

Domain knowledge Matters Improving Prompts with Fix Templates for Repairing

Yun Peng
The Chinese University of
Hong Kong
Hong Kong, China
ypeng@cse.cuhk.edu.hk

Yintong Huo
The Chinese University of
Hong Kong
Hong Kong, China
ythuo@cse.cuhk.edu.hk

Shuzheng Gao
The Chinese University of
Hong Kong
Hong Kong, China
1155203205@link.cuhk.edu.hk

Cuiyun Gao*
Harbin Institute of
Technology
Shenzhen, China
gaocuiyun@hit.edu.cn

Michael R. Lyu
The Chinese University of
Hong Kong
Hong Kong, China
lyu@cse.cuhk.edu.hk

背景

Pythonの人気

- ・ 人工知能やデータサイエンスで多く利用される
- ・ 使用率は増加していつている

動的型システム

- ・ Pythonでは動的型システムが採用されている
 - ー変数の型は実行時に解決される
- ・ プログラムを記述する際の利便性が向上する

欠点として

実行時に多くの型エラーが発生する

型エラーの回避

静的アプローチ

- 静的型チェッカー(mypy)

- ー 型注釈を追加し, 静的に型の衝突をチェック

- 変数の型を静的に推論する

- ー 手動で型注釈を加える労力を軽減

問題: 型エラー修正の際にはあまり助けにならない

型エラーの自動修正

- ルールベースのアプローチ

- ー 9つのテンプレートといくつかの合成ルールを手動で定義

問題: 現実世界の型エラーを十分にカバーできない

最近の自動修正手法

ニューラル機械翻訳(NMT)ベースのアプローチ

- ・バグのあるコード行を正しいコード行に翻訳する

プロンプトベースのアプローチ (AlphaRepair)

- ・バグのあるコード行のいくつかのトークンをマスキングして事前学習されたモデルを呼び出し，トークンを予測
- ・このアプローチで最先端の性能を達成

問題：どのようにマスクするかは手動で設定

ドメイン固有の知識がないと複雑な型エラーを修正するのが困難

+

自動でドメイン知識を組み込むのも難しい

目的

研究の目的

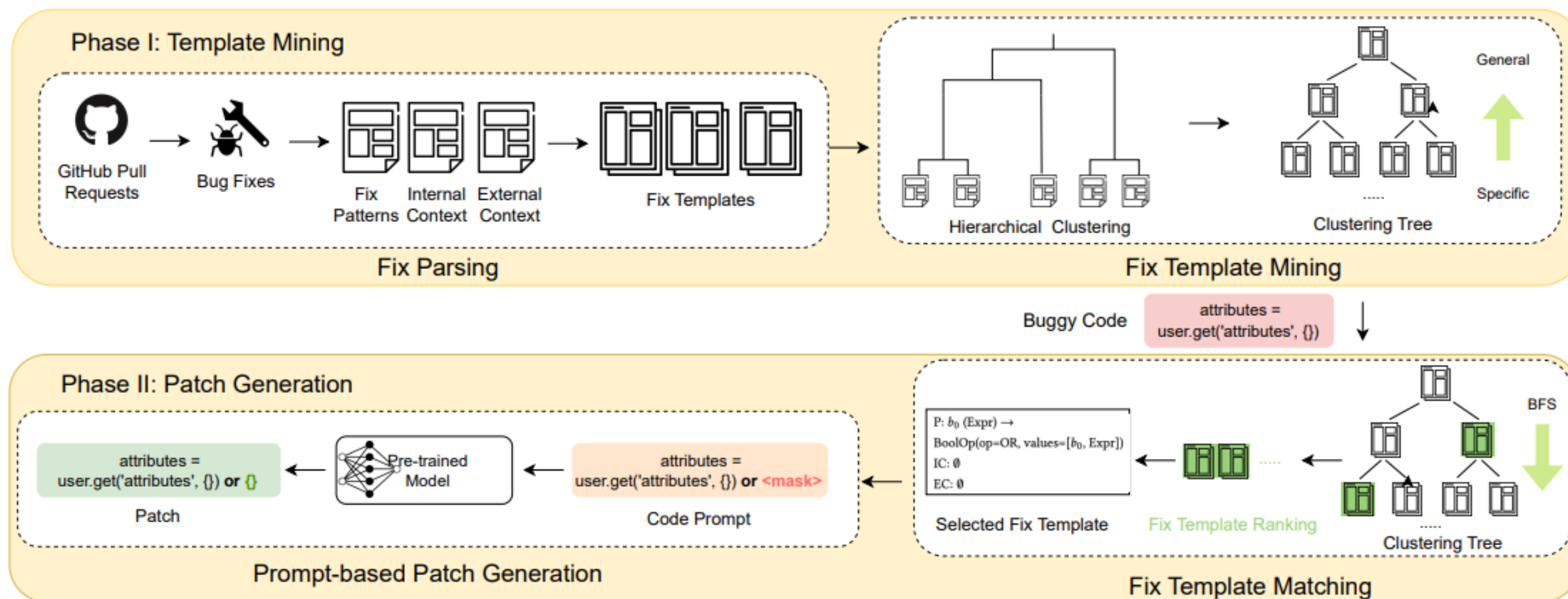
ドメイン知識を含めたプロンプトベースのアプローチを提案する

複雑な修正パターン

```
# Buggy Code: scrapy/scrapy:f042ad
if user:
    - user_pass = '%s:%s' % (unquote(user), unquote(password))
    + user_pass = to_bytes('%s:%s' % (unquote(user), unquote(password)))
      raise ValueError('Port cannot be 0 or less.')
      creds = base64.b64encode(user_pass).strip()
else:
    creds = None
```

提案手法

TypeFix 全体像



提案手法|マイニングフェーズ

目的

修正テンプレートの抽出

手順

修正解析・修正テンプレートマイニング

修正テンプレートの定義

バグのあるコードにマスクを追加する場所と方法を表現

Add, Remove, Insert, Replaceの4つに分類できる

修正パターン・内部コンテキスト・外部コンテキストで構成

修正テンプレート

修正パターン

型エラーを修正するためにバグのあるコードがどのように編集されたかを示す

内部コンテキスト

異なるレベルの型エラーを処理するために修正パターンを適用する位置を特定する

外部コンテキスト

バグのあるプログラム全体における内部コンテキストの位置を示す

これらはすべてテンプレートツリーに基づいて表現される

テンプレートツリー | 修正パターン

テンプレートツリー

- PythonのASTに基づいて定義される
- 元のASTノードタイプ t から8つの基本タイプに再分類(bt)

$bt \in \{\text{Variable, Op, Literal, Builtin, Type, Attribute, Expr, Stmt}\}$

ASTよりも高いレベルの抽象化を実現するために設計

- ほかにもID, 値といった情報を持つ

修正パターンの定義

バグのあるコードを修正済みのコードに変換する操作

内部・外部コンテキスト

内部コンテキスト構成要素

- ・ 修正パターンが適用される最も深いステートメントを表す
テンプレートツリー
- ・ 修正前後で変わらない部分と
置き換えられる部分および置き換え後の関係

外部コンテキスト構成要素

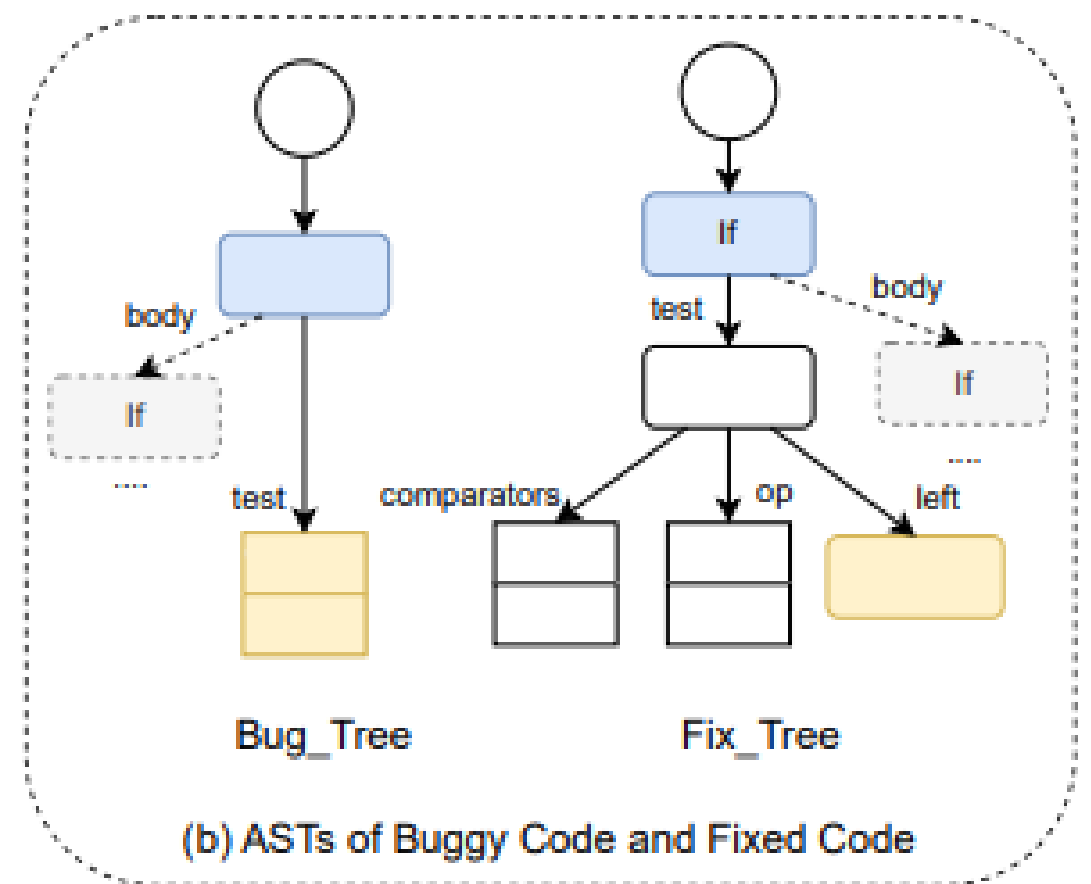
- ・ 内部コンテキストの前にあるステートメントのテンプレートツリー
- ・ 内部コンテキストの後にあるステートメントのテンプレートツリー

修正解析の例

```
if value_type in ('boolean', 'bool'):
    value = boolean(value, strict=False)
- elif value:
+ elif value is not None:
    if value_type in ('integer', 'int'):
```

変更された行を含むサブツリー

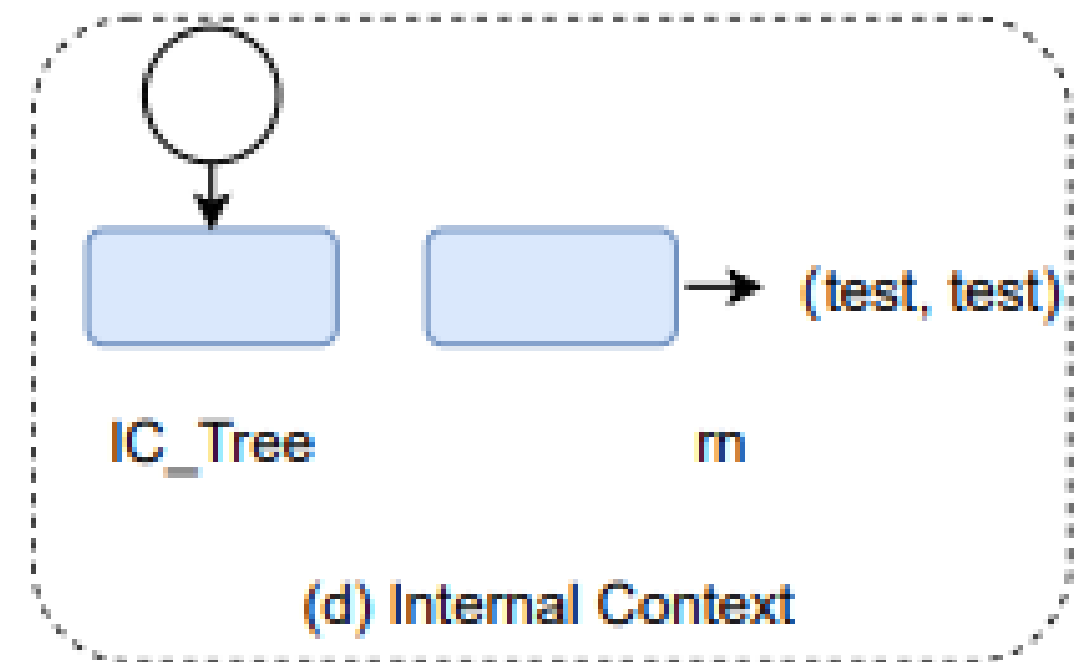
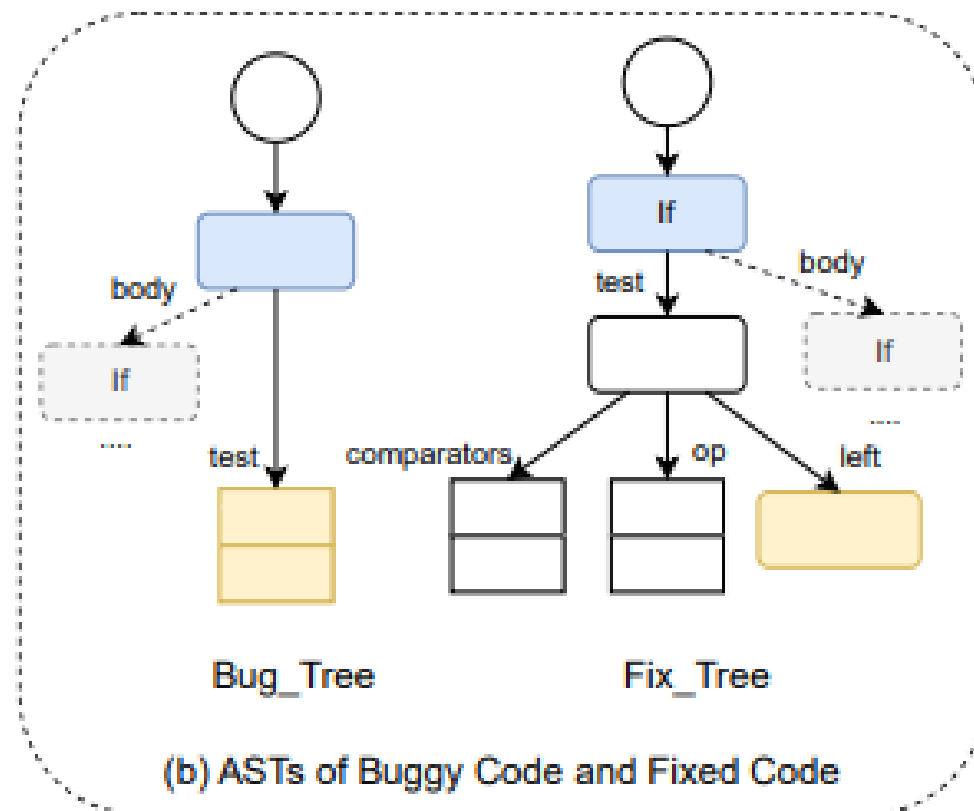
- どちらも同じルートノードを共有しているため式レベルの変更
- 共有しているノードを抽出して新たなテンプレートツリーを作成



修正解析の例

```
if value_type in ('boolean', 'bool'):  
    value = boolean(value, strict=False)  
- elif value:  
+ elif value is not None:  
    if value_type in ('integer', 'int'):
```

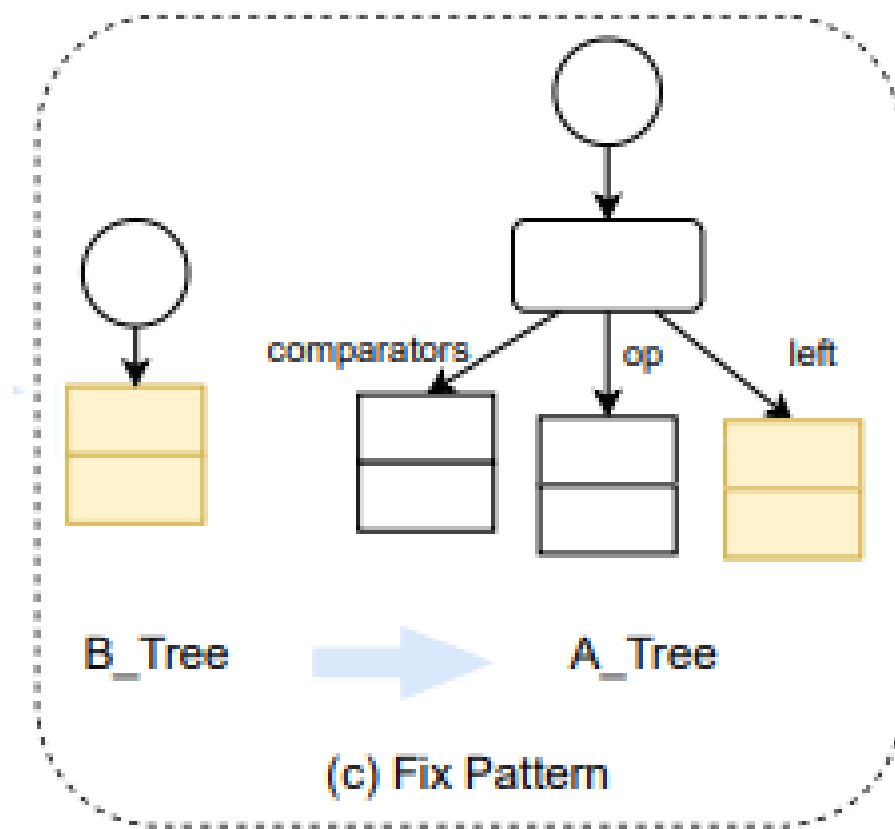
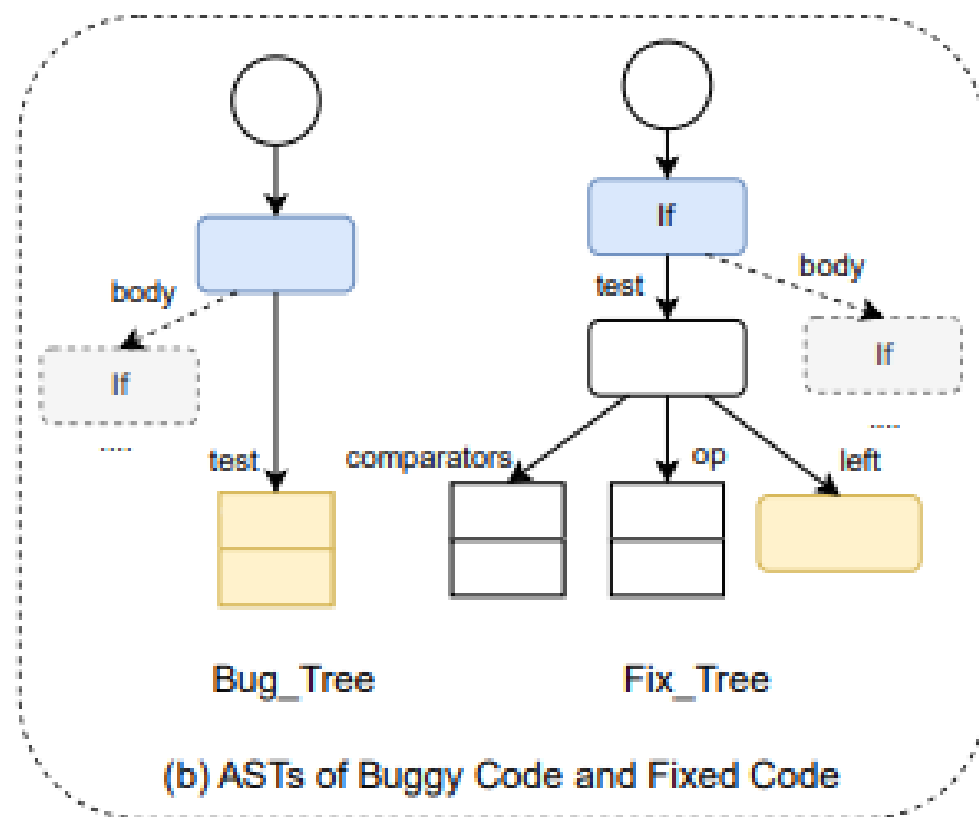
内部コンテキスト



修正解析の例

```
if value_type in ('boolean', 'bool'):  
    value = boolean(value, strict=False)  
- elif value:  
+ elif value is not None:  
    if value_type in ('integer', 'int'):
```

修正パターン

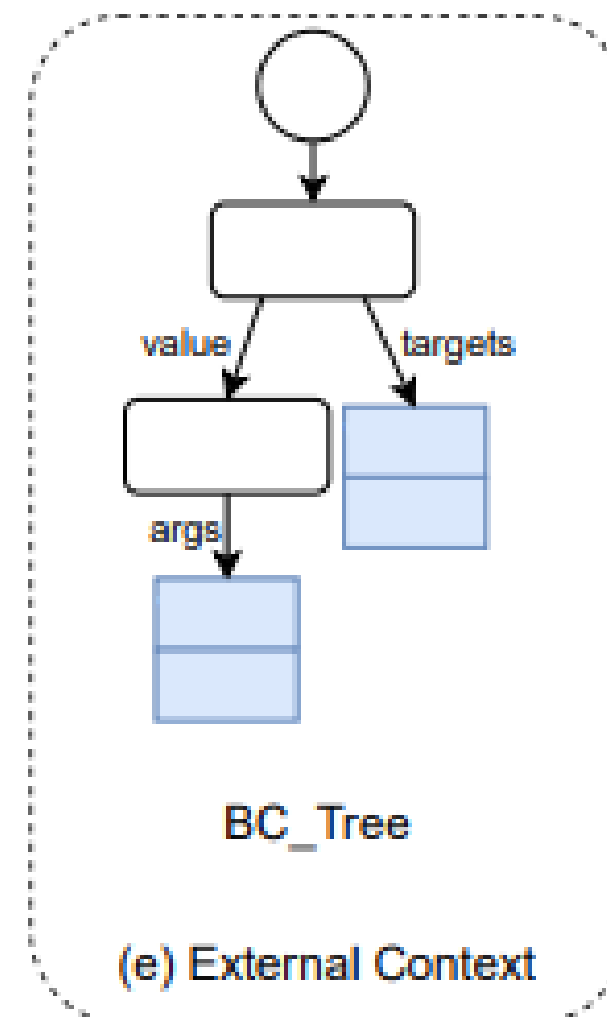


修正解析の例

外部コンテキスト

修正パターンと直接的なデータ依存関係を持つステートメント
(同じ変数を共有するステートメント)

```
if value_type in ('boolean', 'bool'):  
    value = boolean(value, strict=False)  
- elif value:  
+ elif value is not None:  
    if value_type in ('integer', 'int'):
```



修正テンプレートマイニング

階層的クラスタリング

- ・ 具体的な修正テンプレートを抽象化し、一般的なものに統合

類似度

2 種類の類似度指標

- ・ 構造の距離

2つのテンプレートツリーにおいて、

値に関係なく同じタイプを持つノードの比率

- ・ 値の距離

2つのテンプレートツリーにおいて

同じタイプと値を持つノードの比率

テンプレートの抽象化

抽象化の優先順位

- ・ 修正パターン・内部コンテキスト・外部コンテキストのいずれかを一度に抽象化する
- ・ 外部 > 内部 > 修正パターン の優先順位

抽象化のパターン

- ・ 同一ノード
完全に同一で、そのまま保持される
- ・ 値の抽象化
タイプが同じで値が異なる場合は値をABSトークンに設定したノードを作成
- ・ タイプの抽象化
基本タイプが同じでタイプ・値が異なる場合はタイプと値をABSトークンに
- ・ ノードの削除
共通の属性を持たない場合

パッチ生成フェーズ

2つのプロセスで構成

- ・ 修正テンプレートマッチング
- ・ プロンプトベースのパッチ生成

修正テンプレートマッチング

- ・ バグのあるプログラムに適用できる適切な修正テンプレートを
選択してランク付けすることを目的としている

プロンプトベースのパッチ生成

- ・ 選択した修正テンプレートを適用してコードプロンプトを生成し
事前学習されたコードモデルを呼び出すことで候補パッチを生成

修正テンプレートマッチング

テンプレートノードマッチ

- 2つのノードの値が一致
- 2つのノードの基本タイプが一致

テンプレートツリーマッチ

- ルートノードが一致
- ルートノードに連なる子ノードがすべて一致

修正テンプレートマッチ

- バグ行を含む最も深いステートメントおよびその前後の依存関係を持つステートメントを表現
- 外部コンテキスト， 内部コンテキスト， 修正パターンとそれぞれ一致

修正テンプレートのランク付け

第1段階

- ・ バグのある箇所に対して異なる修正方法を提供するテンプレートをグループにまとめる
- ・ 出現頻度に基づいて、修正テンプレートのランク付けを行う

第2段階

- ・ グループ間でランク付けを行う
- ・ グループのうち最も出現頻度が高いテンプレートの抽象化率を調べる（ABSトークンである割合）
- ・ 抽象化率が低いほどドメイン知識が豊富であると判断されランクが高くなる

プロンプトベースのパッチ生成

手順

- ダミーASTノードを修正テンプレートに追加
テンプレートマイニングの段階で削除された子ノードがある可能性
- バグのある部分と置換する
- ABSトークンをマスクとして含むソースコードにする
- マスクの予測とパッチ生成
CodeT5を呼び出して各マスクトークンを予測する

評価実験

結果

Approach	TYPEBUGS		BUGSINPY	
	#Unique	Coverage	#Unique	Coverage
TYPEFIX	24	83 (76.1%)	16	40 (74.1%)
PyTER	10	46 (42.2%)	5	18 (33.3%)

より多くの種類の型エラーを修正することができた

評価実験

生成できなかったもの

```
1 #Type Error 1: apache/airflow:892d4d
2 if conf.getboolean('core', 'store_dag_code',\
3 fallback=False):
4     - DagCode.bulk_sync_to_db([dag.fileloc for dag in orm_dag])
5     + DagCode.bulk_sync_to_db([orm_dag.fileloc])
6 #Type Error 2: pandas-dev/pandas:a3e903
7 elif (is_extension_array_dtype(left) or\
8     - is_extension_array_dtype(right)):
9     + (is_extension_array_dtype(right) and not is_scalar(right)):
10     return dispatch_to_extension_op(op, left, right)
```

学習データを増やすことで対応ができる

総括

Pythonの型エラーを修正するドメイン知識を取り入れたプロンプトベースのアプローチが提案された

従来手法よりも多くの型エラーを修正できるようになった

