Compile Principle - HW of Chapter 5

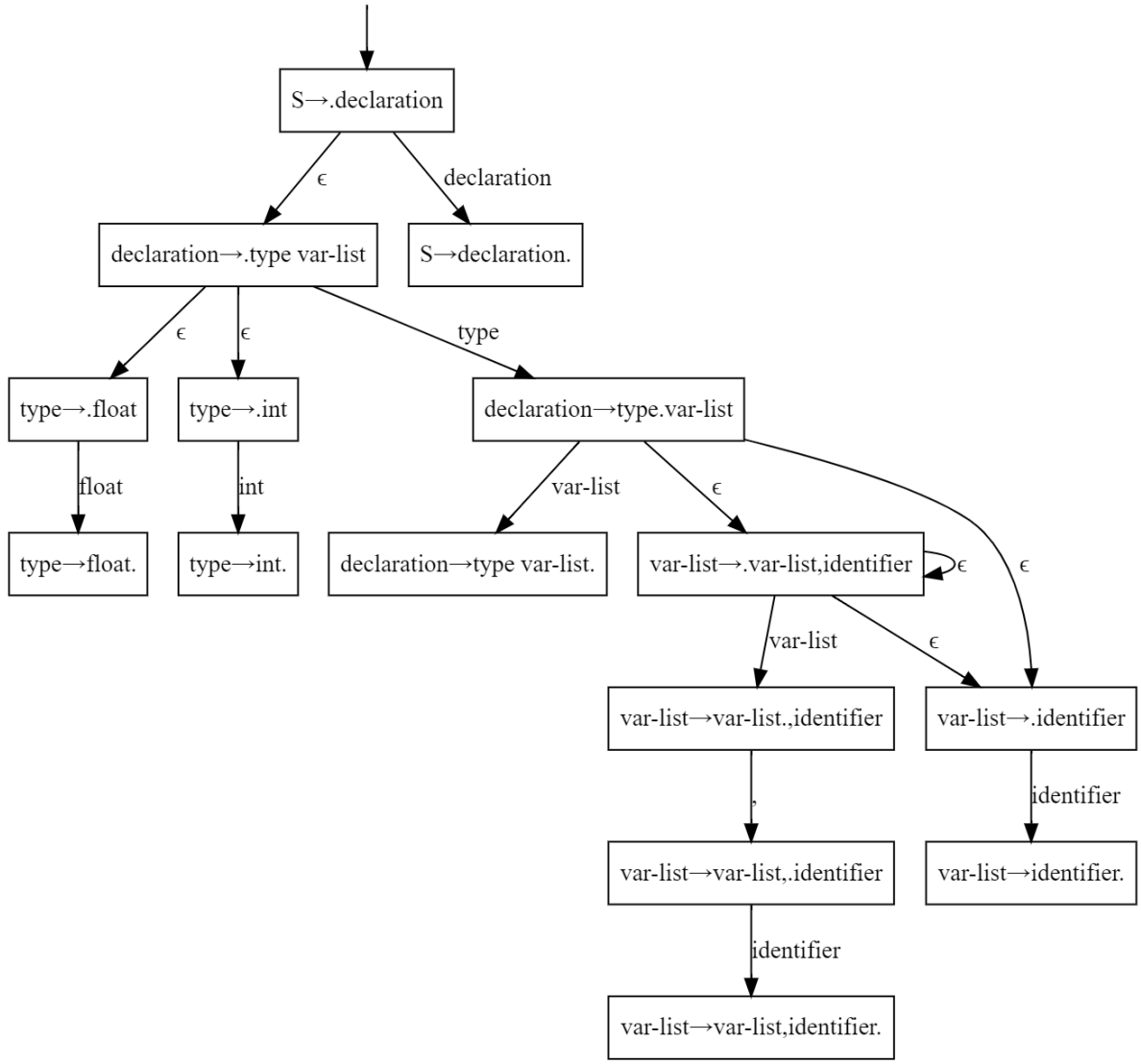
解雲暄　3190105871

> 5.8 Consider the following grammar
>
> ```
> declaration → type var-list
> type → int
>      | float
> var-list → identifier, var-list
>          | identifier
> ```
>
> a. Rewrite it in a form more suitable for bottom-up parsing.
>
> b. Construct the LR(0) DFA for the rewritten grammar.
>
> c. Construct the SLR(1) parsing table for the rewritten grammar.

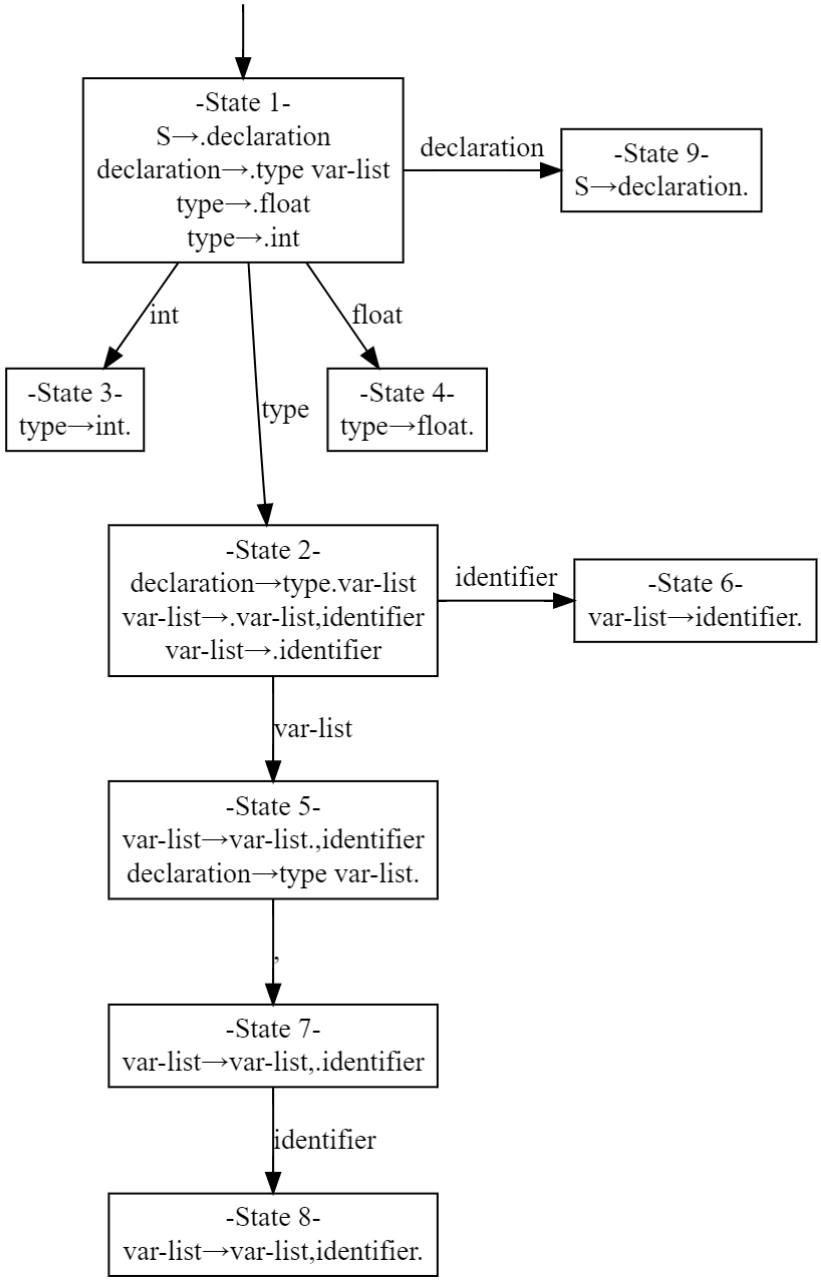
a. Add a start symbol and remove the right recursion:

```
S -> declaration
declaration -> type var-list
type -> int
type -> float
var-list -> var-list, identifier
var-list -> identifier
```

b. The LR(0) NFA is as follows:



We can simplify the NFA and get the requested DFA:



c. We can know that:

$Follow(S) = \{\$\} \subseteq Follow(declaration) \subseteq Follow(var\text{-}list)$
$First(var\text{-}list) = \{identifier\} \subseteq Follow(type)$
$\{',' \} \subseteq Follow(var\text{-}list)$

Therefore, the follow set of each non-terminal is:

non-terminal	S	declaration	var-list	type
follow set	{$}	{$}	{',' , $}	{identifier}

So we can construct the SLR(1) Parsing Table with the DFA and follow sets:

state	shift					goto		
	int	float	identifier	,	$	var-list	declaration	type
1	s3	s4					g9	g2
2			s6			g5		
3			r3					
4			r4					
5				s7	r2			
6				r6	r6			
7			s8					
8				r5	r5			
9					Accept			

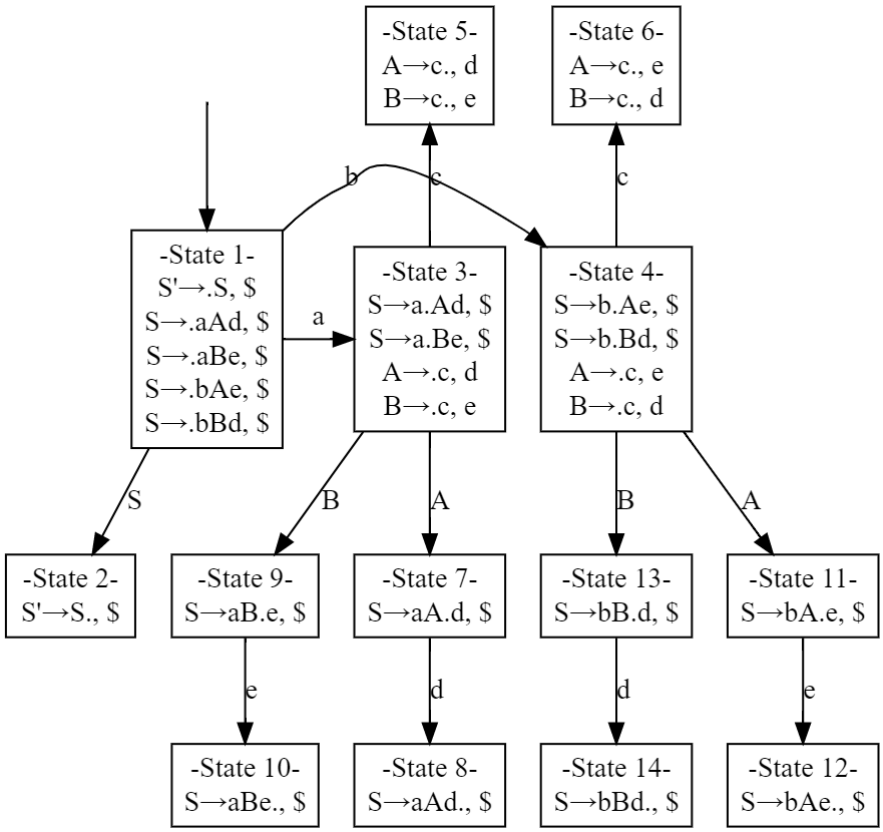
> 5.12 Show the following grammar is LR(1) but not LALR(1):

```
S → a A d
  | b B d
  | a B e
  | b A e
A → c
B → c
```

We rewrite the grammar to be:

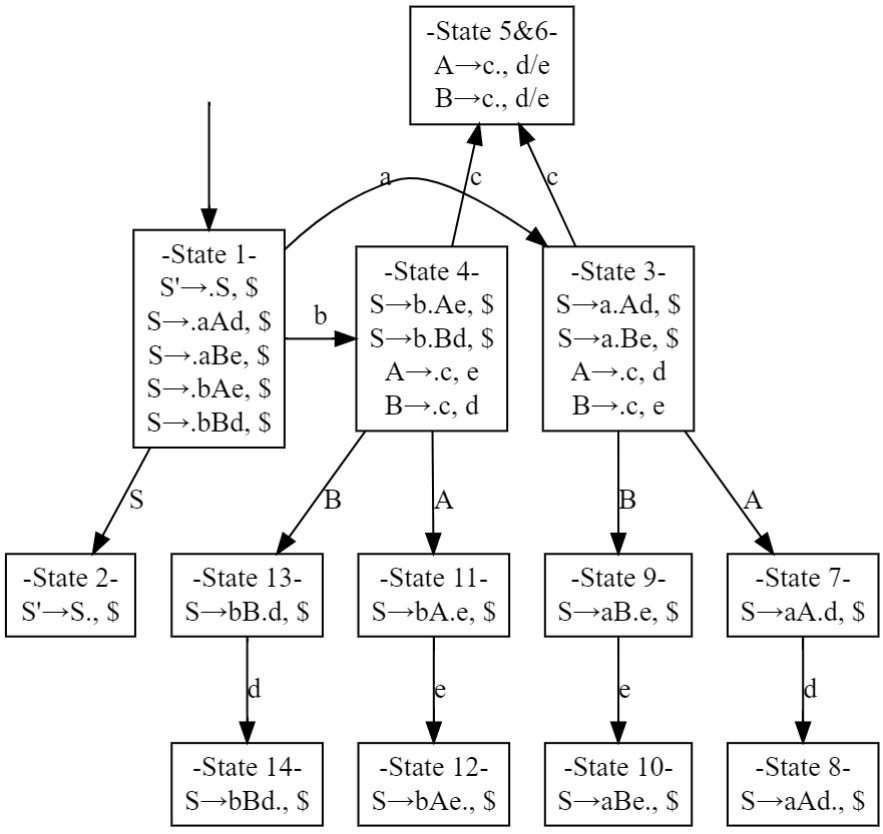
```
S'-> S
S -> aAd
S -> bBd
S -> aBe
S -> bAe
A -> c
B -> c
```

So we can get the LR(1) DFA:



We can see that there is no conflicts in the LR(1) DFA above, so this grammar is LR(1).

But the LALR(1) DFA is:



That is, we merge the original State 5 and State 6, but it will introduce a reduce-reduce conflict. So this grammar is not LALR(1).