

FEATMAKER: Automated Feature Engineering for Search Strategy of Symbolic Execution

KLEE'26 @ Munich, Germany

Jaehan Yoon, Sooyoung Cha

Sungkyunkwan University, South Korea

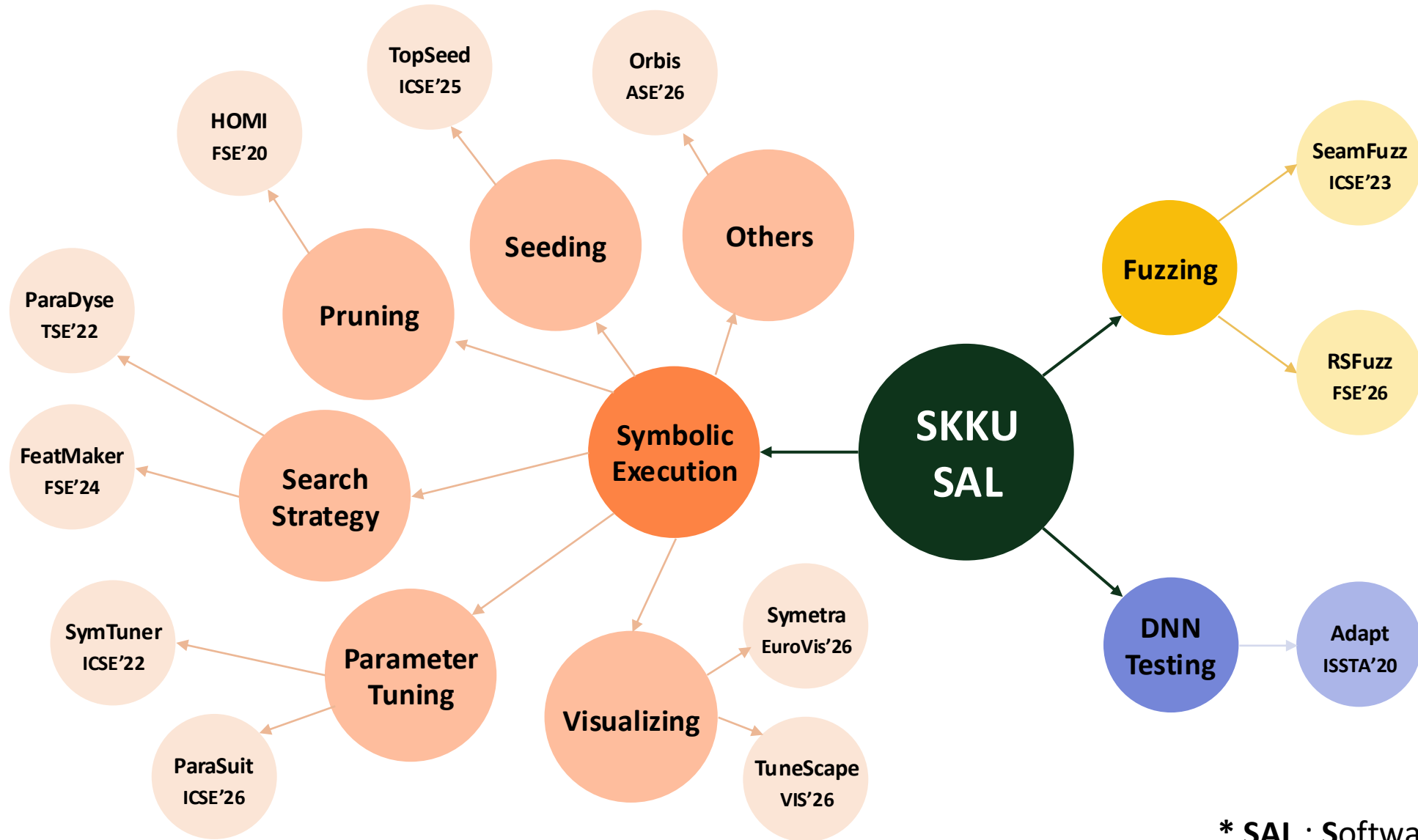




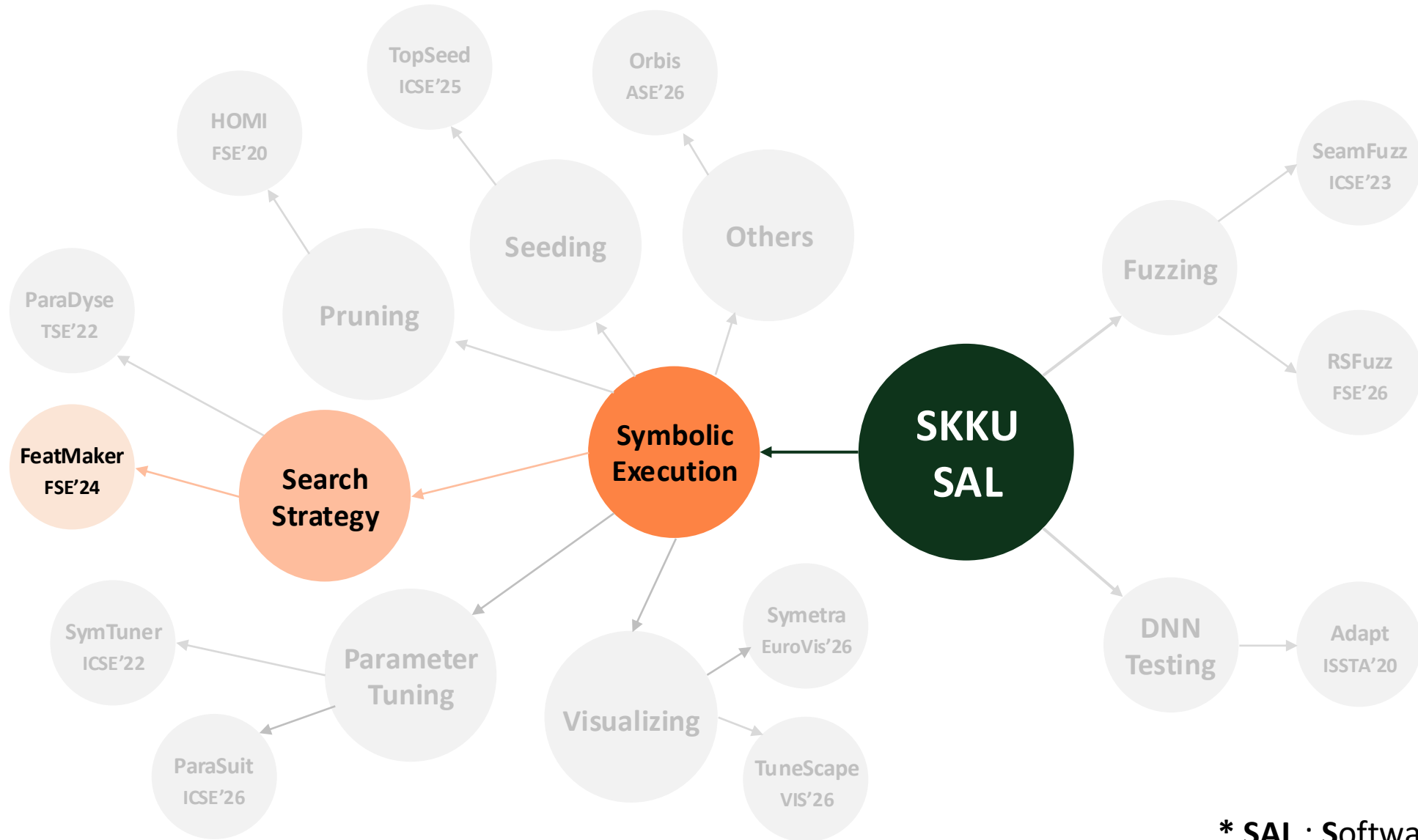
Sungkyunkwan University (SKKU)





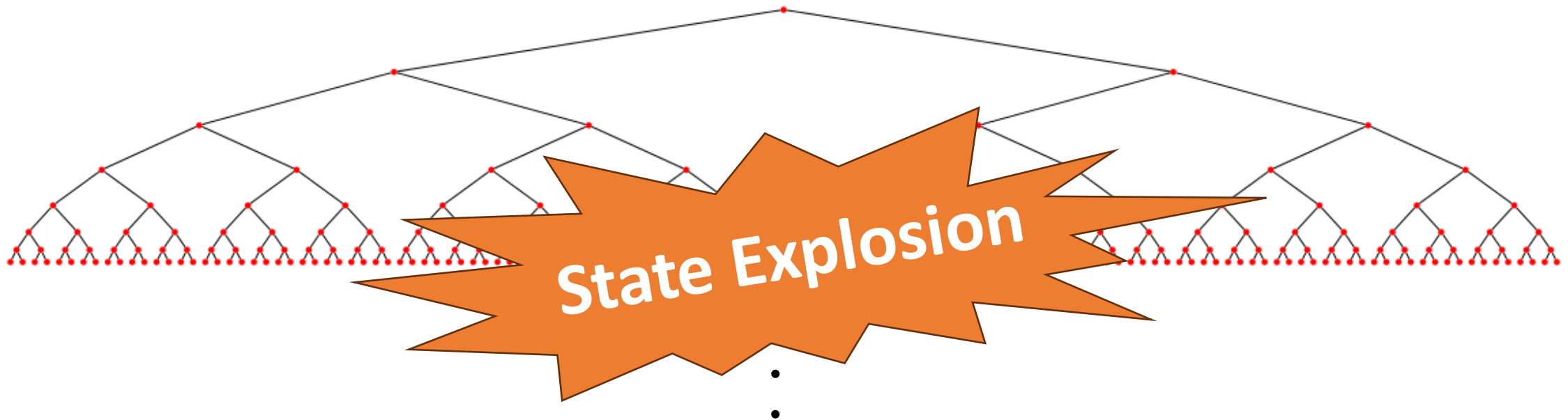


* SAL : Software Analysis Lab

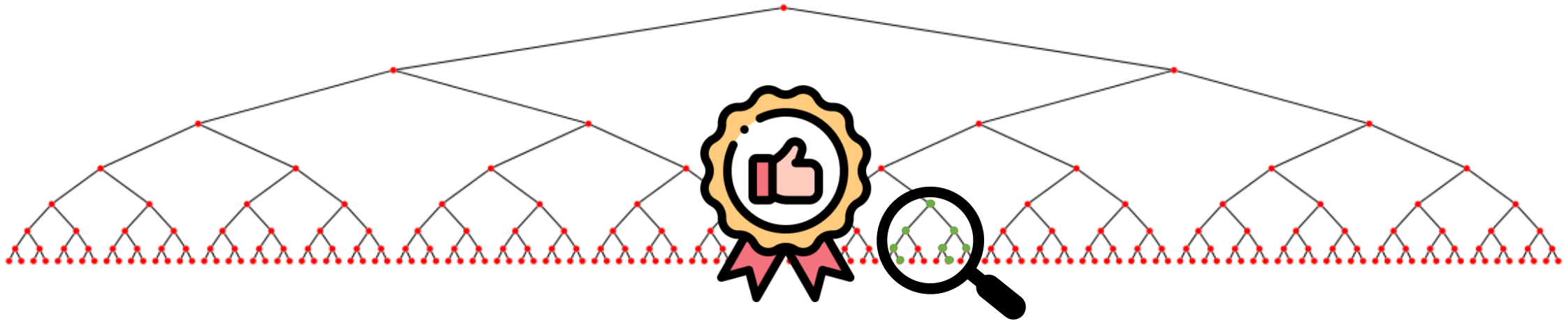


* **SAL** : Software Analysis Lab

- # of states increases **exponentially** with # of conditional statements.
 - ✓ grep-3.4 : 2^{8566} states
 - ✓ Executing all states is **impractical** for real-world programs.

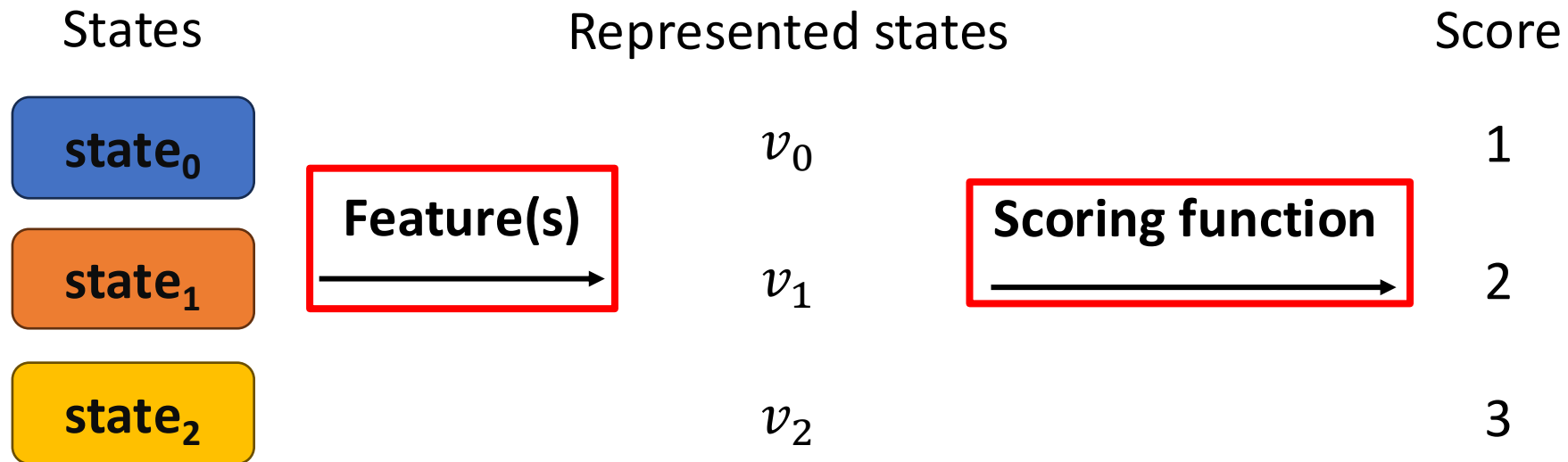


- # of states increases **exponentially** with # of conditional statements.
 - ✓ grep-3.4 : 2^{8566} states
 - ✓ Executing all states is **impractical** for real-world programs.
 - ✓ Identifying **promising states** is the most important issue.



- Strategies for symbolic execution.

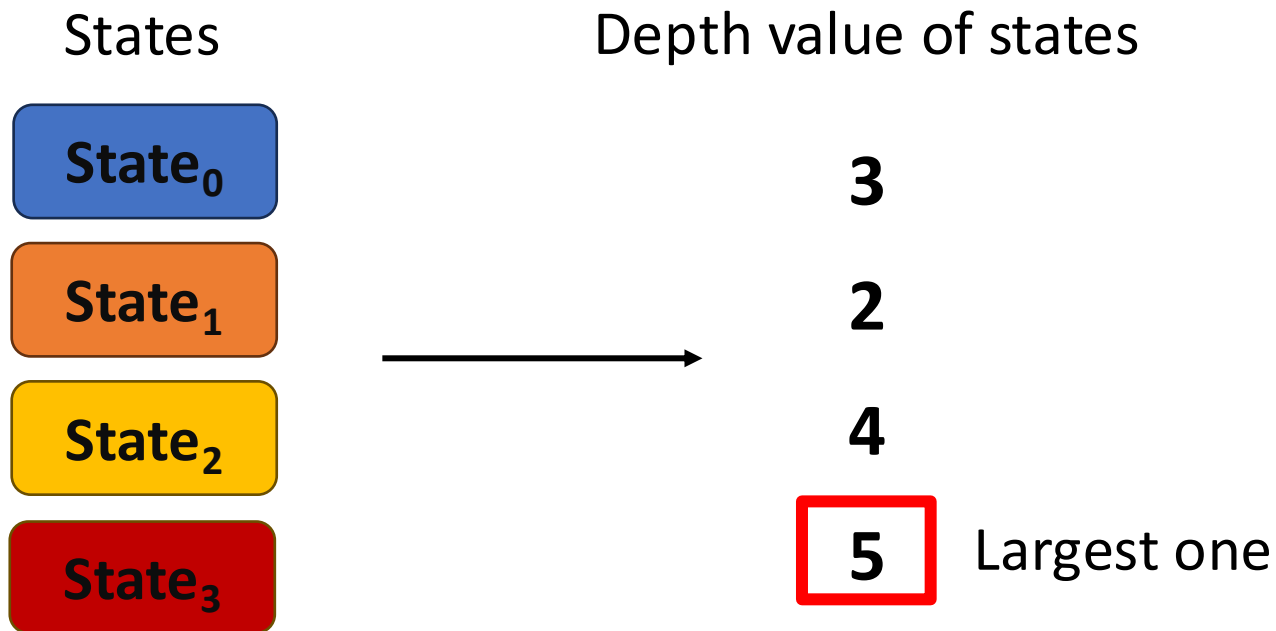
- ✓ **State feature** : represents states in the same format.
- ✓ **Scoring function** : prioritizes states.



- Strategies with a **single feature**.

- ✓ **Depth strategy** (OSDI 08)

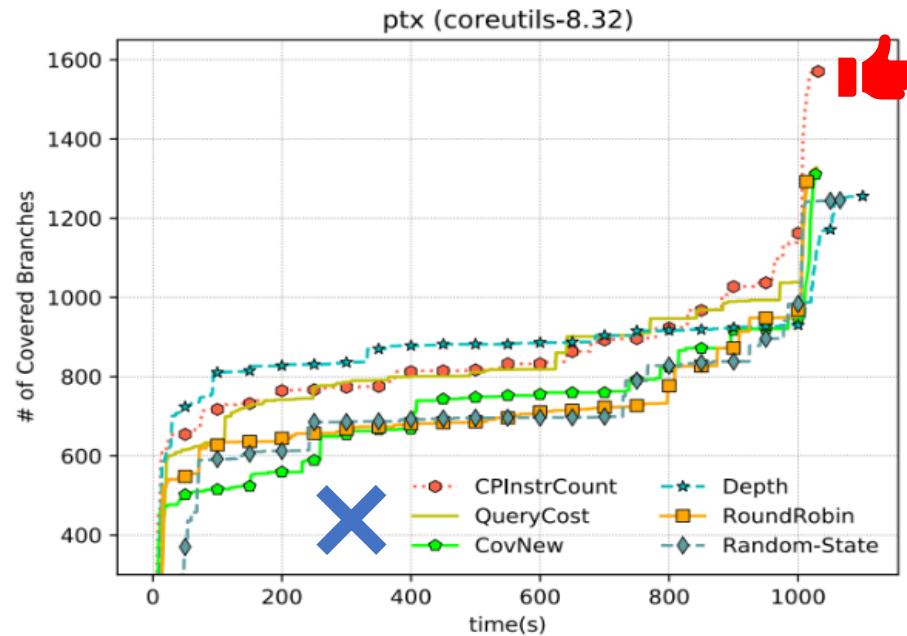
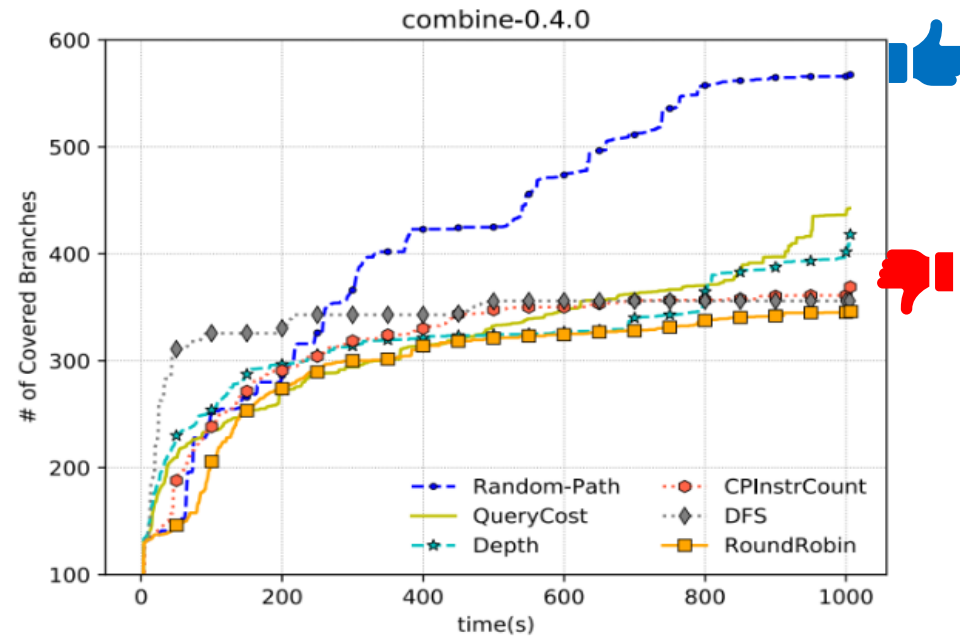
- A Feature : # of conditions exercised by each state
- A scoring function : larger one first



Existing strategies and limitations

● Limitation of strategies with a single feature.

- ✓ Same feature & same scoring function for all programs.
- ➔ Performance **varies** depending on the target program.

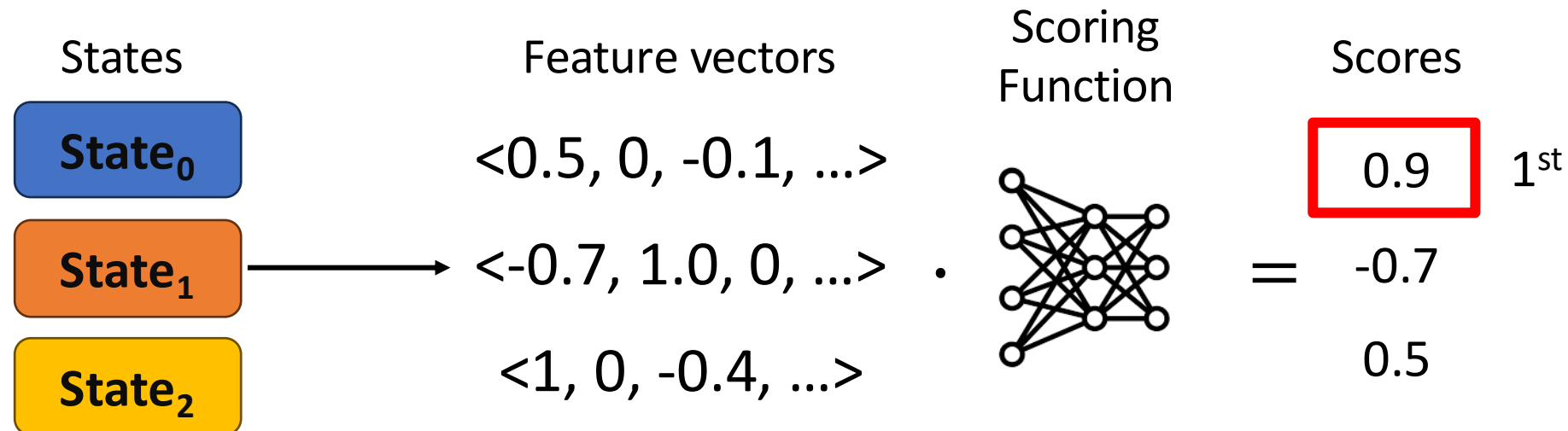


- Enhancing Dynamic Symbolic Execution by Automatically Learning Search Heuristics. (TSE 22)

Existing strategies and limitations

● Strategies with **multiple features**.

- ✓ Approaches to mitigate the limitation of a single feature.
- ✓ **LEARCH** (CCS 21)
 - Features : 10 features convert the state into a 47-dimension vector.
 - A scoring function : feed-forward neural network.



Existing strategies and limitations

- Limitation of strategies with multiple features.
 - ✓ Need to design all features **manually**.

Feature	Description
stack	Size of <i>state</i> 's current call stack.
successor	Number of successors of <i>state</i> 's current basic block.
testCase	Number of test cases generated so far.
coverage	(1) Number of new instructions covered by <i>state</i> 's branch. (2) Number of new instructions covered along <i>state</i> 's path. (3) Number of new source lines covered by <i>state</i> 's branch. (4) Number of new source lines covered along <i>state</i> 's path.
constraint	Bag-of-word representation of <i>state</i> 's path constraints.
depth	Number of forks already performed along <i>state</i> 's path.
cpicnt	Number of instructions visited in <i>state</i> 's current function.
icnt	Number of times for which <i>state</i> 's current instruction has been visited.
covNew	Number of instructions executed by <i>state</i> since the last time a new instruction is covered.
subpath	Number of times for which <i>state</i> 's subpaths [48] have been visited. The length of the subpaths can be 1, 2, 4, or 8.

10 features for Learch (CCS 21)

#	Description
1	10% states with the smallest number of executed instructions
2	10% states with the highest number of executed instructions
3	10% states with the smallest number of executed instructions since the last instruction was newly
	•
	•
	•
23	10% states with the shallowest loop depth in currently executing function
24	10% states with the deepest loop depth in currently executing function
25	10% states with the shallowest loop depth in the caller of currently executing function
26	10% states with the deepest loop depth in the caller of currently executing function

26 features for Paradyse (TSE 22)

- Enhancing Dynamic Symbolic Execution by Automatically Learning Search Heuristics. (TSE 22)

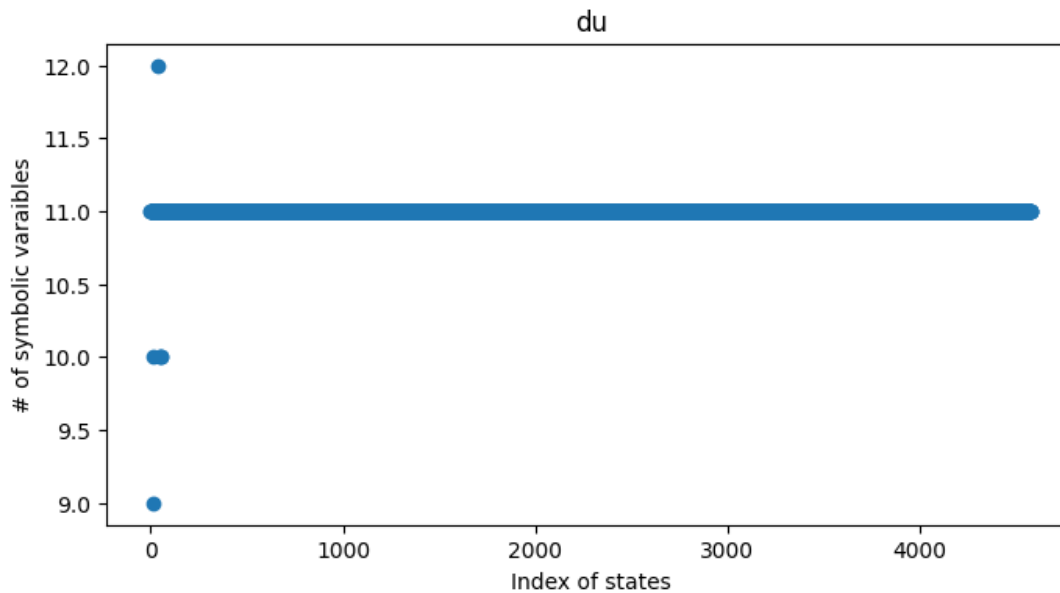
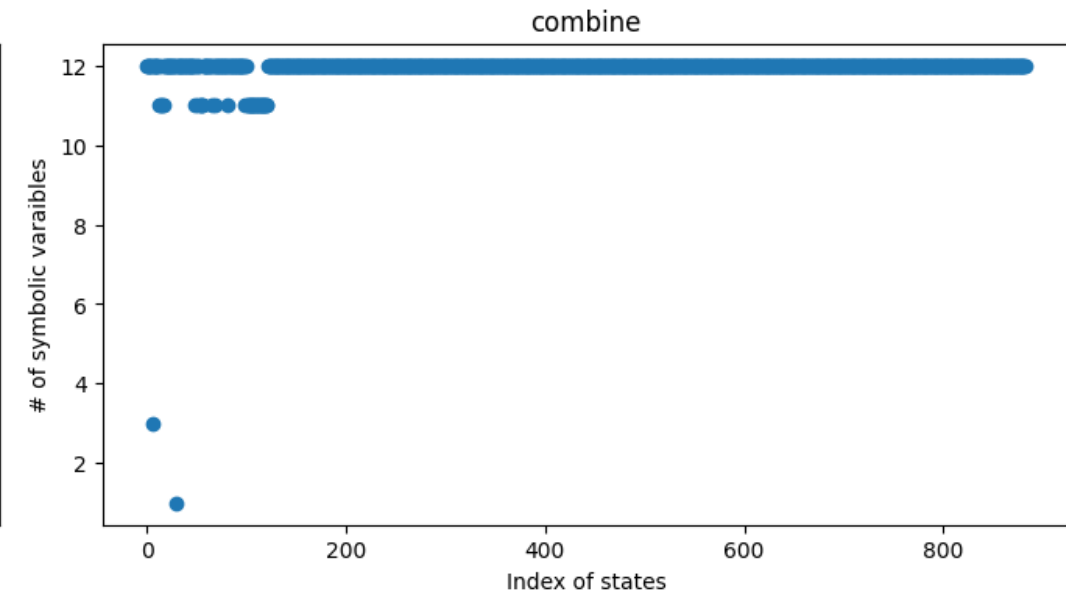
Existing strategies and limitations

● Limitation of strategies with multiple features.

✓ The quality of **manually designed** features is **not guaranteed**.

E.g.) Some features from Paradyse were **useless** for some programs.

- Most states have the same value.



- Limitation of strategies with multiple features.

- ✓ Designed features can't be used with **different symbolic execution tools**.
 - ✓ KLEE : Symbolic execution tool published in OSDI 08
 - ✓ CREST : Concolic testing tool published in ASE 08

26 features $\neq$ 40 features

TSE' 22 (26 features)

ParaDySE implemented on **KLEE**

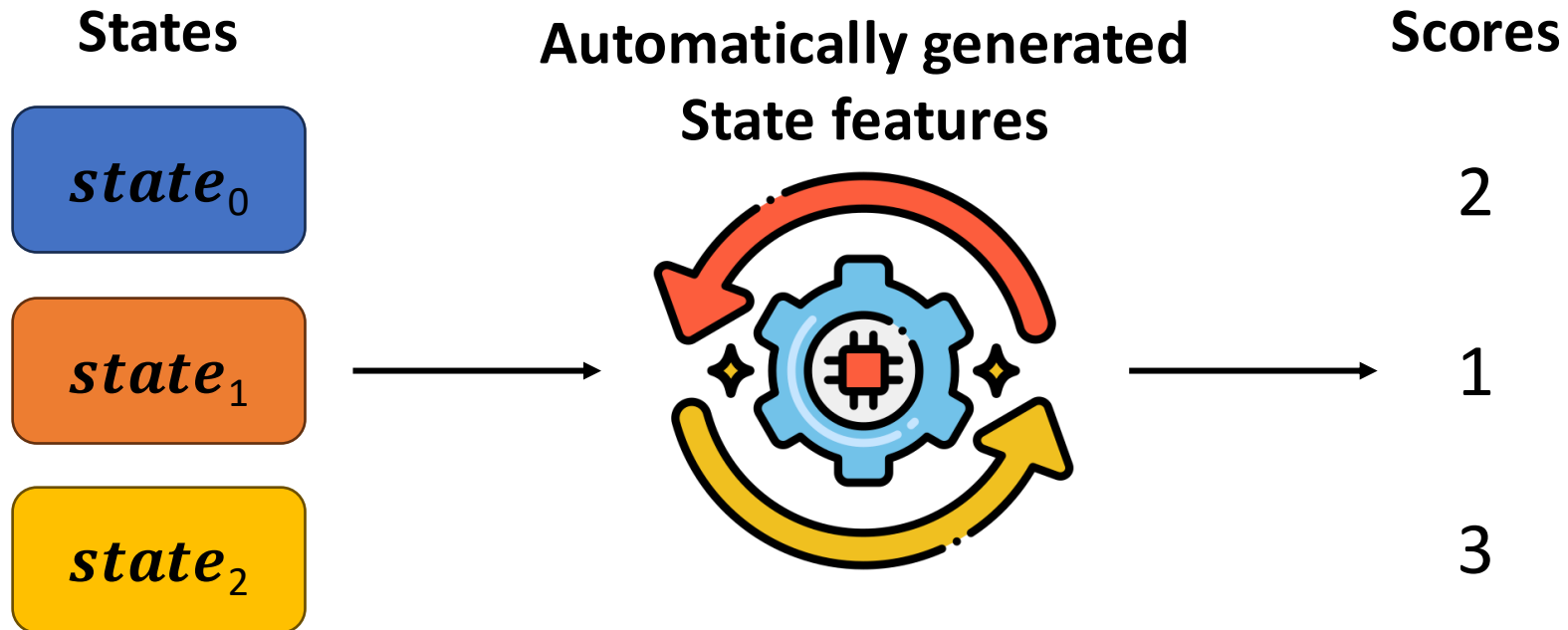
ICSE' 18 (40 features)

ParaDySE implemented on **CREST**

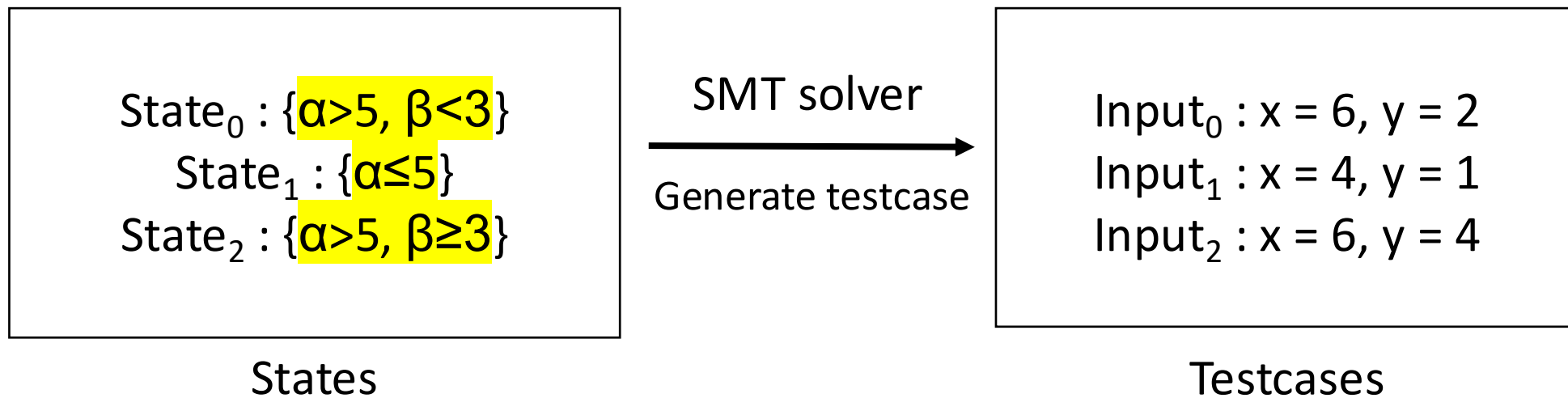
- KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs (OSDI 08)
- Heuristics for Dynamic Test Generation (ASE 08)

- **Automatic generation of state features from scratch.**

- ✓ Use only **general information** of symbolic execution tools **without prior knowledge**.



- Use **branch conditions** as state features.
 - ✓ The most important data to **determine the inputs of test cases**.
 - ✓ Most general information **maintained by all symbolic execution tools**.



- Use each branch condition as a **boolean feature**.
 - ✓ Whether a state has the condition in its branch conditions or not.

State₀ : { $\alpha > 5$, $\beta < 3$ }
State₁ : { $\alpha \leq 5$ }
State₂ : { $\alpha > 5$, $\beta \geq 3$ }

States

Features

$\langle \alpha > 5, \alpha \leq 5, \beta < 3, \beta \geq 3 \rangle$

*state*₀ < 1, 0, 1, 0 >

*state*₁ < 0, 1, 0, 0 >

*state*₂ < 1, 0, 0, 1 >

- Use **weight vector** as a **scoring function**.

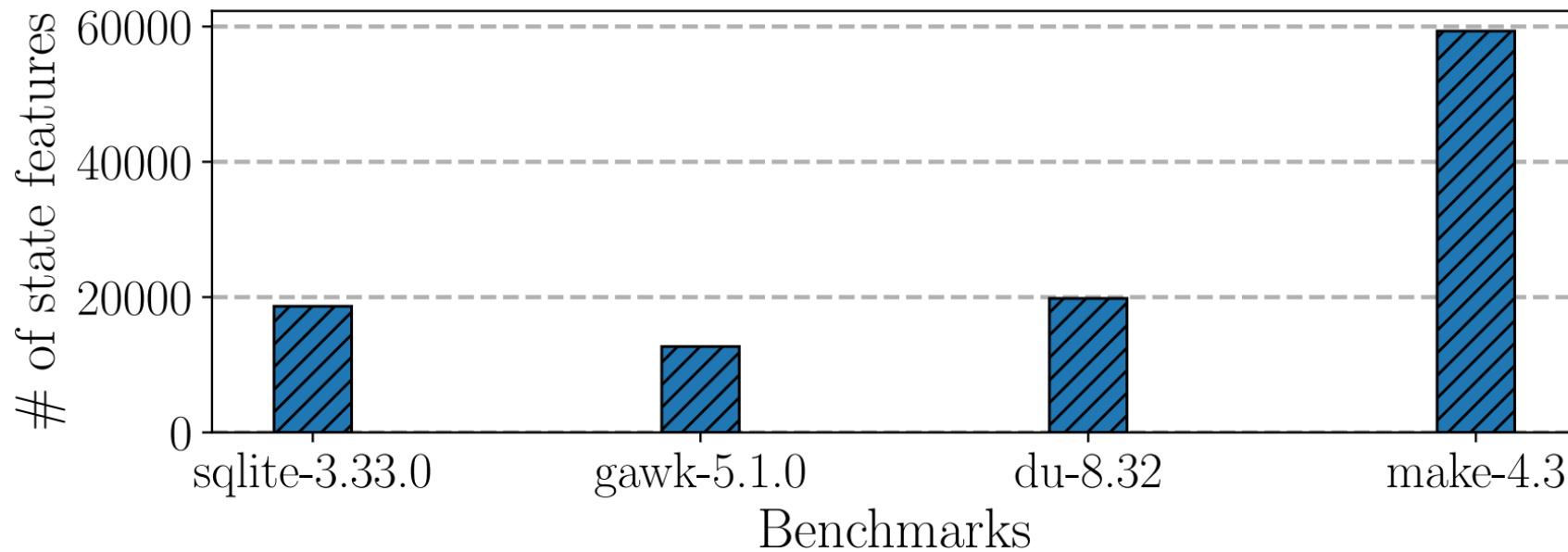
- ✓ Assign weight to each feature and **multiply with represented states** to get scores.

- E.g.) $\langle 0.5, 2.1, -1.7, 3 \rangle$ for **4** features.

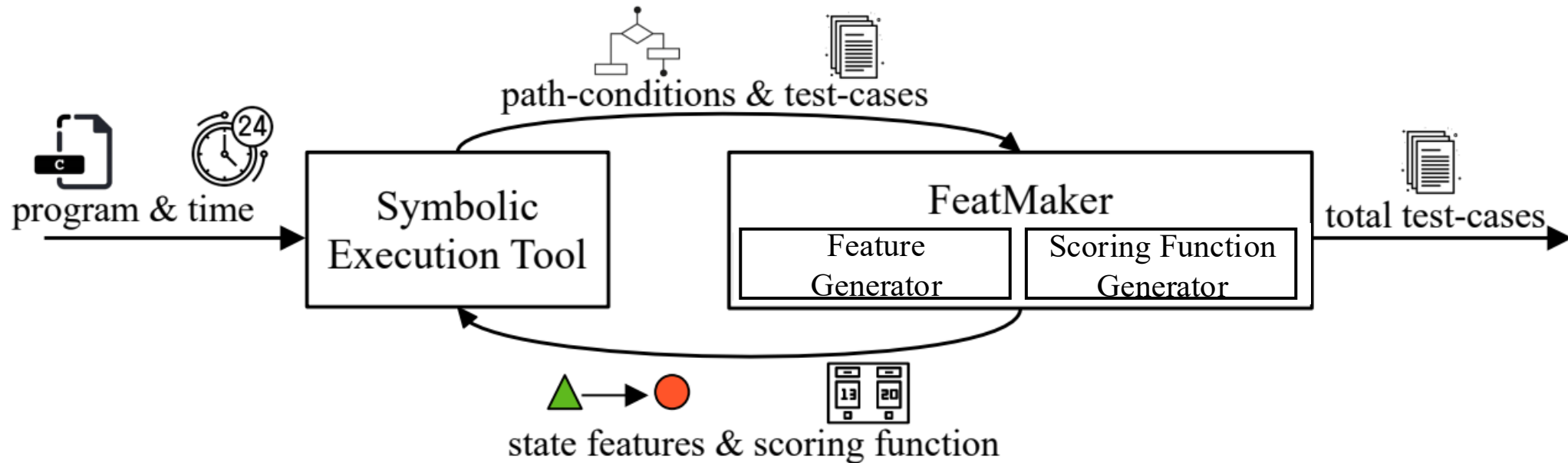
- ✓ Update weights with a **data-driven approach** in testing budget.

Feature vectors		weights	Scores	
$state_0$	$\langle 1, 0, 1, 0 \rangle$	0.5	.	-1.2
$state_1$	$\langle 0, 1, 0, 0 \rangle$	2.1		2.1
$state_2$	$\langle 1, 0, 0, 1 \rangle$	-1.7		3.5
		3		
			=	

- Collecting all conditions results in an **impractically large weight vector**.
 - ✓ **60,000** conditions are generated in 24h execution for make-4.3.
 - ✓ We should **retain only the necessary** branch conditions.

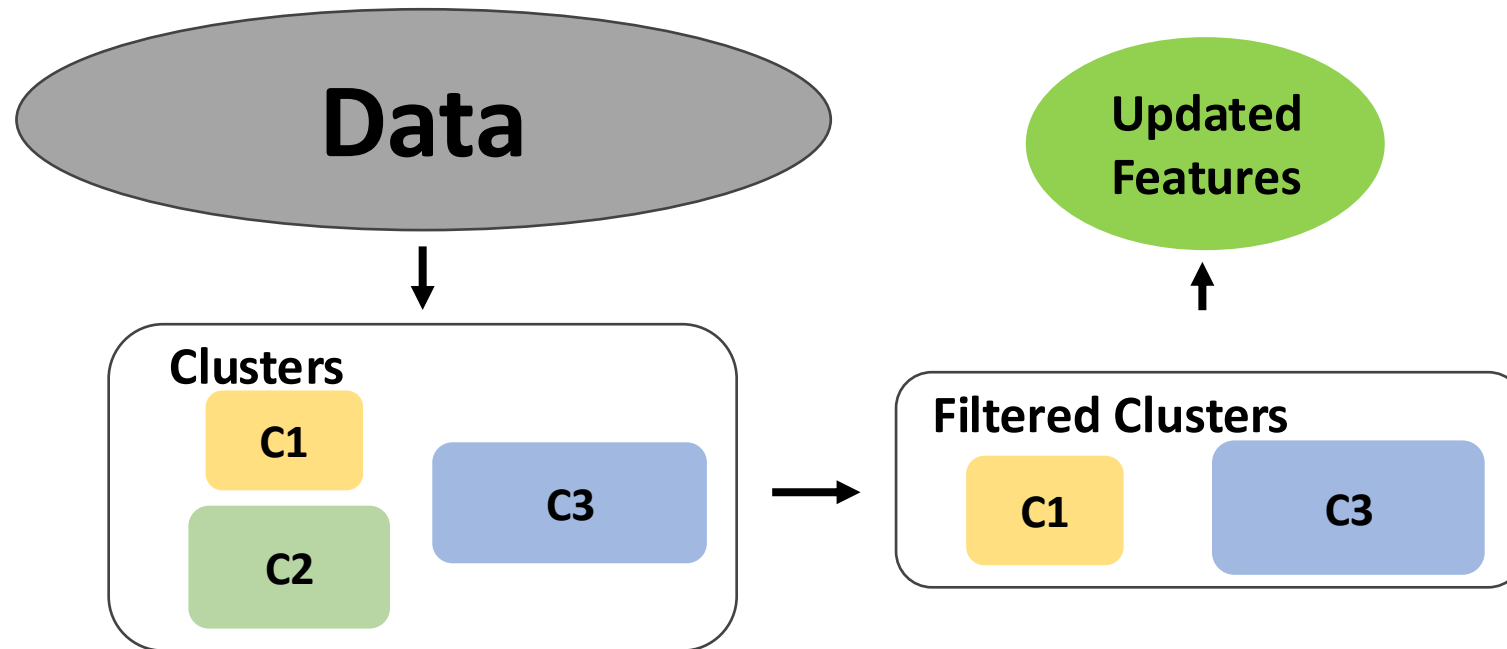


- **Iteratively update** state features & scoring function.
 - ✓ Collect **data** for updating through **multiple short symbolic executions**.



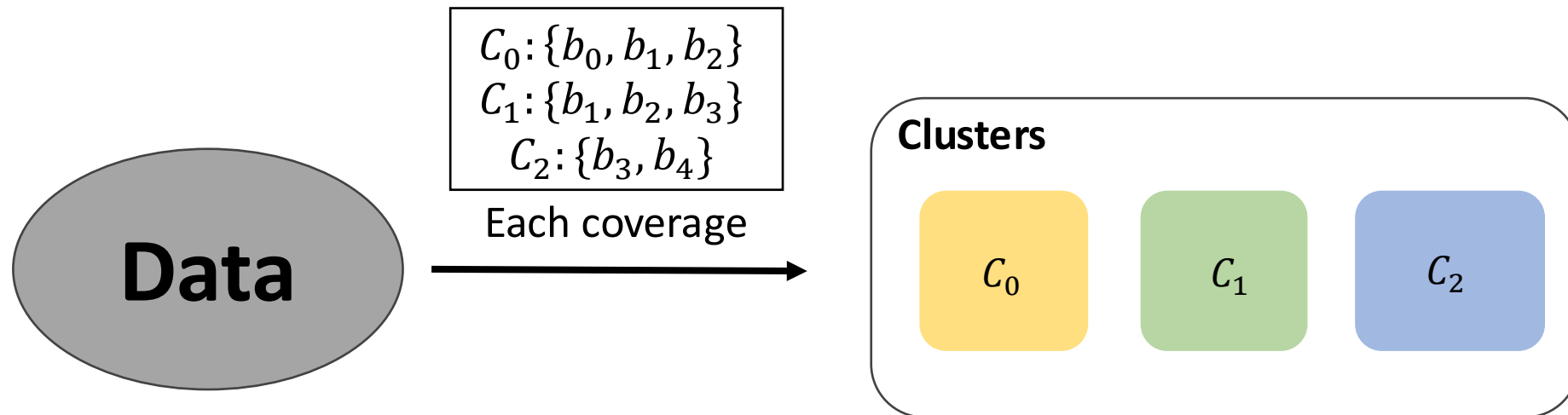
- **Collect necessary branch conditions from filtered data.**

- ✓ Data : (branch conditions, Coverage) of each test case.
 - Coverage (covered branches) can be achieved by executing test cases.
- ✓ **Classify** data into clusters and **remove unnecessary** clusters.



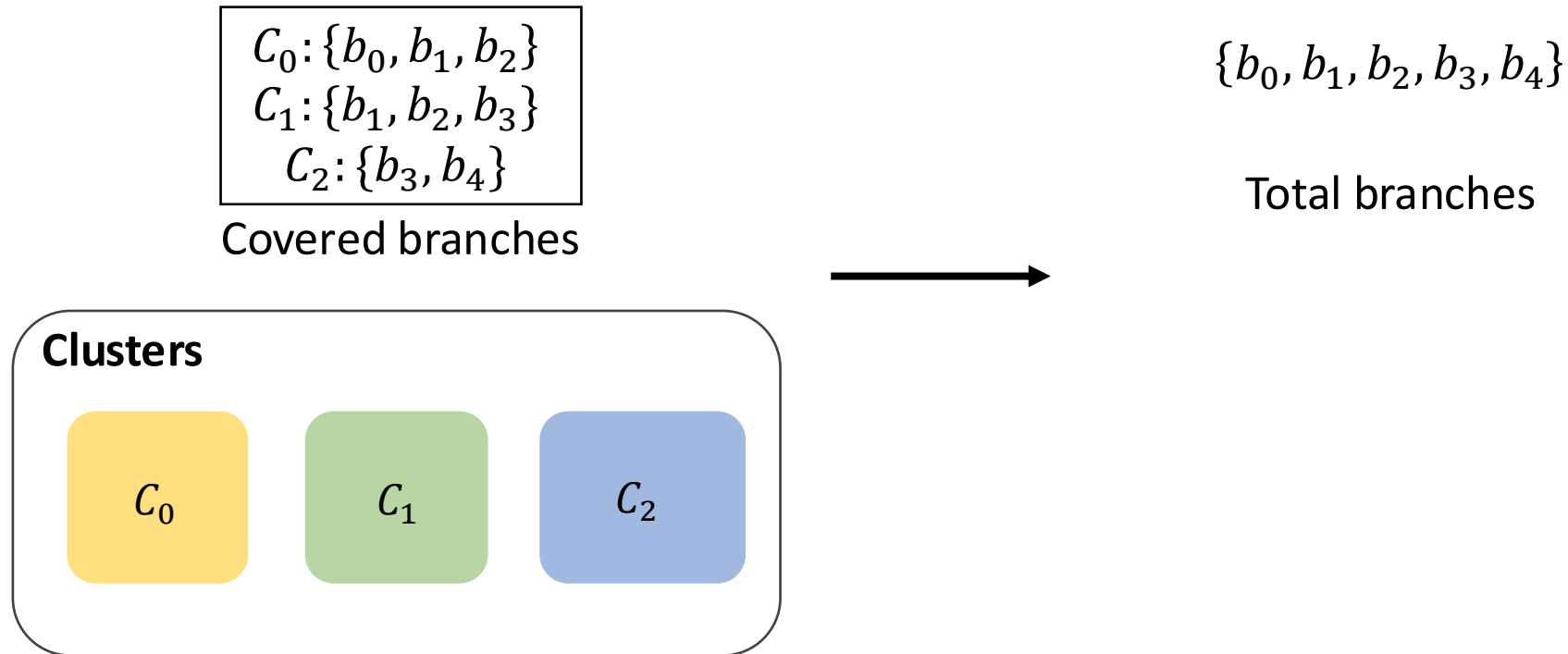
FeatMaker – Feature Generator

- **Cluster data** based on the **coverages**.
 - ✓ classify data with the same coverage(covered branches).
 - C_k : k^{th} Unique Coverage



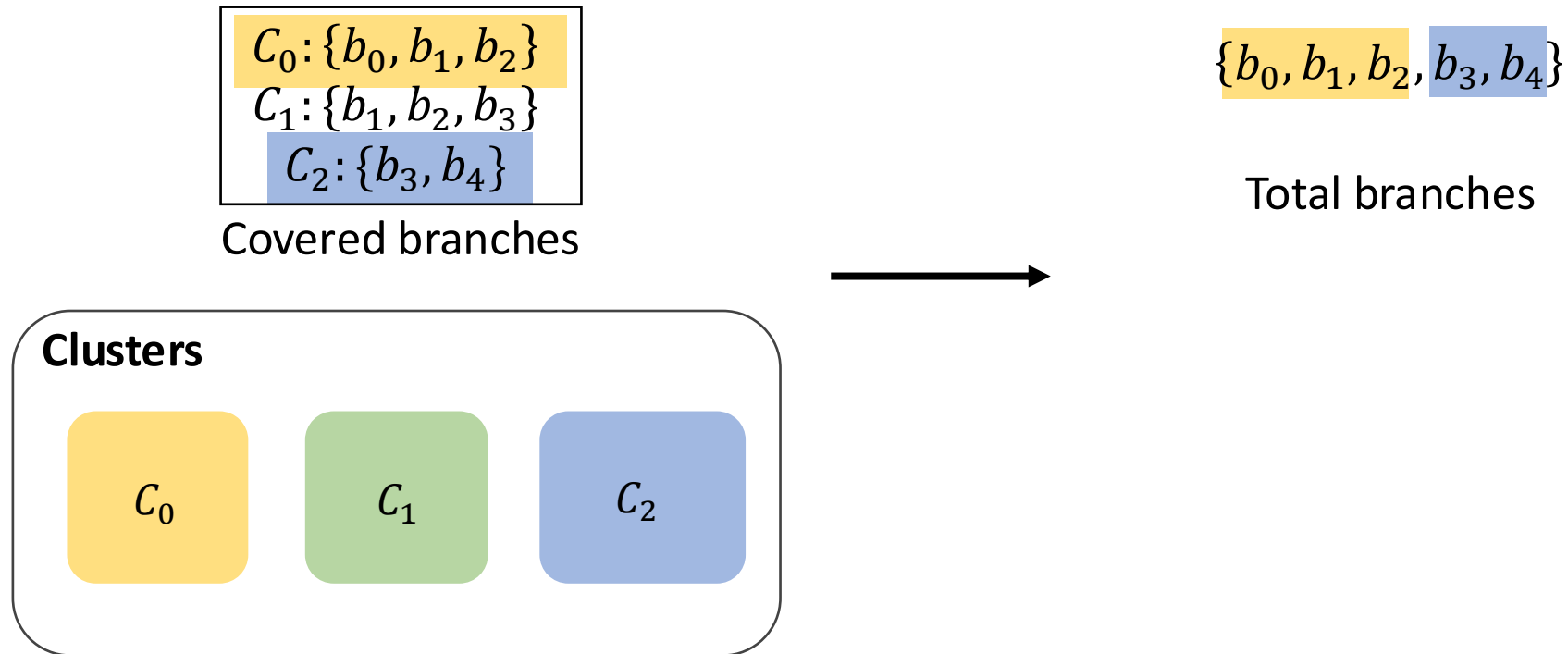
2. **Retain the minimum number of clusters** covering all branches.

✓ Utilized set cover problem.



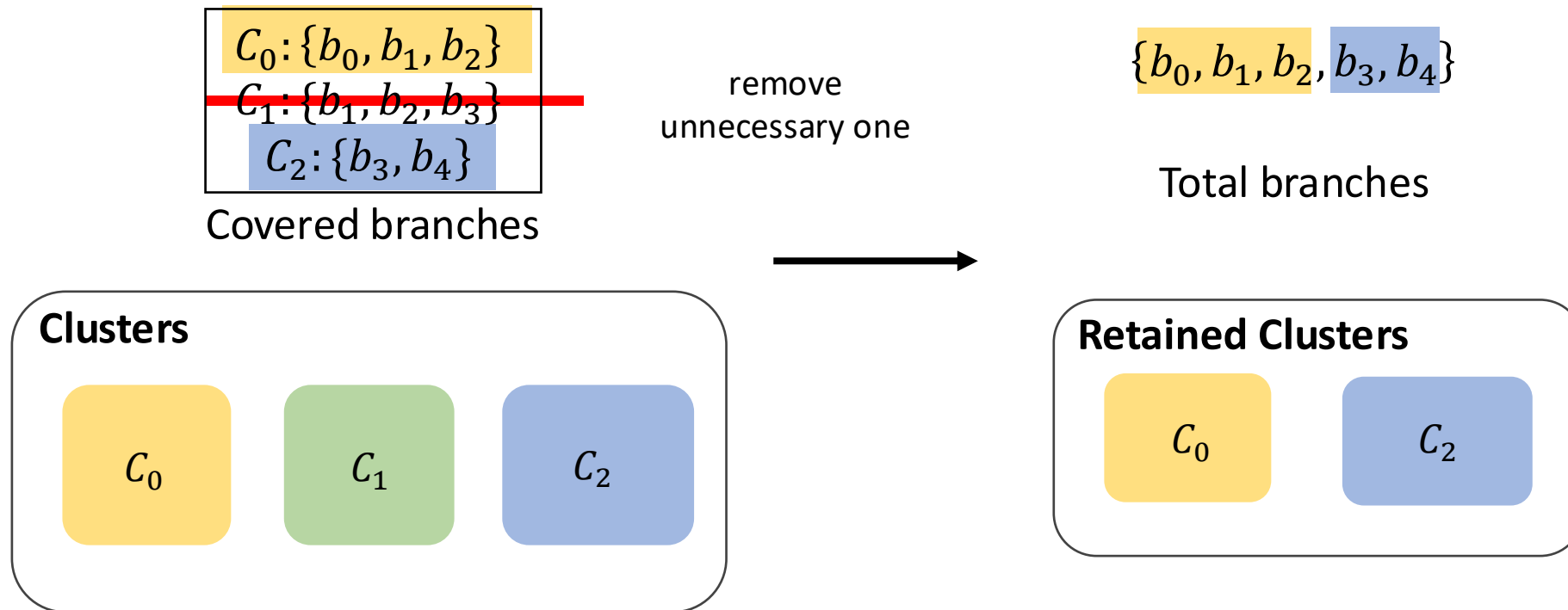
2. Retain the minimum number of clusters covering all branches.

✓ Utilized set cover problem.

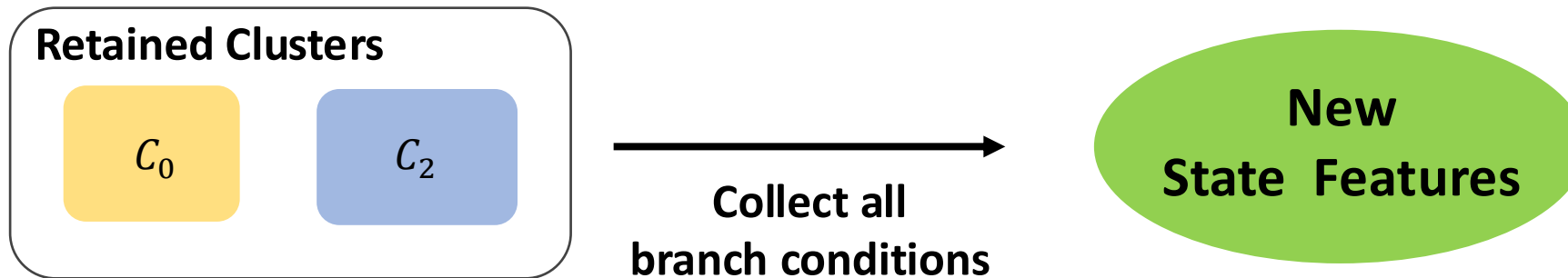


2. **Retain the minimum number of clusters** covering all branches.

✓ Utilized set cover problem.



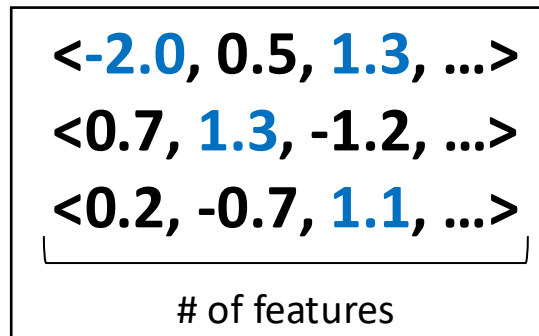
3. **Gather branch conditions** from retained clusters.
 - ✓ Collect branch conditions from all elements in each cluster.



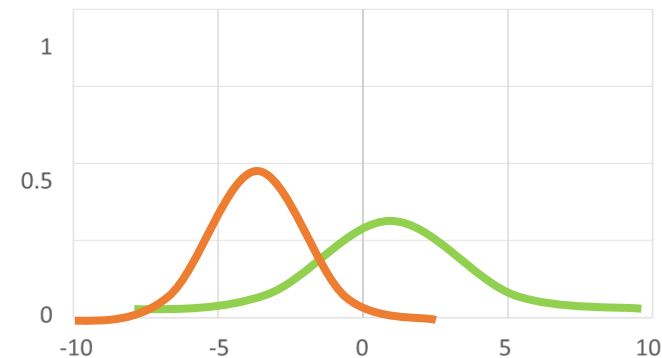
- **Generate a new scoring function similar to well-performed ones.**

- ✓ Identify **good** weight values for each feature.
- ✓ Sample new weights from the distribution of **good** weight values.

Used scoring functions



Identify good weight values

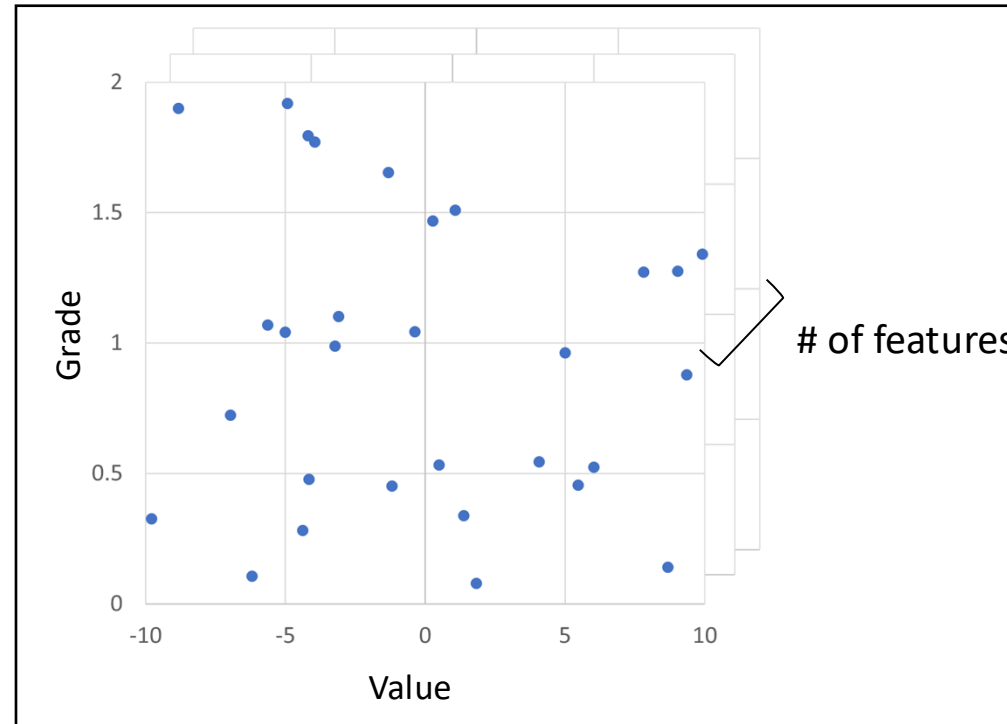


Sample new weight value

FeatMaker – Scoring function Generator

- **Grade used weights of each feature** based on the coverages.
 - ✓ covered **more** branches.
 - ✓ covered **new** branches.

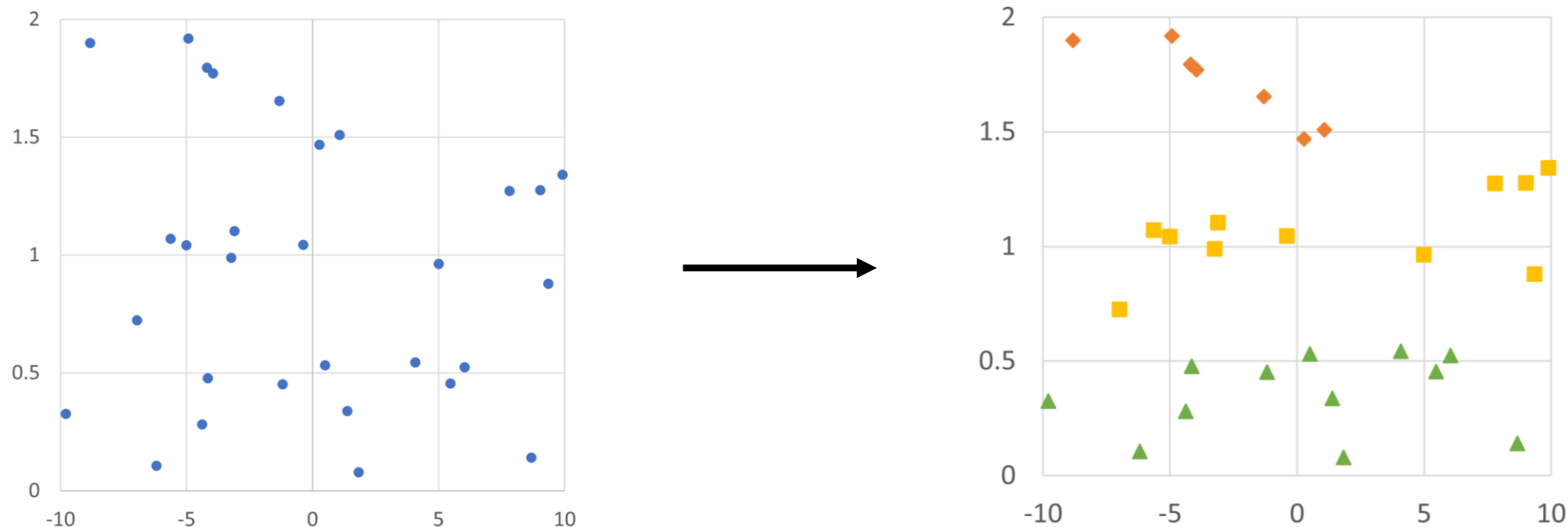
Grade all used values for each feature



FeatMaker – Scoring function Generator

2. **Identify** *good* / *bad* weights for each feature.

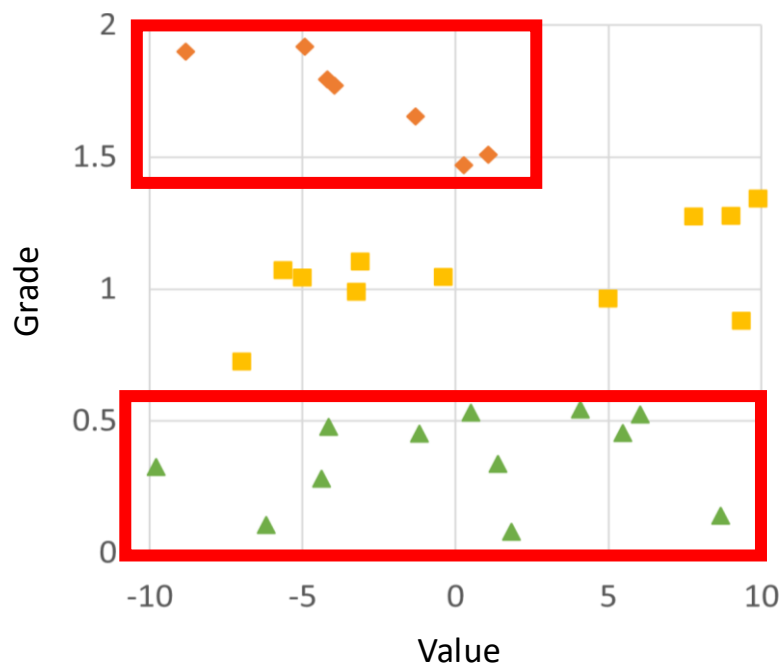
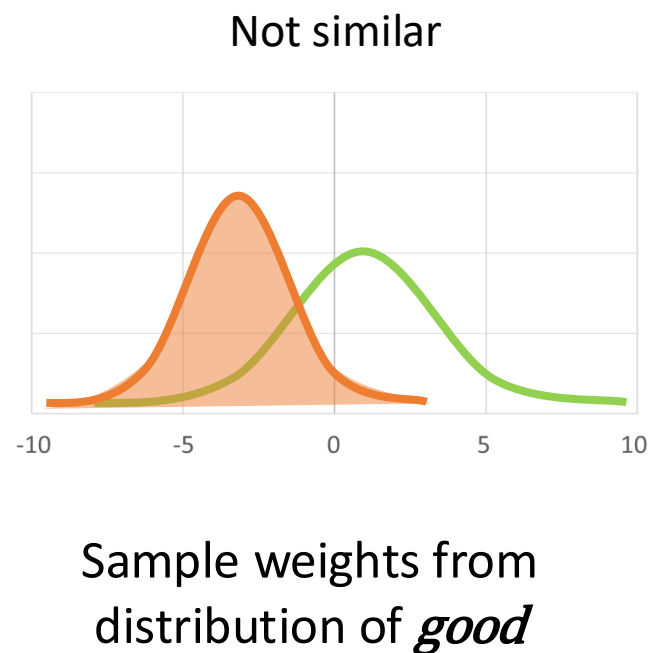
✓ k-means clustering with $k=3$ based on the scored grades.



FeatMaker – Scoring function Generator

3. Sample new weights **similar to well-performed weights.**

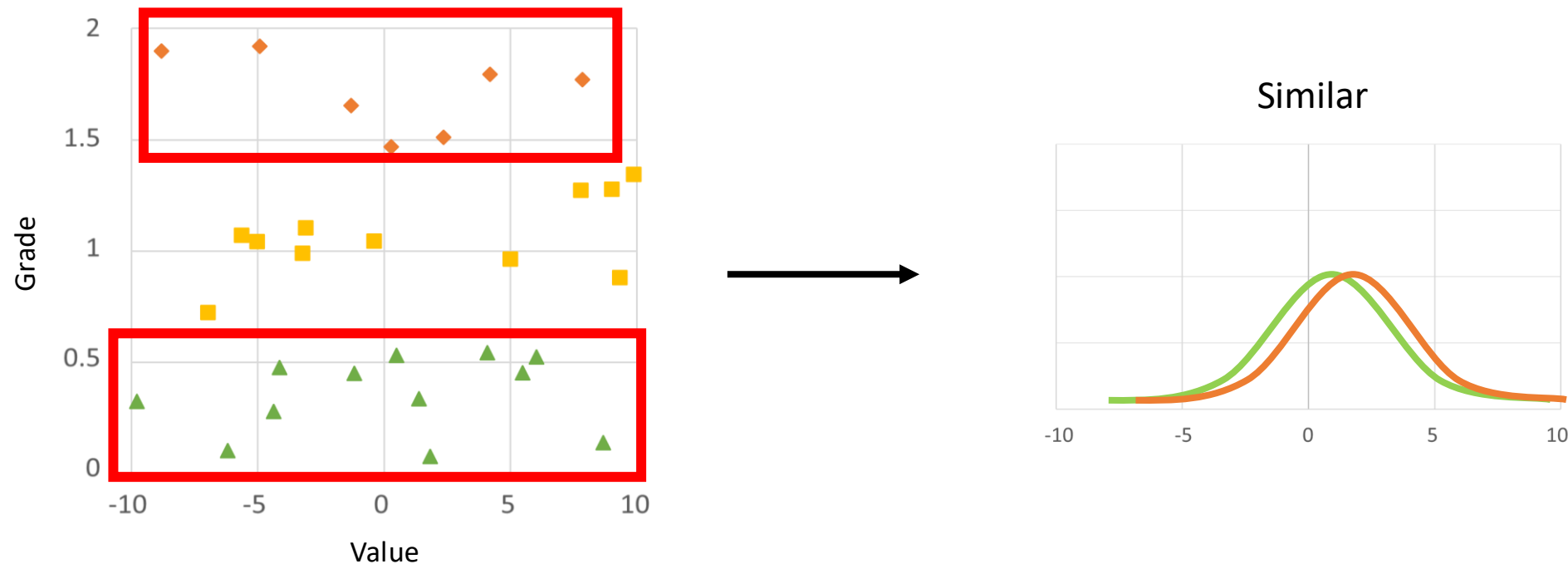
- ✓ Sample from distribution of **good** if distributions of **good/bad** are not similar.



FeatMaker – Scoring function Generator

3. Sample new weights **similar to well-performed weights.**

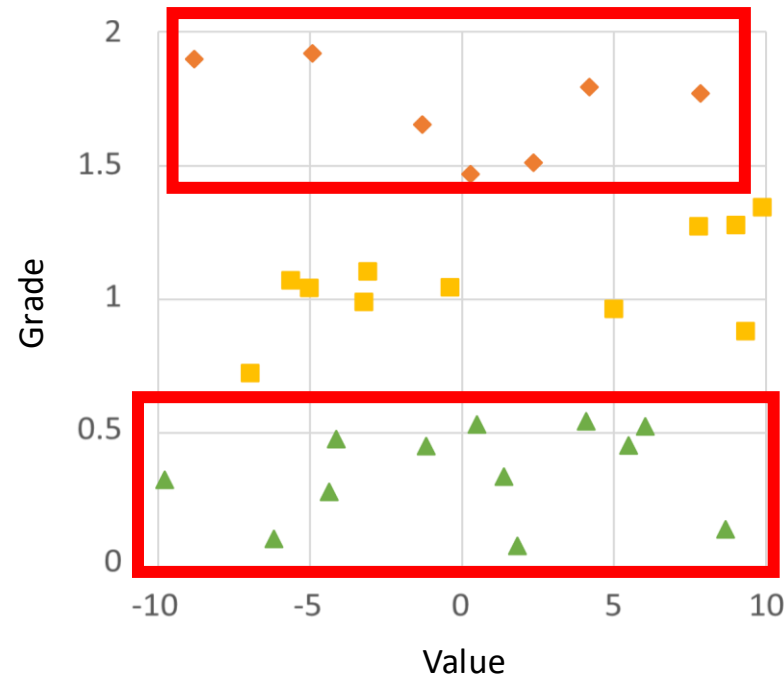
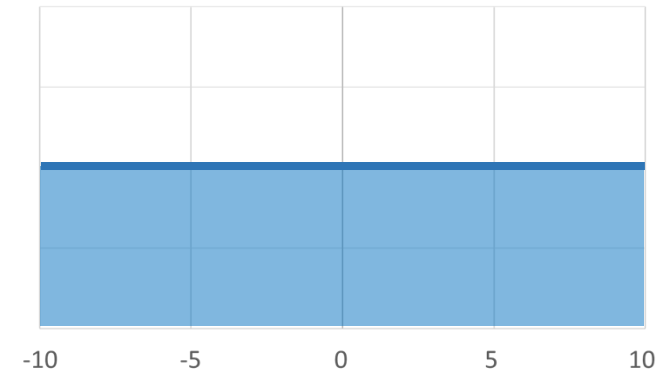
- ✓ Sample from uniform distribution if distributions of **good/bad** are similar.
- ✓ Sample from uniform distribution for not used features.



FeatMaker – Scoring function Generator

3. Sample new weights **similar to well-performed weights.**

- ✓ Sample from uniform distribution if distributions of **good/bad** are similar.
- ✓ Sample from uniform distribution for not used features.

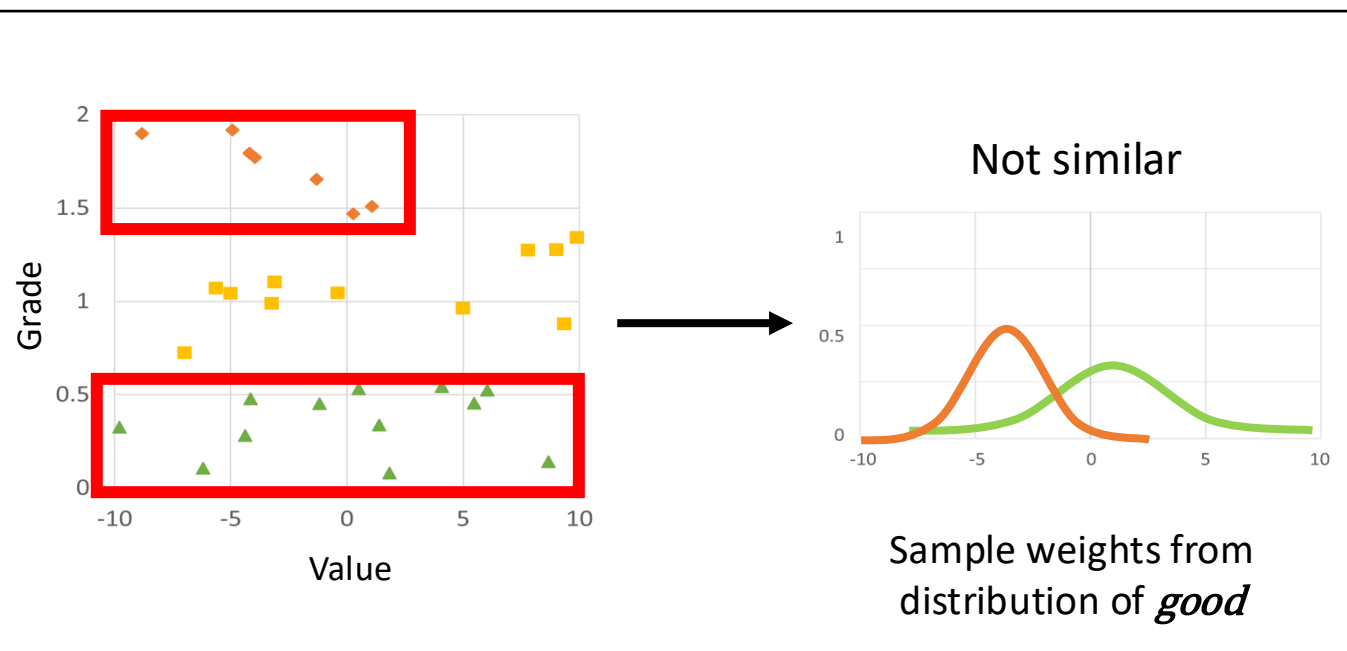


Sample weights from
Uniform distribution

FeatMaker – Scoring function Generator

3. Sample new weights **similar to well-performed weights**.

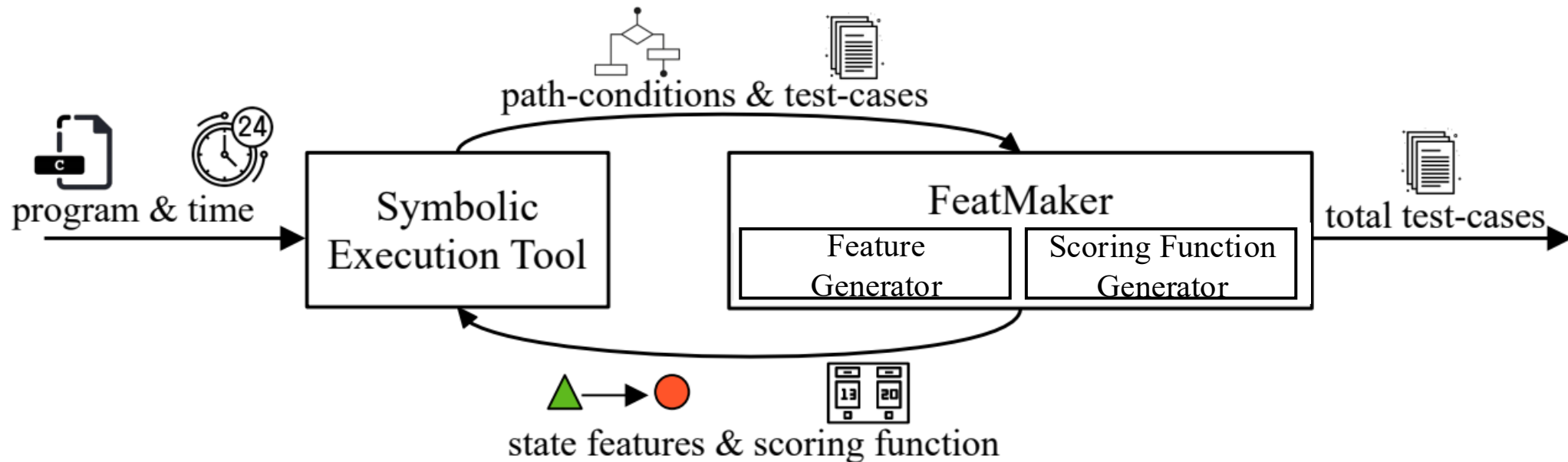
✓ Sample from distribution of **good** if distributions of **good/bad** are not similar.



x # of features → New Scoring Function

- Iteratively update **state features** & **scoring function**.

- ✓ Generate state features **reducing unnecessary data**.
- ✓ Generate scoring functions in a **data-driven approach**.



●Experimental settings

- ✓ Implemented on **KLEE-2.1**, a widely-adopted symbolic execution tool.
- ✓ 24-hour experiments on **15 benchmarks** with **4 baselines**.
 - Search strategies with multiple features
 - PARADYSE (TSE' 22)
 - LEARCH (CCS' 21)
 - Search strategies with a single feature
 - SGS (OOPSLA' 13)
 - DEPTH (OSDI' 08)

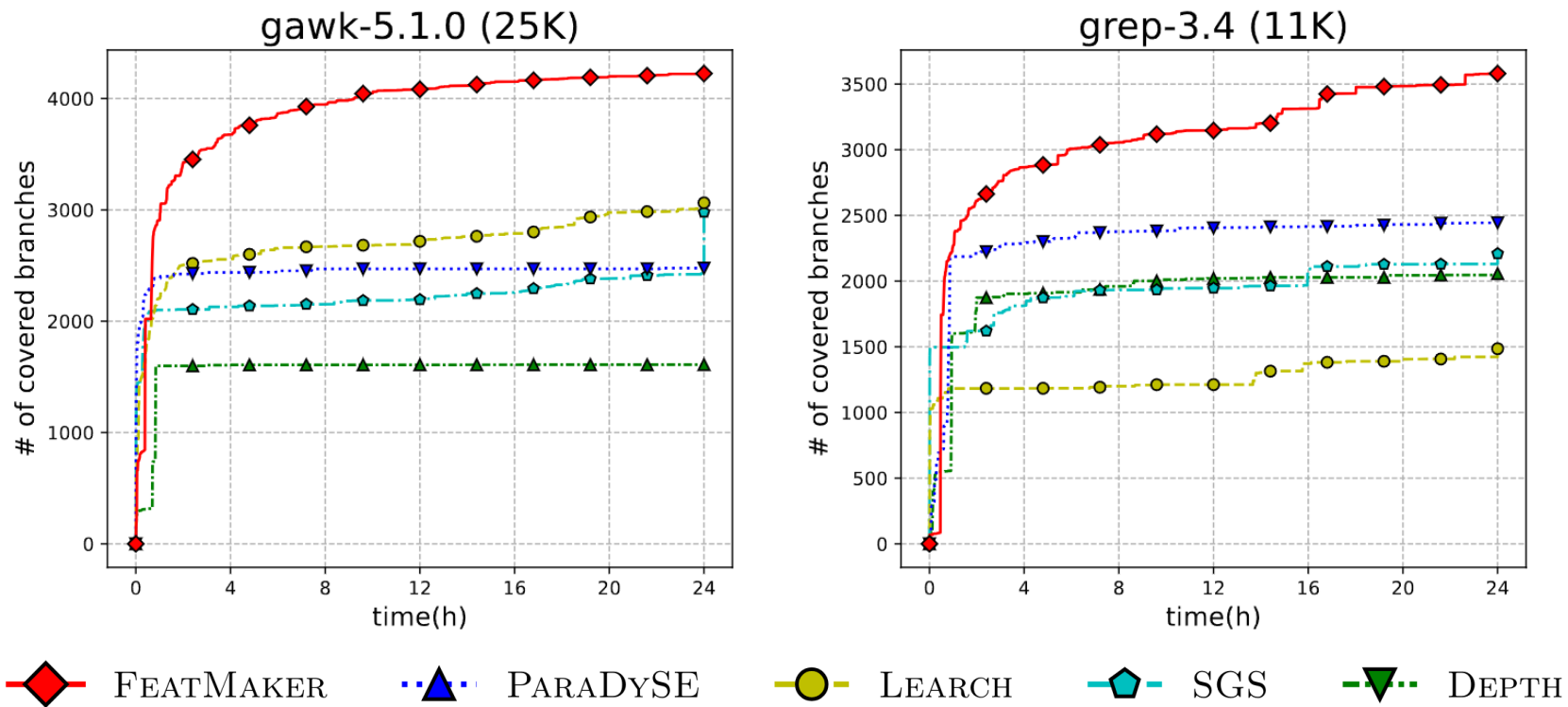
●Experimental setting

- ✓ 15 Benchmarks : collected from baseline works.
- ✓ Programs with 42K Lines to 4K lines

Programs	LOC	# of Branches	Programs	LOC	# of Branches
sqlite-3.33.0	42K	29,337	patch-2.7.6	8K	6,227
gawk-5.1.0	25K	18,299	ptx (coureutils-8.32)	6K	5,318
gcal-4.1	23K	15,799	expr (coureutils-8.32)	6K	4,608
find (findutils-4.7.0)	13K	10,071	csplit (coureutils-8.32)	6K	4,560
grep-3.4	11K	8,566	ls (coureutils-8.32)	5K	3,824
diff (diffutils-3.7)	10K	7,688	trueprint-5.4	4K	2,518
du (coureutils-8.32)	8K	6,707	combine-0.4.0	4K	2,359
make-4.3	8K	6,604			

● Code coverage

- ✓ Achieved best performance on all 15 benchmarks.
- ✓ FeatMaker covered 37% more branches on gawk & 45% more on grep.



●Code coverage

- ✓ Recorded ranks & total covered branches.
- ✓ **30% more branches** than SOTA techniques.

Rank	FEATMAKER	PARADYSE	LEARCH	SGS	DEPTH
1 st	15	0	0	0	0
2 nd	0	8	5	2	0
3 rd	0	3	5	4	3
4 th	0	4	3	5	3
5 th	0	0	2	4	9
Total Coverage	34256	26472	26288	23209	15521

●Code coverage

- ✓ Recorded ranks & total covered branches.
- ✓ Strategies with **multiple features performed better** than strategies with a single feature.

Rank	FEATMAKER	PARADYSE	LEARCH	SGS	DEPTH
1 st	15	0	0	0	0
2 nd	0	8	5	2	0
3 rd	0	3	5	4	3
4 th	0	4	3	5	3
5 th	0	0	2	4	9
Total Coverage	34256	26472	26288	23209	15521

●Code coverage

- ✓ Recorded ranks & total covered branches.
- ✓ SOTA techniques **still achieve varying results** depending on the target program.

Rank	FEATMAKER	PARADYSE	LEARCH	SGS	DEPTH
1 st	15	0	0	0	0
2 nd	0	8	5	2	0
3 rd	0	3	5	4	3
4 th	0	4	3	5	3
5 th	0	0	2	4	9
Total Coverage	34256	26472	26288	23209	15521

● Bug finding

- ✓ Only crash-occurring bugs in original source code.
- ✓ Detected **15 bugs** in 6 benchmarks and **5 bugs** were not detected by baselines.

Benchmarks	Error-Types	Error Locations	FEATMAKER	PARADySE	LEARCH	SGS	DEPTH
gawk-5.1.0	Abnormal termination	Line: 1337 in main.c	✓	×	×	×	×
	Segmentation fault	Line: 740 in /src/file-io.c	✓	✓	✓	✓	×
	Segmentation fault	Line: 3956 in /src/gcal.c	✓	✓	×	×	×
gcal-4.1	Segmentation fault	Line: 4934 in /src/gcal.c	✓	×	×	×	×
	Segmentation fault	Line: 5001 in /src/gcal.c	✓	×	×	×	×
	Segmentation fault	Line: 72 in /libc/string/strncasecmp.c	✓	✓	✓	×	×
	Abnormal termination	Line: 27 in /libc/string/strepy.c	✓	×	✓	×	×
	Abnormal termination	Line: 29 in /libc/string/memcpy.c	✓	×	×	×	×
find-4.7.0	Segmentation fault	Line: 131 in /find/tree.c	✓	×	×	×	×
	Segmentation fault	Line: 3143 in /find/parser.c	✓	×	✓	×	×
patch-2.7.6	Segmentation fault	Line: 2396 in /src/pch.c	✓	✓	✓	×	×
	Segmentation fault	Line: 385 in /src/field.c	✓	✓	✓	✓	✓
combine-0.4.0	Segmentation fault	Line: 458 in /src/field.c	✓	✓	×	×	×
	Segmentation fault	Line: 633 in /src/df_options.c	✓	✓	×	×	×
	Segmentation fault	Line: 646 in /src/df_options.c	✓	×	×	×	×

●Bug finding

✓ Not fixed in latest version(4.9.0) of **find** (<https://savannah.gnu.org/bugs/?64717>).

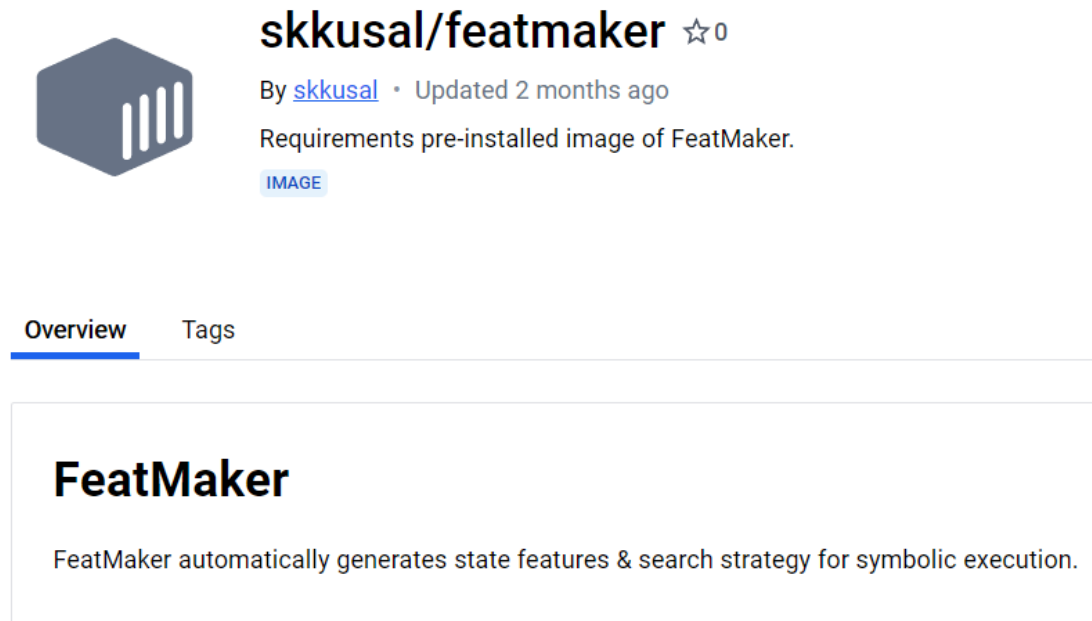
Input : find -amin NAN

bug #64717: find-4.9.0: aborted after assertion at line 3179 of parser.c

Submitter:	None	<div>Submit changes and close</div>	
Submitted:	Tue 26 Sep 2023 12:03:18 PM UTC	<div>Submit changes and re-open</div>	
		<div>Preview</div>	
Category:	find	Severity:	3 - Normal
Item Group:	None	Status:	Fixed
Privacy:	Public	Assigned to:	berny
Originator Name:	Jaehan Yoon	Originator Email:	-email is unavailable-
Open/Closed:	Open	Release:	4.9.0
Fixed Release:	None		

* Mandatory Fields

- Our tool is publicly available at the following links.
 - ✓ Docker container image : hub.docker.com/r/skkusal/featmaker
 - ✓ Github : github.com/skkusal/featmaker



Docker Hub QR code

- Existing state features have clear limitations.
 - ✓ Performance **varies** depending on the target program.
 - ✓ All features had to be **manually designed** based on human insights.
- FeatMaker: a novel technique that automatically generates state features.
 - ✓ Uses **branch conditions** as boolean features.
 - ✓ Retains only the necessary branch conditions.
- Achieved significant improvement in performance.
 - ✓ Covered **30% more branches** than SOTA techniques.
 - ✓ Detected **15 bugs** and **6 bugs were not detected** by SOTA techniques.

Thank You!



Docker Hub QR code

- The number of state features

- ✓ Removed 83% of branch conditions on average.

