PROJECT Specification

Task Booklet 04

COS341 2020

INTRODUCTION

- You must once again perform static semantic analysis on your SPL program's AST. For this part you must ultimately determine whether variables have been assigned a **value** before they are used.
- You will be crawling your AST again to guarantee that every use of a variable has a value at its usage point. Otherwise 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.

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 → input (VAR)
  - VAR **gets** a value at this point
- IO → output (VAR)
  - VAR must have a value

ASSIGNMENT

- ASSIGN → VAR = *stringLiteral*
  - VAR **gets** a value at this point
- ASSIGN → VAR = VAR*
  - VAR* must have a value, consequently VAR gets a value
- ASSIGN → VAR = NUMEXPR
  - NUMEXPR must have a value, consequently VAR gets a value
- ASSIGN → VAR = BOOL
  - BOOL must have a value, consequently 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 → add (NUMEXPR , NUMEXPR)
  - if both NUMEXPR have a value then CALC gets a value
- CALC → sub (NUMEXPR , NUMEXPR)
  - if both NUMEXPR have a value then CALC gets a value
- CALC → mult (NUMEXPR , NUMEXPR)
  - if both NUMEXPR have a value then CALC gets a value

COND_BRANCH

- COND_BRANCH → if (BOOL) then { CODE }
  - BOOL must have a value
- COND_BRANCH → if (BOOL) then { CODE } else { CODE }
  - BOOL must have a value

BOOL

- BOOL → eq (VAR , VAR)
  - if both VAR have a value then BOOL gets a value
- BOOL → (VAR < VAR)
  - if both VAR have a value then BOOL gets a value
- BOOL → (VAR > VAR)
  - if both VAR have a value then BOOL gets a value

BOOL

- BOOL → not BOOL*
  - if BOOL* has a value then BOOL gets a value
- BOOL → and (BOOL* , BOOL*)
  - if both BOOL* have a value then BOOL gets a value
- BOOL → or (BOOL* , BOOL*)
  - if both BOOL* have a value then BOOL gets a value

BOOL

- BOOL → T
  - BOOL **gets** a value at this point
- BOOL → F
  - BOOL **gets** a value at this point
- 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 no errors and no variables without a value:

```
num a; bool b;
input(b);
if (b) then {
  a = 5
} else {
  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
}
```

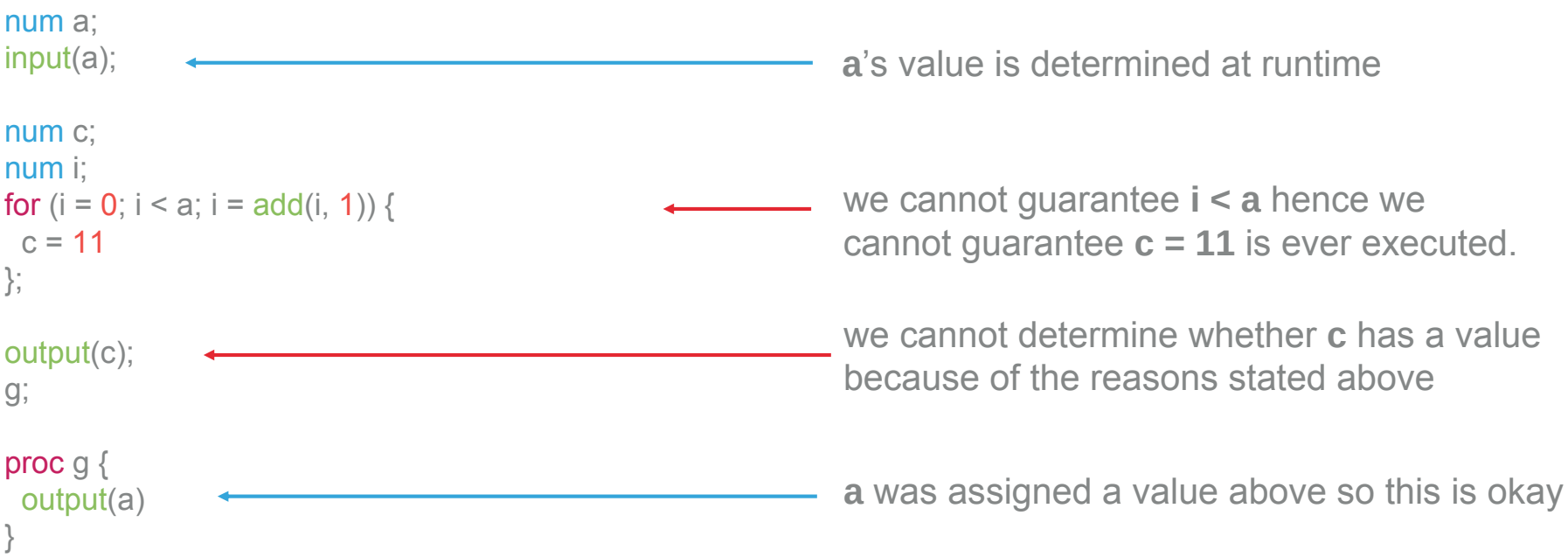
0 < (5 or 10) so this will be executed at least once

(5 or 10) < 11 so this will be executed at least once

b was assigned a value in the call to **p**

EXAMPLE (2)

- The following program has one **error** and has one variable without a known value the rest of the program is valid:



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 for the parser, 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.

And now: HAPPY CODING ☺ ☺ ☺