Simplified LLVM Instruction Modelling (SLIM) for Analysis and Optimizations

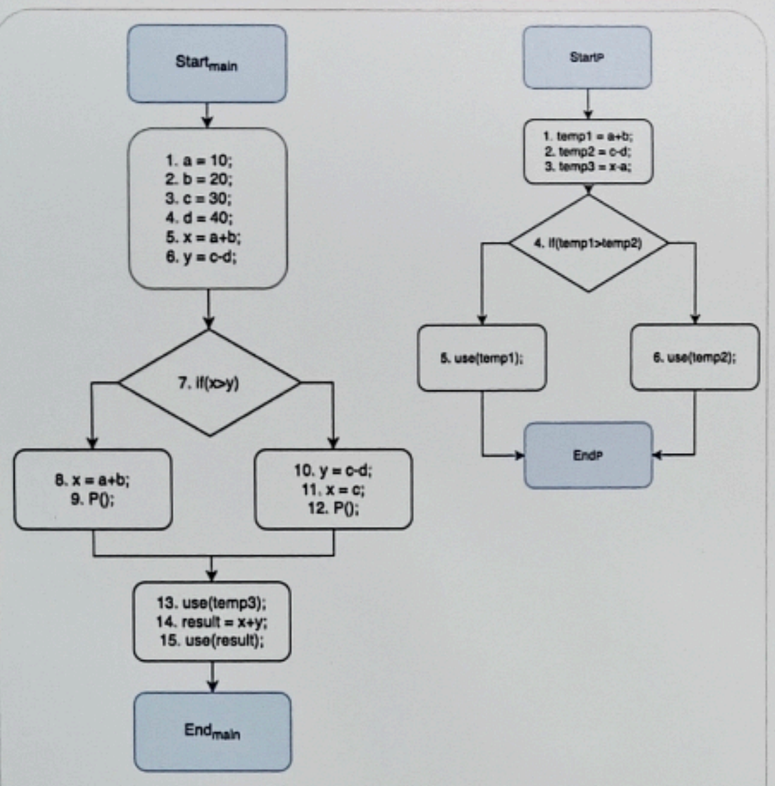
Om Godage, Aman Sharma
Guide: Prof. Uday Khedker
Department of Computer Science and Engineering, IIT Bombay

Sample Program

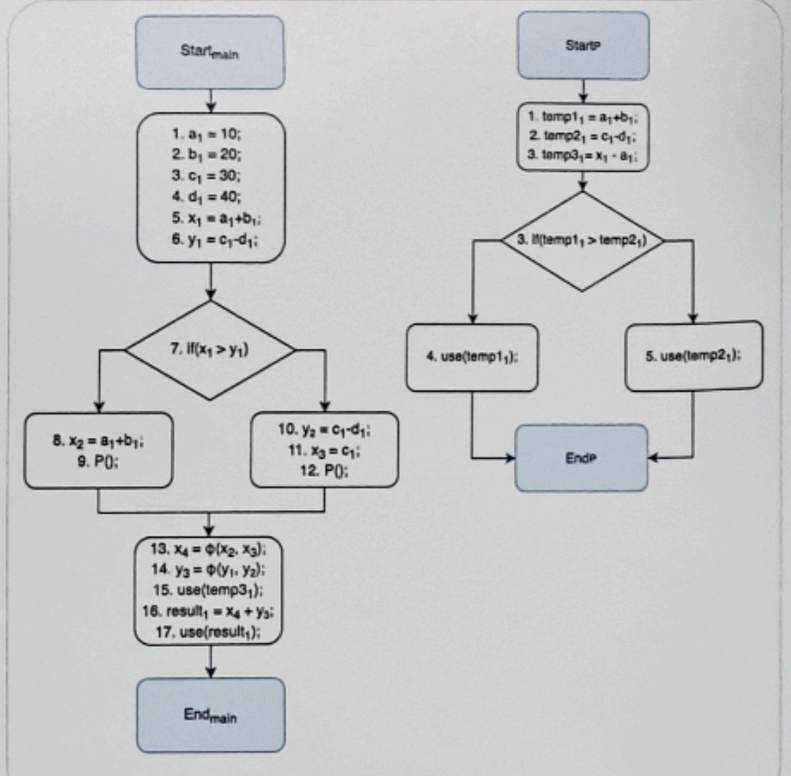
```
int a, b, c, d, x, y;
int temp3;

void main() {
    a = 10;
    b = 20;
    c = 30;
    d = 40;

    x = a + b;
    y = c - d;

    if (x > y) {
        x = a + b;
        P();
    } else {
        y = c - d;
        x = c;
        P();
    }
    use(temp3);
    int result = x + y;
    use(result);
}
```

```
void P() {
    int temp1 = a + b;
    int temp2 = c - d;
    temp3 = x - a;

    if (temp1 > temp2)
        use(temp1);
    else
        use(temp2);
}
```

CFG



SSA Form



What is SLIM?

- **Instruction-Level abstraction** over **LLVM** IR suitable for performing program analysis
- **Features**
  - Hiding **Verbosity** (debug and alignment information)
  - Assignments depicted as **use-def** instead of memory loads/stores
  - Field and array accesses depicted directly as **pointer accesses** instead of GEP
- **Uses**
  - **VASCO** (interprocedural analysis using value sensitive contexts)
  - **CoS-SSA**

LLVM Output

```
; Function Attrs: noinline nounwind uwtable
define dso_local void @P() #0 !dbg !26 {
entry:
  %i = load i32, i32* @a, align 4, !dbg !30
  %i1 = load i32, i32* @b, align 4, !dbg !31
  %add = add nsw i32 %i, %i1, !dbg !32
  call void @llvm.dbg.value(metadata i32 %add, metadata !33, metadata !DIExpression()), !dbg !34
  %i2 = load i32, i32* @c, align 4, !dbg !35
  %i3 = load i32, i32* @d, align 4, !dbg !36
  %sub = sub nsw i32 %i2, %i3, !dbg !37
  call void @llvm.dbg.value(metadata i32 %sub, metadata !38, metadata !DIExpression()), !dbg !34
  %cmp = icmp sgt i32 %add, %sub, !dbg !39
  br i1 %cmp, label %if.then, label %if.else, !dbg !41

if.then:                                          ; preds = %entry
  %call = call i32 (i8*, ...) @use(i8* noundef getelementptr inbounds
          ([3 x i8], [3 x i8]* @.str, i64 0, i64 0), i32 noundef %add), !dbg !42
  br label %if.end, !dbg !42

if.else:                                          ; preds = %entry
  %call1 = call i32 (i8*, ...) @use(i8* noundef getelementptr inbounds
           ([3 x i8], [3 x i8]* @.str, i64 0, i64 0), i32 noundef %sub), !dbg !43
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  ret void, !dbg !44
}
```

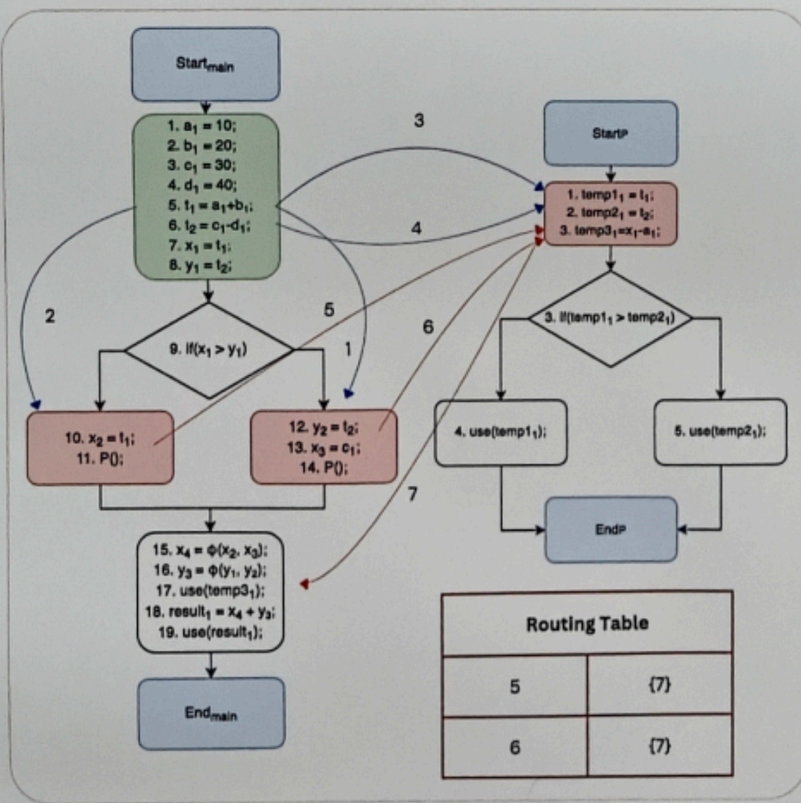
SLIM Output

```
[program.c] Function: P
-----------------------------------
Basic block 8: entry (Predecessors: [])
 [0][7] i_P = a
 [1][7] i1_P = b
 [2][7] add_P = i_P + i1_P
 [3][8] i2_P = c
 [4][8] i3_P = d
 [5][8] sub_P = i2_P - i3_P
 [6][9] cmp_P = add_P > sub_P
 [7][9] branch (cmp_P) if.then_P, if.else_P


Basic block 9: if.then_P (Predecessors: [entry])
 [8][src: 10] call_P = call printf(.str, add_P)
 [9][10] branch if.end_P


Basic block 10: if.else_P (Predecessors: [entry])
 [10][src: 12] call1_P = call printf(.str, sub_P)
 [11]branch if.end_P


Basic block 11: if.end_P (Predecessors: [if.else_P, if.then_P])
 [12][14] return
```

Global Value Numbering

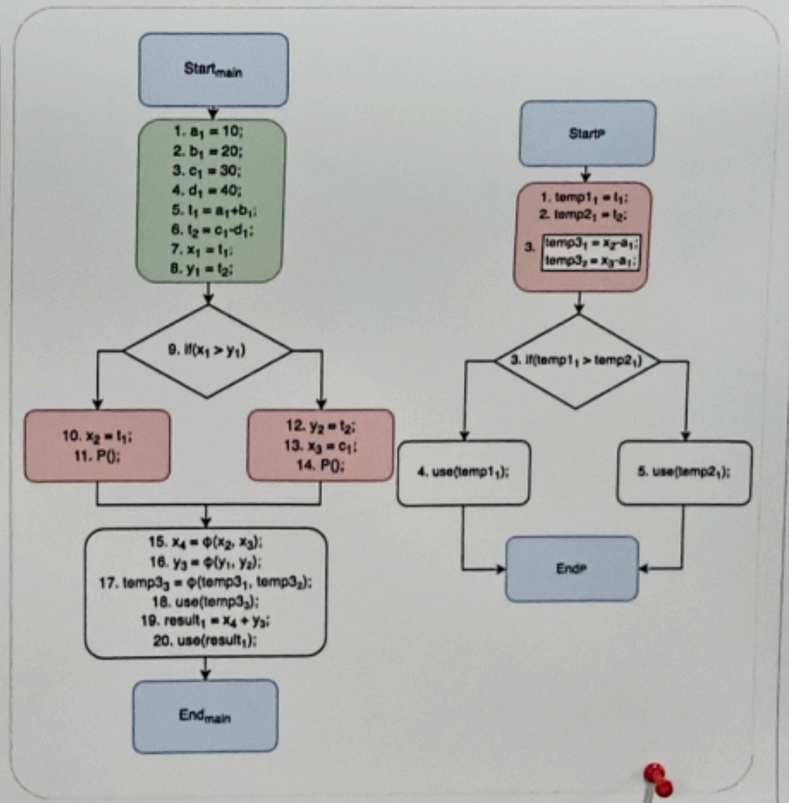
- **Unique Value Assignment**: Assigns unique numbers to equivalent expressions globally
- **Detects Deeper Redundancies**: Detects redundancies beyond **CSE**
- **SSA Dependency**: Relies on **SSA** form for accurate analysis, and applies **interprocedurally**
- **Optimization Scope**: Enables optimizations like redundancy removal and copy propagation

Optimization + DU Form



Routing Table	
5	{7}
6	{7}

CoS-SSA Form



Future Work

- **SLIM:**
  - Support for Transformations
  - Reverse Mapping to LLVM IR
  - Supporting CoS-SSA APIs (e.g. adding CoS-φ nodes)
- **GVN:**
  - Use SLIM to perform GVN interprocedurally
  - Use CoS-DU to get a more optimized program than interprocedural GVN