Compile Principle - HW of Chapter 6

解雲暄 3190105871

> 6.7 Consider the following grammar for simple Pascal-style declarations:
>
> ```
> decl -> var-list: type
> var-list -> var-list, id | id
> type -> integer
> type -> real
> ```
>
> Write the attribute grammar for this grammar.

```
decl -> var-list: type  var-list.dtype = type.dtype
var-list1->var-list2,id var-list2.dtype = var-list1.dtype
                        id.dtype = var-list1.dtype
var-list -> id          id.dtype = var-list.dtype
type -> integer         type.dtype = integer
type -> real            type.dtype = real
```

> 6.8 Consider the grammar of 6.7. Rewrite the grammar so that the type of a variable can be purely synthesized attribute, and give a new attribute grammar for the type has this property.

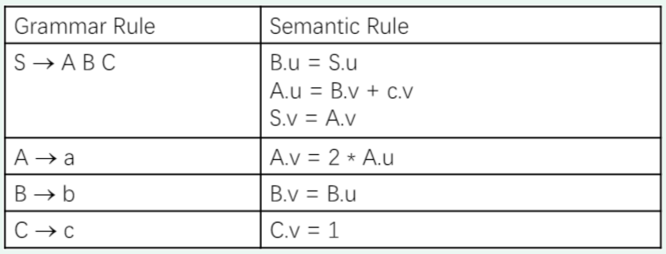
The problem is that, the grammar in 6.7 passes the type of variables top-down (`decl` to `id`), but if we need synthesized attribute, we should pass the type bottom-up. Therefore we need to find a right associative way, which needs right recursion grammar.

`decl` will be expanded to `id, id, ..., id : type`, so we rewrite the grammar to be:

```
decl -> id right
right -> , id right | : type
type -> integer | real
```

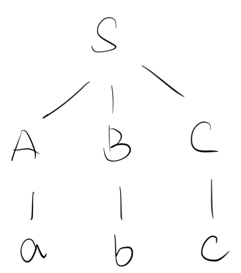
Therefore the corresponding attribute grammar is:

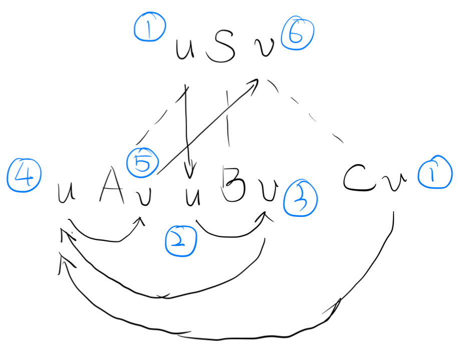
```
decl -> id right         id.dtype = right.dtype
right1 -> , id right2    right1.dtype = right2.dtype
                         id.dtype = right2.dtype
right -> : type          right.dtype = type.dtype
type -> integer          type.dtype = integer
type -> real             type.dtype = real
```

> 6.13 Consider the following attribute grammar:
>
> 
>
> | Grammar Rule | Semantic Rule |
> |---|---|
> | S → A B C | B.u = S.u
A.u = B.v + c.v
S.v = A.v |
> | A → a | A.v = 2 * A.u |
> | B → b | B.v = B.u |
> | C → c | C.v = 1 |
>
> (a)Draw the parse tree for string "abc", and draw the dependency graph for the associated attributes. Describe a correct order for the evaluation of the attributes.

The answer is in the figure below. The order of evaluation of the attributes has been shown on the graph. Note that there are two ① as `S.u` and `C.v` has no dependency and can be parallel evaluated.

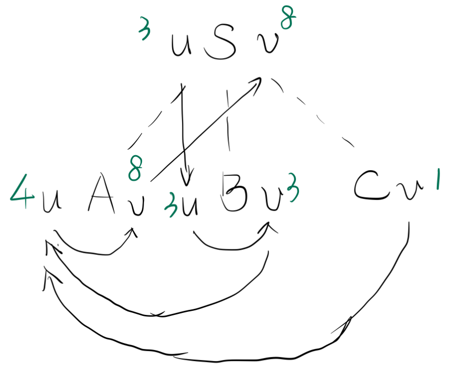
That is, the order is: `S.u, C.v; B.u; B.v; A.u; A.v; S.v`。


Parse Tree


Dependency Graph

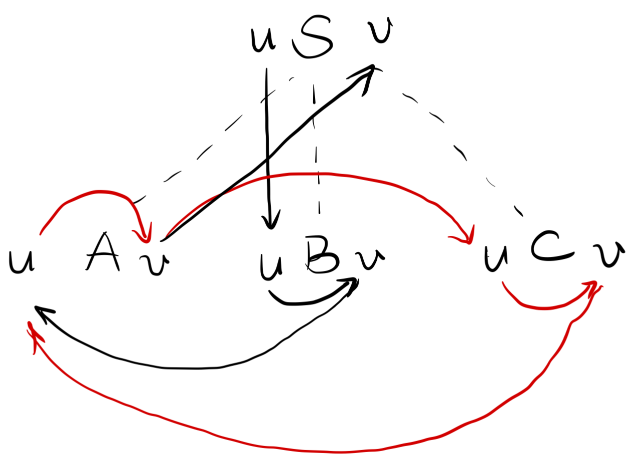
> (b)Suppose that S.u is assigned the value of 3 before attribute evaluation begins. What is the value of S.v when the evaluation has finished?

`S.v = 8` :


Dependency Graph

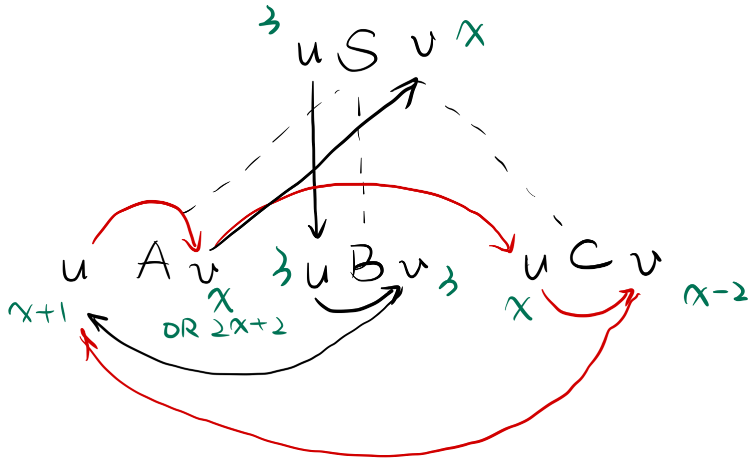
> (c) Suppose the attribute equations are modified as follows:
>
> | Grammar Rule | Semantic Rule |
> |---|---|
> | S → A B C | B.u = S.u
C.u = A.v
A.u = B.v + c.v
S.v = A.v |
> | A → a | A.v = 2 * A.u |
> | B → b | B.v = B.u |
> | C → c | C.v = C.u - 2 |
>
> What value does S.v have after attribute evaluation, if S.u=3 before evaluation begins?

The dependency graph is shown below:



We can see that the dependencies signed by red arrows form a cycle, making the value unable to calculate.

If we just treat one of the value as an unknown quantity and try to solve the equation, we will get:



Where $2x + 2 = x$

Therefore `S.v` = x = -2.