

DUETCS: Code Style Transfer through Generation and Retrieval

発表者：ダニエル

Binger Chen
TU Berlin
Berlin, Germany
chen@tu-berlin.de

Ziawasch Abedjan
Leibniz Universität Hannover & L3S Research Center
Hannover, Germany
abedjan@lbs.uni-hannover.de

背景

- | コードスタイルとは、書式設定、命名、モジュール化など、コードの構成方法に関するルールを指します
- | コードスタイルはコードの理解に直接影響を及ぼし、開発と保守の両方にとって重要です
- | リンターやフォーマッタなどの既存のソリューションは、特定の定義済みルールに限定されており、命名や構造化などのより広範なスタイル機能はカバーしていません

目的

- | 事前に定義されたスタイルルールを必要とせずにコードスタイルを自動的に転送できる教師なしメソッドを開発します。
- | ビッグコード データベースとメタ学習技術が、任意のコードスタイルの抽出と適用にどのように役立つかを探ります。
- | コード内のスタイルとコンテンツの機能を区別する

WHAT IS DUETCS?

- DUETCS is a system designed to automatically change the **coding style** of a program while keeping its functionality intact.
- It takes in two inputs:
 - A **program** that you want to change the style of (its content is kept the same).
 - A **set of code examples** written in the desired style you want to apply.
- It works in **two modes**:
 - Retrieval mode**: It searches the Big Code database to find a piece of code that matches both the content of your original program and the new style.
 - Generation mode**: It creates a new piece of code using an **LSTM-based model** that combines your program's content with the new style features.

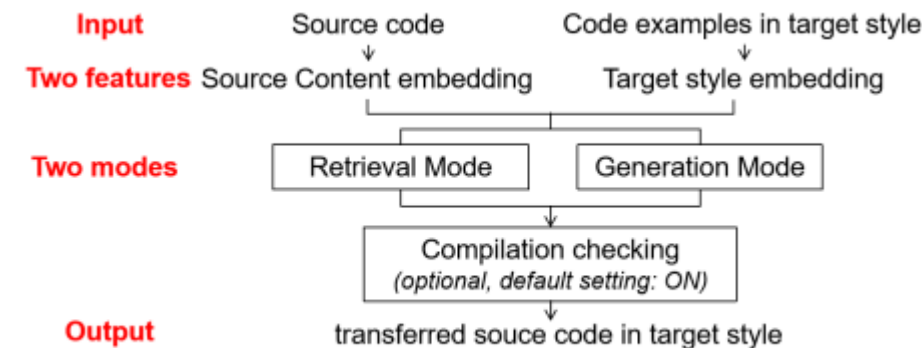


Fig. 1: DUETCS Overview

調査方法/ 研究デザイン

Character-Level Encoding:

- To handle names, a **character-level embedding** is used for variable names. Characters are labeled based on their position within the word:
 - for the **beginning** of the word.
 - <M> for characters in the **middle** of the word.
 - <E> for the **end** of the word.

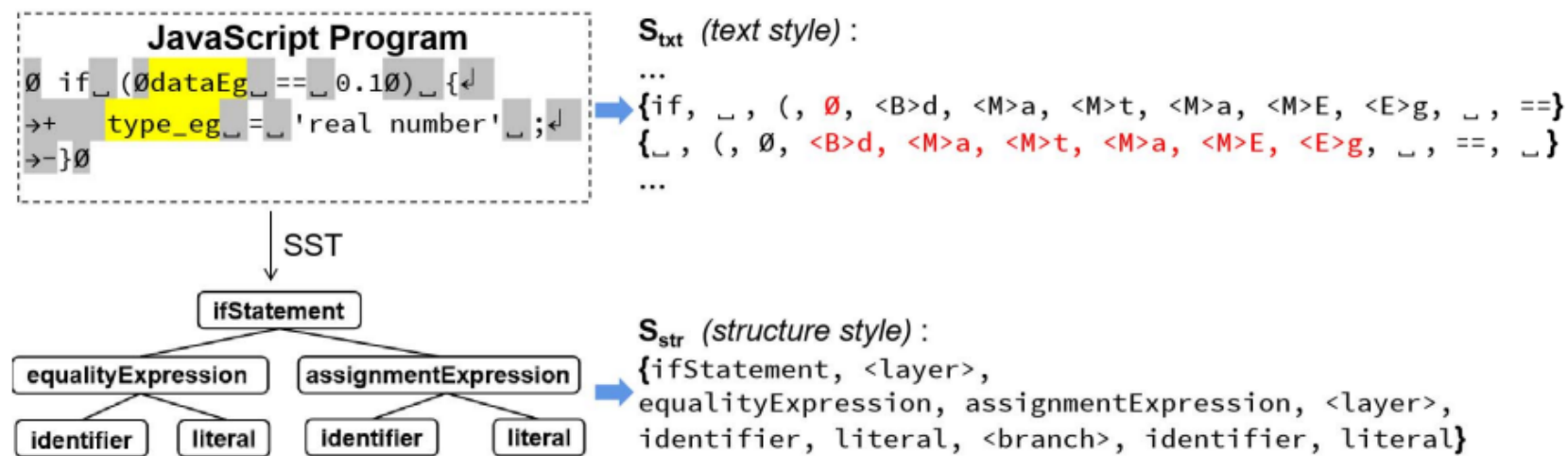


Fig. 2: Vectorization of text and structure style features

調査方法/ 研究デザイン

Structure Style Representation:

- The **Simplified Syntax Tree (SST)** represents the **structure** of the code while eliminating redundant nodes.
- The tree is traversed using **breadth-first search (BFS)**, and **special tokens** are added to indicate new layers or branches:
 - <layer> to indicate the **start of a new layer**.
 - <branch> to indicate the **start of a new branch**.

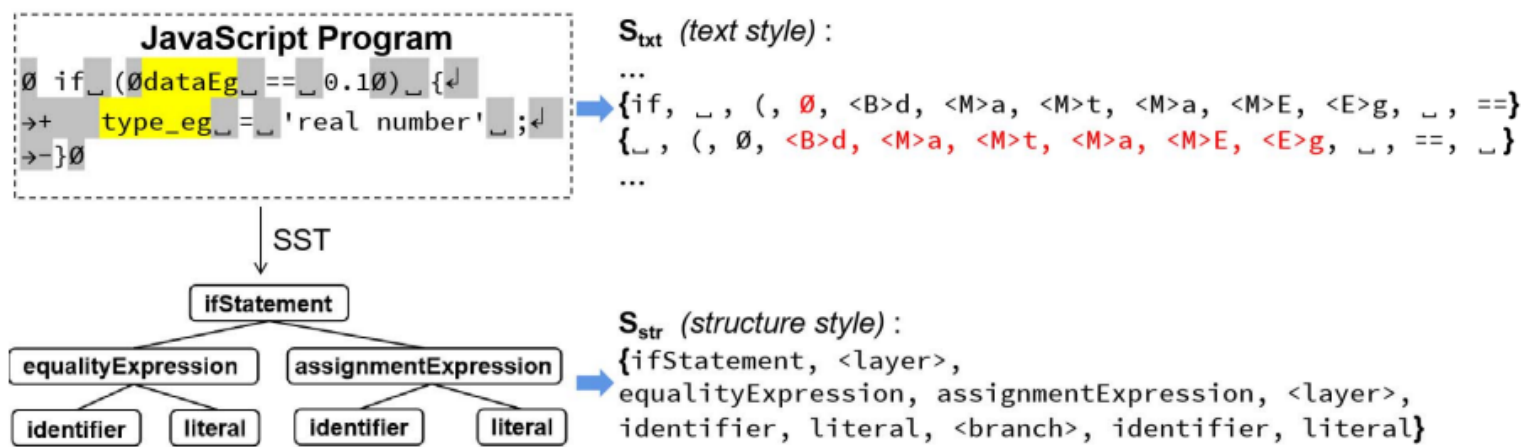


Fig. 2: Vectorization of text and structure style features

調査方法/ 研究デザイン

- Conducted extensive experiments to evaluate DUETCS and its variations for different :-
 - languages (C++, JavaScript, Java),
 - tasks (code style transfer, formatting, method name prediction)
 - style aspects (formatting, naming, structure style).
- Two types of datasets:
 - Training dataset: Public Git Archive (PGA) database, covering over 260,000 bookmarked GitHub repositories.
 - Testing datasets: Codeforces, Jsformat, and Java-small datasets.
 - Fetch all projects written in three popular languages: JavaScript, Java, and C++
 - Split the files into functions as training samples and remove duplicates
 - Obtain a 239GB dataset with 13,652 projects and around 2,490 functions per project

調查結果

DUETCS on the JSformat dataset.

The evaluation focuses on three style aspects:

- formatting
- naming
- Structure

STYLE-ANALYZER can only learn **formatting style**

DUETCS can recover around 90% of the original style for both **formatting** and **structure** style

Metrics

- Precision (Prec): percentage of correct predictions out of all predictions, indicating performance on inconsistent style correction.
- Prediction Rate (PredR): percentage of actual predicted tokens out of all formatting tokens that need to be predicted, indicating performance on inconsistent style detection.

TABLE III: Performance on JSformat&Java-small dataset

	Method	Formatting		Naming		Structure		(train/ test)
		Prec	PredR	Prec	PredR	Prec	PredR	
JSformat	DUETCS	89.2	71.9	48.7		91.9	87.3	
	DUETCS (G)	91.2	58.7	33.5	100	96.6	76.5	1,269/
	DUETCS (R)	97.2	43.6	21.1		90.6	63.2	6028
	DUETCS (Ideal)	93.1	75.4	54.6		94.7	91.2	
	STYLE-ANALYZER	92.5	91.6	-		-		139,615/ 663,171 token seq.
Java-small	DUETCS	93.1	62.0	53.9		81.2	93.7	
	DUETCS (G)	92.1	53.6	39.4	100	80.9	78.8	66,512/
	DUETCS (R)	89.1	31.5	20.3		97.8	45.7	721,280
	DUETCS (Ideal)	94.5	68.3	59.7		86.3	94.2	
	ConVAttention			50.3				
	Path+CRFs			8.4				
	C2V	-		18.5	100	-		665,115/ 56,165
	C2S			50.7				
	CTrans			54.9				

調查結果

Both generation and retrieval modes can achieve **good precision**, but their ability to detect style inconsistency is **worse** than the full system

Combining both modes is a proper strategy

The **prediction rate (PredR)** of DUETCS is about 20% lower than STYLE-ANALYZER

For **naming style**, the PredR is always 100% because the location of method name is fixed

TABLE III: Performance on JSformat&Java-small dataset

	Method	Formatting		Naming		Structure		(train/ test)
		Prec	PredR	Prec	PredR	Prec	PredR	
JSformat	DUETCS	89.2	71.9	48.7		91.9	87.3	
	DUETCS (G)	91.2	58.7	33.5	100	96.6	76.5	1,269/
	DUETCS (R)	97.2	43.6	21.1		90.6	63.2	6028
	DUETCS (Ideal)	93.1	75.4	54.6		94.7	91.2	
	STYLE-ANALYZER	92.5	91.6	-		-		139,615/ 663,171 token seq.
Java-small	DUETCS	93.1	62.0	53.9		81.2	93.7	
	DUETCS (G)	92.1	53.6	39.4	100	80.9	78.8	66,512/
	DUETCS (R)	89.1	31.5	20.3		97.8	45.7	721,280
	DUETCS (Ideal)	94.5	68.3	59.7		86.3	94.2	
	Con VAttention			50.3				
	Path+CRFs			8.4				
	C2V	-		18.5	100	-		665,115/ 56,165
	C2S			50.7				
	CTrans			54.9				

調査結果

Input	Ground truth	Output of DUETCS
<pre> #include<bits/stdc++.h> using namespace std; int main(){ int t; cin>>t; while(t--){ int n; cin>>n; bool x,y=1; int a; for(int i=0;i<n;i++){ cin>>a; x=1; for(int j=2;j<=i+2;j++){ if(a%j!=0){ x=0; break; } } if(x) y=0; } if(y) cout<<"YES"<<endl; else cout<<"NO"<<endl; } return 0; } </pre>	<pre> #include<bits/stdc++.h> using namespace std; int main() { int t; cin >> t; while (t--) { int n; cin >> n; bool ok = true; for (int i = 1; i <= n; i++) { int x; cin >> x; bool found = false; for (int j = i + 1; j >= 2; j--) { if (x % j) { found = true; break; } } ok &= found; } if (ok) { cout << "YES\n"; } else { cout << "NO\n"; } return 0; } } </pre>	<pre> #include<bits/stdc++.h> using namespace std; int main() { int t; cin >> t; while (t--) { int n; cin >> n; bool ok, count = true; for (int i = 1; i <= n; i++) { int x; cin >> x; for (int j=1; j<=i+2; j++) { if (x % j) { count = false; break; } } If (count) { ok = false; } } if (ok) { cout << "YES"<<endl; } else { cout << "NO"<<endl; } return 0; } } </pre>

Fig. 6: Example of code style transfer and failed cases (arrows show the style differences between source and target, blue color shows that output captured the desired style and orange shows what was missed by the output)

考察

- DUETCS は、コード生成と取得を活用してコードをスタイル化する新しいアプローチです
- DUETCS は、既存のベースラインと比較して、より価値のある情報をユーザーに提供できます
- 完璧な転送コードを生成できるわけではないが、出力には、ユーザーのスタイル化を支援するのに十分なスタイル情報が取り込まれていることが結果からわかります