4th International KLEE Workshop on Symbolic Execution

15-16 April 2024, Lisbon, Portugal, Co-located with ICSE 2024

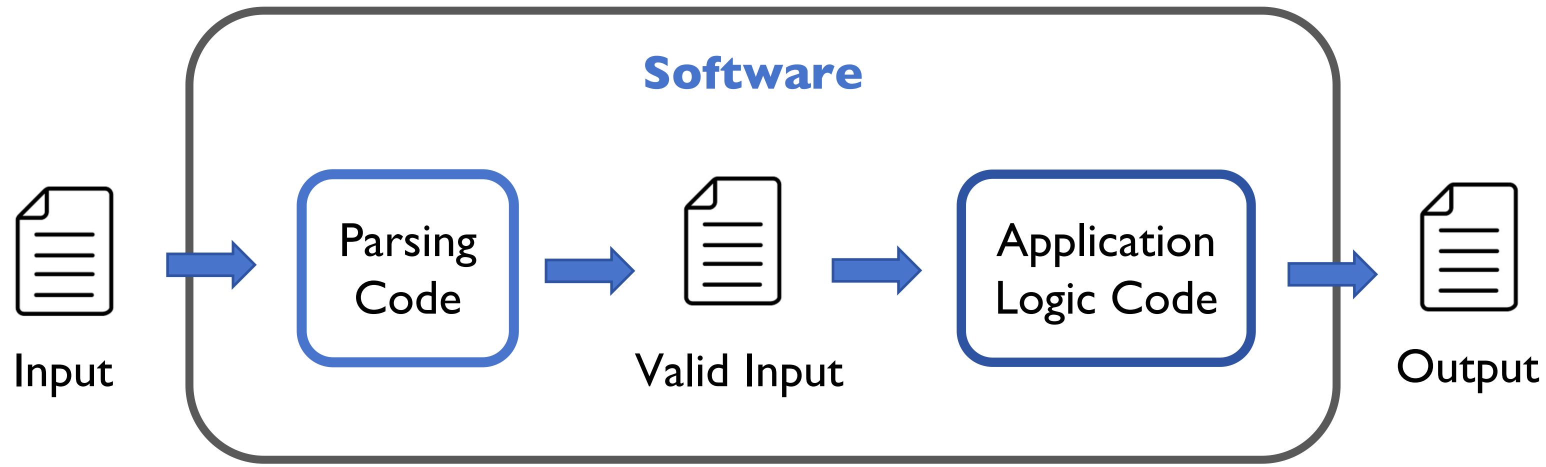
Poster: Input Grammar Oriented Symbolic Execution

Weijiang Hong (hongweijiang17@nudt.edu.cn)

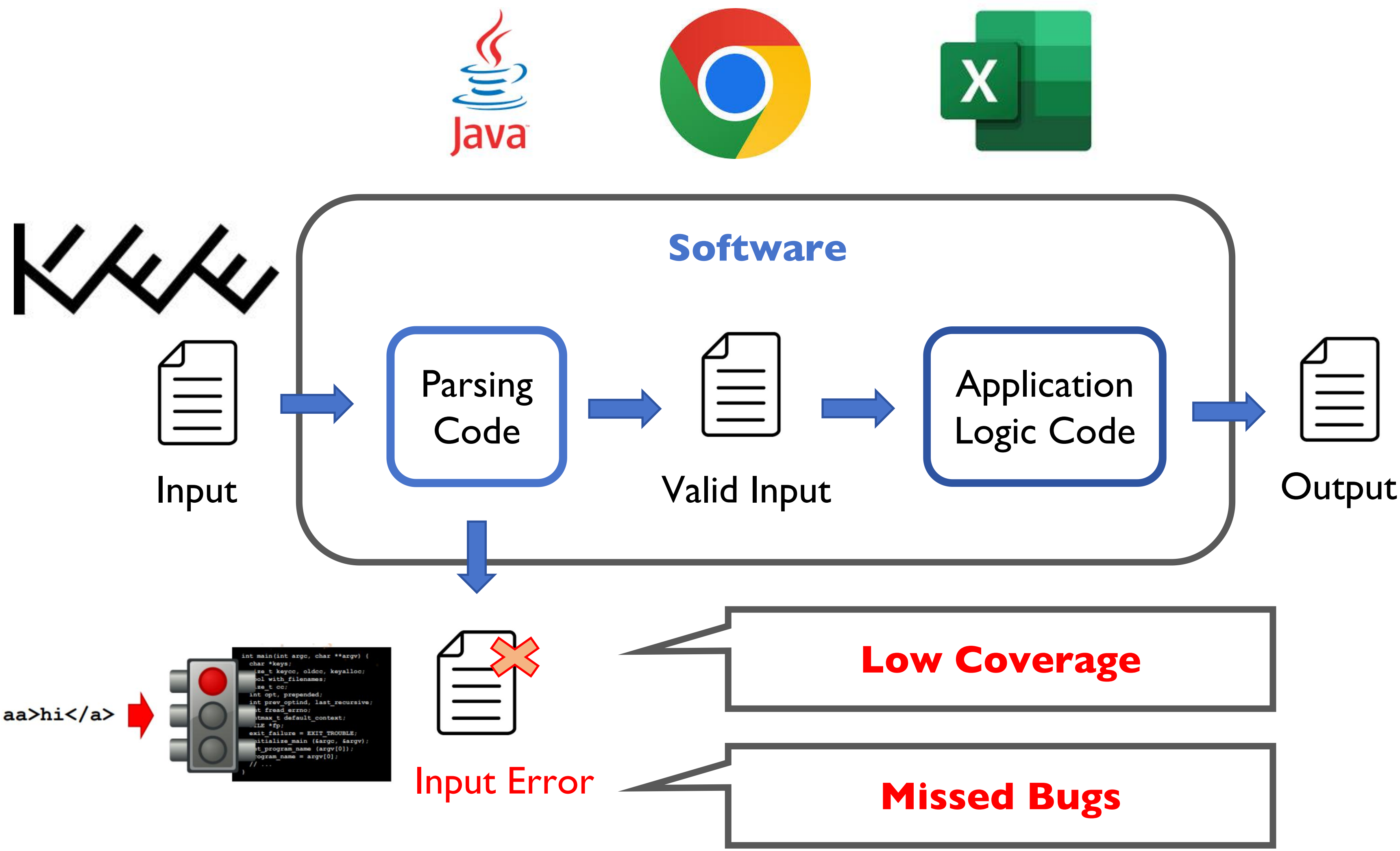
joint work with Ke Ma, Yunlai Luo, Zhenbang Chen, Yufeng Zhang and Ji Wang

College of Computer, NUDT& College of Computer Science and Electronic Engineering, HNU, China

1. Background

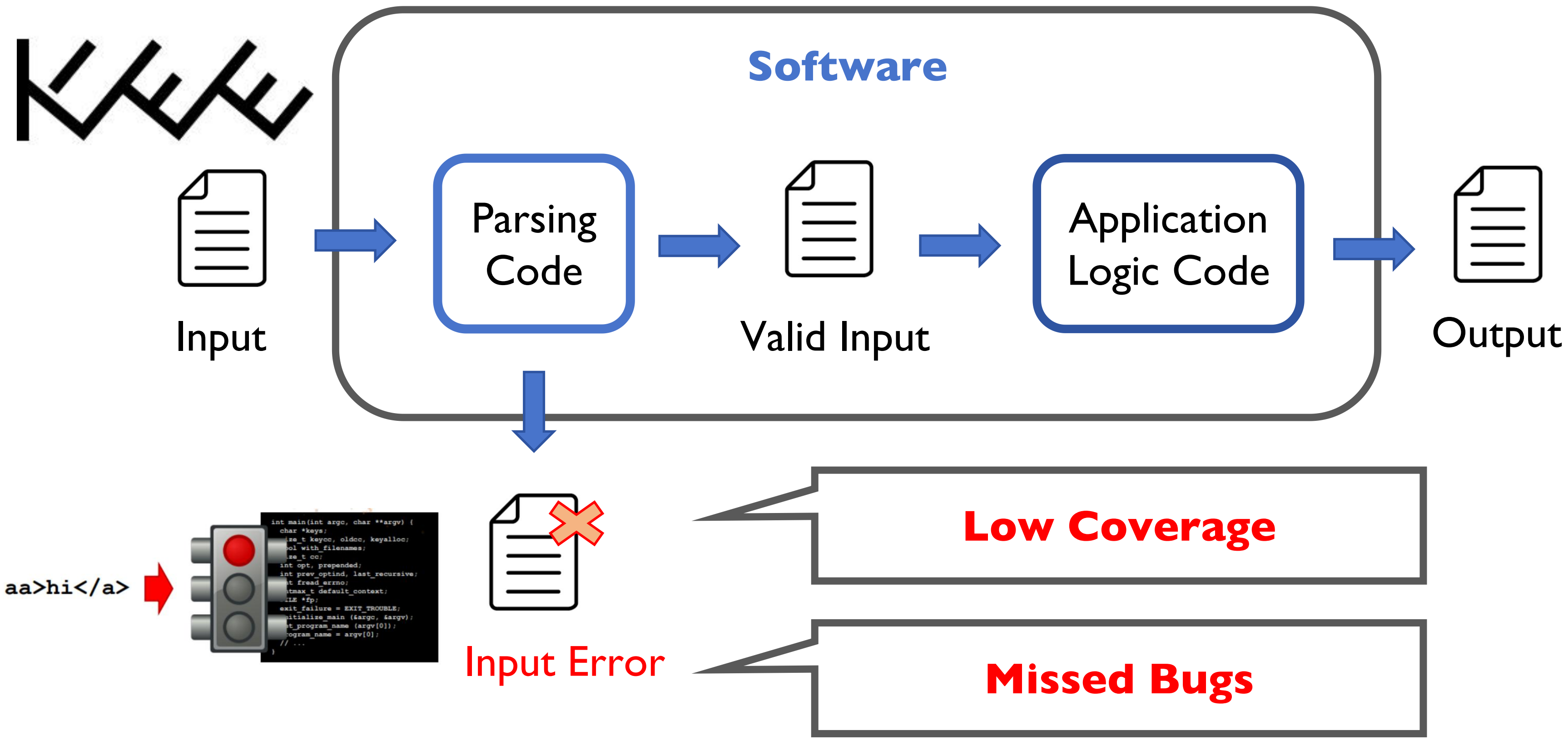


I. Background



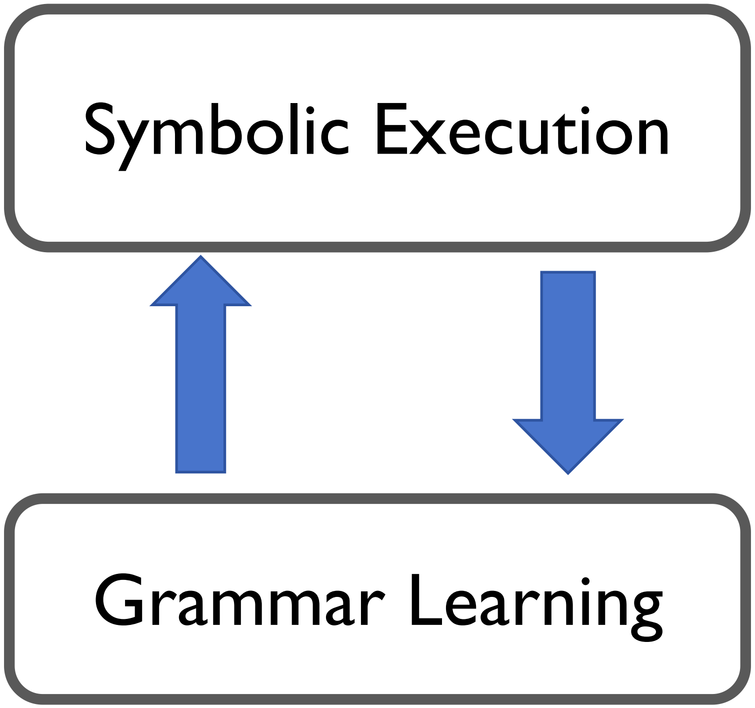
I. Background

The underlying issue is that **symbolic execution** lacks awareness of the program's input format grammar.



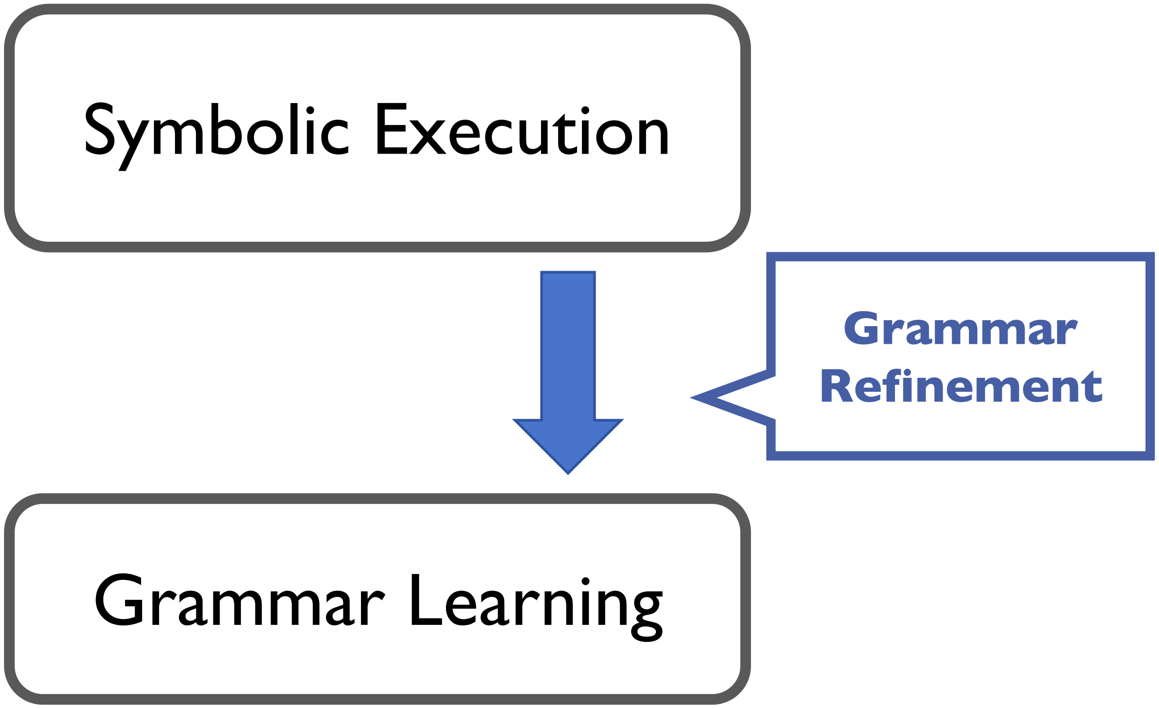
2. Key Idea

The underlying issue is that **symbolic execution** lacks awareness of the program's input format grammar.



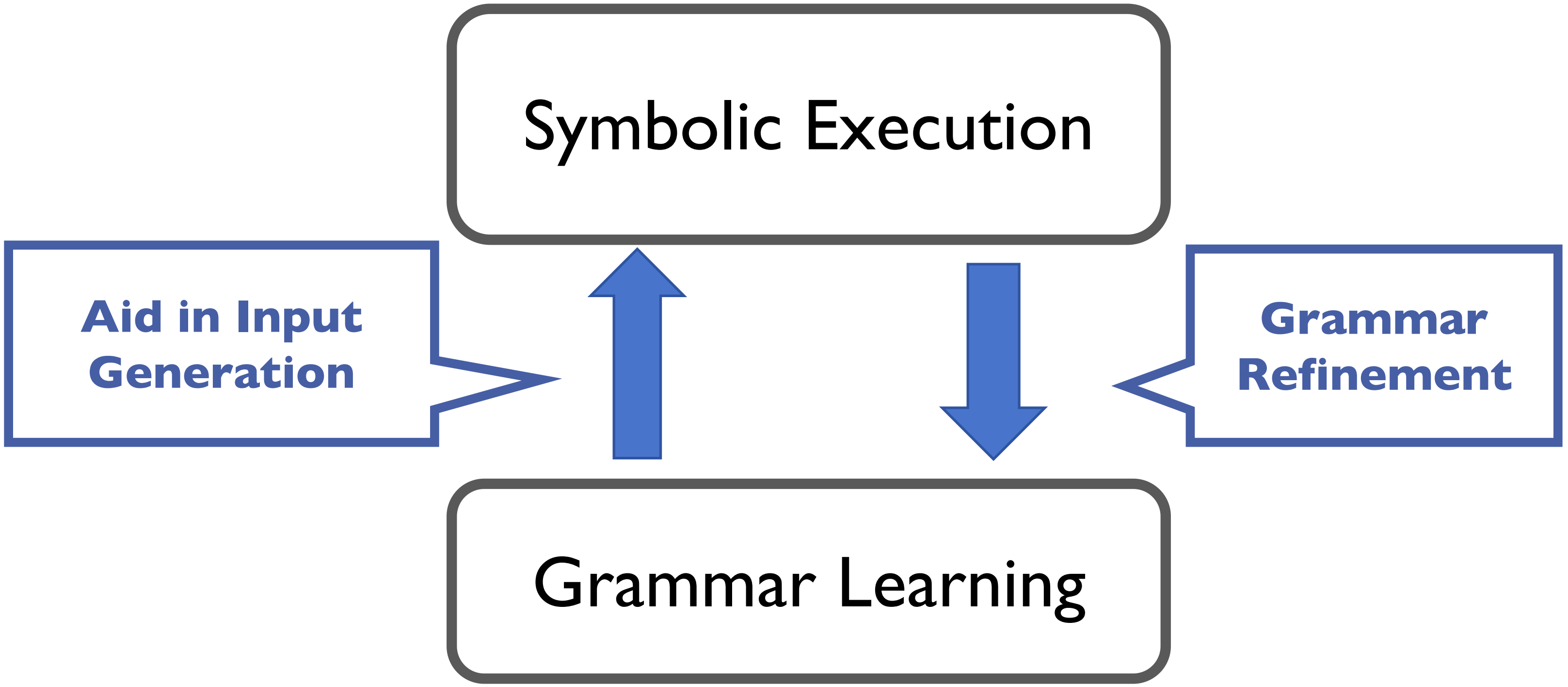
2. Key Idea

The underlying issue is that **symbolic execution** lacks awareness of the program's input format grammar.

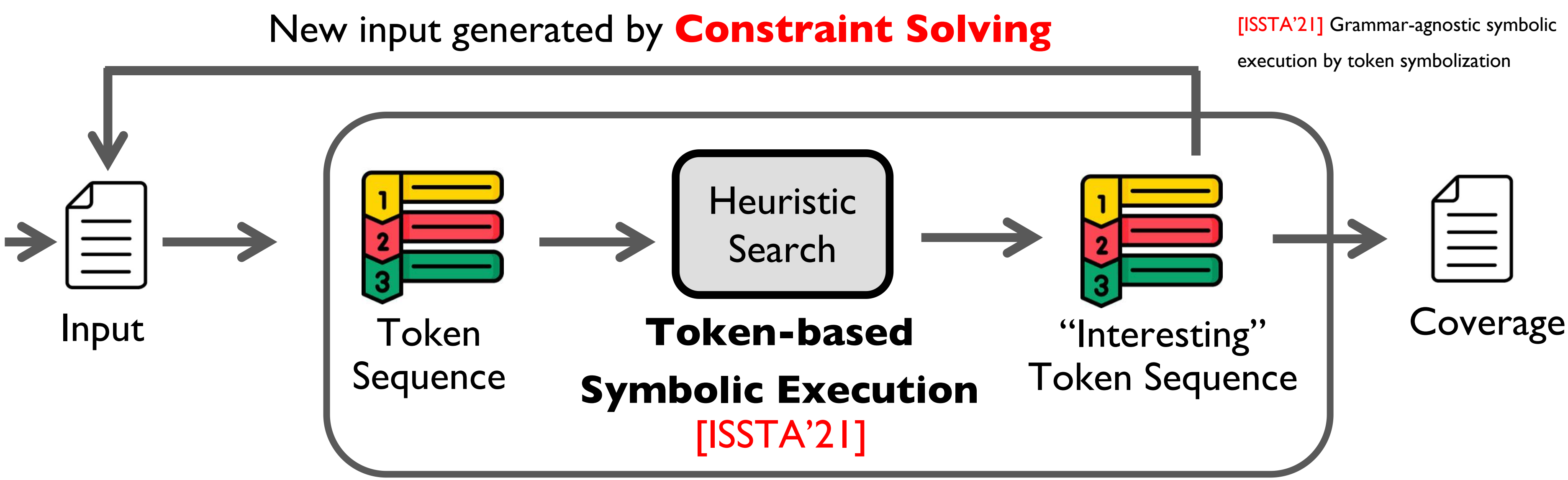


2. Key Idea

The underlying issue is that **symbolic execution** lacks awareness of the program's input format grammar.



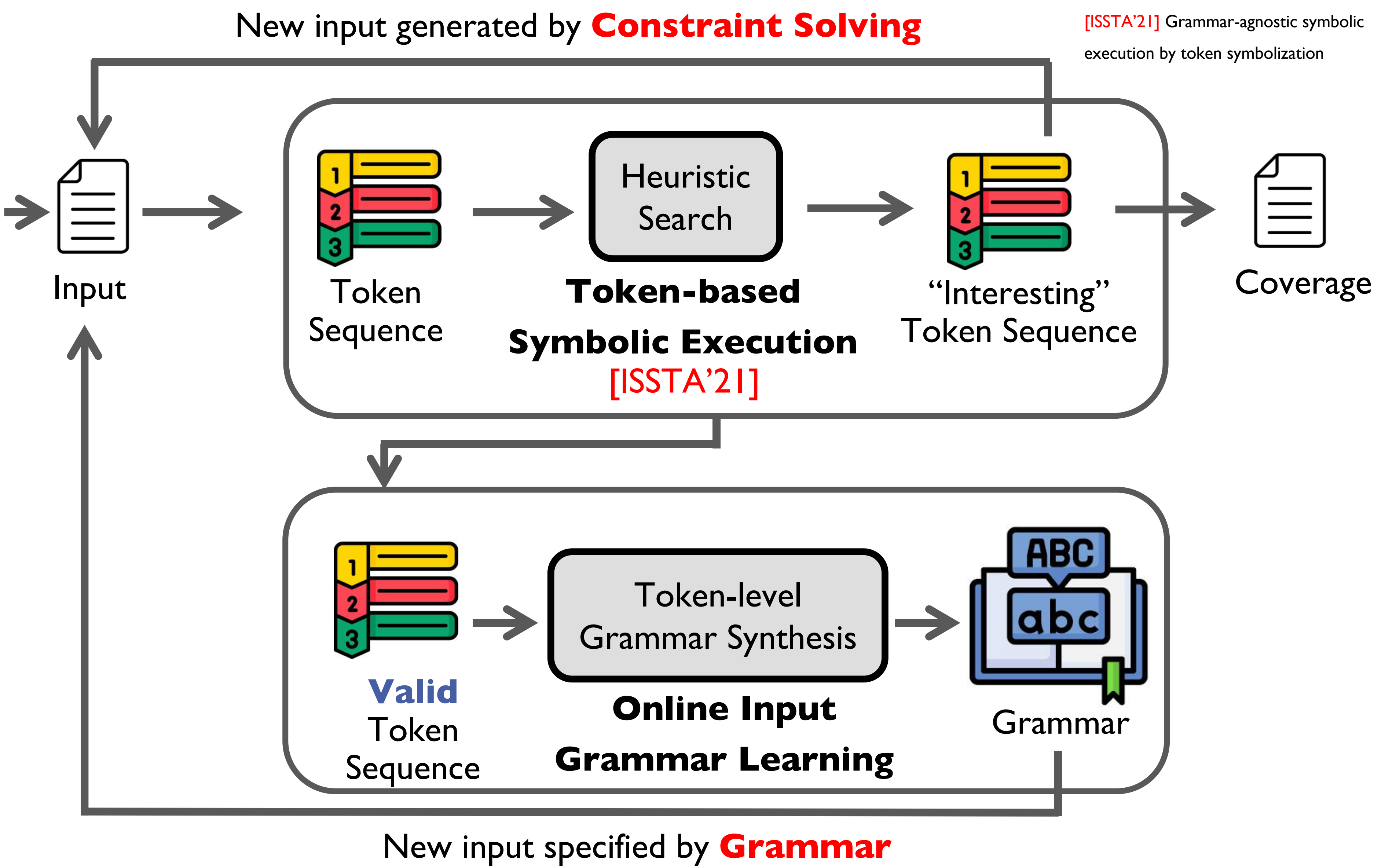
3. Framework



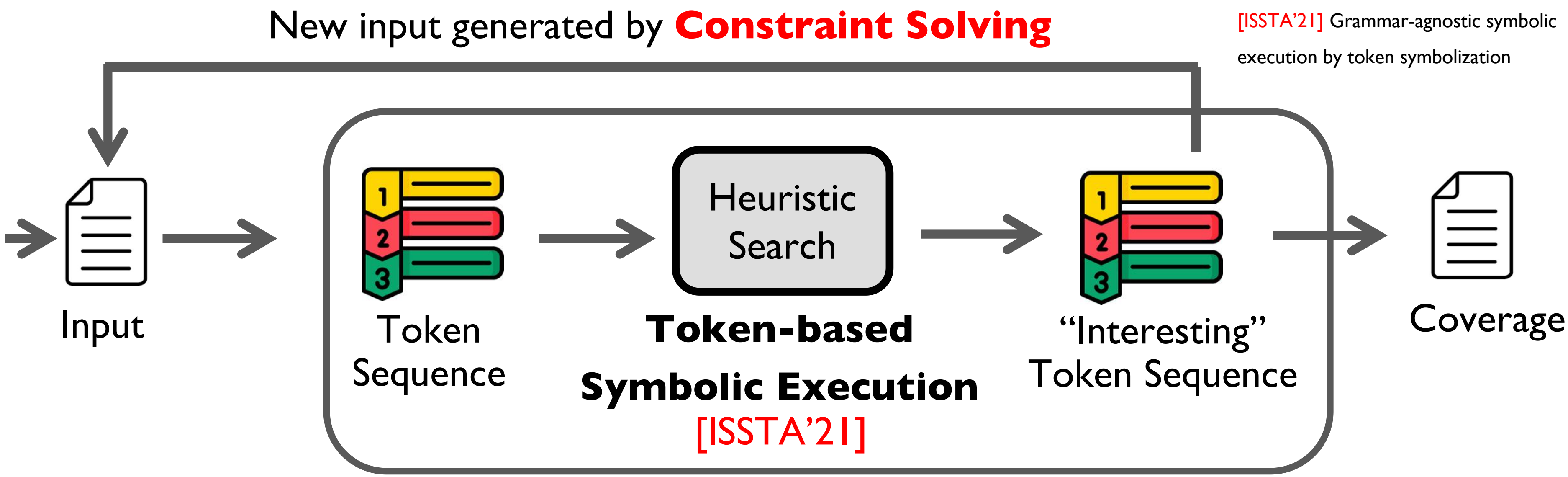
[ISSTA'21] Grammar-agnostic symbolic execution by token symbolization

3. Framework

[ISSTA'21] Grammar-agnostic symbolic execution by token symbolization



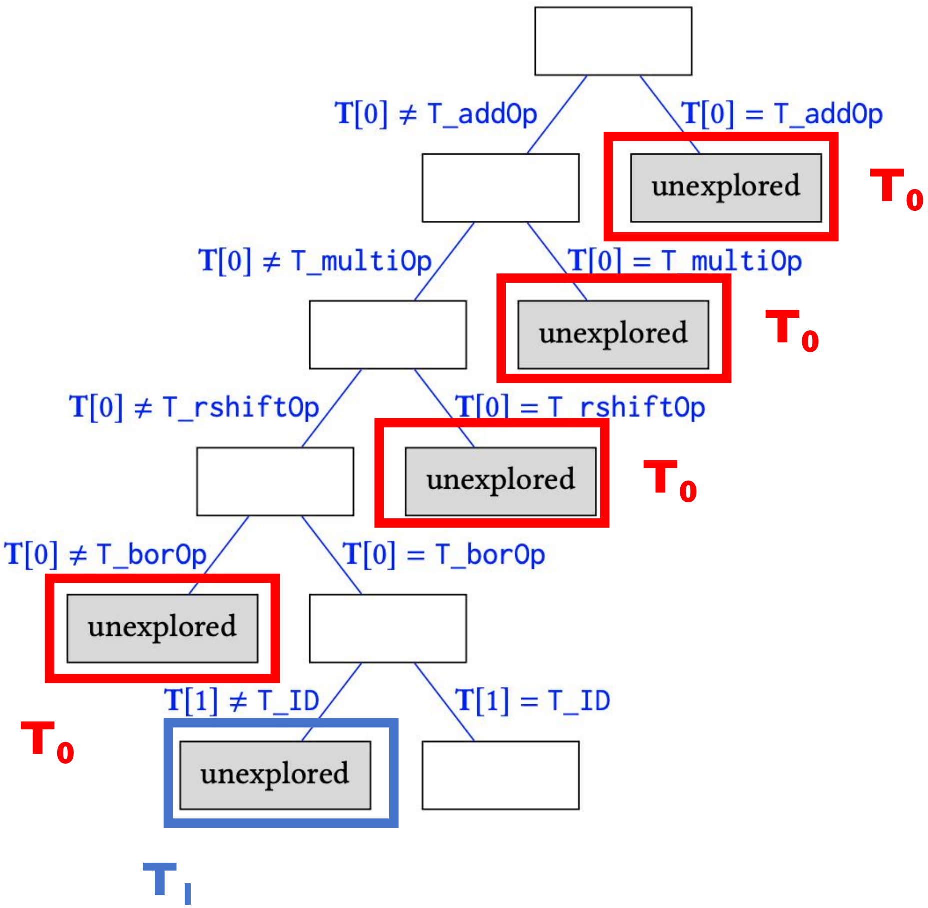
3.1 Heuristic Search



[ISSTA'21] Grammar-agnostic symbolic execution by token symbolization

How to design the **heuristic search** that tries to generate new inputs, covering more syntax rules in the parsing code?

3.1 Heuristic Search



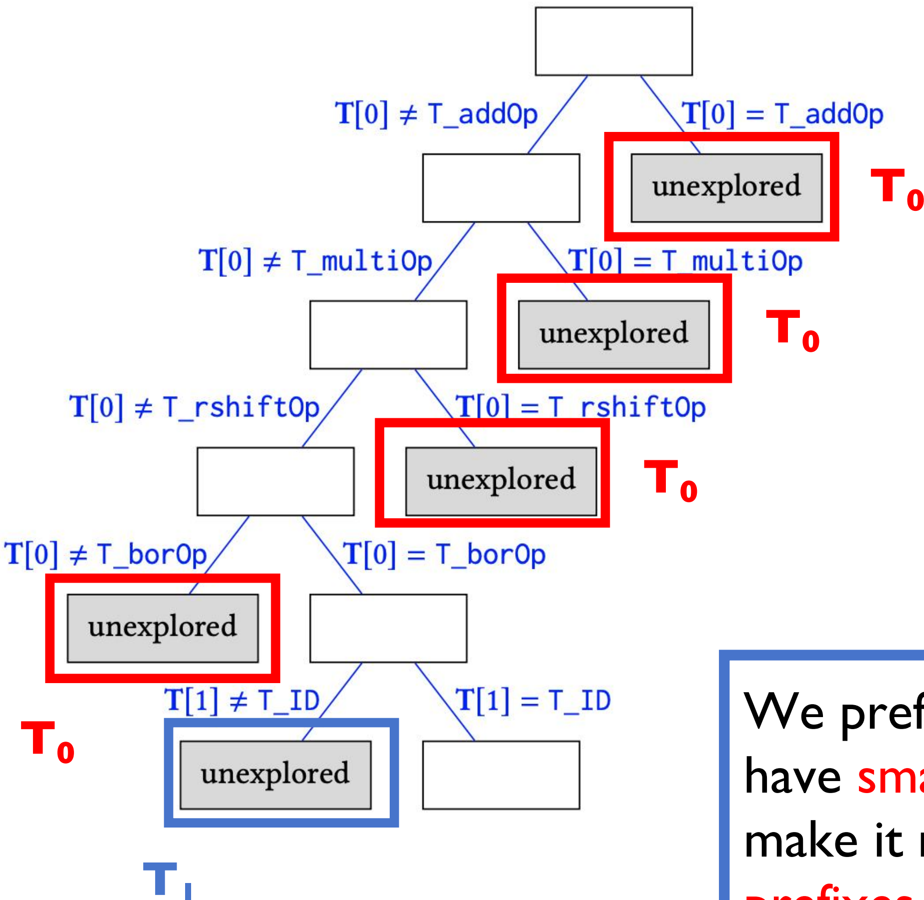
Input: bitor a

Token Sequence: ⟨T_borOp, T_ID⟩

Path Condition:

T[0] ≠ T_addOp ∧ T[0] ≠ T_multiOp ∧ T[0] ≠ T_rfhiftOp ∧ T[0] = T_borOp ∧ T[1] = T_ID

3.1 Heuristic Search



Input: bitor a

Token Sequence: ⟨T_borOp, T_ID⟩

Path Condition:

T[0] ≠ T_addOp ∧ T[0] ≠ T_multiOp ∧ T[0] ≠ T_rfhiftOp ∧ T[0] = T_borOp ∧ T[1] = T_ID

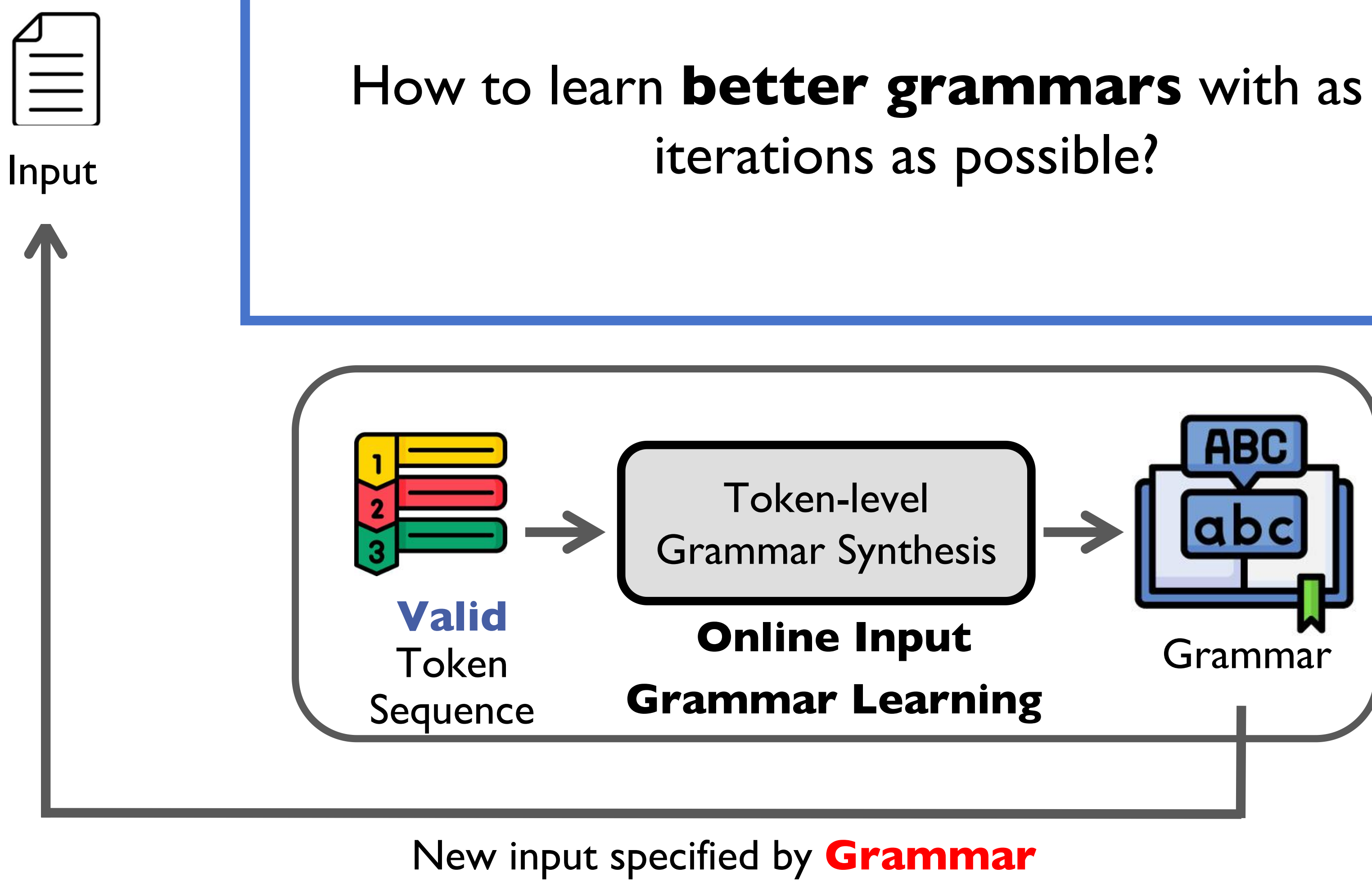
We prefer to select the unexplored branches that have smaller token indexes in priority, which make it more easier to generate different token prefixes.

New Path Condition:

T[0] = T_addOp

3.2 Grammar Synthesis

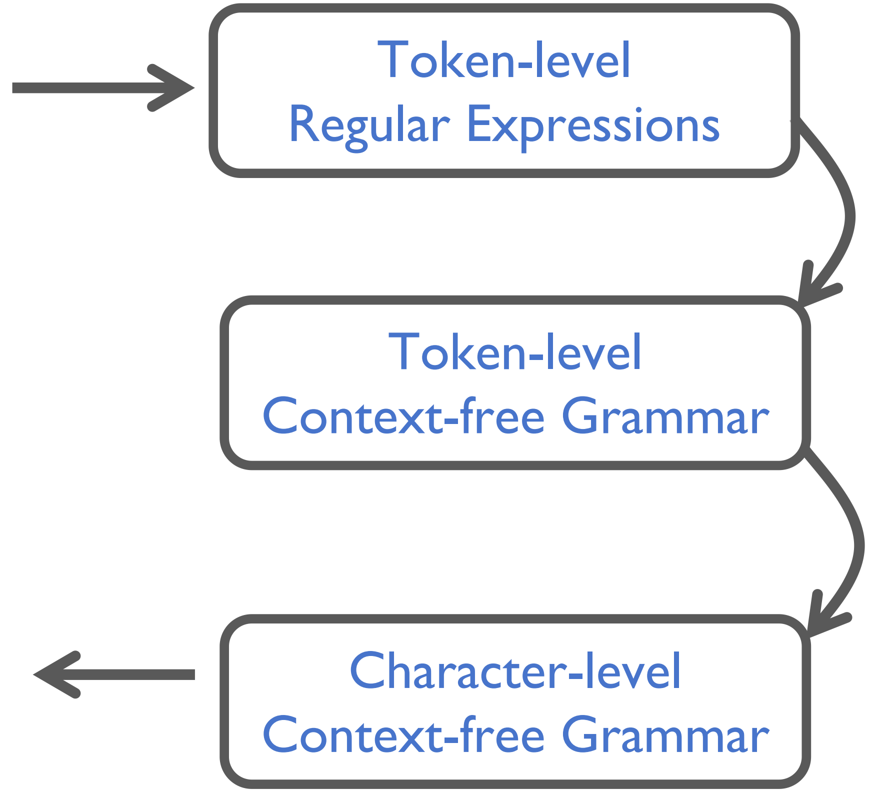
How to learn **better grammars** with as few iterations as possible?



3.2 Grammar Synthesis

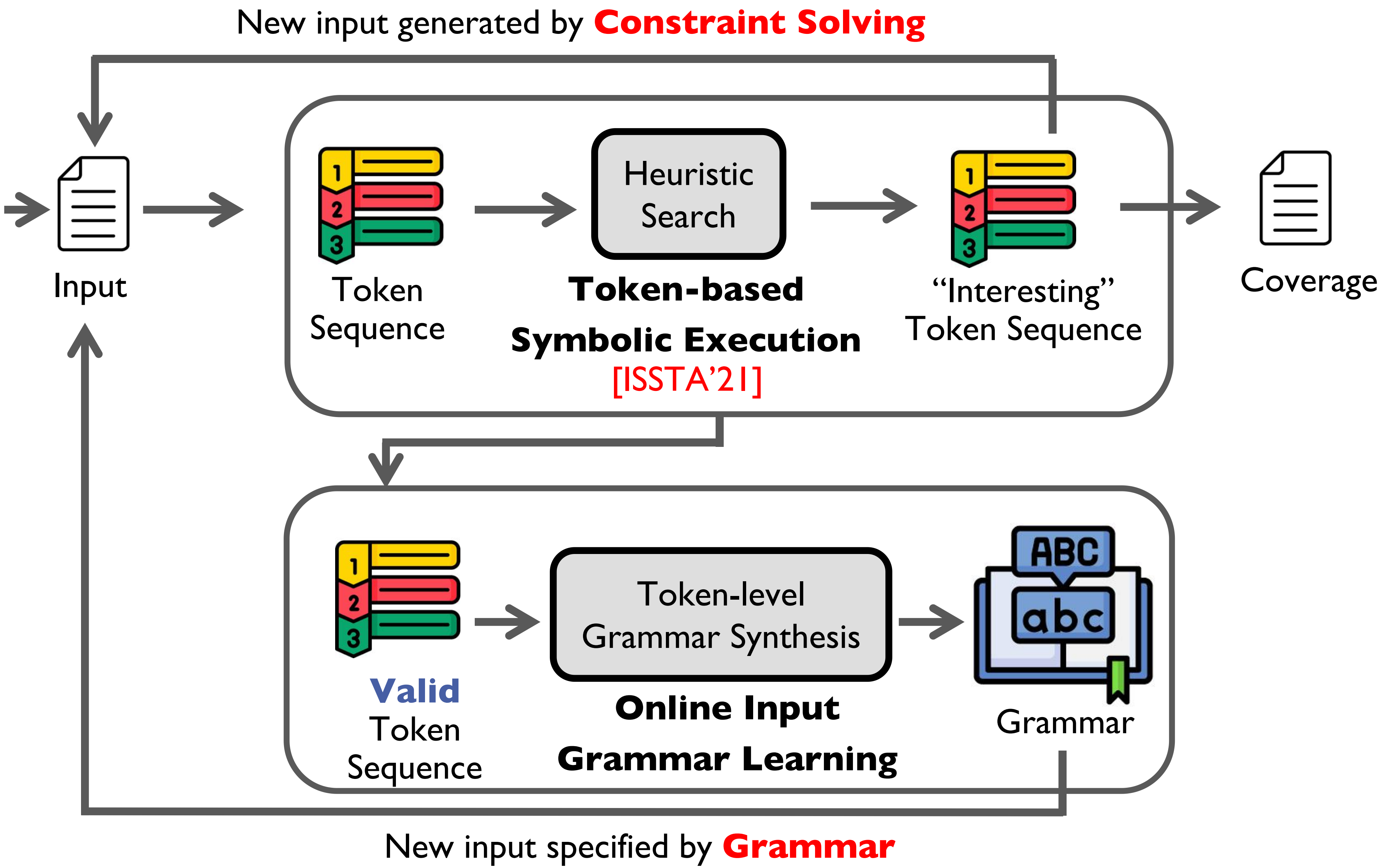
Valid Input: a+a+1

Token Sequence:
⟨T_ID, T_OP, T_ID, T_OP, T_NUM⟩



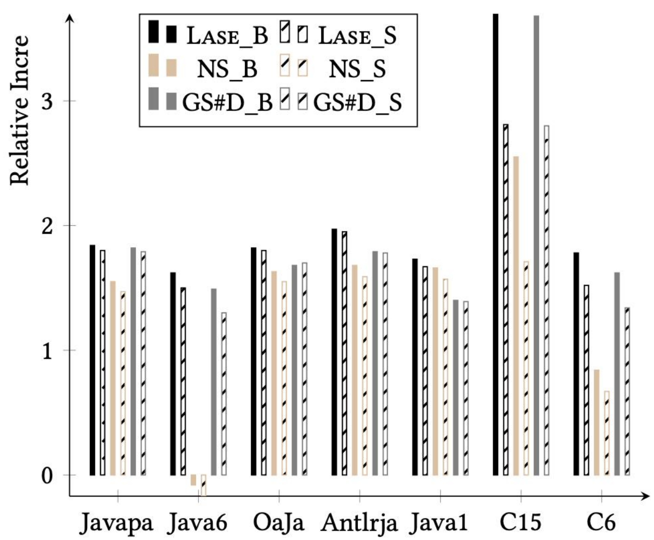

```
⟨expr⟩  ::=  ⟨expr⟩⟨op⟩⟨term⟩ | ⟨term⟩
⟨term⟩  ::=  ⟨num⟩ | ⟨ID⟩
 ⟨num⟩  ::=  '0' | '1' | ... | '9'
  ⟨ID⟩  ::=  'a' | 'b' | ... | 'y' | 'z'
  ⟨op⟩  ::=  '+'
```

3. Framework

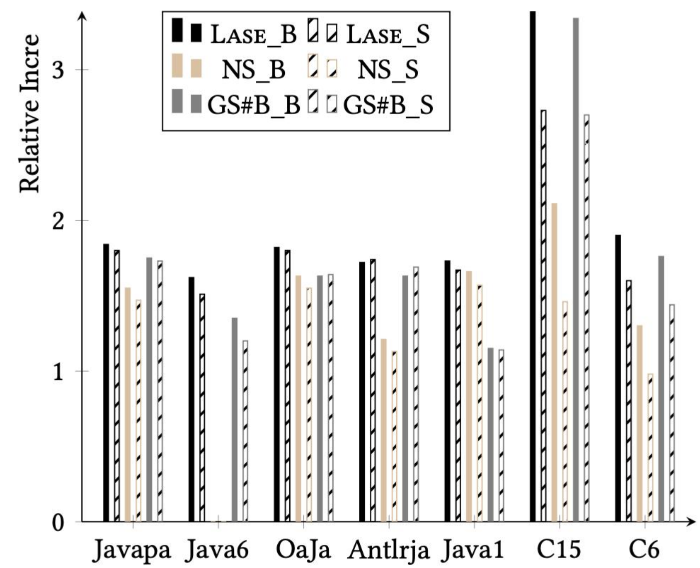

New input generated by **Constraint Solving**
Input
Token Sequence
Heuristic Search
Token-based Symbolic Execution
[ISSTA'21]
"Interesting" Token Sequence
Coverage
Valid Token Sequence
Token-level Grammar Synthesis
Online Input Grammar Learning
ABC
abc
Grammar
New input specified by **Grammar**

4. Results

- for Java compiler **Janino**, on average, we achieve a 50.92% increase in statement coverage under the BFS strategy, and a 57.68% increase in statement coverage under the DFS strategy.
- for C compiler **CLoli**, on average, we achieve a 289.57% under BFS strategy, and a 342.09% increase in statement coverage under the DFS strategy.



Relative increase compared to GADSE under **DFS**



Relative increase compared to GADSE under **BFS**

Thank you!

Q & A