02242: Program analysis

Technical University of Denmark

First draft

Ibrahim Nemli - s093477
Kim Rostgaard Christensen - s084283
Peter Gammelgaard Poulsen - s093263

October 7, 2013

Abstract

This is the report documenting and describing the work done in the course 02242: Program Analysis.

1 Introduction

Program analysis is the discipline of extracting and deriving information about the structure, and potential behaviour of computer programs. It has many applications, such as; bug finding, optimizations, code reuse, formal validation and model mapping. In this report, we take a closer look at some of the specialized and general theory used for transforming source code text into analysis-friendly structures, and then performing the actual analysis on them.

2 Program representation

The first problems we need to solve is which data structures should be used for our abstract syntax tree, which will serve as the first step of our intermediate representation.

2.1 Abstract syntax tree data structure

We have decided that we could use an unbalanced tree for storage of our abstract syntax tree in our target programming language. Constraining the tree construction will give us the necessary check for syntax errors.
A simple tree-traversal will give us the appropriate unique label numbers.

2.2 Flow graph graph data structure

For the flow graph, we have chosen a network graph data structure. This gives an intuitive way of linking statements to potential paths.

2.3 Program graph data structure

Notes Below

Constructing flow graphs

Functions: Slide 11+12 of from week 2

label(S): The set of nodes of the flow graph S

Init(S): The initial node of the flow graph S. Unique node where the execution of the program starts.

Final(S): The final node of the flow graph S. A set of nodes where the execution of the program may terminate.

Block(S): A set of the blocks/statements in the program under inquisition.

Flow(S): The edges of the flow graph for S. A set of pairs is returned.

Label(S): $\{1, 2, 3, 4, 5, 6\}$

Initial(S): 1

Final(S): $\{6\}$

Blocks(S): { y:=x, z:=1 y>0, z:=z*y, y;=y-1, Y:=0 }

Flow(S): $\{(1, 2), (2, 3), (3, 4), (4, 5), (5, 3), (3, 6)\}$

```
[y:=x]1
[z:=1]2
while [x>0]3 do
    [z:= z*y]4
    [Y:= y-1]5
od
[y:=0]6
```

Figure 1: Source code example

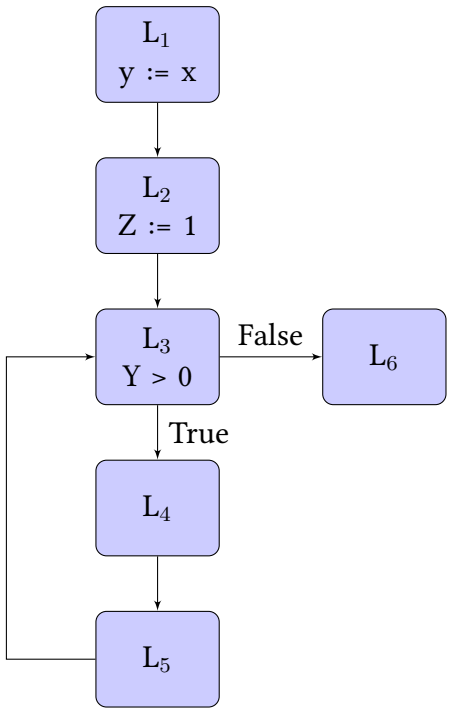

Figure 2: Flow Graph

Constructing program graph

Functions: Slide 14, week 2.
pg_{qs}^{qt}: The program graph for statement S with with initial node being q_s[1], and q_t[2]
Nodes are constructed by the algorithm.
The edges are represented by the tuple (q_s, x, q_T) where q_s and q_t are nodes, and x is and elementary block statement.

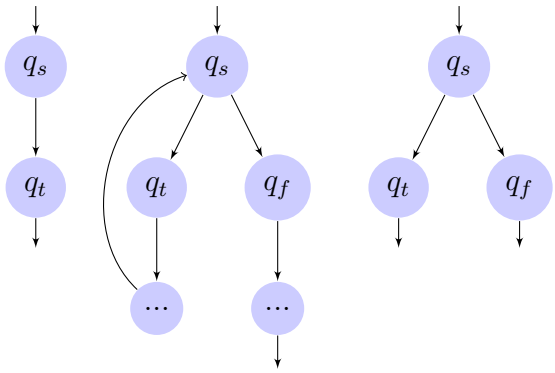

Figure 3: Assignments, loops and branches

Label(S): $\{1, 2, 3, 4, 5, 6, 7\}$

Initial(S): 1

Final(S): $\{7\}$

Blocks(S): $\{[y := x], [z := 1], [y > 0], [z := z * y], [y; = y - 1], [\neg(y > 0)], [Y := 0]\}$

Flow(S): $\{(1, 2), (2, 3), (3, 4), (4, 5), (5, 3), (3, 6), (6, 7)\}$

$a_\bullet, a_\circ$

```
Operator_A_Enum : {Plus, Minus, Divide, Multiply}
```

[1] s for source.
[2] t for target.

Variable_Enum	{ int, Array }
Operator_R_Enum	{Less, Less_Equal, Equal, Not_Equal, Greater, Greater_Equal}
Operator_B_Enum	{And, Or}

Table 1: Enumerations

```
program
[int x]1
[int y]2
[int <]3
[y := x]4
[z := 1]5
while [y>0]6 do
    [z:= z*y]7
    [Y:= y-1]8
od
[y:=0]9
end
```

Figure 4: Slicing code example

Exercise 2 - Program Slicing

(a) Example program

The point of interest is label 8. The result of program slice analysis is; [y:=x]^4, [y:=y-1]^8, [int x]^1.

The point of interest is label 7. The result of program slice analysis is; [y:=x]^4, [z:=1]^5, [z:=z*y]^7, [int z]^3, [int y]^2.

:

int A[n]:

int x	$\text{kill}_{RD}([\text{int x}]^2) = \varnothing$ $\text{gen}_{RD}([\text{int x}]^2) = \{(x, 2)\}$
A[n]	$\text{kill}_{RD}([\text{A[n]}]^2) = \varnothing$ $\text{gen}_{RD}([\text{A[n]}]^2) = \{(A[0], l), ...(A[i-a], l)\}$
A[a_1 := a_2	$\text{kill}_{RD}([\text{A[a}_1\text{]}]^2) = \{$ (A[a_1],l' } $\text{gen}_{RD}([\text{A[n]}]^2) = \{(A[0], l), ...(A[i-a], l)\}$

Table 2: RD Equations

(b) Extending Reaching Definitions Analysis table

(c) Flow graph

(d) Program slice calculation algorithm

Exercise 3

Where

Exercise 4 - Buffer overflow

$$(L, \mathcal{F}, F, E, \iota, \text{f.}) \tag{1}$$

We define our complete lattice to; $(?, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \bot)$, and our interval is defined to: Interval = $\{\bot\} \bigcup \{[Z_2, Z_2] | Z_1 < Z_2, Z_1, Z_2, \in Z' \bigcup \{-\infty, \infty\}\}$, where Z' = $\{Z | \min \leq Z \leq \max\}$ and $-\infty \leq \infty$ minimum $\in Z$
$\sqsubseteq$-Ordering: $\subseteq$ (subset ordering).
$\text{Interval}_1 \sqsubseteq \text{Interval}_2$ if, and only if $\beta(\text{Interval}_1) \subseteq \beta(\text{Interval}_2)$, where $\beta(\bot) = \varnothing$
Least upper bound: $\bigsqcup Y = \bigcup Y$
Greatest lower bound: $\bigsqcap Y = \bigcap Y$
$\bot$: Bottom, least element : $\varnothing$
$\top$: Top, greatest element : $[-\infty, \infty]$

This is a complete lattice, as every subset in the partial ordered set have a least upper bound and a greatest lower bound.
It also holds the Ascending Chain Property, because the lattice has a finite height, and will thus, eventually stabilize.

Internal analysis