STS Software Technology Systems

Compiler Construction
Summer Term 16
June 9, 2016
Mattsen/Schupp

Code Generation with LLVM

How to complete an exercise successfully?

Follow the rules as described in the Lecture!

How to get additional information?

You are encouraged to discuss past and present exercise sheets with the teaching assistants. Either approach the teaching assistant during the exercise session, or visit us during the weekly office hours. We are also available through e-mail or on the StudIP forum. We try to reply as quickly as possible and in general, you should get a reply the next weekday, but we cannot guarantee this.

Code Generation with LLVM

Your task is to compile our small CPP language to LLVM IR. In principle, there are two ways of doing this. You could either generate LLVM IR directly, meaning generating a string that contains LLVM IR as text representation, or use the LLVM API to generate the IR's AST directly. The second approach is considerably faster and is therefore preferred.

LLVM IR is structured in basic blocks, meaning a sequence of non-branching instructions, terminated by a control-flow-affecting instruction (terminator), e.g., jumps and returns. Furthermore, LLVM IR makes evaluation order explicit. The LLVM IR for the expression $a + b + c$ would, e.g., be compiled to something similar to the following:

```
%1 = add i32 %a, %b
%2 = add i32 %1, %c
```

The LLVM language reference[1] gives you a more precise overview. If you use Haskell as your implementation language, look at the modules `llvm-general`[2] and `llvm-general-pure`[3]. As you can see in the sample code above, you need to provide the result type of instructions, e.g., the `add` is of type `i32` (32 bit integer). Therefore, it would be beneficial if your type checker provides the necessary annotations.

Task

Implement a compiler that takes programs in our simple CPP language, prints valid, corresponding LLVM IR and, optionally, x86 assembly. For a rough description of the semantics, please refer to the course book's website[4], section operational semantics. In difference to the specification, you do **not** have to provide the six built-in functions!

Deadlines of open tasks

a) Type Checker: 09.06.2016, 13:14 (Upload to StudIP and demonstrate)

b) Code Generator: 23.06.2016, 13:15 (Upload to StudIP and demonstrate)

[1] http://llvm.org/docs/LangRef.html
[2] http://hackage.haskell.org/package/llvm-general
[3] http://hackage.haskell.org/package/llvm-general-pure
[4] http://www1.digitalgrammars.com/ipl-book/assignments/assignment3/assignment3.html