G-9

不具合を含むソースコードの生成に向けたLLVM IRからの多様なC言語ソースコードの生成

A Study on the Variaty of Source Code Generation via LLVM IR for Generating Buggy Source Code

○須川優斗[1]，村上知里[2]

*Yuto Sugawa[1], Chisato Murakami[2]

Abstract: In recent years, the use of machine learning for software bug detection has been increasing. It is essential to provide diverse and unbiased training source code to develop more general-purpose classifiers. One approach to providing training data is using of generation techniques with machine learning. This study discussed source code generation using translation models based on intermediate representations. We examined how the learning conditions of the translation models affect the accuracy and diversity of the generated source code.

１．はじめに

近年，機械学習によるソフトウェアの不具合判別技術が増加している．正常なソースコードは，GitHubなどから入手できる．しかし，学習用データとなる不具合を含むソースコードは，不具合を含まないソースコードに比べると，入手が難しい．高精度の判別器を作成するためには，さまざまな不具合について偏りがない学習用データを確保する必要がある．また，同一の仕様を満たすソースコードは，複数の表現で記述することができる．現実的な不具合検出の利用を考えれば，ソースコード表現に揺らぎを持つ学習用データであることが望ましい．そこで我々は，任意の不具合を含み，かつ，表現に揺らぎを持つソースコード生成器の開発を行っている．提案する生成器は，入力データである中間表現に不具合を注入し，不具合を含む中間表現から不具合を含むソースコードを生成する．中間表現はソースコードの構文情報を持つテキストデータであり，不具合を含まない中間表現はソースコードと可逆である．また，中間表現はプログラミング言語に依存しないため，さまざまな言語への適用が期待できる．

本検討では，中間表現にLLVM IR，ソースコードにC言語で記述されたファイルを採用した．提案する生成器の開発に向けた最初のステップとして，不具合を含まないLLVM IRからCファイルを生成することを目的とした．LLVM IRから，①対となるCファイルを復元生成できること，②表現の揺らぎを持つ多様なファイルを生成できることが求められる．今回は，生成モデルの学習条件を変更した際に①と②へ与える影響を検討した．

２．原理

2.1 Sequence-to-sequence (Seq2seq)

Seq2seqモデル[1]は，あるデータ列（入力データ）を別のデータ列（ターゲットデータ）に変換するモデルである．このモデルはEncoderとDecoderの2つの機構により構成される．本検討では，EncoderとDecoderを順方向の Gated Recurrent Unit (GRU) で構築し，Attentionを使用した．Figure 1は構築したモデルの概要である．

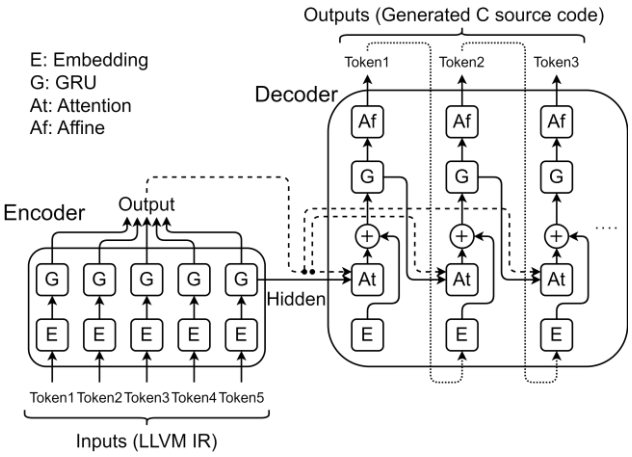

Figure 1. Our model architecture

2.2 Teacher forcing

通常の学習では，DecoderにEncoderの前ステップの出力を入力する．Teacher forcingを適用すると，一定の確率でターゲットデータを入力する．学習の初期段階における誤りの蓄積を防ぎ，構文の関係を効率的に学習することが期待できる．

３．手法

3.1 データセットの準備

CファイルをCodeflaws[2]から入手し，実行可能なファイルに前処理を行い，110 件のデータを用意した．LLVM IRは，前処理を行ったCファイルからClangで生成した．110対のデータのうち，100対を学習用データ，10対をテスト用データとした．各データは記号区切りでトークン化した．

1：日大理工・院（前）・情報　2：日大理工・教員・情報

3.2 モデルの学習

入力データを LLVM IR，ターゲットデータを C ファイルとして学習を行った．学習は 1000 epoch 行い，100 epoch の学習ごとに，学習したモデルを保存した．入力データは前詰め，ターゲットデータは後ろ詰めでパディングした．Decoder 学習時の Teacher forcing (TF) 適用率を 0 %, 50 %, 100 %に変化させ，3 種類の学習条件（TF 0 %, TF 50 %, TF 100 %）を用意した．

4.3 評価

各学習条件で最小損失であったモデルを選択した．選択したモデルにテスト用データである 10 件の LLVM IR を入力し，C ファイルを生成した．生成した C ファイルを，一致性と多様性の観点で評価した．

一致性では，学習条件ごとに，入力した LLVM IR の対である C ファイルと生成した C ファイルについて，先頭から連続して一致したトークン数を算出した．ここで，変数名が異なっていても語順が一致していれば，一致とした．

多様性では，テスト用データ 1 件の LLVM IR から C ファイルを 100 回生成し，学習条件ごとに 1000 件の結果を用意した．まず，単一の LLVM IR から生成した C ファイルが表現の揺らぎを持つことを確認するために，多様な C ファイルの数を算出した．次に 1000 件のうち，次の 3 項目 (1) コンパイル可能である，(2) 入力した LLVM IR の対である C ファイルから作成したテストを通過する，(3) 入力した LLVM IR の対である C ファイルと生成した C ファイルは同じ語順である，に当てはまるファイル数を算出した．

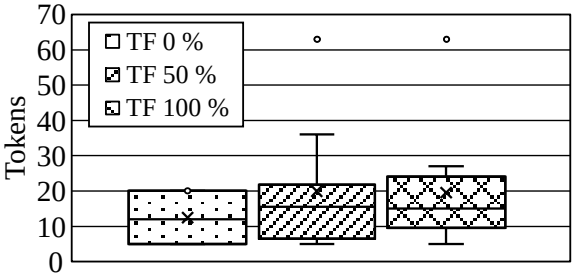

Figure 2. Consistency evaluation

```
# include < stdio.h > int main ( void ) { int y ; scanf ( " % d " , & y ) ; if ( y % 2 = = 0 & & y ! = 2 ) { printf ( " yes " ) ; } else { printf ( " no " ) ; } return 0 ; }
```

Figure 3. A generated C source code

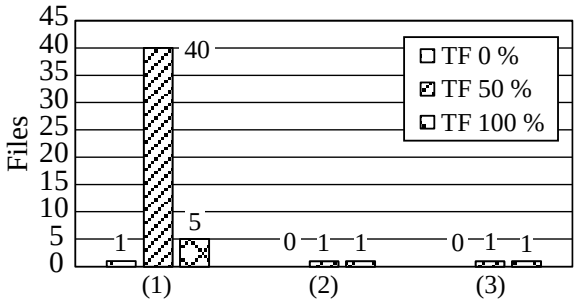

Figure 4. Variety evaluation

４．結果と考察

一致性の結果を Figure 2 に示す．Figure 2 から，TF が 50 %以上の場合，平均で 20 トークン程度，最大で 63 トークンを正しく生成することができた．TF 100 %の場合に 63 トークンを復元した C ファイルを Figure 3 に示す．しかし，最小は 5 トークンであり，最大と最小のトークン数で大きな差が発生した．

多様性の結果を Figure 4 に示す．各学習条件における多様な C ファイルの数は，TF 0 % : 1000 件，TF 50 % : 691 件，TF 100 % : 673 件であった．Figure 4 (1) から，TF 50 %および 100 %の場合に，40 件および 5 件のコンパイル可能なソースコードが生成することができた．しかし (2) および (3) では，TF 50 %と TF 100 %のそれぞれで，1 件ずつの生成であった．

本検討では，Teacher forcing を適用した場合の一致性と多様性に対する影響を定量評価することができた．いずれの評価でも，Teacher forcing は 50 %以上が望ましいと言える．一方で，コンパイル可能で多様な C ファイルを生成する割合を高める必要がある．

５．おわりに

我々の最終目標は，任意の不具合を含み，かつ，表現に揺らぎを持つソースコード生成器を開発することである．本検討では，最初のステップとして，不具合を含まない LLVM IR から C ファイルを生成することを目的とした．C ファイルの生成モデルの学習時に Teacher forcing を変化させ，1 件の LLVM IR から，①対となる C ファイルを復元生成できること，②表現の揺らぎを持つ多様なファイルを生成できること，への影響を評価した．その結果，Teacher forcing が 50 %以上の場合に，①と②を達成することができた．しかし達成割合が低いため，今後は，結果を生成モデルにフィードバックする機構を追加する予定である．

６．参考文献

[1] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” 2014, *arXiv*: *1409.3215*.

[2] Shin Hwei Tan, Jooyong Yi, Yulis, S. Mechtaev, and A. Roychoudhury, “Codeflaws: A programming competition benchmark for evaluating automated program repair tools,” in *Proc. 2017 IEEE/ACM 39th Int. Conf. Softw. Eng. Companion (ICSE-C)*, pp. 180–182, May 2017, doi: 10.1109/ICSE-C.2017.76.