Modular

# Efficient Data-Flow Analysis on Region-Based Control Flow in MLIR

Weiwei Chen
weiwei.chen@modular.com
EuroLLVM 2024

# Agenda

01 Data-flow Analysis

02 Region-based Control Flow Representation in MLIR

03 An Efficient SCCP

04 Conclusions

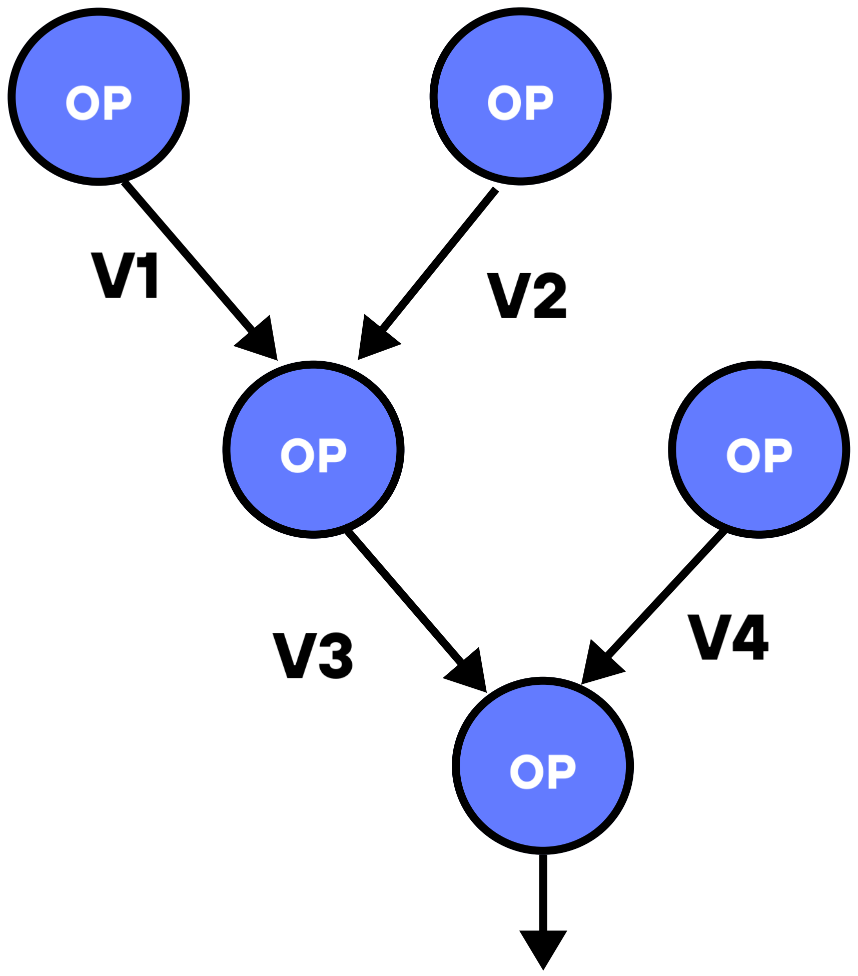
05 Questions?

# Data-flow Analysis

- Gathers information that is propagated along the control-flow graph (CFG) of a program.
- Static analysis that covers all the edges of how data is flowed in the program.
- Analysis states can be use for optimizations like Sparse Conditional Constant Propagation (SCCP), Value Range Analysis, Bit-Vector Analysis, etc.

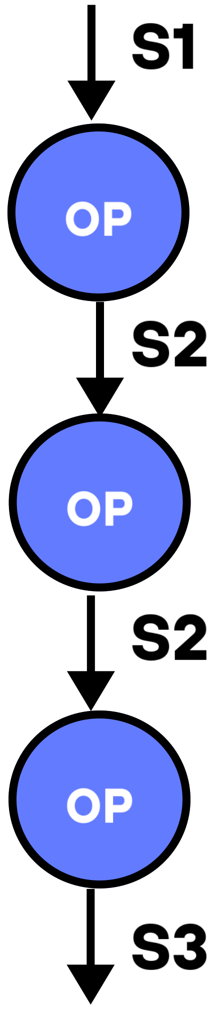
# Classic Data-flow Analysis

**Sparse**




$$S(v_{i+2}) = f_{op}(S(v_i), S(v_{i+1}))$$

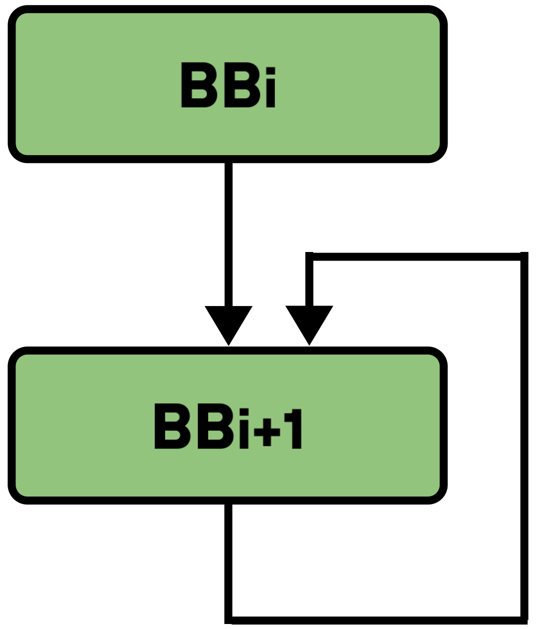
**Dense**




$$S_{i+1} = f_{op}(S_i)$$

[MLIR Dataflow Analysis](#) by Jeff Niu, Tom Eccles, *2023 EuroLLVM*

# Classic Data-flow Analysis States




### Sparse

$$S(arg(BB_i, n)) = S(out(BB_i, n)) \mathbf{V}\ S(out(BB_{i+1}, n))$$

### Dense

$$S(begin(BB_i)) = S(end(BB_i) \mathbf{V}\ S(end(BB_{i+1}))$$

### Analysis State Lattice

- $\bot$: uninitialized (bottom)
- $\top$: over-defined (top)
- $\vee$: join (union)
- $\wedge$: meet (intersect)
- $X$: lattice element

$$\top \vee \mathbf{X} = \top$$
$$\bot \vee \mathbf{X} = \mathbf{X}$$
$$\top \wedge \mathbf{X} = \mathbf{X}$$
$$\bot \wedge \mathbf{X} = \bot$$

$$\mathbf{X}_i \vee \mathbf{X}_j = \text{unique UB}(X_i, X_j)$$
$$\mathbf{X}_i \wedge \mathbf{X}_j = \text{unique LB}(X_i, X_j)$$

### Boolean Constraints[1]

[1] Using the Clang data-flow framework for null-pointer analysis by Viktor Cseh, *2023 EuroLLVM*.

# Data-flow Analysis in LLVM and MLIR

- **LLVM:**
  - SCCP, IPSCCP, etc.
  - SCCPSolver, Clang Dataflow framework[1]
- **MLIR:**
  - Dead Code Analysis, IntegerRangeAnalysis, LivenessAnalysis, etc.
  - Extensible and composable DataFlowFrameWork[2]

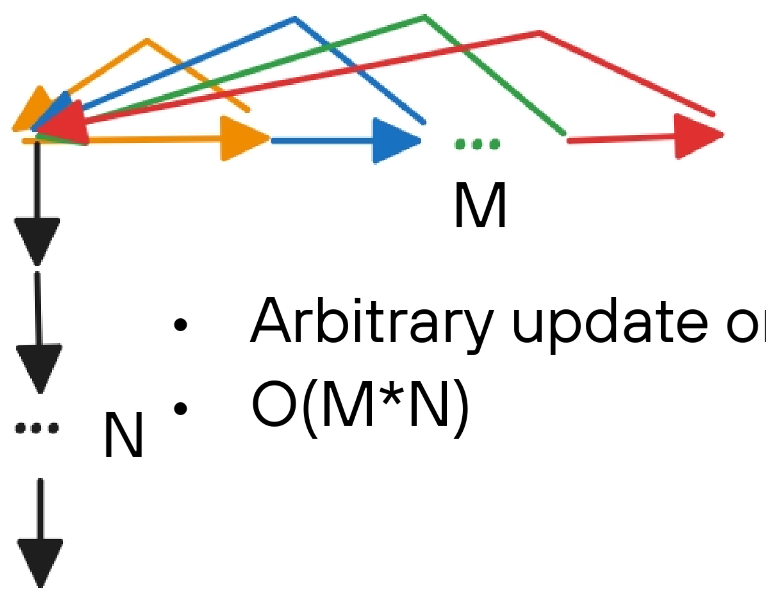
## Analysis follows the **general** control flow graph (CFG):

- $\phi$ nodes add complexity 🤕
- CFG can be irreducible 🤯
- Logically difficult to debug 😵‍💫

## Iterates an analysis state solver to fix point:

$$\mathbf{S_{i+1}(P_{n+1}) = S_i(P_{n+1}) \vee f_{op}(S_i(P_n))}$$

$$\mathbf{S_{i+1}(P_{n+1}) == S_i(P_{n+1})}$$




- Arbitrary update order
- O(M*N)

[1] Data flow analysis: an informal introduction *Clang Documentation.*

[2] MLIR Dataflow Analysis by Jeff Niu, Tom Eccles, *2023 EuroLLVM.*

# Region-based Control Flow Representation in MLIR

- Structured Control Flow Representation (like mlir.scf)
- Support early exits:
  - break, continue.
  - exits in the middle of basic blocks.
  - pure region-based representation.
- No arbitrary control flow, only branch back to parent region(s).
- High-level control flow representation matches well with program logic.
- Easy for frontends to emit directly, i.e.Mojo🔥

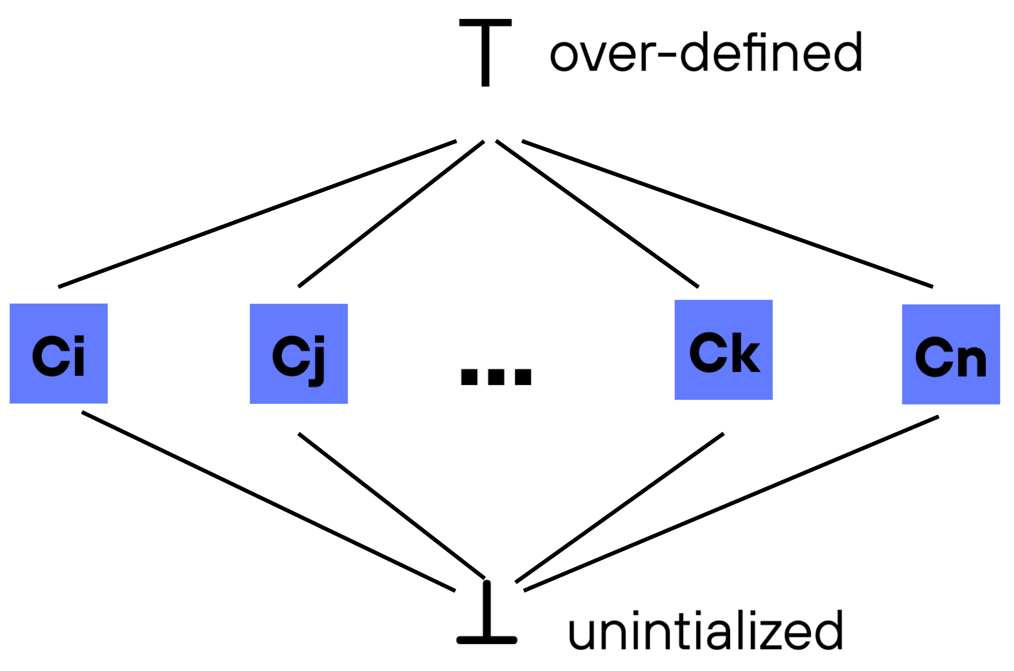
```
func.func @foobar() {
  rcf.loop {
    %0 = call @rand_bool() : () → i1
    rcf.if %0 {
      rcf.break
    }
    call @do_something() : () → ()
    rcf.continue
  }
  return
}
```

# Region-based Control Flow Representation in MLIR

- Region operations:
  rcf.loop, rcf.if, rcf.for, …
- Region terminators:
  rcf.yield, rcf.break, rcf.continue
- Control flow interfaces for passes use abstraction.
- Co-exist with CFG and mlir.scf.

```
class RCFNode : public mlir::OpInterface<ControlFlowNode, …> {
public:
  /// Given potential constant values of the operands of this operation, return
  /// the indices of the entry region of the operation, which is the region to
  /// the beginning of which control-flow branches upon visiting the start of
  /// this operation, and the operands with which to branch to that region.
  /// Return `None` to indicate that control-flow branches directly to after the
  /// operation.
  void getEntryTargets(ArrayRef<Attribute> operands,
                       SmallVectorImpl<RCFTarget> & targets);

  /// Verifier.
  static mlir::LogicalResult verify(mlir::Operation *op);

  …
};

class RCFTerminator : public mlir::OpInterface<ControlFlowTerminator, …> {
public:
  /// This method is invoked on the proper ancestors of a control-flow
  /// terminator to determine the nearest valid parent operation. The method
  /// should return true if the provided operation is a valid parent operation
  /// to the terminator, and false to keep searching.
  bool isParentNode(Operation * op);

  /// Return the branch target of the terminator relative to its control-flow
  /// parent and the operands with which to branch to that region. For instance,
  /// to branch to the beginning of the first region, the method should return
  /// `0`. To branch to the after the parent operation, the method should return
  /// `None`.
  void getBranchTargets(ArrayRef<Attribute> operands,
                        SmallVectorImpl<RCFTarget> & targets);

  /// Verifier.
  static mlir::LogicalResult verify(mlir::Operation *op);

  …
};
```

# **S**parse **C**onditional **C**onstant **P**ropagation 🐝

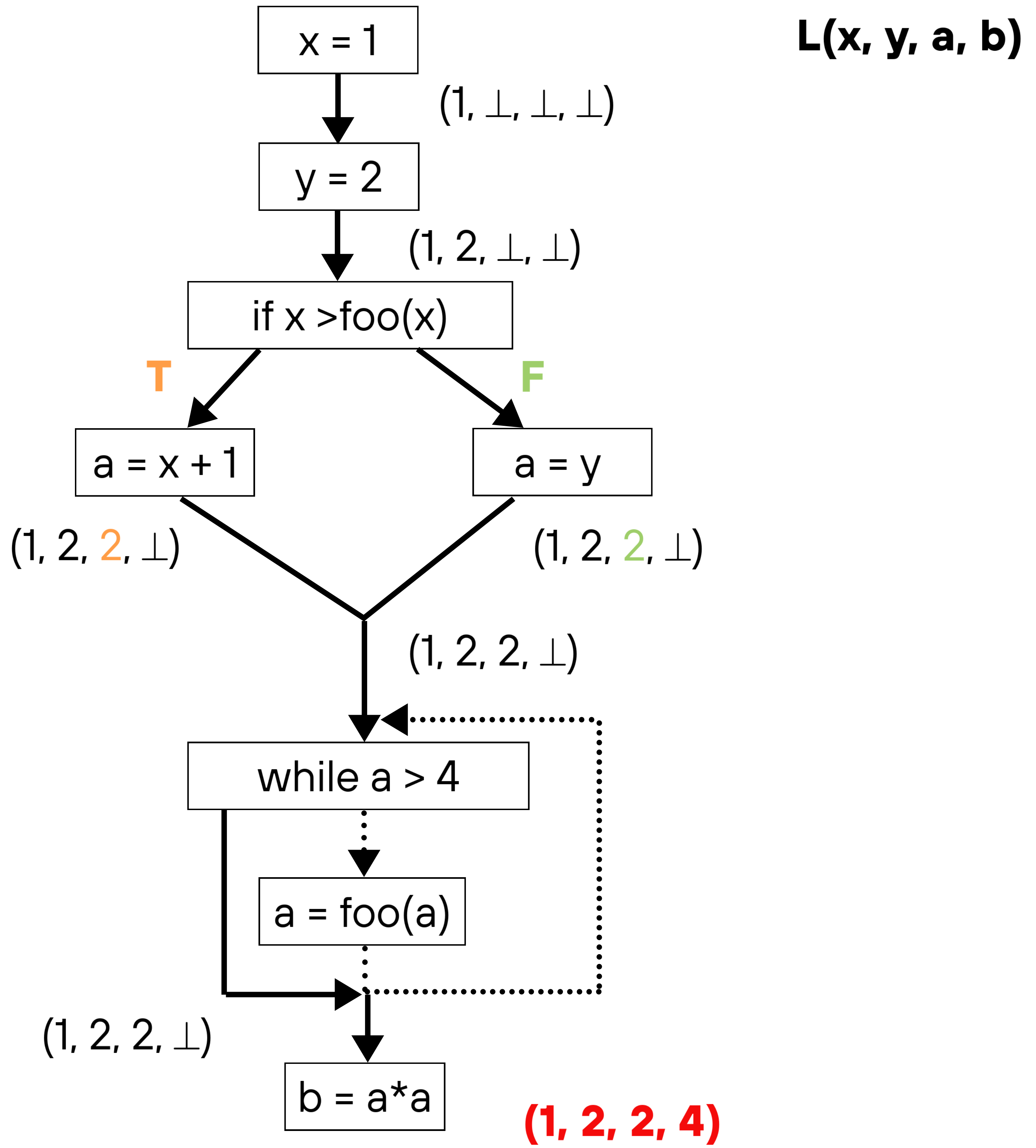



$\top \lor \text{any} = \top$

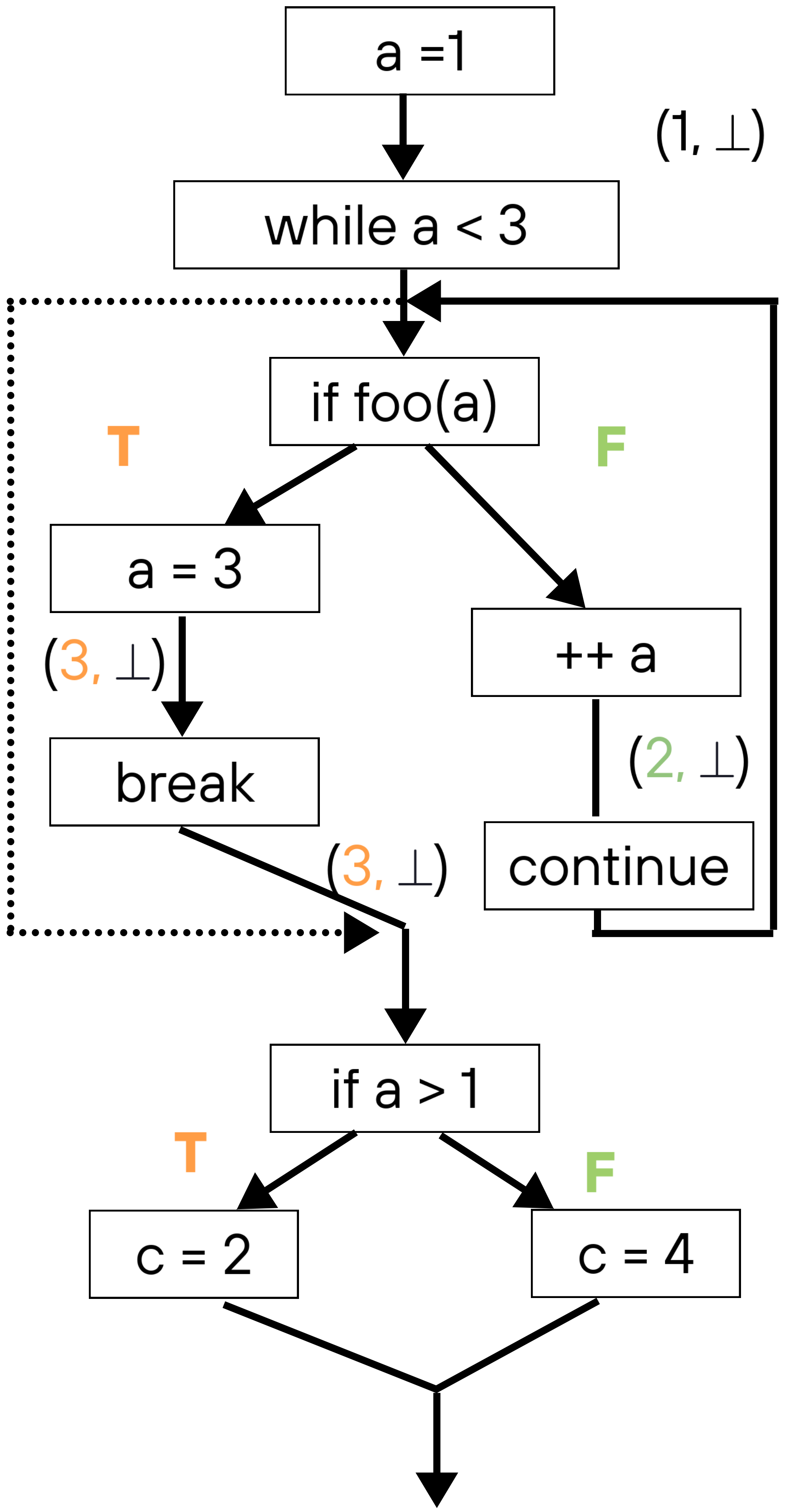
$\bot \lor \text{any} = \text{any}$

$C_i \lor C_k = C_i \text{ iff } C_i == C_k$

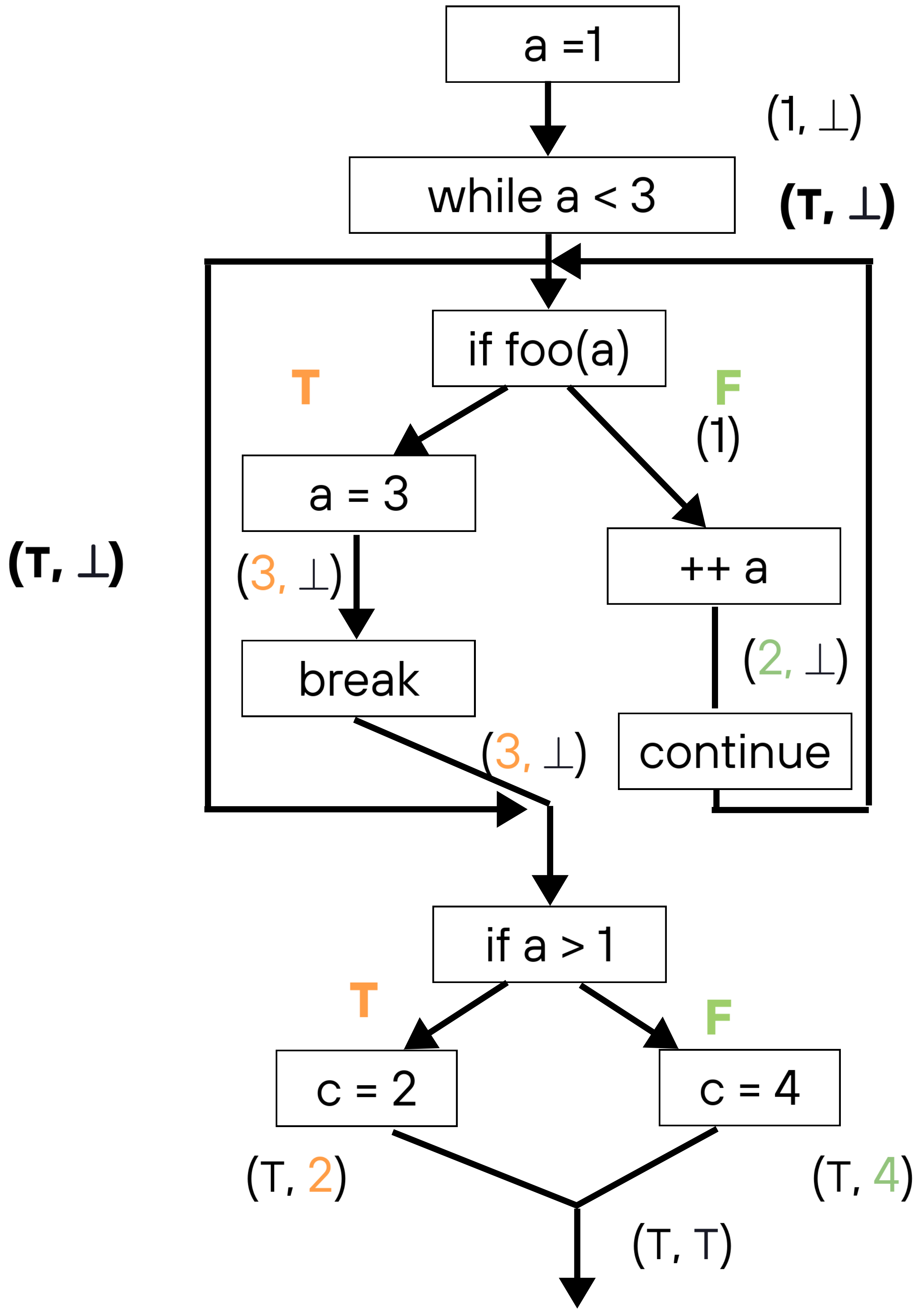
$C_i \lor C_k = \top \text{ iff } C_i \, != C_k$



# Sparse Conditional Constant Propagation 🐝

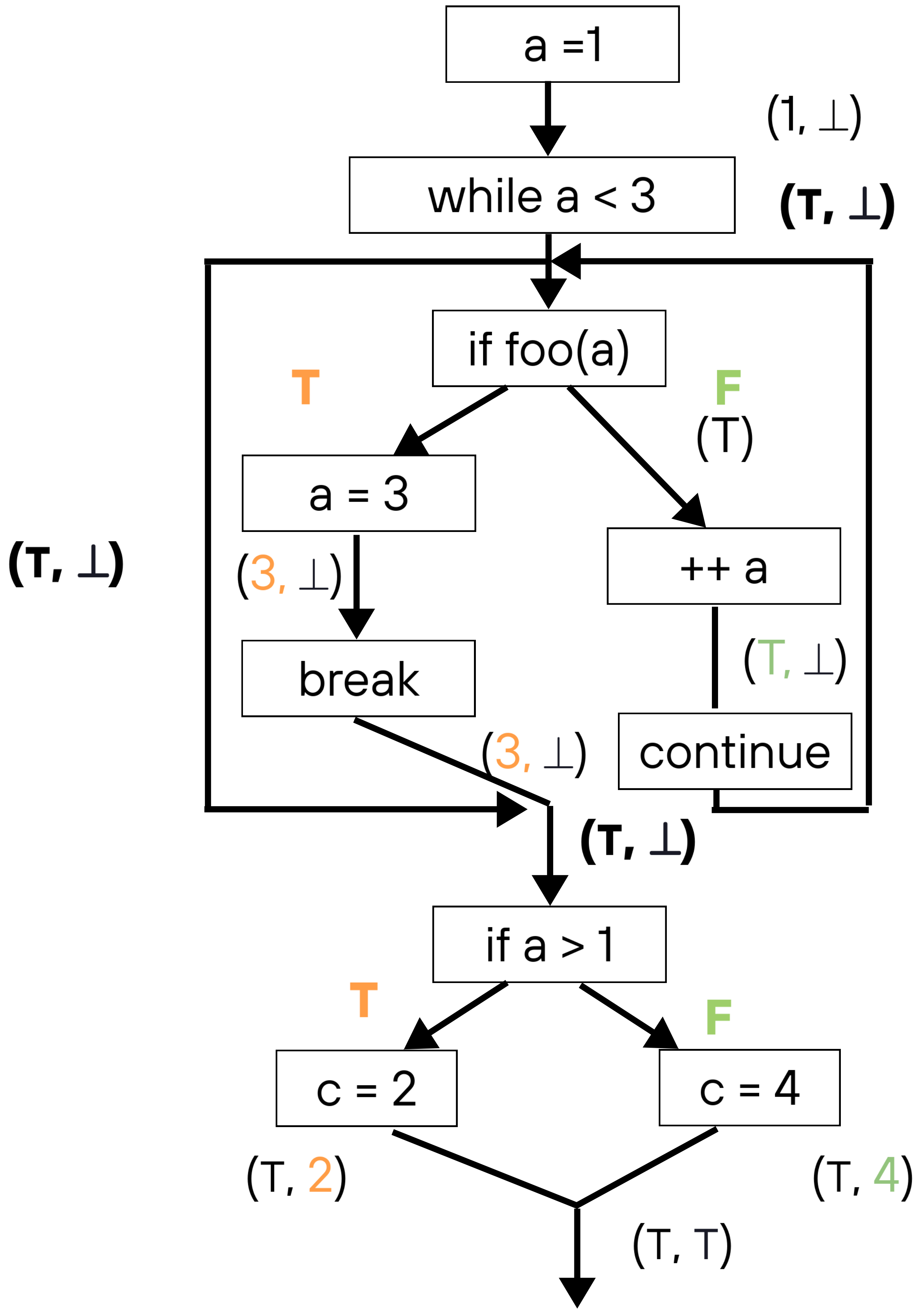


# Sparse Conditional Constant Propagation 🐝



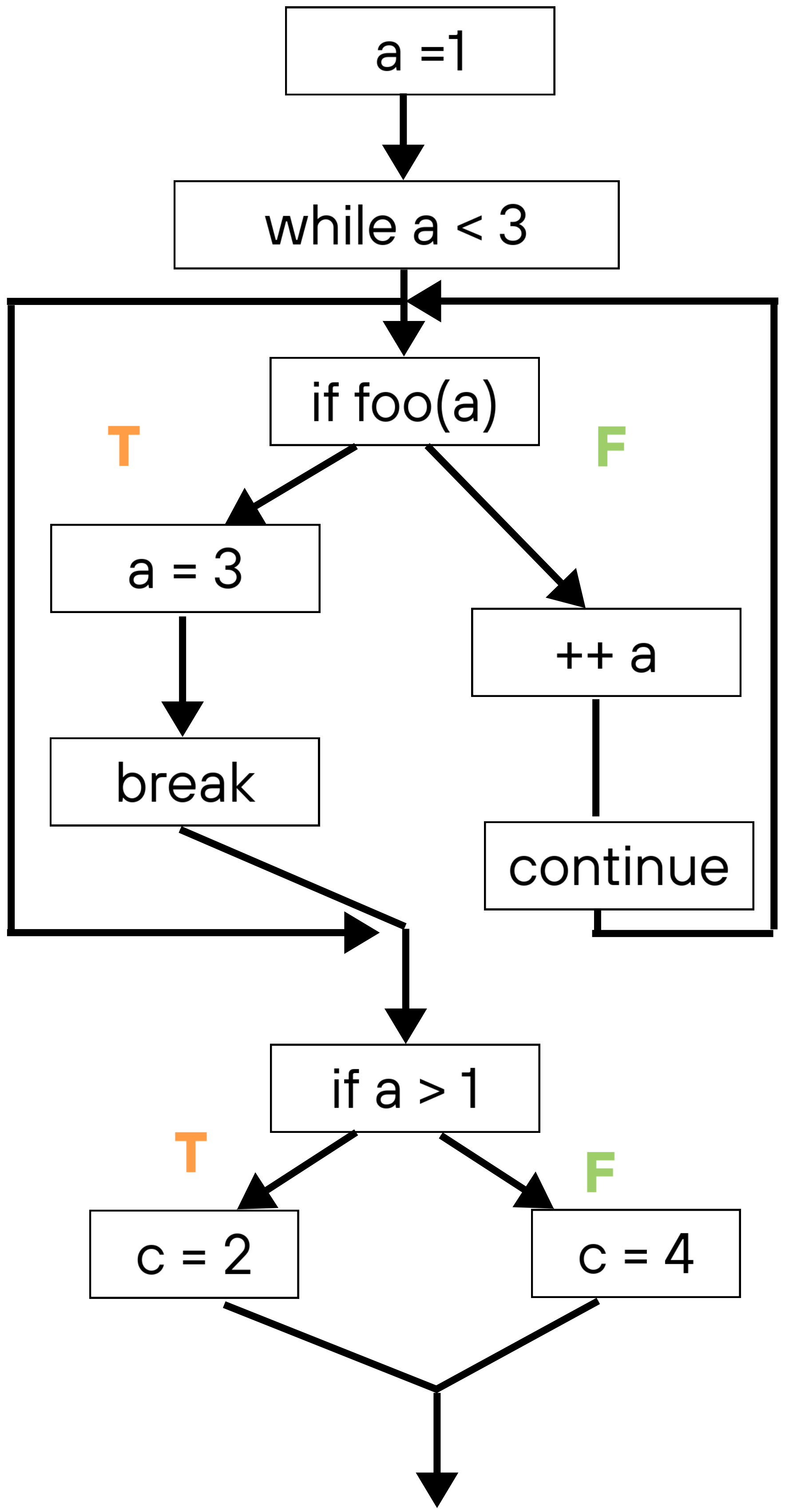

*c is over-defined*

# Sparse Conditional Constant Propagation 🐝




c is over-defined

# Sparse Conditional Constant Propagation 🐝

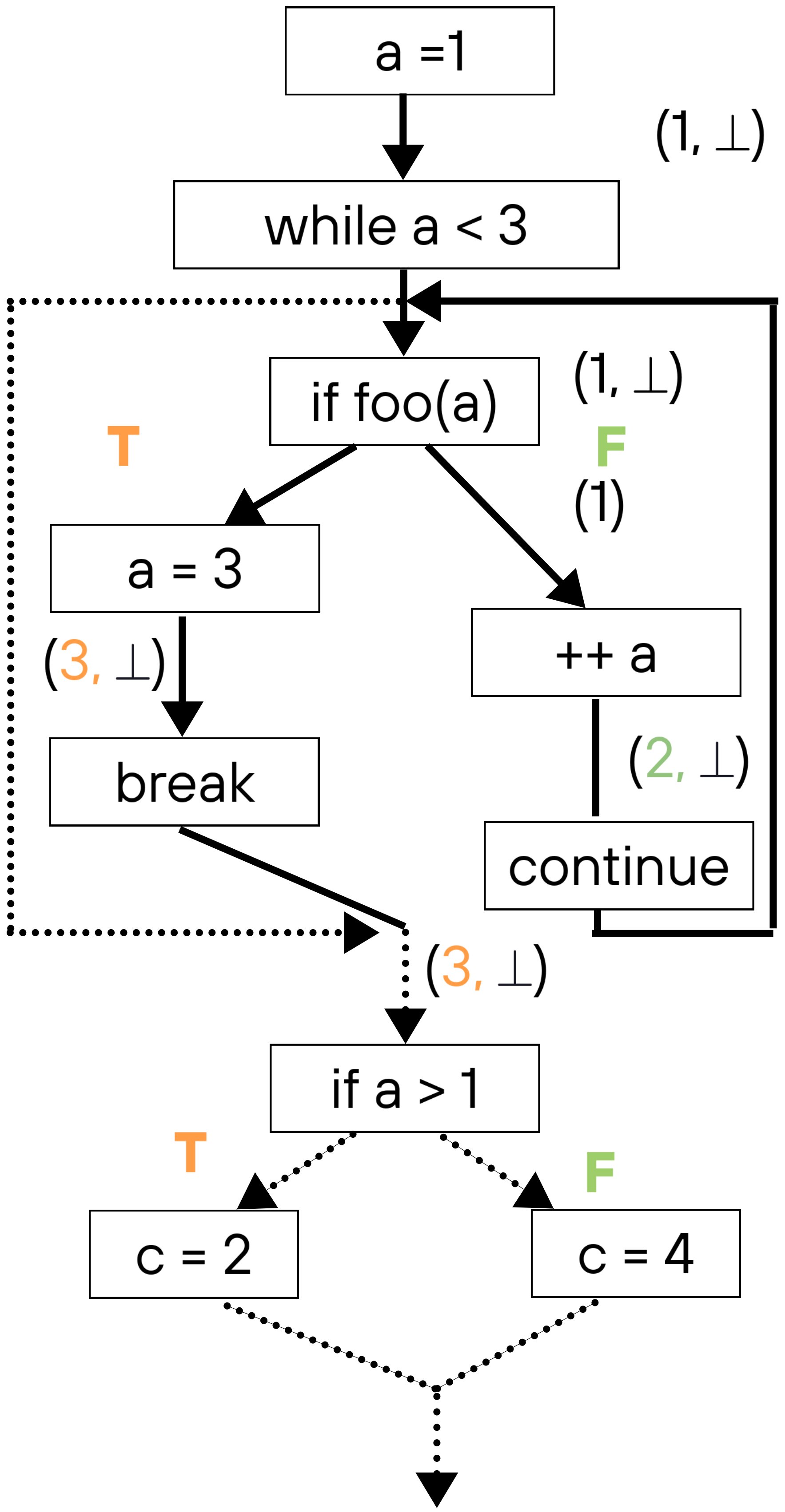



*c must be 2 ?*

# Sparse Conditional Constant Propagation 🐝

Interpret Loop based on control flow

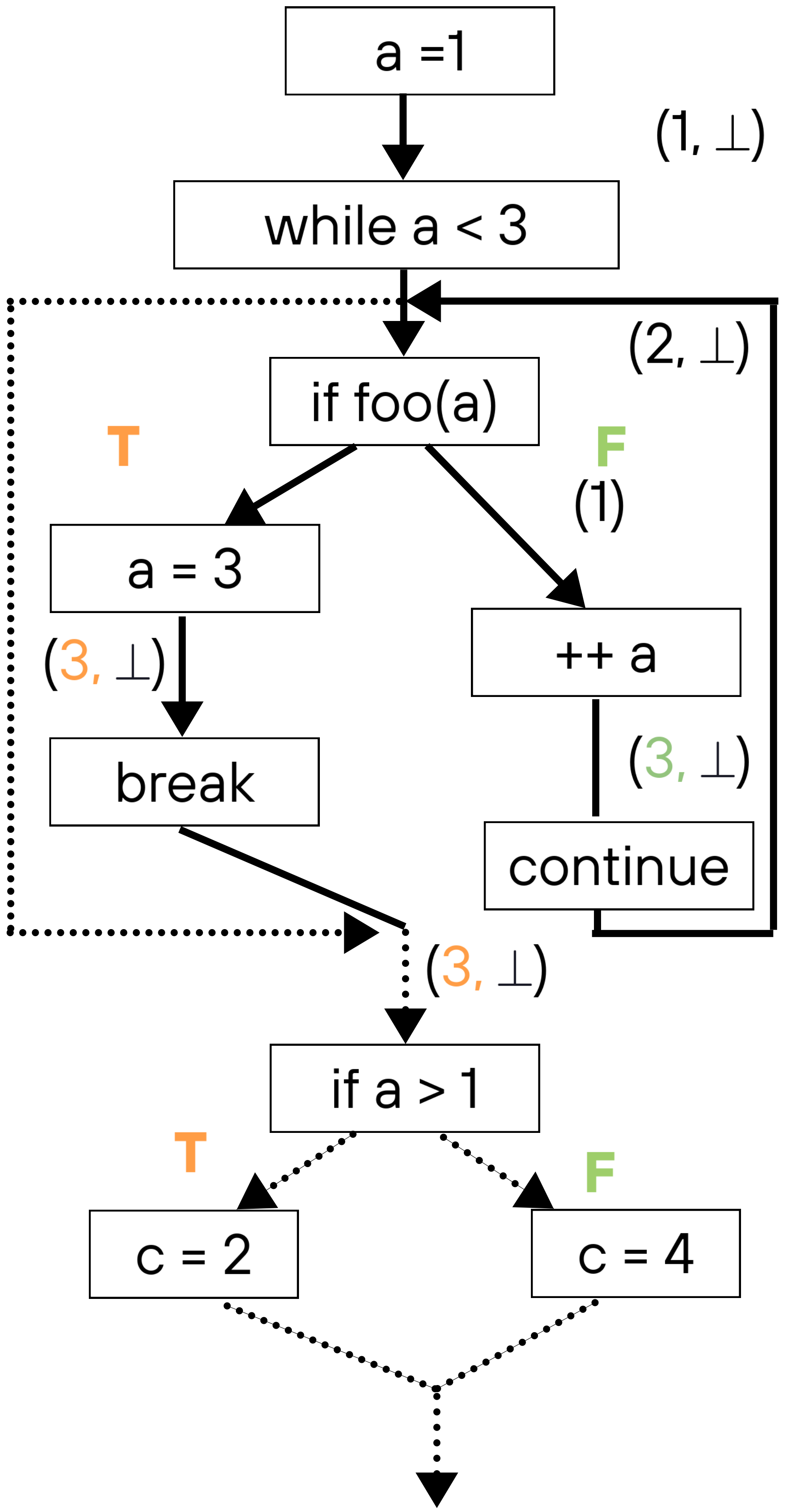
iter = 1



# Sparse Conditional Constant Propagation 🐝

Interpret Loop based on control flow

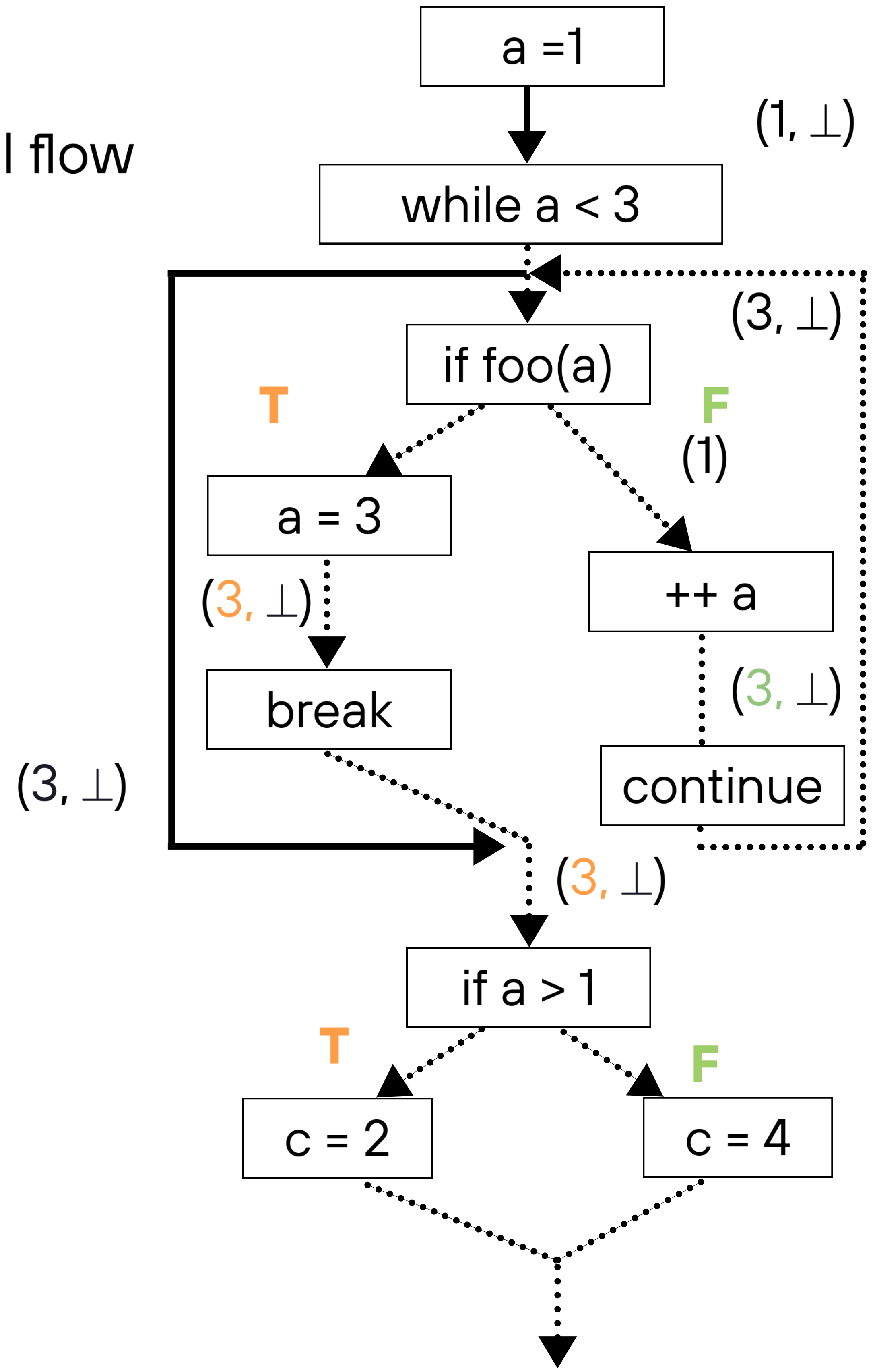
iter = 2



# Sparse Conditional Constant Propagation 🐝

Interpret Loop based on control flow

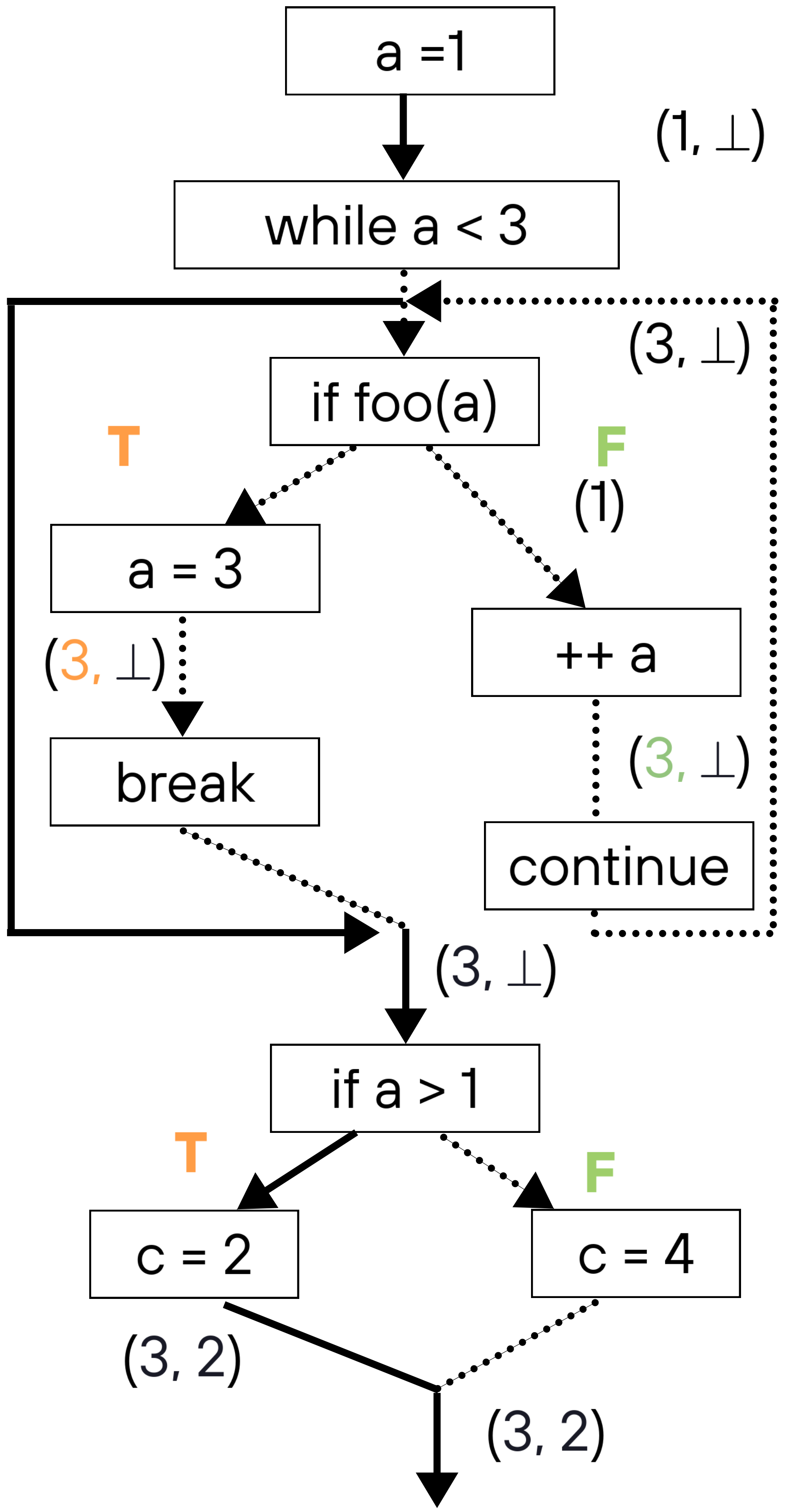
iter = 3



# Sparse Conditional Constant Propagation 🐝

Interpret Loop based on control flow

iter = 3

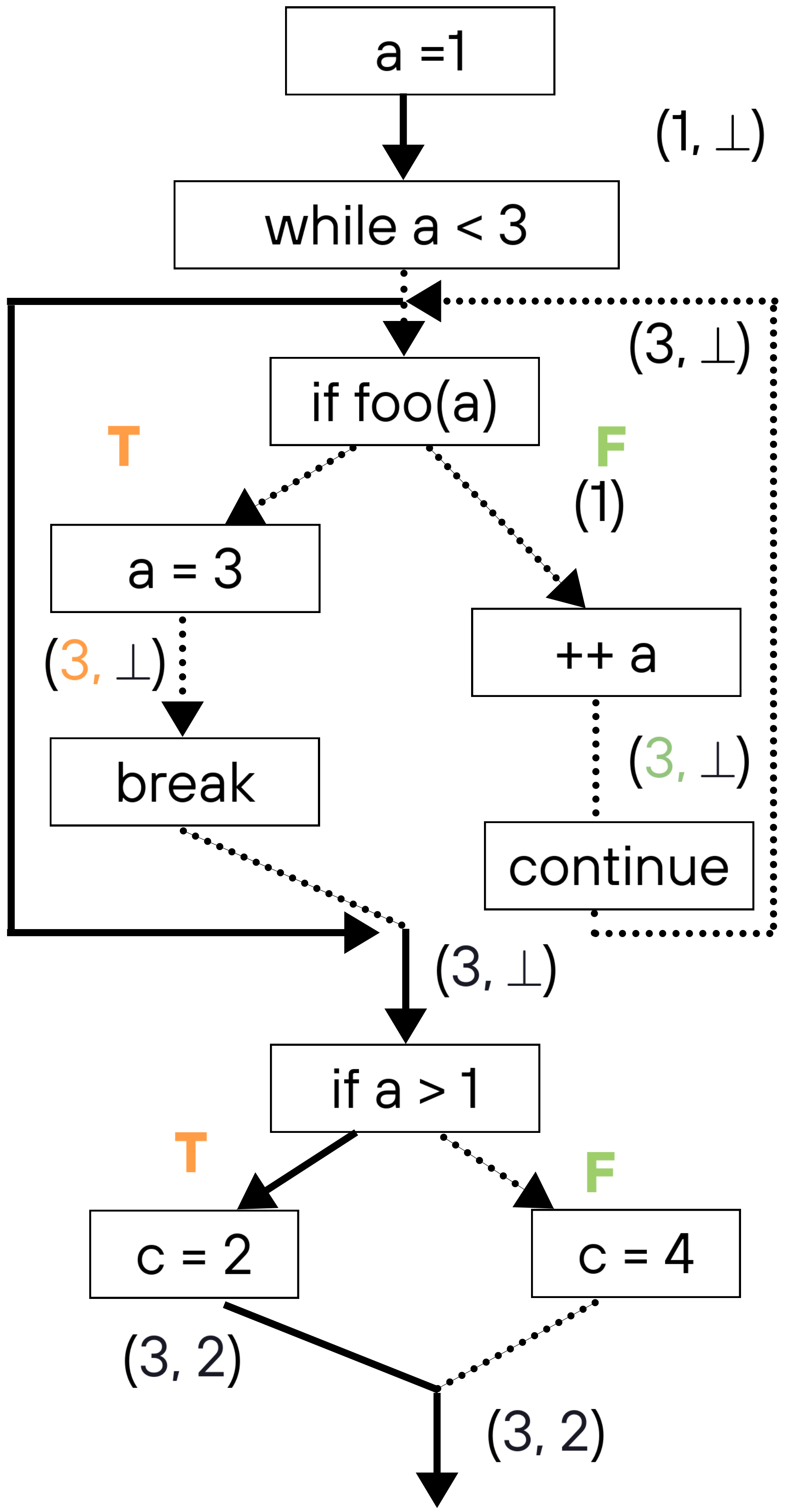



*c must be 2 !*

# Sparse Conditional Constant Propagation 🐝

Interpret Loop based on control flow

- More accurate result. ✅
- Can explode compilation time ⚠️
  - Large loop iterations.
  - Nested loops.
  - Use heuristics.

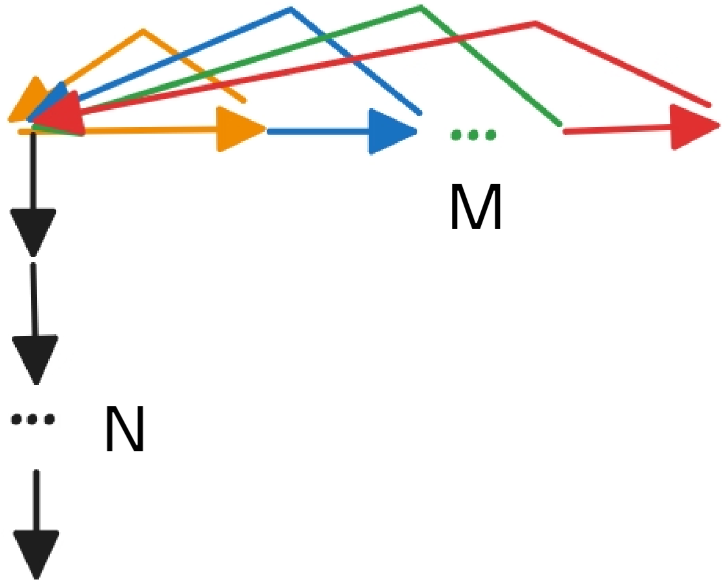



c must be 2 !

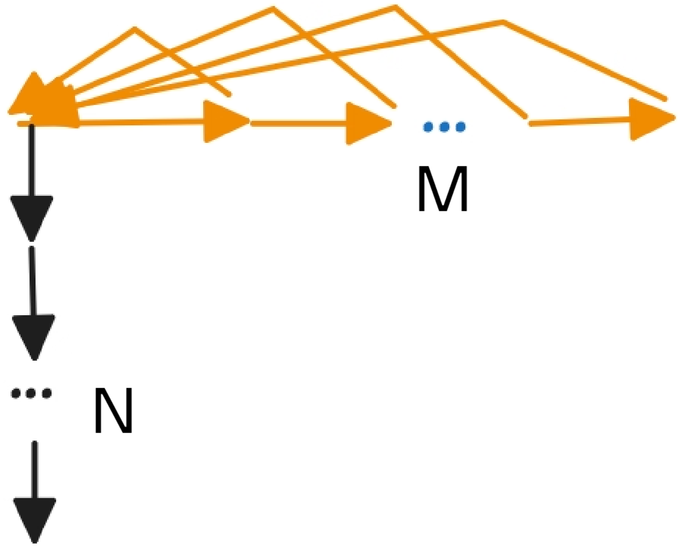
# SCCP on Region-based CF

Fix-point solver:

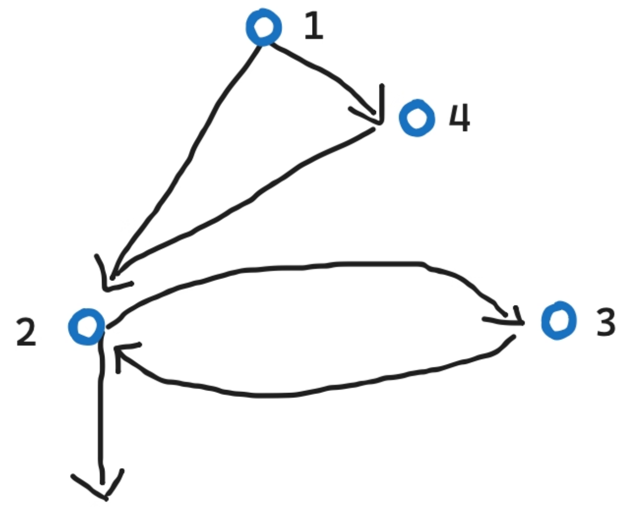
- Arbitrary update order
- O(M*N)



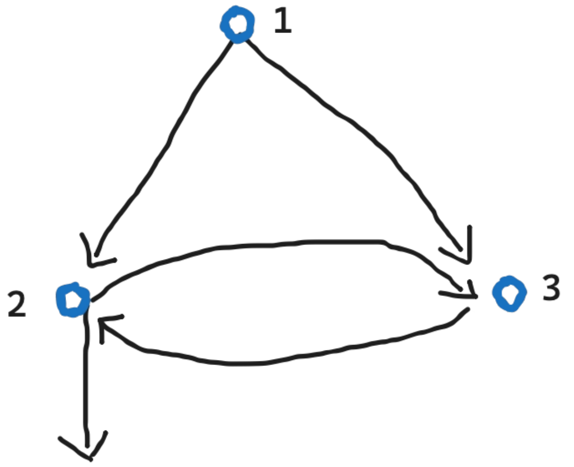

- Converge SCC first
- Then update tail
- O(M + N)




Reducible control flow ✅




Irreducible control flow 😖




- Loops are SCCs
- Forward linear analysis outside of SCCs
- For SCCs:
  - Localize analysis within SCC
  - Join SCC output and input states
  - Up to 2x linear analysis within SCC
- Complexity: O(2x #operations).
- Theoretical SCCP complexity: [1] O(# SSA edges) + O(# control flow edges)
- Heuristics based loop interpretation for better analysis.

[1] **Constant propagation with conditional branches** by M. Wegman, F. K. Zadeck
*ACM-SIGACT Symposium on Principles of Programming Languages*, January 1985.

# Experiments

| Model name | QPS wo/sccp | QPS w/sccp | Compilation Time (s) wo/sccp | Compilation Time (s) w/sccp |
|---|---|---|---|---|
| dlrm-rm2-multihot | 39.66 | 39.87 **(1.006x)** | **201.163 s** ± 0.475 s | **200.471 s** ± 0.481 s |
| resnet50-v1.5 | 110.29 | 111.59 **(1.012x)** | **38.092 s** ± 11.945 s | **38.316 s** ± 12.509 s |
| gpt2 | 124.71 | 125.34 **(1.005x)** | **30.873 s** ± 0.432 s | **29.189 s** ± 0.139 s |

**Benchmark environment:**

- c5n.metal
- Disable hyper-threading and turbo-boost.
- CPU freq: 2.9G Hz.

**Bechmark Methodology:**

- Run each model multiple time for a set period of time.
- Statistical results.

# Conclusions

- Structured region-based control flow representation:
  - Allows early exits.
  - Can co-exist with mlir.scf and CFG.
  - Reducible control flow that guarantees best case complexity for data-flow analysis.
  - Logically easy to debug due to close match to the high-level programing language.
  - Applicable to other efficient analyses: range value, bit-vector, memory scoping, ...
- We are planning to upstream:
  - Region-based control flow representation — RFC.
  - First-class support for successors and predecessors.
  - Data-flow analyses based on the control flow representation.

**[RFC] Region-based control-flow with early exits in MLIR**

MLIR

Mogball — Feb 14

**Region-based control-flow with early exits**

This RFC proposes the additional of a new region-based control-flow paradigm to MLIR, but one that enables early exits via operations like `break` or `continue` in contrast with SCF.

# Questions?