ASTraMut:
Learning Java Mutation Operators from Real-World Bug-Fix Patterns

CS453 Project Team 5
20266008 Doyeol Oh 20266118 Hyunji Park 20266161 Junseo Jang 20265304 Dongjae Lee

Why *PIT*'s Mutants Fall Short

- **Mutation testing**: tweak the code in small ways (mutants) → see how many your tests catch
- ***PIT*** [Coles, 2024] — common tool for mutation-test Java

But these mutants from *PIT* are...

- Not realistic [Just et al., FSE'14]
- Trivial at industry scale [Beller et al., FSE'21]
- Misleading metric
- Limited Scope

Name	Transformation	Example
Cond. Bound.	Replaces one relational operator instance with another one (single replacement).	$< \rightsquigarrow \leq$
Negate Cond.	Negates one relational operator (single negation).	$== \rightsquigarrow !=$
Remove Cond.	Replaces a cond. branch with true or false.	`if (...)` ⇝ `if (true)`
Math	Replaces a numerical op. by another one (single replacement).	$+ \rightsquigarrow -$

Learn Operators from Real Bug Fixes

- Pattern Learning
  - Learns fix patterns **Buggy ↦ Patched** from bug-fix commits
- Generate Mutation Operator
  - Flip them — use **Patched ↦ Buggy** as mutation rules
  - Take clean code, inject a shape that actually showed up as a bug
  - Reported to be way more actionable than the textbook ones in industrial evaluation

Getafix: Learning to Fix Bugs Automatically

Johannes Bader
Facebook
jobader@fb.com

Andrew Scott
Facebook
andrewscott@fb.com

Michael Pradel
Facebook
michael@binaervarianz.de

Satish Chandra
Facebook
satch@fb.com

The Care and Feeding of Wild-Caught Mutants

David Bingham Brown
Univ. of Wisconsin–Madison, USA
bingham@cs.wisc.edu

Michael Vaughn
Univ. of Wisconsin–Madison, USA
mvaughn@cs.wisc.edu

Ben Liblit
Univ. of Wisconsin–Madison, USA
liblit@cs.wisc.edu

Thomas Reps
Univ. of Wisconsin–Madison and GrammaTech, Inc., USA
reps@cs.wisc.edu

ASTRAMUT's Pattern Learning Algorithm

- **Extract edits**
  - Pull out edits from each commit at multiple granularities
- **Anti-unify & Cluster**
  - Generalize pairs of edits → introduces pattern variable where things differ
- **Elimination & Rank**
  - Hierarchical agglomerative clustering → patterns generalize bottom-up

→ ASTRAMUT = a from-scratch Java implementation of this idea, targeted at mutation testing

Inspired by

Getafix: Learning to Fix Bugs Automatically

Johannes Bader
Facebook
jobader@fb.com

Andrew Scott
Facebook
andrewscott@fb.com

Michael Pradel
Facebook
michael@binaervarianz.de

Satish Chandra
Facebook
satch@fb.com

Pipeline Overview

- **Edit Preprocessor**
  - Slice each diff into candidates at every AST level.
  - Canonicalize early — names, types, magic numbers → pattern variable / sentinel.
- **Anti-Unification + Hierarchical Clustering**
  - Keep what's identical, replace what differs with shared pattern variable.
  - Merge bottom-up into more general patterns.
- **Mutation Operator Selection and Ranking**
  - Drop the unsafe, the trivial, the noisy.
  - Rank by Score = support × specificity

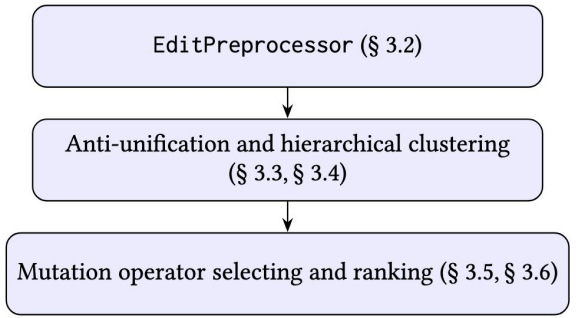

Figure 1: The ASTRAMUT pattern-learning pipeline.

Generalizing Patterns

Canonicalization: Converts identifier to pattern variable

Source: `int result = obj.getItems();`

Identifier	Decision
`int`	literal
`result`	`?h0`
`obj`	`?h1`
`getItems`	`?h2`

Pattern: `int ?h0 = ?h1.?h2();`

- Pattern matches any receiver/method of this shape, not just `obj.getItems()` literally.
- Lets API renames like `length` → `size` generalize across variable names

Generalizing Patterns

Anti-Unification: Keeps what's shared, pattern variables what differs

AntiUnify(p_1, p_2) walks both AST trees. Four cases:

Case	Condition	Action
A	both subtrees identical	keep as-is
B	both marked unmodified	collapse to a single hole
C	share type / label / arity	recurse on children
D	otherwise	mint a new hole keyed by (t_1, t_2)

- Pattern variable allocator shared between Buggy-side and Patched-side
  → same pattern variables on both sides.

Generalizing Patterns

Anti-Unification Example

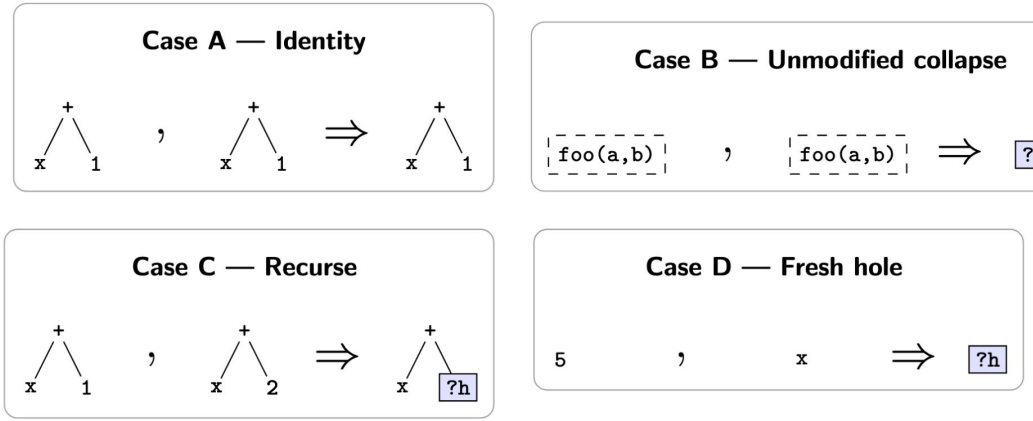

Figure 1: Anti-unification cases A–D, illustrated on small AST pairs. Blue nodes are freshly allocated pattern variables; dashed boxes mark unmodified regions.

Mutation Operator Selection and Ranking

Elimination Rule

Rule	Drops when...	Why
No-op	before = after	trivial identity — no mutant.
Unbounded pvar	pvars(before) $\not\subseteq$ pvars(after)	reverse-apply would need to invent code.
Oversized ground singleton	support = 1, pvars = 0, nodes > threshold	one-shot project-specific edit.
Identifier-only root	both sides bare identifiers, support = 1	raw name swap — no typed context.
Destructive root pvar	root `?h` whose id absent on opposite side	"replace the whole tree."

Ranking

$$\text{specificity}(p) = 1 - \frac{|\text{pvars}(p)|}{|\text{nodes}(p)|}, \qquad \text{score}(p) = \text{support}(p) \cdot \text{specificity}(p).$$

Actual Pattern Learning

Table 1: Training datasets used for learning mutation operators.

Dataset	Fix pairs	Patch scope
ManySStuBs4J [11]	63 923	Statement
Bugs2Fix [16]	46 680	Method

Table 3: Final mutation-operator-set sizes after all five elimination rules and cross-run deduplication.

Training data	Cohort size	Patterns
ManySStuBs4J	63 923	257
Bugs2Fix	46 680	234
Merged	110 603	**436**

- Trained on **110,603** real bug-fix pairs — *ManySStuBs4J* (one-line fixes) + *Bugs2Fix* (full methods).
- Got **436** mutation operators — these include *PIT*'s simple rewrites (swap booleans, flip < to <=, etc..)
  - and bigger ones *PIT* just doesn't have (rename APIs, tweak predicates, change block shapes).

Coverage between learned operators and *PIT*

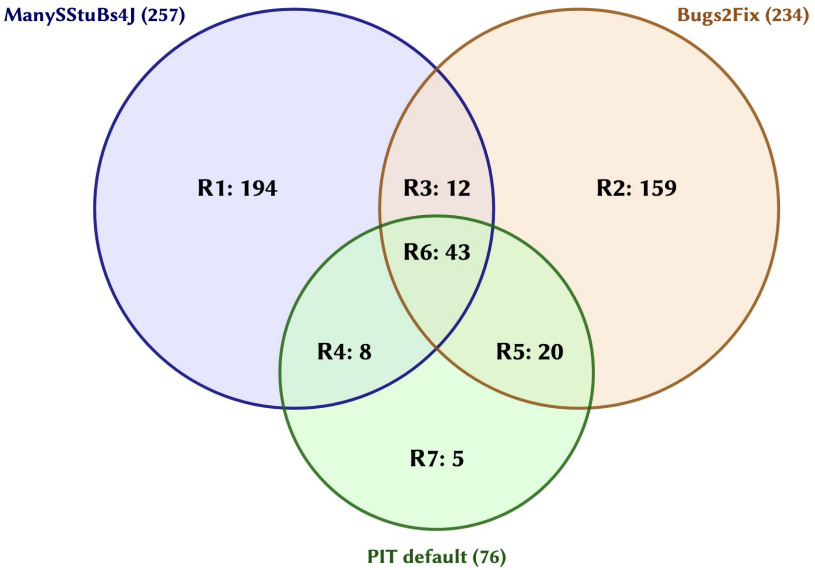

Figure 5: The Overlap between learned pattern sets and PIT mutation operators

	Region (set membership)	#	Top-3 patterns

R1	ManySStuBs4J *only*	194	`__MAGIC__` → `0`
			`StringBuffer` → `StringBuilder`
			`MethodInvocation` → `PrefixExpression`

R2	Bugs2Fix *only*	159	`private` → `public`
			`public` → `private`
			`protected` → `public`

R3	ManySStuBs4J ∩ Bugs2Fix only	12	`1` → `__MAGIC__`
			`0` → `__MAGIC__`
			`__MAGIC__` → `1`

R4	ManySStuBs4J ∩ PIT only	8	`>>` → `>>>`
			`<=` → `!=`
			`+` → `|`

R5	Bugs2Fix ∩ PIT only	20	`==` → `<`
			`++` → `--`
			`/` → `+`

R6	All three (ManySStuBs4J ∩ Bugs2Fix ∩ PIT)	43	`1` → `0`
			`true` → `false`
			`false` → `true`

R7	PIT default *only*	5	`INVERT_NEGS`
			`VOID_METHOD_CALLS`
			`EMPTY_RETURNS`
			`NULL_RETURNS`
			`RETURN_VALS`

Experimental Setup 💻

8 mutation-operator configuration compared on *Defects4J*.

Table 3: Variants of mutation operator set

Variant	# Operators	Coverage
ASTRAMUT (Merged, Top 100)	100	top-100 of 436 by score
ASTRAMUT (Merged, Full)	436	all learned mutation operators from both datasets
ASTRAMUT (ManySStuBs4J, Top 100)	100	top-100 of 257 by score
ASTRAMUT (ManySStuBs4J, Full)	257	all learned from ManySStuBs4J
ASTRAMUT (Bugs2Fix, Top 100)	100	top-100 of 234 by score
ASTRAMUT (Bugs2Fix, Full)	234	all learned from Bugs2Fix
PIT (Default)	11	conditionals, arithmetic, return values
PIT (Full)	26	default + constructor/method call, switch, `BigDecimal`

Experimental Setup 💻

Procedure

- ❏ 1. Identify methods modified by the developer patch
- ❏ 2. Run *Defects4J*-provided relevant tests on those methods only
- ❏ 3. $\text{mutation score} = \frac{|\text{killed mutants}|}{|\text{generated mutants}|}$ (1 if no mutants generated)
- ❏ 4. Report arithmetic mean across all evaluated bugs

Research Questions

- ❏ RQ1: Does **ASTRAMUT** Generate More Difficult Mutants to Kill?
- ❏ RQ2: Can **ASTRAMUT** Learn Mutation Operators Not in *PIT*?

Experiment Results

Table 2: Mutation-testing results on Defects4J by mutation-operator variant.

Variant	Avg. generated mutants	Avg. killed mutants	Avg. mutation score	Δ vs. PIT (Full)
ASTraMut (Merged, Top 100)	115.40	70.85	**0.6926**	−1.94 %
ASTraMut (Merged, Full)	158.64	79.20	0.6933	−1.84 %
ASTraMut (ManySStuBs4J, Top 100)	86.55	50.26	0.6995	−0.96 %
ASTraMut (ManySStuBs4J, Full)	90.92	52.98	0.6987	−1.08 %
ASTraMut (Bugs2Fix, Top 100)	106.58	66.65	0.6941	−1.73 %
ASTraMut (Bugs2Fix, Full)	127.76	81.29	0.7061	−0.03 %
PIT (Default)	41.10	25.41	0.7399	+4.76 %
PIT (Full)	135.21	84.07	0.7063	0.00 %

6 Patterns *PIT* Can't Produce ✨

Table 4: Representative learned patterns from `patterns-merged.json` that fall outside PIT's built-in operator catalogue.

	Pattern (*Patched* ↦ *Buggy*)	Why this is outside PIT
E1	`StringBuilder` ↦ `StringBuffer`	Library-specific class names are not part of PIT's fixed bytecode mutator set.
E2	`IllegalArgumentException` ↦ `IllegalStateException`	Exception classes are semantic identifiers, not relational, arithmetic, return, constant, or call-removal targets.
E3	`Long` ↦ `Integer`	PIT can change primitive return values, but does not rewrite boxed type names in source-level type positions.
E4	`?h0` ↦ `!?h0`	This wraps a whole predicate; `NEGATE_CONDITIONALS` rewrites conditional bytecode rather than learning arbitrary predicate templates.
E5	`?h0 || ?h1` ↦ `?h0`	The learned rewrite weakens a guard by dropping one disjunct, not by replacing a comparator or forcing the entire conditional.
E6	`Block[?h]` ↦ `<empty Block>`	The mutation changes method-body shape; PIT removes selected calls or assignments but does not learn block-level rewrite templates.

- These patterns catch the bug shapes *PIT*'s fixed mutators don't cover.
- *PIT* was not designed to reach:
  - API renames, whole-predicate negation, disjunction broadening, method-body restructuring.

Answers to Research Questions

- RQ1: Does ASTRAMUT Generate More Difficult Mutants to Kill?
  - Yes. Mutation score **−1.94 %** vs. best *PIT*, on the same tests.
  - And does it with fewer mutants too: **115 vs. *PIT*'s 135**.

- RQ2: Can ASTRAMUT Learn Mutation Operators Not in *PIT*?
  - Yes. Only **71 / 436 (12.6 %)** overlap with *PIT*.
  - The rest are shapes *PIT*'s fixed catalogue can't express
    - identifier swaps, predicate templates, block-shape rewrites, etc..

Contributions of ASTraMut

- **End-to-end automation, from scratch in Java**
  - GumTree diffs → Anti-Unification → Clustering → Mutation engine, all wired up in one pipeline
  - No manual operator design — just point it at a bug-fix corpus

- **Operators learned from real-world bug-fix datasets**
  - *ManySStuBs4J* (63,923 single-statement fixes) + *Bugs2Fix* (46,680 method-level pairs)
  - → 436 mutation operators drawn from how developers actually fix code

- **Lower score = harder-to-kill mutants**
  - Mutation score **−1.94 %** vs. best *PIT*, on the same tests, using fewer operators.

Summary

- **Problem**
  - *PIT* misses entire fault classes
- **Method**
  - 110,603 Fixes → Anti-Unification → Clustering → Filters → 436 Operators
- **Finding**
  - *Merged Top-100* vs *PIT* (**-1.94% – -6.39%**), Lower mutation score
  - 6 pattern categories outside *PIT*'s reach

https://github.com/duncan020313/cs453-team5-learning-mutation-operator

Any Questions?