Gabriel Arnell

Lab 1

Crafting a Compiler 1.11

The Measure Of Software Similarity (MOSS) [SWA03] tool can detect similarity of programs written in a variety of modern programming lan-guages. Its main application has been in detecting similarity of programs submitted in computer science classes, where such similarity may indi-cate plagiarism (students, beware!). In theory, detecting equivalence of two programs is undecidable, but MOSS does a very good job of finding similarity in spite of that limitation. Investigate the techniques MOSS uses to find similarity. How does MOSS differ from other approaches for detecting possible plagiarism?

As described by Danny Yang, a basic way to detect plagiarism would be to simply compare strings of text in the source code. MOSS does not review or compare every substring piece of text. In fact, it throws lots of information out the window when computing similarity. When MOSS processes a document it will remove all useless or easily changeable information (like whitespace or variable names). The processed document will then be ran through a "winnowing" hash which means the document is grouped into discrete sections (like groups of 5 characters) that are hashed in sequence. Specific hash results from this output are taken as the document's 'fingerprint'. This fingerprint is then used to compare against other code to measure the similarity of fingerprints. Overall, the method of winnowing and hashing makes it extremely unlikely that it will return a false negative because even a slight difference will return a completely different hash, while also making attempts to hide the plagiarism very difficult because it is unpredictable what sections actually get fingerprinted.

Source:
https://yangdanny97.github.io/blog/2019/05/03/MOSS

Crafting a Compiler 3.1

Assume the following text is presented to a C scanner:

```
main(){
    const float payment = 384.00;
    float bal;
    int month = 0;
    bal=15000;
    while (bal>0){
        printf("Month: %2d  Balance: %10.2f\n", month, bal);
        bal=bal-payment+0.015*bal;
        month=month+1;
    }
}
```

What token sequence is produced? For which tokens must extra information be returned in addition to the token code?

function (extra information that it is ‘main’)
left_paren
right_paren
left_bracket
constant
float_type
Identifier (extra information that it is ‘payment’
assign
Decimal_literal (extra info: it is 384.00)
Endstatement
Float_type
Identifier (extra information that it is ‘bal’);
Endstatement
Int_type
Identifier (extra info: it is ‘month’)
assign
Int_literal (extra info: 0)
Endstatement
Identifier (extra info: it is ‘bal’)
assign
Int_literal (info: 15000)
While
Left_paren
Identifier (extra info: it is ‘bal’)
Greater_than
Int_literal (info: 0)
Right_paren
Left_bracket
Printformat
Left_paren
Quote
Char (M)
Char (o)
Char (n)
Char (t)
Char (h)
Char (:)
Char (*space*)
Char (%)
Char (d)
Char (*space*)

Char (*space*)
Char (B)
Char (a)
Char (l)
Char (a)
Char (n)
Char (c)
Char (e)
Char (:)
Char (*space*)
Char (%)
Char (1)
Char (0)
Char (.)
Char (2)
Char (f)
Char (*newline*)
Quote
Comma
Identifier (month)
Comma
Identifier (bal)
Right_paren
Endstatement
Identifier (bal)
assign
subtraction_op
Identifier (bal)
Identifier (payment)
Addition_op
Decimal_literal (0.015)
Multiplication_op
Identifier (bal)
Endstatement
Identifier (month)
assign
Identifier (month)
Addition_op
Int_literal (1)
Endstatement
Right_bracket
Right_bracket

Dragon: 1.1.4

A compiler that translates a high-level language into another high-level language is called a source-to-source translator. What advantages are there to using C as a target language for a compiler?

Unless the compiler is translating the high-level language into a language that is interpreted, it will eventually have to be compiled down anyway. Translating it to C will make it a compiled langage as C gets compiled into machine code. This allows the program to run much faster than other languages that are being interpreted like JavaScript. It also allows you to catch mistakes that your high-level language may not have picked up on initially like type miss-matches.

Dragon: 1.6.1

For the block-structured C code of Fig. 1.13(a), indicate the values assigned to w,x,y, and z.

```
int w, x, y, z;
int i = 4; int j = 5;
{   int j = 7;
    i = 6;
    w = i + j;
}
x = i + j;
{   int i = 8;
    y = i + j;
}
z = i + j;
```

(a) Code for Exercise 1.6.1

w is assigned to i+j where j is the variable in the block's scope with a value of 7. The i variable is also assigned 6 in the block, so w = 6+7 = 13.

x is assigned to i + j (out of block scope). The i was changed to 6 in the block so x = 6+5=11

y is assigned to i (inner scope = 8) + j (oouter scope = 5) to get 13.
z is assigned 6(outer i)+5(j) = 11.