Dataflow Analysis

git fetch
git checkout exercises/day-4-analysis-1

Warmup Quiz: Can we optimize this code?

```
1 %longer = list.push_back %li, %item : !list.list<i32>
2 %res = list.is_empty %longer : !list.list<i32> -> i1
```

Warmup Quiz: Can we optimize this code?

```
1 %longer = list.push_back %li, %item : !list.list<i32>
2 %res = list.is_empty %longer : !list.list<i32> -> i1
```

%longer is never empty

Warmup Quiz: Can we optimize this code?

```
1 %longer = list.push_back %li, %item : !list.list<i32>
2 %mapped = list.map %list with ...
3 %res = list.is_empty %mapped : !list.list<i32> -> i1
```

Warmup Quiz: Can we optimize this code?

```
1 %longer = list.push_back %li, %item : !list.list<i32>
2 %mapped = list.map %list with ...
3 %res = list.is_empty %mapped : !list.list<i32> -> i1
```

%mapped is never empty

Warmup Quiz: Can we optimize this code?

```
%lres = scf.if %c -> list {
  %l_true = list.push_back %l, %x : !list.list<i32>
  scf.yield %l_true : !list.list<i32>
} else {
  %l_false = list.push_back %l, %y : !list.list<i32>
  %l_false2 = list.push_back %l_false, %y : !list.list<i32>
  scf.yield %l_false : !list.list<i32>
}
%res = list.is_empty %lres : !list.list<i32> -> i1
```

Warmup Quiz: Can we optimize this code?

```
1 %lres = scf.if %c -> list {
2   %l_true = list.push_back %l, %x : !list.list<i32>
3   scf.yield %l_true : !list.list<i32>
4 } else {
5   %l_false = list.push_back %l, %y : !list.list<i32>
6   %l_false2 = list.push_back %l_false, %y : !list.list<i32>
7   scf.yield %l_false : !list.list<i32>
8 }
9 %res = list.is_empty %lres : !list.list<i32> -> i1
```

Warmup Quiz: Can we optimize this code?

```
1 %lres = scf.if %c -> list {
2   %l_true = list.push_back %l, %x : !list.list<i32>
3   scf.yield %l_true : !list.list<i32>
4 } else {
5   %l_false = list.push_back %l, %y : !list.list<i32>
6   %l_false2 = list.push_back %l_false, %y : !list.list<i32>
7   scf.yield %l_false : !list.list<i32>
8 }
9 %res = list.is_empty %lres : !list.list<i32> -> i1
```

Warmup Quiz: Can we optimize this code?

```
1 %lres = scf.if %c -> list {
2   %l_true = list.push_back %l, %x : !list.list<i32>
3   scf.yield %l_true : !list.list<i32>
4 } else {
5   %l_false = list.push_back %l, %y : !list.list<i32>
6   %l_false2 = list.push_back %l_false, %y : !list.list<i32>
7   scf.yield %l_false : !list.list<i32>
8 }
9 %res = list.is_empty %lres : !list.list<i32> -> i1
```

Warmup Quiz: Can we optimize this code?

```
1 %lres = scf.if %c -> list {
2   %l_true = list.push_back %l, %x : !list.list<i32>
3   scf.yield %l_true : !list.list<i32>
4 } else {
5   %l_false = list.push_back %l, %y : !list.list<i32>
6   %l_false2 = list.push_back %l_false, %y : !list.list<i32>
7   scf.yield %l_false : !list.list<i32>
8 }
9 %res = list.is_empty %lres : !list.list<i32> -> i1
```

Warmup Quiz: Can we optimize this code?

```
1 %lres = scf.if %c -> list {
2   %l_true = list.push_back %l, %x : !list.list<i32>
3   scf.yield %l_true : !list.list<i32>
4 } else {
5   %l_false
6   %l_false2
7   scf.yield %l_false : !list.list<i32>
8 }
9 %res = list.is_empty %lres : !list.list<i32> -> i1
```

This is dataflow analysis

Examples of dataflow analyses

SparseConstantPropagation

```
%c42 = arith.constant 42 : i32
%x = scf.if %cond -> i32 {
  scf.yield %c42
} else {
  scf.yield %c42
}
vector.print %x : i32
```

SparseConstantPropagation

```
1 %c42 = arith.constant 42 : i32    // %c42 = 42
2 %x = scf.if %cond -> i32 {        // %x   = 42
3   scf.yield %c42
4 } else {
5   scf.yield %c42
6 }
7 vector.print %x : i32
```

SparseConstantPropagation

```
1 %c42 = arith.constant 42 : i32    // %c42 = 42
2 %x = scf.if %cond -> i32 {        // %x   = 42
3   scf.yield %c42
4 } else {
5   scf.yield %c42
6 }
7 vector.print %x : i32
```

IntegerRangeAnalysis

```
%x = ...
%c15 = arith.constant 15 : i32
%c16 = arith.constant 16 : i32

%y = arith.andi %x, %c15 : i32
%r = arith.remui %y, %c16 : i32

vector.print %r : i8
```

IntegerRangeAnalysis

```
1 %x = ...                                  // %x   = [-2^31, 2^31-1]
2 %c15 = arith.constant 15 : i32            // %c15 = [15, 15]
3 %c16 = arith.constant 16 : i32            // %c16 = [16, 16]
4
5 %y = arith.andi %x, %c15 : i32            // %y   = [0, 15]
6 %r = arith.remui %y, %c16 : i32           // %r   = [0, 15]
7
8 vector.print %r : i8
```

IntegerRangeAnalysis

```
1 %x = ...                                  // %x   = [-2^31, 2^31-1]
2 %c15 = arith.constant 15 : i32            // %c15 = [15, 15]
3 %c16 = arith.constant 16 : i32            // %c16 = [16, 16]
4
5 %y = arith.andi %x, %c15 : i32            // %y   = [0, 15]
6 %r = arith.remui %y, %c16 : i32           // %r   = [0, 15]
7
8 vector.print %r : i8
```

LivenessAnalysis

```
1 func.func @liveness(%cond: i1, %a: i32, %b: i32) -> i32 {
2
3   %x, %dead = scf.if %cond -> (i32, i32) {
4     scf.yield %a, %b : i32, i32
5   } else {
6     scf.yield %b, %a : i32, i32
7   }
8   return %x : i32
9 }
```

LivenessAnalysis

```
1 func.func @liveness(%cond: i1, %a: i32, %b: i32) -> i32 {
2     // %a alive, %b alive, %cond alive
3   %x, %dead = scf.if %cond -> (i32, i32) {  // %x alive, %dead dead
4     scf.yield %a, %b : i32, i32
5   } else {
6     scf.yield %b, %a : i32, i32
7   }
8   return %x : i32
9 }
```

LivenessAnalysis

```
1 func.func @liveness(%cond: i1, %a: i32, %b: i32) -> i32 {
2     // %a alive, %b alive, %cond alive
3   %x, %dead = scf.if %cond -> (i32, i32) {  // %x alive, %dead dead
4     scf.yield %a, %b : i32, i32
5   } else {
6     scf.yield %b, %a : i32, i32
7   }
8   return %x : i32
9 }
```

DenseSizeRangeAnalysis

```
%list = ...
%len = list.length %list : !listlang.list -> index
%c0 = arith.constant 0 : index
%is_empty = arith.cmpi eq, %len, %c0 : index
scf.if %is_empty {
  %len2 = list.length %list : !listlang.list -> index
  use %len2
} else {
  ...
}
```

DenseSizeRangeAnalysis

```
1 %list = ...                                                    // {}
2 %len = list.length %list : !listlang.list -> index            // {}
3 %c0 = arith.constant 0 : index                                // {}
4 %is_empty = arith.cmpi eq, %len, %c0 : index                  // {}
5 scf.if %is_empty {                                            // {}
6   %len2 = list.length %list : !listlang.list -> index         // {%list}
7   use %len2                                                   // {%list}
8 } else {
9   ...                                                         // {}
10 }
```

DenseSizeRangeAnalysis

```
1  %list = ...                                                    // {}
2  %len = list.length %list : !listlang.list -> index             // {}
3  %c0 = arith.constant 0 : index                                 // {}
4  %is_empty = arith.cmpi eq, %len, %c0 : index                   // {}
5  scf.if %is_empty {                                             // {}
6    %len2 = list.length %list : !listlang.list -> index          // {%list}
7    use %len2                                                    // {%list}
8  } else {
9    ...                                                          // {}
10 }
```

Some Passes using dataflow analysis

- SCCP
- IntRangeOptimizations
- IntRangeNarrowing
- RemoveDeadValues
- XeGPUPropagateLayout
- CIRCT –canonicalize (without the framework)

How does dataflow analysis work?

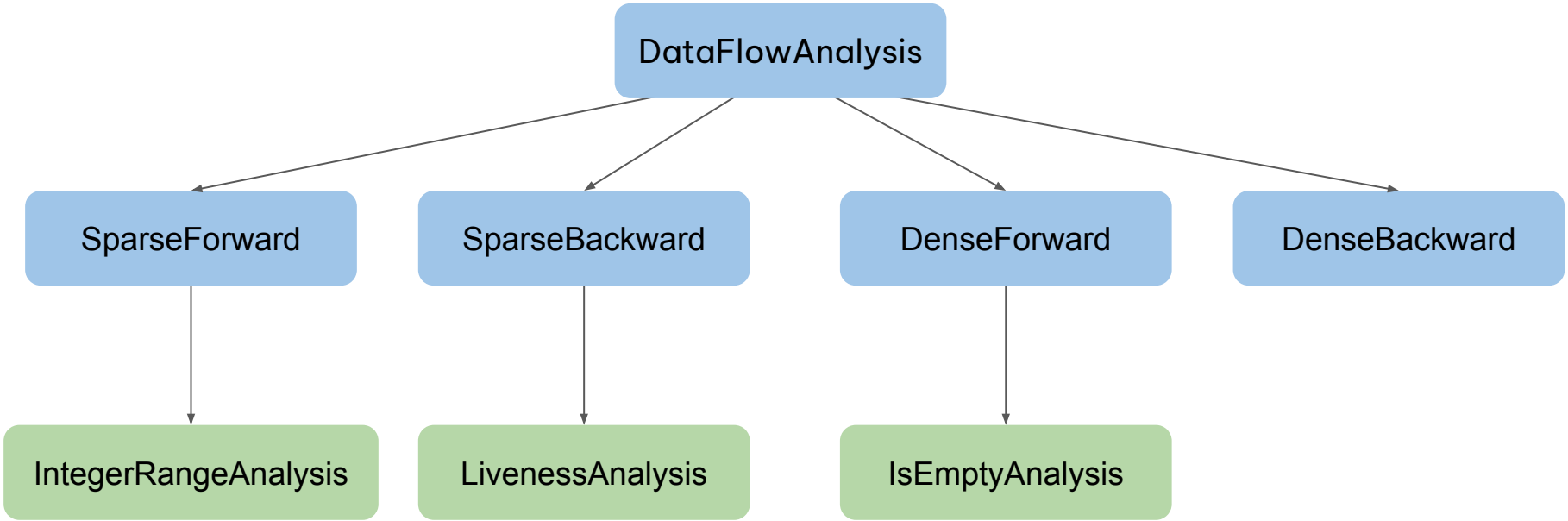
Steps in dataflow analysis

- Initialize some variables
- Do a first pass across the IR
- Use a worklist to handle loop fixpoint computation

Each update is either:

- Using an operation rule to propagate information
- Merging two incoming information

MLIR Dataflow Analysis hierarchy



Example : ConstantAnalysis

```
1 func.func @foo(%cond: i1) -> i32 {   // %cond = uninitialized
2   %r = scf.if %cond -> i32 {         // %r    = uninitialized
3     %c15 = arith.constant 15 : i32   // %c15  = uninitialized
4     scf.yield %c15
5   } else {
6     %c16 = arith.constant 16 : i32   // %c16  = uninitialized
7     scf.yield %c16
8   }
9   func.return %r : i32
10 }
```

- Initialization

Example : ConstantAnalysis

```
1 func.func @foo(%cond: i1) -> i32 {   // %cond = unknown
2   %r = scf.if %cond -> i32 {         // %r    = uninitialized
3     %c15 = arith.constant 15 : i32   // %c15  = uninitialized
4     scf.yield %c15
5   } else {
6     %c16 = arith.constant 16 : i32   // %c16  = uninitialized
7     scf.yield %c16
8   }
9   func.return %r : i32
10 }
```

- Initialization

Example : ConstantAnalysis

```
1 func.func @foo(%cond: i1) -> i32 {    // %cond = unknown
2   %r = scf.if %cond -> i32 {          // %r    = uninitialized
3     %c15 = arith.constant 15 : i32    // %c15  = 15
4     scf.yield %c15
5   } else {
6     %c16 = arith.constant 16 : i32    // %c16  = uninitialized
7     scf.yield %c16
8   }
9   func.return %r : i32
10 }
```

- Initialization
- **First pass**

Example : ConstantAnalysis

```
1 func.func @foo(%cond: i1) -> i32 {    // %cond = unknown
2   %r = scf.if %cond -> i32 {          // %r    = 15
3     %c15 = arith.constant 15 : i32    // %c15  = 15
4     scf.yield %c15
5   } else {
6     %c16 = arith.constant 16 : i32    // %c16  = uninitialized
7     scf.yield %c16
8   }
9   func.return %r : i32
10 }
```

- Initialization
- **First pass**

Example : ConstantAnalysis

```
1 func.func @foo(%cond: i1) -> i32 {    // %cond = unknown
2   %r = scf.if %cond -> i32 {          // %r    = 15
3     %c15 = arith.constant 15 : i32    // %c15  = 15
4     scf.yield %c15
5   } else {
6     %c16 = arith.constant 16 : i32    // %c16  = 16
7     scf.yield %c16
8   }
9   func.return %r : i32
10 }
```

- Initialization
- **First pass**

Example : ConstantAnalysis

```
1 func.func @foo(%cond: i1) -> i32 {    // %cond = unknown
2   %r = scf.if %cond -> i32 {          // %r    = unknown
3     %c15 = arith.constant 15 : i32    // %c15  = 15
4     scf.yield %c15
5   } else {
6     %c16 = arith.constant 16 : i32    // %c16  = 16
7     scf.yield %c16
8   }
9   func.return %r : i32
10 }
```

- Initialization
- **First pass**

Example : ConstantAnalysis with loops

```
1 func.func @foo(%lb: index, %ub: index) -> i32 { // %lb, %ub = uninitialized
2   %c1 = arith.constant 1 : index                // %c1      = uninitialized
3   %c15 = arith.constant 15 : i32                // %c15     = uninitialized
4   %r = scf.for %i = %lb to %ub step %c1         // %r       = uninitialized
5       iter_args(%x = %c15) -> i32 {             // %i, %x   = uninitialized
6     %next = arith.addi %x, %c1 : i32            // %next    = uninitialized
7     scf.yield %next : i32
8   }
9   %res = arith.addi %r, %c1 : i32               // %res     = uninitialized
10  func.return %res : i32
11 }
```

- **Initialization**

Example : ConstantAnalysis with loops

```
1 func.func @foo(%lb: index, %ub: index) -> i32 { // %lb, %ub = unknown
2   %c1 = arith.constant 1 : index                // %c1      = uninitialized
3   %c15 = arith.constant 15 : i32                // %c15     = uninitialized
4   %r = scf.for %i = %lb to %ub step %c1         // %r       = uninitialized
5       iter_args(%x = %c15) -> i32 {             // %i, %x   = uninitialized
6     %next = arith.addi %x, %c1 : i32            // %next    = uninitialized
7     scf.yield %next : i32
8   }
9   %res = arith.addi %r, %c1 : i32               // %res     = uninitialized
10  func.return %res : i32
11 }
```

- Initialization

Example : ConstantAnalysis with loops

```
1  func.func @foo(%lb: index, %ub: index) -> i32 { // %lb, %ub = unknown
2    %c1 = arith.constant 1 : index                // %c1      = 1
3    %c15 = arith.constant 15 : i32                // %c15     = uninitialized
4    %r = scf.for %i = %lb to %ub step %c1         // %r       = uninitialized
5        iter_args(%x = %c15) -> i32 {             // %i, %x   = uninitialized
6      %next = arith.addi %x, %c1 : i32            // %next    = uninitialized
7      scf.yield %next : i32
8    }
9    %res = arith.addi %r, %c1 : i32               // %res     = uninitialized
10   func.return %res : i32
11 }
```

- Initialization
- **First pass**

Example : ConstantAnalysis with loops

```
1 func.func @foo(%lb: index, %ub: index) -> i32 { // %lb, %ub = unknown
2   %c1 = arith.constant 1 : index                // %c1      = 1
3   %c15 = arith.constant 15 : i32                // %c15     = 15
4   %r = scf.for %i = %lb to %ub step %c1         // %r       = uninitialized
5       iter_args(%x = %c15) -> i32 {             // %i, %x   = uninitialized
6     %next = arith.addi %x, %c1 : i32            // %next    = uninitialized
7     scf.yield %next : i32
8   }
9   %res = arith.addi %r, %c1 : i32               // %res     = uninitialized
10  func.return %res : i32
11 }
```

- Initialization
- **First pass**

Example : ConstantAnalysis with loops

```
1 func.func @foo(%lb: index, %ub: index) -> i32 { // %lb, %ub = unknown
2   %c1 = arith.constant 1 : index                // %c1       = 1
3   %c15 = arith.constant 15 : i32                // %c15      = 15
4   %r = scf.for %i = %lb to %ub step %c1         // %r        = uninitialized
5       iter_args(%x = %c15) -> i32 {             // %i, %x    = unknown, 15
6     %next = arith.addi %x, %c1 : i32            // %next     = uninitialized
7     scf.yield %next : i32
8   }
9   %res = arith.addi %r, %c1 : i32               // %res      = uninitialized
10  func.return %res : i32
11 }
```

- Initialization
- **First pass**

Example : ConstantAnalysis with loops

```
1 func.func @foo(%lb: index, %ub: index) -> i32 { // %lb, %ub = unknown
2   %c1 = arith.constant 1 : index                // %c1       = 1
3   %c15 = arith.constant 15 : i32                // %c15      = 15
4   %r = scf.for %i = %lb to %ub step %c1         // %r        = uninitialized
5       iter_args(%x = %c15) -> i32 {             // %i, %x    = unknown, 15
6     %next = arith.addi %x, %c1 : i32            // %next     = 16
7     scf.yield %next : i32
8   }
9   %res = arith.addi %r, %c1 : i32               // %res      = uninitialized
10  func.return %res : i32
11 }
```

- Initialization
- **First pass**

Example : ConstantAnalysis with loops

```
1 func.func @foo(%lb: index, %ub: index) -> i32 { // %lb, %ub = unknown
2   %c1 = arith.constant 1 : index                // %c1       = 1
3   %c15 = arith.constant 15 : i32                // %c15      = 15
4   %r = scf.for %i = %lb to %ub step %c1         // %r        = 16
5       iter_args(%x = %c15) -> i32 {             // %i, %x    = unknown
6     %next = arith.addi %x, %c1 : i32            // %next     = 16
7     scf.yield %next : i32
8   }
9   %res = arith.addi %r, %c1 : i32               // %res      = uninitialized
10  func.return %res : i32
11 }
```

- Initialization
- **First pass**

Example : ConstantAnalysis with loops

```
1  func.func @foo(%lb: index, %ub: index) -> i32 { // %lb, %ub = unknown
2    %c1 = arith.constant 1 : index                // %c1      = 1
3    %c15 = arith.constant 15 : i32                // %c15     = 15
4    %r = scf.for %i = %lb to %ub step %c1         // %r       = 16
5        iter_args(%x = %c15) -> i32 {             // %i, %x   = unknown
6      %next = arith.addi %x, %c1 : i32            // %next    = 16
7      scf.yield %next : i32
8    }
9    %res = arith.addi %r, %c1 : i32               // %res     = 17
10   func.return %res : i32
11 }
```

- Initialization
- **First pass**

Example : ConstantAnalysis with loops

```
1 func.func @foo(%lb: index, %ub: index) -> i32 { // %lb, %ub = unknown
2   %c1 = arith.constant 1 : index                // %c1       = 1
3   %c15 = arith.constant 15 : i32                // %c15      = 15
4   %r = scf.for %i = %lb to %ub step %c1         // %r        = 16
5       iter_args(%x = %c15) -> i32 {             // %i, %x    = unknown
6     %next = arith.addi %x, %c1 : i32            // %next     = 16
7     scf.yield %next : i32
8   }
9   %res = arith.addi %r, %c1 : i32               // %res      = 17
10  func.return %res : i32
11 }
```

- Initialization
- First pass
- **Fixpoint**

Example : ConstantAnalysis with loops

```
1  func.func @foo(%lb: index, %ub: index) -> i32 { // %lb, %ub = unknown
2    %c1 = arith.constant 1 : index                // %c1      = 1
3    %c15 = arith.constant 15 : i32                // %c15     = 15
4    %r = scf.for %i = %lb to %ub step %c1         // %r       = 16
5        iter_args(%x = %c15) -> i32 {             // %i, %x   = unknown
6      %next = arith.addi %x, %c1 : i32            // %next    = unknown
7      scf.yield %next : i32
8    }
9    %res = arith.addi %r, %c1 : i32               // %res     = 17
10   func.return %res : i32
11 }
```

- Initialization
- First pass
- **Fixpoint**

Example : ConstantAnalysis with loops

```
1  func.func @foo(%lb: index, %ub: index) -> i32 { // %lb, %ub = unknown
2    %c1 = arith.constant 1 : index                // %c1      = 1
3    %c15 = arith.constant 15 : i32                // %c15     = 15
4    %r = scf.for %i = %lb to %ub step %c1         // %r       = unknown
5        iter_args(%x = %c15) -> i32 {             // %i, %x   = unknown
6      %next = arith.addi %x, %c1 : i32            // %next    = unknown
7      scf.yield %next : i32
8    }
9    %res = arith.addi %r, %c1 : i32               // %res     = 17
10   func.return %res : i32
11 }
```

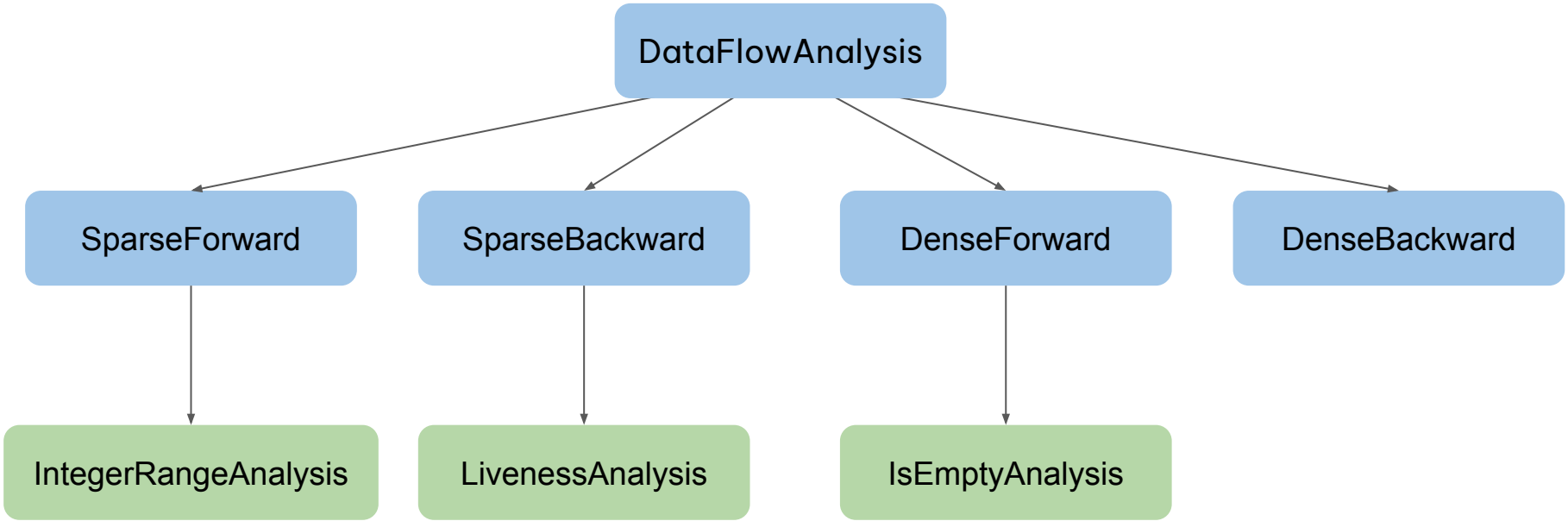
- Initialization
- First pass
- **Fixpoint**

Example : ConstantAnalysis with loops

```
1  func.func @foo(%lb: index, %ub: index) -> i32 { // %lb, %ub = unknown
2    %c1 = arith.constant 1 : index                // %c1      = 1
3    %c15 = arith.constant 15 : i32                // %c15     = 15
4    %r = scf.for %i = %lb to %ub step %c1         // %r       = unknown
5        iter_args(%x = %c15) -> i32 {             // %i, %x   = unknown
6      %next = arith.addi %x, %c1 : i32            // %next    = unknown
7      scf.yield %next : i32
8    }
9    %res = arith.addi %r, %c1 : i32               // %res     = unknown
10   func.return %res : i32
11 }
```

- Initialization
- First pass
- **Fixpoint**

MLIR Dataflow Analysis hierarchy



Example : LivenessAnalysis

```
1 // %cond uninitialized, %a uninitialized, %b uninitialized
2 func.func @liveness(%cond: i1, %a: i32, %b: i32) -> i32 {
3   // %x uninitialized, %dead uninitialized
4   %x, %dead = scf.if %cond -> (i32, i32) {
5     scf.yield %a, %b : i32, i32
6   } else {
7     scf.yield %b, %a : i32, i32
8   }
9   return %x : i32
10 }
```

Example : LivenessAnalysis

```
1 // %cond uninitialized, %a uninitialized, %b uninitialized
2 func.func @liveness(%cond: i1, %a: i32, %b: i32) -> i32 {
3   // %x live, %dead uninitialized
4   %x, %dead = scf.if %cond -> (i32, i32) {
5     scf.yield %a, %b : i32, i32
6   } else {
7     scf.yield %b, %a : i32, i32
8   }
9   return %x : i32
10 }
```

Example : LivenessAnalysis

```
1 // %cond uninitialized, %a uninitialized, %b uninitialized
2 func.func @liveness(%cond: i1, %a: i32, %b: i32) -> i32 {
3   // %x live, %dead dead
4   %x, %dead = scf.if %cond -> (i32, i32) {
5     scf.yield %a, %b : i32, i32
6   } else {
7     scf.yield %b, %a : i32, i32
8   }
9   return %x : i32
10 }
```

Example : LivenessAnalysis

```
1 // %cond uninitialized, %a live, %b uninitialized
2 func.func @liveness(%cond: i1, %a: i32, %b: i32) -> i32 {
3   // %x live, %dead dead
4   %x, %dead = scf.if %cond -> (i32, i32) {
5     scf.yield %a, %b : i32, i32
6   } else {
7     scf.yield %b, %a : i32, i32
8   }
9   return %x : i32
10 }
```

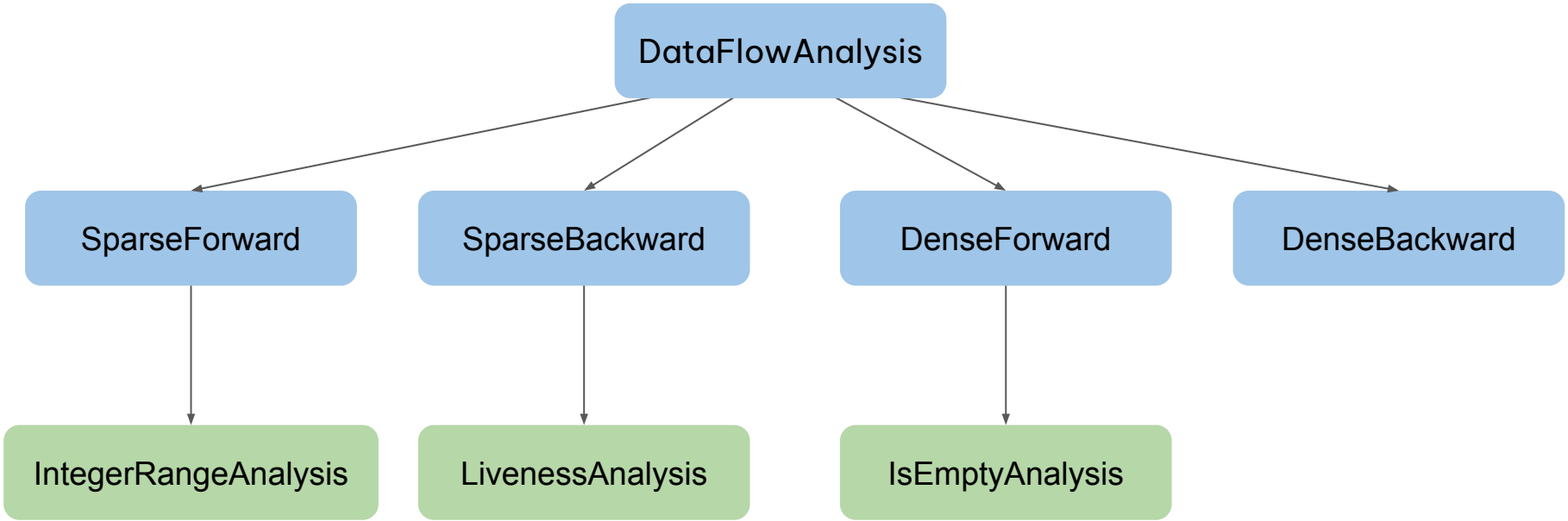
Example : LivenessAnalysis

```
1 // %cond uninitialized, %a live, %b live
2 func.func @liveness(%cond: i1, %a: i32, %b: i32) -> i32 {
3   // %x live, %dead dead
4   %x, %dead = scf.if %cond -> (i32, i32) {
5     scf.yield %a, %b : i32, i32
6   } else {
7     scf.yield %b, %a : i32, i32
8   }
9   return %x : i32
10 }
```

Example : LivenessAnalysis

```
1 // %cond live, %a live, %b live
2 func.func @liveness(%cond: i1, %a: i32, %b: i32) -> i32 {
3   // %x live, %dead dead
4   %x, %dead = scf.if %cond -> (i32, i32) {
5     scf.yield %a, %b : i32, i32
6   } else {
7     scf.yield %b, %a : i32, i32
8   }
9   return %x : i32
10 }
```

MLIR Dataflow Analysis hierarchy



Example : DenseIsEmpty

```
%list = ...                                                  // uninitialized
%len = list.length %list : !listlang.list -> index           // uninitialized
%c0 = arith.constant 0 : index                               // uninitialized
%is_empty = arith.cmpi eq, %len, %c0 : index                 // uninitialized
scf.if %is_empty {                                           // uninitialized
  %len2 = list.length %list : !listlang.list -> index        // uninitialized
  ...
} else {
  ...                                                        // uninitialized
}
```

Example : DenseIsEmpty

```
%list = ...                                                    // {}
%len = list.length %list : !listlang.list -> index             // uninitialized
%c0 = arith.constant 0 : index                                 // uninitialized
%is_empty = arith.cmpi eq, %len, %c0 : index                   // uninitialized
scf.if %is_empty {                                             // uninitialized
  %len2 = list.length %list : !listlang.list -> index          // uninitialized
  ...
} else {
  ...                                                          // uninitialized
}
```

Example : DenseIsEmpty

```
1  %list = ...                                                    // {}
2  %len = list.length %list : !listlang.list -> index             // {}
3  %c0 = arith.constant 0 : index                                 // uninitialized
4  %is_empty = arith.cmpi eq, %len, %c0 : index                   // uninitialized
5  scf.if %is_empty {                                             // uninitialized
6    %len2 = list.length %list : !listlang.list -> index          // uninitialized
7    ...
8  } else {
9    ...                                                          // uninitialized
10 }
```

Example : DenseIsEmpty

```
1  %list = ...                                                    // {}
2  %len = list.length %list : !listlang.list -> index             // {}
3  %c0 = arith.constant 0 : index                                 // {}
4  %is_empty = arith.cmpi eq, %len, %c0 : index                   // uninitialized
5  scf.if %is_empty {                                             // uninitialized
6    %len2 = list.length %list : !listlang.list -> index          // uninitialized
7    ...
8  } else {
9    ...                                                          // uninitialized
10 }
```

Example : DenseIsEmpty

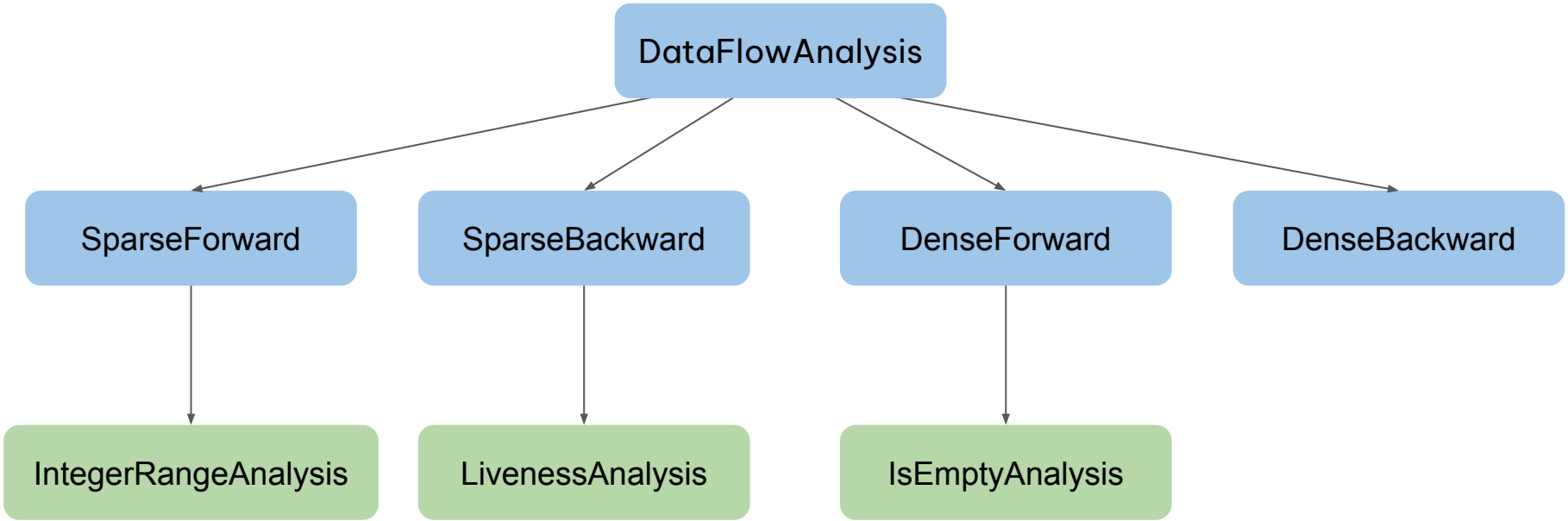
```
1  %list = ...                                                    // {}
2  %len = list.length %list : !listlang.list -> index             // {}
3  %c0 = arith.constant 0 : index                                 // {}
4  %is_empty = arith.cmpi eq, %len, %c0 : index                   // {}
5  scf.if %is_empty {                                             // uninitialized
6    %len2 = list.length %list : !listlang.list -> index          // uninitialized
7    ...
8  } else {
9    ...                                                          // uninitialized
10 }
```

Example : DenseIsEmpty

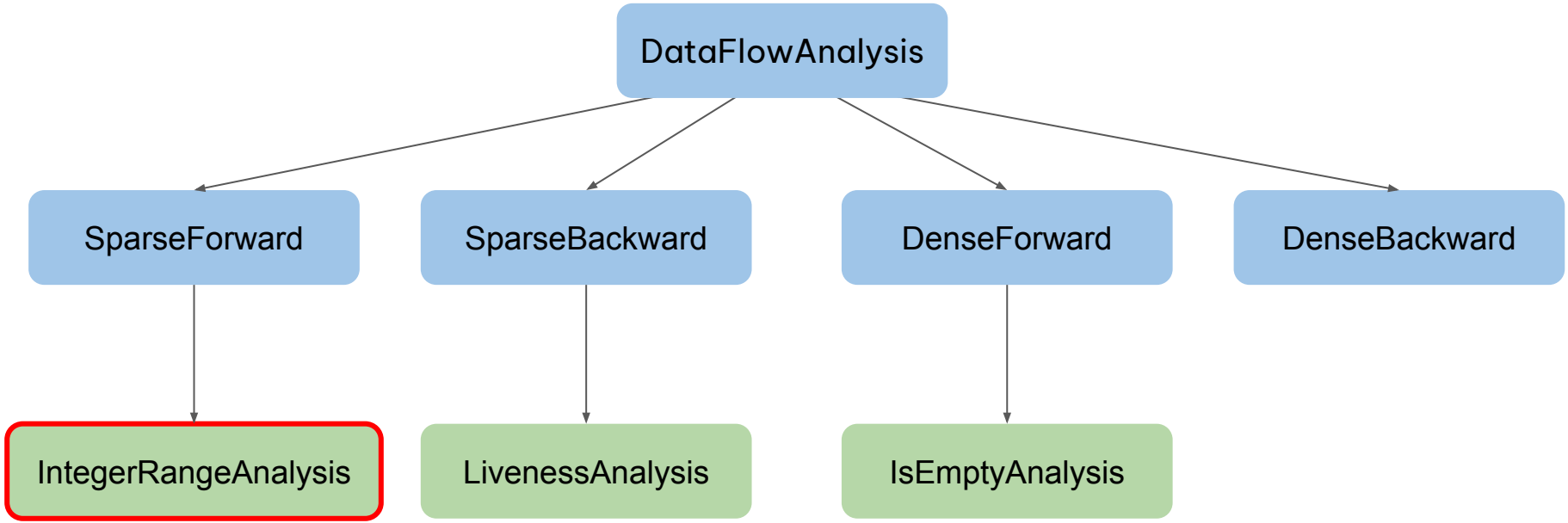
```
1  %list = ...                                                    // {}
2  %len = list.length %list : !listlang.list -> index             // {}
3  %c0 = arith.constant 0 : index                                 // {}
4  %is_empty = arith.cmpi eq, %len, %c0 : index                   // {}
5  scf.if %is_empty {                                             // uninitialized
6    %len2 = list.length %list : !listlang.list -> index          // {%list}
7    ...
8  } else {
9    ...                                                          // {}
10 }
```

Extending a Dataflow Analysis

MLIR Dataflow Analysis hierarchy



MLIR Dataflow Analysis hierarchy



Extending IntegerRangeAnalysis

```
func.func @loop_stack() -> (i1) {
  %c0 = arith.constant 0 : i32
  %c1 = arith.constant 1 : i32
  %c10 = arith.constant 10 : i32

  %init = list.from_elements %c0 : (i32) -> !list.list<i32>
  %list = scf.for %i = %c0 to %c10 step %c1
      iter_args(%current = %init) -> !list.list<i32> : i32 {
    %new_list = list.push_front %current, %i : !list.list<i32>
    scf.yield %new_list : !list.list<i32>
  }

  %rest = list.pop_front %list : !list.list<i32>
  %next = list.peek_front %rest : !list.list<i32> -> i32
  %check = arith.cmpi ult, %next, %c10 : i32
  return %check : i1
}
```

Extending IntegerRangeAnalysis

```
1  func.func @loop_stack() -> (i1) {
2    %c0 = arith.constant 0 : i32
3    %c1 = arith.constant 1 : i32
4    %c10 = arith.constant 10 : i32
5
6    %init = list.from_elements %c0 : (i32) -> !list.list<i32>
7    %list = scf.for %i = %c0 to %c10 step %c1
8        iter_args(%current = %init) -> !list.list<i32> : i32 {
9      %new_list = list.push_front %current, %i : !list.list<i32>
10     scf.yield %new_list : !list.list<i32>
11   }
12
13   %rest = list.pop_front %list : !list.list<i32>
14   %next = list.peek_front %rest : !list.list<i32> -> i32
15   %check = arith.cmpi ult, %next, %c10 : i32
16   return %check : i1
17 }
```

No ranges are propagated through list operations

Extending IntegerRangeAnalysis

```
1  class InferIntRangeInterface {
2  public:
3    // All operand ranges are known.
4    void inferResultRanges(
5        ArrayRef<ConstantIntRanges> operandRanges,
6        SetIntRangeFn setResultRange);
7
8    // Operand ranges may be uninitialized / unknown.
9    void inferResultRangesFromOptional(
10       ArrayRef<IntegerValueRange> operandRanges,
11       SetIntLatticeFn setResultRange);
12 };
```

Most analyses define their own interface

Extending IntegerRangeAnalysis

```
1 // ListDialect.td
2 def List_LengthOp : List_Op<"length", [
3     Pure,
4     DeclareOpInterfaceMethods<InferIntRangeInterface,
5                               ["inferResultRanges"]>
6   ]> {
```

Recall how to add an interface in MLIR.

Exercise: Extending IntegerRangeAnalysis

```cpp
// ListOps.cpp
void FromElementsOp::inferResultRanges(ArrayRef<ConstantIntRanges> argRanges,
                                       SetIntRangeFn setResultRanges) {
  // Get the bitwidth of the list element type.
  unsigned elementWidth = ConstantIntRanges::getStorageBitwidth(
      getResult().getType().getElementType());

  // If the list is empty, the result range is set at an arbitrary constant (0).
  if (argRanges.empty()) {
    setResultRanges(getResult(), ConstantIntRanges::constant(APInt::getZero(elementWidth)));
    return;
  }

  // Otherwise, compute the union of all argument ranges.
  ConstantIntRanges resultRange = argRanges.front();
  for (const ConstantIntRanges &range : argRanges.drop_front())
    resultRange = resultRange.rangeUnion(range);
  setResultRanges(getResult(), resultRange);
}
```

Exercise: Extending IntegerRangeAnalysis

```cpp
// ListOps.cpp
void FromElementsOp::inferResultRanges(ArrayRef<ConstantIntRanges> argRanges,
                                       SetIntRangeFn setResultRanges) {
  // Get the bitwidth of the list element type.
  unsigned elementWidth = ConstantIntRanges::getStorageBitwidth(
      getResult().getType().getElementType());

  // If the list is empty, the result range is set at an arbitrary constant (0).
  if (argRanges.empty()) {
    setResultRanges(getResult(), ConstantIntRanges::constant(APInt::getZero(elementWidth)));
    return;
  }

  // Otherwise, compute the union of all argument ranges.
  ConstantIntRanges resultRange = argRanges.front();
  for (const ConstantIntRanges &range : argRanges.drop_front())
    resultRange = resultRange.rangeUnion(range);
  setResultRanges(getResult(), resultRange);
}
```

Implement this in ListOps.cpp

Debug with ./scripts/print-integer-range-analysis.sh

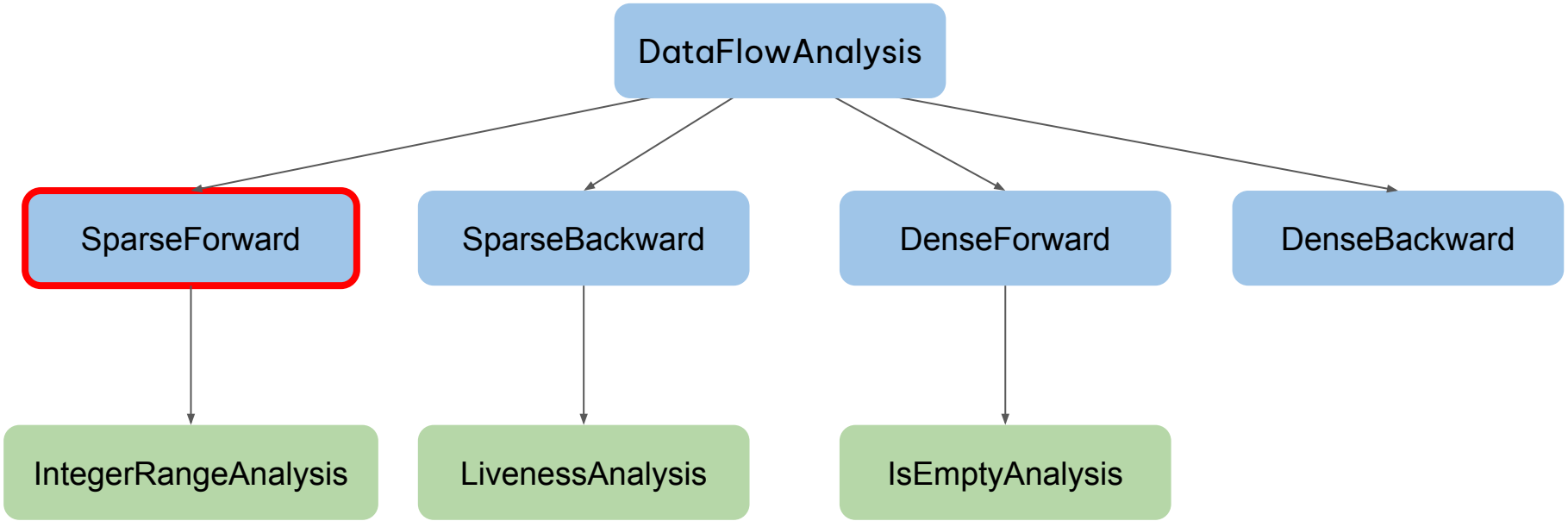
Test with ./scripts/test-integer-range.sh

Extended IntegerRangeAnalysis

```
func.func @loop_stack() -> (i1, i32, !list.list<i32>) {
  %c0_i32 = arith.constant {list.integer_ranges = {result0 = "signed : [0, 0]"}} 0 : i32
  %c1_i32 = arith.constant {list.integer_ranges = {result0 = "signed : [1, 1]"}} 1 : i32
  %c10_i32 = arith.constant {list.integer_ranges = {result0 = "signed : [10, 10]"}} 10 : i32

  %0 = list.from_elements %c0_i32 {list.integer_ranges = {result0 = "signed : [0, 0]"}} : (i32) -> !list.list<i32>
  %1 = scf.for %arg0 = %c0_i32 to %c10_i32 step %c1_i32 iter_args(%arg1 = %0) -> (!list.list<i32>)  : i32 {
      %5 = list.push_front %arg1, %arg0 {list.integer_ranges = {result0 = "signed : [0, 9]"}} : !list.list<i32>
      scf.yield %5 : !list.list<i32>
  } {list.integer_ranges = {result0 = "signed : [0, 9]"}}

  %2 = list.pop_front %1 {list.integer_ranges = {result0 = "signed : [0, 9]"}} : !list.list<i32>
  %3 = list.peek_front %2 {list.integer_ranges = {result0 = "signed : [0, 9]"}} : !list.list<i32> -> i32
  %4 = arith.cmpi ult, %3, %c10_i32 {list.integer_ranges = {result0 = "signed : [-1, -1]"}} : i32
  return %4, %3, %2 : i1, i32, !list.list<i32>
}
```

Implementing your own sparse forward analysis

MLIR Dataflow Analysis hierarchy



A generic framework for analyses

SparseForwardDataFlowAnalysis provides:

- Information storage
- Fixpoint computation

User provides:

- Type of information stored
- Initialization
- Propagation of information

Goal: SizeRangeAnalysis

```
%empty = list.empty : !list.list<i32>                          // %empty : [0]
%list = scf.if %condition -> (!list.list<i32>) {               // %list : [0, 1]
  scf.yield %empty : !list.list<i32>
} else {
  %one = list.push_back %empty, %value : !list.list<i32>       // %one : [1, 1]
  scf.yield %one : !list.list<i32>
}
%length = list.length %list : !list.list<i32> -> i32
```

dataflow::Lattice

```
1 class SizeRangeValue {
2   // Join the information of two SizeRangeValues.
3   SizeRangeValue SizeRangeValue::join(const SizeRangeValue &lhs,
4                                       const SizeRangeValue &rhs);
5 };
6
7 class SizeRangeLattice : public dataflow::Lattice<SizeRangeValue> {
8   // Forward to the join function of SizeRangeValue.
9   ChangeResult join(const dataflow::AbstractSparseLattice &rhs);
10  ChangeResult join(const SizeRangeValue &rhs);
11 }
12
13 class SizeRangeAnalysis : public dataflow::SparseForwardDataFlowAnalysis<SizeRangeLattice>;
```

Mathematics of dataflow::Lattice

- A Lattice represents a set of values
  - e.g. [0, 2] represents all lists of size {0, 1, 2}
- The join of two Lattices should be the smallest Lattice that represents the union of the values.
  - e.g. [0, 2] join [4, 5] = [0, 5]

Exercise: Implementing SizeRangeValue::join

Implement this in SizeRangeAnalysis.cpp

Solution of join

DataflowAnalysis class

```
class SizeRangeAnalysis
    : public dataflow::SparseForwardDataFlowAnalysis<SizeRangeLattice> {

  LogicalResult visitOperation(Operation *op, ArrayRef<const SizeRangeLattice *> operands,
                               ArrayRef<SizeRangeLattice *> results) override;

  void setToEntryState(SizeRangeLattice *lattice) override;
};
```

Exercise: Implement setToEntryState

```
class SizeRangeAnalysis
    : public dataflow::SparseForwardDataFlowAnalysis<SizeRangeLattice> {

  LogicalResult visitOperation(Operation *op, ArrayRef<const SizeRangeLattice *> operands,
                               ArrayRef<SizeRangeLattice *> results) override;

  void setToEntryState(SizeRangeLattice *lattice) override;
};
```

Exercise: Implement visitOperation

```
LogicalResult
SizeRangeAnalysis::visitOperation(Operation *op,
                                  ArrayRef<const SizeRangeLattice *> operands,
                                  ArrayRef<SizeRangeLattice *> results) {
  if (isa<list::EmptyOp>(op)) {
    set(results.front(), SizeRangeValue::getExact(0));
    return success();
  }
  ...
}
```

Solution: Implement visitOperation

```
LogicalResult
SizeRangeAnalysis::visitOperation(Operation *op,
                                  ArrayRef<const SizeRangeLattice *> operands,
                                  ArrayRef<SizeRangeLattice *> results) {
  if (isa<list::EmptyOp>(op)) {
    set(results.front(), SizeRangeValue::getExact(0));
    return success();
  }
  ...
}
```

Running and Using an Analysis

How to use dataflow analyses

```
1 DataFlowSolver solver;
2 dataflow::loadBaselineAnalyses(solver);
3 solver.load<dataflow::IntegerRangeAnalysis>();
4
5 if (failed(solver.initializeAndRun(getOperation()))) {
6   getOperation()->emitError("failed to run integer range analysis");
7   signalPassFailure();
8   return;
9 }
```

How to use dataflow analyses

```
1 solver.lookupState<IntegerValueRangeLattice>(value);
```

How to use dataflow analyses

```
1 solver.lookupState<IntegerValueRangeLattice>(value);
```

Be careful when querying values after modifying the IR!

How to update dataflow analyses

```
1 static void copyIntegerRange(DataFlowSolver &solver, Value oldVal, Value newVal) {
2   auto *oldState = solver.lookupState<IntegerValueRangeLattice>(oldVal);
3   if (!oldState)
4     return;
5   solver.getOrCreateState<IntegerValueRangeLattice>(newVal)->join(*oldState);
6 }
```

Implement analysis-based optimizations

- Replace empty lists with `list.empty`

Implement analysis-based optimizations

Solution

"Unbounded" lattices

Do you see an issue here?

```
1  func.func @push_back_loop(%upper_bound: i32) -> !list.list<i32> {
2    %c0 = arith.constant 0 : i32
3    %c1 = arith.constant 1 : i32
4
5    %empty = list.empty : !list.list<i32>
6    %list = scf.for %i = %c0 to %upper_bound step %c1
7        iter_args(%current = %empty) -> !list.list<i32> : i32 {
8      %next = list.push_back %current, %i : !list.list<i32>
9      scf.yield %next : !list.list<i32>
10   }
11
12   return %list : !list.list<i32>
13 }
```

Solution : Widening

```
// In SizeRangeLattice::join
SizeRangeValue joined = SizeRangeValue::join(current, rhs);

if (!current.isUninitialized() && current.hasUpperBound() &&
    (!joined.hasUpperBound() || *joined.getUpper() > *current.getUpper())) {
  ++upperBoundGrowths;
  if (upperBoundGrowths >= 4)
    joined = SizeRangeValue::getRange(joined.getLower(), std::nullopt);
}

```

Combining analyses

Combining analyses is very powerful

```
1  func.func @sccp_example(%unknown: i32) -> i32 {
2    %true = arith.constant true
3    %c42 = arith.constant 42 : i32
4    cf.cond_br %true, ^then, ^else
5
6  ^then:
7    cf.br ^merge(%c42 : i32)
8
9  ^else:
10   %x = arith.addi %unknown, %c42 : i32
11   cf.br ^merge(%x : i32)
12
13 ^merge(%result: i32):
14   return %result : i32
15 }
```

Dead code analysis needs information from ConstantAnalysis

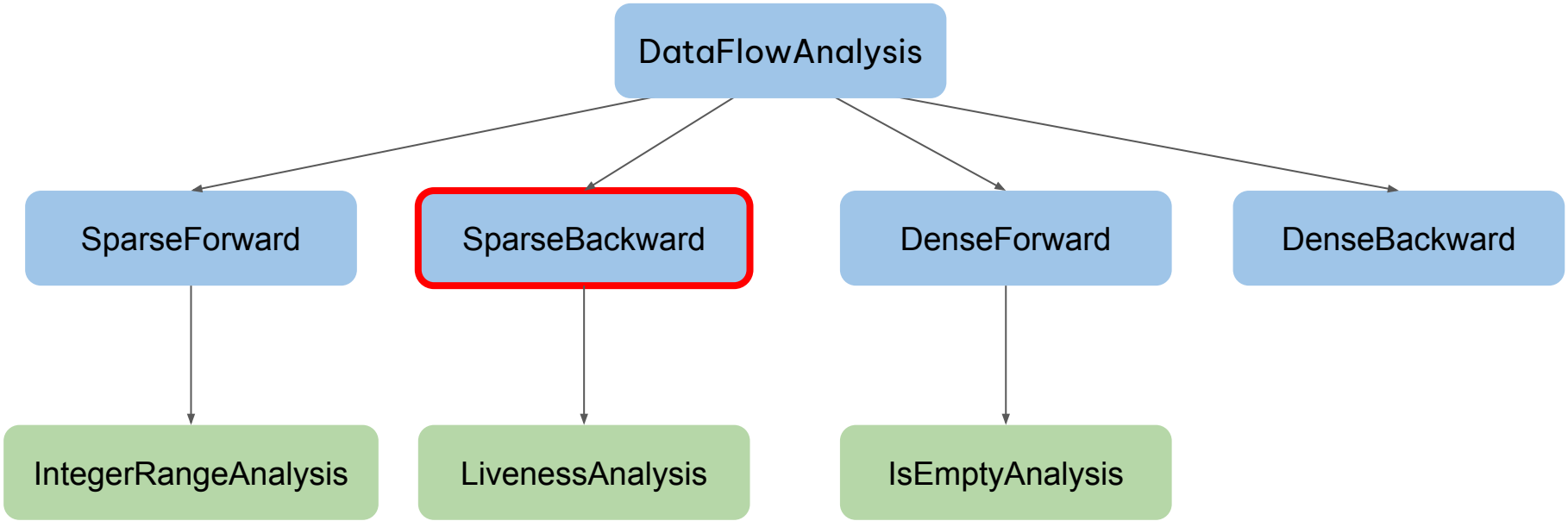
Combining analyses in MLIR

In `MyAnalysis::visitOperation`:

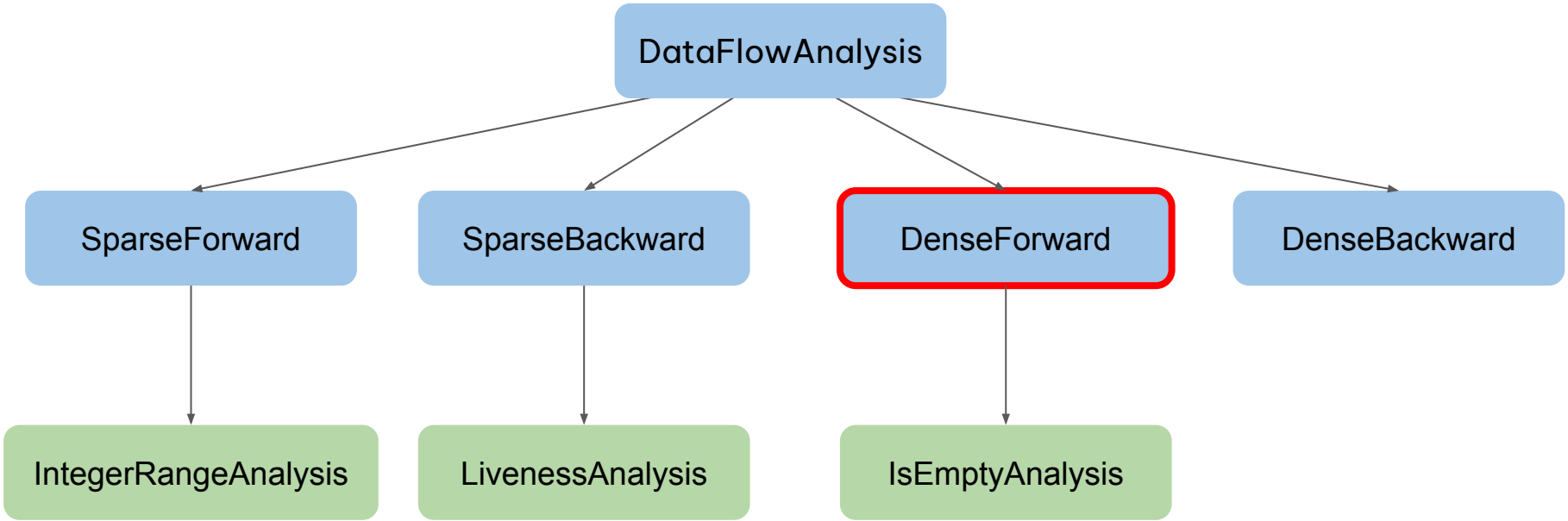
```
// Register a dependency so this transfer function is revisited whenever the
// size analysis learns more about the input list.
const auto *size = getOrCreateFor<SizeRangeLattice>(getProgramPointAfter(op), input);
const SizeRangeValue &sizeRange = size->getValue();

if (sizeRange.isUninitialized())
  return success();
...
```

Other kind of Dataflow Analysis

MLIR Dataflow Analysis hierarchy



MLIR Dataflow Analysis hierarchy



DenseForwardAnalysis

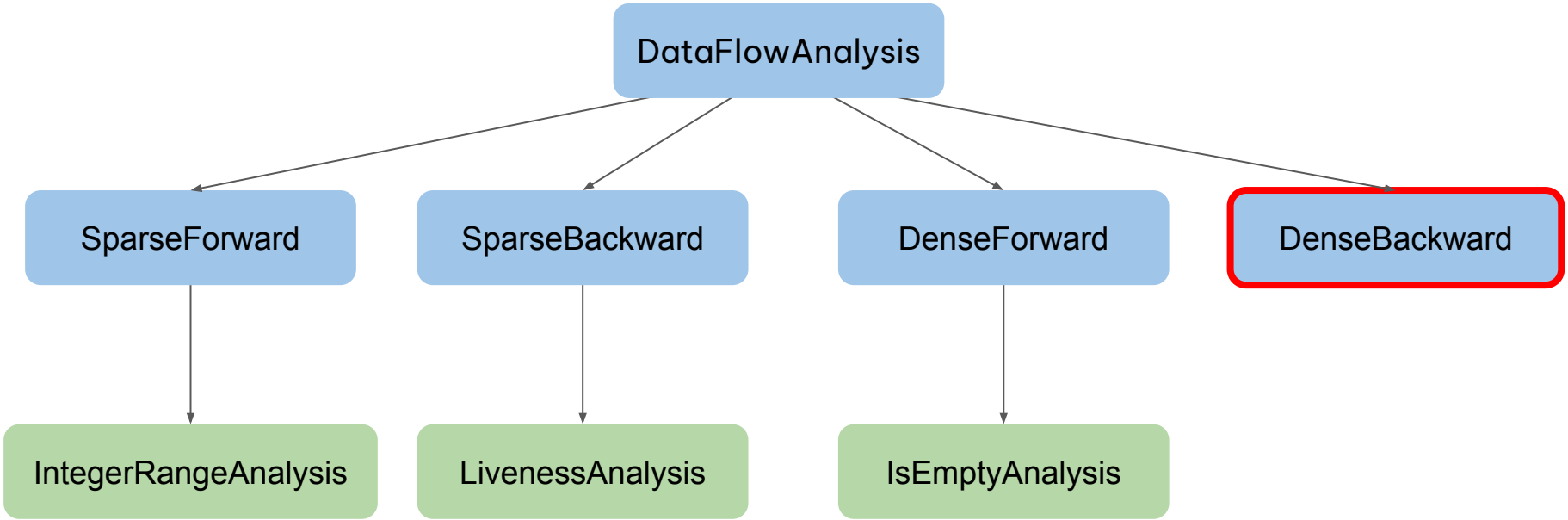
Dense analyses are control-flow aware

```
1  %list = ...                                                  // {}
2  %len = list.length %list : !listlang.list -> index           // {}
3  %c0 = arith.constant 0 : index                               // {}
4  %is_empty = arith.cmpi eq, %len, %c0 : index                 // {}
5  scf.if %is_empty {                                           // {}
6    %len2 = list.length %list : !listlang.list -> index        // {%list}
7    use %len2                                                  // {%list}
8  } else {
9    ...                                                        // {}
10 }
```

DenseForwardAnalysis

```
class ExampleDenseAnalysis
    : public dataflow::DenseForwardDataFlowAnalysis<ExampleDenseLattice> {

  LogicalResult visitOperation(Operation *op, const ExampleDenseLattice &before,
                               ExampleDenseLattice *after) override;

  void visitBlockTransfer(Block *block, ProgramPoint *point, Block *predecessor,
                          const ExampleDenseLattice &before,
                          ExampleDenseLattice *after) override;

  void visitRegionBranchControlFlowTransfer(
      RegionBranchOpInterface branch, std::optional<unsigned> regionFrom,
      std::optional<unsigned> regionTo, const ExampleDenseLattice &before,
      ExampleDenseLattice *after) override;

  void setToEntryState(ExampleDenseLattice *lattice) override;
};
```

MLIR Dataflow Analysis hierarchy



MLIR Dataflow Analysis hierarchy