ECE/CS 5544 Assignment 2 Part 1

Akhilesh Marathe, Alexander Owens

DataFlow.cpp:

DataFlow.cpp is used as the “driver” code, i.e where most of the initialization and prep takes place. We also define the meet, and other utility functions here. Since the direction and meet operators are dependent on the type of analysis performed, we take them as inputs for the objects of classes derived from DataFlow. Here, we tried to use boolean arguments, but due to a multitude of errors, decided to fall back on enums. We initialize the output with the boundary block based on the direction of flow, with the traversal order stored as well.

Analysis	Direction	Meet Operator
Available Expressions	Forward	Intersection
Reaching Definitions	Forward	Union
Liveness	Backward	Union

Available Expressions, Reaching Definitions and Liveness:

While the premise (in code) is similar for all 3 with DataFlow being the driver code, the actual transfer functions, direction, and meet operators differ according to the table given above.

- Available Expressions:

To perform available expression analysis, we iterate over each instruction in each block, and check if any of the previous operands has been overwritten by the current instruction. If it has, the operand is killed, and thus its corresponding instruction is killed as well. This process continues till each block has been visited, and the output stored in the transferVar struct. Later, an intersection is performed to achieve the resultant available expressions at the end.

```
FUNCTION :: main
[BB Name]entry
IN[entry]: { }
OUT[entry]: {%argc + 50 ,%add + 96 }
Gen[entry]: {%argc + 50 ,%add + 96 }
Kill[entry]: {%add + 96 ,%add - 50 ,96 * %add ,%add + 50 }

[BB Name]if.then
IN[if.then]: {%argc + 50 ,%add + 96 }
OUT[if.then]: {%argc + 50 ,%add + 96 ,%add - 50 ,96 * %add }
Gen[if.then]: {%add - 50 ,96 * %add }
Kill[if.then]: { }

[BB Name]if.else
IN[if.else]: {%argc + 50 ,%add + 96 }
OUT[if.else]: {%argc + 50 ,%add + 96 ,96 * %add ,%add + 50 }
Gen[if.else]: {96 * %add ,%add + 50 }
Kill[if.else]: { }

[BB Name]if.end
IN[if.end]: {%argc + 50 ,%add + 96 ,96 * %add }
OUT[if.end]: {%argc + 50 ,%add + 96 ,96 * %add ,50 - 96 ,%sub4 + %f.0 }S
Gen[if.end]: {50 - 96 ,%sub4 + %f.0 }
Kill[if.end]: {%sub4 + %f.0 }

```

- Reaching Definitions:

Similar to Available Expressions, here too, we traverse in the forward direction, over each block. We iteratively check the validity of each definition, and add it to the kill set if false. We use a union operator here, in contrast to available expressions.

```
FUNCTION :: test
BB Name
IN[]: {i32 %0 | i32 %1}
OUT[]: {i32 %0 | i32 %1 | %3 = sub nsw i32 %0, 1}
Gen[]: {%3 = sub nsw i32 %0, 1}
Kill[]: {%3 = sub nsw i32 %0, 1}

BB Name
IN[]: {i32 %0 | i32 %1 | %3 = sub nsw i32 %0, 1 | %.01 = phi i32 [ %1, %2 ], [ %7, %11 ] | %.0 = phi i32 [ 0, %2 ], [ %12, %11 ] |
  %5 = icmp slt i32 %.0, 10 | %7 = sub nsw i32 %.01, 2 | %8 = icmp sgt i32 %7, 4 | %12 = add nsw i32 %.0, 1}
OUT[]: {i32 %0 | i32 %1 | %3 = sub nsw i32 %0, 1 | %.01 = phi i32 [ %1, %2 ], [ %7, %11 ] | %.0 = phi i32 [ 0, %2 ], [ %12, %11 ] |
  %5 = icmp slt i32 %.0, 10 | %7 = sub nsw i32 %.01, 2 | %8 = icmp sgt i32 %7, 4 | %12 = add nsw i32 %.0, 1}
Gen[]: {%.01 = phi i32 [ %1, %2 ], [ %7, %11 ] | %.0 = phi i32 [ 0, %2 ], [ %12, %11 ] | %5 = icmp slt i32 %.0, 10}
Kill[]: {%.01 = phi i32 [ %1, %2 ], [ %7, %11 ] | %.0 = phi i32 [ 0, %2 ], [ %12, %11 ] | %5 = icmp slt i32 %.0, 10}

BB Name
IN[]: {i32 %0 | i32 %1 | %3 = sub nsw i32 %0, 1 | %.01 = phi i32 [ %1, %2 ], [ %7, %11 ] | %.0 = phi i32 [ 0, %2 ], [ %12, %11 ] |
  %5 = icmp slt i32 %.0, 10 | %7 = sub nsw i32 %.01, 2 | %8 = icmp sgt i32 %7, 4 | %12 = add nsw i32 %.0, 1}
OUT[]: {i32 %0 | i32 %1 | %3 = sub nsw i32 %0, 1 | %.01 = phi i32 [ %1, %2 ], [ %7, %11 ] | %.0 = phi i32 [ 0, %2 ], [ %12, %11 ] |
  %5 = icmp slt i32 %.0, 10 | %7 = sub nsw i32 %.01, 2 | %8 = icmp sgt i32 %7, 4 | %12 = add nsw i32 %.0, 1}
Gen[]: {%7 = sub nsw i32 %.01, 2 | %8 = icmp sgt i32 %7, 4}
Kill[]: {%7 = sub nsw i32 %.01, 2 | %8 = icmp sgt i32 %7, 4}

BB Name
IN[]: {i32 %0 | i32 %1 | %3 = sub nsw i32 %0, 1 | %.01 = phi i32 [ %1, %2 ], [ %7, %11 ] | %.0 = phi i32 [ 0, %2 ], [ %12, %11 ] |
  %5 = icmp slt i32 %.0, 10 | %7 = sub nsw i32 %.01, 2 | %8 = icmp sgt i32 %7, 4 | %12 = add nsw i32 %.0, 1}
OUT[]: {i32 %0 | i32 %1 | %3 = sub nsw i32 %0, 1 | %.01 = phi i32 [ %1, %2 ], [ %7, %11 ] | %.0 = phi i32 [ 0, %2 ], [ %12, %11 ] |
  %5 = icmp slt i32 %.0, 10 | %7 = sub nsw i32 %.01, 2 | %8 = icmp sgt i32 %7, 4 | %12 = add nsw i32 %.0, 1}
Gen[]: {}
Kill[]: {}

BB Name
IN[]: {i32 %0 | i32 %1 | %3 = sub nsw i32 %0, 1 | %.01 = phi i32 [ %1, %2 ], [ %7, %11 ] | %.0 = phi i32 [ 0, %2 ], [ %12, %11 ] |
  %5 = icmp slt i32 %.0, 10 | %7 = sub nsw i32 %.01, 2 | %8 = icmp sgt i32 %7, 4 | %12 = add nsw i32 %.0, 1}
OUT[]: {i32 %0 | i32 %1 | %3 = sub nsw i32 %0, 1 | %.01 = phi i32 [ %1, %2 ], [ %7, %11 ] | %.0 = phi i32 [ 0, %2 ], [ %12, %11 ] |
  %5 = icmp slt i32 %.0, 10 | %7 = sub nsw i32 %.01, 2 | %8 = icmp sgt i32 %7, 4 | %12 = add nsw i32 %.0, 1}
Gen[]: {}
Kill[]: {}

BB Name
IN[]: {i32 %0 | i32 %1 | %3 = sub nsw i32 %0, 1 | %.01 = phi i32 [ %1, %2 ], [ %7, %11 ] | %.0 = phi i32 [ 0, %2 ], [ %12, %11 ] |
  %5 = icmp slt i32 %.0, 10 | %7 = sub nsw i32 %.01, 2 | %8 = icmp sgt i32 %7, 4 | %12 = add nsw i32 %.0, 1}
OUT[]: {i32 %0 | i32 %1 | %3 = sub nsw i32 %0, 1 | %.01 = phi i32 [ %1, %2 ], [ %7, %11 ] | %.0 = phi i32 [ 0, %2 ], [ %12, %11 ] |
  %5 = icmp slt i32 %.0, 10 | %7 = sub nsw i32 %.01, 2 | %8 = icmp sgt i32 %7, 4 | %12 = add nsw i32 %.0, 1}
Gen[]: {%12 = add nsw i32 %.0, 1}
Kill[]: {%12 = add nsw i32 %.0, 1}

BB Name
IN[]: {i32 %0 | i32 %1 | %3 = sub nsw i32 %0, 1 | %.01 = phi i32 [ %1, %2 ], [ %7, %11 ] | %.0 = phi i32 [ 0, %2 ], [ %12, %11 ] |
  %5 = icmp slt i32 %.0, 10 | %7 = sub nsw i32 %.01, 2 | %8 = icmp sgt i32 %7, 4 | %12 = add nsw i32 %.0, 1}
OUT[]: {i32 %0 | i32 %1 | %3 = sub nsw i32 %0, 1 | %.01 = phi i32 [ %1, %2 ], [ %7, %11 ] | %.0 = phi i32 [ 0, %2 ], [ %12, %11 ] |
  %5 = icmp slt i32 %.0, 10 | %7 = sub nsw i32 %.01, 2 | %8 = icmp sgt i32 %7, 4 | %12 = add nsw i32 %.0, 1}
Gen[]: {}
Kill[]: {}
```

- Liveness Analysis

Through Backward traversal, we get the IN and OUT sets of each block, which contain the variables which are live at the IN of every block. Finally, we use the union operator to obtain the final set.

```
FUNCTION :: main
[BB Name]entry
IN[entry]: { }
OUT[entry]: {%argc + 50 ,%add + 96 }
Gen[entry]: {%argc + 50 ,%add + 96 }
Kill[entry]: {%add + 96 ,%add - 50 ,96 * %add ,%add + 50 }

[BB Name]if.then
IN[if.then]: {%argc + 50 ,%add + 96 }
OUT[if.then]: {%argc + 50 ,%add + 96 ,%add - 50 ,96 * %add }
Gen[if.then]: {%add - 50 ,96 * %add }
Kill[if.then]: { }

[BB Name]if.else
IN[if.else]: {%argc + 50 ,%add + 96 }
OUT[if.else]: {%argc + 50 ,%add + 96 ,96 * %add ,%add + 50 }
Gen[if.else]: {96 * %add ,%add + 50 }
Kill[if.else]: { }

[BB Name]if.end
IN[if.end]: {%argc + 50 ,%add + 96 ,96 * %add }
OUT[if.end]: {%argc + 50 ,%add + 96 ,96 * %add ,50 - 96 ,%sub4 + %f.0 }
Gen[if.end]: {50 - 96 ,%sub4 + %f.0 }
Kill[if.end]: {%sub4 + %f.0 }

```