An Oz Subset

This document describes a subset of Oz \cite{VH04} that is designed to be slightly larger than the declarative kernel language of chapter 2 (of \cite{VH04}). The subset includes several syntactic sugars, but not so many that manipulation becomes unwieldy.

The purpose of this subset, and the programs in the following directory:
\url{http://www.eecs.ucf.edu/~leavens/COP4020/reducer}
is to show how to define various manipulations of Oz programs. In particular, we provide definitions of Free and Bound Variable identifiers and desugaring into the declarative kernel language. Also provided is a definitional interpreter. All of these programs are written in Oz (although using a larger portion of the language).

In what follows, I provide the lexical (Section 1) and context-free (Section 2 on page 3) grammars for what the tools use in lexical analysis and parsing. I also define the abstract syntax trees (Section 3 on page 3) that are used in other tools (Section 4 on page 7).

Notational Conventions

Throughout this document we use the following conventions for describing grammars. The square brackets `[' and ']' surround optional parts of a production. Thus

(A) ::= example [ ⟨B⟩ ] ⟨C⟩

means the same thing as

(A) ::= example ⟨C⟩
| example ⟨B⟩ ⟨C⟩

Furthermore, the notation “...” following an optional construct means zero or more repetitions of the enclosed sentential form. Thus

(A) ::= example [ ⟨B⟩ ] ... ⟨C⟩

means the same thing as

(A) ::= example ⟨Bs⟩ ⟨C⟩
⟨Bs⟩ ::= ⟨B⟩ ⟨Bs⟩
| ⟨Empty⟩
⟨Empty⟩ ::=

Single quotes (' and ') are used to indicate when notations such as ‘[ ’ and ‘ ]’ are to be taken literally as terminal symbols, as opposed to their default interpretation as meta-level grammatical symbols.

1 Microsyntax for An Oz Subset

The lexical analyzer in OzLexer.oz, which uses the code in LexerTools.oz, does lexical analysis for the Oz subset. The lexer recognizes all the keywords (really reserved words) of the entire Oz language, as well as all the special keyword tokens. However, it currently does not handle the Oz \texttt{\textbackslash insert} directive.

The microsyntax recognized by the lexer is given in Figure 1 on the following page. This grammar which is to be understood lexically, that is, unlike a normal grammar, no spaces or comments may appear between the characters of a token. Thus, in particular, in a ⟨label⟩, there can be no space or comment before the left parenthesis.

The microsyntax and the lexer do not not consider \texttt{unit} to be a keyword, unlike the Oz documentation, but merely an ⟨Atom⟩.
Figure 1: The lexical grammar recognized by the tools in this directory, which is a subset of the Oz lexical grammar. Note that an (Atom) that has the same sequence of characters as a (keyword) or a (Bool) will be recognized as that (keyword) or a (Bool), unless it is quoted.
2 Context Free Grammar for the Oz Subset

The parser in MyOzParser.oz recognizes the grammar in Figure 2 on the next page, which builds on the lexical grammar as usual.

In the figure, the names in the right column are not part of the grammar, but instead show what type of abstract syntax tree(s) record(s) is (are) made from the production(s) on the left. Compare these with the abstract syntax definitions in Section 3.

3 Abstract Syntax Trees for the Oz Subset

The grammar in Figure 3 on page 5 describes the abstract syntax trees produced by the parser and used by other tools. These data structures are all records with distinguishing labels. Thus the grammar in the figure represents the abstract syntax of our Oz subset. For example the following Oz statement:

A = B

is represented by (parsed into) the following abstract syntax tree:

unifyStmt(varId('A') varId('B'))

Note that the grammar uses Oz atoms to represent variable identifiers.

A more complex example of an abstract syntax tree is given in Figure 4 on page 6.
Figure 2: Concrete syntax recognized by the parser, which is a subset of Oz.
Figure 3: Abstract syntax for an Oz Subset, based on the textbook’s section 2.6 [VH04]. The optional (Position) field is used for error messages, but is often omitted in examples made by hand. The type (String) is the type of character strings in Oz, (Atom) is the type of atoms in Oz, (Int) is the type of Integers in Oz, etc.
The following Oz statement:

```oz
fun {AddToEach A#B Ls}
    case Ls of
    (X#Y)|T then ({Plus A X}|{Plus B Y})|{AddToEach A#B T}
else nil
end
end
```

is represented by the following record structure (with all positions omitted):

```scheme
namedFunStmt('AddToEach'
    [recordPat(atomExp('#'))
     varIdPat('Ls')
    caseExp(varId('Ls')
     recordPat(atomExp('|')
      [posFld(recordExp(atomExp('#')
       [posFld(varIdPat('X'))
        posFld(varIdPat('Y'))])
      posFld(varIdPat('T'))])
    recordExp(atomExp('|')
      [posFld(recordExp(atomExp('#')
       [posFld(applyExp(varId('Plus')
        [varId('A')
         varId('X')])
      posFld(applyExp(varId('Plus')
        [varId('B')
         varId('Y')])))]
    posFld(applyExp(varId('AddToEach')
      [recordExp(atomExp('#')
       [posFld(varIdPat('A'))
        posFld(varIdPat('B'))]
    varId('T')]]))
atomExp(nil)))
```

Figure 4: Example showing how Oz is parsed into the record structures (abstract syntax trees) from Figure 3 on the preceding page.
4 The Main Functions

The reducer provides the following main functions to users.

1. The function
   \[\text{FreeVarIdsStmt}: \langle \text{fun} \{\$ <\text{Statement}>\}: <\text{Set} <\text{String}>\rangle\]
   takes a \langle Statement \rangle in the grammar of Figure 3 on page 5 and returns a set of all the variable identifiers that occur free in its argument. Its implementation is in FreeVarIds.oz. There are tests in the file FreeVarIdsTest.oz.
   See the file TestOutput.txt for results of testing.

2. The function
   \[\text{BoundVarIdsStmt}: \langle \text{fun} \{\$ <\text{Statement}>\}: <\text{Set} <\text{String}>\rangle\]
   takes a \langle Statement \rangle in the grammar of Figure 3 on page 5 and returns a set of all the variable identifiers that occur bound in its argument. Its implementation is in BoundVarIds.oz. There are tests in the file BoundVarIdsTest.oz.
   See the file TestOutput.txt for results of testing.

3. The function
   \[\text{DesugarStmt}: \langle \text{fun} \{\$ <\text{Statement}>\}: <\text{Statement}\rangle\]
   takes a \langle Statement \rangle in the grammar of Figure 3 on page 5 and returns a \langle Statement \rangle in the subset of that grammar that represents Oz statements in the kernel language of the textbook [VH04, Section 2.3].
   An implementation is in Desugar.oz. There are tests in the file DesugarTest.oz.
   See the file TestOutput.txt for results of testing.

4. The function
   \[\text{Reduce1}: \langle \text{fun} \{\$ <\text{Config}>\}: <\text{Pair String Config}>\rangle\]
   takes a non-terminal configuration and returns a new rule name and a configuration for the next step in the operational semantics of Oz, starting at the given configuration.
   The implementation is in Reducer.oz. There are tests in the file ReducerTest.oz.
   See Configuration.oz for details on configurations.

References