Task4 - SCSR

Andrea Munarin (879607), Simone Jovon (882028)

March 2023

1 Exercise 1

Consider the procedures main, ralph and joe, as seen in class:

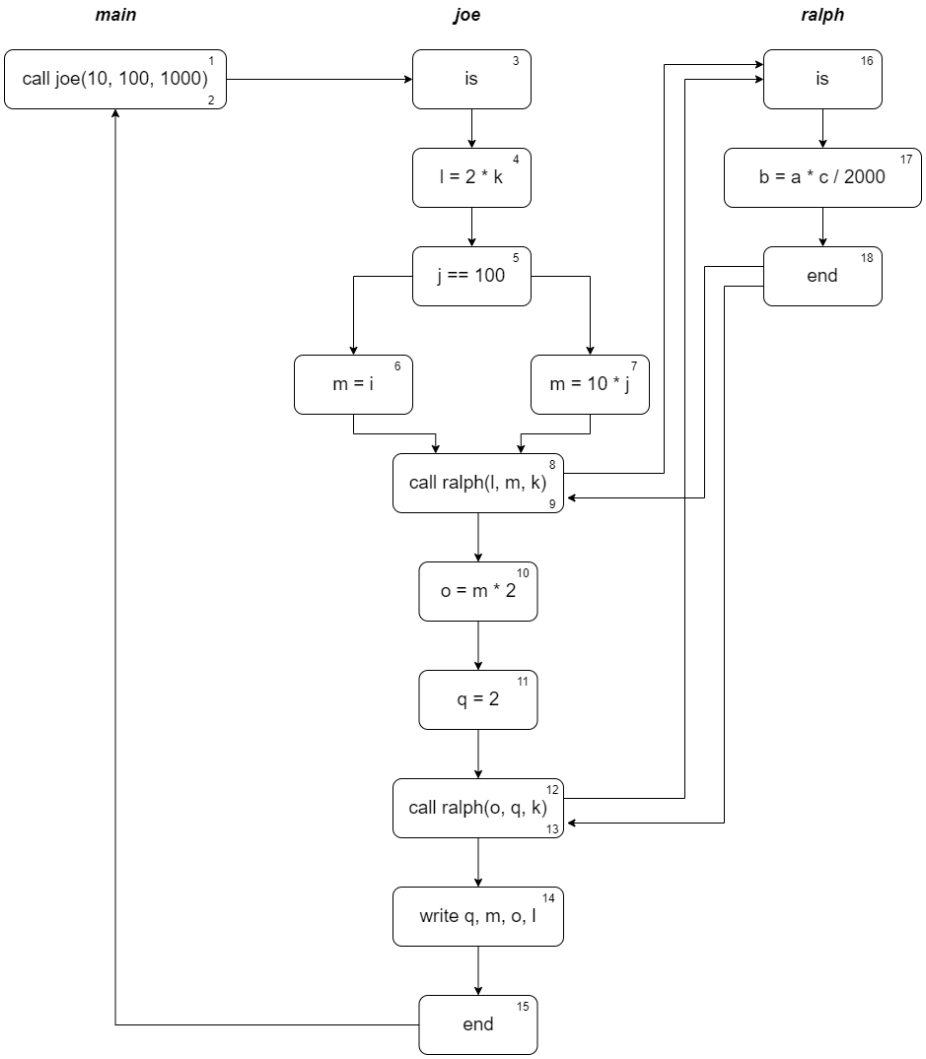
```
procedure main
    call joe(10, 100, 1000)

procedure joe(i, j, k)
    l <- 2 * k
    if(j = 100)
        then m <- 10 * j
        else m <- i
    call ralph(l, m, k)
    o <- m * 2
    q <- 2
    call ralph(o, q, k)
    write q, m, o, l

procedure ralph(a, b, c)
    b <- a * c / 2000
```

SOLUTION

1. *Build the interprocedural control flow graph*



2. *Compute the inter-procedural constant propagation analysis, formalizing the transfer functions associated to entering in and exiting from a called procedure*
We need to extend the constant propagation specification in order to work also with procedure class. In order to define it let introduce this notation:

- `V` = {Set of all variables present in the entire program}
- `map[B]` = {(x, y) such that in block B there is a procedure call in which the argument y is passed to parameter x}

Now, using the above notation and def, gen set (from class lectures) we provide the recursive equation. The crucial equations are those that formalize entering and exiting from a procedure:

- $CP_{in}(l) = \emptyset$ if p is the initial point in the graph
- $CP_{in}(l) = \bigcup\{CP_{out}(l')|(l', l) \in For(l'; l) \in F\}$
- $CP_{out}(l) = gen_{RD}(l) \cup (CP_{in}(l) \setminus kill_{RD}(l))$
- $CP_{out}(l_c) = CP_{map}(l) \cup \{(x', k)|(x, k) \in CP_{in}(l_c) \wedge (?, x) \notin CP_{map}(l)\}$
  - Consider two sets in our analysis. In our instruction `map`, we store the argument value passed to our parameter variable. Next, we create a duplicate of the output set by renaming all the variables, thus avoiding scenarios where a variable with the same name appears in two distinct functions with different scopes.
  - It's important to note that if our program includes nested function calls, the variable's scope is recorded by appending a new tick mark each time. For instance, if our output set contains (x', 10), and a new function call is made, we obtain (x", 10).
- $CP_{out}(l_r) = \{(x, v)|(y, x) \in CP_{map}(l) \wedge x \in \texttt{V} \wedge (y, v) \in CP_{in}(l_r)\} \cup \{(x, k)|(x', k) \in CP_{in}(l_r)\}$
  - To return to our previous environment, we apply the computed function value to our arguments (if they are variables rather than constant values). Afterward, we rename the variables to their previous names, since we are returning to a previous scope. It's worth noting that we don't retain any information about the variables defined in the function, since they are destroyed at the end of the scope.

One last thing important to formalize is the all the possible path that the program can traverse. Then we will proceed through the analysis using this set: `inter-flow`={(1, 3, 15, 2), (8, 16, 18, 9), (12, 16, 18, 13)}

Let's now compute the inter-procedural constant propagation analysis for our piece of code:

- $CP_{in}(1) = \emptyset$
  $CP_{out}(1) = \{$(i, 10), (j, 100), (k, 1000)$\}$
  - We start from the empty set.
  - Then we need to apply the transfer function related to a procedure call $CP_{out}(l_c)$. We store the argument/parameter couple in our set.
- $CP_{in}(3) = \{$(i, 10), (j, 100), (k, 1000)$\}$
  $CP_{out}(3) = \{$(i, 10), (j, 100), (k, 1000)$\}$
- $CP_{in}(4) = \{$(i, 10), (j, 100), (k, 1000)$\}$
  $CP_{out}(4) = \{$(i, 10), (j, 100), (k, 1000), (l, 2*k)$\}$
  - Here we can apply first Rule 1 and then Rule 2[1] (defined in class to reduce the value during the constant propagation analysis). So it can be obtained the following result: $l = 2 * 1000 \rightarrow l = 2000$
- $CP_{in}(5) = \{$(i, 10), (j, 100), (k, 1000), (l, 2000)$\}$
  $CP_{out}(5) = \{$(i, 10), (j, 100), (k, 1000), (l, 2000)$\}$
  - For constant propagation we know that j is always equal to 100 when reaching definition 5, so we are sure that block 6 is never reached
- $CP_{in}(7) = \{$(i, 10), (j, 100), (k, 1000), (l, 2000)$\}$
  $CP_{out}(7) = \{$(i, 10), (j, 100), (k, 1000), (l, 2000), (m, 10*j)$\}$
  - With Rule 1 and Rule 2 we obtain: $m = 10 * 100 = 1000$
- $CP_{in}(8) = \{$(i, 10), (j, 100), (k, 1000), (l, 2000), (m, 1000)$\}$
  $CP_{out}(8) = \{$(c, 1000), (a, 2000), (b, 1000), (i', 10), (j', 100)$\}$
  - Within the procedure call we need to substitute the name of the variable using the $CP_{map}(8) = \{(a, l), (b, m), (c, k)\}$. We also have to rename the variable not visible in this scope: $i \rightarrow i' \wedge j \rightarrow j'$
- $CP_{in}(16) = \{$(c, 1000), (a, 2000), (b, 1000), (i', 10), (j', 100)$\}$
  $CP_{out}(16) = \{$(c, 1000), (a, 2000), (b, 1000), (i', 10), (j', 100)$\}$
- $CP_{in}(17) = \{$(c, 1000), (a, 2000), (b, 1000), (i', 10), (j', 100)$\}$
  $CP_{out}(17) = \{$(c, 1000), (a, 2000), (b, a*c/2000), (i', 10), (j', 100)$\}$
  - We apply Rule 1 two times and then Rule 2: $b = 2000 * c/2000 = 2000 * 1000/2000 = 1000$
- $CP_{in}(18) = \{$(c, 1000), (a, 2000), (b, 1000), (i', 10), (j', 100)$\}$
  $CP_{out}(18) = \{$(c, 1000), (a, 2000), (b, 1000), (i', 10), (j', 100)$\}$

[1] Remember that we can apply these rules only because we have only one instance of k present in our set

- $CP_{in}(9)$ = {(c, 1000), (a, 2000), (b, 1000), (i', 10), (j', 100)}
  $CP_{out}(9)$ = {(k, 1000), (l, 2000), (m, 1000), (i, 10), (j, 100)}
  - We restore the variable using the return recursive equation.

From now on we proceed with the same reasoning for all the blocks:

- $CP_{in}(10)$ = {(k, 1000), (l, 2000), (m, 1000), (i, 10), (j, 100)}
  $CP_{out}(10)$ = {(k, 1000), (l, 2000), (m, 1000), (i, 10), (j, 100), (o, m*2)}
  - Rule1 + Rule2 = (o, 2000)
- $CP_{in}(11)$ = {(k, 1000), (l, 2000), (m, 1000), (i, 10), (j, 100), (o, 2000)}
  $CP_{out}(11)$ = {(k, 1000), (l, 2000), (m, 1000), (i, 10), (j, 100), (o, 2000), (q, 2)}
- $CP_{in}(12)$ = {(k, 1000), (l, 2000), (m, 1000), (i, 10), (j, 100), (o, 2000)}
  $CP_{out}(12)$ = {(c, 1000), (l', 2000), (m', 1000), (i', 10), (j', 100), (a, 2000), (b, 2)}
- $CP_{in}(16)$ = {(a, 2000), (b, 2), (c, 1000), (l', 2000), (m', 1000), (i', 10), (j', 100)}
  $CP_{out}(16)$ = {(a, 2000), (b, 2), (c, 1000), (l', 2000), (m', 1000), (i', 10), (j', 100)}
- $CP_{in}(17)$ = {(a, 2000), (b, 2), (c, 1000), (l', 2000), (m', 1000), (i', 10), (j', 100)}
  $CP_{out}(17)$ = {(a, 2000), (b, a*c/2000), (c, 1000), (l', 2000), (m', 1000), (i', 10), (j', 100)}
  - Rule1 + Rule2 = (b, 1000)
- $CP_{in}(18)$ = {(a, 2000), (b, 1000), (c, 1000), (l', 2000), (m', 1000), (i', 10), (j', 100)}
  $CP_{out}(18)$ = {(a, 2000), (b, 1000), (c, 1000), (l', 2000), (m', 1000), (i', 10), (j', 100)}
- $CP_{in}(13)$ = {(a, 2000), (b, 1000), (c, 1000), (l', 2000), (m', 1000), (i', 10), (j', 100)}
  $CP_{out}(13)$ = {(o, 2000), (q, 1000), (k, 1000), (l, 2000), (m, 1000), (i, 10), (j, 100)}
- $CP_{in}(14)$ = {(o, 2000), (q, 1000), (k, 1000), (l, 2000), (m, 1000), (i, 10), (j, 100)}
  $CP_{out}(14)$ = {(o, 2000), (q, 1000), (k, 1000), (l, 2000), (m, 1000), (i, 10), (j, 100)}
  - So the program will write: 1000, 1000, 2000, 2000

- $CP_{in}(15) = \{$(o, 2000), (q, 1000), (k, 1000), (l, 2000), (m, 1000), (i, 10), (j, 100)$\}$
  $CP_{out}(15) = \{$(o, 2000), (q, 1000), (k, 1000), (l, 2000), (m, 1000), (i, 10), (j, 100)$\}$
- $CP_{in}(2) = \{$(o, 2000), (q, 1000), (k, 1000), (l, 2000), (m, 1000), (i, 10), (j, 100)$\}$
  $CP_{out}(2) = \emptyset$