**1.<Program> ::= <Statement\_List>**

M([[ Statement\_List1]], m0) =

let

val m1 = M(Statement\_List1, m0)

in

m1

end

or M(stmt\_list1, m0)

**2. <Statement\_List> ::= <Statement> <Statement\_List>**

M([[ Statement1 Statement\_List1 ]], m0) =

let

val m1 = M(Statement1, m0)

val m2 = M(Statement\_List1, m1)

in

m2

end

or M(stmtList1, M(stmt1, m0))

**3.<Statement\_List> ::= <Epsilon>**

M([[ Epsilon1 ]] , m0) = m0

**4.<Statement> ::= <Declaration> “;”**

M([[ Declaration1 ; ]], m0) =

let

val m1 = M(Declaration1, m0)

in

m1

end

or M(declaration, m0)

**5.<Statement> ::= <Assignment> “;”**

M([[ Assignment1 ;]], m0) =

let

val m1 = M(Assignment1, m0)

in

m1

end

or M(Assignmnet1, m0)

**6.<Statement> ::= <Conditional\_Statement>**

M([[ Conditional\_Statement1 ]], m0 ) =

let

val m1 = M ( Conditional\_Statement1 , m0 )

in

m1

end

or M(Conditional1, m0)

**7.<Statement> ::= <Iterative\_Statement>**

M([[ Iterative\_Statement1 ]], m0 ) =

let

val m1 = M (Iterative\_Statement1 , m0 )

in

m1

end

or M(iterative1, m0)

**8.<Statement> ::= <Block\_Statement>**

M([[ Block\_Statement1 ]], m0 ) =

let

val m1 = M( Block\_Statement1 , m0 )

in

m1

end

**9. <Statement> ::= <PrePost> “;”**

M([[ PrePost1 ;]], m0 ) =

let

val m1 = M ( PrePost1, m0 )

in

m1

end

**10.<Statement> ::= <Print\_Statement> “;”**

M([[ Print\_Statement1 ; ]], m0 ) =

let

val m1 = M( Print\_Statement1 , m0 )

in

m1

end

**11.<Declaration> ::= “bool” <Id> “=” <Expression>**

M([[ bool Id1 = Expression1 ]], m0) =

let

val m1 = updateEnv(Id1, bool, new( ), m0)

val loc = getLoc(accessEnv(Id1, m1))

val (v1, m2) = E’(Expression1, m1)

val m3 = updateStore(loc, v1, m2)

in

m3

end

**12.<Declaration> ::= “int” <Id> “=” <Expression>**

M([[ int Id1 = Expression1 ]], m0) =

let

val m1 = updateEnv(Id1, int, new( ), m0)

val loc = getLoc(accessEnv(Id1, m1))

val (v1, m2) = E’(Expression1, m1)

val m3 = updateStore(loc, v1, m2)

in

m3

end

**13.<Assignment> ::= <Id> “=” <Expression>**

M([[ id1 = Expression1 ]], m0) =

let

val (v1 ,m1) = E’( Expression1, m0)

val loc = getLoc(accessEnv(id1 , m1))

val m2 = updateStore(loc, v1, m1)

in

m2

end

**14.<Conditional\_Statement> ::= “if” <Expression> “then” <Block\_Statement>**

M ([[ if Expression1 then Block\_Statement1 ]] , m0) =

let

val (v1 , m1) = E’ ( Expression1 , m0 )

in

if v1 then

let

val m2 = M (Block\_Statement1 , m1 )

in

m2

end

else

m1

end

**15.<Conditional\_Statement> ::= “if” <Expression> “then”**

**<Block\_Statement>“else”**

**<Block\_Statement>**

M([[ if Expression1 then Block\_Statement1 else Block\_Statement2 ]], m0) =

let

val (v1 , m1) = E’ ( Expression1 , m0 )

in

if v1 then

let

val m2 = M (Block\_Statement1 , m1 )

in

m2

end

else

let

val m3 = M ( Block\_Statement2 , m1)

in

m3

end

end

**16. <Iterative \_Statement> ::= <For\_Block>**

M([[ For\_Block1 ]], m0 =

let

val m1 = M ( For\_Block1 , m0 )

in

m1

End

**17. <Iterative \_Statement> ::= <While\_Block>**

M([[ While\_Block1 ]], m0) =

let

val m1 = M ( While\_Block1, m0 )

in

m1

end

**18.<For\_block> ::= “for” “(“<Initiation>“;” <Expression> ”;”**

**<PrePost>“)” <Block\_Statement>**

M([[ for (Initiation1; Expression1; PrePost1) Block\_Statement1 ]], m0) =

let

val m1 = M(Initiation1, m0)

val m2 = O(Expression1, Block\_Statement1, PrePost1, m1)

in

m2

end

O(Expression1, Block\_Statement1, PrePost1, m0) =

let

val (v1,m1) = E’(Expression1, m0)

in

if v1 then O(Expression1, Block\_Statement1, PrePost1, M(PrePost1,

M(Block\_Statement1, m1)))

else

m1

end

**19. <Initiation> ::= <Assignment>**

M([[ Assignment1]], m0) =

let

val m1 = M(Assignment1, m0)

in

m1

end

**20. <Initiation> ::= <Declaration>**

M([[ Declaration1  ]], m0) =

let

val m1 = M(Declaration1, m0)

in

m1

end

**21. <While\_Block> ::= “while” “(“<Expression>”)” <Block\_Statement>**

M ([[ while (Expression1) Block\_Statement1 ]], m0 ) = N ( Expression1 ,

Block\_Statement1 , m0 )

N ( Expression1 , Block\_Statement1 , m0 ) =

let

val (v1 , m1 ) = E’ ( Expression1 , m0 )

in

if v1 then N ( Expression1 , Block\_Statement1 ,

M( Block\_Statement1 , m1 ) )

else m1

end

**22. <Block\_Statement> ::= “{” <Statement\_List> “}”**

M([[ { Statement\_List1 } ]], (env0 , S0 )) =

let

val (env1, S1) = M(Statement\_List1,(env0 , S0))

in

(env0 , S1)

end

**23. <Print\_Statement> ::= “print” “(“ <Expression> “)”**

M ([[ print ( Expression1 ) ]], m0) =

let

val (v1 , m1 ) = E’ ( Expression1 , m0 )

print(v1)

in

m1

end

**24. <Expression> ::= <Expression> “or” <Conjunction>**

E’([[ Expression1 or Conjunction1 ]], m0) =

let

val (v1,m1) = E’(Expression1, m0)

in

if v1 then (v1, m1)

else

let

val (v2,m2) = E’(Conjunction1, m1)

in

(v2, m2)

end

end

**25.<Expression> ::= <Conjunction>**

E’([[ Conjunction1 ]], m0) =

let

val (v1,m1) = E’( Conjunction1, m0)

in

(v1, m1)

end

**26. <Conjunction> ::= <Conjunction> “and” <Equality>**

E’([[ Conjunction1 and Equality1 ]], m0) =

let

val (v1, m1) = E’( Conjunction1, m0)

in

if v1 then

let

val (v2, m2) = E’(Equality1, m1)

in

(v2, m2)

end

else

(v1, m1)

end

**27.<Conjunction> ::= <Equality>**

E’([[ Equality1 ]], m0) =

let

val (v1,m1) = E’(Equality1, m0)

in

(v1,m1)

end

**28. <Equality> ::= <Equality> “=” <Comparator>**

E’([[ Equality1 = Comparator1 ]],m0) =

let

val (v1, m1) = E’(Equality1,m0)

val (v2,m2) = E’(Comparator1,m1)

in

(v1=v2,m2)

end

**29. <Equality> ::= <Equality> “!=” <Comparator>**

E’([[ Equality1 != Comparator1 ]],m0) =

let

val (v1,m1) = E’(Equality1, m0)

val (v2,m2) = E’(Comparator1,m1)

in

(v1 != v2, m2)

end

**30.<Equality> ::= <Comparator>**

E’([[ Comparator1 ]], m0) =

let

val (v1, m1) = E’(Comparator1, m0)

in

(v1, m1)

end

**31. <Comparator> ::= <Add\_Sub> “>” <Add\_Sub>**

E’([[ Add\_Sub1 > Add\_Sub2 ]], m0) =

let

val (v1, m1) = E’(Add\_Sub1, m0)

val (v2, m2) = E’(Add\_Sub2, m1)

in

(v1 > v2, m2)

end

**32. <Comparator> ::= <Add\_Sub> “<” <Add\_Sub>**

E’([[ Add\_Sub1 < Add\_Sub2 ]], m0) =

let

val (v1, m1) = E’(Add\_Sub1, m0)

val (v2, m2) = E’(Add\_Sub2, m1)

in

(v1 < v2, m2)

end

**34. <Comparator> ::= <Add\_Sub>**

E’([[ Add\_Sub1 ]], m0) =

let

val (v1, m1) = E’(Add\_Sub1, m0)

in

(v1, m1)

end

**35. <Add\_Sub> ::= <Product>**

E’([[ Product1 ]], m0) =

let

val (v1, m1) = E’(Product1, m0)

in

(v1,m1)

end

**36. <Add\_Sub> ::= <Add\_Sub> “+” <Product>**

E’([[ Add\_Sub1 + Product1 ]], m0) =

let

val (v1, m1) = E’(Add\_Sub1,m0)

val (v2,m2) = E’(Product1,m1)

in

(v1 + v2,m2)

end

**37. <Add\_Sub> ::= <Add\_Sub> “-” <Product>**

E’([[ Add\_Sub1 - Product1 ]], m0) =

let

val (v1,m1) = E’(Add\_Sub1, m0)

val (v2,m2) = E’(Product1,m1)

in

(v1 - v2,m2)

end

**38. <Product> ::= <Product> “\*” <Negation>**

E’([[ Product1 \* Negation1]], m0) =

let

val (v1,m1) = E’(Product1,m0)

val (v2,m2) = E’(Negation1,m1)

in

(v1 \* v2, m2)

end

**39. <Product> ::= <Product> “div” <Negation>**

E’([[ Product1 div Negation1 ]], m0) =

let

val (v1,m1) = E’(Product1,m0)

val (v2,m2) = E’(Negation1,m1)

in

(v1 / v2, m2)

end

**40. <Product> ::= <Product> “mod” <Negation>**

E’([[ Product1 mod Negation1 ]], m0) =

let

val (v1,m1) = E’(Product1,m0)

val (v2,m2) = E’(Negation1,m1)

in

(v1 % v2, m2)

end

**41. <Product> ::= <Negation>**

E’([[ Negation1 ]], m0) =

let

val (v1,m1) = E’(Negation1, m0)

in

(v1,m1)

end

**42. <Negation> ::= “not” “(“<Negation>”)”**

E’([[ not(Negation1) ]], m0) =

let

val (v1,m1) = E’(Negation1,m0)

in

(not v1,m1)

end

**43. <Negation> ::= “-” <Negation>**

E’([[ - Negation1 ]], m0) =

let

val (v1,m1) = E’(Negation1,m0)

in

(-v1,m1)

end

**44. <Negation> ::= <Exponential>**

E’([[ Exponential1 ]], m0) =

let

val (v1,m1) = E’(Exponential1,m0)

in

(v1,m1)

end

**45. <Exponential> ::= <Base> “^” <Exponential>**

E’([[ Base1 ^ Exponential1 ]], m0) =

let

val (v1, m1) = E’(Exponential1, m0)

val (v2, m2) = E’(Base1, m1)

in

(exp(v2, v1), m2)

end

**46. <Exponential> ::= <Base>**

E’([[ Base1 ]], m0) =

let

val (v1, m1) = E’(Base1, m0)

in

(v1, m1)

end

**47. <Base> ::= <Integer\_Const>**

E’([[ Integer\_Const1 ]], m0) =

let

val v1 = Integer\_Const1

in

(v1, m0)

end

**48. <Base> ::= <Boolean\_Const>**

E’([[ Boolean\_Const1 ]], m0) =

let

val v1 = Boolean\_Const1

in

(v1,m0)

end

**49. <Base> ::= “(“ <Expression> “)”**

E’([[ ( Expression1 ) ]], m0) =

let

val (v1,m1) = E’(Expression1,m0)

in

(v1,m1)

end

**50. <Base> ::= “|” <Expression> “|”**

E’([[ | Expression1 | ]], m0) =

let

val (v1,m1) = E’(Expression1,m0)

in

if v1 < 0 then

(-v1,m1)

else

(v1,m1)

end

end

**51. <Base> ::= <PrePost>**

E’([[ ( PrePost1) ]], m0) =

let

val (v1,m1) = E’(PrePost1,m0)

in

(v1,m1)

end

**52. <Base> ::= <Id>**

E’ ([[ Id ]] , m0 ) =

let

val loc = getLoc (accessEnv( Id1, m0)

val v1 = accessStore( loc ,m0 )

in

(v1,m0)

end

**53. <PrePost> ::= “++” <Id>**

M ([[ ++ Id1 ]] , m0) =

let

val ( v1 , m1 ) = E’ ( [[id1]] , mo )

val loc = getLoc (accessEnv ( v1 , m1)

val m2 = updateStore (loc , v1+1, m1)

in

m2

end

E’ ([[ ++ Id1 ]] , m0) =

let

val ( v1 , m1 ) = E’ ( [[id1]] , mo )

val loc = getLoc (accessEnv ( v1 , m1)

val m2 = updateStore (loc , v1+1, m1)

in

(v1+1,m2)

End

**54. <PrePost> ::= “--” <Id>**

E’ ([[ -- Id1 ]] , m0) =

let

val ( v1 , m1 ) = E’ ( [[id1]] , mo )

val loc = getLoc (accessEnv ( v1 , m1)

val m2 = updateStore (loc , v1-1, m1)

in

(v1-1,m2)

end

**55. <PrePost> ::= <Id> “++”**

E’ ([[ Id1  ++ ]] , m0) =

let

val ( v1 , m1 ) = E’ ( [[id1]] , mo )

val loc = getLoc (accessEnv ( v1 , m1)

val m2 = updateStore (loc , v1+1, m1)

in

(v1,m2)

end

**56. <PrePost> ::= <Id> “--”**

E’ ([[ Id1 -- ]] , m0) =

let

val ( v1 , m1 ) = E’ ( [[id1]] , mo )

val loc = getLoc (accessEnv ( v1 , m1)

val m2 = updateStore (loc , v1-1, m1)

in

(v1,m2)

end

**57. <PrePost> ::= “++” <Id>**

M ([[ ++ Id1 ]] , m0) =

let

val ( v1 , m1 ) = E’ ( [[id1]] , mo )

val loc = getLoc (accessEnv ( v1 , m1)

val m2 = updateStore (loc , v1+1, m1)

in

m2

end

**58. <PrePost> ::= “--” <Id>**

M ([[ -- Id1 ]] , m0) =

let

val ( v1 , m1 ) = E’ ( [[id1]] , mo )

val loc = getLoc (accessEnv ( v1 , m1)

val m2 = updateStore (loc , v1-1, m1)

in

m2

end

**59. <PrePost> ::= <Id> “++”**

M ([[ Id1  ++ ]] , m0) =

let

val ( v1 , m1 ) = E’ ( [[id1]] , mo )

val loc = getLoc (accessEnv ( v1 , m1)

val m2 = updateStore (loc , v1+1, m1)

in

m2

end

**60. <PrePost> ::= <Id> “--”**

M ([[ Id1 -- ]] , m0) =

let

val ( v1 , m1 ) = E’ ( [[id1]] , mo )

val loc = getLoc (accessEnv ( v1 , m1)

val m2 = updateStore (loc , v1-1, m1)

in

m2

end

21. <Do\_While\_Block> ::= "do" <Block\_Statement> “while”

“(“<Expression>”)”

M ([[ do Block\_Statement1 while (Expression1)]], m0 ) = N ( Expression1,

Block\_Statement1 , m0 )

N ( Expression1 , Block\_Statement1 , m0 ) =

let

m1 = M(Block\_Statement1, m0)

val (v1 , m2 ) = E’ ( Expression1 , m1 )

in

if v1 then N ( Expression1 , Block\_Statement1 ,

M( Block\_Statement1 , m2 ) )

else m2

end