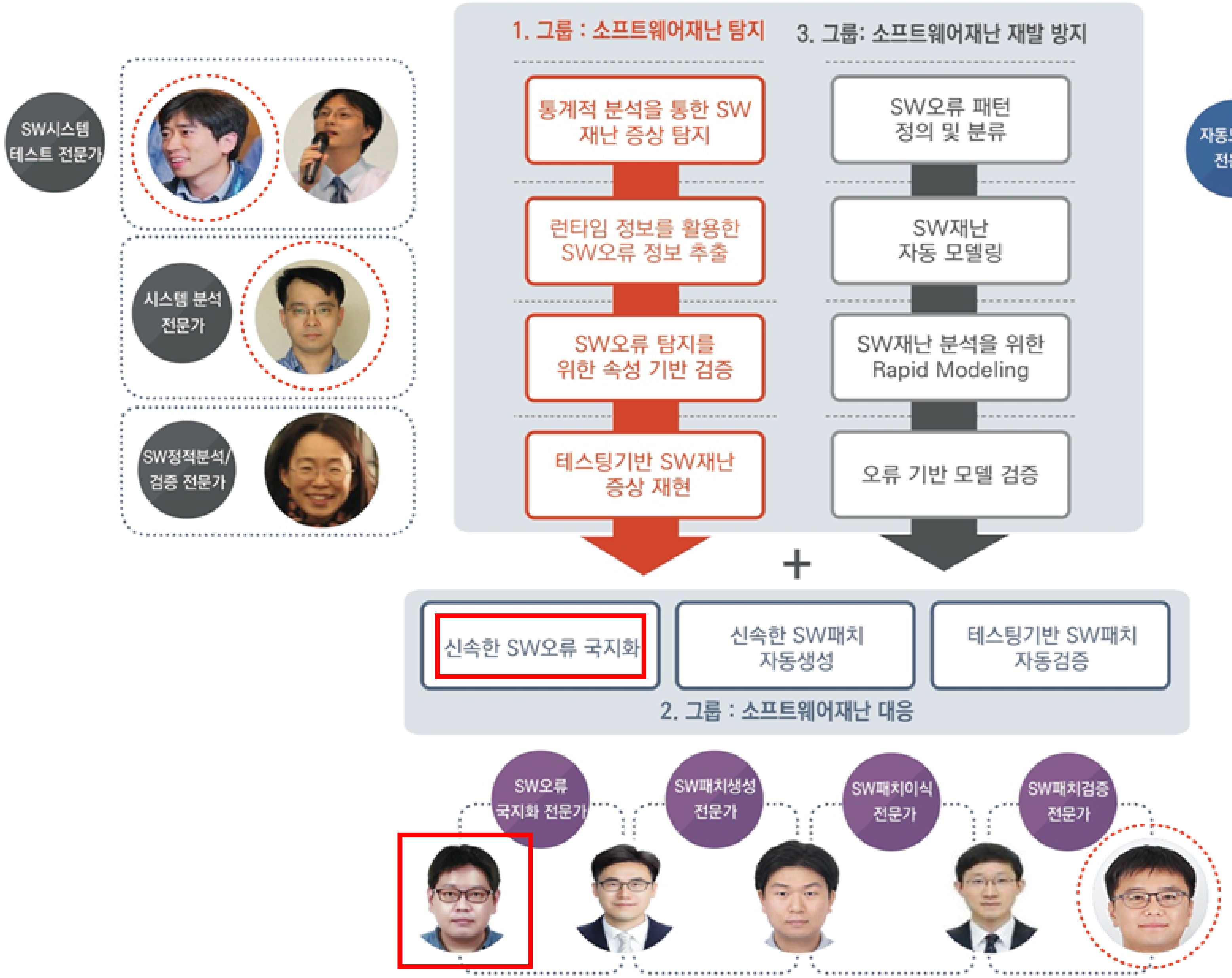
Learning-based Mutant Reduction for Debugging

Yunho Kim

Dept. of Computer Science

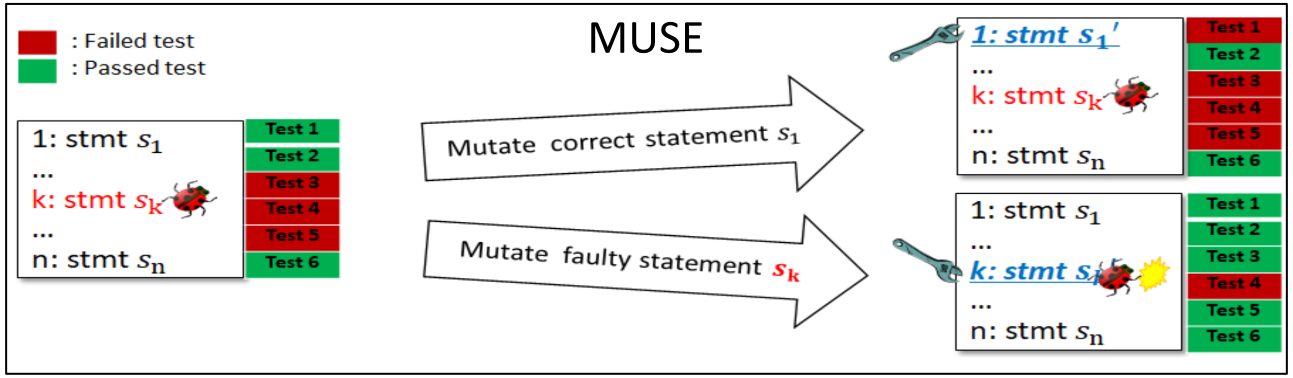
Hanyang University

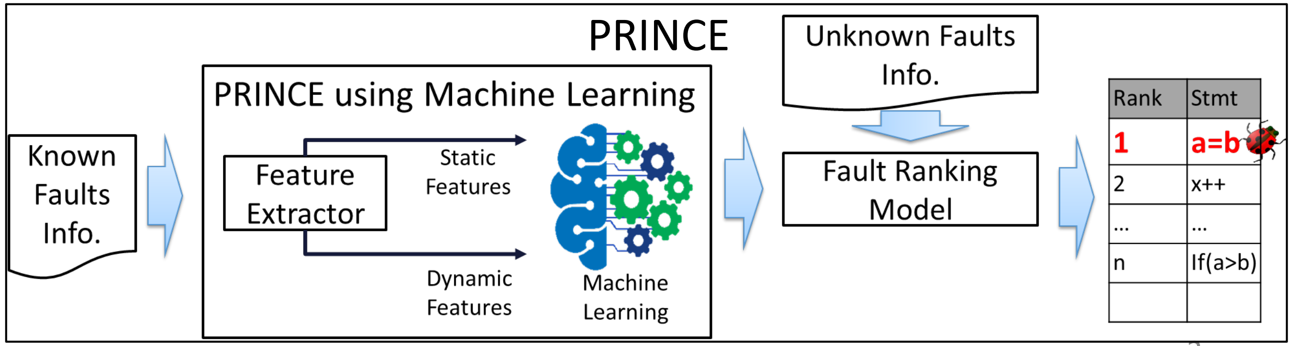
My role in STAAR



Mutation-based Fault Localization (MBFL)

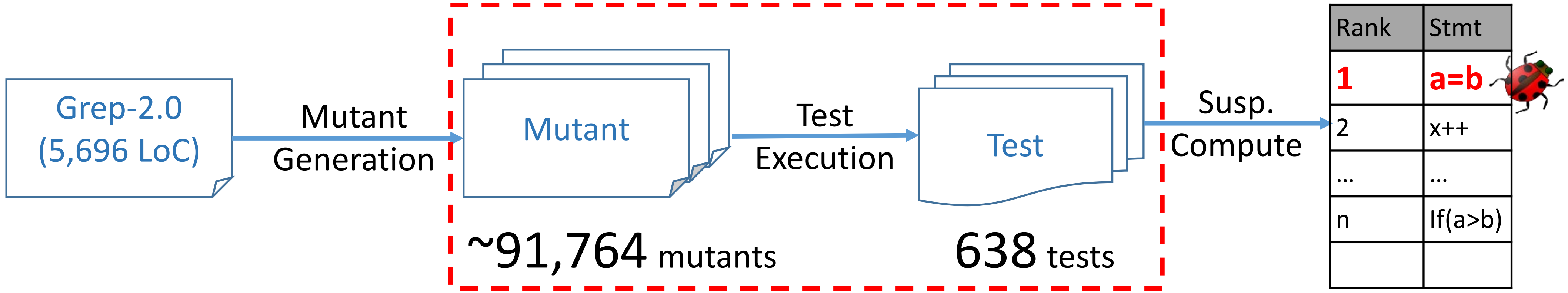
- Accurate fault localization technique using mutation analysis
    1. MUSE [ICST 14]
        - Accurate mutation-based fault Localization
          Expense metric of MUSE is **6.0%**
    2. MUSEUM [ASE 15, IST 17]
        - Accurate mutation-based fault localization for multilingual programs
    3. PRINCE [TOSEM 19]
        - Machine learning-based fault localization using various features including MBFL and SBFL
        - Expense metric of PRINCE is **2.4%**





Challenge: High Runtime Cost

- MBFL suffers from **high runtime cost** due to execution of all generated mutants against test suites.
- Example: Localize a fault in grep-2.0



- Fault localization takes **~117 hours**

Existing Solutions for Challenge

- Use MUSE [ICST 14] to show the effects of mutant reduction on MBFL

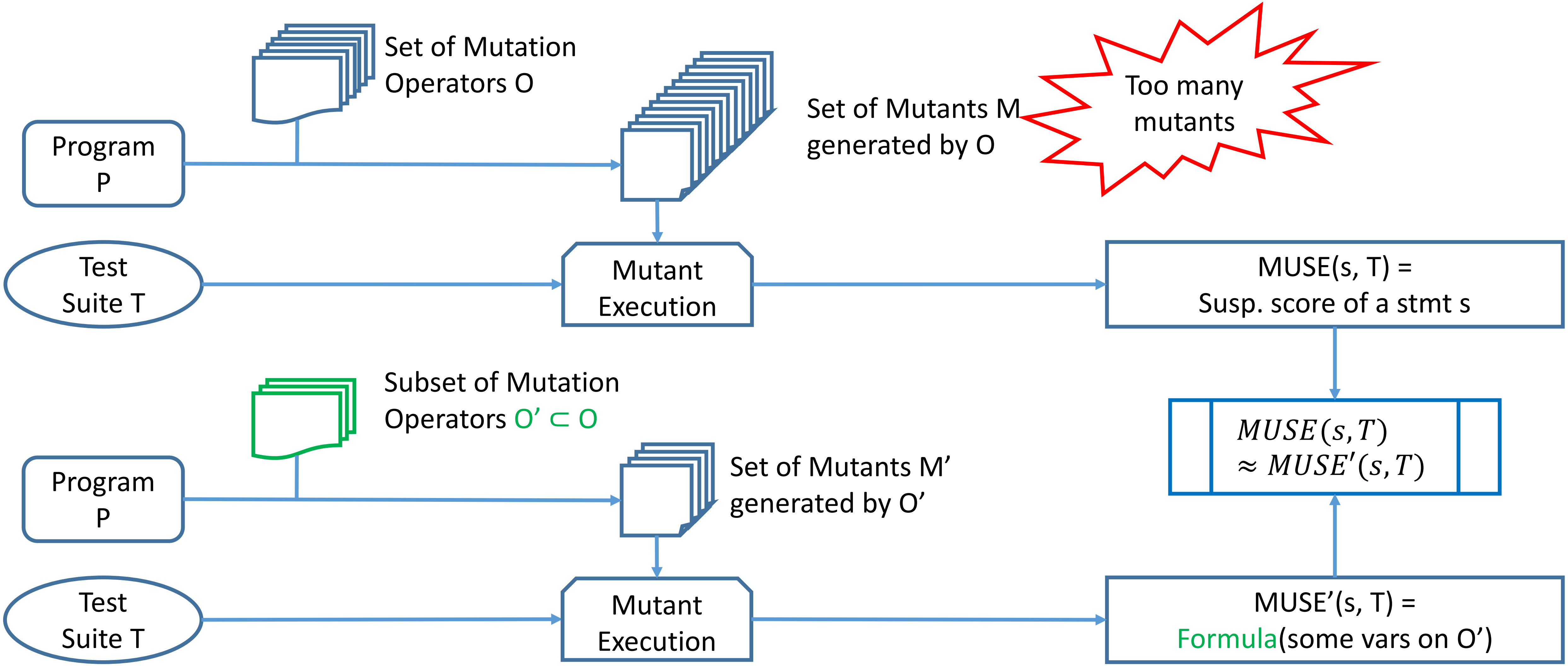
Category	Technique	Brief description	Target Programs	Mutant Reduction %
Random Selection	Wong and Mathur (J. Sys. Soft. 95)	Randomly select 10-40% of generated mutants	4 Fortran programs	60-90%
Mutation Operator based	Offutt et al. (TOSEM'96)	5 Fortran expression-level mutation operators	10 Fortran programs	77.6%
	Barbosa et al. (STVR'01)	10 C mutation operators identified through proposed 6-step guidelines	27 C programs	65.0%
	Namin et al. (ICSE'08)	28 C mutation operators identified using Cost-based Linear Regression	7 C programs (Siemens)	92.6%
	Deng et al. (ICST'13)	Only Statement-Deletion (SSDL-only) mutation operator	40 Java Classes	81.0%

… Are NOT appropriate for Debugging

- We use MUSE [ICST 14] to show the effects of mutant reduction on MBFL
- Several reduction techniques make MUSE worse than Op2 (12.1%)

Category	Technique	Brief description	Mutant Reduction %	MUSE's Results (Exam score %)
No Selection			0%	5.1%
Random Selection	Wong and Mathur (J. Sys. Soft. 95)	Randomly select 10-40% of generated mutants	60-90%	**16.2%-31.5%**
Mutation Operator based Selection	Offutt et al. (TOSEM'96)	5 Fortran expression-level mutation operators	77.6%	**24.5%**
	Barbosa et al. (STVR'01)	10 C mutation operators identified through proposed 6-step guidelines	65.0%	**17.2%**
	Namin et al. (ICSE'08)	28 C mutation operators identified using Cost-based Linear Regression	92.6%	**15.1%**
	Deng et al. (ICST'13)	Only Statement-Deletion (SSDL-only) mutation operator	81.0%	**18.5%**

Mut. Op. based Mutant Reduction for Debugging



Two Issues

- Issue 1: What should we train the mutant reduction model against?
  - Choice 1: Train the model against ideal results
  - Choice 2: Train the model against MUSE (i.e., using all mutants)

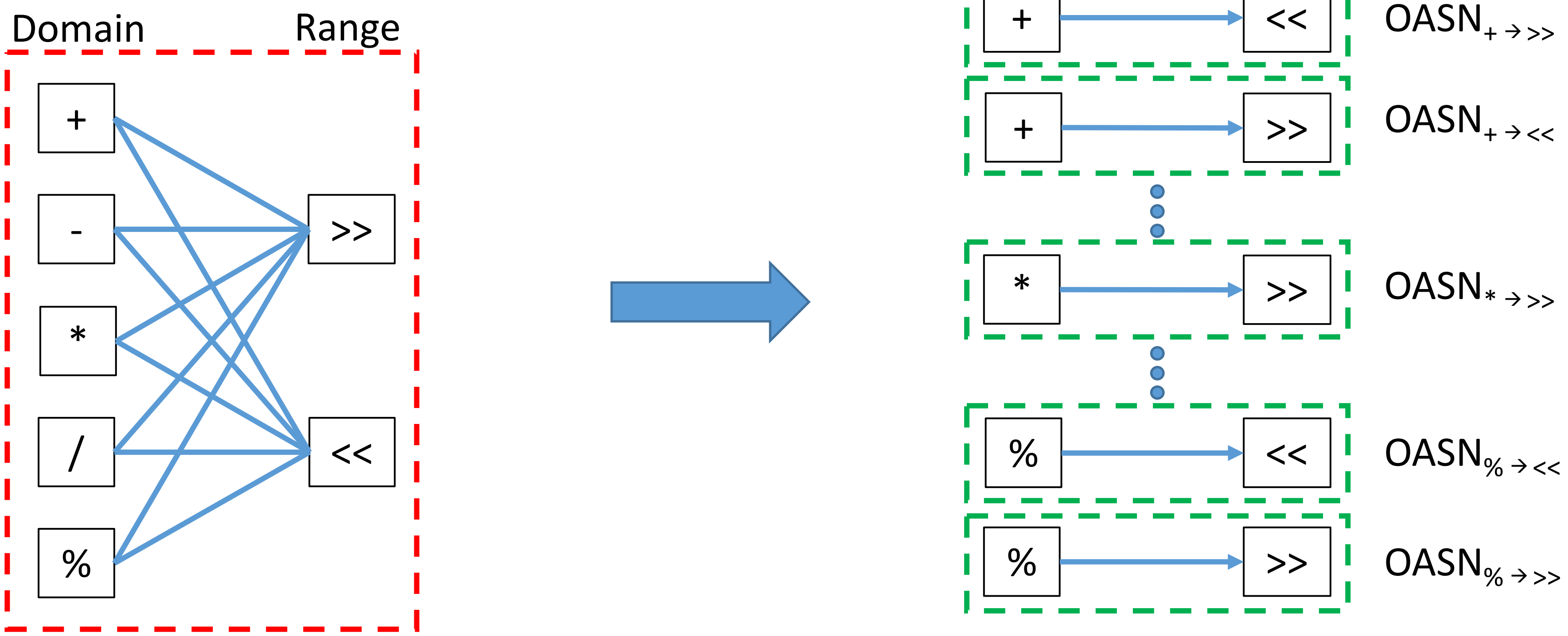
- Issue 2: Which variable should we use to construct the mutant reduction model?

Fine-grained Mutation Operators - Example

A mutation operator represents a rule to change target program code to create a mutant.

Coarse-grained Operator
OASN
Rule: Arithmetic Operator → Shift Operator

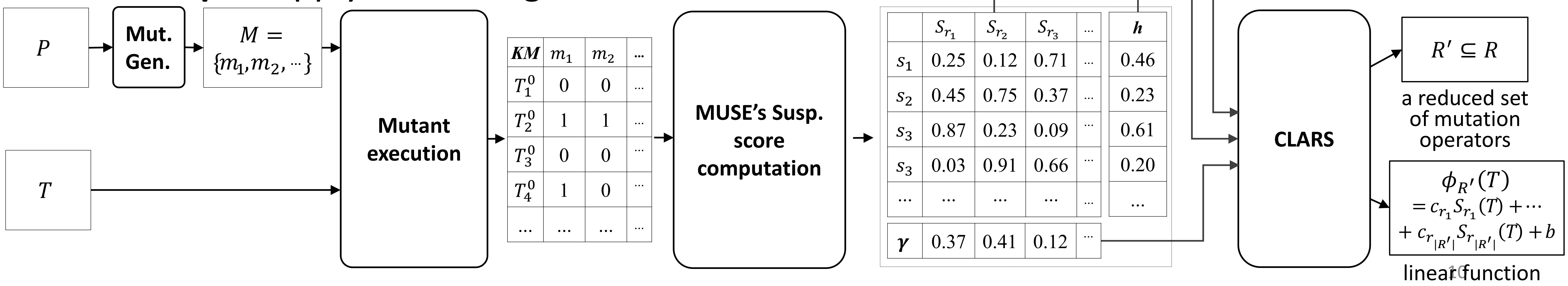
10 Fine-grained Operators
refined from OASN



<u>Domain</u>: set of tokens a mut. op. <u>mutates</u>
<u>Range</u> : set of tokens a mut. op. <u>mutates to</u>

Overall Process

- **Step 1**: Conduct mutation analysis on each program in training data
  - Generate a killmap for each program
- **Step 2**: For each statement s in a program P, calculate:
  - MUSE(s,T) = $\dfrac{\left(\sum_{m\in mut(s)} \frac{|f_P(s)\cap p_m|}{\text{f2p}+1} - \frac{|p_P(s)\cap f_m|}{\text{p2f}+1}\right)}{\left(|mut(s)| + 1\right)}$
  - MUSE_{Srn}(s,T) = $\dfrac{\left(\sum_{m\in mut_{s_{r1}}(s)} \frac{|f_P(s)\cap p_m|}{\text{f2p}+1} - \frac{|p_P(s)\cap f_m|}{\text{p2f}+1}\right)}{\left(|mut_{s_{r1}}(s)| + 1\right)}$
- Cost matrix = # mutants generated by each operator
- **Step 3**: Apply CLARS to generate models



Research Questions

- RQ1. Effect of the mutant reduction on mutation-based fault localization
  - **Efficiency**: How much execution time does the reduction technique can reduce?
  - **Effectiveness**: How accurate is MUSE with the proposed mutant reduction in localizing a target fault in terms of expense metric?
- RQ2. Effect of the fine-grained mutation operators
  - How much do fine-grained mutation operators affect the number of selected mutants and the accuracy of MBFL compared to coarse-grained mutation operators?
- RQ3. Comparison with random mutant selection
- RQ4. Comparison with existing mutation-operator based mutant selection techniques

Target Programs

- We target 75 faulty versions of 5 SIR programs
  - Bash and Vim are not included because of time constraints
- Eliminate trivially equivalent, duplicated mutants (md5 checksum comparison)

Prog.	#faulty Vers	LoC	#tests
flex	19	7254	567
grep	18	5696	809
gzip	16	3040	208
make	19	9820	1006
sed	3	3980	360
Average	15.0	5879.8	2755.8

Techniques to Compare

Technique	Description	Related RQ
AllMutLearn	Apply CLARS with fine-grained mutation operators and learn	RQ1-4
IdealLearn	Apply CLARS with fine-grained mutation operators and learn	RQ1
AllMutLearn^{TRD}	Apply CLARS with coarse-grained mutation operators and learn	RQ2
RND^{SN}	Randomly selects the same number of mutants generated AllMutLearn	RQ3
Offutt et al. Barbosa et al. Namin et al. Deng et al.	Existing mutation operator-based mutant reduction techniques	RQ4

Experiment Setup

- CLARS learning setup
  - Run CLARS 1,000 iterations
- Evaluation setup
  - 10-fold cross validation for total 75 faulty versions
- Machine setup
  - HW: AMD Ryzen 5950X (max 4.9Ghz) 16C32T, 64GB memory
  - OS: Ubuntu 20.04 LTS

Results – RQ1: Effects of Mutant Reduction

- AllMutLearn is **11.5 times faster** and **13.7% more accurate** than MUSE
  - **~10 hours for each fault**, on average
  - Note that execution time of IdealMutLearn and AllMutLearn does not include learning time
- IdealMutLearn is worse than AllMutLearn in terms of both time and accuracy

Target Programs	MUSE		IdealMutLearn		AllMutLearn	
	Time(h)	Expense(%)	Time(h)	Expense(%)	Time(h)	Expense(%)
flex	9038.5	13.7	1012.3	15.2	953.2	5.5
grep	10786.1	1.3	806.7	9.5	813.6	2.4
gzip	6222.4	5.3	706.5	8.9	309.4	5.9
make	10097.2	3.9	1544.4	6.4	1022.3	5.4
sed	7088.4	1.2	691.4	5.3	661.5	2.7
Average	**8646.5**	**5.1**	**952.2**	**9.1**	**752.0**	**4.4**

Results – RQ2: Effects of Fine-grained Mut. Ops.

- Fine-grained mutation operator makes AllMutLearn **1.5 times faster** and **2.3 times more accurate** than AllMutLearn^{TRD}

Target Programs	AllMutLearn^{TRD}		AllMutLearn	
	Time(h)	Expense(%)	Time(h)	Expense(%)
flex	1477.5	13.1	953.2	5.5
grep	1179.7	4.8	813.6	2.4
gzip	485.8	12.6	309.4	5.9
make	1349.4	12.7	1022.3	5.4
sed	972.4	6.8	661.5	2.7
Average	**1093.0**	**10.0**	**752.0**	**4.4**

Results – RQ3: Comparison with Random

- AllMutLearn is **4.2 times more accurate** than RND^{SN}

Target Programs	RND^{SN}		AllMutLearn	
	Time(h)	Expense(%)	Time(h)	Expense(%)
flex	953.2	23.9	953.2	5.5
grep	813.6	13.2	813.6	2.4
gzip	309.4	25.1	309.4	5.9
make	1022.3	21.0	1022.3	5.4
sed	661.5	9.3	661.5	2.7
Average	**752.0**	**18.5**	**752.0**	**4.4**

Results – RQ4: Comparison with Existing Mut. Op. based Reduction

- AllMutLearn is at least **3.4 times more accurate** than the existing operator-based mutant reduction for debugging

Target Programs	Offutt et al. (TOSEM’96)		Barbosa et al. (STVR’01)		Namin et al. (ICSE’08)		Deng et al. (ICST’13)		AllMutLearn	
	Time(h)	Expense(%)	Time(h)	Expense(%)	Time(h)	Expense(%)	Time(h)	Expense(%)	Time(h)	Expense(%)
flex	2044.9	35.6	2910.4	15.3	695.6	13.5	1597.1	12.3	953.2	5.5
grep	2343.6	21.3	3850.6	19.5	806.2	12.6	1926.4	20.3	813.6	2.4
gzip	1379.9	26.5	2199.6	11.3	414.4	21.3	1229.5	19.3	309.4	5.9
make	2058.2	19.3	3604.7	19.3	821.9	12.3	1822.5	23.1	1022.3	5.4
sed	1651.3	19.7	2580.2	20.5	508.8	15.6	1239.0	17.4	661.5	2.7
Average	**1895.6**	**24.5**	**3029.1**	**17.2**	**649.4**	**15.1**	**1562.9**	**18.5**	**752.0**	**4.4**

Conclusion

- Learning-based mutant reduction can significantly decrease execution as well as increase accuracy of MBFL
- Fine-grained mutation operators are effective to construct a better mutant reduction model

Future Work

- Through analysis to identify when AllMutLearn works well or bad
- Apply mutant reduction to PRINCE techniques
  - PRINCE utilizes various features to improve accuracy and efficiency
  - PRINCE with mutant reduction will be better than MUSE with mutant reduction
- Reduce more execution time
  - Predictive mutation analysis

Q & A