ISSTA 2021 Online
ACM SIGSOFT International Symposium on Software Testing and Analysis

Grammar-Agnostic Symbolic Execution by Token Symbolization

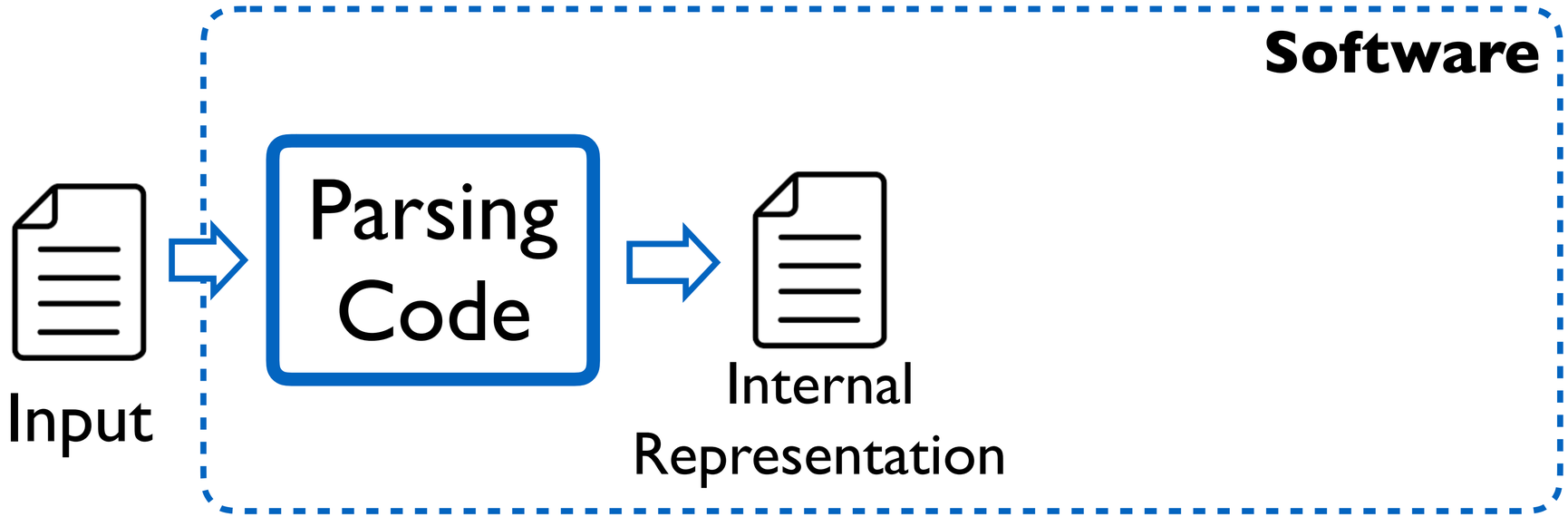
Zhenbang Chen
(zbchen@nudt.edu.cn)

Joint work with Weiyu Pan, Guofeng Zhang, Yunlai Luo, Yufeng Zhang, and Ji Wang

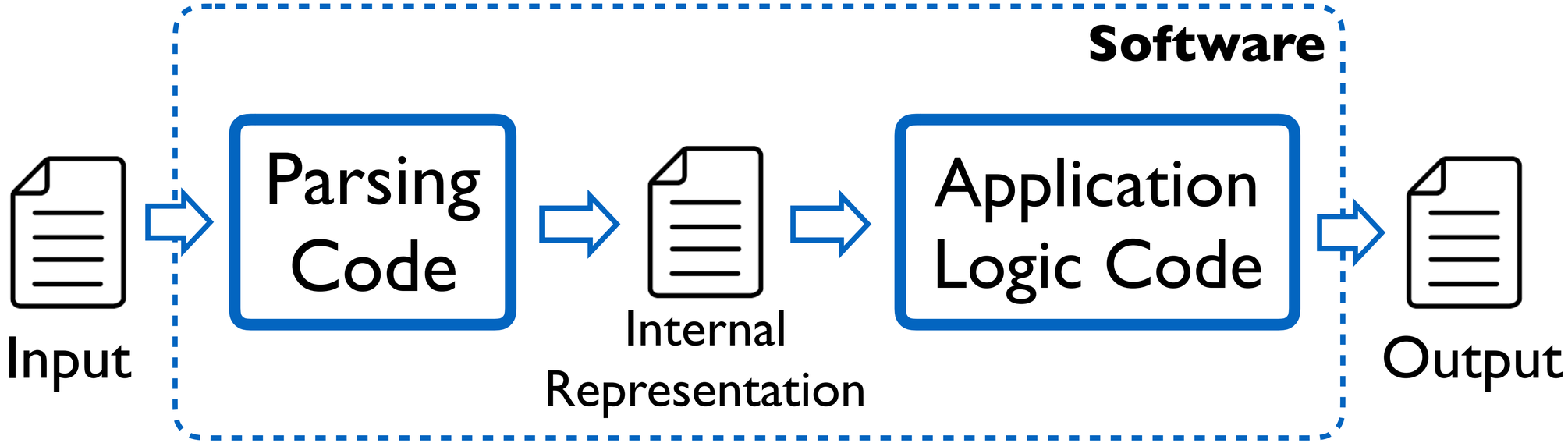
Parsing

Parsing

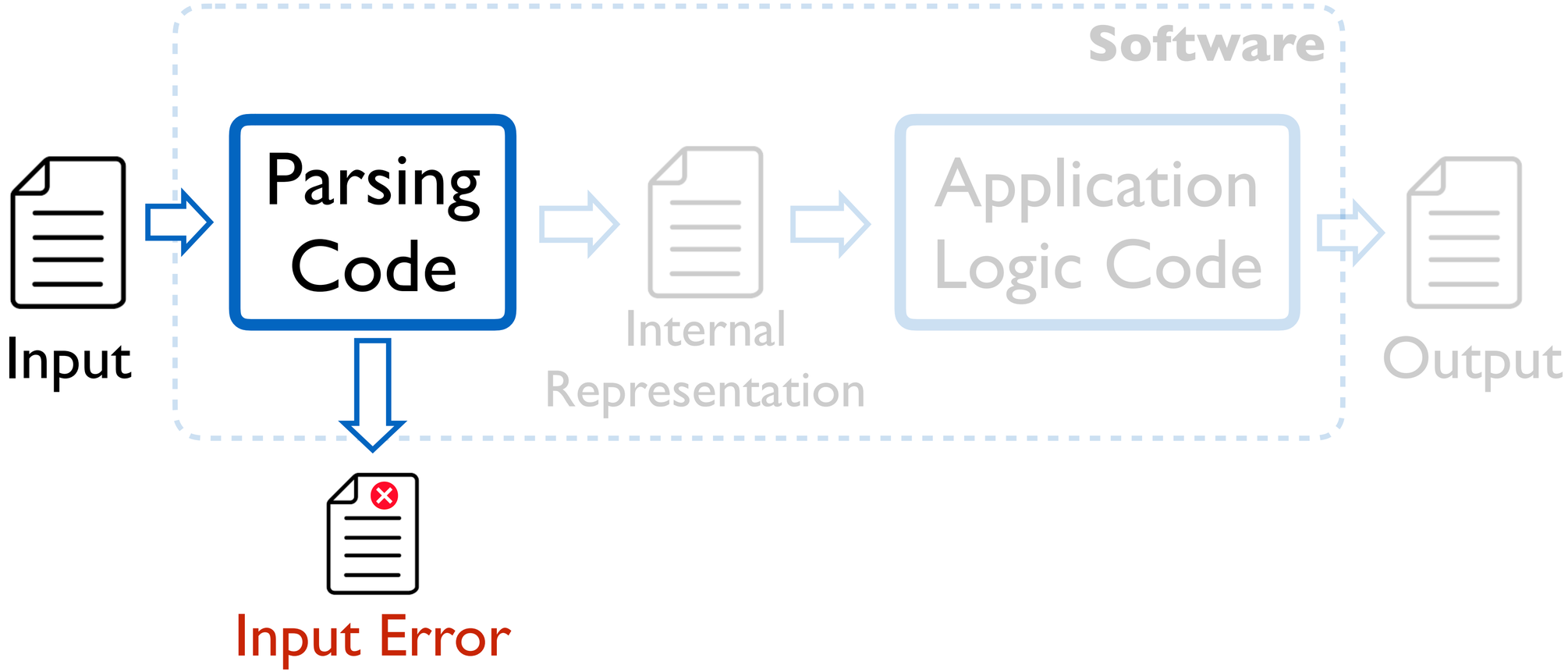
Parsing



Parsing


Software
Parsing Code
Application Logic Code
Input
Internal Representation
Output

Parsing


Software
Parsing Code
Application Logic Code
Input
Internal Representation
Output
Input Error

Parsing Code

Parsing code exists extensively

Parsing Code

Parsing code exists extensively

Compilers

. . .

Parsing Code

Parsing code exists extensively

Compilers

. . .

Web Browsers

. . .

Parsing Code

Parsing code exists extensively

Compilers

. . .

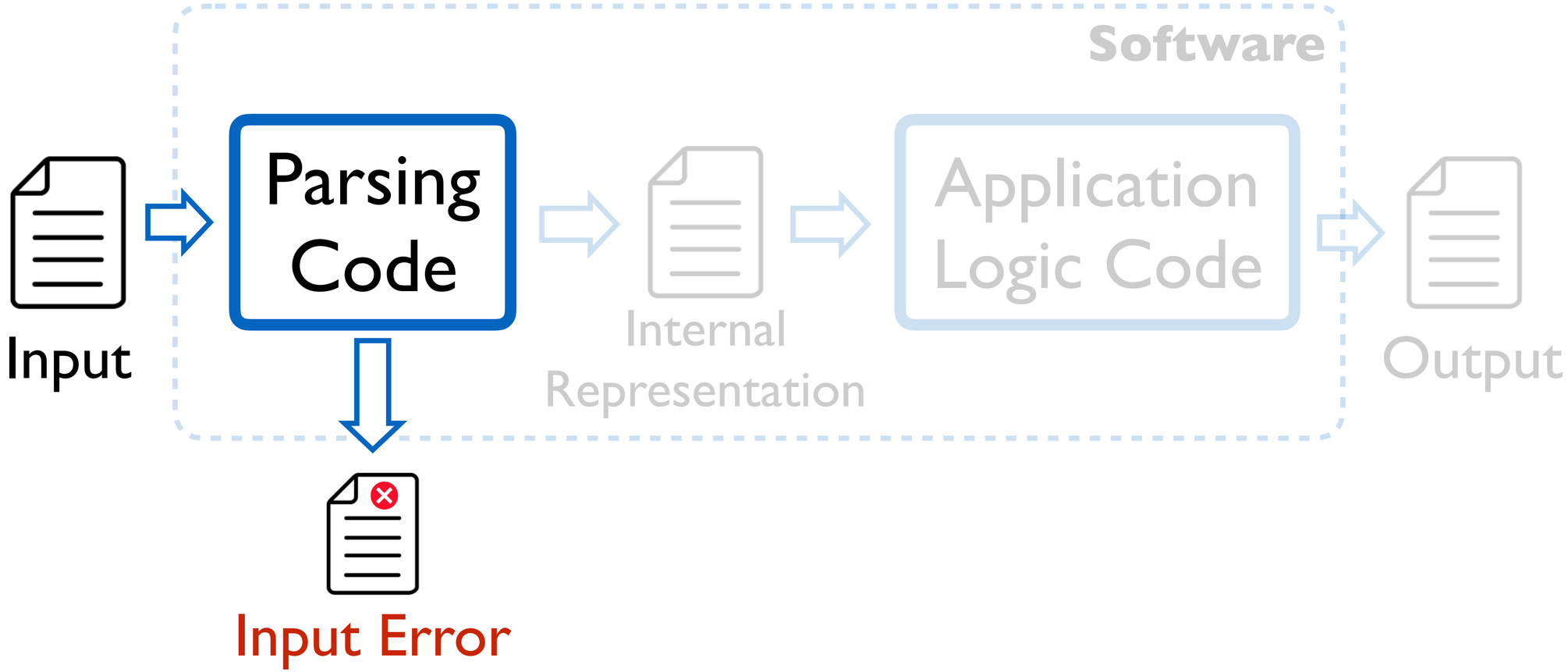
Web Browsers

. . .

Document Processing

. . .

Challenges in Testing Complex Parsing Code



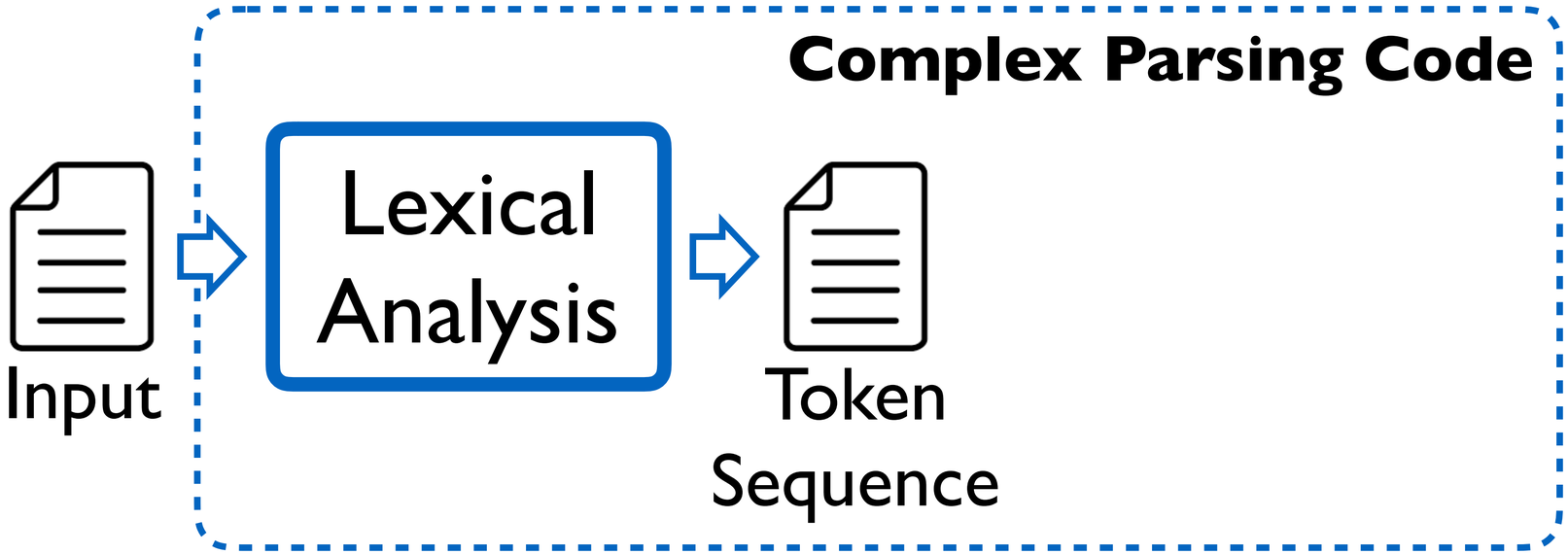
Challenges in Testing Complex Parsing Code

An input grammar usually exists

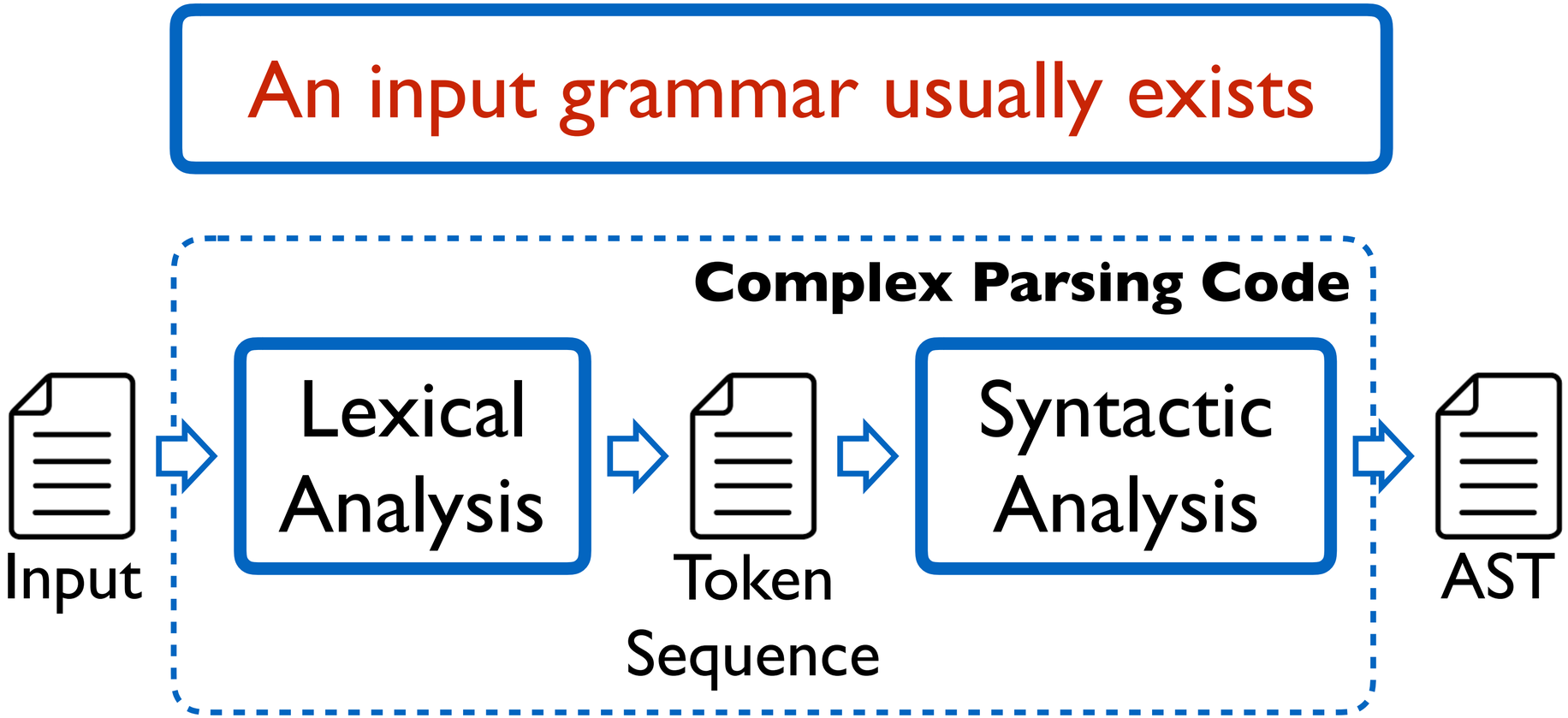
Complex Parsing Code

Challenges in Testing Complex Parsing Code

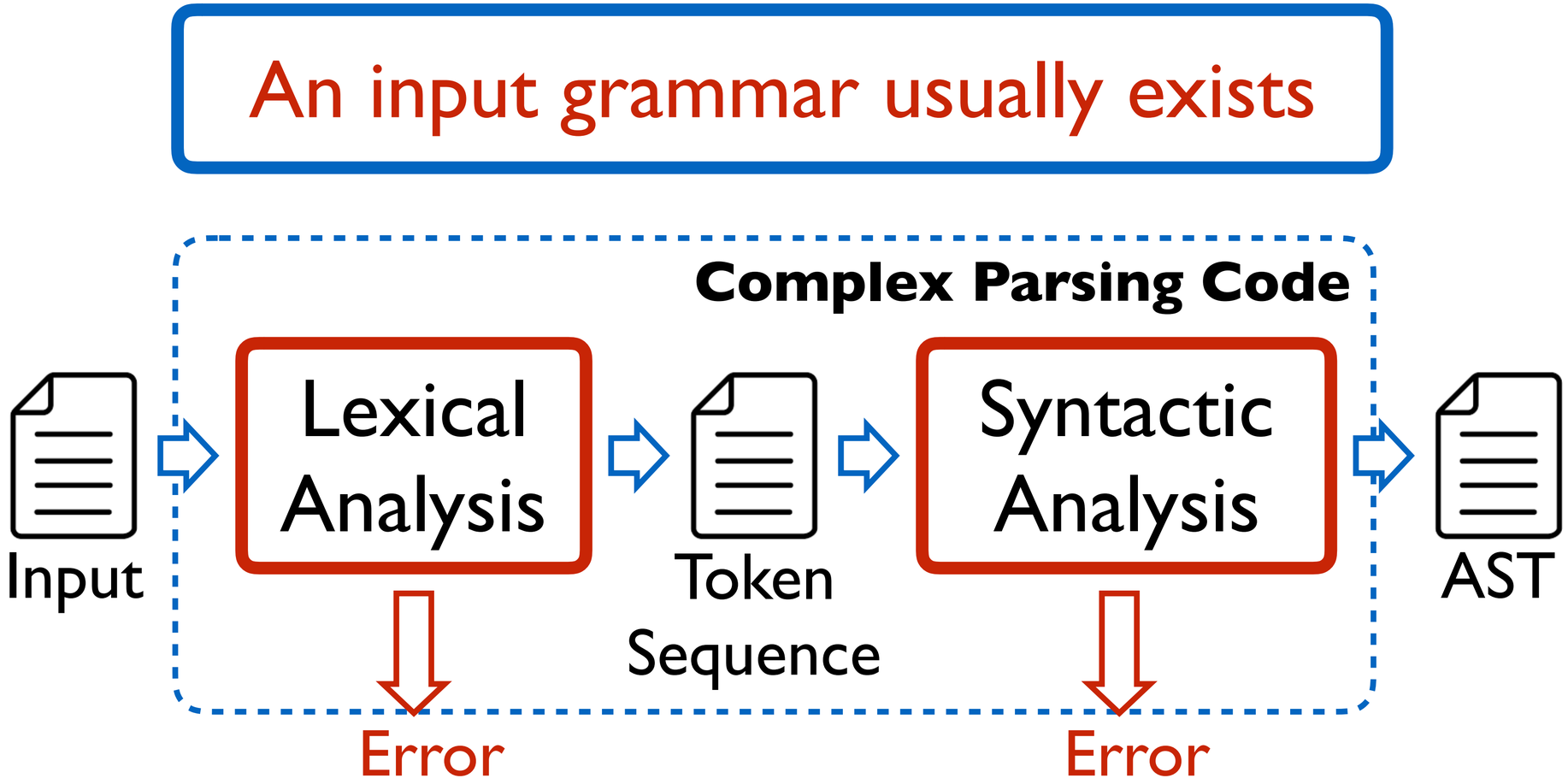
An input grammar usually exists



Challenges in Testing Complex Parsing Code



Challenges in Testing Complex Parsing Code



Existing Work

Existing Work

- Random testing
- Coverage-oriented fuzzing
  - AFL, JQF, …

Existing Work

- Random testing
- Coverage-oriented fuzzing
  - AFL, JQF, …

Not efficient

Existing Work

- Random testing
- Coverage-oriented fuzzing
  - AFL, JQF, …
- Grammar-directed fuzzing
  - Black-box fuzzing[ASE'19], Zest[ISSTA'19], …

Existing Work

- Random testing
- Coverage-oriented fuzzing
  - AFL, JQF, …
- Grammar-directed fuzzing
  - Black-box fuzzing[ASE'19], Zest[ISSTA'19], …
- Grammar-directed symbolic execution
  - Godefroid et al. [PLDI'08], CESE[ASE'07], …

Existing Work

- Random testing
- Coverage-oriented fuzzing
  - AFL, JQF, …
- Grammar-directed fuzzing
  - Black-box fuzzing[ASE'19], Zest[ISSTA'19], …
- Grammar-directed symbolic execution
  - Godefroid et al. [PLDI'08], CESE[ASE'07], …

Need Input Grammar

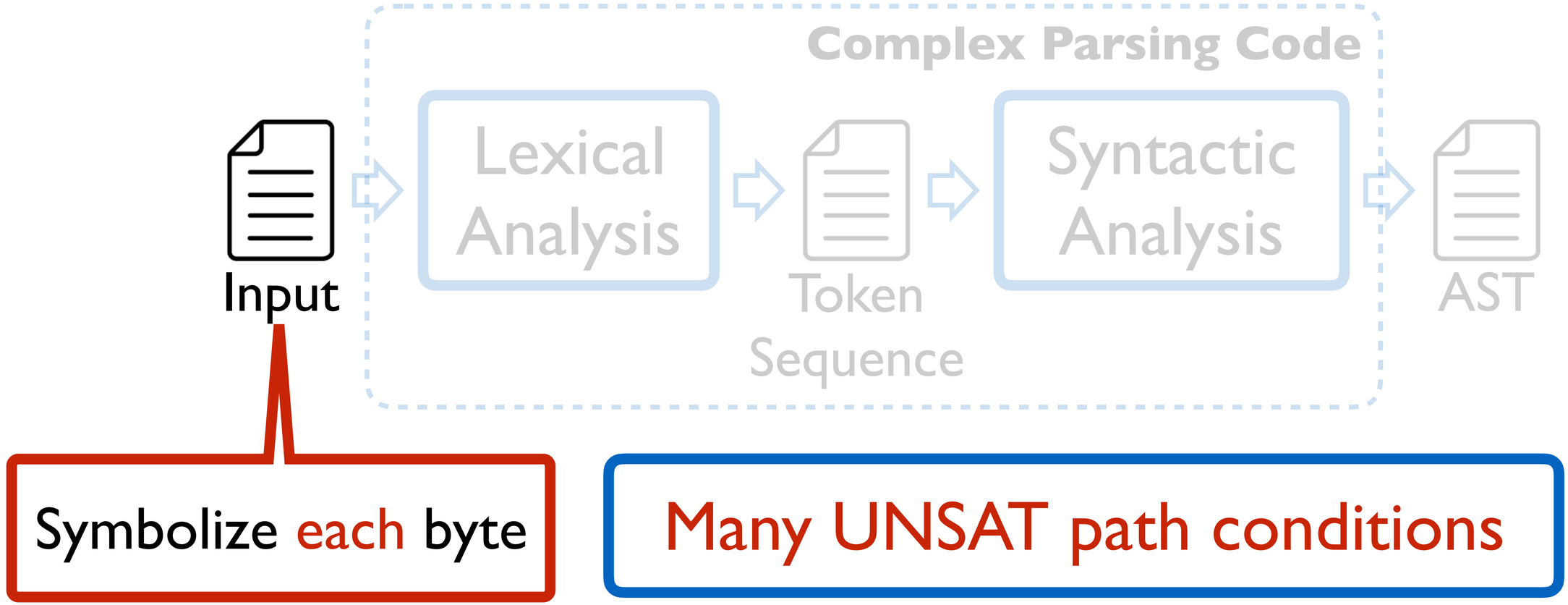
Existing Work

- Random testing
- Coverage-oriented fuzzing
  - AFL, JQF, …

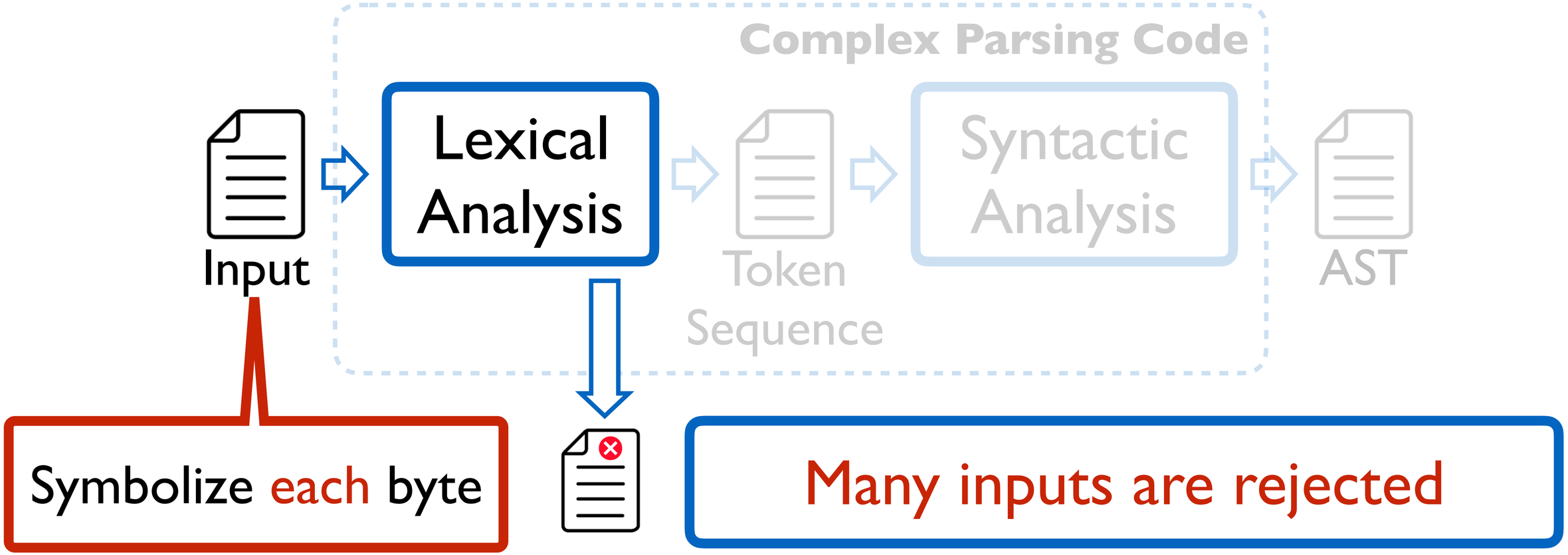
Need Input Grammar

- Grammar-directed fuzzing
  - Black-box fuzzing[ASE'19], Zest[ISSTA'19], …
- Grammar-directed symbolic execution
  - Godefroid et al. [PLDI'08], CESE[ASE'07], …

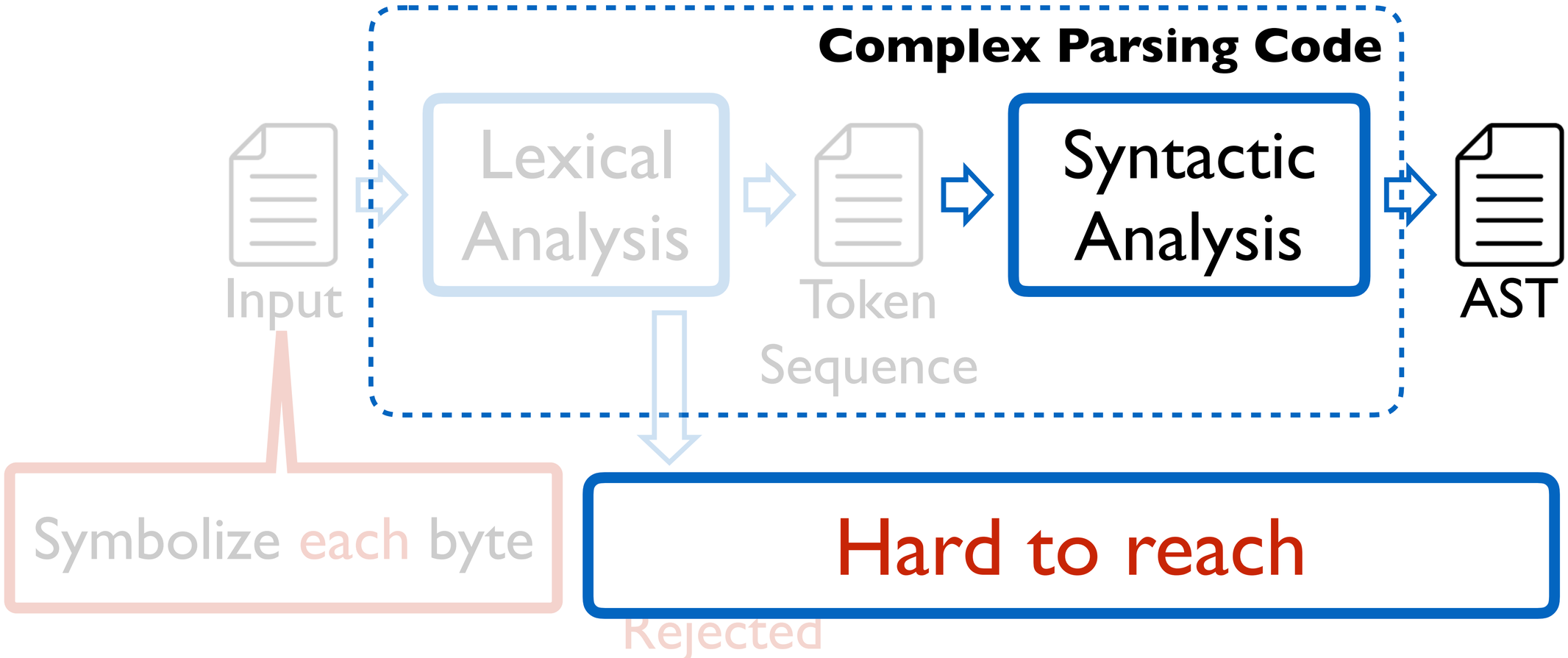
Symbolic Execution of Complex Parsing Programs without Grammar



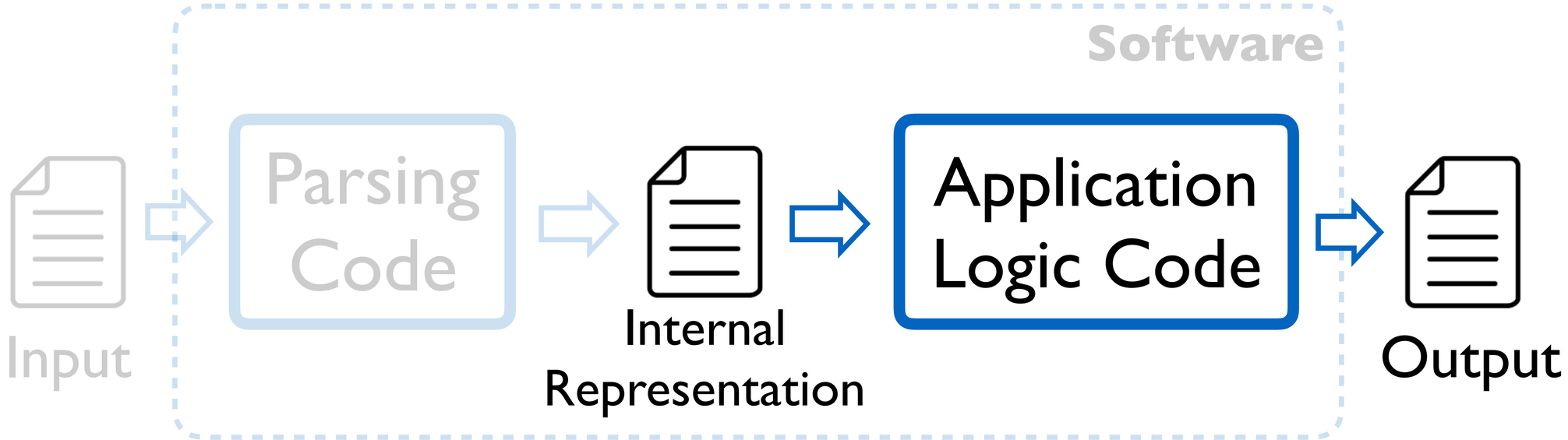
Symbolic Execution of Complex Parsing Programs without Grammar



Symbolic Execution of Complex Parsing Programs without Grammar

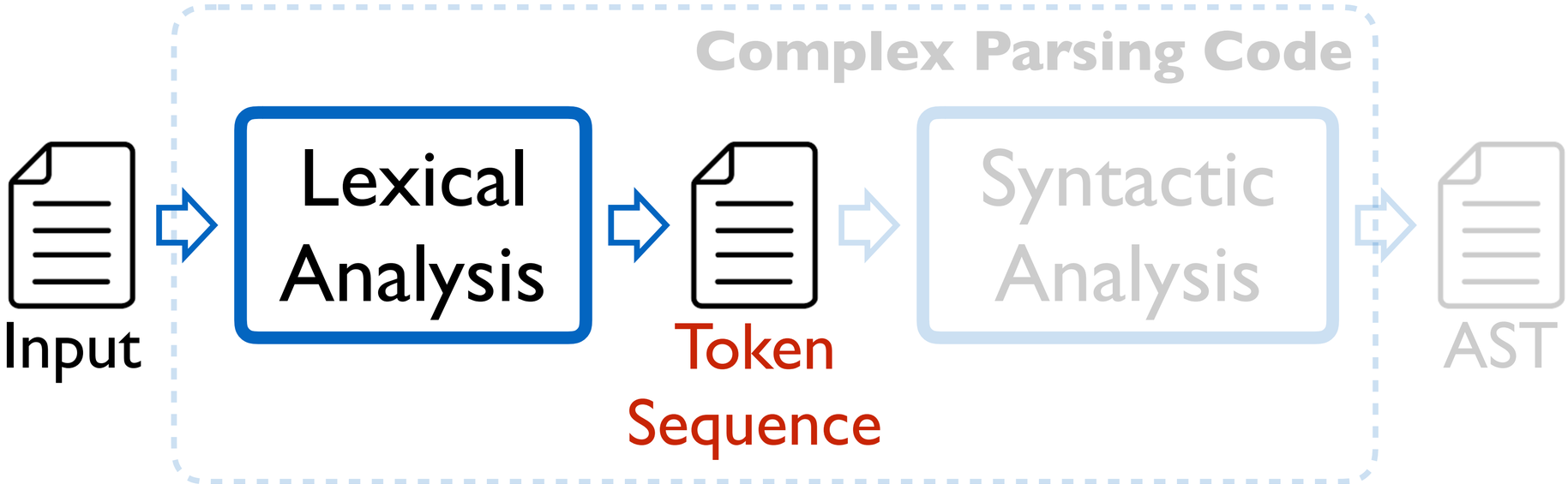


Symbolic Execution of Complex Parsing Programs without Grammar

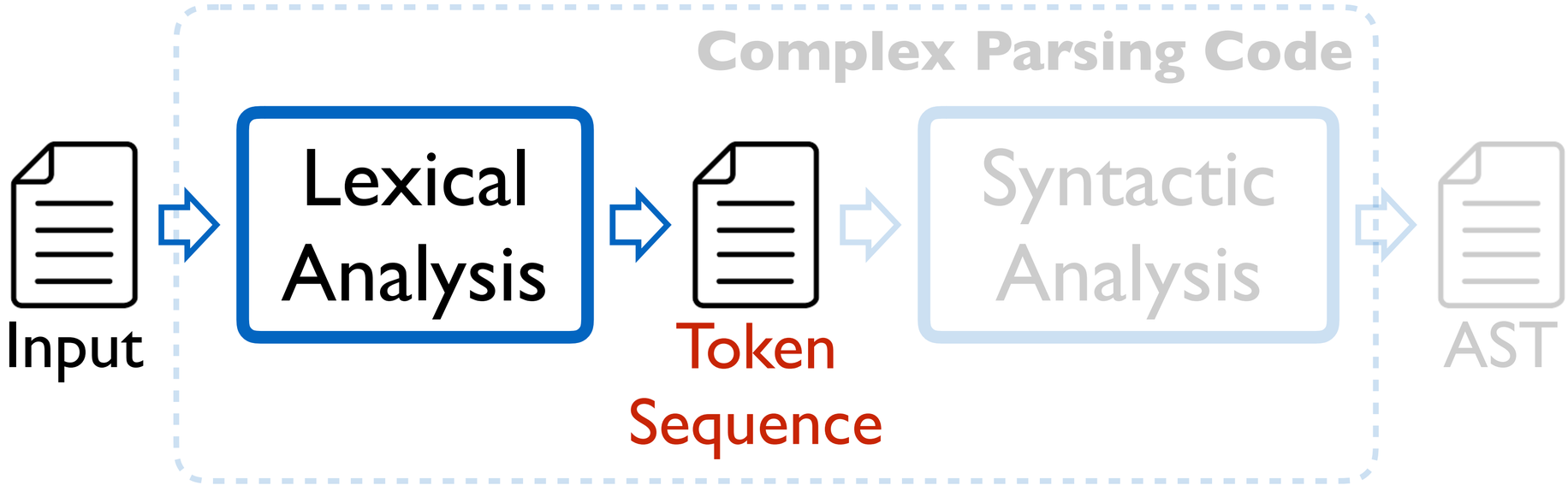


Analyze the application logic code is even harder

Key Insight

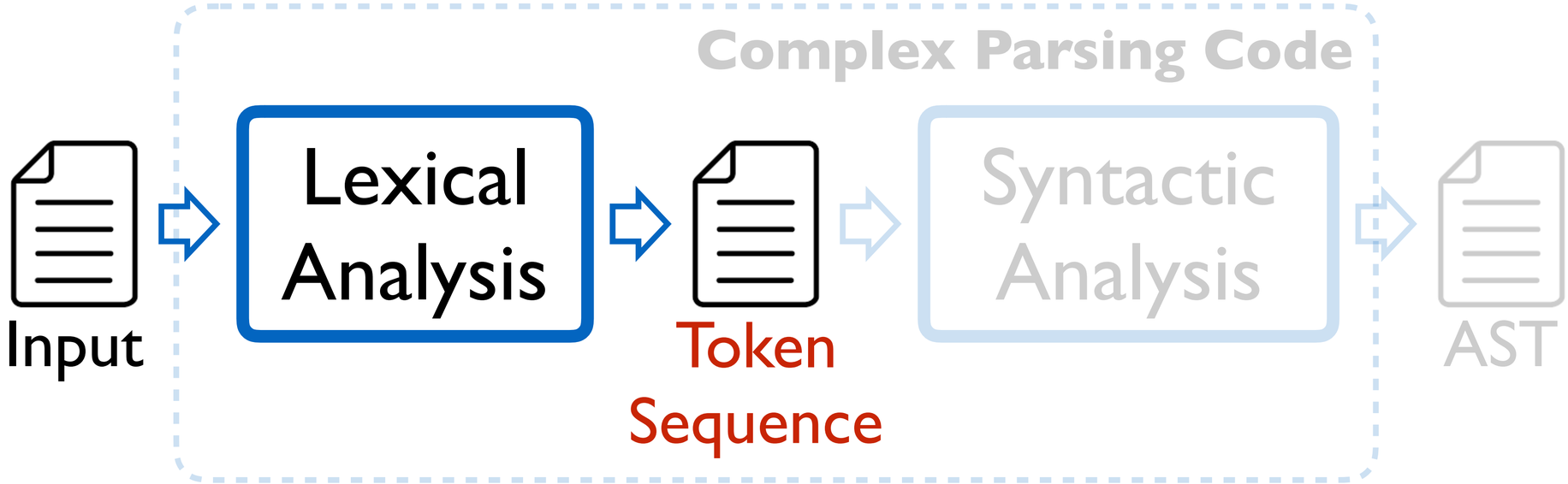

Complex Parsing Code
Lexical Analysis
Syntactic Analysis
Input
Token Sequence
AST

Key Insight



$\langle \text{EXP} \rangle \rightarrow \langle \text{NUM} \rangle \langle \text{OP} \rangle \langle \text{NUM} \rangle$

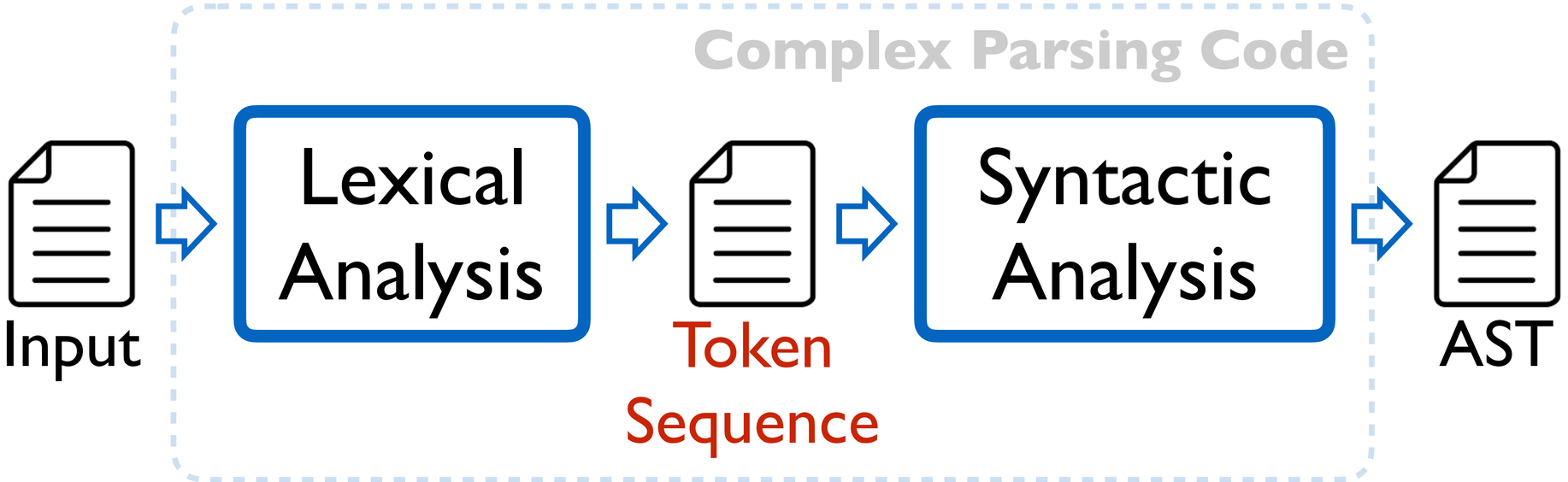
Key Insight



$\langle EXP \rangle \rightarrow \langle NUM \rangle \langle OP \rangle \langle NUM \rangle$

"11 + 22"

Key Insight

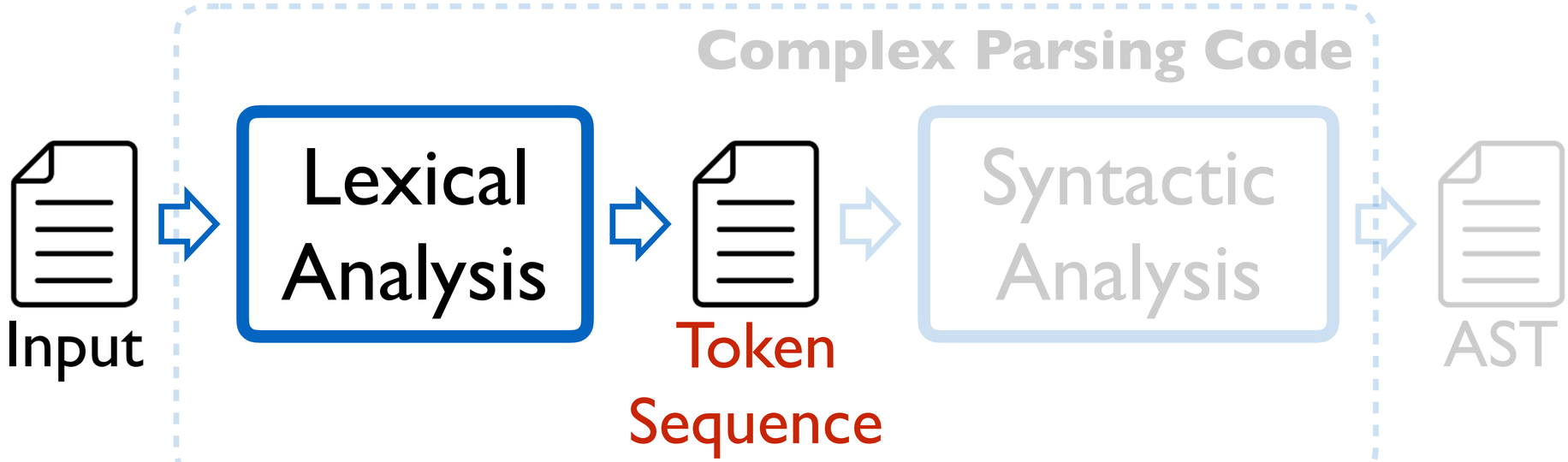


$\langle EXP \rangle \rightarrow \langle NUM \rangle \langle OP \rangle \langle NUM \rangle$

"11 + 22"

T_NUM T_OP T_NUM

Key Insight



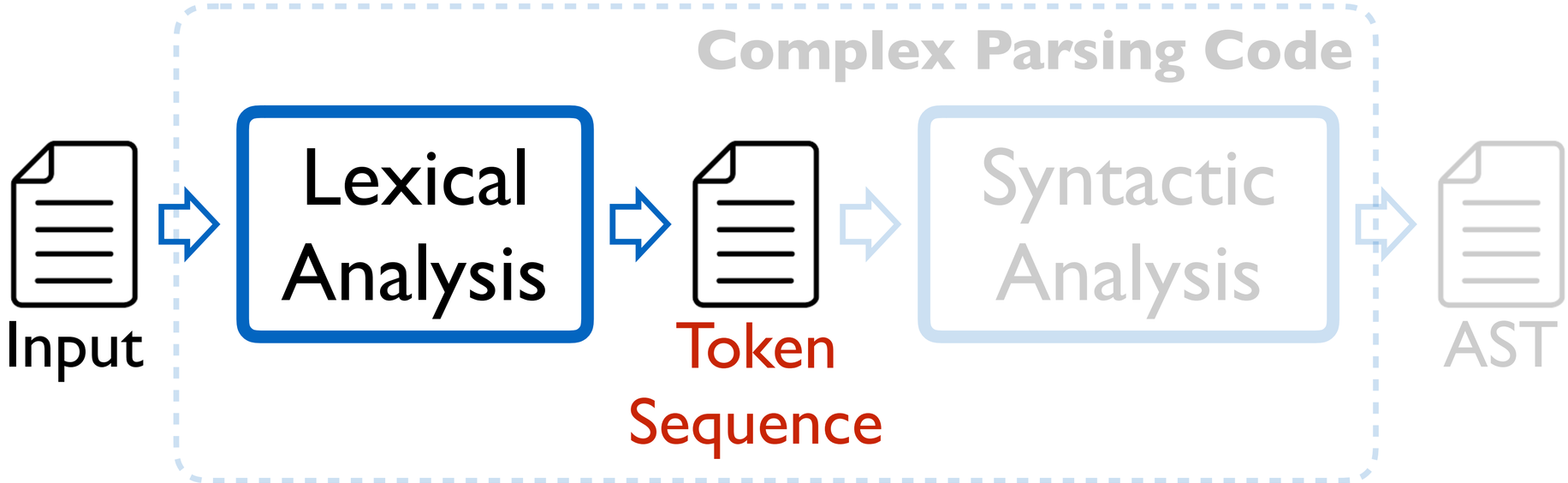
$\langle EXP \rangle \rightarrow \langle NUM \rangle \langle OP \rangle \langle NUM \rangle$

"11 + 22"

T_NUM T_OP T_NUM

A token value represents many byte-level values

Key Insight



⟨EXP⟩ → ⟨NUM⟩⟨OP⟩⟨NUM⟩

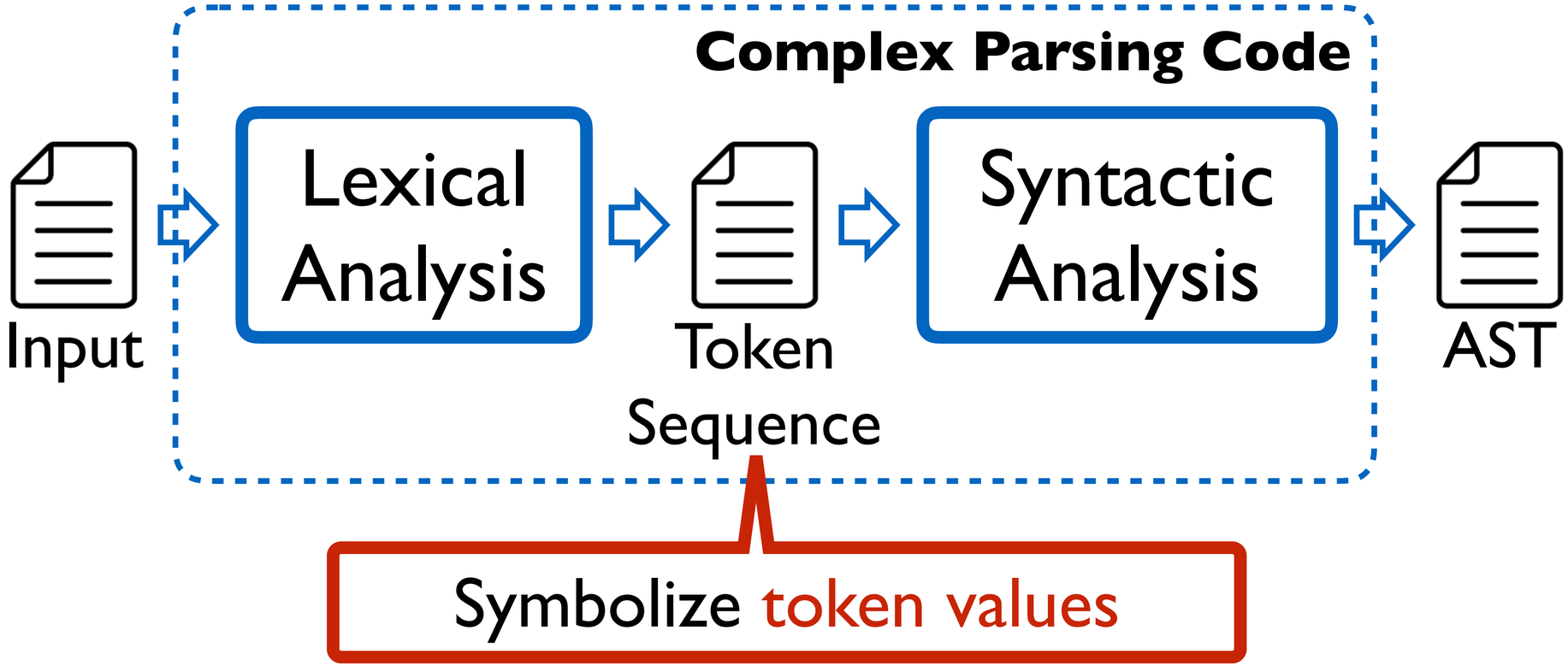
"11 + 22"

T_NUM T_OP T_NUM

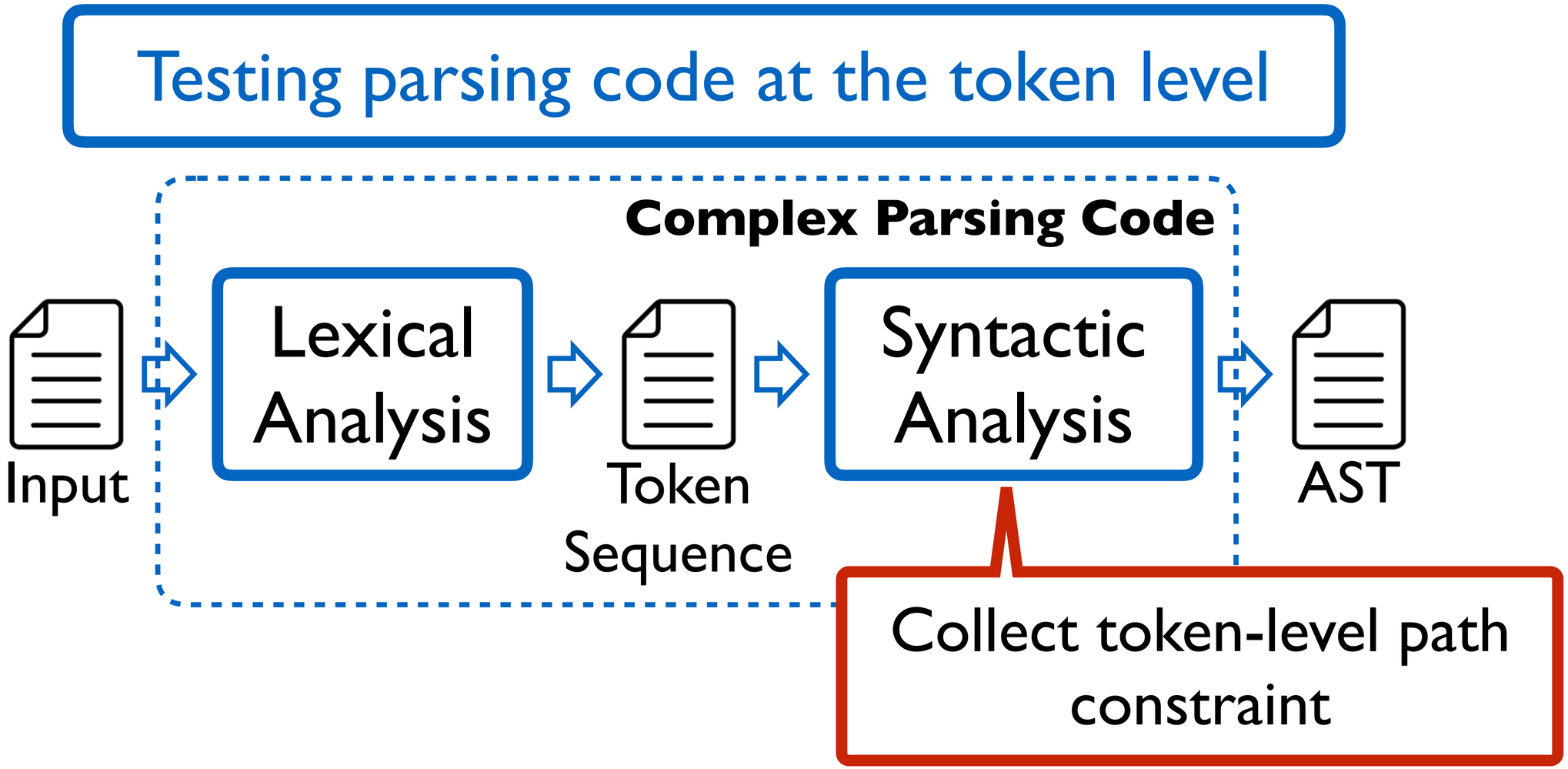
Token provides an abstraction for the input grammar

Key Idea

Testing parsing code at the token level



Key Idea


Testing parsing code at the token level
Complex Parsing Code
Input
Lexical Analysis
Token Sequence
Syntactic Analysis
AST
Collect token-level path constraint

Technique Problems

Technique Problems

- Input generation for a token sequence

T_NUM T_OP T_NUM

Technique Problems

- Input generation for a token sequence

T_NUM T_OP T_NUM

- Analyze the application code in priority

Technique Problems

- Input generation for a token sequence

Summarize the tokenization code first

- Analyze the application code in priority

Technique Problems

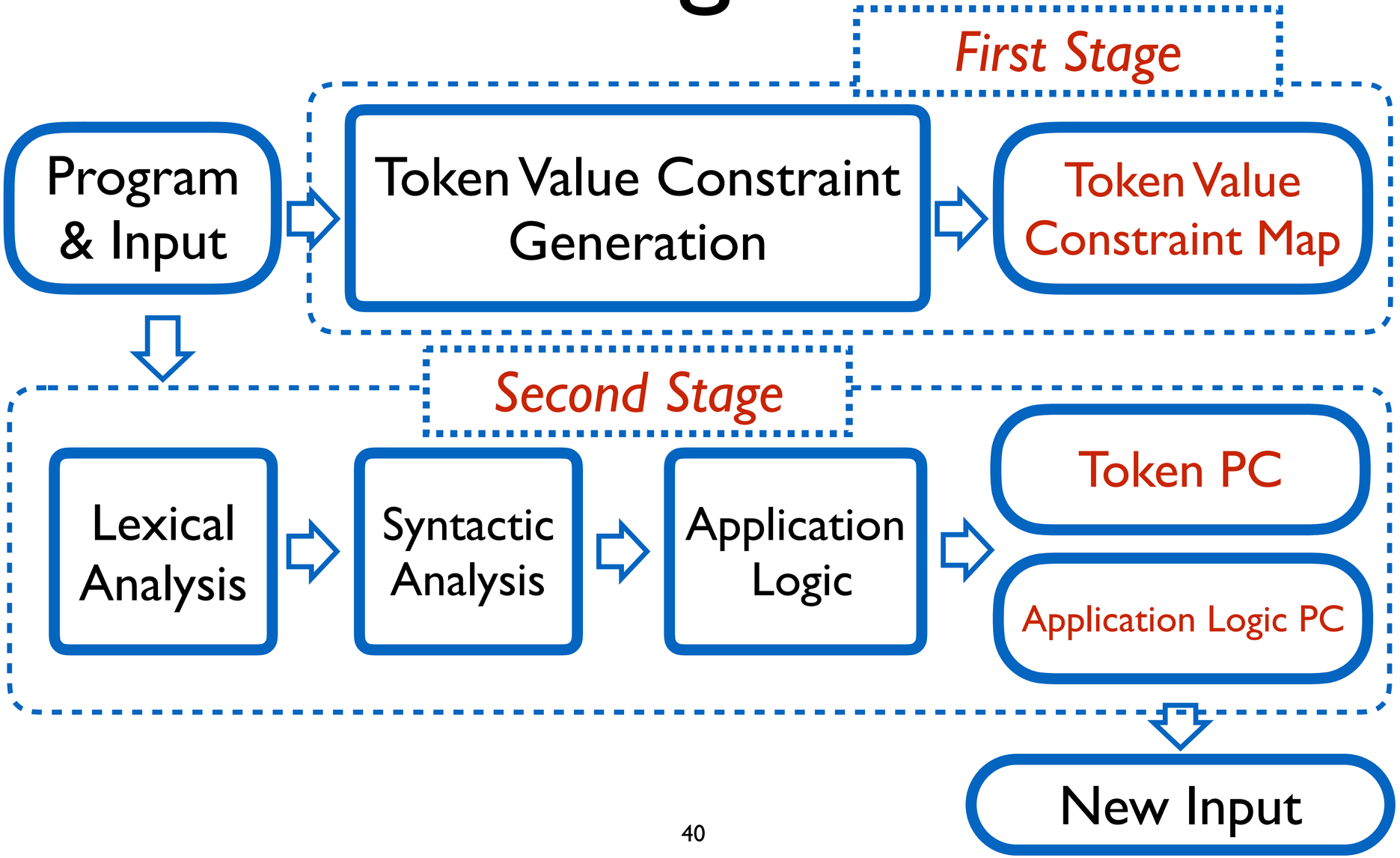
- Input generation for a token sequence

Summarize the tokenization code first

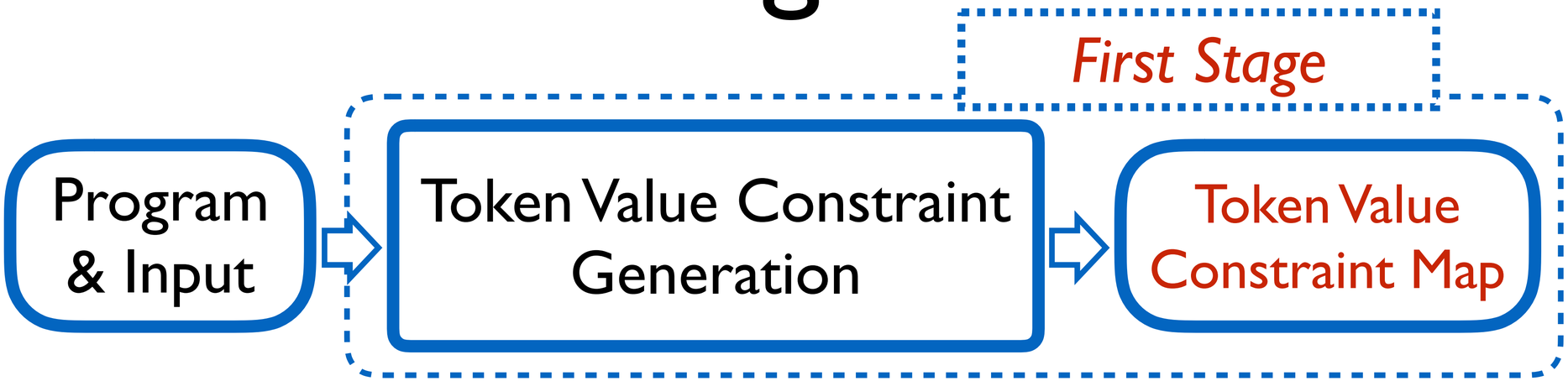
- Analyze the application code in priority

Record byte-level path constraint in application logic

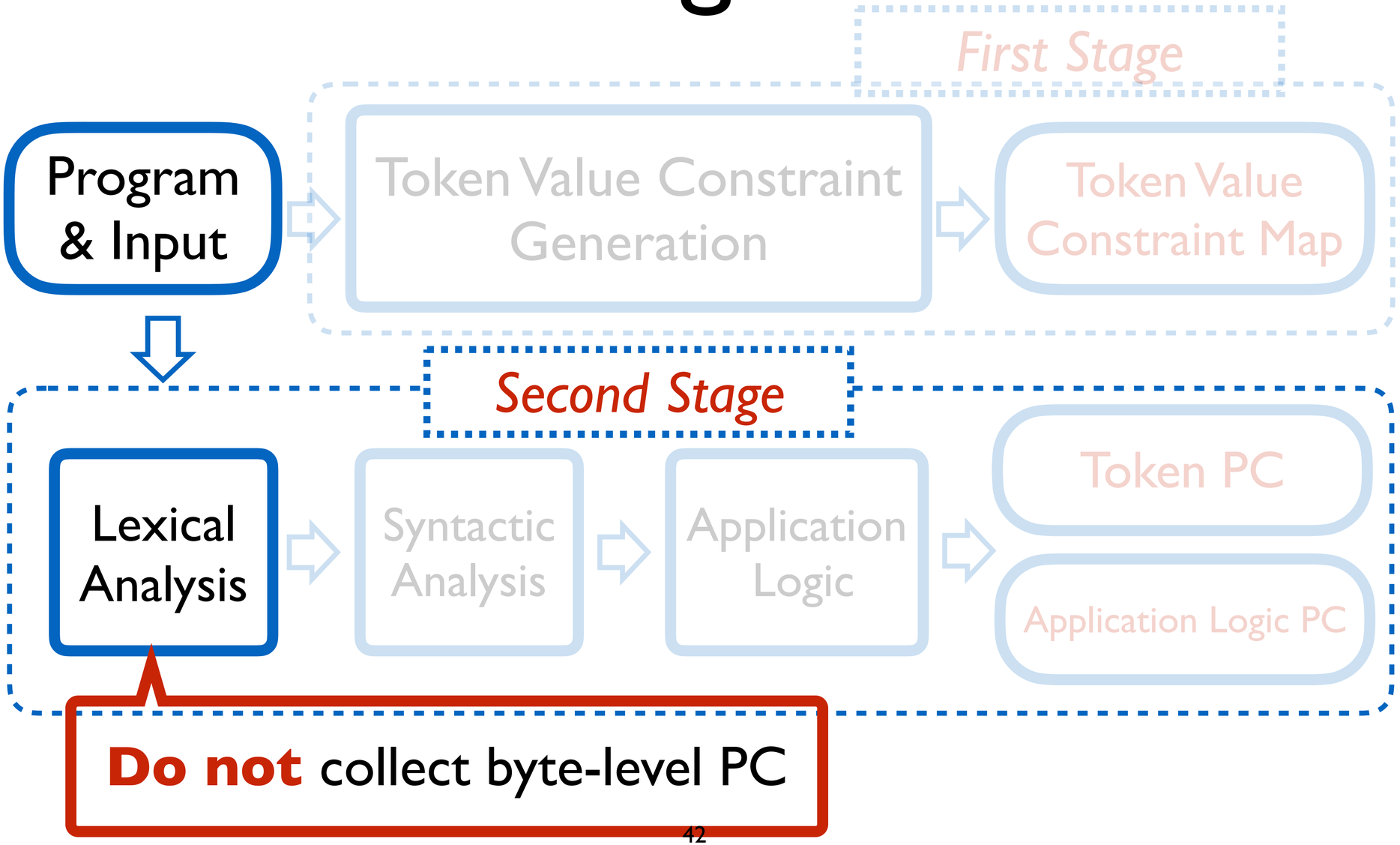
Grammar-agnostic DSE



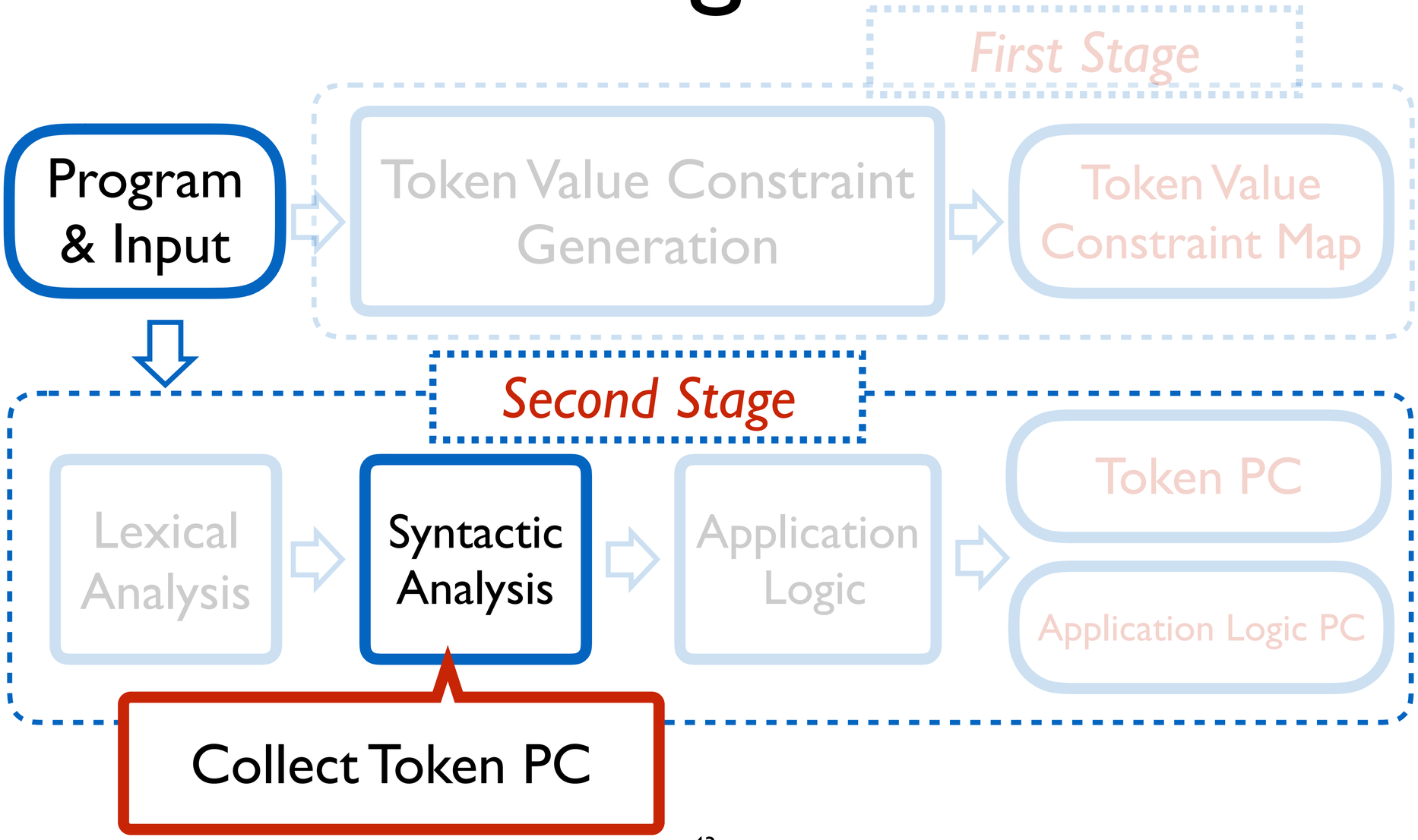
Grammar-agnostic DSE



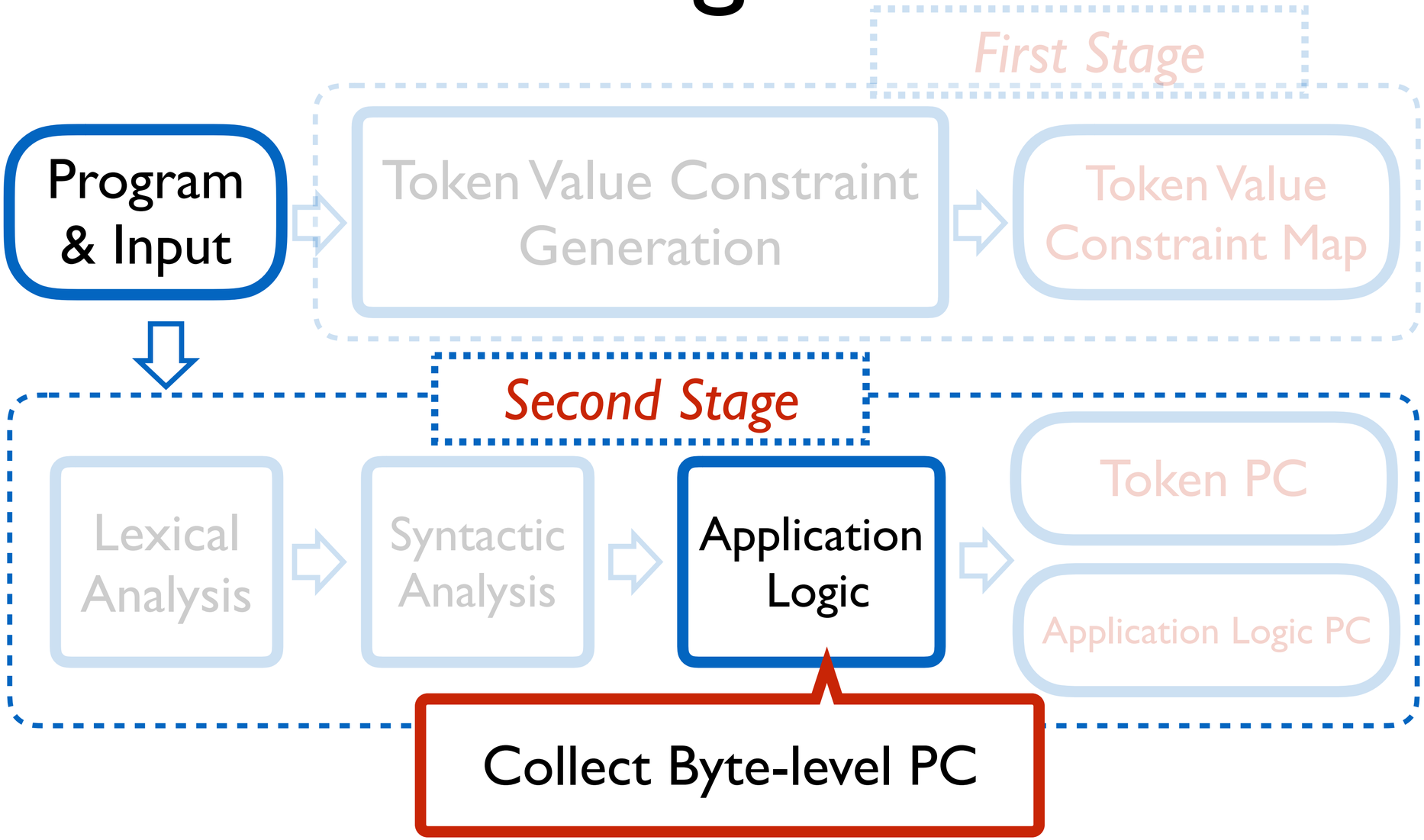
Grammar-agnostic DSE



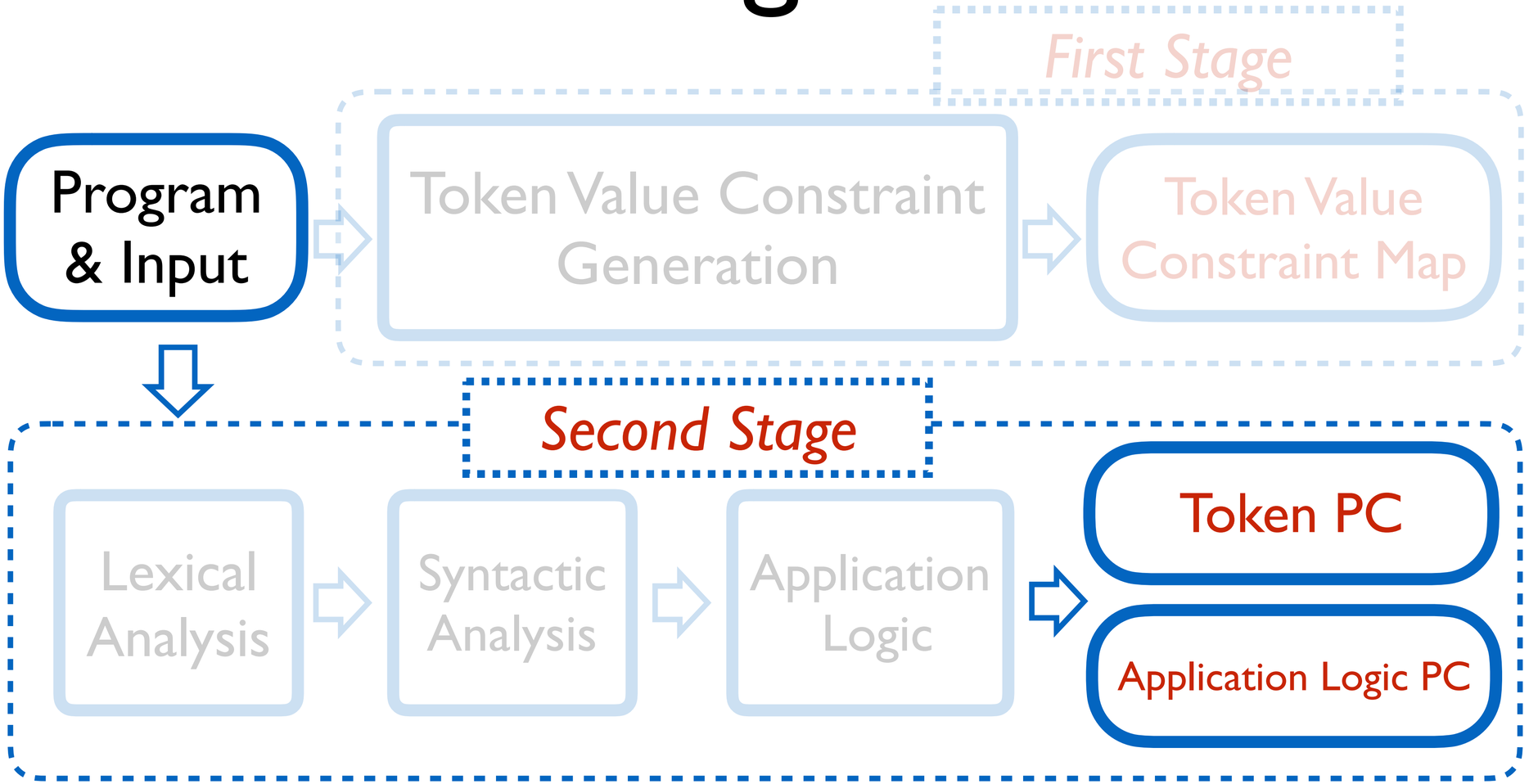
Grammar-agnostic DSE



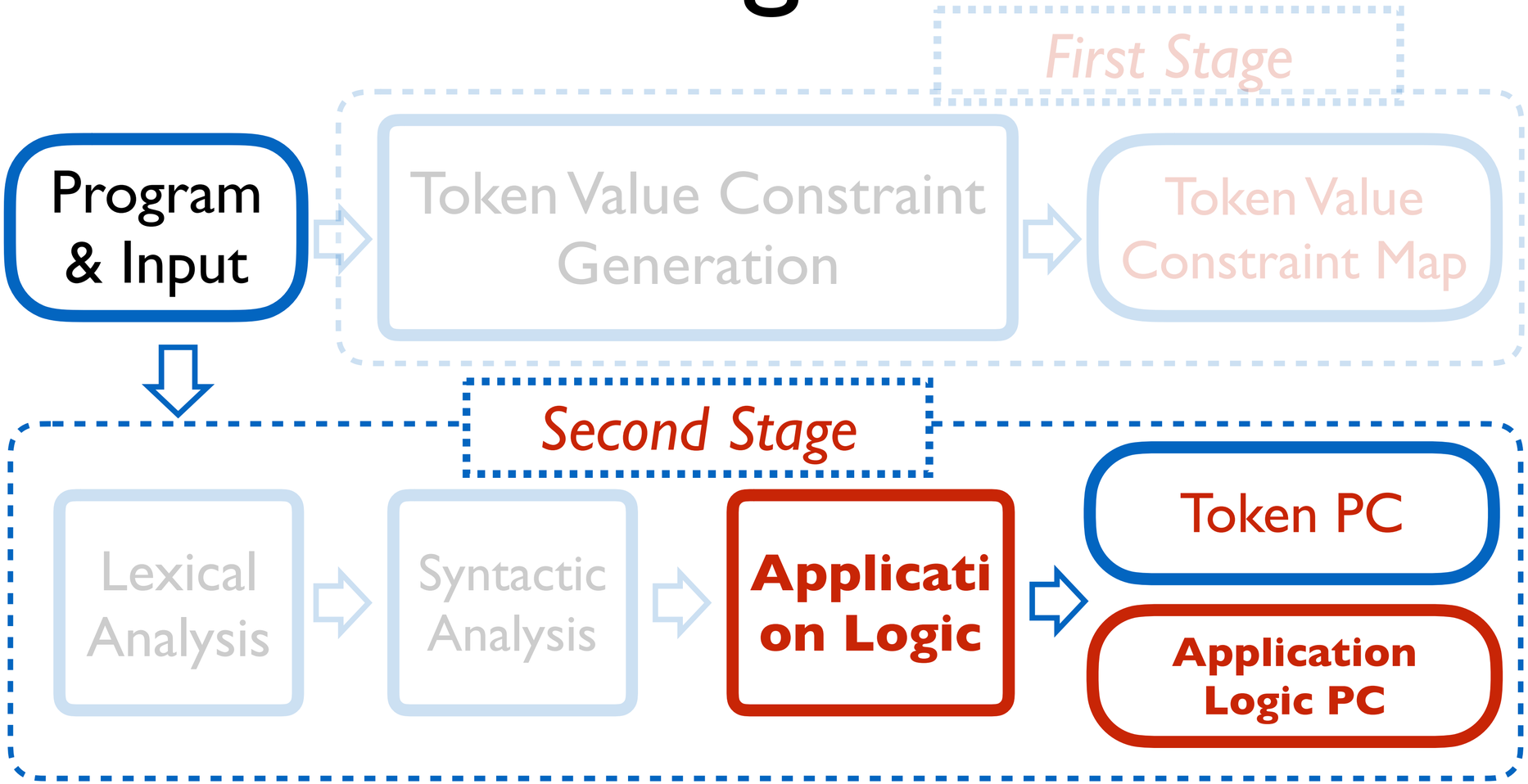
Grammar-agnostic DSE



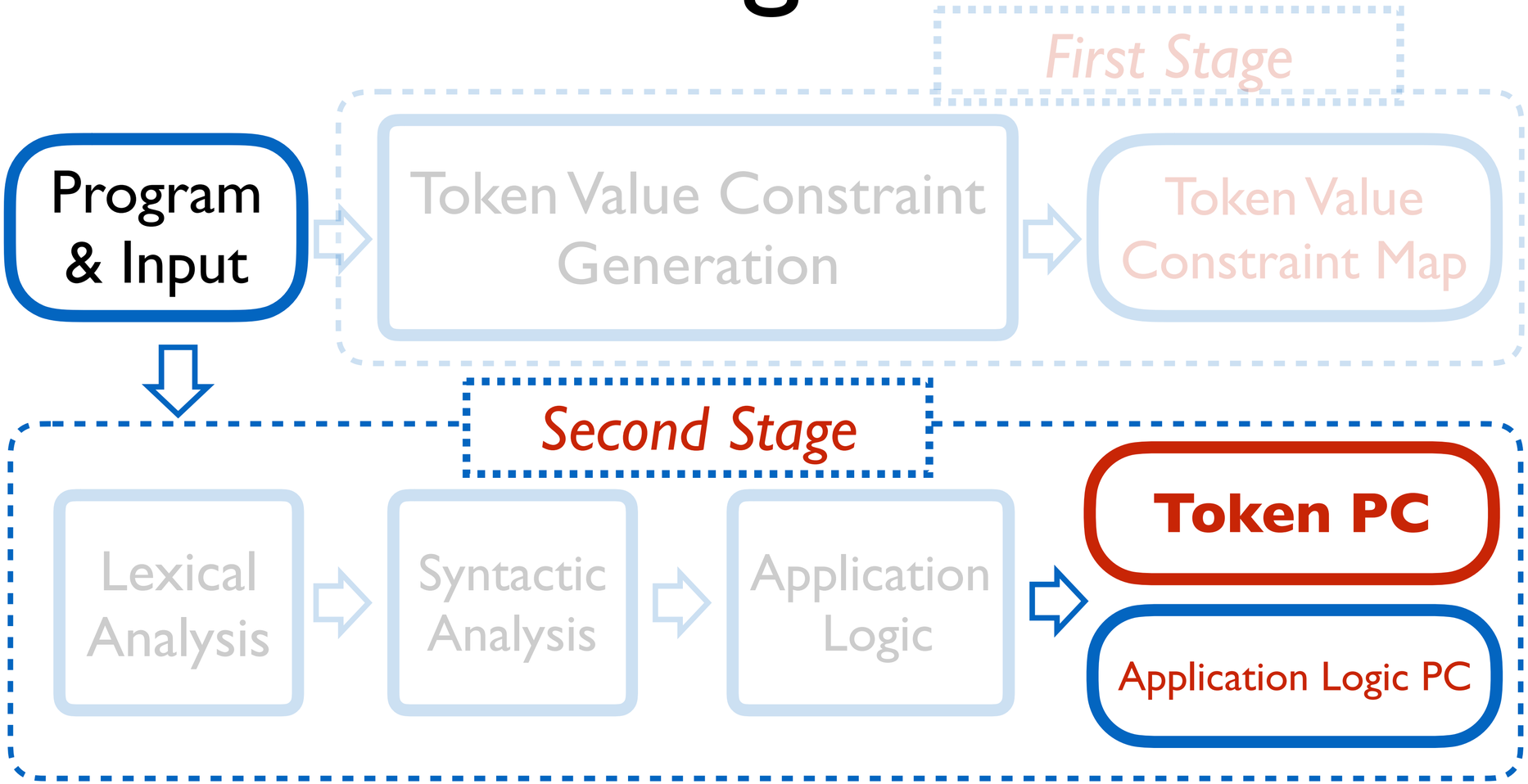
Grammar-agnostic DSE



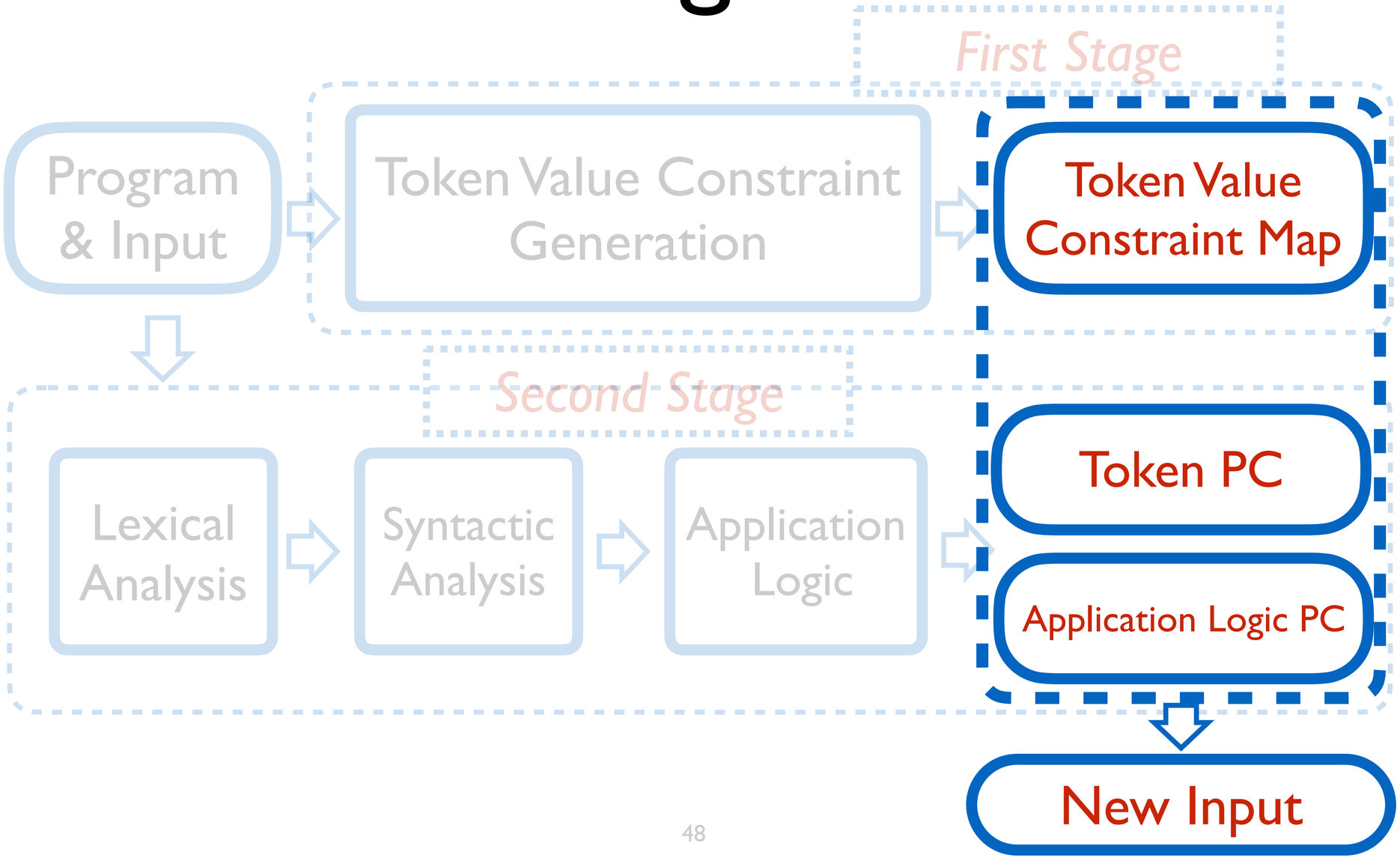
Grammar-agnostic DSE



Grammar-agnostic DSE



Grammar-agnostic DSE



Motivation Example

⟨Expr⟩ ::= ⟨ID⟩ | ⟨Number⟩ | ⟨Number⟩ ⟨Op⟩ ⟨Expr⟩
⟨ID⟩ ::= `'a'` | `'b'` | ... | `'y'` | `'z'`
⟨Number⟩ ::= `'00'` | `'01'` | ... | `'98'` | `'99'`
⟨Op⟩ ::= `'*'` | `'+'`

Motivation Example

⟨Expr⟩ ::= ⟨ID⟩ | ⟨Number⟩ | ⟨Number⟩ ⟨Op⟩ ⟨Expr⟩
⟨ID⟩ ::= `'a'` | `'b'` | ... | `'y'` | `'z'`
⟨Number⟩ ::= `'00'` | `'01'` | ... | `'98'` | `'99'`
⟨Op⟩ ::= `'*'` | `'+'`

11+12 31+a 49+z 11*a ...

Motivation Example

⟨Expr⟩ ::= ⟨ID⟩ | ⟨Number⟩ | ⟨Number⟩ ⟨Op⟩ ⟨Expr⟩
⟨ID⟩ ::= 'a' | 'b' | ... | 'y' | 'z'
⟨Number⟩ ::= '00' | '01' | ... | '98' | '99'
⟨Op⟩ ::= '*' | '+'

11+12 31+a 49+z 11*a …

11+1a 0*11 a-11 a+11 ⋯

Motivation Example

$$
\begin{aligned}
\langle \text{Expr} \rangle &::= \langle \text{ID} \rangle \mid \langle \text{Number} \rangle \mid \langle \text{Number} \rangle \ \langle \text{Op} \rangle \ \langle \text{Expr} \rangle \\
\langle \text{ID} \rangle &::= \texttt{'a'} \mid \texttt{'b'} \mid \ldots \mid \texttt{'y'} \mid \texttt{'z'} \\
\langle \text{Number} \rangle &::= \texttt{'00'} \mid \texttt{'01'} \mid \ldots \mid \texttt{'98'} \mid \texttt{'99'} \\
\langle \text{Op} \rangle &::= \texttt{'*'} \mid \texttt{'+'}
\end{aligned}
$$

```
1 public void entry(String a) throws ParseException{
2  // inputReader's type is Reader
3  inputReader = new StringReader(a);
4  parseExpr();
5  // application logic starts
6  if (a.charAt(a.length() - 1) == 'z') {
7   assert(false); //bug
8  }
9 }
```

Motivation Example

$$
\begin{aligned}
\langle \text{Expr} \rangle &::= \langle \text{ID} \rangle \mid \langle \text{Number} \rangle \mid \langle \text{Number} \rangle \ \langle \text{Op} \rangle \ \langle \text{Expr} \rangle \\
\langle \text{ID} \rangle &::= \texttt{'a'} \mid \texttt{'b'} \mid \ldots \mid \texttt{'y'} \mid \texttt{'z'} \\
\langle \text{Number} \rangle &::= \texttt{'00'} \mid \texttt{'01'} \mid \ldots \mid \texttt{'98'} \mid \texttt{'99'} \\
\langle \text{Op} \rangle &::= \texttt{'*'} \mid \texttt{'+'}
\end{aligned}
$$

```
1 public void entry(String a) throws ParseException{
2  // inputReader's type is Reader
3  inputReader = new StringReader(a);
4  parseExpr();
5  // application logic starts
6  if (a.charAt(a.length() - 1) == 'z') {
7   assert(false); //bug
8  }
9 }
```

Motivation Example

⟨Expr⟩ ::= ⟨ID⟩ | ⟨Number⟩ | ⟨Number⟩ ⟨Op⟩ ⟨Expr⟩

⟨ID⟩ ::= 'a' | 'b' | ... | 'y' | 'z'

⟨Number⟩ ::= '00' | '01' | ... | '98' | '99'

⟨Op⟩ ::= '*' | '+'

```
11 void parseExpr() throws ParseException {
12  int token = getNextToken();
13  if (token == T_NUM){
14   parseOp();
15   return;
16  } else if (token == T_ID){
17   if (getNextToken() == T_EOF) return;
18  }
19  throw new ParseException();
20 }
```

Motivation Example

```
32 int getNextToken() throws ParseException {
33  int res = inputReader.read();
34  if (res == -1) return T_EOF;
35  char c = (char) res;
36  if (c >= '*' && c <= '+'){
37   return T_OP;
38  } else if (c >= 'a' && c <= 'z'){
39   return T_ID;
40  } else if (c >= '0' && c <= '9'){
41   char next_c = (char) inputReader.read();
42   if (next_c >= '0' && next_c <= '9'){
43    return T_NUM;
44   }
45  }
46  throw new ParseException();
47 }
```

⟨Expr⟩ ::= ⟨ID⟩ | ⟨Number⟩ | ⟨Number⟩ ⟨Op⟩ ⟨Expr⟩

⟨ID⟩ ::= 'a' | 'b' | ... | 'y' | 'z'

⟨Number⟩ ::= '00' | '01' | ... | '98' | '99'

⟨Op⟩ ::= '*' | '+'

T_ID

T_NUM

T_OP

Pure DSE

Motivation Example

- Pure DSE

Input: "12+13"

```
32 int getNextToken() throws ParseException {
33  int res = inputReader.read();
34  if (res == -1) return T_EOF;
35  char c = (char) res;
36  if (c >= '*' && c <= '+'){
37   return T_OP;
38  } else if (c >= 'a' && c <= 'z'){
39   return T_ID;
40  } else if (c >= '0' && c <= '9'){
41   char next_c = (char) inputReader.read();
42   if (next_c >= '0' && next_c <= '9'){
43    return T_NUM;
44   }
45  }
46  throw new ParseException();
47 }
```

First Path's Path Condition

$$
\begin{aligned}
&a[0] \geq \text{'*'} \wedge a[0] > \text{'+'} \wedge a[0] < \text{'a'} \wedge \\
&a[0] \geq \text{'0'} \wedge a[0] \leq \text{'9'} \wedge a[1] \geq \text{'0'} \wedge a[1] \leq \text{'9'} \wedge \\
&a[2] \geq \text{'*'} \wedge a[2] \leq \text{'+'} \wedge \\
&a[3] \geq \text{'*'} \wedge a[3] > \text{'+'} \wedge a[3] < \text{'a'} \wedge \\
&a[3] \geq \text{'0'} \wedge a[3] \leq \text{'9'} \wedge a[4] \geq \text{'0'} \wedge a[4] \leq \text{'9'} \wedge \\
&a[4] \neq \text{'z'}
\end{aligned}
$$

Motivation Example

- Pure DSE

Input: "12+13"

First Path's Path Condition

$$
\begin{aligned}
&a[0] \geq \text{'*'} \wedge a[0] > \text{'+'} \wedge a[0] < \text{'a'} \wedge \\
&a[0] \geq \text{'0'} \wedge a[0] \leq \text{'9'} \wedge a[1] \geq \text{'0'} \wedge a[1] \leq \text{'9'} \wedge \\
&a[2] \geq \text{'*'} \wedge a[2] \leq \text{'+'} \wedge \\
&a[3] \geq \text{'*'} \wedge a[3] > \text{'+'} \wedge a[3] < \text{'a'} \wedge \\
&a[3] \geq \text{'0'} \wedge a[3] \leq \text{'9'} \wedge a[4] \geq \text{'0'} \wedge a[4] \leq \text{'9'} \wedge \\
&a[4] \neq \text{'z'}
\end{aligned}
$$

Negation in DFS style

Motivation Example

- Pure DSE

Input: "12+13"

First Path's Path Condition

$$a[0] \geq \text{'*'} \wedge a[0] > \text{'+'} \wedge a[0] < \text{'a'} \wedge$$
$$a[0] \geq \text{'0'} \wedge a[0] \leq \text{'9'} \wedge a[1] \geq \text{'0'} \wedge a[1] \leq \text{'9'} \wedge$$
$$a[2] \geq \text{'*'} \wedge a[2] \leq \text{'+'} \wedge$$
$$a[3] \geq \text{'*'} \wedge a[3] > \text{'+'} \wedge a[3] < \text{'a'} \wedge$$
$$a[3] \geq \text{'0'} \wedge a[3] \leq \text{'9'} \wedge a[4] \geq \text{'0'} \wedge a[4] \leq \text{'9'} \wedge$$
$$a[4] = \text{'z'}$$

Motivation Example

- Pure DSE

Input: "12+13"

First Path's Path Condition

$$a[0] \geq \text{'*'} \wedge a[0] > \text{'+'} \wedge a[0] < \text{'a'} \wedge$$
$$a[0] \geq \text{'0'} \wedge a[0] \leq \text{'9'} \wedge a[1] \geq \text{'0'} \wedge a[1] \leq \text{'9'} \wedge$$
$$a[2] \geq \text{'*'} \wedge a[2] \leq \text{'+'} \wedge$$
$$a[3] \geq \text{'*'} \wedge a[3] > \text{'+'} \wedge a[3] < \text{'a'} \wedge$$
$$a[3] \geq \text{'0'} \wedge a[3] \leq \text{'9'} \wedge a[4] \geq \text{'0'} \wedge a[4] \leq \text{'9'} \wedge$$
$$a[4] = \text{'z'}$$

Unsatisfiable

Motivation Example

- Pure DSE

```
⟨Expr⟩   ::=  ⟨ID⟩ | ⟨Number⟩ | ⟨Number⟩ ⟨Op⟩ ⟨Expr⟩
⟨ID⟩     ::=  'a' | 'b' | ... | 'y' | 'z'
⟨Number⟩ ::=  '00' | '01' | ... | '98' | '99'
⟨Op⟩     ::=  '*' | '+'
```

Input: "12+13"

First Path's Path Condition

$$a[0] \geq \text{'*'} \wedge a[0] > \text{'+'} \wedge a[0] < \text{'a'} \wedge$$
$$a[0] \geq \text{'0'} \wedge a[0] \leq \text{'9'} \wedge a[1] \geq \text{'0'} \wedge a[1] \leq \text{'9'} \wedge$$
$$a[2] \geq \text{'*'} \wedge a[2] \leq \text{'+'} \wedge$$
$$a[3] \geq \text{'*'} \wedge a[3] > \text{'+'} \wedge a[3] < \text{'a'} \wedge$$
$$a[3] \geq \text{'0'} \wedge a[3] \leq \text{'9'} \wedge a[4] \geq \text{'0'} \wedge \boxed{a[4] > \text{'9'}}$$

Motivation Example

```
⟨Expr⟩   ::=  ⟨ID⟩ | ⟨Number⟩ | ⟨Number⟩ ⟨Op⟩ ⟨Expr⟩
⟨ID⟩     ::=  'a' | 'b' | ... | 'y' | 'z'
⟨Number⟩ ::=  '00' | '01' | ... | '98' | '99'
⟨Op⟩     ::=  '*' | '+'
```

- Pure DSE

Input: "12+13"

First Path's Path Condition

$$a[0] \geq '*' \wedge a[0] > '+' \wedge a[0] < 'a' \wedge$$
$$a[0] \geq '0' \wedge a[0] \leq '9' \wedge a[1] \geq '0' \wedge a[1] \leq '9' \wedge$$
$$a[2] \geq '*' \wedge a[2] \leq '+' \wedge$$
$$a[3] \geq '*' \wedge a[3] > '+' \wedge a[3] < 'a' \wedge$$
$$a[3] \geq '0' \wedge a[3] \leq '9' \wedge a[4] \geq '0' \wedge \boxed{a[4] > '9'}$$

New Input: "12+1z"

Motivation Example

```
⟨Expr⟩   ::=  ⟨ID⟩ | ⟨Number⟩ | ⟨Number⟩ ⟨Op⟩ ⟨Expr⟩
⟨ID⟩     ::=  'a' | 'b' | ... | 'y' | 'z'
⟨Number⟩ ::=  '00' | '01' | ... | '98' | '99'
⟨Op⟩     ::=  '*' | '+'
```

- Pure DSE

Input: "12+13"

First Path's Path Condition

$$a[0] \geq '*' \wedge a[0] > '+' \wedge a[0] < 'a' \wedge$$
$$a[0] \geq '0' \wedge a[0] \leq '9' \wedge a[1] \geq '0' \wedge a[1] \leq '9' \wedge$$
$$a[2] \geq '*' \wedge a[2] \leq '+' \wedge$$
$$a[3] \geq '*' \wedge a[3] > '+' \wedge a[3] < 'a' \wedge$$
$$a[3] \geq '0' \wedge a[3] \leq '9' \wedge a[4] \geq '0' \wedge \boxed{a[4] > '9'}$$

New Input: "12+1z"

New input will be rejected

Motivation Example

- Pure DSE

```
⟨Expr⟩   ::=  ⟨ID⟩ | ⟨Number⟩ | ⟨Number⟩ ⟨Op⟩ ⟨Expr⟩
  ⟨ID⟩   ::=  'a' | 'b' | ... | 'y' | 'z'
⟨Number⟩ ::=  '00' | '01' | ... | '98' | '99'
  ⟨Op⟩   ::=  '*' | '+'
```

Six iterations to cover all the parsing code

Motivation Example

- Pure DSE

⟨Expr⟩ ::= ⟨ID⟩ | ⟨Number⟩ | ⟨Number⟩ ⟨Op⟩ ⟨Expr⟩
⟨ID⟩ ::= 'a' | 'b' | ... | 'y' | 'z'
⟨Number⟩ ::= '00' | '01' | ... | '98' | '99'
⟨Op⟩ ::= '*' | '+'

Input: "12+13"

```
1 public void entry(String a) throws ParseException{
2  // inputReader's type is Reader
3  inputReader = new StringReader(a);
4  parseExpr();
5  // application logic starts
6  if (a.charAt(a.length() - 1) == 'z') {
7  assert(false); //bug
8  }
9 }
```

Impossible to trigger the bug with 5-length strings

Grammar-Agnostic DSE

Motivation Example

- Grammar-agnostic DSE

First Stage

```
32 int getNextToken() throws ParseException {
33  int res = inputReader.read();
34  if (res == -1) return T_EOF;
35  char c = (char) res;
36  if (c >= '*' && c <= '+'){
37   return T_OP;
38  } else if (c >= 'a' && c <= 'z'){
39   return T_ID;
40  } else if (c >= '0' && c <= '9'){
41   char next_c = (char) inputReader.read();
42   if (next_c >= '0' && next_c <= '9'){
43    return T_NUM;
44   }
45  }
46  throw new ParseException();
47 }
```

$$\texttt{T\_OP} \quad t[0] \geq' \text{*}' \wedge t[0] \leq' +'$$

$$\texttt{T\_ID} \quad t[0] \geq' a' \wedge t[0] \leq' z'$$

$$\texttt{T\_NUM} \quad t[0] \geq' 0' \wedge t[0] \leq' 9' \wedge t[1] \geq' 0' \wedge t[1] \leq' 9'$$

Motivation Example

- Grammar-agnostic DSE

Second Stage

Input: "12+13"

First Path's Path Condition

$$\underbrace{\mathbf{T}[0] = \texttt{T\_NUM} \wedge \mathbf{T}[1] = \texttt{T\_OP} \wedge \mathbf{T}[2] = \texttt{T\_NUM}}_{\textbf{Token PC}} \wedge \underbrace{a[4] \neq \texttt{'z'}}_{\textbf{Application Logic PC}}$$

Motivation Example

- Grammar-agnostic DSE

Second Stage

Input: "12+13"

First Path's Path Condition

$$\underbrace{\mathbf{T}[0] = \texttt{T\_NUM} \wedge \mathbf{T}[1] = \texttt{T\_OP} \wedge \mathbf{T}[2] = \texttt{T\_NUM}}_{\textbf{Token PC}} \wedge \underbrace{a[4] = \texttt{'z'}}_{\textbf{Application Logic PC}}$$

Motivation Example

$$\underbrace{\mathbf{T}[0] = \texttt{T\_NUM} \wedge \mathbf{T}[1] = \texttt{T\_OP} \wedge \mathbf{T}[2] = \texttt{T\_NUM}}_{\text{Token PC}} \wedge \underbrace{a[4] = \texttt{'z'}}_{\text{Application Logic PC}}$$

Motivation Example

$$\underbrace{\mathbf{T}[0] = \texttt{T\_NUM} \wedge \mathbf{T}[1] = \texttt{T\_OP} \wedge \mathbf{T}[2] = \texttt{T\_NUM}}_{\text{Token PC}} \wedge \underbrace{a[4] = \texttt{'z'}}_{\text{Application Logic PC}}$$

$\langle \texttt{T\_NUM}, \texttt{T\_OP}, \texttt{T\_NUM} \rangle$

Motivation Example

$$\underbrace{\mathbf{T}[0] = \texttt{T\_NUM} \wedge \mathbf{T}[1] = \texttt{T\_OP} \wedge \mathbf{T}[2] = \texttt{T\_NUM}}_{\text{Token PC}} \wedge \underbrace{a[4] = \text{'z'}}_{\text{Application Logic PC}}$$

$\langle \texttt{T\_NUM}, \texttt{T\_OP}, \texttt{T\_NUM} \rangle$

$\texttt{T\_OP}$ $t[0] \geq' \text{'*'} \wedge t[0] \leq' \text{'+'}$

$\texttt{T\_ID}$ $t[0] \geq' a' \wedge t[0] \leq' z'$

$\texttt{T\_NUM}$ $t[0] \geq' 0' \wedge t[0] \leq' 9' \wedge$
$t[1] \geq' 0' \wedge t[1] \leq' 9'$

Motivation Example

$$\underbrace{\mathbf{T}[0] = \texttt{T\_NUM} \wedge \mathbf{T}[1] = \texttt{T\_OP} \wedge \mathbf{T}[2] = \texttt{T\_NUM}}_{\textbf{Token PC}} \wedge \underbrace{a[4] = \text{'z'}}_{\textbf{Application Logic PC}}$$

$\langle \texttt{T\_NUM}, \texttt{T\_OP}, \texttt{T\_NUM} \rangle$

$\texttt{T\_OP}$ $t[0] \geq \text{'*'} \wedge t[0] \leq \text{'+'}$

$\texttt{T\_ID}$ $t[0] \geq \text{'a'} \wedge t[0] \leq \text{'z'}$

$\texttt{T\_NUM}$ $t[0] \geq \text{'0'} \wedge t[0] \leq \text{'9'} \wedge$
$t[1] \geq \text{'0'} \wedge t[1] \leq \text{'9'}$

$$a[0] \geq \text{'0'} \wedge a[0] \leq \text{'9'} \wedge a[1] \geq \text{'0'} \wedge a[1] \leq \text{'9'} \wedge$$
$$a[2] \geq \text{'*'} \wedge a[2] \leq \text{'+'} \wedge$$
$$a[3] \geq \text{'0'} \wedge a[3] \leq \text{'9'} \wedge a[4] \geq \text{'0'} \wedge a[4] \leq \text{'9'}$$

Motivation Example

$$\underbrace{\mathbf{T}[0] = \texttt{T\_NUM} \wedge \mathbf{T}[1] = \texttt{T\_OP} \wedge \mathbf{T}[2] = \texttt{T\_NUM}}_{\textbf{Token PC}} \wedge \underbrace{a[4] = 'z'}_{\textbf{Application Logic PC}}$$

$\langle \texttt{T\_NUM}, \texttt{T\_OP}, \texttt{T\_NUM} \rangle$

$\texttt{T\_OP}$ $t[0] \geq' '*' \wedge t[0] \leq' '+'$

$\texttt{T\_ID}$ $t[0] \geq' 'a' \wedge t[0] \leq' 'z'$

$\texttt{T\_NUM}$ $t[0] \geq' '0' \wedge t[0] \leq' '9' \wedge$
$t[1] \geq' '0' \wedge t[1] \leq' '9'$

$$a[0] \geq \text{'0'} \wedge a[0] \leq \text{'9'} \wedge a[1] \geq \text{'0'} \wedge a[1] \leq \text{'9'} \wedge$$
$$a[2] \geq \text{'*'} \wedge a[2] \leq \text{'+'} \wedge$$
$$a[3] \geq \text{'0'} \wedge a[3] \leq \text{'9'} \wedge a[4] \geq \text{'0'} \wedge a[4] \leq \text{'9'}$$

Motivation Example

$$\underbrace{\mathbf{T}[0] = \texttt{T\_NUM} \wedge \mathbf{T}[1] = \texttt{T\_OP} \wedge \mathbf{T}[2] = \texttt{T\_NUM}}_{\text{Token PC}} \wedge \underbrace{a[4] = 'z'}_{\text{Application Logic PC}}$$

$\langle \texttt{T\_NUM}, \texttt{T\_OP}, \texttt{T\_NUM} \rangle$

$\texttt{T\_OP}$ $t[0] \geq' '*' \wedge t[0] \leq' '+'$

$\texttt{T\_ID}$ $t[0] \geq' 'a' \wedge t[0] \leq' 'z'$

$\texttt{T\_NUM}$ $t[0] \geq' '0' \wedge t[0] \leq' '9' \wedge t[1] \geq' '0' \wedge t[1] \leq' '9'$

$$a[0] \geq '0' \wedge a[0] \leq '9' \wedge a[1] \geq '0' \wedge a[1] \leq '9' \wedge a[2] \geq '*' \wedge a[2] \leq '+' \wedge a[3] \geq '0' \wedge a[3] \leq '9' \wedge a[4] \geq '0' \wedge a[4] \leq '9'$$

Unsatisfiable

Motivation Example

- Grammar-agnostic DSE

Second Stage

Input: "12+13"

First Path's Path Condition

Token PC

$$\mathbf{T}[0] = \texttt{T\_NUM} \wedge \mathbf{T}[1] = \texttt{T\_OP} \wedge \mathbf{T}[2] \neq \texttt{T\_NUM}$$

Motivation Example

- Grammar-agnostic DSE

Second Stage

Input: "12+13"

First Path's Path Condition

Token PC

$$\mathbf{T}[0] = \texttt{T\_NUM} \wedge \mathbf{T}[1] = \texttt{T\_OP} \wedge \mathbf{T}[2] \neq \texttt{T\_NUM}$$

$$\bigwedge_{i=1}^{3} \mathbf{T}[i] \in \{\texttt{T\_ID}, \texttt{T\_NUM}, \texttt{T\_OP}\}$$

Motivation Example

- Grammar-agnostic DSE

Second Stage

Input: "12+13"

First Path's Path Condition

Token PC

$$\mathbf{T}[0] = \texttt{T\_NUM} \wedge \mathbf{T}[1] = \texttt{T\_OP} \wedge \mathbf{T}[2] \neq \texttt{T\_NUM}$$

$$\bigwedge_{i=1}^{3} \mathbf{T}[i] \in \{\texttt{T\_ID}, \texttt{T\_NUM}, \texttt{T\_OP}\} \Rightarrow \langle \texttt{T\_NUM}, \texttt{T\_OP}, \texttt{T\_ID} \rangle$$

Motivation Example

$\langle \texttt{T\_NUM}, \texttt{T\_OP}, \texttt{T\_ID} \rangle$

$\texttt{T\_OP}$ $t[0] \geq' \text{'*'} \wedge t[0] \leq' \text{'+'}$

$\texttt{T\_ID}$ $t[0] \geq' a' \wedge t[0] \leq' z'$

$\texttt{T\_NUM}$ $t[0] \geq' 0' \wedge t[0] \leq' 9' \wedge$
$t[1] \geq' 0' \wedge t[1] \leq' 9'$

$$a[0] \geq \text{'0'} \wedge a[0] \leq \text{'9'} \wedge a[1] \geq \text{'0'} \wedge a[1] \leq \text{'9'} \wedge$$
$$a[2] \geq \text{'*'} \wedge a[2] \leq \text{'+'} \wedge$$
$$a[3] \geq \text{'}a\text{'} \wedge a[3] \leq \text{'}z\text{'}$$

Motivation Example

$\langle \texttt{T\_NUM}, \texttt{T\_OP}, \texttt{T\_ID} \rangle$

$a[0] \geq \text{'0'} \wedge a[0] \leq \text{'9'} \wedge a[1] \geq \text{'0'} \wedge a[1] \leq \text{'9'} \wedge$
$a[2] \geq \text{'*'} \wedge a[2] \leq \text{'+'} \wedge$
$a[3] \geq \text{'a'} \wedge a[3] \leq \text{'z'}$

T_OP $t[0] \geq' \text{'*'} \wedge t[0] \leq' \text{'+'}$

T_ID $t[0] \geq' a' \wedge t[0] \leq' z'$

T_NUM $t[0] \geq' 0' \wedge t[0] \leq' 9' \wedge$
$t[1] \geq' 0' \wedge t[1] \leq' 9'$

New Input: "11+a"

Motivation Example

- Grammar-agnostic DSE

Second Stage

Input: "11+a"

Second Path's Path Condition

$$\underbrace{\mathbf{T}[0] = \texttt{T\_NUM} \wedge \mathbf{T}[1] = \texttt{T\_OP} \wedge \mathbf{T}[2] \neq \texttt{T\_NUM} \wedge \mathbf{T}[2] = \texttt{T\_ID}}_{\textbf{Token PC}} \wedge \underbrace{a[3] \neq \prime\texttt{z}\prime}_{\textbf{Application Logic PC}}$$

Motivation Example

- Grammar-agnostic DSE

Second Stage

Input: "11+a"

Second Path's Path Condition

$$\underbrace{\mathbf{T}[0] = \texttt{T\_NUM} \wedge \mathbf{T}[1] = \texttt{T\_OP} \wedge \mathbf{T}[2] \neq \texttt{T\_NUM} \wedge \mathbf{T}[2] = \texttt{T\_ID} \wedge}_{\textbf{Token PC}}$$

$$\underbrace{a[3] = \text{'z'}}_{\textbf{Application Logic PC}}$$

Motivation Example

- Grammar-agnostic DSE

Second Stage

Input: "11+a"

Second Path's Path Condition

$$\underbrace{\mathbf{T}[0] = \texttt{T\_NUM} \wedge \mathbf{T}[1] = \texttt{T\_OP} \wedge \mathbf{T}[2] \neq \texttt{T\_NUM} \wedge \mathbf{T}[2] = \texttt{T\_ID} \wedge}_{\textbf{Token PC}}$$

$$\underbrace{a[3] = 'z'}_{\textbf{Application Logic PC}}$$

"11+z"

⟨Expr⟩ ::= ⟨ID⟩ | ⟨Number⟩ | ⟨Number⟩ ⟨Op⟩ ⟨Expr⟩
⟨ID⟩ ::= `'a' | 'b' | ... | 'y' | 'z'`
⟨Number⟩ ::= `'00' | '01' | ... | '98' | '99'`
⟨Op⟩ ::= `'*' | '+'`

Grammar-Agnostic DSE

Cover all the parsing code at the 2nd iteration. Trigger the bug at the 3rd iteration

Evaluation

- Implementation
  - Java Programs
  - JPF-based DSE engine

Evaluation

- Implementation
  - Java Programs
  - JPF-based DSE engine
- Research questions
  - Effectiveness
    - Bug detection and coverage
  - Efficiency

Evaluation

- Benchmark
  - 19 open-source Java parsing programs
  - 11 types of grammars
    - 5~128 token types

Evaluation

- Baseline
  - Byte-level symbolization
  - JQF: coverage-guided fuzzing [ISSTA'19]
  - Grammar: grammar-guided black-box fuzzing [ASE'19]
- Setup
  - 1 hour analysis time

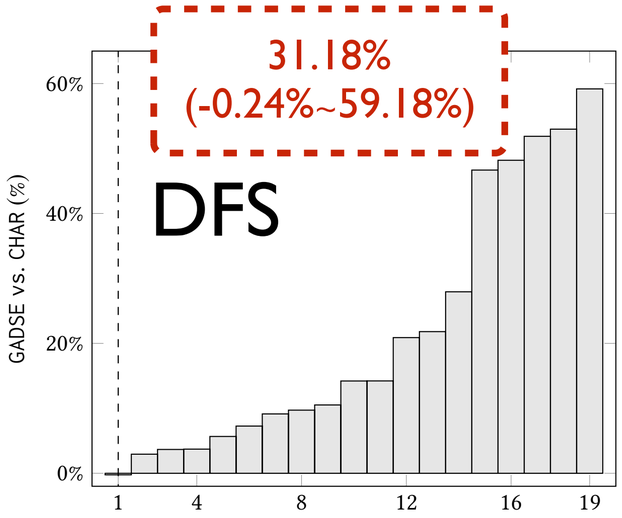
Results of Bug Finding

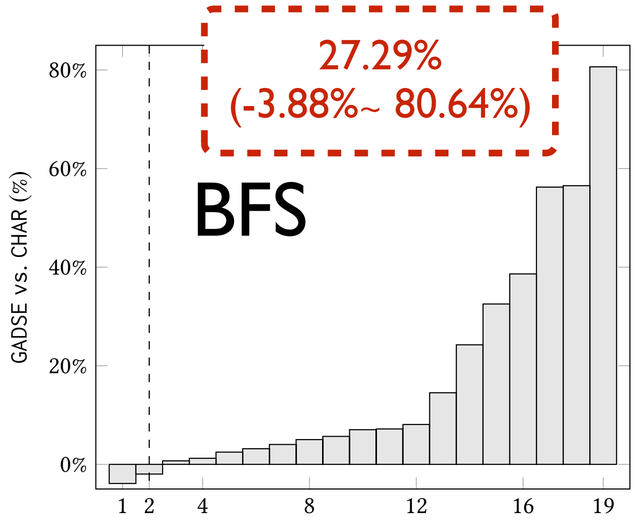
Name	Project	Type	GADSE	CHAR	JQF	GRAMMAR
Bug 1	J2latex	NumberFormatException	143s	>1h	185s	No
Bug 2	CMMParser	NullPointerException	36s	>1h	>1h	Yes
Bug 3	CMMParser	NumberFormatException	41s	>1h	>1h	Yes
Bug 4	Jsijcc	NullPointerException	456s	>1h	>1h	No
Bug 5	Jsijcc	ClassCastException	163s	77s	>1h	No
Bug 6	Jsijcc	ClassCastException	44s	>1h	>1h	Yes

Find 6 unknown bugs, and each needs less than 8 minutes

Results of Coverage

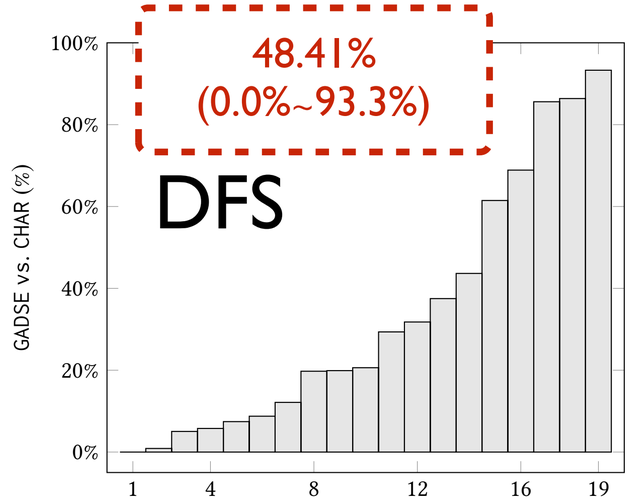
Statement Coverage

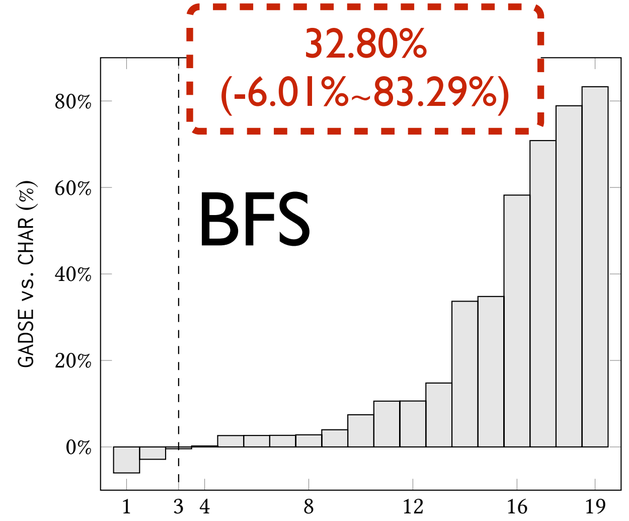




$$\frac{N_{\mathbf{GADSE}} - N_{\mathbf{CHAR}}}{N_{\mathbf{CHAR}}}$$

Branch Coverage

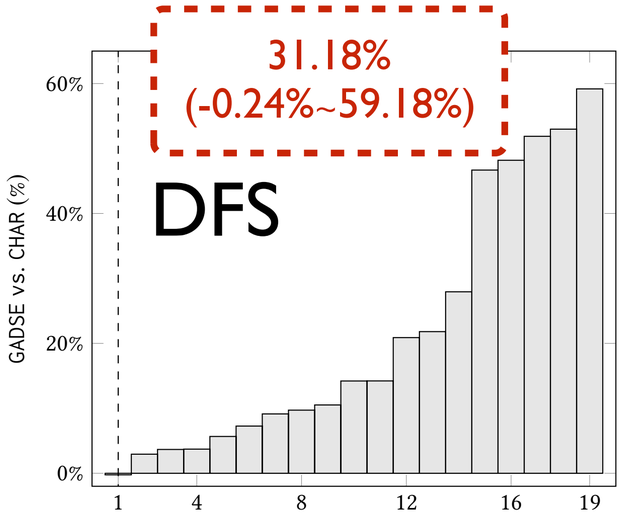


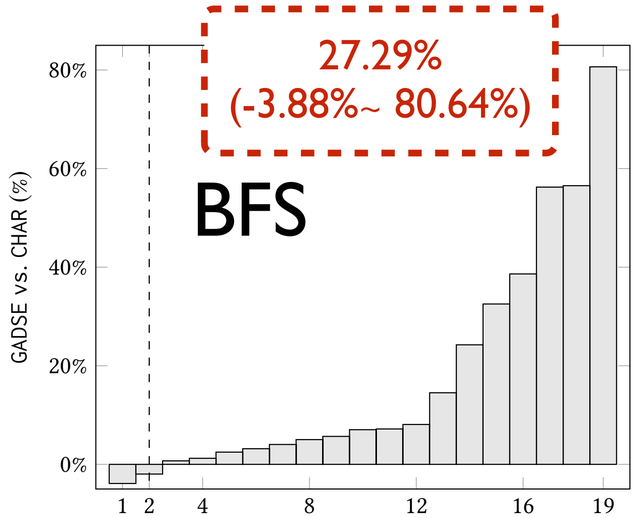


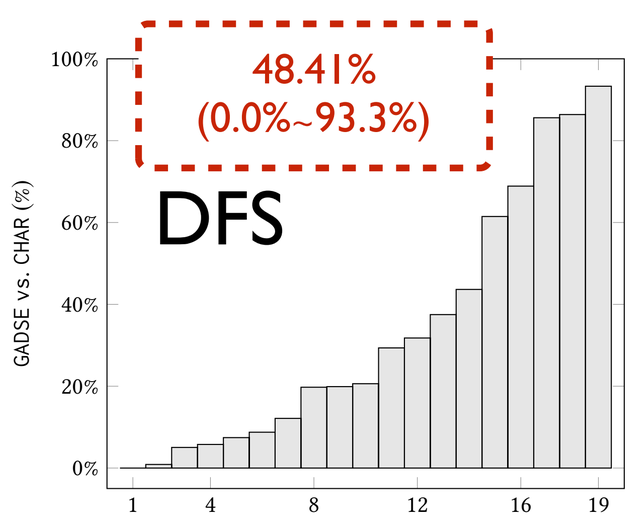
Results of Coverage

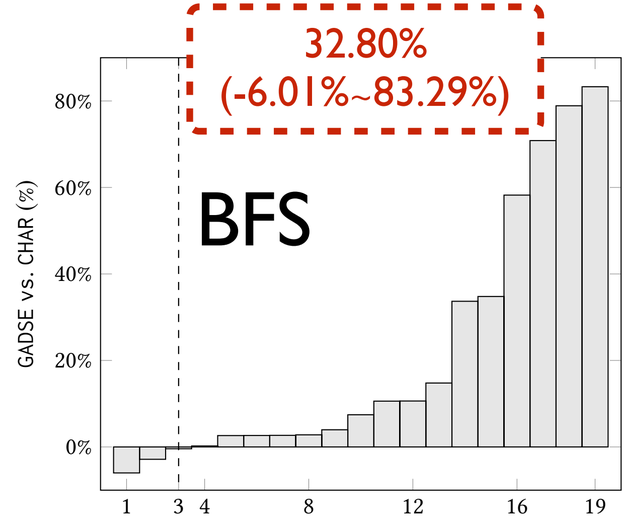
Statement Coverage

Branch Coverage





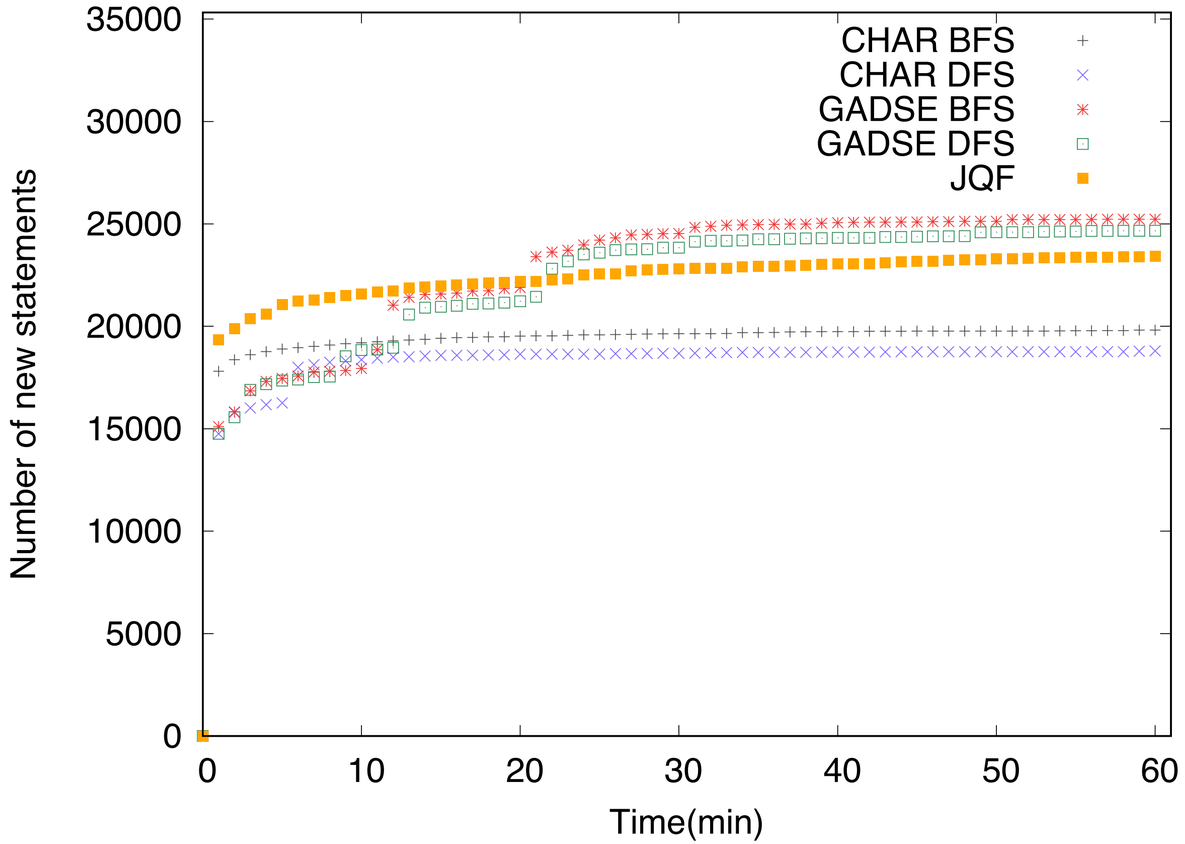




Better than byte-level symbolization in both statement and branch coverages

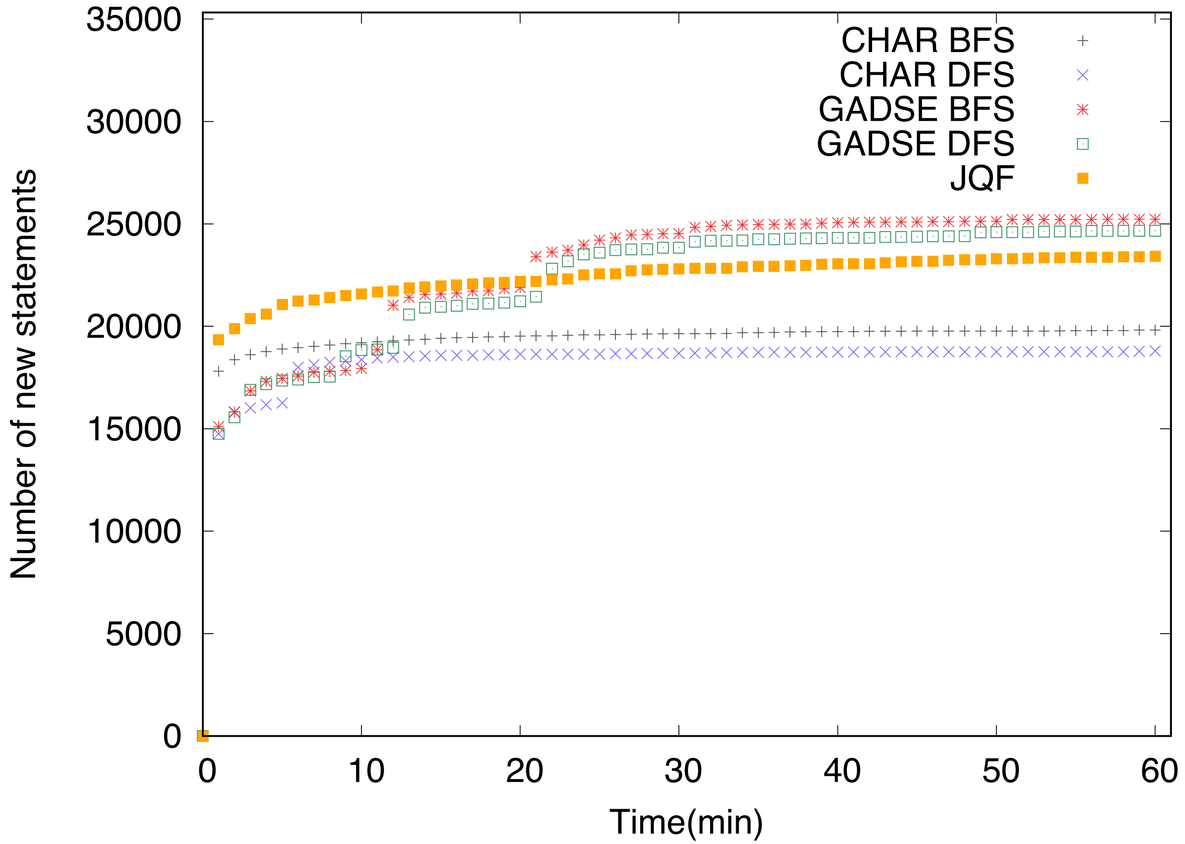
Results of Coverage

Statement Coverage



Results of Coverage

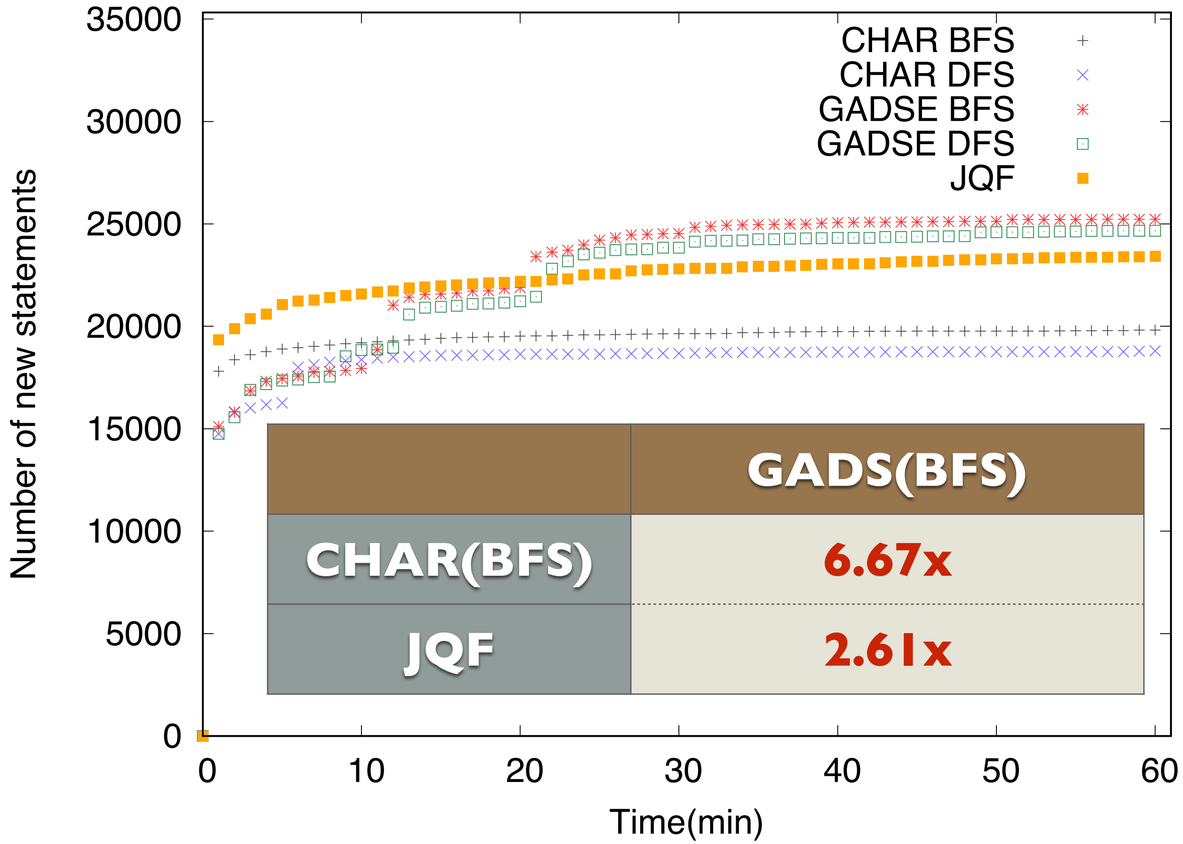
Statement Coverage



GADSE achieve best results in BFS mode

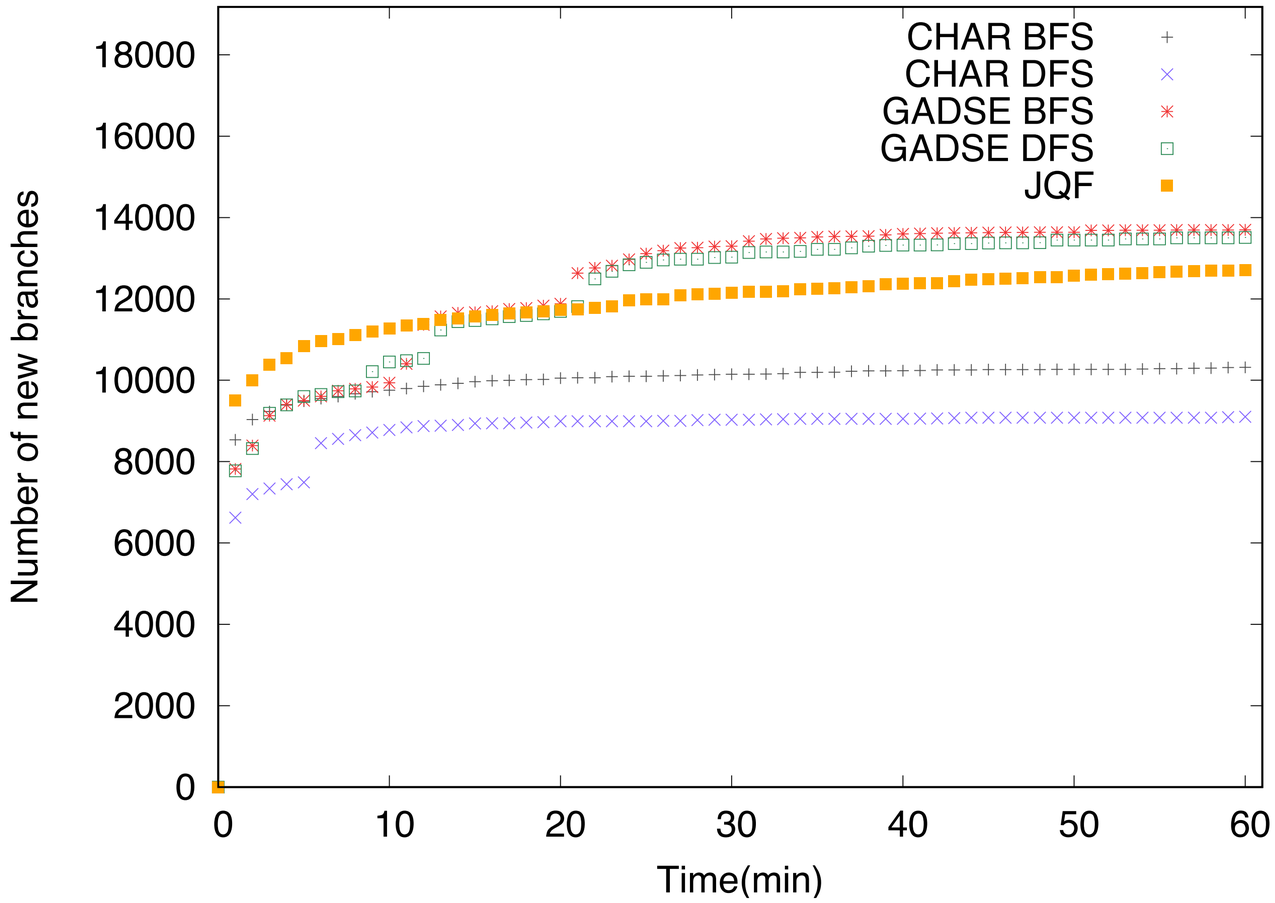
Results of Coverage

Statement Coverage



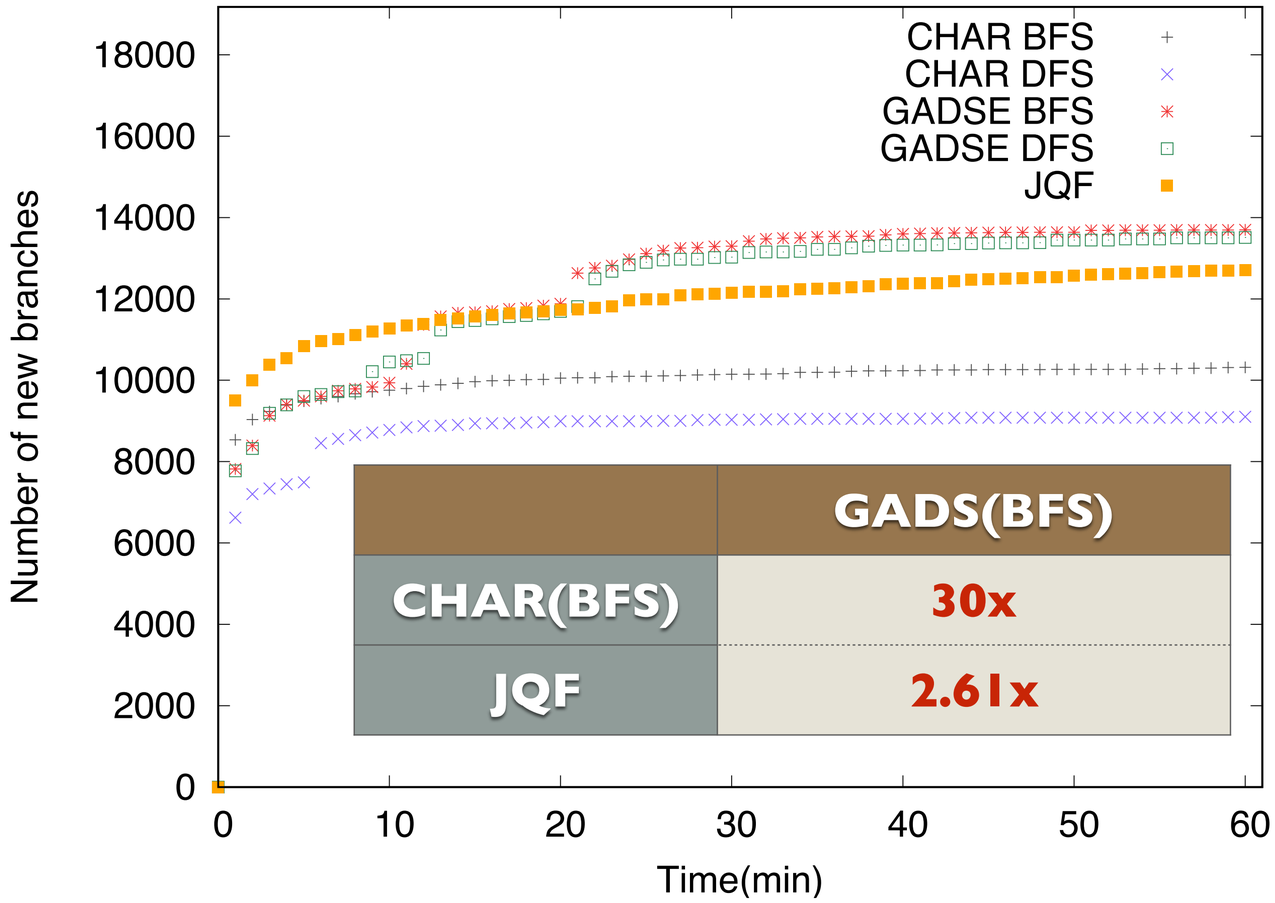
Results of Coverage

Branch Coverage

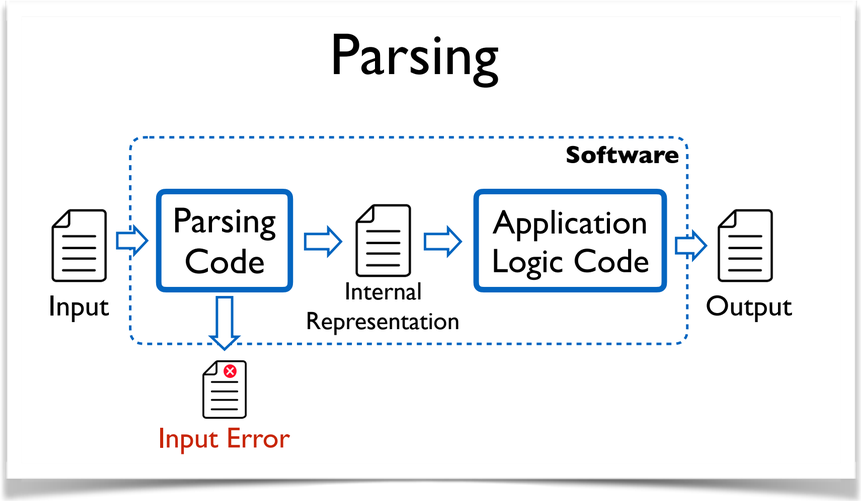


Results of Coverage

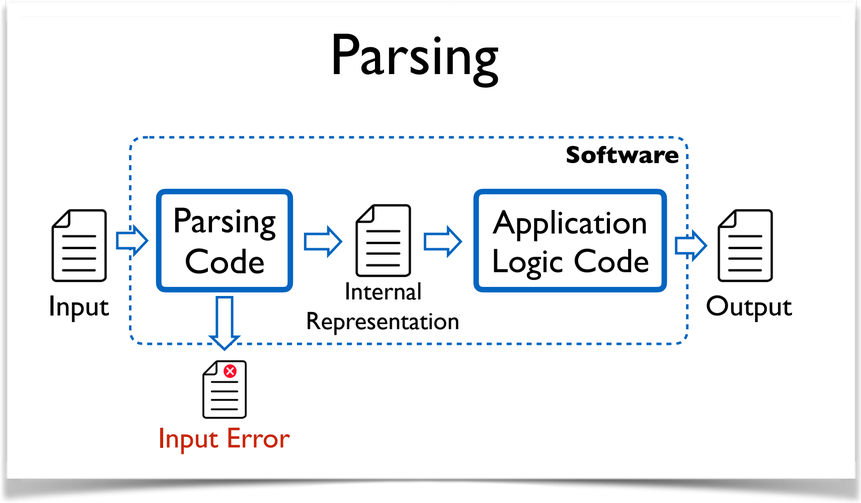
Branch Coverage

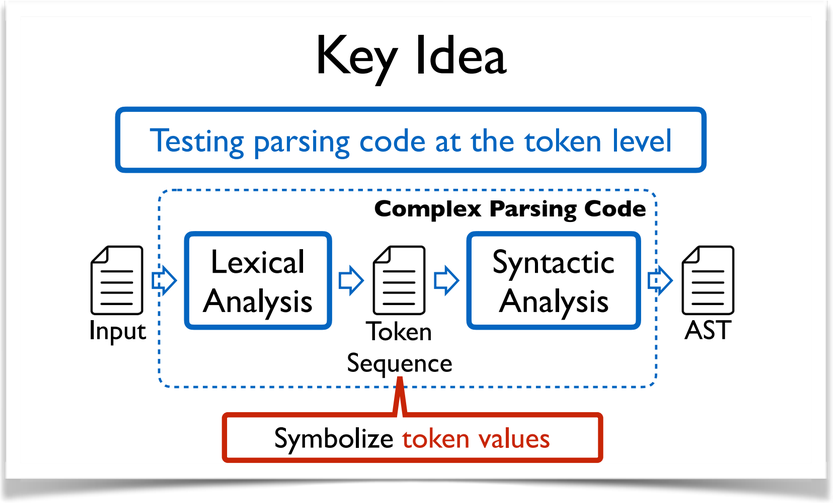


Conclusion


Parsing
Software
Parsing Code
Application Logic Code
Input
Internal Representation
Output
Input Error

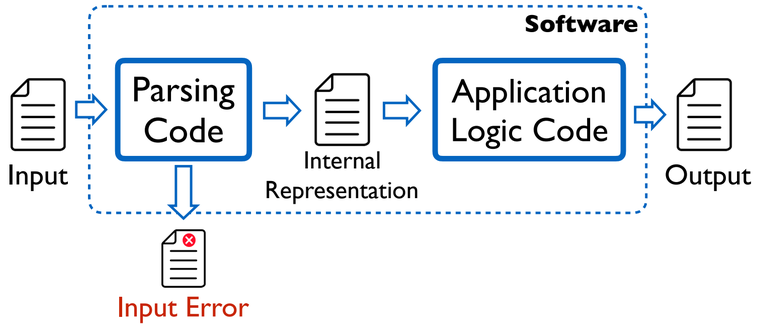
Conclusion


Parsing
Software
Parsing Code
Application Logic Code
Input
Internal Representation
Output
Input Error

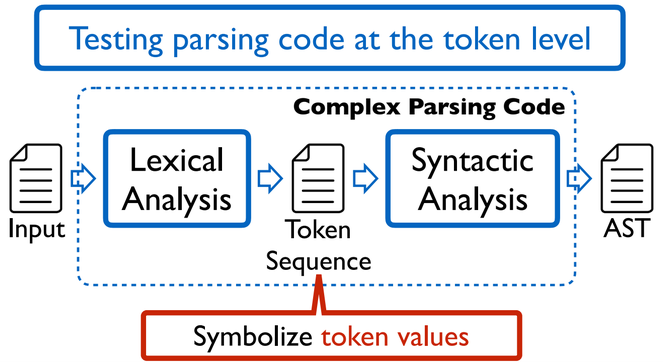

Key Idea
Testing parsing code at the token level
Complex Parsing Code
Lexical Analysis
Syntactic Analysis
Input
Token Sequence
AST
Symbolize token values

Conclusion

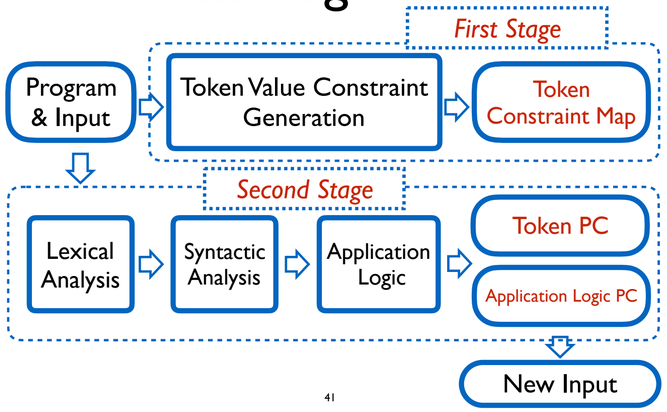
Parsing



Key Idea



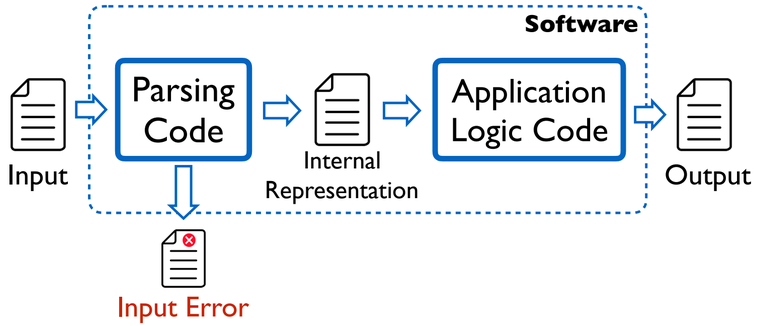
Grammar-agnostic DSE



41

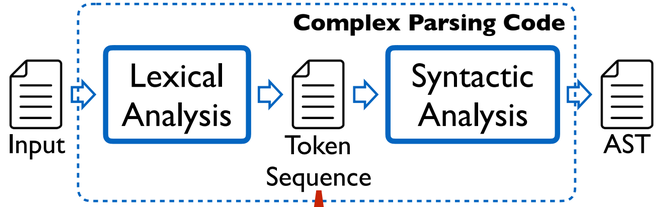
Conclusion

Parsing



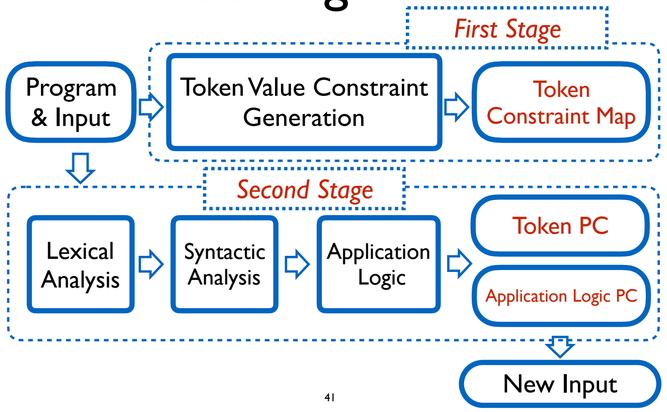
Key Idea

Testing parsing code at the token level



Symbolize token values

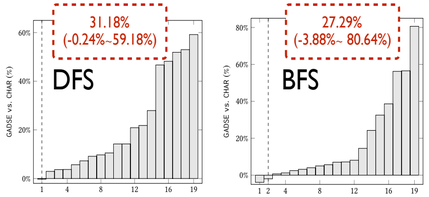
Grammar-agnostic DSE



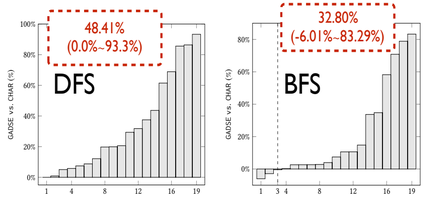
41

Results of Coverage

Statement Coverage



Branch Coverage



Better than byte-level symbolization in both statement and branch coverages

Thank you!

Q&A