17-1 / Logic Programming

prolog
datalog

relations and inference rules
then ask queries

"github jeapostrophe teachlog"

```
rel edge/2.      edge(a,b). edge(b,c). edge(c,d). edge(a,d).
rel path/2.      edge(a,e). edge(e,c). edge(e,b).
rel cycle/1.
```

$$\begin{tikzcd}
A \arrow[r] \arrow[rrr, bend left] \arrow[dr] & B \arrow[r] & C \arrow[r] & D \\
 & E \arrow[u] \arrow[ur] & &
\end{tikzcd}$$

```
path(X,Y) :- edge(X,Y).
path(X,Z) :- edge(X,Y), path(Y,Z).
cycle(X) :- path(X,X).
```

cycle(X)?

edge(c,a).
cycle(X)?
next.

17-2 / Non-deterministic computation

$nd ::= ans(v)$ $\quad v \in$ Answers

$\quad | \ fail$

$\quad | \ choice(nd, nd)$ $\quad nd \in$ computations

$\quad | \ bind(nd, f)$ $\quad f \in$ Answer $\to$ computation

$run : ND \ X \Rightarrow stream(X)$

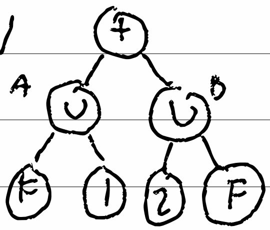
$run \ p = sols \ [\langle p, kret \rangle]$

$k ::= kret \ | \ kbind(f, k)$

17-3/

```
sols : List < Pair < ND , Kont>> -> Stream<Ans>
  sols [] = []
  sols <p, k> : q =
   case p of
     bind p' f -> sols <p', kbind (f, k> : q
     choice p1 p2 -> sols <p1 , k> : <p2 , k> : q
     fail -> sols q
     ans(v) -> case k of   kret -> yield v ;
                                   sols q
                           kbind (f, k) -> sols <f v , k> : q
```

17-4)



```
A = choice fail ans(1)
B = choice ans(2) fail
C = bind A (λ av →
    D bind B (λ bv →
      E ans (av + bv))

[<C, kret>] ⇒ [<A, kbind (λ av→ D) kret>]
            ⇒ [<fail, "">, <ans(1), "">]
            ⇒ [<ans(1), "">]
            ⇒ [<D 1, kret>]
            ⇒ [<B, kbind (λbv → E 1) kret>]
            ⇒ [<ans(2), "">, <fail, "">]
            ⇒ [<E 1 2 = ans(3), kret>, <fail, "">]
            ⇒ 3 : sols [<fail, "">] ⇒ 3 : [] ⇒ [3]
```

17-5)

```
searchTop : rules x query -> stream(ans)
Search Top rules q =
  run (bind (search N rules ∅ [q])
            (λ (env) (ans (extract env q))))


SearchN : rules x env x list(query) -> nd(env)
SearchN rules env qs = let p = (ans env) in
    for q ∈ qs do  p = bind p (λ env' ->
                          search* rules env' rules q)
    return p
```

17-6/

```
search* : rules × env × rules × g ⇒ nd (env)
   search* allrules env rules g =
     case rules of    [] ⇒ fail
                      r::mrs ⇒ choice (search* allrules env
                                                mrs g)
                                       (search1 allrules env
                                                 r g)
```

```
path(X,Z) :- edge(X,Y),
             path(Y,Z).
```

```
search1 : rules × env × rule × g ⇒ nd (env)
search1 allrules env rule1      g =
   let (head, body) = (rule1) in
   bind (unify env head g)
        (λ (env') (search N allrules env' body))
```

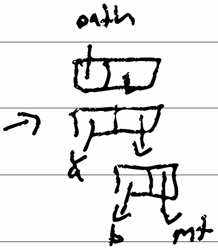
$\boxed{7-7}$

$$\begin{tikzcd}
 & \text{searchT} \arrow[dr] & \\
 & & \text{searchN} \arrow[d] \\
 & & \text{search*} \arrow[d] \\
\text{unify} & & \text{searchI} \arrow[ll] \arrow[uu, bend right=90]
\end{tikzcd}$$

17-8/

```
unify $\emptyset$ path(X,Y) path(a,b)
    = $[X \mapsto a, Y \mapsto b]$
unify $[X \mapsto a]$ path(X,Y) path(a,b)
    = $[X \mapsto a, Y \mapsto b]$
unify $[X \mapsto z]$ path(X,Y) path(a,b)
    = fail

unify env x x = ans(env)
unify env Var(a) rhs = case env(a) of
                           $\bot$ $\Rightarrow$ ans(env[a $\mapsto$ rhs])
                           av $\Rightarrow$ unify env av rhs
unify env lhs Var(a) = unify env Var(a) lhs
unify env cons(la,ld) cons(ra,rd) = bind (unify env la ra)
                                         ($\lambda$ env' $\Rightarrow$ unify env' ld rd)
```



17-9/ extract : env x query ⇒ query

```
extract env []        = []
            cons(a,d) = cons (extract env a)
                             (extract env d)
            var(x)    = extract env env(x)
            v         = v
```