MO601 – Project 3
LISC
Learning Instruction Semantics from Code Generators

093311 – Alberto Arruda de Oliveira

Overview

- Binary analysis and instrumentation
  - Program monitoring (Valgrind, Pin)
  - Virtualization
  - Malware Analysis
  - Etc.

- Modeling of instruction Semantics
  - Translation of assembly to intermediate language representation

- Requires previously existing models, created manually
  - Limited Support of new architectures

LISC
Objective

- Automated modeling of instruction semantics

- Compilers: Intermediate Language -> Assembly Code
  - GCC, LLVM, etc.

- LISC:
  - Learn from code generators
  - Automatically extract semantics
  - Test extracted semantics

LISC
Benefits

- Automated instruction semantics modeling
  - Machine Learning Approach
  - Reduce manual efforts
  - Broaden Architecture Support

- Architecture Neutrality
  - Code Generators

- Well tested compiler code
  - Among the most tested code

Project 3
Objective

- Test LISC in a already used architecture of choice
  - x86, ARM, AVR

- Reproduce **at least one** of LISC’s evaluation approaches:
  - Completeness
  - Architecture-Neutrality
  - Performance
  - Correctness

LISC Evaluation Completeness

- Are all the instructions lifted correctly?

- 99.5% of Ubuntu/x86 and 99.8% of Debian/ARM binaries
  - Missed Instructions are mostly NOPS

- Evaluation:
  - Testing programs P_{test} and training programs $P_{train} \subset P_{test}$
  - Build a transducer using P_{train}
  - Translate the assembly of P_{test}
  - Metrics:
    - Fraction of unique instructions translated
    - Total number of non-translated instructions

LISC Evaluation
Architecture Neutrality

- Can the system be easily adapted to a new architecture?
- 9 person-hours to support ARM and 3 person-hours to support AVR
- **Evaluation:**
  - **Code that is architecture specific**
- **Project 4:** Support a new architecture!

LISC Evaluation Performance

- How long does training/testing takes?

- 10 minutes training
  - 1.4GB of code generator logs
  - 4830 transducer states

- Lifting binaries from Ubuntu/x86 and Debian/ARM archiectures (around 100M instructions): around 8 hours

- Linear-time algorithm for lifting binaries

LISC Evaluation Correctness

- Semantic equivalence test
  - *ArCheck*

- Consistency Check
  - Distinct Intermediate Language Snippets -> Same Assembly Instruction *A*
  - *A* may translate to two different snippets

- Loopback Test: Given a binary with a list of instructions α
  - Lift α to Intermediate language γ using LISC
  - Use GCC to generate code α' for γ
  - Check if α' is consistent with α

Project 4
Possible Ideas

- New check of architecture neutrality
  - Generate language model for a non-supported architecture

- Alternative machine learning technique
  - Can a different approach to training and testing be employed?

- Evaluation Flaws
  - Is there some aspect of current evaluation that fails to cover possible scenarios?

THANK YOU!