CS314 Fall 2018 Assignment3 Solution

September 2018

1 (a)

FIRST sets for left hand sides:

FIRST(<program>)={def}
FIRST(<arguments>)={(}
FIRST(<funcname>)={f,g}
FIRST(<morevars>)={,,ϵ}
FIRST(<block>)=FIRST(<stmtlist>)={\t}
FIRST(<morestmts>)={\n,ϵ}
FIRST(<stmt>)={FIRST(<assign>), FIRST(<ifstmt>), FIRST(<returnstmt>)} ={a,b,c,if,return}
FIRST(<assign>)=FIRST(<variable>)={a,b,c}
FIRST(<condition>)=FIRST(<variable>)={a,b,c}
FIRST(<ifstmt>)={if}
FIRST(<returnstmt>)={return}
FIRST(<expr>)=FIRST(<term>)={a,b,c,0,1,2}
FIRST(<term>)={FIRST(<variable>), FIRST(<digit>)}={a,b,c,0,1,2}
FIRST(<variable>)={a,b,c}
FIRST(<digit>)={0,1,2}

FIRST sets for right hand sides:

FIRST(def<funcname> <arguments>:\n<block>EOF)={def}
FIRST(f)={f}
FIRST(g)={g}
FIRST((<variable> <morevars>))={(}
FIRST(,<variable> <morevars>)={,}
FIRST(\t<stmt> <morestmts>)={\t}
FIRST(\n<stmtlist>)={\n}
FIRST(if<condition>:<assign>\n\t else :<assign>)={if}
FIRST(return<variable>)={return}
FIRST(a)={a}
FIRST(b)={b}
FIRST(c)={c}
FIRST(0)={0}
FIRST(1)={1}
FIRST(2)={2}

FOLLOW sets:

FOLLOW(<funcname>)=FIRST(<arguments>)={(}
FOLLOW(<arguments>)={:}
FOLLOW(<morevars>)={)}
FOLLOW(<block>)={EOF}
FOLLOW(<stmslist>)=FOLLOW(<block>)={EOF}
FOLLOW(<morestmts>)=FOLLOW(<stmslist>)={EOF}
FOLLOW(<stmt>)={FIRST(<morestmts>), FOLLOW(<morestmts>)}={\n, EOF}
FOLLOW(<assign>)=FOLLOW(<stmt>)={\n, EOF}

FOLLOW(<condition>)={:}
FOLLOW(<ifstmt>)=FOLLOW(<stmt>)={\n, EOF}
FOLLOW(<returnstmt>)=FOLLOW(<stmt>)={\n, EOF}
FOLLOW(<expr>)={FOLLOW(<assign>), FOLLOW(<condition>)}={\n, EOF, :}
FOLLOW(<term>)={+, FOLLOW(<expr>)}={+, \n, EOF, :}
FOLLOW(<variable>)=
{=, <=, FOLLOW(<term>), FIRST(<morevars>)-ϵ, FOLLOW(<morevars>, FOLLOW(<returnstmt>))}
={=, <= , +, \n, EOF, :, , ,)}
FOLLOW(<digit>)={FOLLOW(<term>)}={+, \n, EOF, :}
PREDICT sets:
(1)PREDICT(<program>::=def<funcname> <arguments>:\n<block>EOF)={def}
(2)PREDICT(<funcname>::=f)={f}
(3)PREDICT(<funcname>::=g)={g}
(4)PREDICT(<arguments>::=(<variable> <morevars>))={(}
(5)PREDICT(<morevars>::=,<variable> <morevars>)={,}
(6)PREDICT(<morevars>::=ϵ)=FOLLOW(<morevars>)={)}
(7)PREDICT(<block>::=<stmtlist>)=FIRST(<stmtlist>)={\t}
(8)PREDICT(<stmtlist>::=\t<stmt> <morestmts>)={\t}
(9)PREDICT(<morestmts>::=\n<stmtlist>)={\n}
(10)PREDICT(<morestmts>::=ϵ)=FOLLOW(<morestmts>)={EOF}
(11)PREDICT(<stmt>::=<assign>)=FIRST(<assign>)={a, b, c}
(12)PREDICT(<stmt>::=<ifstmt>)=FIRST(<ifstmt>)={if}
(13)PREDICT(<stmt>::=<returnstmt>)=FIRST(<returnstmt>)={return}
(14)PREDICT(<assign>::=<variable>=<expr>)=FIRST(<variable>)={a,b,c}
(15)PREDICT(<condition>::=<variable> <=<expr>)=FIRST(<variable>)={a,b,c}
(16)PREDICT(<ifstmt>::=if<condition>:<assign>\n\t else :<assign>)={if}
(17)PREDICT(<returnstmt>::=return<variable>)={return}
(18)PREDICT(<expr>::=<term>+<term>)=FIRST(<term>)={a, b, c, 0, 1, 2}
(19)PREDICT(<term>::=<variable>)=FIRST(<variable>)={a,b,c}
(20)PREDICT(<term>::=<digit>)=FIRST(<digit>)={0,1,2}
(21)PREDICT(<variable>::=a)={a}
(22)PREDICT(<variable>::=b)={b}
(23)PREDICT(<variable>::=c)={c}
(24)PREDICT(<digit>::=0)={0}
(25)PREDICT(<digit>::=1)={1}
(26)PREDICT(<digit>::=2)={2}

45pt: 15pts for FIRST of non-terminal symbols, 1pt for each; 4pts for FOLLOW(<morevars>) *and* FOLLOW(<morestmt>) *, 2pt for each; 26pts for PREDICT, 1pt for each*

2 (b)

	def	:	\n	EOF	f	g	(	)	,	\t	=	<=	if	else	return	+	a	b	c	0	1	2
<program>	(1)																					
<funcname>					(2)	(3)																
<arguments>							(4)															
<morevars>								(6)	(5)													
<block>										(7)												
<stmtlist>										(8)												
<morestmts>			(9)	(10)																		
<stmt>													(12)		(13)		(11)	(11)	(11)			
<assign>																	(14)	(14)	(14)			
<condition>																	(15)	(15)	(15)			
<ifstmt>													(16)									
<returnstmt>															(17)							
<expr>																	(18)	(18)	(18)	(18)	(18)	(18)
<term>																	(19)	(19)	(19)	(20)	(20)	(20)
<variable>																	(21)	(22)	(23)			
<digit>																				(24)	(25)	(26)

Empty entities in the table imply error.
21pt: 1pt each row, additionally, a missed column should be penalized with 1pt, except columns without any entities

3 (c)

```
void main(){
    token = next_token();
    if (program()) {
        print("accept")
    } else {
        print("error");
    }
}

bool program() {
    if (token != def)
        return false;
    token = next_token();
    if (!funcname())
        return false;
    if (!arguments())
        return false;
    if (token != :)
        return false;
    token = next_token();
    if (token != \n)
        return false;
    token = next_token();
    if (!block())
        return false;
    if (token != EOF)
        return false;
    token = next_token();
    return true;
}

bool funcname() {
    switch(token) {
        case f:
        case g:
            token = next_token();
            return true;
        default:
            return false;
    }
}

bool arguments() {
    if (token != ()
        return false;
    token = next_token();
    if (!variable())
        return false;
```

```
    if (!morevars())
        return false;
    if (token != ))
        return false;
    token = next_token();
    return true;
}

bool morevars() {
    switch(token) {
        case ):
            return true;
        case ,:
            token = next_token();
            if (!variable())
                return false;
            return morevars()
        default:
            return false;
    }
}

bool block() {
    if (token == \t)
        return stmtlist();
    else
        return false;
}

bool stmtlist() {
    if (token != \t)
        return false;
    token = next_token();
    if (!stmt())
        return false;
    if (!morestmts())
        return false;
    return true;
}

bool morestmts() {
    switch(token) {
        case \n:
            token = next_token();
            return stmtlist();
        case EOF:
            return true;
        default:
            return false;
    }
}

bool stmt() {
    switch(token) {
```

```
        case if:
            return ifstmt();
        case return:
            return returnstmt();
        case a:
        case b:
        case c:
            return assign();
        default:
            return false;
    }
}

bool assign() {
    switch(token) {
        case a:
        case b:
        case c:
            if (!variable())
                return false;
            if (token != =)
                return false;
            token = next_token();
            if (!expr())
                return false;
            return true;
        default:
            return false;
    }
}

bool condition() {
    switch(token) {
        case a:
        case b:
        case c:
            if (!variable())
                return false;
            if (token != <=)
                return false;
            token = next_token();
            if (!expr())
                return false;
            return true;
        default:
            return false;
    }
}

bool ifstmt() {
    if (token != if)
        return false;
    token = next_token();
    if (!condition())
```

```
        return false;
    if (token != :)
        return false;
    token = next_token();
    if (!assign())
        return false;
    if (token != \n)
        return false;
    token = next_token();
    if (token != \t)
        return false;
    token = next_token();
    if (token != else)
        return false;
    token = next_token();
    if (token != :)
        return false;
    token = next_token();
    return assign()
}

bool returnstmt() {
    if (token != return)
        return false;
    token = next_token();
    return variable();
}

bool expr() {
    switch(token) {
        case a:
        case b:
        case c:
        case 0:
        case 1:
        case 2:
            if (!term())
                return false;
            if (token != +)
                return false;
            token = next_token();
            return term();
        default:
            return false;
    }
}

bool term() {
    switch(token) {
        case a:
        case b:
        case c:
            return variable();
        case 0:
```

```
        case 1:
        case 2:
            return digit();
        default:
            return false;
    }
}

bool variable() {
    switch(token) {
        case a:
        case b:
        case c:
            token = next_token();
            return true;
        default:
            return false;
    }
}

bool digit() {
    switch(token) {
        case 0:
        case 1:
        case 2:
            token = next_token();
            return true;
        default:
            return false;
    }
}
```

34pt: 2pt for each function