语法分析程序测试方案

测试目标

所有语法范畴的正确情况和错误情况。

正确情况

Goal

测试用例`custom/test-goal.txt`：

```
class Main {
    public static void main(String[] args) {{
        mc = new MyClient();
        if (true + false) {
            mc = new MyClient();
        } else {
            mc = new MyClient();
        }
        while (!false) {
            handle = mc.start(10, 10);
        }
    }}
}

class NewHappend {
    public int main2(int[] args) {
        int mc;

        mc = new MyClient();
        return mc;
    }
}

class NewHappend2 {
    public int main2(int[] args) {
        int mc;
        int handle;

        mc = new MyClient();
        while(!false){
            handle = mc.start(488, 388);
        }
        return mc[0];
    }
}
```

测试结果-AST：

```
<Goal>[GoalType]
```

```
├── <MainClass>[MainClassType]: Main, args
│   └── <Statement>[StateType, StateKind]
│       ├── <Statement>[StateType, AssignKind]: mc
│       │   └── <Expression>[ExpType, NewIdKind]: MyClient
│       ├── <Statement>[StateType, IfKind]
│       │   ├── <Expression>[ExpType, AddKind]
│       │   │   ├── true
│       │   │   └── false
│       │   ├── <Statement>[StateType, StateKind]
│       │   │   └── <Statement>[StateType, AssignKind]: mc
│       │   │       └── <Expression>[ExpType, NewIdKind]: MyClient
│       │   └── <Statement>[StateType, StateKind]
│       │       └── <Statement>[StateType, AssignKind]: mc
│       │           └── <Expression>[ExpType, NewIdKind]: MyClient
│       └── <Statement>[StateType, WhileKind]
│           ├── <Expression>[ExpType, NotKind]
│           │   └── false
│           └── <Statement>[StateType, StateKind]
│               └── <Statement>[StateType, AssignKind]: handle
│                   └── <Expression>[ExpType, CallKind]: start
│                       ├── mc
│                       ├── 10
│                       └── 10
├── <ClassDeclaration>[ClassDecType]: NewHappend
│   └── <MethodDeclaration>[MethodDecType]: int, main2
│       ├── <Argument>[ArgType]: int[], args
│       ├── <VarDeclaration>[VarDecType]: int, mc
│       ├── <Statement>[StateType, AssignKind]: mc
│       │   └── <Expression>[ExpType, NewIdKind]: MyClient
│       └── mc
└── <ClassDeclaration>[ClassDecType]: NewHappend2
    └── <MethodDeclaration>[MethodDecType]: int, main2
        ├── <Argument>[ArgType]: int[], args
        ├── <VarDeclaration>[VarDecType]: int, mc
        ├── <VarDeclaration>[VarDecType]: int, handle
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── <Expression>[ExpType, NewIdKind]: MyClient
        ├── <Statement>[StateType, WhileKind]
        │   ├── <Expression>[ExpType, NotKind]
        │   │   └── false
        │   └── <Statement>[StateType, StateKind]
        │       └── <Statement>[StateType, AssignKind]: handle
        │           └── <Expression>[ExpType, CallKind]: start
        │               ├── mc
        │               ├── 488
        │               └── 388
        └── <Expression>[ExpType, BraKind]
            ├── mc
            └── 0
```

MainClass

测试用例`custom/test-main-class.txt`：

```
class Main {
    public static void main(String[] args) {
        System.out.println(1);
    }
}
```

测试结果-AST：

```
<Goal>[GoalType]
└── <MainClass>[MainClassType]: Main, args
    └── <Statement>[StateType, PrintKind]
        └── 1
```

ClassDeclaration

测试用例`custom/test-class-dec.txt`：

```
class Main {
    public static void main(String[] args) {{
        mc = new MyClient();
    }}
}

class NewHappend extends Main {
    int var;

    public int main2(int[] args) {
        int mc;

        return mc;
    }
}
```

测试结果-AST：

```
<Goal>[GoalType]
├── <MainClass>[MainClassType]: Main, args
│   └── <Statement>[StateType, StateKind]
│       └── <Statement>[StateType, AssignKind]: mc
│           └── <Expression>[ExpType, NewIdKind]: MyClient
└── <ClassDeclaration>[ClassDecType]: NewHappend, Main
    ├── <VarDeclaration>[VarDecType]: int, var
    └── <MethodDeclaration>[MethodDecType]: int, main2
        ├── <Argument>[ArgType]: int[], args
        ├── <VarDeclaration>[VarDecType]: int, mc
        └── mc
```

VarDeclaration

测试用例`custom/test-var-dec.txt`：

```
class Main {
```

```
    public static void main(String[] args) {{
        mc = new MyClient();
    }}
}

class NewHappend {
    public int main2(int[] args) {
        int mc;
        int[] handle;
        boolean xxx;
        Main a;
        Main = 2;

        return mc;
    }
}
```

测试结果-AST：

```
<Goal>[GoalType]
├── <MainClass>[MainClassType]: Main, args
│   └── <Statement>[StateType, StateKind]
│       └── <Statement>[StateType, AssignKind]: mc
│           └── <Expression>[ExpType, NewIdKind]: MyClient
└── <ClassDeclaration>[ClassDecType]: NewHappend
    └── <MethodDeclaration>[MethodDecType]: int, main2
        ├── <Argument>[ArgType]: int[], args
        ├── <VarDeclaration>[VarDecType]: int, mc
        ├── <VarDeclaration>[VarDecType]: int[], handle
        ├── <VarDeclaration>[VarDecType]: boolean, xxx
        ├── <VarDeclaration>[VarDecType]: Main, a
        ├── <Statement>[StateType, AssignKind]: Main
        │   └── 2
        └── mc
```

MethodDeclaration

测试用例`custom/test-method-dec.txt`：

```
class Main {
    public static void main(String[] args) {{
        mc = new MyClient();
        if (true + false) {
            mc = new MyClient();
        } else {
            mc = new MyClient();
        }
        while (!false) {
            handle = mc.start(10, 10);
        }
    }}
}
```

```
class NewHappend {
    public int main2(int[] args, boolean b, int c) {
        int mc;
        int handle;

        mc = new MyClient();
        while(!false){
            handle = mc.start(488, 388);
        }
        return mc[0];
    }
}
```

测试结果-AST：

```
<Goal>[GoalType]
├── <MainClass>[MainClassType]: Main, args
│   └── <Statement>[StateType, StateKind]
│       ├── <Statement>[StateType, AssignKind]: mc
│       │   └── <Expression>[ExpType, NewIdKind]: MyClient
│       ├── <Statement>[StateType, IfKind]
│       │   ├── <Expression>[ExpType, AddKind]
│       │   │   ├── true
│       │   │   └── false
│       │   ├── <Statement>[StateType, StateKind]
│       │   │   └── <Statement>[StateType, AssignKind]: mc
│       │   │       └── <Expression>[ExpType, NewIdKind]: MyClient
│       │   └── <Statement>[StateType, StateKind]
│       │       └── <Statement>[StateType, AssignKind]: mc
│       │           └── <Expression>[ExpType, NewIdKind]: MyClient
│       └── <Statement>[StateType, WhileKind]
│           ├── <Expression>[ExpType, NotKind]
│           │   └── false
│           └── <Statement>[StateType, StateKind]
│               └── <Statement>[StateType, AssignKind]: handle
│                   └── <Expression>[ExpType, CallKind]: start
│                       ├── mc
│                       ├── 10
│                       └── 10
└── <ClassDeclaration>[ClassDecType]: NewHappend
    └── <MethodDeclaration>[MethodDecType]: int, main2
        ├── <Argument>[ArgType]: int[], args
        ├── <Argument>[ArgType]: boolean, b
        ├── <Argument>[ArgType]: int, c
        ├── <VarDeclaration>[VarDecType]: int, mc
        ├── <VarDeclaration>[VarDecType]: int, handle
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── <Expression>[ExpType, NewIdKind]: MyClient
        ├── <Statement>[StateType, WhileKind]
        │   ├── <Expression>[ExpType, NotKind]
        │   │   └── false
        │   └── <Statement>[StateType, StateKind]
        │       └── <Statement>[StateType, AssignKind]: handle
        │           └── <Expression>[ExpType, CallKind]: start
        │               ├── mc
        │               ├── 488
```

```
        │               └── 388
        └── <Expression>[ExpType, BraKind]
            ├── mc
            └── 0
```

Type

测试用例`custom/test-type.txt`：

```
class Main {
    public static void main(String[] args) {{
        mc = new MyClient();
        if (true + false) {
            mc = new MyClient();
        } else {
            mc = new MyClient();
        }
        while (!false) {
            handle = mc.start(10, 10);
        }
    }}
}

class NewHappend {
    public int main2(int[] args, boolean b, int c) {
        int mc;
        int handle;

        mc = new MyClient();
        while(!false){
            handle = mc.start(488, 388);
        }
        return mc[0];
    }
}
```

测试结果-AST：

```
<Goal>[GoalType]
├── <MainClass>[MainClassType]: Main, args
│   └── <Statement>[StateType, StateKind]
│       ├── <Statement>[StateType, AssignKind]: mc
│       │   └── <Expression>[ExpType, NewIdKind]: MyClient
│       ├── <Statement>[StateType, IfKind]
│       │   ├── <Expression>[ExpType, AddKind]
│       │   │   ├── true
│       │   │   └── false
│       │   ├── <Statement>[StateType, StateKind]
│       │   │   └── <Statement>[StateType, AssignKind]: mc
│       │   │       └── <Expression>[ExpType, NewIdKind]: MyClient
│       │   └── <Statement>[StateType, StateKind]
│       │       └── <Statement>[StateType, AssignKind]: mc
│       │           └── <Expression>[ExpType, NewIdKind]: MyClient
```

```
|       └── <Statement>[StateType, WhileKind]
|           ├── <Expression>[ExpType, NotKind]
|           |   └── false
|           └── <Statement>[StateType, StateKind]
|               └── <Statement>[StateType, AssignKind]: handle
|                   └── <Expression>[ExpType, CallKind]: start
|                       ├── mc
|                       ├── 10
|                       └── 10
└── <ClassDeclaration>[ClassDecType]: NewHappend
    └── <MethodDeclaration>[MethodDecType]: int, main2
        ├── <Argument>[ArgType]: int[], args
        ├── <Argument>[ArgType]: boolean, b
        ├── <Argument>[ArgType]: int, c
        ├── <VarDeclaration>[VarDecType]: int, mc
        ├── <VarDeclaration>[VarDecType]: int, handle
        ├── <Statement>[StateType, AssignKind]: mc
        |   └── <Expression>[ExpType, NewIdKind]: MyClient
        ├── <Statement>[StateType, WhileKind]
        |   ├── <Expression>[ExpType, NotKind]
        |   |   └── false
        |   └── <Statement>[StateType, StateKind]
        |       └── <Statement>[StateType, AssignKind]: handle
        |           └── <Expression>[ExpType, CallKind]: start
        |               ├── mc
        |               ├── 488
        |               └── 388
        |
        └── <Expression>[ExpType, BraKind]
            ├── mc
            └── 0
```

Statement

测试用例`custom/test-statement.txt`：

```
class Main {
    public static void main(String[] args) {{

        if (true + false) {
            mc = new MyClient();
        } else {
            mc = new MyClient();
        }

        while (!false) {
            handle = mc.start(10, 10);
        }

        System.out.println(1+1);

        mc = new MyClient();

        mc[2] = MyClient.apply();
```

```
    }}
}
```

测试结果-AST：

```
<Goal>[GoalType]
└── <MainClass>[MainClassType]: Main, args
    └── <Statement>[StateType, StateKind]
        ├── <Statement>[StateType, IfKind]
        │   ├── <Expression>[ExpType, AddKind]
        │   │   ├── true
        │   │   └── false
        │   ├── <Statement>[StateType, StateKind]
        │   │   └── <Statement>[StateType, AssignKind]: mc
        │   │       └── <Expression>[ExpType, NewIdKind]: MyClient
        │   └── <Statement>[StateType, StateKind]
        │       └── <Statement>[StateType, AssignKind]: mc
        │           └── <Expression>[ExpType, NewIdKind]: MyClient
        ├── <Statement>[StateType, WhileKind]
        │   ├── <Expression>[ExpType, NotKind]
        │   │   └── false
        │   └── <Statement>[StateType, StateKind]
        │       └── <Statement>[StateType, AssignKind]: handle
        │           └── <Expression>[ExpType, CallKind]: start
        │               ├── mc
        │               ├── 10
        │               └── 10
        ├── <Statement>[StateType, PrintKind]
        │   └── <Expression>[ExpType, AddKind]
        │       ├── 1
        │       └── 1
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── <Expression>[ExpType, NewIdKind]: MyClient
        └── <Statement>[StateType, ArrAssignKind]: mc
            ├── 2
            └── <Expression>[ExpType, CallKind]: apply
                └── MyClient
```

Expression

测试用例`custom/test-expression.txt`：

```
class Main {
    public static void main(String[] args) {{
        mc = 1+1;
        mc = 1-1;
        mc = 1*1;
        mc = 1&&1;
        mc = 1<1;
        mc = mc[1];
        mc = mc.length;
        mc = mc.start(1, 2, 3);
        mc = 1234;
```

```
        mc = true;
        mc = false;
        mc = args;
        mc = this;
        mc = new int [123];
        mc = new MC();
        mc = !true;
        mc = (false);

        mc = 1+2*3*mc.length+(1+1)*3 && !true;
    }}
}
```

测试结果-AST：

```
<Goal>[GoalType]
└── <MainClass>[MainClassType]: Main, args
    └── <Statement>[StateType, StateKind]
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── <Expression>[ExpType, AddKind]
        │       ├── 1
        │       └── 1
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── <Expression>[ExpType, SubKind]
        │       ├── 1
        │       └── 1
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── <Expression>[ExpType, MultiKind]
        │       ├── 1
        │       └── 1
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── <Expression>[ExpType, AndKind]
        │       ├── 1
        │       └── 1
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── <Expression>[ExpType, LtKind]
        │       ├── 1
        │       └── 1
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── <Expression>[ExpType, BraKind]
        │       ├── mc
        │       └── 1
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── <Expression>[ExpType, LenKind]
        │       └── mc
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── <Expression>[ExpType, CallKind]: start
        │       ├── mc
        │       ├── 1
        │       ├── 2
        │       └── 3
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── 1234
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── true
        ├── <Statement>[StateType, AssignKind]: mc
```

```
        │   └── false
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── args
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── this
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── <Expression>[ExpType, NewIntKind]
        │       └── 123
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── <Expression>[ExpType, NewIdKind]: MC
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── <Expression>[ExpType, NotKind]
        │       └── true
        ├── <Statement>[StateType, AssignKind]: mc
        │   └── <Expression>[ExpType, ParKind]
        │       └── false
        └── <Statement>[StateType, AssignKind]: mc
            └── <Expression>[ExpType, AndKind]
                ├── <Expression>[ExpType, AddKind]
                │   ├── <Expression>[ExpType, AddKind]
                │   │   ├── 1
                │   │   └── <Expression>[ExpType, MultiKind]
                │   │       ├── 2
                │   │       └── <Expression>[ExpType, MultiKind]
                │   │           ├── 3
                │   │           └── <Expression>[ExpType, LenKind]
                │   │               └── mc
                │   └── <Expression>[ExpType, MultiKind]
                │       ├── <Expression>[ExpType, ParKind]
                │       │   └── <Expression>[ExpType, AddKind]
                │       │       ├── 1
                │       │       └── 1
                │       └── 3
                └── <Expression>[ExpType, NotKind]
                    └── true
```

一个全覆盖的用例

测试用例`custom/test1.txt`：

```
class Main {
    public static void main(String[] args) {{
        mc = new MyClient();
        if (true + false) {
            mc = new MyClient();
        } else {
            mc = new MyClient();
        }
        while (!false) {
            handle = mc.start(10, 10);
        }
    }}
}

class NewHappend {
```

```
    public int main2(int[] args) {
        int mc;
        int handle;

        mc = new MyClient();
        while(!false){
            handle = mc.start(488, 388);
        }
        return mc[0];
    }
}

class Client {
    int in;
    int out;
    int[] messagelist;
    int index;
    public boolean init(){
        index = 0;
        messagelist = new int[10];
        in = 0;
        out = 0;
        return true;
    }
    public int run(int host, int port){
        int handle;
        handle = this.Juggling();

        return 0;
    }

    public int getMsg(){
        int tmp;
        tmp = messagelist.length;
        if(this.isVoid()){
            tmp = tmp - 1;
        }
        else{
            tmp = tmp * 2;
        }
        if(index < 10){
            messagelist[index] = tmp;
            index = index + 1;
        }
        else{
            index = 0;
        }
        return tmp;
    }

    public boolean isVoid(){
        boolean flag;
        if(0 < messagelist.length){
            flag = false;
        }
        else{
            flag = true;
        }
```

```
        return flag;
    }

    public int Juggling(){
        boolean t;
        int tmp1;
        int tmp2;
        int tmp3;
        tmp1 = 2;
        tmp2 = 3;
        tmp3 = 4;
        while((tmp2 < tmp3)&&(tmp1<tmp2)){
            tmp1 = tmp3 - tmp2;
            tmp2 = tmp2 - tmp1;
            tmp3 = tmp2 * tmp1;
            t = this.HolyLight();
        }
        return (tmp1+tmp2*tmp3)*messagelist.length;
    }


    public boolean HolyLight(){
        in = in + 1;
        out = out - 1;
        System.out.println(false);

        return 0;
    }
}

class MyClient extends Client{

    public int start(int host, int port){
        int handle;
        handle = this.run();

        return handle;
    }
}
```

测试结果-AST：

```
<Goal>[GoalType]
├── <MainClass>[MainClassType]: Main, args
│   └── <Statement>[StateType, StateKind]
│       ├── <Statement>[StateType, AssignKind]: mc
│       │   └── <Expression>[ExpType, NewIdKind]: MyClient
│       ├── <Statement>[StateType, IfKind]
│       │   ├── <Expression>[ExpType, AddKind]
│       │   │   ├── true
│       │   │   └── false
│       │   ├── <Statement>[StateType, StateKind]
│       │   │   └── <Statement>[StateType, AssignKind]: mc
│       │   │       └── <Expression>[ExpType, NewIdKind]: MyClient
│       │   └── <Statement>[StateType, StateKind]
│       │       └── <Statement>[StateType, AssignKind]: mc
```

```
|       |           └── <Expression>[ExpType, NewIdKind]: MyClient
|       └── <Statement>[StateType, WhileKind]
|           ├── <Expression>[ExpType, NotKind]
|           |   └── false
|           └── <Statement>[StateType, StateKind]
|               └── <Statement>[StateType, AssignKind]: handle
|                   └── <Expression>[ExpType, CallKind]: start
|                       ├── mc
|                       ├── 10
|                       └── 10
├── <ClassDeclaration>[ClassDecType]: NewHappend
|   └── <MethodDeclaration>[MethodDecType]: int, main2
|       ├── <Argument>[ArgType]: int[], args
|       ├── <VarDeclaration>[VarDecType]: int, mc
|       ├── <VarDeclaration>[VarDecType]: int, handle
|       ├── <Statement>[StateType, AssignKind]: mc
|       |   └── <Expression>[ExpType, NewIdKind]: MyClient
|       ├── <Statement>[StateType, WhileKind]
|       |   ├── <Expression>[ExpType, NotKind]
|       |   |   └── false
|       |   └── <Statement>[StateType, StateKind]
|       |       └── <Statement>[StateType, AssignKind]: handle
|       |           └── <Expression>[ExpType, CallKind]: start
|       |               ├── mc
|       |               ├── 488
|       |               └── 388
|       └── <Expression>[ExpType, BraKind]
|           ├── mc
|           └── 0
├── <ClassDeclaration>[ClassDecType]: Client
|   ├── <VarDeclaration>[VarDecType]: int, in
|   ├── <VarDeclaration>[VarDecType]: int, out
|   ├── <VarDeclaration>[VarDecType]: int[], messagelist
|   ├── <VarDeclaration>[VarDecType]: int, index
|   ├── <MethodDeclaration>[MethodDecType]: boolean, init
|   |   ├── <Statement>[StateType, AssignKind]: index
|   |   |   └── 0
|   |   ├── <Statement>[StateType, AssignKind]: messagelist
|   |   |   └── <Expression>[ExpType, NewIntKind]
|   |   |       └── 10
|   |   ├── <Statement>[StateType, AssignKind]: in
|   |   |   └── 0
|   |   ├── <Statement>[StateType, AssignKind]: out
|   |   |   └── 0
|   |   └── true
|   ├── <MethodDeclaration>[MethodDecType]: int, run
|   |   ├── <Argument>[ArgType]: int, host
|   |   ├── <Argument>[ArgType]: int, port
|   |   ├── <VarDeclaration>[VarDecType]: int, handle
|   |   ├── <Statement>[StateType, AssignKind]: handle
|   |   |   └── <Expression>[ExpType, CallKind]: Juggling
|   |   |       └── this
|   |   └── 0
|   ├── <MethodDeclaration>[MethodDecType]: int, getMsg
|   |   ├── <VarDeclaration>[VarDecType]: int, tmp
|   |   ├── <Statement>[StateType, AssignKind]: tmp
|   |   |   └── <Expression>[ExpType, LenKind]
|   |   |       └── messagelist
```

```
|   |   ├── <Statement>[StateType, IfKind]
|   |   |   ├── <Expression>[ExpType, CallKind]: isVoid
|   |   |   |   └── this
|   |   |   ├── <Statement>[StateType, StateKind]
|   |   |   |   └── <Statement>[StateType, AssignKind]: tmp
|   |   |   |       └── <Expression>[ExpType, SubKind]
|   |   |   |           ├── tmp
|   |   |   |           └── 1
|   |   |   └── <Statement>[StateType, StateKind]
|   |   |       └── <Statement>[StateType, AssignKind]: tmp
|   |   |           └── <Expression>[ExpType, MultiKind]
|   |   |               ├── tmp
|   |   |               └── 2
|   |   ├── <Statement>[StateType, IfKind]
|   |   |   ├── <Expression>[ExpType, LtKind]
|   |   |   |   ├── index
|   |   |   |   └── 10
|   |   |   ├── <Statement>[StateType, StateKind]
|   |   |   |   ├── <Statement>[StateType, ArrAssignKind]: messagelist
|   |   |   |   |   ├── index
|   |   |   |   |   └── tmp
|   |   |   |   └── <Statement>[StateType, AssignKind]: index
|   |   |   |       └── <Expression>[ExpType, AddKind]
|   |   |   |           ├── index
|   |   |   |           └── 1
|   |   |   └── <Statement>[StateType, StateKind]
|   |   |       └── <Statement>[StateType, AssignKind]: index
|   |   |           └── 0
|   |   └── tmp
|   ├── <MethodDeclaration>[MethodDecType]: boolean, isVoid
|   |   ├── <VarDeclaration>[VarDecType]: boolean, flag
|   |   ├── <Statement>[StateType, IfKind]
|   |   |   ├── <Expression>[ExpType, LtKind]
|   |   |   |   ├── 0
|   |   |   |   └── <Expression>[ExpType, LenKind]
|   |   |   |       └── messagelist
|   |   |   ├── <Statement>[StateType, StateKind]
|   |   |   |   └── <Statement>[StateType, AssignKind]: flag
|   |   |   |       └── false
|   |   |   └── <Statement>[StateType, StateKind]
|   |   |       └── <Statement>[StateType, AssignKind]: flag
|   |   |           └── true
|   |   └── flag
|   ├── <MethodDeclaration>[MethodDecType]: int, Juggling
|   |   ├── <VarDeclaration>[VarDecType]: boolean, t
|   |   ├── <VarDeclaration>[VarDecType]: int, tmp1
|   |   ├── <VarDeclaration>[VarDecType]: int, tmp2
|   |   ├── <VarDeclaration>[VarDecType]: int, tmp3
|   |   ├── <Statement>[StateType, AssignKind]: tmp1
|   |   |   └── 2
|   |   ├── <Statement>[StateType, AssignKind]: tmp2
|   |   |   └── 3
|   |   ├── <Statement>[StateType, AssignKind]: tmp3
|   |   |   └── 4
|   |   ├── <Statement>[StateType, WhileKind]
|   |   |   ├── <Expression>[ExpType, AndKind]
|   |   |   |   ├── <Expression>[ExpType, ParKind]
|   |   |   |   |   └── <Expression>[ExpType, LtKind]
```

```
|   |   |   |   |       ├── tmp2
|   |   |   |   |       └── tmp3
|   |   |   |   └── <Expression>[ExpType, ParKind]
|   |   |   |       └── <Expression>[ExpType, LtKind]
|   |   |   |           ├── tmp1
|   |   |   |           └── tmp2
|   |   |   └── <Statement>[StateType, StateKind]
|   |   |       ├── <Statement>[StateType, AssignKind]: tmp1
|   |   |       |   └── <Expression>[ExpType, SubKind]
|   |   |       |       ├── tmp3
|   |   |       |       └── tmp2
|   |   |       ├── <Statement>[StateType, AssignKind]: tmp2
|   |   |       |   └── <Expression>[ExpType, SubKind]
|   |   |       |       ├── tmp2
|   |   |       |       └── tmp1
|   |   |       ├── <Statement>[StateType, AssignKind]: tmp3
|   |   |       |   └── <Expression>[ExpType, MultiKind]
|   |   |       |       ├── tmp2
|   |   |       |       └── tmp1
|   |   |       └── <Statement>[StateType, AssignKind]: t
|   |   |           └── <Expression>[ExpType, CallKind]: HolyLight
|   |   |               └── this
|   |   └── <Expression>[ExpType, MultiKind]
|   |       ├── <Expression>[ExpType, ParKind]
|   |       |   └── <Expression>[ExpType, AddKind]
|   |       |       ├── tmp1
|   |       |       └── <Expression>[ExpType, MultiKind]
|   |       |           ├── tmp2
|   |       |           └── tmp3
|   |       └── <Expression>[ExpType, LenKind]
|   |           └── messagelist
|   └── <MethodDeclaration>[MethodDecType]: boolean, HolyLight
|       ├── <Statement>[StateType, AssignKind]: in
|       |   └── <Expression>[ExpType, AddKind]
|       |       ├── in
|       |       └── 1
|       ├── <Statement>[StateType, AssignKind]: out
|       |   └── <Expression>[ExpType, SubKind]
|       |       ├── out
|       |       └── 1
|       ├── <Statement>[StateType, PrintKind]
|       |   └── false
|       └── 0
└── <ClassDeclaration>[ClassDecType]: MyClient, Client
    └── <MethodDeclaration>[MethodDecType]: int, start
        ├── <Argument>[ArgType]: int, host
        ├── <Argument>[ArgType]: int, port
        ├── <VarDeclaration>[VarDecType]: int, handle
        ├── <Statement>[StateType, AssignKind]: handle
        |   └── <Expression>[ExpType, CallKind]: run
        |       └── this
        └── handle
```

错误情况

Goal

测试用例`custom/test-goalE.txt`:

```
class Main {
    public static void main(String[] args) {{
        mc = new MyClient();
        if (true + false) {
            mc = new MyClient();
        } else {
            mc = new MyClient();
        }
        while (!false) {
            handle = mc.start(10, 10);
        }
    }}
}

class NewHappend {
    public int main2(int[] args) {
        int mc;

        mc = new MyClient();
        return mc;
    }
}

class NewHappend2 {
    public int main2(int[] args) {
        int mc;
        int handle;

        mc = new MyClient();
        while(!false){
            handle = mc.start(488, 388);
        }
        return mc[0];
    }
}

somethingUnexpectedSyntax
```

测试结果-报错:

```
(37:26): Error: expect [class, [EOF]] after `}`
somethingUnexpectedSyntax
                        ^

Detail error information:
In LR state 11, can not find next action on [ IDENTIFIER,
somethingUnexpectedSyntax ]
The rules in this state are(is) below:
{
  content: [
    'class',
```

```
    'Identifier',
    '<ClassDeclarationO1>',
    '{',
    '<ClassDeclarationR1>',
    '<ClassDeclarationR2>',
    '}'
  ],
  count: 7,
  name: '<ClassDeclaration>',
  lookahead: [ 'class', '[EOF]' ]
}
Elements in symbol stack are: [
  '<MainClass>',
  '<ClassDeclaration>',
  'CLASS',
  'IDENTIFIER',
  '<ClassDeclarationO1>',
  'LT_BRACE',
  '<ClassDeclarationR1>',
  '<ClassDeclarationR2>',
  'RT_BRACE'
]
```

MainClass

测试用例`custom/test-main-classE.txt`:

```
class Main {
    public static void mainx(String[] args) {
        System.out.println(1);
    }
}
```

测试结果-报错:

```
(2:29): Error: expect [main] after `void`
    public static void mainx(String[] args) {
                            ^

Detail error information:
In LR state 303, can not find next action on [ IDENTIFIER, mainx ]
The rules in this state are(is) below:
{
  content: [
    'class',       'Identifier',
    '{',           'public',
    'static',      'void',
    'main',        '(',
    'String',      '[',
    ']',           'Identifier',
    ')',           '{',
    '<Statement>', '}',
    '}'
  ],
```

```
  count: 6,
  name: '<MainClass>',
  lookahead: [ 'class', '[EOF]' ]
}
Elements in symbol stack are: [ 'CLASS', 'IDENTIFIER', 'LT_BRACE', 'PUBLIC',
'STATIC', 'VOID' ]
```

ClassDeclaration

测试用例`custom/test-class-decE.txt`：

```
class Main {
    public static void main(String[] args) {{
        mc = new MyClient();
    }}
}

class NewHappend extend Main {
    int var;

    public int main2(int[] args) {
        int mc;

        return mc;
    }
}
```

测试结果-报错：

```
(7:24): Error: expect [<ClassDeclarationO1>, extends, {] after `NewHappend`
class NewHappend extend Main {
                       ^

Detail error information:
In LR state 6, can not find next action on [ IDENTIFIER, extend ]
The rules in this state are(is) below:
{
  content: [
    'class',
    'Identifier',
    '<ClassDeclarationO1>',
    '{',
    '<ClassDeclarationR1>',
    '<ClassDeclarationR2>',
    '}'
  ],
  count: 2,
  name: '<ClassDeclaration>',
  lookahead: [ 'class', '[EOF]' ]
}
{
  content: [ 'extends', 'Identifier' ],
  count: 0,
```

```
  name: '<ClassDeclarationO1>',
  lookahead: [ '{' ]
}
{
  content: [],
  count: 0,
  name: '<ClassDeclarationO1>',
  lookahead: [ '{' ]
}
Elements in symbol stack are: [ '<MainClass>', 'CLASS', 'IDENTIFIER' ]
```

VarDeclaration

测试用例`custom/test-var-decE.txt`：

```
class Main {
    public static void main(String[] args) {{
        mc = new MyClient();
    }}
}

class NewHappend {
    public int main2(int[] args) {
        int mc;
        int[] int;

        return mc;
    }
}
```

测试结果-报错：

```
(10:18): Error: expect [Identifier] after `]`
        int[] int;
                ^

Detail error information:
In LR state 281, can not find next action on [ INT, int ]
The rules in this state are(is) below:
{
  content: [ 'int', '[', ']' ],
  count: 3,
  name: '<Type>',
  lookahead: [ 'Identifier' ]
}
Elements in symbol stack are: [
  '<MainClass>',           'CLASS',
  'IDENTIFIER',            '<ClassDeclarationO1>',
  'LT_BRACE',              '<ClassDeclarationR1>',
  'PUBLIC',                '<Type>',
  'IDENTIFIER',            'LT_PAREN',
  '<MethodDeclarationO1>', 'RT_PAREN',
  'LT_BRACE',              '<VarDeclaration>',
```

```
  'INT',                    'LT_BRACK',
  'RT_BRACK'
]
```

MethodDeclaration

测试用例`custom/test-method-dec.txt`：

```
class Main {
    public static void main(String[] args) {{
        while (!false) {
            handle = mc.start(10, 10);
        }
    }}
}

class NewHappend {
    public int main2(int[] args, boolean b, int ) {
        int mc;
        int handle;

        mc = new MyClient();
        return mc[0];
    }
}
```

测试结果-AST：

```
(10:50): Error: expect [[, Identifier] after `int`
    public int main2(int[] args, boolean b, int ) {
                                                 ^

Detail error information:
In LR state 279, can not find next action on [ RT_PAREN, ) ]
The rules in this state are(is) below:
{
  content: [ 'int', '[', ']' ],
  count: 1,
  name: '<Type>',
  lookahead: [ 'Identifier' ]
}
{
  content: [ 'int' ],
  count: 1,
  name: '<Type>',
  lookahead: [ 'Identifier' ]
}
Elements in symbol stack are: [
  '<MainClass>', 'CLASS',
  'IDENTIFIER',  '<ClassDeclarationO1>',
  'LT_BRACE',    '<ClassDeclarationR1>',
  'PUBLIC',      '<Type>',
```

```
  'IDENTIFIER',  'LT_PAREN',
  '<Type>',      'IDENTIFIER',
  'COMMA',       '<Type>',
  'IDENTIFIER',  'COMMA',
  'INT'
]
```

Type

测试用例`custom/test-type.txt`：

```
class Main {
    public static void main(String[] args) {{
        mc = new MyClient();
    }}
}

class NewHappend {
    public int main2(int[] args) {
        int a;
        double b;
        main c;

        return mc;
    }
}
```

测试结果-报错：

```
(11:13): Error: expect [int, boolean, Identifier, [EOF], {, if, while,
System.out.println, return] after `;`
        main c;
           ^

Detail error information:
In LR state 278, can not find next action on [ MAIN, main ]
The rules in this state are(is) below:
{
  content: [ '<Type>', 'Identifier', ';' ],
  count: 3,
  name: '<VarDeclaration>',
  lookahead: [
    'int',
    'boolean',
    'Identifier',
    '[EOF]',
    '{',
    'if',
    'while',
    'System.out.println',
    'return'
  ]
}
```

```
Elements in symbol stack are: [
  '<MainClass>',           'CLASS',
  'IDENTIFIER',            '<ClassDeclarationO1>',
  'LT_BRACE',              '<ClassDeclarationR1>',
  'PUBLIC',                '<Type>',
  'IDENTIFIER',            'LT_PAREN',
  '<MethodDeclarationO1>', 'RT_PAREN',
  'LT_BRACE',              '<VarDeclaration>',
  '<Type>',                'IDENTIFIER',
  'SEMI'
]
```

Statement - 规则：{ Statement }

测试用例`custom/test-statementE2.txt`：

注：本规则定义main方法的函数体中，只能包含一个statement，或者再包含多个由花括号括起的statement。

```
class Main {
    public static void main(String[] args) {
        System.out.println(1+1);
        System.out.println(1+1);
    }
}
```

测试结果-报错：

```
(4:27): Error: expect [}] after `;`
        System.out.println(1+1);
                           ^

Detail error information:
In LR state 334, can not find next action on [ PRINTLN, System.out.println ]
The rules in this state are(is) below:
{
  content: [ 'System.out.println', '(', '<Expression>', ')', ';' ],
  count: 5,
  name: '<Statement>',
  lookahead: [ '}' ]
}
Elements in symbol stack are: [
  'CLASS',         'IDENTIFIER',
  'LT_BRACE',      'PUBLIC',
  'STATIC',        'VOID',
  'MAIN',          'LT_PAREN',
  'IDENTIFIER',    'LT_BRACK',
  'RT_BRACK',      'IDENTIFIER',
  'RT_PAREN',      'LT_BRACE',
  'PRINTLN',       'LT_PAREN',
  '<Expression>', 'RT_PAREN',
  'SEMI'
]
```

Statement - 规则： if - else

测试用例`custom/test-statementE2.txt`：

```
class Main {
    public static void main(String[] args) {{
        if (true) {
            true + false
        } else {
            1 + 2
        }
    }}
}
```

测试结果-报错：

```
(4:17): Error: expect [<StatementR1>, <Statement>, {, if, while,
System.out.println, Identifier, }] after `{`
            true + false
               ^

Detail error information:
In LR state 217, can not find next action on [ TRUE, true ]
The rules in this state are(is) below:
{
  content: [ '{', '<StatementR1>', '}' ],
  count: 1,
  name: '<Statement>',
  lookahead: [ 'else' ]
}
{
  content: [ '<Statement>', '<StatementR1>' ],
  count: 0,
  name: '<StatementR1>',
  lookahead: [ '}' ]
}
{ content: [], count: 0, name: '<StatementR1>', lookahead: [ '}' ] }
{
  content: [ '{', '<StatementR1>', '}' ],
  count: 0,
  name: '<Statement>',
  lookahead: [
    '{',
    'if',
    'while',
    'System.out.println',
    'Identifier',
    '[EOF]',
    '}'
  ]
}
```

```
{
  content: [
    'if',
    '(',
    '<Expression>',
    ')',
    '<Statement>',
    'else',
    '<Statement>'
  ],
  count: 0,
  name: '<Statement>',
  lookahead: [
    '{',
    'if',
    'while',
    'System.out.println',
    'Identifier',
    '[EOF]',
    '}'
  ]
}
{
  content: [ 'while', '(', '<Expression>', ')', '<Statement>' ],
  count: 0,
  name: '<Statement>',
  lookahead: [
    '{',
    'if',
    'while',
    'System.out.println',
    'Identifier',
    '[EOF]',
    '}'
  ]
}
{
  content: [ 'System.out.println', '(', '<Expression>', ')', ';' ],
  count: 0,
  name: '<Statement>',
  lookahead: [
    '{',
    'if',
    'while',
    'System.out.println',
    'Identifier',
    '[EOF]',
    '}'
  ]
}
{
  content: [ 'Identifier', '=', '<Expression>', ';' ],
  count: 0,
  name: '<Statement>',
  lookahead: [
    '{',
    'if',
    'while',
```

```
    'System.out.println',
    'Identifier',
    '[EOF]',
    '}'
  ]
}
{
  content: [ 'Identifier', '[', '<Expression>', ']', '=', '<Expression>', ';' ],
  count: 0,
  name: '<Statement>',
  lookahead: [
    '{',
    'if',
    'while',
    'System.out.println',
    'Identifier',
    '[EOF]',
    '}'
  ]
}
Elements in symbol stack are: [
  'CLASS',      'IDENTIFIER',
  'LT_BRACE',   'PUBLIC',
  'STATIC',     'VOID',
  'MAIN',       'LT_PAREN',
  'IDENTIFIER', 'LT_BRACK',
  'RT_BRACK',   'IDENTIFIER',
  'RT_PAREN',   'LT_BRACE',
  'LT_BRACE',   'IF',
  'LT_PAREN',   '<Expression>',
  'RT_PAREN',   'LT_BRACE'
]
```

Statement - 规则： while

测试用例`custom/test-statementE3.txt`：

```
class Main {
    public static void main(String[] args) {
        whie (true) {
            System.out.println(1);
        }
    }
}
```

测试结果-报错：

```
(3:15): Error: expect [=, [] after `whie`
        whie (true) {
             ^

Detail error information:
In LR state 335, can not find next action on [ LT_PAREN, ( ]
The rules in this state are(is) below:
```

```
{
  content: [ 'Identifier', '=', '<Expression>', ';' ],
  count: 1,
  name: '<Statement>',
  lookahead: [ '}' ]
}
{
  content: [ 'Identifier', '[', '<Expression>', ']', '=', '<Expression>', ';' ],
  count: 1,
  name: '<Statement>',
  lookahead: [ '}' ]
}
Elements in symbol stack are: [
  'CLASS',      'IDENTIFIER',
  'LT_BRACE',   'PUBLIC',
  'STATIC',     'VOID',
  'MAIN',       'LT_PAREN',
  'IDENTIFIER', 'LT_BRACK',
  'RT_BRACK',   'IDENTIFIER',
  'RT_PAREN',   'LT_BRACE',
  'IDENTIFIER'
]
```

Statement - 规则： System.out.println

测试用例`custom/test-statementE4.txt`：

```
class Main {
    public static void main(String[] args) {
        System.out.println(123;
    }
}
```

测试结果-报错：

```
(3:32): Error: expect [., [, *, -, +, <, &&, )] after `123`
        System.out.println(123;
                               ^

Detail error information:
In LR state 132, can not find next action on [ SEMI, ; ]
The rules in this state are(is) below:
{
  content: [ 'IntegerLiteral' ],
  count: 1,
  name: '<Expression>',
  lookahead: [
    '.',  '[', '*',
    '-',  '+', '<',
    '&&', ')'
  ]
}
Elements in symbol stack are: [
  'CLASS',           'IDENTIFIER',
```

```
  'LT_BRACE',        'PUBLIC',
  'STATIC',          'VOID',
  'MAIN',            'LT_PAREN',
  'IDENTIFIER',      'LT_BRACK',
  'RT_BRACK',        'IDENTIFIER',
  'RT_PAREN',        'LT_BRACE',
  'PRINTLN',         'LT_PAREN',
  'INTEGER_LITERAL'
]
```

Statement - 规则： Identifier = Expression

测试用例`custom/test-statementE5.txt`：

```
class Main {
    public static void main(String[] args) {
        123 = 123;
    }
}
```

测试结果-报错：

```
(3:12): Error: expect [<Statement>, {, if, while, System.out.println,
Identifier] after `{`
        123 = 123;
          ^

Detail error information:
In LR state 311, can not find next action on [ INTEGER_LITERAL, 123 ]
The rules in this state are(is) below:
{
  content: [
    'class',       'Identifier',
    '{',           'public',
    'static',      'void',
    'main',        '(',
    'String',      '[',
    ']',           'Identifier',
    ')',           '{',
    '<Statement>', '}',
    '}'
  ],
  count: 14,
  name: '<MainClass>',
  lookahead: [ 'class', '[EOF]' ]
}
{
  content: [ '{', '<StatementR1>', '}' ],
  count: 0,
  name: '<Statement>',
  lookahead: [ '}' ]
}
{
  content: [
```

```
    'if',
    '(',
    '<Expression>',
    ')',
    '<Statement>',
    'else',
    '<Statement>'
  ],
  count: 0,
  name: '<Statement>',
  lookahead: [ '}' ]
}
{
  content: [ 'while', '(', '<Expression>', ')', '<Statement>' ],
  count: 0,
  name: '<Statement>',
  lookahead: [ '}' ]
}
{
  content: [ 'System.out.println', '(', '<Expression>', ')', ';' ],
  count: 0,
  name: '<Statement>',
  lookahead: [ '}' ]
}
{
  content: [ 'Identifier', '=', '<Expression>', ';' ],
  count: 0,
  name: '<Statement>',
  lookahead: [ '}' ]
}
{
  content: [ 'Identifier', '[', '<Expression>', ']', '=', '<Expression>', ';' ],
  count: 0,
  name: '<Statement>',
  lookahead: [ '}' ]
}
Elements in symbol stack are: [
  'CLASS',      'IDENTIFIER',
  'LT_BRACE',   'PUBLIC',
  'STATIC',     'VOID',
  'MAIN',       'LT_PAREN',
  'IDENTIFIER', 'LT_BRACK',
  'RT_BRACK',   'IDENTIFIER',
  'RT_PAREN',   'LT_BRACE'
]
```

Statement - 规则： Identifier[Expression] = Expression

测试用例`custom/test-statementE6.txt`：

```
class Main {
    public static void main(String[] args) {
        mc[123 = 132;
    }
}
```

测试结果-报错：

```
(3:17): Error: expect [., [, *, -, +, <, &&, ]] after `123`
        mc[123 = 132;
               ^

Detail error information:
In LR state 74, can not find next action on [ EQ, = ]
The rules in this state are(is) below:
{
  content: [ 'IntegerLiteral' ],
  count: 1,
  name: '<Expression>',
  lookahead: [
    '.',  '[', '*',
    '-',  '+', '<',
    '&&', ']'
  ]
}
Elements in symbol stack are: [
  'CLASS',           'IDENTIFIER',
  'LT_BRACE',        'PUBLIC',
  'STATIC',          'VOID',
  'MAIN',            'LT_PAREN',
  'IDENTIFIER',      'LT_BRACK',
  'RT_BRACK',        'IDENTIFIER',
  'RT_PAREN',        'LT_BRACE',
  'IDENTIFIER',      'LT_BRACK',
  'INTEGER_LITERAL'
]
```

Expression

测试用例`custom/test-expressionE1.txt`：

```
class Main {
    public static void main(String[] args) {{
        mc = 1;&&1;
    }}
}
```

测试结果-报错：

```

```

```
(3:18): Error: expect [{, if, while, System.out.println, Identifier, [EOF], }]
after `;`
        mc = 1;&&1;
                ^

Detail error information:
In LR state 210, can not find next action on [ AND, && ]
The rules in this state are(is) below:
{
  content: [ 'Identifier', '=', '<Expression>', ';' ],
  count: 4,
  name: '<Statement>',
  lookahead: [
    '{',
    'if',
    'while',
    'System.out.println',
    'Identifier',
    '[EOF]',
    '}'
  ]
}
Elements in symbol stack are: [
  'CLASS',      'IDENTIFIER',
  'LT_BRACE',   'PUBLIC',
  'STATIC',     'VOID',
  'MAIN',       'LT_PAREN',
  'IDENTIFIER', 'LT_BRACK',
  'RT_BRACK',   'IDENTIFIER',
  'RT_PAREN',   'LT_BRACE',
  'LT_BRACE',   'IDENTIFIER',
  'EQ',         '<Expression>',
  'SEMI'
]
```