Introduction to

Algebraic Program Analysis

Zachary Kincaid[1]   Thomas Reps[2,3]

[1]Princeton University   [2]University of Wisconsin-Madison   [3]GrammaTech, Inc.

January 8, 2018

Program analysis

Design algorithms to answer questions about the dynamic behavior of software

- **Correctness**
  - *Is a program correct w.r.t. some specification?*
- **Security**
  - *Can a program over-read a buffer?*
- **Performance**
  - *How much memory will a program consume?*

Algebraic program analysis

A framework for designing program analyses based on **algebra**.

Semantic algebra = space of program properties + composition operators

- Sequencing: $\otimes$
- Choice: $\oplus$
- Iteration: $*$

Why algebraic program analysis?

- *Compositional*
  - Incremental analysis
  - Easy to parallelize
- Opens the door for new ways to compute loop invariants
  - Abstractions of loops are computed from abstractions of loop bodies

Why algebraic program analysis?

- *Compositional*
  - Incremental analysis
  - Easy to parallelize
- Opens the door for new ways to compute loop invariants
  - Abstractions of loops are computed from abstractions of loop bodies

Why *not* algebraic program analysis?

- Loss of contextual information

Outline

- Background
  - Iterative program analysis
  - Abstract interpretation
- Intraprocedural analysis
  - Overview
  - Path expressions
  - Compositional Recurrence Analysis
  - Proving soundness
- Interprocedural analysis
  - Functional approach
  - Newtonian program analysis
  - Newtonian program analysis via tensor product
  - Newtonian program analysis and Gauss-Jordan elimination

Outline

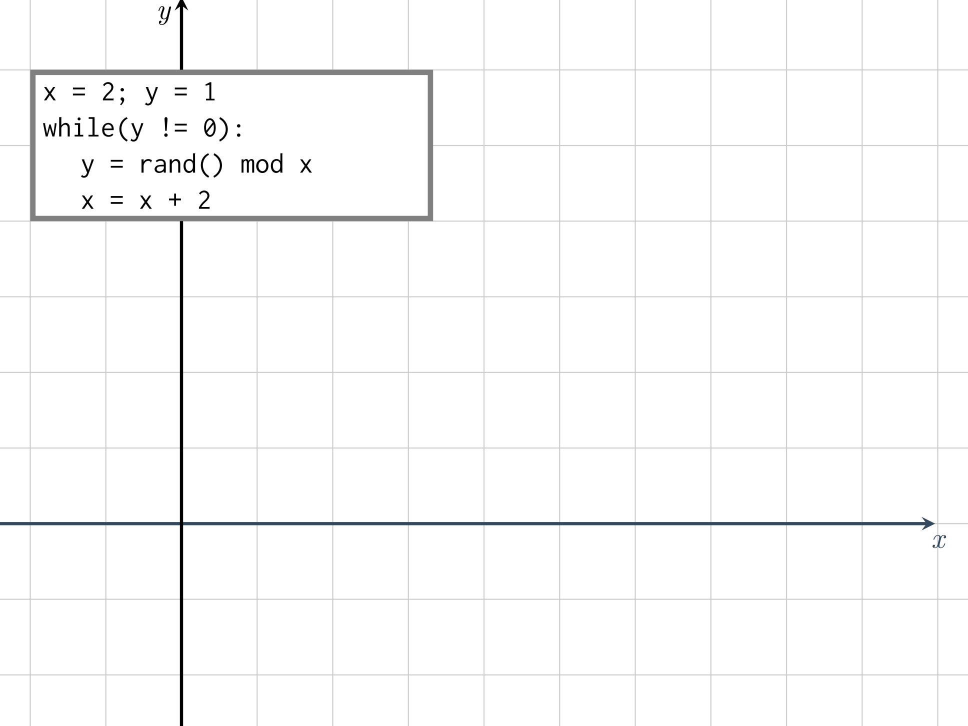
- Background
  - Iterative program analysis
  - Abstract interpretation
- Intraprocedural analysis
  - Overview
  - Path expressions
  - Compositional Recurrence Analysis
  - …oundness

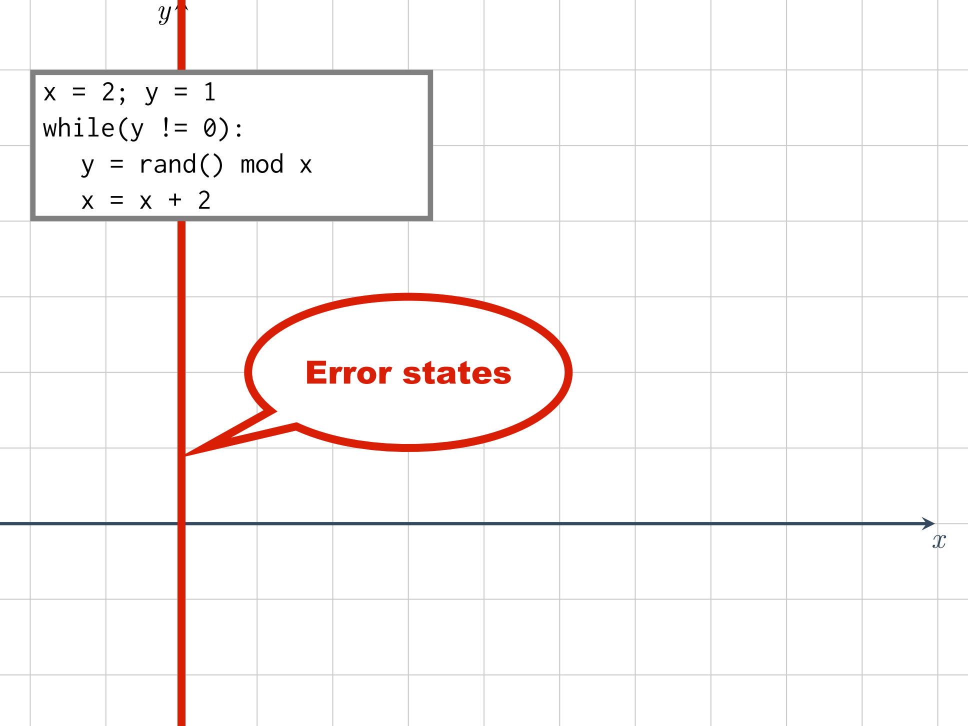
break

- Inte… …dural analysis
  - Functional approach
  - Newtonian program analysis
  - Newtonian program analysis via tensor product
  - Newtonian program analysis and Gauss-Jordan elimination

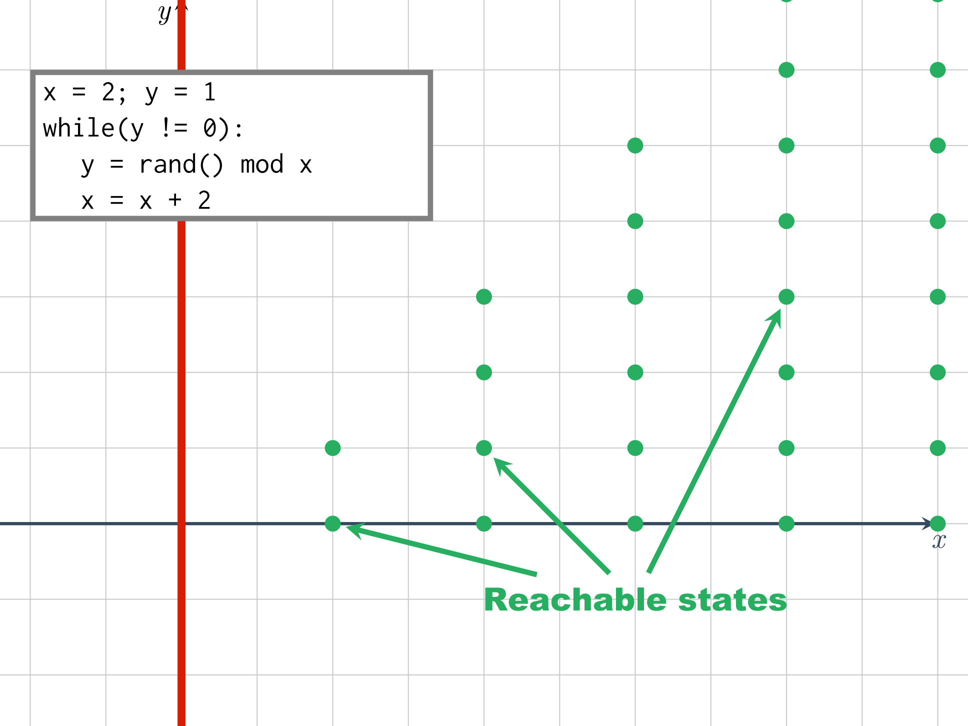
```
x = 2; y = 1
while(y != 0):
   y = rand() mod x
   x = x + 2
```

```
x = 2; y = 1
while(y != 0):
   y = rand() mod x
   x = x + 2
```

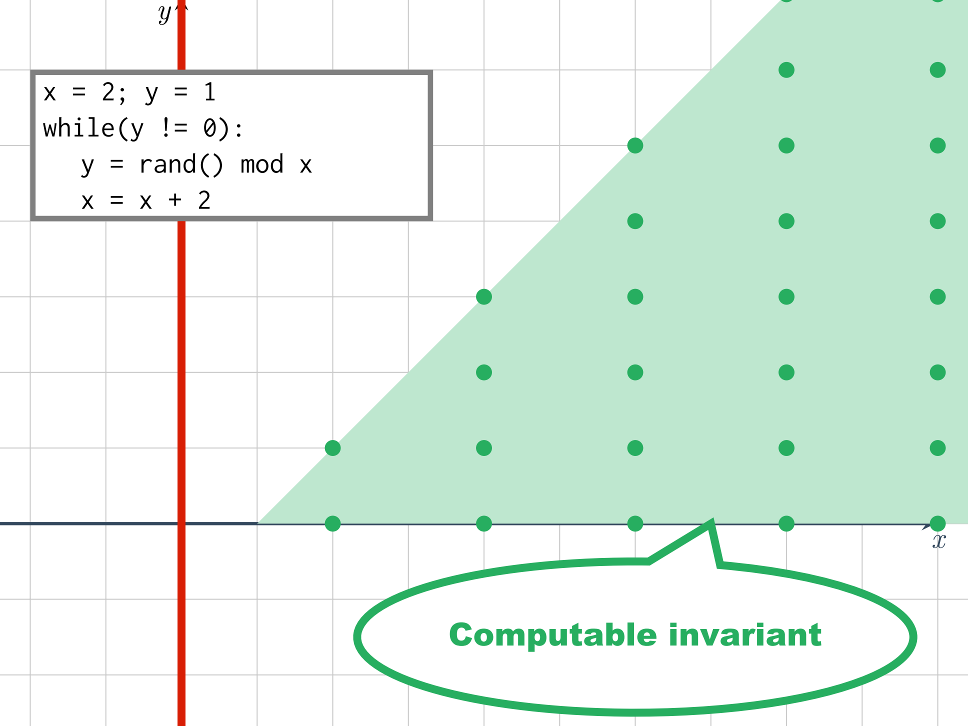


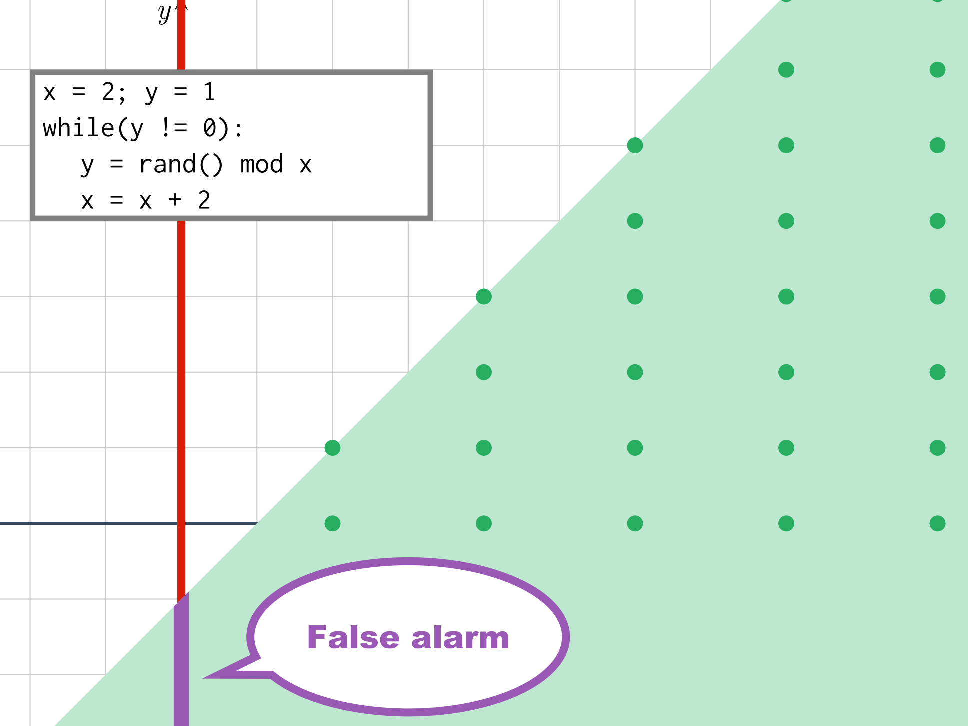


y
x = 2; y = 1
while(y != 0):
y = rand() mod x
x = x + 2
Error states
x



y
x = 2; y = 1
while(y != 0):
y = rand() mod x
x = x + 2
x
Reachable states

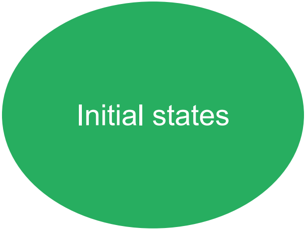

y
x = 2; y = 1
while(y != 0):
y = rand() mod x
x = x + 2
x
Computable invariant


y
x = 2; y = 1
while(y != 0):
y = rand() mod x
x = x + 2
False alarm

Iterative program analysis

Repeatedly evaluate the program under an abstract semantics until convergence upon a property that over-approximates all reachable states.

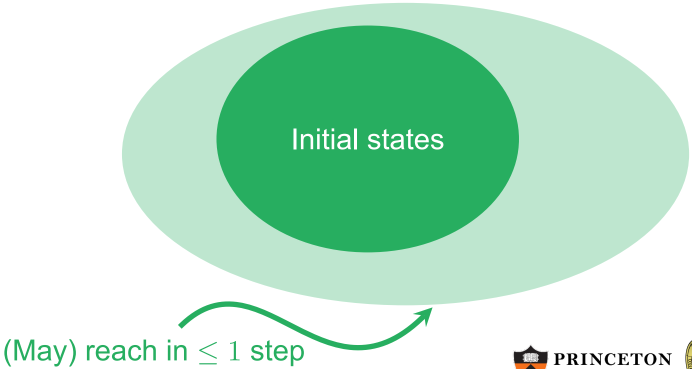
- SLAM, Astrée, ...



Iterative program analysis

Repeatedly evaluate the program under an abstract semantics until convergence upon a property that over-approximates all reachable states.

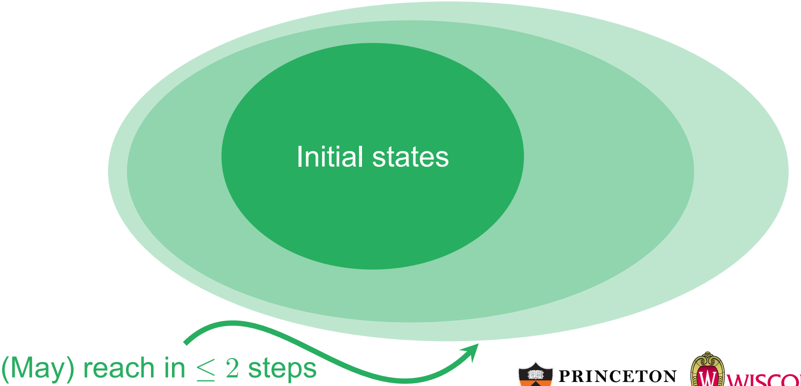
- SLAM, Astrée, ...



Iterative program analysis

Repeatedly evaluate the program under an abstract semantics until convergence upon a property that over-approximates all reachable states.

- SLAM, Astrée, ...

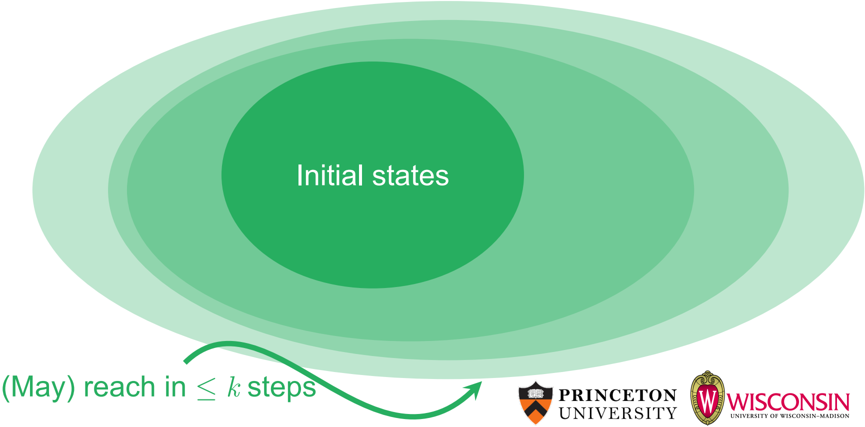


PRINCETON UNIVERSITY WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

Iterative program analysis

Repeatedly evaluate the program under an abstract semantics until convergence upon a property that over-approximates all reachable states.

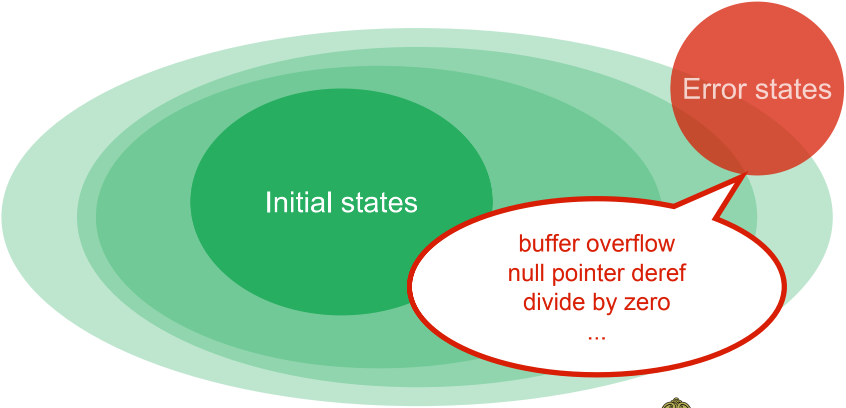
- SLAM, Astrée, ...



Iterative program analysis

Repeatedly evaluate the program under an abstract semantics until convergence upon a property that over-approximates all reachable states.

- SLAM, Astrée, ...



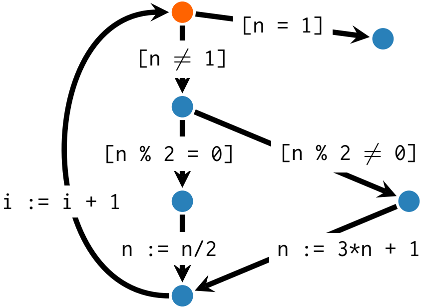
PRINCETON UNIVERSITY WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

Program model

Control flow graph $G = \langle \mathit{Loc}, \mathit{Edge}, \mathit{root} \rangle$

- *Loc*: set of control locations
- *Edge*: set of instruction-labeled edges
- *root*: root (entry location)

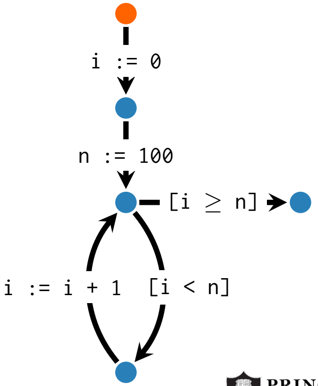
```
while(n ≠ 1){
  if(n % 2 == 0)
    n := n/2;
  else
    n := 3*n+1;
  i := i+1;
}
```



PRINCETON UNIVERSITY WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

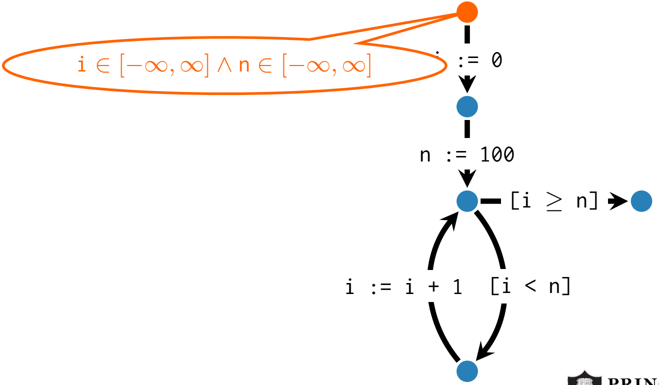
Example: interval analysis

$$\text{Property} \triangleq \underbrace{\text{Var} \to \overbrace{(\mathbb{Z}\cup\{-\infty\})\times(\mathbb{Z}\cup\{\infty\})}^{\text{Interval}}}_{\text{Interval store}}$$



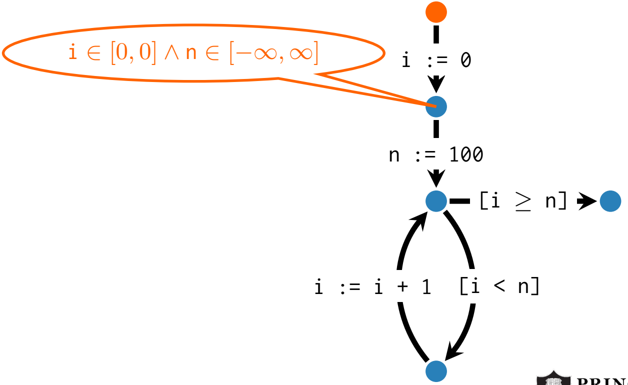
Example: interval analysis

$$\text{Property} \triangleq \underbrace{\text{Var} \to \overbrace{(\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{\infty\})}^{\text{Interval}}}_{\text{Interval store}}$$



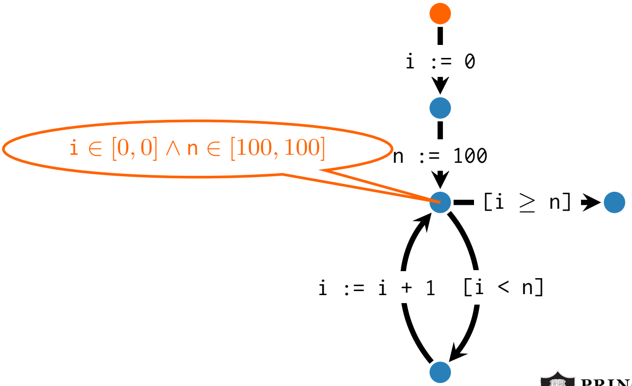
Example: interval analysis

$$\text{Property} \triangleq \underbrace{\text{Var} \to \overbrace{(\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{\infty\})}^{\text{Interval}}}_{\text{Interval store}}$$



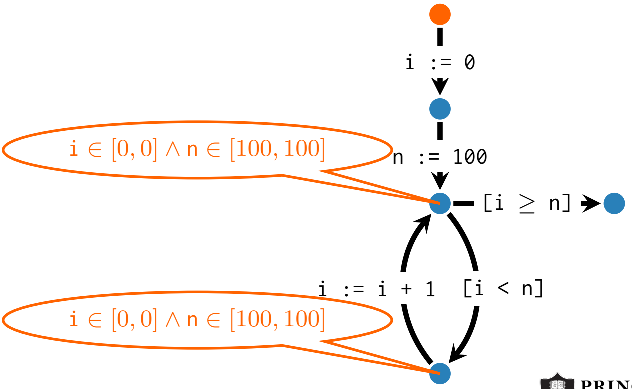
Example: interval analysis

$$\text{Property} \triangleq \underbrace{\text{Var} \to \overbrace{(\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{\infty\})}^{\text{Interval}}}_{\text{Interval store}}$$



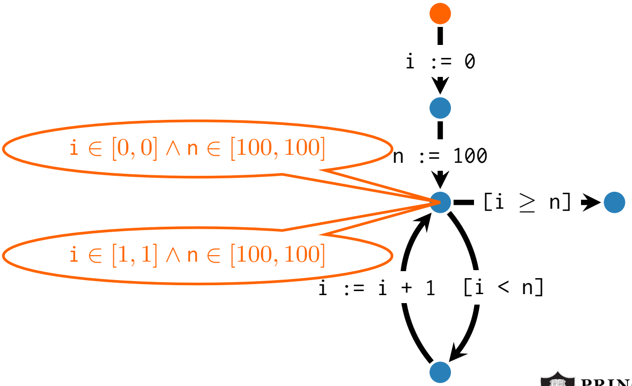
Example: interval analysis

$$\text{Property} \triangleq \underbrace{\text{Var} \rightarrow \overbrace{(\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{\infty\})}^{\text{Interval}}}_{\text{Interval store}}$$



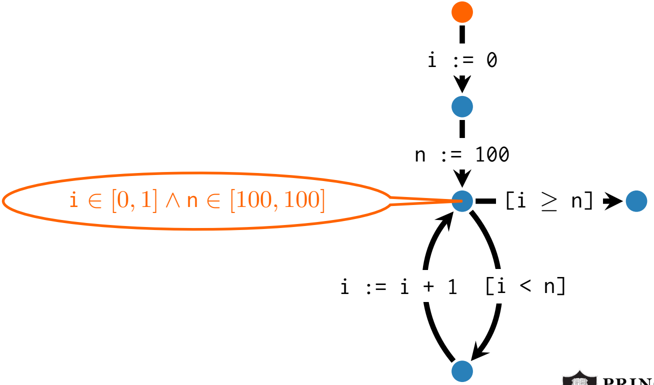
Example: interval analysis

$$\text{Property} \triangleq \underbrace{\text{Var} \to \overbrace{(\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{\infty\})}^{\text{Interval}}}_{\text{Interval store}}$$



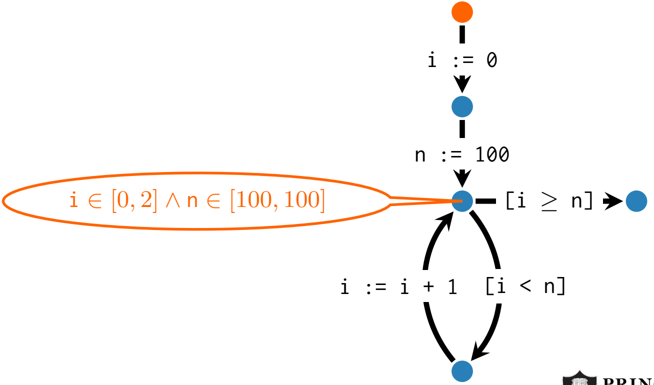
Example: interval analysis

$$\text{Property} \triangleq \underbrace{\text{Var} \to \overbrace{(\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{\infty\})}^{\text{Interval}}}_{\text{Interval store}}$$



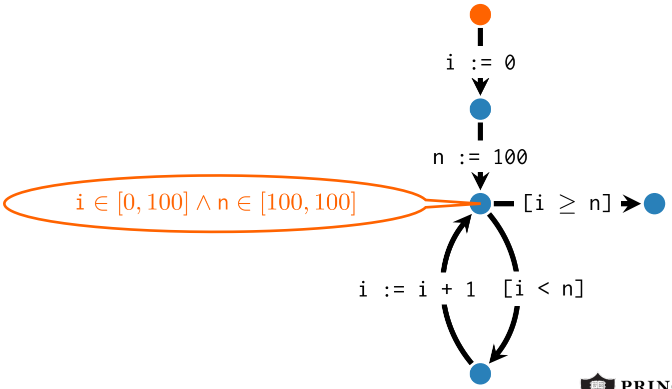
Example: interval analysis

$$\text{Property} \triangleq \underbrace{\text{Var} \to \overbrace{(\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{\infty\})}^{\text{Interval}}}_{\text{Interval store}}$$



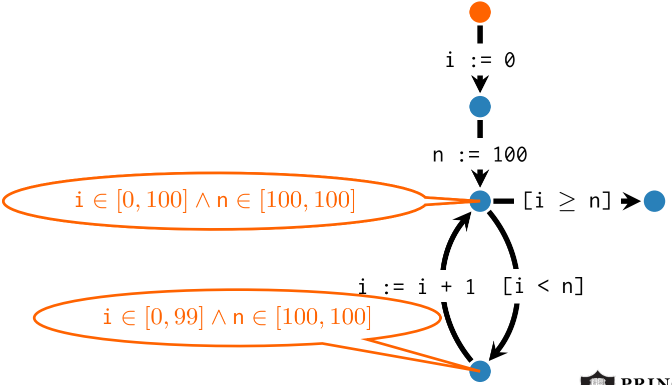
Example: interval analysis

$$\text{Property} \triangleq \underbrace{\text{Var} \to \overbrace{(\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{\infty\})}^{\text{Interval}}}_{\text{Interval store}}$$



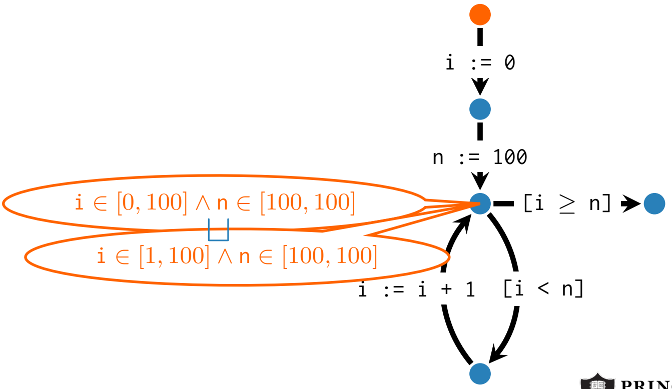
Example: interval analysis

$$\text{Property} \triangleq \underbrace{\text{Var} \rightarrow \overbrace{(\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{\infty\})}^{\text{Interval}}}_{\text{Interval store}}$$



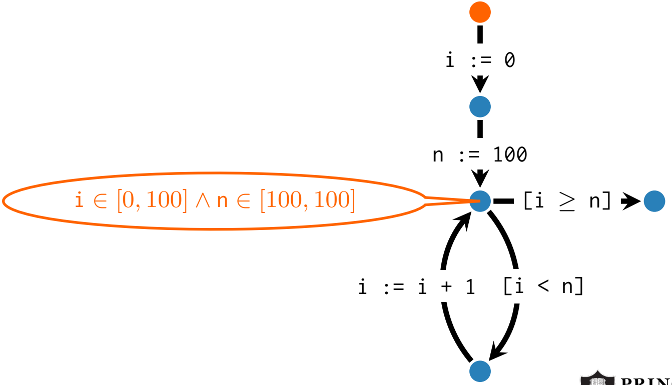
Example: interval analysis

$$\text{Property} \triangleq \underbrace{\text{Var} \to \overbrace{(\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{\infty\})}^{\text{Interval}}}_{\text{Interval store}}$$



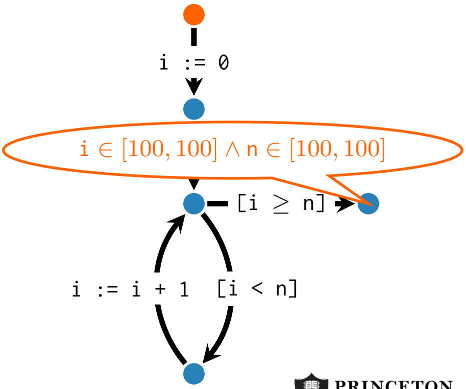
Example: interval analysis

$$\text{Property} \triangleq \underbrace{\text{Var} \to \overbrace{(\mathbb{Z}\cup\{-\infty\})\times(\mathbb{Z}\cup\{\infty\})}^{\text{Interval}}}_{\text{Interval store}}$$



Example: interval analysis

$$\text{Property} \triangleq \underbrace{\text{Var} \to \overbrace{(\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{\infty\})}^{\text{Interval}}}_{\text{Interval store}}$$



PRINCETON UNIVERSITY WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

Approximating a loop

$$\overbrace{p_1 \sqsubseteq p_2 \sqsubseteq p_2 \sqsubseteq \ldots}^{\text{Ascending sequence of properties}}$$

Approximate limit w/ a widening operator

$$\hat{p}_1 = p_1$$
$$\hat{p}_{i+1} = p_i \nabla p_{i+1}$$

PRINCETON UNIVERSITY WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

Designing an iterative analysis

1. **Define**:
   - Abstract domain $\mathcal{L} = \langle L, \sqsubseteq, \sqcup, \bot, \nabla \rangle$
     - L: space of program properties
     - $\sqsubseteq \subseteq L \times L$: approximation order
     - $\sqcup : L \times L \to L$: join (least upper bound) operator
     - $\nabla : L \times L \to L$ widening (extrapolation) operator
   - Property transformer: $\mathcal{L}[\![\cdot]\!]$: *Edge* $\to (L \to L)$ maps each command to a monotone function on L

Designing an iterative analysis

1. **Define**:
   - Abstract domain $\mathcal{L} = \langle L, \sqsubseteq, \sqcup, \bot, \nabla \rangle$
     - L: space of program properties
     - $\sqsubseteq \subseteq L \times L$: approximation order
     - $\sqcup : L \times L \rightarrow L$: join (least upper bound) operator
     - $\nabla : L \times L \rightarrow L$ widening (extrapolation) operator
   - Property transformer: $\mathcal{L}[\![\cdot]\!] : \mathit{Edge} \rightarrow (L \rightarrow L)$ maps each command to a monotone function on L
2. **Apply**: *chaotic iteration algorithm*
   - Computes a map $\mathit{inv} : \mathit{Loc} \rightarrow L$ that is closed under the abstract semantics:

$$\forall (u, v) \in \mathit{Edge}. \mathcal{L}[\![(u, v)]\!](\mathit{inv}(u)) \sqsubseteq \mathit{inv}(v)$$

Outline

Background

Iterative program analysis

Abstract interpretation

Intraprocedural analysis

Overview

Path expressions

Compositional Recurrence Analysis

Proving soundness

Interprocedural analysis

Functional approach

Newtonian program analysis

Newtonian program analysis via tensor product

Newtonian program analysis and Gauss-Jordan elimination

Proving soundness [Cousot & Cousot '77]

1. **Define**:
   - Concrete semantics
     - $\mathcal{C} \triangleq \langle 2^{\textit{Store}}, \subseteq, \cup, \emptyset, \cup \rangle$
     - $\mathcal{C}[\![e]\!](S) \triangleq \{s' : \exists s \in S. s \xrightarrow{e} s'\}$
   - *Concretization function* $\gamma : L \to 2^{\textit{Store}}$
     maps properties to set of stores that satisfy it

$$\gamma([x \mapsto [0,1]; y \mapsto [2,3]]) = \left\{ \begin{array}{ll} [x \mapsto 0; y \mapsto 2], & [x \mapsto 0; y \mapsto 3], \\ [x \mapsto 1; y \mapsto 2], & [x \mapsto 1; y \mapsto 3] \end{array} \right\}$$

Proving soundness [Cousot & Cousot '77]

1. **Define**:
   - Concrete semantics
     - $\mathcal{C} \triangleq \langle 2^{\textsf{Store}}, \subseteq, \cup, \emptyset, \cup \rangle$
     - $\mathcal{C}[\![e]\!](S) \triangleq \{s' : \exists s \in S. s \xrightarrow{e} s'\}$
   - *Concretization function* $\gamma : L \rightarrow 2^{\textsf{Store}}$
     maps properties to set of stores that satisfy it

$$\gamma([x \mapsto [0,1]; y \mapsto [2,3]]) = \left\{ \begin{array}{ll} [x \mapsto 0; y \mapsto 2], & [x \mapsto 0; y \mapsto 3], \\ [x \mapsto 1; y \mapsto 2], & [x \mapsto 1; y \mapsto 3] \end{array} \right\}$$

2. **Prove** transformer simulation: for all properties p, edges e:

$$\mathcal{C}[\![e]\!](\gamma(p)) \subseteq \gamma(\mathcal{L}[\![e]\!](p))$$

Proving soundness [Cousot & Cousot '77]

1. **Define**:
   - Concrete semantics
     - $\mathcal{C} \triangleq \langle 2^{\textit{Store}}, \subseteq, \cup, \emptyset, \cup \rangle$
     - $\mathcal{C}[\![e]\!](S) \triangleq \{s' : \exists s \in S . s \xrightarrow{e} s'\}$
   - *Concretization function* $\gamma : L \to 2^{\textit{Store}}$
     maps properties to set of stores that satisfy it

$$\gamma([x \mapsto [0,1]; y \mapsto [2,3]]) = \left\{ \begin{array}{ll} [x \mapsto 0; y \mapsto 2], & [x \mapsto 0; y \mapsto 3], \\ [x \mapsto 1; y \mapsto 2], & [x \mapsto 1; y \mapsto 3] \end{array} \right\}$$

2. **Prove** transformer simulation: for all properties p, edges e:

$$\mathcal{C}[\![e]\!](\gamma(p)) \subseteq \gamma(\mathcal{L}[\![e]\!](p))$$

3. **Apply** *fixpoint transfer*: Chaotic iteration algorithm computes a map *inv* : *Loc* $\to L$ such that

$$\text{Stores reachable at } v \subseteq \gamma(\textit{inv}(v))$$