Untyped Arithmetic Expressions with extensions

<u>**SYNTAX :**</u>

<u>Syntax For **AND :**</u>

AND t_1 t_2 t_1 and t_2 should be bool type or $t_1 \rightarrow t_1'$ and $t_2 \rightarrow t_2'$
(t_1' and t_2' must be bool type)

<u>Syntax For **OR :**</u>

OR t_1 t_2 t_1 and t_2 should be bool type or $t_1 \rightarrow t_1'$ and $t_2 \rightarrow t_2'$
(t_1' and t_2' must be bool type)

<u>Syntax For **SWITCH :**</u>

SWITCH t_1 CASE **0** : t_2 CASE **SUCC 0 :** t_3 where t_1 should be NAT type and t_2 ,t_3 can be R type(R : bool or NAT)

<u>**Formal Operational Semantics**</u> :

<u>**AND :**</u>

AND true t $\rightarrow$ t
AND false t $\rightarrow$ false
AND t true $\rightarrow$ t
AND t false $\rightarrow$ false
AND t_1 t_2 $\rightarrow$ need further evaluation

<u>**OR :**</u>

OR true t $\rightarrow$ true
OR false t $\rightarrow$ t
OR t_1 t_2 $\rightarrow$ need further evaluation

<u>**SWITCH**</u> :

SWITCH 0 CASE 0 : t_1 CASE SUCC 0 : t_2 $\rightarrow$ t_1
SWITCH SUCC 0 CASE 0 : t_1 CASE SUCC 0 : t_2 $\rightarrow$ t_2

Implementation:

We have updated/added in these files:

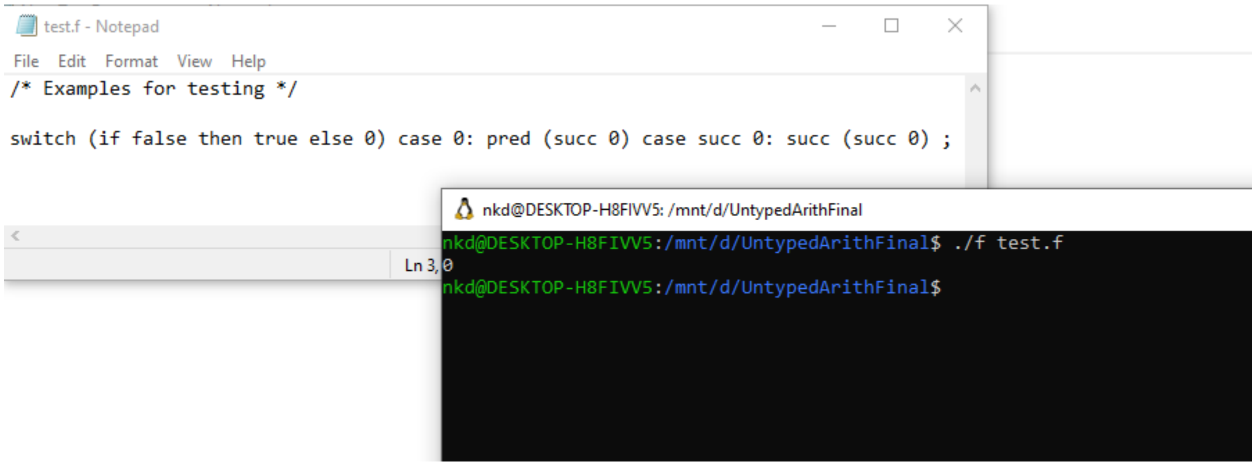
Core.ml : evaluation rules for SWITCH, AND and OR has been added.

Syntax.ml, **Syntax.mli** : dataType, file information and printing code.

Lexer.mll : Keyword declaration.

```
 (* EVALUATION RULE  FOR SWITCH -------*)
 | TmSwitch(_,t1,t2,t3) when isValZero t1 ->
     t2
 | TmSwitch(_,t1,t2,t3) when isValSuccZero t1 ->
     t3
 | TmSwitch(fi,t1,t2,t3) ->
     let t1' = eval1 t1 in
     TmSwitch(fi,t1',t2,t3)
(* EVALUATION RULE FOR AND -------- *)
| TmAnd(fi,TmTrue(_),v2) when isValBool v2 ->
     v2
 | TmAnd(fi,TmFalse(_),v2) when isValBool v2 ->
     TmFalse(dummyinfo)
 | TmAnd(fi,v1,TmTrue(_)) when isValBool v1 ->
     v1
 | TmAnd(fi,v1,TmFalse(_)) when isValBool v1 ->
     TmFalse(dummyinfo)
 | TmAnd(fi,v1,t2) when isValBool v1 ->
     let t2' = eval1 t2 in
     TmAnd(fi,v1,t2')
 | TmAnd(fi,t1,t2) ->
     let t1' = eval1 t1 in
     TmAnd(fi,t1',t2)
(* EVALUATION RUKE FOR OR -------- *)
 | TmOr(fi,TmTrue(_),v2) when isValBool v2 ->
     TmTrue(dummyinfo)
 | TmOr(fi,TmFalse(_),v2) when isValBool v2 ->
     v2
 | TmOr(fi,v1,t2) when isValBool v1 ->
     let t2' = eval1 t2 in
     TmOr(fi,v1,t2')
 | TmOr(fi,t1,t2) ->
     let t1' = eval1 t1 in
     TmOr(fi,t1',t2)
(* BOOLEAN VALUE CHECK  -------- *)
let isValBool t = match t with
   TmTrue(_)             -> true
 | TmFalse(_)            -> true
 | _                     -> false
(* SUCC 0 VALUE CHECK  -------- *)
let isValSuccZero t = match t with
   TmSucc(_,t1) when isValZero t1 -> true
 | _
```

Output



test.f - Notepad

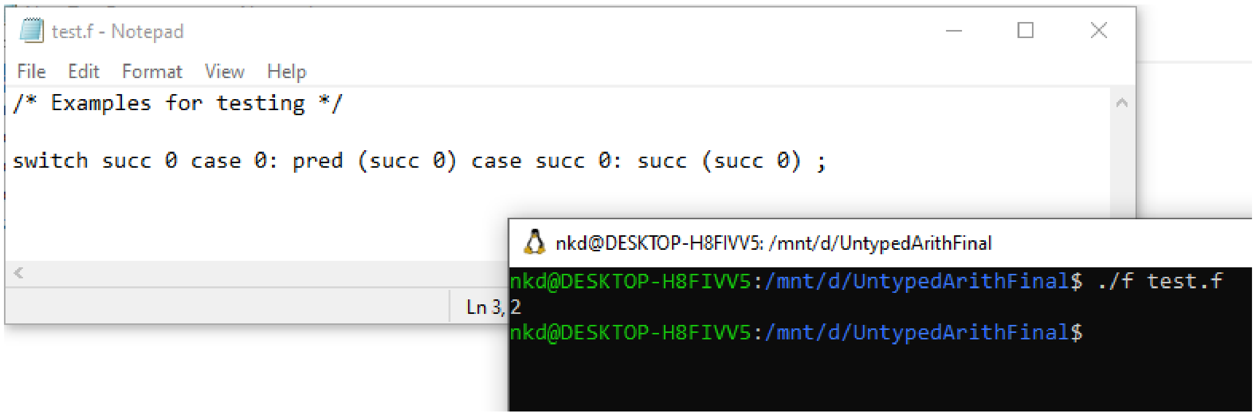
```
/* Examples for testing */

switch (if false then true else 0) case 0: pred (succ 0) case succ 0: succ (succ 0) ;
```

Ln 3,

nkd@DESKTOP-H8FIVV5: /mnt/d/UntypedArithFinal

```
nkd@DESKTOP-H8FIVV5:/mnt/d/UntypedArithFinal$ ./f test.f
0
nkd@DESKTOP-H8FIVV5:/mnt/d/UntypedArithFinal$
```



test.f - Notepad

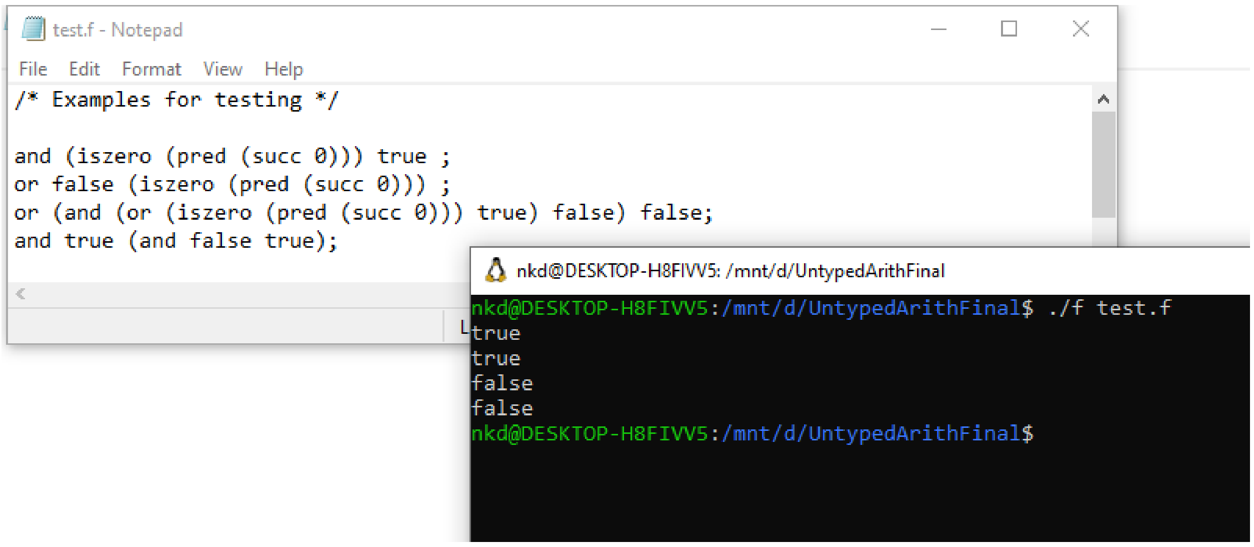
```
/* Examples for testing */

switch succ 0 case 0: pred (succ 0) case succ 0: succ (succ 0) ;
```

Ln 3,

nkd@DESKTOP-H8FIVV5: /mnt/d/UntypedArithFinal

```
nkd@DESKTOP-H8FIVV5:/mnt/d/UntypedArithFinal$ ./f test.f
2
nkd@DESKTOP-H8FIVV5:/mnt/d/UntypedArithFinal$
```



test.f - Notepad

```
/* Examples for testing */

and (iszero (pred (succ 0))) true ;
or false (iszero (pred (succ 0))) ;
or (and (or (iszero (pred (succ 0))) true) false) false;
and true (and false true);
```

nkd@DESKTOP-H8FIVV5: /mnt/d/UntypedArithFinal

```
nkd@DESKTOP-H8FIVV5:/mnt/d/UntypedArithFinal$ ./f test.f
true
true
false
false
nkd@DESKTOP-H8FIVV5:/mnt/d/UntypedArithFinal$
```