

Fine-grained and Accurate Source Code Differencing

Jean-Rémy Falleri , Floréal Morandat, Xavier Blanc,
Matias Martinez, Martin Monperrus

ASE '14 Proceedings of the 29th ACM/IEEE
international conference on Automated
software engineering Pages 313-324

Edit Script

- ソースコードに加えられた操作
- ソースコードの進化を分析する上で重要
- 従来の手法では行単位で追加，削除しか取れない

問題点

- ESの中でコードの移動は頻繁に起こりうる
- より進んだ分析をするために移動の抽出が必要

目的

- 追加・削除に加えて移動も抽出できるES取得手法
GumTreeを構築
- 従来手法よりもきめ細かくESを取得

アプローチ

- ソースコードを抽象構文木(AST)に変換
- 2つのASTのノードをマッピングして差分を取得

GumTreeの概要

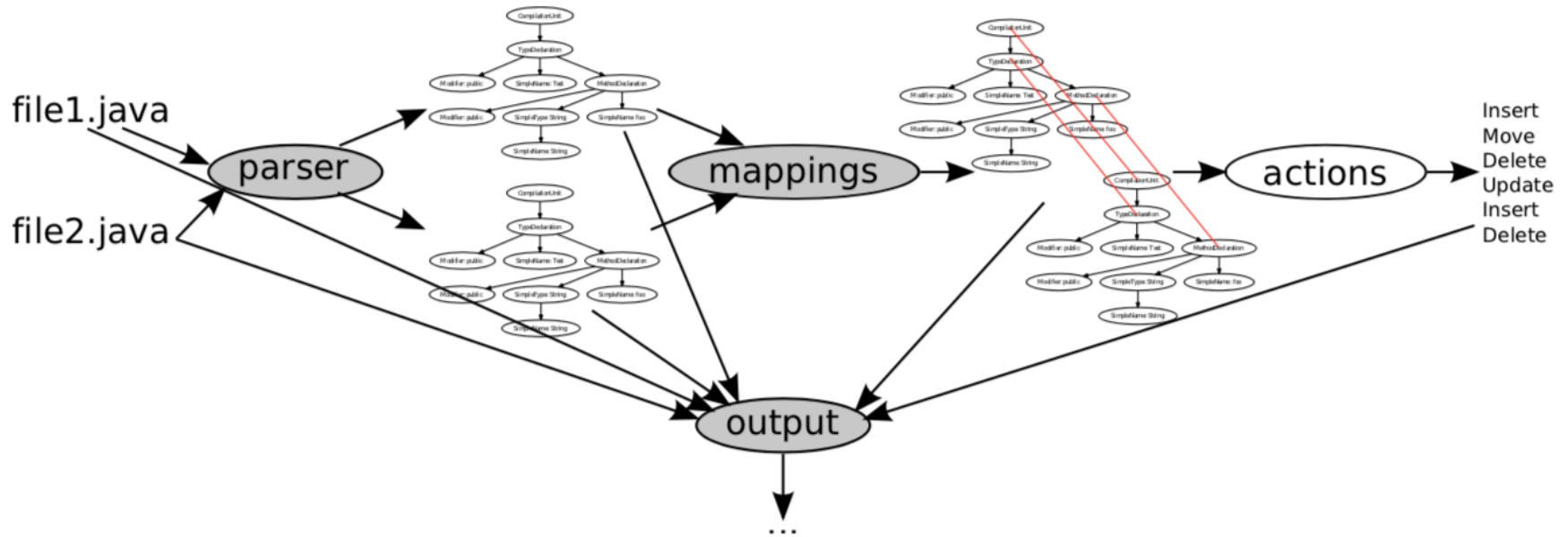


Figure 2: Our pipe and filter architecture. Abstract modules are greyed.

ESの種類

- $updateValue(t, v_n)$
 - ノード t の値を v_n に更新
- $add(t, t_p, i, l, v)$
 - 親ノード t_p の i 番目にノード t を追加
 - 親ノード t_p がない場合はノード t は根となり, 以前の根はノード t の子ノード
 - l はノード t のラベル, v はノード t の値

ESの種類

- $delete(t)$
 - 葉ノード t を削除
- $move(t, t_p, i)$
 - 親ノード t_p の i 番目にノード t を移動
 - ノード t の子ノードもそのまま移動

ノードのマッピング

- [1]で構築されたマッピングアルゴリズムを用いる
 - トップダウンアルゴリズム
 - 2つのAST間の同形の部分木のマッピング
 - ボトムアップアルゴリズム
 - 2つのAST間のノードのマッピング

トップダウンアルゴリズム

- L : height-indexed priority
- A : マッピング候補
- M : マッピング集合
- $\text{push}(t, L)$: L に木 t を追加
- $\text{open}(t, L)$: L に木 t の子ノードを追加
- $\text{peekMax}(L)$: L にある最も高い木を取り出す
- $\text{pop}(L)$: L にある最も高い木を取り出して L から削除
- $\text{isomorphic}(t_1, t_2)$: 木 t_1 , t_2 が同形であるかどうかを返す
- $\text{dice}(t_1, t_2, M) = \frac{2 \times |\{t_1 \in s(t_1) \mid (t_1, t_2) \in M\}|}{|s(t_1)| + |s(t_2)|}$
 - 木 t_1 , t_2 の子孫のうち共通のもの割合
 - $s(t)$: 木 t の子孫

Algorithm 1: The algorithm of the top-down phase.

Data: A source tree T_1 and a destination tree T_2 , a minimum height minHeight , two empty height-indexed priority lists L_1 and L_2 , an empty list $\mathcal{A}$ of candidate mappings, and an empty set of mappings $\mathcal{M}$

Result: The set of mappings $\mathcal{M}$

```

1 push(root( $T_1$ ),  $L_1$ );
2 push(root( $T_2$ ),  $L_2$ );
3 while  $\min(\text{peekMax}(L_1), \text{peekMax}(L_2)) > \text{minHeight}$  do
4   if  $\text{peekMax}(L_1) \neq \text{peekMax}(L_2)$  then
5     if  $\text{peekMax}(L_1) > \text{peekMax}(L_2)$  then
6       foreach  $t \in \text{pop}(L_1)$  do open( $t$ ,  $L_1$ )
7     else
8       foreach  $t \in \text{pop}(L_2)$  do open( $t$ ,  $L_2$ )
9   else
10     $H_1 \leftarrow \text{pop}(L_1)$ ;
11     $H_2 \leftarrow \text{pop}(L_2)$ ;
12    foreach  $(t_1, t_2) \in H_1 \times H_2$  do
13      if isomorphic( $t_1, t_2$ ) then
14        if  $\exists t_x \in T_2 \mid \text{isomorphic}(t_1, t_x) \wedge t_x \neq t_2$ 
15          or  $\exists t_x \in T_1 \mid \text{isomorphic}(t_x, t_2) \wedge t_x \neq t_1$ 
16          then
17            add( $\mathcal{A}$ ,  $(t_1, t_2)$ );
18          else
19            add all pairs of isomorphic nodes of  $s(t_1)$ 
20              and  $s(t_2)$  to  $\mathcal{M}$ ;
21    foreach  $t_1 \in H_1 \mid (t_1, t_x) \notin \mathcal{A} \cup \mathcal{M}$  do open( $t_1$ ,  $L_1$ )
22    foreach  $t_2 \in H_2 \mid (t_x, t_2) \notin \mathcal{A} \cup \mathcal{M}$  do open( $t_2$ ,  $L_2$ )
23  sort  $(t_1, t_2) \in \mathcal{A}$  using dice(parent( $t_1$ ), parent( $t_2$ ),  $\mathcal{M}$ );
24  while  $\text{size}(\mathcal{A}) > 0$  do
25     $(t_1, t_2) \leftarrow \text{remove}(\mathcal{A}, 0)$ ;
26    add all pairs of isomorphic nodes of  $s(t_1)$  and  $s(t_2)$  to  $\mathcal{M}$ ;
27     $\mathcal{A} \leftarrow \mathcal{A} \setminus \{(t_1, t_x) \in \mathcal{A}\}$ ;
28     $\mathcal{A} \leftarrow \mathcal{A} \setminus \{(t_x, t_2) \in \mathcal{A}\}$ ;

```

トップダウンアルゴリズム

Algorithm 1: The algorithm of the top-down phase.

Data: A source tree T_1 and a destination tree T_2 , a minimum height $minHeight$, two empty height-indexed priority lists L_1 and L_2 , an empty list $\mathcal{A}$ of candidate mappings, and an empty set of mappings $\mathcal{M}$

Result: The set of mappings $\mathcal{M}$

```

1 push(root( $T_1$ ),  $L_1$ );
2 push(root( $T_2$ ),  $L_2$ );
3 while  $\min(\text{peekMax}(L_1), \text{peekMax}(L_2)) > minHeight$  do
4   if  $\text{peekMax}(L_1) \neq \text{peekMax}(L_2)$  then
5     if  $\text{peekMax}(L_1) > \text{peekMax}(L_2)$  then
6       foreach  $t \in \text{pop}(L_1)$  do open( $t$ ,  $L_1$ )
7     else
8       foreach  $t \in \text{pop}(L_2)$  do open( $t$ ,  $L_2$ )
9   else
10     $H_1 \leftarrow \text{pop}(L_1)$ ;
11     $H_2 \leftarrow \text{pop}(L_2)$ ;
12    foreach  $(t_1, t_2) \in H_1 \times H_2$  do
13      if isomorphic( $t_1, t_2$ ) then
14        if  $\exists t_x \in T_2 \mid \text{isomorphic}(t_1, t_x) \wedge t_x \neq t_2$ 
15          or  $\exists t_x \in T_1 \mid \text{isomorphic}(t_x, t_2) \wedge t_x \neq t_1$ 
16          then
17            add( $\mathcal{A}$ ,  $(t_1, t_2)$ );
18        else
19          add all pairs of isomorphic nodes of  $s(t_1)$ 
20            and  $s(t_2)$  to  $\mathcal{M}$ ;
21    foreach  $t_1 \in H_1 \mid (t_1, t_x) \notin \mathcal{A} \cup \mathcal{M}$  do open( $t_1$ ,  $L_1$ )
22    foreach  $t_2 \in H_2 \mid (t_x, t_2) \notin \mathcal{A} \cup \mathcal{M}$  do open( $t_2$ ,  $L_2$ )
23  sort  $(t_1, t_2) \in \mathcal{A}$  using dice(parent( $t_1$ ), parent( $t_2$ ),  $\mathcal{M}$ );
24  while  $\text{size}(\mathcal{A}) > 0$  do
25     $(t_1, t_2) \leftarrow \text{remove}(\mathcal{A}, 0)$ ;
26    add all pairs of isomorphic nodes of  $s(t_1)$  and  $s(t_2)$  to  $\mathcal{M}$ ;
27     $\mathcal{A} \leftarrow \mathcal{A} \setminus \{(t_1, t_x) \in \mathcal{A}\}$ ;
28     $\mathcal{A} \leftarrow \mathcal{A} \setminus \{(t_x, t_2) \in \mathcal{A}\}$ ;

```

高さが同じ部分木
についてマッピング
やマッピング候補
を抽出

マッピング候補
からマッピング
のみを抽出

同形の木のペア
が複数ある場合
はAに入れる
ペアが単一の場
合はMに入れる

ボトムアップアルゴリズム

$\text{opt}(t_1, t_2)[2]$: ESを返す

トップダウンで求めたマッピングされた部分木からさらにマッピングできるノードを抽出

Algorithm 2: The algorithm of the bottom-up phase.

Data: Two trees T_1 and T_2 , a set $\mathcal{M}$ of mappings (resulting from the top-down phase), a threshold minDice and a maximum tree size maxSize

Result: The set of mappings $\mathcal{M}$.

```
1 foreach  $t_1 \in T_1 \mid t_1$  is not matched  $\wedge t_1$  has matched  
   children, in post-order do  
2    $t_2 \leftarrow \text{candidate}(t_1, \mathcal{M});$   
3   if  $t_2 \neq \text{null}$  and  $\text{dice}(t_1, t_2, \mathcal{M}) > \text{minDice}$  then  
4      $\mathcal{M} \leftarrow \mathcal{M} \cup (t_1, t_2);$   
5     if  $\max(|s(t_1)|, |s(t_2)|) < \text{maxSize}$  then  
6        $\mathcal{R} \leftarrow \text{opt}(t_1, t_2);$   
7       foreach  $(t_a, t_b) \in \mathcal{R}$  do  
8         if  $t_a, t_b$  not already mapped and  
            $\text{label}(t_a) = \text{label}(t_b)$  then  
9            $\mathcal{M} \leftarrow \mathcal{M} \cup (t_a, t_b);$ 
```

マッピングの例

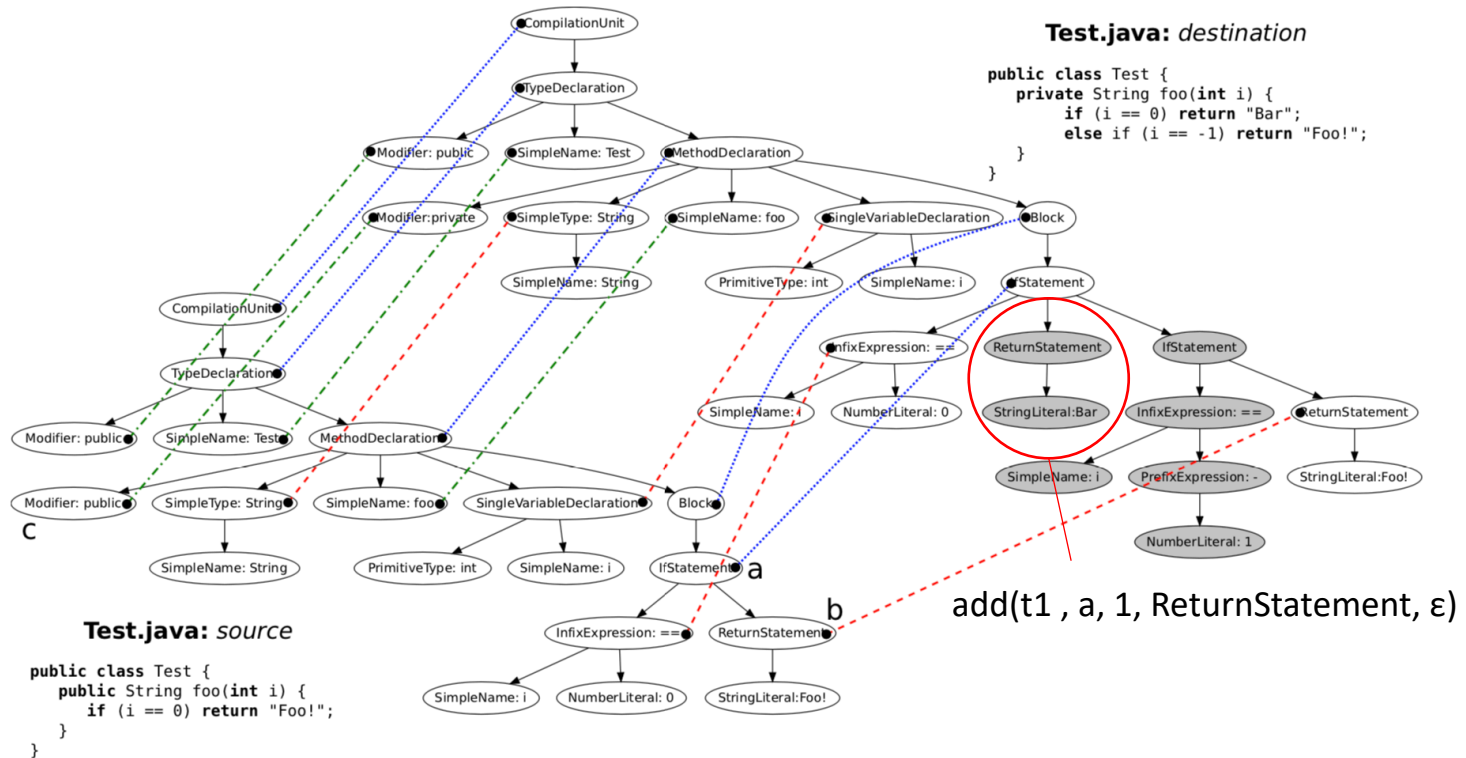


Figure 1: Two sample Java files with their corresponding ASTs and mappings. Nodes with a label only are denoted: Label, nodes with a label and a value are denoted: Label: value. Mappings from the top-down phase are depicted with long-dotted lines (descendants of these nodes are also mapped but it is omitted to enhance readability). Mappings from the bottom-up phase are depicted using short-dotted lines (container mappings) or alternate-dotted lines (recovery mappings). Unmatched nodes are greyed.

GumTreeの評価実験

- 被験者による出力結果の正しさの評価
 - 最新のテキストベースのdiffツールと使いやすさを比較
- 性能の評価
 - マッピングできたノードの多さと検出したESのサイズの小ささを他のASTベースの差分取得ツールと比較
- 移動の検出精度の評価
 - 移動を検出できる他のツールと検出精度を比較

被験者による出力結果の正しさの評価

		Full (3/3)	Majority (2/3)
#1	GT does good job	122	137
	GT does not good job	3	3
	Neutral	0	1
#2	GT better	28	66
	Diff better	3	12
	Equivalent	45	61

GT : GumTree

Table 1: Agreements of the manual inspection of the 144 transactions by three raters for Question #1 (top) and Question #2 (bottom).

- 筆者のうち3人が144ペアのスク립トを差分を見て評価
- 質問
 - GumTreeは役立つか？
 - GumTreeは最新のdiffより使いやすいか

性能の評価

CDG		GT better	CD better	Equiv.
	Mappings	4007 (31.32%)	542 (4.24%)	8243 (64.44%)
CDG	ES size	4938 (38.6%)	412 (3.22%)	7442 (58.18%)
JDTG		GT better	CD better	Equiv.
	Mappings	8378 (65.49%)	203 (1.59%)	4211 (32.92%)
JDTG	ES size	10358 (80.97%)	175 (1.37%)	2259 (17.66%)
JDTG		GT better	RTED better	Equiv.
	Mappings	2806 (21.94%)	1234 (9.65%)	8752 (68.42%)
JDTG	ES size	3020 (23.61%)	2193 (17.14%)	7579 (59.25%)

CDG : ChangeDistiller granularity

JDTG : JDT granularity

CD : ChangeDistiller

Table 2: Number of cases where GumTree is better (resp. worse and equivalent) than ChangeDistiller (top, middle) and RTED (bottom) for 2 metrics, number of mappings and edit script size (ES size), at the CDG granularity (top) and JDTG granularity (middle, bottom).

- 2つのASTベースの差分取得ツールを比較対象として以下の性能を評価
 - マッピングできたノードの数の大きさ
 - 検出したESの粒度の小ささ
- データセットはjavaで書かれた12792ペアのファイル

移動の検出精度の評価

	GT only move op	GT other op
CD only move op	77	1
CD other op	52	12662

Table 3: Comparison of the number of move operations from GumTree and ChangeDistiller for 12 792 file pairs to be compared.

- データは性能の評価に用いたものを使用
- 既存研究よりも検出できた移動の数が多いかを評価

まとめ・所感

- まとめ
 - 既存手法とは別のマッピングアルゴリズムを用いたASTベースの差分抽出手法を構築
 - 既存のツールよりも粒度の細かいESの抽出が可能になった
- 所感
 - 細かい粒度（文や式の単位）で差分が取れるのはありがたい
 - GumTreeの入手・使用が簡単