New Features of Eli Version 4.2

Uwe Kastens

University of Paderborn
D-33098 Paderborn
FRG

A. M. Sloane

Department of Computing
School of Mathematics, Physics, Computing and Electronics
Macquarie University
Sydney, NSW 2109
Australia

W. M. Waite

Department of Electrical and Computer Engineering
University of Colorado
Boulder, CO 80309-0425
USA

$Revision: 3.3 $

Table of Contents

This document gives information about new facilities available in Eli version 4.2 and those modifications made since the previous distributed Eli version 4.1 that might be of general interest. Numerous corrections, improvements, and additions have been made without being described here. They shall just help users to solve their problem without taking notice of Eli's mechanism.

1 New FunnelWeb Typesetter Support

Two typesetter pragmas have been added to FunnelWeb:

```
@p typesetter = latex
@p typesetter = latex2html
```

These pragmas cause the derivation `:fwTex` to create text acceptable to LaTeX and latex2html respectively. They are identical as far as the treatment of text is concerned, but differ in how they handle FunnelWeb macro definitions and invocations. In both cases, normal LaTeX and latex2html markup can be used in the text portions. Macro definitions and calls are implemented using LaTeX markup when `latex` is specified, and latex2html markup when `latex2html` is specified.

In both cases the user must supply an appropriate LaTeX preamble, `\begin{document}` and `\end{document}` as part of the document text. This means that it is possible to combine the outputs from several `:fwTex` derivations into a single document by using appropriate LaTeX `\input` commands.

The preamble must include the LaTeX command `\usepackage{alltt}` for either of these pragmas, and `latex2html` also requires `\usepackage{html}`.

The FunnelWeb section directives are normally translated to LaTeX sectioning commands as follows:

- `@A` `\section`
- `@B` `\subsection`
- `@C` `\subsubsection`
- `@D` `\paragraph`
- `@E` `\subparagraph`

If the `+chapter` parameter is passed to the `:fwTex` derivation, however, the translation is:

- `@A` `\chapter`
- `@B` `\section`
- `@C` `\subsection`
- `@D` `\subsubsection`
- `@E` `\paragraph`

2 New token processor lexerr

The token processor `lexerr` reports that the character sequence is not a token. It does not alter the initial classification, and does not compute a value.

Normally, a lexical analyzer generated by Eli attaches an error report to each character that it does not recognize. While this behavior is adequate in most cases, it is sometimes necessary for the designer to specify a particular sequence of characters to be erroneous. One typical example is disallowing tab characters:

```
TAB     [lexerr]
```

The canned description `TAB` handles all of the coordinate updating (see Section "Maintaining the source text coordinates" in *Lexical Analysis*). Since there is no label on this line, the tab character is classified as a comment. That classification is not changed by `lexerr`, which simply reports a token error at the coordinates of the tab (see Section "Making White Space Illegal" in *Lexical Analysis*).

There is no source file for `lexerr`; it is a component of the scanner itself, but its interface is exported so that it can be used by other modules.

3 OIL

Four constant definitions have been added to the OIL library in an effort to simplify code using it and to bring its interface more into line with other interfaces in the Eli system. These new definitions do not change the library's functionality in any way:

`OilInvalidType`
: The type to which all types can be coerced, and which can be coerced to any type. This is the value returned by `OilErrorType()`.

`OilInvalidOp`
: The invalid operator, resulting from an operator identification failure. This is the value returned by `OilErrorOp()`.

`OilEmptyArgSig`
: The empty signature, used in constructing signatures. This is the value returned by `OilNewArgSig()`.

`OilEmptySetSig`
: The empty type set signature, used in constructing argument lists. This is the value returned by `OilNewSetSig()`.

4 Tree Parsing

Type-'`.tp`'-files specify tree parsers. Tree parsers can be used to transform, interpret and print tree-structured data. They are particularly useful for problems in which the action at a node depends strongly on the context in which that node appears. Code selection is a common example of this kind of problem: The code selected for an operation is largely determined by that operation's context.

Specifications written in the TP language are analyzed for consistency and then compiled into specifications for the BURG processor (Fraser, C. W., R. R. Henry and T. A. Proebsting, "BURG – Fast Optimal Instruction Selection and Tree Parsing", SIGPLAN Notices 27, 4 (April, 1992) 68-76). Eli will now also accept raw BURG specifications (file type '`.burg`'), but we recommend that TP specifications be used instead because they are easier to write and understand.

For a detailed treatment of tree parsers, see *Tree Parsing*.

5 ModLib

This chapter summarizes changes made to the Module Library. For more details, see Section "top" in *Specification Module Library: Abstract Data Types*.

List-Module

A macro `SingleTYPEList(e)` has been added. It creates a singleton list from the element `e`. In LIDO specifications it may be used e.g. in `WITH` clauses of `CONSTITUENTS`.

A function `AddToOrderedSetTYPEList` has been added. It adds an element to a list if it is not yet in that list. In contrast to the function `AddToSetTYPEList` it is assumed that the list is ordered increasingly.

LidoList-Module

A symbol role `TYPEFilterListElem` has been added. On list construction it may be used instead of `TYPEListElem` in order to decide for each element whether it is to be inserted into the list.

6 Caution Needed in Specifying a Compiler

The environment variable `ELI_CC` can be used to specify the C compiler that should be used to compile user-provided and Eli-generated code. Specification of a C compiler in this manner is considered to be "permanent", so Eli assumes that the value of `ELI_CC` will remain unchanged over the life of a cache. If you change the value of `ELI_CC`, you should restart Eli with the `eli -r` command to reset the cache. Otherwise, Eli will mix object code produced by the two compilers with unpredictable results.

You can also specify the C compiler to be used by means of the `+cc` parameter. That specification holds only for the request containing it. Eli will guarantee that all of the object code needed to satisfy that request was produced by the specified compiler. Successive requests specifying different compilers will be handled correctly.

Index