Puerto De La Cruz, Tenerife, Spain
December 10-12, 2011

DESIGN OF AN AUTOMATICALLY GENERATED RETARGETABLE DECOMPILER

Lukáš Ďurfina, Jakub Křoustek, Petr Zemek, Dušan Kolář, Tomáš Hruška,
Karel Masařík, Alexander Meduna

Faculty of Information Technology
Brno University of Technology, Czech Republic

Contents

PART 1

INTRODUCTION AND MOTIVATION

Introduction and Motivation

- Decompilation
  - Reverse translation: binary executables => HLL code (C, Java, etc.)
  - Harder than compilation (a lot…)
- Motivation
  - Reverse engineering (how does it work?)
  - Cross-platform porting
  - Adding features to 3rd party SW
  - Recovery of lost source code
  - Finding bugs, vulnerabilities, malware, etc.
- Our focus
  - Retargetability, automation, high readability

PART 2

STATE OF THE ART

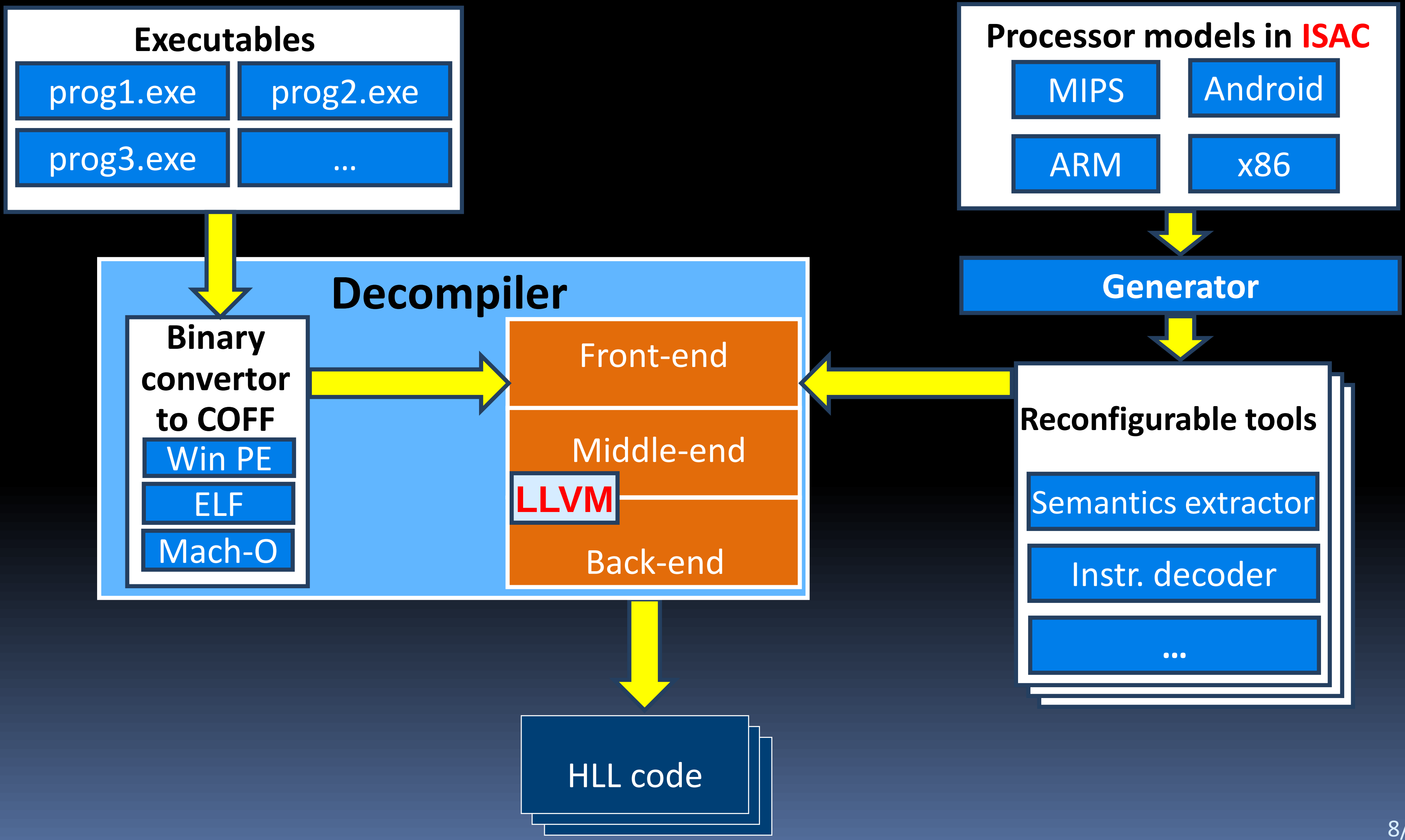
State of the Art

- Reverse engineering
  - Disassemblers, decompilers, …
- Single target architecture decompilation
  - Open problem for > 40 years (HP equivalent problem)
  - Several attempts (dcc, Boomerang, Hex-Rays, REC Decompiler)
- Retargetable decompilation
  - Reconfigurable (based on target platform description)
  - The PILAR System (1970, never completed)
  - No other similar projects

PART 3

RETARGETABLE DECOMPILER

Retargetable Decompiler



Retargetable Decompiler

- Developed within the Lissom project (BUT FIT)
  - In cooperation with AVG Technologies
- Exploitation of existing technologies
  - Architecture Description Language ISAC (BUT FIT)
  - LLVM Compiler System
- Reconfigurable, automatically generated
- Input
  - Platform-dependent binary application
  - Platform model in ISAC
- Output
  - HLL representation of the input application

Front-end

- Goal: Semantic translation of the input program
  - Platform-dependent format => internal COFF format => LLVM IR
- Platform dependent
  - Generated based on the architecture model (in the ISAC ADL)
- Tasks:
  - File format conversion (support of ELF, PE, Mach-O, DEX, E32, …) to COFF
  - Generic "disassembly" (based on formal models)
  - Static code detection (signature based detection, FLIRT, types)
  - Static analysis (code vs. data, control-flow analyses, etc.)
  - Compiler detection (for MS Windows and GNU/Linux)
  - …

Middle-end and Back-end

- Goal: HLL code reconstruction
  - LLVM IR => HLL independent IR => target HLL code
  - Uses existing optimizations and transformations + own passes
- Platform independent
  - Built on top of LLVM Compiler System
- Tasks:
  - Support of different HLLs (Python‘, C)
  - Recognition of high-level constructs (loops, IF statements, etc.)
  - Emission of the target HLL code
  - Post-processing

PART 4

EXPERIMENTAL RESULTS

Experimental Results

- Used platform: Sony PlayStation Portable (PSP)
  - video game hand-held console
  - dual-core processor based on MIPS-4000
  - executables are in the PRX format
- Used compiler: psp-gcc
  - from PSP SDK (4.3.2)
  - with enabled optimizations (-O2)

Experimental Results

- Original code:

```
#include <pspkernel.h>
#include "sum.h"

/* Initialization */
PSP_MODULE_INFO("template", 0, 1, 1);
PSP_MAIN_THREAD_ATTR(0x80004000);

int main(void)
{
    volatile int a = 3;
    int b;
    for (b = 1; b < 100; b++)
        a = sum(a, b);
    return a;
}
```

- Decompiled code:

```
# -------- Global Variables --------
  orange = 0
  banana = 0
  lemon = 0
# --------- Declarations -----------
  int sum(int, int)
# ------ Defined Functions ---------
  def main():
    orange = 3
    result = orange
    for i in range(0, 99):
      banana = result
      lemon = i + 1
      result = sum(banana, lemon)
    return result
```

PART 5

CONCLUSION

Conclusion and Future Work

- Retargetable decompiler
  - Reconfigurable, platform independent
  - No other similar solutions
  - Exploitation of existing technologies
    - ISAC ADL, LLVM Compiler System
- Result
  - Proof of concept: decompilation of a MIPS program (Sony PSP)
  - Produces highly readable HLL code (Python-like language)
- Future research
  - Static analysis (ABI detection)
  - Other HLLs (C, type information)
  - Testing on different architectures (x86, ARM, …)

DISCUSSION