

Enhancing Symbolic Execution with Self-Configuring Parameters

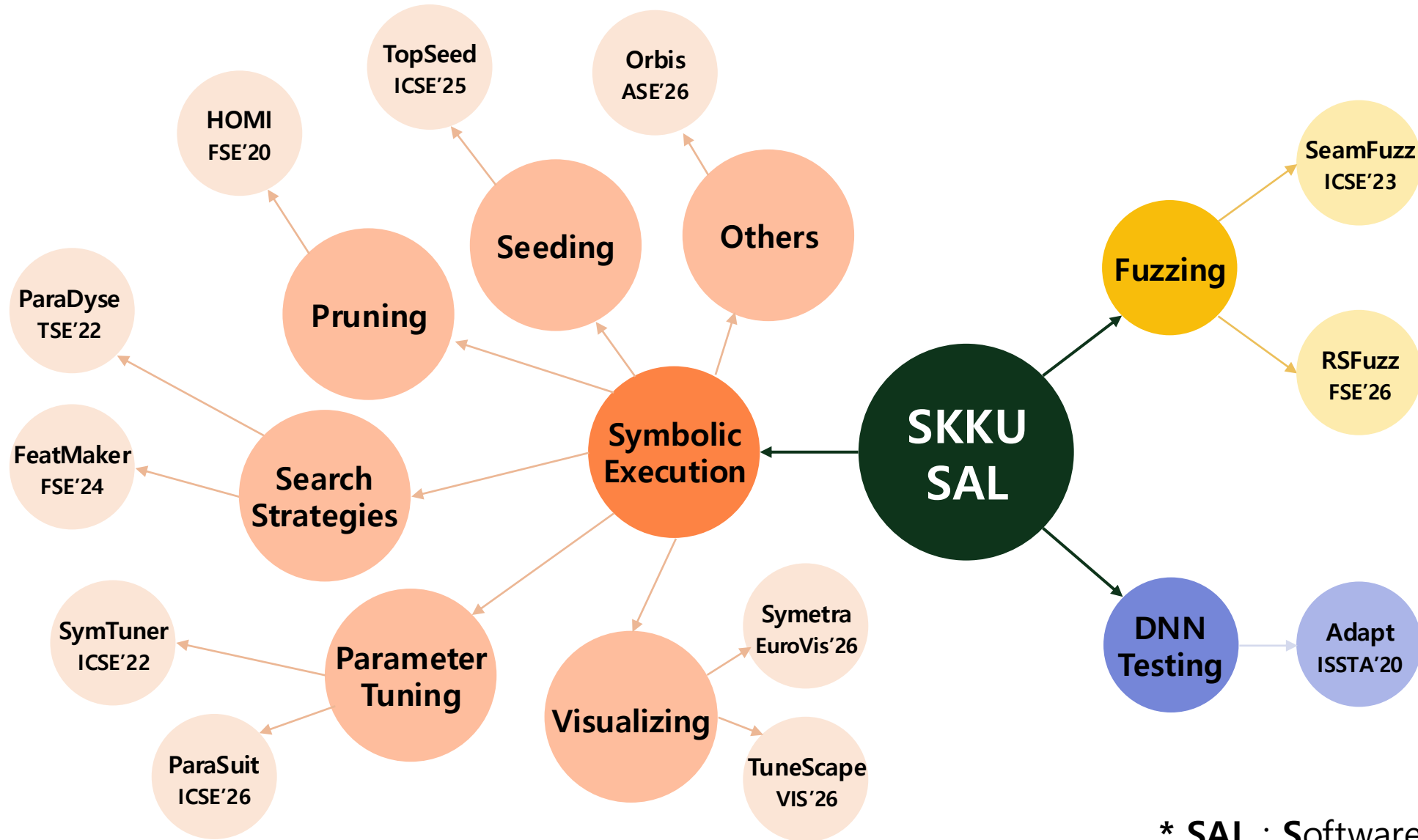
Minjong Kim¹, Sooyoung Cha¹

¹Sungkyunkwan University

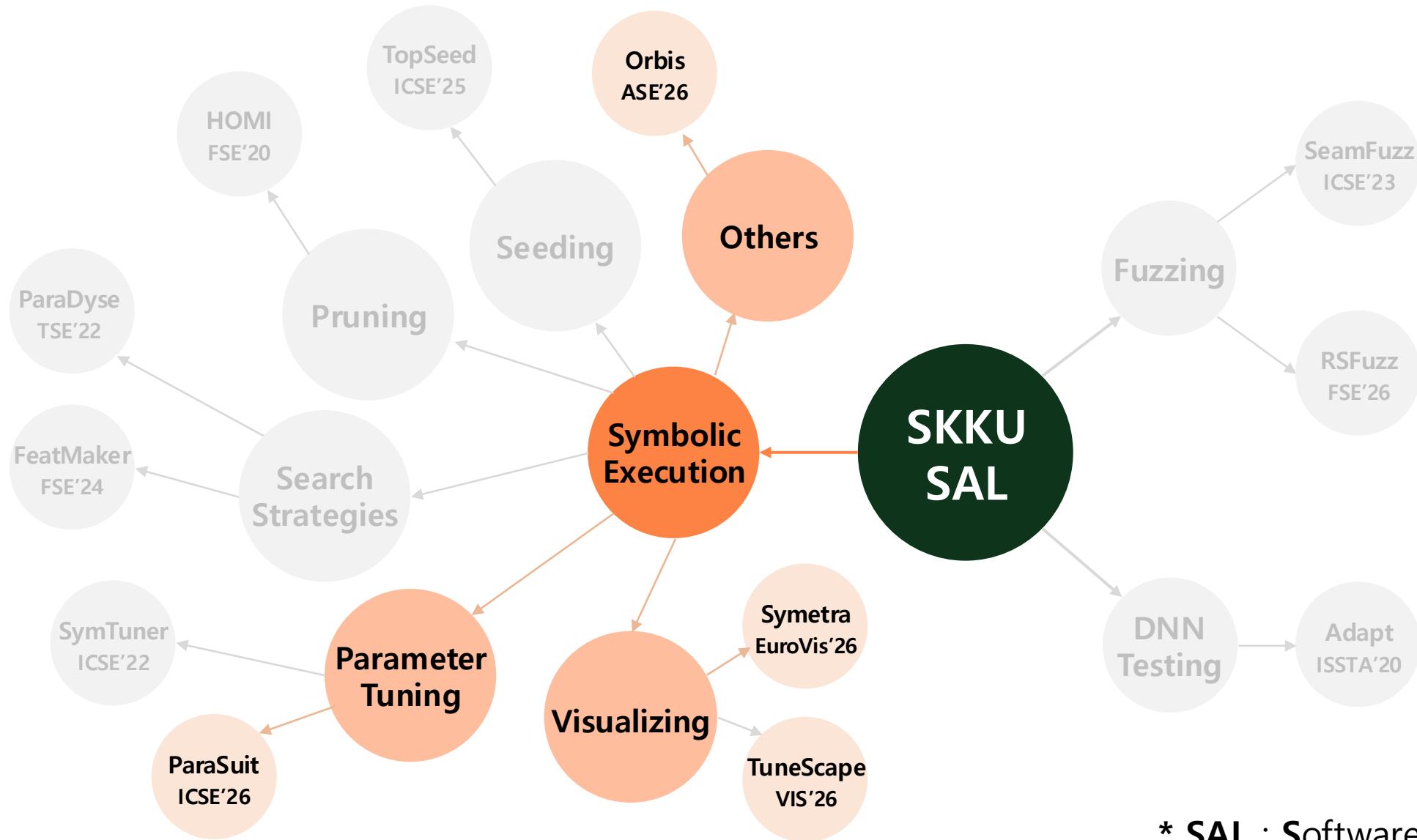
KLEE'26 @ Munich, Germany







* SAL : Software Analysis Lab



* **SAL** : Software **A**nalysis **L**ab



Enhancing Symbolic Execution with Self-Configuring Parameters

Minjong Kim
Sungkyunkwan University
Suwon, Republic of Korea
minjong.kim@skku.edu

Sooyoung Cha*
Sungkyunkwan University
Suwon, Republic of Korea
sooyoung.cha@skku.edu

Abstract

We present PARASUIT, a self-configuring technique that enhances symbolic execution by autonomously adjusting its parameters tailored to each program under test. Modern symbolic execution tools are typically equipped with various external parameters to effectively test real-world programs. However, the need for users to fine-tune a multitude of parameters for optimal testing outcomes makes these tools harder to use and limits their potential benefits. Despite recent efforts to improve this tuning process, existing techniques are not self-configuring; they cannot dynamically identify which parameters to tune for each target program, and for each manually selected parameter, they sample a value from a fixed, user-defined set of candidate values that is specific to that parameter and remains unchanged across programs. The goal of this paper is to automatically configure symbolic execution parameters from scratch for each program. To this end, PARASUIT begins by automatically identifying all available parameters in the symbolic execution tool and evaluating each parameter’s impact through interactions with the tool. It then applies a specialized algorithm to iteratively select promising parameters, construct sampling spaces for each, and update their sampling probabilities based on data accumulated from symbolic execution runs using sampled parameter values. We implemented PARASUIT on KLEE and assessed it across 12 open-source C programs. The results demonstrate that PARASUIT significantly outperforms the state-of-the-art method without self-configuring parameters, achieving an average of 26% higher branch coverage. Remarkably, PARASUIT identified 11 unique bugs, four of which were exclusively discovered by PARASUIT.

CCS Concepts

• Software and its engineering → Software testing and debugging; • Computing methodologies → Machine learning.

1 Introduction

Symbolic execution [15, 16, 38, 73] is undoubtedly one of the prominent software testing methods, actively researched in both academia and industry [14, 39, 40]. Its primary goal is to efficiently generate valuable test-cases that cover hard-to-reach execution paths in the target program or reveal hidden bugs within a time budget. To achieve this, symbolic execution translates program inputs into symbolic variables, runs the target program with those variables, accumulates symbolic branch conditions, and generates test-cases that satisfy these conditions using an SMT solver [30, 37]. However, due to the need to gather symbolic branch conditions and generate test-cases via the SMT solver, symbolic execution faces two well-known open challenges: the path-explosion problem and the high cost of constraint solving. In response to these challenges, various techniques have been proposed, including search strategy [19, 44, 57, 93], path-pruning [8, 22, 90, 91], constraints solving [54, 60, 78, 81], and state-merging [3, 55, 74, 82]. As a result, the effectiveness of modern symbolic execution tools incorporating these techniques has seen marked improvement.

Yet, despite their notable performance enhancements, it has paradoxically become more challenging for users to utilize symbolic execution tools effectively. This challenge stems from the need for users to determine the "parameter values" that influence the effectiveness of these tools to achieve the desired testing outcomes. Specifically, given a target program, users choose which parameters to use from the total parameters provided by a symbolic execution tool, and then assign a suitable value to each selected parameter. Unfortunately, the number of adjustable parameters in practical symbolic execution tools, like KLEE [14], exceeds a hundred, including boolean, string, and integer types. Thus, adjusting these parameter values for testing each program imposes a considerable burden on users. This burden often leads users to settle for the default parameter settings offered by symbolic execution tools. For



Eurographics Conference on Visualization (EuroVis) 2026
N. Andrienko, M. Chen, and B. Wang
(Guest Editors)

COMPUTER GRAPHICS forum
Volume 45 (2026), Number 3

Symetra: Visual Analytics for the Parameter Tuning Process of Symbolic Execution Engines

D. Hong, M. Kim, S. Cha, and J. Jo
Sungkyunkwan University, Republic of Korea

Abstract
Symbolic execution engines such as KLEE automatically generate test cases to maximize branch coverage, but their numerous parameters make it difficult to understand the parameters' impact, leading the user to rely on suboptimal default configurations. While automated tuners have shown promising results, they provide limited insights into why certain configurations work well, motivating the need for Human-in-the-Loop approaches. In this work, we present a visual analytics system, Symetra, designed to support Human-in-the-Loop parameter tuning of symbolic execution engines. To handle a large number of parameters and their configurations, we provide two complementary overviews of their impact on branch coverage values and patterns. Building on these overviews, our system enables collective analysis, allowing the user to contrast groups of configurations and identify differences that may affect branch coverage. We also report on case studies and a Human-in-the-Loop tuning process, demonstrating that experts not only interpreted parameter impacts and identified complementary configurations, but also improved upon fully automated approaches in both branch coverage and tuning efficiency.

CCS Concepts
• Human-centered computing → Visual analytics; • Software and its engineering → Software testing and debugging;

1. Introduction
One important research question in software engineering is how to ensure that a program is robust to arbitrary inputs. This is typically accomplished by generating test cases, executing them, and checking for uncaught exceptions or violated assertions. Symbolic execution [GKS05, CGP⁺08, SMA05, CDE08] is a powerful method that automatically generates such test cases. The key idea is to systematically explore the program's execution paths by treating input branch coverage; the user often requires a set of complementary configurations (i.e., configurations whose test cases cover different sets of branches) to further improve branch coverage. For example, a configuration with the Breadth-First Search (BFS) strategy may provide high overall branch coverage, while one with a Depth-First Search (DFS) strategy may better explore deeply nested branches.

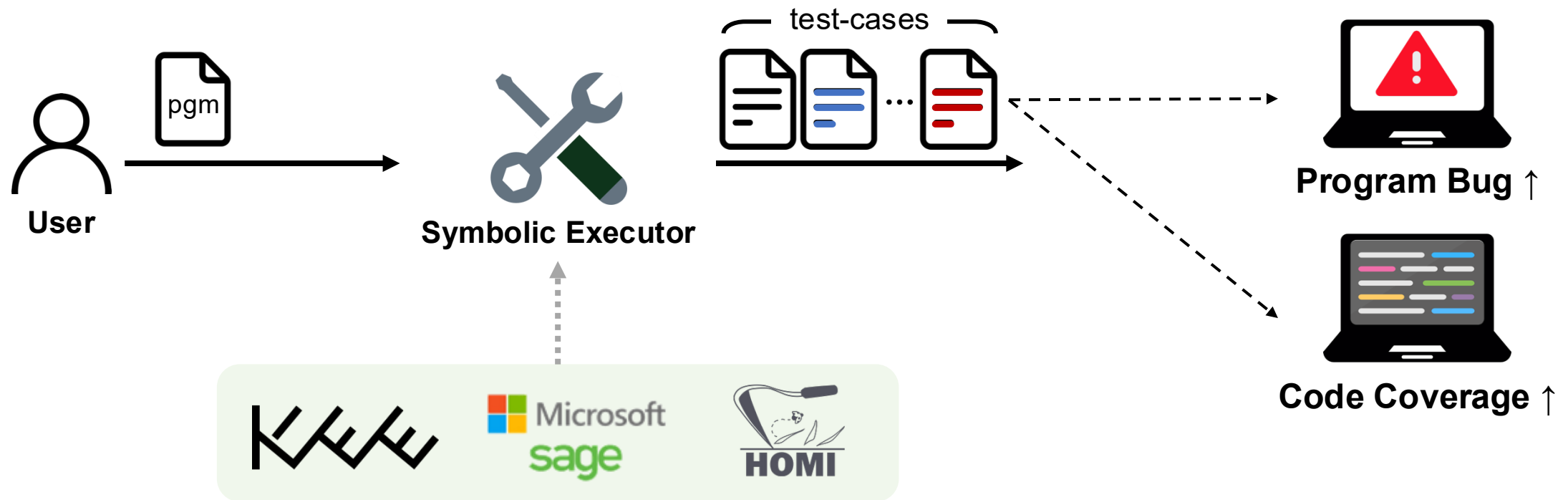
In this work, we aim to enable the user to effectively leverage symbolic execution engines by providing a visual analytics system.

304.05349v1 [cs.HC] 7 Apr 2026

Symetra (EuroVis'26)

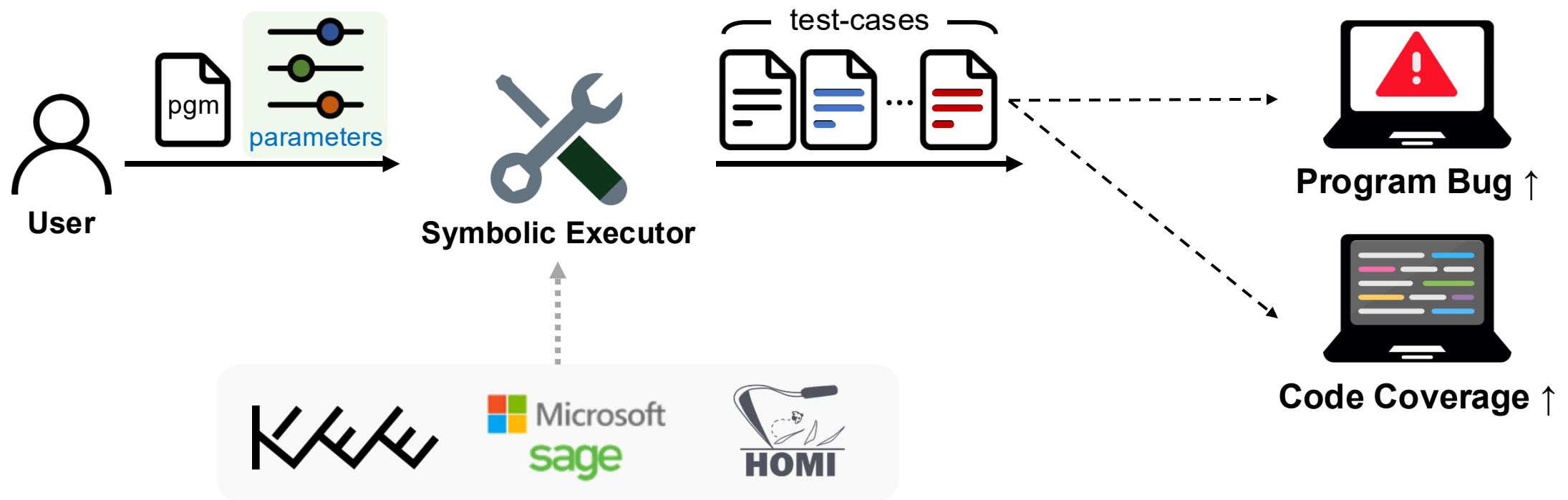
Symbolic Execution

- An **effective** technique for white-box software testing



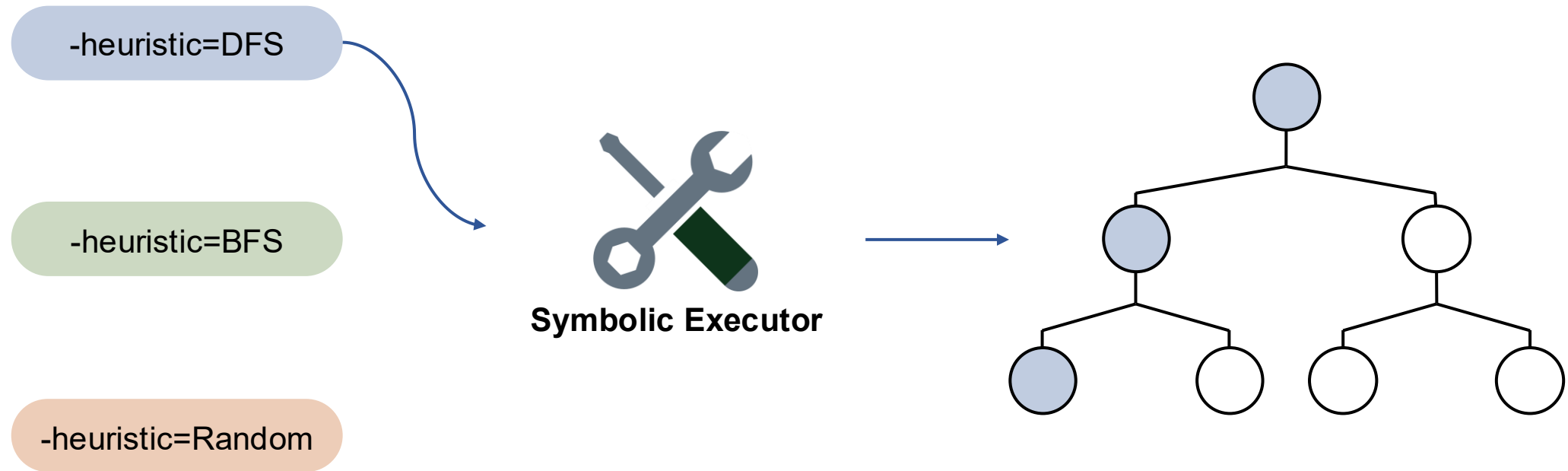
Parameters in Symbolic Execution

- All factors that **affect the performance** of symbolic execution (e.g., option configs)



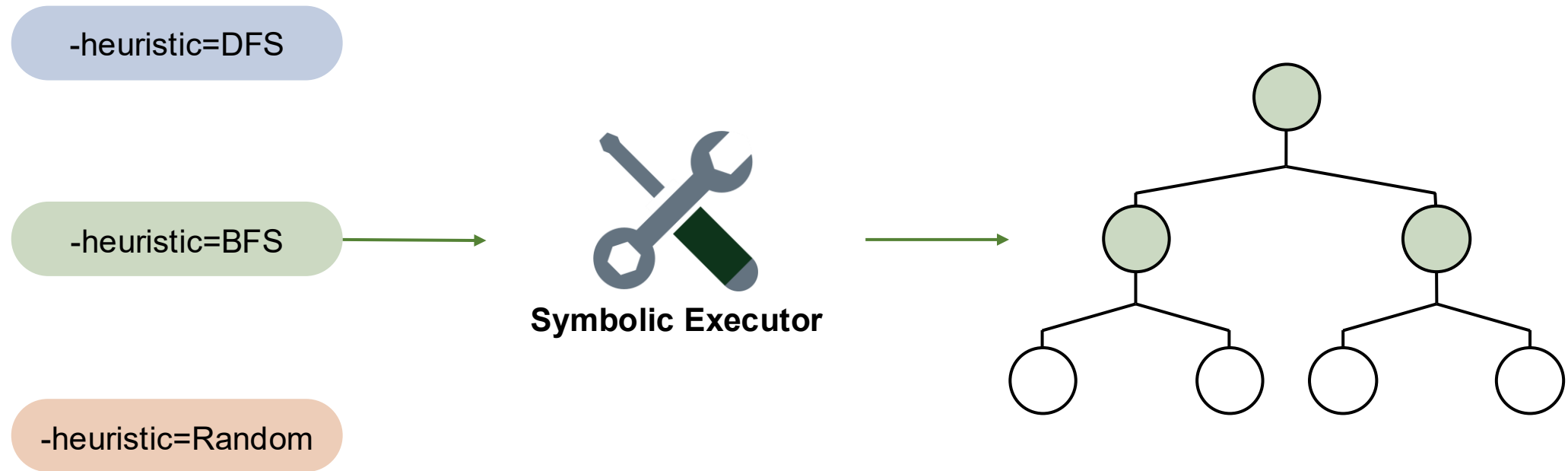
Parameters in Symbolic Execution

- All factors that **affect the performance** of symbolic execution (e.g., option configs)



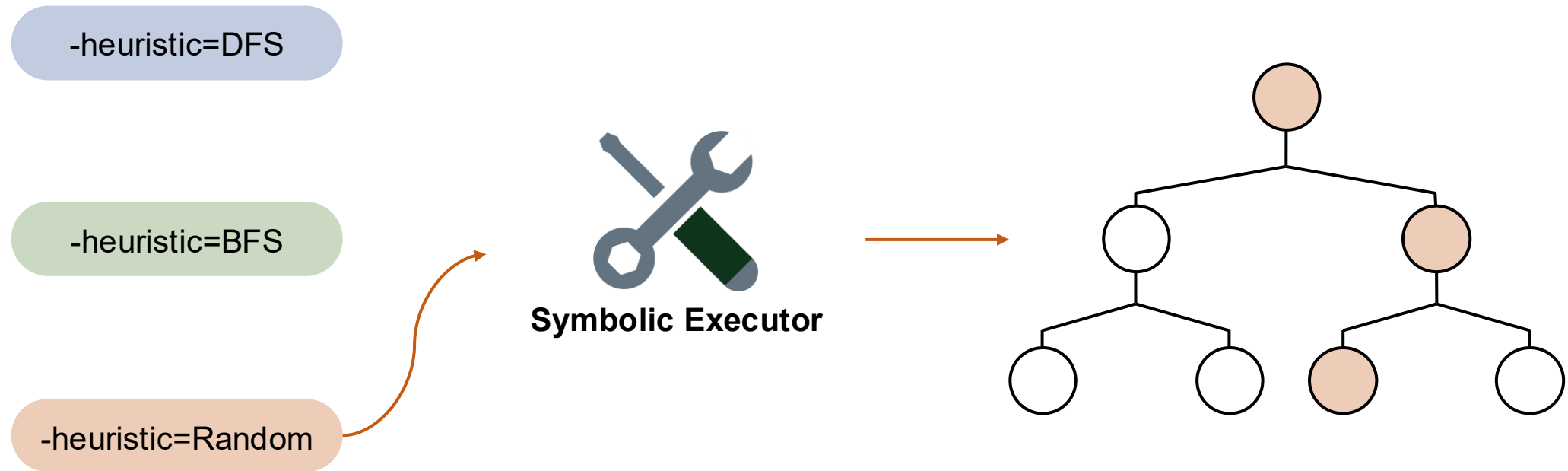
Parameters in Symbolic Execution

- All factors that **affect the performance** of symbolic execution (e.g., option configs)



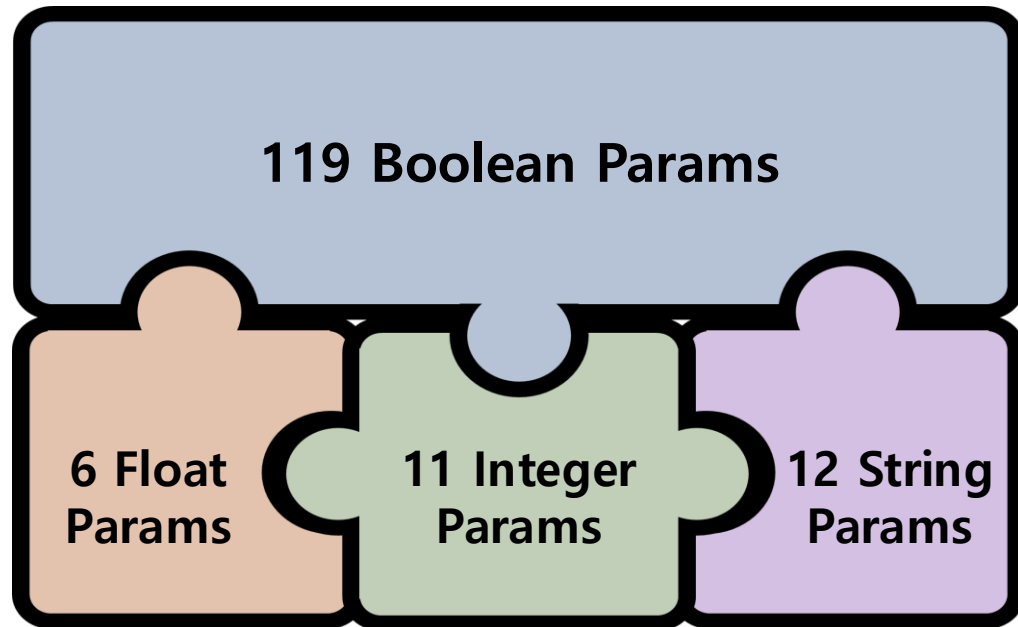
Parameters in Symbolic Execution

- All factors that **affect the performance** of symbolic execution (e.g., option configs)



Difficulty of Parameter Tuning

- There are **148 parameters** in KLEE

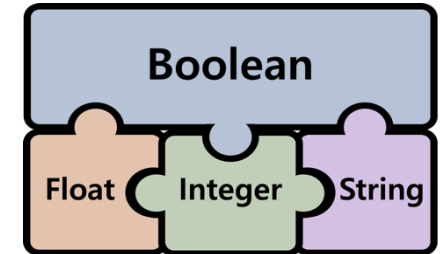



Symbolic Executor

Difficulty of Parameter Tuning

- There are 148 parameters in KLEE
- An example of parameter tuning in KLEE

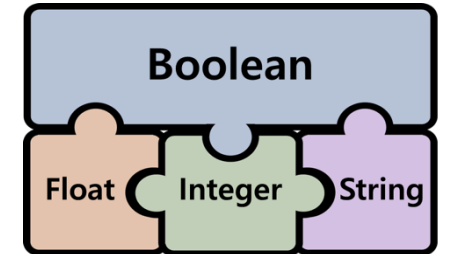
```
$ klee --simplify-sym-indices --max-memory=1000 --optimize  
--use-forked-solver --use-cex-cache --external-calls=all  
--max-sym-array-size=4096 --max-instruction-time=30s  
--max-time=60min --max-memory-inhibit=false  
--max-static-fork-pct=1 --max-static-solve-pct=1  
--max-static-cpfork-pct=1 --switch-type=internal  
--search=random-path --search=nurs:covnew  
--batch-instructions=10000 ./pgm.bc --sym-args 0 1 10  
--sym-args 0 2 2 --sym-files 1 8 --sym-stdin 8 --sym-stdout
```



Difficulty of Parameter Tuning

- There are 148 parameters in KLEE
- An example of parameter tuning in KLEE

```
$ klee --simplify-sym-indices --max-memory=? --optimize  
--use-forked-solver --use-cex-cache --external-calls=?  
--max-sym-array-size=? --max-instruction-time=?  
--max-time=? --max-memory-inhibit=false  
--max-static-fork-pct=? --max-static-solve-pct=?  
--max-static-cpfork-pct=? --switch-type=?  
--search=? --search=?  
--batch-instructions=? ./pgm.bc --sym-args ?  
--sym-args ? --sym-files ? --sym-stdin ? --sym-stdout
```



●Prior Works

- 1) Use the baseline tool's default parameter values **without any optimization**
 - “KLEE was configured with its **default policy**” (TOSEM'25)
 - “We use recommended symbolic arguments setting **provided by KLEE**” (PLDI'24)
 - “The symbolic arguments **used in KLEE [1]**” (FSE'20)
 - “Unmodified KLEE with **default settings**” (ISSTA'19)
 - “A configuration **used in KLEE [1]**” (ASE'18)
 - “The **default configuration** of KLEE” (ASE'17)
 - “The **same argument configuration** as in prior work” (ISSTA'17)

[1] Cristian Cadar, Daniel Dunbar, and Dawson Engler. KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs. (OSDI'08)

Difficulty of Parameter Tuning

●Prior Works

2) SymTuner (ICSE'22)

SYM-TUNER: Maximizing the Power of Symbolic Execution by Adaptively Tuning External Parameters

Sooyoung Cha
Sungkyunkwan University
Republic of Korea
sooyoung.cha@skku.edu

Myungho Lee
Korea University
Republic of Korea
myungho_lee@korea.ac.kr

Seokhyun Lee
Korea University
Republic of Korea
seokhyunlee@korea.ac.kr

Hakjoo Oh*
Korea University
Republic of Korea
hakjoo_oh@korea.ac.kr

ABSTRACT

We present SYM-TUNER, a novel technique for automatically tuning external parameters of symbolic execution tools. Symbolic execution tools have important external parameters (e.g., seed input) that critically affect their performance. To the huge parameter space, however, symbolic execution tools are notoriously difficult even for experts. As a consequence, symbolic execution tools have typically been used in a suboptimal manner that, for example, simply relies on the default parameter settings of the tools and loses the opportunity for better performance. In this paper, we aim to change this situation by automatically configuring symbolic execution parameters. With SYM-TUNER that takes parameter spaces to be tuned, symbolic execution tools are run without manual parameter configurations; instead, appropriate parameter values are learned and adjusted during symbolic execution. To achieve this, we present a learning algorithm that observes the behavior of symbolic execution and accordingly updates the sampling probability of each parameter space. We evaluated SYM-TUNER with KLEE on 12 open-source C programs. The results show that SYM-TUNER increases branch coverage of KLEE by 56% on average and finds 8 more bugs than KLEE with its default parameters over the latest releases of the programs.

ACM Reference Format:

Sooyoung Cha, Myungho Lee, Seokhyun Lee, and Hakjoo Oh. 2022. SYM-TUNER: Maximizing the Power of Symbolic Execution by Adaptively Tuning External Parameters. In *44th International Conference on Software Engineering (ICSE '22)*, May 21–29, 2022, Pittsburgh, PA, USA. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3510003.3510185>

5m ☆ **SymTuner: Maximizing the Power of Symbolic Execution by Adaptively Tuning External Parameters**
Technical Track
Sooyoung Cha Sungkyunkwan University, Myungho Lee Korea University, Seokhyun Lee Korea University, South Korea, Hakjoo Oh Korea University
Pre-print Media Attached

DISTINGUISHED PAPER AWARD

```
--search-random-path --search-headers --search-headers --search-headers
--batch-instructions=10000 ./pgm.bc --sym-args 0 1 10
--sym-args 0 2 2 --sym-files 1 8 --sym-stdin 8 --sym-stdout ...
```

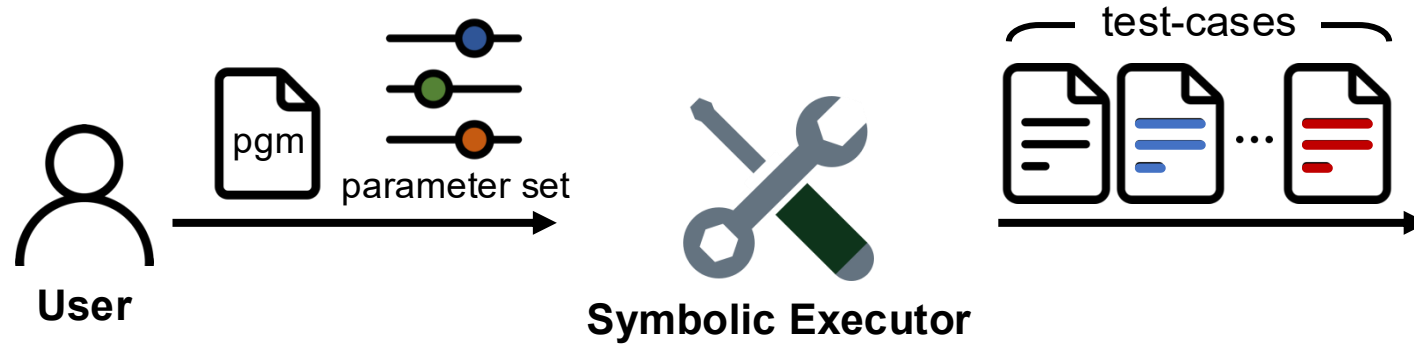
Figure 1: The parameter setting of KLEE used in [49]

and constraint solving [6, 14, 23, 34, 36, 38, 39, 45, 48, 52, 54, 66]. Equipped with these techniques, symbolic execution tools such as KLEE [11] have become publicly available and widely used in both academia and industry [12, 13].

Despite the progress, however, maximizing the performance of modern symbolic execution tools is notoriously difficult in practice. One main reason is that state-of-the-art symbolic executors have a number of important external parameters that critically affect their effectiveness. For example, Figure 1 shows a typical command for running KLEE, which was tailored to GNU Coreutils by the original authors of KLEE [11, 49]. These parameters, for example, are to select which search strategy to use, to decide the symbolic arguments, and to choose the memory budget. It is well-known that these parameters have a huge impact on the runtime performance of symbolic execution and therefore must be carefully tuned for

●Prior Works

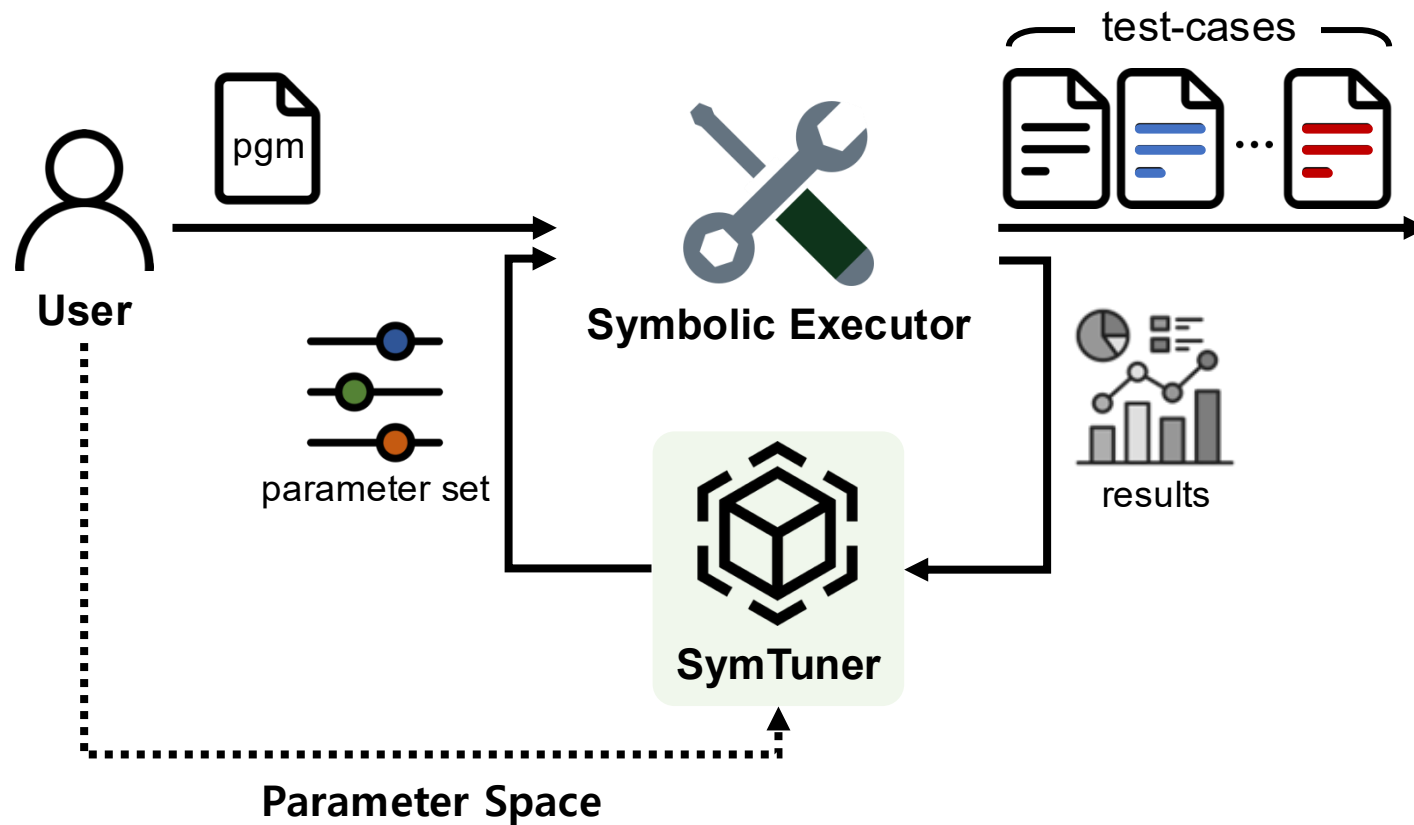
2) SymTuner (ICSE'22)



Difficulty of Parameter Tuning

● Prior Works

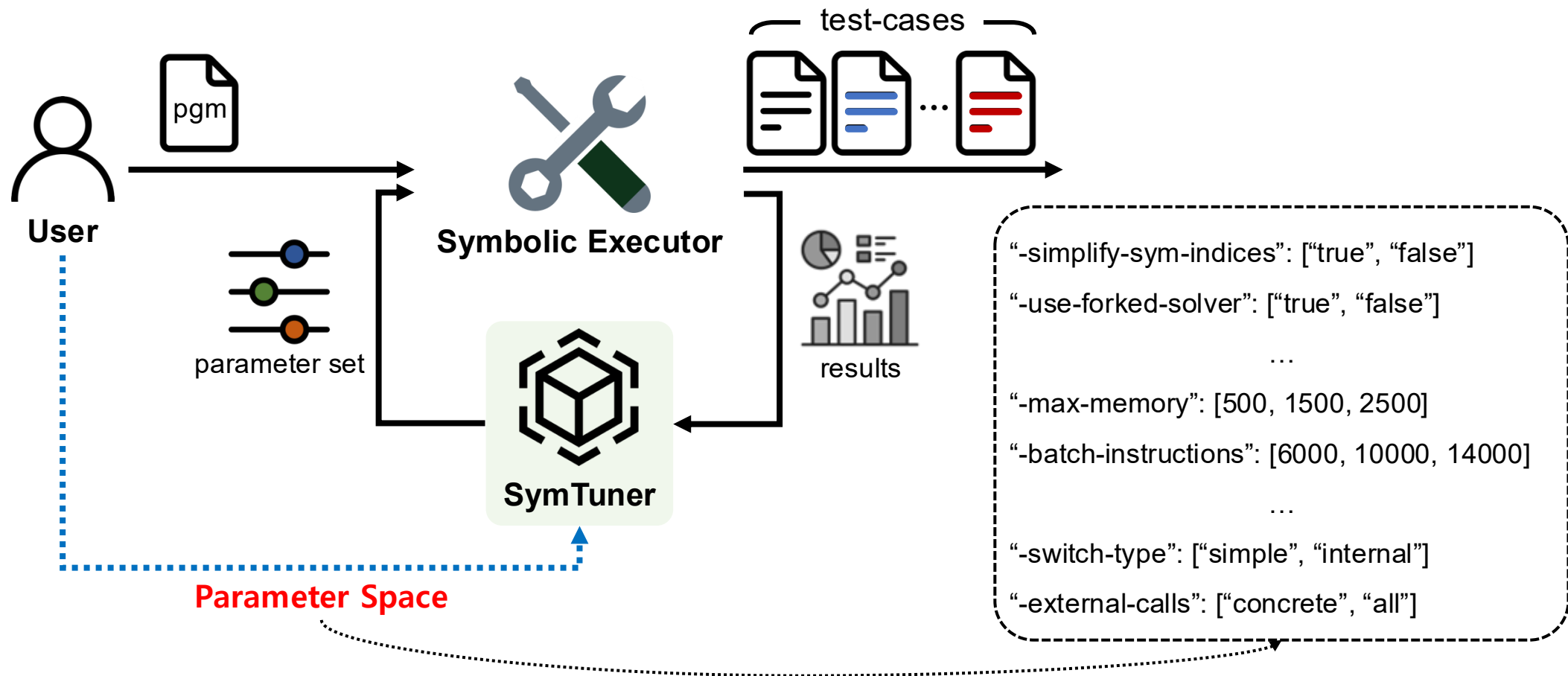
2) SymTuner (ICSE'22)



Difficulty of Parameter Tuning

● Prior Works

2) SymTuner (ICSE'22)



Limitations of State-of-the-art Technique

● Which parameters should be selected for defining the search space?

- KLEE has **148 parameters**,
- while SymTuner defines the search space for **20 of them**.

"-simplify-sym-indices": ["true", "false"]

"-use-forked-solver": ["true", "false"]

...

"-max-memory": [500, 1000, 1500, 2000, 2500]

"-batch-instructions": [6000, 8000, 10000, 12000, 14000]

...

"-switch-type": ["simple", "internal"]

"-external-calls": ["concrete", "all"]

} 20 Parameters

Limitations of State-of-the-art Technique

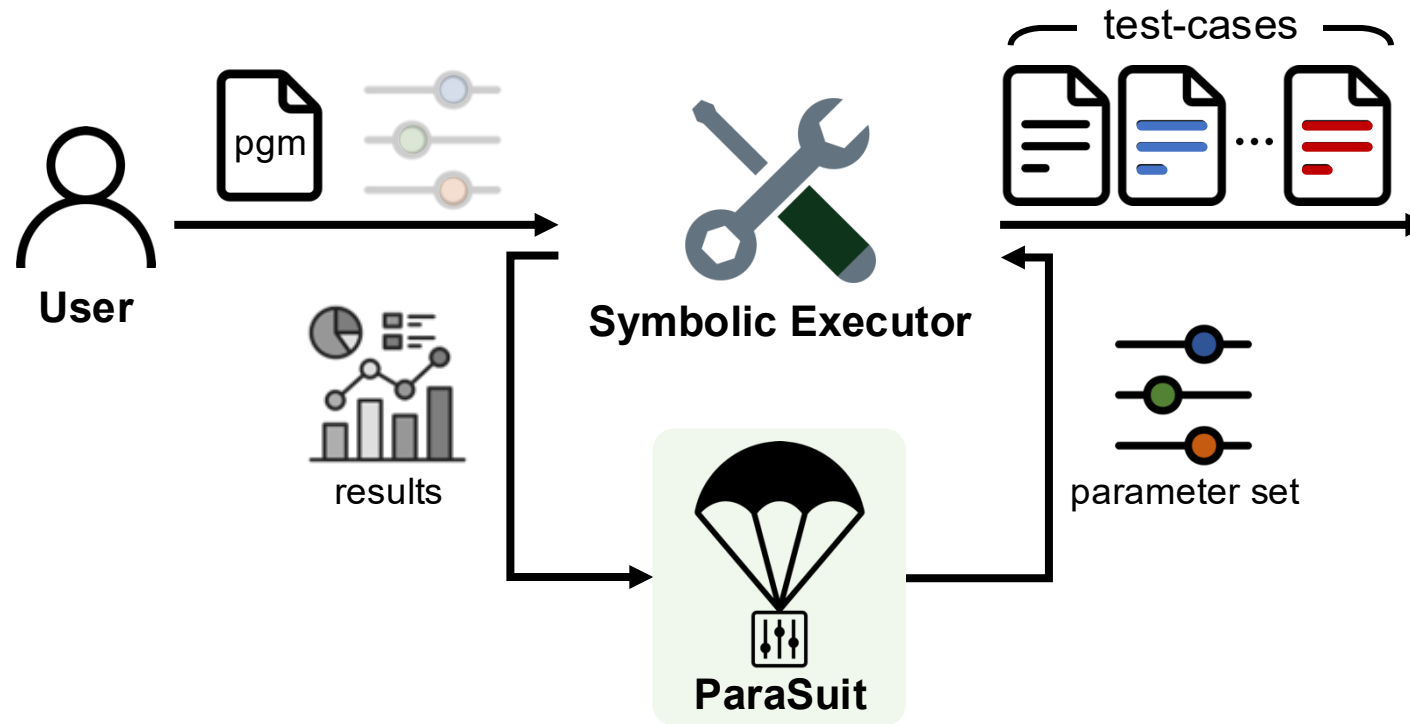
- What values should be defined as the search space for each parameter?
 - Integer/float parameter space is **unbounded**,
 - while SymTuner defines **five uniformly spaced values** for each parameter.

```

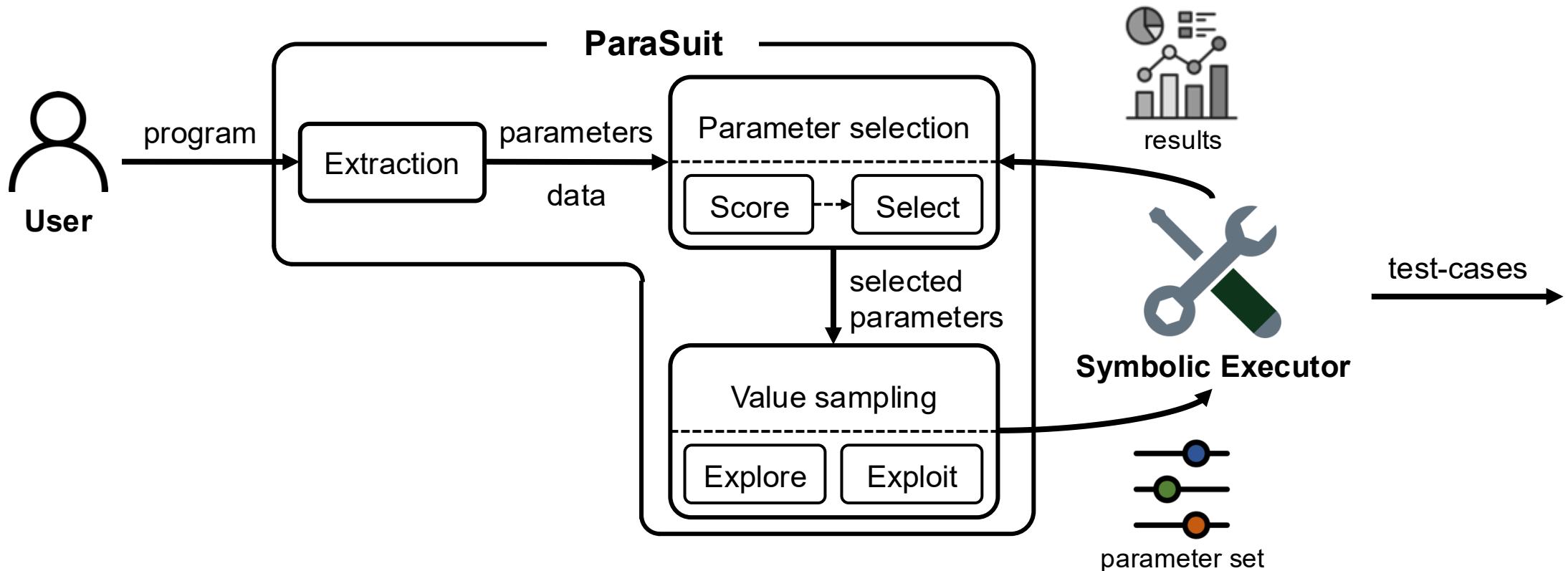
-simplify-sym-indices: ["true", "false"]
-use-forked-solver: ["true", "false"]
...
-max-memory: [500, 1000, 1500, 2000, 2500]
-batch-instructions: [6000, 8000, 10000, 12000, 14000]
                        └─── 5 Values ───┘
...
-switch-type: ["simple", "internal"]
-external-calls: ["concrete", "all"]

```

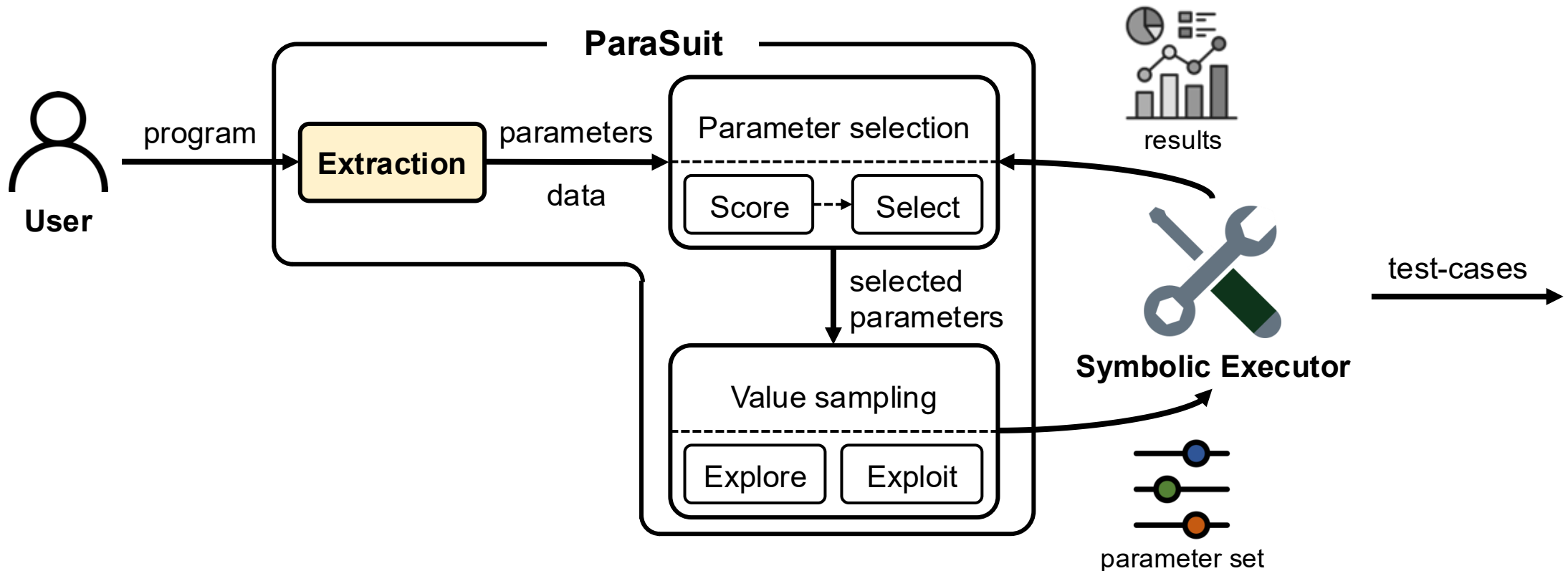
- **Fully automate** the tuning process (parameter selection + value sampling)
- Iteratively define **program-aware** parameter values.



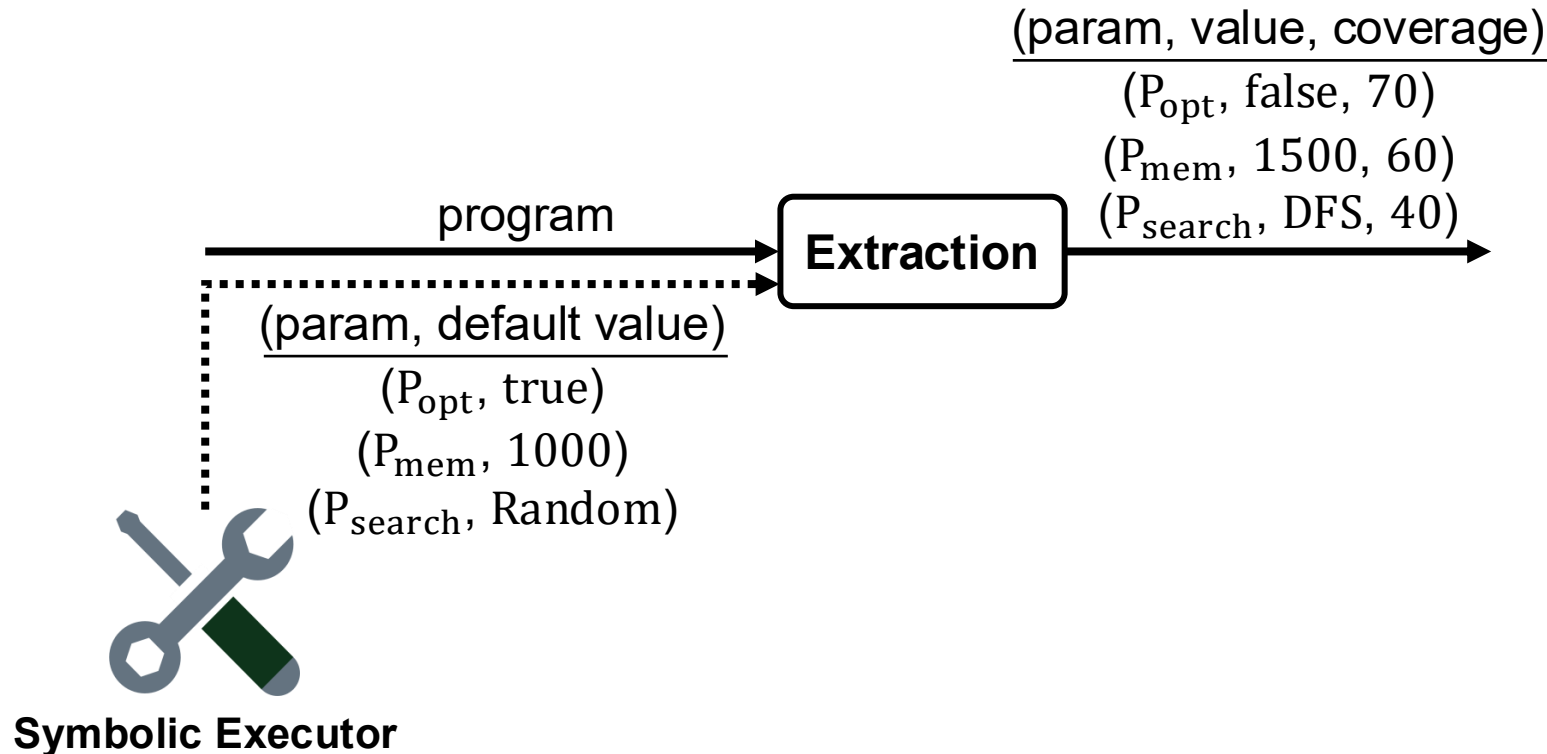
- **Fully automate** the tuning process (parameter selection + value sampling)
- Iteratively define **program-aware** parameter values.



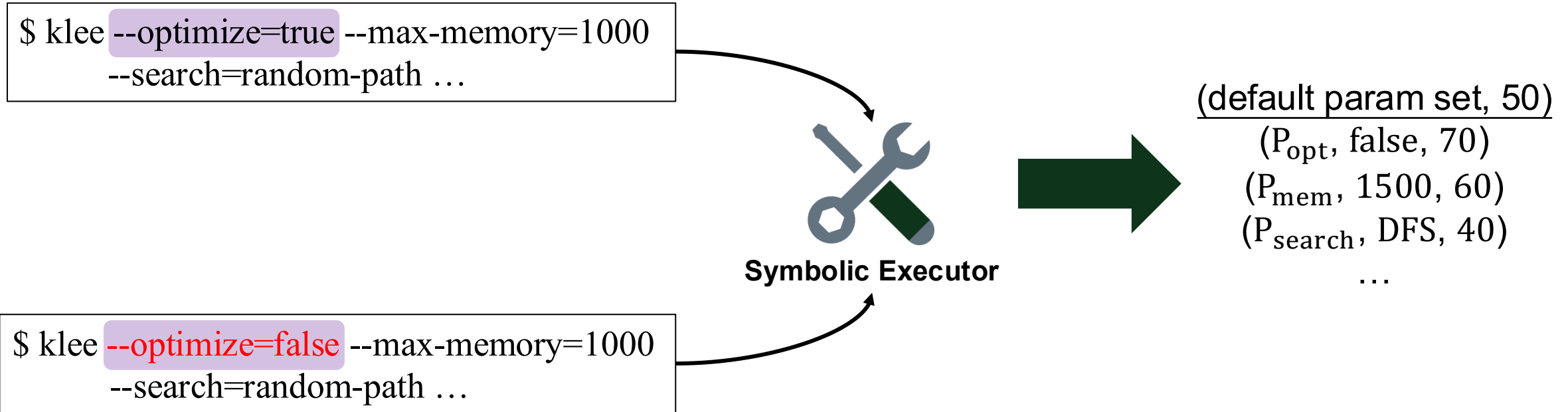
- Extract **initial information** required for parameter selection and value sampling



- Extract **initial information** required for parameter selection and value sampling
 - String / Boolean : Extract **possible values** using help script and source code
 - Integer / Float : Sample **a value around the default** value

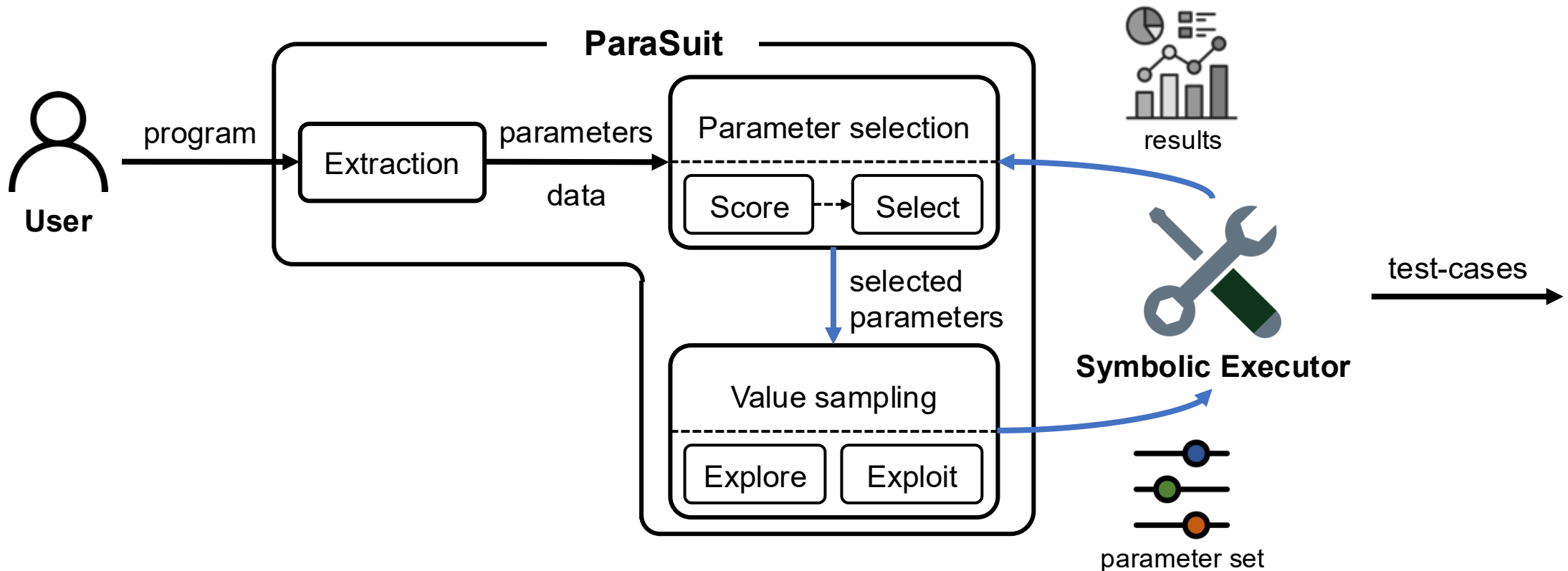


- Try different values for each parameter and store the results as data



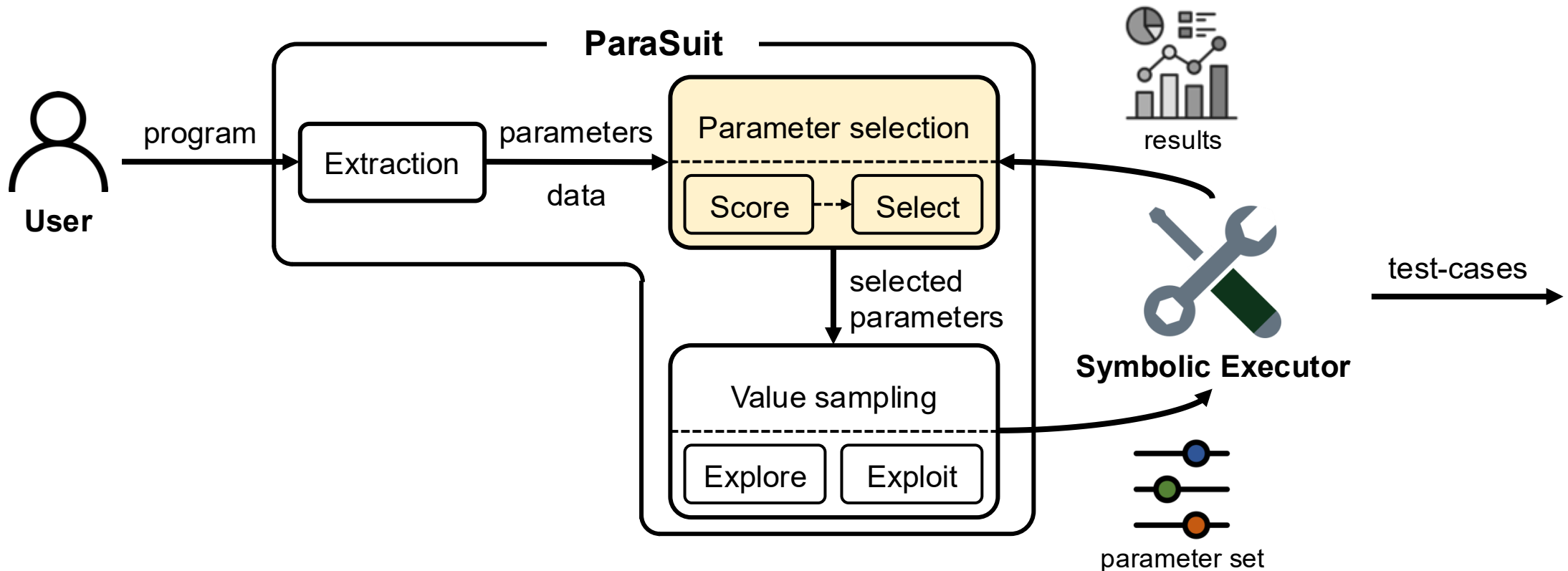
Iterative Process of ParaSuit

- After Extraction stage,
- ParaSuit **repeats its iterative stages** until the time budget is exhausted.

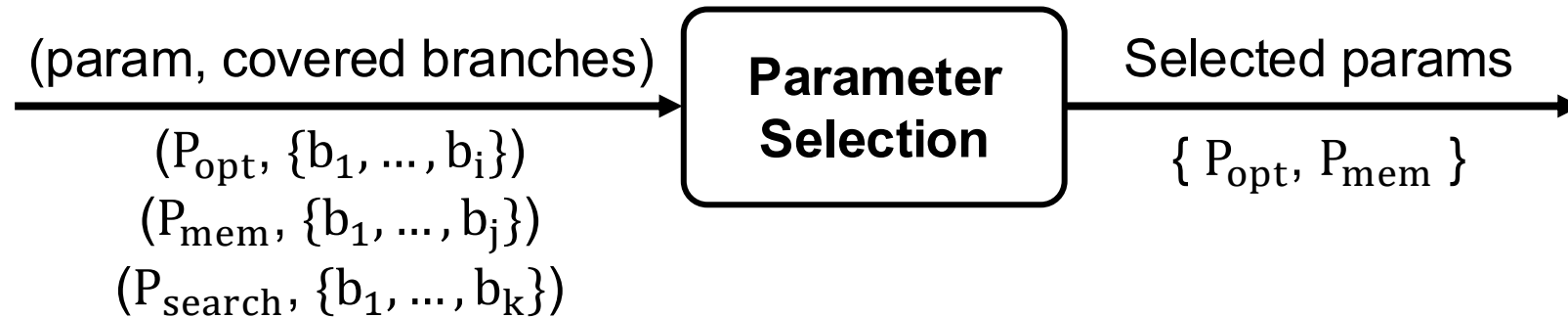


Parameter Selection Stage

- **Probabilistically** select effective parameters using accumulated data

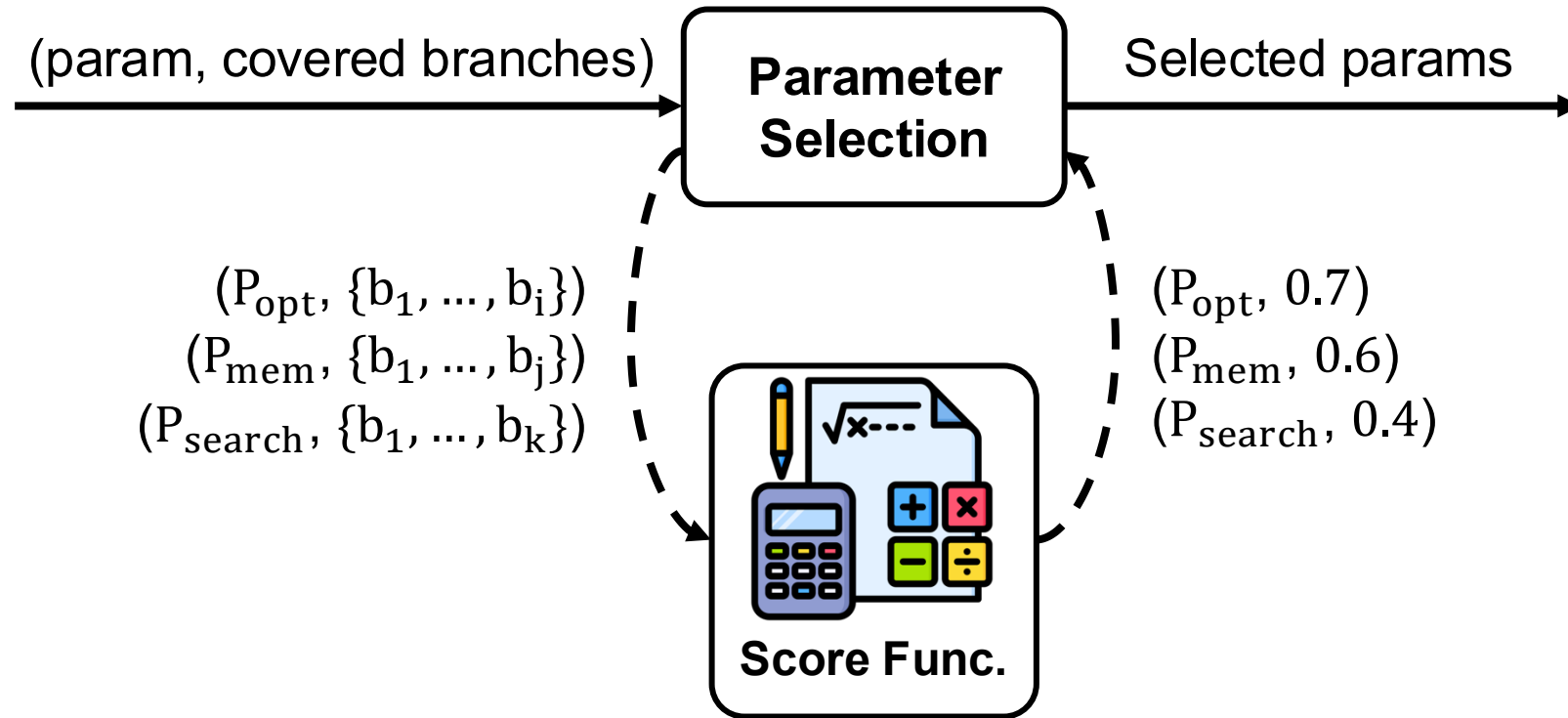


- **Probabilistically** select effective parameters using accumulated data



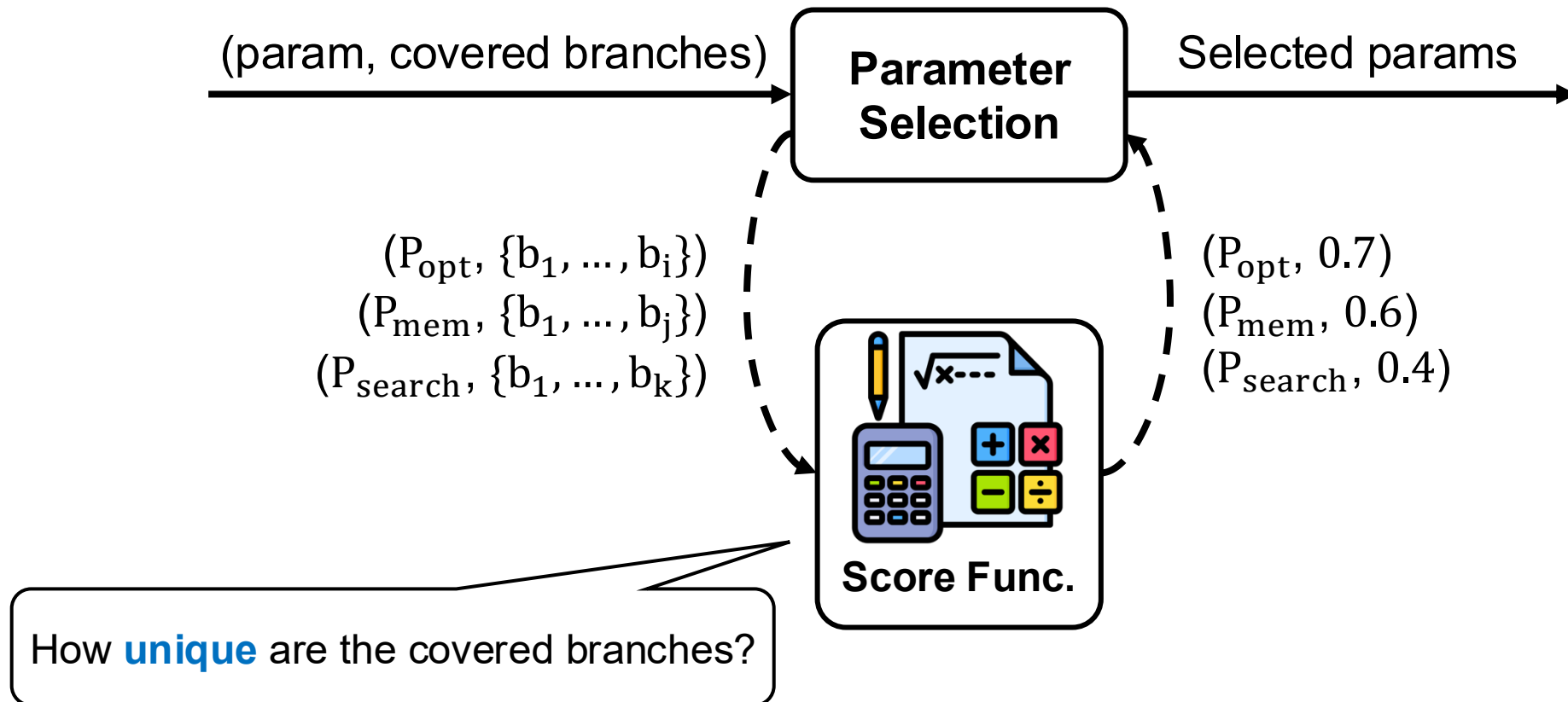
Parameter Selection Stage

- Compute the **score of each parameter** using the score function.



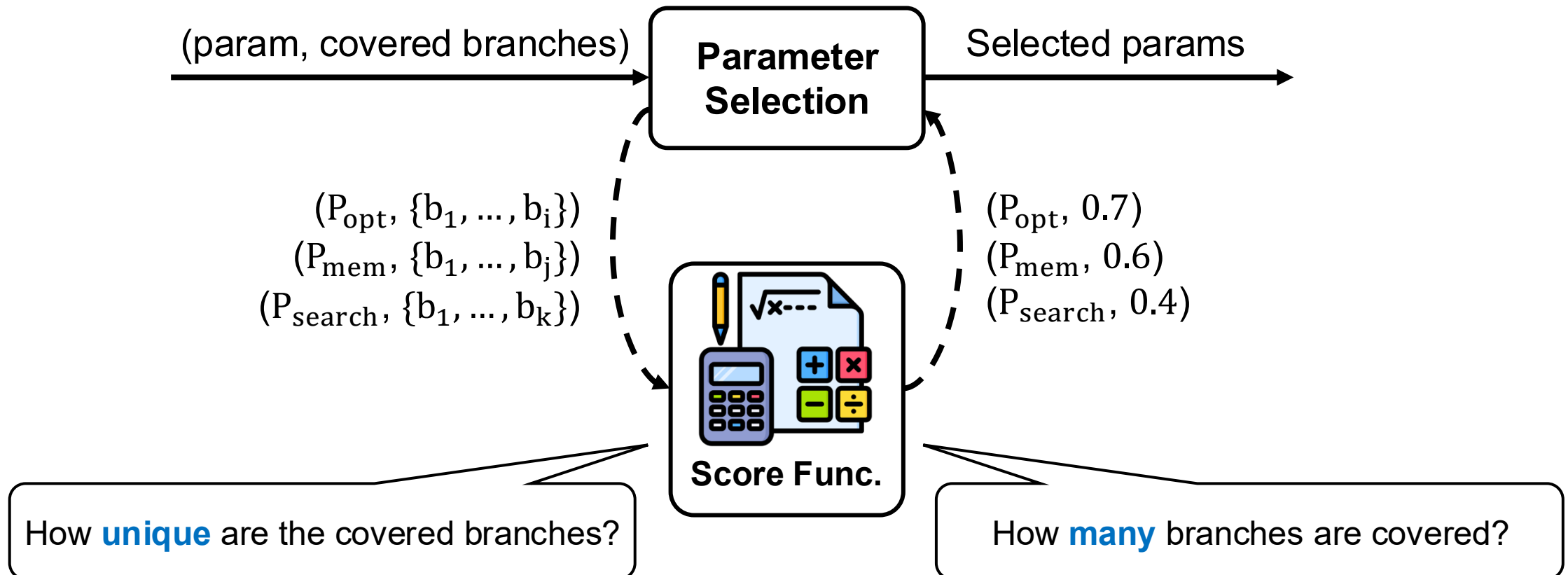
Parameter Selection Stage

- Compute the **score of each parameter** using the score function.



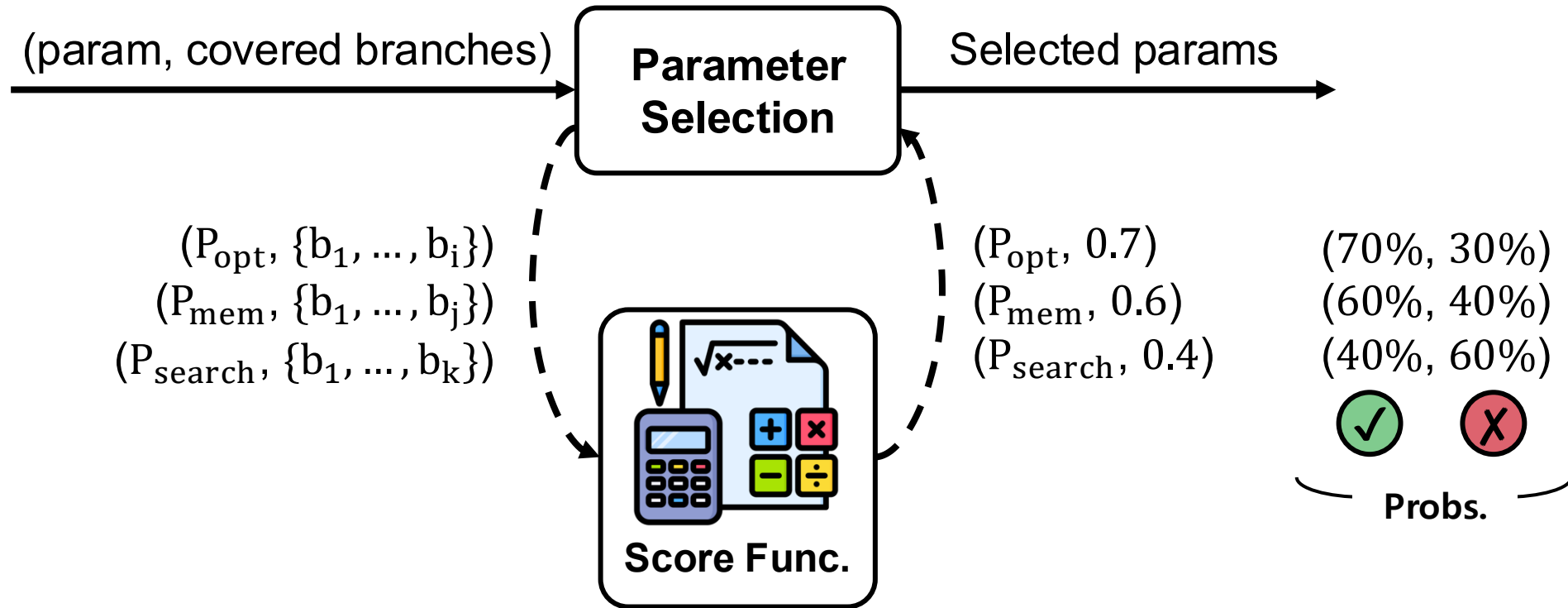
Parameter Selection Stage

- Compute the **score of each parameter** using the score function.



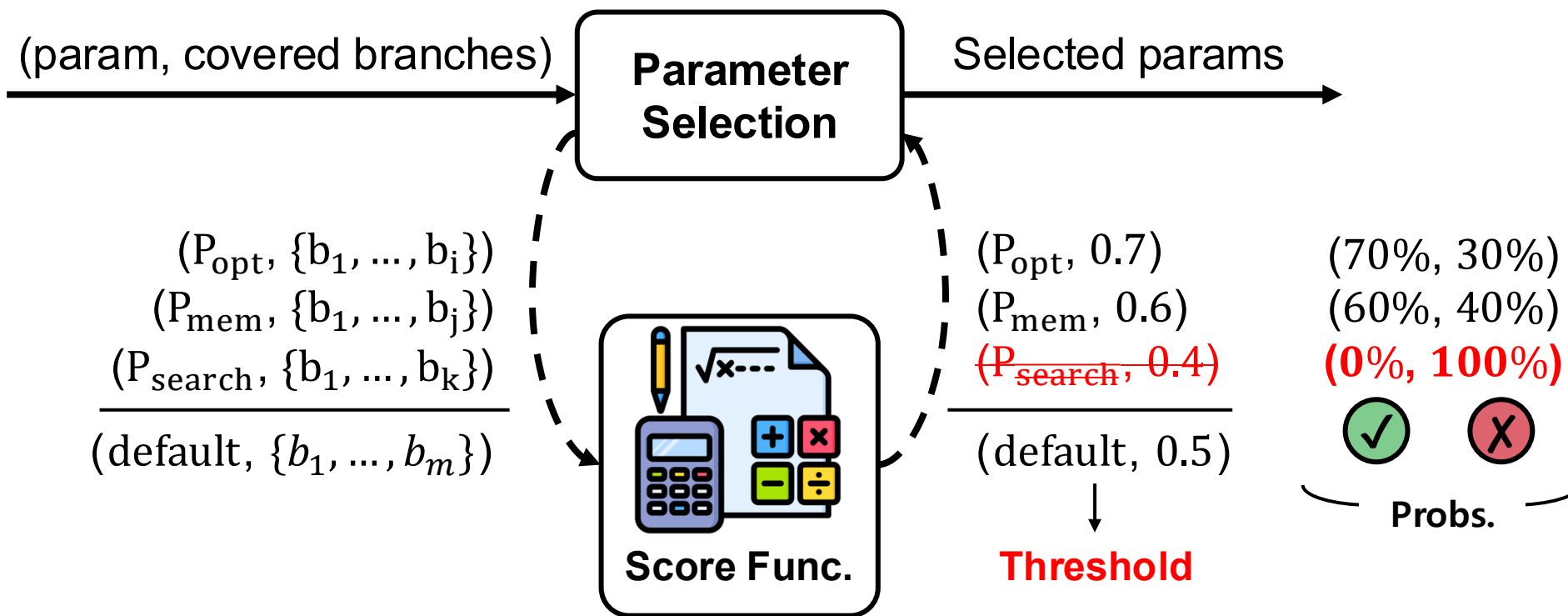
Parameter Selection Stage

- Select parameters **probabilistically** based on their scores



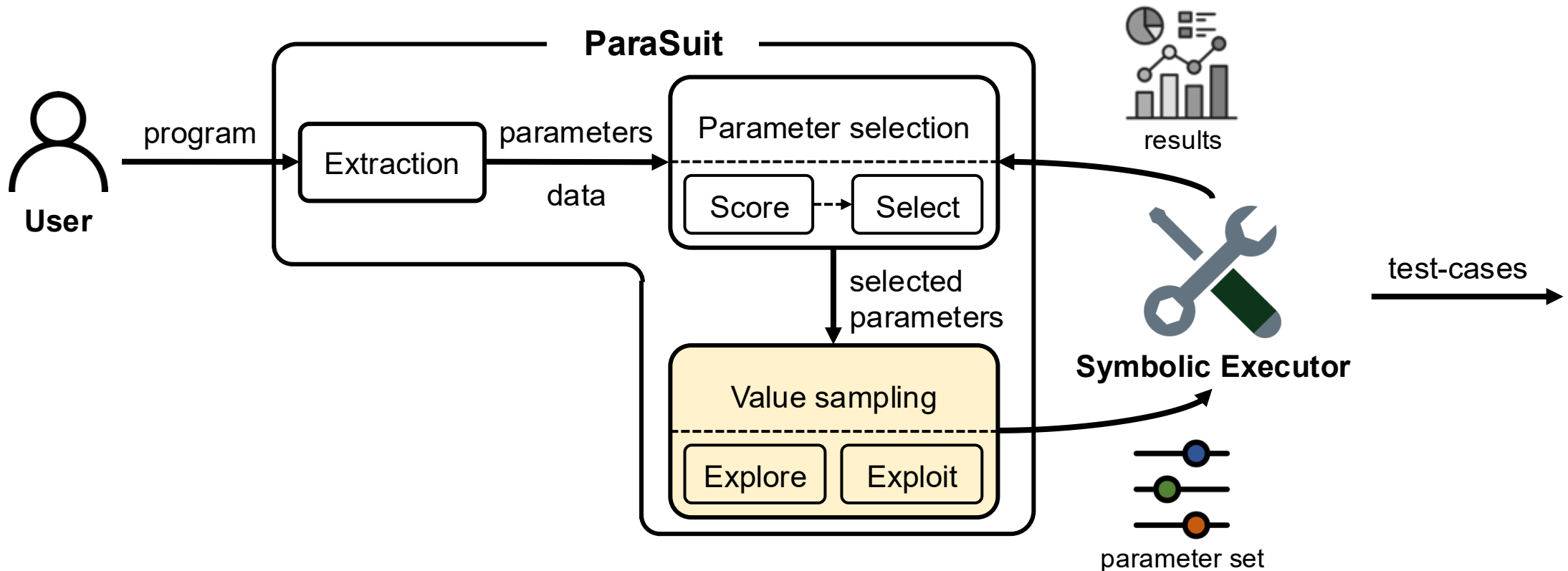
Parameter Selection Stage

- Give **0%** selection probability to parameters with scores **lower than the default set**



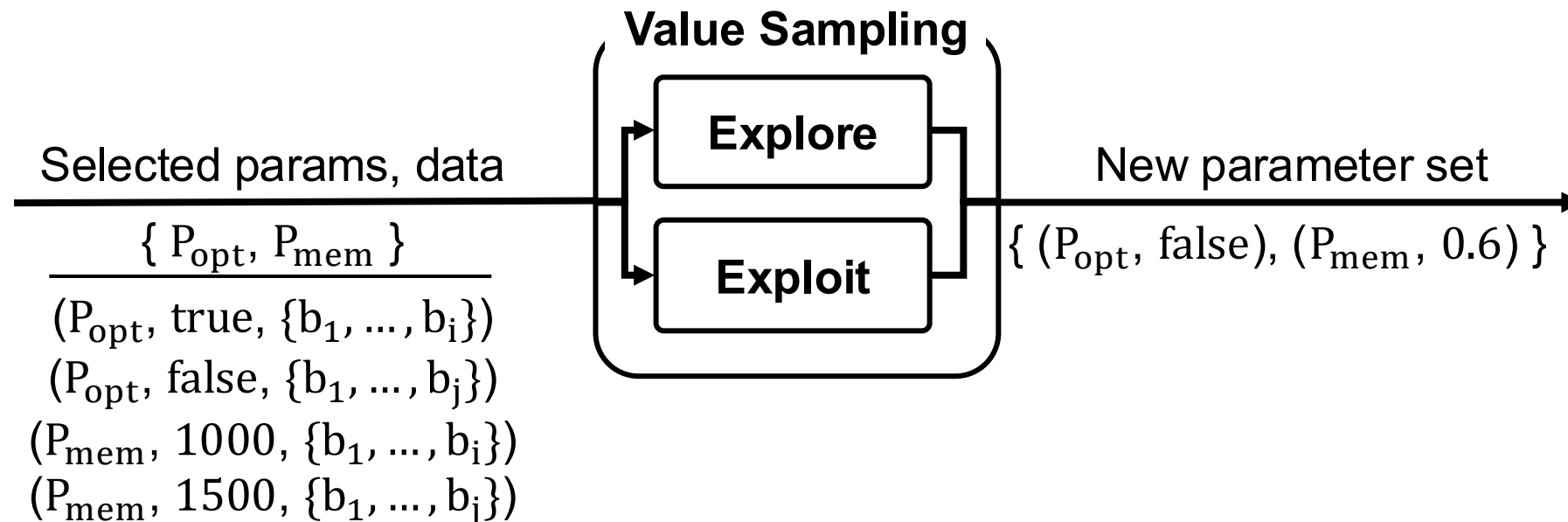
Value Sampling Stage

- Sample the **most appropriate values** for the selected parameters



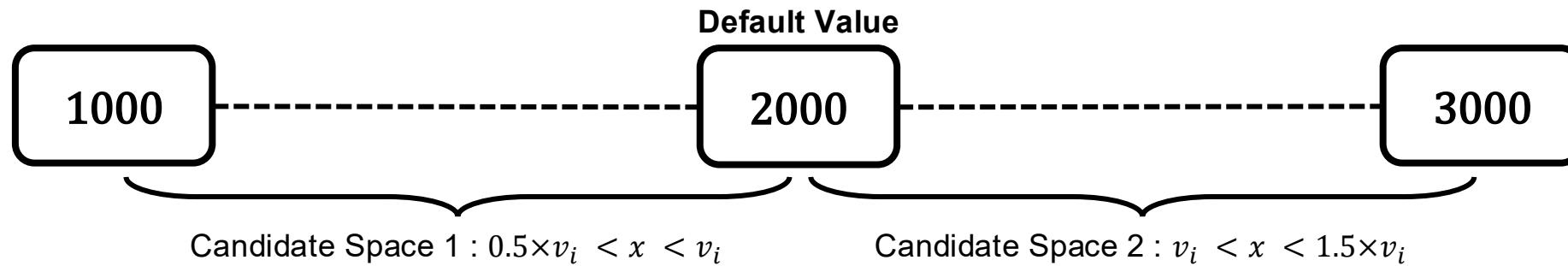
Value Sampling Stage

- Sample the **most appropriate values** for the selected parameters
- Utilizes two policies: Explore / Exploit
 - Explore : Try **diverse values** for each parameter
 - Exploit : Sample a new value that is either **the most effective one or close to it**



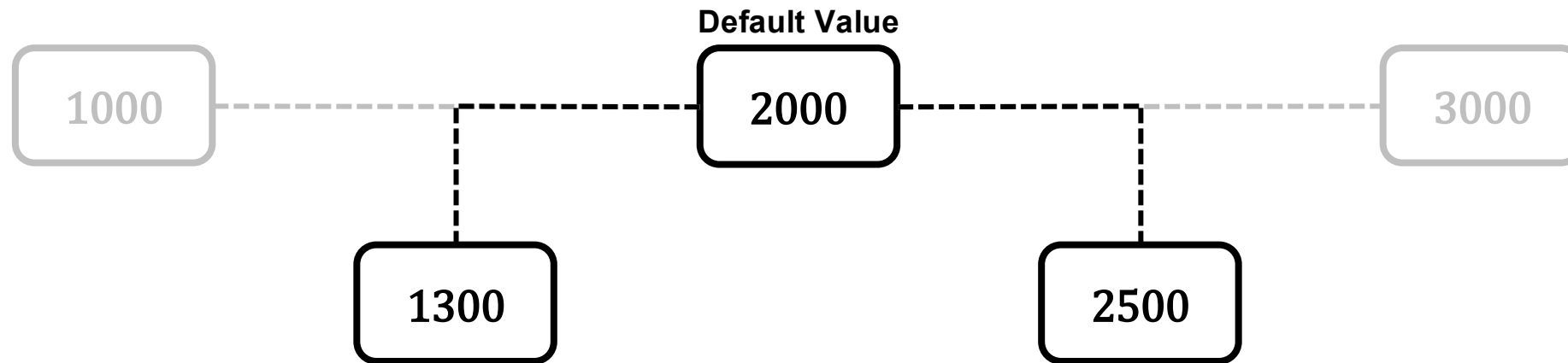
Value Sampling Stage

- Explore: Try **diverse values** for each parameter
 - Perform greedy sampling based on the **default value**



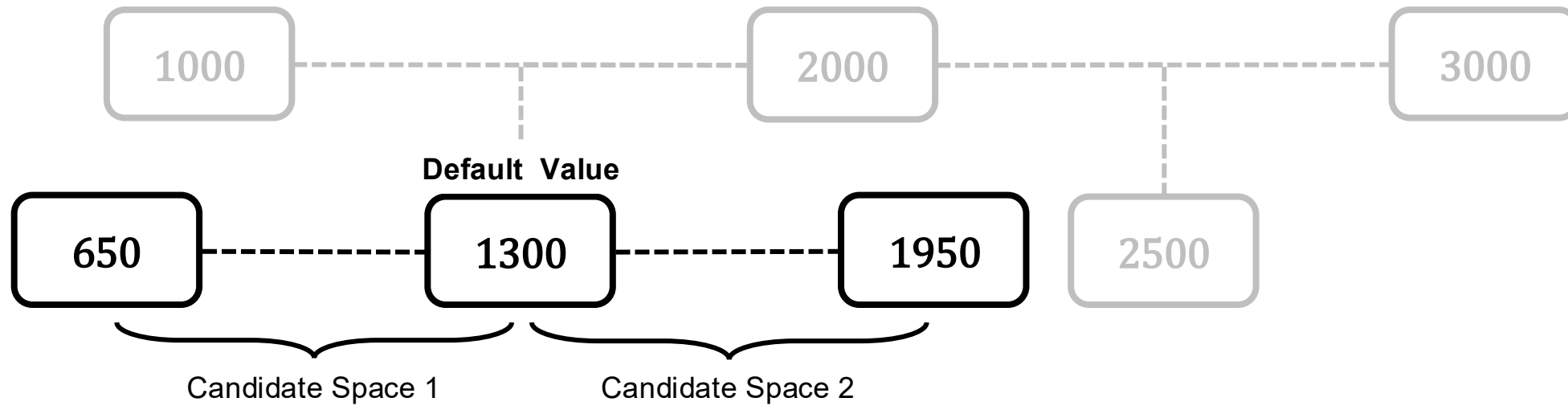
Value Sampling Stage

- Explore: Try **diverse values** for each parameter
 - Perform greedy sampling based on the **default value**



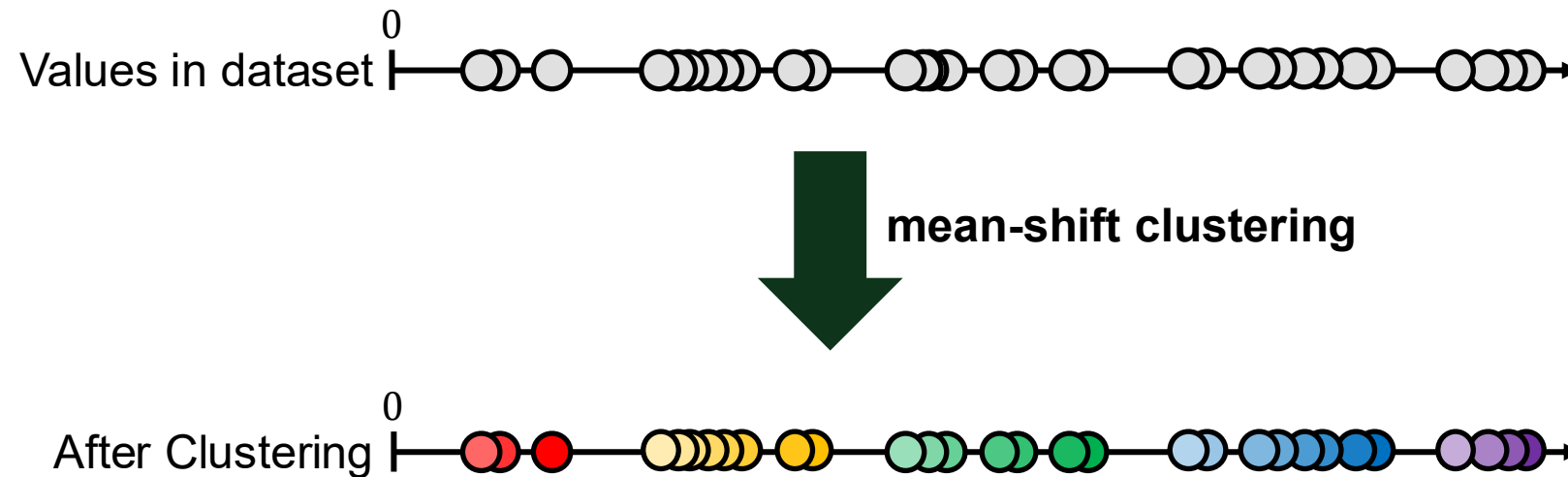
Value Sampling Stage

- Explore: Try **diverse values** for each parameter
 - Perform greedy sampling based on the **default value**



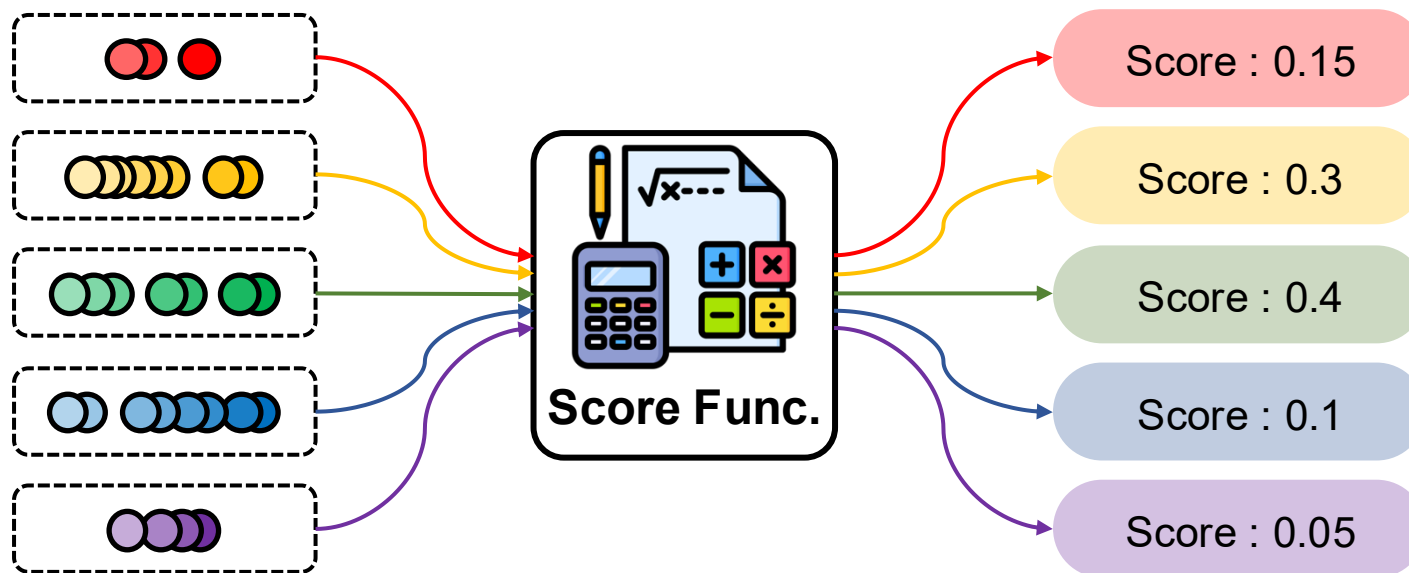
Value Sampling Stage

- Exploit: Sample a new value that is either **the most effective one or close to it**
 - Perform value sampling using **clustering**



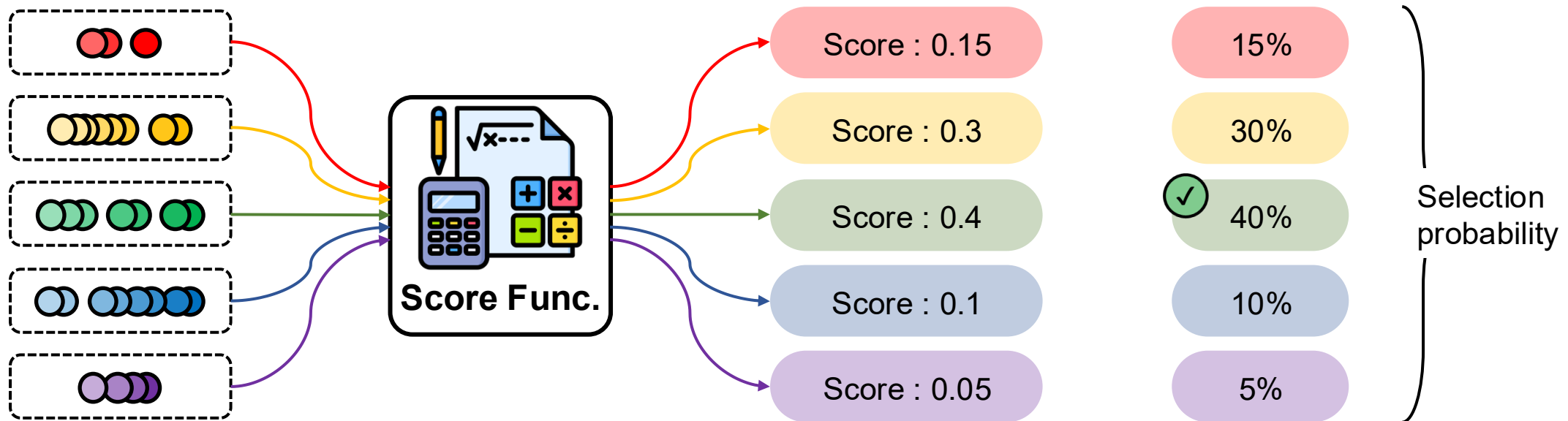
Value Sampling Stage

- Exploit: Sample a new value that is either **the most effective one or close to it**
 - Perform value sampling using **clustering**



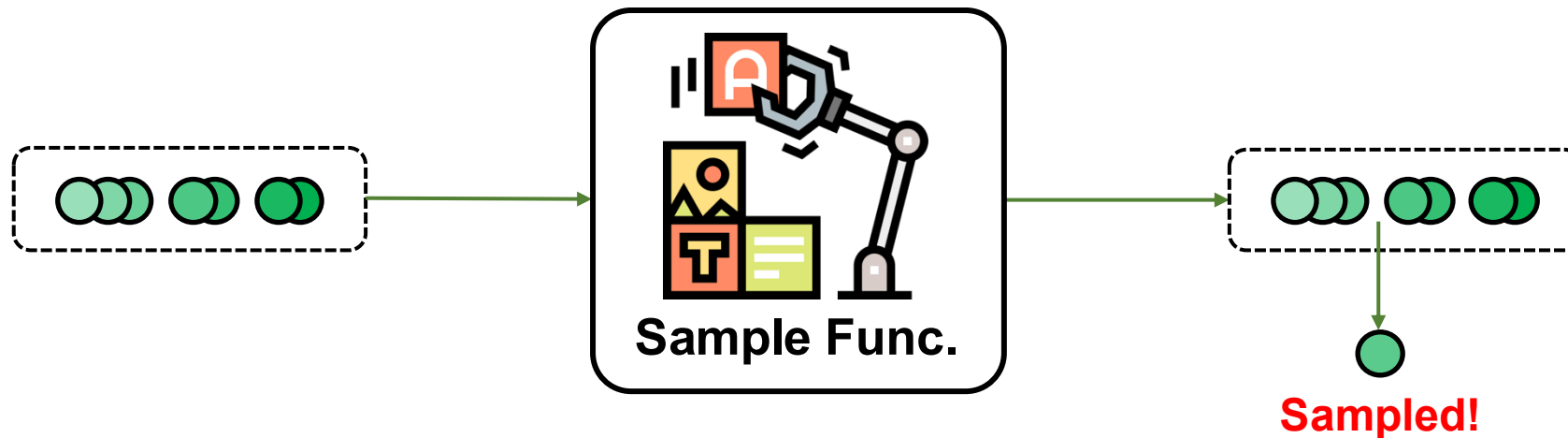
Value Sampling Stage

- Exploit: Sample a new value that is either **the most effective one or close to it**
 - Perform value sampling using **clustering**



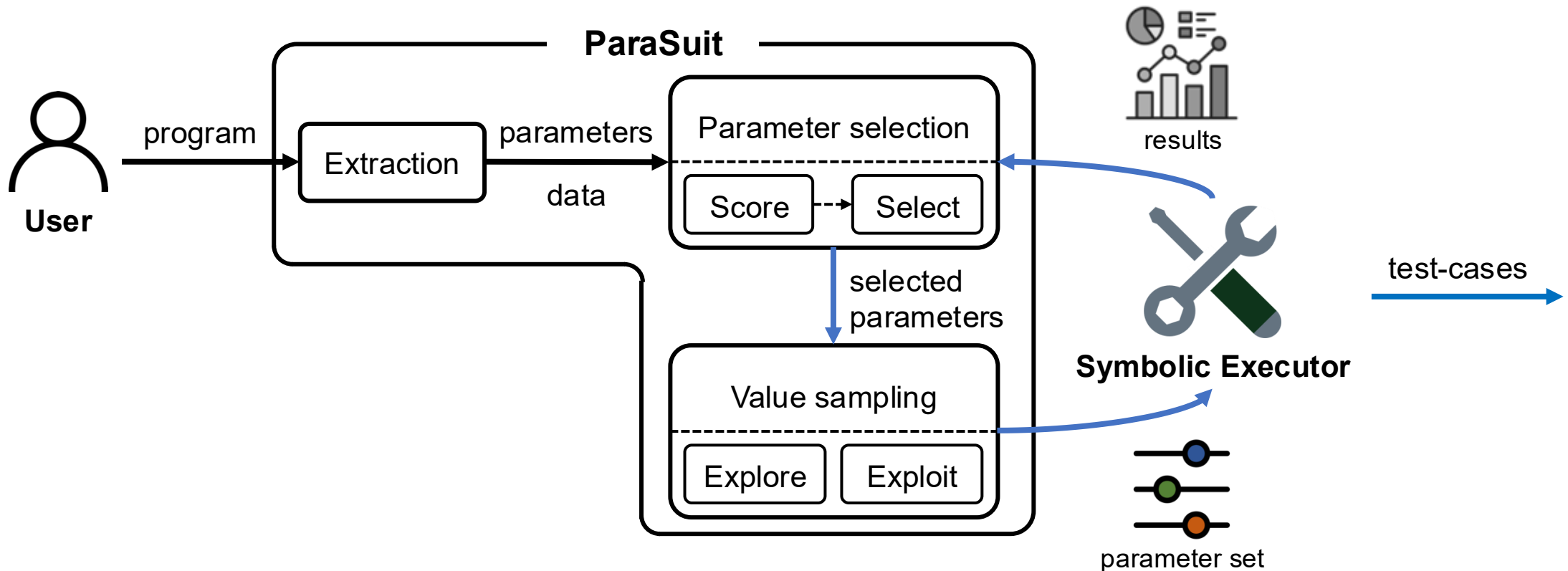
Value Sampling Stage

- Exploit: Sample a new value that is either **the most effective one or close to it**
 - Perform value sampling using **clustering**
 - Sample function: Sample a new value close to the most effective value **within the cluster**.



Iterative Process of ParaSUIT

- When the time budget is exhausted,
- ParaSUIT **outputs all test-cases** generated across the three stages



●Baselines

- SymTuner: **Semi-automatic** parameter tuning technique (ICSE'22)
- KLEE_{fix}: Use the parameter set provided by the KLEE developers **without tuning**
- RandSuit: **Randomly select** parameters and sample their values using **only the explore policy**
- SymTuner_{ALLP}: Define the search **space for all parameters** in KLEE and run SymTuner.

●Benchmarks : 12 GNU open-source C programs

●Operating system : Ubuntu-18.04

●Symbolic execution tool : KLEE-2.1

Experimental Results

● A **significant increase** in branch coverage

- ParaSuit > SymTuner (+25.8%)
- ParaSuit > KLEE_{fix} (+97.4%)
- ParaSuit > RandSuit (+46.8%)
- ParaSuit > Symtuner_{AllP} (+95.6%)

Programs	ParaSuit	SymTuner	KLEE _{fix}	RandSuit	Symtuner _{AllP}
xorriso-1.5.2	6,109.4	2,956.8	2,014.8	2,272.8	2,117.6
gcal-4.1	4,766.0	3,907.0	2,003.2	3,234.6	1,825.0
grep-3.4	3,613.8	3,280.2	2,208.4	3,344.2	2,017.8
gawk-5.1.0	4,393.2	3,747.0	2,841.6	2,893.0	3,261.8
sed-4.8	2,462.6	2,357.8	1,097.0	1,840.8	1,267.6
nano-4.9	1,909.6	1,721.0	1,457.0	1,662.2	1,289.0
...					
total	31,878.8	25,340.4	16,149.4	21,714.2	16,296.0

- Found **11 bugs** across three benchmarks
 - Detected **all bugs** found by the baselines,
 - and additionally identified **four bugs missed by all baselines**.

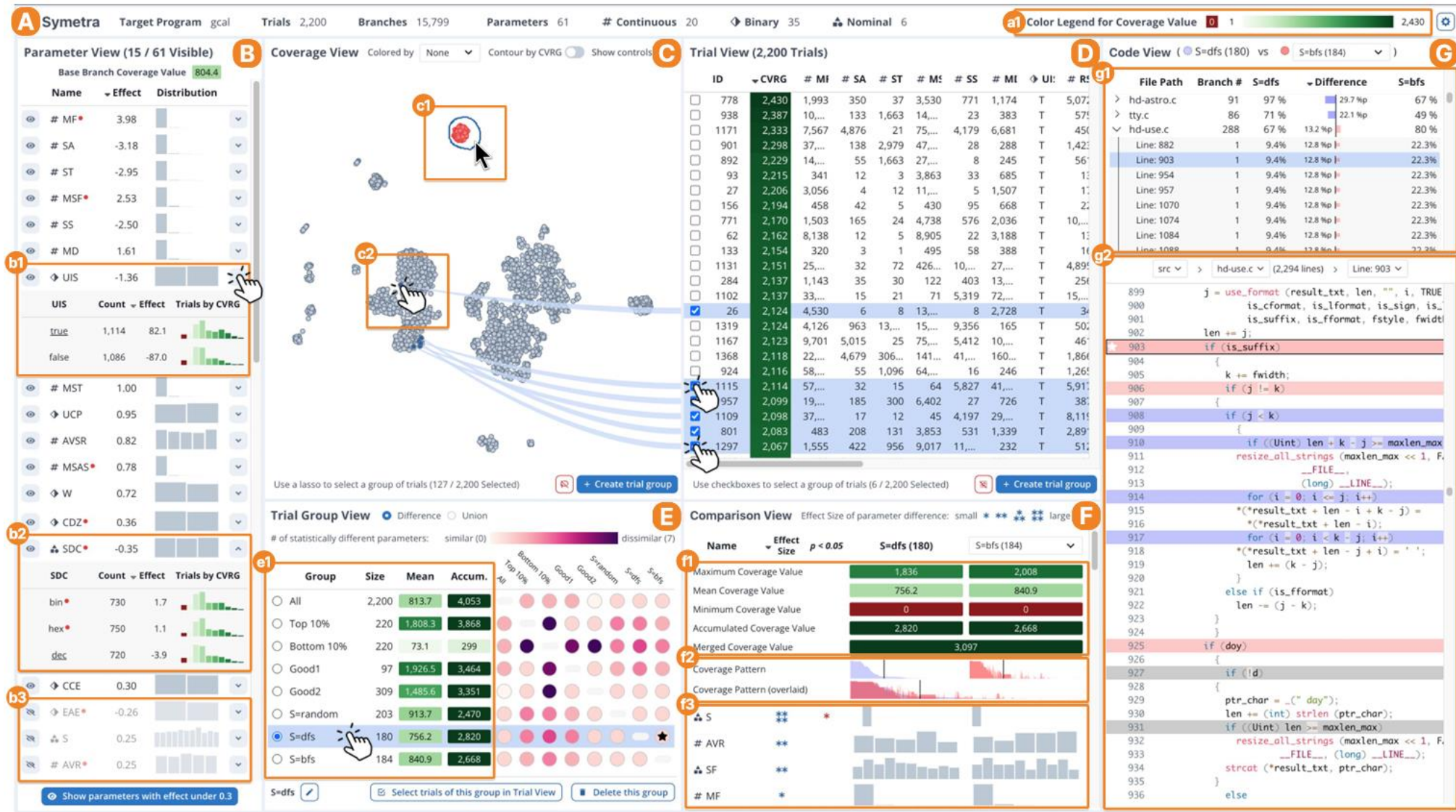
Programs	Error-Types	Error Location	ParaSuit	Symtuner	KLEE _{fix}	RandSuit	Symtuner _{ALLP}
gcal-4.1	Segmentation Fault	Line 740 in /src/file-io.c	✓	✓	✓	✓	✓
	Segmentation Fault	Line 72 in /libc/string/strncasecmp.c	✓	✓	✓	X	X
	Segmentation Fault	Line 19 in libc/stdio/fputc.c	✓	X	X	X	X
	Segmentation Fault	Line 14 in /libc/stdio/filen.c	✓	X	X	X	X
	Segmentation Fault	Line 3956 in /src/gcal.c	✓	✓	✓	X	X
	Segmentation Fault	Line 4934 in /src/gcal.c	✓	X	X	X	X
gawk-5.1.0	Abmornal Termination	Line 29 in /libc/string/memcpy.c	✓	✓	✓	✓	✓
	Abmornal Termination	Line 1337 in main.c	✓	✓	X	✓	X
combine-0.4.0	Segmentation Fault	Line 385 in /src/field.c	✓	✓	✓	✓	✓
	Segmentation Fault	Line 18 in /libc/string/strlen.c	✓	X	X	X	X
	Abmornal Termination	Line 668 in /src/dstring.c	✓	✓	X	X	X
# Found Bugs			11	7	5	4	3

Comparing with Other Tuning Methods

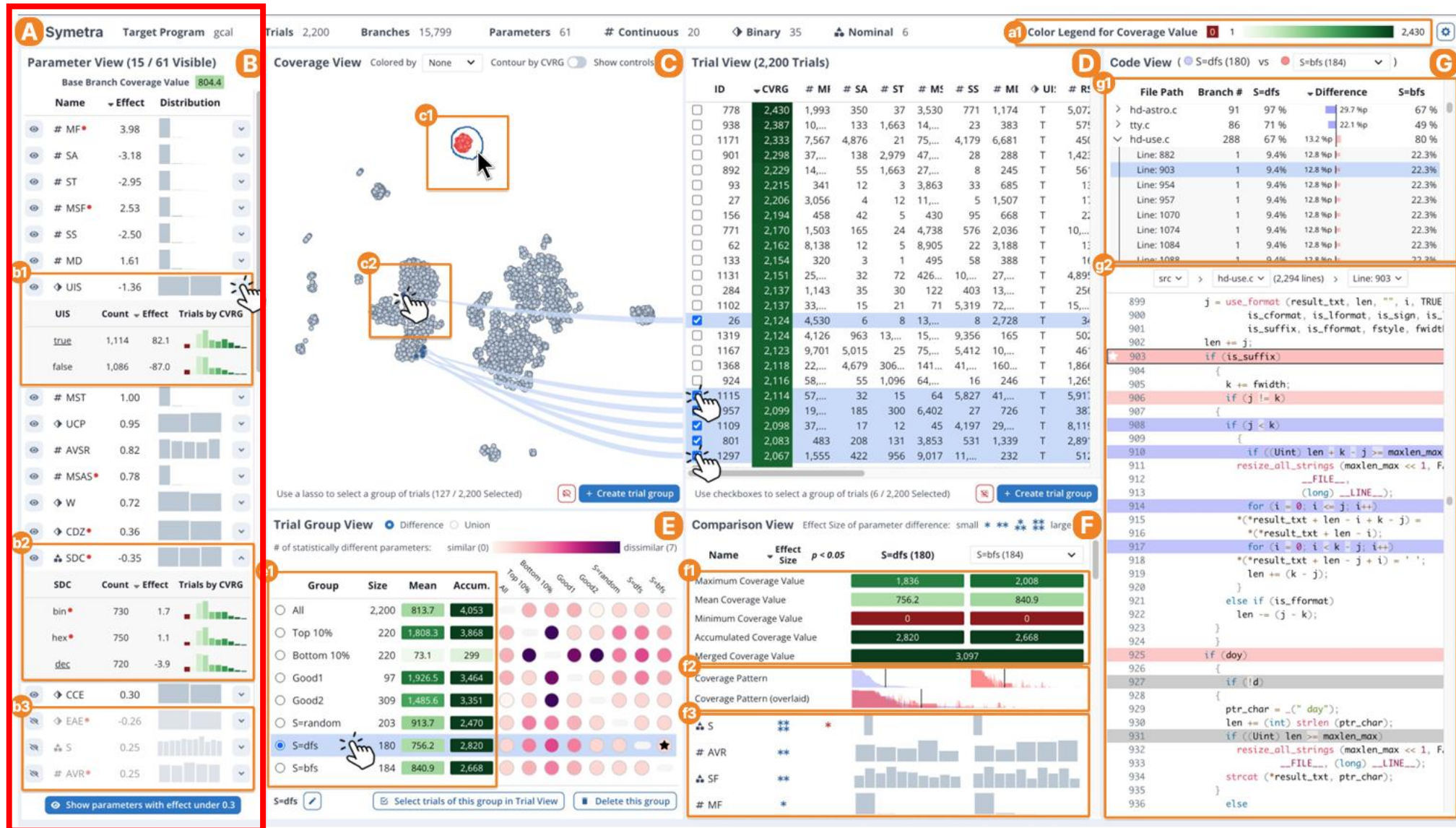
- Our value sampling algorithm **outperforms** other parameter tuning methods
 - Baselines: Bayesian Optimization(Bayesian), Genetic Algorithm(Genetic), TPE, CMA-ES
 - ParaSuit > Bayesian (+14.1% / +4 bugs)
 - ParaSuit > Genetic (+15.3% / +5 bugs)
 - ParaSuit > TPE (+10.8% / +5 bugs)
 - ParaSuit > CMA-ES (+12.5% / +3 bugs)

Programs	ParaSuit		Bayesian		Genetic		TPE		CMA-ES	
	Coverage	Bugs	Coverage	Bugs	Coverage	Bugs	Coverage	Bugs	Coverage	Bugs
xorriso-1.5.2	6,109.4	-	4,417.2	-	4,383.0	-	5,568.8	-	5,361.0	-
gcal-4.1	4,766.0	7	4,617.0	5	4,551.6	5	4,650.6	5	4,411.6	5
grep-3.4	3,613.8	-	3,559.2	-	3,348.8	-	3,474.2	-	3,455.4	-
gawk-5.1.0	4,393.2	1	4,116.0	1	3,902.4	-	3,929.6	-	3,801.2	1
sed-4.8	2,462.6	-	2,184.4	-	2,408.0	-	2,132.2	-	2,135.7	-
nano-4.9	1,909.6	-	1,889.8	-	1,945.3	-	1,873.8	-	1,917.0	-
...										
total	31,878.8	11	25,340.4	7	16,149.4	6	21,714.2	6	16,296.0	8

Visualizing Parameter-tuning Results



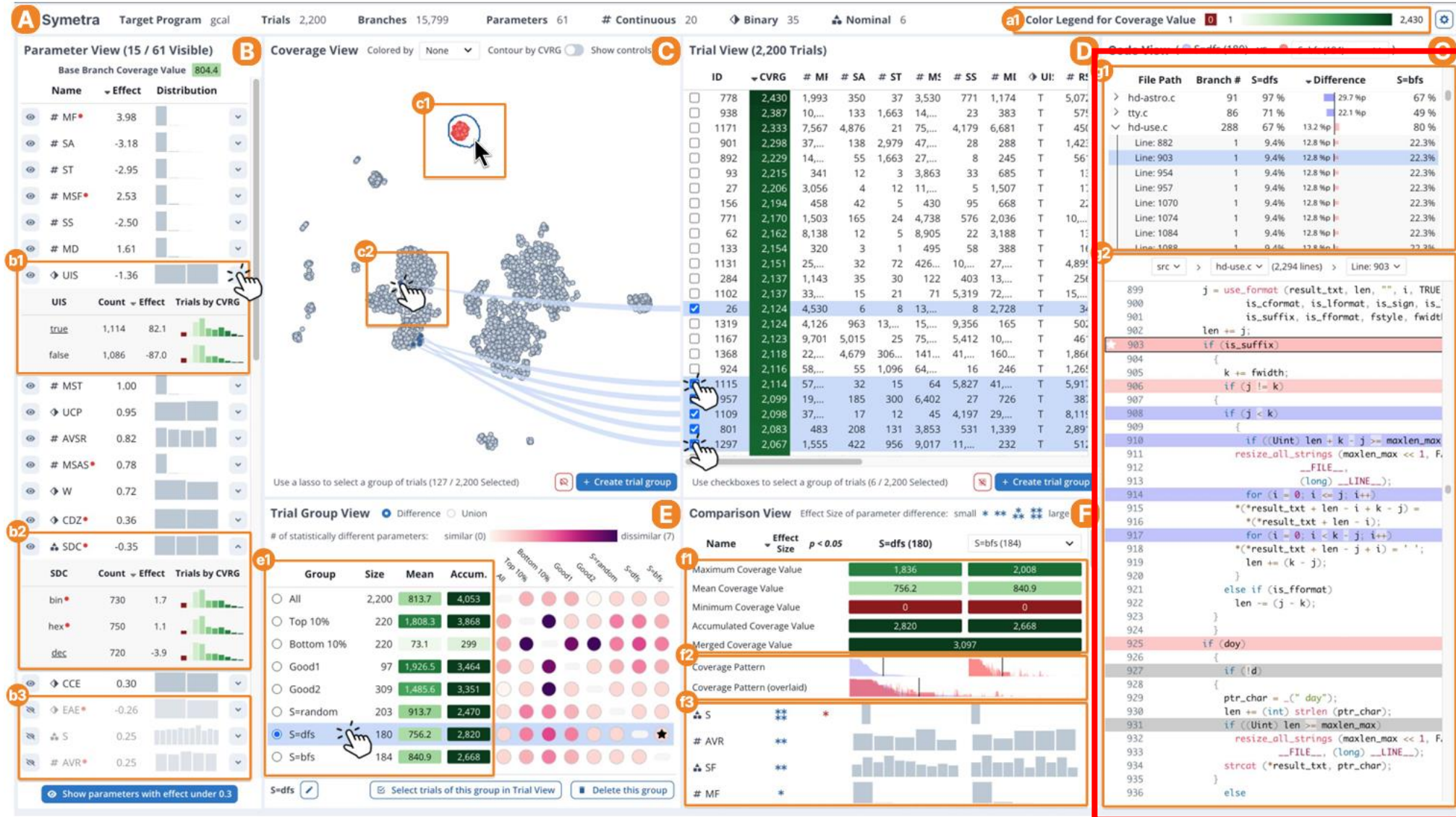
Visualizing Parameter-tuning Results



Visualizing Parameter-tuning Results



Visualizing Parameter-tuning Results



ParaSuit (ICSE'26)



Paper



Artifact

Symetra (EuroVis'26)



Paper



Artifact

Enhancing Symbolic Execution with Self-Configuring Parameters

Minjong Kim¹, Sooyoung Cha¹

😊 Thanks 😊

KLEE'26 @ Munich, Germany

