L12. First-class Function

유형	강의
복습 여부	✓
Status	Done

Where are we now ?

- **AE**
    - supports addition and subtraction
- **WAE**
    - supports identifiers
    - implement substitution
    - implement deferred substitution
- **F1WAE**
    - supports function
    - implement substitution
    - implement deferred substitution
    - return type
        - ex. 2
- **FWAE**
    - supports first-class function
    - implement substitution
    - return type
        - ex. (num 2)
- **FAE**
    - supports first-class function
    - implement **deferred substitution**
    - return type
        - ex. (numV 2)

No More ‘with’

Remove ‘with’ syntax in our abstract syntax. But we will still use it in our concrete syntax.

{with {x 10} x}

is the same as

{{fun {x} x} 10}

In general,

{with {<id> <FWAE>$_1$} <FWAE>$_2$}

is the same as

{{fun {<id>} <FWAE>$_2$} <FWAE>$_1$}

Let's assume

(with '<id> <FWAE>$_1$ <FWAE>$_2$)

⇒ (app (fun '<id> <FWAE>$_2$) <FWAE>$_1$)

Goal of our interpreter

Support static scope.

```
{with {x 3} {with {f {fun {y} {+ x y}}} {with {x 5} {f 4}}}}
```

When dealing with the function call, ‘f’ will be replaced with the function body and also provided a cache for the bound identifier ‘x’ in ‘{fun {y} {+ x y}}’.

FAE

```
<FAE> ::= <num>
        | {+ <FAE> <FAE>}
        | {- <FAE> <FAE>}
        | {with {<id> <FAE>} <FAE>}   ← Keep this for concrete syntax!
                                        Remove this for abstract syntax!
        | <id>
        | {fun {<id> <FAE>}
        | {<FAE> <FAE>}
```

- We'll still use 'with' in example code (concrete syntax).
- No more case lines in interp and other functions for 'with'
- No more test cases for interp and other functions using 'with'

```
; FAE
(define-type FAE
  [num (n number?)]
  [add (lhs FAE?) (rhs FAE?)]
  [sub (lhs FAE?) (rhs FAE?)]
  [id (name symbol?)]
  [fun (param symbol?) (body FAE?)]
  [app (ftn FAE?) (arg FAE?)])
```

Parser

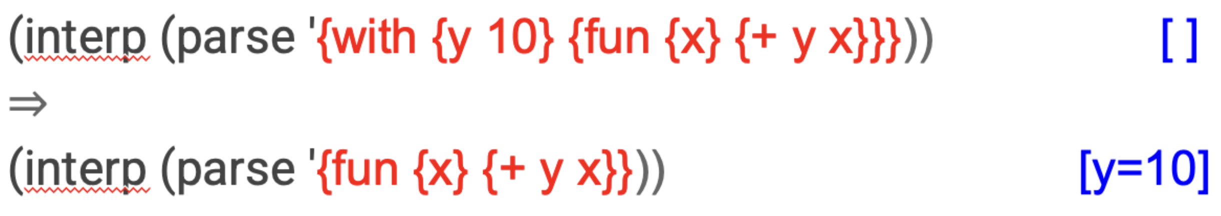
```
; [contract] parse : sexp -> FAE
(define (parse sexp)
  (match sexp
    [(? number?) (num sexp)]
    [(list '+ l r) (add (parse l) (parse r))]
    [(list '- l r) (sub (parse l) (parse r))]
    [(list 'with (list i v) e) (app (fun i (parse e)) (parse v))]
    [(? symbol?) (id sexp)]
    [(list 'fun (list p) b) (fun p (parse b))]
    [(list f a) (app (parse f) (parse a))]
    [else (error 'parse "bad syntax:~a" sexp)]))
```

```
(parse '{with {x 3} {+ x x}})
(parse '{with {x 3} {with {f {fun {y} {+ x y}}} {with {x 5} {f 4}}}})
```

```
(app (fun 'x (add (id 'x) (id 'x))) (num 3))
(app (fun 'x (app (fun 'f (app (fun 'x (app (id 'f) (num 4))) (num 5))) (fun 'y (add (id 'x) (id 'y))))) (num 3))
```

FAE with Deferred Substitution

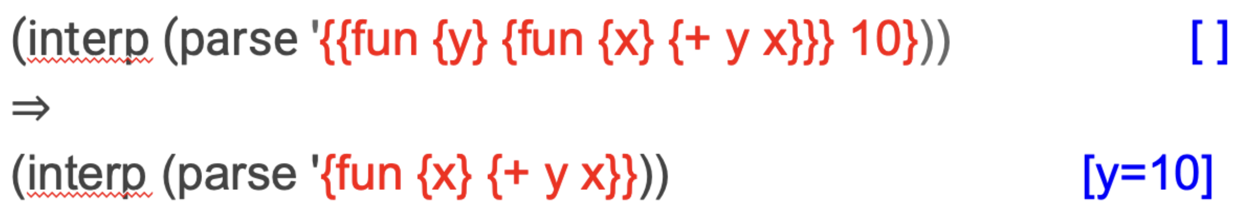
Ex1. (interp (parse '{with {y 10} {fun {x} {+ y x}}}))


(interp (parse '{with {y 10} {fun {x} {+ y x}}})) [ ]

⇒

(interp (parse '{fun {x} {+ y x}})) [y=10]

아래는 Ex1의 다른 표현


(interp (parse '{{fun {y} {fun {x} {+ y x}}} 10})) [ ]

⇒

(interp (parse '{fun {x} {+ y x}})) [y=10]

Ex2. (interp (parse '{{with {y 10} {fun {x} {+ y x}}} {with {y 7} y}}))

(interp (parse '{{with {y 10} {fun {x} {+ y x}}} [ ]
{with {y 7} y}}))

Argument expression:
(interp (parse '{with {y 7} y})) [ ]
⇒ (interp (parse 'y)) [y=7]
⇒ 7

Function expression:
(interp (parse '{{with {y 10} {fun {x} {+ y x}}} [ ]
⇒ (interp (parse '{fun {x} {+ y x}})) [y=10]
⇒ ??? ⇐ We need a new way to represent this function and this cache together.

Function: {fun {x} {+ y x}} [y=10]
Argument: 7

To apply, interpret the function body with the given argument:
(interp (parse '{+ y x})) [x=7 y=10]

Complete Interpreter

> Any bound ids which are not a parameter of a function need to be kept in its substitution cache with its corresponding value so that we can avoid dynamic scope and we will not forget the pending substitution for the function.

Implement a new type, 'FAE-Value' that supports numbers and functions with cache.

```
; FAE
(define-type FAE
  [num (n number?)]
  [add (lhs FAE?) (rhs FAE?)]
  [sub (lhs FAE?) (rhs FAE?)]
  [id (name symbol?)]
  [fun (param symbol?) (body FAE?)]
  [app (ftn FAE?) (arg FAE?)])

; FAE-Value
(define-type FAE-Value
  [numV (n number?)]
  [closureV (param symbol?) (body FAE?) (ds DefrdSub?)])

; DefrdSub
(define-type DefrdSub
  [mtSub]
  [aSub (name symbol?) (value FAE-Value?) (ds DefrdSub?)])
```

```
; Interpreter
; [contract] FAE DefrdSub -> FAE-Value
(define (interp fae ds)
  (type-case FAE fae
    [num (n) (numV n)]
    [add (l r) (num+ (interp l ds) (interp r ds))]
    [sub (l r) (num- (interp l ds) (interp r ds))]
    [id (s) (lookup s ds)]
    [fun (p b) (closureV p b ds)]
    [app (f a) (local
                 [(define f-val (interp f ds))
                  (define a-val (interp a ds))]
                 ; function definition is saved in closureV type
                 (interp (closureV-body f-val)
                         (aSub (closureV-param f-val)
                               a-val
                               (closureV-ds f-val))
                         )
                 )
         ]
    )
  )
```

```
; num-op
(define (num-op op)
  (lambda (x y)
    (numV (op (numV-n x) (numV-n y)))))

(define num+ (num-op +))
(define num- (num-op -))

; lookup : symbol DefrdSub -> number
(define (lookup name ds)
  (type-case DefrdSub ds
    [mtSub () (error 'lookup "free identifier")]
    [aSub (i v saved) (if (symbol=? i name)
                          v
                          (lookup name saved))]
    )
  )
```

```
(test (interp (app (fun 'x (app (fun 'f (app (fun 'x (app (id 'f) (num 4))) (num 5))) (fun 'y (add (id 'x) (id 'y))))) (num 3)) (mtSub)) (n
(test (interp (parse '{with {y 10} {fun {x} {+ y x}}}) (mtSub))(closureV 'x (add (id 'y) (id 'x)) (aSub 'y (numV 10) (mtSub))))
```

```
good (interp (app (fun 'x (app (fun 'f (app (fun 'x (app (id 'f) (num 4))) (num 5))) (fun 'y (add (id 'x) (id 'y))))) (num 3)) (mtSub)) at line 82
  expected: (numV 7)
  given: (numV 7)

good (interp (parse '(with (y 10) (fun (x) (+ y x)))) (mtSub)) at line 86
  expected: (closureV 'x (add (id 'y) (id 'x)) (aSub 'y (numV 10) (mtSub)))
  given: (closureV 'x (add (id 'y) (id 'x)) (aSub 'y (numV 10) (mtSub)))
```

Differences between FAE and F1WAE

F1WAE

```
; [contract] interp : F1WAE list-of-FuncDef DefrdSub -> number
(define (interp f1wae fundefs ds)
  (type-case F1WAE f1wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs ds) (interp r fundefs ds))]
    [sub (l r) (- (interp l fundefs ds) (interp r fundefs ds))]
    [with (i v e) (interp e fundefs (aSub i (interp v fundefs ds) ds))]
    [id (s) (lookup s ds)]
    [app (f a) (local
                 [(define a_fundef (lookup-fundef f fundefs))]
                 (interp (fundef-body a_fundef)
                         fundefs
                         (aSub (fundef-arg-name a_fundef)
                               (interp a fundefs ds)
                               (mtSub))))]))
```

In F1WAE, function definition was separately defined. So, substitution wasn't needed for the entire function. From 'fundefs', we obtained the function.

```
; (parse '{f 1})
; (parse-fd '{deffun {f x} {+ x 3}})
(test (interp (app 'f (num 1)) (list (fundef 'f 'x (add (id 'x) (num 3)))) (mtSub)) 4)
```

```
good (interp (app 'f (num 1)) (list (fundef 'f 'x (add (id 'x) (num 3)))) (mtSub)) at line 76
   expected: 4
   given: 4
```

FAE (incomplete)

```
; FAE
(define-type FAE
  [num (n number?)]
  [add (lhs FAE?) (rhs FAE?)]
  [sub (lhs FAE?) (rhs FAE?)]
  [id (name symbol?)]
  [fun (param symbol?) (body FAE?)]
  [app (ftn FAE?) (arg FAE?)])
```

```
; interp FAE DefrdSub -> FAE
(define (interp fae ds)
  (type-case FAE fae
    [num (n) fae]
    [add (l r) (num+ (interp l) (interp r))]
    [sub (l r) (num- (interp l) (interp r))]
    [id (s) (lookup s ds)]
    [fun (p b) fae]
    [app (f a) (local
                 ([define ftn (interp f ds)])
                 (interp (fun-body ftn)
                         (aSub (fun-param ftn)
                               (interp a ds)
                               ds))
                 )
         ]
    )
  )
```

In FAE, a function is a value. So when it is assigned to an identifier, it must be dealt with like other identifiers.

In the above code, `app (f a)`, `f` can be either function definition itself or identifier of the function definition.

We can assign function definition to specific identifier and save that information in deferred substitution cache. So deferred substitution cache must have the identifier of the function and its value, which is function definition. Then, if we call that function using the identifier, we just substitute the identifier with value (our original function definition).

```
(interp (parse '{with {x 3} {with {f {fun {y} {+ x y}}} {with {x 5} {f 4}}}}) (mtSub))
```

`f` is saved in deferred substitution cache with value, `{fun {y} {+ x y}}` .

In `{f 4}` , when we call f, using information from the deferred substitution cache, we must substitute `f` with `{fun {y} {+ x y}}` .

Q. Dynamic Scope again ?

In `{fun {y} {+ x y}}` , if we consider static scope, x is 3.

However, in our current implementation, because `{f 4}`, it is affected by the binding identifier `x` in `{x 5}` , `x` is 5.

Q. How can we solve this problem ?

```
{with {x 3} {with {f {fun {y} {+ x y}}} {with {x 5} {f 4}}}}
```

So we have to find a way to ignore the `{x 5}` part when `{f 4}` is called.

When getting the corresponding function body of the identifier, why don't we always provide static value of `x` bound identifier together? Let's provide some kind of cache for this function definition.

This is why we need FAE-Value type for our return type.

FAE (Complete)

We created new data structure, FAE-Value to support static scope.

FAE-Value has two types, `numV` and `closureV`.

Previously, return type was FAE that could hold `num`, `add`, `fun` variants and so on. But from now on, we can only provide exact type for our return value. `numV` represents number, and `closureV` represents function definition with deferred substitution cache.

```
; FAE-Value
(define-type FAE-Value
  [numV (n number?)]
  [closureV (param symbol?) (body FAE?) (ds DefrdSub?)])
```

```
; Interpreter
; [contract] FAE DefrdSub -> FAE-Value
(define (interp fae ds)
  (type-case FAE fae
    [num (n) (numV n)]
    [add (l r) (num+ (interp l ds) (interp r ds))]
    [sub (l r) (num- (interp l ds) (interp r ds))]
    [id (s) (lookup s ds)]
    [fun (p b) (closureV p b ds)]
    [app (f a) (local
                 [(define f-val (interp f ds))
                  (define a-val (interp a ds))]
                 ; function definition is saved in closureV type
                 (interp (closureV-body f-val)
                         (aSub (closureV-param f-val)
                               a-val
                               (closureV-ds f-val))
                         )
                 )
         ]
    )
  )
```

```
; parse '{with {x 3} {with {f {fun {y} {+ x y}}} {with {x 5} {f 4}}}}
(interp (app (fun 'x (app (fun 'f (app (fun 'x (app (id 'f) (num 4))) (num 5))) (fun 'y (add (id 'x) (id 'y))))) (num 3)) (mtSub))
```

Q. what happens when {f 4} is interpreted ?

```
(interp (app (id 'f) (num 4))
        (aSub 'x
              (numV 5)
              (aSub 'f
                    (closureV 'y (add (id 'x) (id 'y)) (aSub 'x (numV 3) mtSub))
                    (aSub 'x (numV 3) mtSub)
                    )
              )
        )
```

```
f-val
(interp 'f
        (aSub 'x
              (numV 5)
              (aSub 'f
                    (closureV 'y (add (id 'x) (id 'y)) (aSub 'x (numV 3) mtSub))
                    (aSub 'x (numV 3) mtSub)
                    )
              )
        )
(lookup 'f
        (aSub 'x
              (numV 5)
              (aSub 'f
                    (closureV 'y (add (id 'x) (id 'y)) (aSub 'x (numV 3) mtSub))
                    (aSub 'x (numV 3) mtSub)
                    )
              )
        )
(closureV 'y (add (id 'x) (id 'y)) (aSub 'x (numV 3) mtSub))
```

We can get both the function definition of ‘f and the cache information.

```
a-val
(interp (num 4)
        (aSub 'x
              (numV 5)
              (aSub 'f
                    (closureV 'y (add (id 'x) (id 'y)) (aSub 'x (numV 3) mtSub))
                    (aSub 'x (numV 3) mtSub)
                    )
              )
        )
(numV 4)
```

When interpreting the function body of f-val, the cache information related to ‘f is also considered.

```
(interp (add (id 'x) (id 'y))
        (aSub 'y
              (numV 4)
              (aSub 'x (numV 3) mtSub)
              )
        )
```

the value of identifier ‘x is correctly substituted with 3, not 5.

```
(num+ (interp (id 'x)
              (aSub 'y
                    (numV 4)
                    (aSub 'x (numV 3) mtSub)
                    )
              )
      (interp (id 'y)
              (aSub 'y
                    (numV 4)
                    (aSub 'x (numV 3) mtSub)
                    )
              )
      )

(lookup 'x (aSub 'y
                 (numV 4)
                 (aSub 'x (numV 3) mtSub)
                 )
        )
(numV 3)

(lookup 'y (aSub 'y
                 (numV 4)
                 (aSub 'x (numV 3) mtSub)
                 )
        )
(numV 4)

(num+ (numV 3) (numV 4))
(numV (+ 3 4))
(numV 7)
```

We get the expected result.