Contents

[XXX BLANK]

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 also 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.

The NJOY program is composed by a set of modules. The modules are independent programs which performs a specific task. The modules uses input and output files, specified by the input instructions, to communicate with each other. Data flows from one module to another through a chain of files. As such, the NJOY input instructions is a sequential listing of instructions to be carried out by the NJOY program, and its execution order is of outermost importance.

The following grammar[1] [5], describe a general outline of what the NJOY input instructions look like[2]:

---

[1]XXX: describe grammar?

[2]Note that this grammar is by no means intended to be a complete, nor correct, grammar definition of the NJOY input instructions. Its purpose is solely illustrational.

Algorithm 2 General structure of the NJOY input instructions

```
program        ::= module_list
module_list    ::= module NEWLINE module_list
                 | STOP
module         ::= MODULE_NAME NEWLINE card_list
card_list      ::= card NEWLINE card_list
                 | card NEWLINE
card           ::= variable_list
variable_list  ::= variable SPACE variable_list opt_slash
                 | variable optional_slash
                 | SLASH
variable       ::= VARIABLE
opt_slash      ::= SLASH
                 | EMPTY
```

The start symbol is `program`. Keywords and special symbols are capitalized. `EMPTY` denotes the empty string. `STOP` denotes the stop instruction. `MODULE_NAME` indicates a module name, such as `acer` or `reconr`. `NEWLINE`, `SLASH`, and `SPACE` denotes a newline, a slash and a blank space, respectively. `VARIABLE` indicates a variable declaration, such as numerical quantities or strings.

As indicated by the grammar, each declaration of a module and card is always suffixed by a newline. A variable list is separated by spaces (or by commas, but it is less frequently used). Comments are available and prefixed by `$`.

XXX input sample file here.

- XXX: Repeatable cards.

- XXX: Similarities to control flow statements; some variables declare which cards are present.

- XXX: Terminology: input deck, decks of cards, ...

2.2 Designing the New NJOY Input Format

XXX Define a grammar for the new NJOY input format.

- Influenced by the C programming language.

2.3 Writing the Grammar

XXX

- Static scope? Only top-level declarations?

- Block structure. e.g. `module {` *variable-declarations* `}`

2.4 Building the Translator

XXX Python , PLY (compiler compiler) , UN*X environment.

2.4.1 Lexical Analysis

XXX

- Used a lexical analyzer to generate tokens and detect errors. PLY Lex.

- Wrote regular expressions to recognize tokens.

2.4.2 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.

2.4.3 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].

2.4.4 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.

2.5 Testing

XXX Continuously testing during all phases.

- Unit Testing?

- Used NJOY test problems as examples

  - Compared output from translator with expected output.

3 Results

3.1 NJOY Input Format (`NIF`)

- Describe design?

  – Strings and numbers

- The Grammar?

- Example. NJOY input instructions vs NIF.

- 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.

Algorithm 3 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`)

- 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

- Analyzer

- Mangler / Massager

- Translator

  - Translation process of AST. Flatten the tree structure.

  - Code generation: generating the target language instructions.

- Emitter

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.

- 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**] Mcfarlane, R.E., "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**] XXX [Online]. Available: `http://www.nndc.bnl.gov/endf/`

[**3**] XXX (Use reference to MACRO article instead of this URI. Availability of article?) [Online]. Available: `http://www.fysast.uu.se/tk/en/content/macro-project`

[**4**] XXX "User Input for NJOY99, updated through version 364" [Online]. Available: `http://t2.lanl.gov/codes/njoy99/Userinp.364`

[**5**] XXX Dragon book, grammar chapter.