Programming Assignment 5

Leo Kivikunnas 525925, Jaakko koskela 526050

April 2020

1 Augmented LL(1) Grammar

Program −> #START Declaration−list #END
Declaration−list −> Declaration Declaration−list | EPSILON
Declaration −> Declaration−initial Declaration−prime
Declaration−initial −> Type−specifier #PID ID
Declaration−prime −> #FUNCTION Fun−declaration−prime |
    #VARIABLE Var−declaration−prime
Var−declaration−prime −> ; | #ARRAY [#ARRAY_SIZE NUM] ;
Fun−declaration−prime −> #BEGINSCOPE (#START_PARAM_COUNTER
    Params #STOP_PARAM_COUNTER) Compound−stmt #ENDSCOPE
Type−specifier −> int | void
Params −> int #PID ID Param−prime Param−list | void
    Param−list−void−abtar
Param−list−void−abtar −> #PID ID Param−prime Param−list |
    EPSILON
Param−list −> , Param Param−list | EPSILON
Param −> Declaration−initial Param−prime
Param−prime −> #ARRAY #ARRAY_PARAM [ ] | #PARAM EPSILON
Compound−stmt −> { Declaration−list #STATEMENTS_BEGIN
    Statement−list }
Statement−list −> Statement Statement−list | EPSILON
Statement −> Expression−stmt | Compound−stmt | Selection−stmt
    | Iteration−stmt | Return−stmt | Switch−stmt
Expression−stmt −> #START_TYPE_CHECK Expression #TYPE_CHECK ;
    | #CONTINUE continue ; | #BREAK break ; | #TYPE_CHECK ;
Selection−stmt −> if (#START_TYPE_CHECK Expression
    #TYPE_CHECK_IN_BRACKETS) #SAVE Statement else #JPF_SAVE
    Statement #JP
Iteration−stmt −> while #ENTER_WHILE (#START_TYPE_CHECK
    Expression #TYPE_CHECK_IN_BRACKETS) #SAVE Statement
    #EXIT_WHILE
Return−stmt −> return Return−stmt−prime #RETURN
Return−stmt−prime −> ; #EMPTY_RETURN | #START_TYPE_CHECK
    Expression #TYPE_CHECK ;
Switch−stmt −> switch #ENTER_SWITCH_CASE (#START_TYPE_CHECK
    Expression #TYPE_CHECK_IN_BRACKETS) { Case−stmts

Default−stmt } #EXIT_SWITCH_CASE

Case−stmts −> Case−stmt Case−stmts | EPSILON

Case−stmt −> case #IMMEDIATE NUM #SAVE_CASE : Statement−list

Default−stmt −> default : #DEFAULT_CASE Statement−list |
    EPSILON

Expression −> Simple−expression−zegond | #USE_PID
    #ADD_TO_TYPE_CHECK ID B

B −> #NOT_FUNCTION_CALL #ASSIGNMENT_CHAIN = Expression
    #ASSIGN | #NOT_FUNCTION_CALL [#INDEXING
    #START_TYPE_CHECK Expression #TYPE_CHECK #INDEXING_DONE]
    H | Simple−expression−prime

H −> #ASSIGNMENT_CHAIN = Expression #ASSIGN | G D C

Simple−expression−zegond −> Additive−expression−zegond C

Simple−expression−prime −> Additive−expression−prime C

C −> Relop Additive−expression #RELOP | EPSILON

Relop −> < #LT | == #EQ

Additive−expression −> Term D

Additive−expression−prime −> Term−prime D

Additive−expression−zegond −> Term−zegond D

D −> Addop Term #ADDOP D | EPSILON

Addop −> #PLUS + | #MINUS −

Term −> Factor G

Term−prime −> Factor−prime G

Term−zegond −> Factor−zegond G

G −> * Factor #MULT G | EPSILON

Factor −> (#START_TYPE_CHECK Expression
    #TYPE_CHECK_IN_BRACKETS) | #USE_PID #ADD_TO_TYPE_CHECK
    ID Var−call−prime | #ADD_TO_TYPE_CHECK #IMMEDIATE NUM

Var−call−prime −> #FUNCTION_CALL (#START_ARGUMENT_COUNTER
    Args #STOP_ARGUMENT_COUNTER) #FUNCTION_CALLED |
    #NOT_FUNCTION_CALL Var−prime

Var−prime −> [#INDEXING #START_TYPE_CHECK Expression
    #TYPE_CHECK #INDEXING_DONE] | EPSILON

Factor−prime −> #FUNCTION_CALL (#START_ARGUMENT_COUNTER Args
    #STOP_ARGUMENT_COUNTER) #FUNCTION_CALLED |
    #NOT_FUNCTION_CALL EPSILON

Factor−zegond −> (#START_TYPE_CHECK Expression
    #TYPE_CHECK_IN_BRACKETS) | #ADD_TO_TYPE_CHECK #IMMEDIATE
    NUM

Args −> Arg−list | EPSILON

Arg−list −> #ARGUMENT #START_TYPE_CHECK Expression
    #ARGUMENT_TO_PASS #TYPE_CHECK Arg−list−prime

Arg−list−prime −> , #ARGUMENT #START_TYPE_CHECK Expression
    #ARGUMENT_TO_PASS #TYPE_CHECK Arg−list−prime | EPSILON

2   Intermediate Code generation

Our intermediate code generator can handle all the compulsory requirements.
Additionally, we implemented code generation for nested scopes as specified in

the second extra requirement.