コードクローン検出の後処理としての統一的ラベリング手法

肥後研究室 M2 清水ささら

Department of Computer Science, Graduate School of Information Science and Technology,
The University of Osaka. Software Engineering Laboratory. - https://sel.ist.osaka-u.ac.jp

コードクローン

ソースコード中の一致，類似したコード片[1]

ソースコードの保守性を下げる要因となる

類似の度合いにより4つに分類される[2]

Type1：コメント，空白文字の違いを除いて一致する

Type2：リテラル，型，識別子の違いを除いて一致する

Type3：文の変更，挿入，削除を除いて一致する

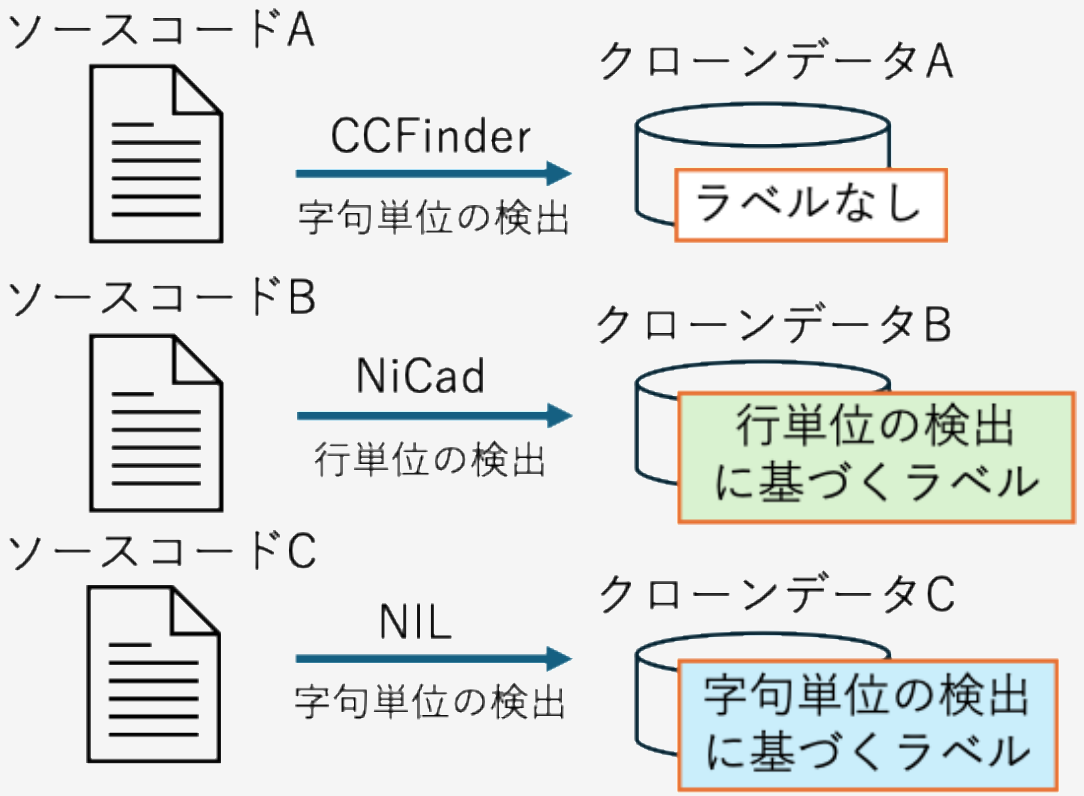
Type4：構文が異なるが，同じ機能を持つ

[1]井上克郎, 神谷年洋, 楠本真二. コードクローン検出法. コンピュータソフトウェア, Vol. 18, No. 5, pp. 529–536, 2001.

[2] C. Roy and J. Cordy. A Survey on Software Clone Detection Research. School of Computing TR No.2007–541, 2007.

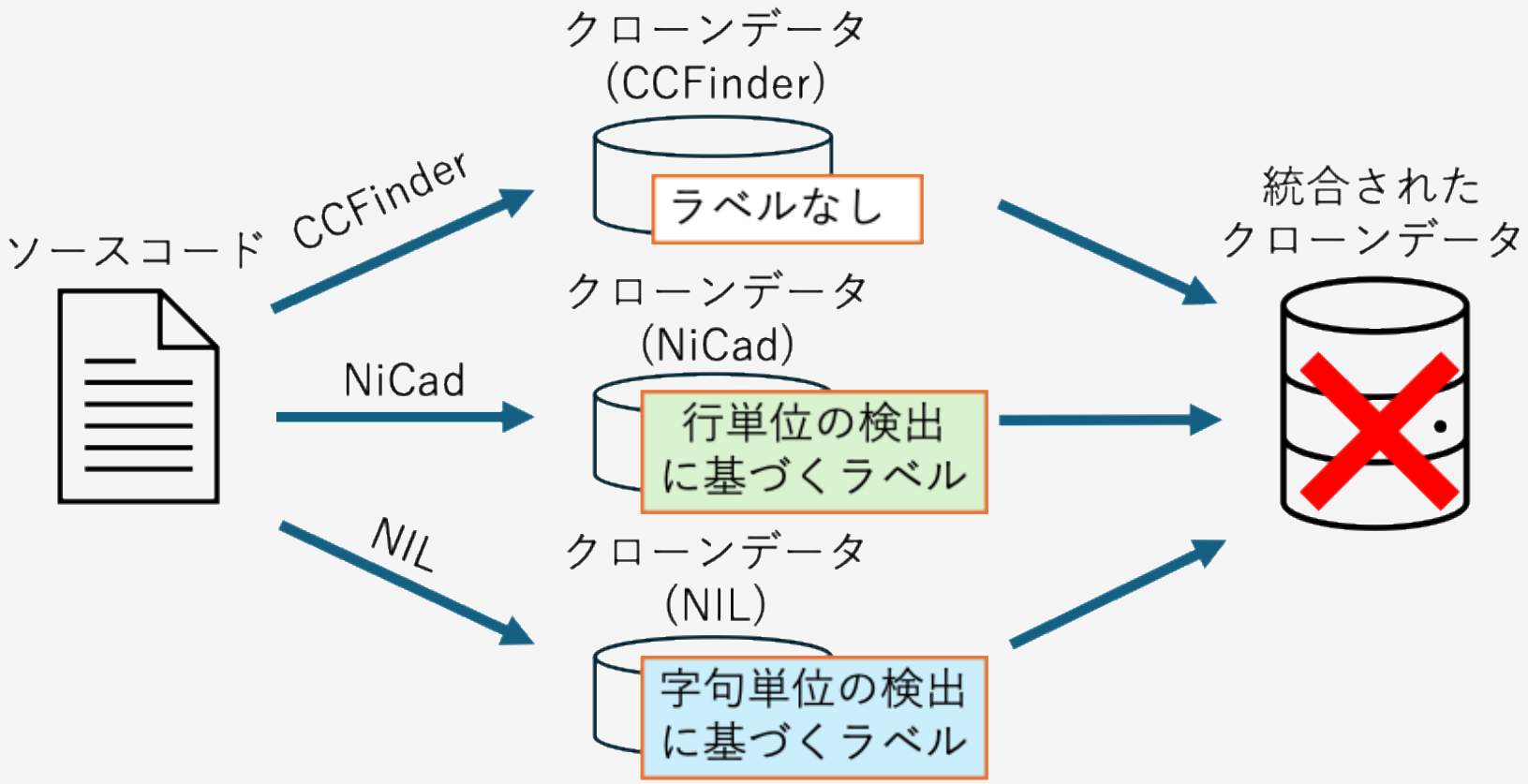
コードクローン検出ツール

- ソースコードに対して単一のクローン検出ツールが用いられる
- 単一のツールでは検出できるクローンに偏りがある



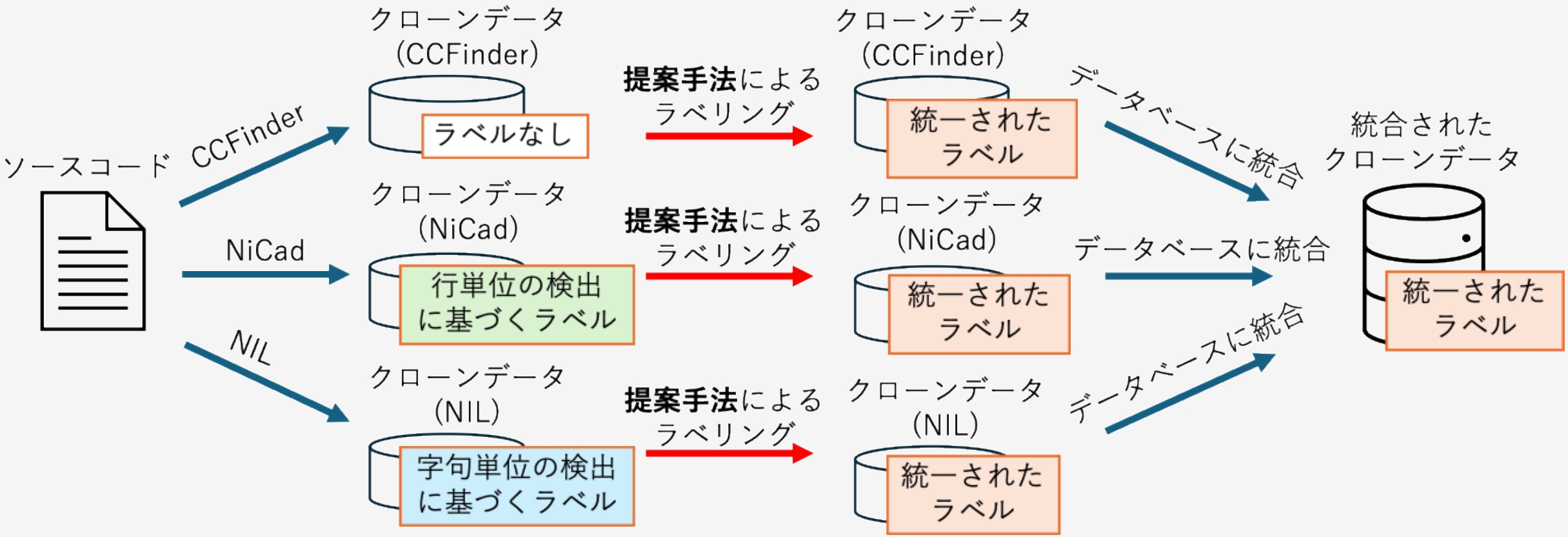
課題

- ラベリングが統一されていないため一つのデータベースに統合ができない



提案手法による解決策

- クローン検出後の統一的なラベリングを行う



研究目的と手段

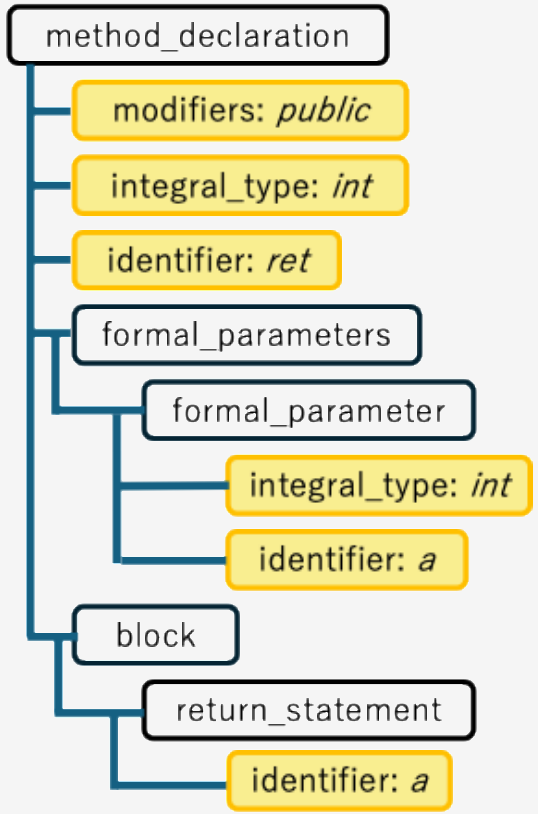
目的：コードクローン検出後の統一的なラベリング

提案手法：抽象構文木(AST)に基づくラベリング

抽象構文木(AST)

ソースコードの構文構造を木構造で表現したデータ構造

```
public int ret(int a) {
    return a
}
```



葉ノード，斜体は値

提案手法：ASTベースのラベリング

提案手法では以下のように分類を行った

入力：クローンペアとして検出された2つのコード片

出力：分類結果

- **Type1**：ASTが同一
- **Type2**：葉ノードを除いたASTが同一
- **Type3**：単文以下のノードを除いたASTが同一
- **Type4**：上記以外

Type1：ASTが同一

ASTの形状と値が同一

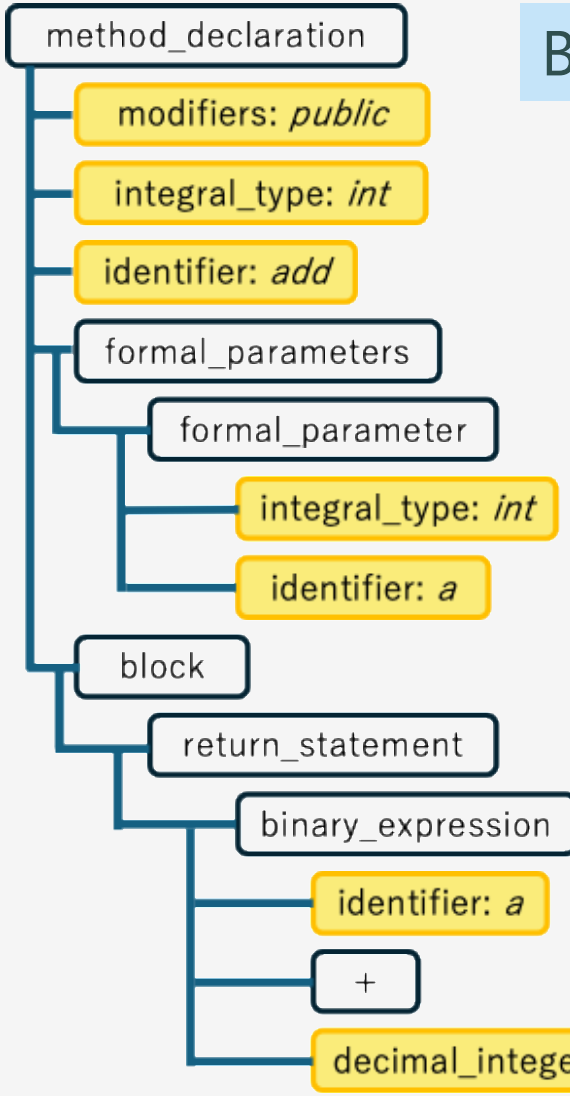
コードA

```
public int add(int a) {
    return a + 10;
}
```

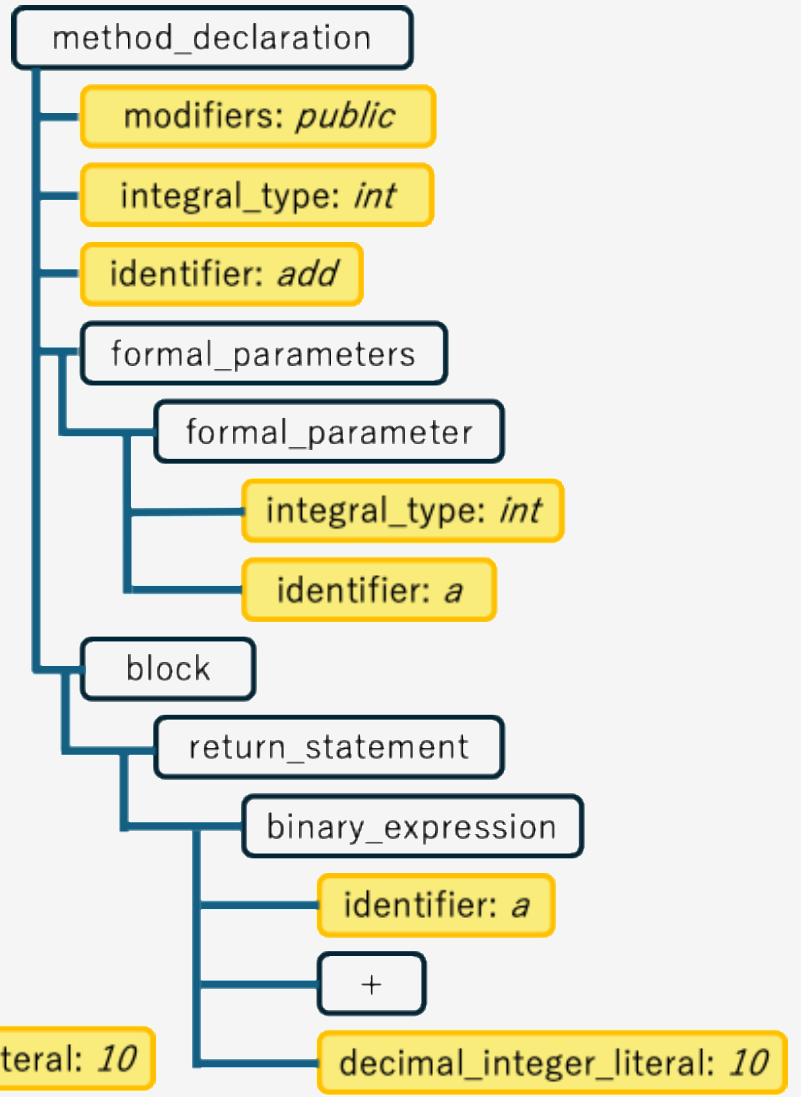
コードB

```
public int add(int a) {
    return a + 10;
}
```

A



B



Type2：葉ノードを除いたASTが同一

葉ノードを除いた
ASTの形状が同一

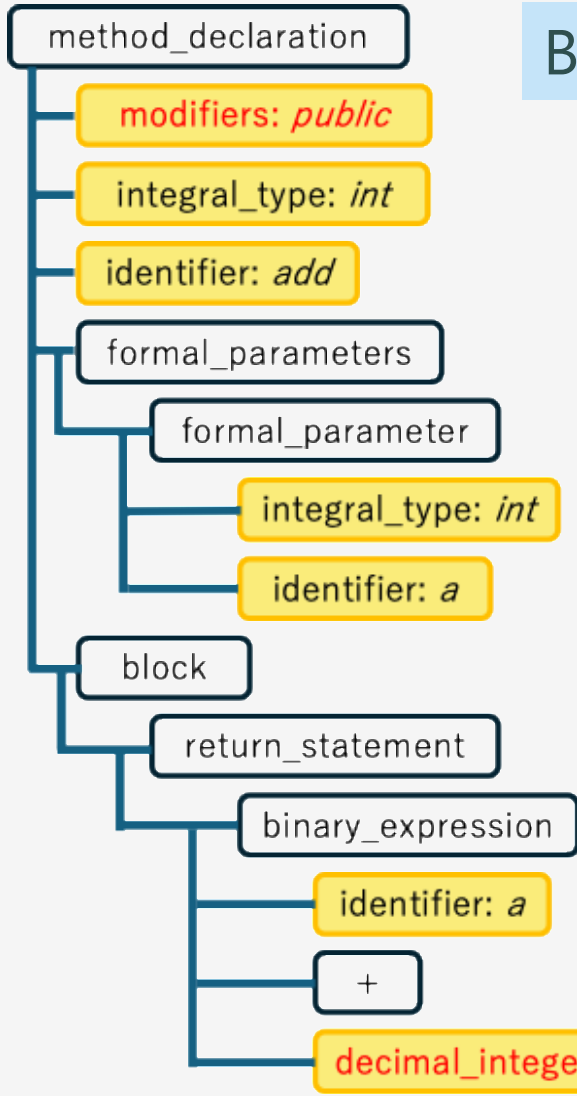
コードA

```
public int add(int a) {
    return a + 10;
}
```

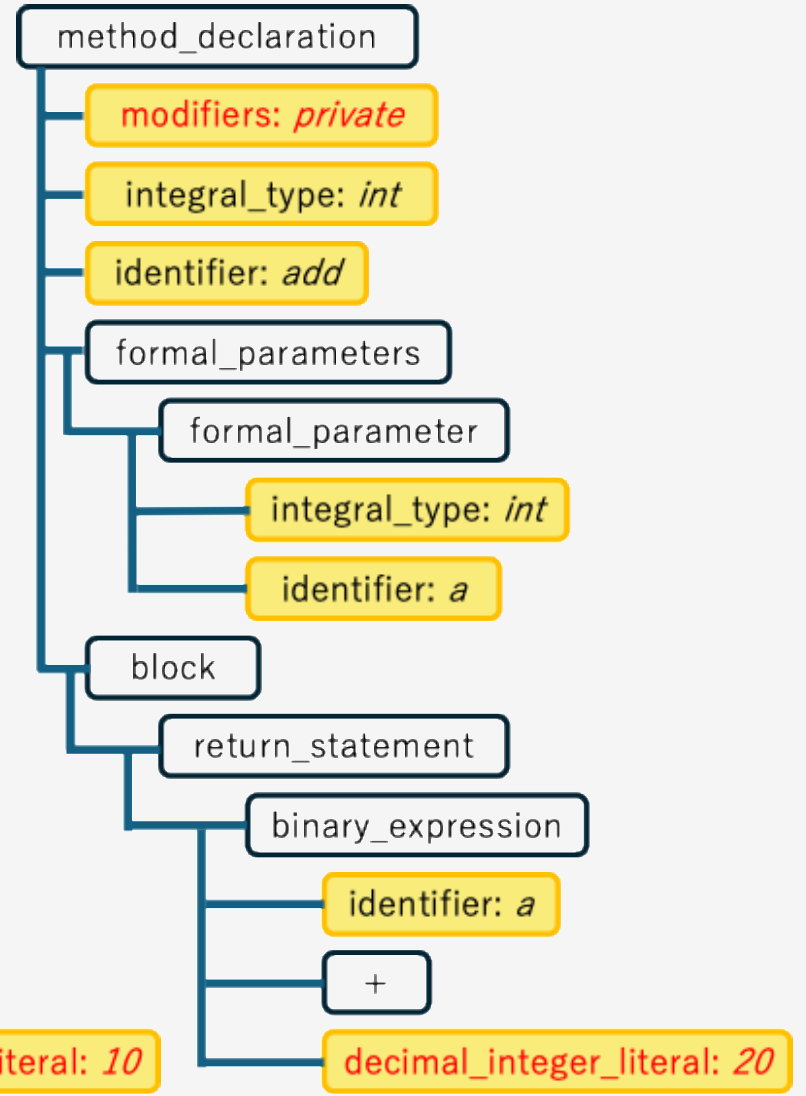
コードB

```
private int add(int a) {
    return a + 20;
}
```

A



B



Type2：葉ノードを除いたASTが同一

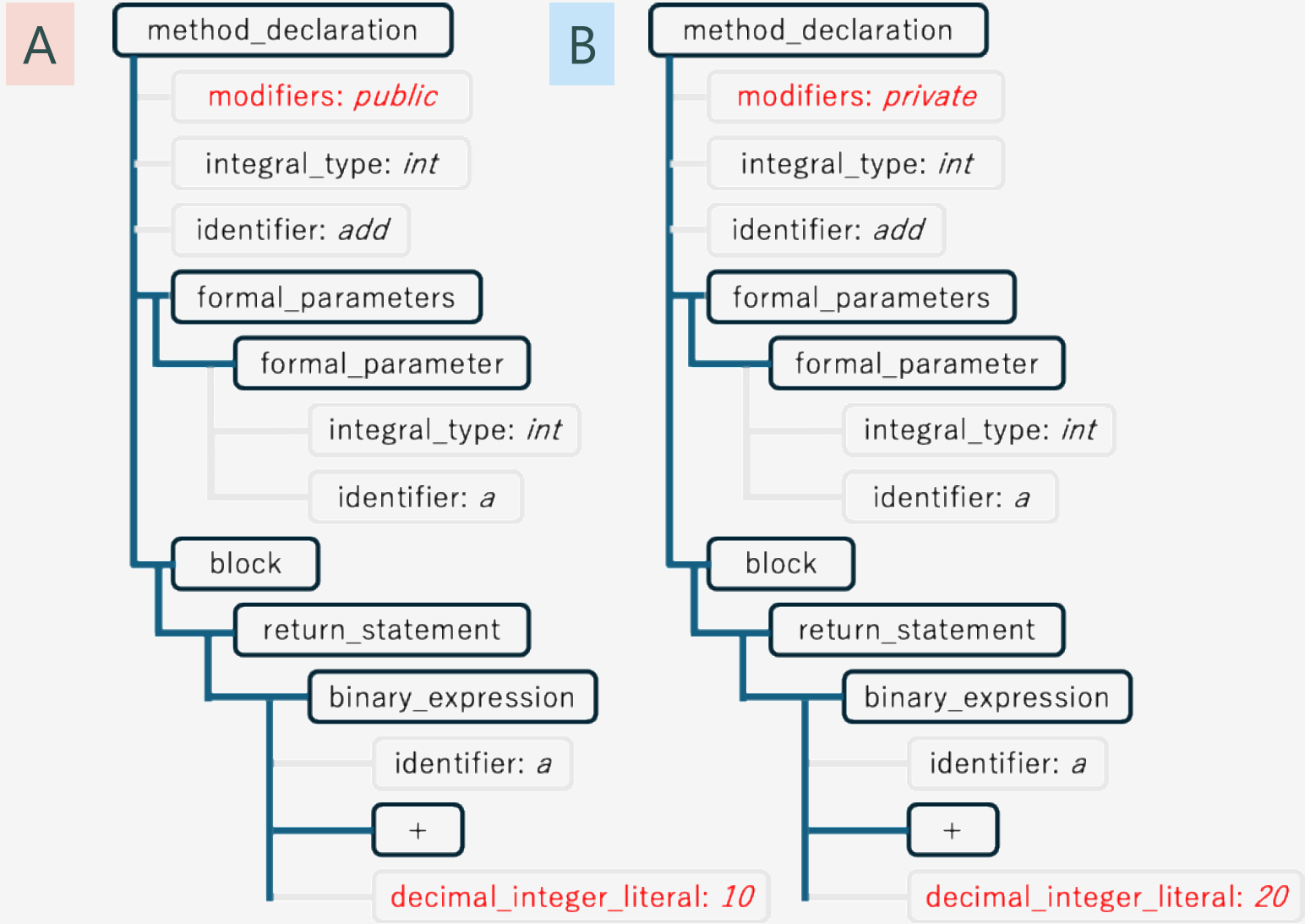
葉ノードを除いた
ASTの形状が同一

コードA

```
public int add(int a) {
    return a + 10;
}
```

コードB

```
private int add(int a) {
    return a + 20;
}
```



Type3：単文以下のノードを除いたASTが同一

Javaの構文定義では以下の文が単文として定義される

単文	例
式文	`x = a + b;` `System.out.println(s);`
変数宣言文	`int count = 0;`
return文	`return result;`
throws文	`throw new IllegalArgumentException();`
break文	`break;`
continue文	`continue;`
yield文	`yield 100;`
assert文	`assert list != null;`

Type3に分類される例

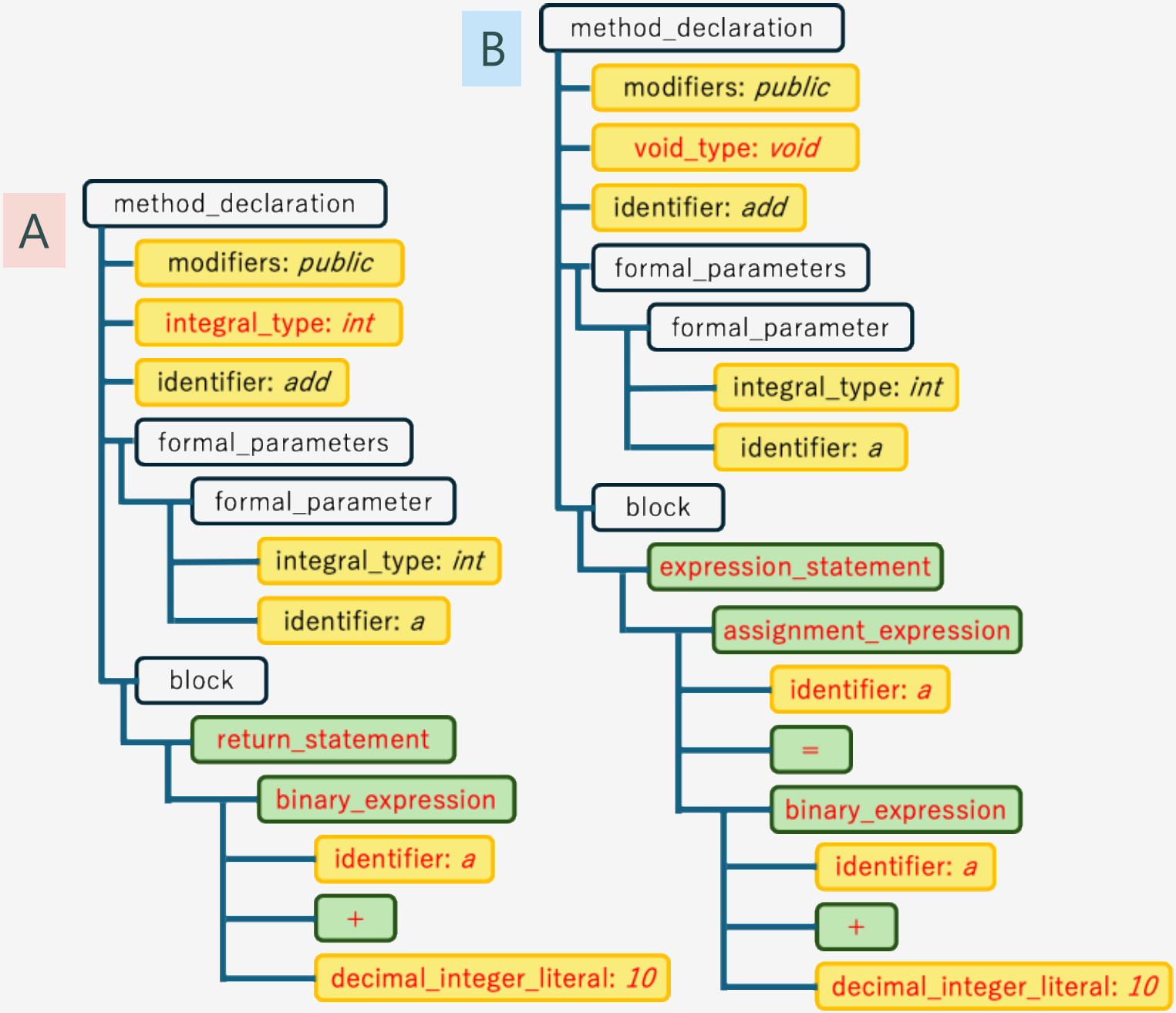
葉ノード，単文以下の
ノードを除いたASTが同一

コードA

```
public int add(int a) {
    return a + 10;
}
```

コードB

```
public void add(int a) {
    a = a + 10;
}
```



Type3に分類される例

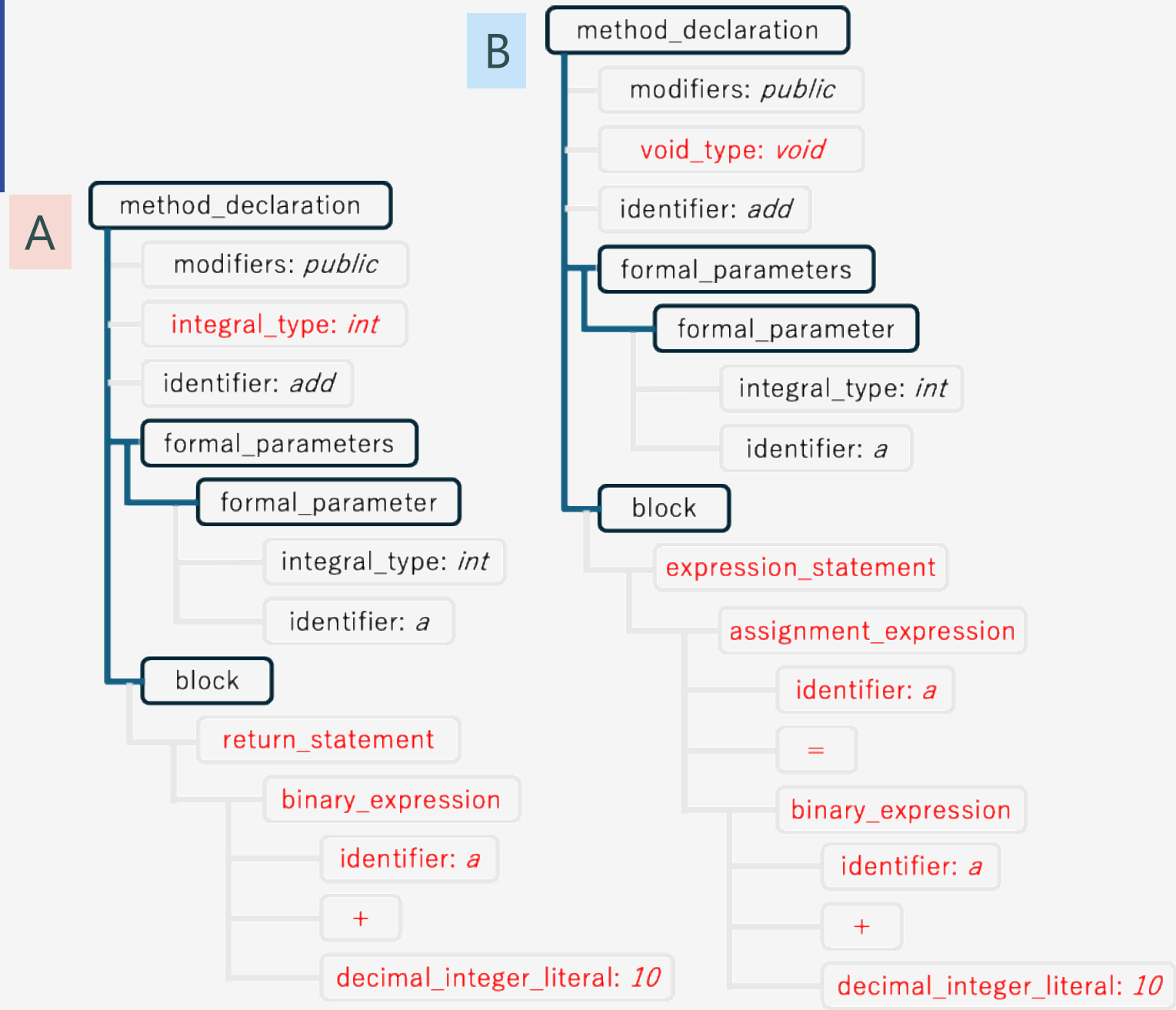
葉ノード，単文以下の
ノードを除いたASTが同一

コードA

```
public int add(int a) {
    return a + 10;
}
```

コードB

```
public void add(int a) {
    a = a + 10;
}
```



実装

AST解析ライブラリである**Tree-sitter**を利用

- 多言語対応したAST解析ライブラリ
- 現在はJavaに対する実装，将来的に他言語への拡張

評価実験

被験者実験

提案手法のラベリングが人間の感覚とずれているのかを評価

コードクローンの大規模データセット**BigCloneBench**を利用

- 既存のコードクローン研究に利用されているため

被験者：大学院博士前期課程の学生6人

既存データセット：BigCloneBench

BigCloneBench(BCB)[3]

- 大規模なコードクローンデータセット
- 約800万のクローンペアが存在
- Type1, Type2, Type3(Strong), Type3(Moderate), Type3(Weak)に分類
  - Type3は一致する行の割合で以下のように分類
    - 一致率 0.7~1.0：**Strong**
    - 一致率 0.5~0.7：**Moderate**
    - 一致率 0.0~0.5：**Weak**
- 既存の研究ではType3(Weak)をType4として利用

[3] J. Svajlenko, J. F. Islam, I. Keivanloo, C. K. Roy, and M. M. Mia, “Towards a big data curated benchmark of inter-project code clones,” in Int. Conf. on Software Maintenance and Evolution (ICSME), 2014.

実験用データ：BigCloneBenchの分類結果

BigCloneBench

提案手法	Type1	Type2	Type3	Type4	合計
Type1	48,116	0	0	0	48,116
Type2	0	4,234	3,649	1	7,884
Type3	0	0	21,286	46,973	68,259
Type4	0	0	85,337	8,403,230	8,488,567
合計	48,116	4,234	110,272	8,450,204	8,612,826

BCBでType3，Type4に分類されたクローンが提案手法では異なる分類

実験用データ：BigCloneBenchの分類結果

BigCloneBench

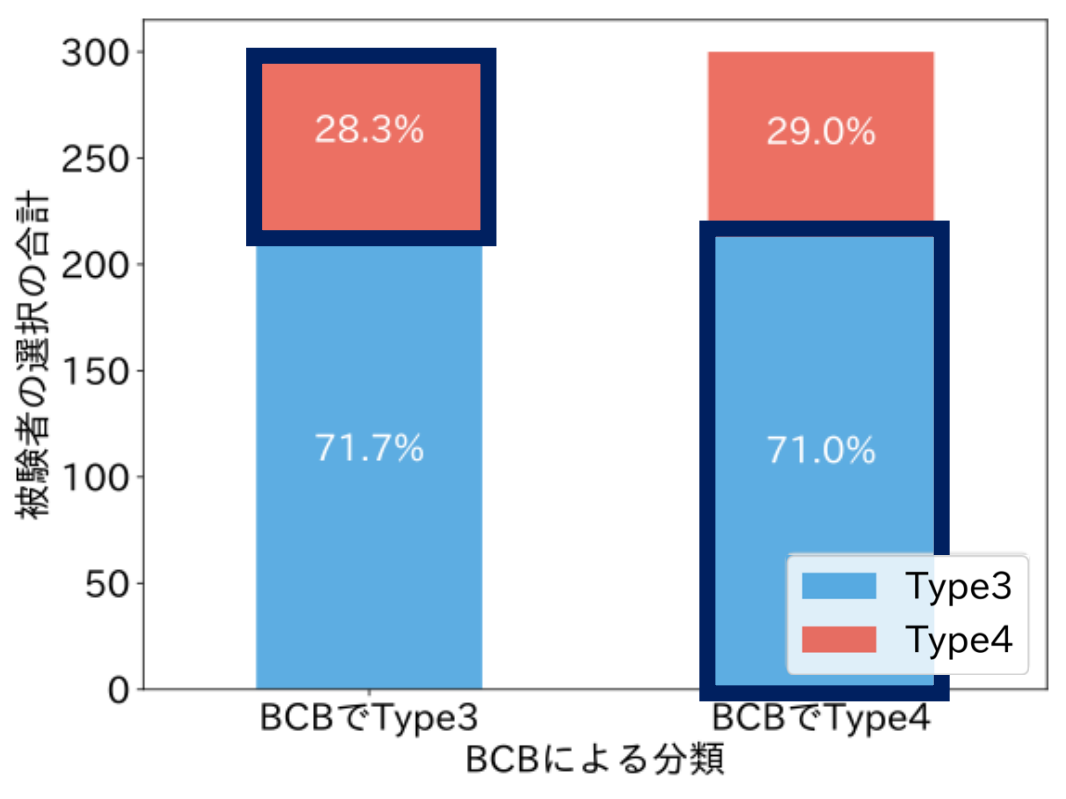
提案手法		Type1	Type2	Type3	Type4	合計
	Type1	48,116	0	0	0	48,116
	Type2	0	4,234	3,649	1	7,884
	Type3	0	0	21,286	46,973	68,259
	Type4	0	0	85,337	8,403,230	8,488,567
	合計	48,116	4,234	110,272	8,450,204	8,612,826

以下のクローンペア**100個**についてType3/4の判定

- BCBでType3，提案手法でType4
- BCBでType4，提案手法でType3

評価結果

- BCBでType3提案手法でType4は，**提案手法**が人間の感覚に**反する**
- BCBでType4提案手法でType3は，**提案手法**が人間の感覚に**近い**



人間の感覚に反した分類のコードを確認

提案手法Type4，6人全員がType3

- **制御構造の条件式のみが異なる**
- 仮引数が異なる

コード例：提案手法でType4，被験者はType3

while文の条件式が異なる

```
private void writeFile (...) throws IOException {
        ...
        while (read = inFile.read (buf)) > 0 \&\& ! stopped) outFile.write (buf, 0,
read);
        ...
}
```

```
private void writeFile (...) throws IOException {
        ...
        while (read = inFile.read (buf)) > 0) outFile.write (buf, 0, read);
        ...
}
```

まとめ

- コードクローン検出ツールはラベリングが一致しない
- 複数のツールの検出結果を単一のデータセットに統合できない
- 課題解決のため，クローン検出後のラベリング手法を提案
- 被験者実験を通じて，提案手法を評価した

今後の課題

- 提案手法でラベリングされたデータセットでLLMへの学習と精度評価
- Java以外の言語への拡張

付録

BigCloneBenchの問題

問題点

- Type3とType4が明確に分類されていない
- ラベル付が適切でない
  このデータセットを用いた研究結果の妥当性が危うくなる
- 特に機械学習の学習データセットとして用いるには問題がある[4]

[4] Krinke, Jens & Ragkhitwetsagul, Chaiyong. (2025). How the Misuse of a Dataset Harmed Semantic Clone Detection.

BigCloneBenchの分類結果

BigCloneBench

提案手法	Type1	Type2	Strongly -Type3	Moderately -Type3	Weakly -Type3
Type1	48,116	0	0	0	0
Type2	0	4,234	3,637	12	1
Type3	0	0	7,984	13,302	46,973
Type4	0	0	10,345	74,992	8,403,230
合計	48,116	4,234	21,966	88,306	8,450,204