Deep Specification Mining
(Supplemental Material)

June 4, 2018

1 Detailed Results

Tables 2, 3, 4, and 5 describe detailed precision, recall, and F-measure results of our proposed approach and the baselines for each of the target library classes shown in Table 1. The first two tables consider the first evaluation scheme (Scheme I), while the last two consider the second evaluation scheme (Scheme II). They provide additional details for the bar graphs drawn in Figures 4 and 5 of the main manuscript.

Table 1: Target Library Classes. "# Methods" represents the number of class methods that are analyzed, "# Generated Test Cases" is the number of test cases generated by Randoop, "# Recorded Method Calls" is the number of recorded method calls in the execution traces, "NFST" stands for `NumberFormatStringTokenizer`.

Target Library Class	**# Methods**	**# Generated Test Cases**	**# Recorded Method Calls**
`ArrayList`	18	42,865	22,996
`HashMap`	11	53,396	67,942
`Hashtable`	8	79,403	89,811
`HashSet`	8	23,181	257,428
`LinkedList`	7	13,731	4,847
`NFST`	5	15,8998	95,149
`Signature`	5	79,096	205,386
`Socket`	21	80,035	130,876
`StringTokenizer`	5	148,649	336,924
`StackAr`	7	549,648	13,2826
`ZipOutputStream`	5	162,971	43,626

Table 2: Scheme I: Traditional 1-tail, Traditional 2-tail, and CONTRACTOR++ . "P" is Precision, "R" is Recall, and "F" is F-measure, "N/A" means that the result is not available.

Target Library Class	Traditional 1-tails			Traditional 2-tails			CONTRACTOR++		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
`ArrayList`	45.17%	8.25%	13.96%	45.67%	7.67%	13.13%	86.30%	22.77%	36.03%
`HashMap`	44.83%	17.73%	25.41%	46.80%	4.80%	8.71%	52.60%	100.00%	68.94%
`HashSet`	49.79%	13.21%	20.88%	54.58%	13.21%	21.27%	57.11%	48.11%	52.22%
`Hashtable`	49.80%	36.89%	42.39%	61.16%	23.14%	33.58%	100.00%	86.53%	92.78%
`LinkedList`	47.62%	18.99%	27.15%	50.57%	17.25%	25.72%	100.00%	75.47%	86.02%
`NFST`	45.72%	16.80%	24.57%	53.08%	16.80%	25.52%	29.00%	31.94%	30.40%
`Signature`	52.66%	74.03%	61.54%	56.75%	74.03%	64.25%	100.00%	50.24%	66.88%
`Socket`	76.56%	23.44%	35.89%	78.09%	19.74%	31.52%	67.91%	46.43%	55.15%
`StackAr`	18.00%	15.30%	16.54%	18.00%	15.30%	16.54%	52.03%	26.26%	34.91%
`StringTokenizer`	55.42%	50.55%	52.88%	55.63%	50.55%	52.97%	100.00%	11.92%	21.30%
`ZipOutputStream`	30.17%	100.00%	46.36%	33.47%	81.31%	47.42%	45.77%	100.00%	62.80%
Average	46.89%	34.11%	33.42%	50.34%	29.44%	30.97%	71.88%	54.52%	55.22%

Table 3: Scheme I: SEKT 1-tail, SEKT 2-tail, and TEMI. "P" is Precision, "R" is Recall, and "F" is F-measure, "N/A" means that the result is not available .

Target Library Class	SEKT 1-tails			SEKT 2-tails			Optimistic TEMI		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
`ArrayList`	47.71%	8.11%	13.86%	44.24%	7.67%	13.07%	88.12%	9.33%	16.87%
`HashMap`	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
`HashSet`	49.79%	13.21%	20.88%	54.58%	13.21%	21.27%	100.00%	13.21%	23.34%
`Hashtable`	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
`LinkedList`	48.08%	18.45%	26.67%	50.25%	16.21%	24.52%	100.00%	3.90%	7.51%
`NFST`	45.66%	16.80%	24.56%	55.42%	16.80%	25.78%	100.00%	6.27%	11.80%
`Signature`	53.41%	74.03%	62.05%	56.33%	74.03%	63.98%	100.00%	24.27%	39.06%
`Socket`	68.69%	23.24%	34.73%	75.52%	17.46%	28.37%	N/A	N/A	N/A
`StackAr`	18.00%	15.30%	16.54%	18.00%	15.30%	16.54%	N/A	N/A	N/A
`StringTokenizer`	53.85%	50.55%	52.15%	N/A	N/A	N/A	N/A	N/A	N/A
`ZipOutputStream`	31.50%	100.00%	47.91%	N/A	N/A	N/A	N/A	N/A	N/A
Average	46.30%	35.52%	33.26%	50.62%	22.95%	27.65%	97.62%	11.40%	19.72%

2 Additional Discussion

According to the empirical evaluation, DSM outperforms all baseline mining algorithms that only analyze method ordering in execution traces to construct FSAs (i.e., k-tails). However, DSM is not better than CONTRACTOR++ and Optimistic TEMI for a few target classes: `ArrayList`, `LinkedList`, `Hashtable`,

Table 4: Scheme II: Traditional 1-tail, Traditional 2-tail, and CONTRACTOR++. "P" is Precision, "R" is Recall, and "F" is F-measure.

Target Library Class	**Traditional 1-tails**			**Traditional 2-tails**			**CONTRACTOR++**		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
`ArrayList`	70.33%	2.92%	5.61%	93.14%	1.57%	3.08%	86.30%	22.77%	36.03%
`HashMap`	43.12%	32.95%	37.35%	57.05%	13.58%	21.93%	52.60%	100.00%	68.94%
`HashSet`	57.01%	14.38%	22.96%	66.52%	8.67%	15.35%	57.11%	48.11%	52.22%
`Hashtable`	76.68%	80.24%	78.42%	94.28%	60.05%	73.37%	100.00%	86.53%	92.78%
`LinkedList`	100.00%	15.14%	26.30%	100.00%	4.20%	8.07%	100.00%	75.47%	86.02%
`NFST`	86.00%	57.22%	68.72%	90.82%	35.49%	51.04%	29.00%	31.94%	30.40%
`Signature`	100.00%	100.00%	100.00%	100.00%	91.22%	95.41%	100.00%	50.24%	66.88%
`Socket`	48.51%	30.30%	37.30%	82.03%	12.69%	21.98%	67.91%	46.43%	55.15%
`StackAr`	89.93%	27.88%	42.57%	89.93%	27.88%	42.57%	52.03%	26.26%	34.91%
`StringTokenizer`	100.00%	100.00%	100.00%	100.00%	81.31%	89.69%	100.00%	11.92%	21.30%
`ZipOutputStream`	70.23%	100.00%	82.51%	74.05%	82.57%	78.08%	45.77%	100.00%	62.80%
Average	76.53%	51.00%	54.70%	86.17%	38.11%	45.50%	71.88%	54.52%	55.22%

Table 5: Scheme II: SEKT 1-tail, SEKT 2-tail, and TEMI. "P" is Precision, "R" is Recall, and "F" is F-measure, "N/A" means that the result is not available .

Target Library Class	**SEKT 1-tails**			**SEKT 2-tails**			**Optimistic TEMI**		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
`ArrayList`	100.00%	2.22%	4.34%	100.00%	1.57%	3.08%	92.72%	22.19%	35.82%
`HashMap`	43.12%	32.95%	37.35%	57.05%	13.58%	21.93%	52.60%	100.00%	68.94%
`HashSet`	57.01%	14.38%	22.96%	66.52%	8.67%	15.35%	65.91%	48.11%	55.62%
`Hashtable`	100.00%	69.05%	81.69%	100.00%	48.66%	65.47%	100.00%	86.53%	92.78%
`LinkedList`	100.00%	10.16%	18.45%	100.00%	3.41%	6.59%	100.00%	75.47%	86.02%
`NFST`	79.60%	57.22%	66.58%	92.89%	33.75%	49.51%	35.00%	31.94%	33.40%
`Signature`	100.00%	91.22%	95.41%	100.00%	81.46%	89.78%	100.00%	50.24%	66.88%
`Socket`	85.17%	22.28%	35.32%	94.29%	11.73%	20.87%	69.37%	46.43%	55.62%
`StackAr`	89.93%	27.88%	42.57%	89.93%	27.88%	42.57%	N/A	N/A	N/A
`StringTokenizer`	100.00%	85.54%	92.21%	100.00%	73.57%	84.77%	87.92%	0.00%	0.00%
`ZipOutputStream`	76.16%	100.00%	86.47%	74.57%	62.23%	67.84%	49.48%	100.00%	66.20%
Average	84.63%	46.63%	53.03%	88.66%	33.32%	42.52%	75.30%	56.09%	56.13%

and `Socket`. Both CONTRACTOR++ and Optimistic TEMI mainly rely on likely invariants rather than the ordering of method invocations observed in the execution traces to construct FSAs.

There is a trade-off between using likely invariants and raw method orderings in execution traces to construct FSAs. Program invariants can be helpful in inferring specifications, but they are more expensive to process and infer. This is because we need to record additional information in the traces such as values of all visible variables in entry and exit points of every invoked method. The more values of variables captured in the execution traces, the more accurate the inferred invariants. On the other hand, utilizing method orderings in execution traces for specification mining is less costly. However, execution traces provide limited information of important properties that might be valuable to enhance quality of inferred FSAs.