HW 10: Dataflow Analysis Foundations

CSCI 434T
Fall, 2007

Overview

We continue to examine dataflow analysis and optimization. Last week, we saw several examples of dataflow problems, and looked at how they could be used to optimize a program. We now take a step back and develop a general framework in which to describe and reason about dataflow analysis. With this framework, it is easier to reason about the correctness and precision of a particular analysis, and to formulate new analyses. You will do both of these tasks in the following problems. The first question picks on last week's themes, in the context of IC TAC instructions; the next few cover the lattice theory underlying dataflow analysis; and the last two ask you to design new dataflow analyses to compute specific information about a program.

Readings

- Dragon 9.3 – 9.5.2
- Dragon 9.5.3 – 9.5.5 *(Optional. A more advanced analysis- have a look at problem 9.5.1 if you do this reading.)*

Exercises

1. This question covers a few more details of basic dataflow analysis problems, in the context of IC.

   (a) Describe the *e_gen*, *e_kill*, *def*, and *use* sets for each of the following instructions:

   - `t = a[i]`
   - `b[j] = s`
   - `x = p.g`
   - `o.f = y`
   - `z = q.m(r)`

   (b) Optimize the TAC generated for the following code snippets. You are free to use any optimization we have talked about, but pay particular attention to common subexpression elimination, copy propagation, and dead code elimination.

   i.

```
class A {
   int f,g;
   A moo(int z) { ...  }
   int f(A a) {
     int v = a.f;
     int w = a.g;
     A b = a.moo(v);
     b.f = w;
     z = (a.f + b.f) * a.g;
     return z;
   }
}
```

```
int f(A a) {
   v = a.f;
   w = a.g;
   b = a.moo(v)
   b.f = w;
   z = a.f;
   t1 = b.f;
   z = z + t1;
   t2 = a.g;
   z = z * t2;
}
```

ii.

```
void f(int[] b) {
   int i,j,x;

   ...

   x = b[i] + b[j];
   b[j] = b[i] + b[j];
   b[i] = b[i] + b[j];
}
```

```
t1 = b[i];
t2 = b[j];
x = t1 + t2;

t3 = b[i];
t4 = b[j];
t5 = t3 + t4;
b[j] = t5;

t6 = b[i];
t7 = b[j];
t8 = t6 + t7;
b[i] = t8;
```

2. Given $P = \{\text{red}, \text{blue}, \text{yellow}, \text{purple}, \text{orange}, \text{green}\}$, let us define the partial order $\sqsubseteq$ as follows:

$$\begin{aligned} \text{red} &\sqsubseteq \text{purple} \\ \text{blue} &\sqsubseteq \text{purple} \\ \text{yellow} &\sqsubseteq \text{orange} \\ \text{red} &\sqsubseteq \text{orange} \\ \text{blue} &\sqsubseteq \text{green} \\ \text{yellow} &\sqsubseteq \text{green} \\ p &\sqsubseteq p \quad \text{for all } p \in P \end{aligned}$$

   (a) Draw the diagram (as in Dragon Figure 9.22) for P.

   (b) Would we still have a partial order if I added $\text{purple} \sqsubseteq \text{yellow}$? Why or why not? Would we still have a partial order if I added $\text{purple} \sqsubseteq \text{yellow}$ and $\text{yellow} \sqsubseteq \text{red}$? Why or why not?

   (c) Now add two additional colors to P:

$$\begin{aligned} \text{white} &\sqsubseteq p && \text{where } p \in \{\text{red}, \text{blue}, \text{yellow}\} \\ p &\sqsubseteq \text{black} && \text{where } p \in \{\text{purple}, \text{orange}, \text{green}\} \end{aligned}$$

      i. The resulting structure is a semi-lattice. Draw it.

      ii. If we add $\text{blue} \sqsubseteq \text{red}$, do we still have a lattice?

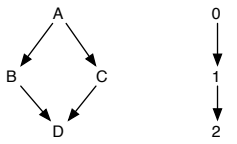
      iii. Compute all lower bounds for $\{\text{black}, \text{purple}, \text{orange}\}$.

      iv. Compute the greatest lower bound for $\{\text{black}, \text{purple}, \text{orange}\}$.

      v. Which of the following functions $f_i : P \to P$ are monotone?

- $f_1(p) = \begin{cases} \text{white} & \text{if } p \in \{\text{red}, \text{blue}, \text{yellow}\} \\ p & \text{otherwise} \end{cases}$
- $f_2(p) = \begin{cases} \text{yellow} & \text{if } p \in \{\text{red}, \text{blue}, \text{green}\} \\ p & \text{otherwise} \end{cases}$
- $f_3(p) = \begin{cases} \text{orange} & \text{if } p \text{ contains the letter } a \\ \text{blue} & \text{otherwise} \end{cases}$

3. (a) Draw the product of the following two lattices L and N:



(b) Suppose V is the set of all subsequences of "moo" ({ moo, mo, oo, m, o, ϵ }) and let $x \wedge y$ be the longest common subsequence of x and y. Draw the lattice for V. Consider the function $f : V \to V$, where $f(x)$ is x with all *o*'s removed. Is f distributive? Justify in one or two sentences.

4. Dragon 9.3.3

5. Design an optimization to remove redundant run-time `null` pointer checks from IC programs. For simplicity, you may assume that each basic block contains a single TAC instruction. Specifically:

   (a) Design a dataflow analysis by describing the following:
   - The direction D (forward/backward) of your analysis.
   - The domain V of data flow values.
   - The meet operation $\wedge$. Describe the order $\leq$ induced on V by your meet operator.
   - The set of transfer functions F, where $f_I \in F$ is the transfer function for TAC instruction I. You only need to consider the following forms: `x = y`, `x = new C()`, `check_null x`, and `x = null`.
   - v, the initial value for either OUT[ENTER] or IN[EXIT], depending on whether your analysis is forward or backward.

   You may wish to use Figure 9.21 as a guide.

   (b) Is your framework monotone? Explain why. Is your framework distributive? Explain.

   (c) Describe how to use the results of your dataflow analysis to optimize a program.

   (d) Show the results of applying the analysis and optimization to the following:

```
y = new Object();
check_null y;
z = w;
while (...)  {
   check_null y;
   check_null z;
}
if (...)  {
   y = k;
   check_null y;
   check_null z;
}
check_null y;
x = y;
check_null x;
```

   (e) In Dragon, page 629, the authors claim that for a lattice-theoretic partial order $\leq$:
   - Any answer greater than IDEAL is incorrect.
   - Any answer smaller than or equal than IDEAL is conservative, *i.e.* safe.

   Explain in one or two sentences why your analysis never yields incorrect results. Is your analysis *strictly* conservative in the sense that it sometimes computes answers strictly smaller than IDEAL? If not, explain why in one or two sentences. If it is strictly conservative, write a short program for which your analysis computes a different value than IDEAL. What are the consequences of being conservative in this case?

6. We want to design a dataflow analysis to compute ranges for integer variables in the program. For this, we extend the set N of integer numbers with plus and minus infinity:

$$\mathbb{N}^* = \mathbb{N} \cup \{+\infty, -\infty\}$$

such that $-\infty < n$ and $n < +\infty$ for any integer number n. We then use a lattice over the set

$$L = \{[l, u] \mid l, u \in \mathbb{N}^* \text{ and } l \leq u\} \cup \{\top\}$$

(a) Explain what the element $\top$ represents and why we need it.

(b) Define the partial order and the meet operator $\wedge$ for elements in this lattice (including $\top$).

(c) Using this lattice to compute ranges of variables will fail. Explain why.

(d) To solve the problems from the last part, we define a lattice

$$L' = \{[l, u] \mid l, u \in \{-\infty, -1, 0, 1, +\infty\} \text{ and } l \leq u\} \cup \{\top\}$$

(with the same partial order as before) and build a dataflow analysis that computes ranges in L'. Show the transfer functions for assignments of constants `x = n` and arithmetic operations `x = y + z` and `x = y * z`, where `x`, `y`, and `z` are program variables and `n` is an integer constant.

(e) Using the revised lattice and your transfer function, show how the dataflow analysis works for the following program:

```
x = 0;
while (...) {
  y = x;
  if (...) {
    x = x+1;
  } else {
    y = y-1;
  }
}
```

(f) Assuming programs consist only of the three kinds of statements shown above, does this analysis always yield the same result as the Meet-Over-Paths solution? If yes, show why. If not, show a counter-example program and indicate the MOP and dataflow solutions.