22100113 Seongbin Kim
Compiler Theory
April 12, 2024

Scanner Construction

Section 1: Scanner Introduction

The scanner of a compiler is responsible for spell checking of the input stream of strings. Given a stream of strings, its output is a set of tokens. These tokens are constructed using basic units called *lexmes* and are determined by regular expressions. In the following sections, the detailed explanation of tokens, regular expressions, and finite automata will be given. The paper ends with a user's manual for the submitted *SmallLexer* program.

Section 2: Tokens and the Symbol Table

Scanners, and compilers in general, use formal language to represent the specifications. This is to remove the ambiguity in the language/specification. A single unit should have one and only one meaning. These basic units are called tokens and are produced by the scanner. A token is determined by using regular expressions or finite automata, which are equivalent in power.

Through the code and/or other language specifications, each token is given a distinct meaning. This naturally causes the need of a storage mechanism that holds the translation from a token to its meaning. This storage tool is called the symbol table. It is usually implemented using a hash-like structure to guarantee fast access.

Section 3: Regular Expressions to Non-Deterministic Finite Automata (NFA)

One of the constraints of the given program, *SmallLexer*, was to not use pattern matching related library functions (e.g. *regex*). This naturally brought the necessity to implement finite automata. Finite automata are equivalent in power as regular expressions. Meaning they can represent all the other can represent and cannot represent what the other cannot represent.

To implement finite automata, one needs to implement deterministic finite automata (DFA) because implementing non-deterministic finite automata (NFA) causes overhead in complexity compared to DFA.

However, to convert regular expressions to DFA, it is convenient to go through the stage of converting to NFA.

Using the Thompson's Construction method, NFA can be constructed using regular expressions.

Section 4: Non-Deterministic Finite Automata to Deterministic Finite Automata (DFA)

After construction of NFA from regular expressions, an additional step of converting to DFA helps relieve complexity in the implementation. Using the Subset Construction method, DFA can be constructed using NFA.

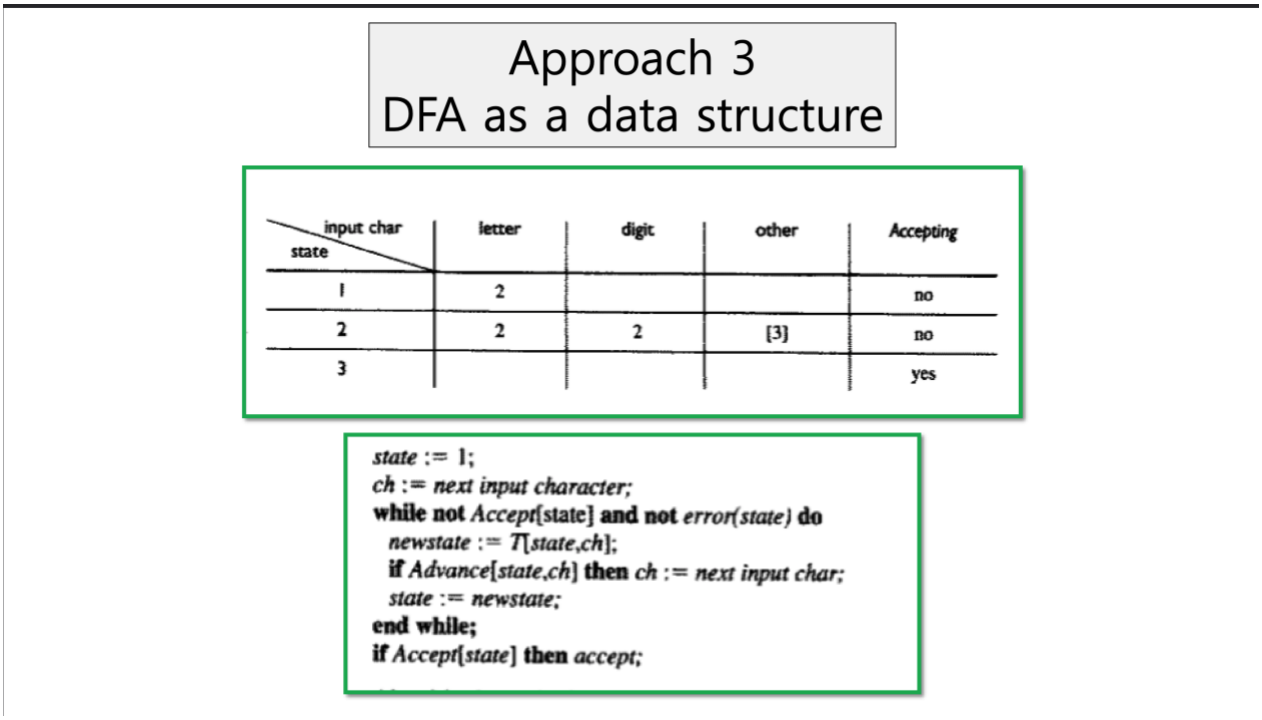
Section 5: Deterministic Finite Automata to Minimal Deterministic Finite Automata (MinDFA)

Constructing DFA from NFA via the Subset Construction method causes an explosion of states. Generally, the converted DFA has potentially 2^n states of the NFA. This requires a process of minimization or else going through the step of constructing the NFA would have been pointless.

By combining states that have the same input and transitions, DFA can be minimized.

Section 6: Minimal Deterministic Finite Automata to Source Code

To provide an abstract and consistent frame for DFA, the minimal DFA of each lexical unit is defined using the method described in one of the best-known compiler books (Louden):



Choosing an array-like data structure representation for DFA provides a template for DFA that accept all different kinds of lexical units.

Section 7: Build Process

The program, *SmallLexer*, is provided using Java code. As all Java source code is provided in the same directory, using the following command the program can be built.

```
$ javac *.java
```

To execute the program, the class with the main function and the input source code file should be given.

```
$ java SmallLexer <filename>
```

Conclusion

Implementing a scanner of a compiler requires substantial amount of theoretical ideas and identically sizeable skill in engineering. For example, the theories and programming of the conversion between regular expressions, NFA, and DFA. A complete implementation of the scanner from tokens to symbol table is provided in Java code.