Summary of the Bachelor thesis “Decompilation of LLVM IR”

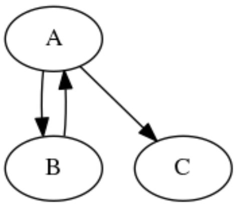
2014-10-08

This is a summary of a Bachelor thesis written by Simon Moll in 2011, entitled “Decompilation of LLVM IR” [1].

Moll’s paper explores various techniques for reconstructing high-level primitives from LLVM IR, which is the low-level intermediate representation of the LLVM compiler infrastructure. Or, to phrase it differently; it demonstrates how to turn machine readable code into human readable code, a process known as decompilation.

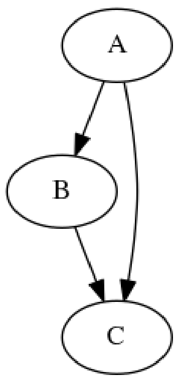
The LLVM IR language is conceptually a platform-independent RISC assembly language whose control flow is structured using basic blocks. Each basic block consists of a sequence of instructions and ends with a terminator instruction (such as a branch or a function return), which indicates the next basic block to execute. The connections between basic blocks, as specified by the execution flow, may be represented using a Control Flow Graph (CFG), in which each basic block represents a node.

In the following illustrations each letter (A-C) represents a node in a CFG, and each arrow represents the flow of execution from one node to another.



```
for (A) {
        B
}
C
```

Figure 1: The CFG (left) and pseudo-code (right) of a for-loop.



```
if (A) {
        B
}
C
```

Figure 2: The CFG (left) and pseudo-code (right) of an if-statement.

A key insight provided by the paper is the idea that various high-level primitives, such as for-loops and if-statements, have different patterns in the CFG. This enables the use of pattern matching to reconstruct high-level primitives from lower level representations.

References

[1] S. Moll, “Decompilation of LLVM IR,” BSc thesis, Saarland University, 2011. [Online] Available: `http://www.cdl.uni-saarland.de/publications/theses/moll_bsc.pdf`. [Accessed: 4 Oct, 2014].