**COL703: Logic for Computer Science**
I semester 2021-22

# Assignment: Parser for a Formal Language for Arguments in Sentential Logic (FLASL)

**Cleaning up Natural language arguments.** Consider the following argument presented in English.

> *If prices rise, then the poor and the salaried class will be unhappy. If taxes are increased then the businessmen will be unhappy. If the poor and the salaried class or the businessmen are unhappy, the Government will not be re-elected. Inflation will rise if Government expenditure exceeds its revenue. Government expenditure will exceed its revenue unless taxes are increased or the Government resorts to deficit financing or takes a loan from the IMF to cover the deficit. If the Government resorts to deficit financing then inflation will rise. If inflation rises, the prices will also rise. The Government will get reelected. Therefore the Government will take a loan from the IMF.*

It is necessary to be able to parse this argument and transform it into the `Argument` abstract syntax tree.

The above argument has several shortcomings, ambiguities and other features which require human intelligence to parse and which do not permit easy automatic parsing and translation into the AST. The argument may be cleaned up using various keywords and punctuation to present it in a form that can be parsed. A cleaner version that is suitable for algorithmic parsing may be the following form, which clearly delineates atomic propositions, propositional connectives and uses punctuation symbols to disambiguate and delimit hypotheses from each other and from the conclusion.

**Keywords.** NOT, AND, OR, IF, THEN, ELSE, IFF, THEREFORE.

**Punctuation symbols.** (, ), .

**AST data-type.**

```
exception Atom_exception
datatype Prop =
     ATOM of string     |
     NOT of Prop        |
     AND of Prop * Prop |
     OR of Prop * Prop  |
     COND of Prop * Prop |
     BIC of Prop * Prop |
     ITE of Prop * Prop * Prop
datatype Argument = HENCE of Prop list * Prop
```

> IF "prices rise" THEN "the poor and the salaried class will be unhappy". IF "taxes are increased" THEN "the businessmen will be unhappy". IF ("the poor and the salaried class are unhappy" OR "the businessmen are unhappy") THEN NOT "the Government will be re-elected". "Inflation rises" IF "Government expenditure exceeds its revenue". IF NOT ("taxes are increased" OR "the Government resorts to deficit financing" OR "the Government takes a loan") THEN "Government expenditure will exceed its revenue". IF "the Government resorts to deficit financing" THEN "inflation rises". IF "inflation rises" THEN "prices rise". "the Government will be re-elected". THEREFORE "the Government takes a loan".

In the above form we have used the following syntactic conventions which define the language FLASL: Formal Language of Arguments in Sentential Logic.

1. All atomic propositions are enclosed in double quotes. We will assume the double quote character never occurs inside an atomic proposition. Atomic propositions are genuinely atomic and hence cannot be nested.
2. Each atomic proposition consists of a sequence of printable words or numerals separated by white-space.
3. No keyword of FLASL can occur as a word in an atomic proposition.
4. Each word may consist of any non-empty sequence of visibly printable ASCII characters (i.e. characters in the range 33-126) except for the punctuation symbols defined above.
5. **white-space equivalence.** Contiguous sequences of white-space characters (`<SPACE>`, `<TAB>`, `<LF>`, `<CR>` etc.) are equivalent to a single `<SPACE>` character. All leading and trailing white-space characters should be ignored.
   (a) Therefore a token for an atomic proposition should be compressed to a string of words, with no leading white-space, no trailing white-space and exactly one `<SPACE>` between words in the string.
   (b) Any contiguous occurrences of tokens are separated by white-space.
6. All words occurring in UPPER-CASE are assumed to be Keywords of FLASL. These words may be used to translate propositions into an element of the dataype `Prop` (defined above).
7. Notice the different uses of the keyword "IF" in the argument above.
8. Assume that the keywords of FLASL do not occur (in UPPER-CASE) inside atoms.
9. Notice the use of the left and right parentheses symbols to disambiguate and aid in the process of parsing.
10. The full-stop symbol "." terminates a (compound) proposition (a hypothesis or the conclusion).
11. THEREFORE is the keyword used to separate the list of hypotheses from the conclusion. The list of hypotheses could be empty but there can be exactly one conclusion in an argument.
12. The correspondence between the constructs of FLASL and the datatypes defined below is as follows:

| FLASL | AST |
|---|---|
| "s" | `ATOM s` |
| NOT $\phi$ | `NOT (`$\phi$`)` |
| $\phi$ AND $\psi$ | `AND (`$\phi$`, `$\psi$`)` |
| $\phi$ OR $\psi$ | `OR (`$\phi$`, `$\psi$`)` |
| IF $\phi$ THEN $\psi$ | `COND (`$\phi$`, `$\psi$`)` |
| $\psi$ IF $\phi$ | `COND (`$\phi$`, `$\psi$`)` |
| $\phi$ IFF $\psi$ | `BIC (`$\phi$`, `$\psi$`)` |
| IF $\phi$ THEN $\psi$ ELSE $\chi$ | `ITE (`$\phi$`, `$\psi$`, `$\chi$`)` |
| $\Phi$ THEREFORE $\psi$ | `HENCE (`$\Phi$`, `$\psi$`)` |

13. Precedence of operators is as defined in the slide titled "Associativity and Precedence" in the Hyper-notes.
14. No particular precedence has been defined for the IF ... THEN ... ELSE. Hence there is a dangling-else problem. All ambiguities of parsing should be handled by suitable parenthesization in FLASL itself.

**Problem statement.** Given an argument written in FLASL as a text file (`arg-inp.flasl`) to transform the argument into an element of the AST defined by the above datatype declarations to obtain an output file (`arg.sml`). Proceed as follows:

**input files.** `arg-inp.flasl`, `ast2flasl.sml`

**output files.** `arg.sml`, `arg-out.flasl`

**submission file.** `flasl2ast.zip`

1. This assignment has to be implemented in SML and may use the associated tools such as ML-Lex, ML-Yacc, ML-Antlr etc.
2. Design a regular expression or a right-linear regular grammar for scanning the text of an argument written in FLASL. You may use ML-Lex to tokenize the argument. Raise exception `ScanError` with line number and character number (references to the input file) as parameters if there is an error.
3. Define an EBNF specification for the language FLASL. The EBNF should specify a context-free grammar which is suitable for either top-down parsing or bottom-up parsing. Depending on the kind of grammar (LL(1) or LALR(1)) you may choose to use either Ml-Antlr or ML-Yacc to parse and generate the AST. You may also choose to write a full parser of your own. But the EBNF specification should appear as a comment at the top of your parser file. Raise an exception `ParseError` with line number and character number (references to the input file) as parameters if there is an error.
4. Write an SML function `ast2flasl: Argument -> string` in a file called `ast2flasl.sml` which takes the file `arg.sml` as input and produces the file `arg-out.flasl`. which reconstructs the argument back from the AST to text as a FLASL file `arg-out.flasl`.
5. All the files of the scanner-parser software along with `ast2flasl.sml` should be zipped into a package called `flasl2ast.zip` and submitted on moodle.

**Self-validation of your software for syntactically well-formed arguments in FLASL**

1. The scanner and parser together represent a function `flasl2ast` which transforms a syntactically well-formed argument in FLASL into AST.
2. The function `ast2flasl: Argument -> string` can be invoked only if there are no errors during scanning and parsing.
3. If `arg-inp.flasl` is syntactically well-formed then the two text-files `arg-inp.flasl` and `arg-out.flasl` should be white-space equivalent.
4. But more importantly self-validation can be done more easily as follows for syntactically well-formed FLASL files i.e. starting from `arg.sml` by cycling through the following diagram identical copies of `arg.sml` and `arg-out.flasl` will be obtained.