Contents

1 Introduction

1.1 Background

The NJOY Nuclear Data Processing System [1] is a software package 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. The listing below 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. The scope of this work has been to design and implement a more user friendly, and readable input format.

The new input format should be designed with algorithmic input file generation in mind, such that input files can be generated by other programs.

The design should be based on some commonly known existing format that is fitting to the task, such that users quickly can embrace the style and nature of the new input format.

2 Methodology

2.1 Understanding the NJOY Input Instructions

The NJOY input instructions [4] had to be understood in order to design the new input format. Each module of the NJOY software package was analyzed separately such that a general structure and common language features could be extracted and used for further analysis.

2.2 Designing the New NJOY Input Format

The new NJOY Input Format (NIF) was based on the analysis of the input instructions. The syntax definition, which is a description of the proper form of programs, of NIF was specified in a notation called context-free grammar [5].

2.3 Building the Translator

The translator, which is supposed to translate NIF into NJOY input instructions, was written in the Python programming language [6] in an UN*X environment.

The PLY [7] tools, PLY Lex and PLY Yacc, was used to generate a lexical analyzer (lexer) and a syntactical analyzer (parser) for the NIF grammar.

2.4 Testing

Testing was carried out continuously during all phases of the design and implementation. The NJOY test problems was used to test both NIF and the translator.

- Python Unit Testing?
- Test plan?
- Used NJOY test problems as examples
  - Compared output from translator with expected output.

3 Results

3.1 NJOY Input Format (`NIF`)

- The Grammar Definition?

Algorithm 2 NJOY Input Format (NIF) Grammar Definition

```
program         ::= module_list
module_list     ::= module module_list
                  | EMPTY
module          ::= MODULE LEFT_BRACE card_list RIGHT_BRACE
card_list       ::= card card_list
                  | EMPTY
card            ::= CARD LEFT_BRACE statement_list RIGHT_BRACE
statement_list  ::= statement statement_list
                  | EMPTY
statement       ::= expression SEMICOLON
expression      ::= assignment
assignment      ::= IDENTIFIER ASSIGNMENT r_value
r_value         ::= FLOAT
                  | INTEGER
                  | STRING
```

Keywords and special symbols are capitalized, `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.

3.2.2 Syntactic Analysis

XXX

- Used a parser generator such to detect and report syntax errors. PLY YACC.
- Constructed an AST 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 No intermediate code necessary.

- Process of translating the *source* language (NIF) into the *target* language (NJOY input instructions).
- Massage the AST into NJOY input instructions.
- Flattened the tree structure.
  - Control flow statements? E.g. for-loop idioms?
  - Traversed the AST. Constructed NJOY input instructions by visiting the nodes (DFS algorithm).
- Detected and reported errors.
- PLY
  - Lexer
    * Reserved words and identifiers. Hardcoded.
    * Comments. Multi-line, single line.
    * Datatypes: numbers and strings.
  - Parser
- Description of Abstract Syntax Tree (AST)
  - List structure
  - Node structure
- Organizer

- Analyzer
- Translator
  - Translation process of AST. 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.
- Hardcoded 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 adviceable. Grammar not verified. "Toy" translator.

5.1 Future Work

- Possible improvements?
  - Recognize more datatypes. 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. 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`

[5] A. V. Aho *et al.*, "A Simple Syntax-Directed Translator" in *Compilers: Principles, Techniques, & Tools*, Second Edition. Boston: Pearson Educ., 2006, ch. 2, sec. 2.1, pp. 40-47.

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

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