Dictation Evaluation Reddit Parser

Garcia, Benjamin Lembke, Logan Smith, Christopher Stelter, Andrew

September 21, 2018

Contents

Part I
Language Tutorial

1 Introduction

Part II
Refrence Guide

This section describes the DERP language, as outlined in the DERP Language white paper submitted to the course CSC 792 (Compilers) at the South Dakota School of Mines & Technology in the Fall of 2018. This reference defines all of the main constructs of the language and gives some examples of their usages; it will not contain hard definitions of any of the specific language constructs defined in plugins to the language; however, it may use some as examples.

2 Syntax Notation

Syntax definitions in this reference will be separated from surrounding text by an empty line, indented, and written in a monospace font. The grammar definitions will follow Backus-Naur form, occasionally using Regular Expression syntax for brevity. Additionally, definitions will use the following conventions:

- Terminals will be unformatted; keywords will be expressed as a sequence of terminal characters.
- Nonterminals will be *italicized*
- Characters which could be considered either terminals or BNF/Regular Expression syntax will be **bold** when they should be interpreted as part of BNF or Regular Expression syntax.
- All terminals shown are lower-case text; however, DERP parsers should match them case-insensitive.
- Production definitions follow the format

```
nonterminal :          textitformula
```

Where the *formula* is any syntactic rules for parsing the *nonterminal*. Furthermore, the rest of this manual will reference specific terminals, and nonterminals inline using the same monospace font and formatting.

3 A Note About Language Extension

Because one of the main features of DERP is that it supports loading and unloading of language extension modules, there are some guarantees we can make in this document which will only hold true for the base language with no loaded modules.

Modules are allowed to define additional parse rules for the following tokens of DERP:

- source
- field
- sortkey

Furthermore, modules are allowed to define any additional parse rules they would like, so long as those parse rules are used within the additional definitions for the above tokens and do not redefine other DERP parse rules. Fields and sources are defined further below in [section #]. For example, a source module for interfacing with a Reddit API module might define the following rules:

```
source :    subreddit string | reddit

field :     title | upvotes | tags | nsfw

nsfw  :     nsfw | not safe for work

sortkey :    popular
```

4 Lexical Conventions

4.1 White Space

White space in the DERP language is interpreted in one of three ways, depending upon its context. Any whitespace character (that is, any character in the set [\r, \n, \t, ' ']) can be used as a part of a string literal used to identify something to the interpreter. The whitespace character \n (a unix-style newline) following any sequence of non-whitespace text indicates the end of a DERP statement to be parsed. Any whitespace character in any other context is ignored.

4.2 Comment Syntax

DERP does not support in-code comments of any sort.

4.3 Common Syntax Elements

A number of common keywords and phrases exist in the DERP language and are used in multiple expressions. Some of them are just common definitions that are useful to have defined, and some are optional syntax elements. The optional syntax elements are injected into expressions to make them sound more like standard English speech when spoken, but they are often not required for the syntax to be valid. The common syntax elements and keywords are defined here.

```
article :    a | an | the

string :    "[a-zA-Z]+"

digit :     [0-9]

number :     digit+([.,]digit+)?
```

5 Language Modes

The DERP language is defined in terms of different modes of operation. Switching to a new mode is achieved through the use of certain expressions which are specific to the current mode. When a mode is triggered, the subset of the language that is valid may change. The next sections give a brief description of each mode, as well as the statements used to switch between them.

5.1 Main Mode

As the name implies, this is the mode that a DERP interpreter should begin in. In this mode, statements that change the loaded plugins, output the instructions for a saved query, read a saved query, clear the currently saved result, or begin query/criteria creation are all valid. See section [section #] for more details about Main Mode statements.

Use a *create_expression* with the selection keyword to switch from Main Mode to Selection Mode

```
create_expression :          create article? new? selection
```

Use a *create_expression* with the criteria keyword to switch from Main Mode to Criteria Mode

```
create_expression :          create article? new? criteria
```

To end your DERP program or close the DERP interpreter use a *stop_expression* in Main Mode

```
stop_expression :          stop | exit
```

5.2 Selection Mode

Selection mode is used to create a new selection statement which can be executed at a later time from Main Mode. While in Selection Mode, statements which build the query, read the query statements, save the query, or exit Selection Mode are all valid. See section [section #] for more details about Selection Mode statements.

To exit Selection Mode (and return to Main Mode), use a *stop_expression*.

5.3 Criteria Mode

Criteria mode is very similar to selection mode; it is used to create criteria rather than selections. The difference between a criterion and a selection is that a criterion cannot be specific to any source; it is a filter that can be applied to a selection, but it is not a valid selection on its own. See section [section #] for more details about Criteria Mode statements.

To exit Criteria Mode (and return to Main Mode), use a *stop_expression*.

6 Data Types

6.1 Post

Posts are the data type that DERP is built around. A Post is nothing more than some body of text and any number of fields containing information about the post. Fields of a post are key-value pairs mapping some name for the field to its value.

While defining the interface of a post is the task of a DERP interpreter writer, some of the functionalities it will likely be required to have are described here.

6.1.1 Retrieving Text Body

Posts should provide some method by which the DERP interpreter can acquire the entire text in the body of the post. It is not required that this information be stored directly in the post object itself, however – Retrieving this data can be delayed until it is requested by the DERP interpreter.

6.2 Source Modules

The first data type that someone writing DERP code will encounter is the Source Module. Source Modules are not created directly by the programmer, but are loaded and unloaded to modify the language. A source consists of, at minimum, new syntax for defining how to reference the source, and a public code interface that the DERP interpreter can use to get results from online.

Source modules are allowed to define new syntax rules for the following DERP nonterminals:

- *source*
- *field*
- *sortkey*

When input is found to match the rule a module provided for source, it indicates that the source module will be able to convert that input to a set of posts.

Source modules are allowed to populate an arbitrary set of key-value fields in the posts they create. Defining new syntax for *field* extends the DERP parser to be able to recognize the keys that the source module may populate in posts it retrieves.

By default, every field is also a sort key; however, it may be desirable to sort posts according to some ordering other than one of their fields. Defining new syntax for *sortkey* extends the language to recognize these new sorting methods.

While defining the interface of a source is the task of a DERP interpreter writer, some of the functionalities it will likely be required to have are described here.

6.2.1 Loading Grammar

As stated above, source modules are allowed to define new syntax and use that syntax to overload parts of the DERP language; therefore, any interface to the source module will likely provide a way for the DERP interpreter to get those changes to the grammar when the module is loaded.

It is anticipated that the grammar definition provided by a module may involve extra definitions of the fields added. These extra definitions can include information such as the data types associated with those fields. Without such a definition, DERP interpreters will not be able to perform type-checking on *qualifiers* before attempting to executing them.

6.2.2 Performing Queries

The most important part of a source module is its ability to take the input that was parsed according to source syntax and produce posts. Because source modules will likely deal with very large databases of information that posts can be retrieved from, this interface will likely need to provide functionalities to start retrieving posts from a specific point in the database and to retrieve a finite number of results from there.

6.2.3 Sorting

If the source module defines some non-standard *sortkey* it will likely also need to provide some method by which the interpreter can sort things when that *sortkey* is specified.

6.2.4 General Matching

One of the special qualifiers defined in [section #], the *on_exp* qualifier, is used as a general-purpose match of posts against some text. If the interpreter does not implement some sort of generic matching routine, it will likely be the responsibility of source modules to determine if a post satisfies this generalized match.

6.3 Criterion

A criterion is a user-defined filter for specifying a subset of the Posts returned from executing a query. Criteria are created by entering Criteria Mode, making a number of statements to define the criterion, and then saving the criterion with a *save_expression*.

6.4 Selection

A selection follows many of the same rules as a Criterion, with the important difference that it can reference a specific source using the syntax rules defined in some source module. Because Selections contain one or more sources to get Posts from, they can be executed with a read statement to return a set of posts.

7 Statements and Expressions

All statements in DERP are a single line of text beginning with some expression and ending with the newline character. There is no provision for splitting a statement across multiple lines. Most expressions listed in this section are valid only in the mode they are listed under, but there is one special expression that is always valid: *stop_expression*. (A version of the *recall_expression* is also valid at all times, but the syntax differs depending on the mode, so each version will be listed under the appropriate mode heading below)

statement : *expression* '\n'

Rather than list every possible expression type here, we use the convention throughout this document that any nonterminal ending with *_expression* is a valid production of *expression*.[1]

7.0.1 Stop

The Stop expression can be used in any context to switch modes back to the previous mode. In Selection or Criteria mode, this means returning to the Main Mode. In Main Mode, the Stop expression is used to end the DERP program. If DERP code is being run directly through an interactive interpreter, this should end the interpreter loop.

The stop expression is simply one of the keywords stop or exit

stop_expression : stop | exit

7.1 Main Mode

The main mode allows users to load and unload modules and to delete, execute, and display the definitions of existing criteria and selections.

7.1.1 Load/Unload

As mentioned above, source modules are imported code. These imports are done with the *load_expression*, and sources modules can be unloaded with the *unload_expression*. While a source is loaded, its parse rules will be used to parse any statements in addition to the standard DERP language rules.

Loading source modules is done with the expression

load_expression : load *string*

Where the string defines the name of the source to be used. It is left to the interpreter application to define the format of the plugin itself and how to make the plugin file accessible to the interpreter.

Unloading source modules is done with the statement

unload_expression : unload *string*

Where the string given should be the same string name that was used to load the module originally. It is not valid to use the unload keyword with a name that was not previously used to load a source module.

[1] It is intentional that on_exp and with_exp do not match this convention.

7.1.2 Recall

The recall expression can be used from Main Mode to retrieve the lines of DERP code that make up a criteria or selection. It follows the syntax

recall_expression : recall *string*

The string following the keyword recall is the name that was given in a preceding save expression from within Criteria Mode or Selection Mode.

7.1.3 Clear

The clear expression is used to delete a created Criteria or Selection. After a name is used with the *clear_expression*, it will not be valid in any future *read_expression*, *recall_expression*, or *match_expression* unless a *create_expression* is used to assign the string a new selection or criterion

clear_expression : clear *string*

7.1.4 Read

Read expressions are used to execute a selection and output the resulting Posts. Optionally, a read expression can have a sort predicate, which defines the order that the resulting Posts should be outputted.

Read expressions follow the syntax

read_expression : read *string sort_predicate*?

sort_predicate : (sorted | ordered) by *sortkey order*?

order : ascending | descending

The string given immediately after the read keyword should correspond to a previously created selection. Any field defined by a source module is to be a valid *sortkey*, however source modules may define additional syntax for *sortkey* to allow sorting by more context-specific means. Any posts returned from the selection for which the *sortkey* is not relevant (for example, if multiple sources are used in the selection, and the *sortkey* is specific to one of the sources) should be sorted to the end of the list in any order.

The DERP language does not define what constitutes 'outputting posts' after they are found using a query. This leaves the interpreter open to any number of formats for conveying post information to the user. It should be noted that DERP selection semantics guarantee that determining which posts can be done on a group of posts, and after they are processed they do not need to be re-processed if another group of posts is retrieved. This allows interpreters to work with stream sources that can provide a ~~semi-infinite~~ very large result as multiple smaller pieces.

7.1.5 Create

The create expression is used exclusively to trigger either selection mode or criteria mode, depending upon whether it is used with the criteria or selection keyword.

Create expressions take the form

create_expression : create *article*? new? (selection | criteria)

7.2 Criteria Mode

After a *create_expression* using the keyword `criteria` is read, all following statements will be interpreted using the criteria creation syntax defined here. Criteria mode ends when the stop statement is found.

Valid expressions in criteria mode can used to add things to the filter or remove them, read back the criteria so far, and save the criteria with a name.

7.2.1 Recall

The *recall_expression* can be used in criteria mode to repeat back the add and remove expressions used for the criterion so far.

recall_expression : recall

While this is not a particularly useful functionality for users who are writing DERP code in a text file to be parsed, it exists in anticipation of DERP interpreters which take input and produce output in some format other text on a screen. For example, it would be useful in an implementation of the DERP interpreter which takes voice commands.

7.2.2 Save

The *save_expression* is used to save the set of *add_expressions* and *remove_expressions* that have been executed since criteria mode was last entered. This set of commands is named according to the second part (the *string*) of the expression. Saved criteria become available for use immediately. Using saved criteria is covered with the `matching` qualifier

save_expression : save as *string*

7.2.3 Add/Remove Posts

The most important expressions in criteria mode are the *add_expression* and *remove_expression*. They build a sequence of executable statements that are used to filter the list of elements. Both expressions have similar form

add_expression : add posts *selector*

remove_expression : remove posts *selector*

Both expressions use the selector, which is a series of one or more qualifiers strung together to indicate what items from a list of posts should be accepted by the criteria.

selector : *qualifier_or*

qualifier_or : *qualifier_or* or *qualifier_and* | *qualifier_and*

qualifier_and : *qualifier_and* and *qualifier* | *qualifier*

Qualifiers can be combined using the words ‘and’ and ‘or’. Precedence rules for these combine keywords follow standard C ||and && precedence. That is, ‘and’ is higher precedence than ‘or’, and both are left-associative. Forcing precedence through the use of parentheses, as C allows, is not permitted in DERP.

qualifiers can be any of a number of defined checks. These checks can look at any field exposed as part of the post and match it against the requirement given by the code. Some examples of valid qualifiers are ``with a date before 1990'', ``which are nsfw'', and ``with over twenty upvotes''[2]

```
qualifier :          with_exp article field date_check date

          |          with_exp string substring_check field

          |          boolean_check field

          |          string_check field string

          |          number_check number field

          |          on_exp string

          |          matching string
```

With the exception of the matching and *on_exp* qualifiers, each of the qualifiers listed here is used to check a single piece of information against a *field* associated with a post. Below are the definitions of sub-expressions used in the qualifiers followed by the definitions of each of the different **_check* qualifier rules.

```
with_exp :          with | without
```

7.2.4 date_check

Date checks, along with the date production, are used in a qualifier to indicate that you would like to filter posts by the date that they were published. The date check can be used to get points from an exact date, or a range beginning or ending at a specific date. Ranges between two separate dates can be achieved through combining multiple date *qualifiers* with the keywords and and or.

Dates contain, at minimum, some year; but they may contain a month, and if they contain a month, they may contain a numbered day of that month. The keywords on, after, and before correspond to the conventional numerical comparisons $=$, $>$, and $<$.

```
qualifier :          with_exp article field date_check date

date_check :          date (on | after | before)

date  :          ( month day?  )?  year

year  :          digit digit digit digit
```

[2]Provided that source modules are loaded which define the syntax

```
Field  :          nsfw | upvotes | date
```

month : january | february | march | april | may | june | july | august | septem-
ber | october | november | december

day : [0-3] *digit*

7.2.5 substring_check

Substring checks are used the compare a string against one of the fields associated with the post and return true if the string is part of the data for that field. For example, if there is a field called "tags", one might use the substring check to determine if "tags" contains the word "saved". The qualifier to do that operation might look something like this: `'with ''saved'' in the tags'`

qualifier : *with_exp string substring_check field substring_check*: in the ?

7.2.6 boolean_check

Boolean checks are used to check the values of fields that contain boolean data. One example is the common Reddit tag 'nsfw'. This tag is so common that a Reddit source module for DERP may make nsfw its own field in posts it creates. DERP code could then filter out posts with this field set to true using the qualifier 'which are not nsfw'

qualifier : *boolean_check field*

boolean_check : which are not?

7.2.7 string_check

String checks are similar to substring checks in that they match a string against the data of a field; however, string checks require that the field's data match exactly what is given in the DERP code. One might write the following qualifier to get posts that are not written by a specific person: `'without the exact author ''fred flintstone'''`. Note, this would require that the source module which generated the post populates the field 'author'

qualifier : *string_check field string*

string_check : *with_exp* the exact

7.2.8 number_check

Number checks are used to do numerical comparisons on the data in post fields.

qualifier : *number_check number field*

number_check : *with_exp*? exactly | *above_expression* | roughly

above_expression : over | greater than

below_expression : under | less than

The keyword exactly, and expressions *above_expression* and *below_expression* are mapped to the traditional mathematical operations $==$, $>$, and $<$, respectively. The keyword roughly indicates a match within some epsilon of the specified value. The value of epsilon used for a roughly match is implementation-defined.

Note that defining the *number_check* production using *with_exp* provides some additional flexibility in how qualifiers can be used to match numbers. To match a number in a way analogous to the standard >= operation, DERP code could specify the single qualifier `'without less than x field'`; however, it may be more natural for the programmer to specify the combined qualifier 'with less than x field or with exactly x field'.

7.2.9 The on_exp match

The *on_exp* is used to do a general match of a post against a topic. Unlike all of the qualifiers listed above, this qualifier is not explicitly tied to any one field in a post. The exact method of doing this check is implementation-defined, and may be part of the source module interface. This qualifier allows for matching in cases where the fields provided are insufficient and in cases where many fields would have to be specified in their own qualifiers to achieve the same result.

qualifier : *on_exp string*

on_exp : on | about

7.2.10 matching

The matching qualifier is the second rule that isn't tied to a specific field or set of fields in a post. Using the matching qualifier, DERP programmers can compose criteria. For example, say a programmer created a criteria to get posts about science from 2017 and named it 'modern science'. The programmer could later create the qualifier `'about cars matching ''modern science'''`. This qualifier would expand to accept any posts from 2017 about the science associated with cars. Obviously, this is a fairly trivial example; the matching qualifier can be used to group qualifiers together and then use them to create arbitrarily complex new qualifiers.

qualifier : matching *string*

For the matching qualifier to be valid, the string must meet a single requirement:

- It must be the name that a criterion was saved in using the save_expression from within criteria creation mode prior to the current DERP command

When the matching qualifier is encountered, the current state of the criterion specified is copied, and stored with the criteria being created. Saving a new criterion with the same name will NOT have any effect on previously defined matching qualifiers.

7.3 Selection Mode

As mentioned above, selection mode is very similar to criteria mode, but has extra capabilities. While *add_expressions* and *remove_expressions* in Criteria Mode cannot be used to indicate posts from a specific source, both expressions CAN indicate one or more sources in Selection Mode. This is indicated by a grammar change in the selector nonterminal.

selector : *qualifier_or* | from *source qualifier_or*?

source : *source* (and | or) *source*

As this Selection Mode syntax shows, selectors may contain one or more sources, and if any sources are specified, then qualifiers are not required. Although sources are defined as being combined through the keywords and and or just like qualifiers, there is no precedence associated with the keywords when they are used to combine sources. Combining sources with either keyword will always indicate posts that are found

in either of the sources. While this may at first seem like an odd specification to have, it makes more sense in the context that DERP is intended to mimic natural language.

The definition of source defined in Selection Mode is really only a utility syntax. The definitions of the nonterminal that will be matched in input come from source modules that are loaded.

Because *remove_expressions* in selection mode are no longer limited to the fields of posts, but can also remove posts based solely on their origin, ambiguities can arise over the behavior of these expressions when posts in a query come from entirely different sources. To keep things simple, the DERP language includes the following semantic rule: *remove_expression* is valid in Selection mode iff

- It does not contain any *source*, and at least one *add_expression* has been executed after selection mode was entered, but before the *remove_expression* in question.
- OR, for every source in the *remove_expression* that matches syntax defined by a source module, that source was used in an *add_expression* after selection mode was entered, but before the *remove_expression* in question.

As an example, this set of statements is valid in Selection Mode...[3]

```
create new selection
    add posts from cnn on Tesla
    remove posts from cnn with "Musk" in title
    add posts from fox
    remove posts with date before 2015
    save as "misc"
stop
```

...but these sets of statements are not

```
Create new selection
    Remove posts about "food"

Create new selection
    add posts from cnn
    remove posts from fox about "metal"
```

[3]Provided that the following syntax has been added

```
source :        cnn | fox
```