Contents

1 Introduction

1.1 Background

The NJOY Nuclear Data Processing System [1] is a software system used for nuclear data management. In particular, it is used to convert Evaluated Nuclear Data Files (ENDF) [2] into different formats, as well as performing operations on the nuclear data.

NJOY is currently being used within the MACRO project [3] at the Division of Applied Nuclear Physics, at the Department of Physics and Astronomy at Uppsala University.

1.2 Problem Description

The NJOY input instructions are complex and hard to read compared to e.g. a high-level programming language. For example, algorithm 1 is a *short* and *simple* NJOY job which illustrates what the input instructions look like.

Algorithm 1 NJOY Test Problem 14

```
1  acer
2  20 21 0 31 32
3  1 0 1/
4  'proton + 7-n-14 apt la150 njoy99 mcnpx'/
5  725 0./
6  /
7  /
8  acer
9  0 31 33 34 35
10 7 1 2/
11 'proton + 7-n-14 apt la150 njoy99 mcnpx'/
12 viewr
13 33 36/
14 stop
```

Without consulting the documentation, one might guess that line 4 and 11 are some kind of descriptive titles, which is correct. But it is not obvious that line 2 denotes input and output files (each number indicates a specific file) which the `acer` module will operate on. It is also hard to deduce that the first number on line 5 denotes the material to be processed, and that the second number denotes the desired temperature in kelvin.

The input instructions can be annotated with descriptive comments, but even then, working with a large and complex job easily becomes a daunting task.

1.3 Objective

The NJOY input instructions is not an optimal input format. Therefore, the scope of this work has been to design and implement a more user friendly, and readable input format.

In order to make the new input format useable with NJOY, it has to be translated into the original NJOY input instructions. As such, the scope of this work also included developing an accompanying translator to the new input format.

2 Methodology

2.1 Introduction

As stated in [4], a translator (*compiler*) is a program that can read a program in one language and translate it into an equivalent program in another language. In the following subsections, principles and techniques for constructing a translator presented in [4], is described.

In [4], the translation process is described as a sequence of phases. Each phase inspects and transforms a representation of the source program to another. Phases such as lexical analysis, syntax analysis, and semantic analysis has been used throughout this work.

2.2 Understanding the NJOY Input Instructions

The NJOY input instructions [5] had to be understood in order to design the new input format. Each module of the NJOY software system was analyzed separately such that a general structure and common language features could be extracted and used for further analysis.

2.3 Designing the New NJOY Input Format

The design of the new NJOY Input Format (NIF) was based on the analysis of the input instructions.

2.3.1 Syntax Definition

The syntax definition of NIF was specified in a notation called context-free grammar [6]. A context-free grammar is a convenient, and natural method of specifying the syntax of a programming language.

For instance, the assignment (*declaration*) of an identifier can have the form

```
material = 725
```

which can be expressed in a context-free grammar as the production

```
expression ::= l_value "=" r_value
```

where `l_value` and `r_value` are other productions expressing the structure of the left and right hand side of the assignment, respectively.

2.4 Building the Translator

The translator, which is supposed to translate NIF into NJOY input instructions, was partly constructed using a lexical-analyzer generator [7] and a parser generator [8]. The translator was written in the Python programming language [9], in an Un*x environment.

2.4.1 Lexical Analysis

Lexical analysis is the process of dividing the source program (e.g. NIF) into sequences of characters, called tokens [10]. Each token describes a group of

characters in the source program as an abstract type.

For example, the identifier `material`, the assignment character, `=`, and the integer `725` could be represented as tokens of the form

`<IDENTIFIER, material>`,

`<ASSIGNMENT, =>`, and

`<INTEGER, 725>`

PLY Lex [11] was used to generate a lexical analyzer (lexer) for NIF. The method of identifying the NIF tokens was implemented by using the notation of regular expressions [12] in PLY Lex.

2.4.2 Syntax Analysis

Syntax analysis is the process of creating a tree-like representation, an abstract syntax tree, composed of the tokens generated by the lexical analyzer [13]. The syntax tree is used to describe the grammatical structure of the source program.

PLY Yacc [11] was used to generate a syntax analyzer (parser) for the grammar definition of NIF. The method of building the syntax tree was implemented by using the facilities provided by the PLY tools.

2.4.3 Semantic Analysis

Semantic analysis is the process of checking the syntax tree for errors that have to do with the *meaning* of the program [14]. Type checking is a part

of the semantic analysis where the translator checks that each operator has valid operands.

For example, the variable `hk` in card 3, module `acer` must be declared as a string and must not exceed 70 characters in length [5]. The translator should report an error if this is violated; such as when `hk` has been declared as an integer, or when the string contains more than 70 characters.

The semantic analyzer was implemented using the information retrieved during the analysis of the NJOY input instructions.

2.5 Testing

Testing was carried out continuously during the design and implementation of NIF and the translator. The NJOY test problems [1] was used to test the functionality of both NIF and the translator.

The NJOY test problems was manually translated into equivalent NIF programs, which were run through the translator. The resulting output was compared with the expected output, to verify that the translator was working appropriately.

The Python unit testing framework [9] was utilized to set up the testing environment.

3 Results

3.1 NJOY Input Format (NIF)

Algorithm 2 is a context-free grammar definition which describes the NJOY Input Format (NIF).

Algorithm 2 NJOY Input Format (NIF) Grammar Definition

```
program         ::= module_list
module_list     ::= module module_list
                  | empty
module          ::= MODULE "{" card_list "}"
card_list       ::= card card_list
                  | empty
card            ::= CARD "{" statement_list "}"
statement_list  ::= statement statement_list
                  | empty
statement       ::= expression ";"
expression      ::= l_value "=" r_value
l_value         ::= array
                  | identifier
r_value         ::= number
                  | string
array           ::= IDENTIFIER "[" INTEGER "]"
identifier      ::= IDENTIFIER
number          ::= FLOAT
                  | INTEGER
string          ::= STRING
```

The start symbol is `program`. The capitalized terminals, such as `MODULE` and `CARD`, are token classes specified by the lexer (see section 3.2.1). Special symbols are denoted within double quotes. `empty` denotes the empty string.

3.2 NJOY Input Format Translator (`nifty`)

3.2.1 Lexical Analysis

XXX

- Used a lexical analyzer to generate tokens and detect errors. PLY Lex.
- Wrote regular expressions to recognize tokens.

TOKEN	DESCRIPTION	EXAMPLE
`MODULE`		`acer`, `reconr`
`CARD`		`card_1`, `card_2`, `card_8a`
`IDENTIFIER`		`mat`, `nin`, `nout`

Table 1: NIF Tokens

3.2.2 Syntactic Analysis

XXX

- Used a parser generator such to detect and report syntax errors. PLY YACC.
- Constructed an syntax tree to represent the structure of NIF.

3.2.3 Semantic Analysis

XXX

- Enforced a type system.
  - OK: `nendf = 20;`, not OK: `nendf = 999;`
- Determine if the program is semantically correct (i.e. find errors that have to do with the meaning of the program, and not the syntax).
- Detected and reported errors. E.g. input file was not in the range [20,99].

3.2.4 Code Generation

XXX

- Process of translating the *source* language (NIF) into the *target* language (NJOY input instructions).
- Massage the syntax tree into NJOY input instructions.
- Flattened the tree structure.
  - Control flow statements? E.g. for-loop idioms?
  - Traversed the syntax tree. Constructed NJOY input instructions by visiting the nodes (DFS algorithm).
- Detected and reported errors.

- PLY
  - Lexer
    - Reserved words and identifiers. Hard coded.
    - Comments. Multi-line, single line.
    - Data-types: numbers and strings.
  - Parser
- Description of Abstract Syntax Tree (AST)
  - List structure
  - Node structure
- Organizer
- Analyzer
- Translator
  - Translation process of syntax tree. Flatten the tree structure.
  - Code generation: generating the target language instructions.
  - Describe structure which is the result of the translator.
- Emitter
  - Translator result is converted to a string.

4 Discussion

- Testing not that rigorous. NJOY is a large and complex program, there's a lot of scenarios (e.g. input instructions) that hasn't been tested.
- Modules still needs to be given in the correct, sequential order. The translator cannot guess the users' intention of the job. Needs to be told what to do. Just a translator.
- The analysis of the input format revealed some common features for each module, and a general structure of the input instructions was evident. Because of this, a solution based on Lex and Yacc was chosen. Easy and fast.
- Influenced by the C programming language.
- Hard coded words and identifiers. Explain why.
- Efficiency?
- Answer the "Why?" questions.
- Significant findings?

5 Conclusion

- Improvement on the existing situation?
  - Readable?
- Challenges?
  - Constructing a decent input format.
  - The physics (even if it's not within the scope). Documentation is full of it, kind of.
- Usable?
  - Production: not advisable. Grammar not verified. "Toy" translator.

5.1 Future Work

- Possible improvements?
  - Recognize more data-types. E.g. materials, temperatures, etc.
- Complete context-free grammar?
  - Expand grammar. More tokens (`TEMPERATURE`, `MATERIAL`, etc).
- GUI editor?
- Efficient implementation, e.g. C?

References

[1] R. E. MacFarlane, "NJOY99 – code system for producing pointwise and multigroup neutron and photon cross-sections from ENDF/B data", Los Alamos Nat. Laboratory, Los Alamos, NM, Rep. RSIC PSR-480, 2000.

[2] M. B. Chadwick *et al.*, "ENDF/B-VII.0: Next Generation Evaluated Nuclear Data Library for Nuclear Science and Technology," *Nuclear Data Sheets*, vol. 107, no. 12, pp. 2931-3060, Dec. 2006.

[3] C. Gustavsson *et al.*, "Massive Computation Methodology for Reactor Operation (MACRO)," in *European Nuclear Conference*, 2010 © European Nuclear Society. ISBN: 978-92-95064-09-6

[4] A. V. Aho *et al.*, *Compilers: Principles, Techniques, & Tools*, Second Edition. Boston: Pearson Educ., 2007.

[5] A. C. Kahler and R. E. MacFarlane. (2010, Mar. 31). *User Input for NJOY99, updated through version 364* [Online]. Available: `http://t2.lanl.gov/codes/njoy99/Userinp.364`

[6] A. V. Aho *et al.*, "Syntax Analysis" in *Compilers: Principles, Techniques, & Tools*, Second Edition. Boston: Pearson Educ., 2007, ch. 4, sec. 4.2, pp. 197-206.

[7] A. V. Aho *et al.*, "Lexical Analysis" in *Compilers: Principles, Techniques, & Tools*, Second Edition. Boston: Pearson Educ., 2007, ch. 3, sec. 3.5, pp. 140-146.

[8] A. V. Aho *et al.*, "Syntax Analysis" in *Compilers: Principles, Techniques, & Tools*, Second Edition. Boston: Pearson Educ., 2007, ch. 4, sec. 4.9, pp. 287-297.

[9] F. L. Drake, Jr., *et al.* (2011, Apr. 16) *Python v2.7.1 documentation* [Online]. Available: `http://docs.python.org/`

[10] A. V. Aho *et al.*, "Lexical Analysis" in *Compilers: Principles, Techniques, & Tools*, Second Edition. Boston: Pearson Educ., 2007, ch. 3, sec. 3.1, pp. 109-114.

[11] D. M. Beazley. (2011, Apr. 16). *PLY (Python Lex-Yacc)* [Online]. Available: `http://www.dabeaz.com/ply/ply.html`

[12] A. V. Aho *et al.*, "Lexical Analysis" in *Compilers: Principles, Techniques, & Tools*, Second Edition. Boston: Pearson Educ., 2007, ch. 3, sec. 3.3, pp. 116-124.

[13] A. V. Aho *et al.*, "Syntax Analysis" in *Compilers: Principles, Techniques, & Tools*, Second Edition. Boston: Pearson Educ., 2007, ch. 4, sec. 4.1, pp. 192-196.

[14] A. V. Aho *et al.*, "Introduction" in *Compilers: Principles, Techniques, & Tools*, Second Edition. Boston: Pearson Educ., 2007, ch. 1, sec. 1.2, pp. 8-9.