1.<Program> ::= <Statement_List>

M([[Statement_List$_1$]], m_0) =

let

val m_1 = M(Statement_List$_1$, m_0)

in

m_1

end

or M(stmt_list1, m0)

2. <Statement_List> ::= <Statement> <Statement_List>

M([[Statement$_1$ Statement_List$_1$]], m_0) =

let

val m_1 = M(Statement$_1$, m_0)

val m_2 = M(Statement_List$_1$, m_1)

in

m_2

end

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

3.<Statement_List> ::= <Epsilon>

M([[Epsilon$_1$]] , m_0) = m_0

4.<Statement> ::= <Declaration> “;”

M([[Declaration$_1$;]], m_0) =

let

val m_1 = M(Declaration$_1$, m_0)

in

m_1

end

or M(declaration, m0)

5.<Statement> ::= <Assignment> “;”

M([[Assignment$_1$;]], m_0) =

let

val m_1 = M(Assignment$_1$, m_0)

in

m_1

end

or M(Assignmnet1, m0)

6.<Statement> ::= <Conditional_Statement>

M([[Conditional_Statement$_1$]], m_0) =

let

val m_1 = M (Conditional_Statement$_1$, m_0)

in

m_1

end

or M(Conditional1, m0)

7.<Statement> ::= <Iterative_Statement>

M([[Iterative_Statement$_1$]], m_0) =

let

val m_1 = M (Iterative_Statement$_1$, m_0)

in

m_1

end

or M(iterative1, m0)

8.<Statement> ::= <Block_Statement>

M([[Block_Statement$_1$]], m_0) =

let

val m_1 = M(Block_Statement$_1$, m_0)

in

m_1

end

9. <Statement> ::= <PrePost> “;”

M([[PrePost$_1$;]], m_0) =

let

val m_1 = M (PrePost$_1$, m_0)

in

m_1

end

10.<Statement> ::= <Print_Statement> “;”

M([[Print_Statement$_1$;]], m_0) =

let

val m_1 = M(Print_Statement$_1$, m_0)

in

m_1

end

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

M([[bool Id_1 = $Expression_1$]], m_0) =

let

val m_1 = updateEnv(Id_1, bool, new( ), m_0)

val loc = getLoc(accessEnv(Id_1, m_1))

val (v_1, m_2) = E'($Expression_1$, m_1)

val m_3 = updateStore(loc, v_1, m_2)

in

m_3

end

12.<Declaration> ::= "int" <Id> "=" <Expression>

M([[int Id_1 = $Expression_1$]], m_0) =

let

val m_1 = updateEnv(Id_1, int, new( ), m_0)

val loc = getLoc(accessEnv(Id_1, m_1))

val (v_1, m_2) = E'($Expression_1$, m_1)

val m_3 = updateStore(loc, v_1, m_2)

in

m_3

end

13.<Assignment> ::= <Id> "=" <Expression>

M([[id_1 = $Expression_1$]], m_0) =

let

val (v_1 ,m_1) = E'($Expression_1$, m_0)

val loc = getLoc(accessEnv(id_1 , m_1))

val m2 = updateStore(loc, v_1, m_1)

in

m2

end

14.<Conditional_Statement> ::= "if" <Expression> "then" <Block_Statement>

M ([[if $Expression_1$ then $Block\_Statement_1$]] , m_0) =

let
val (v_1 , m_1) = E' (Expression$_1$, m_0)
in
if v_1 then
let
val m_2 = M (Block_Statement$_1$, m_1)
in
m_2
end
else
m_1
end

15.<Conditional_Statement> ::= "if" <Expression> "then" <Block_Statement>"else" <Block_Statement>

M([[if Expression$_1$ then Block_Statement$_1$ else Block_Statement$_2$]], m_0) =
let
val (v_1 , m_1) = E' (Expression$_1$, m_0)
in
if v_1 then
let
val m_2 = M (Block_Statement$_1$, m_1)
in
m_2
end
else
let
val m_3 = M (Block_Statement$_2$, m_1)
in
m_3
end
end

16. <Iterative _Statement> ::= <For_Block>

M([[For_Block$_1$]], m_0 =
let
val m_1 = M (For_Block$_1$, m_0)

in

m_1

End

17. <Iterative _Statement> ::= <While_Block>

M([[While_Block$_1$]], m_0) =

let

val m_1 = M (While_Block$_1$, m_0)

in

m_1

end

18.<For_block> ::= "for" "("<Initiation>";" <Expression> ";" <PrePost>")" <Block_Statement>

M([[for (Initiation$_1$; Expression$_1$; PrePost$_1$) Block_Statement$_1$]], m_0) =

let

val m_1 = M(Initiation$_1$, m_0)

val m_2 = O(Expression$_1$, Block_Statement$_1$, PrePost$_1$, m_1)

in

m_2

end

O(Expression$_1$, Block_Statement$_1$, PrePost$_1$, m_0) =

let

val (v_1,m_1) = E'(Expression$_1$, m_0)

in

if v_1 then O(Expression$_1$, Block_Statement$_1$, PrePost$_1$, M(PrePost$_1$, M(Block_Statement$_1$, m_1)))

else

m_1

end

19. <Initiation> ::= <Assignment>

M([[Assignment$_1$]], m_0) =

let

val m_1 = M(Assignment$_1$, m_0)

in

m_1

end

20. <Initiation> ::= <Declaration>

M([[Declaration$_1$]], m_0) =

let

val m_1 = M(Declaration$_1$, m_0)

in

m_1

end

21. <While_Block> ::= "while" "("<Expression>")" <Block_Statement>

M ([[while (Expression$_1$) Block_Statement$_1$]], m_0) = N (Expression$_1$, Block_Statement$_1$, m_0)

N (Expression$_1$, Block_Statement$_1$, m_0) =

let

val (v_1 , m_1) = E' (Expression$_1$, m_0)

in

if v_1 then N (Expression$_1$, Block_Statement$_1$, M(Block_Statement$_1$, m_1))

else m_1

end

22. <Block_Statement> ::= "{" <Statement_List> "}"

M([[{ Statement_List$_1$ }]], (env_0 , S_0)) =

let

val (env_1, S_1) = M(Statement_List$_1$,(env_0 , S_0))

in

(env_0 , S_1)

end

23. <Print_Statement> ::= "print" "(" <Expression> ")"

M ([[print (Expression$_1$)]], m_0) =

let

val (v_1 , m_1) = E' (Expression$_1$, m_0)

print(v_1)

in

m_1

end

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

E’([[$Expression_1$ or $Conjunction_1$]], m_0) =

let

val (v_1,m_1) = E’($Expression_1$, m_0)

in

if v_1 then (v_1, m_1)

else

let

val (v_2,m_2) = E’($Conjunction_1$, m_1)

in

(v_2, m_2)

end

end

25.<Expression> ::= <Conjunction>

E’([[$Conjunction_1$]], m_0) =

let

val (v_1,m_1) = E’($Conjunction_1$, m_0)

in

(v_1, m_1)

end

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

E’([[$Conjunction_1$ and $Equality_1$]], m_0) =

let

val (v_1, m_1) = E’($Conjunction_1$, m_0)

in

if v_1 then

let

val (v_2, m_2) = E'(Equality$_1$, m_1)
in
(v_2, m_2)
end
else
(v_1, m_1)
end

27.<Conjunction> ::= <Equality>

E'([[Equality$_1$]], m_0) =
let
val (v_1,m_1) = E'(Equality$_1$, m_0)
in
(v_1,m_1)
end

28. <Equality> ::= <Equality> "=" <Comparator>

E'([[Equality$_1$ = Comparator$_1$]],m_0) =
let
val (v_1, m_1) = E'(Equality$_1$,m_0)
val (v_2,m_2) = E'(Comparator$_1$,m_1)
in
(v_1=v_2,m_2)
end

29. <Equality> ::= <Equality> "!=" <Comparator>

E'([[Equality$_1$!= Comparator$_1$]],m_0) =
let
val (v_1,m_1) = E'(Equality$_1$, m_0)
val (v_2,m_2) = E'(Comparator$_1$,m_1)
in
(v_1 != v_2, m_2)
end

30.<Equality> ::= <Comparator>

E'([[Comparator$_1$]], m_0) =
let
val (v_1, m_1) = E'(Comparator$_1$, m_0)
in
(v_1, m_1)

end

31. <Comparator> ::= <Add_Sub> “>” <Add_Sub>

E’([[Add_Sub_1 > Add_Sub_2]], m_0) =

let

val (v_1, m_1) = E’(Add_Sub_1, m_0)

val (v_2, m_2) = E’(Add_Sub_2, m_1)

in

(v_1 > v_2, m_2)

end

32. <Comparator> ::= <Add_Sub> “<” <Add_Sub>

E’([[Add_Sub_1 < Add_Sub_2]], m_0) =

let

val (v_1, m_1) = E’(Add_Sub_1, m_0)

val (v_2, m_2) = E’(Add_Sub_2, m_1)

in

(v_1 < v_2, m_2)

end

34. <Comparator> ::= <Add_Sub>

E’([[Add_Sub_1]], m_0) =

let

val (v_1, m_1) = E’(Add_Sub_1, m_0)

in

(v_1, m_1)

end

35. <Add_Sub> ::= <Product>

E’([[$Product_1$]], m_0) =

let

val (v_1, m_1) = E’($Product_1$, m_0)

in

(v_1,m_1)

end

36. <Add_Sub> ::= <Add_Sub> "+" <Product>

E'([[Add_Sub_1 + $Product_1$]], m_0) =

let

val (v_1, m_1) = E'(Add_Sub_1,m_0)

val (v_2,m_2) = E'($Product_1$,m_1)

in

(v_1 + v_2,m_2)

end

37. <Add_Sub> ::= <Add_Sub> "-" <Product>

E'([[Add_Sub_1 - $Product_1$]], m_0) =

let

val (v_1,m_1) = E'(Add_Sub_1, m_0)

val (v_2,m_2) = E'($Product_1$,m_1)

in

(v_1 - v_2,m_2)

end

38. <Product> ::= <Product> "*" <Negation>

E'([[$Product_1$ * $Negation_1$]], m_0) =

let

val (v_1,m_1) = E'($Product_1$,m_0)

val (v_2,m_2) = E'($Negation_1$,m_1)

in

(v_1 * v_2, m_2)

end

39. <Product> ::= <Product> "div" <Negation>

E'([[$Product_1$ div $Negation_1$]], m_0) =

let

val (v_1,m_1) = E'($Product_1$,m_0)

val (v_2,m_2) = E'($Negation_1$,m_1)

in

(v_1 / v_2, m_2)

end

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

E’([[$Product_1$ mod $Negation_1$]], m_0) =

```
    let
        val (v1,m1) = E’(Product1,m0)
        val (v2,m2) = E’(Negation1,m1)
    in
        (v1 % v2, m2)
    end
```

41. <Product> ::= <Negation>

E’([[$Negation_1$]], m_0) =

```
    let
        val (v1,m1) = E’(Negation1, m0)
    in
        (v1,m1)
    end
```

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

E’([[not($Negation_1$)]], m_0) =

```
    let
        val (v1,m1) = E’(Negation1,m0)
    in
        (not v1,m1)
    end
```

43. <Negation> ::= “-” <Negation>

E’([[- $Negation_1$]], m_0) =

```
    let
        val (v1,m1) = E’(Negation1,m0)
    in
        (-v1,m1)
    end
```

44. <Negation> ::= <Exponential>

E’([[$Exponential_1$]], m_0) =
    let
        val (v_1,m_1) = E’($Exponential_1$,m_0)
    in
        (v_1,m_1)
    end

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

E’([[$Base_1$ ^ $Exponential_1$]], m_0) =
    let
        val (v_1, m_1) = E’($Exponential_1$, m_0)
        val (v_2, m_2) = E’($Base_1$, m_1)
    in
        (exp(v_2, v_1), m_2)
    end

46. <Exponential> ::= <Base>

E’([[$Base_1$]], m_0) =
    let
        val (v_1, m_1) = E’($Base_1$, m_0)
    in
        (v_1, m_1)
    end

47. <Base> ::= <Integer_Const>

E’([[$Integer\_Const_1$]], m_0) =
    let
        val v_1 = $Integer\_Const_1$
    in
        (v_1, m_0)
    end

48. <Base> ::= <Boolean_Const>

E’([[$Boolean\_Const_1$]], m_0) =

let
val v_1 = Boolean_Const$_1$
in
(v_1,m_0)
end

49. <Base> ::= "(" <Expression> ")"

E'([[(Expression$_1$)]], m_0) =
let
val (v_1,m_1) = E'(Expression$_1$,m_0)
in
(v_1,m_1)
end

50. <Base> ::= "|" <Expression> "|"

E'([[| Expression$_1$ |]], m_0) =
let
val (v_1,m_1) = E'(Expression$_1$,m_0)
in
if v_1 < 0 then
(-v_1,m_1)
else
(v_1,m_1)
end
end

51. <Base> ::= <PrePost>

E'([[(PrePost$_1$)]], m_0) =
let
val (v_1,m_1) = E'(PrePost$_1$,m_0)
in
(v_1,m_1)
end

52. <Base> ::= <Id>

E' ([[Id]] , m_0) =
let

val loc = getLoc (accessEnv(Id_1, m_0)
val v_1 = accessStore(loc ,m_0)
in
(v_1,m_0)
end

53. <PrePost> ::= "++" <Id>

M ([[++ Id_1]] , m_0) =
let
val (v_1 , m_1) = E' ([[id_1]] , m_o)
val loc = getLoc (accessEnv (v_1 , m_1)
val m_2 = updateStore (loc , v_1+1, m_1)
in
m_2

end

E' ([[++ Id_1]] , m_0) =
let
val (v_1 , m_1) = E' ([[id_1]] , m_o)
val loc = getLoc (accessEnv (v_1 , m_1)
val m_2 = updateStore (loc , v_1+1, m_1)
in
(v_1+1,m_2)

End

54. <PrePost> ::= "--" <Id>
E' ([[-- Id_1]] , m_0) =
let
val (v_1 , m_1) = E' ([[id_1]] , m_o)
val loc = getLoc (accessEnv (v_1 , m_1)
val m_2 = updateStore (loc , v_1-1, m_1)
in
(v_1-1,m_2)
end

55. <PrePost> ::= <Id> “++”

E’ ([[Id_1 ++]] , m_0) =

let

val (v_1 , m_1) = E’ ([[id_1]] , m_o)

val loc = getLoc (accessEnv (v_1 , m_1)

val m_2 = updateStore (loc , v_1+1, m_1)

in

(v_1,m_2)

end

56. <PrePost> ::= <Id> “--”

E’ ([[Id_1 --]] , m_0) =

let

val (v_1 , m_1) = E’ ([[id_1]] , m_o)

val loc = getLoc (accessEnv (v_1 , m_1)

val m_2 = updateStore (loc , v_1-1, m_1)

in

(v_1,m_2)

end

57. <PrePost> ::= “++” <Id>

M ([[++ Id_1]] , m_0) =

let

val (v_1 , m_1) = E’ ([[id_1]] , m_o)

val loc = getLoc (accessEnv (v_1 , m_1)

val m_2 = updateStore (loc , v_1+1, m_1)

in

m_2

end

58. <PrePost> ::= “--” <Id>

M ([[-- Id_1]] , m_0) =

let

val (v_1 , m_1) = E’ ([[id_1]] , m_o)

val loc = getLoc (accessEnv (v_1 , m_1)

val m_2 = updateStore (loc , v_1-1, m_1)

in

m_2

end

59. <PrePost> ::= <Id> “++”

M ([[Id_1 ++]] , m_0) =

let

val (v_1 , m_1) = E’ ([[id_1]] , m_o)

val loc = getLoc (accessEnv (v_1 , m_1)

val m_2 = updateStore (loc , v_1+1, m_1)

in

m_2

end

60. <PrePost> ::= <Id> “--”

M ([[Id_1 --]] , m_0) =

let

val (v_1 , m_1) = E’ ([[id_1]] , m_o)

val loc = getLoc (accessEnv (v_1 , m_1)

val m_2 = updateStore (loc , v_1-1, m_1)

in

m_2

end

21. <Do_While_Block> ::= "do" <Block_Statement> “while” “(“<Expression>”)”

M ([[do $Block\_Statement_1$ while ($Expression_1$)]], m0) = N ($Expression_1$, $Block\_Statement_1$, m_0)

N ($Expression_1$, $Block\_Statement_1$, m_0) =

let

m_1 = M($Block\_Statement_1$, m_0)

val (v_1 , m_2) = E’ ($Expression_1$, m_1)

in

if v_1 then N ($Expression_1$, $Block\_Statement_1$,

M($Block\_Statement_1$, m_2))

else m_2

end