Using Code Generators as De-code Generators: A Novel Approach for Assembly to IR Translation
Niranjan Hasabnis and R. Sekar ([nhasabni, sekar]@cs.stonybrook.edu)

**Problem**
- Manual development of assembly-to-IR translators
  - Used by binary instrumentation and translation systems
  - Complex modern ISA (e.g., Intel v052 is 1400 pages)
  - Cumbersome - Valgrind does not support FMA, SSE4.1.
  - Error-prone - QEMU contains modeling errors [2]

**Solution**
- Build assembly-to-IR translators automatically! How??
  - Our answer: Yes.
  - Solution: Code generator = black-box

**Approach**
1. Compile plenty of packages
2. Learn numeric parameters in assembly.
3. Replace them by placeholder.
4. Map them (=, +, -, ×, ÷) to IR parameters.
5. Repeat 1–3 for every mapping rule from log.

<table>
<thead>
<tr>
<th>IR</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>([set (reg : AX)]</td>
<td>([set (reg : AX)</td>
</tr>
<tr>
<td>(plus (reg : AX)</td>
<td>(const_int 2)])</td>
</tr>
<tr>
<td>(clobber (reg : FLAGS))</td>
<td>add $2, %eax</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IR</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>([set (reg : AX)]</td>
<td>([set (reg : AX)</td>
</tr>
<tr>
<td>(plus (reg : AX)</td>
<td>(const_int P)</td>
</tr>
<tr>
<td>(clobber (reg : FLAGS))</td>
<td>add $P, %eax</td>
</tr>
</tbody>
</table>

Parameterized pairs = assembly-to-IR translator

**Results**
- Compilation logs obtained from OpenSSL and Binutils
- Code generator = GCC
- Built assembly-to-IR translators for x86 and ARM
  - Completeness = how many of coreutils binaries could be translated to IR
  - Testing against binaries produced by LLVM

**Contributions**
1. Novel architecture-neutral approach to build assembly-to-IR translators automatically
2. Already supports 2 architectures - x86 and ARM
3. Already supports multiple compilers - GCC, LLVM compiled coreutils binaries
4. Supporting new architecture took 3 hours.
5. Supports advanced instruction sets (FMA, AVX, SSE4.1) not supported by Valgrind.

**Publication**

**References**

**Future Directions**
- More thorough evaluation
- Build binary translation, instrumentation, analysis systems
- Explore the need of treating registers as parameters