Code Generation

ICS312

Machine-Level and Systems Programming

Henri Casanova (henric@hawaii.edu)

Code Generation

- In the previous set of lecture notes we have create an ANTLR parser for our language
- In this set of lecture notes we make that parser generate code!
  - Only then can we really call it a compiler
- Let’s look at an example program in our source language again to refresh our memory

Example Program

```
int a;
int b;
a = 3;
b = a + 1;
if (b == 4)
   a = 2;
endif
if (a == 3)
    a = a + 1;
    b = b + 6;
endif
print a;
print b;
```

Code Generation

- Code generation is a pretty complex part of compilers, especially because the generated code should be fast
- One easy, but limited option, is to use syntax-directed translation
  - Attach *actions* to the rules of the grammar
  - Use *attributes* to non-terminals and terminals in the grammar
- There is quite a bit of theory here, but instead we'll just do it by example using the ANTLR syntax
  - ANTLR is so easy, that seeing examples is enough!
- First let's just review a few basic elements of how one can get ANTLR to output text, based on the rule of our grammar

ANTLR Syntax-directed translation

- Each time a grammar symbol is evaluated you can insert Java code to be executed!
- Example:

```
program :
    {System.out.println(“Declarations!”);}
     declaration*
     {System.out.println(“Statement!”);}
     statements*
    {System.out.println(“Done!”);}
     ;
```

ANTLR Syntax-directed translation

- Let’s start from the MyLanguageV0NoCode.g4 file on the course Web site and copy it into MyLanguageV0Code.g4 (changing the grammar’s name in it as well)
  - Let’s use a convenient Makefile I’ve set up to make this a bit less painful (Makefile_ANTLR_x86 on the Course Web site, which I’ll rename to “Makefile”)

- Let’s add a tiny bit of Java in this way to our parser to generate the standard parts of an x86 NASM program as we’ve done by hand this semester: preamble, cleanup, etc.

ANTLR Syntax-directed translation

- Each (lexer) token has an attribute called `text` that contains its lexeme
- Example:

```
declaration :
    INT NAME SEMICOLON
    {System.out.println(“Declared ”+$NAME.text);}
    ;
```

ANTLR Syntax-directed translation

- Let’s add a more Java to MyLanguageV0Code.g4 to deal with variable declarations…

ANTLR Syntax-directed translation

- You can give your own names to symbols in case you have multiple occurrences
- Example:

```
something :
    {int a,b;}
    a=NAME EQUAL b=NAME SEMICOLON
    {System.out.println($a.text + ″-″ + $b.text);}
    ;
```

ANTLR Syntax-directed translation

- You can create attributes for non-terminal grammar symbols and use them
- Example:

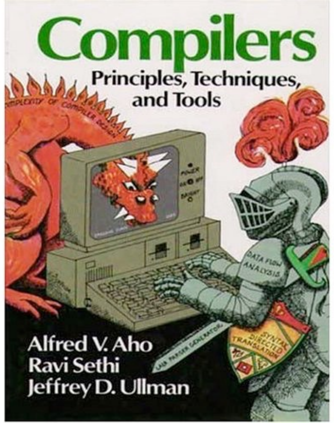
```
something :
       ident SEMICOLON
       {System.out.println(“stuff”+$ident.whatever);}
       ;

ident returns [String whatever]:
       NAME
       {$whatever = "somestring"+$NAME.text;}
       ;
```

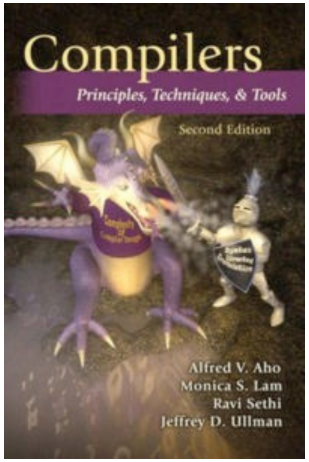
ANTLR Syntax-directed translation

- And with all this we can now implement our compiler
- Our goal: have ANTLR produce x86 assembly code that we can run!
- Let’s do it in class right now on my Linux VM…
    - A (hopefully) similar version is posted on the course Web site
- There will be mistakes, questions, hiccups, and confusion
- But the goal is to learn from this
- Feel free to suggest things to add to our language!
- Let’s look at the generated code and see if we see optimization options!!
- Off we go…. THIS WILL TAKE A WHILE

Conclusion

- There is a LOT of depth to the topic of Compilers
- We've only scratched the surface here
- There are well-known books on compilers





- Let's look at Homework #9 (last one!)
- We'll have an in-class practice quiz on this module next week
- If time permits, we can now talk a bit about code optimization…