Tool Demonstration

The Eli System

Uwe Kastens, Peter Pfahler, Matthias Jung
Universität-GH Paderborn, Fachbereich 17
D-33095 Paderborn, Germany
Email: {uwe, peter, mjung}@uni-paderborn.de

Methods and techniques of compiler construction are applicable to a range of problems that is much broader than the development of compilers for programming languages: Processors for input languages, design languages, specification languages, and intermediate languages in application programs all demand solutions to translation problems.

We have combined a variety of standard tools that implement powerful compiler construction strategies into a domain-specific programming environment called Eli. Using this environment, one can automatically generate complete language implementations from application-oriented specifications. The implementations may involve translation into an arbitrary target language, or may be interpretive.

Eli provides modern compiler construction facilities to users with a wide range of sophistication. It offers complete solutions for commonly-encountered language implementation subtasks and contains libraries of reusable specifications, making possible the production of high-quality implementations from simple problem descriptions.

Eli generates programs whose performance is comparable to that of a good hand-coded implementation. Development time for a processor using Eli is generally about one third of that for comparable hand code, and maintenance is significantly easier because specifications rather than implementations are being maintained.

Eli has been in the field since 1989 and has been used in many projects worldwide. The Eli development team consists of the groups of William M. Waite (University of Colorado at Boulder, USA), Antony M. Sloane (James Cook University of North Queensland, Townsville, Australia), and Uwe Kastens (Universität-GH Paderborn, Germany).

Application Areas

Eli has been and is being used successfully in a number of areas to produce translators, program generators, analyzers and interpreters. Here is a small selection of applications:

- Programming language compilers for ANSI C to SPARC machine code, for Pascal to Pcode, for Pascal to C, and for many user-defined languages.

- Special processors that translate musical notation to PostScript, producing scores, or statistical data to TeX tables or histograms.
- Program generators that produce C simulation programs from descriptions of mechanical systems, or query language calls from a database description, or finite-element models from descriptions of solids.
- Interpreters that do animation from descriptions of graphics, or interactively evaluate decision tables.
- Analyzers to enforce programming style, or interactively validate commands for satellite control, or compute and present statistics.

Advantages of Eli

Eli is a freely-available system that has many advantages over conventional tool sets for compiler construction:

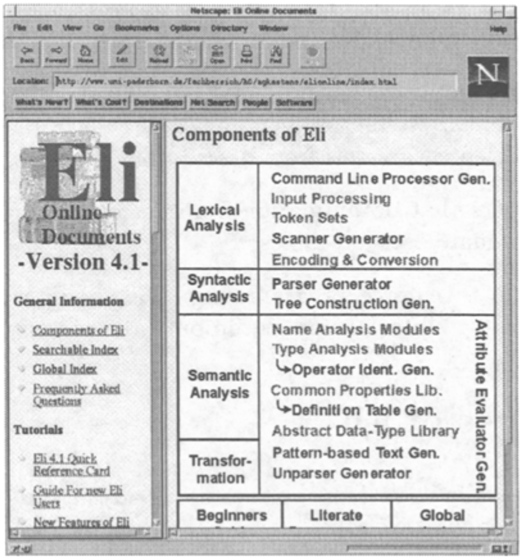
- Problem-oriented instead of tool-oriented:

  The user describes the problems that must be solved. Eli automatically employs the tools and components needed for that particular problem. Because of this, it is possible to improve tools and components without invalidating specifications.
- Broad spectrum of tasks and solutions:

  Eli offers solutions for most of the tasks that must be carried out to implement a language. They range from structural analysis (solved by tools analogous to LEX and YACC) through analysis of names, types and values, to storage of translation data structures and production of the target text. There is even a declarative specification for the command-line parameterization of the generated programs.
- Little prior knowledge required:

  A basic understanding of languages and language structures suffices to begin using Eli. Tutorials with sample problems and solutions, together with extensive on-line documentation and an interactive help system lead the user further if they so desire.
- Integrated system, with flexible overriding:

  All components of Eli are coordinated with one another in such a way that the interfaces of the generated modules fit together. As a result, Eli generates a complete set of C modules, including a Makefile, to solve the specified problem. Mechanisms are provided to allow users to override standard modules and interfaces if necessary to solve specific problems.


Netscape: Eli Online Documents
File Edit View Go Bookmarks Options Directory Window Help
Eli Online Documents -Version 4.1-
General Information
Components of Eli
Searchable Index
Global Index
Frequently Asked Questions
Tutorials
Eli 4.1 Quick Reference Card
Guide For new Eli Users
New Features of Eli

Components of Eli

Lexical Analysis	Command Line Processor Gen. Input Processing Token Sets Scanner Generator Encoding & Conversion	
Syntactic Analysis	Parser Generator Tree Construction Gen.	
Semantic Analysis	Name Analysis Modules Type Analysis Modules ↳Operator Ident. Gen. Common Properties Lib. ↳Definition Table Gen. Abstract Data-Type Library	Attribute Evaluator Gen.
Transfor-mation	Pattern-based Text Gen. Unparser Generator	

Beginners	Literate	Global

Figure 1: Eli's Online Documentation

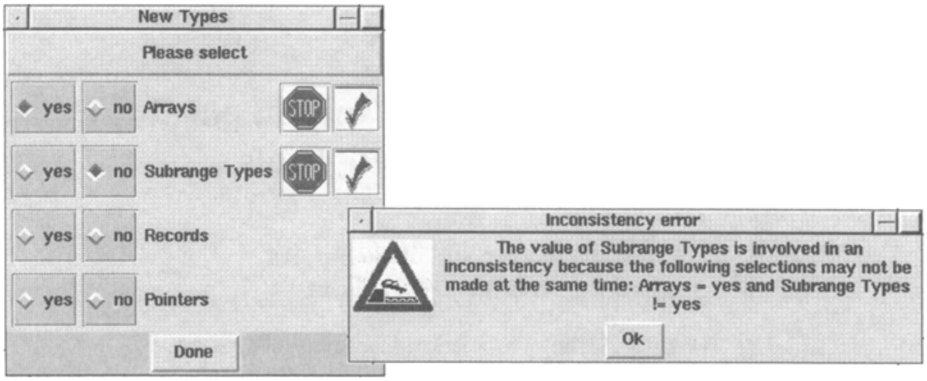
- State-of-the art components:

  Eli's tools and modules are implemented according to current understanding of the best approaches to solving translation problems. The generated programs are reliable and efficient. Eli is widely used and tested.

Tool demonstration

Our tool presentation demonstrates how Eli can be used to generate language implementations from specifications. The goal is to give an impression of how some of the specification formats for Eli look and how solutions for commonly encountered problems can be reused from Eli's module library.

While the process of compiler construction from specifications itself does not produce interesting graphics, Eli provides a hypertext-based online-help system (see Fig. 1) and several graphical debugging tools for language specifications which can be nicely demonstrated. Furthermore, we present the LaCon ("language construction") system approach which we built on top of Eli. A LaCon system is used by experts from application domains to design and implement domain-specific languages by selecting combinable language specification components and feeding them into the language implementation system Eli. LaCon provides a user interface for language design (see Fig. 2) which completely hides the underlying compiler technology for language implementation.



[✓ yes] no Arrays

yes [✓ no] Subrange Types

yes no Records

yes no Pointers

The value of Subrange Types is involved in an inconsistency because the following selections may not be made at the same time: Arrays = yes and Subrange Types != yes

Figure 2: Design and implementation of domain specific languages using LaCon

Further Information

- The Eli Development team:
  - `www.cs.colorado.edu/~eliuser/`
  - `www.cs.jcu.edu.au/~tony/`
  - `www.uni-paderborn.de/fachbereich/AG/agkastens/eli_home.html`

- Papers on Eli
  - Gray, R. W., Heuring, V. P., Levi, S. P., Sloane, A. M., Waite, W. M. *Eli: A Complete, Flexible Compiler Construction System*, Communications of the ACM 35 (February, 1992), 121-131.
  - Waite, W. M. *An Executable Language Definition*, SIGPLAN Notices 28 (February, 1993), 28-40.
  - Kastens, U. *Attribute Grammars in a Compiler Construction Environment*, Lecture Notes in Computer Science 545, Springer Verlag, 1991, 380-400.
  - Kastens, U., Waite, W. M. *"Modularity and Reusability in Attribute Grammars*, Acta Informatica 31 (1994), 601-627.
  - Sloane, A. M. *An Evaluation of an Automatically Generated Compiler*, ACM Transactions on Programming Languages and Systems 17 (September, 1995), 691-703.

- Paper on LaCon
  - Kastens, U., Pfahler, P. *Compositional Design and Implementation of Domain-Specific Languages*, Proc. Systems Implementation 2000, Berlin, Germany, Feb. 1998.