


AAAI - 20

TreeGen: A Tree-Based Transformer Architecture for Code Generation

Zeyu Sun, Qihao Zhu, Yingfei Xiong, Yican Sun, Lili Mou, Lu Zhang

Peking University

PART ONE

Introduction

Code Generation

Open a file, F1

Python: f = open(F1, “r”)
C: FILE *f = fopen(F1, “r”);

Print A

Python: print (A)
Java: System.out.print(A)

list airport

Lambda Calculus: lambda $0 e (airport $0)

Code Generation

Open a file, F1

Print A

List airport

...

Generating code from natural language description.

Code generation is beneficial in various scenarios.

Game Development

Beginner

Programmer

Code Generation

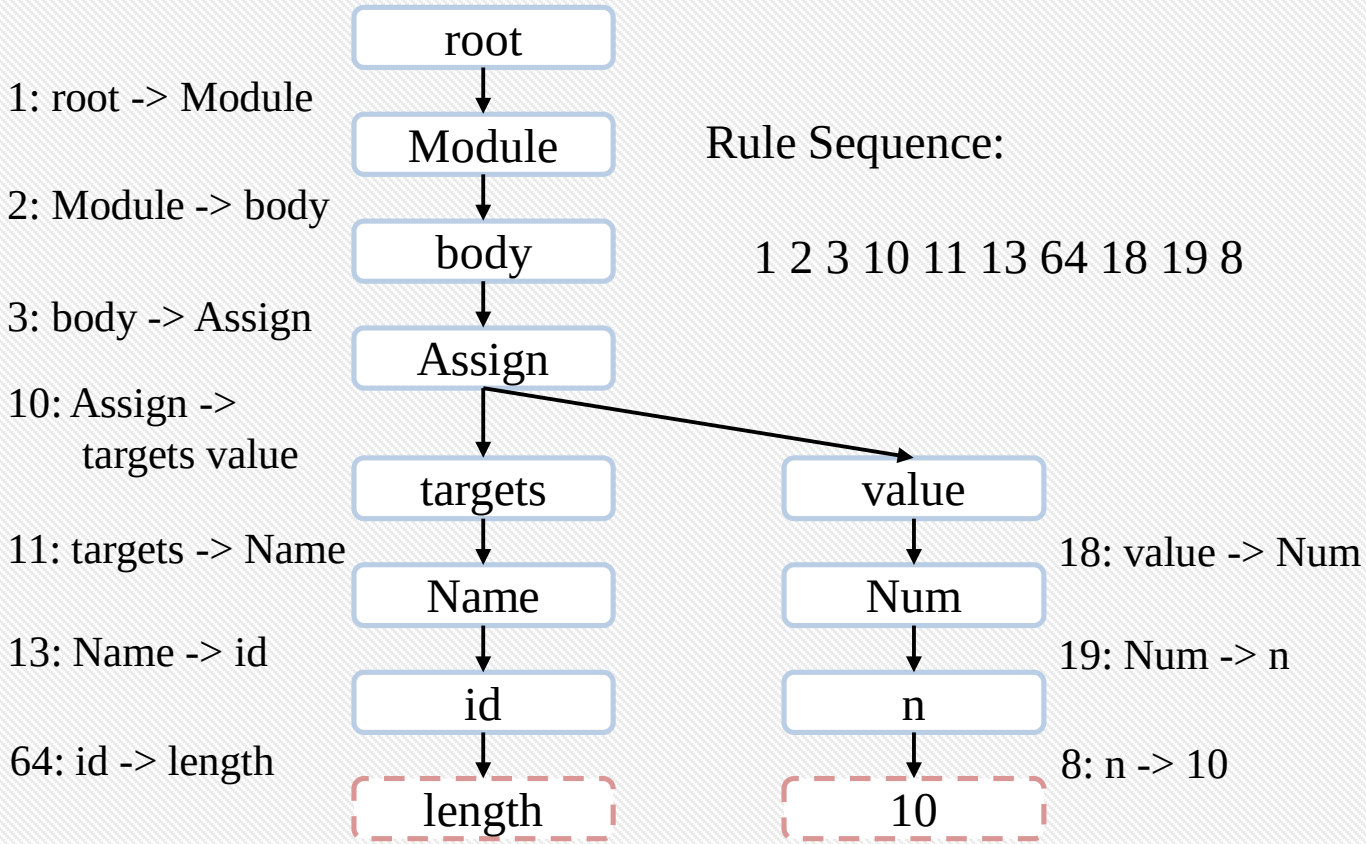
- State-of-the-art approaches generate code by predicting a sequence of grammar rules.

Length = 10

Code Generation

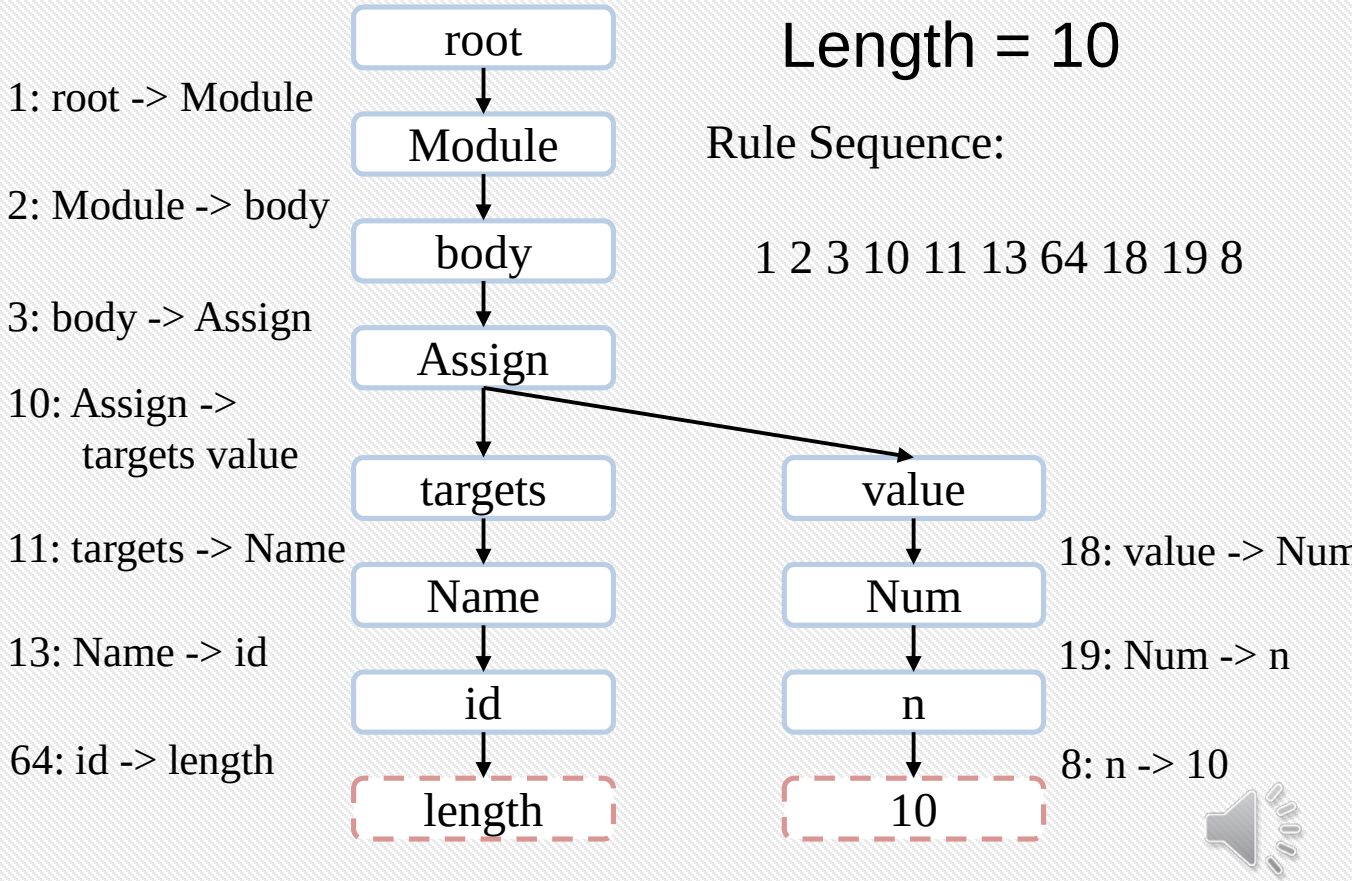
- State-of-the-art approaches generate code by predicting a sequence of grammar rules.

Length = 10



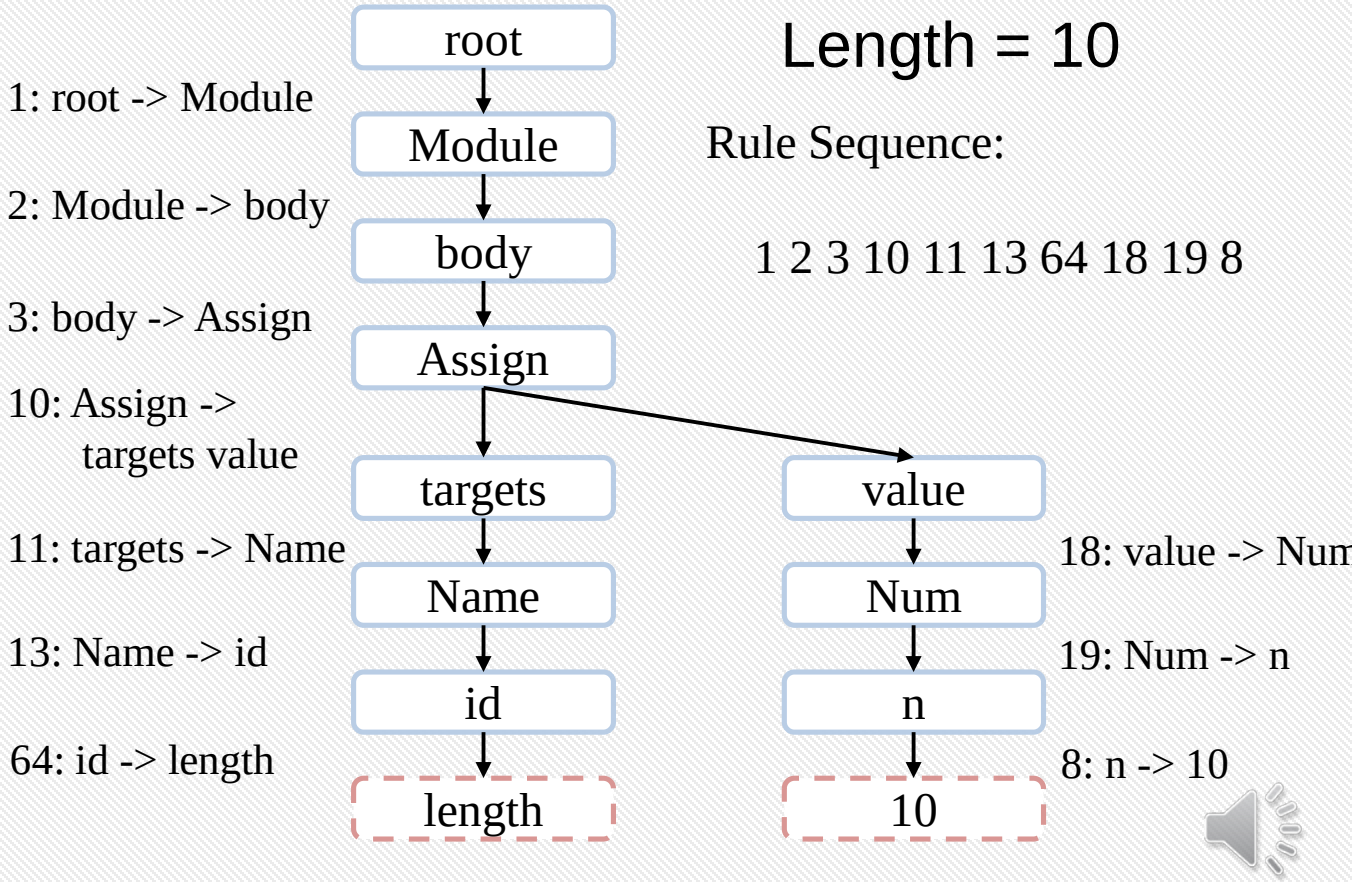
Generating Code (AST)

- We model code generation as a series of classification problems of grammar rules.
  - The programs can be decomposed into several context-free grammar rules and parsed as an AST.



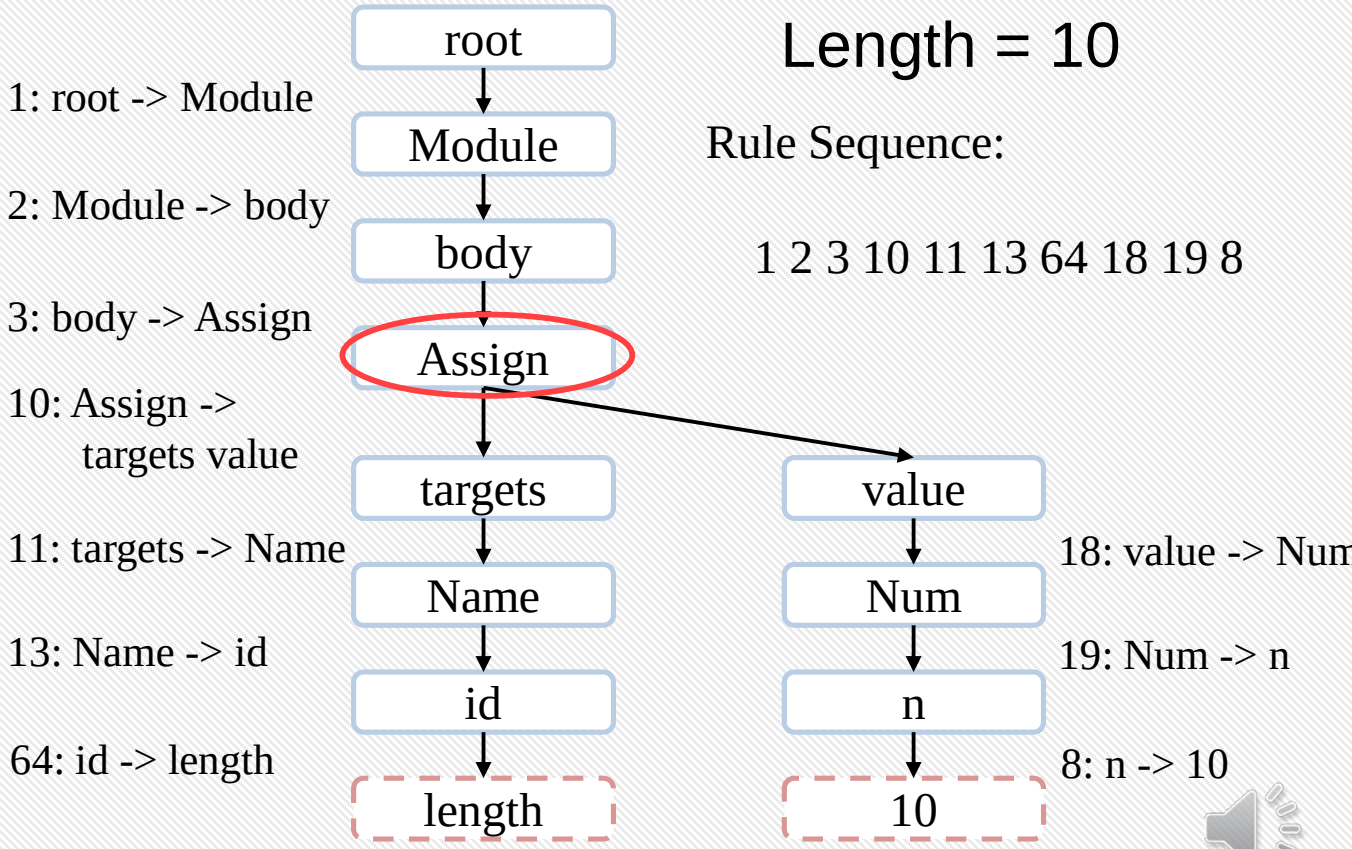
Generating Code (AST)

- We model code generation as a series of classification problems of grammar rules.
  - The programs can be decomposed into several context-free grammar rules and parsed as an AST.
  - AST-based code generation could be thought of as expanding a non-terminal node by a grammar rule.



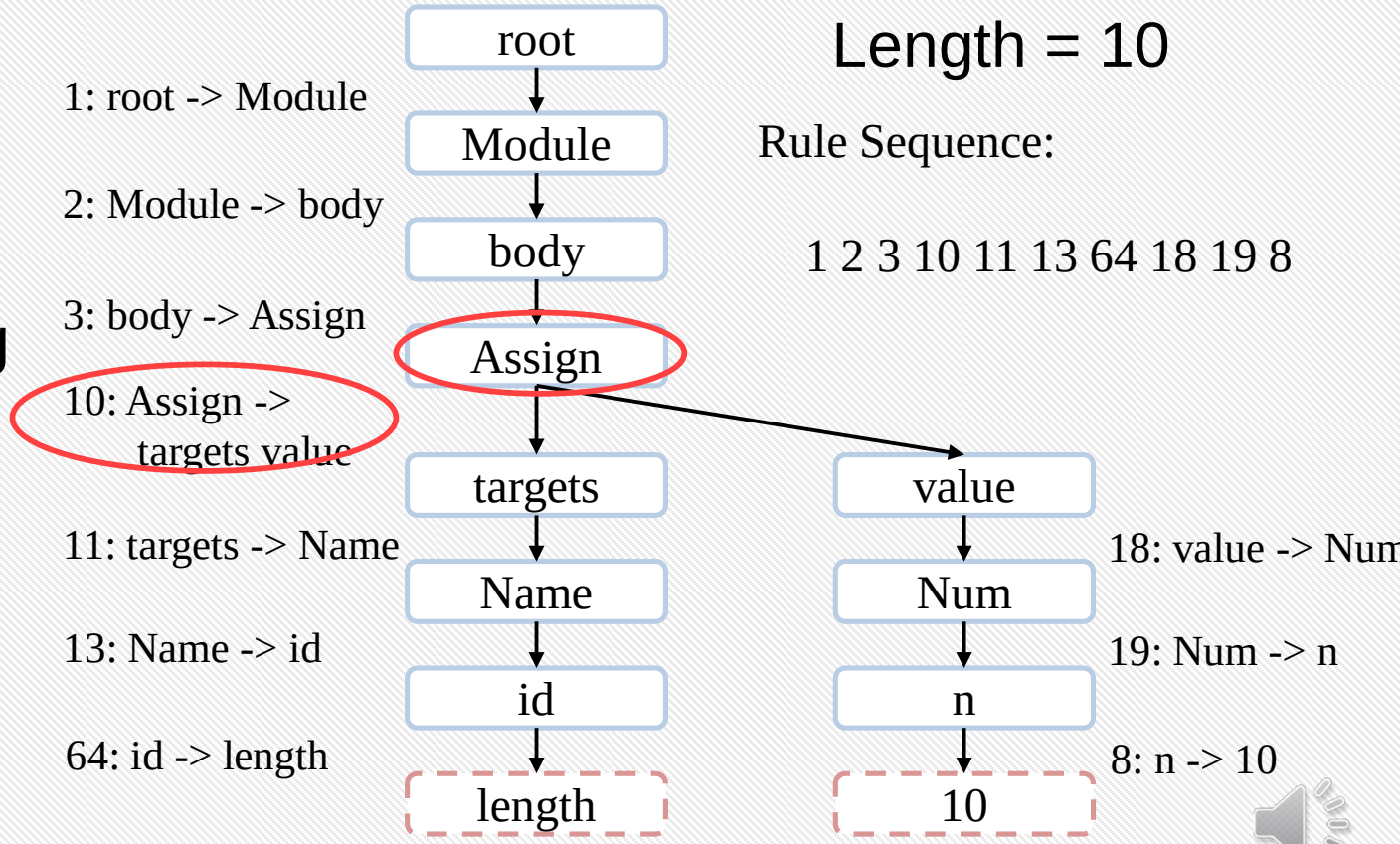
Generating Code (AST)

- We model code generation as a series of classification problems of grammar rules.
  - The programs can be decomposed into several context-free grammar rules and parsed as an AST.
  - AST-based code generation could be thought of as expanding a non-terminal node by a grammar rule.



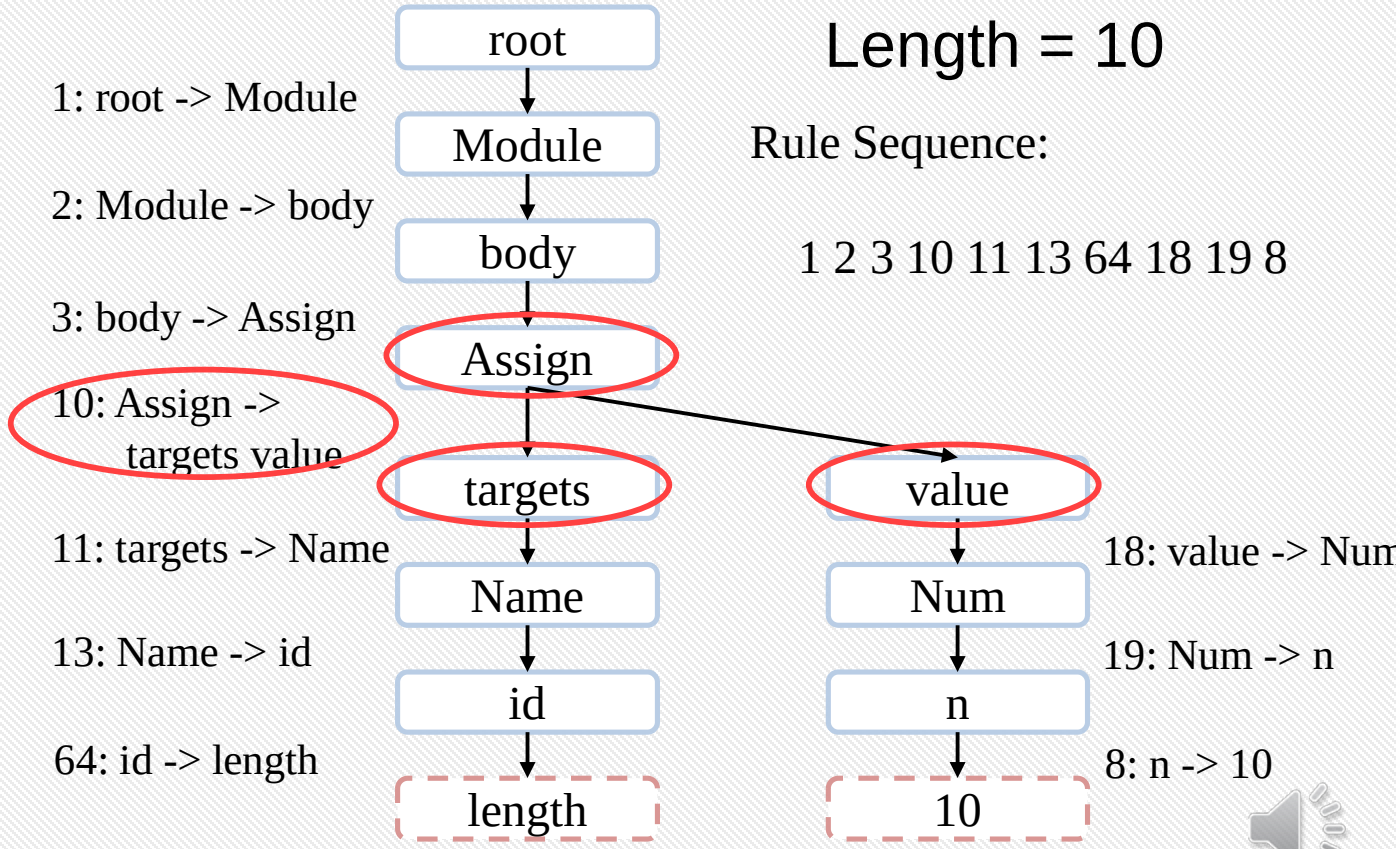
Generating Code (AST)

- We model code generation as a series of classification problems of grammar rules.
  - The programs can be decomposed into several context-free grammar rules and parsed as an AST.
  - AST-based code generation could be thought of as expanding a non-terminal node by a grammar rule.



Generating Code (AST)

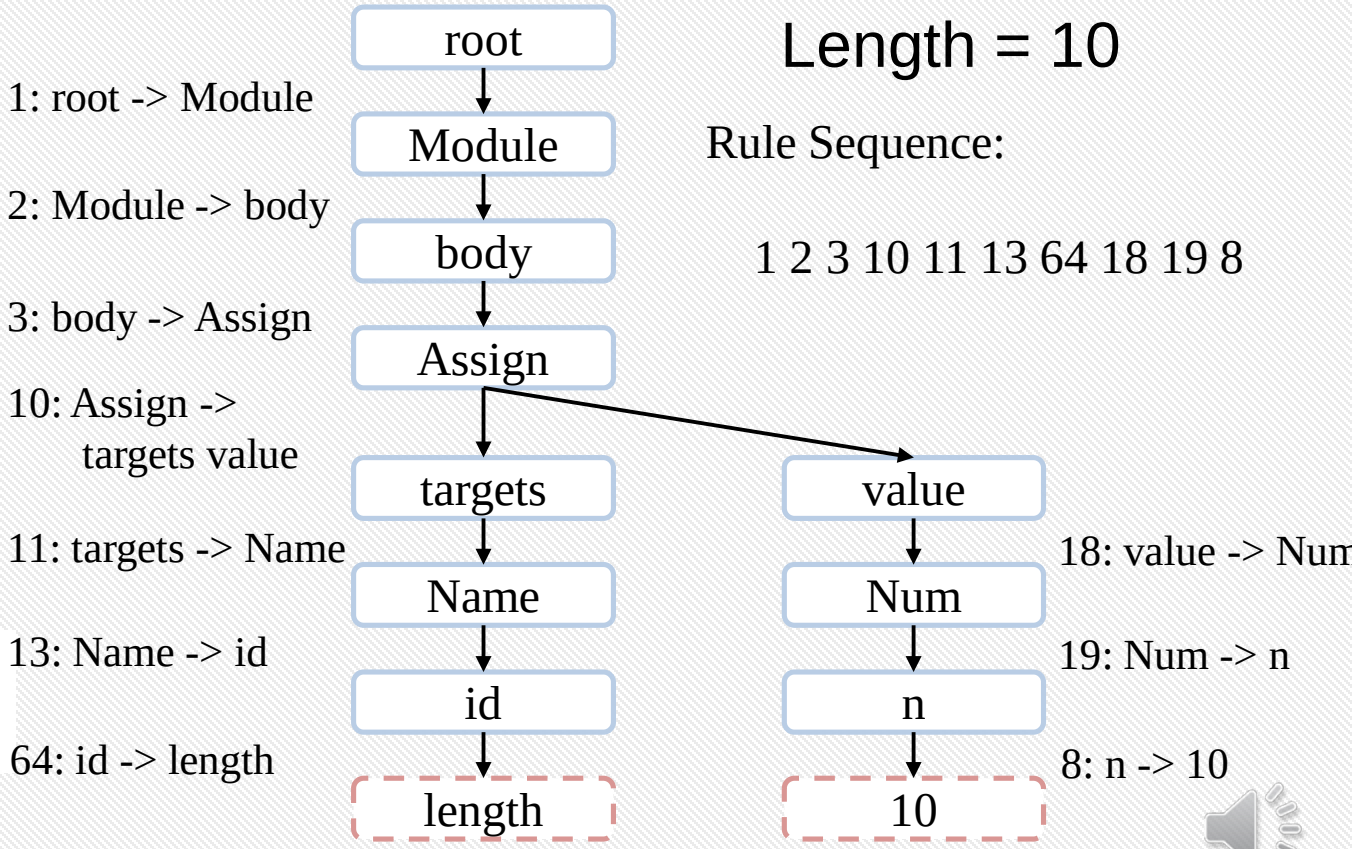
- We model code generation as a series of classification problems of grammar rules.
  - The programs can be decomposed into several context-free grammar rules and parsed as an AST.
  - AST-based code generation could be thought of as expanding a non-terminal node by a grammar rule.



Generating Code (AST)

- We model code generation as a series of classification problems of grammar rules.
  - The programs can be decomposed into several context-free grammar rules and parsed as an AST.
  - AST-based code generation could be thought of as expanding a non-terminal node by a grammar rule.

$$p(\text{code}) = \prod_{i=1}^{P} p(r_i \mid \text{NL input}, r_i, \cdots, r_{i-1})$$



Code Generation

- The classification of grammar rules faces two main challenges.
  - The first challenge is the long-dependency problem (Bengio, Simard, and Frasconi 1994).
  - The second challenge is the representation of code structures.

Code Generation

- The classification of grammar rules faces two main challenges.
  - The first challenge is the long-dependency problem (Bengio, Simard, and Frasconi 1994).

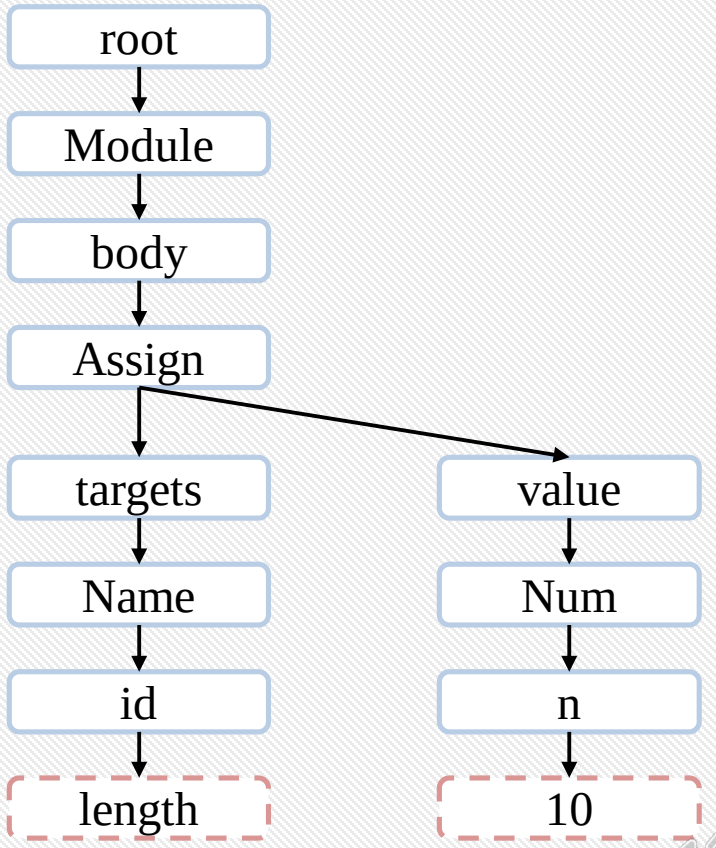
```
10 :Max_Length = 10
.....
100:if len(a) < Max_Length:
```

  - The second challenge is the representation of code structures.

Code Generation

- The classification of grammar rules faces two main challenges.
  - The first challenge is the long-dependency problem (Bengio, Simard, and Frasconi 1994).

```
10 :Max_Length = 10
.....
100:if len(a) < Max_Length:
```

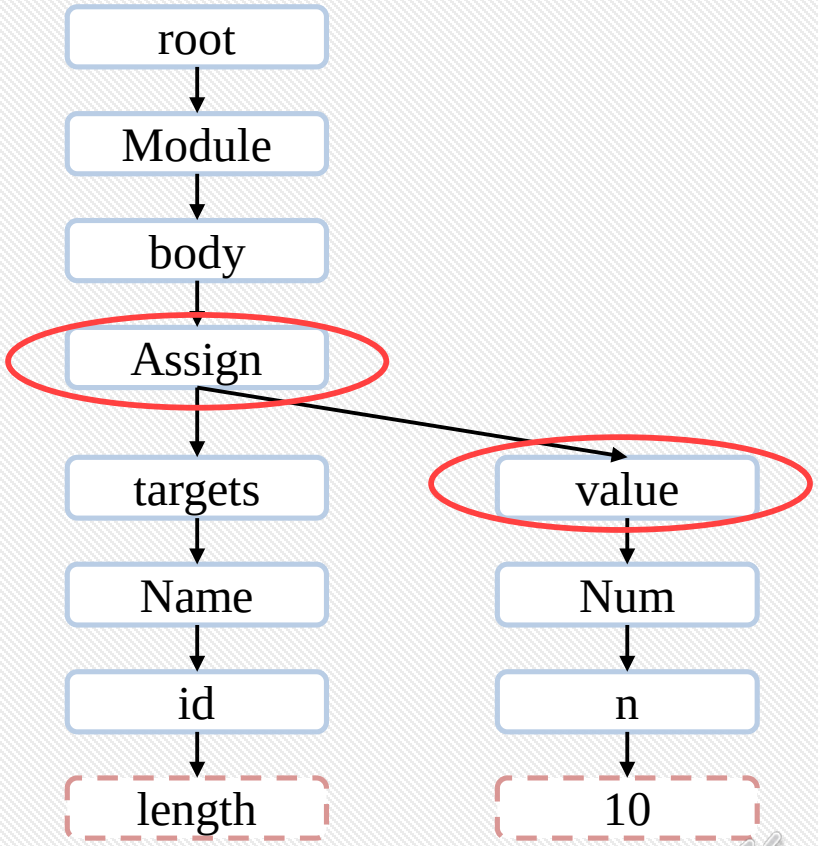


  - The second challenge is the representation of code structures.

Code Generation

- The classification of grammar rules faces two main challenges.
  - The first challenge is the long-dependency problem (Bengio, Simard, and Frasconi 1994).

```
10 :Max_Length
.....
100:if len(a) < Max_Length
```

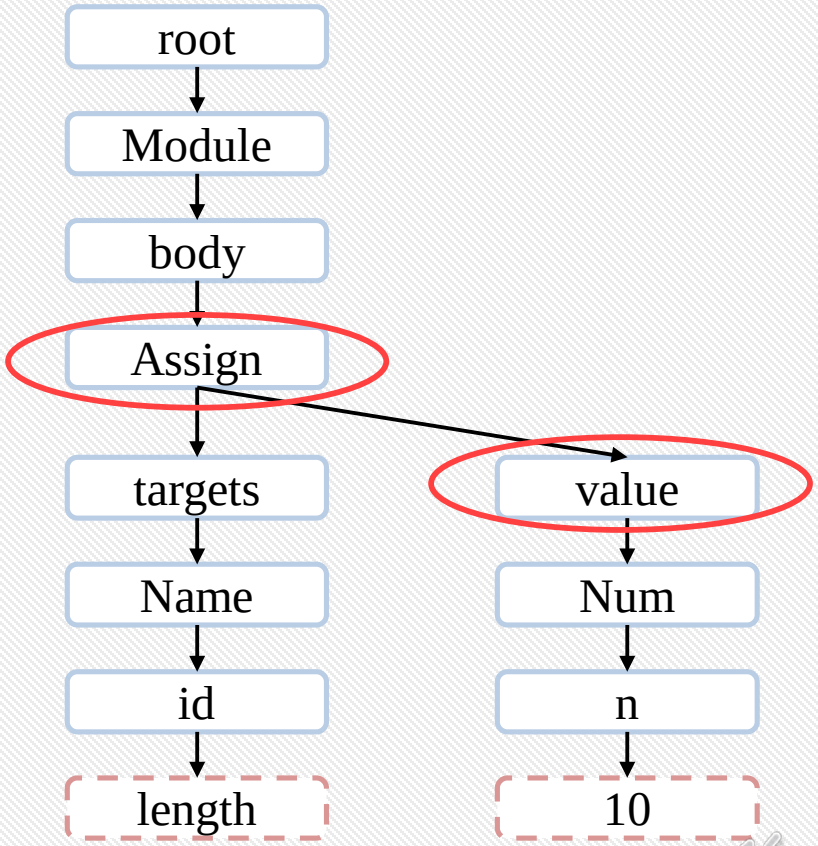


  - The second challenge is the representation of code structures.

Code Generation

- The classification of grammar rules faces two main challenges.
  - The first challenge is the long-dependency problem (Bengio, Simard, and Frasconi 1994).

```
10 :Max_Length
.....
100:if len(a) < Max_Length
```



  - The second challenge is the representation of code structures.
    - However, a “flat” neural architecture cannot capture structure information well.

TreeGen

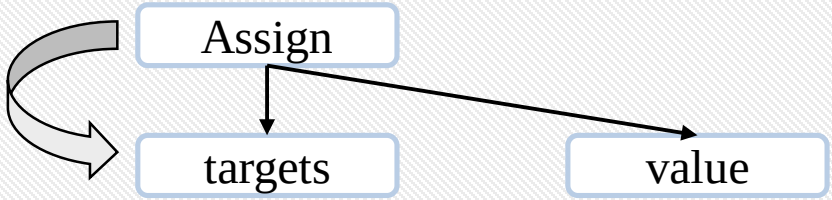
- We propose a novel neural architecture, TreeGen, for the code generation.
  - The first challenge is the long-dependency problem (Bengio, Simard, and Frasconi 1994).

TreeGen

- We propose a novel neural architecture, TreeGen, for the code generation.
  - The first challenge is the long-dependency problem (Bengio, Simard, and Frasconi 1994).
    - TreeGen adopts the Transformer architecture (Vaswani et al. 2017), which is capable of capturing long dependencies.
  - However, the original Transformer architecture cannot utilize tree structures.

TreeGen

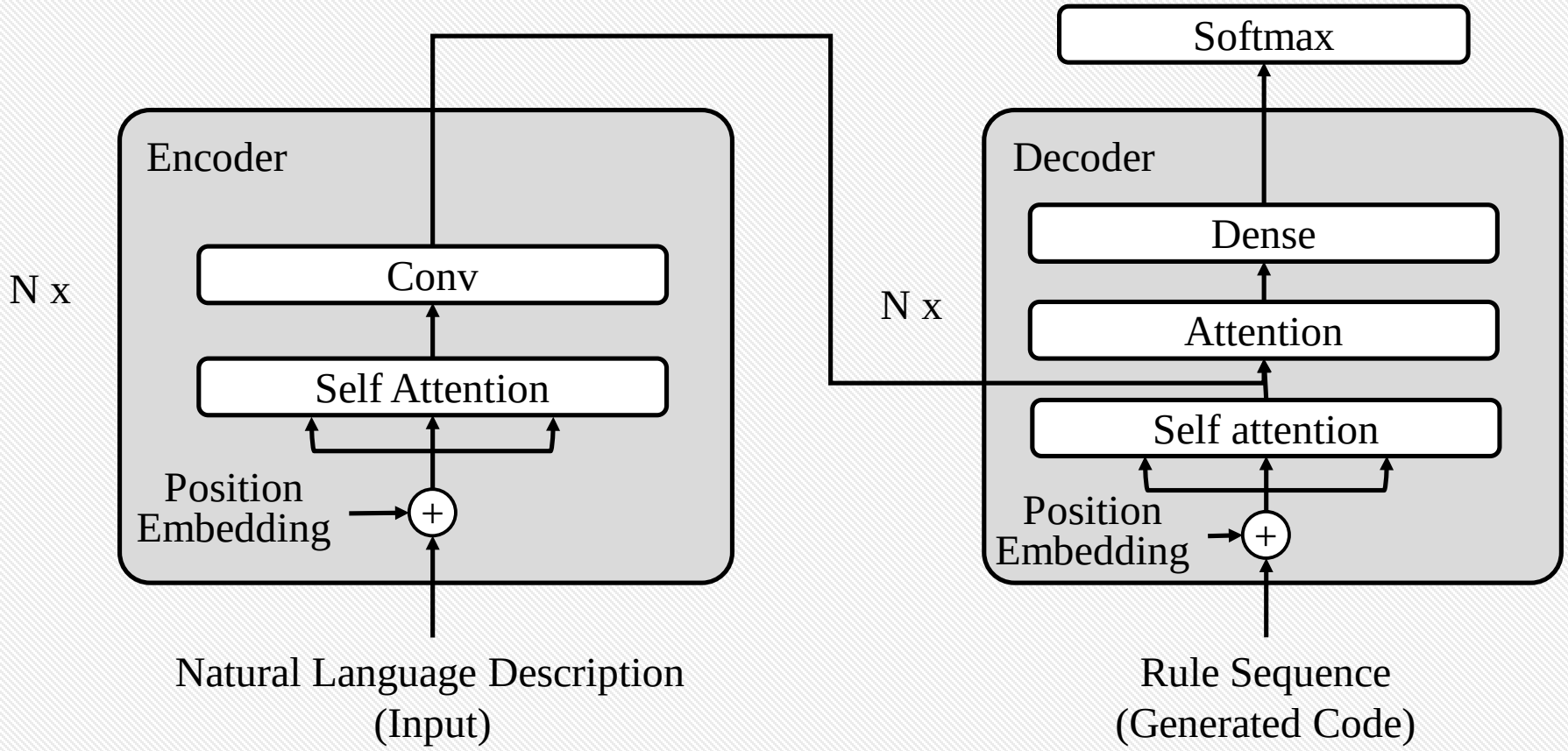
- We propose a novel neural architecture, TreeGen, for the code generation.
  - The second challenge is the representation of code structures.
    - A standard way of utilizing structural information is to combine the vector representations of a node and its structural neighbors (like Graph Neural Network) as the output of a structural convolution sub-layer.



    - However, a standard Transformer architecture does not have such structural convolution sub-layers, and it is not clear where to add them.

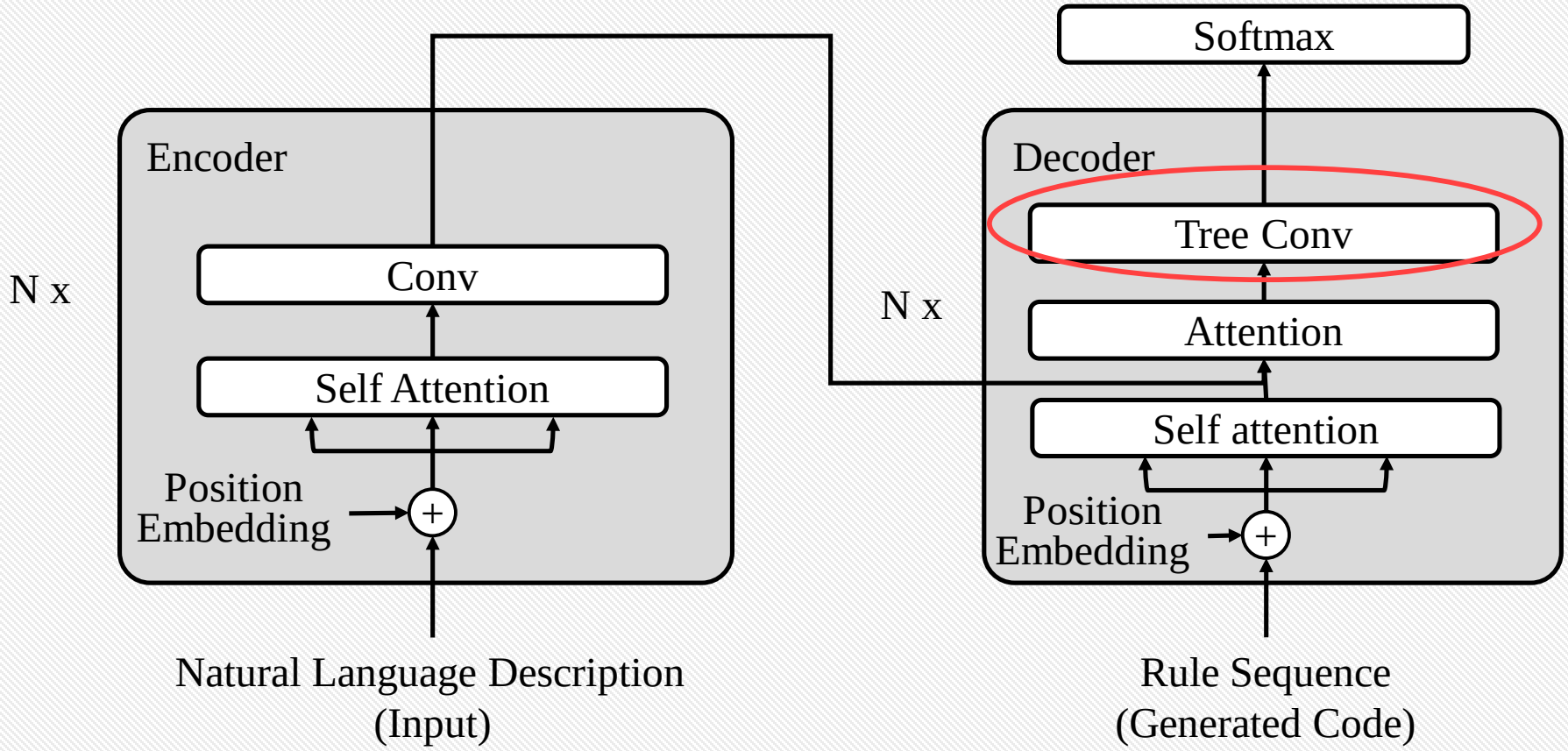
TreeGen

- We propose a novel neural architecture, TreeGen, for the code generation.
  - The second challenge is the representation of code structures.



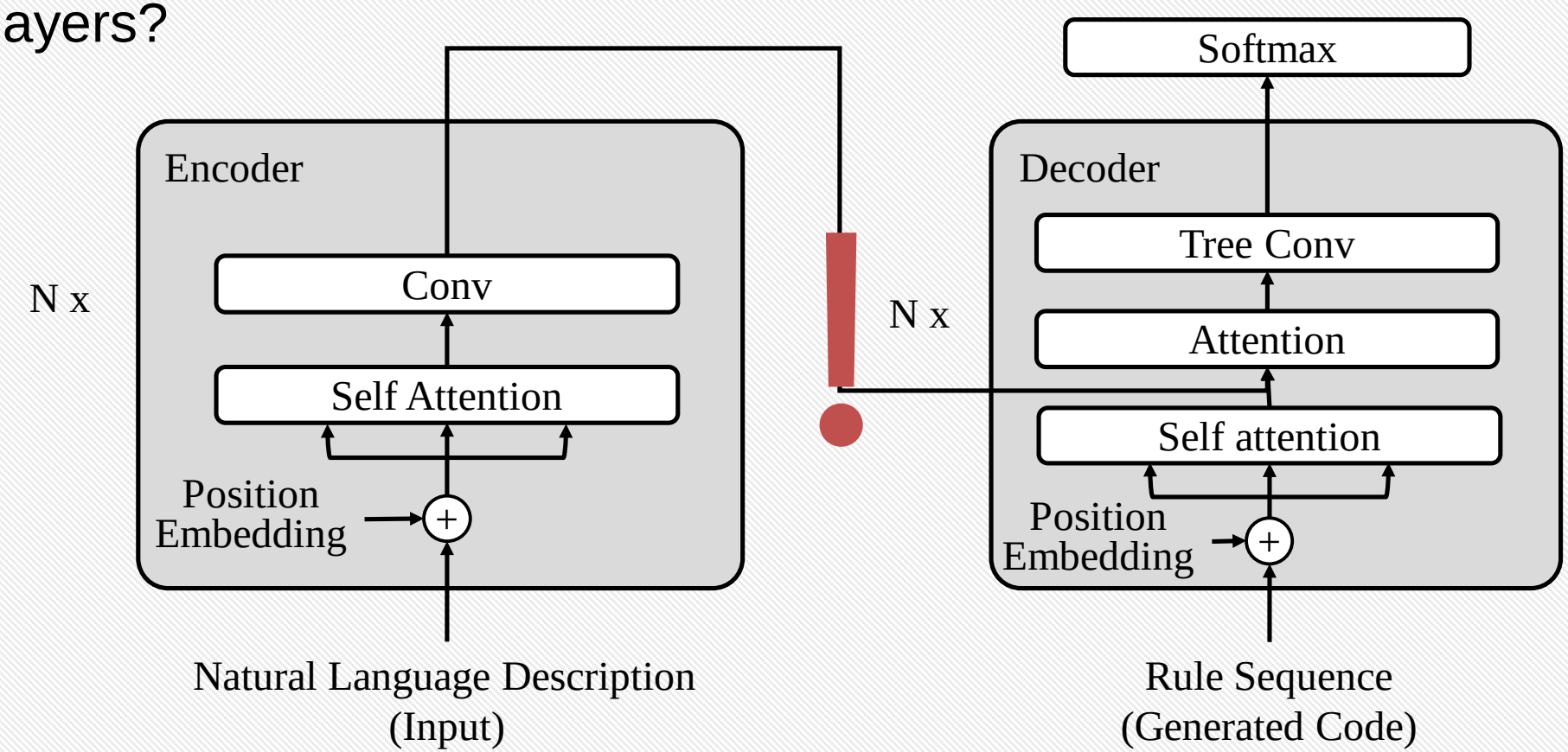
TreeGen

- We propose a novel neural architecture, TreeGen, for the code generation.
  - The second challenge is the representation of code structures.



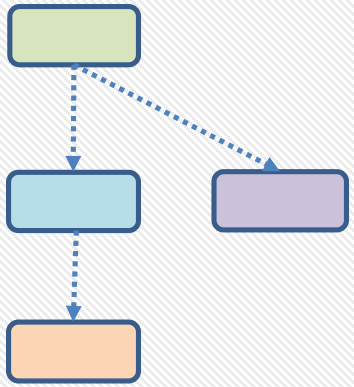
TreeGen

- We propose a novel neural architecture, TreeGen, for the code generation..
  - The second challenge is the representation of code structures.
    - To which transformer blocks should we add the structural convolution sublayers?



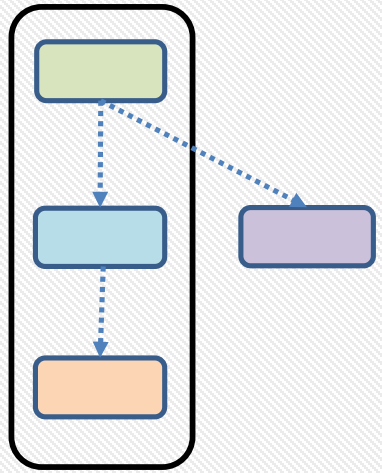
TreeGen

- We propose a novel neural architecture, TreeGen, for the code generation..
  - The second challenge is the representation of code structures.
    - As the vector representation of the nodes is processed by more blocks in the decoder of the Transformer, they gradually mix in more information from other nodes and lose their original information.



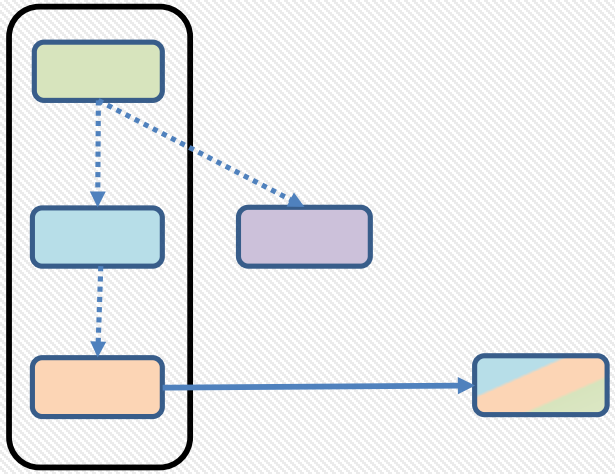
TreeGen

- We propose a novel neural architecture, TreeGen, for the code generation..
  - The second challenge is the representation of code structures.
    - As the vector representation of the nodes is processed by more blocks in the decoder of the Transformer, they gradually mix in more information from other nodes and lose their original information.



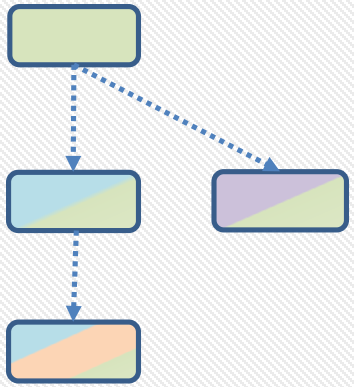
TreeGen

- We propose a novel neural architecture, TreeGen, for the code generation..
  - The second challenge is the representation of code structures.
    - As the vector representation of the nodes is processed by more blocks in the decoder of the Transformer, they gradually mix in more information from other nodes and lose their original information.



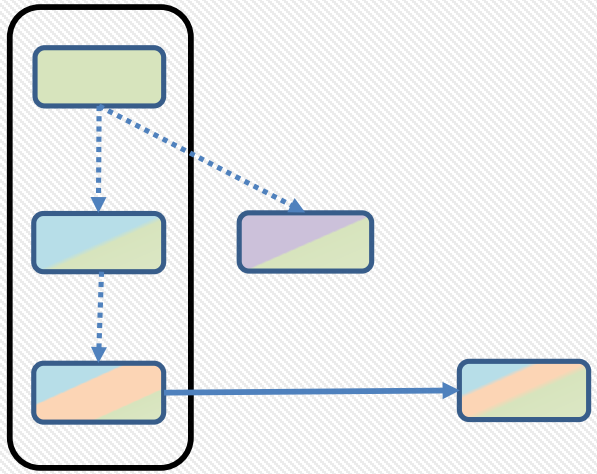
TreeGen

- We propose a novel neural architecture, TreeGen, for the code generation..
  - The second challenge is the representation of code structures.
    - As the vector representation of the nodes is processed by more blocks in the decoder of the Transformer, they gradually mix in more information from other nodes and lose their original information.



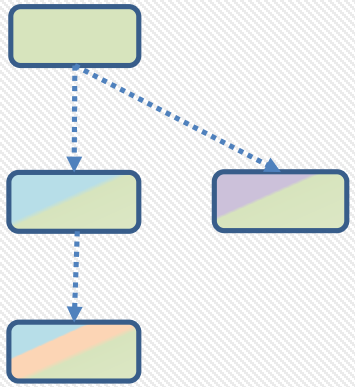
TreeGen

- We propose a novel neural architecture, TreeGen, for the code generation..
  - The second challenge is the representation of code structures.
    - As the vector representation of the nodes is processed by more blocks in the decoder of the Transformer, they gradually mix in more information from other nodes and lose their original information.



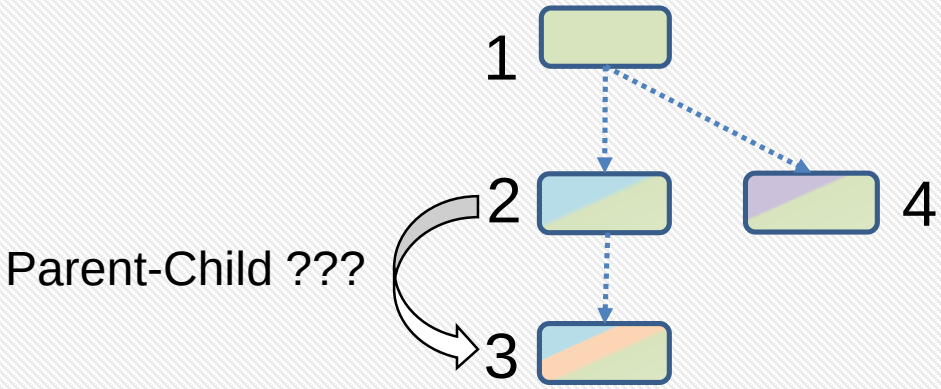
TreeGen

- We propose a novel neural architecture, TreeGen, for the code generation..
  - The second challenge is the representation of code structures.
    - As the vector representation of the nodes is processed by more blocks in the decoder of the Transformer, they gradually mix in more information from other nodes and lose their original information.



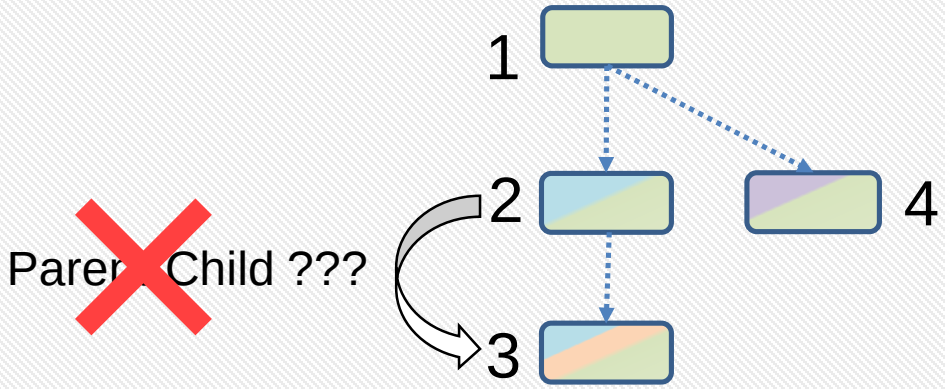
TreeGen

- We propose a novel neural architecture, TreeGen, for the code generation..
  - The second challenge is the representation of code structures.
    - As the vector representation of the nodes is processed by more blocks in the decoder of the Transformer, they gradually mix in more information from other nodes and lose their original information.



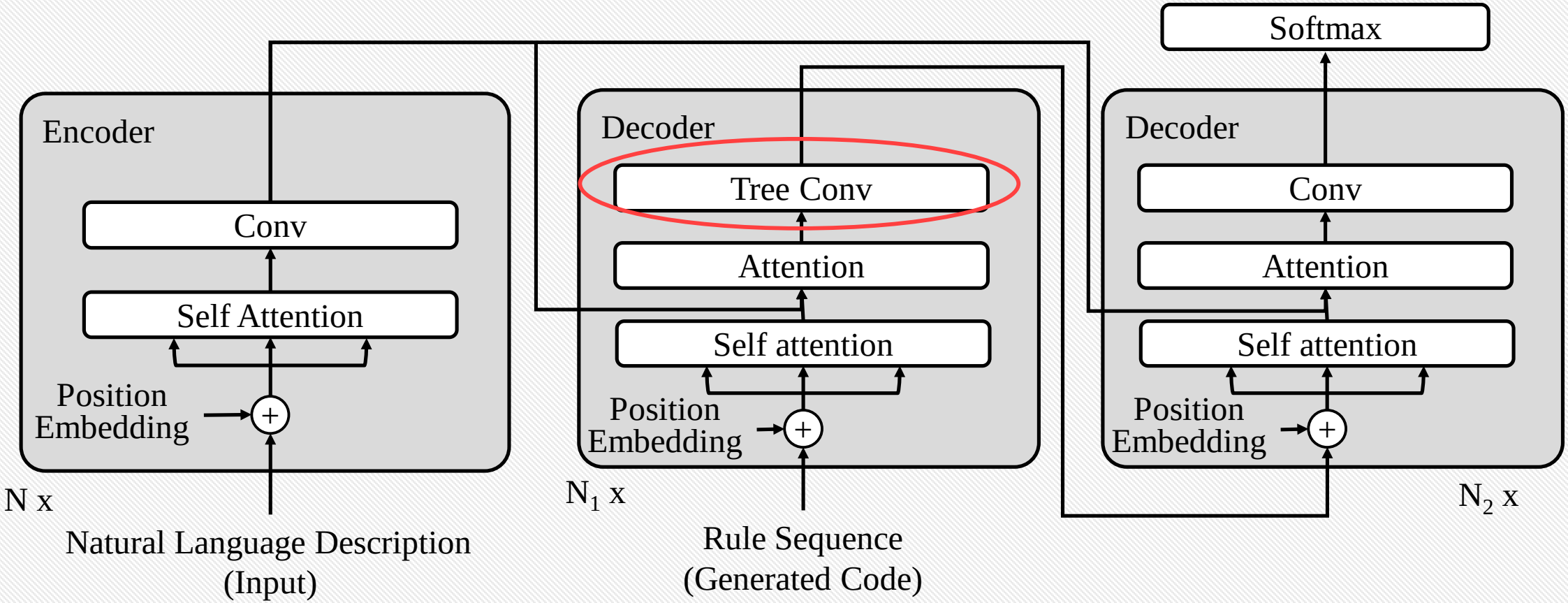
TreeGen

- We propose a novel neural architecture, TreeGen, for the code generation..
  - The second challenge is the representation of code structures.
    - As the vector representation of the nodes is processed by more blocks in the decoder of the Transformer, they gradually mix in more information from other nodes and lose their original information.



TreeGen

- We propose a novel neural architecture, TreeGen, for the code generation.
  - The second challenge is the representation of code structures.



PART TWO

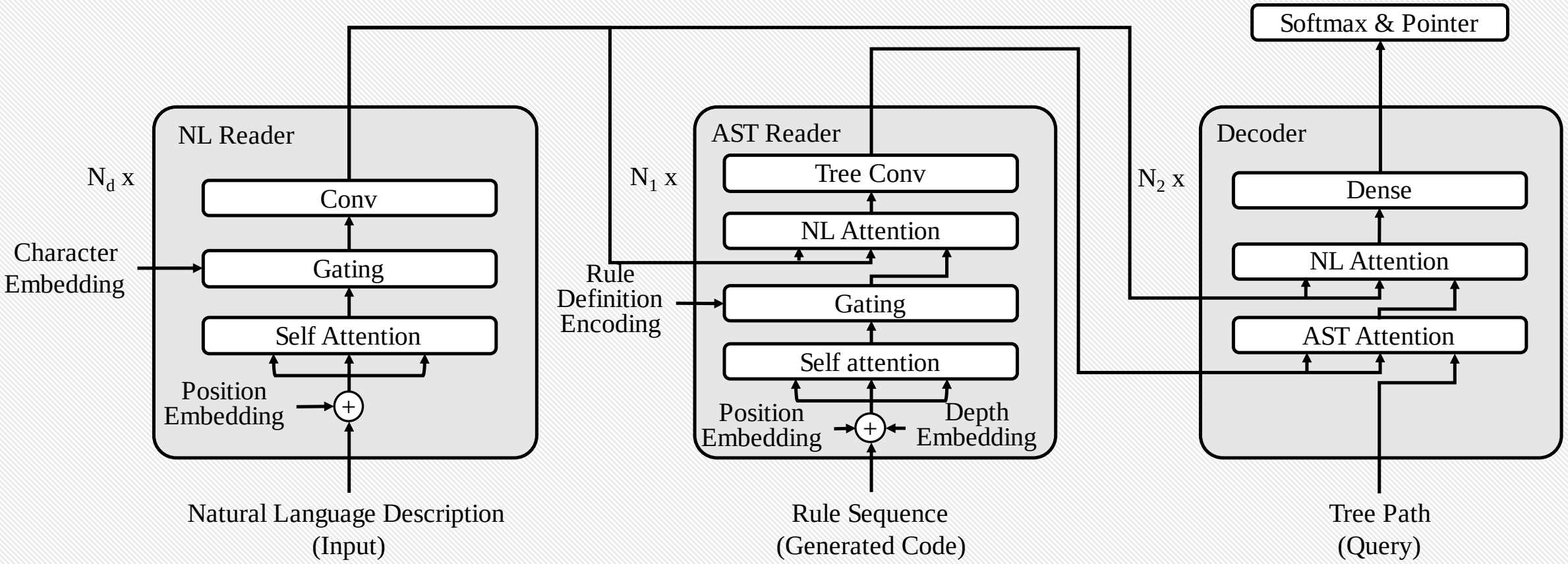
Approach

TreeGen

- The TreeGen architecture consists of three parts:
  - A natural language (NL) reader (encoder) encodes the text description;
  - An AST reader (the first several Transformer decoder blocks) encodes the previously generated partial code with the structural convolution sublayers;
  - A decoder (the rest Transformer decoder blocks) combines the query (the node to be expanded in AST) and the previous two encoders to predict the next grammar rule.

TreeGen

- TreeGen takes natural language description as input and outputs the target AST rule sequence.



PART THREE

Evaluation

Experiments

- We evaluated our approach on two types of benchmarks:
  - A Python code generation benchmark, HearthStone.
  - Two semantic parsing benchmarks, ATIS and GEO.

NAME: Darkscale Healer
ATK: 4
DEF: 5
COST: 5
DUR: -1
TYPE: Minion
PLAYER: Neutral
RACE: NIL
RARITY: Common
DESCRIPTION:
Battlecry: Restore 2 Health to all friendly characters.

```
class DarkscaleHealer(MinionCard):
    def __init__ (self) :
        super().__init__("Darkscale Healer", 5,
            CHARACTER_CLASS.ALL, CARD_RARITY.COMMON,
            battlecry = Battlecry(Heal(2),
                CharacterSelector()))

    def create_minion (self, player):
        return Minion(4, 5)
```

List airport in ci0

Lambda $0 e (and (airport $0))
(loc:t $0 ci0)

Experiment: HearthStone

- We first evaluated our approach on the HearthStone benchmark (Ling et al. 2016).
  - The benchmark contains Python code that implements 665 different cards of HearthStone. (train: 533, dev: 66, test: 66)

Statistics	HS
# Train	533
# Dev	66
# Test	66
Avg. tokens in description	35.0
Max. tokens in description	76.0
Avg. tokens in code	83.2
Max. tokens in code	403

Experiment: HearthStone

- Metrics
  - StrAcc: exactly the same token sequence as the ground truth.
  - BLEU: the similarity between the generated code and the reference code at the token level.
  - Acc+: is evaluated manually and allows variable renaming on top of StrAcc.

Experiment: HearthStone

- We first evaluated our approach on the HearthStone benchmark (Ling et al. 2016).
- Preprocessing:
  - **Plain preprocessing** treats the whole description as plain text.
  - **Structural preprocessing** treats the descriptions as semi-structural one.

Experiment: HearthStone

- Plain preprocessing:

  19.6% → 25.8%

- Structural preprocessing:

  27.3% → 31.8%

	Model	StrAcc	Acc+	BLEU
Plain	LPN (Ling et al. 2016)	6.1	–	67.1
	SEQ2TREE (Dong and Lapata 2016)	1.5	–	53.4
	YN17 (Yin and Neubig 2017)	16.2	~18.2	75.8
	ASN (Rabinovich, Stern, and Klein 2017)	18.2	–	77.6
	ReCode (Hayati et al. 2018)	19.6	–	78.4
	TreeGen-A	**25.8**	**25.8**	**79.3**
Structural	ASN+SUPATT (Rabinovich, Stern, and Klein 2017)	22.7	–	79.2
	SZM19 (Sun et al. 2019)	27.3	30.3	79.6
	TreeGen-B	**31.8**	**33.3**	80.8
	Location of Structural Convolutional Sub-layer			
	$N_1 = 10, N_2 = 0$	25.8	27.3	80.4
	$N_1 = 10(7), N_2 = 0$	27.3	28.8	78.5
	$N_1 = 10(8), N_2 = 0$	25.8	28.8	78.5
	$N_1 = 0, N_2 = 10$	21.2	22.7	79.6
	Ablation test			
	Baseline: Transformer	10.6 ($p = 0.015$)	12.1	68.0
	- Tree Convolution	27.3 ($p = 0.015$)	27.3	80.9
	- Rule Definition Encoding	27.3 ($p < 0.001$)	28.8	**81.8**
	- Char Embedding	15.2 ($p < 0.001$)	18.2	72.9
	- Self-Attention	28.8 ($p < 0.001$)	28.8	81.0

Experiment: HearthStone

- Time Efficiency in Training
  - We further evaluated the complexity of our model on the HearthStone.
  - An Epoch:
    - 18s for TreeGen.
    - 180s for the CNN (Sun et al. 2019).
    - 49s for the RNN (Yin and Neubig 2017).

Experiment II: Semantic Parsing

- We further evaluated our approach on the semantic parsing tasks.
  - Our experiment was conducted on two semantic parsing datasets, ATIS and GEO.

		Exp II	
Statistics	**HS**	**ATIS**	**GEO**
# Train	533	4,434	600
# Dev	66	491	-
# Test	66	448	280
Avg. tokens in description	35.0	10.6	7.4
Max. tokens in description	76.0	48	23
Avg. tokens in code	83.2	33.9	28.3
Max. tokens in code	403	113	144

Experiment II: Semantic Parsing

- Metrics
  - We follow the evaluation of the previous approaches (Dong and Lapata 2016) and use accuracy as the metric.
    - The tree exact match was considered to avoid spurious errors.

Experiment II: Semantic Parsing

- We achieved the best accuracy among neural network-based approaches on ATIS (89.1%) and GEO (89.6%).

	Method	ATIS	GEO
Traditional	ZC07 (Zettlemoyer and Collins 2007)	84.6	86.1
	FUBL (Kwiatkowski et al. 2011)	82.8	88.6
	KCAZ13 (Kwiatkowski et al. 2013)	-	89.0
	WKZ14 (Wang, Kwiatkowski, and Zettlemoyer 2014)	**91.3**	**90.4**
Neural Networks	SEQ2SEQ (Dong and Lapata 2016)	84.2	84.6
	SEQ2TREE (Dong and Lapata 2016)	84.6	87.1
	ASN (Rabinovich, Stern, and Klein 2017)	85.3	85.7
	ASN+SUPATT (Rabinovich, Stern, and Klein 2017)	85.9	87.1
	COARSE2FINE (Dong and Lapata 2018)	87.7	88.2
	TRANX (Yin and Neubig 2018)	86.2	88.2
	Seq2Act (Chen, Sun, and Han 2018)	85.5	88.9
	Graph2Seq (Xu et al. 2018)	85.9	88.1
	SZM19 (Sun et al. 2019)	85.0	-
	TreeGen	**89.1**	**89.6**

Conclusion

- We propose TreeGen for code generation.
  - Transformers to alleviate the long-dependency problem.
  - An AST reader to combine the grammar rules and the AST structure.

- The experimental results show that our model significantly outperforms existing approaches.

Thanks

TreeGen: A Tree-Based Transformer Architecture for Code Generation

Zeyu Sun, Qihao Zhu, Yingfei Xiong, Yican Sun, Lili Mou, Lu Zhang

szy_@pku.edu.cn

https://github.com/zysszy/AAAI20-ORAL-POSTER

https://github.com/zysszy/TreeGen