به نام خدا

ملیکا داودزاده

فاز چهارم پروژه کامپایلر

در فاز پارسر ما نهایتا به یک درخت پارسر رسیدیم که در این فاز می خواهیم با استفاده از آن، درخت دیگری بسازیم که اطلاعات بیشتری به ما بدهد و خطاهای بیشتری را مشخص کنیم.

در فولدر این پروژه فایل های زیر موجود است :

semant.cc

semant.h

cool-tree.h

Test.pl

که در باره هر کدام توضیح داده خواهد شد.

در این فاز ما می خواهیم که عبارات موجود در برنامه ها را از جهات مختلف مانند نوع آنها (type)، بررسی اسکوپ متغییر ها و ارتباط آنها با یکدیگر تفسیر کنیم به صورتی که ببنیم آیا برنامه ما معنای درستی می دهد یا نه. در ابتدا نیاز به یک symbol table داریم تا بعدا بتوانیم به درستی نوع متغیر ها را تشخیص دهیم و در زبان cool که همچیز در قالب کلاس تعریف می شوند بتوانیم به درستی نام گذاری کلاس ها را تشخیص و خطایابی کنیم. Symbol table در فایل semant.cc به صورت پیش فرض قرار گرفته است.

زبان cool چند کلاس اصلی دارد : Object, Str, int, IO, Bool این کلاس ها باید در ابندا شناسایی شوند و چند مورد دیگر نیز مانند اینکه هیچ دو کلاسی نباید همنام باشند، نام کلاس نمیتواند SELF-TYPE باشد و فقط یک کلاس Main وجود دارد باید برسی شود که در کد (فایل semant.cc) به صورت توابعی تمام این مسائل بررسی شده است.

یکی از قسمت های مهمی که در این فاز بررسی می شود بررسی وراثت کلاس ها در زبان cool است که به وسیله گراف وراثت بررسی می شود. کد زیر در برنامه پس از آنکه حالات بالا بررسی شد و مشکلی نداشت نحوه وراثت کلاس ها و خطا های آن مانند وجود دور را بررسی میکند :

```
for (int i = classes->first(); classes->more(i); i = classes->next(i)){
    curr_class = classes->nth(i);


        log << "    " << curr_class->GetName();
```

```
        Symbol parent_name = curr_class->GetParent();
        while (parent_name != Object && parent_name != classes->nth(i)-
>GetName()) {
            // check that the parent of curr_class is present in m_classes
            if (m_classes.find(parent_name) == m_classes.end()) {
                semant_error(curr_class)
                    << "Error! Cannot find class " << parent_name << std::endl;
                return;
            }


            // check that the parent is not Int or Bool or Str or SELF_TYPE
            if (parent_name == Int || parent_name == Str || parent_name ==
SELF_TYPE || parent_name == Bool) {
                semant_error(curr_class)
                    << "Error! Class " << curr_class->GetName() << " cannot
inherit from "
                    << parent_name << std::endl;
                return;
            }


            log << " <- " << parent_name;
            curr_class = m_classes[parent_name];
            parent_name = curr_class->GetParent();
        }


        if (parent_name == Object) {
            log << " <- " << parent_name << std::endl;
        } else {
            semant_error(curr_class) << "Error! Cycle inheritance!" <<
std::endl;
            return;
        }
    }


    log << std::endl;
```

}

یکی دیگر از قسمت ها یمهمی که در ویدیو های کورس گفته شد برای اینکه بتوانیم به درستی نوع متغییر یا عبارت را تشخیص درهیم و باید برای این فاز در نظر گرفته می شد پیدا کردن بزرگترین جد مشترک و هم چنین استفاده از NO-TYPE بود که در کد زیر میبنیم که چطور بدست می آید :

```
ClassTable::FindLCA(Symbol type1, Symbol type2)
{
    std::list<Symbol> path1 = GetInheritancePath(type1);
    std::list<Symbol> path2 = GetInheritancePath(type2);

    Symbol ret;
    std::list<Symbol>::iterator iter1 = path1.begin(), iter2 = path2.begin();

    while (iter1 != path1.end() && iter2 != path2.end()) {
        if (*iter1 == *iter2) {
            ret = *iter1;
        } else {
            break;
        }

        iter1++;
        iter2++;
    }

    return ret;
}
```

در ادامه semant.cc مهمترین قسمت این فاز یعنی چک کردن درست بودن نوع متغیر ها و عبارت ها و درست تعریف شدن متدها انجام می شود که این کار با کمک چیز هایی که در بالا گفتیم مانند LCA و دیگر چیز ها انجام می شود. همچنین syntax tree را کامل تر میکند تا اطلاعات بیشتری در دسترس باشد.

فایل semant.h هدر این فایل می باشد که تغییر خاصی نسبت به فایلی که در دیفالت پروژه بود نکرده است.

پس از انجام این کارها میتوانیم به وسیله فایل Test.pl این فاز را تست کنیم.