

MELT: Mining Effective Lightweight Transformations from Pull Requests

Daniel Ramos
School of Computer Science, INESC-ID
Carnegie Mellon University, USA
danielrr@cmu.edu

Hailie Mitchell
Computer Science Department
Dickinson College, USA
mitchelh@dickinson.edu

Inês Lynce
INESC-ID, Instituto Superior Técnico
Universidade de Lisboa, Portugal
ines.lynce@tecnico.ulisboa.pt

Vasco Manquinho
INESC-ID, Instituto Superior Técnico
Universidade de Lisboa, Portugal
vasco.manquinho@inesc-id.pt

Ruben Martins
School of Computer Science
Carnegie Mellon University, USA
rubenm@andrew.cmu.edu

Claire Le Goues
School of Computer Science
Carnegie Mellon University, USA
clegoues@cs.cmu.edu

ASE

API

| 多くの開発者がサードパーティライブラリを使用

- ライブラリの機能実装とAPIは分離されるべき
- ライブラリの機能変更と一緒にAPIが変更されることがある^{[2][3]}

| ライブラリのバージョン移行

- APIの破壊的変更や非推奨によって利用側の移行が困難に
- バージョン移行作業はバグが混入されやすい^[9]
- バージョン移行の自動化の研究が盛ん^[12-17]

既存研究

| API移行のためのツール

- 移行済ライブラリの利用プロジェクトのコミットをマイニング
- 最新のAPIの利用プロジェクトの情報

| ツールの問題点

- これらのツールが存在するにも関わらず81.5%のプロジェクトが古いバージョンを利用^[18]
- 新バージョンがリリースされた直後に使用できない
 - マイニングは利用プロジェクトが移行した後しか実行できない

目的と貢献

| 目的

- 利用プロジェクトデータに依存せずにライブラリのAPI変更に対応するためのルールを抽出
- 利用プロジェクトの変更ルールの汎化性向上

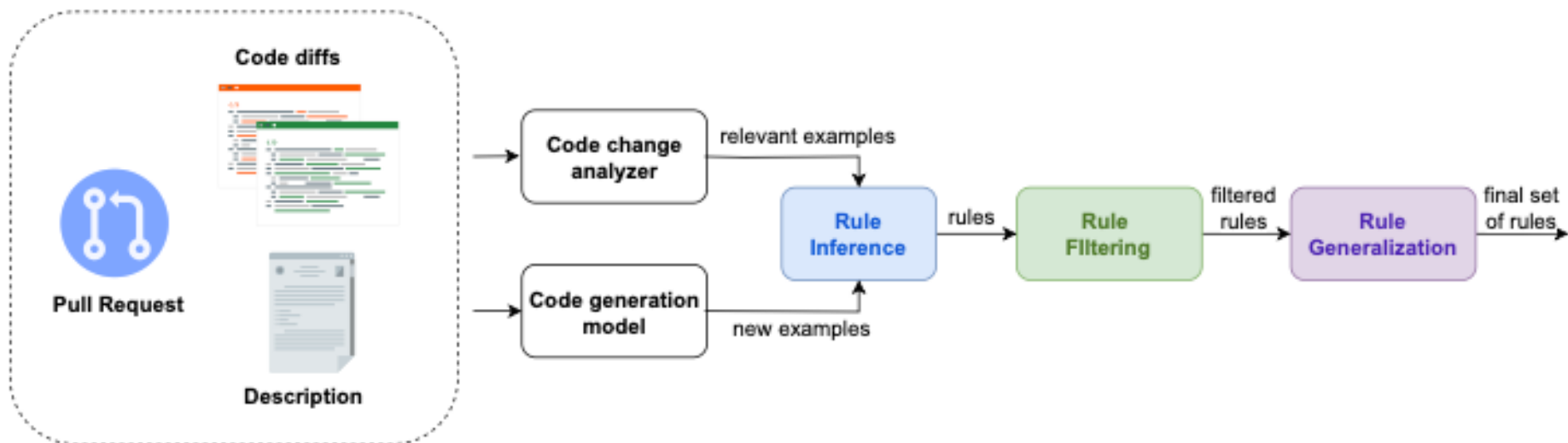
| 貢献

- PR情報を使用することで、ライブラリ側のAPI変更後すぐに本手法の適用が可能
- メンテナが本ツールをCIワークフローに組み込むことが可能

MELTの概要

| ライブラリにマージされたPRを利用

- まず'dupricated'や'breaking change'などのキーワードでフィルタリング
- 変更点や非推奨事項を考慮して利用側コードの変換ルール構築
- PRに含まれるコード, 自然言語両方を使用

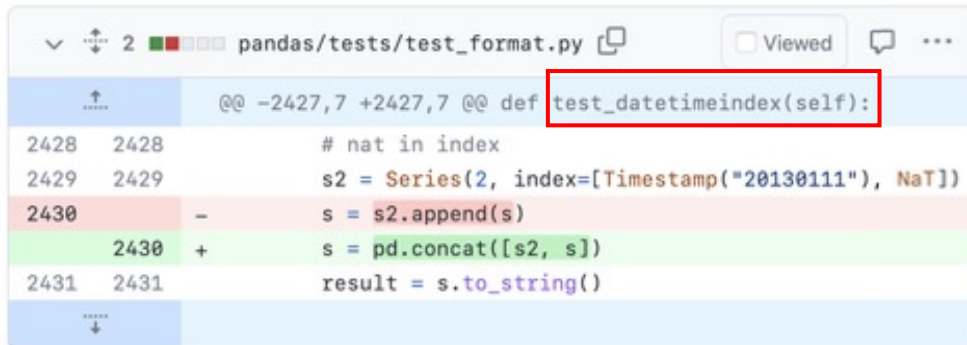


API変更に関連するコードの抽出

| PR内の変更差分がAPIの変更に限るとは限らない

- コード変更によって影響を受ける関数と対応するクラスを特定
- 変更差分とその周辺を保持

pandasのDataFrame.appendが非推奨になった例



The screenshot shows a code diff for the file pandas/tests/test_format.py. The diff highlights the removal of the `append` method and the addition of `pd.concat`. The function `test_datetimeindex(self):` is highlighted with a red box. The diff shows the following changes:

```
@@ -2427,7 +2427,7 @@ def test_datetimeindex(self):
2428 2428         # nat in index
2429 2429         s2 = Series(2, index=[Timestamp("20130111"), NaT])
2430 -         s = s2.append(s)
2430 +         s = pd.concat([s2, s])
2431 2431         result = s.to_string()
```

Code Change Analyzerで
API呼び出し箇所を特定

追加コード例の生成

| API変更に関する利用可能なdiffを持つPRは23.6%

- 自然言語（議論，ドキュメント）を利用可能
- プロンプトを用いてコード例を生成

Algorithm 1 GENERATETRANSITIONEXAMPLES($P, \mathcal{M}, N$)

Input: $\mathcal{P}$: pull request, $\mathcal{M}$: gen model, N : number of samples

Output: E : transition examples

```
1:  $E \leftarrow \emptyset$ 
2: for  $i = 1$  to  $N$  do
3:    $(e_{old}, e_{new}) \leftarrow \text{GENERATEEXAMPLE}(\mathcal{M}, \mathcal{P})$ 
4:    $\mathcal{T}_E \leftarrow \text{GENERATETESTCASES}(\mathcal{M}, \mathcal{P}, e_{old}, e_{new})$ 
5:    $E \leftarrow E \cup \{(e_{old}, e_{new})\}$ 
6:   for each  $\text{test} \in \mathcal{T}_E$  do
7:     if FAILS(test) then
8:        $E \leftarrow E \setminus \{(e_{old}, e_{new})\}$ 
9:     end if
10:  end for
11: end for
12: return  $E$ 
```

使用ライブラリの名前と
PRのデータからコード例の生成



生成されたコードをテスト



テスト失敗したらコード例は破棄

追加コード例の生成

| GPT-4 8kを使用

- テンプレートを使用してPR情報を入力
- 旧用法 e_{old} と新用法 e_{new} を生成
- e_{old} と e_{new} の等価性をテスト

Please review the following Pull Request, which aims to replace a set of APIs with updated alternatives in [\(library_name\)](#). For each breaking/deprecated API and its corresponding new alternative, provide a pair of examples that clearly demonstrate how to transition from old usage to new usage. Ensure that the examples within each pair are directly related and exhibit similar behavior. [\(additional_requirements\)](#).

===== Output format =====

```
```python
Summary: Summary of the breaking changes

import pandas as pd
import numpy as np

def old_usage(x):
 pass

def new_usage(x):
 pass

(...)
```
```

===== Example output =====

```
{concrete_example}
```

===== Input =====

```
{pr_data}
```

Please generate tests for the examples above, to test whether they are equal. You do not need to include the old and new usage examples in the code block.

===== Output format =====

```
```python
import unittest # Do not forget appropriate imports.

class TestEquiv(unittest.TestCase):

 def test_assert1(self):
 # Input initialization
 assert old_usage1(x1) == new_usage1(x1)

 def test_assert2(self):
 # Input initialization
 assert np.allclose(old_usage2(x2), new_usage2(x2),
 rtol=1e-2, atol=1e-2)

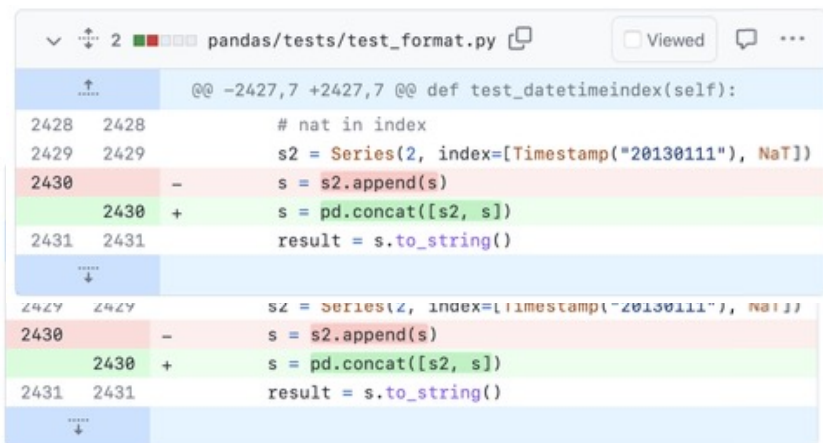
(...)

if __name__ == "__main__":
 unittest.main(verbosity=2, buffer=True)
```
```

ルール生成

| Combyを使用

- ライブラリ名や入る変数は抽象化（テンプレート化）
- 言語によらないテンプレートを生成可能



```
@@ -2427,7 +2427,7 @@ def test_datetimeindex(self):
2428 2428      # nat in index
2429 2429      s2 = Series(2, index=[Timestamp("20130111"), NaT])
2430 -      s = s2.append(s)
2430 +      s = pd.concat([s2, s])
2431 2431      result = s.to_string()

2427 2427      s2 = Series(2, index=[Timestamp("20130111"), NaT])
2430 -      s = s2.append(s)
2430 +      s = pd.concat([s2, s])
2431 2431      result = s.to_string()
```

| Match Template | Rewrite Template |
|---|---|
| <pre>: [[s2]].append(: [[s1]]) where : [[s1]].type == Series : [[s2]].type == Series</pre> | <pre>pd.concat(: [[s2]], : [[s1]])</pre> |
| <pre>: [[df]].append(: [[s]]) where : [[df]].type == DataFrame : [[s]].type == Series</pre> | <pre>pd.concat(: [[df]], DataFrame(: [[s]]).T. infer_objects())</pre> |
| <pre>: [[df]].append(: [[s]]) where : [[df]].type == DataFrame : [[s]].type == Series</pre> | <pre>DataFrame(: [[s]]).T. infer_objects())</pre> |

ルール推論

| API変更に関連するソースファイルをASTへ変換

- 不一致ノードからdiffを特定, InferRuleアルゴリズム^[35]に入力
- メソッド名やクラス名, 引数は抽象化しない
- 型推論言語サーバを使用

(a) Code before migration

```
r = pd.read_csv(  
    filename,  
    compression=comp,  
    encoding=enc,  
    index_col=0,  
    - squeeze=True)
```

(b) Code after migration

```
r = pd.read_csv(  
    filename,  
    compression=comp,  
    encoding=enc,  
    index_col=0).  
+ squeeze()
```

```
: [[a]] =  
: [[i]].read_csv(  
: [[d]],  
compression=: [e],  
encoding=: [f],  
index_col=: [[g]],  
squeeze=: [[h]])
```

→

```
: [[a]] =  
: [[i]].read_csv(  
: [[d]],  
compression=: [e],  
encoding=: [f],  
index_col=: [[g]])  
squeeze()
```

```
: [[a]] =  
: [[i]].read_csv(  
: [[d]],  
compression=: [e],  
encoding=: [f],  
index_col=: [[g]],  
squeeze=True)
```

```
: [[a]] =  
: [[i]].read_csv(  
: [[d]],  
compression=: [e],  
encoding=: [f],  
index_col=: [[g]])  
squeeze()
```

```
where : [[h]].type == int,  
: [[i]].type == pandas
```

ルールの一般化

| 無関係なコンテキストを抽象化

– 引数の一般化

```
: [[a]] =  
  : [[i]].read_csv(  
    [[d]],  
    compression=: [e],  
    encoding=: [f],  
    index_col=: [[g]],  
    squeeze=True)  
→  
: [[a]] =  
  : [[i]].read_csv(  
    : [[d]],  
    compression=: [e],  
    encoding=: [f],  
    index_col=: [[g]])  
  squeeze()
```

```
where : [[h]].type == int,  
      : [[i]].type == pandas
```



```
: [[i]].read_csv(  
  : [args],  
  squeeze=True)  
→  
: [[i]].read_csv(  
  : [args])  
  squeeze()
```

```
where : [[i]].type == pandas
```

RQ

| RQ1 : MELTはどの程度有効なAPI変更ルールを生成できるか

- 1179のPRのうち, 144のPRに対してコード例から461のルールを生成

| RQ2 : 自動生成されたコード例は, PRのコード例をどのように補完するか

- 自動生成されたコード例から生成された156のルールのうち, 114の正しいルールを生成
- 96/114はMELT独自のルール

RQ

| RQ3 : ルールの汎化性は？

- ベースラインと比べてルールのマッチ数は9.07倍に増加
- PRを用いたアプローチが有効であったと示せる

| Library | Original Rule | | Generalized Rule | |
|---------|---|---------|--|---------|
| | Match Template | Matches | Match Template | Matches |
| pandas | <code>:[x]].set_index(:[a], drop=:[b]], inplace=True)</code> | 2 | <code>:[x]].set_index(:[args], inplace=True)</code> | 370 |
| | <code>:[x]].read_csv(:[a]], compression=:[b]], encoding=:[c]], index_col=:[d], squeeze=True)</code> | 0 | <code>:[x]].read_csv(:[args], squeeze=True)</code> | 21 |
| | <code>:[[aai]].apply(:[a], axis=:[b]], reduce=True)</code> | 3 | <code>:[[aai]].apply(:[args], reduce=True)</code> | 4 |
| scipy | <code>jaccard_similarity_score(:[a]], :[b]])</code> | 94 | <code>jaccard_similarity_score(:[args])</code> | 226 |
| | <code>:[x]].filters.gaussian_filter(:[a], :[b], mode=:[c]])</code> | 0 | <code>:[x]].filters.gaussian_filter(:[args])</code> | 86 |
| | <code>:[x]].query(:[a]], :[b]], n_jobs=:[c])</code> | 0 | <code>:[x]].query(:[args], n_jobs=:[y])</code> | 0 |
| | <code>:[x]].hanning(:[a]], :[b]])</code> | 0 | <code>:[x]].hanning(:[args])</code> | 0 |
| numpy | <code>:[x]].alltrue(:[a], axis=:[b])</code> | 7 | <code>:[x]].alltrue(:[args])</code> | 208 |
| | <code>:[x]].histogram(:[a]], bins=:[b], range=:[c], normed=:[y])</code> | 2 | <code>:[x]].histogram(:[args], normed=:[y])</code> | 66 |
| | <code>:[x]].complex(:[a]], :[b]])</code> | 17 | <code>:[x]].complex(:[args])</code> | 20 |
| sklearn | <code>BaggingClassifier(base_estimator=:[a]], n_estimators=:[b]], random_state=:[c]])</code> | 26 | <code>BaggingClassifier(base_estimator=:[x], :[args])</code> | 220 |
| | <code>BaggingRegressor(base_estimator=:[a]], n_estimators=:[b]], random_state=:[c]])</code> | 7 | <code>BaggingRegressor(base_estimator=:[x], :[args])</code> | 116 |
| | <code>KMeans(n_clusters=:[a], init=:[b]], n_init=:[c]), algorithm='full')</code> | 0 | <code>KMeans(:[args], algorithm='full')</code> | 38 |
| | <code>AgglomerativeClustering(n_clusters=:[a], linkage=:[b], affinity=:[c])</code> | 4 | <code>AgglomerativeClustering(:[args], affinity=:[c])</code> | 28 |
| | <code>OneHotEncoder(sparse=:[aac]], categories=:[aan]), drop=:[aaz]])</code> | 0 | <code>OneHotEncoder(sparse=:[x], :[args])</code> | 66 |

RQ

| RQ4：ルールは利用プロジェクトのコード更新に有効か
– sklearn,pandas,scipyライブラリを使用しているプロジェクト

| Library | Total Projects | Affected Projects | Unique Rules | Rule Applications | Additional Warnings | Resolved Warnings | Additional Passing Tests | Additional Failures | Resolved Failures |
|--------------|----------------|-------------------|--------------|-------------------|---------------------|-------------------|--------------------------|---------------------|-------------------|
| sklearn | 20 | 10 | 6 | 27 | 9 | 598 | 2 | 1 | 1 |
| pandas | 20 | 10 | 4 | 23 | 0 | 44 | 7 | 81 | 7 |
| scipy | 20 | 6 | 5 | 23 | 0 | 266 | 0 | 1 | 0 |
| Total | 60 | 26 | 15 | 73 | 9 | 908 | 9 | 83 | 8 |

まとめ

| APIの破壊的変更や非推奨に対応するためのルールを生成するMELTを提案

- PRの情報を使用することで早期にAPI移行を実現
- 抽出/生成したコード例から変更ルールを作成

| 多数の警告エラーを解消可能