فاز چهارم پروژه کامپایلر
Semantic analyzer

ریحانه درفشی ۶۱۰۳۹۶۰۹۸

به منظور تکمیل مرحله semantic analyzer مراحل زیر دنبال شده است :

1. نصب کلاسهای اصلی
2. ایجاد گراف وراث
3. قرار دادن مقادیر در scope table و DFS
4. یافتن کوچکترین جد مشترک
5. بررسی برای وجود دور در گراف وراثت
6. بررسی تایپ exp ها

1. نصب کلاسهای اصلی :

کلاس اصلی در COOL کلاس Object است. در کنار این 4 کلاس اساسی به نام های IO ، Int ، Bool و String وجود دارد. Hashmaps برای map کردن parent به childeren استفاده شده است. متودهای اصلی با استفاده از map به کلاس های مربوط مرتبط شدند و هر متود با لیستی از formal ها جدا از Int و Bool pair شده است.

2. ایجاد گراف وراثت :

در اینجا گراف وراثت Cool ساخته شده است. ابتدا کلاسهای ابتدایی Object و String ، Bool ، Int و IO به نمودار اضافه می شوند. به هر کلاس برای شناسایی یک id منحصر به فرد داده می شود. Error handling اولیه به شرح زیر انجام می شود:

1 .بررسی اینکه نام کلاس "self" نباشد
2. هیچ دو کلاس نمی توانند اسم یکسان داشته باشند.
3. کلاس نمی تواند SELF_TYPE یا هر یک از کلاسهای اصلی را به ارث برساند.
4- فقط یک کلاس main باید وجود داشته باشد.

بعد از error handling اولیه ، گراف وراثت را برای method ها و attribute ها به شرح زیر بررسی میکنیم:

1. بررسی کنید که آیا method یا attribute ای از آن کلاس به ارث رسیده است.
2. اگر بله ، سپس پارامترهای method را بررسی کنید ، از جمله: 1. تعداد پارامترها 2. type پارامترها 3. Return type متد 4.چندتا وجود داشتن از formal ها

در صورت عدم رعایت هر یک از موارد فوق ، پیامهای خطای مناسب برمی گردند.

3. قرار دادن مقادیر در scope table و DFS :

برای type checking انجام دادن exp ها ، یک بار دیگر گراف را طی می کنیم. تابع dfs در انتهای گراف وراثت صدا زده می شود و فقط در صورت رخ ندادن خطا تمام می شود.
1. برای هر کلاس وارد scope table می شویم و مقدار attribute مربوطه را در جدول وارد می کنیم. هم attribute و هم exp body آن بررسی می شوند ، و اطمینان حاصل میشود که return type این عبارت با attribute مطابقت دارد. این کار با استفاده از تابع ExprType انجام می شود که type را بر اساس exp instance باز می گرداند.
2- method ها به روشی مشابه اداره می شوند ، جایی که برای هر method در یک کلاس وارد scope می شویم و formal های مربوط به آن را وارد می کنیم. یک exp type checking مشابه انجام میشود . لازم است در formal ها دقت شود که کوچکترین جد مشترک با type بازگشت method مطابقت داشته باشد.

4. یافتن کوچکترین جد مشترک :

از متد iterative برای بالا رفتن از گراف های هر دو عبارت ورودی استفاده می شود ، به این ترتیب که کوچکترین جد مشترک بازگردانده می شود ، اگر وجود داشته باشد ، اگر نه "no_type" بازگردانده می شود.

5. بررسی برای وجود دور در گراف وراثت :

با استفاده از index ها در گراف ، وجود دور را با استفاده از ریکرژن و بازگشتن به همه گره های مجاور بررسی می کنیم. یک آرایه بولین تعریف میشود و هر گره ای که دیده میشود در این ارایه مارک میشود. اگر شرایطی وجود داشته باشد که یک گره دیده شده دوباره دیده شود ، به این معنی است که یک دور وجود دارد.

6. بررسی تایپ exp ها :

در این مرحله type checking متد های کلاس ها صورت می گیرد. این امر در عملکرد dfs همانطور که گفته شد صورت می گیرد و توسط تابع exprType پیاده سازی می شود. در زیر قوانین مختلف برای عبارات مختلف وجود دارد.

A.Classes and No_expr () :

در اینجا کلاس هایی مانند int ، bool و string l بررسی میشوند و همچنین "no type" و exp type مربوطه بازگردانده می شود.

B.Object class () :

Object داده شده با استفاده از عملکرد lookUpGlobal در symbol table جستجو شده است. اگر هیچ object ای مطابقت نداشته باشد error ایجاد می شود و رشته "object" بازگردانده می شود.

C.Assignment (=) :

1. Id object باید در escope فعلی یا scope اجداد یک نام معتبر باشد.
2. Exp type سمت راست باید یک type به ارث رسیده از attribute در سمت چپ باشد.
3. Return type در صورت valid بودن, تایپ exp سمت راست است. در غیر این صورت "object" است.

D.Static Dispatch (@<type_name>.()) :

1. کلاسي که متد آن صدا زده میشود باید قبلا تعریف شده باشد.
2. متد مورد نظر باید یک متد valid در کلاس فوق باشد.
3. مقدار formal های متد صدا زده شده دقیقاً برابر با متد اصلی است.
4. تایپ exp باید یک تایپ کوچکتر valid از کلاس type_name باشد.
5. return type نهایی برابر تایپ exp است.

E.Dispatch (<type_name>.()) :

Dispatch در داخل static dispatch بررسی می شود و باید با توجه به مواردی که با کلاس که در آن عضو شده است ، مطابق با قوانین فوق باشد.

F. Condition (if then <then_expr> else <else_expr>) :

1. تایپ predicate باید bool باشد
2. Return type شرط کوچک ترین جد مشترک then_expr و else_expr است.

G. Loop (while loop pool) :

1. تایپ predicate exp باید بول باشد.
2. return type لوپ object است

H. Case (case [: [<-]]+ of [: =>]+ esac) :

1. اولین exp باید درست باشد
2. هر یک از شاخه ها باید type معتبر داشته باشد
3. استیتمنت دو شاخه نمیتواند تایپ یکسان داشته باشد
4. Return type عبارت شاخه باید یک valid type کوچک تر از declared type شاخه باشد.
5. return type نهایی case کوچک ترین جد مشترکreturn type شاخه ها است.
3.

I. Block ({ }) :

1. هر یک از statement های موجود در body ارزیابی می شود.
2. return type برابر با تایپ اخرین استیتمنت است

J. Let ({ let :<type_decl> in }) :

1. هیچ identifier ای نمیتواند نام self داشته باشد
2. Exp اولیه ارزیابی میشود و به scope جدید وارد میشود
3. type decleration و assigned exp باید یک زیرکلاس valid از تایپ مشخص شده باشند.
4. تایپ نهایی let statement تایپ body است.

K. Operators (+,-,*,/,<,<=,~) :

1. تایپ عبارت سمت راست و چپ باید int باشد.
2. برای <, => return type برابر bool است در غیر این صورت int است.

L. Equality (=,<,<=,not) :

تایپ سمت چپ و سمت راست باید یکی باشد.
تایپ exp تنها میتواند bool, string و یا int باشد.
'<'و '=>' تنها میتواند از تایپ int در هر دو طرف داشته باشند.
اگر error پیش نیاید return type برابر bool است در غیر این صورت برابر object است.

M. Validating 'new' :

1. کلاس باید یک تایپ تعریف شده valid در گراف وراثت باشد.
2. return type برابر تایپ کلاس است


File Machine View Input Devices Help
compilers@compilers-vm: ~/cool/semantic
File Edit Tabs Help
compilers@compilers-vm: ~/cool/semantic$ ./mysement good.cl
#1
_program
  #1
  _class
    C
    Object
    "good.cl"
    (
    #1
    _attr
      a
      Int
      #1
      _no_expr
      : _no_type
    #1
    _attr
      b
      Bool
      #1
      _no_expr
      : _no_type
    #1
    _method
      init
      #1
      _formal
        x
        Int
      #1
      _formal
        y
        Bool
      C
      #1
      _block
        #1
        _assign
          a
          #1
          _object
            x
          : Int
        : Int
        #1
        _assign
          b
          #1
          _object
            y
          : Bool
        : Bool
        #1
        _object
          self
        : SELF_TYPE
      : SELF_TYPE
)
  #1
  _class
    Main
    Object
    "good.cl"
    (
    #1
    _method
      main
      C
      #1
      _dispatch
        #1
        _new
          C
        : C
        init
        (
        #1
        _int
          1
        : Int
        #1
        _bool
          1
        : Bool
)
      : C
)