CSE 231 – Project Part3

Outline

- Liveness Analysis
  - Sample Code
  - IR sample
  - Analysis
  - Implementation tips
- May-Point-To Analysis
  - Sample Code
  - IR sample
  - Analysis
  - Implementation tips

Liveness Analysis

- Perform Liveness Analysis using the Dataflow Analysis Framework built in Project Part 2
- Liveness Analysis is used to determine if a variable is “alive” at a particular node.
- Remember that Liveness Analysis is a backward analysis

```
template <class Info, bool Direction>
class DataFlowAnalysis {
...
}
```

Set *Direction* appropriately for the analysis

Sample code

```
#include <iostream>
using namespace std;

int main () {
        int x, y, z;
        x = 3;
        y = 4;
        x = 5;
        z = x + y;
        return 0;
}
```

LLVM IR

```
define dso_local i32 @main() #4 {
 /*1*/ %1 = alloca i32, align 4
 /*2*/ %2 = alloca i32, align 4 // int x
 /*3*/ %3 = alloca i32, align 4 // int y
 /*4*/ %4 = alloca i32, align 4 // int z
 /*5*/ store i32 0, i32* %1, align 4
 /*6*/ store i32 3, i32* %2, align 4 // x = 3
 /*7*/ store i32 4, i32* %3, align 4 // y = 4
 /*8*/ store i32 5, i32* %2, align 4 // x = 5
 /*9*/ %5 = load i32, i32* %2, align 4 // z = x + y
 /*10*/ %6 = load i32, i32* %3, align 4 // z = x + y
 /*11*/ %7 = add nsw i32 %5, %6 // z = x + y
 /*12*/ store i32 %7, i32* %4, align 4 // z = x + y
 /*13*/ ret i32 0
}
```

Analysis results

```
define dso_local i32 @main() #4 {
 /*1*/ %1 = alloca i32, align 4
 /*2*/ %2 = alloca i32, align 4 // int x
 /*3*/ %3 = alloca i32, align 4 // int y
 /*4*/ %4 = alloca i32, align 4 // int z
 /*5*/ store i32 0, i32* %1, align 4
 /*6*/ store i32 3, i32* %2, align 4 // x = 3
 /*7*/ store i32 4, i32* %3, align 4 // y = 4
 /*8*/ store i32 5, i32* %2, align 4 // x = 5
 /*9*/ %5 = load i32, i32* %2, align 4 // z = x + y
 /*10*/ %6 = load i32, i32* %3, align 4 // z = x + y
 /*11*/ %7 = add nsw i32 %5, %6 // z = x + y
 /*12*/ store i32 %7, i32* %4, align 4 // z = x + y
 /*13*/ ret i32 0
}
```

```
Edge 2->Edge 1:1|
Edge 3->Edge 2:1|2|
Edge 4->Edge 3:1|2|3|
Edge 5->Edge 4:1|2|3|4|
Edge 6->Edge 5:2|3|4|
Edge 7->Edge 6:2|3|4|
Edge 8->Edge 7:2|3|4|
Edge 9->Edge 8:2|3|4|
Edge 10->Edge 9:3|4|9|
Edge 11->Edge 10:4|9|10|
Edge 12->Edge 11:4|11|
Edge 13->Edge 12:
```

Implementation Tips

- Take a close look at the output required. This should help you in structuring your `LivenessInfo` class.
- Carefully categorize the IR instructions in to the correct category to apply the correct flow function.
- How to access the operands in an Instruction?
  - `getNumOperands(), getOperant(pos)`
  - `op_iterator`
- How to get index of instruction from the operand?
  - Cast to instruction

May-Point-To Analysis

- May-Point-To pointer analysis using the Dataflow Analysis Framework built in Project Part 2

```
/*10*/ %ptr = alloca i32, i32 4
```

Notation:

- **M10** -> DFA identifier of the memory object allocated by this instruction
- **R10** -> DFA identifier of the pointer created by this instruction

Sample code

```
#include <iostream>
using namespace std;

int main () {
        int x;
        int *a;
        x = 3;
        a = &x;
        return 0;
}
```

LLVM IR

```
define dso_local i32 @main() #4 {
 /*1*/ %1 = alloca i32, align 4
 /*2*/ %2 = alloca i32, align 4 // int x
 /*3*/ %3 = alloca i32*, align 8 // int *a
 /*4*/ store i32 0, i32* %1, align 4
 /*5*/ store i32 3, i32* %2, align 4 // x = 3
 /*6*/ store i32* %2, i32** %3, align 8 // a = &x
 /*7*/ ret i32 0
}
```

Analysis results

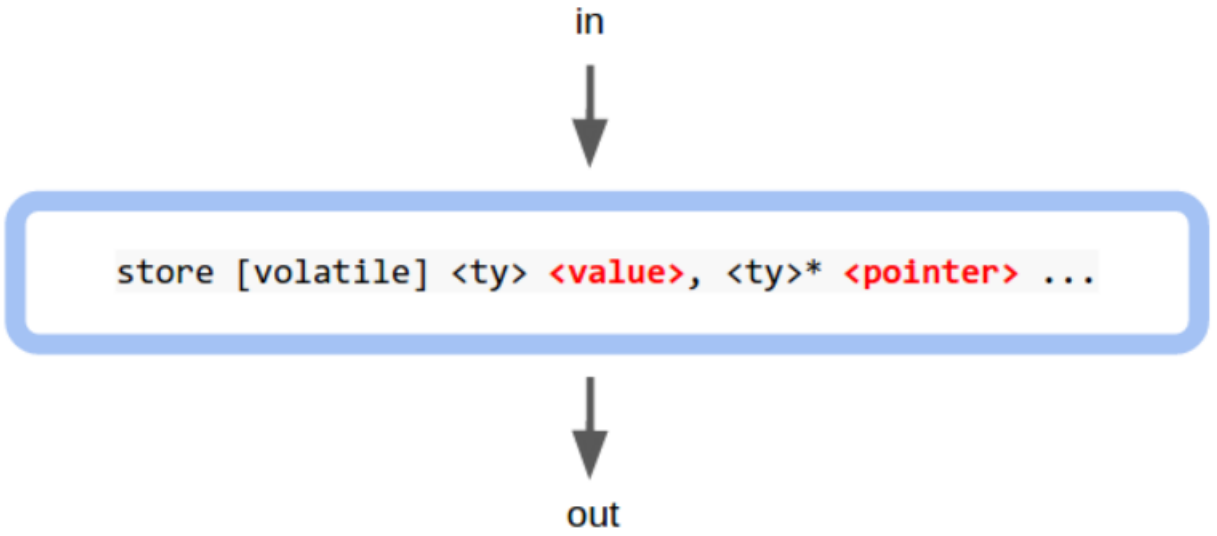
```
define dso_local i32 @main() #4 {
 /*1*/ %1 = alloca i32, align 4
 /*2*/ %2 = alloca i32, align 4 // int x
 /*3*/ %3 = alloca i32*, align 8 // int *a
 /*4*/ store i32 0, i32* %1, align 4
 /*5*/ store i32 3, i32* %2, align 4 // x = 3
 /*6*/ store i32* %2, i32** %3, align 8 // a = &x
 /*7*/ ret i32 0
}
```

```
Edge 0->Edge 1:
Edge 1->Edge 2:R1->(M1/)|
Edge 2->Edge 3:R1->(M1/)|R2->(M2/)|
Edge 3->Edge 4:R1->(M1/)|R2->(M2/)|R3->(M3/)|
Edge 4->Edge 5:R1->(M1/)|R2->(M2/)|R3->(M3/)|
Edge 5->Edge 6:R1->(M1/)|R2->(M2/)|R3->(M3/)|
Edge 6->Edge 7:R1->(M1/)|R2->(M2/)|R3->(M3/)|M3->(M2/)|
```

Flow function of *store*

```
/*1*/ %1 = alloca i32, align 4
/*2*/ %2 = alloca i32, align 4
/*3*/ %3 = alloca i32*, align 8
/*4*/ store i32 0, i32* %1, align 4
/*5*/ store i32 3, i32* %2, align 4
/*6*/ store i32* %2, i32** %3, align 8
/*7*/ ret i32 0
```

```
Edge 0->Edge 1:
Edge 1->Edge 2:R1->(M1/)|
Edge 2->Edge 3:R1->(M1/)|R2->(M2/)|
Edge 3->Edge 4:R1->(M1/)|R2->(M2/)|R3->(M3/)|
Edge 4->Edge 5:R1->(M1/)|R2->(M2/)|R3->(M3/)|
Edge 5->Edge 6:R1->(M1/)|R2->(M2/)|R3->(M3/)|
Edge 6->Edge 7:R1->(M1/)|R2->(M2/)|R3->(M3/)|M3->(M2/)|
```



$$out = in \cup \{R2 \rightarrow M2 \in in \cap R3 \rightarrow M3 \in in\}$$

Implementation Tips

- Take a close look at the output required. This should help you in structuring your `MayPointToInfo` class.
- You only need to handle the 7(+1) IR instructions mentioned in the project writeup.
- How to access the operands in an Instruction?
  - Cast the generic instruction to the specific instruction subclass, eg: `GetElementPtrInst` and use the class specific functions like `getPointerOperand()` to access the correct operand.

```
<result> = getelementptr <ty>, <ty>* <ptrval>{, <ty> <idx>}*
```

- How to get index of instruction from the operand?
  - Cast to instruction