Program Validation by Symbolic and Reverse Execution

Jooyong Lee

BRICS

Department of Computer Science

University of Aarhus

Computer Programs

Introduction
- Computer Programs
- Program Errors (aka ‘Bugs’)
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks

Programs are everywhere.

Computer Programs

Introduction
- Computer Programs
- Program Errors (aka ‘Bugs’)
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks

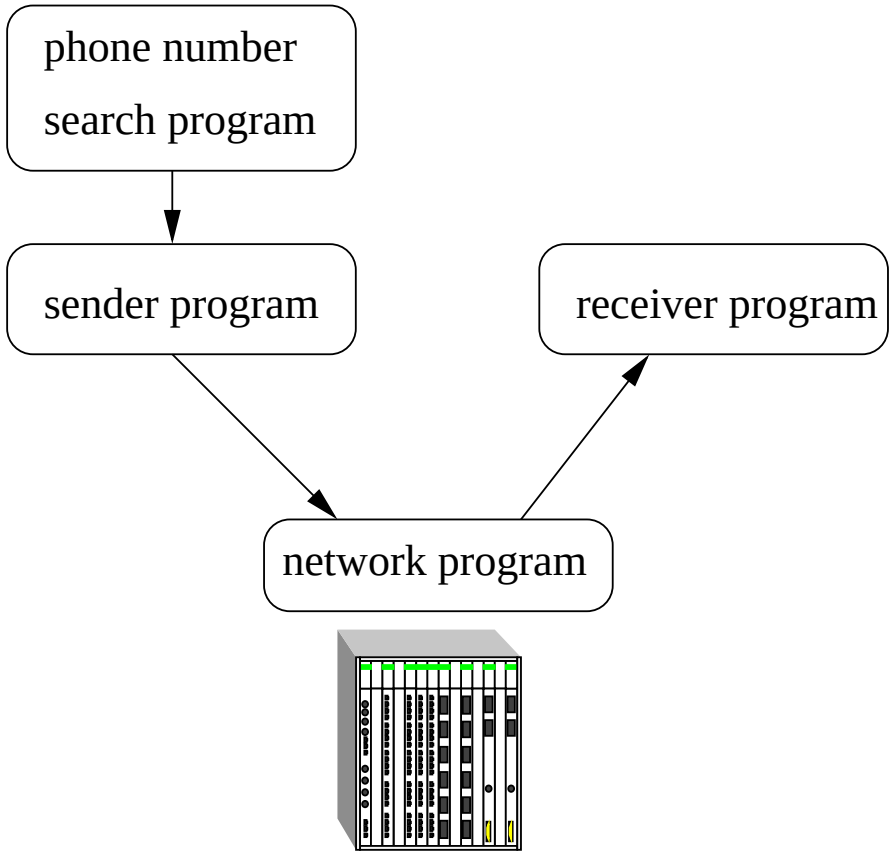
Computer Programs

Introduction
- Computer Programs
- Program Errors (aka 'Bugs')
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks



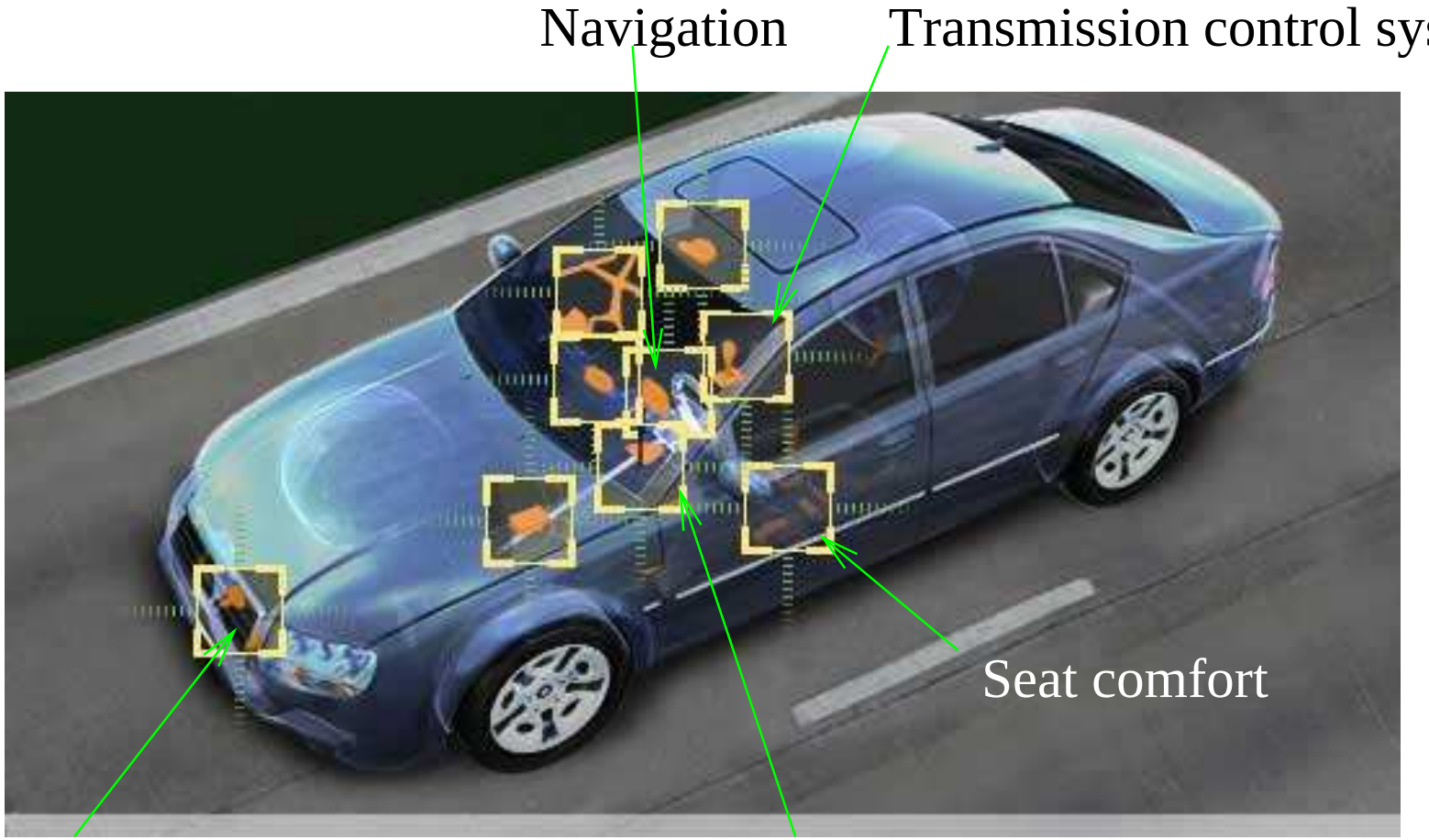
Computer Programs

Introduction
- Computer Programs
- Program Errors (aka ‘Bugs’)
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks



Computer Programs

Introduction
- Computer Programs
- Program Errors (aka ‘Bugs’)
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks

- Ariane 5: an unmanned rocket
- Everything is controlled by programs.

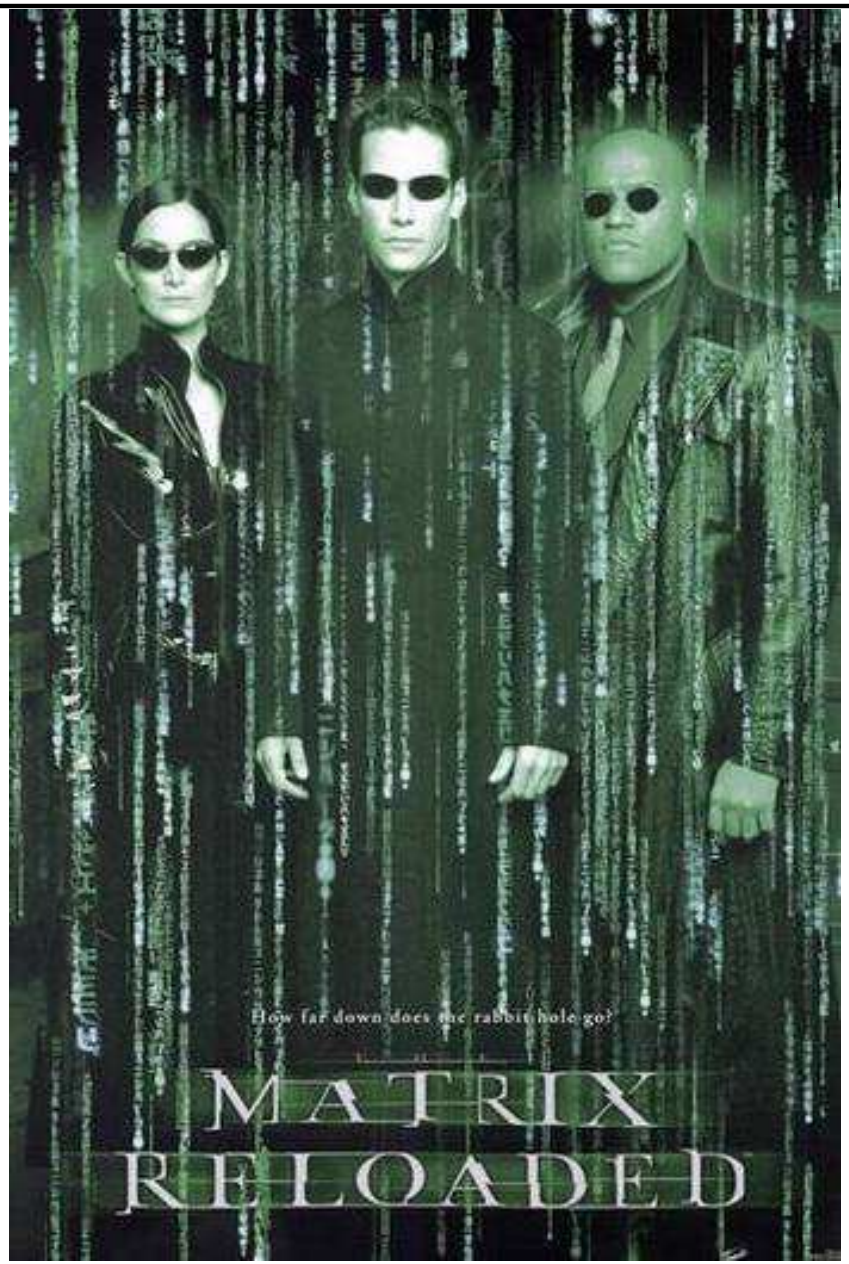
Computer Programs

Introduction
- Computer Programs
- Program Errors (aka ‘Bugs’)
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks



Who knows?
We may be living our lives as programs in the Matrix.

Program Errors (aka ‘Bugs’)

Introduction
- Computer Programs
- Program Errors (aka ‘Bugs’)
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks

Bugs are everywhere.

A problem has been detected and windows has been shut down to prevent damage
to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c

Program Errors (aka ‘Bugs’)

Introduction
- Computer Programs
- Program Errors (aka ‘Bugs’)
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks

Is it really OK?

Program Errors (aka ‘Bugs’)

Introduction
- Computer Programs
- Program Errors (aka ‘Bugs’)
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks

- Recall of Chrysler Pacifica in 2006
- Summary: ON CERTAIN PASSENGER VEHICLES, THE FUEL PUMP MODULE AND THE POWER TRAIN CONTROL MODULE SOFTWARE MAY ALLOW THE ENGINE TO STALL UNDER CERTAIN OPERATING CONDITIONS.
- Consequence: THIS COULD CAUSE A CRASH TO OCCUR WITHOUT PRIOR WARNING.

Program Errors (aka ‘Bugs’)

Introduction
- Computer Programs
- Program Errors (aka ‘Bugs’)
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks

- The Ariane 5 rocket exploded 40 seconds after its launch due to a program bug in 1996.

Topic of My Thesis

Introduction
- Computer Programs
- Program Errors (aka 'Bugs')
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks

CHECKING FOR BUGS

&

LOCATING BUGS

in programs

Topic of My Thesis

Introduction
- Computer Programs
- Program Errors (aka 'Bugs')
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks

PROGRAM VALIDATION

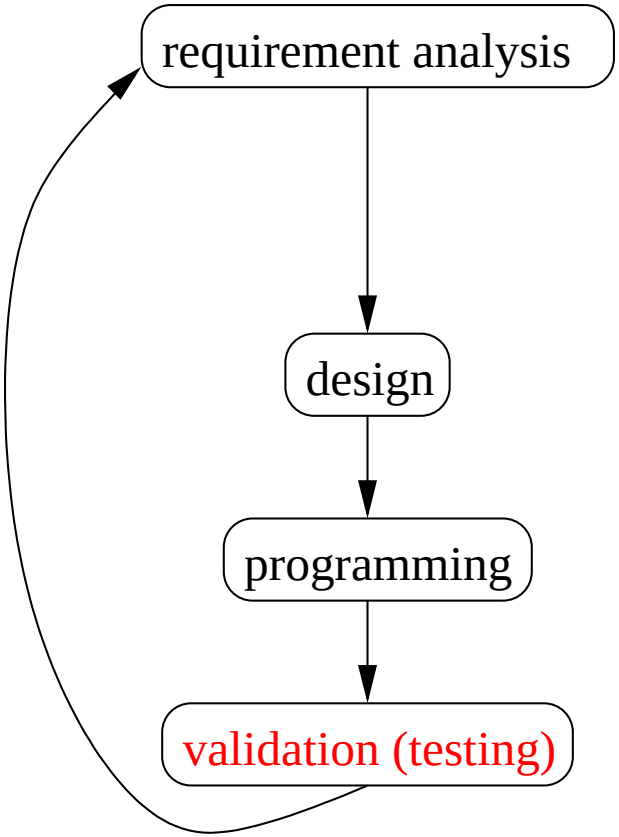
Program Life-cycle

Introduction
- Computer Programs
- Program Errors (aka ‘Bugs’)
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks



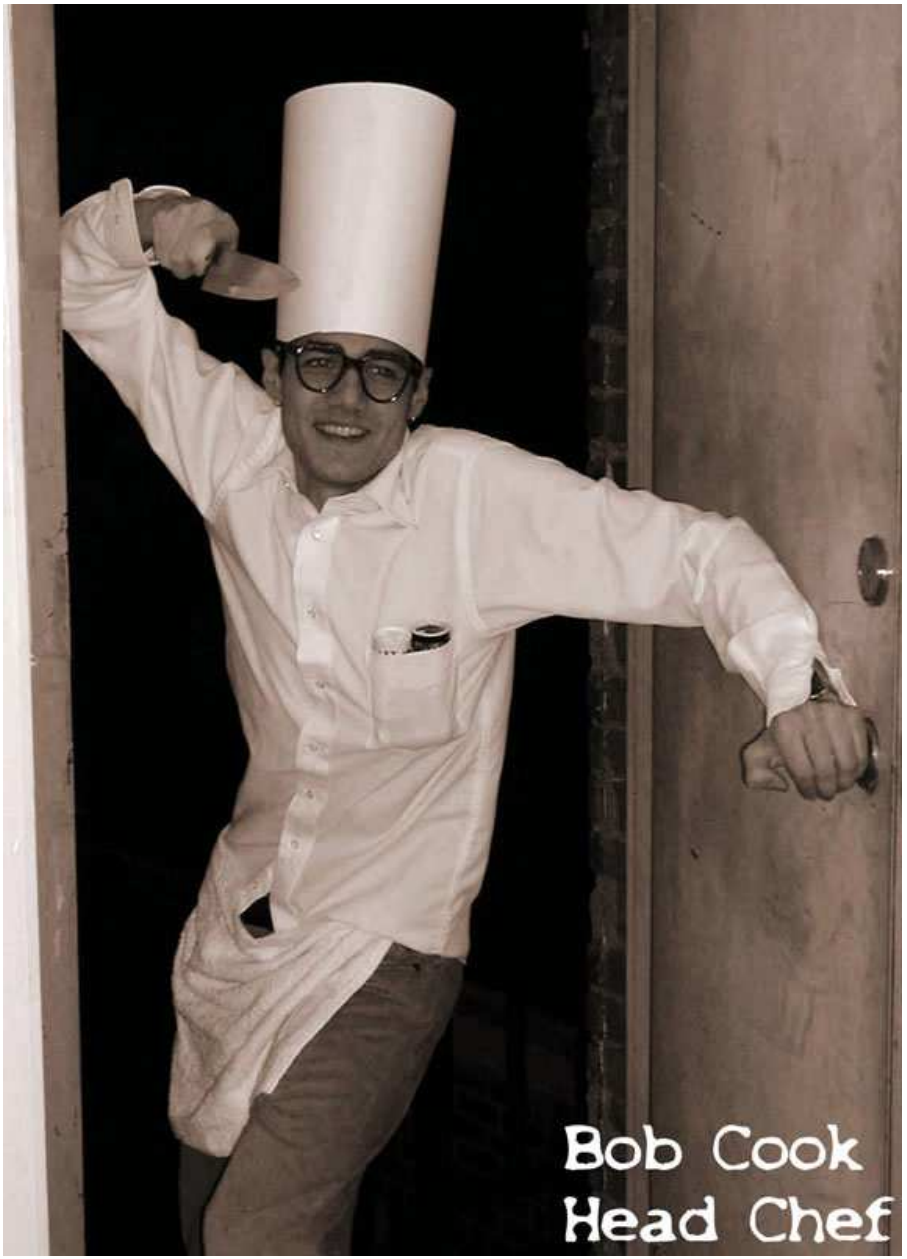
Program Life-cycle

Introduction
- Computer Programs
- Program Errors (aka ‘Bugs’)
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks



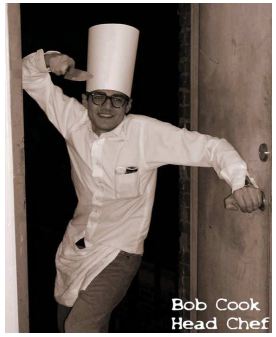
Program Life-cycle

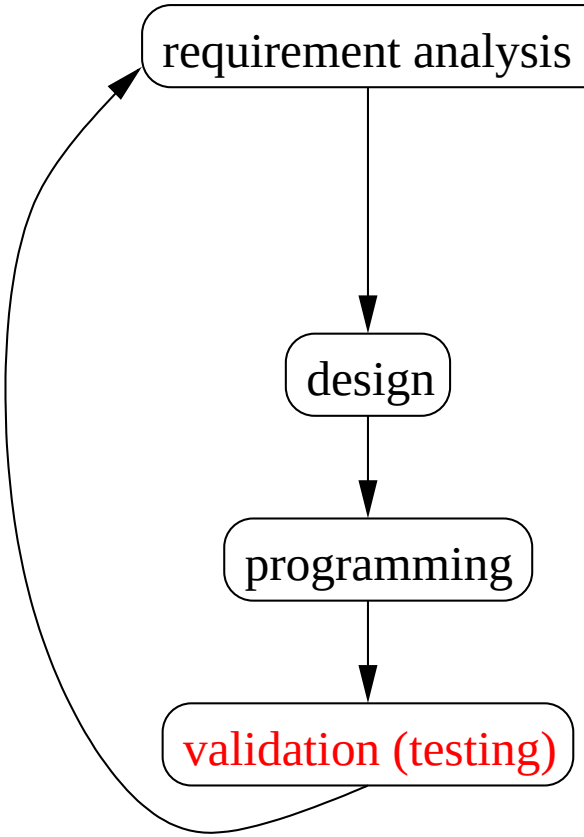
Introduction
- Computer Programs
- Program Errors (aka 'Bugs')
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

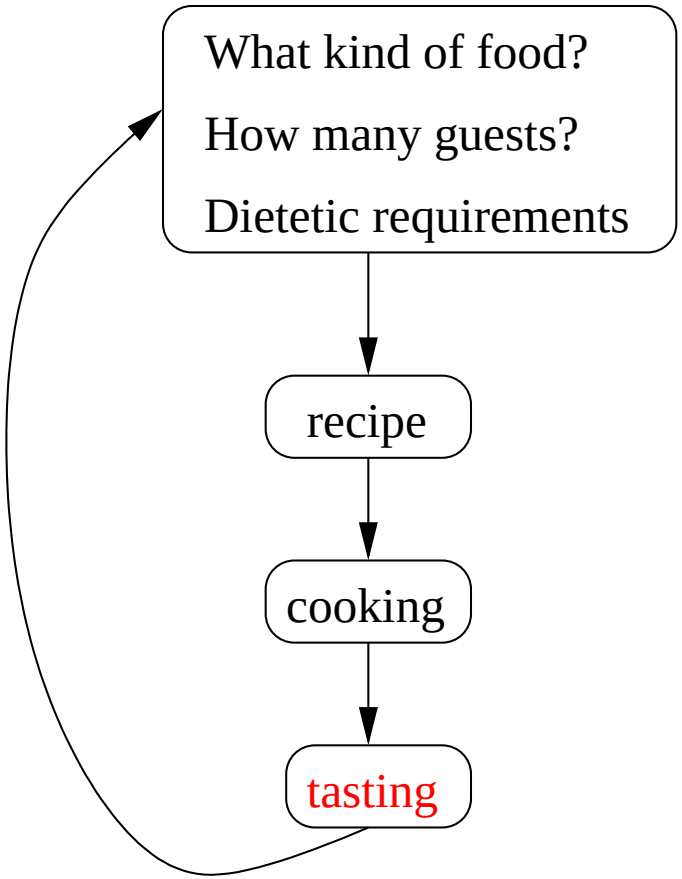
Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks







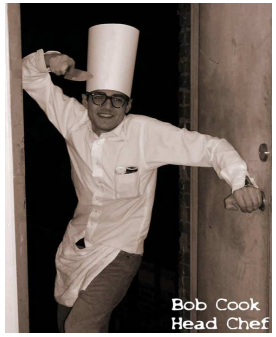
Program Life-cycle

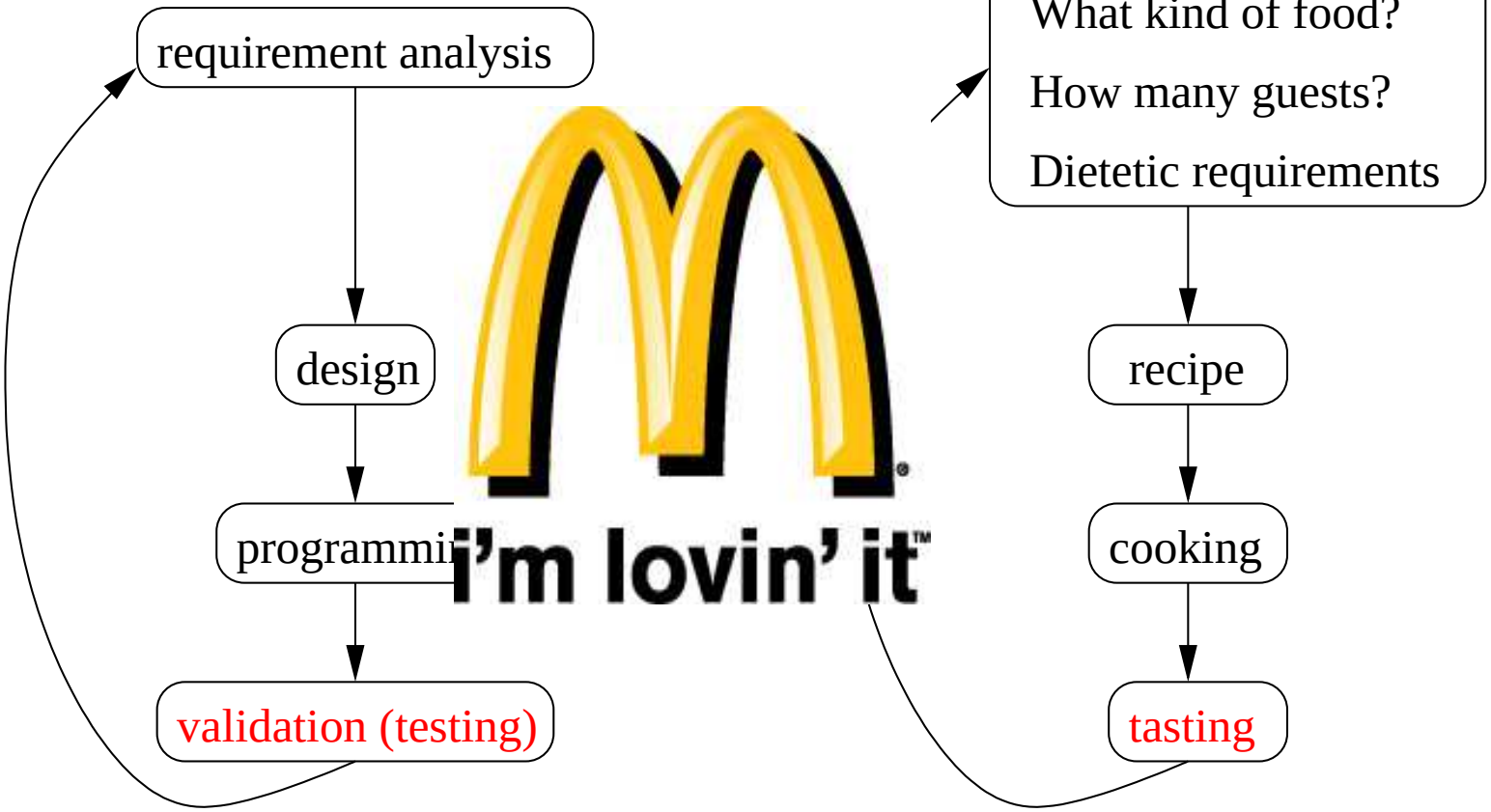
Introduction
- Computer Programs
- Program Errors (aka 'Bugs')
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks





My Thesis (What)

Introduction
- Computer Programs
- Program Errors (aka ‘Bugs’)
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks

Program Validation

My Thesis (How)

Introduction
- Computer Programs
- Program Errors (aka 'Bugs')
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks

Program Validation

by Symbolic and Reverse Execution

- Symbolic Execution
  - Checking for Bugs

- Reverse Execution
  - Locating Bugs

Plan

Introduction
- Computer Programs
- Program Errors (aka 'Bugs')
- Topic of My Thesis
- Program Life-cycle
- My Thesis (What)
- My Thesis (How)
- Plan

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks

1. Introduction
2. Reverse Execution
   (a) What is it?
   (b) Existing Work?
   (c) Problem?
   (d) Contribution: Dynamic Reverse Code Generation
   (e) Soundness
   (f) Case Study
3. Symbolic Execution
   (a) What is it?
   (b) Existing Work?
   (c) Contribution: Bounded Lazier Initialization
   (d) Soundness and Completeness
4. Concluding Remarks

What?

Introduction

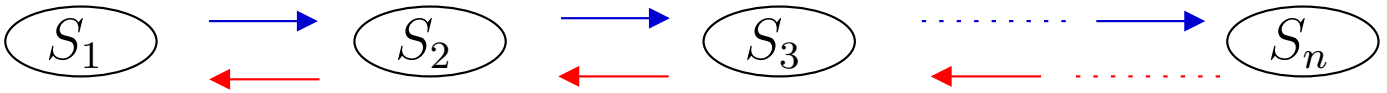
Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

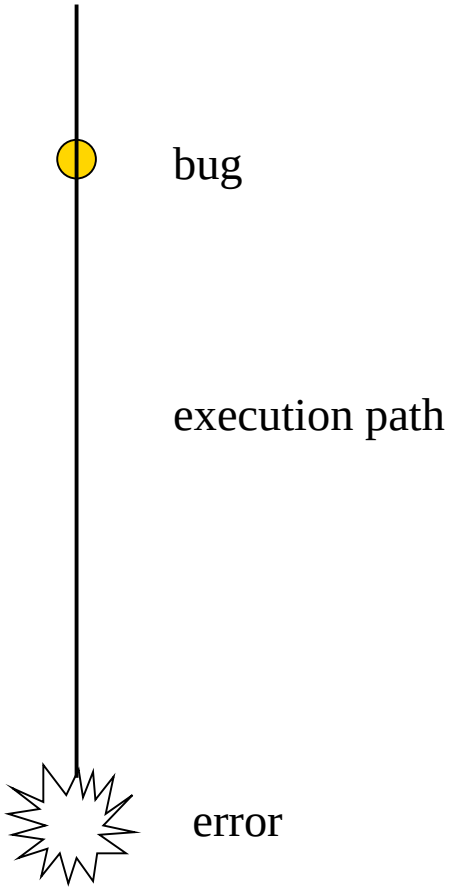
Program Validation by Symbolic Execution

- What is reverse execution?
  - A series of state restorations
- What is execution?
  - A series of state changes



Locating a Bug

- Butterfly Effect: Something as small as the flutter of a butterfly's wings can ultimately cause a typhoon halfway around the world.



Introduction

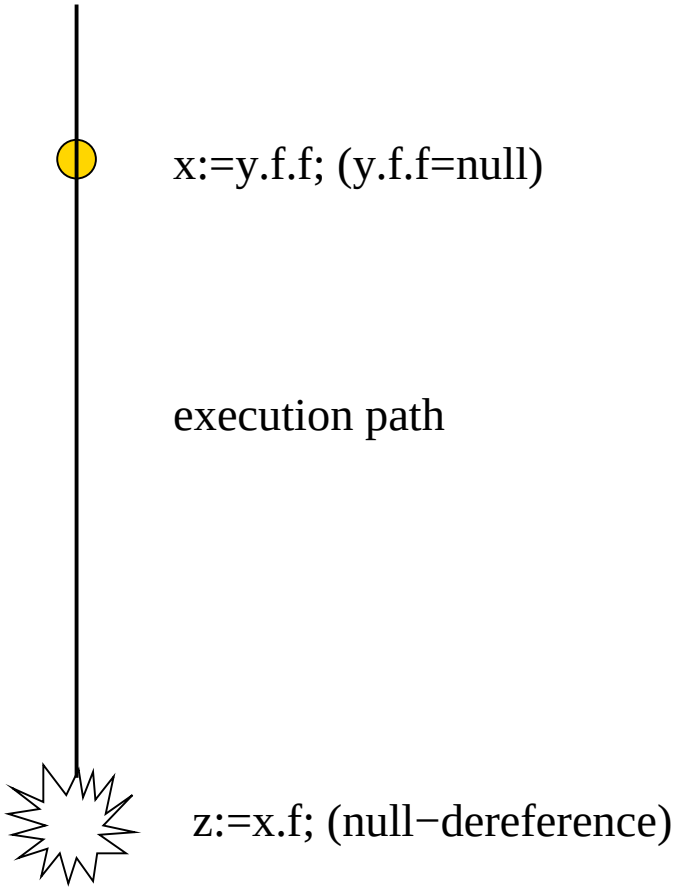
Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Verification by Symbolic Execution

Locating a Bug

- Butterfly Effect: Something as small as the flutter of a butterfly's wings can ultimately cause a typhoon halfway around the world.



Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example (Chapter 8)

Program Verification by Symbolic Execution

Locating a Bug by Iterative Execution

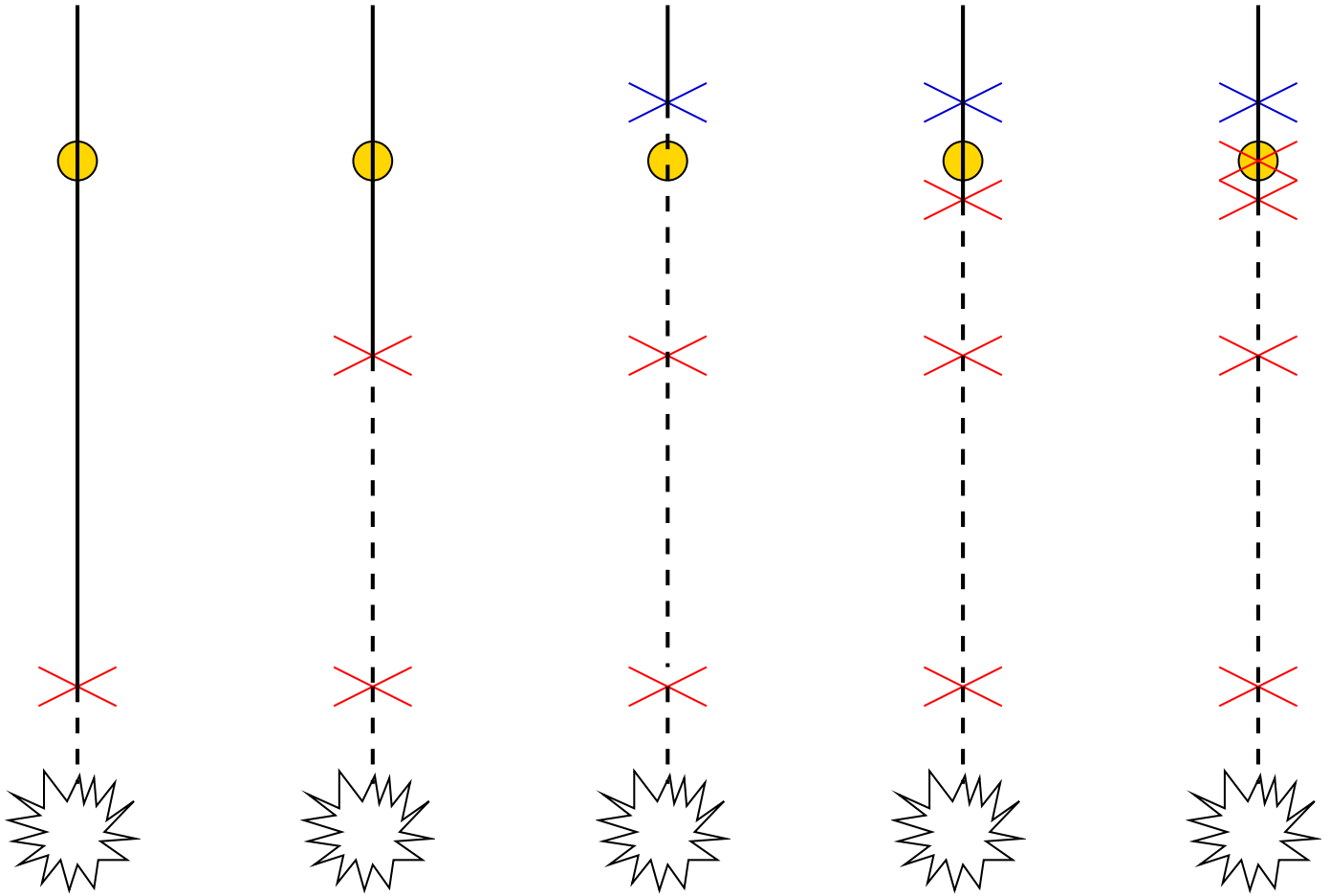
Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

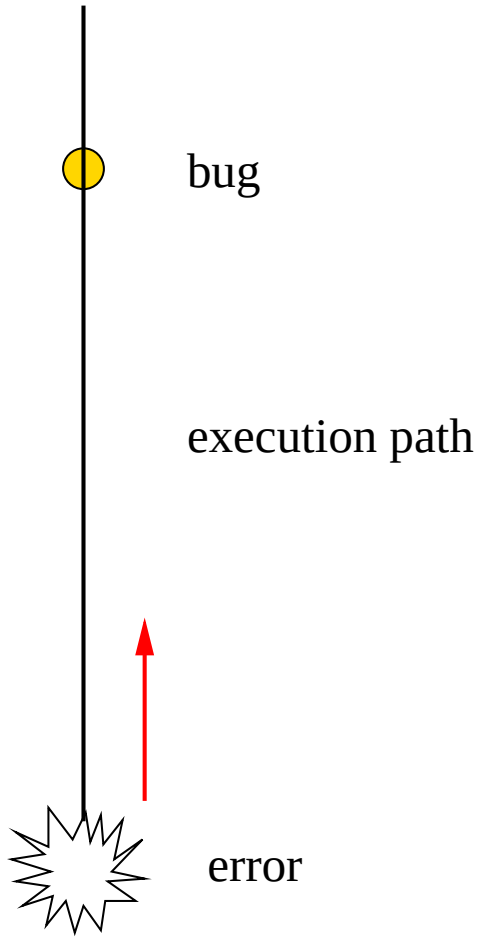
(Chapter 8)

Program Validation by Symbolic Execution



Locating a Bug by Reverse Execution

- As an alternative, reverse execution has been suggested
- Functionally similar to undo in text editors



Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Verification by Symbolic Execution

Reverse-Execution Methods

Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

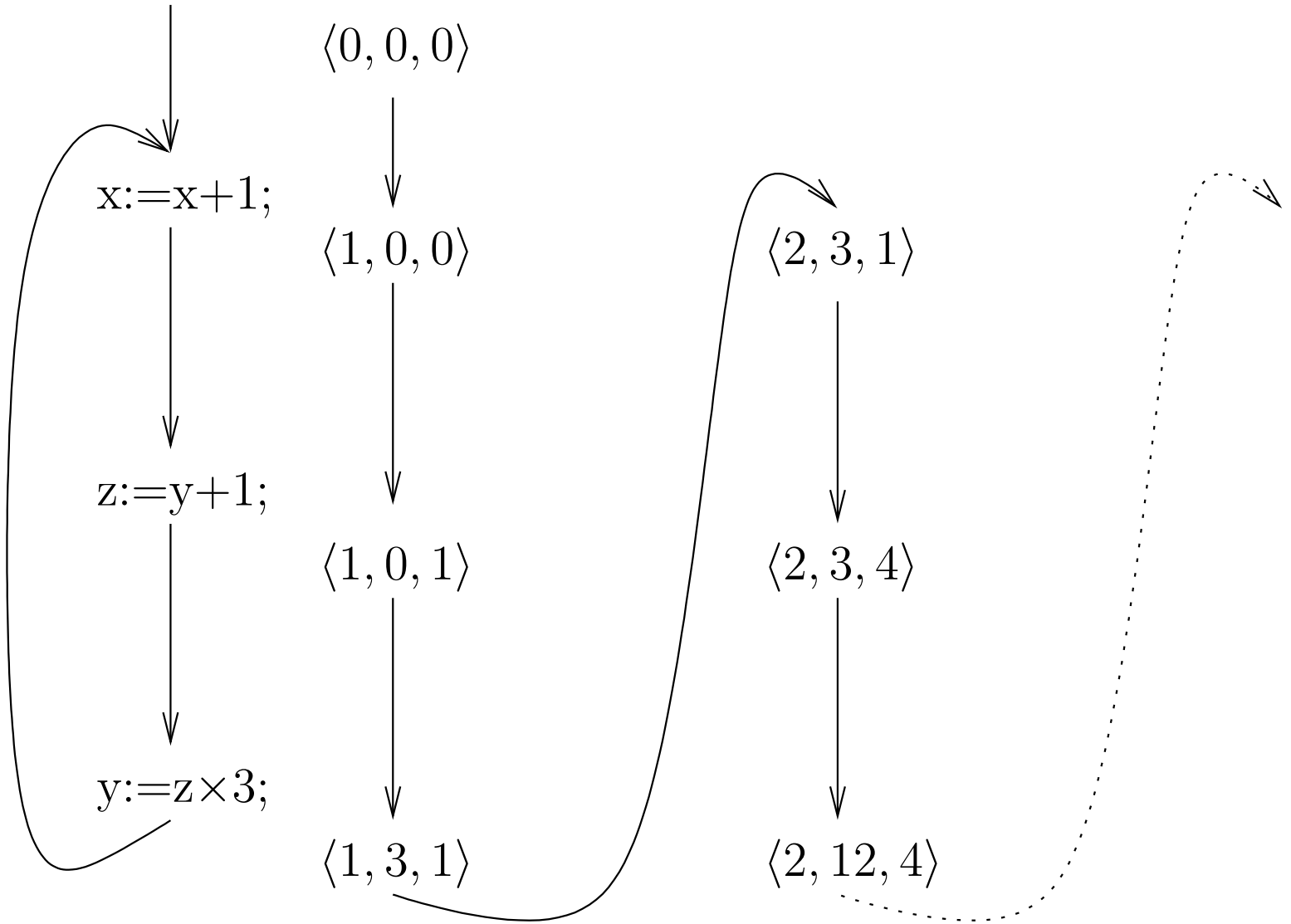
(Chapter 8)

Program Verification by Symbolic Execution

In the literature:

- State-Saving
- Checkpointing
- Reverse Code

Loop Example



Introduction

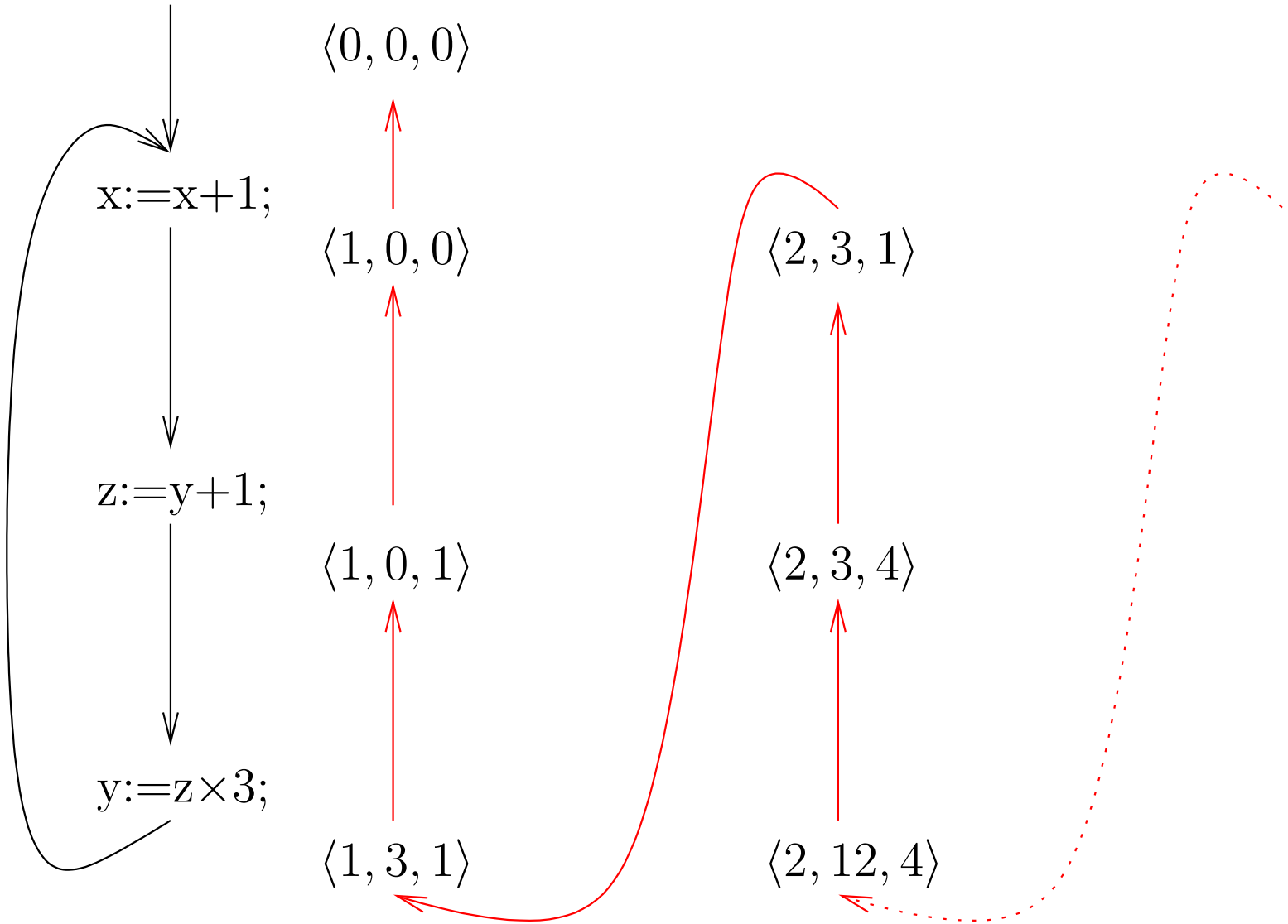
Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Verification by Symbolic Execution

Loop Example



Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

Reverse Execution by State-Saving

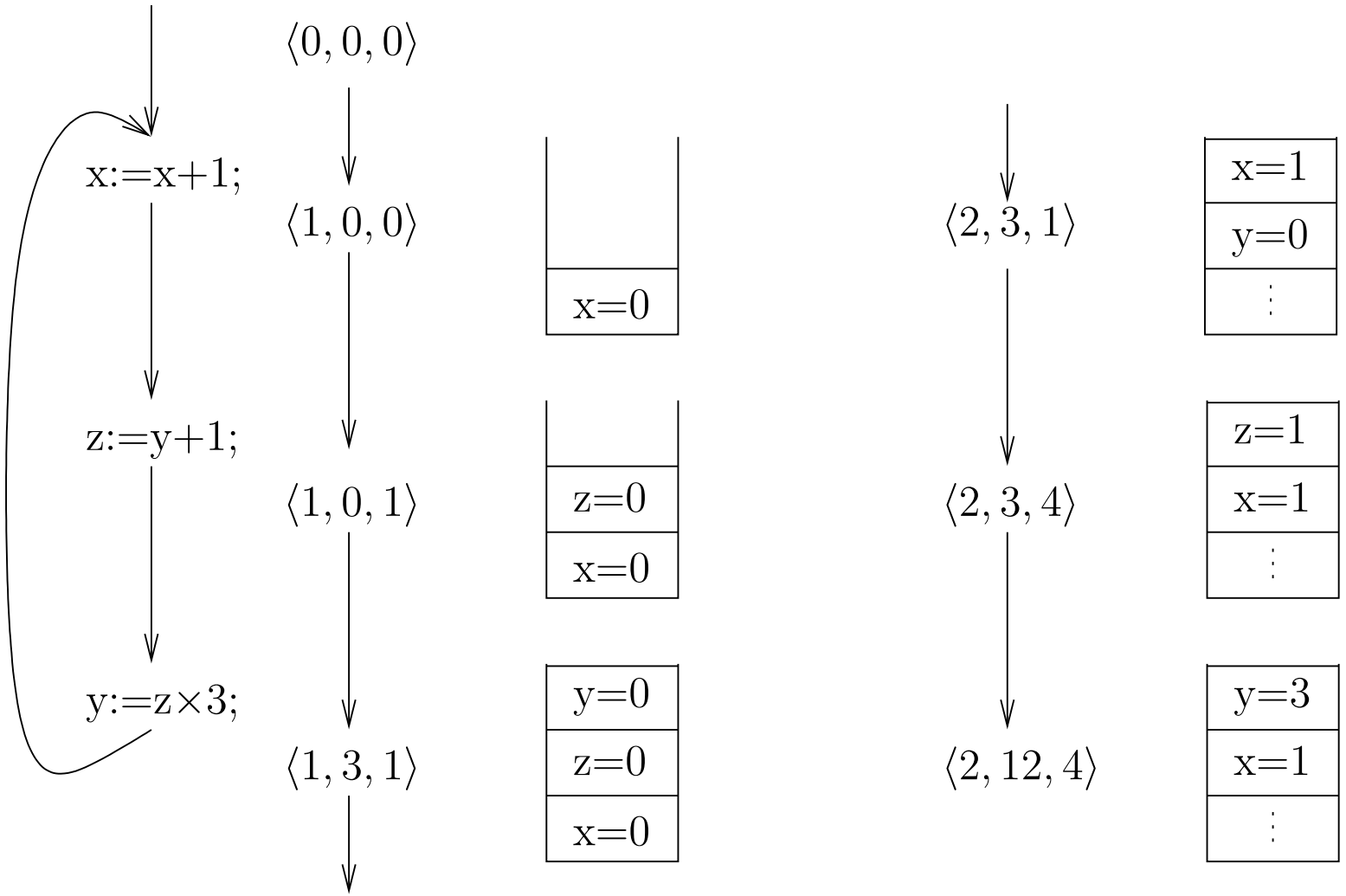
Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Verification by Symbolic Execution



Reverse Execution by State-Saving

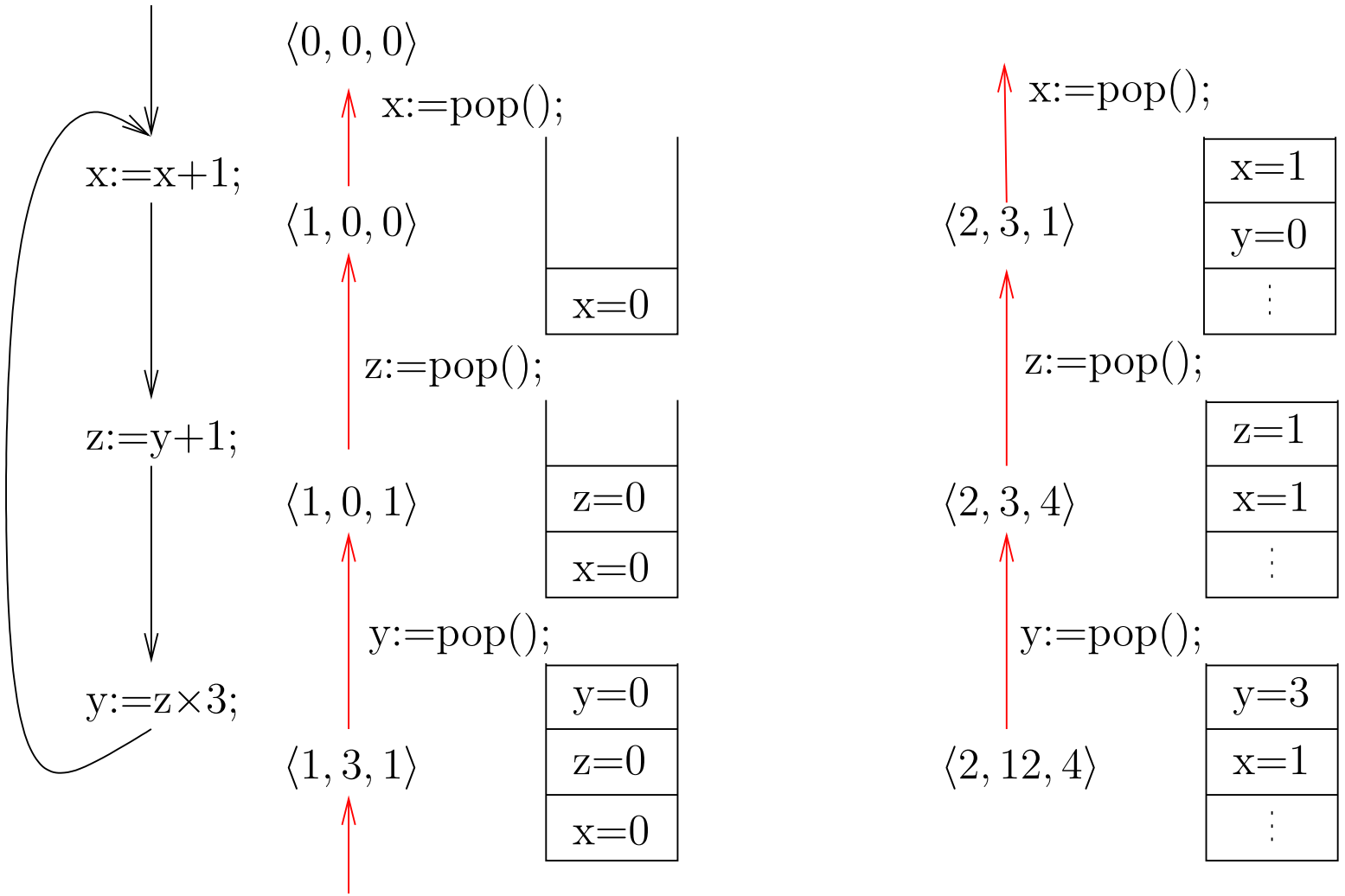
Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Verification by Symbolic Execution



Reverse Execution by Checkpointing

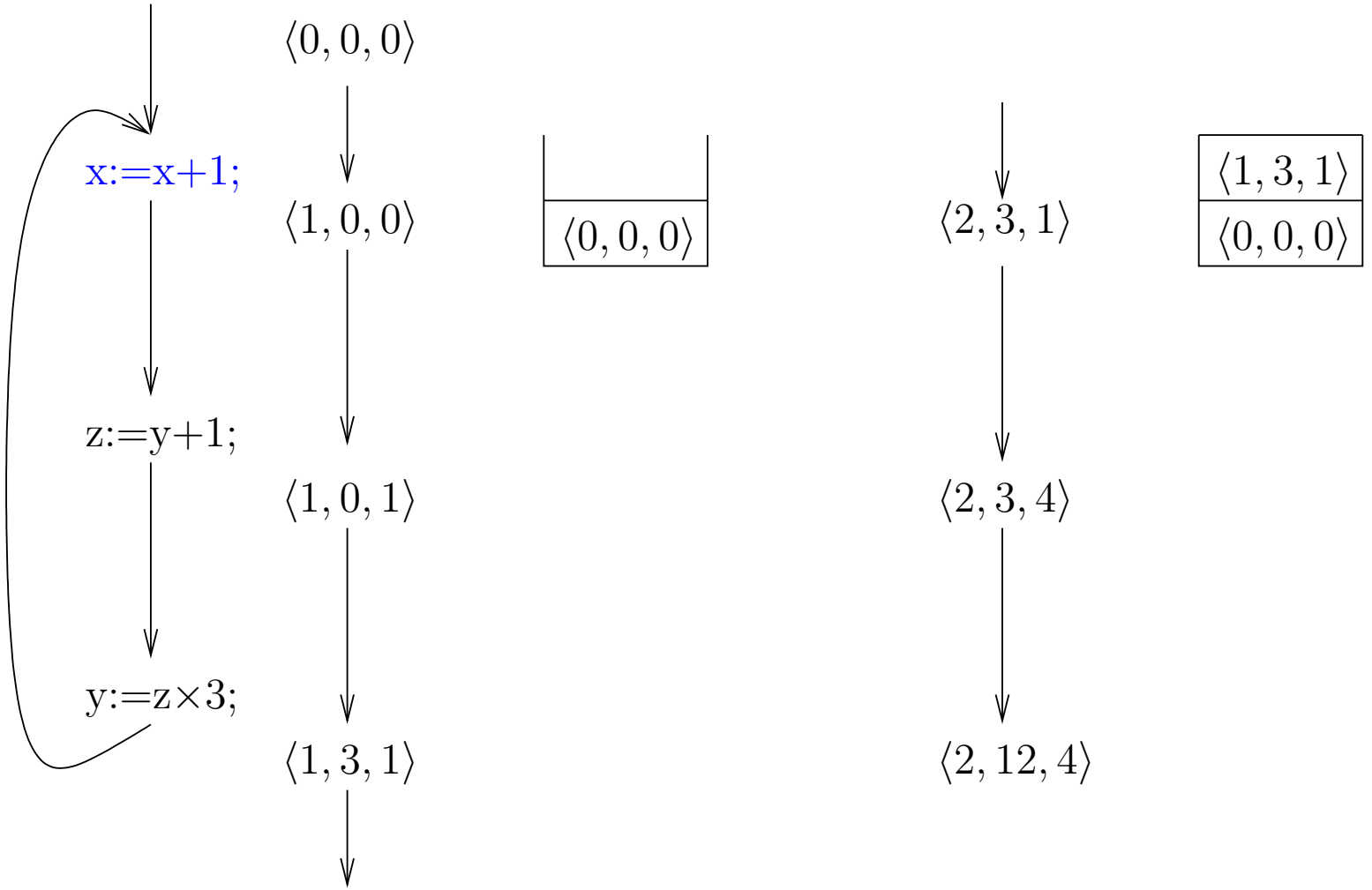
Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Verification by Symbolic Execution

■ Periodic state-saving



Reverse Execution by Checkpointing

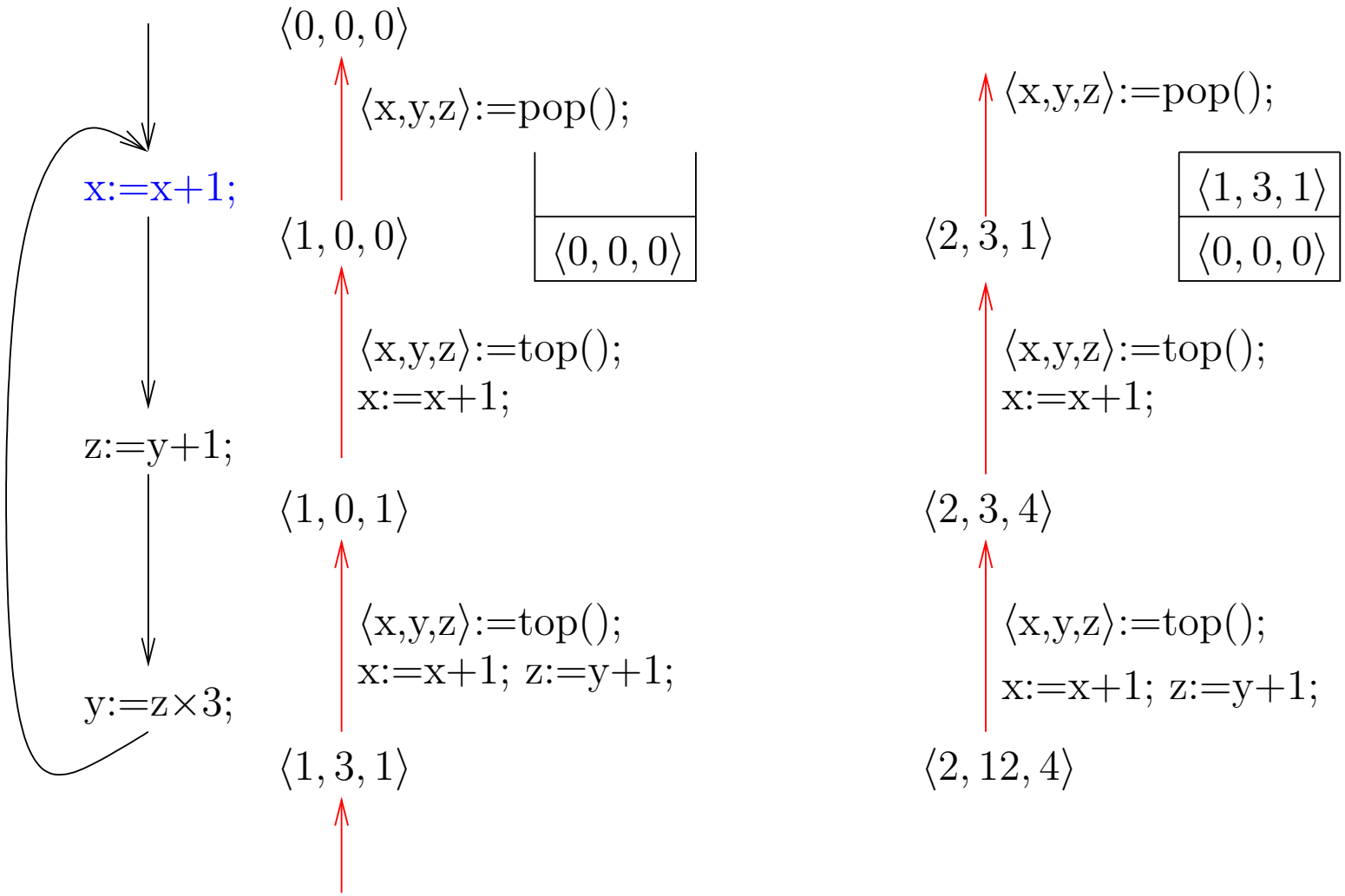
Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

Periodic state-saving



Reverse Execution by Checkpointing

- Periodic state-saving
- It consumes fewer stack elements than state-saving if. . .
- State-saving and checkpointing depend heavily on the use of the stack
- The stack is usually allocated in the memory of a computer

Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

Reverse Execution by Reverse Code

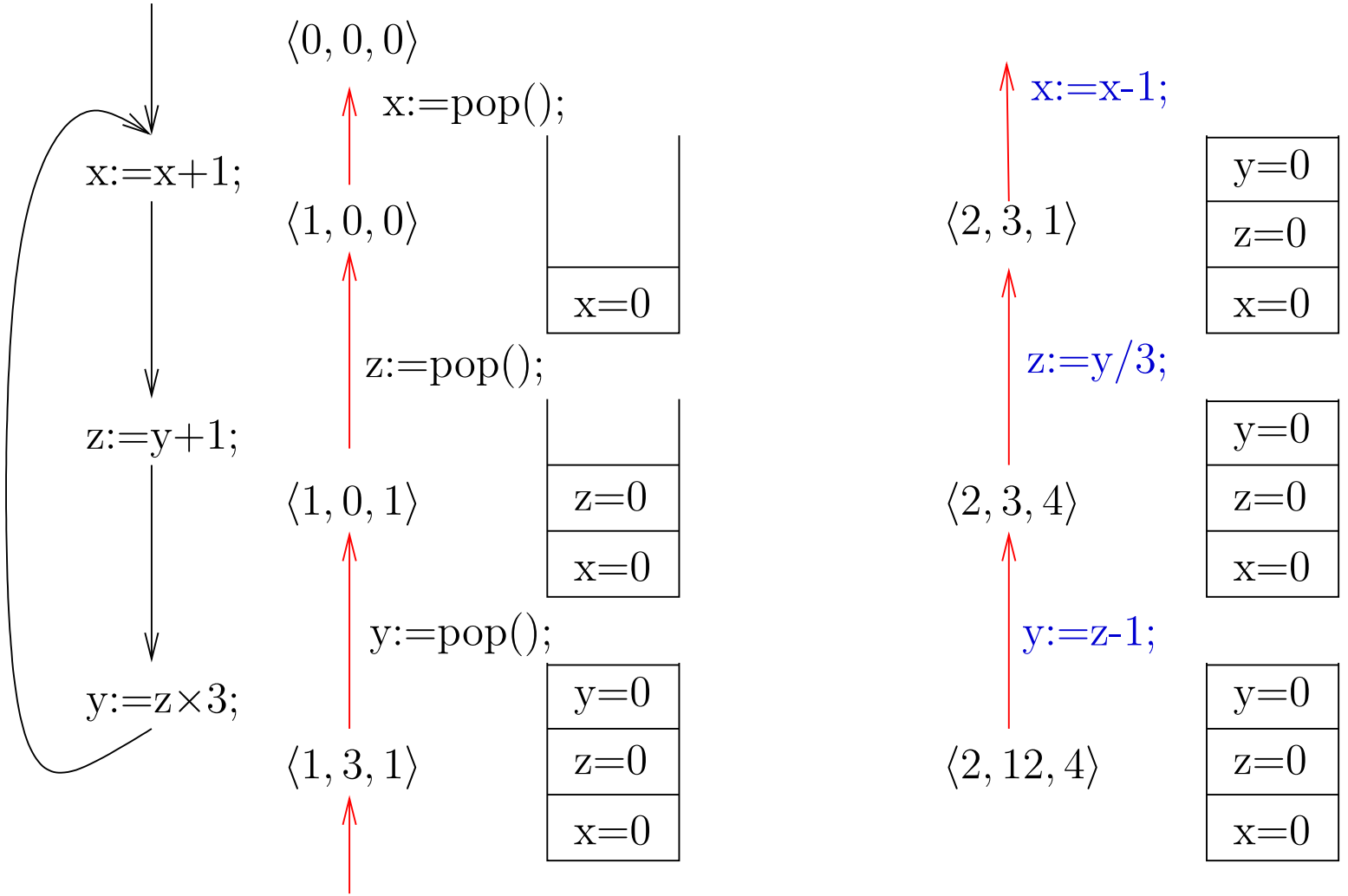
Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

- Reverse code alleviates the situation



Reverse Code Generation

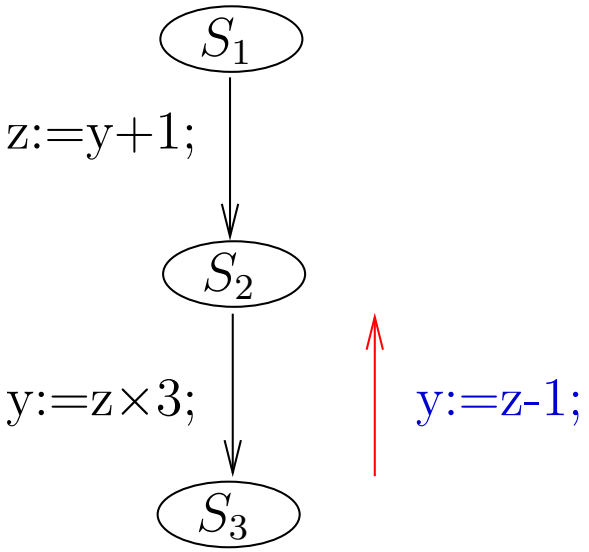
Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

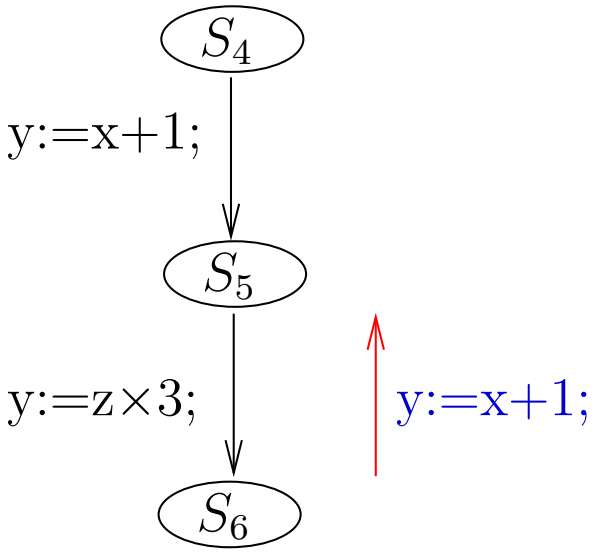
Program Validation by Symbolic Execution

- Extraction-from-use



- y was used in z:=y+1;.
- z=$(\lambda t.t+1)$(y)
- y=$(\lambda t.t+1)^{-1}$(z)
- $(\lambda t.t+1)^{-1}$=$(\lambda t.t-1)$
- y=$(\lambda t.t-1)$(z)
- y:=z-1;.

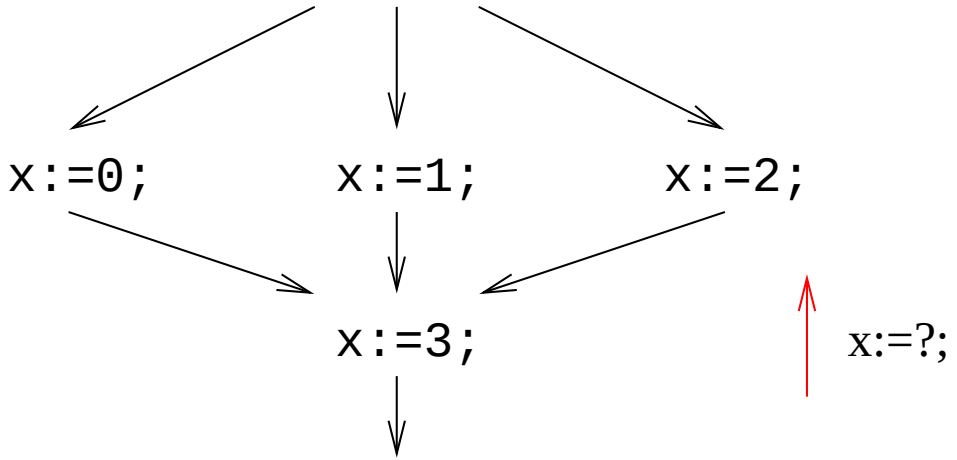
- Redefinition



- y:=x+1; is a reaching definition of y.

Static Reverse Code Generation

- shown in [Akgul and Mooney, PASTE'02]
- Analyzes a source program
- Before executing a program
- Infeasible for non-deterministic programs



Introduction

Reverse Execution

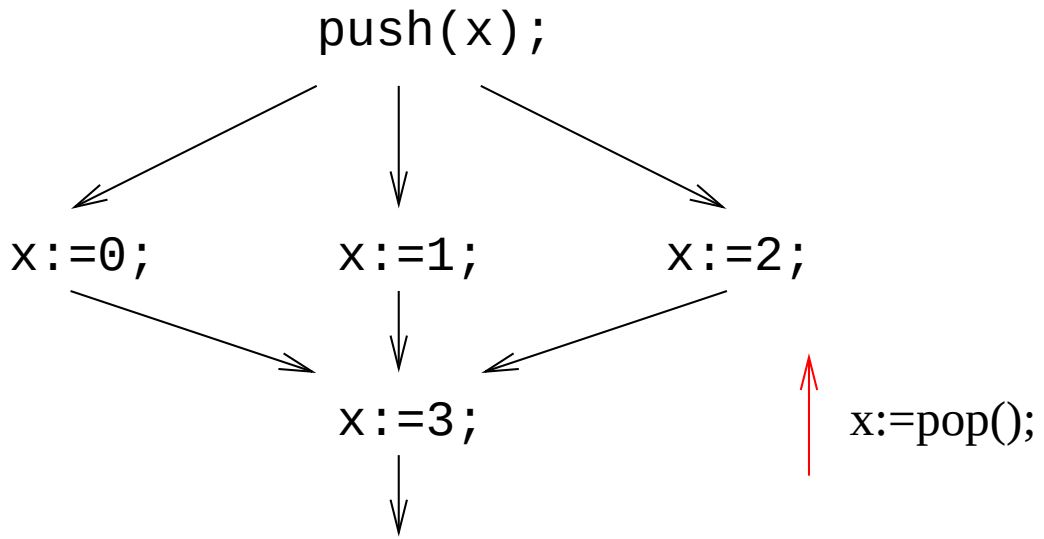
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

Static Reverse Code Generation

- shown in [Akgul and Mooney, PASTE'02]
- Analyzes a source program
- Before executing a program
- Infeasible for non-deterministic programs



Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

Static Reverse Code Generation

Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

- shown in [Akgul and Mooney, PASTE'02]
- Analyzes a source program
- Before executing a program
- Infeasible for non-deterministic programs

- Does it matter?

Static Reverse Code Generation

- shown in [Akgul and Mooney, PASTE'02]
- Analyzes a source program
- Before executing a program
- Infeasible for non-deterministic programs

- Does it matter?

Yes, it does!

Introduction

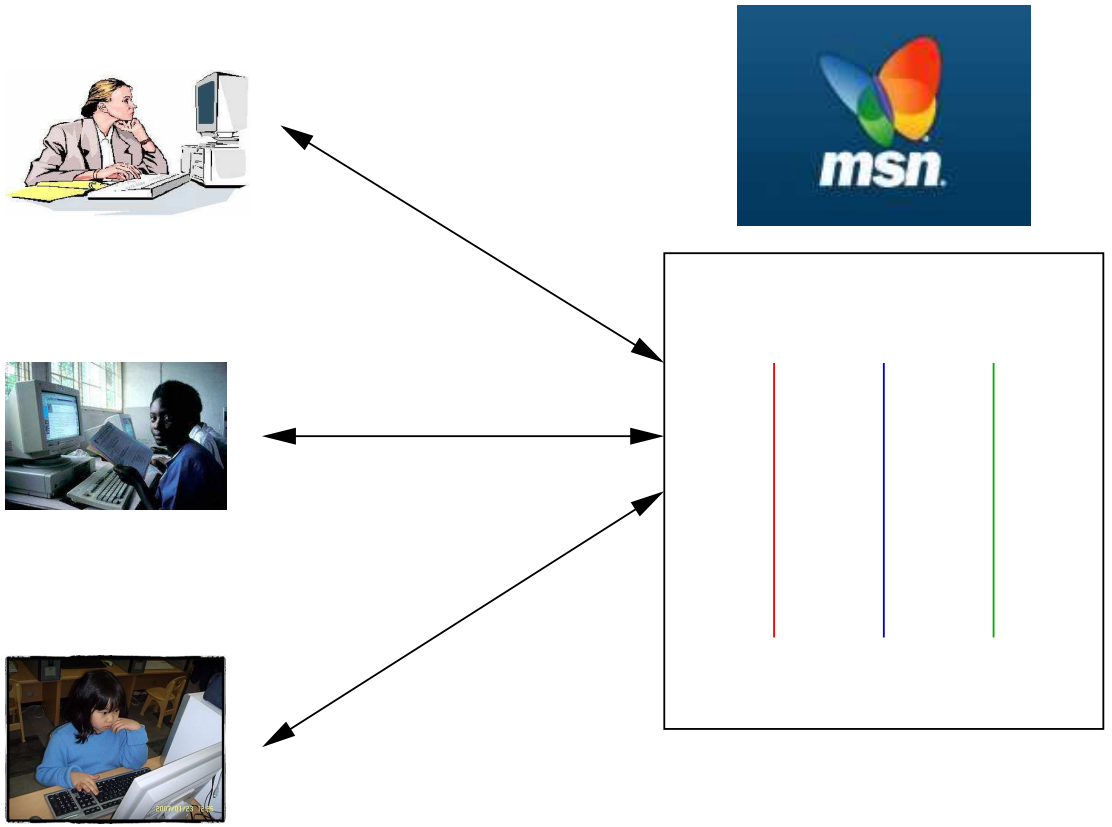
Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

Multi-threaded Programs

- Multi-threaded programs occur everywhere.
- They are non-deterministic.



Introduction

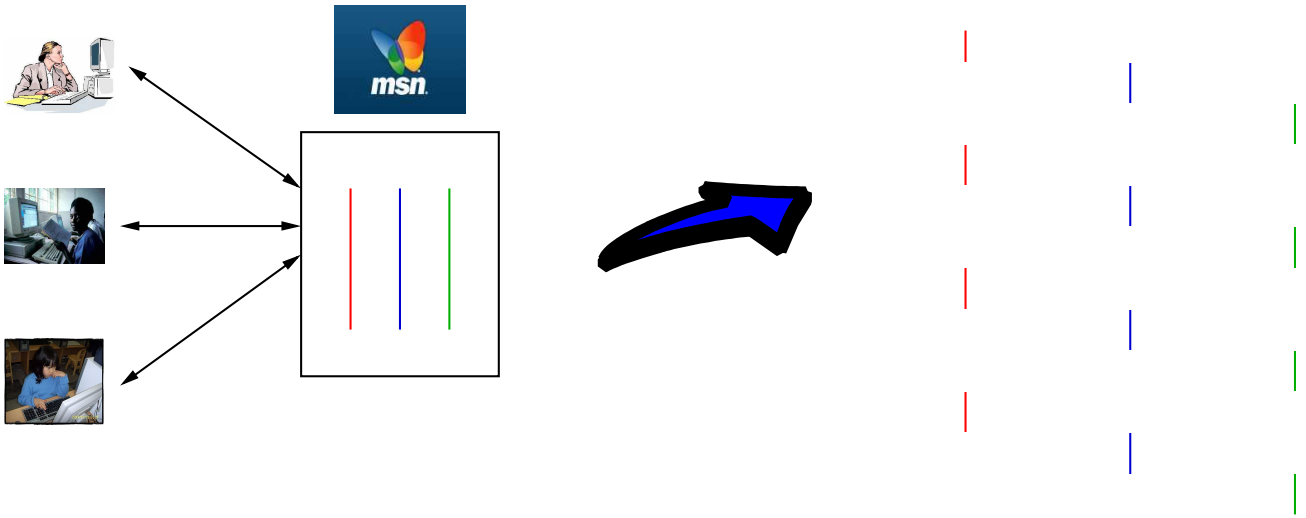
Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

Multi-threaded Programs

- Multi-threaded programs occur everywhere.
- They are non-deterministic.
- Interleaving



Introduction

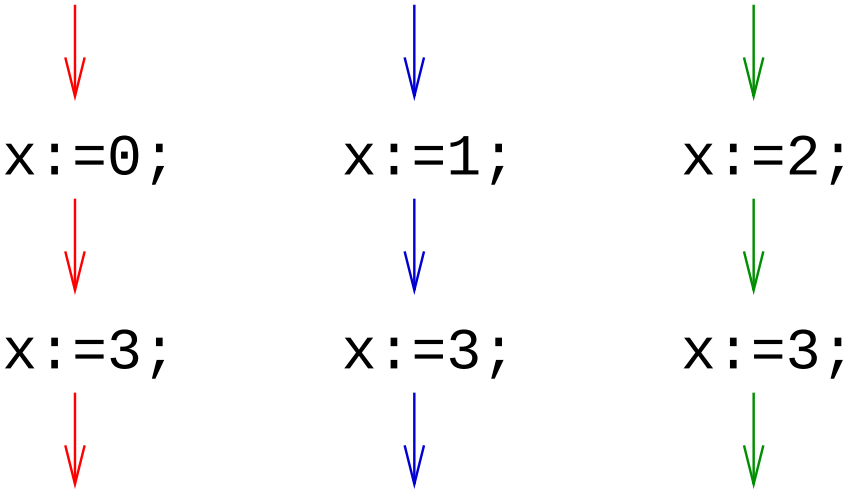
Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

Multi-threaded Programs

- Multi-threaded programs occur everywhere.
- They are non-deterministic.
- Interleaving



Introduction

Reverse Execution

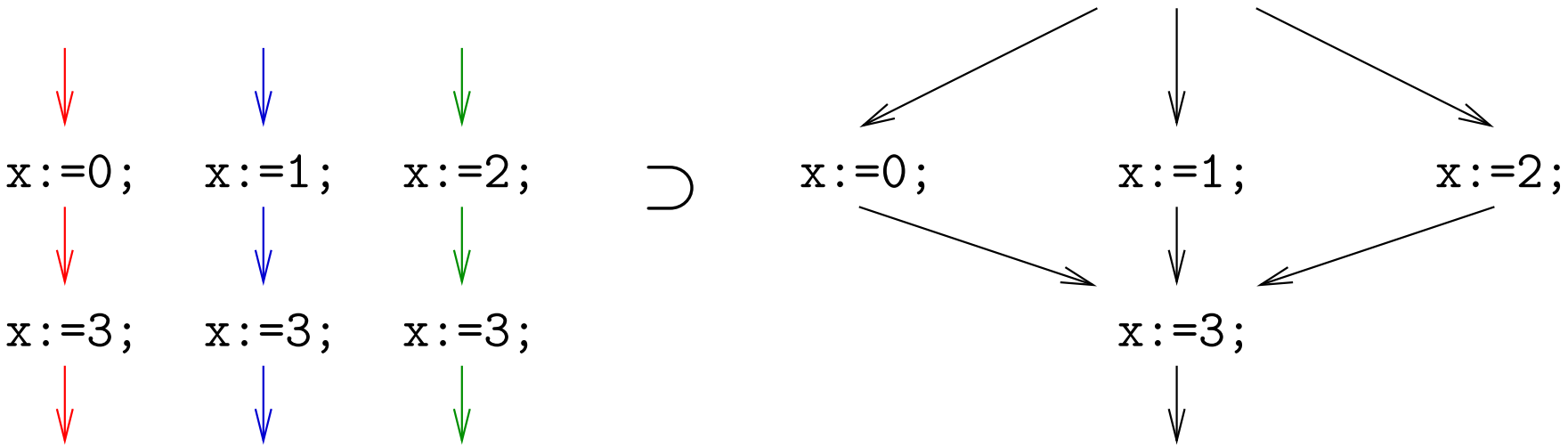
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

Multi-threaded Programs

- Multi-threaded programs occur everywhere.
- They are non-deterministic.
- Interleaving $\Rightarrow$ Non-determinism



Introduction

Reverse Execution

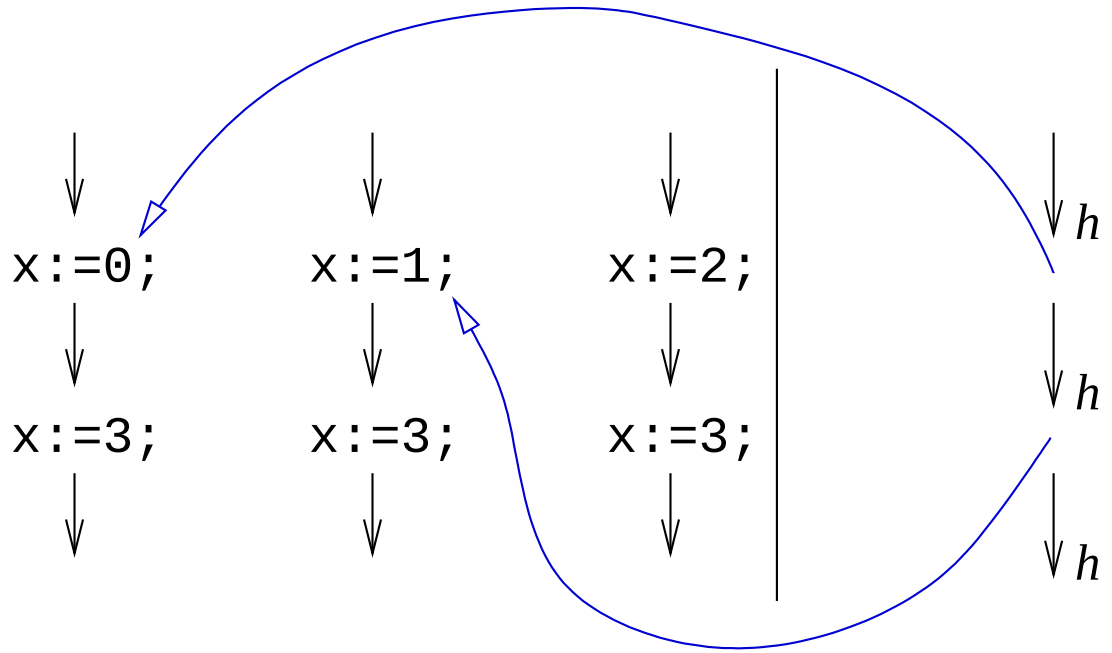
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

Dynamic Reverse Code Generation

- proposed in [Lee, V&D'06] and Chapter 5 of the dissertation
- Scales to multi-threaded programs
- Analyzes a runtime history of statements
- On-demand-basis generation at runtime
- A runtime history is essential for the reverse execution in non-deterministic programs, regardless of methods
- There is no way to infer non-deterministic choices



Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example (Chapter 8)

Program Validation by Symbolic Execution

Dynamic Reverse Code Generation

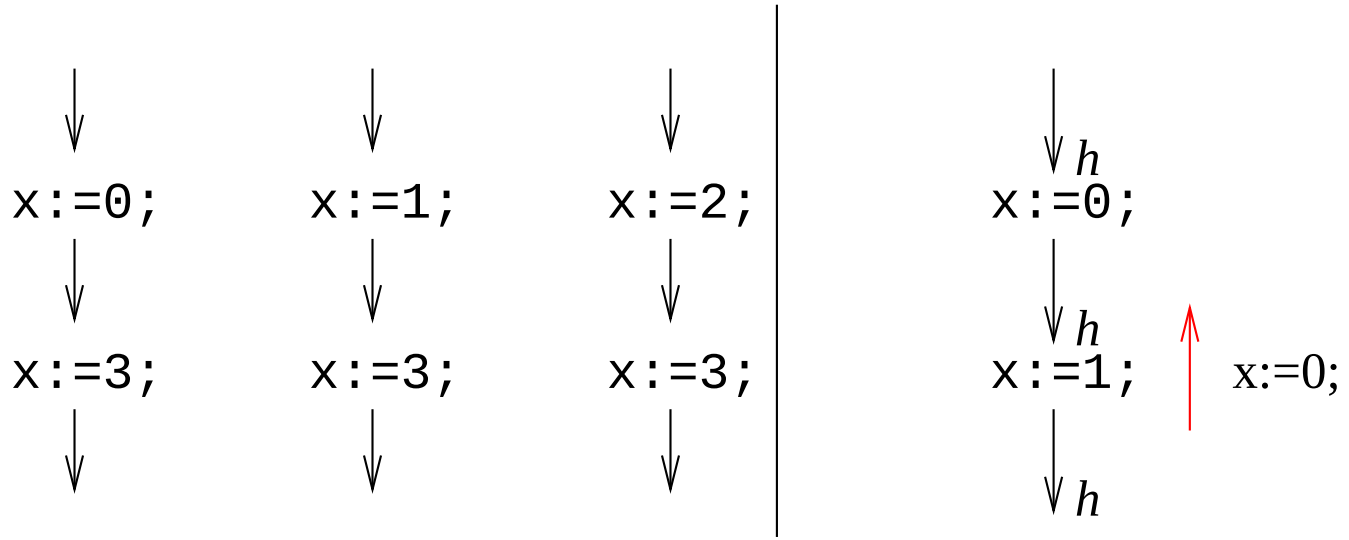
Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example (Chapter 8)

Program Validation by Symbolic Execution

- proposed in [Lee, V&D'06] and Chapter 5 of the dissertation
- Scales to multi-threaded programs
- Analyzes a runtime history of statements
- On-demand-basis generation at runtime
- A runtime history is essential for the reverse execution in non-deterministic programs, regardless of methods
- There is no way to infer non-deterministic choices



Static vs. Dynamic Reverse Code Generation

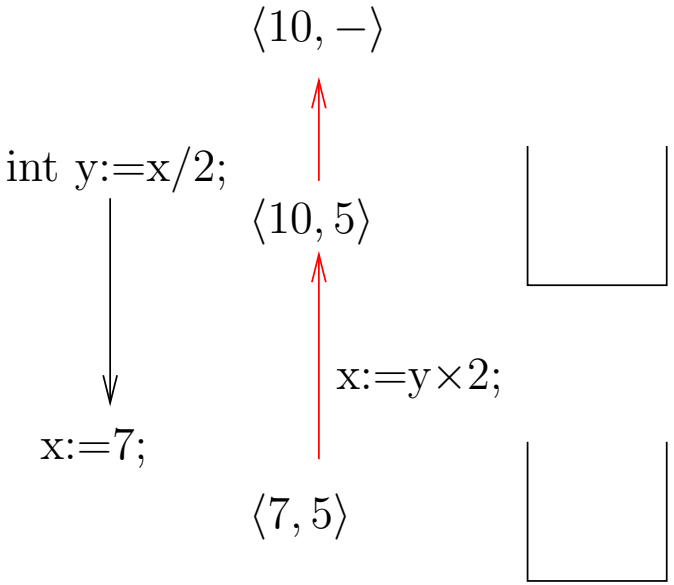
Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

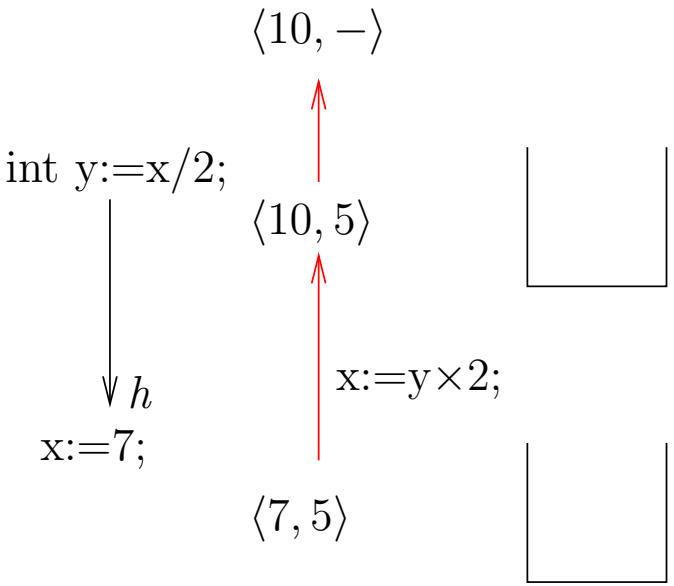
Program Validation by Symbolic Execution

- Source program
- Feasible only for deterministic programs
- Value-insensitive



- Minimal runtime overhead

- Runtime history
- Feasible for non-deterministic programs
- Value-sensitive



- Substantial runtime overhead
  - Multi-core processors
  - Interactive debugging

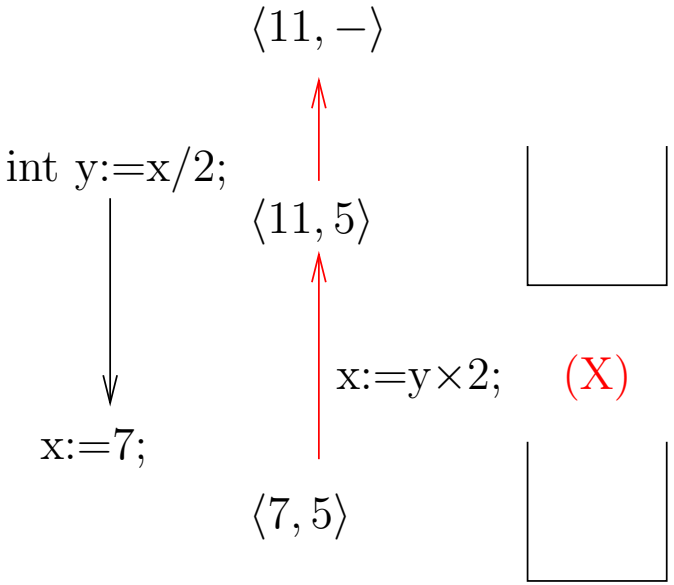
Static vs. Dynamic Reverse Code Generation

Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

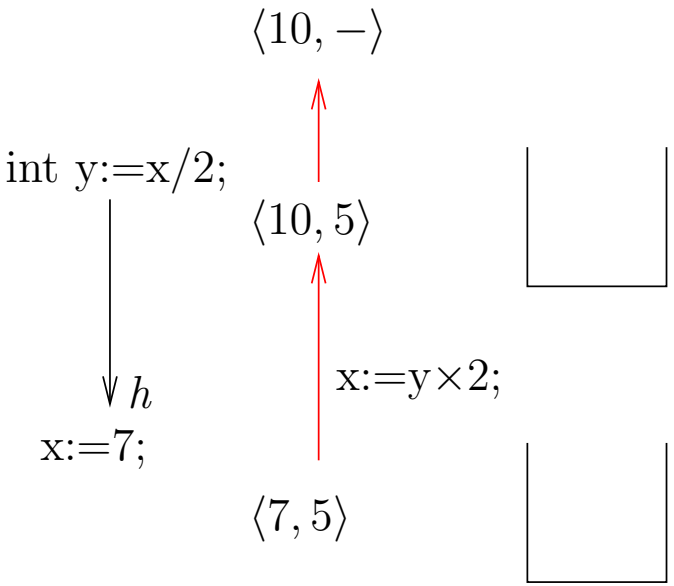
(Chapter 8)
Program Validation by Symbolic Execution

- Source program
- Feasible only for deterministic programs
- Value-insensitive



- Minimal runtime overhead

- Runtime history
- Feasible for non-deterministic programs
- Value-sensitive



- Substantial runtime overhead
  - Multi-core processors
  - Interactive debugging

Static vs. Dynamic Reverse Code Generation

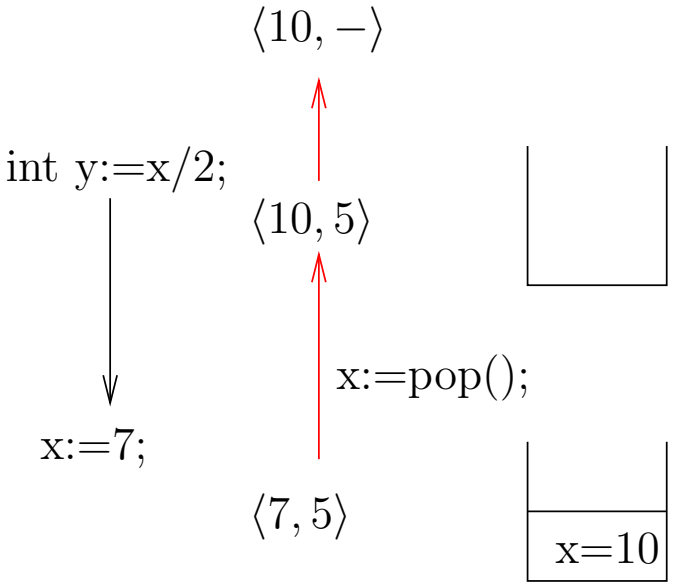
Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

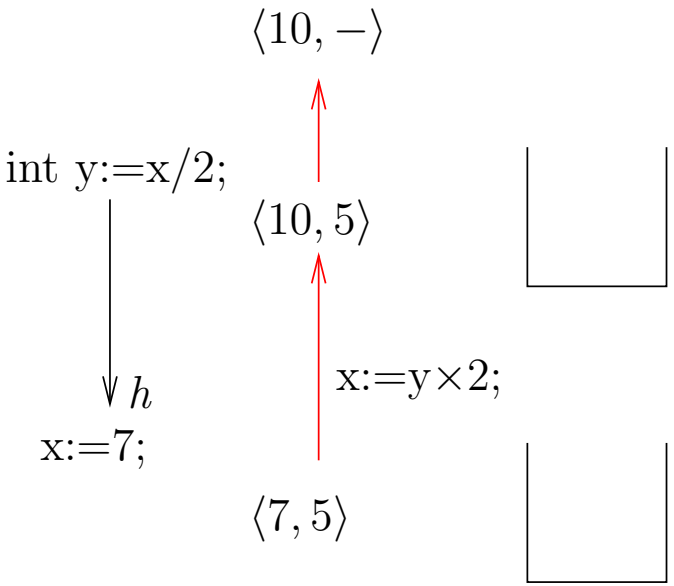
Program Validation by Symbolic Execution

- Source program
- Feasible only for deterministic programs
- Value-insensitive



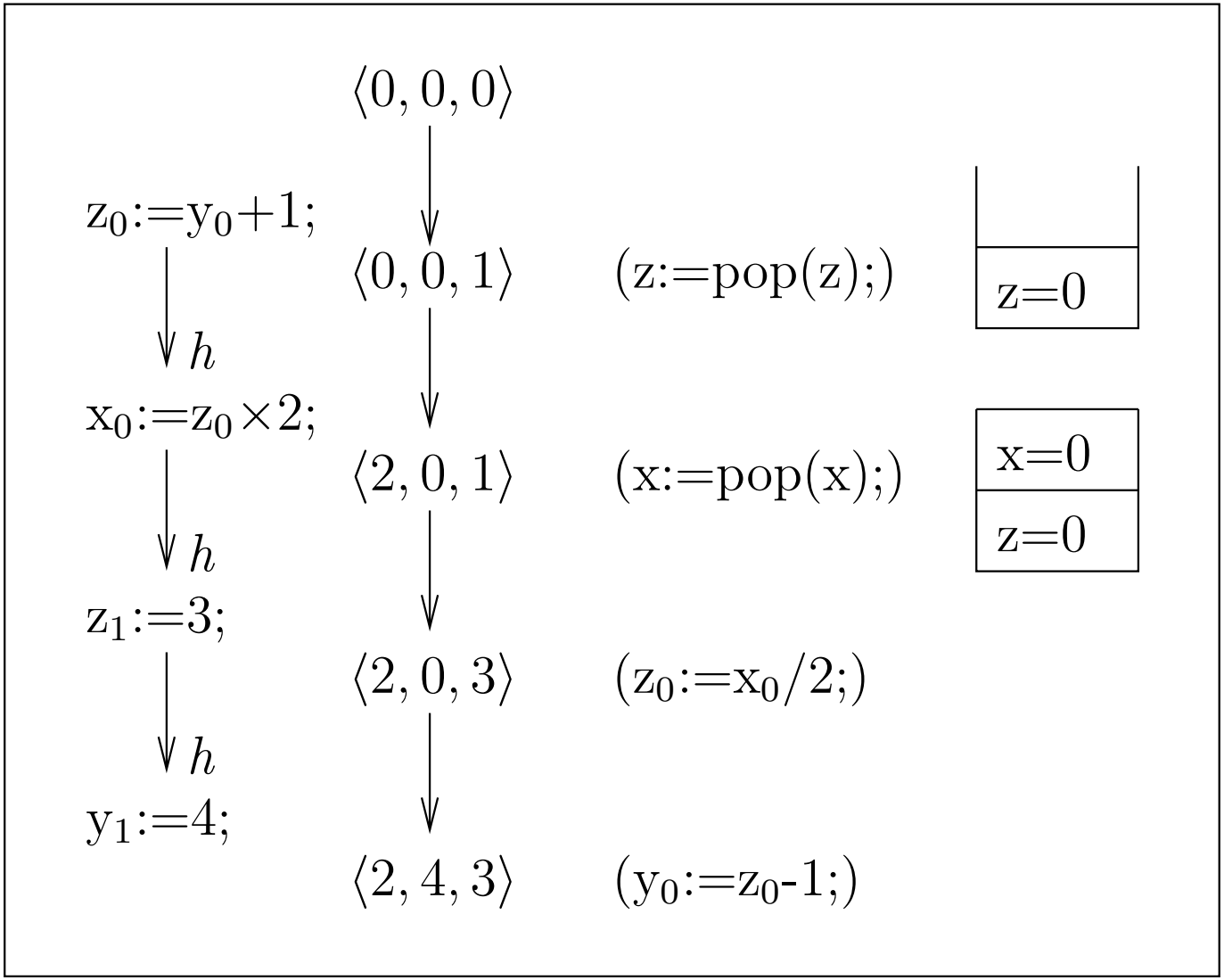
- Minimal runtime overhead

- Runtime history
- Feasible for non-deterministic programs
- Value-sensitive



- Substantial runtime overhead
  - Multi-core processors
  - Interactive debugging

Dynamic Reverse Code Generation



Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Verification by Symbolic Execution

Dynamic Reverse Code Generation

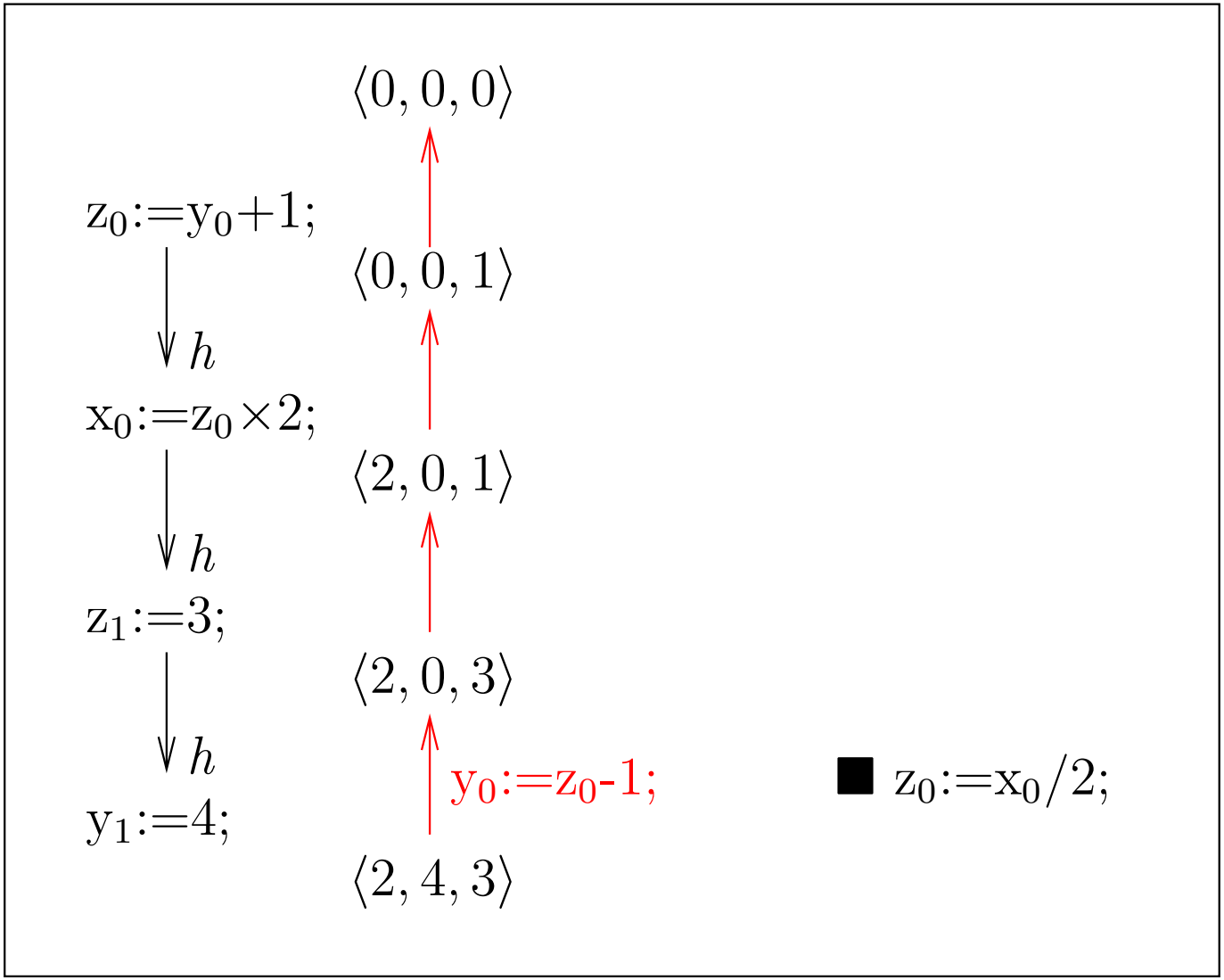
Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution



Dynamic Reverse Code Generation

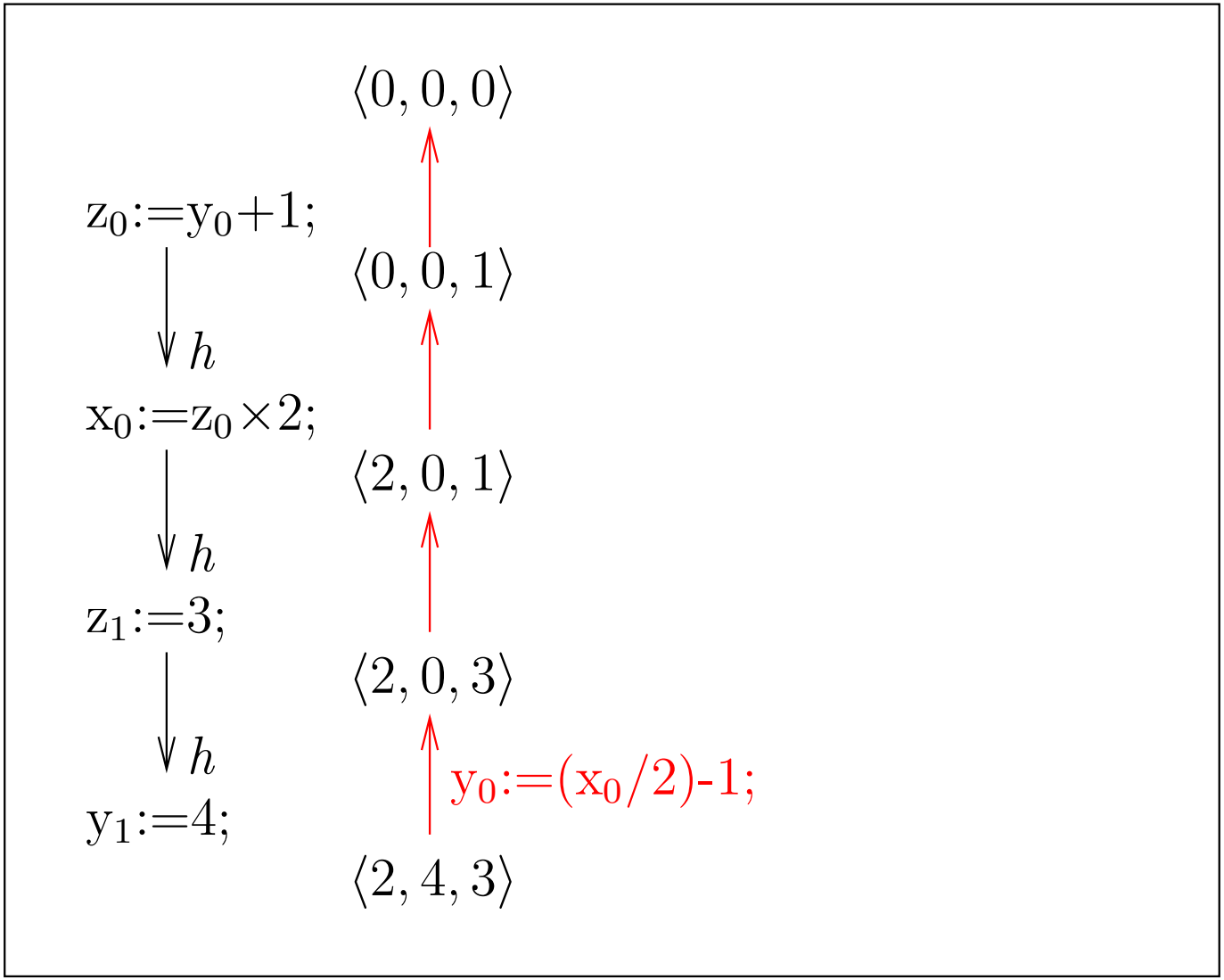
Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution



Dynamic Reverse Code Generation

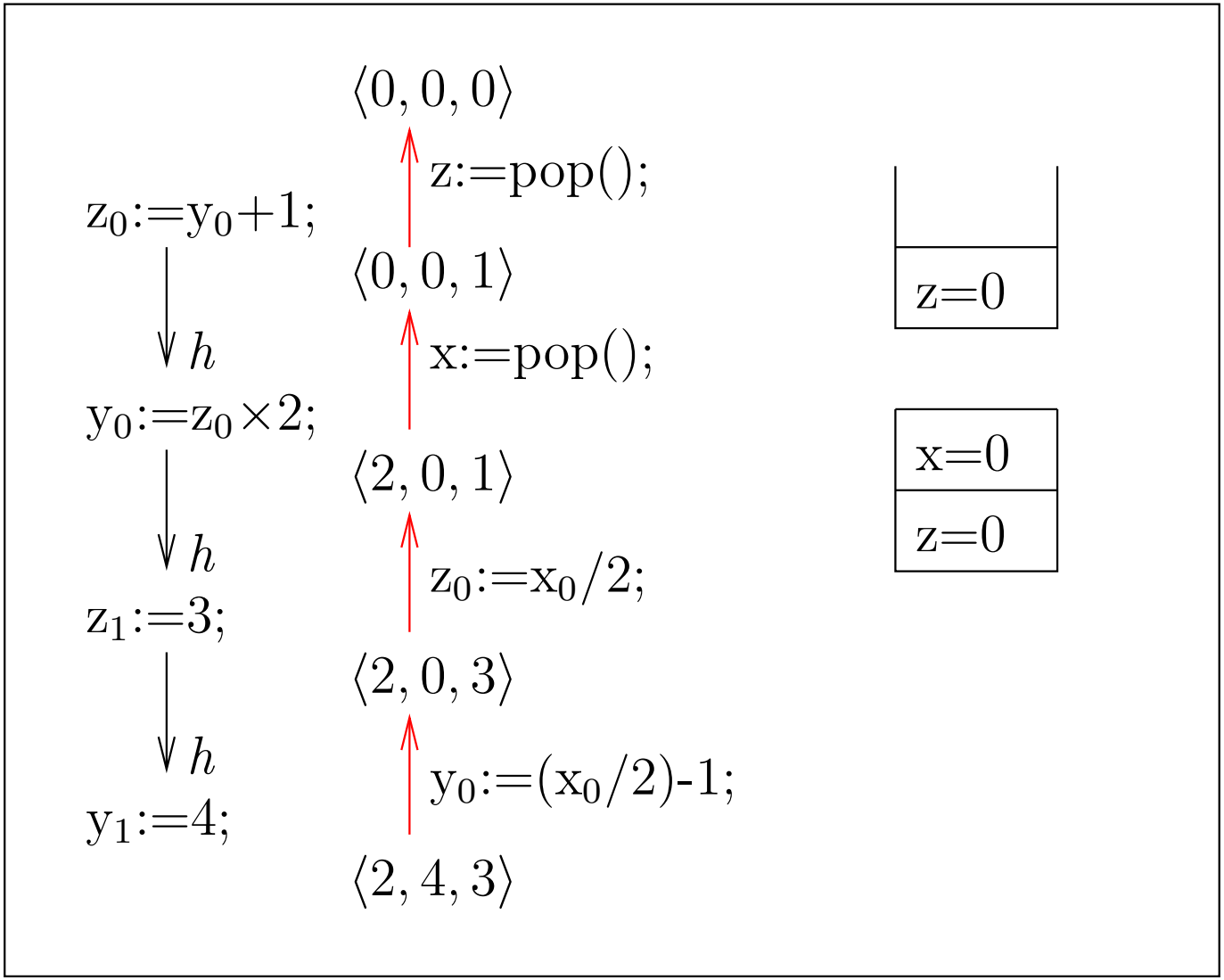
Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution



Multiple Variables

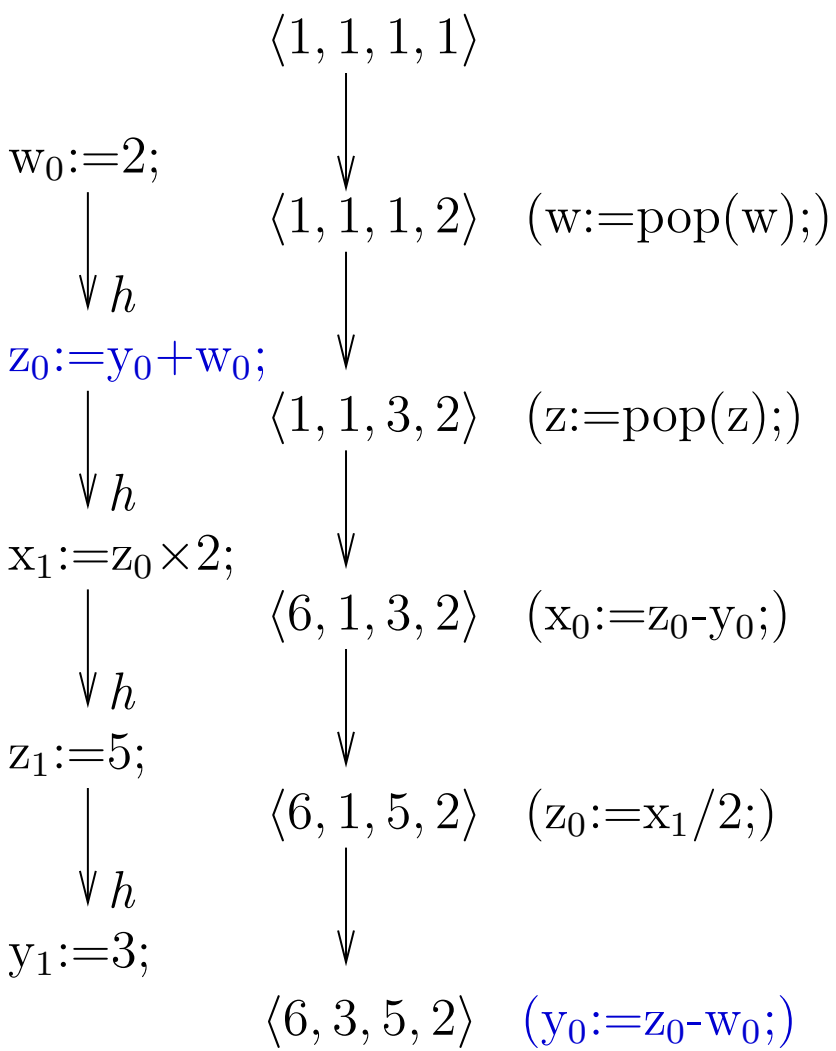
Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution



- $z_0=(\lambda t.t+w_0)(y_0)$
- $(\lambda t.t+w_0)^{-1}=(\lambda t.t-w_0)$
- $y_0=(\lambda t.t-w_0)(z_0)$

Multiple Variables

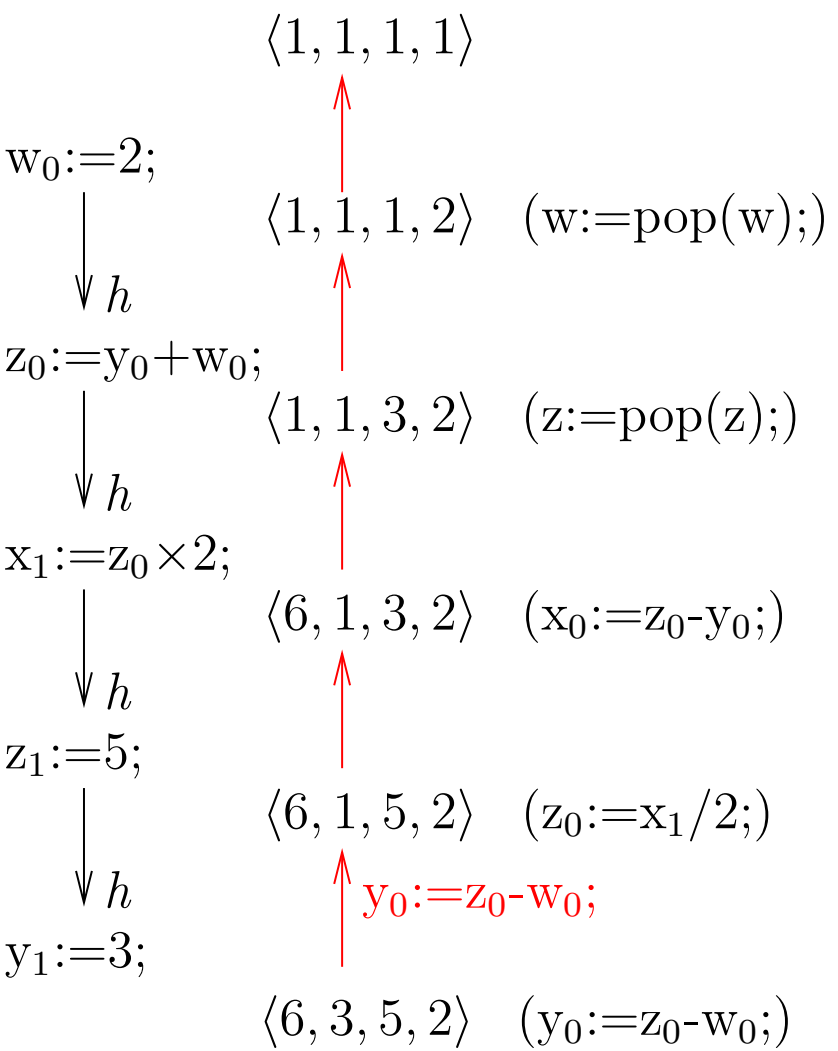
Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution



Multiple Variables

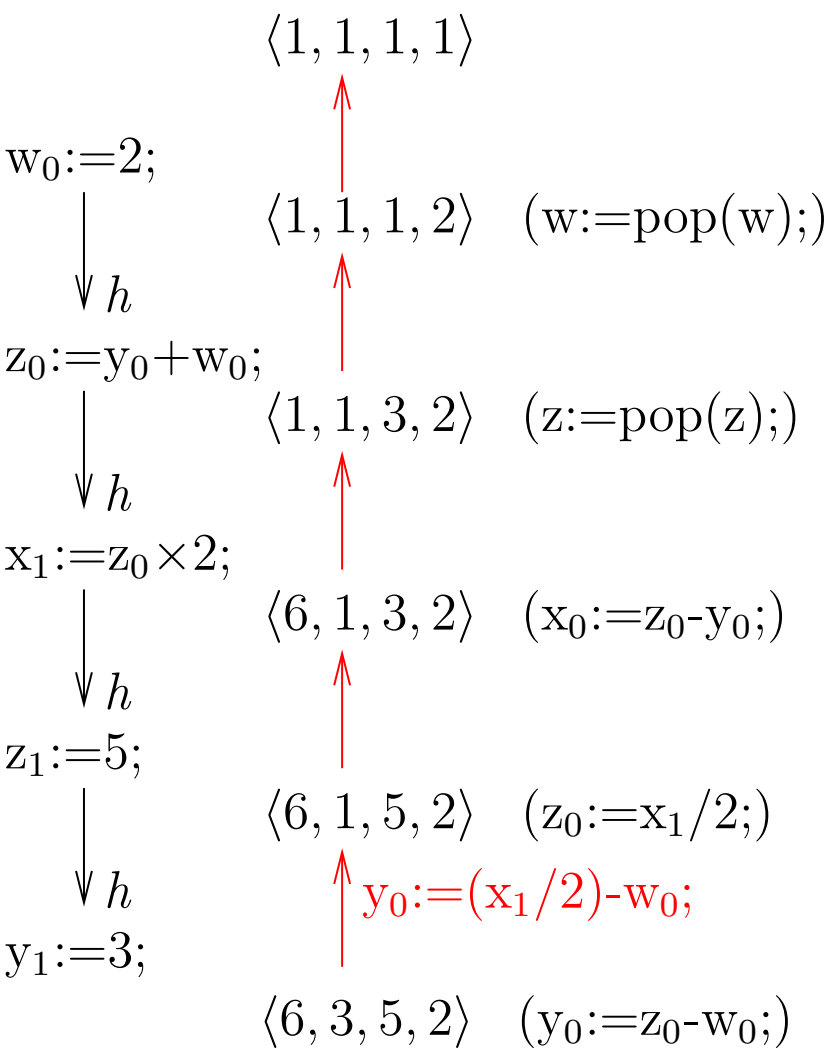
Introduction

Reverse Execution

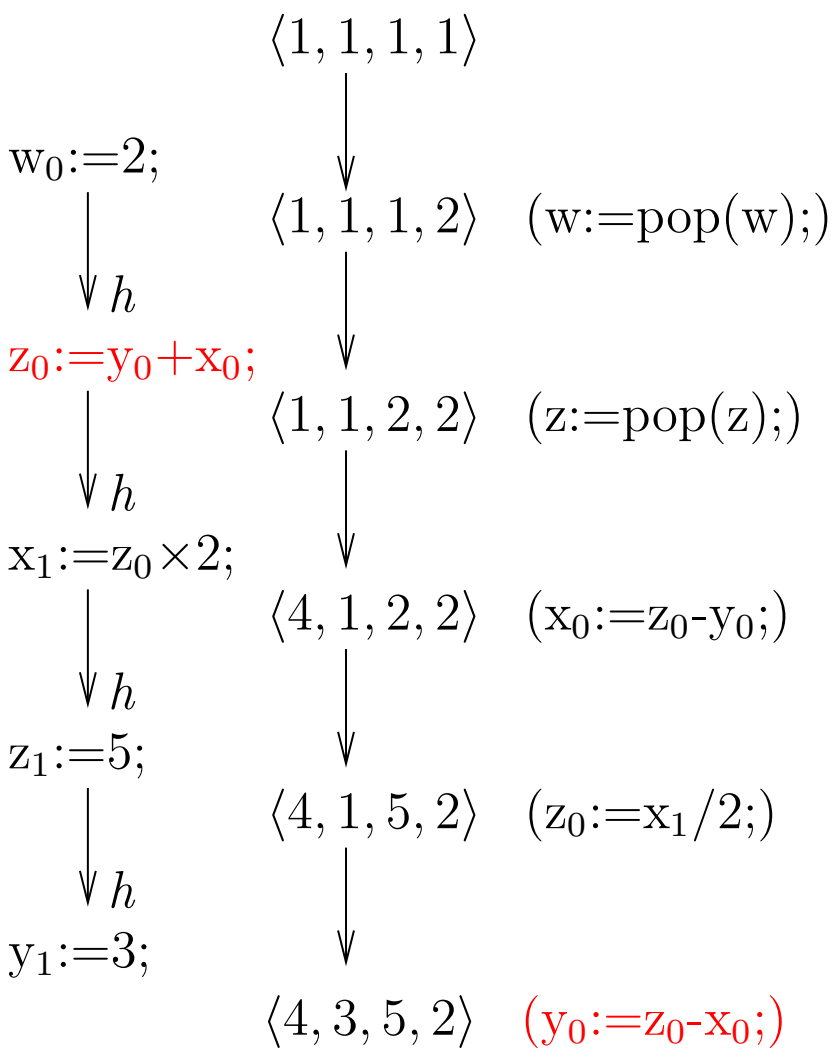
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution



Multiple Variables



- $z_0 = (\lambda t.t + x_0)(y_0)$
- $(\lambda t.t + x_0)^{-1} = (\lambda t.t - x_0)$
- $y_0 = (\lambda t.t - x_0)(z_0)$

Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Verification by Symbolic Execution

Multiple Variables

Introduction

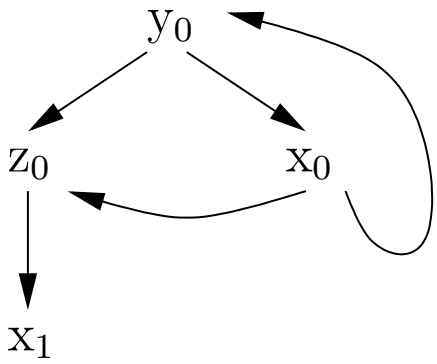
Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

$$
\begin{array}{ll}
 & \langle 1,1,1,1\rangle \\
 & \uparrow \\
w_0{:=}2; & \\
\downarrow h & \\
 & \langle 1,1,1,2\rangle \quad (w{:=}pop(w);) \\
 & \uparrow \\
z_0{:=}y_0{+}x_0; & \\
\downarrow h & \\
 & \langle 1,1,2,2\rangle \quad (z{:=}pop(z);) \\
 & \uparrow \\
x_1{:=}z_0{\times}2; & \\
\downarrow h & \\
 & \langle 4,1,2,2\rangle \quad (x_0{:=}z_0\text{-}y_0;) \\
 & \uparrow \\
z_1{:=}5; & \\
\downarrow h & \\
 & \langle 4,1,5,2\rangle \quad (z_0{:=}x_1/2;) \\
 & \uparrow \\
\downarrow h & \quad y_0{:=}(x_1/2)\text{-}(z_0\text{-}y_0); \\
y_1{:=}3; & \\
 & \langle 4,3,5,2\rangle \quad (y_0{:=}z_0\text{-}x_0;)
\end{array}
$$



Multiple Variables

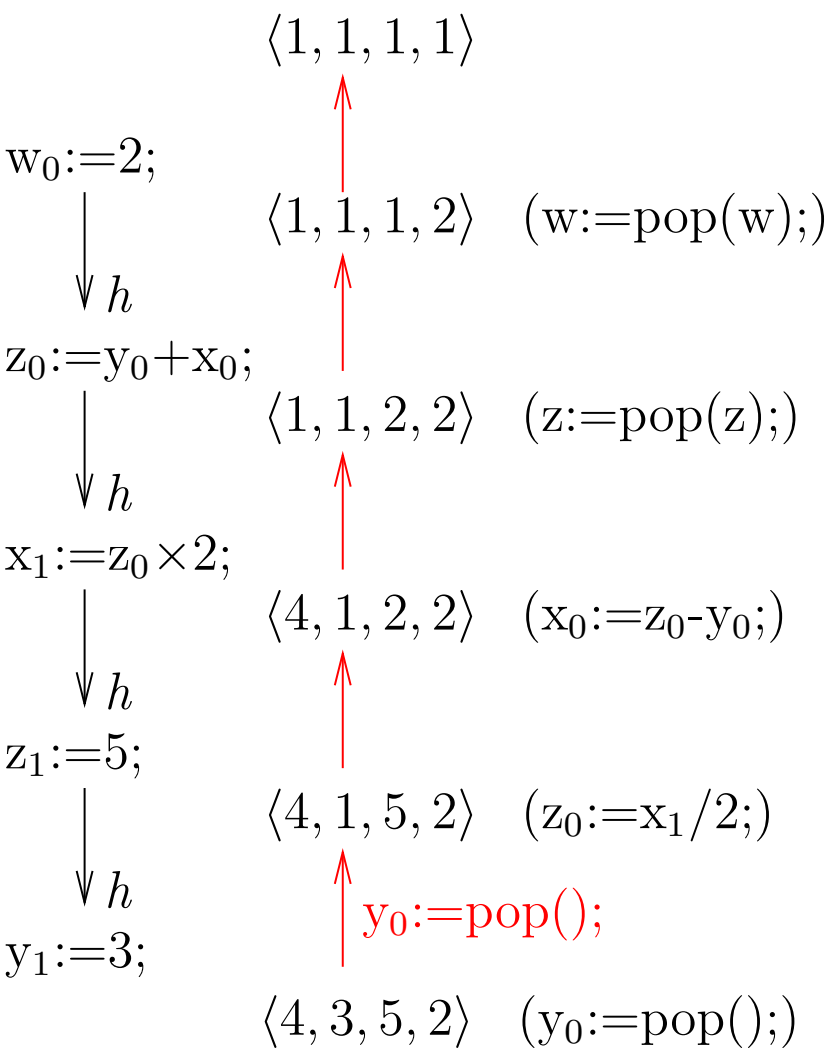
Introduction

Reverse Execution

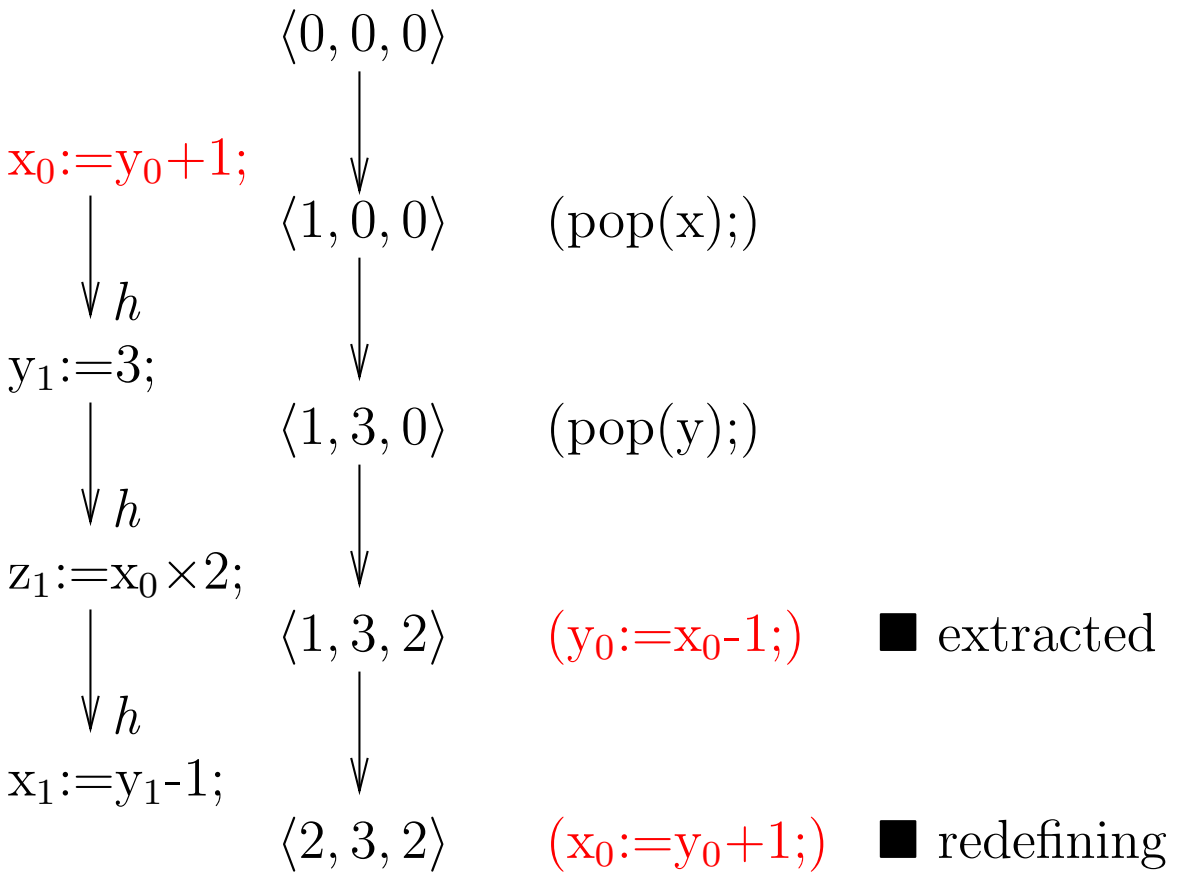
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution



Combination



Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

Combination

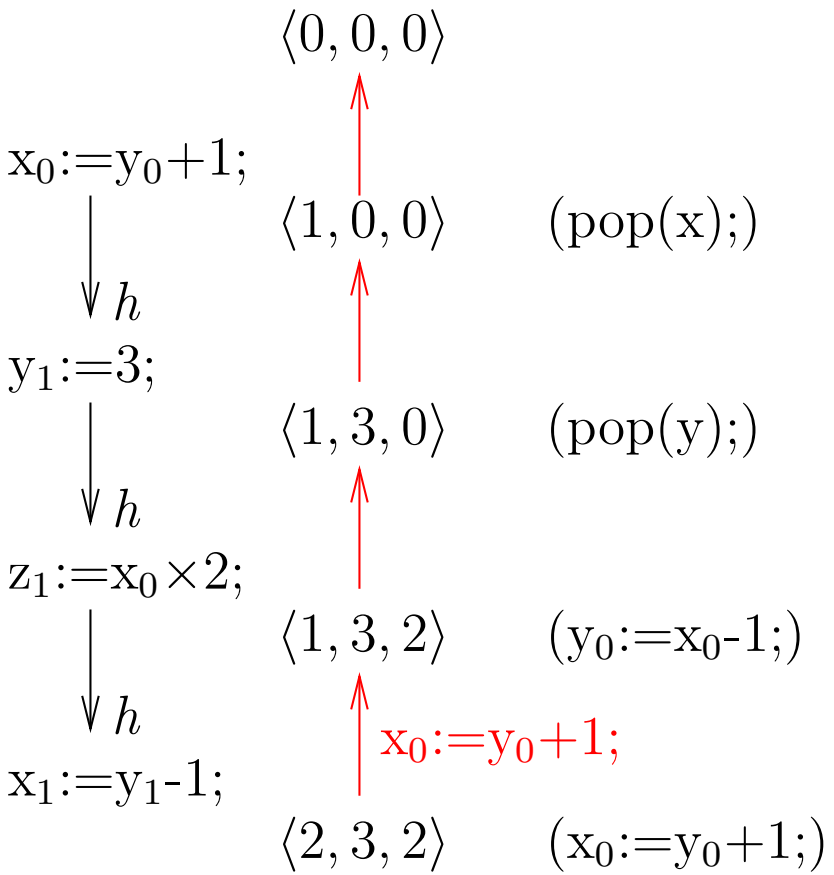
Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution



Combination

Introduction

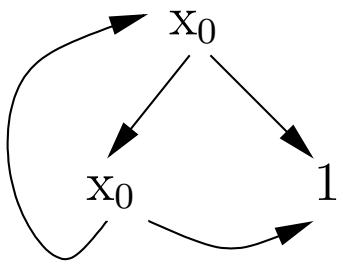
Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

$$
\begin{array}{lll}
 & \langle 0,0,0\rangle & \\
 & \uparrow & \\
x_0{:=}y_0{+}1; & \langle 1,0,0\rangle & (\mathrm{pop}(x);) \\
\downarrow h & \uparrow & \\
y_1{:=}3; & \langle 1,3,0\rangle & (\mathrm{pop}(y);) \\
\downarrow h & \uparrow & \\
z_1{:=}x_0{\times}2; & \langle 1,3,2\rangle & (y_0{:=}x_0{-}1;) \\
\downarrow h & \uparrow & \\
 & x_0{:=}(x_0{-}1){+}1; & \\
x_1{:=}y_1{-}1; & \langle 2,3,2\rangle & (x_0{:=}y_0{+}1;)
\end{array}
$$



Combination

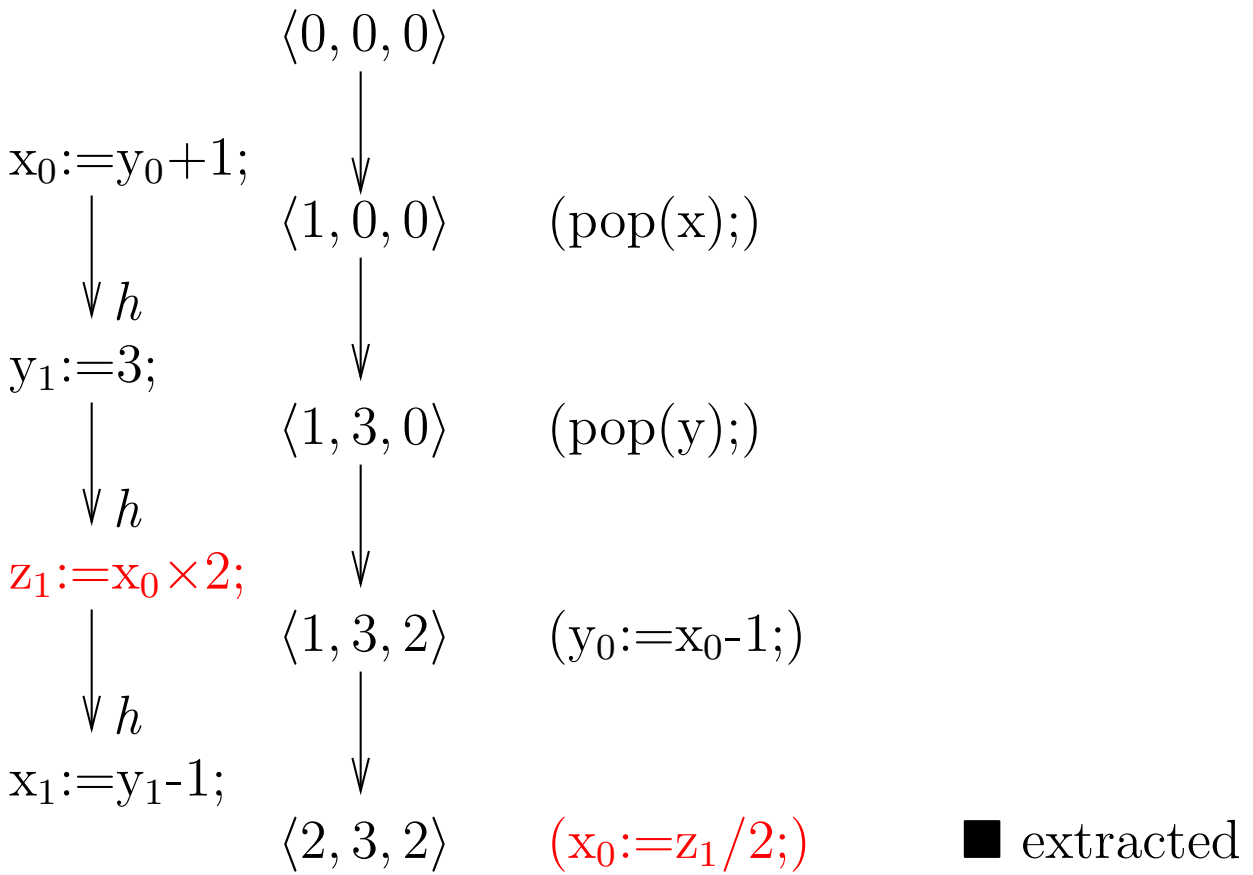
Introduction

Reverse Execution

- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution



Soundness

Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example

(Chapter 8)

Program Validation by Symbolic Execution

The value of v restored by dynamic reverse code generation is the same as the previous value of v.

Proof. Suppose that the reverse code for v is $v := f(u_1, u_2, \ldots, u_n)$

1. The value of u_k either is immediately available or can be restored.
2. A reverse code is deterministically calculated.
3. The restoration of u_k does not involve v.

□

- A deterministic algorithm exploiting extraction-from-use was shown in [Lee, V&D'06].
- Conjecture: also true when redefinition/combination is used.

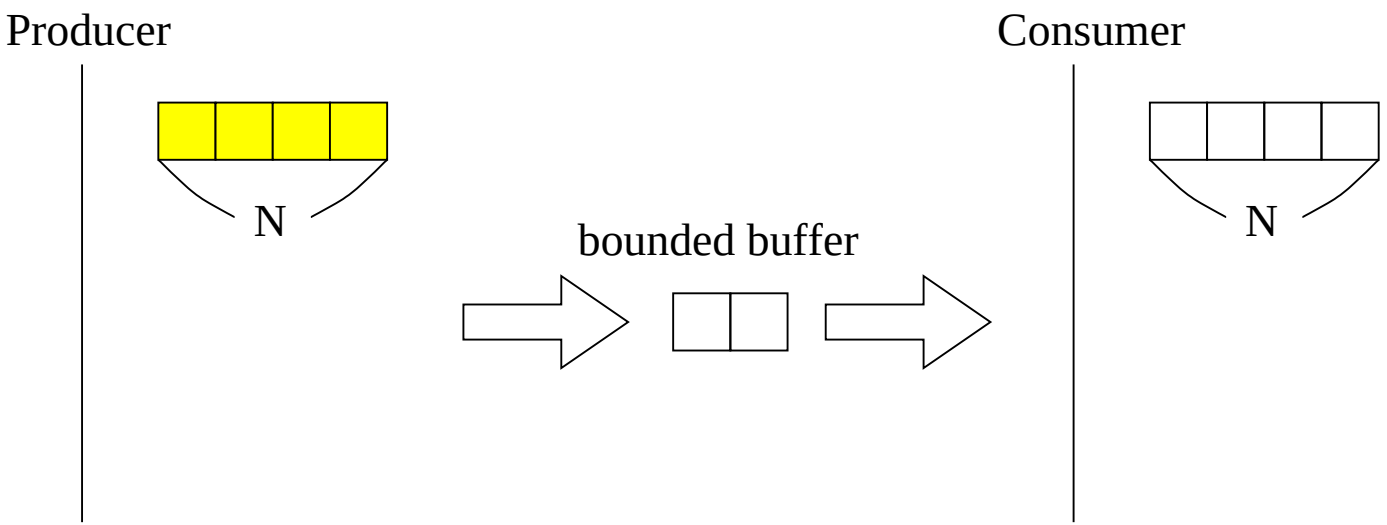
Bounded Buffer Example (Chapter 8)

Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example (Chapter 8)

Program Validation by Symbolic Execution

- It is widely used.



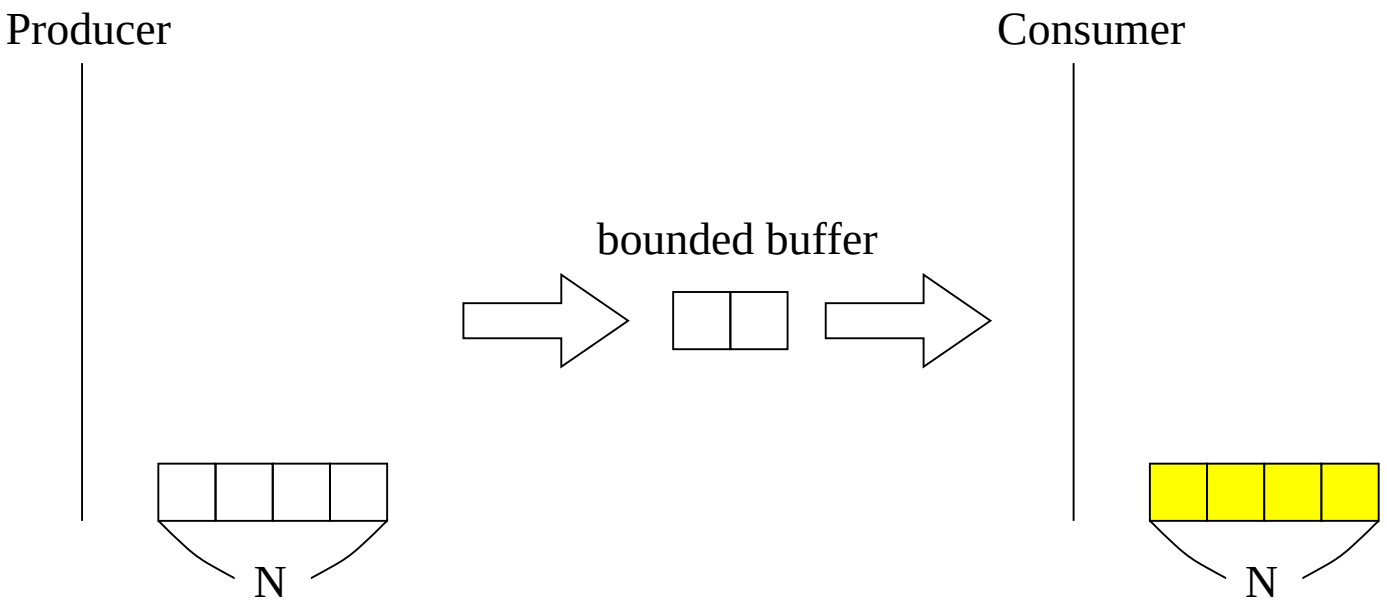
Bounded Buffer Example (Chapter 8)

Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example (Chapter 8)

Program Validation by Symbolic Execution

- It is widely used.



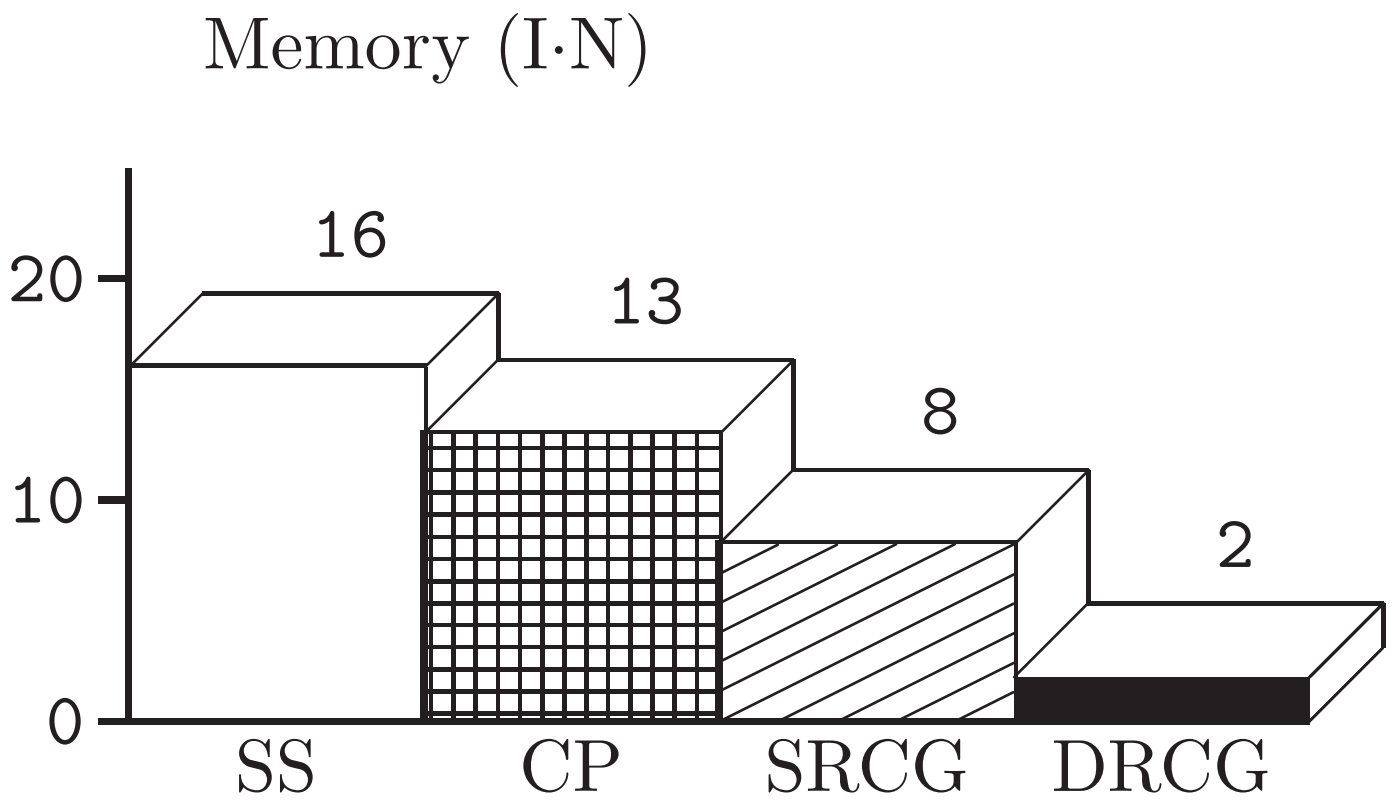
Bounded Buffer Example (Chapter 8)

Introduction

Reverse Execution
- What?
- Locating a Bug
- Locating a Bug by Iterative Execution
- Locating a Bug by Reverse Execution
- Reverse-Execution Methods
- Loop Example
- Reverse Execution by State-Saving
- Reverse Execution by Checkpointing
- Reverse Execution by Checkpointing
- Reverse Execution by Reverse Code
- Reverse Code Generation
- Static Reverse Code Generation
- Multi-threaded Programs
- Dynamic Reverse Code Generation
- Static vs. Dynamic Reverse Code Generation
- Dynamic Reverse Code Generation
- Multiple Variables
- Combination
- Soundness
- Bounded Buffer Example (Chapter 8)

Program Validation by Symbolic Execution

- It is widely used.



Introduction

Reverse Execution

Program Validation by Symbolic Execution
- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks

Program Validation by Symbolic Execution

Testing

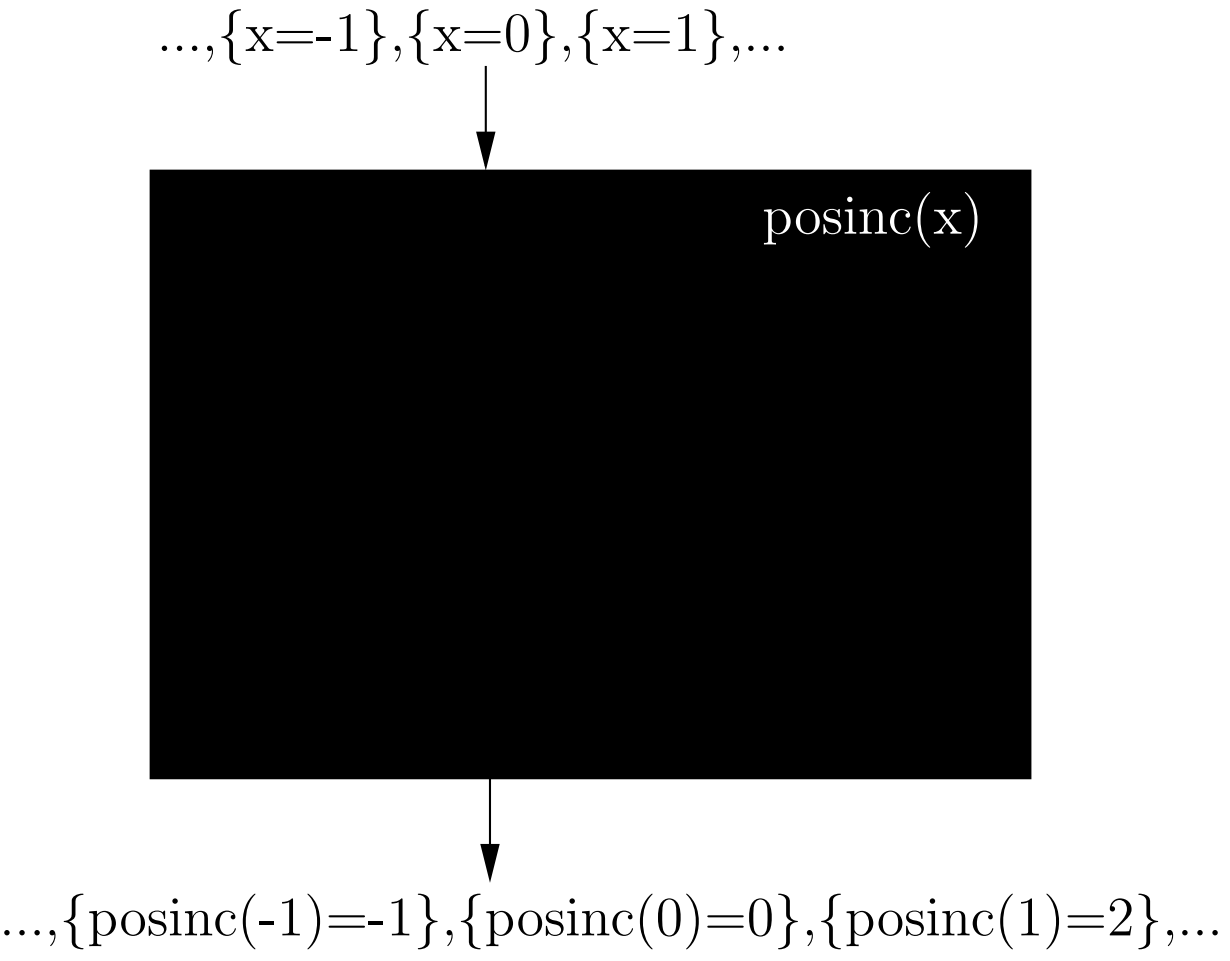
Introduction

Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks



- How many input/output pairs?

Symbolic Execution

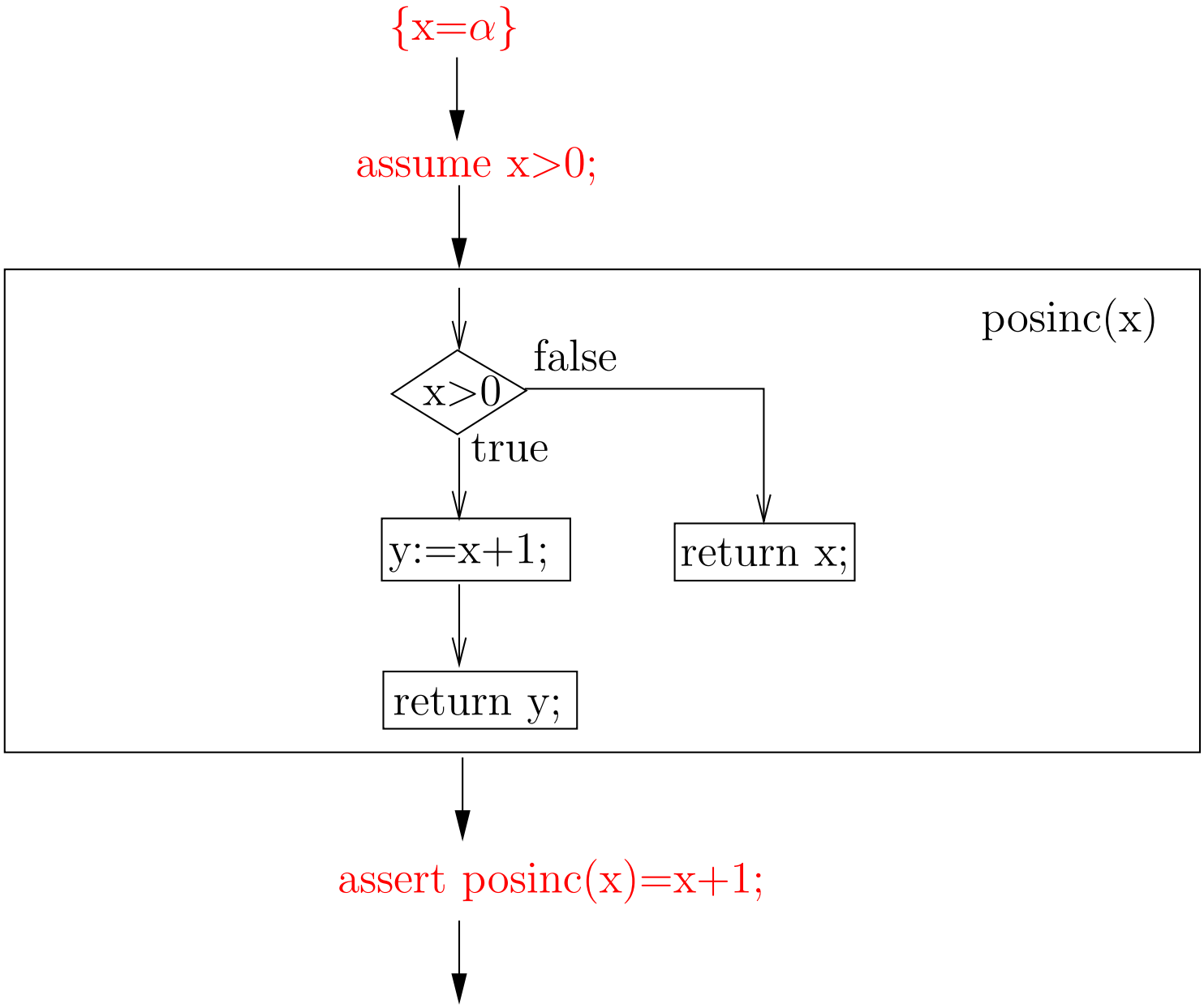
Introduction

Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks



Symbolic Execution

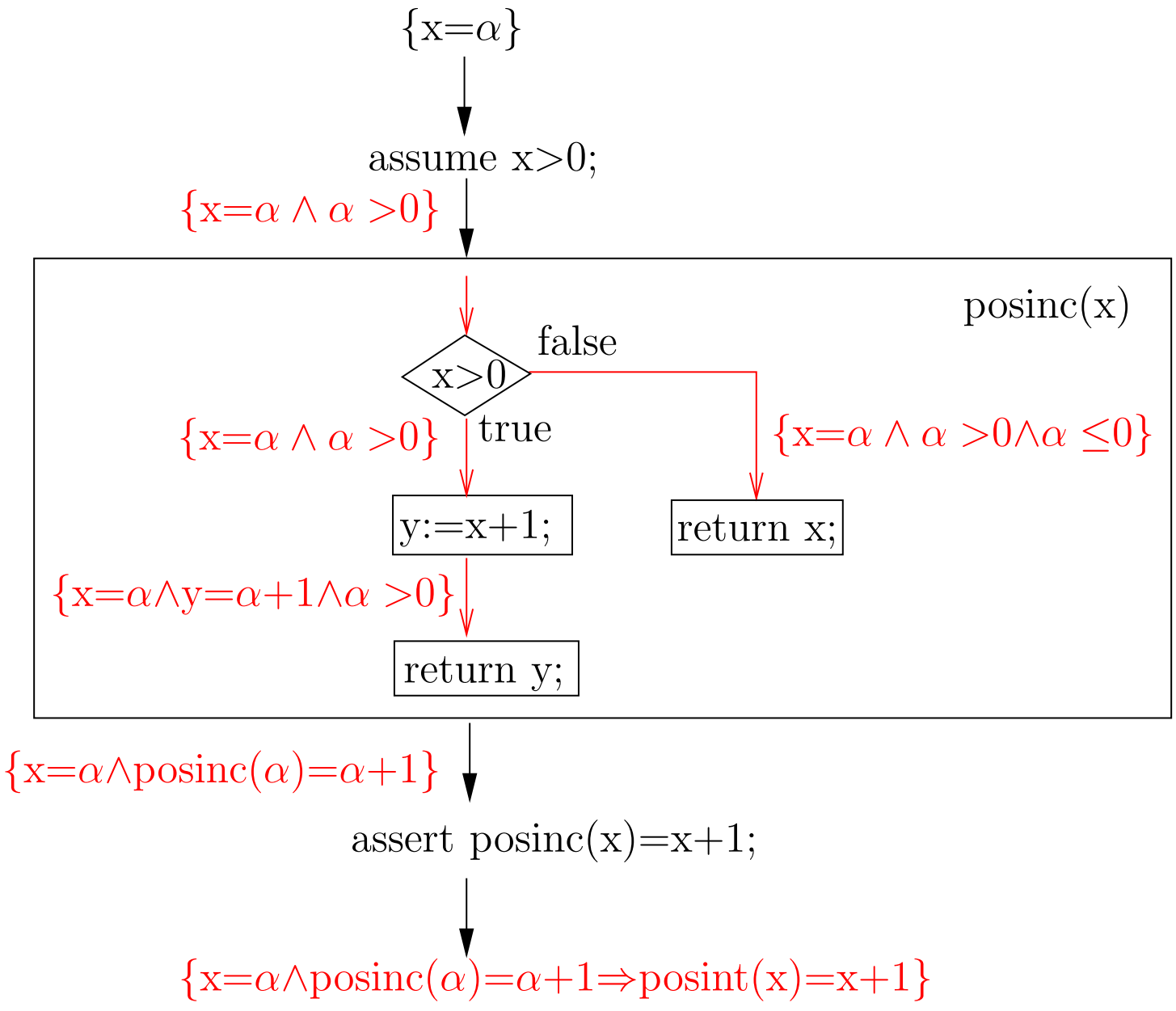
Introduction

Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks



Heap

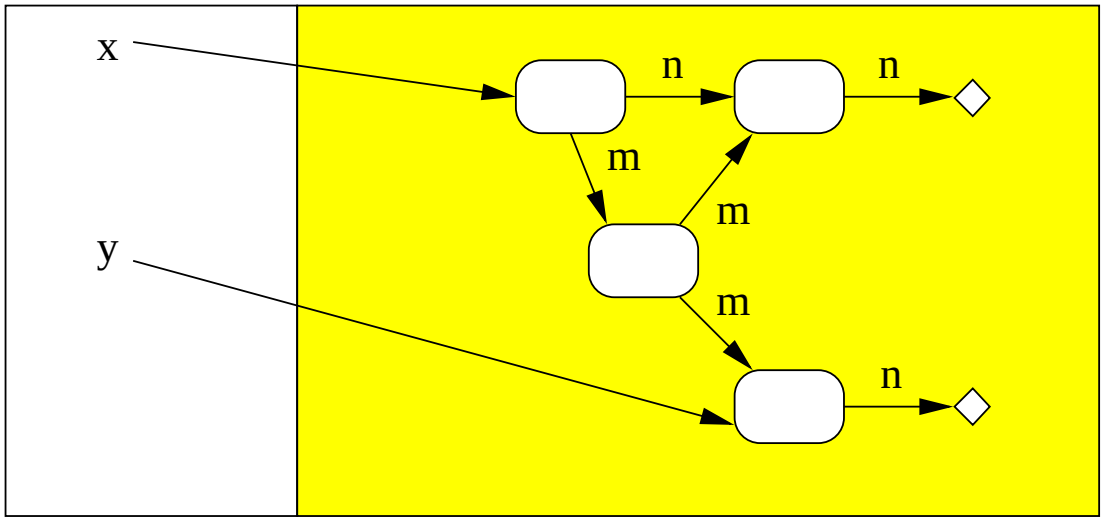
Introduction

Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks



Symbolic Execution on Heap

Introduction

Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks

$$\{x = \alpha \wedge \alpha \mapsto [f : \beta]\}$$
$$y := x.f;$$
$$\{x = \alpha \wedge y = \beta \wedge \alpha \mapsto [f : \beta]\}$$

$$\{x = \alpha \wedge y = \beta \wedge \beta \mapsto [f : -]\}$$
$$y.f := x;$$
$$\{x = \alpha \wedge y = \beta \wedge \beta \mapsto [f : \alpha]\}$$

Constraint Solver

Introduction

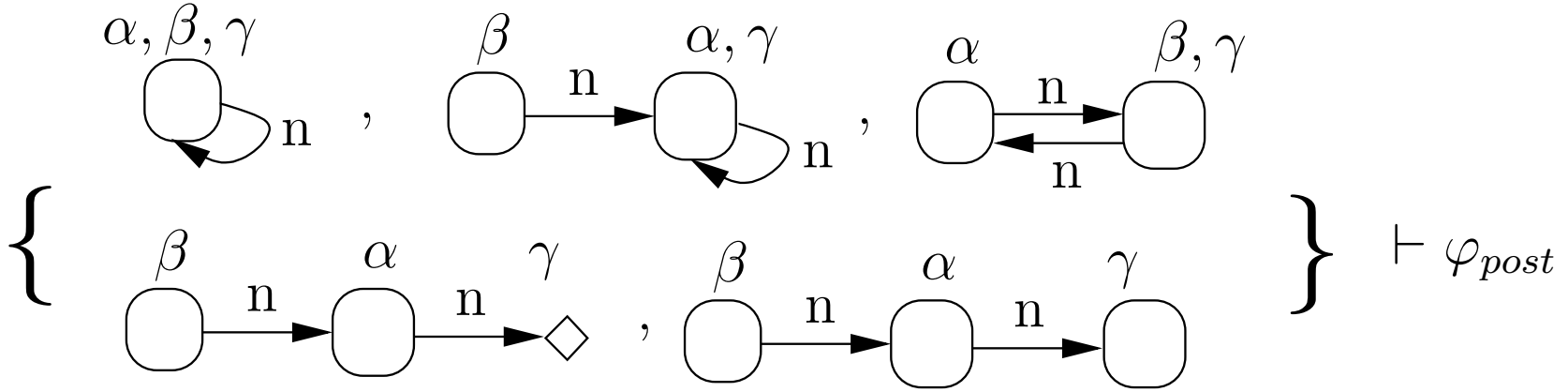
Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks

$\{\mathsf{c} = \alpha \wedge \mathsf{t} = \beta \wedge \mathsf{s} = \gamma \wedge \alpha \mapsto [\mathsf{n} : \gamma] \wedge \beta \mapsto [\mathsf{n} : \alpha]\}$
assert φ_{post};
$\{\mathsf{c} = \alpha \wedge \mathsf{t} = \beta \wedge \mathsf{s} = \gamma \wedge \alpha \mapsto [\mathsf{n} : \gamma] \wedge \beta \mapsto [\mathsf{n} : \alpha] \Rightarrow \varphi_{post}\}$



Lazy Initialization

Introduction

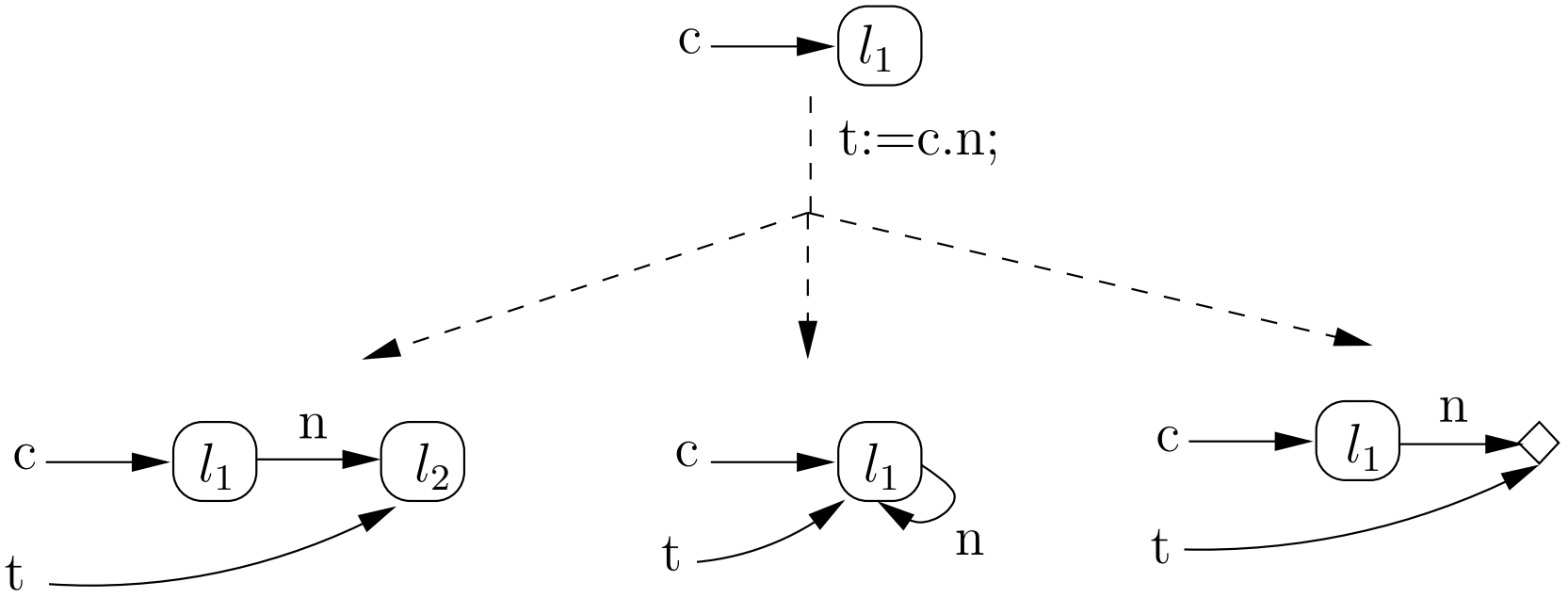
Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks

- introduced in [Khurshid, Păsăreanu and Visser, TACAS'03]



$$type(l_1) \leq type(\mathsf{n})$$

Lazy Initialization

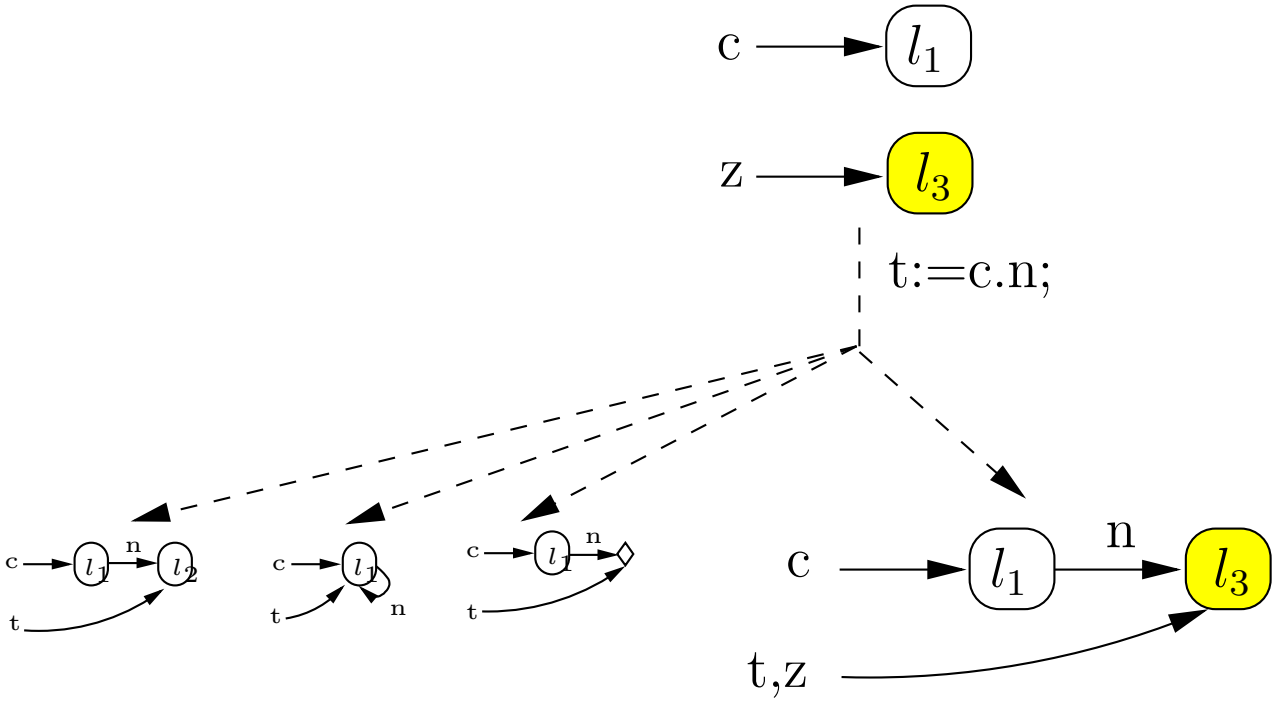
Introduction

Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks



$$type(l_3) \leq type(\mathsf{n})$$

$$age(l_3) \geq age(l_1)$$

Lazier Initialization [Deng, Lee and Robby, ASE'06]

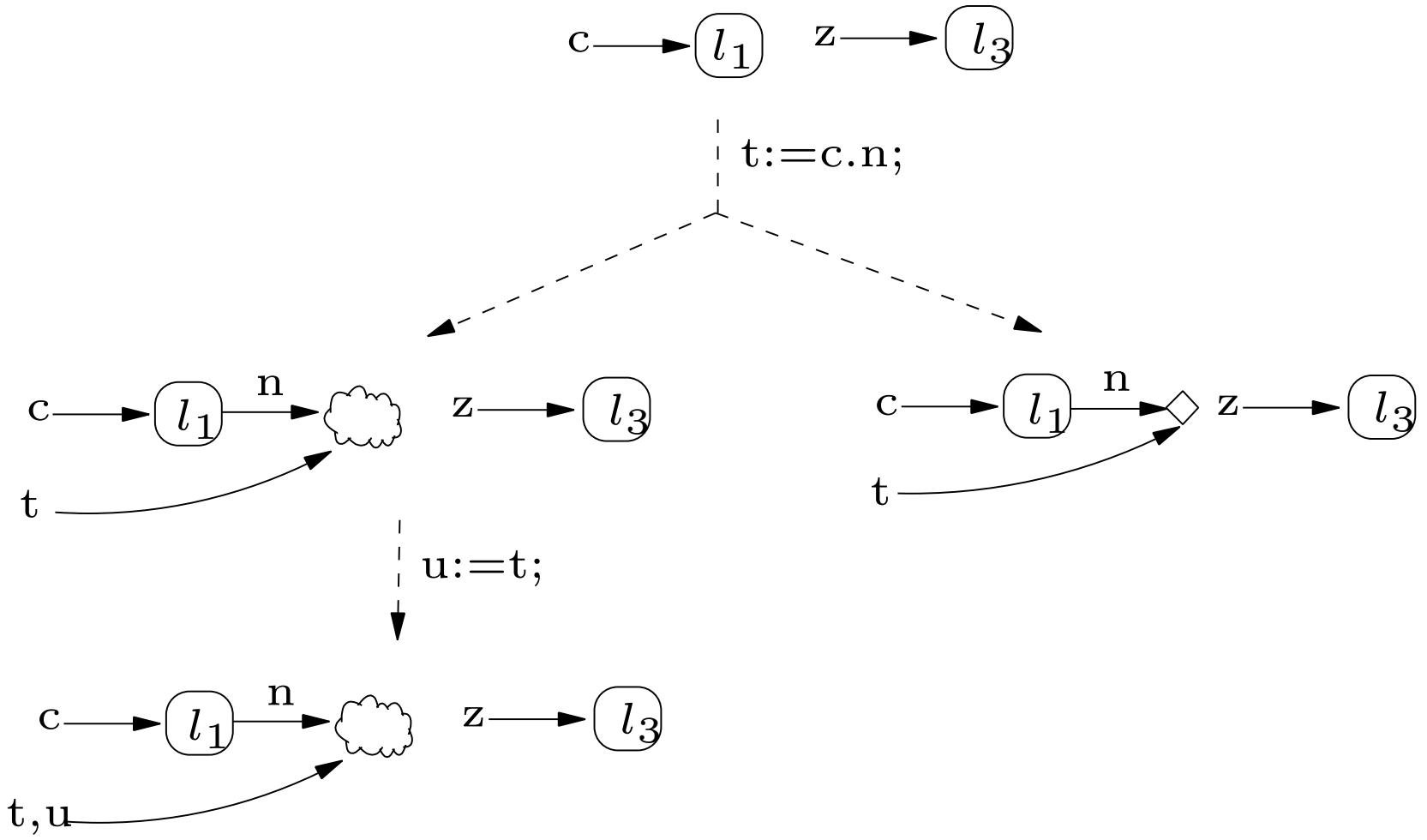
Introduction

Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks



Lazier Initialization [Deng, Lee and Robby, ASE'06]

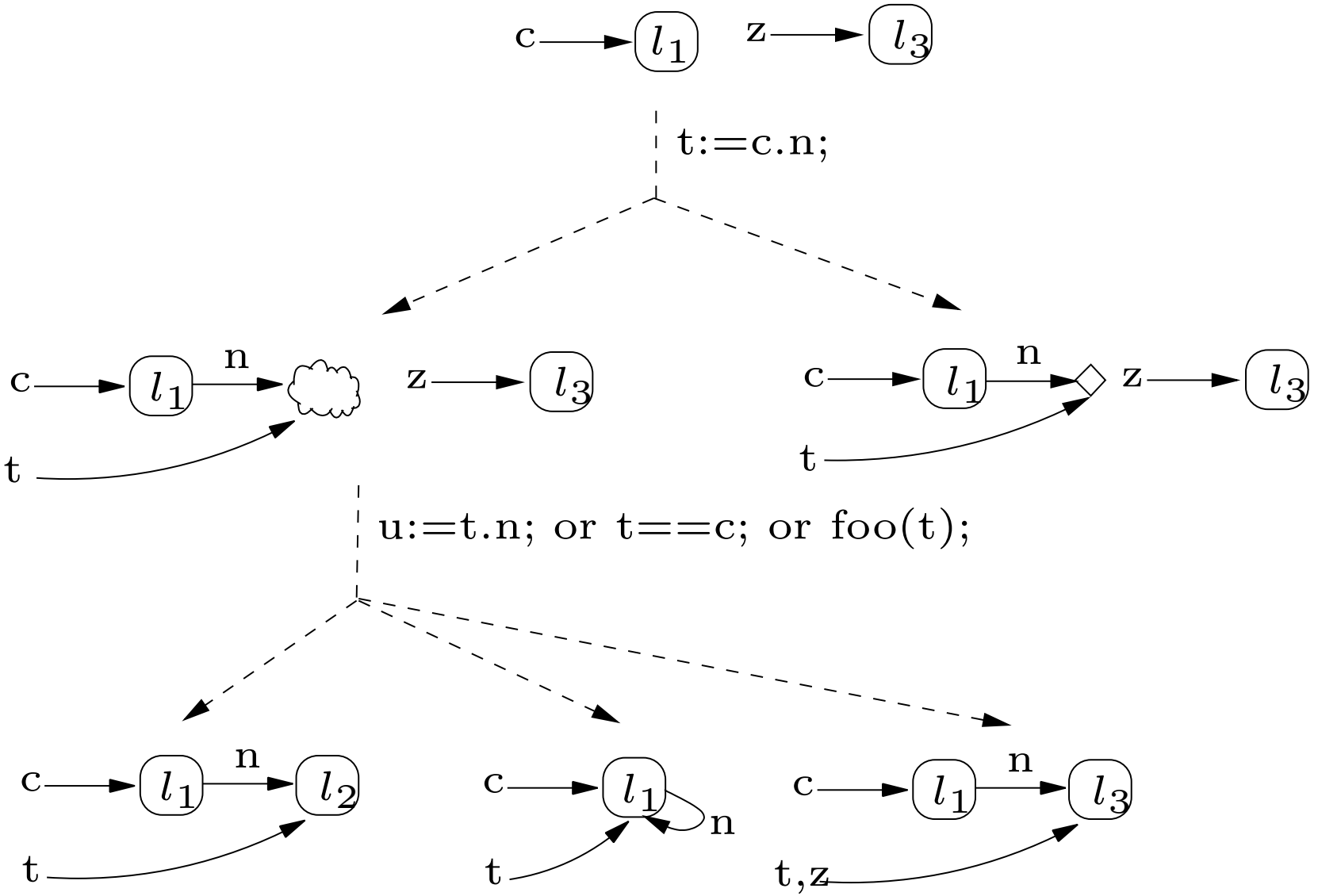
Introduction

Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks



Lazy vs. Lazier

Introduction

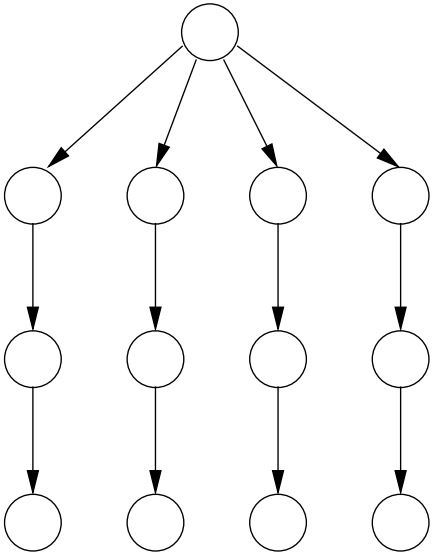
Reverse Execution

Program Validation by Symbolic Execution

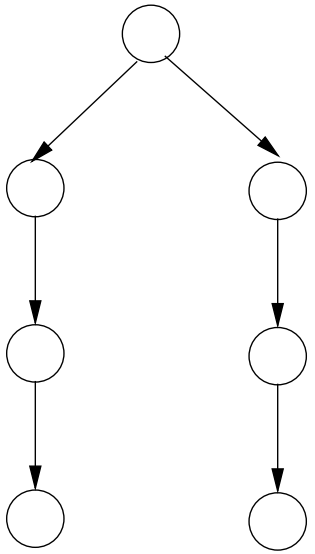
- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

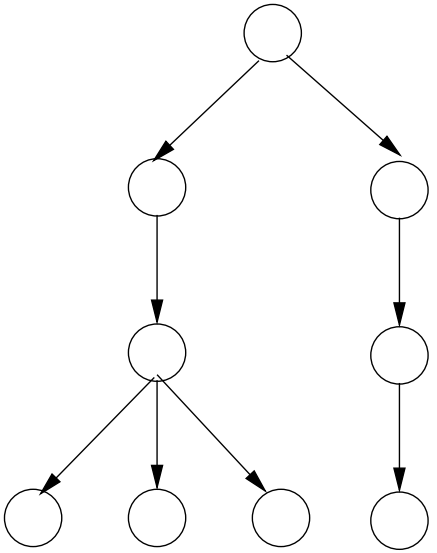
Concluding Remarks

Lazy Initialization



Lazier Initialization





Loop Example

Introduction

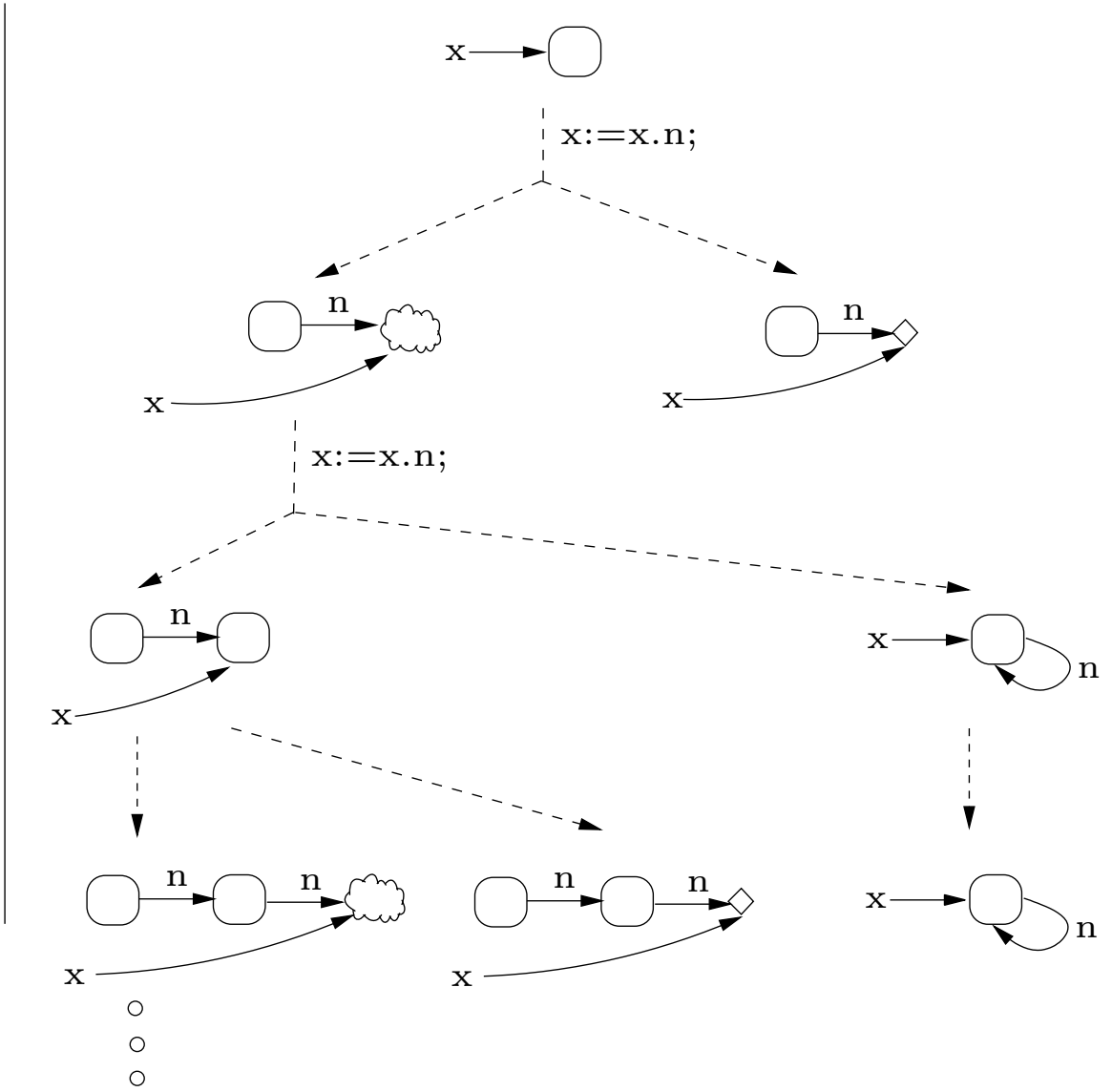
Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks


```
bag.add(x);
while (x≠nil) {
  x:=x.n;
  if (x≠nil) {
    if(bag.has(x))
      break;
    else
      bag.add(x);
  }
}
```



Solutions

Introduction

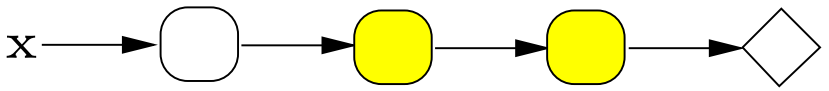
Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks

- Loop-invariant: heavy annotation
- Abstract interpretation:
  - automatic verification
  - false alarm, scalability issue
- Bounding:
  - no false alarm
  - suitable at the early phase of validation
  - easily applicable to any program with a small bound
  - BMC (transition), Alloy (domain of variables)



$(k=2)$

initial object fresh objects

Bounded Symbolic Execution

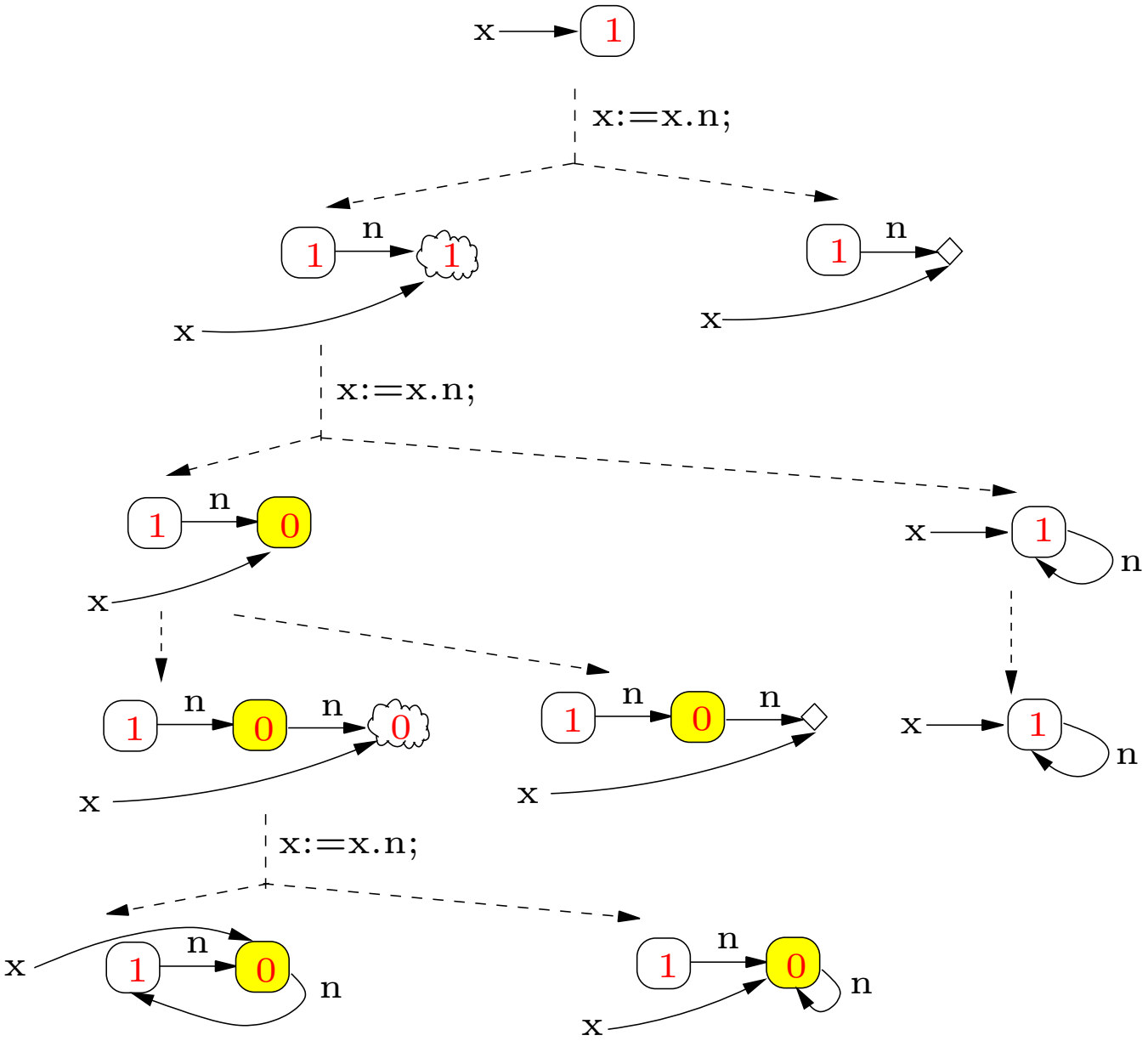
Introduction

Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks



Soundness

Introduction

Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks

If an error is found by our symbolic execution, this error can take place in a concrete execution.

- Assumptions
  1. The underlying theorem prover can tell all the infeasible path conditions.
  2. Specifications of methods are reliable.
- Proof: by a simulation

Completeness

Introduction

Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks

If an error can take place in a concrete execution, this error must be able to be found by our symbolic execution.

- Assumptions
  1. The length of every possible execution path is finite.
  2. A bound in use is sufficiently large.
  3. Specifications of methods are precise enough.
- Proof: by a simulation

Insertion Procedure of AVL Tree

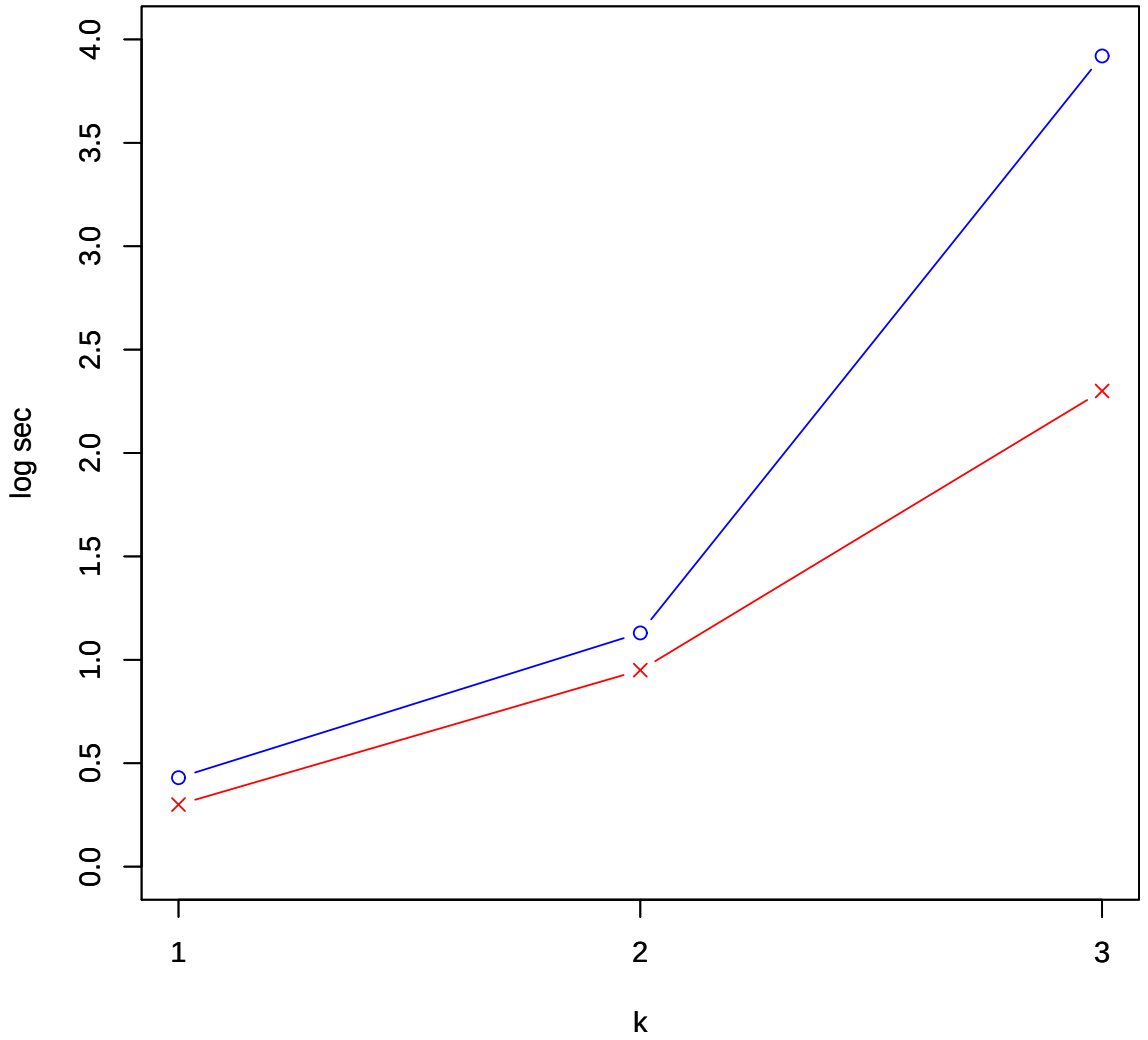
Introduction

Reverse Execution

Program Validation by Symbolic Execution

- Testing
- Symbolic Execution
- Heap
- Symbolic Execution on Heap
- Constraint Solver
- Lazy Initialization
- Lazier Initialization
- Lazy vs. Lazier
- Loop Example
- Solutions
- Bounded Symbolic Execution
- Soundness
- Completeness
- Insertion Procedure of AVL Tree

Concluding Remarks



Introduction

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks
- Recap
- Future Work
- Publications

Concluding Remarks

Recap

Introduction
Reverse Execution
Program Validation by Symbolic Execution
Concluding Remarks
● Recap
● Future Work
● Publications

- Introduction
  - Thesis topic (what): program validation
  - Thesis topic (how): symbolic and reverse execution
- Reverse Execution
  - Reverse execution helps to locate a bug
  - Introduced dynamic reverse code generation (DRCG)
  - DRCG scales to multi-threaded programs
  - DRCG is sound
- Symbolic Execution
  - Symbolic execution helps to check for a bug
  - Introduced bounded lazier initialization (BLI)
  - BLI is useful for finding heap-related bugs
  - BLI is sound in the sense of bug-finding

Future Work

- Reverse Execution
  - Empirical studies focusing on efficient data structures and algorithms
  - Application to other languages of different paradigms
  - Exploration of a programming language suitable for reverse-code generation
- Symbolic Execution
  - Maximum bound inference
  - Loop-invariant inference

Publications

Introduction

Reverse Execution

Program Validation by Symbolic Execution

Concluding Remarks

- Recap
- Future Work
- Publications

- Jooyong Lee.
  Reverse code generation for Java program model checking.
  In *Proceedings of Winter School on Modeling and Verifying Parallel Processes (MOVEP'04)*, pages 89-95, 2004.
- Jooyong Lee.
  Dynamic reverse code generation for backward execution.
  In *Proceedings of the Post-CAV Workshop on Verification and Debugging (V&D'06)*, pages 25-43, 2006.
- Jooyong Lee.
  A case for dynamic reverse-code generation.
  Submitted.
- Xianghua Deng, Jooyong Lee and Robby.
  Bogor/Kiasan: A k-bounded Symbolic Execution for Checking Strong Heap Properties of Open Systems.
  In *Proceedings of Conference on Automated Software Engineering (ASE'06)*, pages 157-166, 2006.