神奈川大学理学部　情報科学科　馬谷研究室

Lox 言語の for 文の実装方式に関する考察

篠田彪雅 201803379

1　はじめに

本調査では、Lox 言語における desugaring について調査を行う。インタープリタでは、プログラムの実行時に制御フローを解析し、文の実行順序や条件分岐、ループの挙動を理解することが重要である。特に、for 文の実行においては、その制御フローが while 文に変換されてからインタープリタによって解釈されている点に着目する。

2　本研究の目的

while 文は、インタープリタ内部で直接解釈されるが、for 文においては desugaring と呼ばれる手法が用いられている。for 文は while 文に変換され、その後インタープリタによって解釈されるため、for 文の制御フローを解析する際には、実際には変換された while 文の制御フローを理解する必要がある。本論文では、具体的に for 文の desugaring 手法に着目し、なぜこのような変換が行われるのかについて考察する。

3　desugaring による for 文の実装

desugaring は、syntactic sugar として提供されている高レベルな構文を、低レベルな基本的な構文に変換することである。Lox では、for ループを使わずに既存の文を使用して同じ処理を行うことができる。ここでは desugaring を使用し for 文の実装がどのように行われているかを明らかにする。

```
private Stmt forStatement() {
...
  if (increment != null) {
    body = new Stmt.Block(
        Arrays.asList(
            body,
            new Stmt.Expression
            (increment)));
  }
  if (condition == null)
  condition = new Expr.Literal(true);
  body = new Stmt.While(condition, body);
  if (initializer != null) {
    body = new Stmt.Block
    (Arrays.asList(initializer, body));
  }
  return body;
}
```

上記は desugaring による for 文実装について解説している。desugaring の実行は、増分子がある場合は、ループ本体の後ろに増分子を評価する式文を追加する。新しいブロック文は、元のループ本体 body と増分子を評価する式文を順に実行するようになる。条件式が省略された場合は、常に真を表すリテラル式として置き換える。これにより、while ループの条件として真のリテラル式が使われるようになる。初期化子がある場合は、ループ全体の前に初期化子を実行するブロックを追加する。この新しいブロック文は、初期化子を実行した後に元のループ本体 body を実行するようになる。これより for ループの各節を解析して得られた情報を使って desugaring が行われる。最終的に、desugaring されたループ本体が得られることになる。while 文は抽象構文木の WhileStmt と BlockStmt の組み合わせとして表現される。この抽象構文木を辿ることで、for 文の挙動を再現することができる。

```
@Override
 public Void visitWhileStmt
 (Stmt.While stmt) {
   while (isTruthy(evaluate
   (stmt.condition))) {
     execute(stmt.body);
```

```
      }
      return null;
    }
  @Override
    public Void visitBlockStmt
    (Stmt.Block stmt) {
      executeBlock(stmt.statements,
      new Environment(environment));
      return null;
    }
```

上記のコードは lox の while 文の解説である。WhileStmt がループ条件を表し、BlockStmt がループ本体を表します。for 文の初期化部分は抽象構文木で assignexpr として表現されている。block 文は抽象構文木のブロック文を評価するためのものです。executeBlock メソッドを呼び出して、ブロック内の文を順番に実行します。また、新しい環境 environment を作成し、その環境内でブロック文内の文を順に実行していきます。このメソッドは抽象構文木の while 文と block 文を評価するためのものである。while 文は条件式を持ち、条件式が true である間はループ本体を繰り返し実行します。evaluate メソッドを使って stmt の条件式を評価し、その結果が true の場合はループ本体 stmt.body を実行します。条件式が false になると、ループから抜ける。

4 直接解釈による for 文の実装

for 文を実装するには、抽象構文木に新しいノードタイプを追加し、それに対応する抽象構文木の Visitor のメソッドを実装する必要である。Lox 言語のインタープリタは、抽象構文木を実行することによってプログラムを実行する。抽象構文木に含まれるノードを順番に評価・実行することで、プログラムの制御フローや処理を行う。

```
@Override
public Void visitForStmt(Stmt.For stmt)
{
  if (stmt.initializer != null) {
    execute(stmt.initializer);
  }
  for (; ;) {
    boolean condition = true;
    if (stmt.condition != null) {
      condition = isTruthy(
      evaluate(stmt.condition));
    }
    if (!condition) {
      break;
    }
    execute(stmt.body);
    if (stmt.increment != null) {
      evaluate(stmt.increment);
    }
  }

  return null;
}
```

このコードは for 文の実行を行うためのメソッドの一部です。このメソッドは抽象構文木をトラバースして、for 文を実行する際の動作を定義している。for 文の条件式が null の場合は常に true として評価されるようになっている。これにより、for(;;) のような無限ループを表現することができる。visitforstmt メソッドは stmt.for 型の抽象構文木ノードを受け取る。そして for 文の初期化子, 条件式, 増分子, ループ本体を評価・実行できる。

5 まとめ

for 文は for を直接解釈することも可能だが基本的な構文を使用することにより既に構築された抽象構文木を評価・実行する際に While 文の制御フローを再現して for 文を実行する方が言語処理系はよりシンプルかつ直接的な構文を扱うことができ、開発者はより理解しやすいコードを書くことができる。今後は、他の制御構造との比較やさらなる最適化の検討を行い、Lox 言語の性能向上に向けた研究を考えていきたい。

参考文献

[1] bob nystrom, craftinginterpreters a tree-walk interpreter
http://craftinginterpreters.com/a-tree-walk-interpreter.html