Programming language theory, 2021 spring, Homework 2

Student:	Khaimovich Timur
email:	t_khaimovich@kbtu.kz
ID:	18BD110452
Group:	Fri 13:00

Exercise I

Exercise II

Theoretical part

I did not remove any rules, but I changed one and added two rules, and changed syntax just a little bit.

1. Added normal form to the syntax
   norm::=
   v
   wrong
2. Change rule E-If to:
   $$\frac{t1 \rightarrow t1'}{if\ t1\ then\ norm\ else\ norm \rightarrow if\ t1'\ then\ norm\ else\ norm}$$
3. Added rule: $\frac{t2 \rightarrow t2'}{if\ t1\ then\ t2\ else\ t3 \rightarrow if\ t1'\ then\ t2'\ else\ t3}$
4. Added rule: $\frac{t3 \rightarrow t3'}{if\ t1\ then\ t2\ else\ t3 \rightarrow if\ t1\ then\ t2\ else\ t3'}$

Practical part

```
/* Ex 3.5.18 */
if (succ(iszero 0)) then (if (true) then succ 0 else 0) else 0;
ubuntu@ubuntu2004:~/Documents/plt/w5/3.5.18$ ./f test.f
(succ (iszero (pred (pred 0))))->iszero (pred (pred 0))->pred (pred 0)->pred 0->(succ (iszero
(pred 0)))->iszero (pred 0)->pred 0->(succ (iszero 0))->iszero 0->(succ true)->wrong->wrong
true->true
false->false
if (succ (iszero 0)) then if true then 1 else 0 else 0->if true
                                                        then 1
                                                        else 0->

if (succ (iszero 0))
then 1
else 0->(succ (iszero 0))->iszero 0->if (succ true) then 1 else 0->(succ true)->
if wrong
then 1
else 0->wrong->wrong
ubuntu@ubuntu2004:~/Documents/plt/w5/3.5.18$
```

Exercise III

Part A

Statement: All stuck terms from the original syntax are now converted to wrong term and original logic works correctly

Proof:

Since we did not remove any rules from the original logic thus it will work in the same way.

To prove the first part of the statement let us consider every case such that g ->* wrong. g cannot be value.

1. g is succ
   if g(t) is succ then t must be true, false or wrong. This is what defined in the rule E-Succ-Wrong
2. g is pred
   Here situation is the same as with succ
3. g is iszero
   Again similar situation
4. g is an if statement
   if statement is wrong when its guard is wrong, or when its branch that will be evaluated is wrong. The rule E-If-Wrong will evaluate g to wrong when the guard is not a Boolean value. If the guard is ok then by the rule E-If all (if they are nested) will be evaluated and

eventually not-if-terms will be left that can be marked wrong by above rules.

Part B

On this screenshot I've implemented new evaluation rules

```
type term =
    TmWrong of info
  | TmTrue of info
  | TmFalse of info
  | TmIf of info * term * term * term
  | TmZero of info
  | TmSucc of info * term
  | TmPred of info * term
  | TmIsZero of info * term
```

Here, I've added new term

```
and printtm_ATerm outer t = match t with
    TmWrong(_) -> pr "wrong"
  | TmTrue(_) -> pr "true"
  | TmFalse(_) -> pr "false"
  | TmZero(fi) ->
      pr "0"
  | TmSucc(_,t1) ->
```

Now, ocaml can print new term

```
/* Ex 3.5.16 */
if succ(1) then 1 else 2;/*E-If-Wrong*/
succ(iszero(1));/*E-Succ-Wrong*/
pred(iszero(pred(1)));/*E-Pred-Wrong*/
iszero(iszero(succ(0)));/*E-IsZero-Wrong*/
ubuntu@ubuntu2004:~/Documents/plt/w5/3.5.18$ ./f test.f
wrong
true
false
wrong
wrong
wrong
wrong
wrong
ubuntu@ubuntu2004:~/Documents/plt/w5/3.5.18$ s
```