DFb - Fb w/ declarations

TFb : environment

$D ::= d, \ldots$

$d ::= \text{Let } x = e;;$

$\Delta ::= \{x \mapsto v, \ldots\}$

$\Delta \vdash e \Rightarrow v$

$\Delta \vdash D \Rightarrow \Delta$

$$\frac{d_1 = \text{Let } x = e;; \quad \Delta_1 \vdash e \Rightarrow v \quad \Delta_2 = \Delta_1[x \mapsto v] \quad \Delta_2 \vdash [d_2, \ldots, d_n] \Rightarrow \Delta_3}{\Delta_1 \vdash [d_1, d_2, \ldots, d_n] \Rightarrow \Delta_3}$$

$$\frac{x \mapsto v \in \Delta}{\Delta \vdash x \Rightarrow v}$$

DTFbSRAM $\approx$ C

C++

* C
* OO
* Type decls
* overloading
* typecasts (C)
* templates

Python

* procedural
* OO (crazy)

* type conversions

* GC

Bounded Polymorphism

$$\text{fun } x \to x \;:\; \forall \tau.\ \tau \to \tau$$

dog <: animal

dog list <: animal list

dog array <?: animal array

```
let d : dog array = new ... in
let a : animal array = d in
a[0] := new Cat;
let x : dog = d[0] in
...
```

dog type

```
{
  get : int → dog
  set : int → dog → ()
  reflist : ...
}
```

<:

animal type

```
{
  get : int → animal
  set : int → animal → ()
  reflist ...
}
```

int → dog <: int → animal

int → dog → () <: int → animal → ()

✗ `List<Animal> list = new ArrayList<Dog>();`

✓ `List<? extends Animal> list = new ArrayList<Dog>();`

Neat Things

* Logic languages
* Rust — ownership
* Haskell — :) :(
  * Monad