# Semantic Foundations for the Analysis of Program Revisions

**Dakota Bryan**
Project Advised by Evan Chang and Matthew Hammer
Program Analysis: Theory and Practice, Fall 2025

# Motivation

Developers have informal notions of the semantics of a program revision. We aim to formalize these semantics and provide an abstract interpretation framework to reason about them.

# Understanding the Problem

How can we describe the semantic effect of a program revision? (Concrete / collecting semantics)

How can we reason about these effects? (Abstract semantics)

# Contributions and Outline

Part 1: We describe the semantic effect of a program revision by providing concrete/collecting semantics

Part 2: We give the requirements for reasoning about these effects by providing parameterized abstract semantics

# Semantics of Program Revisions

How can we describe the semantic effect of a program revision?

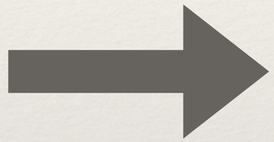
How can we reason about these effects?

Idea: Reachable states of the “old” program are either removed or related.
Reachable states of the “new” program are either added or related.

Remaining Question: When is an “old” /“new” program state removed/ added?

# Example

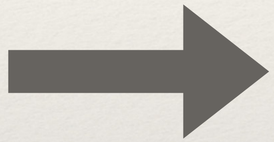
```
x := nonDet()
if (x > 0)
  sgn := 1
else
  sgn := -1
```



```
x := nonDet()
if (x > 0)
  sgn := 1
else
  sgn := -1
if (x = 0)
  sgn := 0
```

# Example: Entire State

```
x := nonDet()
if (x > 0)
  sgn := 1
else
  sgn := -1
```



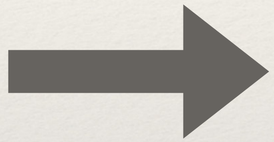
```
x := nonDet()
if (x > 0)
  sgn := 1
else
  sgn := -1
if (x = 0)
  sgn := 0
```

$x \mapsto 0, \mathrm{sgn} \mapsto -1$

$x \mapsto 0, \mathrm{sgn} \mapsto 0$

# Example: State Projection (sgn)

```
x := nonDet()
if (x > 0)
  sgn := 1
else
  sgn := -1
```



```
x := nonDet()
if (x > 0)
  sgn := 1
else
  sgn := -1
if (x = 0)
  sgn := 0
```

$x \mapsto 0, \mathrm{sgn} \mapsto 0$

# Solution: State Correspondence

- A "new" state, $\sigma_2$, is added if for any reachable "old" state, $\sigma_1$, $\sigma_2 \nsim \sigma_1$
- $\sim$ is a user defined relation between "old" states and "new" states

$$\begin{aligned} \text{state correspondence} \quad & s ::= p_1, p_2 \\ \text{state projection} \quad & p ::= \circ \mid p, x \\ \text{program variables} \quad & x \end{aligned}$$

$s = (\text{sgn}), (\text{sgn})$

$$\sigma_1 \sim \sigma_2 \iff \sigma_1(\text{sgn}) = \sigma_2(\text{sgn})$$

# Solution: State Correspondence

How can the effect of a program revision be described? (Concrete/collecting semantics)

How can we reason about these effects?

Solution: Reachable states of the "old" and "new" programs are characterized as removed, related, or added (concretely, sets) via a state correspondence

# Abstraction

How can the effect of a program revision be described?

How can we reason about these effects?

Idea: Over-approximate removed, related, and added sets of states

# Intricacies of Abstraction: Partitioning

Recall: removed = $\{\sigma_1 \in \Sigma_1 \mid \forall \sigma_2 \in \Sigma_2 \,.\, \sigma_1 \nsim \sigma_2\}$

Consider: $\hat{\nsim}$ and $\hat{\sigma}$ where $\gamma(\hat{\sigma}) = \Sigma$

Challenge: If $\hat{\sigma}$ merges all abstract states into one element, then either all states are removed or none of them are

# Intricacies of Abstraction: Partitioning

One Solution: $\hat{\sigma}$ is split into elements representing subsets of $\gamma(\hat{\sigma})$

$$\hat{\text{removed}} = \{\hat{\sigma}_i \in \hat{\sigma}_1 \mid \forall \hat{\sigma}_j \in \hat{\sigma}_2 \,.\, \hat{\sigma}_i \mathrel{\hat{\not\approx}} \hat{\sigma}_j\}$$

# Intricacies of Abstraction: Under-Approximation

$$\hat{\text{removed}} = \{\hat{\sigma}_i \in \hat{\sigma}_1 \mid \forall \hat{\sigma}_j \in \hat{\sigma}_2 \,.\, \hat{\sigma}_i \,\hat{\not\approx}\, \hat{\sigma}_j\}$$

Problem: If $\Sigma_2 \subseteq \gamma(\hat{\sigma}_2)$, then there are "more opportunities" for $\hat{\sigma}_i \,\hat{\not\approx}\, \hat{\sigma}_j$ for some $\hat{\sigma}_i$ resulting in the possibility for $\hat{\text{removed}}$ to be an under-approximation

# Intricacies of Abstraction: Under-Approximation

$$\hat{\text{removed}} = \{\hat{\sigma}_i \in \hat{\sigma}_1 \mid \forall \hat{\sigma}_j \in \hat{\sigma}_2 \,.\, \hat{\sigma}_i \not\hat{\approx} \hat{\sigma}_j\}$$

Solution: We require that the abstraction of reachable stats for the "new" program is an under-approximation ($\gamma(\hat{\sigma}_2) \subseteq \Sigma_2$)

# Parameterized Abstract Semantics

How can the effect of a program revision be described? (Concrete/collecting semantics)

How can we reason about these effects? (Parameterized abstract semantics)

We require:

- An over- and under-approximating abstraction of reachable states,
- In a finitely partitioned abstract domain,
- With the state correspondence operation lifted to this domain

Thank you, Questions?