Language Specification for SUGAR

Health Informatics 3

June 15, 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 and Connection. We plan on implementing a few more.

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 Integer literals

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

5.3 Float literals

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

5.4 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.

5.5 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 Constraints

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

In the previous example we saw a constraint, let's take a closer look.

In the $measurement.id = 1$ we declare that for the given table the column id should be 1, simple enough.

In the $measurement.level > 5$ we declare that level should be larger than 5.

With $table.column <= 2$ and $table.column >= 2$ we declare that the table's column should be less/larger or equal to 2.

Next we have the AND, OR and NOT statement. With AND we declare that both constrains should hold. For example when we have $table.id > 2\ AND\ table.id < 4$, than the id should be larger than 2 and less than 4. With OR we declare that one or more constrains should hold. So when we have $table.name = "Matthijs"\ OR\ table.name = "Bob"$, than name must be "Matthijs" or the name must be "Bob".

For dates we have the BEFORE and AFTER commands, which do exactly what you'd expect. For example $\#1995-01-17\#\ BEFORE\ \#1994-01-17\#$ returns true.

7 Computation

We should support basic operations such as $+, -, *, /, pow(), sqrt()$.

For time operations we have the ADD and MIN operation to add or subtract time. An example would be $table.dateMIN\#1DAYS\#$, which does exactly what you'd expect.

We also support the RELATIVE operation. This calculates the differences between 2 dates in the given unit. So $RELATIVE(\#1995-01-17\#, \#1995-01-18\#, DAYS)$ results in 1. Since this is an integer, you can then apply the other computations to it.

7.1 Functions

For aggregates we should support MAX, MIN, AVG, COUNT, SUM, MEDIAN, STDDEV

8 Chunking

Chunking is just a special kind of constraint. We should support the same operation $>, <, >=, <=, AND,\ OR,\ NOT$

9 Connections

Connections should be directed and should be capable of being many to many. This means implementing such operators:

(First,) Results, Triggered by, Constraints operands.

The first means that we only connect one row to another. The results and triggered by are to be used when checking if there are any connections. The

This results in something like this:
$def\ measurementInput(measurement, input) : Connection = FIRST\ input\ WHERE\ input.time < measurement.time\ AND\ input.date >= measurement.date\ ORDER\ input.date, input.time\ ASC$

$connection(measurementInput, statSensor, website)|is(connected)$

10 Coding

Codings are quite complicated. We want to cover all possible combinations of events. Let's say something like this: $def\ input(input, measurement) : Coding = IF\ input\ WHERE\ measurement.time < (input.time - 5\ min)\ AND\ measurement.date = input.date\ STORE\ "MI", input.time, input.date$

$coding(input, website, statSensor)|append(coded)$

To code the taking of a measurement and immediately entering it in the website.

So you specify IF there is a certain row, for which there is another row or multiple other rows and then you output a code and certain other values. There should be a way to connect rows and the code back together afterwards.

11 Comparison

TBD

12 Conversion

TBD