5. <ha\_stmt> -> ha ( <boolexpr> ) <stmt> [ lol <stmt> ]

M\_ha(ha (<boolexpr>) <stmt>, s) 🡪

if M\_b(<boolexpr>, s) == error

return error;

if M\_b(<boolexpr>, s)

if M\_stmt(<stmt>, s) == error

return error;

return s = M\_stmt(<stmt>, s)

M\_ha(ha (<boolexpr>) <stmt1> lol <stmt2>, s) -->

if M\_b(<boolexpr>, s) == error

return error;

if M\_b(<boolexpr>, s)

if M\_stmt(<stmt1>, s) == error

return error;

return s = M\_stmt(<stmt1>, s)

else

if M\_stmt(<stmt2>, s) == error

return error;

return s = M\_stmt(<stmt2>, s)

6. <during\_stmt> -> during ( <boolexpr> ) <stmt>

M\_during(during (<boolexpr>) <stmt>, s) 🡪

If M\_bool(<boolexpr>, s) == error

return error;

if M\_bool(<boolexpr>, s)

if M\_stmt(<stmt>, s) == error

return error;

return s = M\_stmt(<stmt>, s)

9.

1. String + String does concatenation

Syntax: <eq1> -> <assignment> -> `$`<data\_type> <ident>`;`

<data\_type> -> ger | oat | string | car | bool

<eq1> -> `$$` <ident> { (\* | - | = | \*\*\*) <ident> } `;` \*\*\* = exponentiation

<ident> -> id | oat\_lit | ger\_lit | char\_lit | string\_lit | <bool\_value>

Semantics: <eq1.value> 🡨 data\_type.value ident.value1

If (data\_type.value == string) return string

Else if (data\_type.value == ger) return ger

Else if (data\_type.value == oat) return oat

Else if (data\_type.value == car) return car

Else return bool

<eq1> 🡨 ident.value2 + ident.value3

If (ident.value1 == identvalue2 && ident.value3 = string\_lit) return string\_lit

Else return float

Predicate: ident.value1 == identvalue2

Ident.value3 == string\_lit to add two strings together.

1. String \* Natural repeats the Natural
2. Assign bool to natural is allowed
3. Assign natural to bool is allowed
4. Assign char to natural is allowed
5. Assign natural to char is allowed
6. Assign natural to real is allowed
7. No other types are allowed to be assigned to others outside of their own
8. Dividing by zero is an error
9. Modulo operating by zero is an error

10.

Please refer to the number10 file in github.

11. Axiomatic Semantics (find the weakest preconditions)

1. a = 2 \* (b – 1) – 1 {a > 0}

*a = 2 \* (b – 1) – 1*

*{a > 0}*

*2 \* (b – 1) – 1 > 0*

*2 \* (b – 1) > 1*

*(b – 1) > ½*

*b > ½ + 2/2*

*b > 3/2*

1. if (x < y) x = x + 1

x = x + 1 {x < 0}

else x = 3 \* x

x = 3 \* x {x < 0}

{x < 0}

{x + 1 = 0}

x = x + 1

{x < 0}

{3 \* x < 0}

x = 3 \* x

{x < 0}

{x < -1} {x < -1}

x = x + 1 when it satisfies this

{x < 0} condition it satisfies both

{x < 0} conditions

x = 3 \* x

{x < 0}

1. dd
2. a = 3 \* (2 \* b + a); a = 3 \* (2 \* b + a); a = 3 \* (2 \* b + a);

b = 2 \* a – 1 {2 \* a – 1 > 5} {a > 3}

{b > 5} b = 2 \* a – 1; b = 2 \* a – 1;

{b > 5} {b > 5}

{b + a > ½}

A = 3 \* (2 \* b + a);

{a > 3}

B = 2 \* a – 1;

{b > 5}