

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

Exercise I

```
ubuntu@ubuntu2004:~/Documents/plt/w5/3.5.18$ cat test.f
/* Ex1 */
/* Stuck term */
succ(iszero(pred(pred(0))));
/* Normal Form Term */
true;
/* Value */
false;
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
ubuntu@ubuntu2004:~/Documents/plt/w5/3.5.18$
```

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:

$$\frac{t_1 \rightarrow t_1'}{if\ t_1\ then\ norm\ else\ norm \rightarrow if\ t_1'\ then\ norm\ else\ norm}$$

3. Added rule:
$$\frac{t_2 \rightarrow t_2'}{if\ t_1\ then\ t_2\ else\ t_3 \rightarrow if\ t_1'\ then\ t_2'\ else\ t_3}$$

4. Added rule:
$$\frac{t_3 \rightarrow t_3'}{if\ t_1\ then\ t_2\ else\ t_3 \rightarrow if\ t_1\ then\ t_2\ else\ t_3'}$$

Practical part

```
/* Ex 3.5.18 */
if (succ(iszero 0)) then (if (true) then succ 0 else 0) else 0;
ubuntu@ubuntu2004:~/Documents/pl1/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/pl1/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 $g \rightarrow^* \text{wrong}$. 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

```
core.ml x
let rec isbadnat t = match t with
| .. TmWrong( ) -> true
| .. TmTrue( ) -> true
| .. TmFalse( ) -> true
| _ -> false

let rec isbadbool t = match t with
| .. TmWrong( ) -> true
| .. t when isnumericval t -> true
| _ -> false

let rec isnorm t = match t with
| TmWrong( ) -> true
| t when isval t -> true
| _ -> false

let rec eval1 t = (*printtm t;*)
(*print_string "->";*)
match t with
| TmIf( , TmTrue( ), t2, t3) ->
  t2
| TmIf( , TmFalse( ), t2, t3) ->
  t3
| TmIf( , badbool, t2, t3) when (isbadbool badbool) ->
  .. TmWrong(dummyinfo)
| TmIf(fi, t1, t2, t3) when (isnorm t2) && (isnorm t3) ->
  let t1' = eval1 t1 in
  TmIf(fi, t1', t2, t3)
| TmIf(fi, t1, t2, t3) when not(isnorm t2) ->
  let t2' = eval1 t2 in
  TmIf(fi, t1, t2', t3)
| TmIf(fi, t1, t2, t3) when not(isnorm t3) ->
  let t3' = eval1 t3 in
  TmIf(fi, t1, t2, t3')
| TmSucc( , badnat) when (isbadnat badnat) ->
  .. TmWrong(dummyinfo)
| TmSucc(fi, t1) ->
  let t1' = eval1 t1 in
  TmSucc(fi, t1')
| TmPred( , TmZero( )) ->
  TmZero(dummyinfo)
| TmPred( , badnat) when (isbadnat badnat) ->
  .. TmWrong(dummyinfo)
| TmPred( , TmSucc( , nv1)) when (isnumericval nv1) ->
  nv1
| TmPred(fi, t1) ->
  let t1' = eval1 t1 in
  TmPred(fi, t1')
| TmIsZero( , TmZero( )) ->
  TmTrue(dummyinfo)
| TmIsZero( , badnat) when (isbadnat badnat) ->
  .. TmWrong(dummyinfo)
| TmIsZero( , TmSucc( , nv1)) when (isnumericval nv1) ->
  TmFalse(dummyinfo)
```

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
wrong
ubuntu@ubuntu2004:~/Documents/plt/w5/3.5.18$ s

```