Programming language theory, 2021 spring.

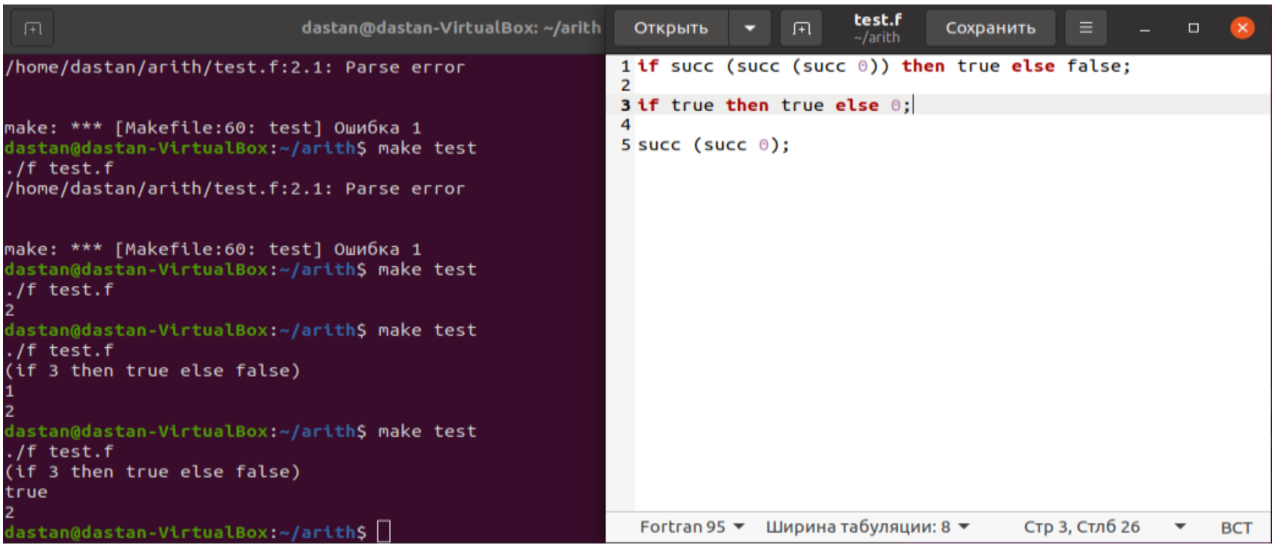
Howe work 2

Attention: read the description in MSTeams as well

Student:	Galymbekov Dastan
email:	galymbekov.dastan@gmail.com/d_galymbekov@kbtu.kz
ID:	18BD110379
Group:	Friday, from 9 to 12

Exercise I.

- Normal form of term is a term which describes not calculatable value, which in on end point, no further evaluations can be done.
- Value is partly every term in lambda calculus. Any value might be in normal form but not every normal form is value. Non-value normal form is stuck.
- Stuck is a normal form of a term, but not a value. Sometimes it called as runtime error.



Exercise II.

Theoretical part

In this part I suppose that in the figure 3.1 Must be removed first two rules which specify “if true then t2 and if false then t3”. Because in this rule first of all evaluated “if”. Instead of it we must use “then”, “else” and then “if”. Therefor, we start our calculations with “then”. Suppose we have 3 terms(t1, t2 and t3). First of all, we calculate “t2” because after then, next term is “t2”.

then:

t2=>t2’
If t1 then t2 else t3 => if t1 then t2’ else t3

There we firstly evaluate t2. Then we must evaluate t3. Because t2 firstly was a term. After evaluating, we take a value. Therefor, next term will be t3.

else:

t3=>t3’
If t1 then v2 else t3 => if t1 then v2 else t3’

There, after completing of evaluation of else and then, we go to t1 which might be true or false. Is this step we also see that we execute a value v3 from term t3. Therefor we will have next lines.

if:

t1=>t1’
If t1 then v2 else v3 => if t1’ then v2 else v3

This steps give us two variants for “**if – true** and **if – false**”.

1. If true then v2 else v3 => v2
2. If false then v2 else v3 => v3

So, in conclusion, we replace “**if – true** and **if – false**” from figure 3.1 with “(**if – true)’** and (**if – false)’**” which will be done after calculations.
Therefor, we also see changes version of our “if” statement.

Practical part

Code for eval1 function.

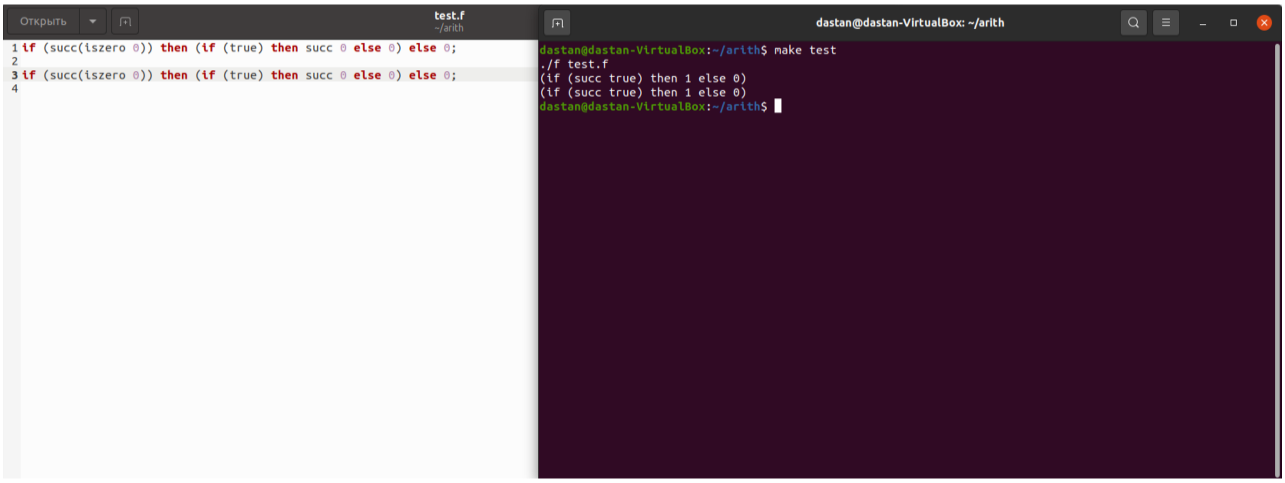
```
exception NoRuleApplies

let rec isnumericval t = match t with
TmZero(_) -> true
| TmSucc(_,t1) -> isnumericval t1
| _ -> false

let rec isval t = match t with
TmTrue(_) -> true
| TmFalse(_) -> true
| t when isnumericval t -> true
| _ -> false

let rec eval1 t = match t with
TmIf(_,TmTrue(_),t2,t3) ->
t2
| TmIf(_,TmFalse(_),t2,t3) ->
t3
| TmIf(fi,t1,t2,t3) when(not(isval t2))->
let t2' = eval1 t2 in
TmIf(fi, t1, t2', t3)
| TmIf(fi,t1,t2,t3) when(not(isval t3))->
let t3' = eval1 t3 in
TmIf(fi, t1, t2, t3')
| TmIf(fi,t1,t2,t3) when(not(isval t1))->
let t1' = eval1 t1 in
TmIf(fi, t1', t2, t3)
| TmSucc(fi,t1) ->
let t1' = eval1 t1 in
TmSucc(fi, t1')
| TmPred(_,TmZero(_)) ->
TmZero(dummyinfo)
| TmPred(_,TmSucc(_,nv1)) when (isnumericval nv1) ->
nv1
| TmPred(fi,t1) ->
let t1' = eval1 t1 in
TmPred(fi, t1')
| TmIsZero(_,TmZero(_)) ->
TmTrue(dummyinfo)
| TmIsZero(_,TmSucc(_,nv1)) when (isnumericval nv1) ->
TmFalse(dummyinfo)
| TmIsZero(fi,t1) ->
let t1' = eval1 t1 in
TmIsZero(fi, t1')
| _ ->
raise NoRuleApplies

let rec eval t =
try let t' = eval1 t
in eval t'
with NoRuleApplies -> t
```


Открыть
test.f
~/arith
1 if (succ(iszero 0)) then (if (true) then succ 0 else 0) else 0;
2
3 if (succ(iszero 0)) then (if (true) then succ 0 else 0) else 0;
4
dastan@dastan-VirtualBox: ~/arith
dastan@dastan-VirtualBox:~/arith$ make test
./f test.f
(if (succ true) then 1 else 0)
(if (succ true) then 1 else 0)
dastan@dastan-VirtualBox:~/arith$

Exercise III.

Part A.

Statement: Let us suppose that if g is stuck, then g will be a wrong

Proof:

Proof: assume that g may be only true, false or 0. Therefor g cannot be a stuck.

Let us check variant if g1 then g2 else g3.

We know that g1 is a normal form(value or stuck) if g1 is not in normal form we can use else if rule to reduce it.

g1 cannot be a true or false term. Because if it will be, by using rules IfTrue and IfFalse, it can be applied, and it will not be stuck.

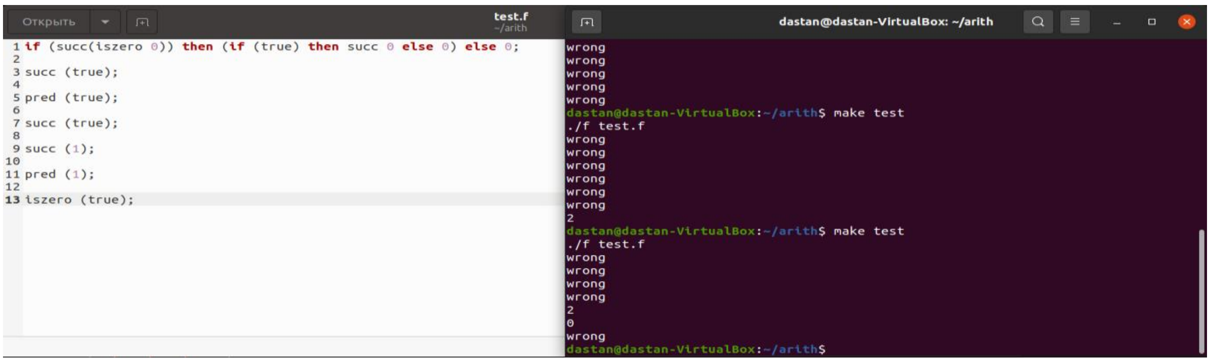
Subcase1: g1is not a value and suppose that g1 is in normal form. Therefor it is stuck. In that casem if g1 will be stuck, theng 1 will give us wrong. Consequently, "if wrong then g2 else g3" will give us wrong.

Subcase2: g1 = succ (g11)

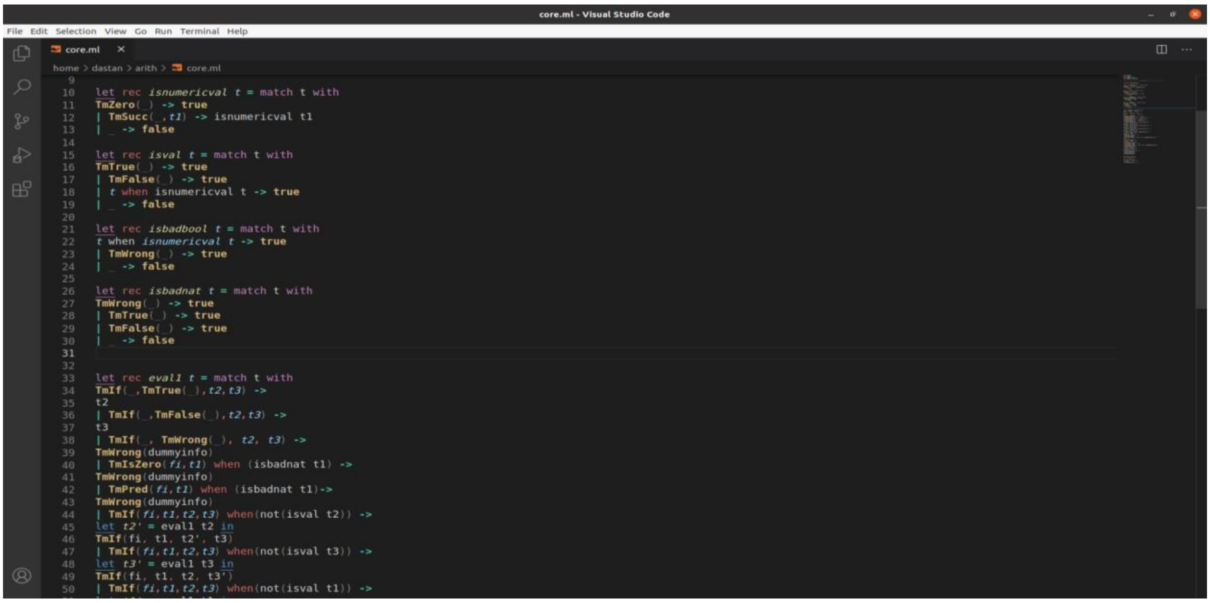
Let us suppose that g11 is a nv. Then, succ (g11) will return us also nv. But then if we apply it like "if(nv) then g2 else g3", we know that nv is a badbool. So, we can modify it like if(badbool) then g2 else g3 which will give us "wrong".

But we also know that if succ equal to treu or false, it will be presented as "if (succ (boll)) then g2 else g3". But if it will be boll, we know that it is a badnat. Therefore, it will give us "if (succ (badnat)) then g2 else g3" which will give us "if (wrong) then g2 else g3" which conaequently will give us wrong.

* Part B.

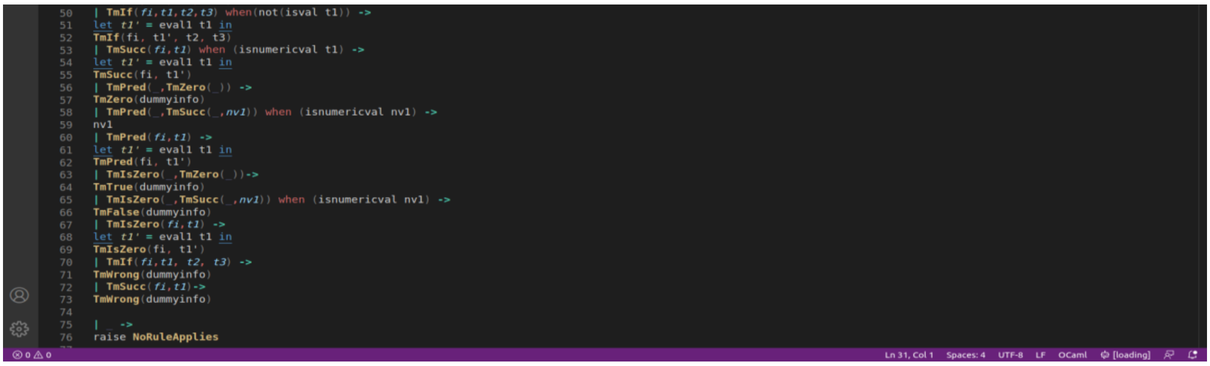


Screenshot of “f” file and compilation result.

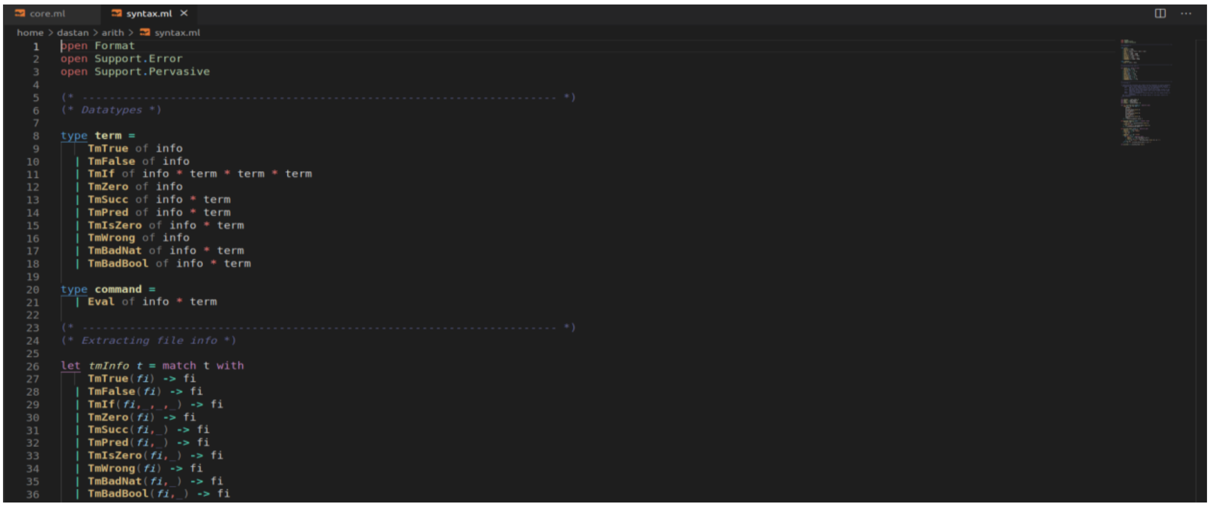


In this screenshot we can see how we create two functions isbadbool and isbadnat. That is clear that if for example isbadnat will apply values such as TmWrong, TmTrue and TmFalse, it will do true. Otherwise, it will apply false.

Also, we see that se replace Tissera and TmPred to the top of our if values in the way that they must be calculated first. Moreover, we can see our TmWrong statement which will print out in the terminal “wrong”.



There you can mention just the second part of code.



In this screenshot there are shown newly added TmWrong which was added both, into syntax.ml and syntax.mli.

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

This screenshot shows that TmWrong statement will print “wrong” message in the terminal.

```
core.ml    syntax.ml    syntax.mli
home > dastan > arith > syntax.mli
(* module Syntax: syntax trees and associated support functions *)

open Support.Pervasive
open Support.Error

(* Data type definitions *)
type term =
    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
  | TmWrong of info
  | TmBadNat of info * term
  | TmBadBool of info * term

type command =
  | Eval of info * term
```

This screenshot shows syntax.mli code which was mentioned before.