a6 tour

Sum types

```
(define-type bool+rat
   [Inl (left bool?)]
   [Inr (right rat?)]
)
```

```
(Inl b)   where b boolean
(Inr r)   where r rat
```

bool + rat

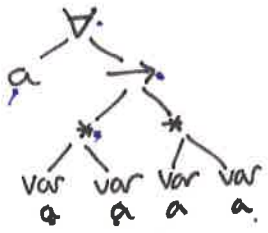
```
(type-case bool+rat s
   [Inl (xL) eL]
   [Inr (xR) eR]
)
```

```
{Sum-case s {Inl xL => eL}
            {Inr xR => eR}}
(Sum-case s xL eL xR eR)
```

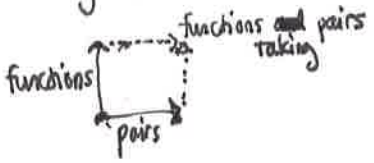
Polymorphic types

$\forall a.(a*a) \rightarrow (a*a)$

for all ,



"Orthogonal" features

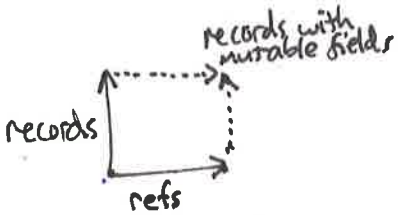


Fun

vs.

Racket

functions of two arguments



vs.

Objects are:
records with some mutable fields (instance variables)
and some immutable fields (methods)
and special operators to create objects
and subtyping...

vs.

define-type'd types are:
product types

and sum types

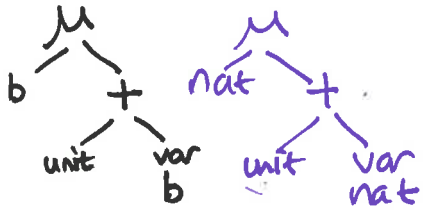
and recursive types

Combining features into one language construct:
sometimes convenient,
but hides similarity

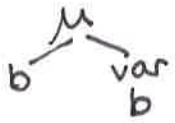
Recursive types

$\mu b.(unit + b)$

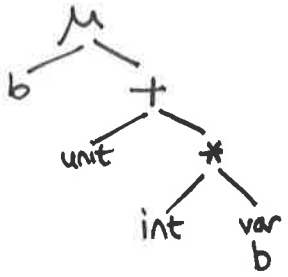
mu



$\mu b.\ b$



$\mu b.\ (unit + (int * b))$



Recursive expressions

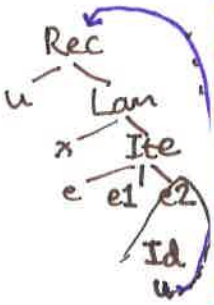
(Rec u e)

(Rec u (Id u))

(Rec u (Lam x (Ite e e1 e2)))

base case

recursive case



Example: ~~cons 1 2 empty~~

(cons 1 (cons 2 empty))