Building Little Languages with Rust

Arcady Goldmints-Orlov

What is a “little language”?

- Specialized for a problem domain
- Limited power (not Turing-complete)
- Often used in or from other languages
- Not compiled to machine code

Examples

- Configuration files
- CSS
- Query languages (SQL is pretty big though!)
- Templating languages
- Not a hard line between “big” and “little”

Motivating Example: Transit Data

- Transit agencies publish realtime data about their buses in protocol buffer format.
- I scrape this data and archive it.
- It is a lot of data! 300 MB per day
  - So I store it in xzipped format
  - This compresses by a factor of 20x
- But I need a way to query subsets of this data
- How do? Little language!

PBQuery

- A simple language to query protocol buffer data
- Inspired by XPath for XML.
- Selects fields ('entity.vehicle')
- Filter expressions (“entity.vehicle[id = '5001']”)
- Expression operators include '=', 'in', '~' (regex match)
- `entity.vehicle[trip.route_id ~ 'Red.*']`

Aside: What is a protocol buffer?

- Efficient binary format for encoding data
- Message has fields (labeled by numbers)
  - Fields can be strings, numbers, or messages
  - Fields can be required, optional, or repeated
- Has a schema describing messages, and mapping names to numeric tags
- There is a compiler that compiles schema into C structs in a shared library.

Parts of my little language

- Parser
- Typechecker
- Evaluator

Parser

- A parser takes a string and returns a string and a thing, or maybe an error
- type ParseResult<'a, T> = Result<(T, &'a str), ParseError>
- fn parser(&'a str) → ParseResult
- Hand coded a recursive descent parser
- Parser returns a parse tree

Type Checker

- Need to translate field names into numbers
- And check that operations on fields are valid, given the defined type
- Takes advantage of protoc-c compiler
- Uses a rusty wrapper on top of FFI
- Returns an expression tree.

The evaluator

- The thing that actually “executes” the little language.
- Consumes the expression tree
- Built on top of an iterator API for streams of (generic) protocol buffers
- Surprisingly compact: Rust has powerful abstractions.

The power of abstractions

```
 fn query_helper<'a, 'b, F>(msg: &'a [u8], expr: Subexpr<'b>,
                                    callback: &mut F) -> usize
    where F : FnMut(PBMessage<'a>) -> bool
{
        let matches = PBIter::new(msg).filter(|p| p.tag == targettag)
                                                .filter(|p| filter.eval(p));
        for m in matches {
            if expr.path.len() == 1 {
                if !callback(m) { break; }
            } else {
                let subpath = Subexpr { path: &expr.path[1..],
                                        filters: &expr.filters[1..], };
                query_helper(m.contents, subpath, callback);
            }
        }
}
```

Learnings

- Rust is great for writing compilers
  - First big rust program was rustc
- Rust's FFI makes it easy to reuse C code
- Rust is great for building composable abstractions.

Conclusion

- Little languages are not hard!
- Go forth and build your own!
- My code is at https://github.com/crzwdjk/pbquery-rs
- Any questions?