Avishai Yaniv 307916023
Alex Abramov 314129438

1.1:

set! expressions purpose is to change a var of an existing val, just like a define expression, which is a different purpose than all other regular form evaluations, thus, must be evaluated as a special form. If we try to evaluate set! as a regular form, the interpreter will evaluate it as an app-exp which is again, not the purpose of set! and can causes errors.

1.2:

The Box usage in the current implementation of the fbinding datatype, reduces fbindings changing capabilities. We can still implement the fbinding datatype without using Box, by using an array for example. It requires us to take care of the actions very carefully, so that we support frames addings whenever *define* is met, or updating existing bindings in case of *letrec*. In addition, we can even ignore Box implementation at all, because Box is only an encapsulation and doesn't change the program logics.

1.3:

e. {f:[T1->T2], h:Number} |- (f g): T2 **- false**

- g is not defined, so f ensure g is of the correct type, which is T1. Therefore, cannot ensure that T2 is the returned value.

f. {f:[T1->T2], h:Number} |- (f h): T2 **- false**

- T1 is of an implicit type, therefore we cannot assume that it can deal with numbers, hence we cannot ensure that T2 is the resulted from numbers.

g. {f:[T1->T2], x:T1} |- (f x x): T2 **- false**

- Given function f which receives only one parameters, it is obvious we can not determine it returns T2 when given two parameters.

1.4:

((lambda (x1) (if (> x1 7) #t #f)) 8)

1. renaming bound vars:
   ((lambda (x1) (if (> x1 7) #t #f)) 8) turn to ((lambda (x) (if (> x 7) #t #f)) 8)

2. Assigning types:

Expression	Variable
((lambda (x) (if (> x 7) #t #f)) 8)	T0
(if (> x 7) #t #f)	Tif
(> x 7)	Ttest
>	T>
x	Tnumber
7	Tnum7
#t	T#t
#f	T#f
8	Tnum8

3. Creating equations:

$T_0 = [T_{num} \rightarrow T_{if}]$
$T_> = [\text{Number} * \text{Number} \rightarrow \text{Boolean}]$
$T_{if} = T_{\#f} = T_{\#t} = T_{test} = \text{boolean}$
$T_{number} = T_{num8} = T_{num7} = \text{Number}$

4. Solving equations:

$T_0 = [\text{Number} \rightarrow \text{Boolean}]$

Result:

```
((lambda (x: number)   : boolean
        (if (> x 7)        : boolean
                #t         : boolean
                #f         : boolean
        )                  : boolean
) 8: number)
        ↓
((lambda ([x: number])   : boolean
        (if (> x 7)
                #t
                #f
        )
) 8: number)
```

1.5:

a:

```
g(x: number, (gRes) => {
        f(gRes: number, (fRes) => {
                h(fRes: number, callback: (number → number));
        }:  (number → number)
}: (number → number)
```

b:

Avishai Yaniv 307916023
Alex Abramov 314129438

```
g(x, (gRes, gErr) => {
        if(gErr) failCallback(gErr);
        Else{
                f(gRes, (fRes, fErr) => {
                        if(fErr) failCallback(fErr);
                        Else{
                                h(fRes, callback);
                        }
                }
        }
}
```

1.6:

```
(define even? (lambda (n cont)
        (if (= n 0)
                (cont #t)
                (odd? (- n 1) (lambda (res) (cont res)))
        )))
(define odd? (lambda (n cont)
        (if (= n 0)
                (cont #f)
                (even? (- n 1) (lambda (res) (cont res)))
        )))
(define f (lambda (x cont)
        (if (even? x cont)
        (cont x)
        (cont (- x 1)))
        ))
```