Control-Flow Graph

Lorenzo Ceragioli

November 13, 2024

IMT Lucca

Control-Flow Graph

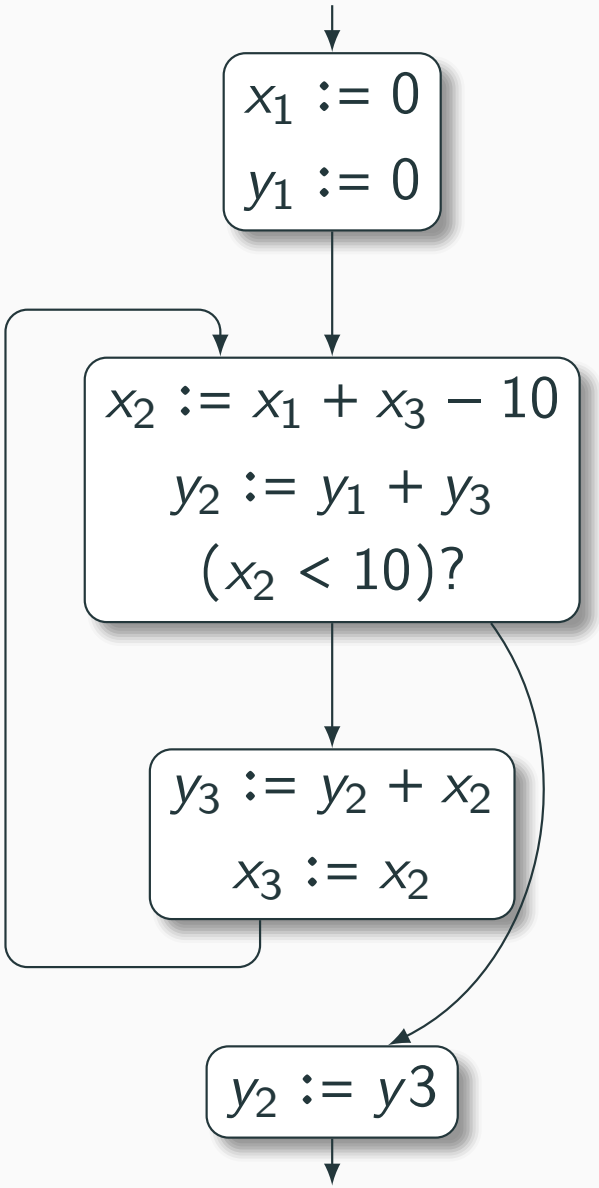
A direct graph where

- **Nodes** are basic blocks (sequence of simple statements)
- **Edges** are possible control flow

Building the CFG is an intermediate step for:

- Static Analysis (Data Flow)
- Compilation (we need to recover exact control flow)

Example



Formally

A directed graph $(N, next, i, f, code)$ where

- N **nodes** are *basic blocks*
- $next : N \longrightarrow \{none\} \cup N \cup (N \times N)$ **edges** are possible control flow
- $i \in N$ is the initial entry node
- $f \in N$ is the final terminal node
- $code : N \longrightarrow L$ (where L is some language, usually sequences of simple statements)

... Ours is a simple intraprocedural case.

MiniImp CFG

MiniImp *simple statements s*

$$s := \texttt{skip} \mid x := a \mid b?$$

$$b := v \mid b \text{ and } b \mid \text{not } b \mid a < a$$

$$a := x \mid n \mid a + a \mid a - a \mid a * a$$

... while control flow constructs define the edges

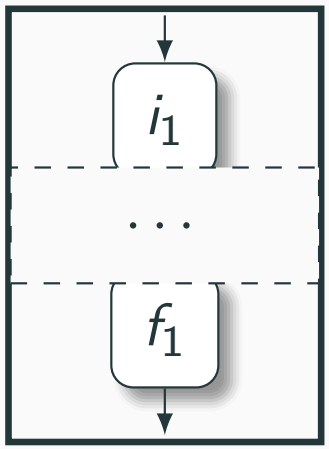
Assumptions

Initial and final nodes of the CFG

- i has no incoming edges
- f has no outgoing edges

We will enforce (and preserve) this constraints while building the CFG inductively

CFG representation

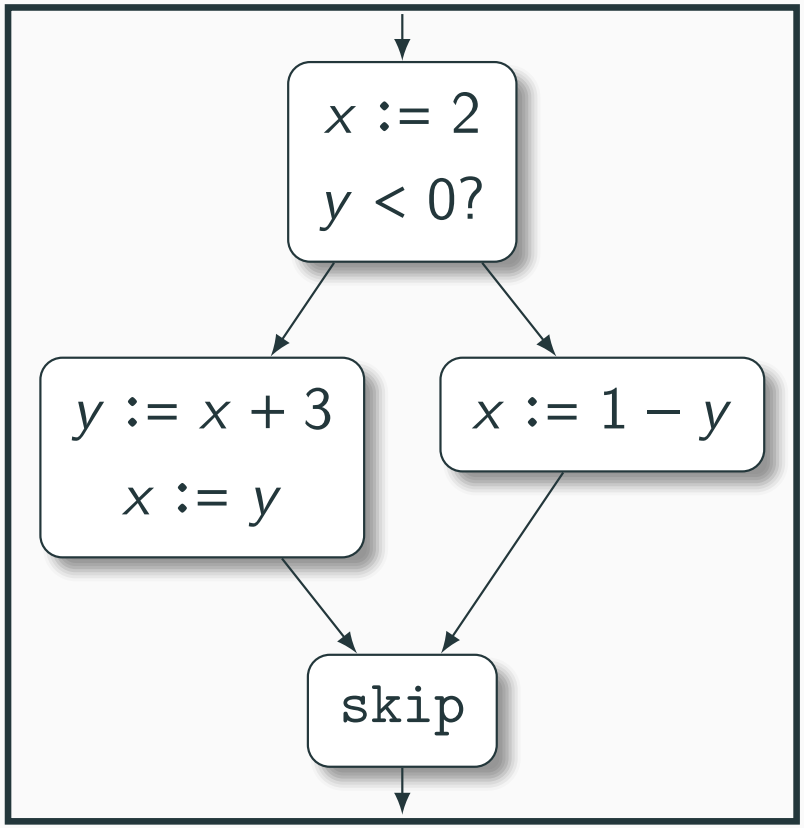


Example: CFG of a simple MiniImp program

MiniImp Program

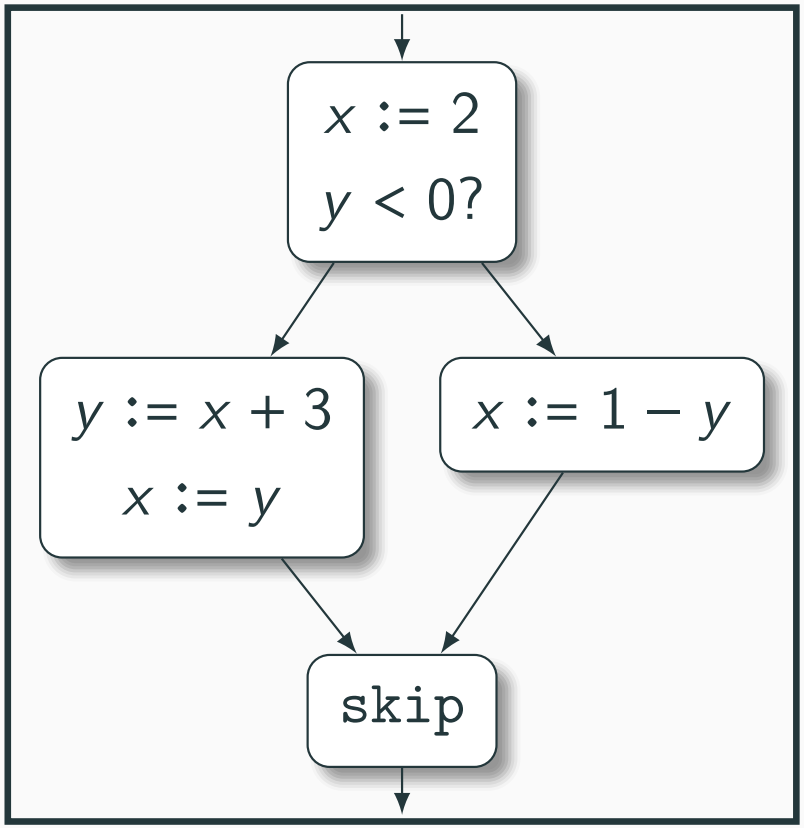
```
1  def main with input y
       output x as
2    x := 2;
3    if y < 0 then (
4      y := x + 3;
5      x := y
6    )
7    else
8      x := 1 - y;
```

MiniImp CFG

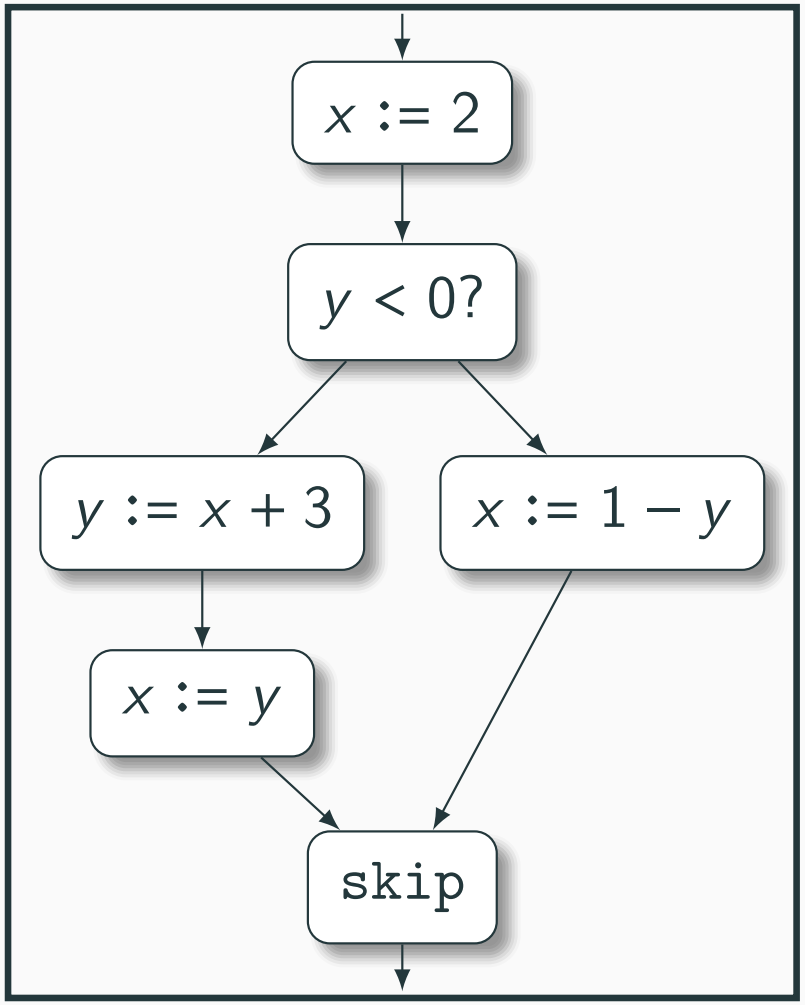


Two Alternatives for CFG

Maximal Blocks



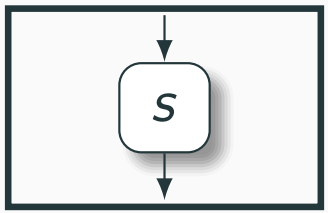
Minimal Blocks



Generating MiniImp CFG

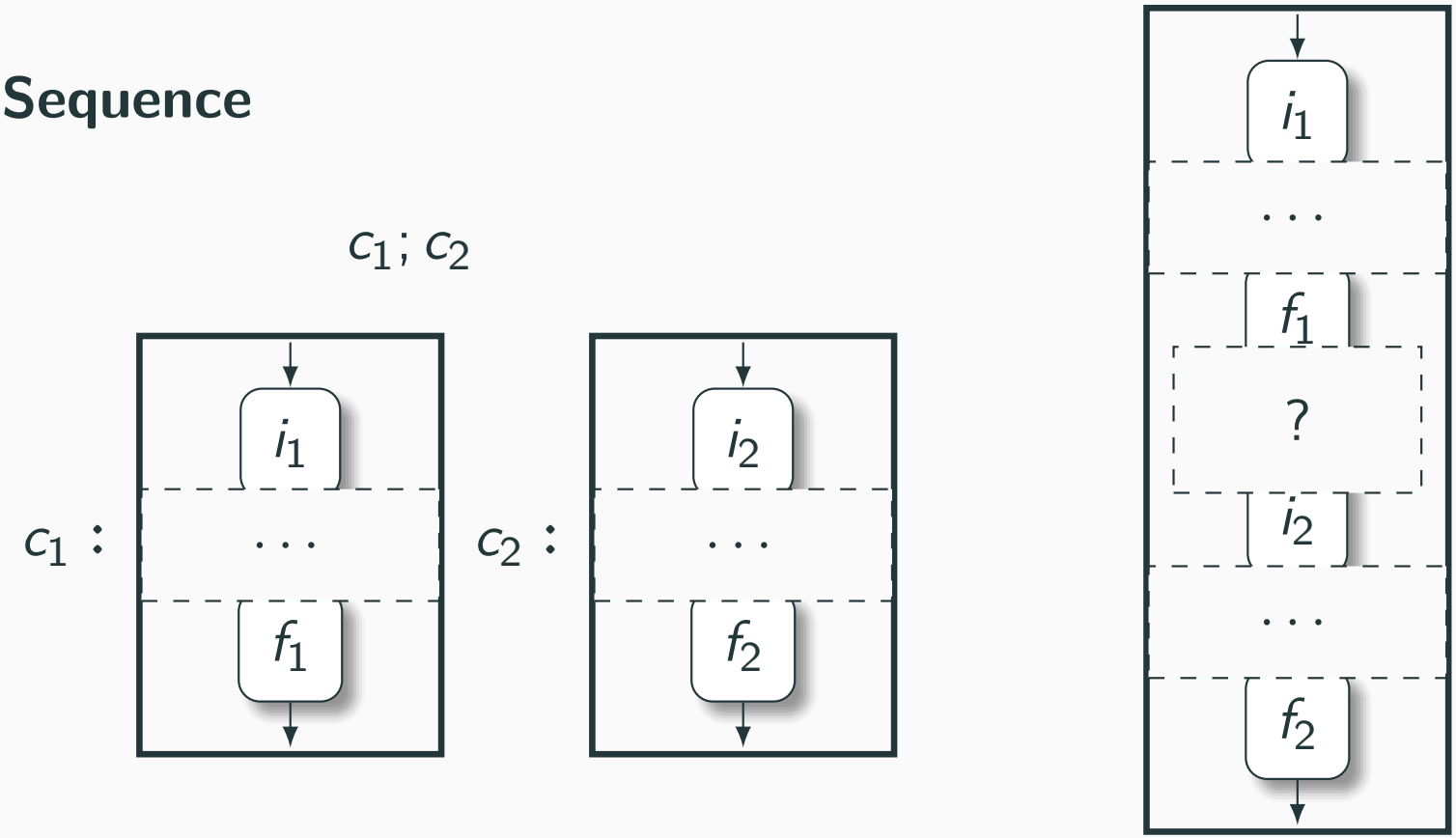
Command

s := skip | x := a



Generating MiniImp CFG

Sequence



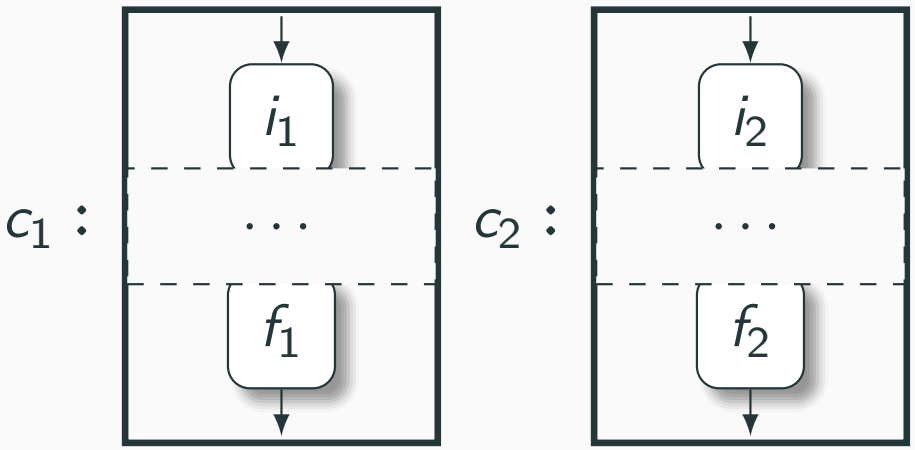
How to fill the box with the question mark "?"?

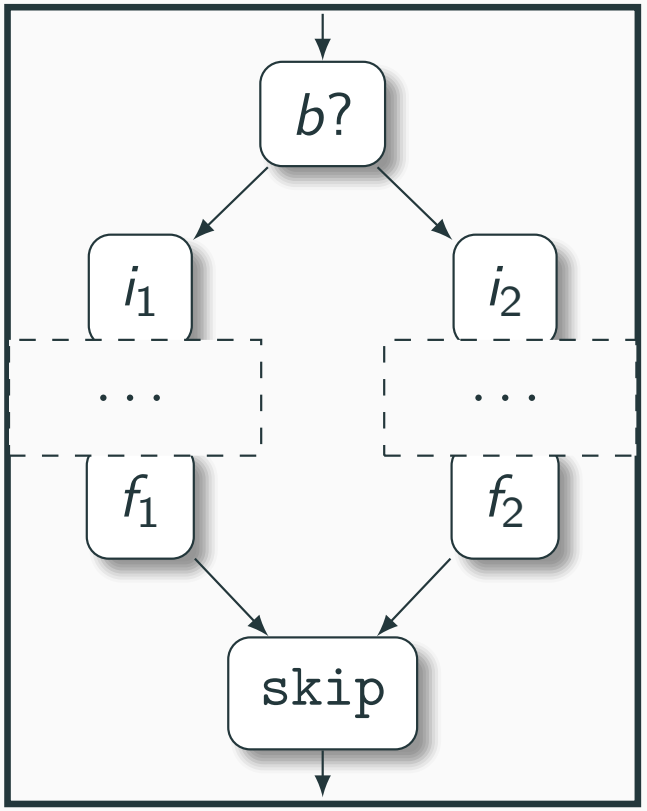
– A block with a skip? An arrow? Merging f_1 and i_2?

Generating MiniImp CFG

Conditional

`if` b `then` c_1 `else` c_2



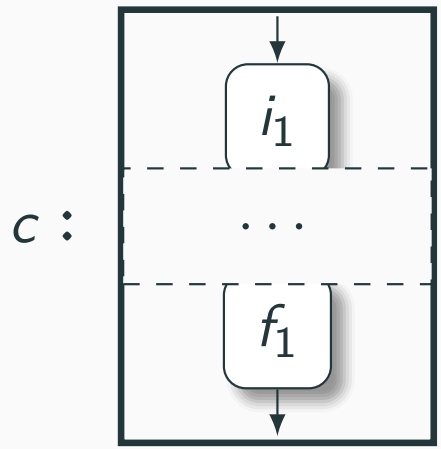


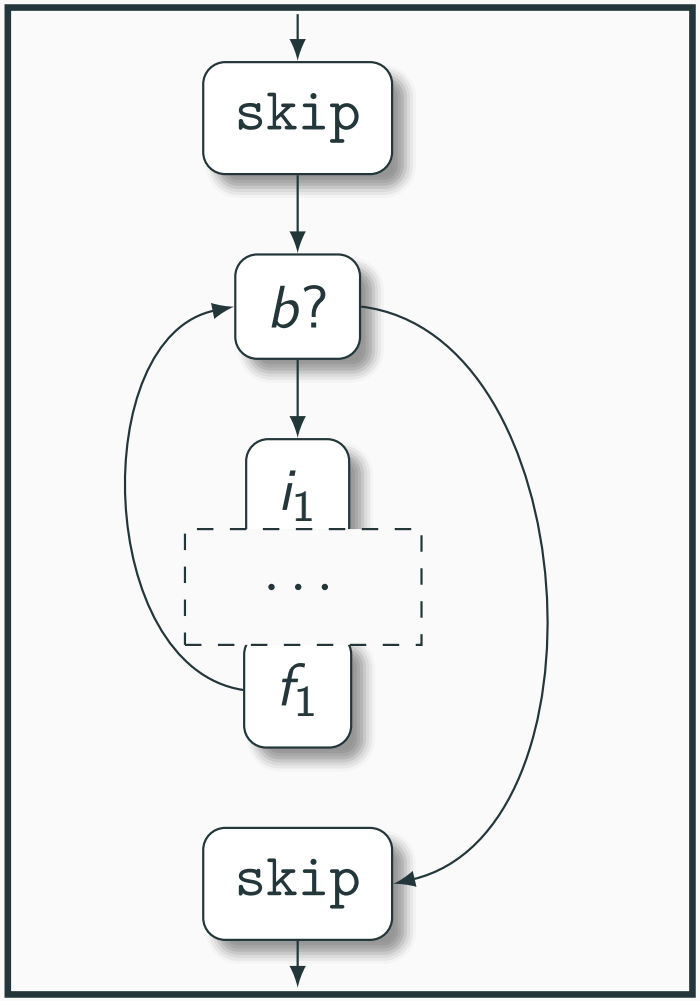
We need the `skip` block because we want a single final block.

Generating MiniImp CFG

Loop

`while` b `do` c





We need the `skip` blocks because the initial node cannot have incoming arrows and the final one cannot have outgoing arrows.

Generating MiniImp CFG

MiniImp Programs

`def main with input` x `output` y `as` c

Its CFG is simply the one of c

Project Fragment

(You should already have the module for MiniImp AST)

1. Define a module for control flow graphs
2. Define a function that given a MiniImp program (AST) returns its CFG
3. Write in the report your implementation choices:
   - which kind of blocks (maximal, minimal, something in between)
   - how you have decided to generate the CFG for sequences
   - any other detail that you consider important