FlashFill++: Scaling Programming by Example by Cutting to the Chase

Paper by: Jose Cambronero et al.

Presented by: Tae Eun Kim

Background

Background

Program synthesis

Background

Program synthesis

- Automatically synthesizing programs

Background

Program synthesis

- Automatically synthesizing programs
- Mostly focused on Domain Specific Language (DSL)

Background

Program synthesis

- Automatically synthesizing programs
- Mostly focused on Domain Specific Language (DSL)

Programming-By-Example (PBE)

Background

Program synthesis

- Automatically synthesizing programs
- Mostly focused on Domain Specific Language (DSL)

Programming-By-Example (PBE)

- Popular method of program synthesis

Background

Program synthesis

- Automatically synthesizing programs
- Mostly focused on Domain Specific Language (DSL)

Programming-By-Example (PBE)

- Popular method of program synthesis
- Input-Output pairs are given as specification

Background

Program synthesis

- Automatically synthesizing programs
- Mostly focused on Domain Specific Language (DSL)

Programming-By-Example (PBE)

- Popular method of program synthesis
- Input-Output pairs are given as specification
- FlashFill[1]

Sumit Gulwani, Automating String Processing in Spreadsheets using Input-Output Examples, POPL, 2011

FlashFill

FlashFill

Legend of Program Synthesis

FlashFill

Legend of Program Synthesis

- Program synthesizer embedded in Microsoft Excel

FlashFill

Legend of Program Synthesis

- Program synthesizer embedded in Microsoft Excel
- Out in the wild for over 10 years

FlashFill

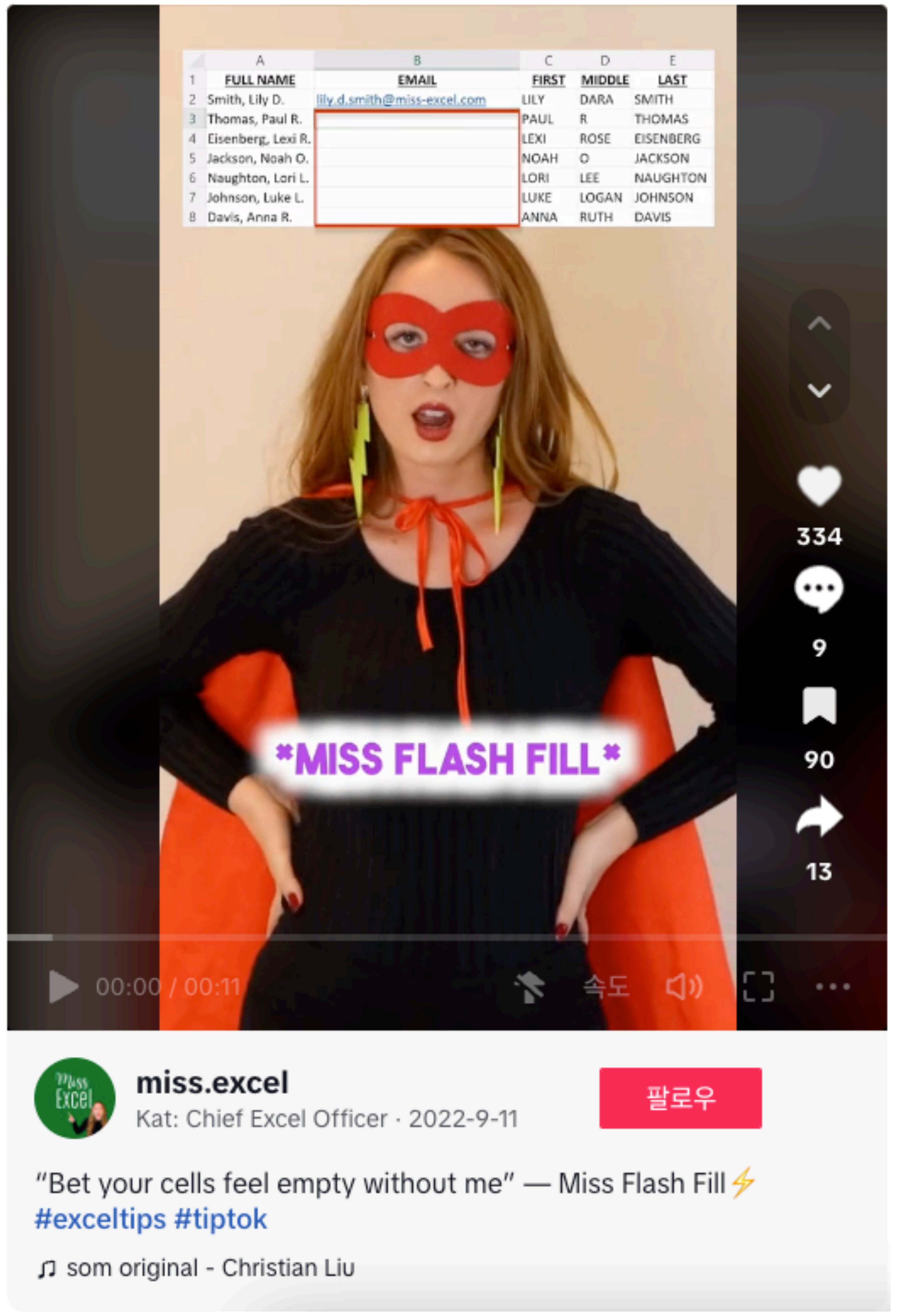
Legend of Program Synthesis

- Program synthesizer embedded in Microsoft Excel
- Out in the wild for over 10 years
- “The best feature in Excel 13”

FlashFill

Legend of Program Synthesis

- Program synthesizer embedded in Microsoft Excel
- Out in the wild for over 10 years
- “The best feature in Excel 13”



	A	B	C	D	E
1	FULL NAME	EMAIL	FIRST	MIDDLE	LAST
2	Smith, Lily D.	lily.d.smith@miss-excel.com	LILY	DARA	SMITH
3	Thomas, Paul R.		PAUL	R	THOMAS
4	Eisenberg, Lexi R.		LEXI	ROSE	EISENBERG
5	Jackson, Noah O.		NOAH	O	JACKSON
6	Naughton, Lori L.		LORI	LEE	NAUGHTON
7	Johnson, Luke L.		LUKE	LOGAN	JOHNSON
8	Davis, Anna R.		ANNA	RUTH	DAVIS

miss.excel
Kat: Chief Excel Officer · 2022-9-11

“Bet your cells feel empty without me” — Miss Flash Fill⚡
#exceltips #tiptok

♫ som original - Christian Liu

FlashFill

Legend of Program Synthesis

- Program synthesizer embedded in Microsoft Excel
- Out in the wild for over 10 years
- “The best feature in Excel 13”

FlashFill

Legend of Program Synthesis

- Program synthesizer embedded in Microsoft Excel
- Out in the wild for over 10 years
- “The best feature in Excel 13”

Power of FlashFill

FlashFill

Legend of Program Synthesis

- Program synthesizer embedded in Microsoft Excel
- Out in the wild for over 10 years
- “The best feature in Excel 13”

Power of FlashFill

- It fills up your empty cell

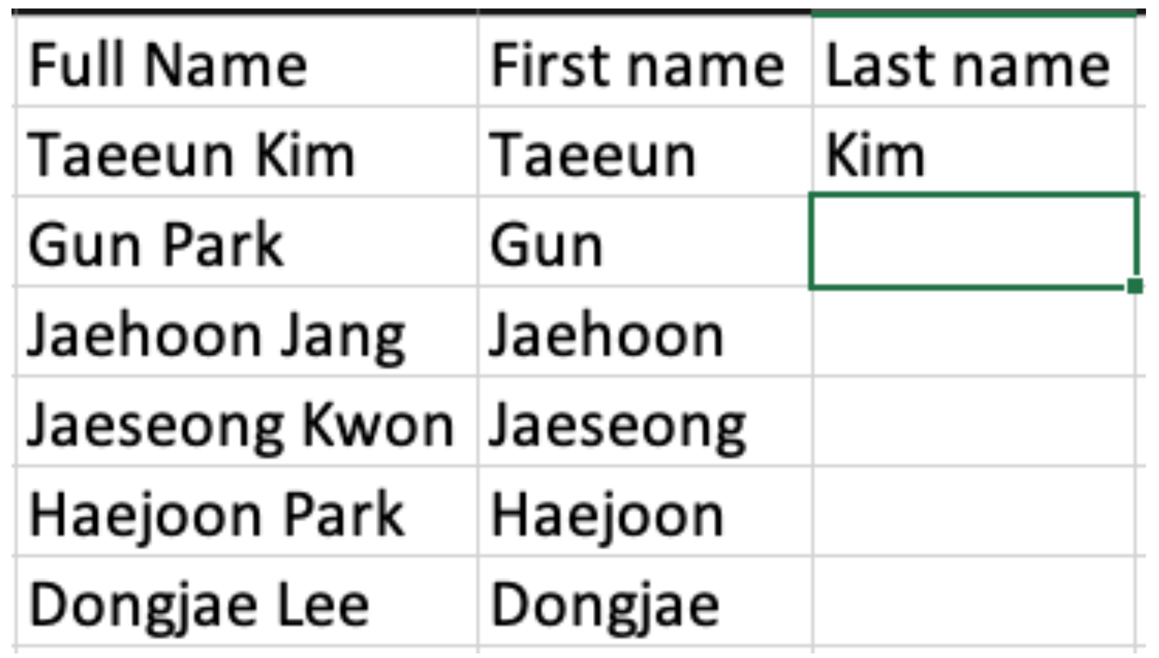
FlashFill

Legend of Program Synthesis

- Program synthesizer embedded in Microsoft Excel
- Out in the wild for over 10 years
- “The best feature in Excel 13”

Power of FlashFill

- It fills up your empty cell



Full Name	First name	Last name
Taeeun Kim	Taeeun	Kim
Gun Park	Gun	
Jaehoon Jang	Jaehoon	
Jaeseong Kwon	Jaeseong	
Haejoon Park	Haejoon	
Dongjae Lee	Dongjae	

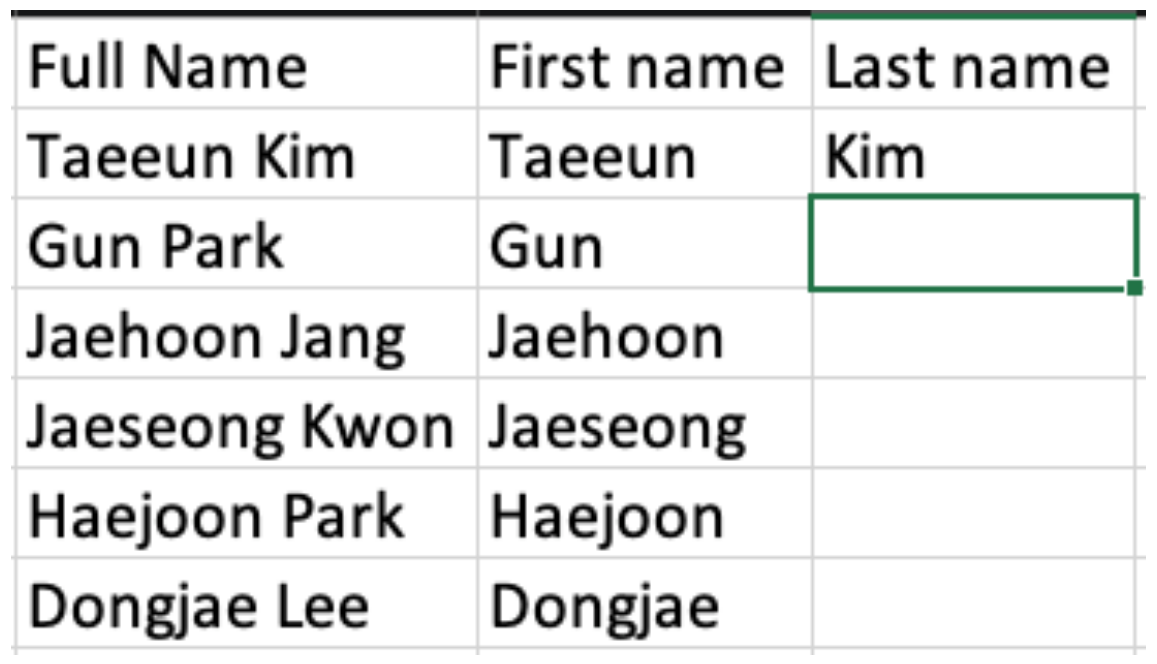
FlashFill

Legend of Program Synthesis

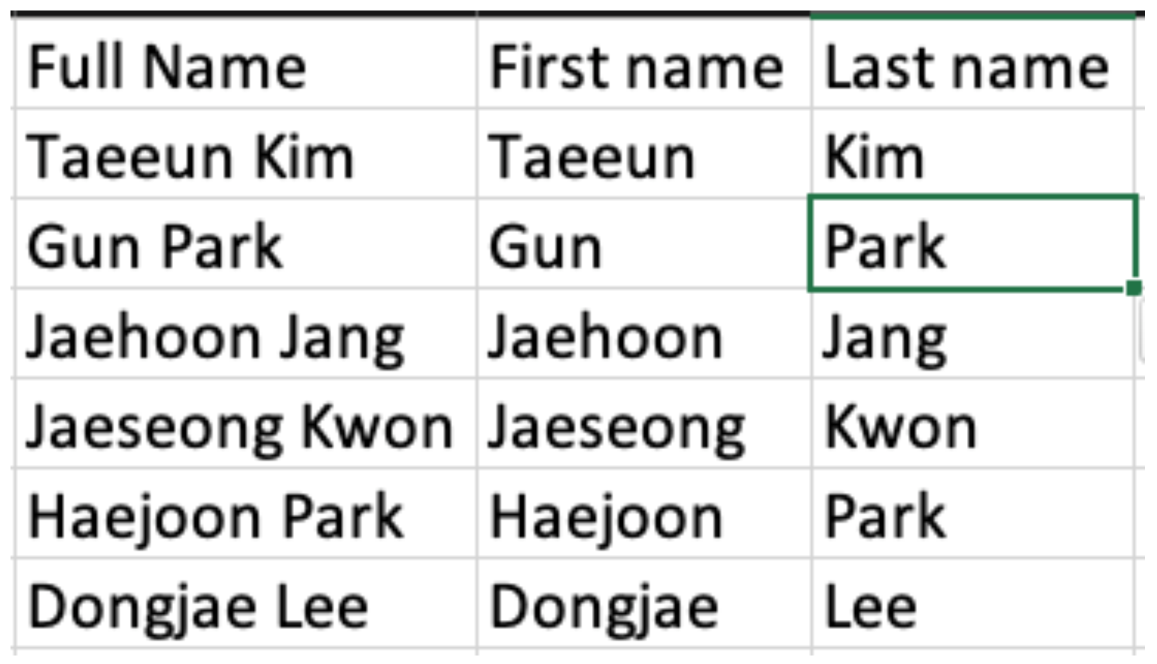
- Program synthesizer embedded in Microsoft Excel
- Out in the wild for over 10 years
- "The best feature in Excel 13"

Power of FlashFill

- It fills up your empty cell



Full Name	First name	Last name
Taeeun Kim	Taeeun	Kim
Gun Park	Gun	
Jaehoon Jang	Jaehoon	
Jaeseong Kwon	Jaeseong	
Haejoon Park	Haejoon	
Dongjae Lee	Dongjae	



Full Name	First name	Last name
Taeeun Kim	Taeeun	Kim
Gun Park	Gun	Park
Jaehoon Jang	Jaehoon	Jang
Jaeseong Kwon	Jaeseong	Kwon
Haejoon Park	Haejoon	Park
Dongjae Lee	Dongjae	Lee

Remaining Challenges

Remaining Challenges

1. **Unreadable code**

Remaining Challenges

1. Unreadable code

- FlashFill only provides solutions, but not the formula

Remaining Challenges

1. Unreadable code

- FlashFill only provides solutions, but not the formula
    - Ex. (“David Walker”, “623179”) -> “D-6231#walker”

Remaining Challenges

1. Unreadable code

- FlashFill only provides solutions, but not the formula
  - Ex. (“David Walker”, “623179”) -> “D-6231#walker”

```
Concatenate(
  Mid(Left(input1, Match(input1, "\p{Lu}+").StartMatch
                   + Len(Match(input1, "\p{Lu}+").FullMatch) - 1),
      Match(input1, "\p{Lu}+").StartMatch),
Concatenate("-",
Concatenate(Mid(Left(input2,Len(input2)-2), Match(input2,"[0-9]+").StartMatch),
Concatenate("#",
  Lower(Mid(
    Left(input1,
      First(LastN(MatchAll(input1, "[\p{Lu}\p{Ll}]+"), 1)).StartMatch
      + Len(First(LastN(MatchAll(input1, "[\p{Lu}\p{Ll}]+"), 1)).FullMatch)-1),
    Last(MatchAll(input1, "[\p{Lu}\p{Ll}]+")).StartMatch)))))
```

Remaining Challenges

1. Unreadable code

- FlashFill only provides solutions, but not the formula
    - Ex. (“David Walker”, “623179”) -> “D-6231#walker”

Remaining Challenges

1. Unreadable code

- FlashFill only provides solutions, but not the formula
    - Ex. (“David Walker”, “623179”) -> “D-6231#walker”
- Difficult to trust the result (thousands of data)

Remaining Challenges

1. Unreadable code

- FlashFill only provides solutions, but not the formula
  - Ex. (“David Walker”, “623179”) -> “D-6231#walker”
- Difficult to trust the result (thousands of data)

2. Ignorance to semantics of the data

Remaining Challenges

1. Unreadable code

- FlashFill only provides solutions, but not the formula
  - Ex. (“David Walker”, “623179”) -> “D-6231#walker”
- Difficult to trust the result (thousands of data)

2. Ignorance to semantics of the data

- Date-Time, Numeric transformation

Remaining Challenges

1. Unreadable code

- FlashFill only provides solutions, but not the formula
  - Ex. (“David Walker”, “623179”) -> “D-6231#walker”
- Difficult to trust the result (thousands of data)

2. Ignorance to semantics of the data

- Date-Time, Numeric transformation
  - Ex. Name of the months

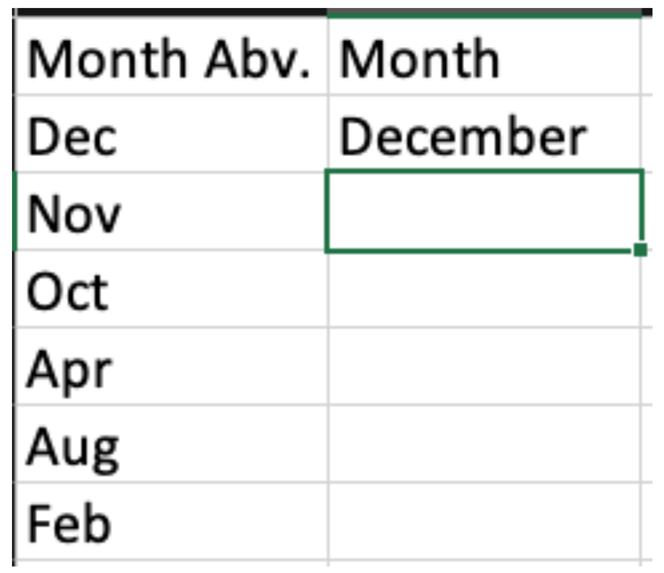
Remaining Challenges

1. Unreadable code

- FlashFill only provides solutions, but not the formula
  - Ex. (“David Walker”, “623179”) -> “D-6231#walker”
- Difficult to trust the result (thousands of data)

2. Ignorance to semantics of the data

- Date-Time, Numeric transformation
  - Ex. Name of the months



Month Abv.	Month
Dec	December
Nov	
Oct	
Apr	
Aug	
Feb	

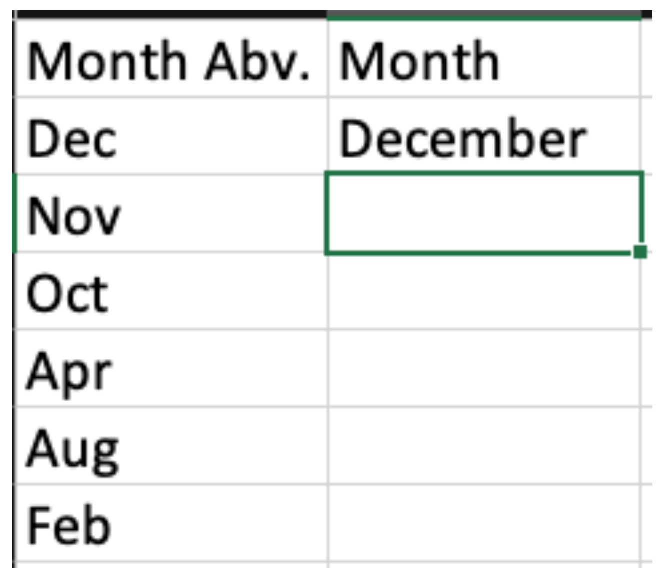
Remaining Challenges

1. Unreadable code

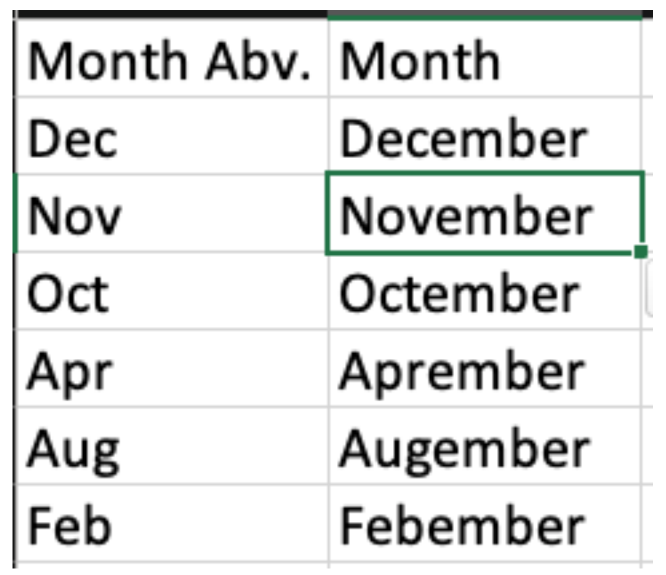
- FlashFill only provides solutions, but not the formula
  - Ex. (“David Walker”, “623179”) -> “D-6231#walker”
- Difficult to trust the result (thousands of data)

2. Ignorance to semantics of the data

- Date-Time, Numeric transformation
  - Ex. Name of the months



Month Abv.	Month
Dec	December
Nov	
Oct	
Apr	
Aug	
Feb	



Month Abv.	Month
Dec	December
Nov	November
Oct	Octember
Apr	Aprember
Aug	Augember
Feb	Febember

Fundamental Reason

Fundamental Reason

Expressiveness of DSL

Fundamental Reason

Expressiveness of DSL

- FlashFill supports 3 operators for strings and date-time

Fundamental Reason

Expressiveness of DSL

- FlashFill supports 3 operators for strings and date-time
- Duet[2] supports 5 operators.

Fundamental Reason

Expressiveness of DSL

- FlashFill supports 3 operators for strings and date-time
- Duet[2] supports 5 operators.
- Easy to synthesize, Difficult to comprehend

Fundamental Reason

Expressiveness of DSL

- FlashFill supports 3 operators for strings and date-time
- Duet[2] supports 5 operators.
- Easy to synthesize, Difficult to comprehend

Need for efficient search

Woosuk Lee, Combining the Top-Down Propagation and Bottom-Up Enumeration for Inductive Program Synthesis POPL, 2021

Fundamental Reason

Expressiveness of DSL

- FlashFill supports 3 operators for strings and date-time
- Duet[2] supports 5 operators.
- Easy to synthesize, Difficult to comprehend

Need for efficient search

- Bigger DSL means bigger search space

Woosuk Lee, Combining the Top-Down Propagation and Bottom-Up Enumeration for Inductive Program Synthesis POPL, 2021

Fundamental Reason

Expressiveness of DSL

- FlashFill supports 3 operators for strings and date-time
- Duet^2 supports 5 operators.
- Easy to synthesize, Difficult to comprehend

Need for efficient search

- Bigger DSL means bigger search space
- Needs efficient search strategy

Woosuk Lee, Combining the Top-Down Propagation and Bottom-Up Enumeration for Inductive Program Synthesis POPL, 2021

Solution: FlashFill++

Solution: FlashFill++

Expressive DSL

Solution: FlashFill++

Expressive DSL

- 40 operators in total

Solution: FlashFill++

Expressive DSL

- 40 operators in total
- 25 for strings and date-time

Solution: FlashFill++

Expressive DSL

- 40 operators in total
- 25 for strings and date-time

```
Left(input1, 1) & "-" & Left(input2, 4) & "#"
  & Lower(Last(FirstN(Split(input1, " "), 2)).Result)
```

Solution: FlashFill++

Expressive DSL

- 40 operators in total
- 25 for strings and date-time

```
Left(input1, 1) & "-" & Left(input2, 4) & "#"
  & Lower(Last(FirstN(Split(input1, " "), 2)).Result)
```

Efficient search

Solution: FlashFill++

Expressive DSL

- 40 operators in total
- 25 for strings and date-time

```
Left(input1, 1) & "-" & Left(input2, 4) & "#"
  & Lower(Last(FirstN(Split(input1, " "), 2)).Result)
```

Efficient search

- Cuts and Precedence

Solution: FlashFill++

Expressive DSL

- 40 operators in total
- 25 for strings and date-time

```
Left(input1, 1) & "-" & Left(input2, 4) & "#"
  & Lower(Last(FirstN(Split(input1, " "), 2)).Result)
```

Efficient search

- Cuts and Precedence
- Solved 92% of benchmarks while baseline performed 65%

Solution: FlashFill++

Expressive DSL

- 40 operators in total
- 25 for strings and date-time

```
Left(input1, 1) & "-" & Left(input2, 4) & "#"
  & Lower(Last(FirstN(Split(input1, " "), 2)).Result)
```

Efficient search

- Cuts and Precedence
- Solved 92% of benchmarks while baseline performed 65%
- 3X faster than the baselines

VSA-driven synthesis

VSA-driven synthesis

Version Space

- Set of programs that satisfy the given I/O examples

VSA-driven synthesis

Version Space

- Set of programs that satisfy the given I/O examples

Version Space Algebra (VSA)

- Operations to compose and manipulate the version space

VSA-driven synthesis

Version Space

- Set of programs that satisfy the given I/O examples

Version Space Algebra (VSA)

- Operations to compose and manipulate the version space

VSA-driven Synthesis

- Program synthesis by expanding and exploring the version space with VSA

VSA-driven synthesis

Version Space

- Set of programs that satisfy the given I/O examples

Version Space Algebra (VSA)

- Operations to compose and manipulate the version space

VSA-driven Synthesis

- Program synthesis by expanding and exploring the version space with VSA
- Top-Down vs Bottom Up

Top-Down Search

Top-Down Search

- Start from Output

Top-Down Search

- Start from Output
- Apply inverse of operators

Top-Down Search

- Start from Output
- Apply inverse of operators
- Expand until solution is found

Top-Down Search

- Start from Output
- Apply inverse of operators
- Expand until solution is found
- Ex. “TaeEun Kim” -> “Kim, T”

Top-Down Search

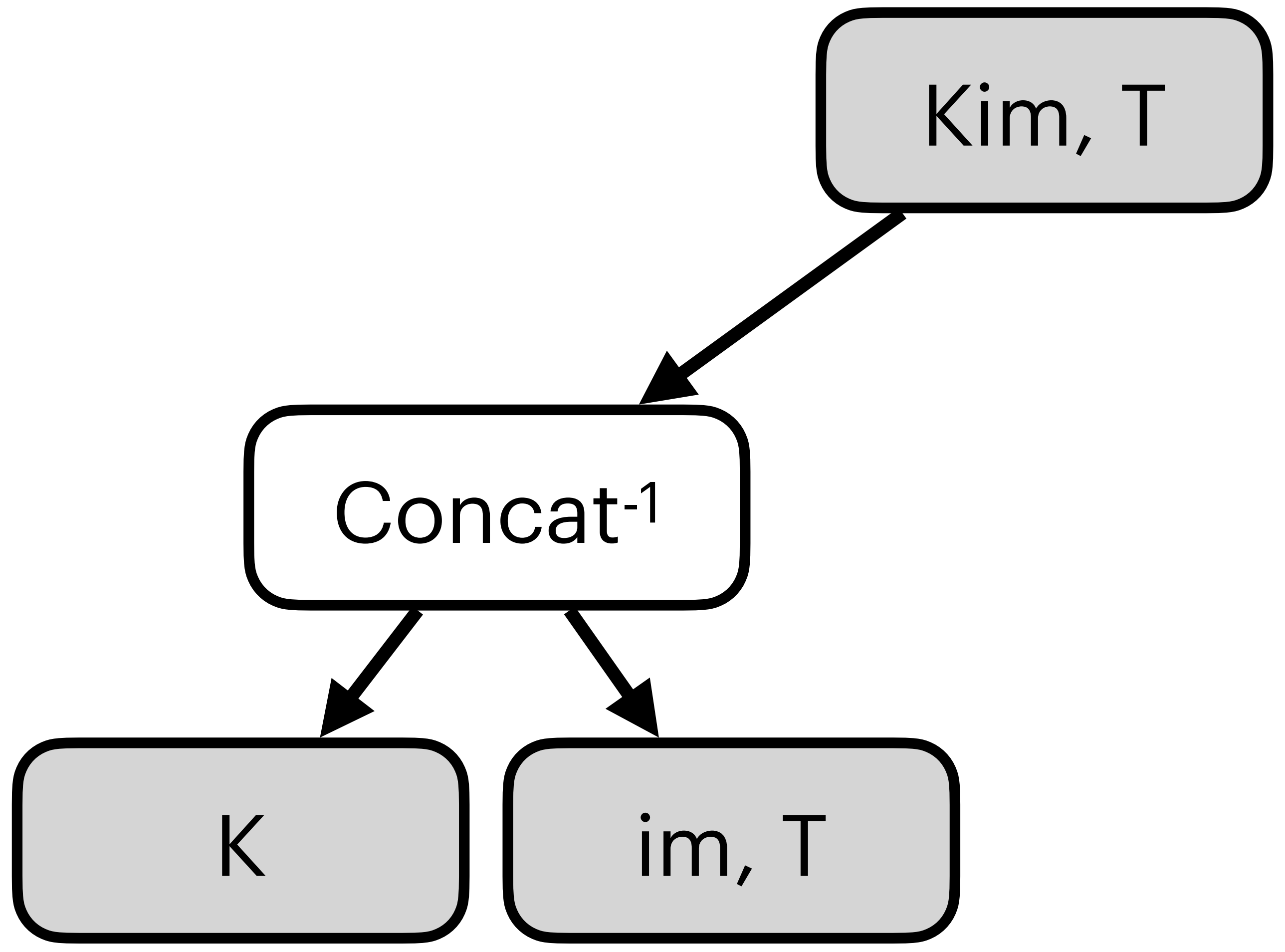
- Start from Output
- Apply inverse of operators
- Expand until solution is found
- Ex. “TaeEun Kim” -> “Kim, T”
  - Substr, Concat is supported

Top-Down Search

- Start from Output
- Apply inverse of operators
- Expand until solution is found
- Ex. “TaeEun Kim” -> “Kim, T”
  - Substr, Concat is supported

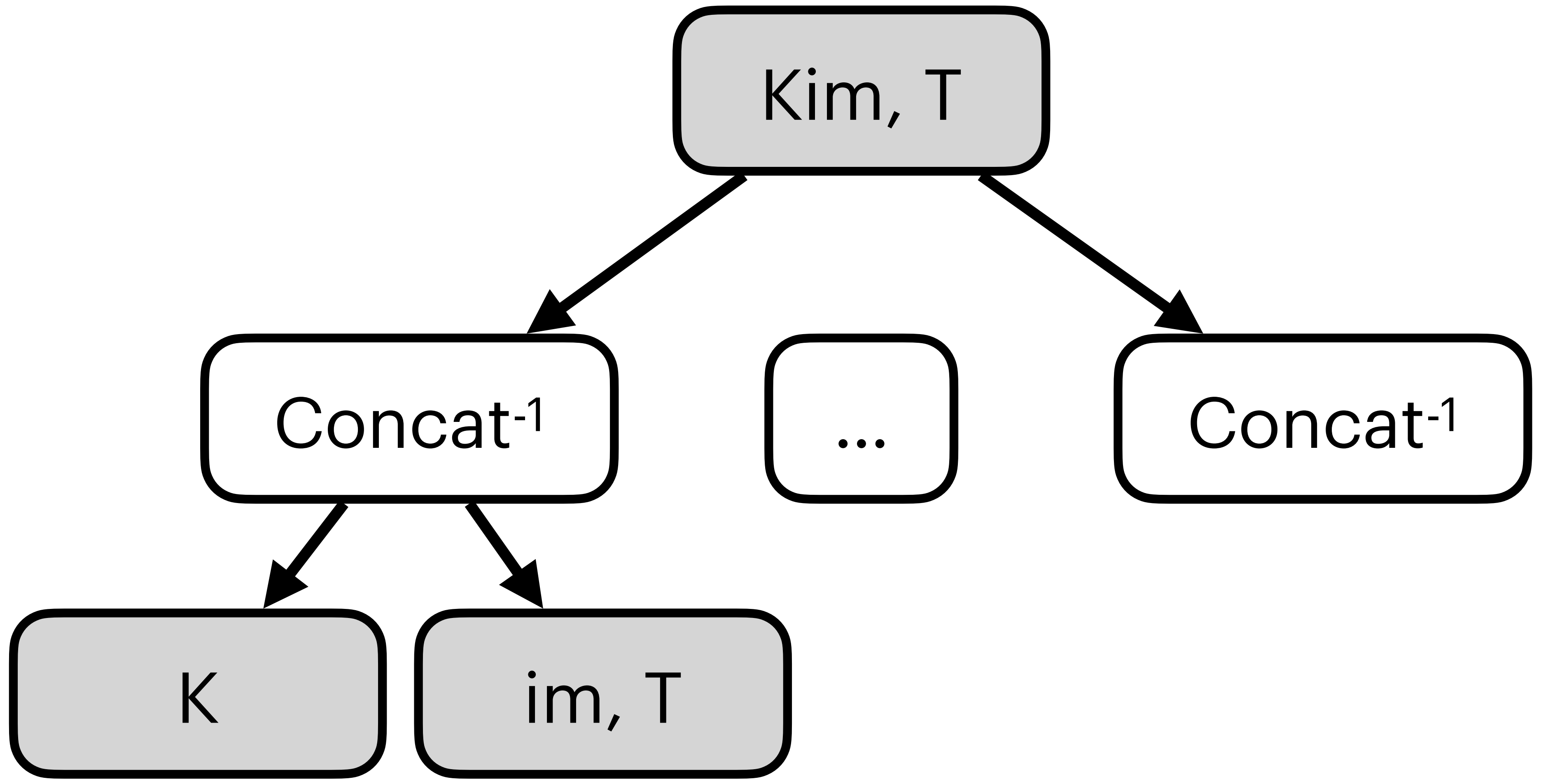
Top-Down Search

- Start from Output
- Apply inverse of operators
- Expand until solution is found
- Ex. “TaeEun Kim” -> “Kim, T”
    - Substr, Concat is supported



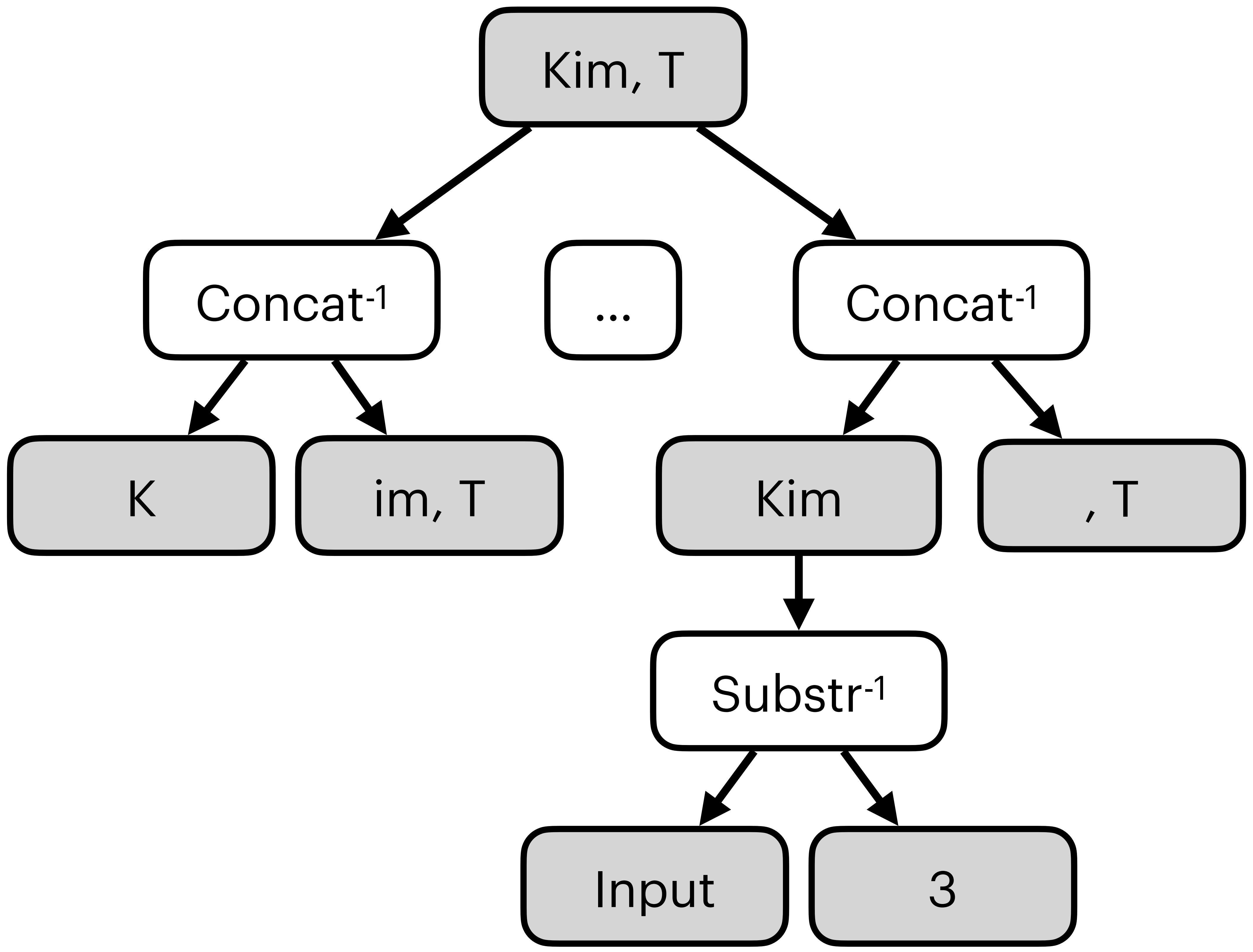
Top-Down Search

- Start from Output
- Apply inverse of operators
- Expand until solution is found
- Ex. “TaeEun Kim” -> “Kim, T”
    - Substr, Concat is supported



Top-Down Search

- Start from Output
- Apply inverse of operators
- Expand until solution is found
- Ex. “TaeEun Kim” -> “Kim, T”
    - Substr, Concat is supported



Bottom-Up Search

Bottom-Up Search

- Start from Input

Bottom-Up Search

- Start from Input
- Apply possible operators

Bottom-Up Search

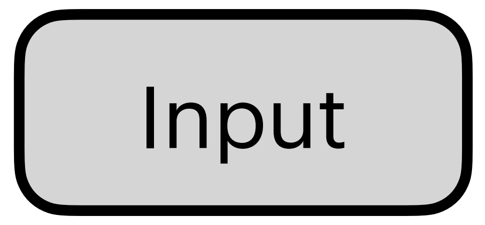
- Start from Input
- Apply possible operators
- Expand until solution is found

Bottom-Up Search

- Start from Input
- Apply possible operators
- Expand until solution is found
- Ex. “TaeEun Kim” -> “Kim, T”
  - Subset, Concat is supported

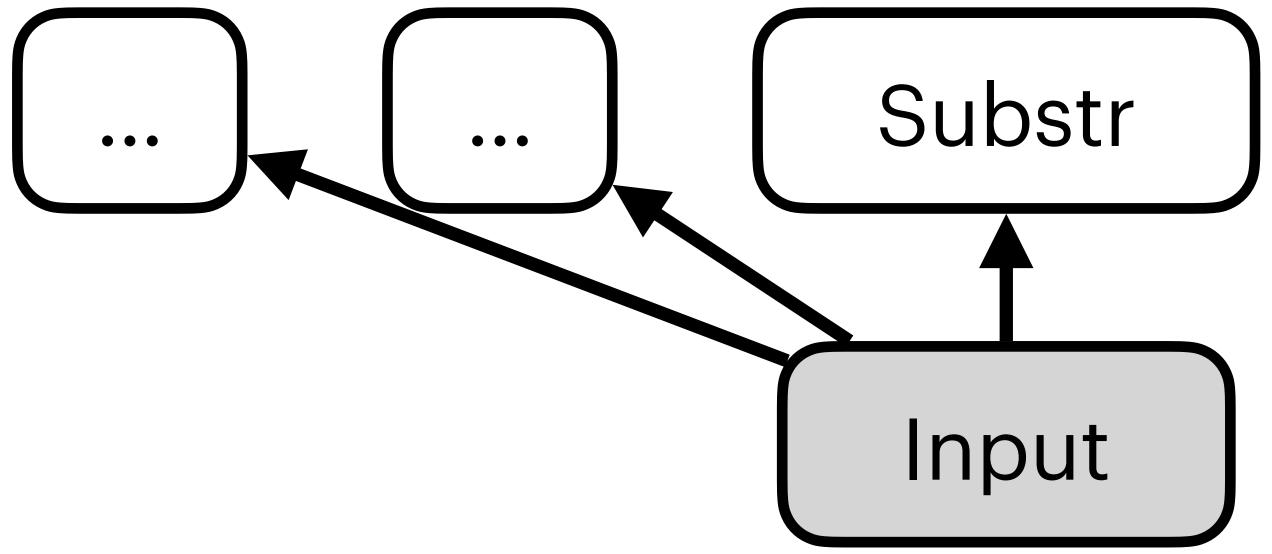
Bottom-Up Search

- Start from Input
- Apply possible operators
- Expand until solution is found
- Ex. “TaeEun Kim” -> “Kim, T”
  - Subset, Concat is supported



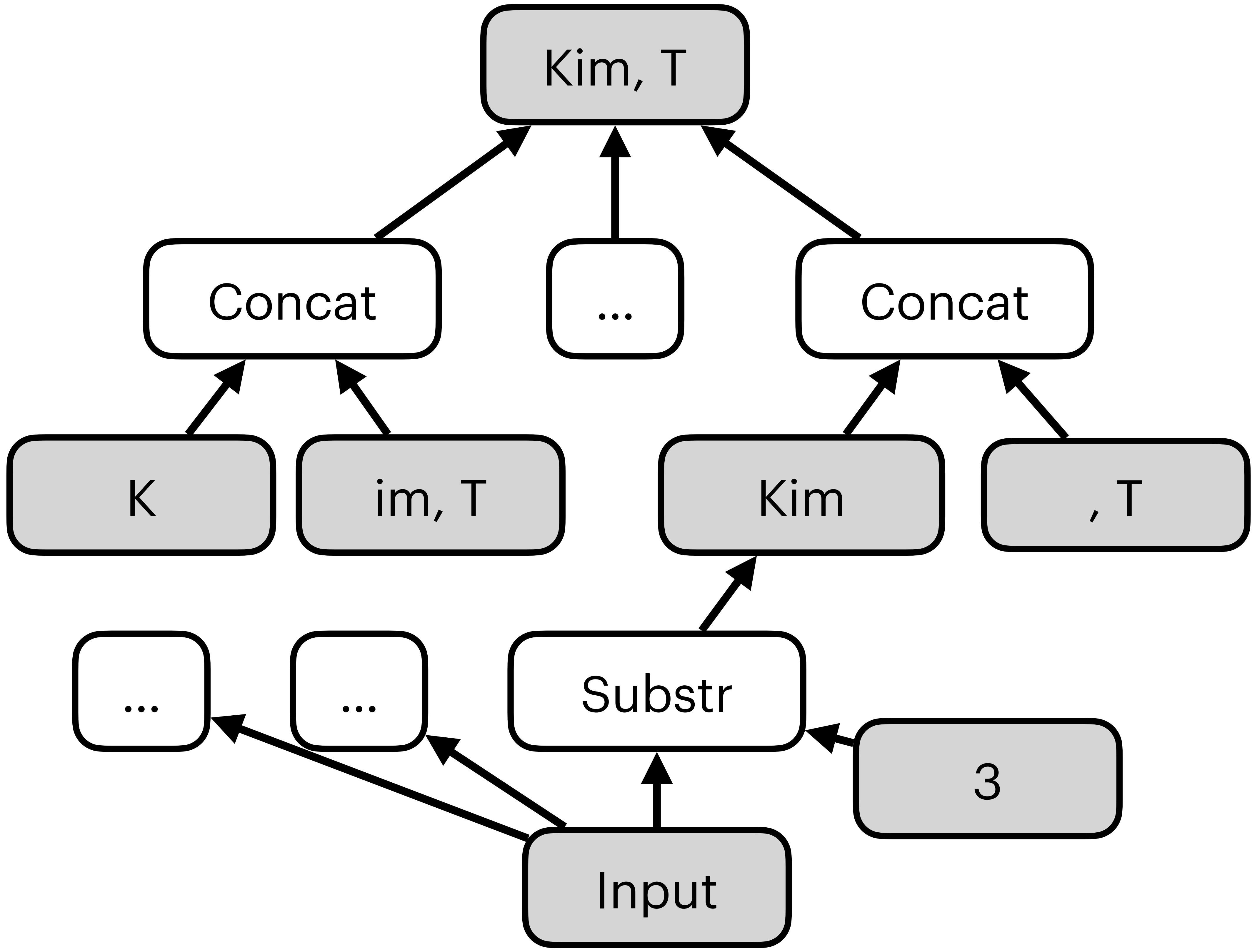
Bottom-Up Search

- Start from Input
- Apply possible operators
- Expand until solution is found
- Ex. “TaeEun Kim” -> “Kim, T”
  - Subset, Concat is supported



Bottom-Up Search

- Start from Input
- Apply possible operators
- Expand until solution is found
- Ex. “TaeEun Kim” -> “Kim, T”
  - Subset, Concat is supported



Limitations of Top-Down and Bottom-Up

Limitations of Top-Down and Bottom-Up

Example

Limitations of Top-Down and Bottom-Up

Example

- Input: “The price is $24.58 and 46 units are available.”

Limitations of Top-Down and Bottom-Up

Example

- Input: “The price is $24.58 and 46 units are available.”
- Output: 24.00

Limitations of Top-Down and Bottom-Up

Example

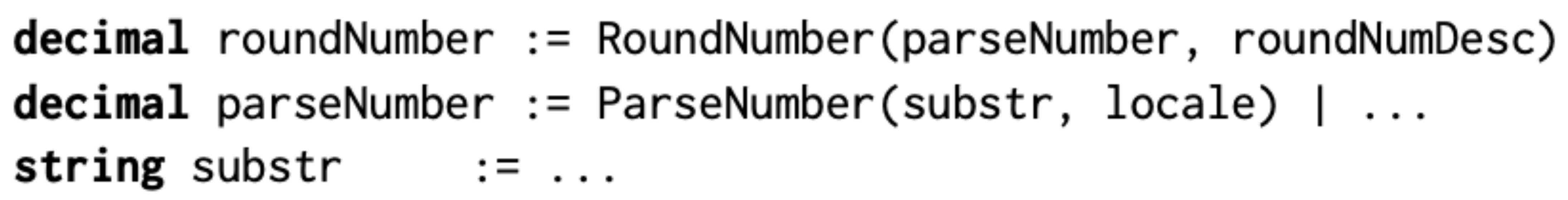
- Input: “The price is $24.58 and 46 units are available.”
- Output: 24.00
- Operations: roundNumber, parseNumber, subset

```
decimal roundNumber := RoundNumber(parseNumber, roundNumDesc)
decimal parseNumber := ParseNumber(substr, locale) | ...
string substr     := ...
```

Limitations of Top-Down and Bottom-Up

Example

- Input: “The price is $24.58 and 46 units are available.”
- Output: 24.00
- Operations: roundNumber, parseNumber, subset
- Start from non-terminal roundNumber.

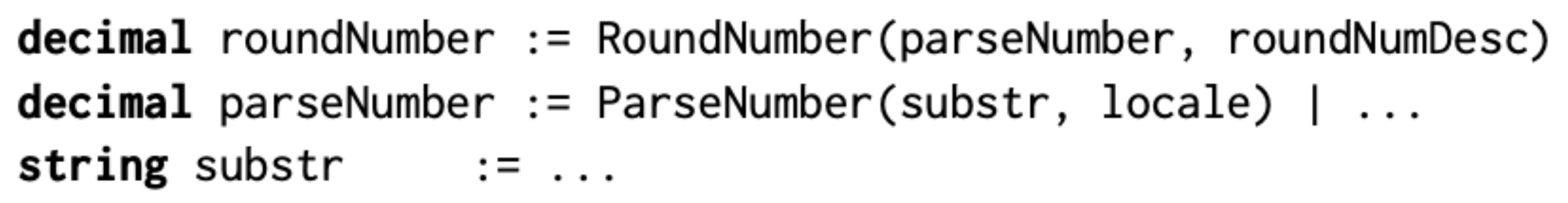


```
decimal roundNumber := RoundNumber(parseNumber, roundNumDesc)
decimal parseNumber := ParseNumber(substr, locale) | ...
string substr     := ...
```

Limitations of Top-Down and Bottom-Up

Example

- Input: “The price is $24.58 and 46 units are available.”
- Output: 24.00
- Operations: roundNumber, parseNumber, subset
- Start from non-terminal roundNumber.
- Expected: roundNumber(parseNumber(substr(Input, …)))

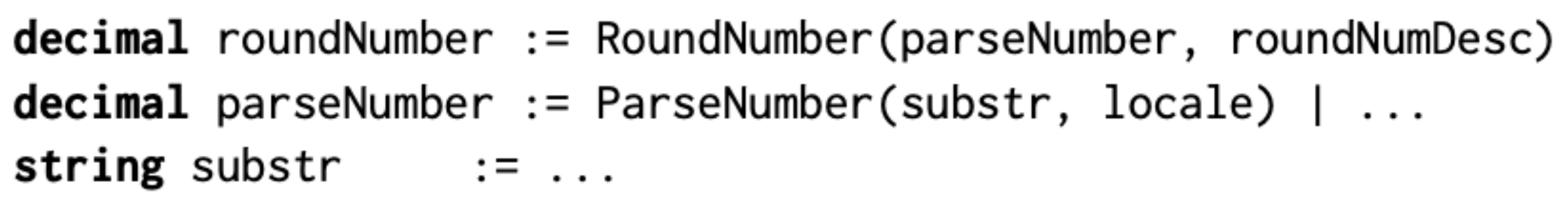


```
decimal roundNumber := RoundNumber(parseNumber, roundNumDesc)
decimal parseNumber := ParseNumber(substr, locale) | ...
string substr      := ...
```

Limitations of Top-Down and Bottom-Up

Example

- Input: “The price is $24.58 and 46 units are available.”
- Output: 24.00
- Operations: roundNumber, parseNumber, subset
- Start from non-terminal roundNumber.
- Expected: roundNumber(parseNumber(substr(Input, …)))



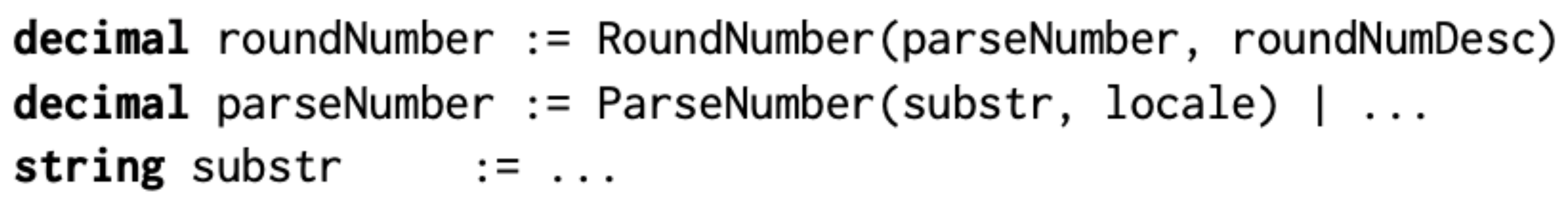
```
decimal roundNumber := RoundNumber(parseNumber, roundNumDesc)
decimal parseNumber := ParseNumber(substr, locale) | ...
string substr       := ...
```

Limitations

Limitations of Top-Down and Bottom-Up

Example

- Input: “The price is $24.58 and 46 units are available.”
- Output: 24.00
- Operations: roundNumber, parseNumber, subset
- Start from non-terminal roundNumber.
- Expected: roundNumber(parseNumber(substr(Input, …)))



```
decimal roundNumber := RoundNumber(parseNumber, roundNumDesc)
decimal parseNumber := ParseNumber(substr, locale) | ...
string substr      := ...
```

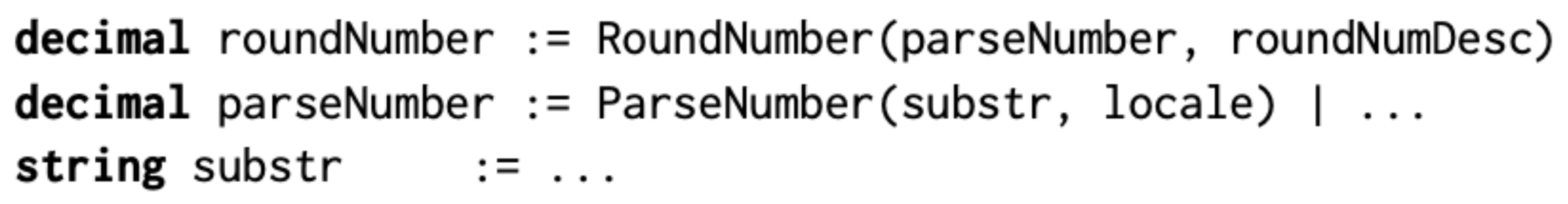
Limitations

- Top-Down: Inverse of roundNumber() is infinite

Limitations of Top-Down and Bottom-Up

Example

- Input: “The price is $24.58 and 46 units are available.”
- Output: 24.00
- Operations: roundNumber, parseNumber, subset
- Start from non-terminal roundNumber.
- Expected: roundNumber(parseNumber(substr(Input, …)))



```
decimal roundNumber := RoundNumber(parseNumber, roundNumDesc)
decimal parseNumber := ParseNumber(substr, locale) | ...
string substr      := ...
```

Limitations

- Top-Down: Inverse of roundNumber() is infinite
- Bottom-Up: Too many possible substring from the input

Cut: Divide into two synthesis

Cut: Divide into two synthesis

Intuition

Cut: Divide into two synthesis

Intuition

- Q. In a solution program,
  what must come into parseNum in order for valid result?

Cut: Divide into two synthesis

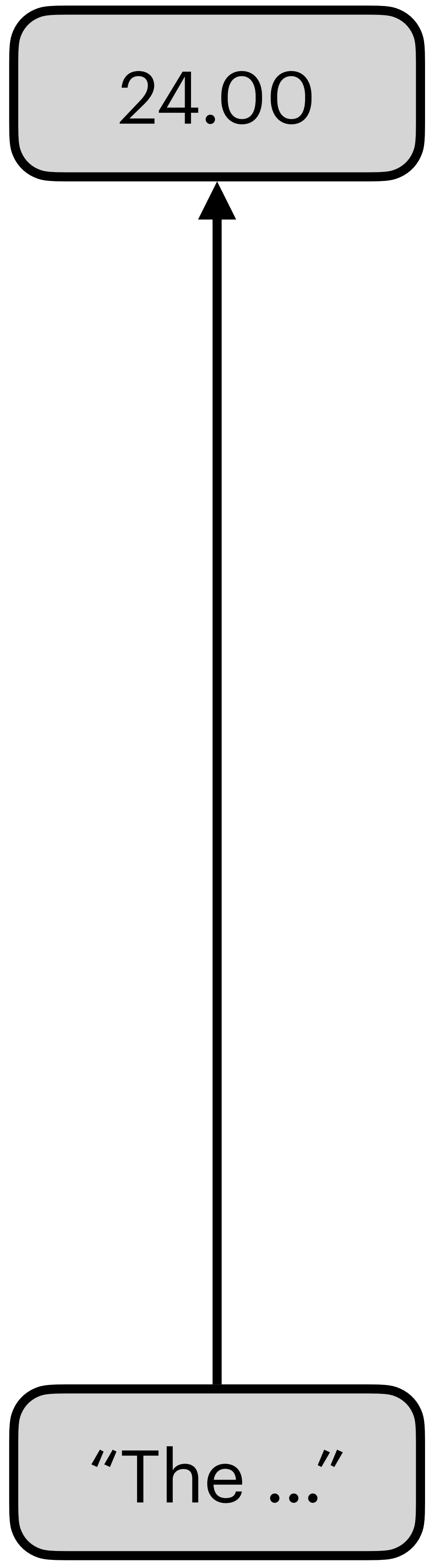
Intuition

- Q. In a solution program,
  what must come into parseNum in order for valid result?
- A. String with only numbers

Cut: Divide into two synthesis

Intuition

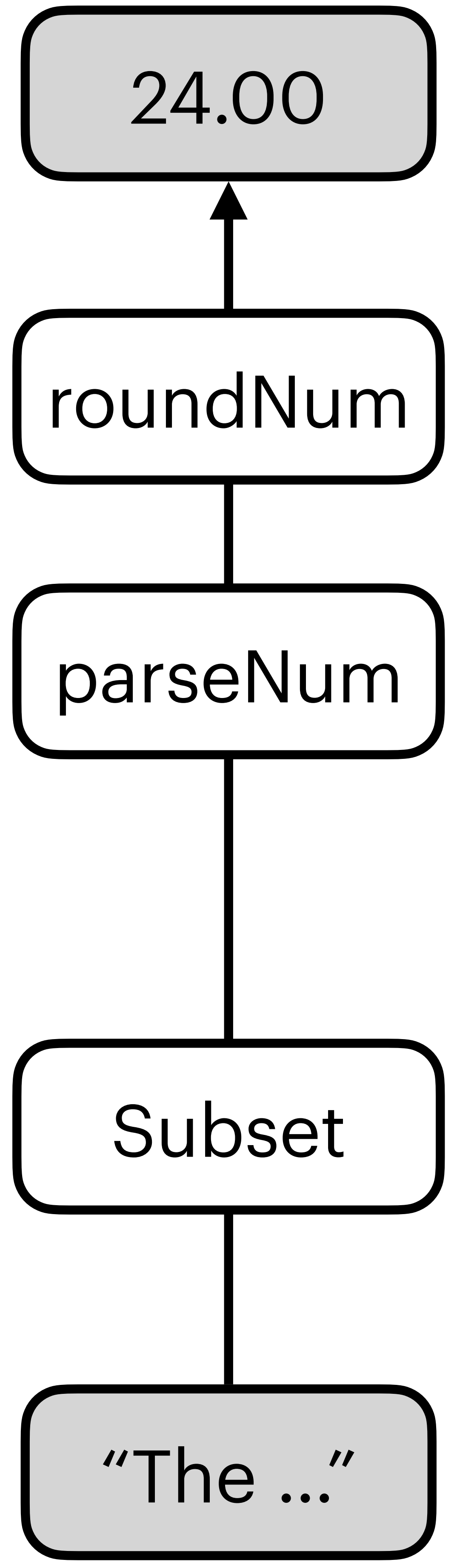
- Q. In a solution program,
  what must come into parseNum in order for valid result?
- A. String with only numbers



Cut: Divide into two synthesis

Intuition

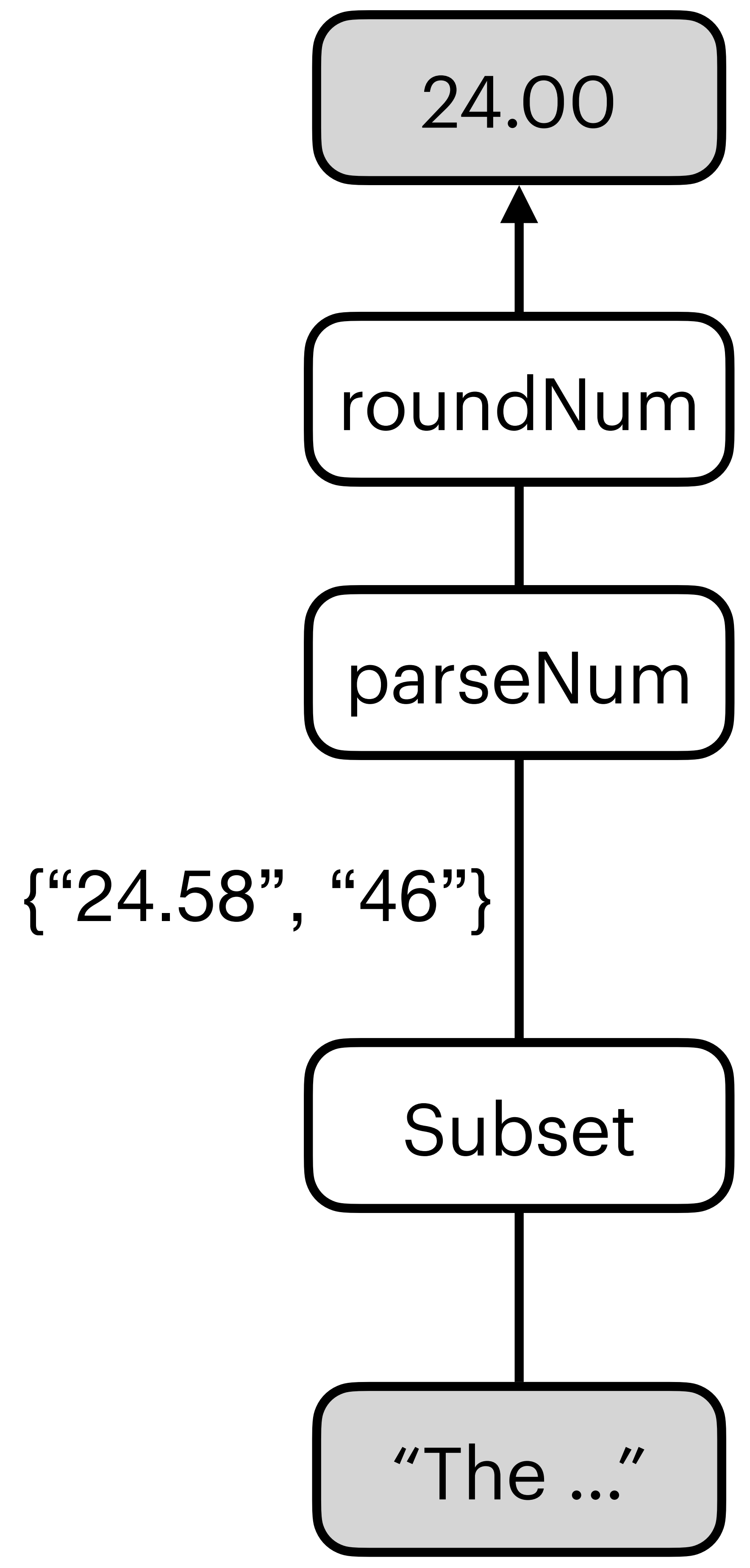
- Q. In a solution program,
  what must come into parseNum in order for valid result?
- A. String with only numbers



Cut: Divide into two synthesis

Intuition

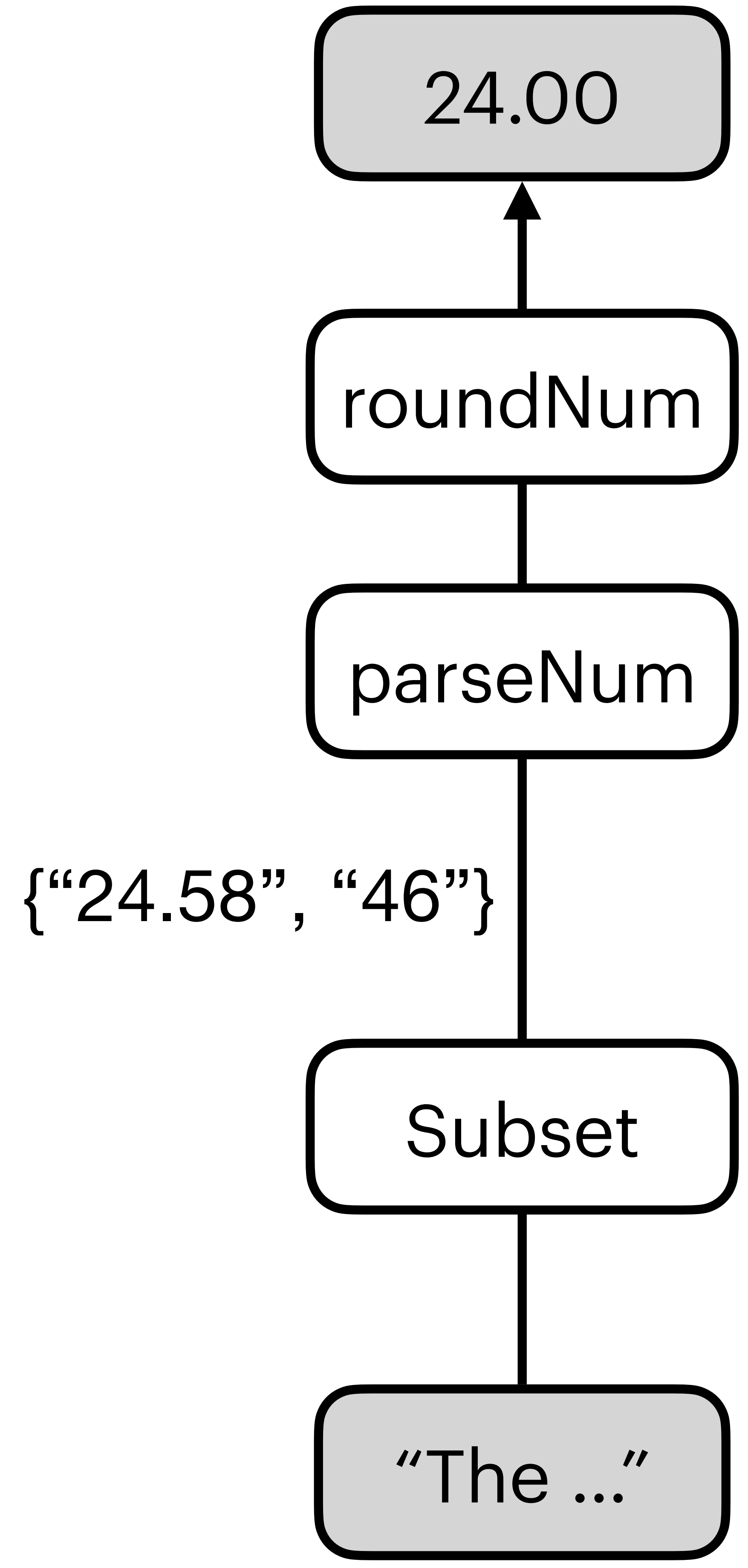
- Q. In a solution program,
  what must come into parseNum in order for valid result?
- A. String with only numbers



Cut: Divide into two synthesis

Intuition

- Q. In a solution program,
  what must come into parseNum in order for valid result?
- A. String with only numbers
- Splits the problem before and after such invariants

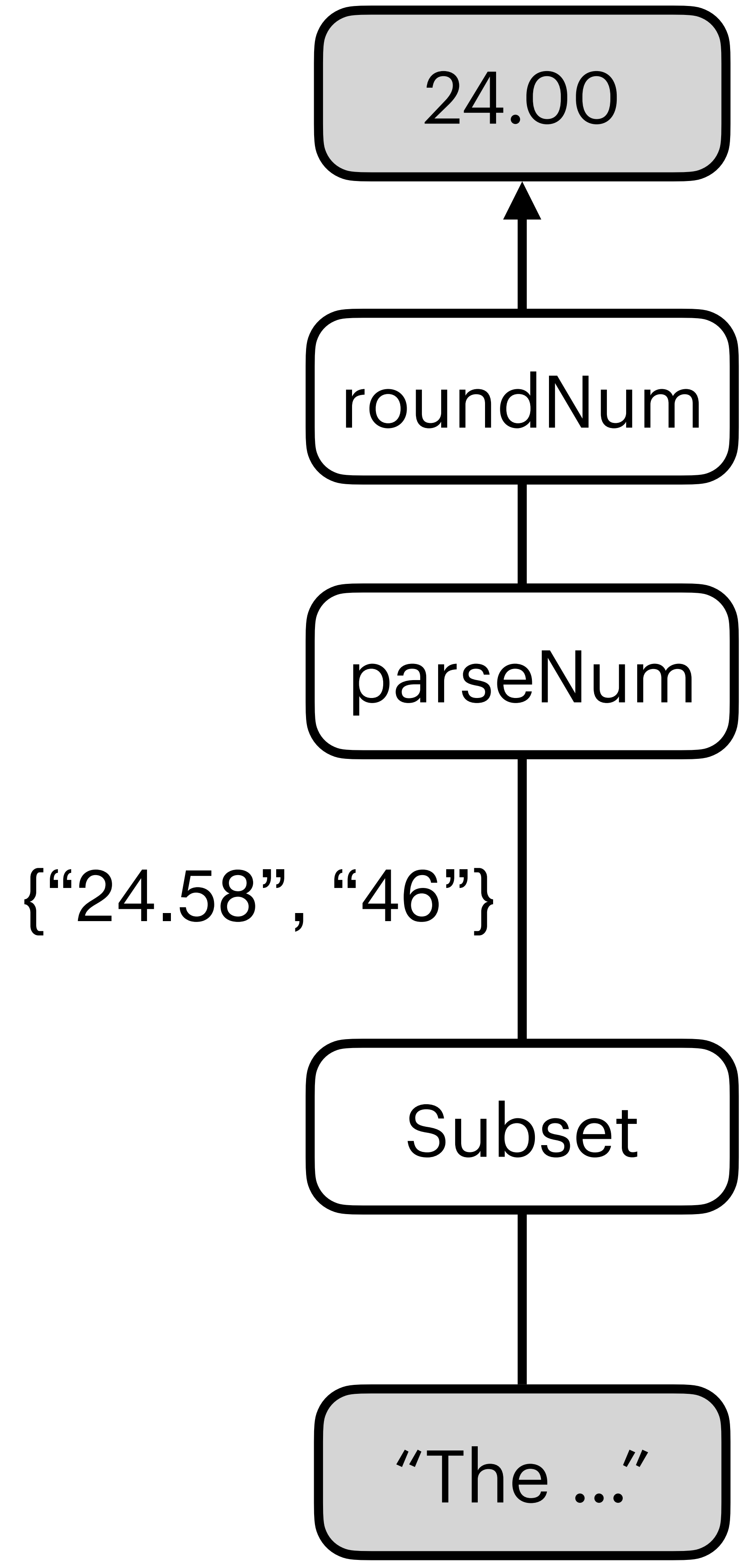


Cut: Divide into two synthesis

Intuition

- Q. In a solution program,
  what must come into parseNum in order for valid result?
- A. String with only numbers
- Splits the problem before and after such invariants

Smaller problems



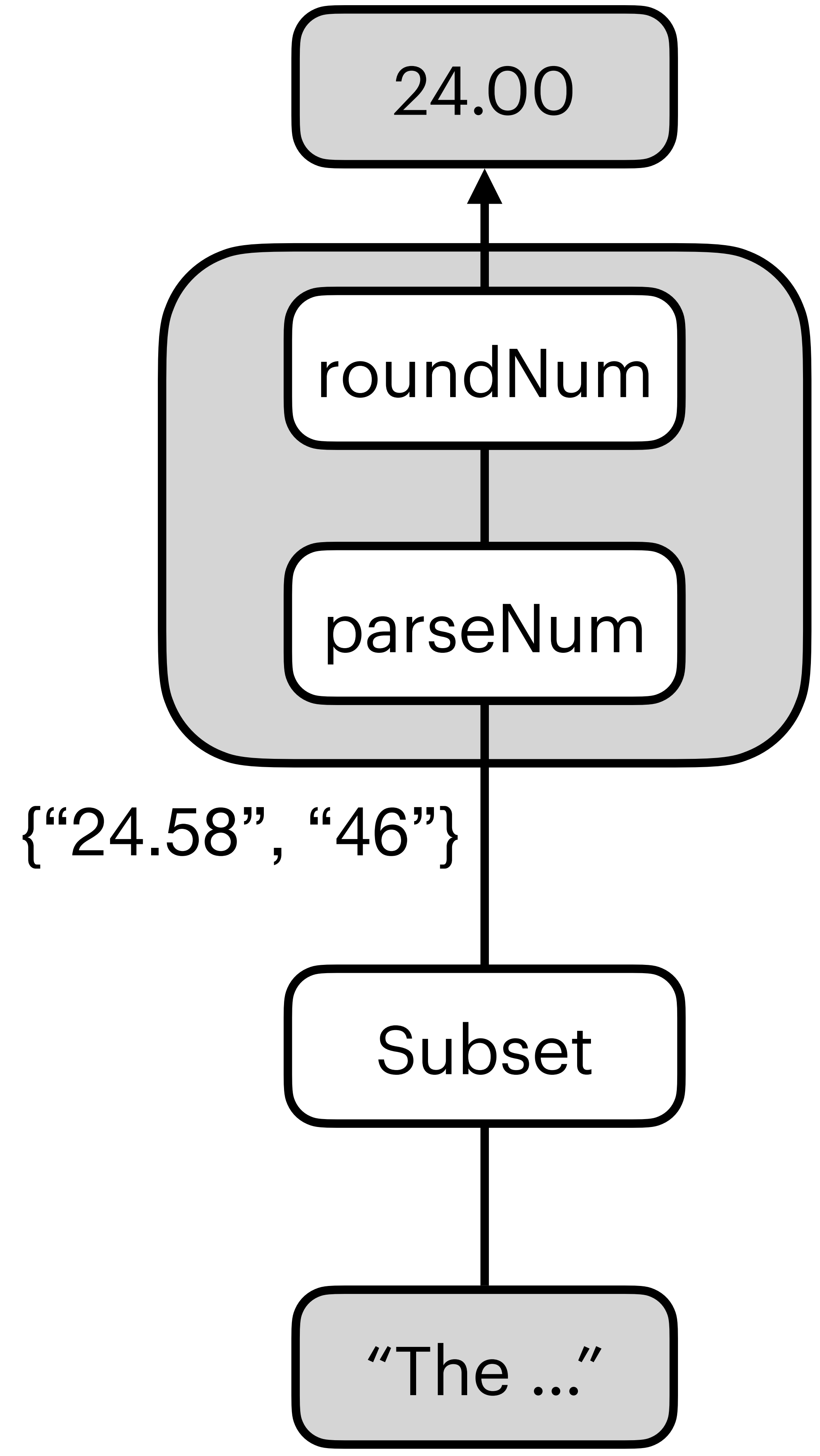
Cut: Divide into two synthesis

Intuition

- Q. In a solution program,
  what must come into parseNum in order for valid result?
- A. String with only numbers
- Splits the problem before and after such invariants

Smaller problems

- "24.58" -> 24.00



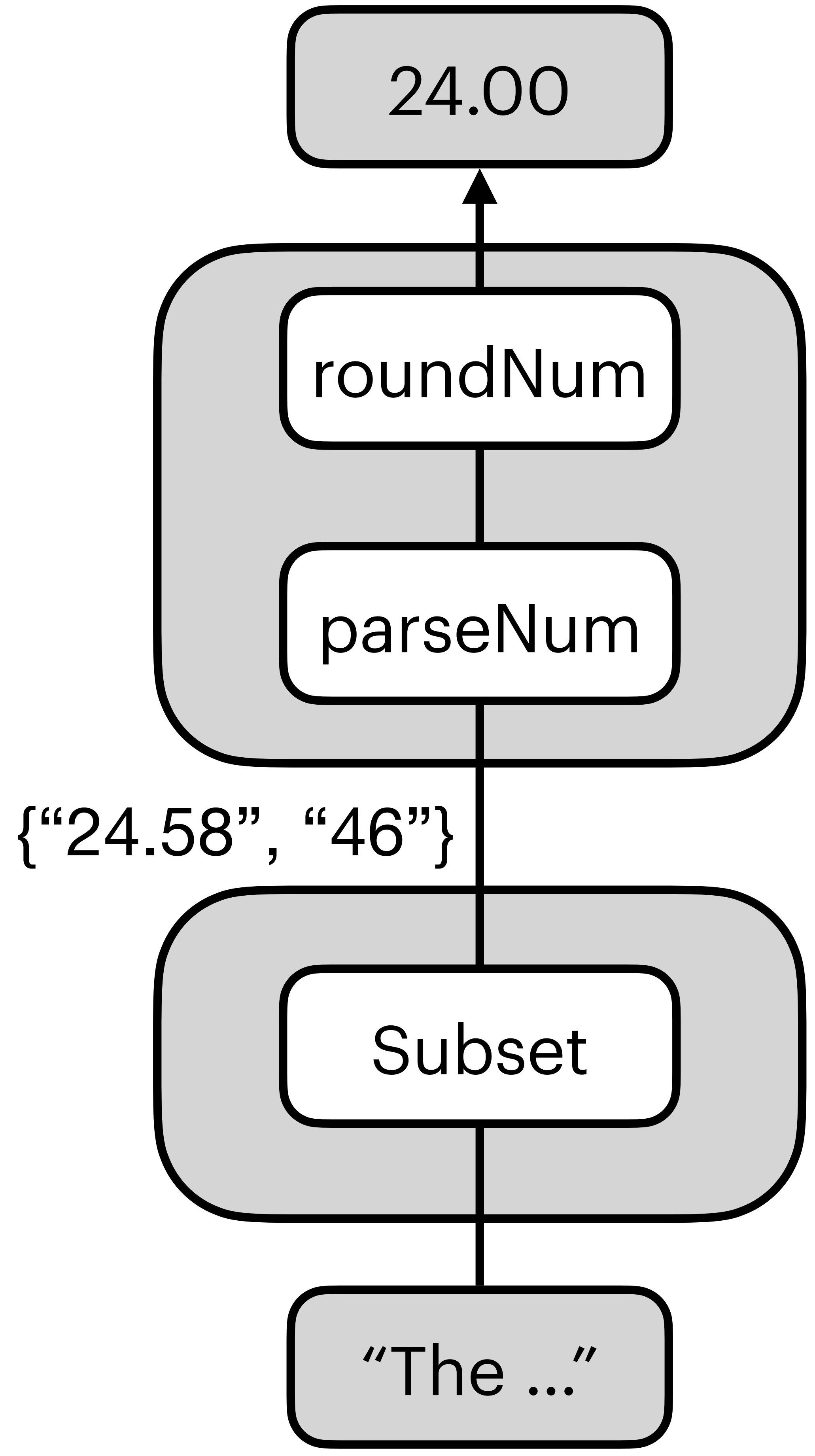
Cut: Divide into two synthesis

Intuition

- Q. In a solution program,
  what must come into parseNum in order for valid result?
- A. String with only numbers
- Splits the problem before and after such invariants

Smaller problems

- “24.58” -> 24.00
- “The …” -> “24.58”



Precedence: Guided synthesis

Precedence: Guided synthesis

Intuition

Precedence: Guided synthesis

Intuition

- More comprehensible, simple programs can be forced to make

Precedence: Guided synthesis

Intuition

- More comprehensible, simple programs can be forced to make
- Some rules will be applied later

Precedence: Guided synthesis

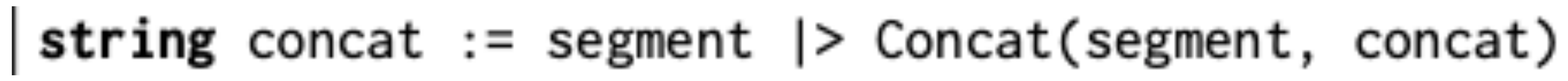
Intuition

- More comprehensible, simple programs can be forced to make
- Some rules will be applied later
- Put guards in DSL

Precedence: Guided synthesis

Intuition

- More comprehensible, simple programs can be forced to make
- Some rules will be applied later
- Put guards in DSL



```
| string concat := segment |> Concat(segment, concat)
```

Evalutation

Evalutation

Baseline

Evalutation

Baseline

- FlashFill, Duet, SmartFill

Evalutation

Baseline

- FlashFill, Duet, SmartFill

Evalutation

Baseline

- FlashFill, Duet, SmartFill

Benchmark

- 886 string transformations

Evalutation

Baseline

- FlashFill, Duet, SmartFill

Benchmark

- 886 string transformations

Criteria

Evalutation

Baseline

- FlashFill, Duet, SmartFill

Benchmark

- 886 string transformations

Criteria

- Correctness, Efficiency, Readability

Correctness

Correctness

RQ1. Can FlashFill++ solve more benchmarks?

Correctness

RQ1. Can FlashFill++ solve more benchmarks?

- Yes, FlashFill++ solves more benchmarks

Correctness

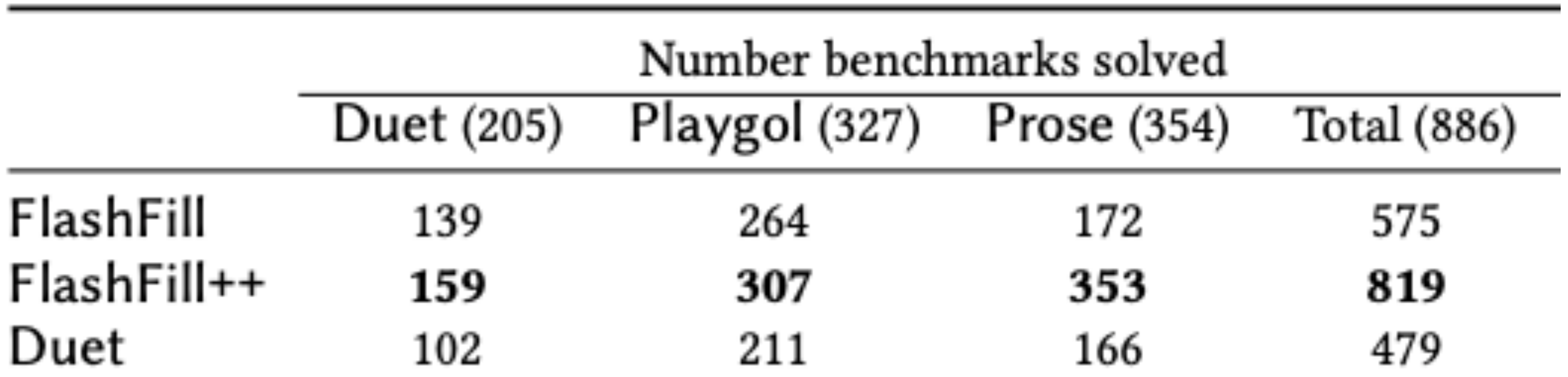
RQ1. Can FlashFill++ solve more benchmarks?

- Yes, FlashFill++ solves more benchmarks
- 92% solved by FlashFill++, 64% by second best baseline

Correctness

RQ1. Can FlashFill++ solve more benchmarks?

- Yes, FlashFill++ solves more benchmarks
- 92% solved by FlashFill++, 64% by second best baseline

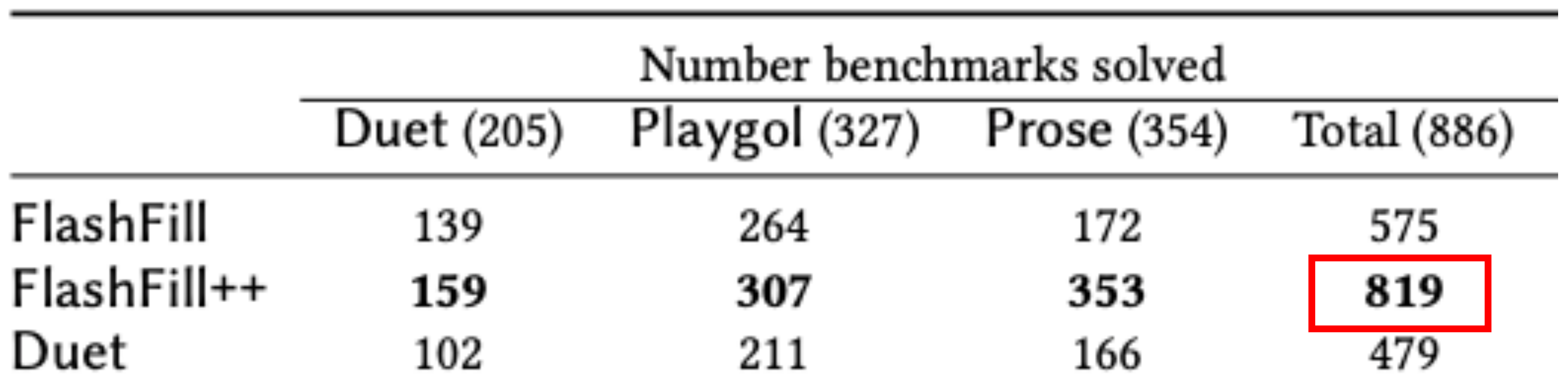


	Number benchmarks solved			
	Duet (205)	Playgol (327)	Prose (354)	Total (886)
FlashFill	139	264	172	575
FlashFill++	**159**	**307**	**353**	**819**
Duet	102	211	166	479

Correctness

RQ1. Can FlashFill++ solve more benchmarks?

- Yes, FlashFill++ solves more benchmarks
- 92% solved by FlashFill++, 64% by second best baseline



	Number benchmarks solved			
	Duet (205)	Playgol (327)	Prose (354)	Total (886)
FlashFill	139	264	172	575
FlashFill++	**159**	**307**	**353**	**819**
Duet	102	211	166	479

Efficiency

Efficiency

RQ2. Is FlashFill++ more efficient?

Efficiency

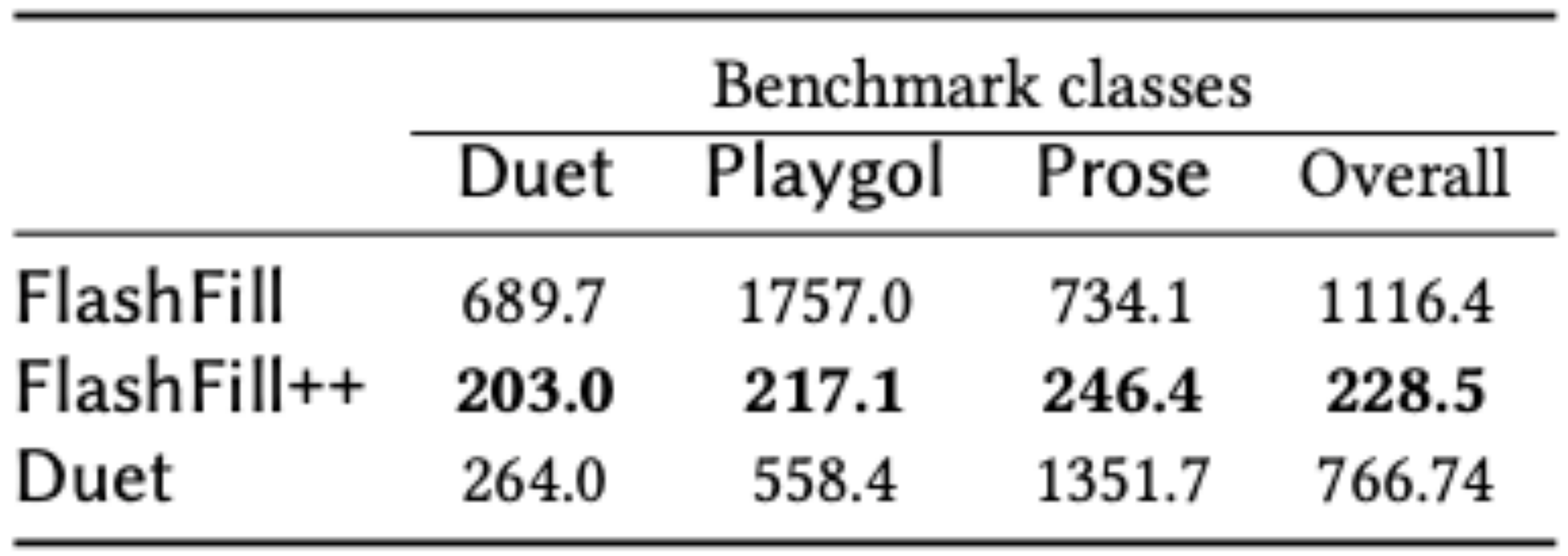
RQ2. Is FlashFill++ more efficient?

- Yes, FlashFill++ is 3X faster than the baselines

Efficiency

RQ2. Is FlashFill++ more efficient?

- Yes, FlashFill++ is 3X faster than the baselines

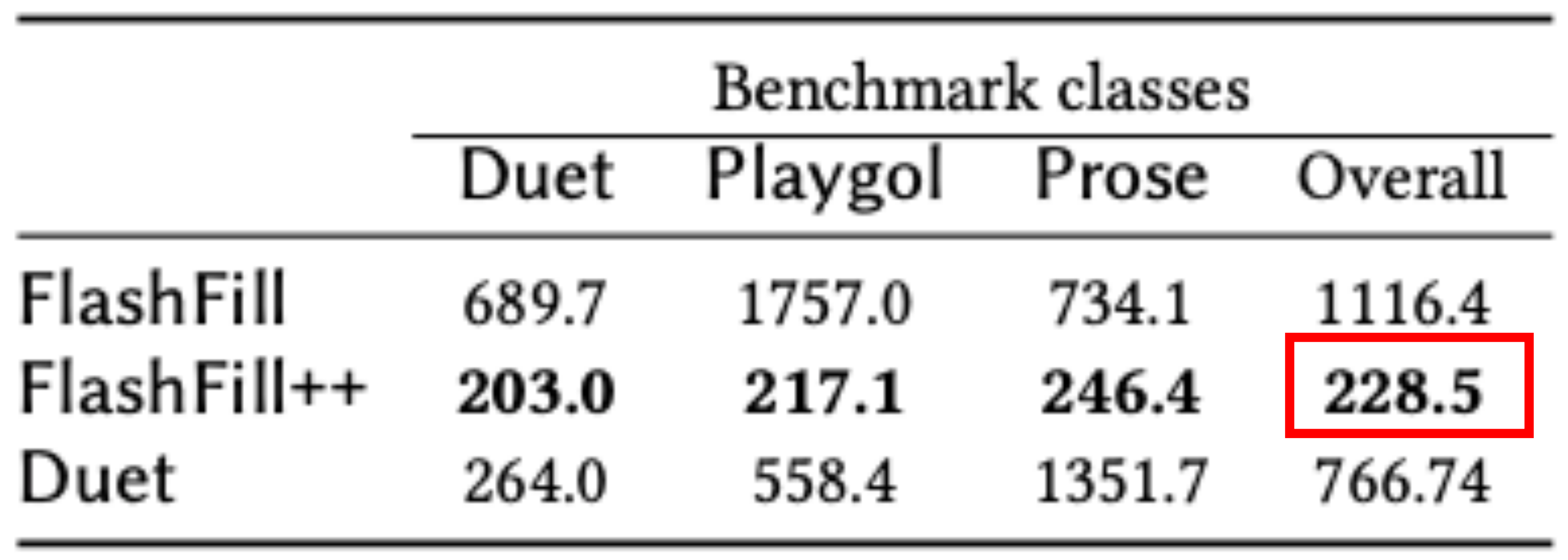


	Benchmark classes			
	Duet	Playgol	Prose	Overall
FlashFill	689.7	1757.0	734.1	1116.4
FlashFill++	**203.0**	**217.1**	**246.4**	**228.5**
Duet	264.0	558.4	1351.7	766.74

Efficiency

RQ2. Is FlashFill++ more efficient?

- Yes, FlashFill++ is 3X faster than the baselines

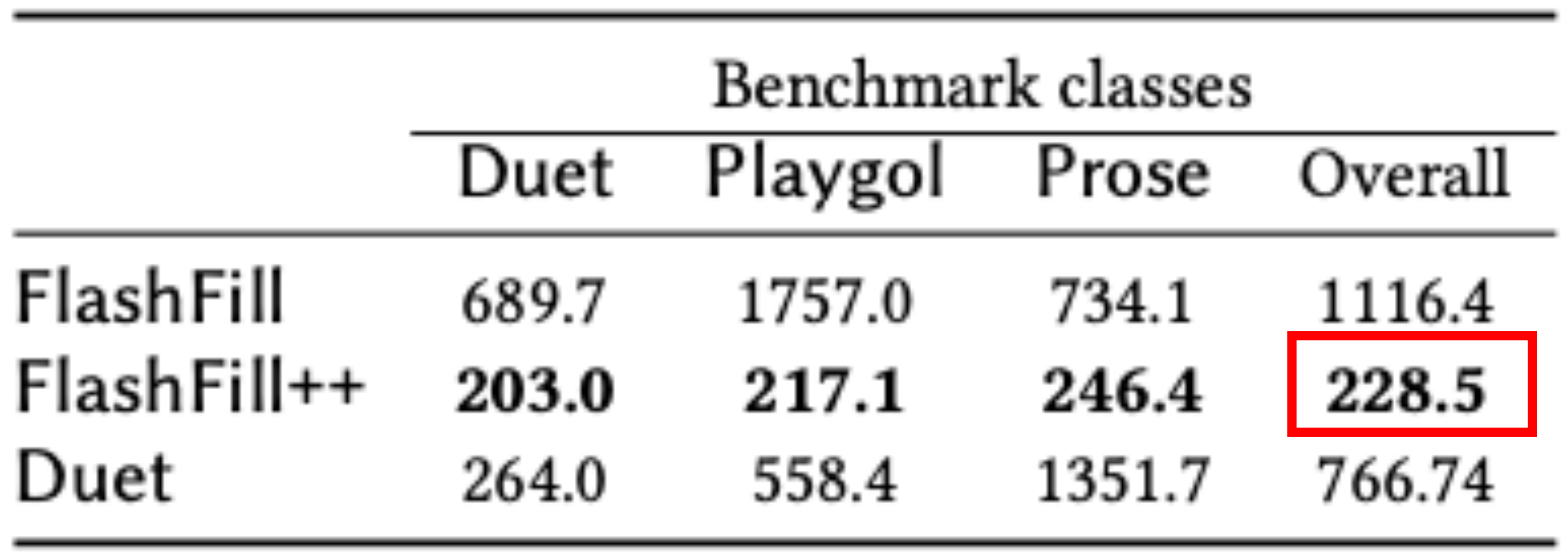


	Benchmark classes			
	Duet	Playgol	Prose	Overall
FlashFill	689.7	1757.0	734.1	1116.4
FlashFill++	**203.0**	**217.1**	**246.4**	**228.5**
Duet	264.0	558.4	1351.7	766.74

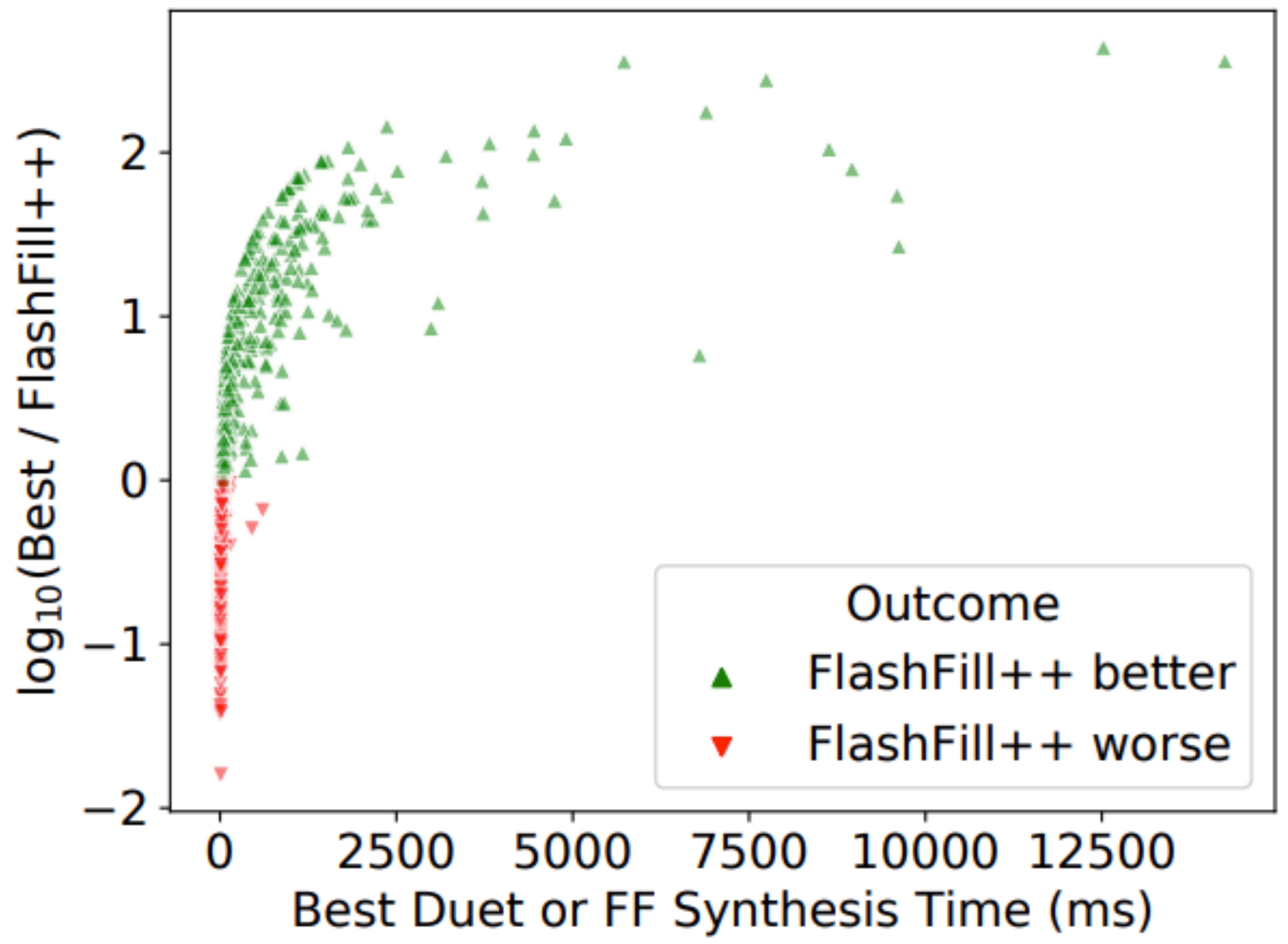
Efficiency

RQ2. Is FlashFill++ more efficient?

- Yes, FlashFill++ is 3X faster than the baselines



	Benchmark classes			
	Duet	Playgol	Prose	Overall
FlashFill	689.7	1757.0	734.1	1116.4
FlashFill++	**203.0**	**217.1**	**246.4**	**228.5**
Duet	264.0	558.4	1351.7	766.74



Readability

Readability

RQ3. Does FlashFill++ generate more readable programs?

Readability

RQ3. Does FlashFill++ generate more readable programs?

- Yes, FlashFill++ is more readable.

Readability

RQ3. Does FlashFill++ generate more readable programs?

- Yes, FlashFill++ is more readable.

- 100 participants on 10 examples

Readability

RQ3. Does FlashFill++ generate more readable programs?

- Yes, FlashFill++ is more readable.

- 100 participants on 10 examples
  - 92% : FlashFill++ is more readable than FlashFill

Readability

RQ3. Does FlashFill++ generate more readable programs?

- Yes, FlashFill++ is more readable.

- 100 participants on 10 examples
  - 92% : FlashFill++ is more readable than FlashFill
  - 81% : FlashFill++ generates code more similar to mine

Comment

Comment

Consistency

Comment

Consistency

- Future work after 13 years

Comment

Consistency

- Future work after 13 years

Response of Academia to the Industry

Comment

Consistency

- Future work after 13 years

Response of Academia to the Industry

- Solving real world problems

Comment

Consistency

- Future work after 13 years

Response of Academia to the Industry

- Solving real world problems

Too much manual effort

Comment

Consistency

- Future work after 13 years

Response of Academia to the Industry

- Solving real world problems

Too much manual effort

- Cuts need to be predefined

Comment

Consistency

- Future work after 13 years

Response of Academia to the Industry

- Solving real world problems

Too much manual effort

- Cuts need to be predefined
- Precedence must be predefined

Summary

Summary

Challenges

Summary

Challenges

- Unreadable code

Summary

Challenges

- Unreadable code
- Ignorance on data types

Summary

Challenges

- Unreadable code
- Ignorance on data types

Solution: FlashFill++

Summary

Challenges

- Unreadable code
- Ignorance on data types

Solution: FlashFill++

- Exploits larger DSL

Summary

Challenges

- Unreadable code
- Ignorance on data types

Solution: FlashFill++

- Exploits larger DSL
- Efficient search

Summary

Challenges

- Unreadable code
- Ignorance on data types

Solution: FlashFill++

- Exploits larger DSL
- Efficient search
  - Cut

Summary

Challenges

- Unreadable code
- Ignorance on data types

Solution: FlashFill++

- Exploits larger DSL
- Efficient search
  - Cut
  - Precedence