

**COMP 621 Analyses and Transformations - Assignment # 3**  
Interprocedural Analysis, Register Allocation and Advanced Analyses  
Due: Tuesday, April 14

## 1 Intra- and Inter-procedural Analysis

Consider the following program:

```
int x,y;
main()
{
    int i,j;
    x = 1;
    y = 4;
    i = x + 1;
    /*-- Point A --*/
    foo(i);
    j = y + 1;
    /*-- Point B --*/
    x = 10;
    bar(i);
    j = x + 1;
    /*-- Point C --*/
}

void foo(int a)
{
    a = a * 2;
    x = a + 1;
    /*-- Point D --*/
}

void bar(int b)
{
    b = b * 2;
    y = b + 2;
    if (x == 10)
        foo(b);
    else
        foo(b+1);
    /*--Point E --*/
}
```

For the above program, give the constant propagation information at the labelled program points, using the following different strategies to estimate the effects of procedure calls.

- a) Conservative Approach: Assume nothing is known about the called procedures and provide the information at program points A, B and C. Clearly state what assumptions you are making for your conservative approach.
- b) Summary Approach: Prepare a summary of each procedure as to which variables can be potentially modified by any call to the procedure (including transitive calls). Use this summary information to get sharper constant information at program points A, B and C.
- c) Full-blown Approach: Precisely estimate the effect of a procedure call, by visiting and analyzing the body of the called function for each call. Provide the constant information at program points A, B, C, D and E. You should tag each piece of dataflow information with a call string (i.e. the complete calling path).

## 2 Register Allocation

Use the following example program.

```
0 { n = read();
1   m = read();
2   if (n > 0)
3     { i = n;
4       sum = 0;
5       while (i != 0)
6         { sum = sum + i;
7           i = i - 1;
8         }
9       write(sum);
10  }
11 else
12   { i = n;
13     prod = 1;
14     while (i != 0)
15       { prod = prod * i;
16         i = i + 1;
17       }
18     write(prod * m);
19   }
20 }
```

- (a) Draw the register interference graph for the example program. Assume that the underlying architecture can use the same register as both a source and destination (i.e. instructions like `MOV R1, R1` and `ADD R1, R2, R1` are allowed). Based on this interference graph, give a minimal register assignment to the program (you may just indicate which register number is assigned to each program variable).
- (b) Is there any program transformation that could be used to reduce register pressure in this program? If so, indicate the transformation.

## 3 Special Topics

Choose any **two** of the following special topics questions to answer. If any question is not clear to you, you may post a query to the google group and the author of the question should provide a clarification.

- (a) **Points-to using BDDs:** It is easy to see how the algorithm described in Figure 6 of *Points-to Analysis Using BDDs* corresponds to standard inference rules for the propagation of the points-to relation. In this exercise, we will examine the implementation of Rule 2:

$$\frac{o_2 \in pt(l) \quad l \rightarrow q.f \quad o_1 \in pt(q)}{o_2 \in pt(o_1.f)}$$

Does the following series of set operations give the same result for *fieldPt* as Lines 2.1-2.3 of the algorithm presented in the original paper?

$$temp1 = \{(v', (o, f)) | \exists v, v'. (v', (v, f)) \in Stores, (v, o) \in pointsTo\}$$

$$fieldPt = \{((o_1, f), o_2) | \exists v. (v, (o_1, f)) \in temp1 \wedge (v, o_2) \in pointsTo\}$$

You may present your findings in terms of an example, using the following relations:

$$\begin{aligned} edgeSet &= \{(b, a), (a, b), (b, a)\} \\ pointsTo &= \{(a, A), (b, B), (c, C), (b, A), (a, B), (c, B), (c, A)\} \\ Loads &= \emptyset \\ Stores &= \{(c, (a, f))\} \end{aligned}$$

- (b) **Program Slicing:** For the following program. What is the static slice for the slicing criterion  $\langle z, 12 \rangle$ ? Be aware that the original lines cannot be changed, only removed. Explain how you computed the slice.

```
int x, y, z, a;
x = 2;
y = x + 5;
z = 20;
a = read();
if (a > 0)
  y = y + y;
else
  z = z + z;
print(y);
print(z);
```

- (c) **Loop dependence analysis:** For the following loop:

```
for (i = 1; i <= 7; i = i+2)
  X[i] = X[8-i];
```

Indicate all the:

- Flow dependences (True dependences)
- Anti-dependences
- Output dependences

- (d) **Array bound checks:** Is the array reference on line L4 safe? Explain your reasoning using the VCG approach.

```
L1: i = j + 4;  
L2: a[i] = .....;  
L3: i = b[20];  
L4: a[j] = .....;
```

Hint: For the statements of the form  $i = j + c$ , add constraint  $i, j \leq c$ , and  $j, i \leq -c$ . For  $a[i] = \dots$  add constraints  $i, a \leq -1$  and  $0, i \leq 0$ . For  $i = \dots$  Detach  $i$  and look at the remaining VCG.

- (e) **Dynamic versus Static type-checking:** There are pros and cons for dynamic type-checking versus static type-checking. Discuss the cons, both in terms of execution speed and programmer development time.