Proposal for a machine-verifiable version of the ADQL grammar

A case for PEG

Jon Juaristi Campilllo[1,2] Markus Demleitner[1,2]

[1]Astronomisches Rechen-Institut, Zentrum für Astronomie, Universität Heidelberg

[2]German Astrophysical Virtual Observatory

2019 IVOA Interop meeting, 16 May, Observatoire de Paris

Contents

Context

- ADQL grammar is defined using Backus-Naur Form (BNF)...
- ...which is not machine-verifiable.
- Ideas about writing a machine-verifiable version have been around.
- We suggest Parsing Expression Grammar (PEG) as an alternative.

What is PEG?

- Parsing Expression Grammar
- Proposed by Brian Ford (MIT) in 2004[1]
- *"Recognition-based formal foundation for describing machine-oriented syntax"*
- Straightforward syntax: slightly differing from BNF.
- Both lexing and parsing.

```
# Hierarchical syntax
Grammar    <- Spacing Definition+ EndOfFile
Definition <- Identifier LEFTARROW Expression

Expression <- Sequence (SLASH Sequence)*
Sequence   <- Prefix*
Prefix     <- (AND / NOT)? Suffix
Suffix     <- Primary (QUESTION / STAR / PLUS)?
Primary    <- Identifier !LEFTARROW
            / OPEN Expression CLOSE
            / Literal / Class / DOT

# Lexical syntax
Identifier <- IdentStart IdentCont* Spacing
IdentStart <- [a-zA-Z_]
IdentCont  <- IdentStart / [0-9]

Literal    <- ['] (!['] Char)* ['] Spacing
            / ["] (!["] Char)* ["] Spacing
Class      <- '[' (!']' Range)* ']' Spacing
Range      <- Char '-' Char / Char
Char       <- '\\' [nrt'"\[\]\\]
            / '\\' [0-2][0-7][0-7]
            / '\\' [0-7][0-7]?
            / !'\\' .

LEFTARROW  <- '<-' Spacing
SLASH      <- '/' Spacing
AND        <- '&' Spacing
NOT        <- '!' Spacing
QUESTION   <- '?' Spacing
STAR       <- '*' Spacing
PLUS       <- '+' Spacing
OPEN       <- '(' Spacing
CLOSE      <- ')' Spacing
DOT        <- '.' Spacing

Spacing    <- (Space / Comment)*
Comment    <- '#' (!EndOfLine .)* EndOfLine
Space      <- ' ' / '\t' / EndOfLine
EndOfLine  <- '\r\n' / '\n' / '\r'
EndOfFile  <- !.
```

Figure: PEG defined in PEG

[1] https://bford.info/pub/lang/peg

Characteristics

There are a couple of features which stand out compared to BNF:

- Choice operator (|) is not present.
- Replaced by ordered choice (/) instead.
- No longest match option: ordered choice means it goes for the first match every time.
- Has both negative (!) and positive (&) lookahead operators

Our development

- We have written a PEG definition of the ADQL grammar (**following the 2.1 draft**).
- Based on first suggestion done by Grégory Mantelet (CDS) in April 2017.[2]
- Tested using the `Arpeggio` tool created by Igor Dejanovic.[3]

Available at the `lyonetia` project originally maintained by Dave Morris (ROE) under the `src/peg` folder:
`https://github.com/ivoa/lyonetia/tree/master/src/peg`

[2] `http://mail.ivoa.net/pipermail/dal/2017-April/007667.html`
[3] `http://github.com/textX/Arpeggio`

Preface

Before the things to take into account are shown, some small notes:

- We have had some issues *translating* the BNF definition into PEG.
- The most important ones that should be discused will be noted here.
- This will serve as a guide for future developments
- Ways to fix these quirks should be addressed as soon as possible.

General considerations

Tried to keep close to the BNF, but many things work differently.

- Some rules have been rewritten due to PEG's nature.
- Biggest (and most obvious) change: terminals are not rules anymore.
- Not alphabetically ordered.
- Whitespace management: special rules to check single, multiple, newline, etc.
- Some errors found in the BNF have been fixed, e.g., `HAVING` depends on `GROUP BY`.

Longest match

- Not available
- Ordered choice, due to it own nature, looks for the first matching pattern.
- Existing rules needed to be adapted accordingly and modularised (i.e., more rules!).
- Ambiguous parameters can be affected by this (we have had problems with `value_expression`).

Tokenisation

- Generally works fine: terms are divided accordingly.
- Issues when defining identifiers which contain reserved keywords.
- Currently solved but worth a look.

Example of a faulty string we had: `USER_TABLE`

- The parser matches the SQL reserved keyword `USER`.
- It will fail, when the whole string is actually correct.

Left recursion

- PEG is left-recursive and the parsing works left-to-right top-down.
- Well-formed PEG shouldn't have left recursion.
- Must be taken into account when writing new rules or fixing the existing ones.

Example: rule `A <- A 'a' / 'b'`

- Will infinitely recurse until the parser complains (*maximum recursion depth exceeded*).
- In order to fix it, it should be rewritten as `A <- 'b' 'a'*`

Testing

- Tests inherited from the original `lyonetia` BNF solution have been used (and in some cases, fixed).
- A series of tests solely dedicated to whitespaces, one of the tricky parts of PEG, have also been added.
- New tests which delve into cases battling other PEG quirks.
- Not all properties of the grammar are covered: **more tests are needed**.

Arpeggio

- The tool we found where we could run our tests.
- No particular reason for choosing it. Advantage: written in Python, which we are familiar with.
- It uses a custom PEG syntax: currently using a patched converter.
- Suggestions for using another tool (or creating one) **for validation purposes** are more than welcome.

What's next?

- **Disclaimer: the development shown here is not settled and it will take a bit of time until it becomes stable.**
- The adequate people should discuss its feasibility in the (near) future
- Our suggestion, should the usage of this definition become part of the standard, would be doing it **starting from ADQL 2.2**.
- Dave Morris will give more details on that in his talk.

Thanks for your attention

Questions, suggestions, any other feedback...

- Any help is welcome!
- Feel free to clone the repository, report issues and create your pull requests.
- You can drop us an email: contact me at `juaristi@uni-heidelberg.de` or Markus Demleitner at `msdemlei@ari.uni-heidelberg.de`.