Building a Parser for Domain-Specific Languages with Rust and PEGs

/sassman

@5422m4n

Definition of a context-free grammar

1. Token aka terminal symbols
    a. are the foundation of a language
    b. they are strings also known as keywords
    c. for example in rust we have "let", "if", "else"
2. Non-terminal symbols
    a. are syntax variables that describe a set of strings
    b. they combine other non-terminal and terminal symbols
    c. for example in rust an "expression" or "statement"
3. One non-terminal is the start-symbol
    a. the produced string set represents the grammar defined language
4. Production rules
    a. One non-terminal
    b. followed by → or sometimes "::="
    c. followed by a series of non-terminal and terminal symbols
    d. for example:

```
expr → expr op expr
op   → +
```

Parsing Expression Grammars (PEGs)

- The formalism was introduced by Bryan Ford in 2004 (MIT)
- Closely related to the family of top-down parsing languages introduced in the early 1970s
- A formal grammar that describes a formal language with a set of rules
- Feels more like RegEx based grammars with greatly improved readability
- Looks similar to Context Free Grammars (CFGs) but behaves different: ambiguous vs first match

src: https://bford.info/pub/lang/peg.pdf

Why Pest.rs?

- Pest is a library for writing plain-text parsers in Rust, no other codegen tools needed
- The grammar DSL (PEG) is easy to learn and maintain
- Fast enough for most cases*
- Excellent editor integration for Zed / VSCode / RustRover
- Bonus Tracks
    - Compiles to WASM and runs in the browser
    - Online grammar playground on the website

*) JSON Parsing: pest 150MB/s vs nom 366MB/s vs serde 472MB/s

src: https://pest.rs

Syntax of pest grammars

// excerpt of the important ones

- Terminal symbols
    - strings in double quotes
    - chars in single quotes
    - case-insensitive strings have a ^ prefix
- Non-terminal symbols
    - are expressed as rules like
        - rule_name = { "terminal" }
- Set of predefined Non-terminals
    - SPACE_SEPARATOR
    - NEWLINE
- Sequence with ~
- Ordered choice with |
- Rules that don't produce parsed token have a _ prefix

src: https://pest.rs/book/grammars/syntax.html

HTTP Language Example

// First Iteration, GET, URL params , Headers and no Body

```
1    GET http://foo.de:9000/path/b/c?a1=1a&2b=b2 HTTP/1.1
2    Content-Type: application/xml
3    Authorization: Basic 1234
```

src: https://github.com/sassman/edu-parsers-rs/blob/talk-iter-1/http-pest/get_example.http

HTTP Language Example

The Pest Grammar for the URL part

```
GET http://foo.de:9000/path/b/c?a1=1a&2b=b2 HTTP/1.1
```

```
wp       = _{ SPACE_SEPARATOR | "\t" }
alpha    = _{ 'a'..'z' | 'A'..'Z' }
digit    = _{ '0'..'9' }
alphanum = _{ (alpha | digit)+ }
symbols  = _{ "%20" }
urlchar  = _{ alphanum ~ symbols* }

method  = { "GET" | "POST" | "PUT" | "DELETE" }
version = { "HTTP/1.1" }

scheme      = { "http" | "https" }
host        = { alphanum ~ ("." ~ alphanum)* }
port        = { digit+ }
path        = { ("/" ~ urlchar*)+ }
param       = { param_name ~ "=" ~ param_value }
param_name  = { urlchar }
param_value = { urlchar }

url = { scheme ~ "://" ~ host ~ (":" ~ port)? ~ path? ~ ("?" ~ param ~ ("&" ~ param)*)? }
```

src: https://github.com/sassman/edu-parsers-rs/blob/talk-iter-1/http-pest/request.pest

HTTP Language Example

The Pest Grammar for the Headers

```
Content-Type: application/xml
Authorization: Basic 1234
```

```
header_name  = { (alphanum | "-")+ }
header_value = { (LETTER | NUMBER | SYMBOL | "/" | wp)* }
header       = { header_name ~ wp* ~ ":" ~ wp* ~ header_value }
headers      = { header ~ (NEWLINE ~ header)* }
```

src: https://github.com/sassman/edu-parsers-rs/blob/talk-iter-1/http-pest/request.pest

HTTP Language Example

The Pest Grammar for the Request

```
GET http://foo.de:9000/path/b/c?a1=1a&2b=b2 HTTP/1.1
Content-Type: application/xml
Authorization: Basic 1234
```

```
request = { SOI ~ method ~ wp ~ url ~ (wp ~ version)? ~ (NEWLINE ~ headers)? }
```

src: https://github.com/sassman/edu-parsers-rs/blob/talk-iter-1/http-pest/request.pest

HTTP Language Example

The Pest Grammar all together

```
wp       = _{ SPACE_SEPARATOR | "\t" }
alpha    = _{ 'a'..'z' | 'A'..'Z' }
digit    = _{ '0'..'9' }
alphanum = _{ (alpha | digit)+ }
symbols  = _{ "%20" }
urlchar  = _{ alphanum ~ symbols* }

method  = { "GET" | "POST" | "PUT" | "DELETE" }
version = { "HTTP/1.1" }

scheme      = { "http" | "https" }
host        = { alphanum ~ ("." ~ alphanum)* }
port        = { digit+ }
path        = { ("/" ~ urlchar*)+ }
param       = { param_name ~ "=" ~ param_value }
param_name  = { urlchar }
param_value = { urlchar }

url = { scheme ~ "://" ~ host ~ (":" ~ port)? ~ path? ~ ("?" ~ param ~ ("&" ~ param)*)? }

header_name  = { (alphanum | "-")+ }
header_value = { (LETTER | NUMBER | SYMBOL | "/" | wp)* }
header       = { header_name ~ wp* ~ ":" ~ wp* ~ header_value }
headers      = { header ~ (NEWLINE ~ header)* }

request = { SOI ~ method ~ wp ~ url ~ (wp ~ version)? ~ (NEWLINE ~ headers)? }

```

src: https://github.com/sassman/edu-parsers-rs/blob/talk-iter-1/http-pest/request.pest

HTTP Language Example

Parsing in Rust

```rust
use pest::{error::Error, iterators::Pairs, Parser};
use pest_derive::Parser;

#[derive(Parser)]
#[grammar = "../request.pest"]
pub struct RequestParser;

impl RequestParser {
    pub fn parse_request(input: &str) -> Result<Pairs<Rule>, Error<Rule>> {
        RequestParser::parse(Rule::request, input)
    }
}
```

src: https://github.com/sassman/edu-parsers-rs/blob/talk-iter-1/http-pest/src/lib.rs#L7-L15

HTTP Language Example

Parsing in Rust

```rust
use pest::{error::Error, iterators::Pairs, Parser};
use pest_derive::Parser;

#[derive(Parser)]
#[grammar = "../request.pest"]
pub struct RequestParser;

impl RequestParser {
    pub fn parse_request(input: &str) -> Result<Pairs<Rule>, Error<Rule>> {
        RequestParser::parse(Rule::request, input)
    }
}

#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_get_request_example_with_recursion() {
        let input = include_str!("../get_example.http");
        let pairs = RequestParser::parse_request(input).unwrap();
        let mut indentation = String::new();

        // ..
    }
}
```

src: https://github.com/sassman/edu-parsers-rs/blob/talk-iter-1/http-pest/src/lib.rs#L7-L41

HTTP Language Example

Parsing in Rust

```rust
#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_get_request_example_with_recursion() {
        let input = include_str!("../get_example.http");
        let pairs = RequestParser::parse_request(input).unwrap();
        let mut indentation = String::new();

        fn dive_in(pairs: Pairs<Rule>, indentation: &mut String) {
            for pair in pairs {
                let next_pair = pair.clone().into_inner();
                if next_pair.clone().count() > 0 {
                    println!("{indentation}- {:?}", pair.as_rule());
                    indentation.push_str("  ");
                    dive_in(next_pair, indentation);
                } else {
                    println!("{indentation}- {:?}: {:?}", pair.as_rule(), pair.as_str());
                }
            }
        }

        dive_in(pairs, &mut indentation);
    }
}
```

src: https://github.com/sassman/edu-parsers-rs/blob/talk-iter-1/http-pest/src/lib.rs#L7-L41

HTTP Language Example

Parsing in Rust

```rust
#[test]
fn test_get_request_example_with_recursion() {
    let input = include_str!("../get_example.http");
    let pairs = RequestParser::parse_request(input).unwrap();

    fn dive_in(pairs: Pairs<Rule>, indentation: &str) {
        for pair in pairs {
            let next_pair = pair.clone().into_inner();
            if next_pair.clone().count() > 0 {
                println!("{indentation}- {:?}", pair.as_rule());
                let mut i = indentation.to_owned();
                i.push_str("  ");
                dive_in(next_pair, &i);
            } else {
                println!("{indentation}- {:?}: {:?}", pair.as_rule(), pair.as_str());
            }
        }
    }

    dive_in(pairs, "");
}
```

cargo test

```
running 1 test
- request
  - method: "GET"
  - url
    - scheme: "http"
    - host: "foo.de"
    - port: "9000"
    - path: "/path/b/c"
    - param
      - param_name: "a1"
      - param_value: "1a"
    - param
      - param_name: "2b"
      - param_value: "b2"
  - version: "HTTP/1.1"
  - headers
    - header
      - header_name: "Content-Type"
      - header_value: "application/xml"
    - header
      - header_name: "Authorization"
      - header_value: "Basic 1234"
test tests::test_get_request_example_with_recursion ... ok
```

src: https://github.com/sassman/edu-parsers-rs/blob/talk-iter-1/http-pest/src/lib.rs#L7-L41

HTTP Language Example

Parsing in Rust

```rust
#[test]
fn test_get_request_example_with_rule_matching() {
    let input = include_str!("../get_example.http");
    let mut pairs = RequestParser::parse_request(input).unwrap();

    let request = pairs.next().unwrap();
    assert_eq!(request.as_rule(), Rule::request);
    println!(
        "{:?} with this content: {:?}",
        request.as_rule(),
        request.as_str()
    );

    let mut all_request_parts = request.into_inner();
    let http_method = all_request_parts.next().unwrap();
    assert_eq!(http_method.as_rule(), Rule::method);
    println!(
        "{:?} with this content: {:?}",
        http_method.as_rule(),
        http_method.as_str()
    );

    let url = all_request_parts.next().unwrap();
    assert_eq!(url.as_rule(), Rule::url);
    println!("{:?} with this content: {:?}", url.as_rule(), url.as_str());

    let scheme = url.into_inner().next().unwrap();
    assert_eq!(scheme.as_rule(), Rule::scheme);
    println!(
        "{:?} with this content: {:?}",
        scheme.as_rule(),
        scheme.as_str()
    );
}
```

src: https://github.com/sassman/edu-parsers-rs/blob/talk-iter-1/http-pest/src/lib.rs

HTTP Language Example

Parsing in Rust

```rust
#[test]
fn test_get_request_example_with_rule_matching() {
    let input = include_str!("../get_example.http");
    let mut pairs = RequestParser::parse_request(input).unwrap();

    let request = pairs.next().unwrap();
    assert_eq!(request.as_rule(), Rule::request);
    println!(
        "{:?} with this content: {:?}",
        request.as_rule(),
        request.as_str()
    );

    let mut all_request_parts = request.into_inner();
    let http_method = all_request_parts.next().unwrap();
    assert_eq!(http_method.as_rule(), Rule::method);
    println!(
        "{:?} with this content: {:?}",
        http_method.as_rule(),
        http_method.as_str()
    );

    let url = all_request_parts.next().unwrap();
    assert_eq!(url.as_rule(), Rule::url);
    println!("{:?} with this content: {:?}", url.as_rule(), url.as_str());

    let scheme = url.into_inner().next().unwrap();
    assert_eq!(scheme.as_rule(), Rule::scheme);
    println!(
        "{:?} with this content: {:?}",
        scheme.as_rule(),
        scheme.as_str()
    );
}
```

```
running 1 test
request with this content: "GET http://foo.de:9030/path/b/c?e1=1a&2b=b2 HTTP/1.1\nContent-Type: application/xml\nAuthorization: Basic 1234"
method with this content: "GET"
url with this content: "http://foo.de:9030/path/b/c?e1=1a&2b=b2"
schema with this content: "http"
test tests::test_get_request_example_with_rule_matching ... ok
```

src: https://github.com/sassman/edu-parsers-rs/blob/talk-iter-1/http-pest/src/lib.rs

HTTP Language Example

Error messages

```
#[test]
fn test_error_messages_1() {
    let input = r#"GET ftp://ftp.de:21/file"#;
    assert_eq!(
        RequestParser::parse_request(input)
            .err()
            .unwrap()
            .to_string(),
        r#" --> 1:5
  |
1 | GET ftp://ftp.de:21/file
  |     ^---
  |
  = expected scheme"#
    )
}
}

```

src: https://github.com/sassman/edu-parsers-rs/blob/talk-iter-1/http-pest/src/lib.rs

Limitations of PEGs: Ambiguous Grammar

Dangling else - grammar

```
1    Statement → if Condition then Statement |
2                if Condition then Statement else Statement |
3                ...
4    Condition → ...
```

```
1    if a then if b then s else s2
```

```
1    if a then begin if b then s end else s2
```

```
1    if a then begin if b then s else s2 end
```

src: https://en.wikipedia.org/wiki/Ambiguous_grammar

Dangling Else with Pest

```
stmt = {
    "if" ~ wp ~ cond ~ wp ~ "then" ~ wp ~ stmt
  | "if" ~ wp ~ cond ~ wp ~ "then" ~ wp ~ stmt ~ wp ~ "else" ~ wp ~ stmt
  | "pass"
}
cond = {
  'a'..'z' ~ wp ~ "==" ~ wp ~ 'a'..'z'
}
wp = _{ SPACE_SEPARATOR }
```

```
if a == b then if c == d then pass else pass
```

```
- stmt
  - cond: "a == b"
  - stmt
    - cond: "c == d"
    - stmt: "pass"
```

Dangling Else with Pest

```
stmt = {
    "if" ~ wp ~ cond ~ wp ~ "then" ~ wp ~ stmt ~ wp ~ "else" ~ wp ~ stmt
  | "if" ~ wp ~ cond ~ wp ~ "then" ~ wp ~ stmt
  | "pass"
}
cond = {
  'a'..'z' ~ wp ~ "==" ~ wp ~ 'a'..'z'
}
wp = _{ SPACE_SEPARATOR }
```

```
if a == b then if c == d then pass else pass
```

```
- stmt
  - cond: "a == b"
  - stmt
    - cond: "c == d"
    - stmt: "pass"
    - stmt: "pass"
```

Dangling Else with Pest (comparision)

```
stmt = {
    "if" ~ wp ~ cond ~ wp ~ "then" ~ wp ~ stmt ~ wp ~ "else" ~ wp ~ stmt
  | "if" ~ wp ~ cond ~ wp ~ "then" ~ wp ~ stmt
  | "pass"
}
```

vs

```
stmt = {
    "if" ~ wp ~ cond ~ wp ~ "then" ~ wp ~ stmt
  | "if" ~ wp ~ cond ~ wp ~ "then" ~ wp ~ stmt ~ wp ~ "else" ~ wp ~ stmt
  | "pass"
}
```

```
if a == b then if c == d then pass else pass
```

```
- stmt
  - cond: "a == b"
  - stmt
    - cond: "c == d"
    - stmt: "pass"
    - stmt: "pass"
```

vs

```
- stmt
  - cond: "a == b"
  - stmt
    - cond: "c == d"
    - stmt: "pass"
```

Theory Takeaway

- Write your grammar like you would top-down parse it
- Keep specific cases before the general cases

Conclusion

- pest.rs let's you maintain a higher level of abstraction of your DSL
- gives focus to your DSL and grammar
- comes with batteries included such as lexer and also tools for ast production see crates `from-pest` and `pest-ast`
- very fast interations on your DSL
- portability via WASM

Thank you

Q&A