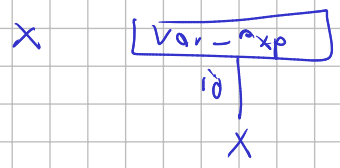
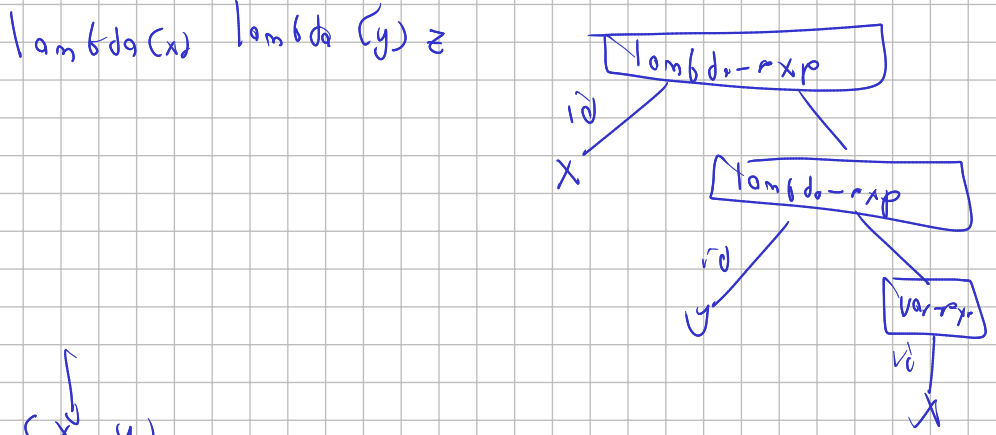
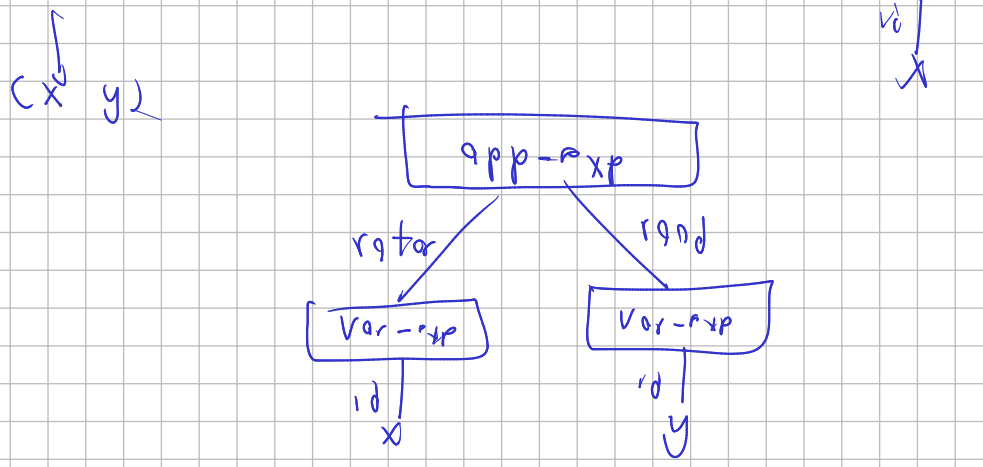
```
<lc-exp> ::= <identificador>
             var-exp(id)
         ::= "lambda" "(" <identificador> ")" <lc-exp>
             lambda-exp(id, exp)
         ::= "(" <lc-exp> <lc-exp> ")"
             app-exp(rator, rand)
```

```
x
lambda(x) lambda (y) z
(x y)
```







Los tipos de datos definidos recursivamente se pueden trabajar con AST (Arboles de Sintaxis abstracta)

```
(define-datatype <tipo> <predicado>
     (<variante> (nombre tipo) (nombre tipo) ...
)
```

```
<lc-exp> ::= <identificador>
             var-exp(id)
         ::= "lambda" "(" <identificador> ")" <lc-exp>
             lambda-exp(id, exp)
         ::= "(" <lc-exp> <lc-exp> ")"
             app-exp(rator, rand)
```

```
(define-datatype lc-exp lc-exp?
        (var-exp (id symbol?))
        (lambda-exp (id symbol?)
                    (exp lc-exp?)
        (app-exp (rator lc-exp?)
                 (rand lc-exp?)
        )
)
```

Parser y unparser

Parser: sintaxis concreta (codigo fuente) => AST

Unparser: AST => sintaxis concreta (entendible por un programador)