

Accurate Method and Variable Tracking in Commit History

Mehran Jodavi, Nikolaos Tsantalis

In Proceedings of the 30th ACM Joint European Software Engineering Conference and
Symposium on the Foundations of Software Engineering, 2022, pp. 183–195

コミット履歴におけるコード追跡性

ソフトウェアエンジニアリングタスクを円滑に進める
うえで重要

- 他のブランチの変更をマージする
- バグが混入したコミットを発見する
- コードの背後にある根拠を読み取る など

開発者はクラスやメソッド、変数など細かい粒度で追跡できることを期待している^{[14][26]}

しかし、既存のツールではメソッド抽出などのリファクタリングを考慮した追跡が実現できていない

- 現在の SOTA はCodeShovel^[14]

研究目的

メソッドや変数など細かい粒度での追跡を可能とする
手法（CodeTracker）の提案

（CodeTracker の貢献）

- 従来手法で追跡できない範囲への対応
 - リファクタリングが適用されたメソッドの追跡維持
 - 変数の追跡
- コードだけでなく、ドキュメントやアノテーションの追跡
- 高い性能でのメソッドや変数などの追跡が可能
- 実行時間の短縮化 など

取得する情報の粒度

Container

- class や interface など

Method

- class に実装されている
メソッド/関数

Variables

- メソッド内で宣言されて
いる変数

```

Zuul-core/src/main/java
package com.netflix.zuul;

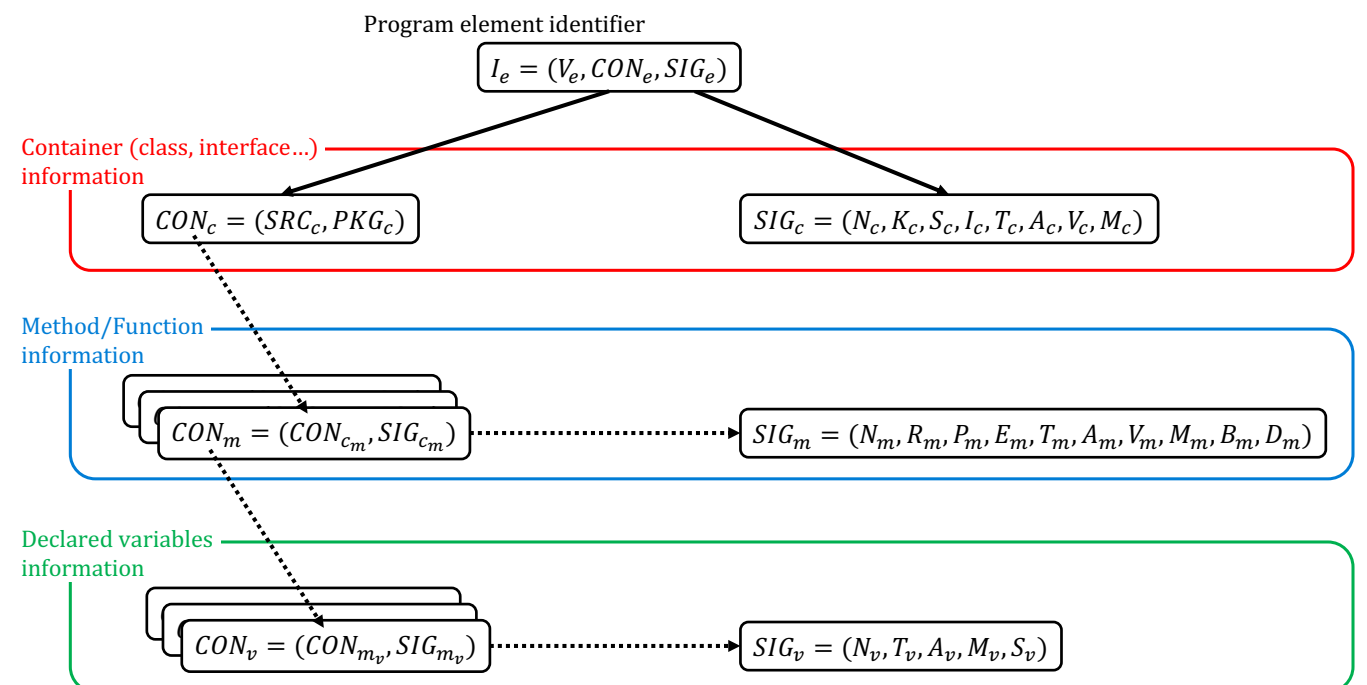
public class FilterProcessorImpl {
    protected final FilterLoader filterLoader;
    protected ZuulFilter getErrorEndpoint(ZuulMessage msg) {
        SessionContext context = msg.getContext();
        String endpointName = context.getErrorEndpoint();
        ...
        ZuulFilter errorEndpoint = getFilterByNameAndType(
            endpointName, FilterType.ENDPOINT);
        if(errorEndpoint == null) {
            String errorStr = "..." + endpointName;
            LOG.error("..." + errorStr, context.getError());
        }
        return errorEndpoint;
    }
}

```

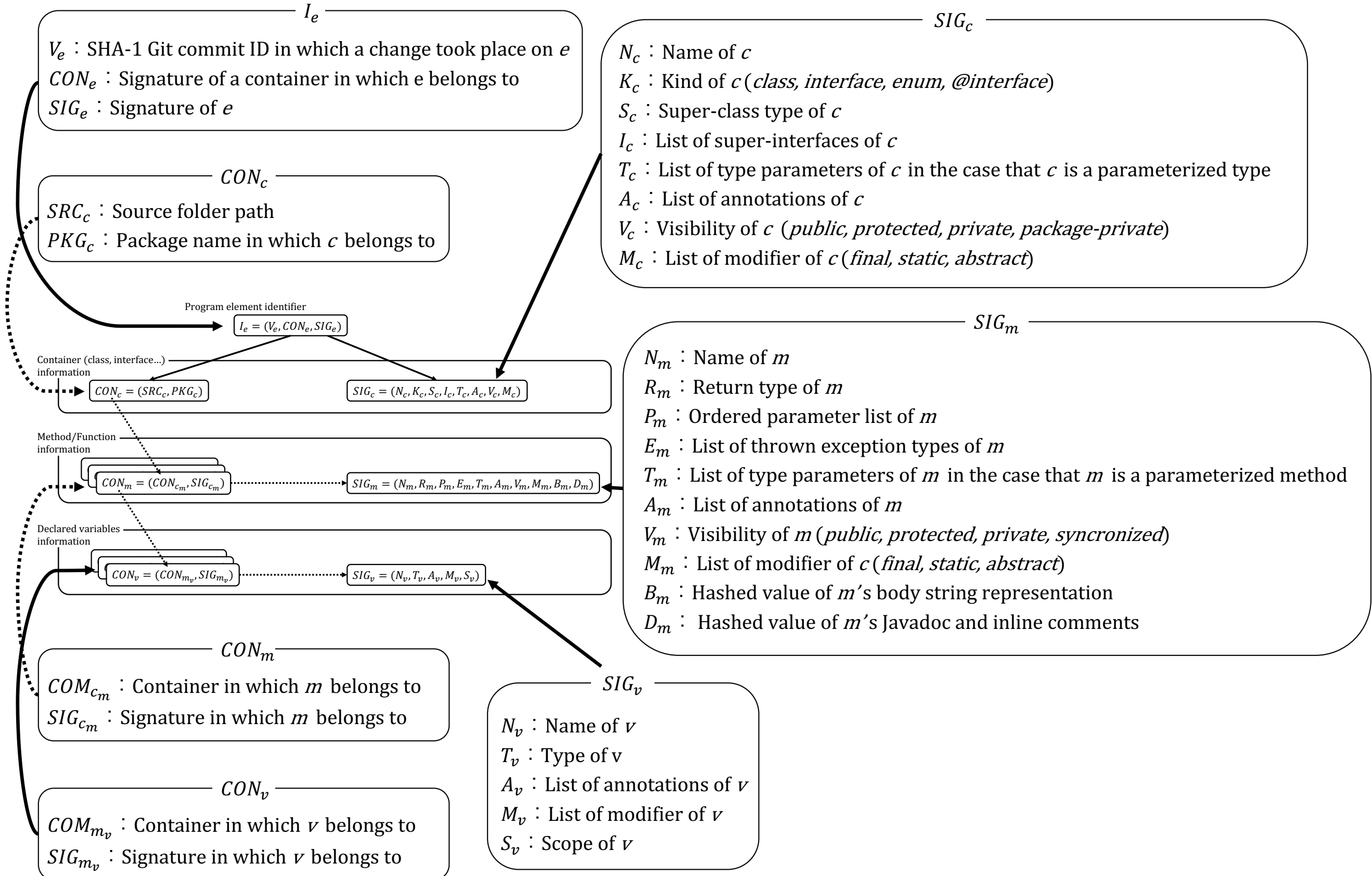
Container

Variable

Method



プログラム断片の対応付けに用いる識別子の構成



追跡手順（簡略版）

Input

- Git repository URL
- Starting SHA-1 commit ID (or HEAD by default)
- File path containing the program element of interest
- name of the program element
- start line number of the program element's declaration

(1) `git log --follow "filePath"`

- ファイル名の変更も考慮しつつ、指定したファイルに関係するコミットを全て取得

(2) 指定したプログラム断片 e を宣言行番号を用いて特定し、コミット前後のリビジョンで識別子が等価どうか確認する

- 等価である場合, “NoChange”と報告（異なる場合は次のステップ3へ）

(3-1) 指定したプログラム断片 e がメソッドである場合, 呼び出しメソッドに追加や変更がないか確認. 呼び出しメソッドが変化していれば RefactoringMiner を実行し, その操作を記録し結びつける

- “ExtractMethod”や“InlinedMethod”がおこなわれていた場合, 抽出されたメソッドの結びつけなどをおこなうため

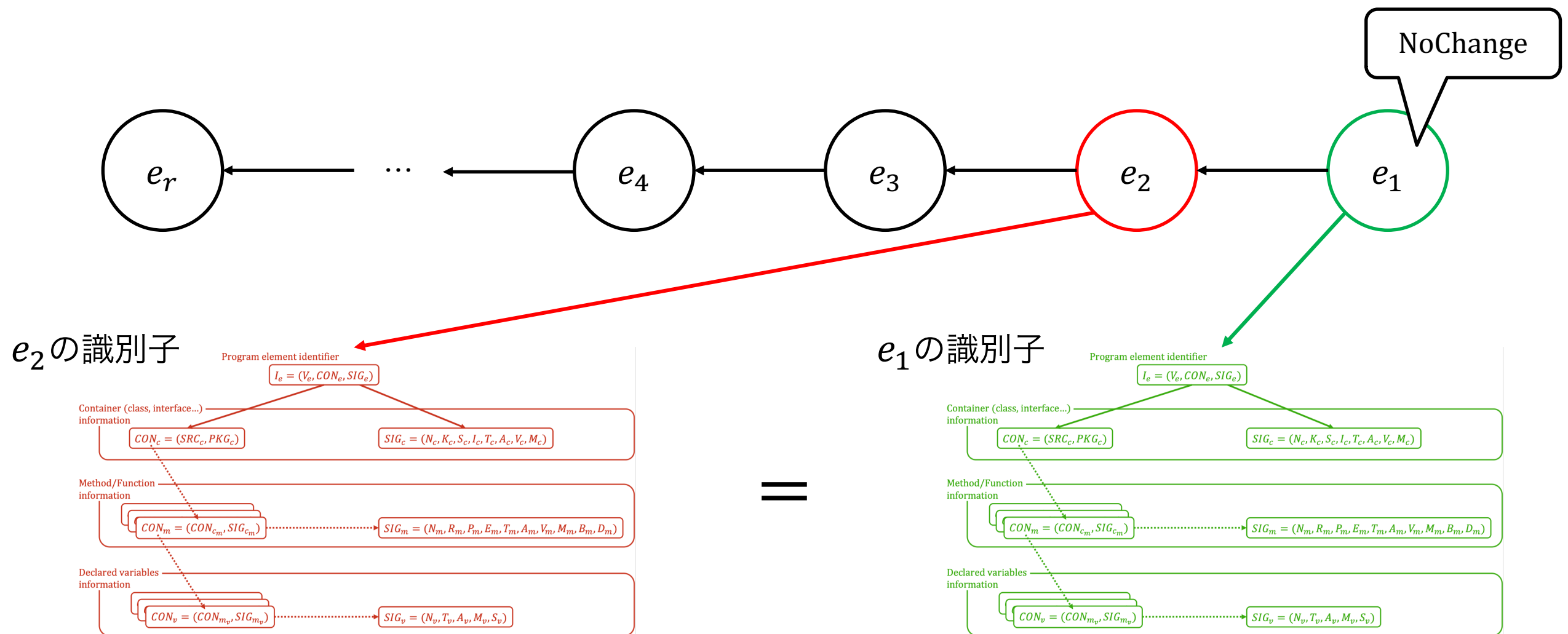
(3-2) 指定したプログラム断片 e が変数である場合, 宣言行番号や型などを用いて変数におこなわれた操作を記録し結びつける

Output

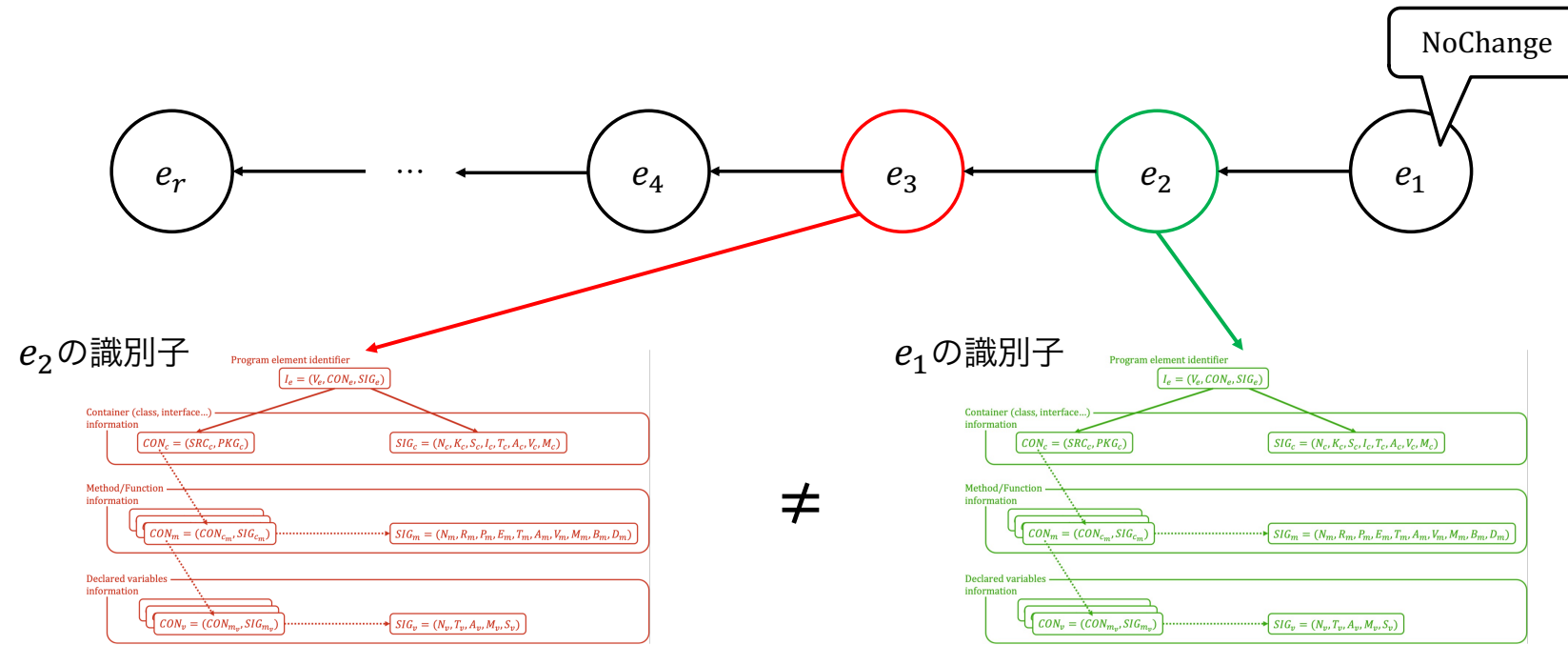
- Graph
- List of commits in which program element e change and along with corresponding kinds of changes

リファクタリングなしの場合

収集した識別子が完全一致する場合はそのまま結びつけ，“NoChange”と報告

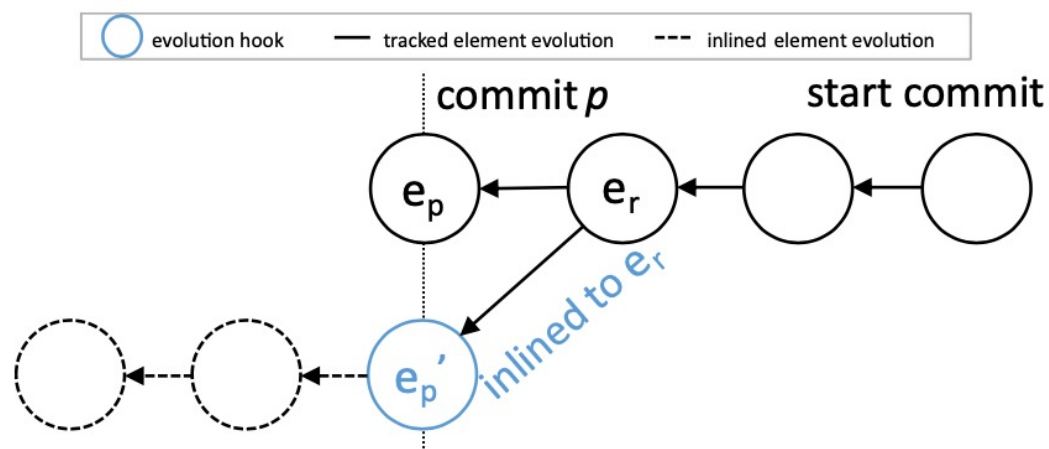


リファクタリングが含まれる場合

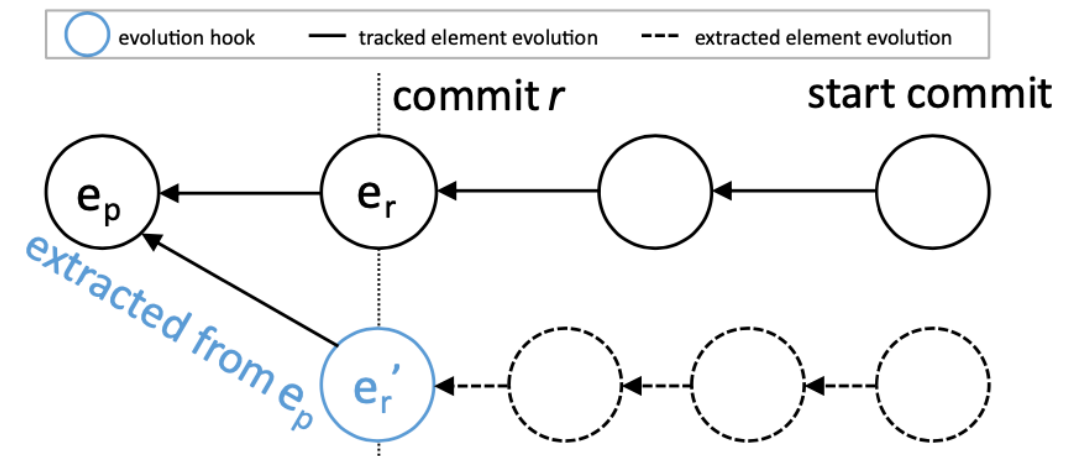


「呼び出しメソッドの数が異なる」や「呼び出しメソッド名が異なる」場合、RefactoringMiner を実行

“InlinedMethod”の結びつけ



“ExtractMethod”の結びつけ



Research Question

RQ1

- CodeShovel と比較して、CodeTracker のメソッド追跡精度はどの程度か？

RQ2

- CodeTracker の変数追跡精度はどの程度か？

RQ3

- CodeShovel と比較して、CodeTracker の実行時間はどうか？

RQ4

- RefactoringMiner のデフォルト動作に比べて、CodeTracker の実行時間の高速化はどの程度か？

RQ1：メソッド追跡精度

CodeShovel^[14]で用いられたデータセットを利用

- CodeShovel はモデルのパラメータ調整が必要なため、Training set と Testing set を用意している
- 変数が少ないメソッドに限定（手動検証するため）

Change Kind	Training set			Testing set		
	C	R	N	C	R	N
Body Change	2276	234	29	459	68	26
File Move	238	24	27	160	7	27
Parameter Change	220	30	6	68	19	8
Return Type Change	47	10	4	15	2	1
Modifier Change	44	12	2	16	4	1
Exception Change	40	9	0	5	3	2
Rename	14	8	7	16	6	2
Method Move	19	23	3	14	27	3
Introduced	81	18	19	83	17	17
Documentation Change	—	—	439	—	—	93
Annotation Change	—	—	42	—	—	24
Total	2979	368	578	836	153	204

C：先行研究と共通して用いるデータ R：手動で検証した結果、誤っており除去したデータ N：新たに追加したデータ

RQ1：メソッド追跡精度

コミットレベルでのメソッド追跡精度

Table 3: Method tracking precision/recall at commit level

Dataset	Tool	TP	FP	FN	Precision	Recall
Training	CodeShovel	2751	219	74	92.63	97.38
	CodeTracker	2825	1	0	99.96	100
Testing	CodeShovel	776	68	65	91.94	92.27
	CodeTracker	839	0	0	100	100
Overall	CodeShovel	3527	287	139	92.48	96.21
	CodeTracker	3664	1	0	99.97	100

該当するメソッドに変更があったコミットに関して、ほぼ確実に特定することが可能

変更の種類レベルでのメソッド追跡精度

Table 4: Method tracking precision/recall at change level

Dataset	Tool	TP	FP	FN	Precision	Recall
Training	CodeShovel	3412	304	145	91.82	95.92
	CodeTracker	3557	3	0	99.92	100
Testing	CodeShovel	915	136	125	87.06	87.98
	CodeTracker	1037	3	3	99.71	99.71
Overall	CodeShovel	4327	440	270	90.77	94.13
	CodeTracker	4594	6	3	99.87	99.93

該当するメソッドにあった変更の種類の特定もほぼ確実に可能

RQ2：変数追跡精度

CodeShovel は変数追跡に対応していないため、CodeTracker のみでの評価

変数の追跡もほぼ確実におこなうことができる

Table 2: Number of instances per change kind for variables.

Change Kind	Training set	Testing set
Introduced	967	378
Rename	145	27
Annotation Change	13	4
Type Change	264	67
Modifier Change	123	52
Total	1512	528

データセットの内訳

Table 5: Variable tracking precision/recall for CodeTracker

Dataset	Level	TP	FP	FN	Precision	Recall
Training	Commit	1459	2	1	99.86	99.93
	Change	1510	4	2	99.74	99.87
Testing	Commit	512	1	0	99.81	100
	Change	527	3	1	99.43	99.81
Overall	Commit	1971	3	1	99.85	99.95
	Change	2037	7	3	99.66	99.85

CodeTracker の変数追跡精度

まとめ

目的

- メソッドや変数など細かい粒度での追跡を可能とする手法（CodeTracker）の提案

貢献

- 従来手法で追跡できない範囲への対応
 - リファクタリングが適用されたメソッドの追跡維持
 - 変数の追跡
- コードだけでなく、ドキュメントやアノテーションの追跡
- 高い性能でのメソッドや変数などの追跡が可能
- 実行時間の短縮化 など

対応可能な操作範囲（ラベル）

青文字は新たに対応可能となったもの

メソッドレベルで対応可能な操作

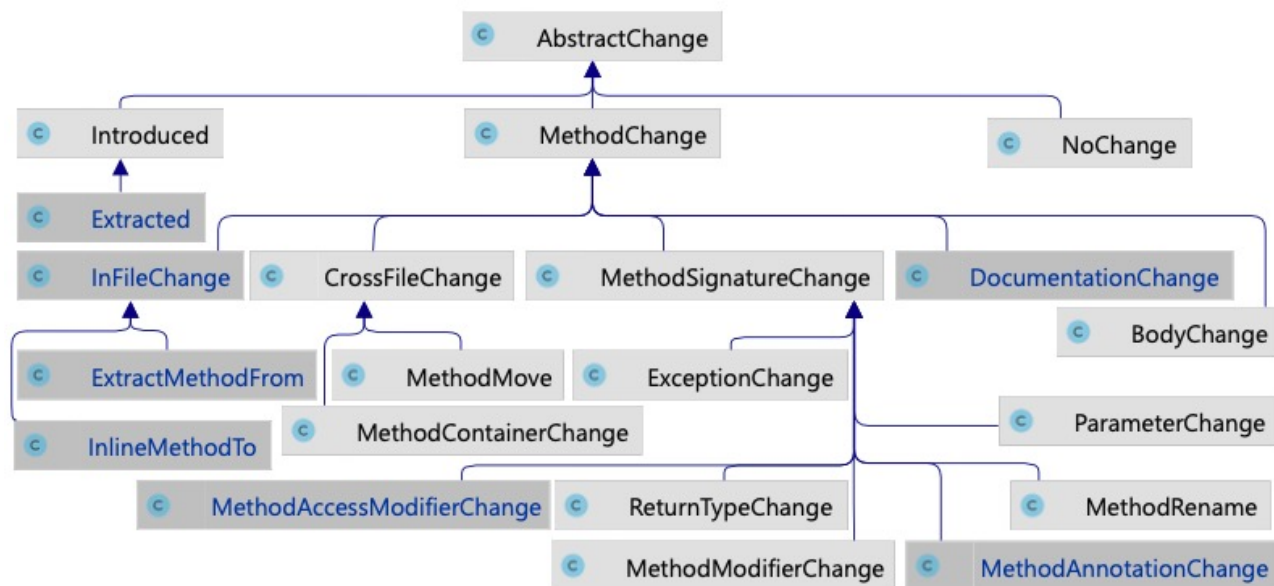


Figure 2: Hierarchy of supported change kinds for methods adopted by CodeShovel [14]. Newly supported change kinds are highlighted in blue colour.

変数レベルで対応可能な操作

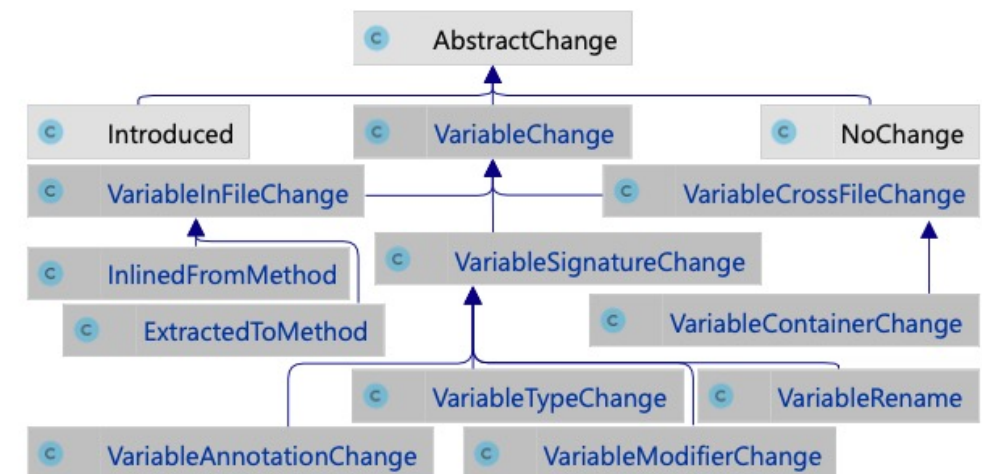


Figure 3: Hierarchy of supported change kinds for variables.