# GLUKOZA grammars

Prototypes for the Alexa Diabetes Challenge

Anton Antonov
Accendo Data LLC
Clearsense LLC
May 2017

## On-boarding new user dialogs grammar

```
ebnfNewUser = "
<command> = <age-statement> | <sex-statement> | <race-statement> | <weight-statement> | <height-statement> | <name-statement> | <diabetic-statement> | <combined-statement> ;
<i-am-start> = [ 'so' ] , [ 'it' , 'seems' ] , ( 'i' , 'am' ) ;
<name-statement> = ( <i-am-start> | 'my' , 'name' , 'is' ) &> <person-name> <@ UserName ;
<person-name> = '_LetterString' , [ '_LetterString' ] ;
<age-statement> = <i-am-start> &> 'Range[0,100]' <& [ 'years' , 'old' ] | 'Range[0,100]' <& ( 'years' , 'old' ) <@ UserAge ;
<sex-statement> = <i-am-start> &> ( 'male' | 'female' ) <@ UserSex ;
<race-statement> = ( <i-am-start> | 'my' , 'race' , 'is' ) &> ( 'african-american' | 'asian' | 'black' | 'hispanic' | 'native' , 'american' | 'white' ) <@ UserRace ;
<height-statement> = ( <i-am-start> | 'my' , 'height' , 'is' ) &> <length-spec> <@ UserHeight ;
<length-spec> = 'Range[0,300]' ;
<weight-statement> = [ <i-am-start> | 'my' , 'weight' , 'is' ] &> 'Range[10,600]' , ( 'pounds' | 'kilograms' | 'lbs' | 'kg' ) <@ UserWeight ;
<combined-statement> = <name-statement> , [ 'and' ] &> ( <age-statement> | <diabetic-statement> | <weight-statement> ) ;
<diabetic-statement-filler> = [ 'the' | 'a' ] | [ 'the' ] , [ 'in' ] , [ 'the' ] , ( 'house' | 'here' | 'now' ) ;
<diabetic-statement> = ( <i-am-start> , [ <diabetic-statement-filler> ] ) &> ( ( [
    'type' , ( 'one' | 'two' ) ] , 'diabetic' ) | ( 'prediabetic' | 'pre-diabetic' ) ) <& [ <diabetic-statement-filler> ] <@ UserDiabet ;
";
```

```
ebnfNewUser = StringReplace[ebnfNewUser, "<" ~~ x : (Except[{"<", ">"}] ..) ~~ ">" :> "<" <> "user-" <> x <> ">"];
```

```
res = GenerateParsersFromEBNF[ParseToEBNFTokens[ebnfNewUser]];
res // LeafCount
```

```
1435
```

```
grammarWords = {"african-american", "asian", "black", "hispanic", "native", "american", "white", "female", "male", "type", "diabetic", "prediabetic", "the", "an", "and"};
```

```
pPERSONNAME1[xs_] := ParsePredicate[StringQ[#] && ! MemberQ[Evaluate[grammarWords], #] &][xs];
pUSERPERSONNAME := pPERSONNAME1 ⊗ ParseOption1[pPERSONNAME1];
```

```
queries = {"i am nina", "i am female", "i am 26 years old", "i am asian", "my race is asian", "my weight is 170 pounds", "i am 70 kilograms", "i am type two diabetic", "i am prediabetic",
   "i am nina markova and i am diabetic", "my name is nina markova and my weight is 160 pounds", "i am the diabetic in the house", "i am pablo and it seems i am a diabetic now"};
ParsingTestTable[ParseShortest[pUSERCOMMAND],
 queries]
```

| # | Statement | Parser output |
|---|---|---|
| 1 | i am nina | {{{}, UserName[{nina, {}}]}} |
| 2 | i am female | {{{}, UserSex[female]}} |
| 3 | i am 26 years old | {{{}, UserAge[26]}} |
| 4 | i am asian | {{{}, UserRace[asian]}} |
| 5 | my race is asian | {{{}, UserRace[asian]}} |
| 6 | my weight is 170 pounds | {{{}, UserWeight[{170, pounds}]}} |
| 7 | i am 70 kilograms | {{{}, UserName[{70, kilograms}]}} |
| 8 | i am type two diabetic | {{{}, UserDiabet[{{type, two}, diabetic}]}} |
| 9 | i am prediabetic | {{{}, UserDiabet[prediabetic]}} |
| 10 | i am nina markova and i am diabetic | {{{}, {UserName[{nina, markova}], UserDiabet[{{}, diabetic}]}}} |
| 11 | my name is nina markova and my weight is 160 pounds | {{{}, {UserName[{nina, markova}], UserWeight[{160, pounds}]}}} |
| 12 | i am the diabetic in the house | {{{}, UserDiabet[{{}, diabetic}]}} |
| 13 | i am pablo and it seems i am a diabetic now | {{{}, {UserName[{pablo, {}}], UserDiabet[{{}, diabetic}]}}} |

```
ParseShortest[pUSERCOMMAND][ToTokens["i am hall clement and i am type one diabetic"]]
```

```
{{{}, {UserName[{hall, clement}], UserDiabet[{{type, one}, diabetic}]}}}
```

# Time interval

Download and read the file TimeSpecificationsGrammar.ebnf from MathematicaForPrediction at GitHub; see [1].

```
ebnfTimeInterval = Get["~/MathematicaForPrediction/EBNF/TimeSpecificationsGrammar.ebnf"];
```

**ebnfTimeInterval**

```
<time-interval> = <time-interval-spec> | <number-of-time-units> | <week-of-year> | <month-of-year> ;
<time-unit> = 'hour' | 'day' | 'week' | 'month' | 'year' | 'lifetime' ;
<time-units> = 'hours' | 'days' | 'weeks' | 'months' | 'years' | 'lifetimes' ;
<number-of-time-units> = <number> , <time-units> | ( 'a' | 'one' ) , <time-unit> ;
<named-time-intervals> =  <day-name-relative> | <time-interval-relative> | <month-name> ;
<time-interval-relative> = [ 'the' ] &> ( 'next' | 'last' ) , ( <time-unit> | ( 'few' | <number> ) , <time-units> ) ;
<time-interval-spec> =  'between' , <time-spec> , 'and' , <time-spec> | 'from' , <time-spec>
  , 'to' , <time-spec> | [ 'in' | 'during' ] &> <named-time-intervals> | 'between' , <number> , 'and' , <number-of-time-units> ;
<time-spec> = <right-now> | <day-name> | <week-number> |  <week-of-year> | <month-name> | <month-of-year> |
  <holiday-name> | <hour-spec> | <holiday-offset> <@ TimeSpec[If[ListQ[#],Flatten[#],#]]& ;
<right-now> = 'now' | 'right' , 'now' | 'just' , 'now' ;
<day-name-relative> = 'today' | 'yesterday' | 'tomorrow' | 'the' , 'day' , 'before' , 'yesterday' ;
<day-name-long> = 'monday' | 'tuesday' | 'wednesday' | 'thursday' | 'friday' | 'saturday' | 'sunday' ;
<day-name-long-plurals> = 'mondays' | 'tuesdays' | 'wednesdays' | 'thursdays' | 'fridays' | 'saturdays' | 'sundays' ;
<day-name-abbr> = 'mon' | 'tue' | 'wed' | 'thu' | 'fri' | 'sat' | 'sun' ;
<day-name> = <day-name-long> | <day-name-abbr> | <day-name-long-plurals> ;
<week-number> = 'week' , <week-number-range> ;
<week-number-range> = 'Range[1,52]' ;
<week-of-year> = [ 'the' ] , 'week' , <week-number-range>  , 'of' , <year> ;
<month-name-long> = 'january' | 'february' | 'march' | 'april' | 'may' | 'june' | 'july' | 'august' | 'september' | 'october' | 'november' | 'december' ;
<month-name-abbr> = 'jan' | 'feb' | 'mar' | 'apr' | 'may' | 'jun' | 'jul' | 'aug' | 'sep' | 'oct' | 'nov' | 'dec' ;
<month-name> = <month-name-long> | <month-name-abbr> ;
<month-of-year> = <month-name> , [ 'of' ] , [ <year> ] ;
<year> = 'year' , <year-number-range> | <year-number-range> ;
<year-number-range> = 'Range[1900,2100]' ;
<holiday-name> = 'ramadan' | 'christmas' | 'thanksgiving' | 'memorial' , 'day' | 'lincoln' , 'day' | 'new' , 'year' | 'mother' , 'day' ;
<holiday-offset> = <number-of-time-units> , ( 'before' | 'after' ) , <holiday-name> ;
<number> = '_?NumberQ' ;
<hour-spec> = 'Range[0,23]' , [ ( 'am' | 'pm' ) ] ;
<full-date-spec> = ( <number> , <month-name> | <month-name> , <number> ) , <number> , [ <number> , ( 'am' | 'pm' ) ] ;
```

```
res = GenerateParsersFromEBNF[ToTokens[ebnfTimeInterval]];
res // LeafCount
```

2536

```
queries = {"yesterday", "from monday to right now", "in april", "last week", "from january to april", "2 months", "a lifetime"};
ParsingTestTable[ParseJust[pTIMEINTERVAL], queries]
```

| # | Statement | Parser output |
|---|---|---|
| 1 | yesterday | {{{}, yesterday}} |
| 2 | from monday to right now | {{{}, {from, {TimeSpec[monday], {to, TimeSpec[{right, now}]}}}}} |
| 3 | in april | {{{}, april}} |
| 4 | last week | {{{}, {last, week}}} |
| 5 | from january to april | {{{}, {from, {TimeSpec[january], {to, TimeSpec[april]}}}}, {{}, {from, {TimeSpec[january], {to, TimeSpec[{april}]}}}}, {{}, {from, {TimeSpec[{january}], {to, TimeSpec[april]}}}}, {{}, {from, {TimeSpec[{january}], {to, TimeSpec[{april}]}}}}} |
| 6 | 2 months | {{{}, {2, months}}} |
| 7 | a lifetime | {{{}, {a, lifetime}}} |

# Medicine prescriptions module

TBD...

# Diabetes education module grammar

TBD...

# Cohort specification

```
ebnfCohort = "
<cohort> = <cohort-single-user> | <cohort-predicate> <@ Cohort ;
<cohort-single-user> = 'me' | [ 'the' ] , [ 'current' ] , 'user' ;
<cohort-predicate> =  ( 'all' | 'overweight' | 'diabetic' | 'prediabetic' ) , 'users' ;
";
```

TBD...

# Exercise activities logging module

TBD...

# Time series analysis module

Modify the original definition of the time series conversational agent grammar in the file TimeSeriesConversationalEngineGrammar.ebnf from MathematicaForPrediction at GitHub; see [2].

## Original definition

## New definition

Note that <time-interval> and <cohort-spec> are defined above.

```
ebnfTimeSeries = "
<preamble> = 'find' | 'compute' | 'calculate' | 'show' ;
<nutrition-element> = (  'calorie' | 'calories' | 'protein' | 'carbs' | 'sodium' | 'fat' ) <& [ 'intake' ] <@ TSNutritionElement ;
<nutrition-spec-straight> = [ 'the' ] , ( <nutrition-element> <& ( 'of' | 'for' ) ) , <cohort-spec> , [ <time-interval> ] <@ TSNutritionSpec[#]& ;
<nutrition-spec-reverse> = [ 'the' ] &> <cohort-spec> , <nutrition-element> , [ <time-interval> ] <@ TSNutritionSpec[Reverse[#]]& ;
<nutrition-spec> = <nutrition-spec-straight> | <nutrition-spec-reverse> ;
<cohort-spec> =  'me' | [ 'the' ] , [ 'current' ] , 'user' | ( 'all' | 'overweight' | 'diabetic' | 'prediabetic' ) , 'users' <@ TSCohortSpec[Flatten[{#}]]& ;
<medicine-spec> = '_String' <@ TSMedicineSpec ;
<finance-element> = [ 'total' ] , 'price' | [ 'intake' ] , 'volume' <@ TSFinancialElement ;
<finance-spec-straight> = [ 'the' ] &> <finance-element> , 'of' , <medicine-spec> <@ TSFinancialData ;
<finance-spec-reverse> =  [ 'the' ] &> <medicine-spec> , <finance-element> <@ TSFinancialData[Reverse[#]]& ;
<finance-spec> = <finance-spec-straight> | <finance-spec-reverse> ;
<past-data-spec> = [ 'the' ] &> 'last' , [ 'loaded' ] , ( 'data' | 'file' ) <@ TSPastData[Flatten[{#}]]& ;
<data-spec> = <nutrition-spec> | <finance-spec> | <past-data-spec> ;
<regression-quantile-bsplines> = [ ( '1' | 'one' | 'a' ) ] , 'regression' , 'quantile' | 'quantile' , 'regression' <@ TSBSplineQRegression[1]& ;
<regression-quantiles-bsplines> = [ 'Range[1,40]' ] <& ( 'regression' , 'quantiles' ) <@ TSBSplineQRegression[#]& ;
<regression-quantiles> = <regression-quantile-bsplines> |  <regression-quantiles-bsplines> ;
<outliers> = [ 'the' ] &> [ ( 'top' | 'bottom' | 'largest' | 'smallest' | 'all' ) ] , 'outliers' <@ TSOutliers[Flatten[#]]& ;
<least-squares> = ( 'least' , 'squares' , [ 'fit' ] , [ 'with' | 'of' ] ) &> '_String' <@ TSLeastSquaresFit[#]& ;
<operation-spec> = <regression-quantiles> | <least-squares> | <outliers> ;
<operate-command> = [ <preamble> ] &> <operation-spec> <@ TSOperateCommand[#]& ;
<operate-on-data-command> = <operate-command> , ( 'for' | 'on' | 'in'  | 'over' | 'of' ) &> <data-spec> <@ TSOperateOnDataCommand[#]& ;
<load-file-command> = ( 'load' , [ 'data' ] , 'file'  ) &> ( '_String' ) <@ TSLoadFile[#]& ;
<load-data-command> = ( ( 'load' | 'get' | 'consider' ) , [ 'the' ] , [ 'data' ] ) &> <data-spec> <@ TSLoadData[#]& ;
<start-over> = 'start' , 'over' | 'clear' <@ TSStartOver[Flatten[{#}]]& ;
<clear-graphics> = 'clear' , ( 'plots' | 'plots' | 'graphics' ) <@ TSClearGraphics ;
<what-operations> = 'what' , ( ( 'operations' , 'are'  |  [ 'are' ] , [ 'the' ] , 'operations' ) , [ 'implemented'
    | 'in' ] ) | [ 'what' ] , ( 'operation' | 'operations' ) , ( 'I' , 'can' | 'to' ) , ( 'use' | 'do' ) <@ TSWhatOperations[Flatten[{#}]]& ;
<help-all> = 'help' | [ 'all' ] , 'commands' <@ TSHelp[Flatten[{#}]]& ;
<help> = <help-all> | <what-operations> ;
<plot-joined> = [ 'plot' | 'plots' ] , 'joined' | 'Joined' , '->' , 'True' | 'Joined->True' <@ TSPlotJoined ;
<plot-not-joined> = [ 'plot' | 'plots' ] ,  ( 'not' | 'non' ) , 'joined' | 'Joined' , '->' , 'False' | 'Joined->False' <@ TSPlotNotJoined ;
<plot-data> = 'plot' , 'data' <@ TSPlotData ;
<plot-command> = <plot-data> | <plot-joined> | <plot-not-joined> ;
<command> = <load-data-command> | <load-file-command> | <operate-command> | <operate-on-data-command> | <clear-graphics> | <start-over> | <help> | <plot-command> <@ TSCommand ;
";
```

```
ebnfTimeSeries = StringReplace[ebnfTimeSeries, "<" ~~ x : (Except[{"<", ">"}] ..) ~~ ">" :> "<" <> "ts-" <> x <> ">"];
```

```
(*ebnfTimeSeries=StringReplace[ebnfTimeSeries,"<ts-time-interval>"→"<time-interval>"]*)
```

```
res = GenerateParsersFromEBNF[ToTokens[ebnfTimeSeries]];
res // LeafCount
```

```
3179
```

```
pTSTIMEINTERVAL = ParseApply[TSTimeInterval, pTIMEINTERVAL];
```

### Parser testing

```
queries = {"load the calorie intake of the current user", "load the calories of diabetic users in april",
   "get the calories of diabetic users between new year and now", "load the total price of diatex", "bottom outliers", "find the top outliers"};
ParsingTestTable[ParseJust[pTSCOMMAND], queries]
```

| # | Statement | Parser output |
|---|---|---|
| 1 | load the calorie intake of the current user | {{{}, TSCommand[TSLoadData[TSNutritionSpec[{{}, {TSNutritionElement[calorie], {TSCohortSpec[{the, current, user}], {}}}}]]]}} |
| 2 | load the calories of diabetic users in april | {{{}, TSCommand[<br>TSLoadData[TSNutritionSpec[{{}, {TSNutritionElement[calories], {TSCohortSpec[{diabetic, users}], TSTimeInterval[april]}}}]]]}} |
| 3 | get the calories of diabetic users between new year and now | {{{}, TSCommand[TSLoadData[TSNutritionSpec[{{}, {TSNutritionElement[calories],<br>{TSCohortSpec[{diabetic, users}], TSTimeInterval[{between, {TimeSpec[{new, year}], {and, TimeSpec[now]}}}]}}]]]}} |
| 4 | load the total price of diatex | {{{}, TSCommand[TSLoadData[TSFinancialData[{TSFinancialElement[{total, price}], {of, TSMedicineSpec[diatex]}}]]]}} |
| 5 | bottom outliers | {{{}, TSCommand[TSOperateCommand[TSOutliers[{bottom, outliers}]]]}} |
| 6 | find the top outliers | {{{}, TSCommand[TSOperateCommand[TSOutliers[{top, outliers}]]]}} |

# References

[1] Anton Antonov, Time specifications grammar in EBNF, (2016), MathematicaForPrediction at GitHub. URL: https://github.com/antononcube/MathematicaForPrediction/blob/master/EBNF/TimeSpecificationsGrammar.ebnf .

[2] Anton Antonov, Time series conversational engine grammar in EBNF, (2016), MathematicaForPrediction at GitHub. URL: https://github.com/antononcube/MathematicaForPrediction/blob/master/EBNF/TimeSeriesConversationalEngineGrammar.ebnf .