

# **Automating string processing in spreadsheets using input-output examples**

Junliang Yan

# Spreadsheets

	A	B	C
1	Name	City	Phone
2	A	Nanjing	12345678910
3	B	Yangzhou	10987654321
4	C	Zhenjiang	11111111111
5	D	Suzhou	22222222222
6	E	Changzhou	33333333333
7	F	Wuxi	66666666666
8	G	Nantong	99999999999

# Collect cities of residence

Name	Adress	City	
A	Jiangsu, nanjing, nju		
B	Hubei, wuhan, hust		
C	Shandong, jinan, sdu		
D	Hunan, changsha, csu		

Copy-Paste one by one?

# Collect cities of residence

Name	Adress	City	
A	Jiangsu, nanjing, nju		
B	Hubei, wuhan, hust		
C	Shandong, jinan, sdu		
D	Hunan, changsha, csu		

✗ Copy-Paste one by one?

✓ Flashfill using input-output example!

# Flashfill using input-output examples

# Flashfill using input-output examples

- String Manipulation Language

```
SubStr2(  
    Input(2),  
    TokenSeq(AlphaToken),  
    2  
)
```

Program

# Flashfill using input-output examples

- String Manipulation Language
- **Synthesize** a program with **input-output examples**

1	Input(2)	Output
A	Jiangsu, nanjing, nju	nanjing

Input-output examples

Synthesize  
➔

```
SubStr2(  
    Input(2),  
    TokenSeq(AlphaToken),  
    2  
)
```

Program

# **String Manipulation Language**

**Construct Output Strings**



# Trace

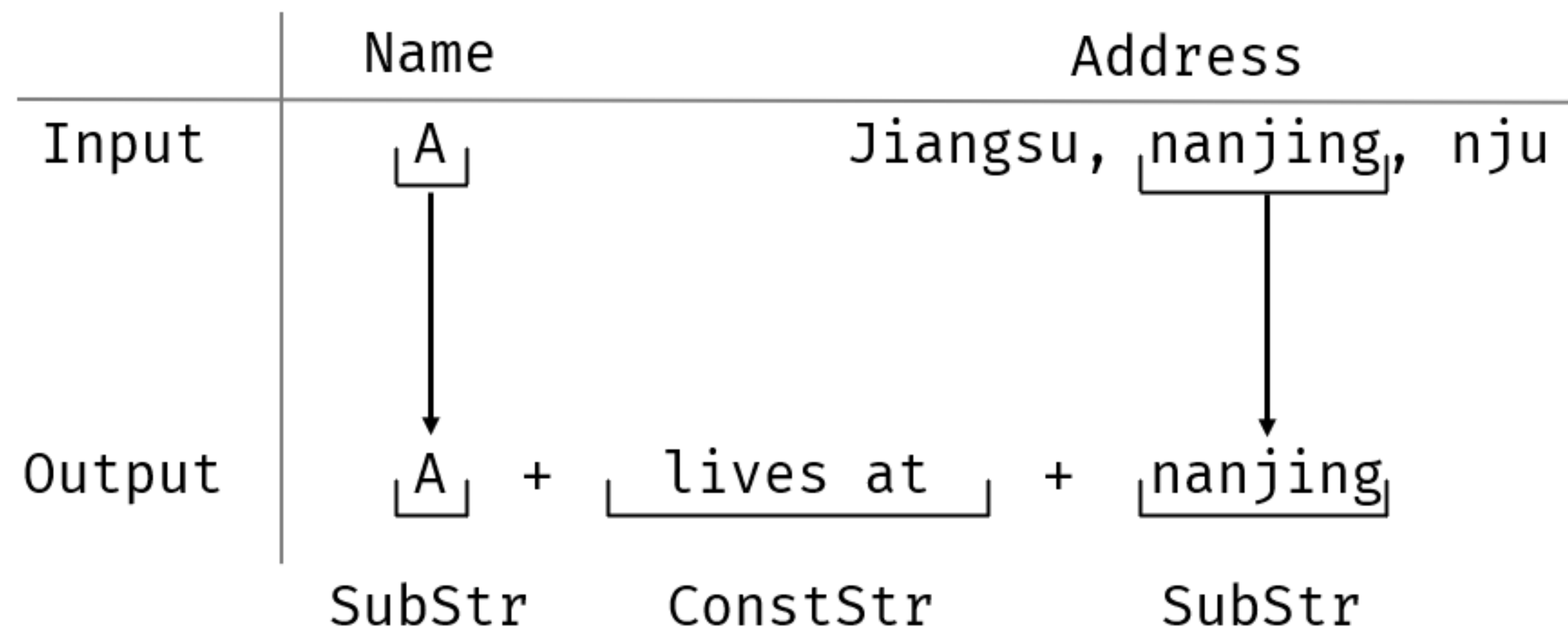
Trace Expr is a concatenation of atom expressions, substrings of inputs or a constant string.

Trace(ConstStr(...), SubStr(...), ...)

# Trace

Trace Expr is a concatenation of atom expressions, substrings of inputs or a constant string.

Trace(ConstStr(...), SubStr(...), ...)



Output: A lives at nanjing

# SubString

SubStr (            ,            )

# SubString

SubStr ( Input, , )

Input : **Index**, which input string is used.

# SubString

SubStr(Input, Left, Right)

Input : **Index**, which input string is used.

Left, Right: **Position Expressions**, the range of substring.

Name	Address	City
A	Jiangsu, nanjing , nju	nanjing

SubStr( , , )

# SubString

SubStr( Input, Left, Right )

Input : **Index**, which input string is used.

Left, Right: **Position Expressions**, the range of substring.

Name	Address	City
A	Jiangsu, nanjing , nju	nanjing

SubStr( **Address**, , )

# SubString

`SubStr(Input, Left, Right)`

`Input` : **Index**, which input string is used.

`Left, Right`: **Position Expressions**, the range of substring.

Name	Address	City
A	Jiangsu,  nanjing , nju	nanjing

`SubStr(Address, CPos(9), CPos(-6))`

# SubString

SubStr(Input, Left, Right)

Input : **Index**, which input string is used.

Left, Right: **Position Expressions**, the range of substring.

Name	Address	City
A	Jiangsu,  nanjing , nju	nanjing

✗ SubStr(Address, CPos(9), CPos(-6))

✓ SubStr(Address, Pos( $\epsilon$ , RE, 2), Pos(RE,  $\epsilon$ , 2))

where RE = LowercaseTokens



# Regular Expressions

Only use a **small subset** of regular expressions.

→ **A Sequence of Tokens.**

# Regular Expressions

Only use a **small subset** of regular expressions.

→ **A Sequence of Tokens.**

```
R = TokenSequence(LowercaseTokens, NumericTokens)
```

```
R = [a-z]+ [0-9]+
```

# Regular Expressions

Only use a **small subset** of regular expressions.

→ **A Sequence of Tokens.**

```
R = TokenSequence(LowercaseTokens, NumericTokens)
```

```
R = [a-z]+ [0-9]+
```

No kleen star (  $[a-z]^*$  ).

No disjunct operation (  $[a-z] \mid [0-9]$  ).

# Regular Expressions

Only use a **small subset** of regular expressions.

→ **A Sequence of Tokens.**

```
R = TokenSequence(LowercaseTokens, NumericTokens)
```

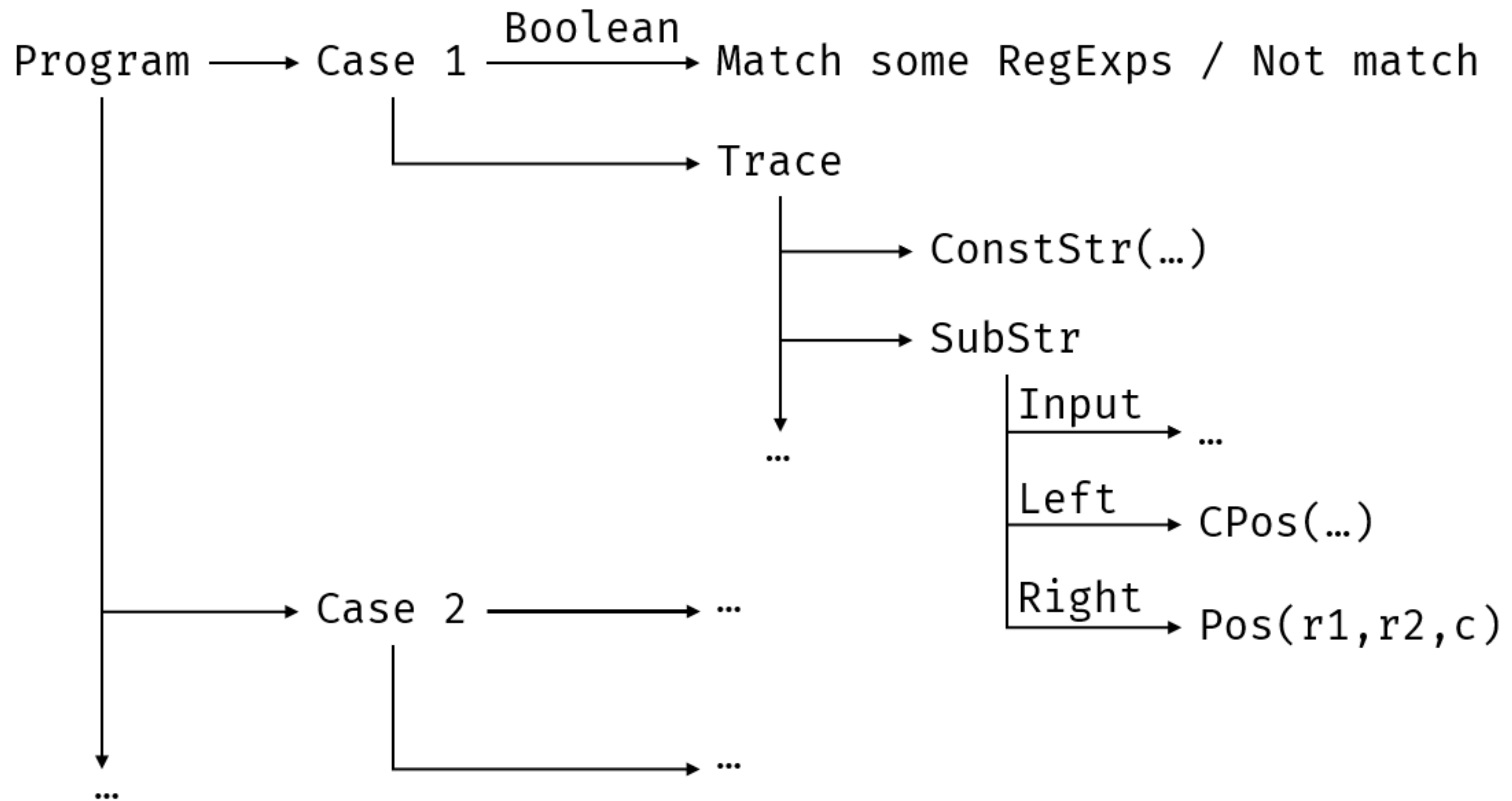
```
R = [a-z]+ [0-9]+
```

No kleen star (  $[a-z]^*$  ).


No disjunct operation (  $[a-z] \mid [0-9]$  ).

➔ **Efficient Algorithm**

# Conditionals




# Example

Num	Rev	Code	
123	321	case 123: return 321;	
456	654	case 456: return 654;	
147	741		
258	852		

```
Trace("case_", SubStr(Num, CPos(0), CPos(-1)),  
      ":", return ", SubStr(Rev, CPos(0), CPos(-1)),  
      ":",")
```

# Example

Item	Output 
Check-in: 2000 mora	2000 mora
New character: 180 fate	180 fate
Intertwined fate: 160 primogem	
New weapon: 240 fate	

```
Trace(SubStr(Item,  
Pos(TokenSeq(Colon, Space), TokenSeq(Numeric), -1),  
CPos(-1)))
```

# **Alogrithom**

**Synthesize a program with input-output examples**



# Goal

Given some input-output examples  $(i_1, o_1), \dots, (i_n, o_n)$ ,  
Synthesize a program  $P$  such that  $P(i_1) = o_1, \dots, P(i_n) = o_n$ .

1.

2.

3.

# Goal

Given some input-output examples  $(i_1, o_1), \dots, (i_n, o_n)$ ,  
Synthesize a program  $P$  such that  $P(i_1) = o_1, \dots, P(i_n) = o_n$ .

1. Synthesize  $n$  programs  $P_k$  such that

$$P_1(i_1) = o_1, P_2(i_2) = o_2, \dots, P_n(i_n) = o_n$$

- 2.

- 3.

# Goal

Given some input-output examples  $(i_1, o_1), \dots, (i_n, o_n)$ ,  
Synthesize a program  $P$  such that  $P(i_1) = o_1, \dots, P(i_n) = o_n$ .

1. Synthesize  $n$  programs  $P_k$  such that

$$P_1(i_1) = o_1, P_2(i_2) = o_2, \dots, P_n(i_n) = o_n$$

2. Intersect programs into non-empty partitions **greedily**

$$(P_1 \cap P_2, \{(i_1, o_1), (i_2, o_2)\}), (P_3 \cap \dots, \{(i_3, o_3), \dots\}), \dots$$

- 3.

# Goal

Given some input-output examples  $(i_1, o_1), \dots, (i_n, o_n)$ ,  
Synthesize a program  $P$  such that  $P(i_1) = o_1, \dots, P(i_n) = o_n$ .

1. Synthesize  $n$  programs  $P_k$  such that

$$P_1(i_1) = o_1, P_2(i_2) = o_2, \dots, P_n(i_n) = o_n$$

2. Intersect programs into non-empty partitions **greedily**

$$(P_1 \cap P_2, \{(i_1, o_1), (i_2, o_2)\}), (P_3 \cap \dots, \{(i_3, o_3), \dots\}), \dots$$

3. Construct boolean classification for partitions

$$(\text{Boolean Expression}, P_1 \cap P_2), \dots$$

# Generate Trace

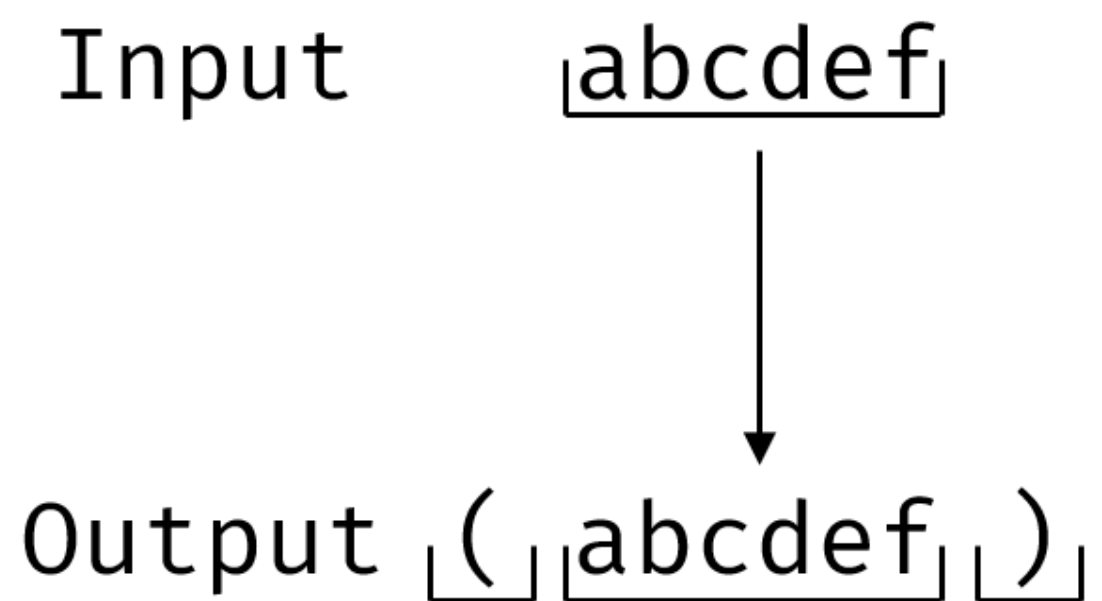
**Goal.** Synthesize  $n$  programs  $P_k$  such that

$$P_1(i_1) = o_1, P_2(i_2) = o_2, \dots, P_n(i_n) = o_n$$

# Generate Trace

**Goal.** Synthesize  $n$  programs  $P_k$  such that

$$P_1(i_1) = o_1, P_2(i_2) = o_2, \dots, P_n(i_n) = o_n$$



# Generate Trace

**Goal.** Synthesize  $n$  programs  $P_k$  such that

$$P_1(i_1) = o_1, P_2(i_2) = o_2, \dots, P_n(i_n) = o_n$$

Input abcdef



Output (abcdef)

Input a bcdef

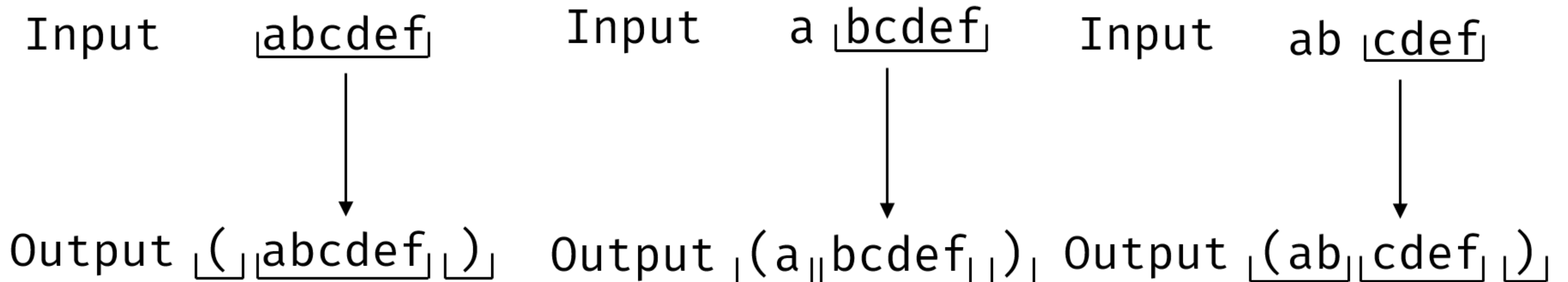


Output (a bcdef)

# Generate Trace

**Goal.** Synthesize  $n$  programs  $P_k$  such that

$$P_1(i_1) = o_1, P_2(i_2) = o_2, \dots, P_n(i_n) = o_n$$

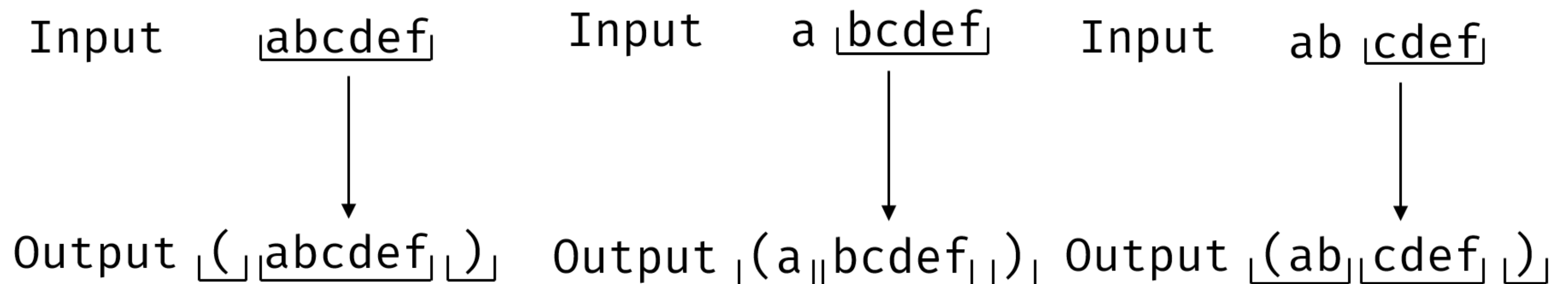




# Generate Trace

**Goal.** Synthesize  $n$  programs  $P_k$  such that

$$P_1(i_1) = o_1, P_2(i_2) = o_2, \dots, P_n(i_n) = o_n$$

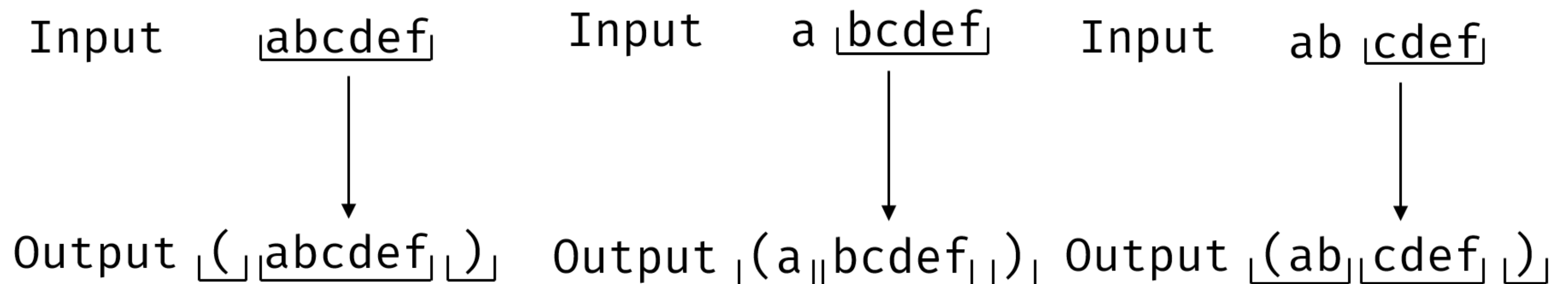


## Iterative **all** possible trace expressions

# Generate Trace

**Goal.** Synthesize  $n$  programs  $P_k$  such that

$$P_1(i_1) = o_1, P_2(i_2) = o_2, \dots, P_n(i_n) = o_n$$



## Iterative **all** possible trace expressions ?

The number is **exponential** in the length of output (  $2^{n-1}$  ).







# Generate Partitions

**Goal.** Intersect programs into non-empty partitions

$$(P_1 \cap P_2, \{(i_1, o_1), (i_2, o_2)\}), (P_3 \cap \dots, \{(i_3, o_3), \dots\}), \dots$$

# Boolean Classification

**Goal.** Construct boolean classification for partitions

$$((\text{Predicate} \wedge \dots) \vee \dots, P_1 \cap P_2), \dots$$

where

$$\text{Predicate} := \text{Match}(\text{Input}, \text{RegExp}, \text{Times}) \mid \neg \text{Match}(\dots)$$

**END**

**Q & A**