have their parameters between parentheses (if they have no parameters parentheses are still required).

Next we have our pipes, which we have described in the previous section. They basically let the output of the one process be the input of the next.

We also encounter a macro, this will be discussed in the next section.

4 Macros

Macros are a way of splitting the processes you want to perform on the data (your process chain) and exactly what that process should do.

Let's begin with another example:

$def\ idCheck : Constraint = measurement.id = 1\ AND\ (measurement.level > 5)\ OR\ NOT(true)\ AND\ (measurement.level > measurement.value)$;

All macros start with the word def. It is followed by the identifier for the macro (which can then be used in a process chain). In this case the identifier is idCheck. After this we see the type. We currently have several types in our language: Constraint, GroupByColumn, GroupByConstraint, Join, Connection, Computation and Comparison.

After the type, we require an = to specify the actual content of the macro. The actual working depends on the type of macro. We close the macro with an ;.

5 Literals

Since all languages require some hardcoding to make everyone's live easier, we also support it. You have several types of literals.

5.1 String literals

Example: "$Hello$"
String literals are denoted by ""s. Unfortunately that means we don't support the use of " in string literals. We encourage any future developers to work on supporting this.

5.2 Boolean literals

Examples: $true$, $false$
Boolean literals are true or false.

5.3 Integer literals

Example: 5
Integer literals are quite simple. They are any number not containing a dot.

5.4 Float literals

Example: 5.0
Float literals are also doable. They are any number containing a dot.

5.5 Date literals

Examples: $\#1995-01-17\#$, $\#13:33\#$, $\#1995-01-23\ 23:35:54\#$

Dates are more tricky. We only support one date format which is year-month-day. This is to prevent ambiguity between the day-month-year and month-day-year formats. The time format is hour:minutes:seconds or hour:minutes. In the latter case the seconds are set to 0.

Dates are always enclosed between hashtags.

Combined the two will result in a datetime.

5.6 Period literals

Examples: $\#5\ DAYS\#$, $\#0\ MOTNHS\#$, $\#5\ YEARS\#$

Periods are a number of days, months or years. This is most useful when adding or subtracting a period of time from a date.

Periods are also enclosed between hashtags and the supported units are DAYS, MONTHS and YEARS.

6 Operators

In our language we support many operations on numbers and booleans.

For numbers we support:

- $+$
- $-$
- $*$
- $/$
- $\hat{}$
- $\%$
- $SQRT$
- $>$
- $<$
- $<=$
- $>=$

For booleans we support:

- AND
- OR
- NOT

All values support the equality operation $=$.

6.1 Date operations

To operate on dates we have several functions which can be called.

6.1.1 Relative

In order to work more freely with dates we support the RELATIVE function. This functions calculates the differences between 2 dates in a given time unit.

For example: $RELATIVE(\#1995-01-17\#, \#2015-06-18\#, DAYS)$ Gives the result 7457.

The first 2 arguments are the dates to compare and the third argument is the unit. The supported units are DAYS, MONTHS and YEARS.

6.1.2 Combine

The combine function combines a date and a time to give a date time.

For example: $COMBINE(\#1995-01-17\#, \#12:12\#)$
Returns the equivalent of $\#1995-01-17\ 12:12\#$.

The first argument is the date and the second the time.

6.1.3 To date and To time

To get either the time part or date part of a given date time you can call TO_DATE or TO_TIME.

For example: $COMBINE(TO\_DATE(\#1995-01-17\ 12:12\#), TO\_TIME(\#1995-01-17\ 12:12\#))$
Results in $\#1995-01-17\ 12:12\#$ and is a rather useless operation.

TO_DATE and TO_TIME both only take one date time argument.

6.1.4 After and Before

To compare dates we have the BEFORE and AFTER operations.

For example: $\#1995-01-17\#\ BEFORE\ \#1995-01-18\#$
Results in true.

All different temporal values are supported, so BEFORE and AFTER work with times, dates and date times. Do note: that mixing only works between dates and date times. Since $\#1995-01-17\#\ BEFORE\ \#12:00\#$ is nonsense.

6.1.5 Add and Min

To add or subtract time we support the ADD and MIN operations.

For example: $\#1995-01-17\#\ ADD\ \#1\ YEARS\#$
Results in $\#1996-01-17\#$.

The left side argument should either be a date or a date time and the right side argument should be a period. The supported units are DAYS, MONTHS and YEARS.

7 Constraints

Constraints can be somewhat complicated and rightfully so, therefore the declaration should be powerfull, but not overly complicated.

In the end a constraint is simply a boolean expression. This means that both the following are valid constraints:

$def\ simple : Constraint = true;$

$def\ complicated : Constraint = NOT(RELATIVE(\#1995-01-15\#, admire2.date, DAYS) > 2);$

8 Computation

To perform a computation on a table and save the result we support the computation analysis. For example:

$def\ comp : Computation = NAME\ relativeDates\ NEW\ SET\ COLUMNS$
$RELATIVE(\#1995-01-17\#, TO\_DATE(admire2.date), DAYS)\ AS\ date;$
$from(admire2)|computation(comp);$

Results in a new table being called relativeDates containing a single column called date in which there is a integer which is the number of days between my birthday and the given measurement.

The computation has the following syntax:

$defIdentifier : Computation = NAME\ Identifier$
$(NEW|INCLUDE\ EXISTING)$
$SET\ COLUMNS\ (Expression\ AS\ Identifier,)+;$

If INCLUDE EXISTING is used instead of NEW, the columns of the original table are included in the result. As shown you can include multiple columns by separating them with a comma.

9 Group by

We support chunking in the form of a group by. We have 2 kinds of group by operations. One is the group by column. You specify a column in which a single value forms a single group. The other is the group by constraint in which you specify a set of constraints, in which each constraint forms a group. For example:

$def\ group : GroupByColumn = NAME\ perday$
$ON\ admire2.date$
$FROM\ MAX(admire2.measurement)\ AS\ max,$
$AVERAGE(admire2.measurement)\ AS\ avg;$
$from(admire2)|groupBy(group)$

$def\ group : GroupByConstraint = NAME\ dayparts$
$ON\ admire2.time\ BEFORE\ \#12:00\#\ AS\ morning,$
$admire2.time\ AFTER\ \#12:00\#\ AS\ afternoon$
$FROM\ MAX(admire2.measurement)\ AS\ max,$
$AVERAGE(admire2.measurement)\ AS\ avg;$
$from(admire2)|groupBy(group)$

The group by column has the following syntax:

$def\ Identifier : GroupByColumn = NAME\ Identifier$
$ON\ ColumnIdentifier$
$FROM\ (Function(ColumnIdentifier)\ AS\ Identifier,)+;$

The group by constraint has the following syntax:

$def\ Identifier : GroupByColumn = NAME\ Identifier$
$ON\ (BooleanExpression\ AS\ Identifier,) +$
$FROM\ (Function\ AS\ Identifier,)+;$

10 Connections and joins

In order to combine 2 tables, we support both joins and connections. For example:

$def\ joinDif : Join = JOIN\ admire2$
$WITH\ filtered\ AS\ difference$
$ON\ admire2.date = filtered.Date$
$AND\ NOT(admire2.Value = filtered.Value);$
$join(joinDif)$

$def\ con : Connection = admire2\ ON\ admire.date$
$WITH\ filtered\ ON\ filtered.Date\ AS\ connected$
$FROM\ admire2.value\ AND\ filtered.Value;$
$connection(con)$

The syntax is:

$def\ Identifier : Join = (FULL\ JOIN|LEFT\ JOIN|RIGHT\ JOIN|JOIN)$
$Identifier\ WITH\ Identifier\ AS\ Identifier$
$ON\ BooleanExpression$
$(FROM\ (ColumnIdentifier\ AND\ ColumnIdentifier,)+)?$

$def\ Identifier : Connection = Identifier\ ON\ ColumnIdentifier$
$WITH\ Identifier\ ON\ ColumnIdentifier\ AS\ Identifier$
$ON\ BooleanExpression$
$(FROM\ (ColumnIdentifier\ AND\ ColumnIdentifier,)+)?$

11 Comparison

Comparisons allow you to compare 2 tables. For example:

$def\ comp : Comparison = admire2\ WITH\ filtered\ AS\ compared$
$ON\ admire2.date\ TO\ filtered.Date;$
$compare(comp)$

The syntax is: $def\ Identifier : Comparison = Identifier\ WITH\ Identifier\ ASIdentifier$
$ON\ ColumnIdentifier\ TO\ ColumnIdentifier;$

There is also a time between process which calculates the time between the rows of a table. For example: $from(admire2)|timeBetween(date)$

12 Codes

Coding is done by specifying the rows in a table to code and specifying the codes. For example:

$def\ gtNine : Constraint = admire2.value > 9;$
$from(admire2)|constraint(gtNine)|setCode(\text{"}above9\text{"},\ admire2)$

To then use the code in a boolean expression we support the HAS_CODE operation.

So $def\ codeCheck : Constraint = HAS\_CODE("above9");$
$from(admire2)|constraint(codeCheck)$ will return the same as the gtNine constraint above.

13 Other processes

13.1 Reading and writing tables

The from process has been in various examples. It simply reads the table given in the parameter and outputs it into the next process.

The is process does the opposite of the from, it takes an input and saves it to the model.

So renaming a table: $from(admire2)|is(testTable)$

13.2 Sorting tables

Sorting a table is done through the sort process. For example: $from(test1)|sort(test1.value, "ASC")|is(sorte$

Is you'd wanted it descending just replace "ASC" with "DESC".

13.3 Set operations

We support both the union and difference set operations.

So the following will add one table to another and then remove all the rows to return to the original: $union(test1, test2)|difference(test1, test2)$

14 Examples

In this section we will provide some examples. We make use of the data from the website and the admire2 patient. The table that contains the admire2 meaurements is called admire2txt and the table for the website is called websitexlsx.

14.1 Start

Before we will start to answer the questions we will reduce the website data. We reduce the website, so that it only contains the row that belongs to the admire2 patients. Next we will reduce that table even further. We select only the rows that correspond to a creatinine value.

$def\ filterUser : Constraint = "admire2" = websitexlsx.Login;$
$def\ con : Constraint = filteredweb.CustomMeasurementId = 346;$

$from(websitexlsx)|constraint(filterUser)|is(filteredweb)|$
$from(filteredweb)|constraint(con)|is(filtered)$

Figure 1: 1_filter.txt

Next we will add a column to the sensor that contains both the time en date.

$def\ comp : Computation = NAME\ sensor$
$INCLUDE\ EXISTING\ SET\ COLUMNS$
$COMBINE(admire2txt.date, admire2txt.time)\ AS\ datetime;$

$from(admire2txt)|computation(comp)$

Figure 2: 2_addDateTime.txt

If you want you can reduce the amount of columns in the sensor.

$def\ comp : Computation = NAME\ sensorred$
$NEW\ SET\ COLUMNS$
$sensor.Value\ AS\ Value,$
$sensor.date\ AS\ Date,$
$sensor.time\ AS\ Time,$
$sensor.datetime\ AS\ DateTime;$

$from(sensor)|computation(comp)$

Figure 3: 3_filterSensor.txt

14.2 Questions

In this section we will give examples analyses for some of the example questions.

14.2.1 What time of the day and on what day do people measure themselves?

The first example shows if the patient measures in the morning, afternoon or evening.

$def\ groupByDay : GroupByColumn = NAME\ measureMoment$
$ON\ filteredweb.Moment\ FROM$
$COUNT(filteredweb.Moment)\ AS\ number_of_measures;$

$from(filteredweb)|groupBy(groupByDay)$

Figure 4: 4_timeMeasured1.txt

The next example checks for each hour how often the patient measured.

$def\ group : GroupByConstraint\ =\ NAME\ measureMoment2\ ON$
$admire2txt.time\ BEFOR\ \#06:00\#\ AS\ early,$
$admire2txt.time\ AFTER\ \#06:00\#$
$AND\ admire2txt.time\ BEFORE\ \#07:00\#\ AS\ six,$
$admire2txt.time\ AFTER\ \#07:00\#$
$AND\ admire2txt.time\ BEFORE\ \#08:00\#\ AS\ seven,$
$admire2txt.time\ AFTER\ \#08:00\#$
$AND\ admire2txt.time\ BEFORE\ \#09:00\#\ AS\ eight,$
$admire2txt.time\ AFTER\ \#09:00\#$
$AND\ admire2txt.time\ BEFORE\ \#10:00\#\ AS\ nine,$
$admire2txt.time\ AFTER\ \#10:00\#$
$AND\ admire2txt.time\ BEFORE\ \#11:00\#\ AS\ ten,$
$admire2txt.time\ AFTER\ \#11:00\#$
$AND\ admire2txt.time\ BEFORE\ \#12:00\#\ AS\ eleven,$
$admire2txt.time\ AFTER\ \#12:00\#\ AS\ late$
$FROM\ COUNT(admire2txt.time)\ AS\ count;$

$from(admire2txt)|groupBy(group)$

Figure 5: 5_timeMeasured2.txt

The next example shows on which days the person does a measurement.

$def\ groupBy : GroupByColumn = NAME\ everyDay\ ON$
$((RELATIVE(admire2txt.date, \#2015-06-21\#, DAYS))\%7)$
$FROM\ COUNT(admire2txt.date)\ AS\ count;$
$from(admire2txt)|groupBy(groupBy)$

Figure 6: 6_dayOfWeek.txt

14.2.2 What time of the day and on what day do they enter measure measurement?

The next two examples are almost the same as the previous one. But these examples uses the website instead of the statsensor.

$def\ group : GroupByConstraint = NAME\ enterMoment\ ON$
$TO\_TIME(filteredweb.CreatedDate)\ BEFORE\ \#06:00\#\ AS\ early,$
$TO\_TIME(filteredweb.CreatedDate)\ AFTER\ \#06:00\#$
$AND\ TO\_TIME(filteredweb.CreatedDate)\ BEFORE\ \#07:00\#\ AS\ six,$
$TO\_TIME(filteredweb.CreatedDate)\ AFTER\ \#07:00\#$
$AND\ TO\_TIME(filteredweb.CreatedDate)\ BEFORE\ \#08:00\#\ AS\ seven,$
$TO\_TIME(filteredweb.CreatedDate)\ AFTER\ \#08:00\#$
$AND\ TO\_TIME(filteredweb.CreatedDate)\ BEFORE\ \#09:00\#\ AS\ eight,$
$TO\_TIME(filteredweb.CreatedDate)\ AFTER\ \#09:00\#$
$AND\ TO\_TIME(filteredweb.CreatedDate)\ BEFORE\ \#10:00\#\ AS\ nine,$
$TO\_TIME(filteredweb.CreatedDate)\ AFTER\ \#10:00\#$
$AND\ TO\_TIME(filteredweb.CreatedDate)\ BEFORE\ \#11:00\#\ AS\ ten,$
$TO\_TIME(filteredweb.CreatedDate)\ AFTER\ \#11:00\#$
$AND\ TO\_TIME(filteredweb.CreatedDate)\ BEFORE\ \#12:00\#\ AS\ eleven,$
$TO\_TIME(filteredweb.CreatedDate)\ AFTER\ \#12:00\#\ AS\ late$
$FROM\ COUNT(filteredweb.Moment)\ AS\ count;$

$from(filteredweb)|groupBy(group)$

Figure 7: 7_enterMoment.txt

$def\ groupBy : GroupByColumn = NAME\ enterDay\ ON$
$((RELATIVE(TO_DATE(filteredweb.CreatedDate), \#2015-06-21\#, DAYS))\%7)FROM\ COUNT(filteredweb.Moment)\ AS\ count;$

$from(filteredweb)|groupBy(groupBy)$

Figure 8: 8_enterDay.txt

14.2.3 Is there a difference between StatSensor measurement and what patients enter into Mijnnierinzicht?

In the example we first join the two tables. Next we perform a computation. In that computation we calculate the difference and select only the relevant tables.

$def\ joinDif : Join = JOIN\ admire2txt\ WITH\ filtered\ AS\ difference$
$ON admire2txt.date = filtered.Date$
$ANDNOT(admire2txt.Value = filtered.Value);$

$def\ comp : Computation = NAME\ joinDifReduced$
$NEW\ SET\ COLUMNS$
$difference.date\ AS\ Date,$
$difference.admire2txt_Value\ AS\ SensorValue,$
$difference.filtered_Value\ AS\ WebValue,$
$(difference.filtered_Value - difference.admire2txt\_Value)\ AS\ dif;$

$join(joinDif)|from(difference)|computation(comp)$

Figure 9: 9_difference.txt

14.2.4 How often do patients measure themselves before they enter data into Mijnnierinzicht?

$def\ groupByDay : GroupByColumn = NAME\ timesMeasured$
$ON\ admire2txt.date\ FROM$
$COUNT(admire2txt.date)\ AS\ number\_of\_measures;$

$from(admire2txt)|groupBy(groupByDay)$

Figure 10: 10_timeMeasured.txt

14.2.5 If a patient did measure multiple time, what measure do he/she eventually enter into Mijnnierinzicht?

First we filter the situations where the patient measured multiple times. Next use a join to select only the rows that belong to those dates. Finally we select only the relevant columns.

$def\ joinMult : Join = JOIN\ timesMeasured\ WITH\ filtered\ AS\ multipleMeasures$
$ON\ timesMeasured.Chunk = filtered.Date$
$AND\ timesMeasured.number > 1;$

$def\ joinMultAdmire : Join = JOIN\ multipleMeasures\ WITH$
$admire2txt\ AS\ multipleMeasuresSensor$
$ON\ multipleMeasures.Chunk = admire2txt.date;$

$def\ comp : Computation = NAME\ multipleShort$
$NEW\ SET\ COLUMNS$
$multipleMeasuresSensor.multipleMeasures_Value\ AS\ ValueWeb,$
$multipleMeasuresSensor.admire2txt_Value\ AS\ ValueAdmire,$
$multipleMeasuresSensor.Chunk\ AS\ Date,$
$multipleMeasuresSensor.number\ AS\ Number;$

$join(joinMult)|join(joinMultAdmire)|from(multipleMeasuresSensor)|computation(comp)$

Figure 11: 11_timeMeasured.txt

14.2.6 How well do patients follow up advice of Mijnnierinzicht to re-measure again?

We select the rows that are a second measurement. Furthermore we select the rows where the website advices to remeasure. Than we we set code "Done" on the rows where the patient had to remeasure and also remeasures. Next we set a code "NotDone" on all the other code. Finally we count the rows based on the codes.

$def\ measured : Constraint = filteredweb.CustomMeasurementId = 415;$

$def\ remeasure : Constraint =$
$filtered.KreatinineAlgorithmActionId = "1";$

$def\ done : Constraint = remeasure.Date = second.Date;$

$def\ group : GroupByConstraint = NAME\ remeasureAmount\ ON$
$HAS\_CODE("Done")\ AS\ Done,$
$HAS\_CODE("NotDone")\ AS\ NotDone$
$FROM\ COUNT(remeasure.Moment)\ AS\ Count;$

$from(filteredweb)|constraint(measured)|is(second)|$
$from(filtered)|constraint(remeasure)|is(remeasure)|$
$from(remeasure, second)|constraint(done)|$
$is(remeasure, temp)|setCode("Done", remeasure)|$
$difference(remeasure, temp)|setCode("NotDone", remeasure)|$
$groupBy(group)$

Figure 12: 12_countRemeasure.txt

14.2.7 What are the conditions under which people overwrite their initial data entered in Mijnnierinzicht?

$def\ con : Constraint = filteredweb.CreatedDate\ BEFORE\ filteredweb.ModifiedDate; from(filteredwe$

Figure 13: 13_updated.txt

14.2.8 Find cases where Mijnnierinzicht advice to contact the hospital

First we check for the situations where the dayRating is 5. In these situations the patient should contact to the hospital. Next we select the rows that have a rating of 3 or 4. Than we select the rows with a rating of 4 and where the day is the day after one of the previous selected rows. In these situations the patient should also contact the hospital. Finally we combine the two cases, so we have one result for all the situation where the patient had to contact the hospital.

$def\ filterUser : Constraint = "admire56" = websitexlsx.Login;$

$def\ con : Constraint = filteredweb56.KreatinineAlgorithmDayRatingId = "5";$

$deffilter : Constraint =$
$filteredweb56.KreatinineAlgorithmDayRatingId = "4"\ OR$
$filteredweb56.KreatinineAlgorithmDayRatingId = "3";$

$def\ hosp : Constraint =$
$filteredweb56.KreatinineAlgorithmDayRatingId = "4"\ AND$
$RELATIVE(temp.Date, filteredweb56.Date, DAYS) = 1;$

$from(websitexlsx)|constraint(filterUser)|is(filteredweb56)|$
$from(filteredweb56)|constraint(con)|is(hosp1)|$
$from(filteredweb56)|constraint(filter)|is(temp)|$
$from(filteredweb56, temp)|constraint(hosp)$
$|is(filteredweb56, hosp2)|union(hosp1, hosp2)|is(hospital)$

Figure 14: 14_hospital.txt

14.2.9 What are the conditions under which people start deviating from their normal measurement routine?

In the first example we set a code normal on the measurements in the second period that are normal measurements. We also sort the result on the time. This is not necessary, but it makes inspecting the result easier.
The normal routine is every other day. First we select only the rows from the second period. Next we select the rows where there is also a row two days in the future. Than we remove the rows for where there is also a row one day in the future. Next we set a code on the normal rows.

$def\ comp : Computation = NAME\ temp$
$INCLUDE\ EXISTING\ SET\ COLUMNS$
$MIN(filtered.Date)\ AS\ min;$

$def\ filter : Constraint =$
$RELATIVE(temp.min, temp.Date, DAYS) > 21\ AND$
$RELATIVE(temp.min, temp.Date, DAYS) < 64;$

$def\ normal : Constraint =$
$RELATIVE(temp.Date, temp2.Date, DAYS) = 2;$

$def\ toOften : Constraint =$
$RELATIVE(temp.Date, temp2.Date, DAYS) = 1;$

$from(filtered)|computation(comp)|$
$from(temp)|constraint(filter)|is(temp)|is(temp2)|$
$from(temp, temp2)|constraint(normal)|is(temp, normal)|$
$from(temp, temp2)|constraint(toOften)|is(temp, temp3)|$
$difference(normal, temp3)|is(normal)|$
$from(temp)|sort(temp.Date, "ASC")|from(normal)|setCode("normal", temp)$

Figure 15: 15_normalPeriod2.txt

In the next example we filter the normal and the abnormal measurements from the third period. We add a column which specifies the week of the measurement. We select only the weeks from the third period. The patient should measure once a week. So we count how often the patient measures.

$def\ comp : Computation = NAME\ temp$
$INCLUDE\ EXISTING\ SET\ COLUMNS$
$MIN(filtered.Date)\ AS\ min;$
$def\ filter : Constraint = temp2.week > 9\ AND\ temp2.week < 19;$

$def\ normalCount : Constraint = measures3.Count = 2;$

$def\ normal : Constraint = temp3.Chunk = temp2.week;$

$def\ week : Computation = NAME\ temp2$
$INCLUDE\ EXISTING\ SET\ COLUMNS$
$((RELATIVE(temp.min, temp.Date, DAYS)) -$
$((RELATIVE(temp.min, temp.Date, DAYS))\%7))/7\ AS\ week;$

$def\ groupBy : GroupByColumn = NAMEmeasures3\ ON\ temp2.week$
$FROM\ COUNT(measures3.week)\ AS\ Count;$

$from(filtered)|computation(comp)|$
$from(temp)|computation(week)|computation(filter)|is(temp2)|$
$groupBy(groupBy)|constraint(normalCount)|is(temp3)|$
$from(temp2, temp3)|constraint(normal)|is(temp2, normal3)|$
$difference(temp2, normal3)|is(notNormal3)$

Figure 16: 16_normalPeriod3.txt