RefSyn の手動解析記録

1 対象関数: append

現在のノードを先頭とする連結リストの末尾に、新しいノードを追加するメソッド。引数として渡された値を val フィールドにもつノードを新たに作成し、それをリストの最後に接続。

1.1 初期プログラム

```
class Node {
  append(arg) { /* TBD */ }
}
var lst = new Node();
lst.val = 2;
lst.append(0);
lst.append(3);
```

1.2 操作ログ

1.2.1 lst.append(0) に対する操作

- addNode: __temp1, Node, false, undefined
- addNode: __temp2, 0, true, string
- addEdge: __temp1, __temp2, val
- addEdge: main-new1, __temp1, next

1.2.2 lst.append(3) に対する操作

- addNode: __temp3, Node, false, undefined
- addNode: __temp4, 3, true, string
- addEdge: __temp3, __temp4, val
- addEdge: temp1, __temp3, next

1.3 手動で復元したコード

1.3.1 lst.append(0)

```
var temp1 = new Node();
var temp2 = 0;
temp1.val = temp2;
this.next = temp1;
```

1.3.2 lst.append(3)

```
var temp1 = new Node();
var temp2 = 3;
temp1.val = temp2;
this.next.next = temp1;
```

1.4 一般化と部分関数の推定

```
var temp1 = new Node();
var temp2 = f();     // f() = 引数 arg を返す関数
temp1.val = temp2;
g().next = temp1;    // g() = 現在のリストの末尾を返す関数
```

1.5 想定の合成された関数

```
append(arg) {
  var temp1 = new Node();
  function f() {
    return arg;
  }
  var temp2 = f();
  temp1.val = temp2;
  function g() {
    if(!this.next) {
      return this;
    } else {
      this.next.g();     // できないけど
    }
  }
  g().next = temp1;
}
```

1.6 oracle の関数

```
append(arg) {
  if(!this.next) {
    this.next = new Node();
    this.net.val = arg;
  } else {
    this.next.append(arg);
  }
}
```

2 対象関数: prepend

現在のノードを先頭とする連結リストの先頭に、新しいノードを追加するメソッド。引数として渡された値を val フィールドにもつノードを新たに作成し、それをリストの先頭に接続。

2.1 初期プログラム

```
class Node {
  prepend(arg) { /* TBD */ }
}
var lst = new Node();
lst.val = 2;
lst = lst.prepend(0);
lst = lst.prepend(3);
```

2.2 操作ログ

2.2.1 lst.prepend(0) に対する操作

- addNode: __temp1, Node, false, undefined
- addNode: __temp2, 0, true, string
- addEdge: __temp1, __temp2, val
- addEdge: __temp1, main-new1, next
- addVariable: __temp1, return

2.2.2 lst.prepend(3) に対する操作

- addNode: __temp3, Node, false, undefined
- addNode: __temp4, 3, true, string

- `addEdge`: __temp3, __temp4, val
- `addEdge`: __temp3, __temp1, next
- `addVariable`: __temp3, return

2.3 手動で復元したコード

2.3.1 `lst.prepend(0)`

```
var temp1 = new Node();
var temp2 = 0;
temp1.val = temp2;
temp1.next = this;
return temp1;
```

2.3.2 `lst.prepend(3)`

```
var temp3 = new Node();
var temp4 = 3;
temp3.val = temp4;
temp3.next = this;
return temp3;
```

temp1 を this にすることは実行時情報から判断できる？

2.4 一般化と部分関数の推定

```
var temp1 = new Node();
var temp2 = f();     // f() = 引数 arg を返す関数
temp1.val = temp2;
temp1.next = this;
return temp1;
```

2.5 想定の合成された関数

```
prepend(arg) {
  var temp1 = new Node();
  function f() {
    return arg;
  }
  var temp2 = f();
  temp1.val = temp2;
  temp1.next = this;
  return temp1;
}
```

2.6 oracle の関数

```
prepend(arg) {
  var temp1 = new Node();
  temp1.val = arg;
  temp1.next = this;
  return temp1;
}
```

3 対象関数: removeLast

現在のノードを先頭とする連結リストの最後尾のノードを消すメソッド。

3.1 初期プログラム

```
class Node {
    removeLast() { /* TBD */ }
}
var lst = new Node();
lst.val = 2;
lst.next = new Node();
lst.next.val = 0;
lst.next.next = new Node();
lst.next.next.val = 3;
lst.removeLast();
lst.removeLast();
```

3.2 操作ログ

3.2.1 lst.removeLast() に対する操作

- deleteNode: main-new3

3.2.2 lst.removeLasr() に対する操作

- deleteNode: main-new2

next とか val とかも消しても良い
Node だけ消せば可視化されない

3.3 手動で復元したコード

3.3.1 lst.removeLast()

```
this.next.next = undefined
```

3.3.2 lst.removeLast()

```
this.next = undefined
```

next 先の Node が消えた
null ではなく、undefined
constructor があるなら null でよい

3.4 一般化と部分関数の推定

```
f().next = undefined;    // f() = 現在のリストの末尾を返す関数
```

3.5 想定の合成された関数

```
removeLast() {
  function f() {
    if(!this.next.next) {
      return this.next;
    } else {
      this.next.g();      // できないけど
    }
  }
  f().next = undefined;
}
```

3.6 oracle の関数

```
removeLast() {
  if(this.next.next === undefined) {
    this.next = undefined;
  } else {
    this.next.removeLast();
  }
}
```

4 対象関数: removeFirst

現在のノードを先頭とする連結リストの先頭のノードを消すメソッド。

4.1 初期プログラム

```
class Node {
    removeFirst() { /* TBD */ }
}
var lst = new Node();
lst.val = 2;
lst.next = new Node();
lst.next.val = 0;
lst.next.next = new Node();
lst.next.next.val = 3;
lst = lst.removeFirst();
lst = lst.removeFirst();
```

4.2 操作ログ

4.2.1 lst.removeFirst() に対する操作

- addVariable: main-new2, return

4.2.2 lst.removeFirst() に対する操作

- addVariable: main-new3, return

4.3 手動で復元したコード

4.3.1 lst.prepend(0)

```
return this.next;
```

ここの変換は非自明？

4.3.2 lst.prepend(3)

```
return this.next;
```

4.4 一般化と部分関数の推定

```
return this.next;
```

4.5 想定の合成された関数

```
removeFirst() {
  return this.next;
}
```

4.6 oracle の関数

```
removeFirst() {
  return this.next;
}
```

5 対象関数: removeVal(arg)

連結リスト内の val フィールドが arg の値を持つノードを削除するメソッド。

5.1 初期プログラム

```
class Node {
  removeVal(arg) { /* TBD */ }
}
var lst = new Node();
lst.val = 2;
lst.next = new Node();
lst.next.val = 0;
lst.next.next = new Node();
lst.next.next.val = 3;
lst.next.next.next = new Node();
lst.next.next.next.val = 1;
lst = lst.removeVal(0);
lst = lst.removeVal(1);
```

5.2 操作ログ

5.2.1 lst.removeVal(0) に対する操作

- editEdgeReference: main-new1, main-new2, main-new3, next
- addVariable: main-new1, return

5.2.2 lst.removeVal(1) に対する操作

- deleteEdge: main-new3, main-new4, next
- addVariable: main-new1, return

5.3 手動で復元したコード

5.3.1 lst.removeVal(0)

```
this.next = this.next.next;
return this;
```

5.3.2 lst.removeVal(1)

```
this.next.next = null;
return this;
```

5.4 一般化と部分関数の推定

```
f().next = g()
// f は val フィールドに arg をもつノードの一個前
// g は val フィールドに arg をもつノードの一個後
```

5.5 想定の合成された関数

```
removeVal(arg) {
  f().next = g();
  return this;
  f() {
    if(this.val === arg) {

    }
  }
  g() {
    if(this.val === arg) {

    }
  }
}
```

5.6 oracle の関数

```
removeVal(arg) {
  if(this.val === arg) {
    return this.next;
  }
  if(this.next !== null) {
    this.next = this.next.removeval(arg);
  }
  return this;
}
```

6 対象関数: removeAt(i)

連結リストの i 番目のノードを削除するメソッド。

6.1 初期プログラム

```
class Node {
  removeAt(i) { /* TBD */ }
}
var lst = new Node();
lst.val = 2;
lst.next = new Node();
lst.next.val = 0;
lst.next.next = new Node();
lst.next.next.val = 3;
lst.next.next.next = new Node();
lst.next.next.next.val = 1;
lst.removeAt(2);
lst.removeAt(1);
```

6.2 操作ログ

6.2.1 `lst.removeAt(2)` に対する操作

- `editEdgeReference`: main-new2, main-new3, main-new4, next
- `addVariable`: main-new1, return

6.2.2 `lst.removeAt(1)` に対する操作

- `editEdgeReference`: main-new1, main-new2, main-new4, next
- `addVariable`: main-new1, return

6.3 手動で復元したコード

6.3.1 `lst.append(0)`

```
this.next.next = this.next.next.next;
return this;
```

6.3.2 `lst.append(3)`

```
this.next = this.next.next;
return this;
```

6.4 一般化と部分関数の推定

```
f().next = g();
\\ f() = i-1 番目のノードを返す関数
\\ g() = i+1 番目のノードを返す関数
return this;
```

6.5 想定の合成された関数

```
removeAt(i) {
  f().next = g();
  return this;
  \\ f() = i−1 番目のノードを返す関数
  \\ g() = i+1 番目のノードを返す関数
}
```

6.6 oracle の関数

```
removeAt(i) {
  if(i=== 0) {
    return this.next;
  }
  this.next = this.next.remove(i - 1);
  return this;
}
```

7 対象関数: insertAfter(i, arg)

連結リストの i 番目の後ろに、val フィールドに arg を持つ新しいノードを追加するメソッド。

7.1 初期プログラム

```
class Node {
  insertAfter(i, arg) { /* TBD */ }
}
var lst = new Node();
lst.val = 2;
lst.next = new Node();
lst.next.val = 0;
lst.insertAfter(0,3);
lst.insertAfter(1,4);
```

7.2 操作ログ

7.2.1 `lst.insertAfter(0,3)` に対する操作

- `addNode`: __temp1, Node, false, undefined
- `addNode`: __temp2, 3, true, string
- `addEdge`: __temp1, __temp2, val
- `addEdge`: __temp1, main-new2, next
- `editEdgeReference`: main-new1, main-new2, __temp1, next

7.2.2 `lst.insertAftere(1,4)` に対する操作

- `addNode`: __temp3, Node, false, undefined
- `addNode`: __temp4, 4, true, string
- `addEdge`: __temp3, __temp4, val
- `addEdge`: __temp3, main-new2, next
- `editEdgeReference`: __temp1, main-new2, __temp3, next

7.3 手動で復元したコード

7.3.1 `lst.insertAfter(0,3)`

```
var temp1 = new Node();
var temp2 = 3;
temp1.val = temp2;
temp1.next = this.next;
this.next = temp1;
```

7.3.2 `lst.insetAfter(1,4)`

```
var temp1 = new Node();
var temp2 = 4;
temp1.val = temp2;
temp1.next = this.next.next;
this.next.next = temp1;
```

7.4 一般化と部分関数の推定

```
var temp1 = new Node();
var temp2 = f();         // f() = 引数 arg を返す関数
temp1.val = temp2;
temp1.next = g()     // g() = i+1 番目のノードを返す関数
h().next = temp1;    // h() = i-1 番目のノードを返す関数
```

7.5 想定の合成された関数

```
insertAfter(i,arg) {
  var temp1 = new Node();
  var temp2 = arg;
  temp1.val = temp2;
  temp1.next = find(i+1);
  find(i-1).next = temp1;
}
```

7.6 oracle の関数

```
insertAfter(i, arg) {
  function find(node, index) {
    if (node === null) {
      return null;
    }
    if (index === 0) {
      return node;
    }
    return find(node.next, index - 1);
  }
  const targetNode = find(this, i);
  if (targetNode !== null) {
    const newNode = new Node();
    newNode.val = arg;
    newNode.next = targetNode.next;
    targetNode.next = newNode;
  }
}
```

8 対象関数: concat(lst)

現在のノードを先頭とする連結リストの末尾に、別の連結リストを追加するメソッド。

8.1 初期プログラム

```
class Node {
  concat(otherList) { /* TBD */ }
}
var lst = new Node();
lst.val = 2;
lst.next = new Node();
lst.next.val = 0;
lst.next.next = new Node();
lst.next.next.val = 3;
var list = new Node();
list.val = 1;
list.next = new Node();
list.next.val = 4;
var l = new Node();
l.val = 5;
lst.concat(list);
lst.concat(l);
```

8.2 操作ログ

8.2.1 `lst.concat(list)` に対する操作

- addEdge: main-new3, main-new4, next

8.2.2 `lst.concat(l)` に対する操作

- addNode: main-new5, main-new6, next

8.3 手動で復元したコード

8.3.1 `lst.concat(list)`

```
this.next.next = list;
```

8.3.2 `lst.concat(l)`

```
this.next.next.next.next.next = l;
```

8.4 一般化と部分関数の推定

```
f().next = g();
\\ f() = 最後のノードを返す関数
\\ g() = 引数を返す関数
```

8.5 想定の合成された関数

```
concat(otherList) {
  f().next = otherList;
  \\ f() = の最後のノードを返す関数this
}
```

8.6 oracle の関数

```
concat(otherList) {
  if(!this.next) {
    this.next = otherList;
  } else {
    this.next.concat(otherList);
  }
}
```

9 対象関数: set(i,arg)

連結リストの i 番目の値を arg に変更するメソッド。

9.1 初期プログラム

```
class Node {
  set(i,arg) { /* TBD */ }
}
var lst = new Node();
lst.val = 2;
lst.next = new Node();
lst.next.val = 0;
lst.next.next = new Node();
lst.next.next.val = 3;
lst.next.next.next = new Node();
lst.next.next.next.val = 1;
lst.set(2,5);
lst.set(1,4);
```

9.2 操作ログ

9.2.1 lst.set(2,5) に対する操作

- addNode: __temp1, 5, true, string
- editEdgeReference: main-new3, main-new3-val, __temp1, val

9.2.2 lst.set(1,4) に対する操作

- addNode: __temp2, 4, true, string
- addEdge: main-new2, main-new2-val, __temp2, val

9.3 手動で復元したコード

9.3.1 lst.set(2,5)

```
var temp1 = 5;
this.next.next.val = temp1;
```

9.3.2 lst.set(1,4)

```
var temp1 = 4;
this.next.val = temp2;
```

9.4 一般化と部分関数の推定

```
var temp1 = f();     \\ f() = 引数 arg を返す関数
g().val = temp1;     \\ g() = 番目のノードを返す関数i
```

9.5 想定の合成された関数

```
set(i,arg) {
  var temp1 = arg;
  find(i).val = temp1;
}
```

9.6 oracle の関数

```
set(i,arg) {
  if(i === 0) {
    this.val = arg;
  }
  if(i > 0) {
    this.next.set(i-1,arg);
  }
}
```

10　対象関数: swap(i, j)

現在のノードを先頭とする連結リストの i 番目と j 番目の要素を交換するメソッド。

10.1　初期プログラム

```
class Node {
  swap(i,j) { /* TBD */ }
}
var lst = new Node();
lst.val = 2;
lst.next = new Node();
lst.next.val = 0;
lst.next.next = new Node();
lst.next.next.val = 3;
lst.next.next.next = new Node();
lst.next.next.next.val = 1;
lst.swap(0, 3);
lst.swap(1,2);
```

10.2　操作ログ

10.2.1　lst.swap(0,3) に対する操作

- addNode: __temp1, 2, true, string
- addNode: __temp2, 1, true, string
- editEdgeReference: main-new1, main-new1-val, __temp1, val
- editEdgeReference: main-new4, main-new4-val, __temp2, val

10.2.2　lst.swap(1,2) に対する操作

- addNode: __temp3, 3, true, string
- addNode: __temp4, 0, true, string
- editEdgeReference: main-new2, main-new2-val, __temp3, val
- editEdgeReference: main-new3, main-new3-val, __temp4, val

10.3　手動で復元したコード

操作列だけみると実際には違う挙動をするプログラムが出てくる参照元がなくなったが使うものは先に temp とかで変数宣言しておく？

10.3.1 lst.swap(0,3)

```
var temp1 = 2;
var temp2 = 1;
this.val = temp1;
this.next.next.next.val = temp2;
```

10.3.2 lst.swap(1,2)

```
var temp1 = 3;
var temp2 = 0;
this.next.val = temp1;
this.next.next.val = temp2;
```

10.4 一般化と部分関数の推定

```
var temp1 = f();     \\ f() = i 番目のノードの val フィールド
の値を返す関数
var temp2 = g();     \\ g() = j 番目のノードの val フィールド
の値を返す関数
h().val = temp1;     \\ h() = j 番目のノードを返す関数
i().val = temp2;     \\ i() = i 番目のノードを返す関数
```

10.5 想定の合成された関数

```
swap(i,j) {
  var temp1 = getVal(i);
  var temp2 = getVal(j);
  getNode(j).val = temp1;
  getNode(i).val = temp2;
}
```

10.6 oracle の関数

```
swap(i,j) {
  function find(node, index) {
    if(index === 0) {
      return node;
    }
    return find(node.next, index - 1);
  }
  var nodeI = find(this, i);
  var nodeJ = find(this, j);
```

```
  var tempVal = nodeI.val;
  nodeI.val = nodeJ.val;
  nodeJ.val = tempVal;
}
```