DCC888 – Range Analyses

Name: _____ ID: _____

1. This question refers to the program below, which is typically found in the partition function used in quicksort:

```
0.  e = 9
1.  b = 0
2.    while b < e:
3.      b = b + 1
4.      e = e - 1
5.    if b < e:
6.      return True
7.    else
8.      return False
```

   (a) Draw the control flow graph of the function `partition`. You do not need to assume that the program is in static single assignment form. Assume a three-address assembly, with the following instructions:

   - Addition: $v_0 = v_1 + v_2$
   - Subtraction: $v_0 = v_1 - v_2$
   - Conditional branch: if $a < b$ goto L
   - Unconditional branch: goto L
   - Return statement: return v

   (b) Convert the CFG of the previous question to the **extended** static single assignment form. Remember to rename variables after the conditional. You will need two new instructions:

   - Phi-functions: $v = \phi(v_0, v_1)$

- Intersections: $v = v_0 \sqcap [l, u]$

Mark the futures present in intersections with the string **ft**.

(c) Draw the constraint graph for the CFG seen in the previous question. Indicate the strongly connected components in this graph by writing these sets next to the graph.

(d) Show the result of the growth analysis when applied on the constraint graph that you drew before. The result of the growth analysis is a table $I[V]$, which gives the intervals computed for each constraint variable V. You might need to widen, to avoid infinity iterations. The widening operator is given below, assuming that you are interpreting the instruction Y, and that its abstract interpretation evaluates to $e(Y)$:

$$I[Y] = \begin{cases} \text{if } I[Y] = [\bot, \bot] \text{ then } e(Y) \\ \text{elif } e(Y)_\downarrow < I[Y]_\downarrow \text{ and } e(Y)_\uparrow > I[Y]_\uparrow \text{ then } [-\infty, \infty] \\ \text{elif } e(Y)_\downarrow < I[Y]_\downarrow \text{ then } [-\infty, I[Y]_\uparrow] \\ \text{elif } e(Y)_\uparrow > I[Y]_\uparrow \text{ then } [I[Y]_\downarrow, \infty] \end{cases}$$

We let $[l, u]_\downarrow = l$ and $[l, u]_\uparrow = u$. We let $\bot$ denote non-initialized intervals, so that $[\bot, \bot] \sqcup [l, u] = [l, u]$. This operation only happens at ϕ-nodes, because we evaluate constraints in topological order.

(e) Show the result of future resolution for this program [1]. Future resolution follows the rules below:

$$\frac{Y = X \sqcap [l, \mathbf{ft}(V) + c] \qquad I[V]_{\uparrow} = u}{Y = X \sqcap [l, u + c]} \quad u, c \in \mathbb{Z} \cup \{-\infty, \infty\}$$

$$\frac{Y = X \sqcap [\mathbf{ft}(V) + c, u] \qquad I[V]_{\downarrow} = l}{Y = X \sqcap [l + c, u]} \quad l, c \in \mathbb{Z} \cup \{-\infty, \infty\}$$

(f) Show the result of the narrowing analysis after future resolution [2].

(g) Does range analysis let you infer that the conditional at lines 5-8 of our example is redundant? Explain your answer.

[1] Technically, growth analysis, future resolution and narrowing are performed for each strongly connected component, a component at a time. So, assume, in this exercise, that you already have tight ranges for the previous component, when solving futures for the next one

[2] Again, assume that you have tight ranges for a component, when doing narrowing on the next component

2. In this exercise we will implement an analysis that infers prefixes of strings. We will be using a very simple language, that allows concatenation of strings, and indexing. The primitive operations available in this language are given in the figure below (Left):

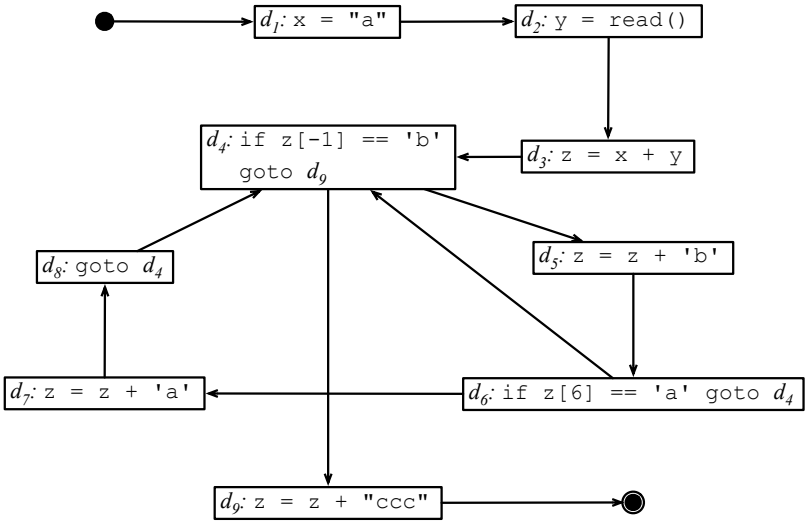
`x = "s"`:
assigns string s to variable x

`s[i]`:
reads the i-th index of string s

`s[-i]`:
reads the cell of s at position 'length(s) - i'

`x = y + z`:
appends string z at the end of string y, and assigns the resulting string to variable x

`read()`:
returns an unknown string



(a) Considering the program on the right, what will be the value associated with variable z, after we execute label d_9?

(b) The goal of our analysis is to approximate the string that is associated with a variable. Hence, we bind to each string s and abstract state $[\![s]\!] = c_0c_1 \ldots c_n\bullet$, where each c_i is an ASCII character, and $\bullet$ represents any occurrence of zero or more characters. We also have a special character, _, which represents exactly one occurrence of any character. Draw the lattice that supports this analysis.

(c) Where is information acquired in this analysis, considering our target programming language, with array statements and control flow primitives?

(d) Explain how is the meet operator used in your analysis. Remember that if $a < b$, then $a \wedge b = a$.

(e) Design a suite of transfer functions that we can use to implement the analysis.

(f) The lattice for this analysis is naturally infinite, as a string can be infinitely long. We need to devise a widening operator that we can apply on the abstract states of the variables, or our abstract interpretation might not terminate. Design a widening operator, and explain when it should be applied.

(g) The widening operator may add too much imprecision onto our analysis. We need a narrowing operator, that can take advantage of the conditional tests to give us back some information. Design this operator, and explain when it should be invoked.

(h) Show the results that your analysis produces for our example program.