Report

I have made constructors as exp(X) which is true if X is true.

Test Cases:

1.hastype([],num(1),T).

T = tint.

2.hastype([],t,T).

T = tbool.

3. hastype([],add(num(1), num(2)),T).

T = tint.

4. hastype([("x",tbool)],or(variable("x"), f),T).

T = tbool.

5. hastype([("x",tint),("y",tint)],add(num(1),mul(variable("y"), num(2))),T).

T = tint.

6. hastype([("x",tint)],if_then_else(t,num(1),num(2)),T).

T = tint.

7. hastype([("x",tint)],if_then_else(t,[num(1),t],[num(2),f]),T).

T = 'tint * tbool'.

8. typeElaborate([],assign("x",num(1)),G).

G = [("x", tint)]

9.typeElaborate([],parallel(assign("x",num(1)),assign("y",num(1))),G).

G = [[("y", tint)], [("x", tint)]]

10. typeElaborate([],seq(assign("x",num(1)),assign("y",add(num(1), variable("x")))),G).

G = [("y", tint), ("x", tint)]

11. typeElaborate([],seq(assign("x",num(1)),assign("y",[add(num(1), variable("x")),t])),G).

G = [("y", 'tint * tbool'), ("x", tint)]

Note:

For functions I needed to assume some type for input. I assumed that expression itself will have type of input. As making it abstract is not possible.

For example if I assumed type of input to be some variable and let prolog decides what it is, its infinite loop.

As prolog will check each input and let say it start with tuple type then it will go forever.(if our answer is not tuple)

12.

hastype([],funct("x",tint,funct("y",tint,add(variable("x"),variable("y")))),T).

T = arrowtype(tint, arrowtype(tint, tint)).

13.

hastype([],let_d_in_e(assign("x",num(1)),funct("y",tint,[variable("x"),variable("y")])),T).

T = arrowtype(tint, 'tint * tint')