Ryan Hays

Prof. Bowers

CPSC 326

1/29/2019

HW2

1. S -> { } | { { } } | { { s } s

S -> { } | { } { } | { s { s } s }

1. This grammar will produce strings that begin with an ‘a’ or a ‘b’. If it begins with an ‘a’, it can recursive call ‘s’ again to output more ‘a’s, call ‘t’ to output any number of ‘b’s, or it could output an empty variable. Then after any number of recalls is finished, it will end with another a. Starting from the ‘t’ grammar will output a ‘b’ and recursively output any number of more b’s. Starting with the ‘b’ will result in strings made only of b’s.

left-most

abaaa

s -> a *s* a | a *t* a *s* | e (e is empty)

t -> b*t* | b

s -> a *t* a *s* [t -> b]

s -> a b a s [s -> a s a]

s -> a b a a s a [s -> e]

s -> a b a a a

right-most

abaaa

s -> a *s* a | a *t* a *s* | e (e is empty)

t -> b*t* | b

s -> a t a s [s -> a s a]

s-> a t a a s a [s -> e]

s -> a t a a a [t -> b]

s -> a b a a a

1. a -> p

b -> q

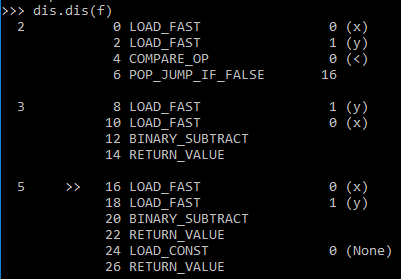
c -> or

d -> and not

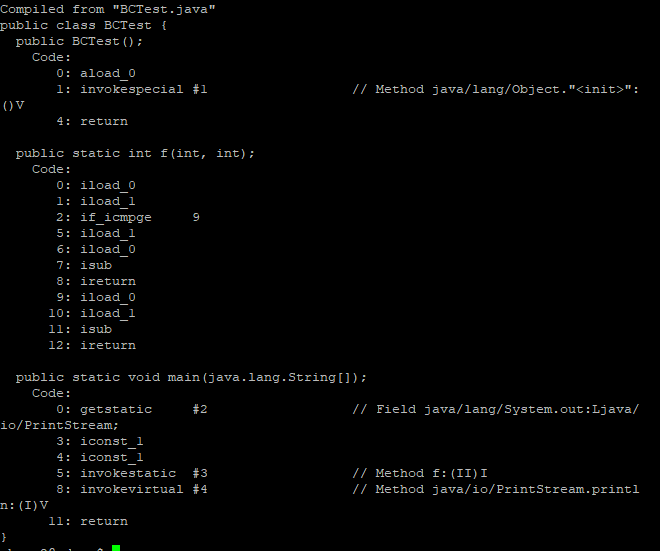
e -> and

f -> r

g -> s

1. 

This is bytecode which is an implementation detail of the python interpreter. This appears to be similar to machine code. It loads two variables into a local scope, compares the two variables by means of a less than (<) sign, and jumps to line 16 if it returns false. Otherwise it subtracts x from y and returns the value. If it took the jump it will instead subtract y from x and return the value.

1. 

This code looks similar but longer than the one done in python. We can see the byteload loading two variables inside the static int f function. Then it calls a comparison which I would image also branches to another line based on the result. One branch then calls a subtract on the variables and a return. Then I’m assuming the other end of the branch is reloading the variables and then calling subtract and return. In main it appears to call the static function and the two constants ‘1’ we used to pass it.