Compilers (L.EIC026)

Spring 2022

Faculdade de Engenharia da Universidade do Porto
Departamento de Engenharia Informática

Second Test (Make-Up Grade Exam)
With Solutions

July 06, 2022 from 09:30 to 10:30

Please label all pages you turn in with your name and student number.

Name: ______________________________ **Number:** ________

Grade:

Problem 1 [10 points]:
Problem 2 [20 points]:
Problem 3 [20 points]:

Total:

INSTRUCTIONS:

1. **This is a close-book and close-notes exam**.
2. Please identify all the pages where you have answers that you wish to be graded. Also make sure to label each of the additional pages with the problem you are answering.
3. Use black or blue ink. No pencil answers allowed.
4. Staple or bind additional answer sheets together with this document to avoid being misplaced or worse, lost. Make sure this cover page is stapled at the front.
5. Please avoid laconic answers so that we can understand you understood the concepts being asked.

Problem 1. Run-Time Environments [10 points]

In the PASCAL language one can define functions and procedures that are local to other procedures, *i.e.,* they are only visible within the scope of that immediately enclosing procedure. This is analogous to the use of nested blocks in the C language. For example, in the code below procedure `f3` is only visible inside the code of procedure `f2`. But neither inside the code of `f1` nor inside the procedure `main`.

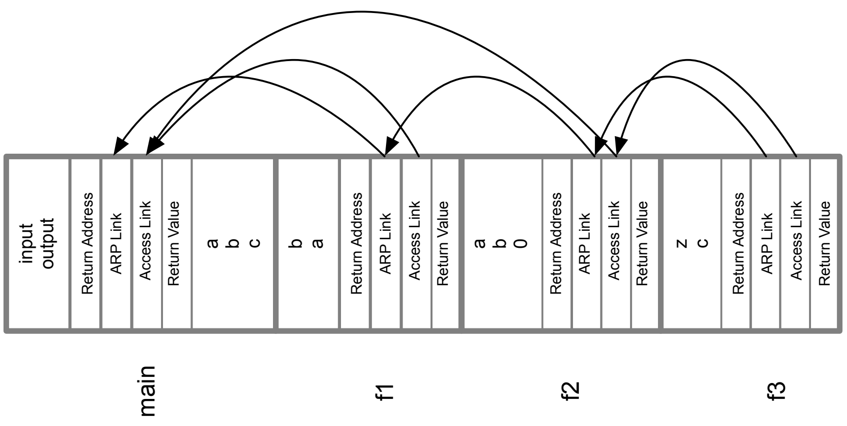
```
01:  procedure main
02:    integer a, b, c;
03:    procedure f1(a,b);
04:      integer a, b;
05:      call f2(0,b,a);
06:    end;
07:    procedure f2(x,y,z);
08:      integer x, y, z;
09:      procedure f3(m,n);
10:          integer m, n;
11:          ...
12:      end;
13:      procedure f4(m,n);
14:        integer m, n;
15:         ...
16:      end;
17:      call f3(y,z);
18:      call f4(c,x);
19:    end;
20:  ...
21:  call f1(a,b);
22:  end;
```

For this code answer the following questions:

a) On line `17` and `18`, which variables are used as the actual values of the parameters of both calls to `f3` and `f4` respectively.
b) Draw (using simplified Activation Records layout) when the control flow of the program reaches line `11` (i.e., inside procedure `f3`). Draw the links regarding the ARs and the Access Links used to access non-local variables. Explain how the compiler can generate code to access the local variable x of the procedure `f2` in the body of procedure `f3`.

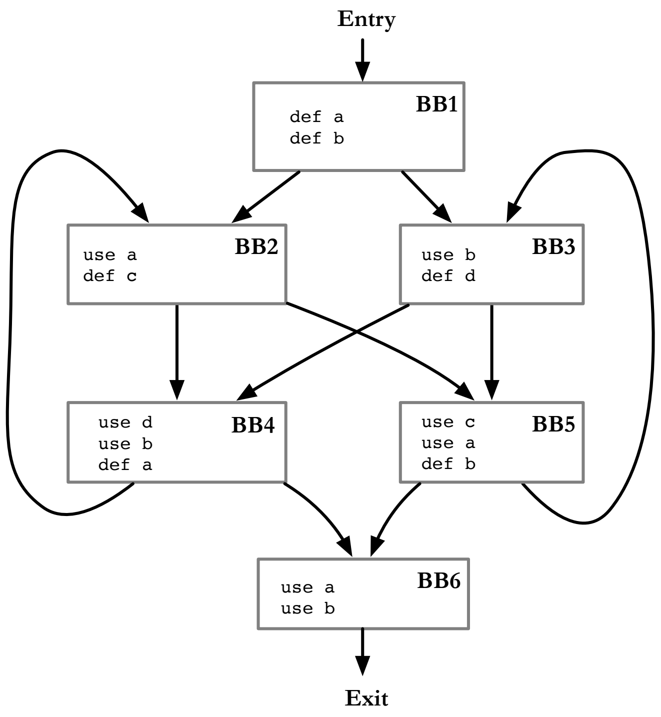
Answers:

a) The arguments for the call to procedure `f3` on line `17` includes variables `y` and `z` which are local variables to procedure `f2`. The arguments for the call to procedure `f4` on line `18` includes variables `c` and `x` which are local variables to `main` and procedure `f2`.
b) See figure below where only the Access Link and ARP Link are shown.



Problem 2. Control-Flow Analysis and Register Allocation [20 points]

Consider the CFG as shown below corresponding to a procedure with local variables `a`, `b`, `c` and `d`.



For this code determine the following:

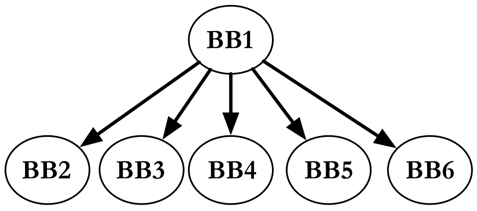
a. [05 points] Dominator tree and the natural loops in this code (if any) along with the corresponding back edge(s).

b. [10 points] Determine the DU-chains and the corresponding webs for the variables `a`, `b`, `c` and `d`, as well as the corresponding interference graph.

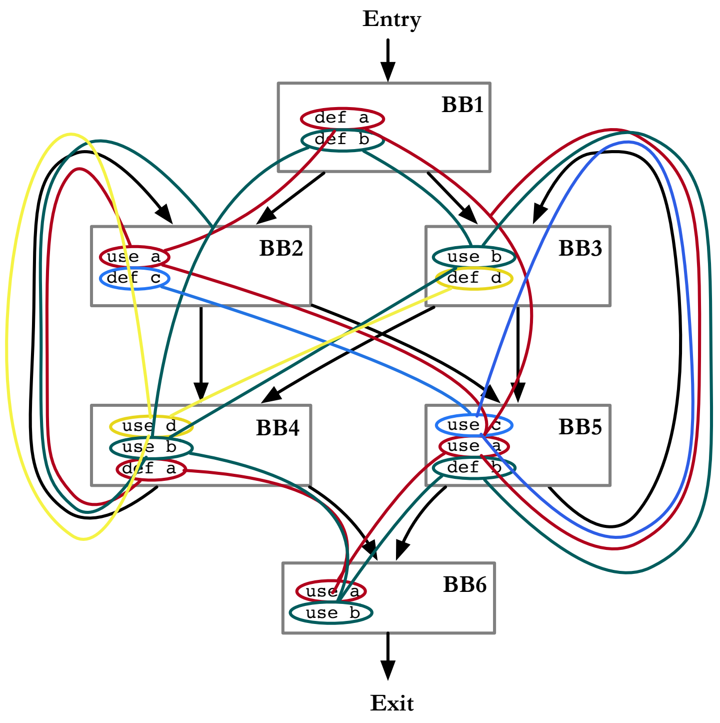
c. [05 points] Can you color the resulting interference graphs with 2 colors? If you cannot, suggest and present the rationale for a strategy that will reduce the connectivity of the corresponding interference graph so that it becomes 2-colorable.

Answers:

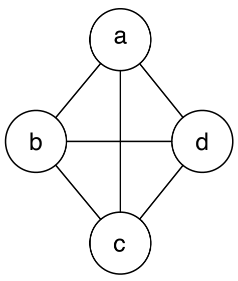
a. The dominator tree is as shown below. As can be seen there is no edge whose node pointed to by the head dominates the node pointed to by the tail. As such there are no natural loops in this CFG.



b. The DU-chains for all the variables a and b cover the entire CFG. for variables c and d, they encompass the basic blocks 2, 3 and 4 for d and 2, 3, 5 for c. As such the interference graph is as shown below.



CFG with Variables Webs

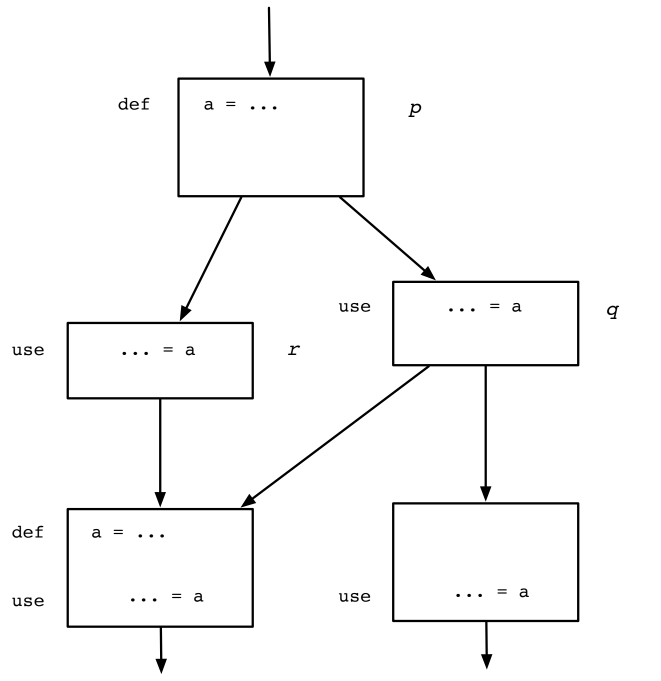


Interference Graph

As it can be seen this interference graph includes a clique of size 4 and as such it cannot be colored with less than 4 colors. One strategy would be not to assign registers to the variables c and d, and thus assign the only two registers to the variables a and b. This is based on the notion that the number of references to these two variables is possibly smaller than the number of references to variables a and b. This rationale is based on the first "global allocation algorithm" and if explicit memory references are inserted, then, ideally in terms of the dynamic execution of the code, the number of memory accesses is smaller than they would be if we were to choose variables a and b.

Problem 3: Iterative Data-Flow Analysis [20 points]

In class we discussed the Live-Variable Analysis problem, where one variable is said to be live at program point *p*, if its value is used along some control-flow path from *p*. In other word, the value of the variable at p can be used in another point *q* without being possibly redefined. In the example below, we say that the variable *a* is live at *p* because there is a control-flow path starting at *p* where the current value of *a* is still possibly used (in this case at *q*). Conversely, the same variable is dead at *r* since its value is no longer used beyond that point as the variable *a* is redefined in all the control-flow paths beyond *r*.



Regarding this data-flow analysis problem answer the following:

a. [05 points] What is the set of values in the corresponding lattice and the initial values?

b. [05 points] What is the direction of the problem, backwards or forward and why?

c. [05 points] What is the basic block transfer function for this data-flow problem, i.e., the GEN and KILL and the equations the iterative approach needs to solve?

d. [05 points] At control-flow merge points, what is the meet operator, and why?

Answers:

a. Lattice values consist of unordered sets of variables. All blocks initialize their IN values to the empty set.

b. A possible formulation of this problem defines the input values as a function of the output values of each basic block, so the direction is backwards. The information that a variable is still live can be propagated backwards in the control flow revealing that in the forward direction a given variable is still needed. This need is killed or eliminated at the point where the variable is defined. Before that definition, the variable is in fact dead. On merge of control-flow paths (backwards, of course) a variable is alive if at least in one of the control-flow paths the variable is alive, which suggests that the merge function is the union of the IN sets of the successors of a basic block.

c. As suggested in the description, the transfer function (either at the basic block level or at the instruction level) is defined by the two equations below.

$$\text{OUT}(B) = \cup_{s\ \text{succ}(B)}\ \text{IN}(s)$$
$$\text{IN}(B) = \text{Use}(B) \cup (\text{OUT}(B) - \text{Def}(B))$$

d. The merge is the Union, since the problem clearly states that a variable is alive if there is at least one path along which the value of a given definition of the variable can still be used. Notice, that by initializing the IN values to the empty sets and using the Union function, the solutions of the equations can only grow and are limited by the size of the universe set that includes all the variables in the program under consideration.