SPLCompiler

Team Members

11812102 Weijie Huang

11810517 Zhaoqi Xiao

11810206 Tao Sun

Design & Implementation

Global variables initialization

In the original reduction rules, initializing a global variable is not supported, i.e.,

```
int x = 10;     // this is not support.
int main() {

}
```

In the project 2, we do not only modify the reduction rule to make it possible, but also add the type checking function. Also, we have provide the extra test case (test_4.spl) to test this feature.

Scope

We also add the scope for the senmatic checking. We use `std::pair<string, int>` as the key type of our red-black tree map. Whenever we get into a `Def`, we will start a new scope. However, it is worth to be minded that when there is a function declaration, the scope will be put ahead. To be specific:

```
// scope 0
int f(int a, int b) {
        // scope 1
        int x; // a, b and c are all in scope 1
}
```

As the variable a and b are in the parameter declaration, which is ahead of the `Def` in the scope 1, so we should use some technique to deal with this situation. And the time when we should get into a new scope and get out of a scope is critical and is needed to be designed carefully.

Structural equivalence

For structural equivalence, we assume that, two structure are the same iff: the variables in the struct are the same, no matter in the type and number and the order of the parameter does not matter.

For instance:

```c
struct st1 {
        int a, b;
        char c;
        float d;
};

struct st2 {
        char c;
        int a;
        int b;
        float d;
}
```

`st1` and `st2` are the same. And also, if the structures in two structures are the same, we will think them as the same type. You can do assignment operation on two structural equivalent type.

B.T.W, I think the order-nomatter equivalence is harder than the order-matter one. Because the former you need to use a bucket sort to check the type equivalence, while in the latter one, you just need to iterate the link list.

Summary

The detail in the implementation is too much to write in the report. To be honest, we wrote a lot of code... a lot of a lot of code...