EEL: Machine-Independent Executable Editing
James R. Larus and Eric Schnarr
Presented by C. Shen
CMSC714, Fall 2005

Outline
- Introduction
- EEL Abstractions
  - Executables
    - Routines
    - CFG: Control-Flow Graph
    - Instructions
    - Code Snippets
- System-Dependent EEL
- EEL Status
- Conclusions

Introduction

Executables
- Source files
- Libraries

Executable Editing
- Remove existing instructions and add foreign code in executable.
- Executable editing is widely used for emulation, observation, and optimization.

EEL
- EEL: Executable Editing Library
- A C++ library for building tools to analyze and modify an executable program
- EEL can edit fully-linked executables.
- EEL emphasizes portability across systems.
- Mostly machine-independent interface
- Applications: qpt (A Quick Program Profiling and Tracing System)
Five major machine-independent abstractions that allow a tool to examine and modify an executable.
- **Internal representation:** register-transfer level (RTL) instruction description

**EEL Abstractions**

**Executables**

- EEL executable objects are an abstraction of executable files
  - Object, library, or static and dynamically-linked programs
- EEL refines symbol information
  - Data tables, hidden routines, and multiple entry points
- EEL maintains symbol table information for the edited program
  - Debugging information for the edited executable

**EEL Abstractions**

**Routines**

- Routines are named objects in a program’s text segment that contain instruction and data.
  - Hold information about an entity in the text segments
  - Provide interfaces to EEL’s control-/data-flow analysis
- Control-flow analysis may split routine

**EEL Abstractions**

**CFG**

- **CFG:** Control-Flow Graph
  - Directed graph
  - Nodes: basic blocks
  - Edges: control flow between blocks
- The primary program representation in EEL
  - Represent a routine as a CFG
- Why CFG?
  - Implement profiling and tracing on CFG edges
  - Adjust addresses in branch and jump instructions
  - Provide an architecture-independent way of representing control flow

**EEL Abstractions**

**CFG (cont’d)**

- Architecture-independent control-flow representation
  - Basis for program analysis
  - EEL uses internally
  - Normalization

**EEL Abstractions**

**CFG (cont’d)**

- Tools edit CFG
  - Delete instruction
  - Add new code before/after instruction or along edge
  - Accumulate edits without changing the CFG (batch style editing)
- After editing CFG
  - Produce a new version of the routine
  - Incorporate the changes
  - Involve laying out blocks and snippets
  - Update control-transfers instructions (calls, branches, jumps)
EEL Abstractions

Instructions
- RISC-like machine instructions
  - Memory references (loads and stores)
  - Control transfers (calls, returns, system calls, jumps, and branches)
  - Computations
  - Invalid (data)
- C++ classes
  - Combine for more complex instructions
  - E.g. autoincrement load = a memory reference + a computation

EEL Abstractions

Instructions (cont’d)
- Inquiries about an instruction’s effect on a program’s state
- Inquiries independent of an underlying machine
  - Code is similar to the original algorithm

EEL Abstractions

Snippets
- Foreign code added to an executable
  - EEL allocates registers from unused (dead) or freed registers at insertion point.
  - Code in snippet is not machine-independent.

EEL Abstractions

Snippets
- Foreign code added to an executable
  - EEL allocates registers from unused (dead) or freed registers at insertion point.
  - Code in snippet is not machine-independent.

System-Dependent EEL
- Instrumentation code is machine-specific

System-Dependent EEL
- A EEL tool for binary instruction analysis and manipulation
  - Customize annotated C++ code from high-level machine description

Outline
- Introduction
- EEL Abstractions
  - Executables
  - Routines
  - CFG: Control-Flow Graph
  - Instructions
  - Code Snippets
  - System-Dependent EEL
  - EEL Status
  - Conclusions

Code Snippets
- EEL allocates registers from unused (dead) or freed registers at insertion point.
- Code in snippet is not machine-independent.
System-Dependent EEL spawn (cont’d)

Example: portion of spawn’s SPARC description

```c
// Instruction field definitions:

// Instruction[32] fields:
op 33-64, rgl 22-24, opr 39-24, opr 1-13,
si 20-24, rld 14-19, rld 6-4, ildef 18-13,
imm13 11-0, imm2 2-1, disp2 2-0,
disp6 7-2, cond 21-20, aflag 20-8,
add 4-13

// Control-transfer instruction syntax:

//

//

//

//

//

```

System-Dependent EEL spawn (cont’d)

Added description of instruction semantics

```c
// General purpose register set

//

//

//

```

Outline

- Introduction
- EEL Abstractions
  - Executables
  - Routines
  - CFG: Control-Flow Graph
  - Instructions
  - Code Snippets
- System-Dependent EEL
- EEL Status
- Conclusions

EEL Status

- EEL runs on SPARC processors under SunOS & Solaris
- Spawn not yet distributed
- QPT2: a EEL-based profiler
- Other applications:
  - Active Memory (a memory system simulation platform)
  - Elsie (a direct-execution architectural simulator)
  - Wisconsin Wind Tunnel architectural simulator
  - Blizzard-S’s fine-grain access control

Conclusions

- Tools to modify executables have proven their value in many areas
  - Monitor program behavior and performance
  - Architectural experiments
- EEL is a highly portable library for editing executable programs
  - Provides mostly architecture- and system-independent set of operations
  - Provides machine-independent CFG and program analysis
  - Simplify the analysis and manipulation of most programs