Google Summer of Code 2018 Proposal

Format-Preserving YAML

Wisnu Adi Nurcahyo <wisn.adn@gmail.com>
Telkom University, Indonesia

Haskell

organization

Summary

Parse YAML with Haskell native code into a data structure and manipulate it natively in Haskell then transform it back into YAML.

Motivation

Why Haskell Organization?

Haskell is my favorite pure functional programming language. The main reason is to improve my Haskell skill. Therefore, Haskell organization is the best choice. Indonesia has a lot of programmers. However, just a few of them who use Haskell. In Lambda Jakarta (a functional programmer community), there are about 60 members who joining the Haskell channel. In Komunitas Haskell Indonesia (Indonesian Haskeller community) on Facebook, there are about 216 members who have joined. If we compare with the largest Indonesian programmer community, PHP Indonesia, which is about 149,782 members who have joined, only less than 2% of them who have ever know Haskell. Joining GSoC in Haskell organization is a really big start to me for improving my functional programming skill through Haskell and bring the Haskell community in Indonesia up. How is it even possible for me? I'm now a Bandung liaison in Lambda Indonesia, a functional programmer community in Indonesia, which have planned to build Lambda Bandung.

Why Working on Parser?

I have a big interest in the programming language. That makes me want to be a computer scientist. I think that starting from the parser is a good start. Since I have interest in functional programming, I want to be a computer scientist who does research in the programming language, especially with functional programming paradigm. Although now I'm just a sophomore, I have a big curiosity and a strong ambition to reach my dream. That's made read all unnecessary stuff in my undergraduate study. However, it is necessary for my dream. I read some stuff such as lambda calculus, category theory, parser generator, parser combinator, and so on.

Benefits to Community

1. We could manipulate YAML content via Haskell code without losing a single thing (even its comment).
2. Easily modify Haskell application config (YAML). Could be used by The Haskell Tool Stack and other tools.

Project Details

The main idea of format-preserving is to parse, manipulate, and transform it back to its original form. To put it simply, what we need is to create a lexical analyzer to store YAML tokens based on YAML Spec. Then, create a data structure for YAML structure so we could easily manipulate it later in the Haskell program. When the data structure ready, we parse the tokens and construct it using our own data structure. Create some functions to manipulate it and a special function to transform it back into YAML structure. Since the main purpose of this project is its correctness, we need to determine what "correctness proof framework" we will use.

Sometime Stack (The Haskell Tool Stack) ask us to add an extra dependency, manually. This is one of some problems that often occurred. How is this project actually help us? To understand better, there is an example below. Suppose that we use the latest Hakyll that need a `pandoc-citeproc-0.13` which is missing in the latest stable Stack LTS. Stack ask us to add the extra dependency to solve this problem. If Stack could add the extra dependency by itself, wouldn't it be nice?

Technical Demonstration

YAML Structure

```
extra-deps:
  - hakyll-4.11.0.0
```

Lexer YAML Tokens Representation

```
SCALAR("extra-deps") MAP(EOL SPACES(2) SEQUENCE([SCALAR("hakyll-4.11.0.0")])) EOF
```

Parsed YAML Into a Data Structure

```
YAML (MAP "extra-deps" (SEQUENCE [SCALAR "hakyll-4.11.0.0"]))
```

YAML Manipulation Demo

```
let yaml = YAML (MAP (SCALAR "extra-deps") (SEQUENCE [SCALAR "hakyll-4.11.0.0"]))
    yaml' = manipulate yaml (pushToSequence "extra-deps" "pandoc-citeproc-0.13")
in yaml'

-- > YAML (MAP (SCALAR "extra-deps") (SEQUENCE [SCALAR "hakyll-4.11.0.0", SCALAR
"pandoc-citeproc-0.13"]))
```

YAML Final Structure

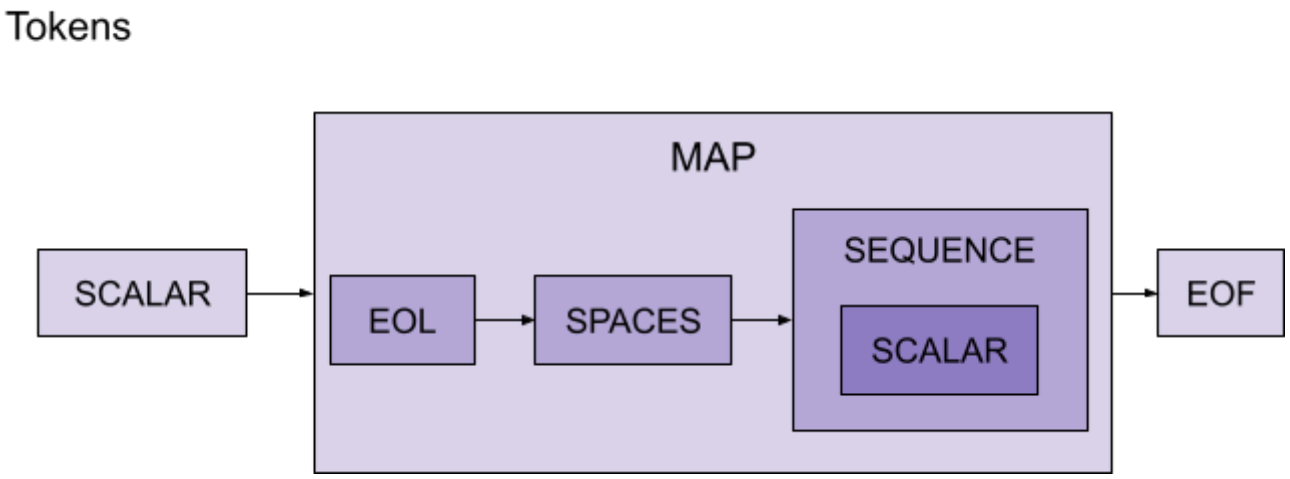
```
extra-deps:
  - hakyll-4.11.0.0
  - pandoc-citeproc-0.13
```

Conclusion

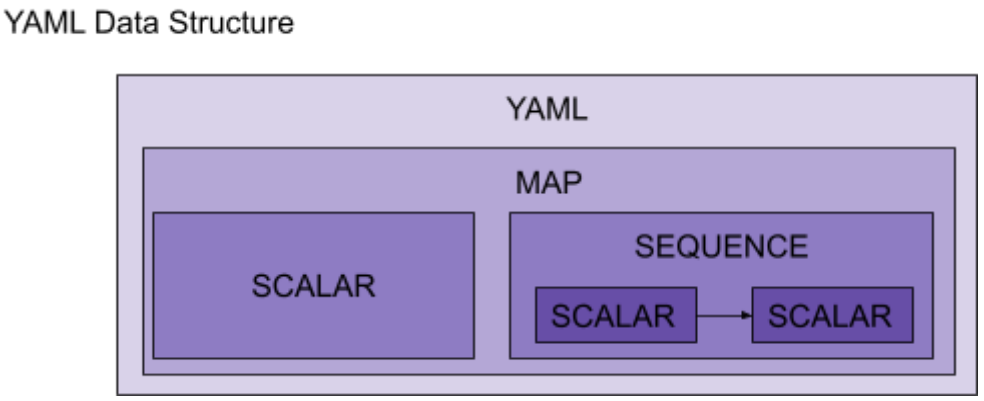
What we need to do now is just run `stack install` without manually editing the YAML file. There will be more advantage from this project.

How it Works

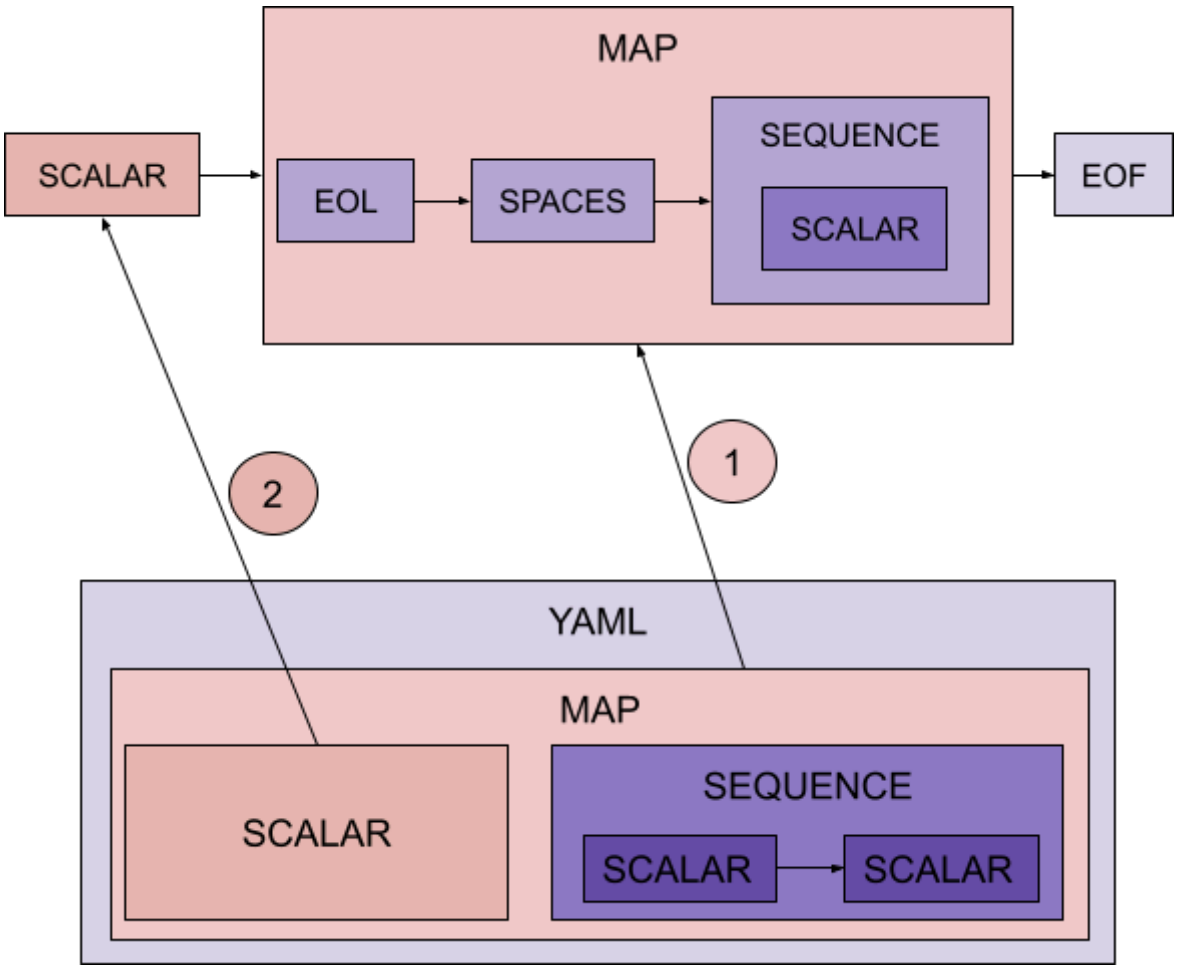
How could that the last item in the “extra-deps” sequence got an exactly the same indentation with its previous item? Since our library will transform it back to the same as its YAML style before, here is how the transformation process goes.



First, we retrieve the YAML tokens from the lexical analyzer. That’s how I represent the tokens. We keep the tokens in the safe place since we will need this later.

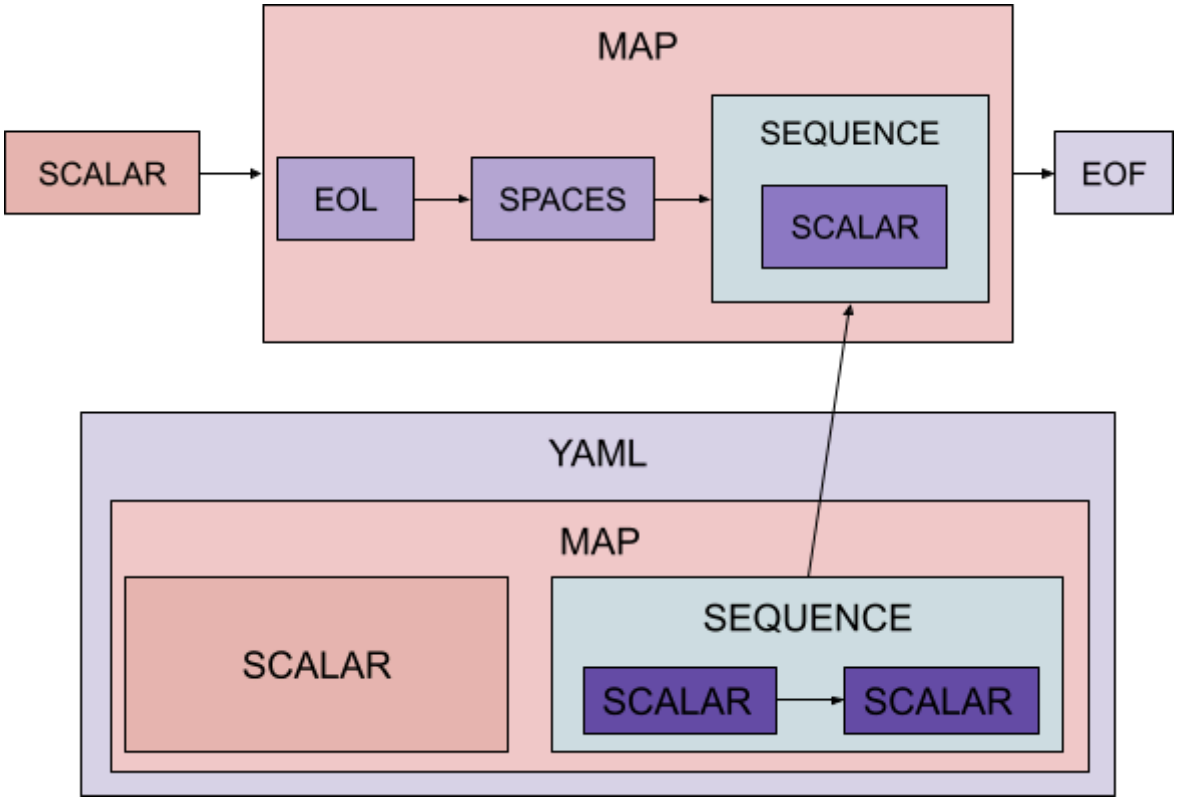


Second, we retrieve the YAML data structure from the parser. Suppose that the YAML above has been manipulated by the user. then, we want to transform the YAML data structure to its original YAML structure. The main idea is to transform the YAML data structure into a list of YAML tokens. We use the current list of tokens as the reference.

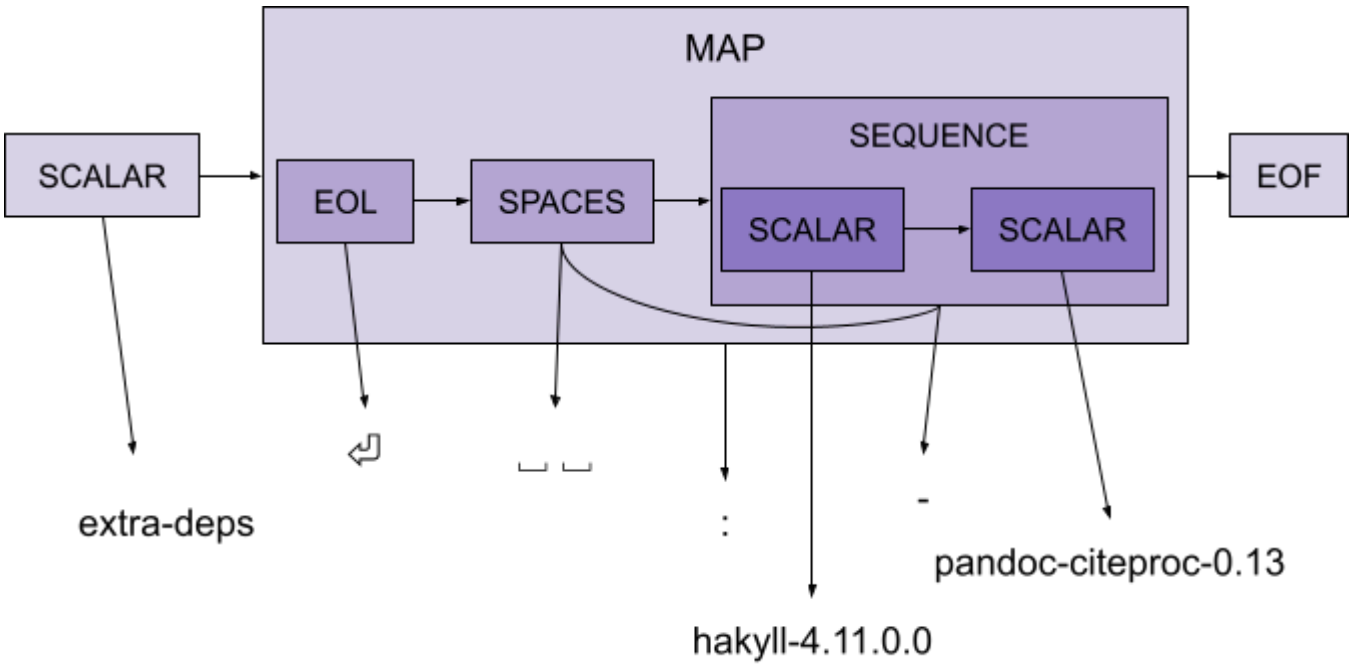


Here is our program checking for the similarity between the YAML data structure and the YAML tokens. For each YAML context, such as SCALAR (EOL, EOF, and SPACES are excluded), we keep it. Let it be a previousContext. Then, the current position of the "iteration" will be currentContext.

1. The program now walking from YAML to the MAP. We check the tokens. We found a MAP in the 2nd position of the tokens. Then, we take the first argument of the MAP in the YAML data structure (which is SCALAR) and compare it with the previousContext.
2. Then, we check the SCALAR (from the YAML data structure in the MAP) with the previousContext which is a SCALAR as well. Not surprisingly, it is matched. Therefore, we are in the right place.



We walk to the MAP tokens. There are EOL, SPACES, and SEQUENCE. Since EOL and SPACES are not a YAML context, we ignore them and then end up in the last tokens which are SEQUENCE. We duplicating the YAML tokens and replace the SEQUENCE with the new one. It is a SEQUENCE with two SCALAR items. Hence, even if it is not a SEQUENCE, suppose a SCALAR, we will replace it with SCALAR.



Here is our latest list of YAML tokens. Please remember that our past tokens are not replaced with this one since we are duplicating it first. A curve line in the tokens above representing as a “reference”. That is the last step. Transform latest tokens into its original form.

```
extra-deps:⏎
␣␣- hakyll-4.11.0.0
␣␣- pandoc-citeproc-0.13
```

Conclusion

The main idea is to check the similarity between the YAML tokens and the YAML data structure. We also need to use “reference” to determine how the new item will be written. Therefore, we get an exactly the same style and structure.

Timeline

April 23, 2018 - May 14, 2018

- Read and learn more about YAML Spec.
- Read and learn more about correctness proof.
- Learn and implement more simple lexical analyzer.
- Learn and implement more simple parser.
- Deciding communication method with the mentor.
- Deciding correctness proof “framework” with the mentor.
- Writing first GSoC blog post.

May 14, 2018 - June 15, 2018

- Work on the lexical analyzer.
- Designing YAML native data structure.
- First attempt to prove the code.
- Communicate with the mentor.

June 11 - 15, 2018

- Writing second GSoC blog post.
- Communicate with the mentor.
- Designing parser.

June 15, 2018 - July 9, 2018

- Work on the parser.

- Communicate with the mentor.
- Second attempt to prove the code.

July 9 - 13, 2018

- Writing third GSoC blog post.
- Communicate with the mentor.
- Designing YAML manipulation functions.

July 13, 2018 - August 6, 2018

- Work on YAML manipulation functions.
- Communicate with the mentor.
- Third attempt to prove the code.

August 6 - 14, 2018

- Writing forth GSoC blog post.
- Communicate with the mentor.
- Submit code.
- Makes sure all done.

Related Work

Here are some examples work that related to the proposal idea.

1. Writing a lexical analyzer and a parser generator for a simple arithmetic calculator in C++ (part 1, part 2, unfinished).
2. TBA. Haskell and/or parsing related stuff.

Code Sample

1. wisn/hapoid: Portable object translation file Linter for Bahasa Indonesia. My first Haskell project.
2. wisn/movee: An OMDb API consumer. Programming Club exercise at Telkom University.
3. wisn/hanum: An OpenStreetMap attributes linter with custom presets. Currently, abandoned.
4. wisn/language-po: A Format-Preserving Portable Object in Haskell.

Haskell related answers on StackOverflow could be found here. Contribution will be added later.

Personal Information

Name	:	Wisnu Adi Nurcahyo
Nickname	:	Wisnu
GitHub	:	https://github.com/wisn
OpenHub	:	https://www.openhub.net/accounts/wisn
StackOverflow	:	https://stackoverflow.com/users/6914498/wisnu-adi-nurcahyo
Email	:	wisn.adn@gmail.com, nurcahyo@protonmail.com
Medium	:	https://medium.com/@nurcahyo
Twitter	:	https://twitter.com/Wisn98
Occupation	:	Telkom University, Undergraduate Student
Major	:	Informatics Engineering, Sophomore
Timezone	:	UTC+7
Country	:	Indonesia
Work Location	:	Bandung, Indonesia