CS131 Midterm

2021 Spring

1.abc Syntax Error vs Semantic Error

Types of Errors

Error Type	Example	Detector
Lexical	`x # y = 1`	Lexer
Syntax	`x = 1 y = 2`	Parser
Semantic	`int x; y = x(1)`	Type Checker
Correctness	Can compile, but wrong output	User / Static Analysis / Model Checker / ···

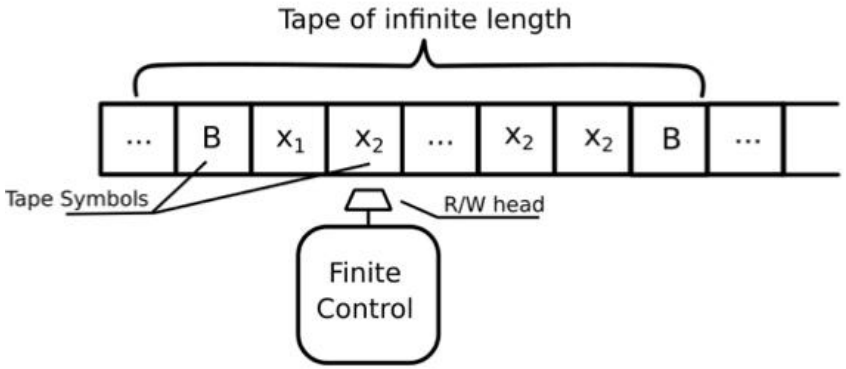
(b) (2 points) Which phase of a compiler may generate an error of undefined variables.

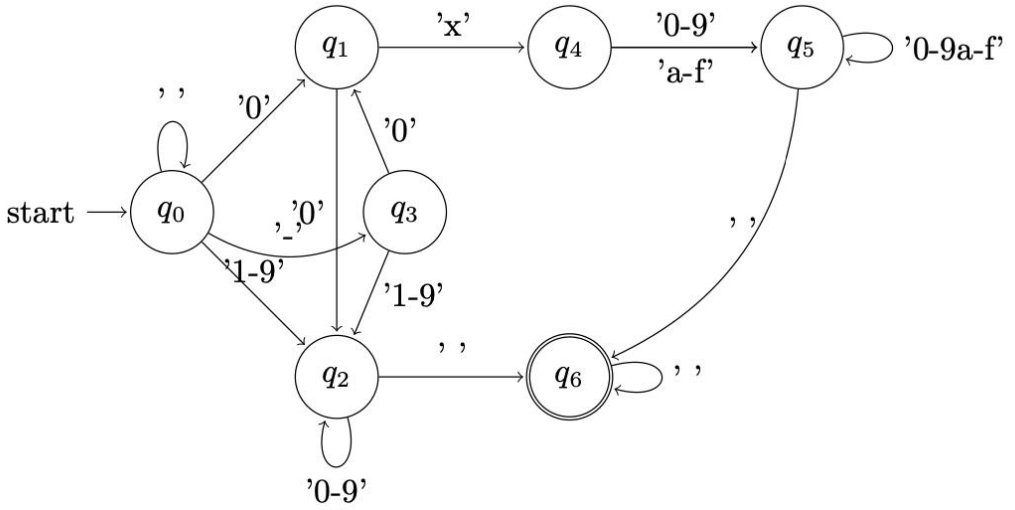
(b) **Semantic Analysis**

- Yacc accept id: {a,b} , bison accept let add : int { a : int; a+b } b is not defined, checked after bison.

1.d regular expression of unsigned number

- Compiler is highly connected with Games, Computer Arch and Operating System. Everything is Autometable.





- We've covered software data decoder that accepts positive and negative integers in hexadecimal and decimals at discussion1.

1.d atoi https://leetcode-cn.com/problems/string-to-integer-atoi/solution/8-zi-fu-chuan-zhuan-huan-zheng-shu-atoi-og1d9/

解题思路

1. 循环筛掉前置的空格
2. 判断符号位，如果符号位超过一个则不合法，直接返回0
3. 最后遍历所有数字位，把符号位和数字结合
4. 限制范围在 `-2**31` 到 `2**31-1` 之间

```
    // 4. 处理开头全是0的情况
    while(start < length && s[start] == '0')
        ++start;

    if(start == length)
        return 0;

    // 转换整数
    long long res = 0;
    while(start < length)
    {
        if(!isdigit(s[start])) // 5. 判断这个字符串目前的开头是不是数字，不是数字就不用处理了。
            break;
        res = 10*res + (s[start] - '0'); //挨个加
        if(res > int_max) //判断是否越界
        {
            if(flag)
                res = int_max;
            else
                res = -int_max-1;
            return res;
        }
        ++start;
    }

    return flag?res : -res;
```

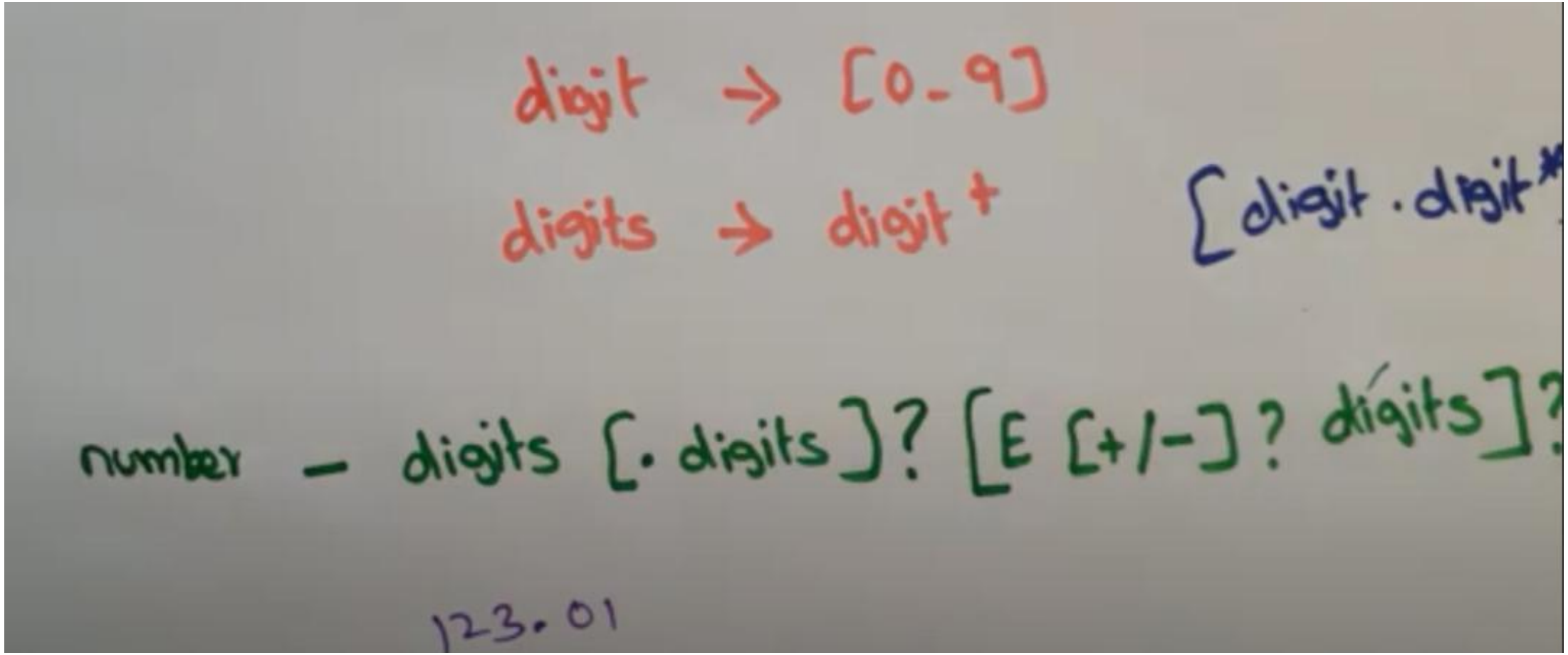
```
class Solution {
private:
    int int_max = 2147483647; //32位最大正数。判断是否越界
public:
    int myAtoi(string s) {
        if(s.empty()) // 1. 首先判断字符串内要有东西。
            return 0;
        int start = 0;
        int length = s.size();

        // 2. 丢掉前导空格
        while(start < length && s[start] == ' ')
            ++start;
        if(start == length)
            return 0;

        // 3. 判断正负号。
        int flag = 1; //flag标志为正负号， 1为正数， 0为负数
        if(s[start] == '-')
        {
            ++start;
            flag = 0;
        }
        else if(s[start] == '+')
            ++start;

        // 判断是否字母
        if(!isdigit(s[start]))
            return 0;
```

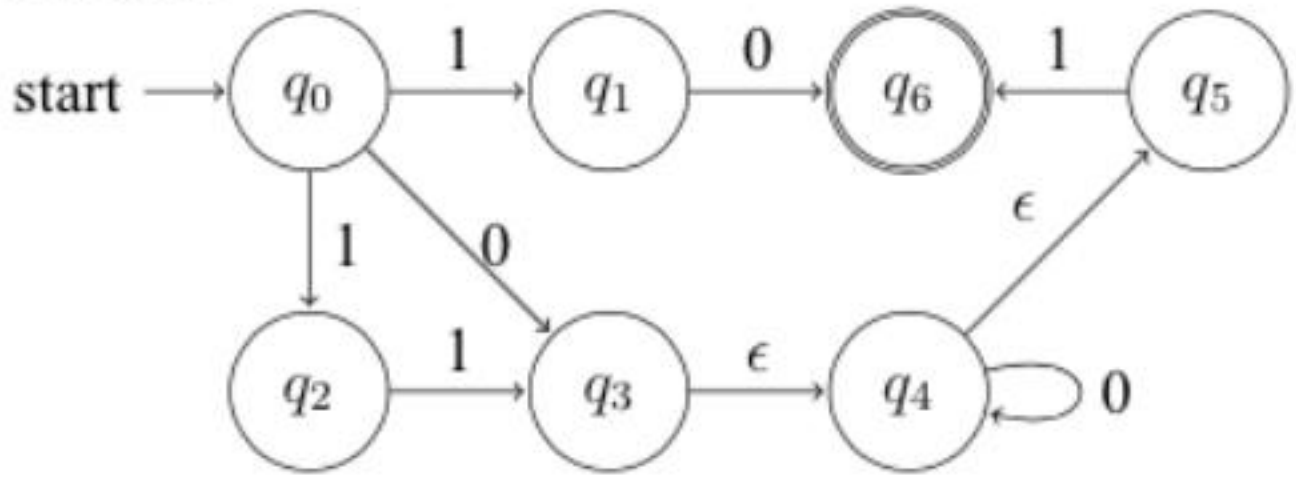
1.d strtod Unsigned Number to Double



2.a How to transform Regular Expr at speed?

- Where to diverge?
- Where to converge?
- The simpler an NFA you write, the simpler you get for the next 2 probs.

Solution:



2.b Subset Construction

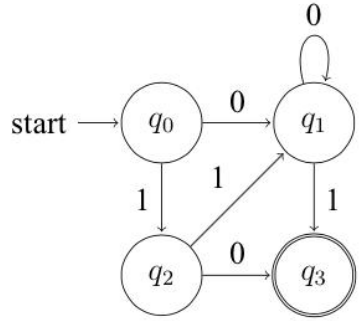
For every NFA N, there is a DFA M s.t. $L(M) = L(N)$:

- Idea: *Subset Construction*, setting $Q' = 2^Q$; **see my PL note - "Convert NFA $\rightarrow$ DFA"**
- This means a language L' is regular iff L' is recognized by an NFA!
- This means using NFAs in place of DFAs can make proofs about RLs much easier!

```
void subsetConstruction() {
    S0 = epsClosure({s0});
    DStates = {(S0, unmarked)};
    while (DStates has any unmarked State U) {
        Mark State U;
        for (each possible input char c) {
            V = epsClosure(move(U, c));
            if (V is not empty) {
                if (V is not in DStates)
                    Include V in DStates, unmarked;
                Add the Transition U--c->V;
            }
        }
    }
}
```

2.c Minimize the DFA

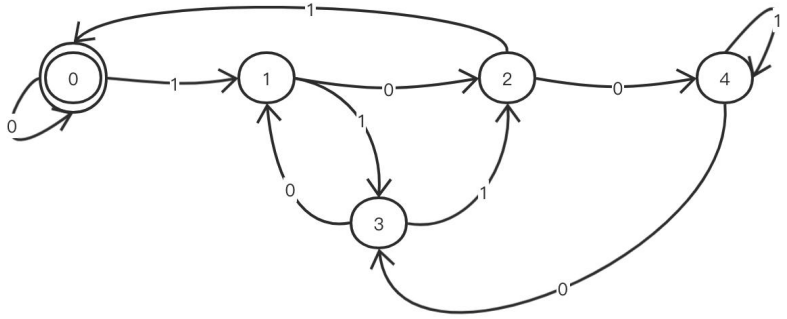
Solution: I_0=$\{q_0, q_1, q_2, q_3\}$, $I_3 = \{q_4\}$
$\rightarrow I_0 = \{q_0, q_1, q_3\}, I_2 = \{q_2\}, I_3 = \{q_4\}$
$\rightarrow I_0 = \{q_0\}, I_1 = \{q_1, q_3\}, I_2 = \{q_2\}, I_3 = \{q_4\}$




```
void minimize() {
    PI = {G_a, G_n};
    do {
        for (every group G in PI) {
            for (every pair of States (s, t) in G) {
                if (for every possible input char c, transition s--c-> and t--c->
                    go to states in the same group)
                    s, t are in the same subgroup;
                else
                    s, t should split into different subgroups;
            }
            Split G according to the above information;
        }
    } while (PI changed in this iteration);
    Every Group in PI is a state in the minimal DFA;
}
```

3.How to interpret the all binary into state machine? $G = \{\text{binary sequences that can be divided by } 5\}.$

- How many state should the Autometa have? mod 5: 5.
- What's the transmission function? think like DP. (小学奥数)
  - The (current state*2 + the last digit of the binary)%5
  - The current bit is to tell which number mod 5 = 0/1/2/3/4
  - The rest is very easy to calculate.



n*2%5 == n%5*2%5

3.pre The connection between the SDT we've covered.(The reverse process)

Every Digit is a state to take care of.

i. Write the syntax-directed translation scheme (SDT) with S-attributed definition.

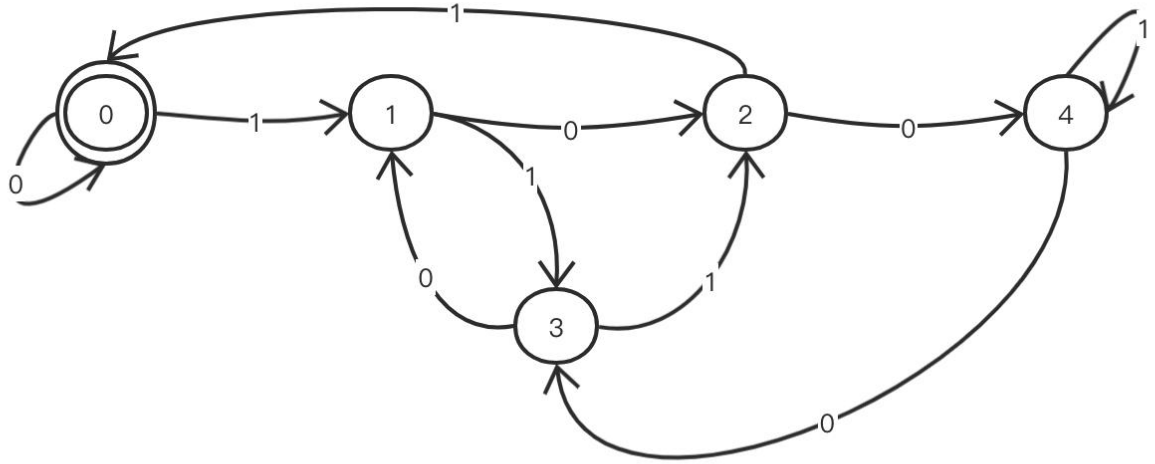
$$
\begin{array}{rcll}
S & \rightarrow & A.B & \{S.val = A.val + B.val\} \\
S & \rightarrow & A & \{S.val = A.val\} \\
A & \rightarrow & A_1\textbf{digit} & \{A.val = A_1.val * 2 + \textbf{digit}.val\} \\
A & \rightarrow & \textbf{digit} & \{A.val = \textbf{digit}.val\} \\
B & \rightarrow & \textbf{digit}B_1 & \{B.val = B_1.val/2 + \textbf{digit}.val/2\} \\
B & \rightarrow & \textbf{digit} & \{B.val = \textbf{digit}.val/2\} \\
\textbf{digit} & \rightarrow & 0 & \{\textbf{digit}.val = 0\} \\
\textbf{digit} & \rightarrow & 1 & \{\textbf{digit}.val = 1\}
\end{array}
$$

ii. Write the syntax-directed translation scheme (SDT) with L-attributed definition.

$$
\begin{array}{rcl}
S & \rightarrow & \{A.in = 0\}A.B\{S.syn = A.syn + B.syn\} \\
S & \rightarrow & \{A.in = 0\}A\{S.syn = A.syn\} \\
A & \rightarrow & \textbf{digit}\{A_1.in = A.in * 2 + \textbf{digit}.val\}A_1\{A.syn = A_1.syn\} \\
A & \rightarrow & \textbf{digit}\{A.syn = A.in * 2 + \textbf{digit}.val\} \\
B & \rightarrow & \textbf{digit}B_1\{B.syn = B_1.syn/2 + \textbf{digit}.val/2\} \\
B & \rightarrow & \textbf{digit}\{B.syn = \textbf{digit}.val/2\} \\
\textbf{digit} & \rightarrow & 0\{\textbf{digit}.val = 0\} \\
\textbf{digit} & \rightarrow & 1\{\textbf{digit}.val = 1\}
\end{array}
$$

3.a Easy transformation

- How many state should the CFG have? mod 5: 5.
- The accepting state is the starting point.
  - The current state is to accept one digit and go into the next one.
  - Remind the starting point can accept ε



Solution: $S \to 0S|1A|\varepsilon$

$A \to 0B \mid 1C$

$B \to 1S \mid 0D$

$C \to 0A \mid 1B$

$D \to 0C \mid 1D$

3.b First sets and Follow sets

- All the follow set is {$}
  - They only can look ahead $. nonterminal->terminal nonterminal

- S's first set is {0,1,ε }, Every body else's first set is {0,1}

	0	1	$
S	$S \to 0S$	$S \to 1A$	$S \to \epsilon$
A	$A \to 0B$	$A \to 1C$	
B	$B \to 0D$	$B \to 1S$	
C	$C \to 0A$	$C \to 1B$	
D	$D \to 0C$	$D \to 1D$	

3.c How to write code in Rust?

```rust
impl Solution {
    pub fn prefixes_div_by5(a: Vec<i32>) -> Vec<bool> {
        let mut state: i32 = 0;
        let mut result = vec![];
        let stateSet = [[0, 1], [2, 3], [4, 0], [1, 2], [3, 4]];
        for i in a {
            state = stateSet[state as usize][i as usize];
            result.push(state == 0);
        }
        result
    }
}
```

3.c How to write recursive predictive parsing program?

Solution:

```
void match(terminal t){
    if (lookahead==t) lookahead = nextToken();
    else error();
}
void S(){
    if (lookahead=='0'){match("0");S();}
    else if(lookahead=='1'){match("1");A();}
    else if(lookahead=='$'){succeed();}
    else error();
}
void A(){
    if (lookahead=='0'){match("0");B();}
    else if(lookahead=='1'){match("1");C();}
    else error();
}
void B(){
    if (lookahead=='0'){match("0");D();}
    else if(lookahead=='1'){match("1");S();}
    else error();
}
void C(){
    if (lookahead=='0'){match("0");A();}
    else if(lookahead=='1'){match("1");B();}
    else error();
}
void D(){
    if (lookahead=='0'){match("0");C();}
    else if(lookahead=='1'){match("1");D();}
    else error();
}
```

3.c How to write recursive descent program?

- #define ACC 1
- bool term(token tok) {return *ptr++ ==tok;}
- bool S() {return (term('1') && A()) || (term('0') && S()) || ACC;}
- bool A() {return (term('1') && C()) || (term('0') && B()); }
- bool B() {return (term('1') && S()) || (term('0') && D()); }
- bool C() {return (term('1') && B()) || (term('0') && A()); }
- bool D() {return (term('1') && D()) || (term('0') && C()); }

Reduce branches

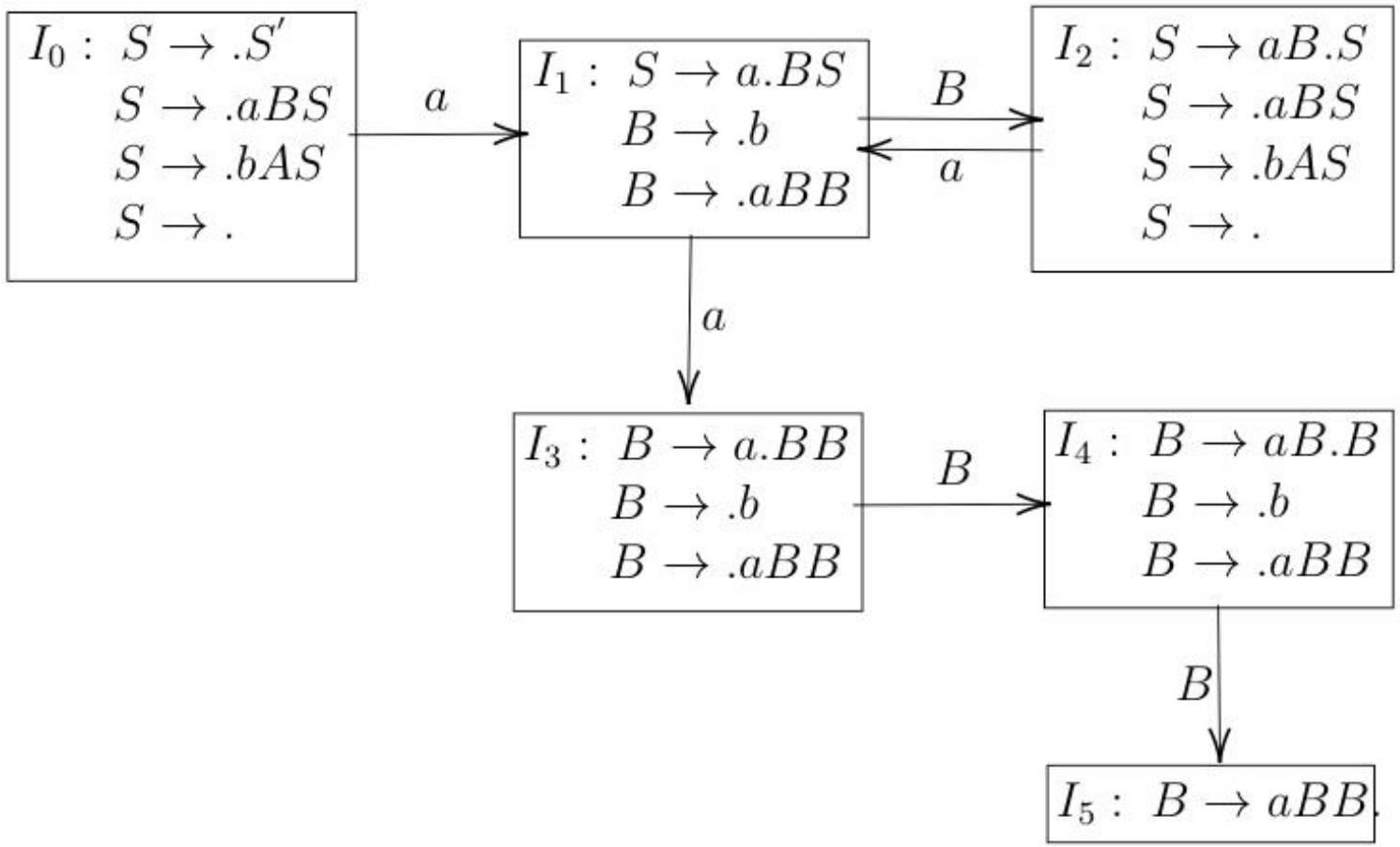
4.a To tell a grammar ambiguous

Solution: G_2 is ambiguous, because it has 2 left most deriviation.

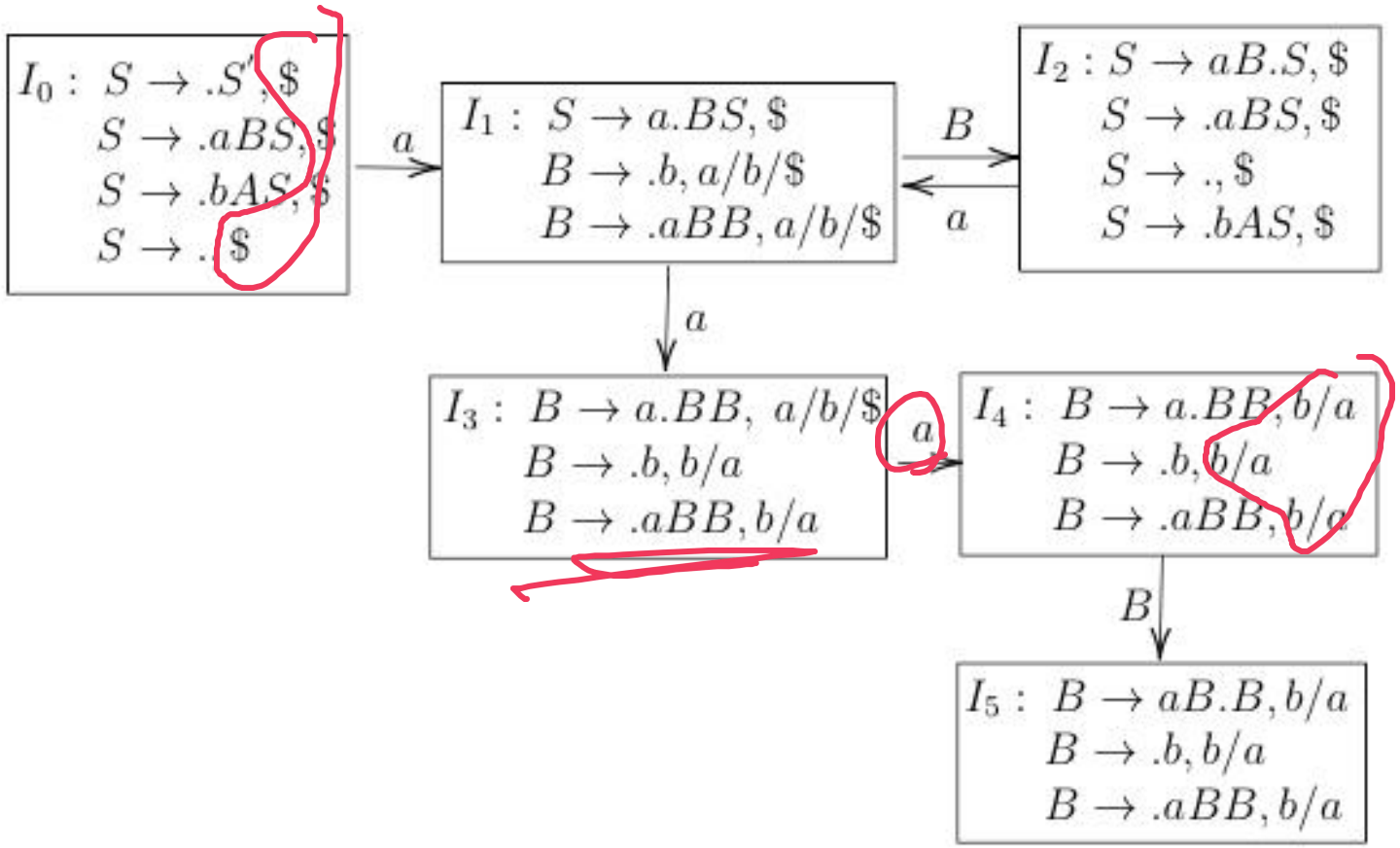
$S =>_{lm} aB =>_{lm} aaBB =>_{lm} aabSB =>_{lm} aabB =>_{lm} aabaBB =>_{lm}$
$aababSB =>_{lm} aababB =>_{lm} aababbS =>_{lm} aababb$

$S =>_{lm} aB =>_{lm} aaBB =>_{lm} aabSB =>_{lm} aabaBB =>_{lm} aababSB =>_{lm}$
$aababB =>_{lm} aababbS =>_{lm} aababb$

4.b To accept a input word - aBaaBB



4.b To accept a input word - aBaaaB



For DFA, no need to accept the input word.

4.c To write LALR(1)

Solution:

state	Action Table			Goto Table			
	a	b	\$	S'	S	A	B
0	s_2	s_3	r_2		1		
1			acc				
2	s_6	s_5					4
3	s_8	s_9				7	
4	s_2	s_3	r_2		10		
5	r_6	r_6	r_6				
6	s_6	s_5					11
7	s_2	s_3	r_2		12		
8	r_4	r_4	r_4				
9	s_8	s_9				13	
10			r_1				
11	s_6	s_5					14
12			r_3				
13	s_8	s_9				15	
14	r_7	r_7	r_7				
15	r_5	r_5	r_5				

5. Basic Block

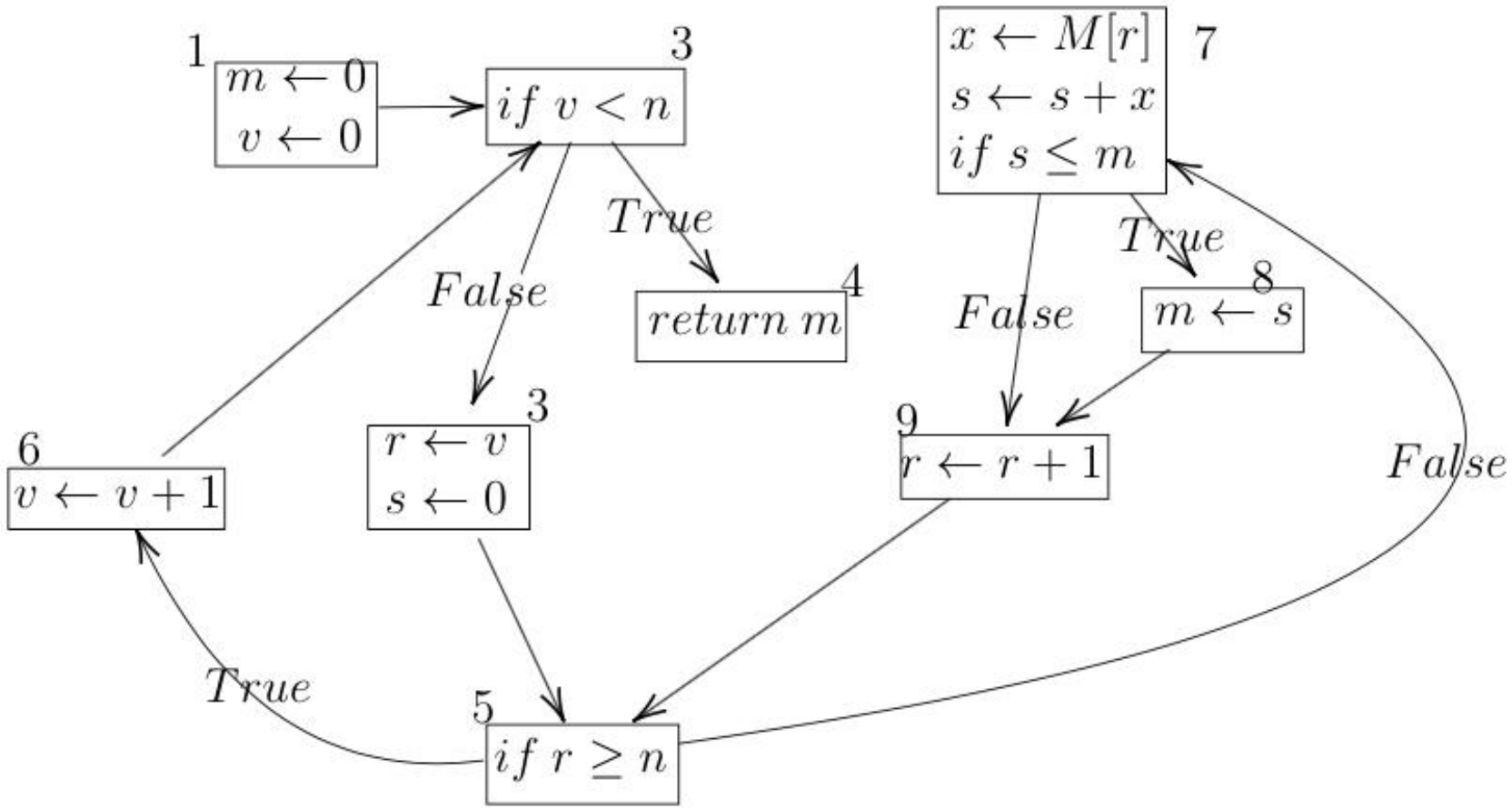
- Have the same expressing ability with SSA(Static Single Assignment), CFG(Control Flow Graph), but have different uses. The former for type checking, higher level optimization, the latter for static analysis.
- LLVM IR is partially SSA.

A **Basic Block** is a consecutive sequence of Statements $S_1, \ldots, S_n$, where flow must enter this block only at S_1, AND if S_1 is executed, then $S_2, \ldots, S_n$ are executed strictly in that order, unless one Statement causes halting.

- The **Leader** is the first Statement of a Basic Block
- A **Maximal Basic Block** is a maximal-length Basic Block

SSA means every variable will only be assigned value ONCE (therefore *single*). Useful for various kinds of optimizations.

5. Basic Block to SSA



Solution:

m_1=0
v_1=0
3: v_3=$\phi(v_1, v_2)$
m_3=$\phi(m_1, m_2)$
if v_3<n
    return m_3
r_1=v_3
s_1=0
5: r_3=$\phi(r_1, r_2)$
if r_3
    v_2=v_3+1
    goto 3
7: x_1=M[r_3]
s_2=s_1+x_1
if $s_2 \leq m_1$
    m_2=s_2
9: r_2=r_1+1
goto 5