프로그래밍 언어 개론
과제 보고서

[PL00]HW11_201601980_KimSeongHan

개발환경 : Ubuntu

제출일 (2021-05-20)

201601980 / 김성한(00분반)

<과제 설명>

Homework – 2. fp-style

- Boolean과 조건문에 syntactic sugar를 적용한 fp-style CFVAE을 해석하는 interpreter를 작성 하시오.
  - (Ast 와 Store에 대한 정의는 skeleton code 참고, 자세한 언어 정의는 이론 자료 참고)

[Rule LT] : $e_1 < e_2$
추상메모리 σ에서 e_1이 n_1으로 계산되고,
추상메모리 σ에서 e_2가 n_2로 계산된다고 가정하면,
추상메모리 σ에서 $e_1 < e_2$는 $n_1 <_Z n_2$로 계산
($n_1 <_Z n_2$ = <λx. λy. x, ⦰> if n_1 is less than n_2
<λx. λy. y, ⦰> otherwise)

[LT] : $e_1 < e_2$

$$\frac{\sigma \vdash e_1 \Downarrow_E n_1 \quad \sigma \vdash e_2 \Downarrow_E n_2}{\sigma \vdash e_1 < e_2 \Downarrow_E n_1 <_Z n_2}$$

```
# example
(1 > 3) (0 + 7) 3 // ⇒(λx. λy. y) 7 3 ⇒3 ((1 > 3)? 0 + 7 : 3)
((1 - 1) < (0 + 1)) (0 + 7) 3 // ⇒(λx. λy. x) 7 3 ⇒7 (((1 - 1) < (0 + 1))? 0 + 7 : 3)
(1 + 1) (0 + 7) 3 // runtime error ( Not a function: 1 + 1 )
```

[과제 제시 내용]

● 이번 과제는 지난 실습에서 구현한 vanilla-style의 CFVAE interpreter를 fp-style로 바꿔 구현하는 과제였다.

● 기존 vanila-style의 CFVAE와는 다르게 Ast.expr의 타입에서 Bool 과 Cond 가 syntatic sugar에 의하여 bool타입의 true 와 false의 값은 ClosureV타입으로 표현되고, 이에따라 Cond 연산은 기존 App에 의해 표현이 된다.

true ⇒ λx. λy. x
false ⇒ λx. λy. y

기존 true와 false는 각각 x 와 y 두 개의 인자를 받아 true인 경우엔 첫 번째 인자인 x false인 경우엔 두 번째 인자인 y를 반환하게 된다.

● 이와 같이 true와 false를 정의하게 되면 Cond에 대한 연산은 App으로 표현할 수 있게되어 boolean 타입과 조건문에 대해 syntatic sugar를 적용할 수 있게된다.

<코드 구현 및 실행 결과 화면>

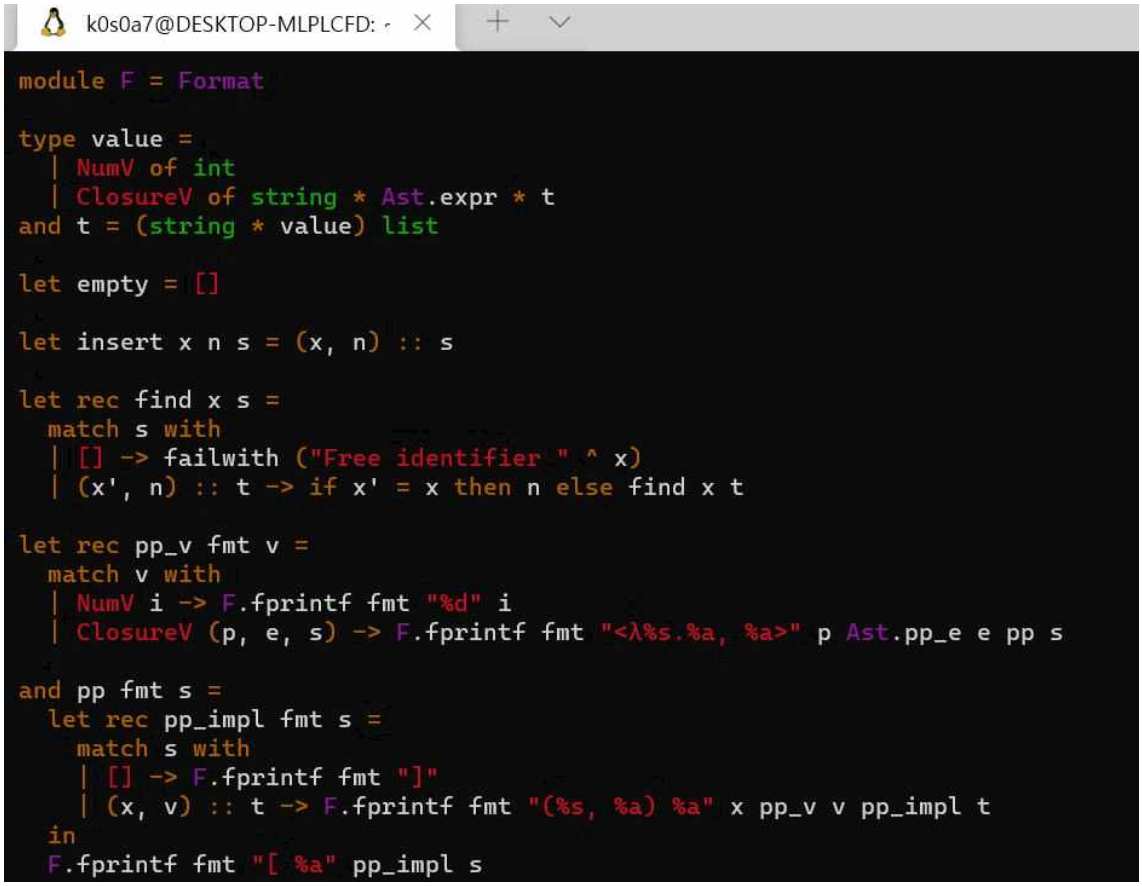
k0s0a7@DESKTOP-MLPLCFD:

```
module F = Format

type expr =
  | Num of int
  | Add of expr * expr
  | Sub of expr * expr
  | Id of string
  | LetIn of string * expr * expr
  | App of expr * expr
  | Fun of string * expr
  | Lt of expr * expr

type fvae =
  | Prog of expr

let rec pp_e fmt e =
  match e with
  | Num n -> F.fprintf fmt "(Num %d)" n
  | Add (e1, e2) -> F.fprintf fmt "(Add %a %a)" pp_e e1 pp_e e2
  | Sub (e1, e2) -> F.fprintf fmt "(Sub %a %a)" pp_e e1 pp_e e2
  | Id x -> F.fprintf fmt "(Id %s)" x
  | LetIn (x, e1, e2) -> F.fprintf fmt "(LetIn %s %a %a)" x pp_e e1 pp_e e2
  | App (e1, e2) -> F.fprintf fmt "(App %a %a)" pp_e e1 pp_e e2
  | Fun (p, e) -> F.fprintf fmt "(Fun %s -> %a)" p pp_e e
  | Lt (e1, e2) -> F.fprintf fmt "(Lt %a < %a)" pp_e e1 pp_e e2

let pp fmt (Prog e) =
  F.fprintf fmt "(Prog %a)" pp_e e
```

[ast.ml의 내부 코드]

● 이번 Functional Programming style의 ast.ml 코드에는 Bool 타입과 Cond 타입이 없어졌음을 확인할 수 있었다.

● 그 외의 타입은 기존 타입과 같으며 Lt패턴에서 true와 false에 대한 ClosureV를 반환하고 그에 의해 App에서 연산을 수행하는 동작을 생각 해볼수 있었다.



[store.ml의 내부 코드]

● store.ml의 코드는 지난주 과제였던 FVAE의 store와 같은 형태이며 추상메모리에는 NumV 타입의 값과 ClosureV 타입의 함수가 저장이 된다. 이에 따라 함수가 값으로 인식될 수 있다.

● 그렇기에 store에서 값으로써 다룰 타입을 정의하기 위해 value를 NumV는 값에대한 타입, ClosureV는 함수에 대한 타입으로 정의하여 선언하였고 이때 ClosureV의 타입정의에 사용되는 t를 참조하기 위하여 and를 이용하였다. 이렇게 store.ml 하나의 파일 내부에 두 개의 타입을 and를 통해 정의해줌으로써 순환참조를 예방하고 이어 두 타입의 상호참조를 허용한다.

● 나머지 find 및 insert 그리고 empty는 이전 store.ml과 바뀐게 없으며 정의된 타입에 맞추어 해당하는 인자들에 대하여 find는 반환하고 insert는 저장하는 역할을 한다.

```
let rec interp_e (s : Store.t) (e : Ast.expr) : Store.value =
        match e with
        | Num n -> NumV n
        | Add (e1,e2) ->
                        begin
                                let exp1 = interp_e s e1 in
                                let exp2 = interp_e s e2 in
                                match exp1,exp2 with
                                | NumV n1, NumV n2 -> NumV(n1 + n2)
                                | _ , _ -> failwith (Format.asprintf "Invalid addition: %a + %a" Ast.pp_e e1 Ast.pp_e e2)
                        end
        | Sub (e1,e2) ->
                        begin
                                let exp1 = interp_e s e1 in
                                let exp2 = interp_e s e2 in
                                match exp1,exp2 with
                                | NumV n1, NumV n2 -> NumV(n1 - n2)
                                | _ , _ -> failwith (Format.asprintf "Invalid subtraction: %a - %a" Ast.pp_e e1 Ast.pp_e e2)
                        end
        | Id str -> Store.find str s
        | LetIn (str,e1,e2) -> interp_e (Store.insert str (interp_e s e1) s) e2
        | App (e1,e2) ->
                        begin
                                let v1 = interp_e s e2 in
                                match interp_e s e1 with
                                | ClosureV (x,e3,s') -> interp_e (Store.insert x v1 s') e3
                                | _ -> failwith (Format.asprintf "Not a function: %a" Ast.pp_e e1)

                        end

        | Fun (str,e) -> ClosureV(str,e,s)
        | Lt (e1,e2) ->
                        begin
                                let v1 = interp_e s e1 in
                                let v2 = interp_e s e2 in
                                match v1,v2 with
                                | NumV n1, NumV n2 -> if n1 < n2 then ClosureV("x",Fun("y",Id("x")),[]) else ClosureV("x",Fun("y",Id("y")),[])
                                | _, _ -> failwith (Format.asprintf "Invalid less-than: %a < %a" Ast.pp_e e1 Ast.pp_e e2)
                        end
```

[interpreter.ml 의 interp_e 함수]

● interpreter.ml 의 interp_e 함수는 지난 FVAE와 크게 달라진 점이 없다. 또한 이번 실습에서 진행한 vanila-style의 코드와 다르게 bool 타입 및 Cond 조건문에 대한 부분이 syntatic sugar에 의해 우리가 구현해야할 부분은 Lt패턴으로 생각할 수 있었다.

● 이번에 추가 된 Lt(e1,e2) 패턴은 초기 e1과 e2를 연산하여 NumV타입의 값 두 개 v1,v2를 반환받는다. 이에따라 각 v1,v2의 값을 비교하여 v1이 v2보다 작다면 true 만일 그렇지 않다면 false에 대한 ClosureV를 반환하게된다. 이에 따라 코드를 작성하게 되었으며 각 true와 false에 대한 ClosureV는 <λx. λy. x, ∅> 와 <λx. λy. y, ∅>로 표현될 수 있었으므로 true = ClosureV(“x” , Fun(“y”,Id(“x”)), []) 로 false = ClosureV(“x” , Fun(“y”,Id(“y”)), []) 로 표현될 수 있었다. 추가적으로 Lt에 대한 Bigstep-Operation은

[Rule LT] : $e_1 < e_2$
추상메모리 σ에서 e_1이 n_1으로 계산되고,
추상메모리 σ에서 e_2가 n_2로 계산된다고 가정하면,
추상메모리 σ에서 $e_1 < e_2$는 $n_1 <_Z n_2$로 계산
($n_1 <_Z n_2$ = <λx. λy. x, ∅> if n_1 is less than n_2
<λx. λy. y, ∅> otherwise)

[LT] : $e_1 < e_2$

$$\frac{\sigma \vdash e_1 \Downarrow_E n_1 \quad \sigma \vdash e_2 \Downarrow_E n_2}{\sigma \vdash e_1 < e_2 \Downarrow_E n_1 <_Z n_2}$$

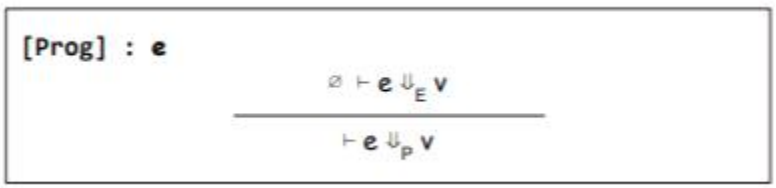
으로 표현할 수 있다.

```
let interp (p : Ast.fvae) : Store.value =
      match p with
     |Prog expr -> interp_e [] expr
```

[interpreter.ml 의 interp 함수]

● interp함수로 초기 인자로 들어온 Prog타입을 interp_e [] expr을 통해 interp_e함수를 호출하고 값을 반환받는 시작함수의 기능을 한다. 이때 반환받는 값의 타입은 Store.value 타입으로 함수로 반환받을수도있고 아니라면 NumV(정수값) 타입으로 반환받을 수도 있다.

● Prog의 Bigstep-Operation은



으로 표현할 수 있으며 이는 빈 추상메모리에서 e를 계산한 결과가 v라고 가정하면 e 는 v로 계산하는 역할을 하게된다. 따라서 위의 interp함수에서 패턴매칭시 같은기능을 수행하도록 구현하였다.

```
k0s0a7@DESKTOP-MLPLCFD:~/week11/hw/fp-style$ dune exec ./main.exe
AST: (Prog (Fun x -> (Fun y -> (Id x))))
RES: <λx.(Fun y -> (Id x)), [ ]>
AST: (Prog (Fun x -> (Fun y -> (Id y))))
RES: <λx.(Fun y -> (Id y)), [ ]>
AST: (Prog (Lt (Num 1) < (Num 3)))
RES: <λx.(Fun y -> (Id x)), [ ]>
AST: (Prog (Lt (Num 3) < (Num 2)))
RES: <λx.(Fun y -> (Id y)), [ ]>
AST: (Prog (Lt (Num 1) < (Add (Num 0) (Num 1))))
RES: <λx.(Fun y -> (Id y)), [ ]>
AST: (Prog (Lt (Sub (Num 1) (Num 1)) < (Add (Num 0) (Num 1))))
RES: <λx.(Fun y -> (Id x)), [ ]>
AST: (Prog (App (App (Fun x -> (Fun y -> (Id x))) (Num 1)) (Num 3)))
RES: 1
AST: (Prog (App (App (Fun x -> (Fun y -> (Id y))) (Num 1)) (Num 3)))
RES: 3
AST: (Prog (App (App (Lt (Sub (Num 1) (Num 1)) < (Add (Num 0) (Num 1))) (Add (Num 0) (Num 7))) (Num 3)))
RES: 7
AST: (Prog (App (App (Add (Num 1) (Num 0)) (Add (Num 0) (Num 7))) (Num 3)))
Runtime Error : Not a function: (Add (Num 1) (Num 0))
AST: (Prog (App (App (Fun x -> (Fun y -> (Id x))) (Num 1)) (Add (Num 3) (Num 10))))
RES: 1
AST: (Prog (App (App (Fun x -> (Fun y -> (Id y))) (Num 1)) (Add (Num 3) (Num 10))))
RES: 13
AST: (Prog (LetIn x (Num 3) (App (App (Lt (Id x) < (Num 4)) (Add (Id x) (Num 10))) (Sub (Id x) (Num 10)))))
RES: 13
AST: (Prog (LetIn x (Num 3) (App (App (LetIn x (Num 2) (Lt (Id x) < (Num 3))) (Add (Id x) (Num 10))) (Sub (Id x) (Num 10)))))
RES: 13
AST: (Prog (LetIn x (Num 3) (App (App (LetIn x (Num 2) (Lt (Id x) < (Num 3))) (LetIn y (Num 10) (Add (Id x) (Id y)))) (LetIn y (Num 99) (Sub (Id x) (Id y
))))))
RES: 13
AST: (Prog (LetIn x (Num 3) (App (App (LetIn y (Num 2) (Lt (Id y) < (Id x))) (Add (Id x) (Num 10))) (Sub (Id x) (Num 10)))))
RES: 13
AST: (Prog (LetIn x (Fun x -> (Add (Id x) (Num 10))) (App (Id x) (App (App (LetIn x (Num 2) (Lt (Id x) < (Num 3))) (App (Id x) (Num 1))) (App (Id x) (Num
 99))))))
RES: 21
AST: (Prog (LetIn x (App (App (LetIn x (Num 2) (Lt (Id x) < (Num 3))) (Fun x -> (Add (Id x) (Num 10)))) (Fun x -> (Sub (Id x) (Num 10)))) (App (Id x) (Nu
m 9))))
RES: 19
k0s0a7@DESKTOP-MLPLCFD:~/week11/hw/fp-style$
```

[main.exe 실행결과]

```
let _ = F.printf "AST: %a\n" pp p1 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p1) in (* <λx.(Fun y -> (Id x)), [ ]> *)
let _ = F.printf "AST: %a\n" pp p2 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p2) in (* <λx.(Fun y -> (Id y)), [ ]> *)
let _ = F.printf "AST: %a\n" pp p3 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p3) in (* <λx.(Fun y -> (Id x)), [ ]> *)
let _ = F.printf "AST: %a\n" pp p4 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p4) in (* <λx.(Fun y -> (Id y)), [ ]> *)
let _ = F.printf "AST: %a\n" pp p5 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p5) in (* <λx.(Fun y -> (Id y)), [ ]> *)
let _ = F.printf "AST: %a\n" pp p6 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p6) in (* <λx.(Fun y -> (Id x)), [ ]> *)
let _ = F.printf "AST: %a\n" pp p7 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p7) in (* 1 *)
let _ = F.printf "AST: %a\n" pp p8 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p8) in (* 3 *)
let _ = F.printf "AST: %a\n" pp p9 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p9) in (* 7 *)
let _ = F.printf "AST: %a\n" pp p10 in
let _ = try F.printf "RES: %a\n" Store.pp_v (interp p10) with Failure e -> F.printf "Runtime Error : %s\n" e in (* Not a function: (Add (Num 1) (Num 0)) *)
let _ = F.printf "AST: %a\n" pp p11 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p11) in (* 1 *)
let _ = F.printf "AST: %a\n" pp p12 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p12) in (* 13 *)
let _ = F.printf "AST: %a\n" pp p13 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p13) in (* 13 *)
let _ = F.printf "AST: %a\n" pp p14 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p14) in (* 13 *)
let _ = F.printf "AST: %a\n" pp p15 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p15) in (* 13 *)
let _ = F.printf "AST: %a\n" pp p16 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p16) in (* 13 *)
let _ = F.printf "AST: %a\n" pp p17 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p17) in (* 21 *)
let _ = F.printf "AST: %a\n" pp p18 in
let _ = F.printf "RES: %a\n" Store.pp_v (interp p18) in (* 19 *)
```

[main.ml에 작성된 테스트케이스 결과]

● main.exe 실행 결과 main.ml에 작성된 테스트케이스 결과와 일치함을 확인할 수 있었다.

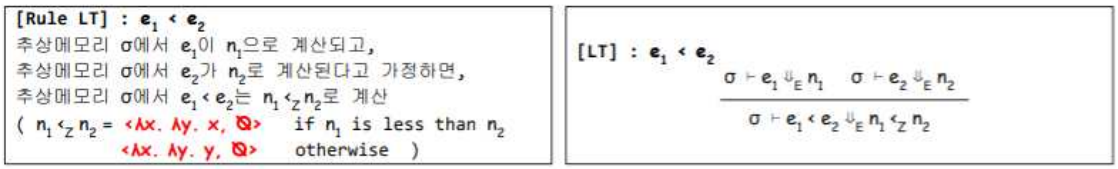
<과제 구현 중 어려웠던 부분 및 해결 과정>

```
let rec interp_e (s : Store.t) (e : Ast.expr) : Store.value =
        match e with
        | Num n -> NumV n
        | Add (e1,e2) ->
                        begin
                                let exp1 = interp_e s e1 in
                                let exp2 = interp_e s e2 in
                                match exp1,exp2 with
                                | NumV n1, NumV n2 -> NumV(n1 + n2)
                                | _ , _ -> failwith (Format.asprintf "Invalid addition: %a + %a" Ast.pp_e e1 Ast.pp_e e2)
                        end
        | Sub (e1,e2) ->
                        begin
                                let exp1 = interp_e s e1 in
                                let exp2 = interp_e s e2 in
                                match exp1,exp2 with
                                | NumV n1, NumV n2 -> NumV(n1 - n2)
                                | _ , _ -> failwith (Format.asprintf "Invalid subtraction: %a - %a" Ast.pp_e e1 Ast.pp_e e2)
                        end
        | Id str -> Store.find str s
        | LetIn (str,e1,e2) -> interp_e (Store.insert str (interp_e s e1) s) e2
        | App (e1,e2) ->
                        begin
                                let v1 = interp_e s e2 in
                                match interp_e s e1 with
                                | ClosureV (x,e3,s') -> interp_e (Store.insert x v1 s') e3
                                | _ -> failwith (Format.asprintf "Not a function: %a" Ast.pp_e e1)

                        end

        | Fun (str,e) -> ClosureV(str,e,s)
        | Lt (e1,e2) ->
                        begin
                                let v1 = interp_e s e1 in
                                let v2 = interp_e s e2 in
                                match v1,v2 with
                                | NumV n1, NumV n2 -> if n1 < n2 then ClosureV("x",Fun("y",Id("x")),[]) else ClosureV("x",Fun("y",Id("y")),[])
                                | _, _ -> failwith (Format.asprintf "Invalid less-than: %a < %a" Ast.pp_e e1 Ast.pp_e e2)
                        end
```

[interpreter.ml 의 interp_e 함수]

● 이번 과제에서 가장 어려웠던 부분은 Lt패턴에서 반환되게 되는 true와 false에 대한 ClosureV를 표현하는 것이었다.

● 이론 자료에 제시되었던



Lt에 대한 Bigstep-Operation에서 보아도 뭔가 감이 잘 오지 않았다. 처음에는 ClosureV에 들어가는 string 타입은 한 개인데 어떻게 두 개로 표현할지에 대해 가장 많은 고민을 했었는데 답이 나오지 않아 질문을 통해 해당 부분을 해결하였다. 질문에 대한 답변을 받았을 때 내가 이해를 잘못했었구나라는 생각이 들었고, 다시 이해하여 해당 부분을 바로 잡을 수 있었다.

<새로 고찰한 내용 또는 느낀점>

● 이번 과제에서는 syntatic sugar를 적용하여 boolean 타입과 Cond 조건문을 해석할 수 있는 interpreter를 구현하는 과제였다. 사실 처음 boolean 타입을 ClosureV로 표현하게 됨으로써 어떻게 동작하게될까 하면서 머리로는 바로 생각하기가 어려웠다. 하지만 이번 과제를 진행하며 조금씩 이해해가면서 동작에 대해 이해할 수 있었고, true와 false에 대한 표현에 대해서도 다시 한번 생각해 볼 수 있었던 것 같다. 더 열심히 해야겠다.