

PROJECT 2: PART C

DUE DATE: 19 APRIL

COS341 2019

INTRODUCTION

- ▶ You must once again perform semantic analysis on your SPL program. For this part you must ultimately determine whether variables have been assigned a value before they are used.
- ▶ You will be crawling your AST to guarantee that every use of a variable has a value at its usage point. If not you must display an error and continue checking to find all errors.

VALUE CHECKING RULES

- ▶ The following slides will define the rules for your value checker.
- ▶ Each rule will be defined in terms of a grammar production where certain terminal and non-terminals will be given value checking rules.

Note: See the specification for project 1 part B for the definition of a production

PROGRAM

- ▶ **PROG** → **CODE** ; **PROC_DEFS**
 - ▶ if some variable v_i has a value before some call to p_i in **CODE** then v_i also has a value in the definition of p_i inside **PROC_DEFS**
- ▶ **CODE** → **INSTR** ; **CODE**
 - ▶ if some v_i in **INSTR** has a value then it also has a value in **CODE**

CALL

- ▶ **CALL** → *userDefinedIdentifier*
 - ▶ Value checking must be applied to the procedure referred to by *userDefinedIdentifier* as if the procedure were inlined.

IO

- ▶ $IO \rightarrow \text{input} (VAR)$
 - ▶ VAR gets a value
- ▶ $IO \rightarrow \text{output} (VAR)$
 - ▶ VAR must have a value

ASSIGNMENT

- ▶ $\text{ASSIGN} \rightarrow \text{VAR} = \text{stringLiteral}$
 - ▶ VAR gets a value
- ▶ $\text{ASSIGN} \rightarrow \text{VAR} = \text{VAR}^*$
 - ▶ VAR^* must have a value then VAR gets a value
- ▶ $\text{ASSIGN} \rightarrow \text{VAR} = \text{NUMEXPR}$
 - ▶ NUMEXPR must have a value then VAR gets a value
- ▶ $\text{ASSIGN} \rightarrow \text{VAR} = \text{BOOL}$
 - ▶ BOOL must have a value then VAR gets a value

NUMEXPR

- ▶ **NUMEXPR** → **VAR**
 - ▶ if **VAR** has a value then **NUMEXPR** gets a value
- ▶ **NUMEXPR** → *integerLiteral*
 - ▶ **NUMEXPR** gets a value
- ▶ **NUMEXPR** → **CALC**
 - ▶ if **CALC** has a value then **NUMEXPR** gets a value

CALC

- ▶ $CALC \rightarrow add (NUMEXPR , NUMEXPR)$
 - ▶ if both $NUMEXPR$ have a value then $CALC$ gets a value
- ▶ $CALC \rightarrow sub (NUMEXPR , NUMEXPR)$
 - ▶ if both $NUMEXPR$ have a value then $CALC$ gets a value
- ▶ $CALC \rightarrow mult (NUMEXPR , NUMEXPR)$
 - ▶ if both $NUMEXPR$ have a value then $CALC$ gets a value

COND_BRANCH

- ▶ COND_BRANCH \rightarrow if (BOOL) then { CODE }
- ▶ BOOL must have a value
- ▶ COND_BRANCH \rightarrow if (BOOL) then { CODE } else { CODE }
- ▶ BOOL must have a value

BOOL

- ▶ $\text{BOOL} \rightarrow \text{eq}(\text{VAR}, \text{VAR})$
 - ▶ if both VAR have a value then BOOL gets a value
- ▶ $\text{BOOL} \rightarrow (\text{VAR} < \text{VAR})$
 - ▶ if both VAR have a value then BOOL gets a value
- ▶ $\text{BOOL} \rightarrow (\text{VAR} > \text{VAR})$
 - ▶ if both VAR have a value then BOOL gets a value

BOOL

- ▶ $\text{BOOL} \rightarrow \text{not } \text{BOOL}^*$
 - ▶ if BOOL^* has a value then BOOL gets a value
- ▶ $\text{BOOL} \rightarrow \text{and} (\text{BOOL}^* , \text{BOOL}^*)$
 - ▶ if both BOOL^* have a value then BOOL gets a value
- ▶ $\text{BOOL} \rightarrow \text{or} (\text{BOOL}^* , \text{BOOL}^*)$
 - ▶ if both BOOL^* have a value then BOOL gets a value

BOOL

- ▶ **BOOL** → **T**
 - ▶ **BOOL** gets a value
- ▶ **BOOL** → **F**
 - ▶ **BOOL** gets a value
- ▶ **BOOL** → **VAR**
 - ▶ if **VAR** has a value then **BOOL** gets a value

COND LOOP

- ▶ **COND_LOOP** → `while (BOOL) { CODE }`
 - ▶ **BOOL** must have a value
- ▶ **COND_LOOP** → `for(VAR=0;VAR*<VAR*;VAR*=add(VAR*;1))
{CODE}`
 - ▶ all **VAR*** must have a value

EXAMPLE (1)

- ▶ The following program has only one error:

```
num a;  
a = 10;
```

```
num c; num d; num i;  
for (i = 0; i < a; i = add(i, 1)) {  
    c = 11  
};
```

```
while ((a < c)) { d = 0 };
```

```
num b;
```

```
p;
```

```
output(b)
```

```
proc p {
```

```
    b = 10
```

```
}
```

← This is an error since **c** might not have a value

← No error. **b** was assigned a value in the call to **p**

EXAMPLE (2)

- ▶ The following program has no errors:

```
bool a;  
input(a);  
num b;  
  
if (a) then {  
    b = 10  
} else {  
    b = 3  
}
```

```
output(b)
```

← No error: **b** has a value since all paths provide a value for b.

YOUR TASK

- ▶ Assume all productions and terminals have no value at the start. Apply the rules given to you above. **Only** the rules that are described with “... **must have a value** ...” should produce an error, such as:
 - ▶ VALUE ERROR [line: 4, col: 3]: cannot assign undefined value to variable
 - ▶ VALUE ERROR [line: 6, col: 7]: if statement condition cannot be undefined
- ▶ You must output messages for all of these errors that occur in the program.
- ▶ You must also mark all nodes that don't have a value in your symbol table with “No-Value” and all nodes with a value must be marked with “Has-Value”.

See slides 6,7,10 and 14 for rules with that produce error messages.

ADDITIONAL NOTES

- ▶ If you changed your grammar from project 1 part B you must find a way to adapt the given rules to you in this spec to match your modified grammar.
- ▶ Plagiarism is not allowed! You or your group may not use any code written by someone not within your own group.