

Compiler Testing with Sized Types

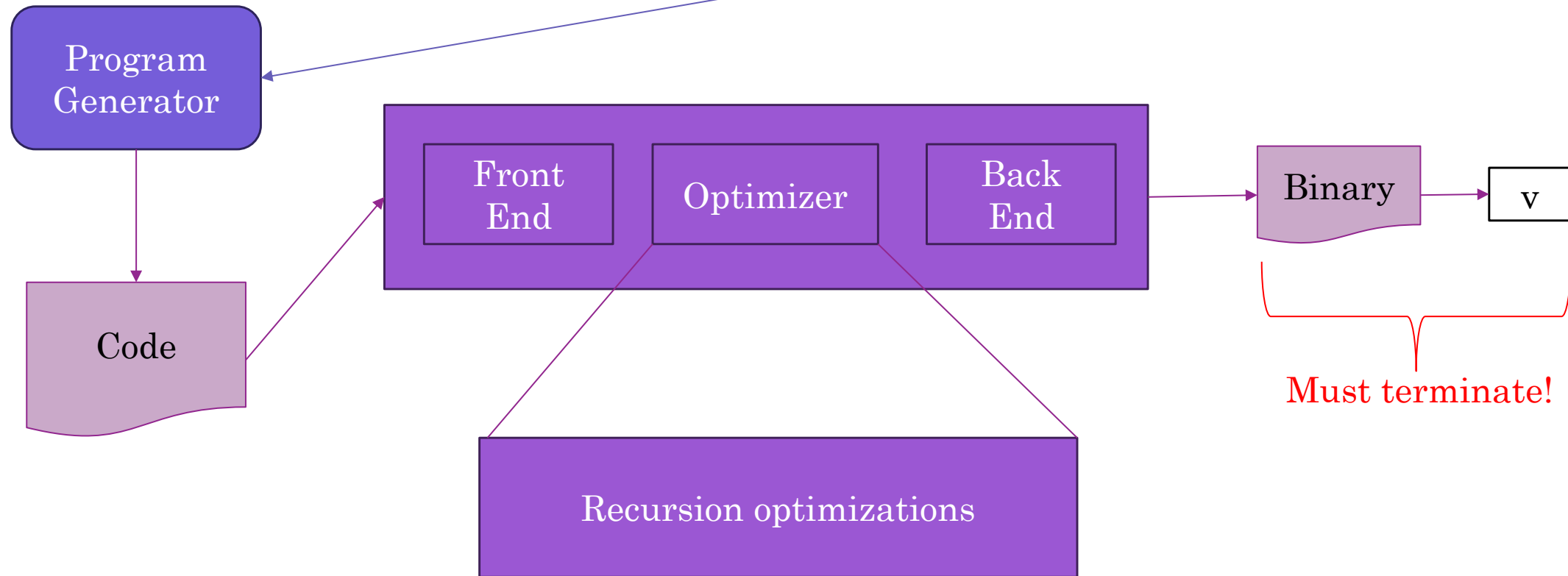
Caspar Popova, University of Maryland



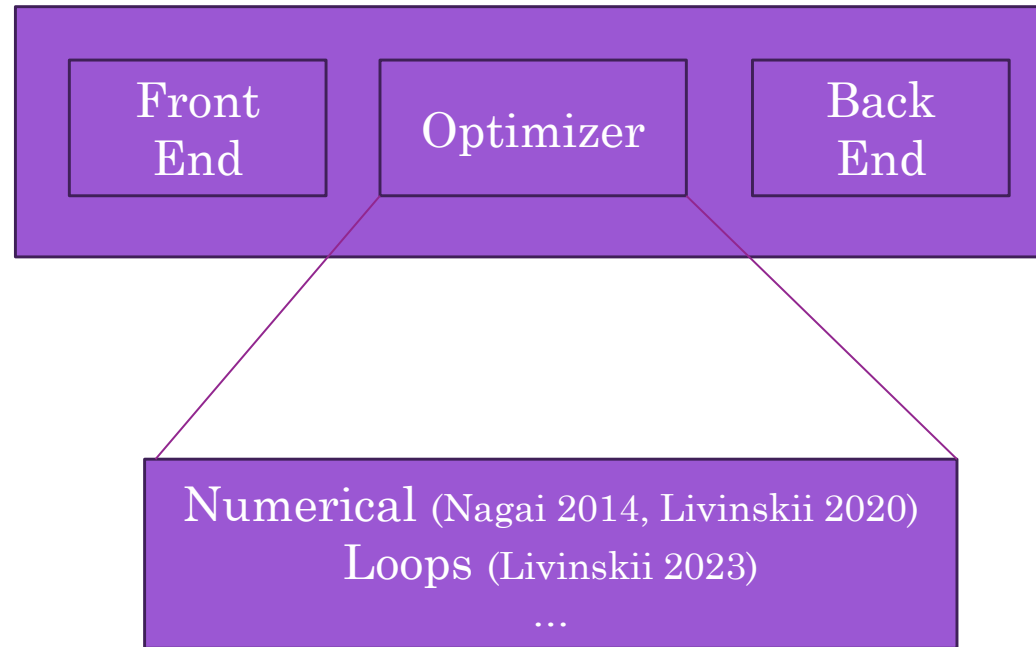
Joint work with Harry Goldstein, Leonidas Lampropoulos

NJPLS ~ December 05, 2025

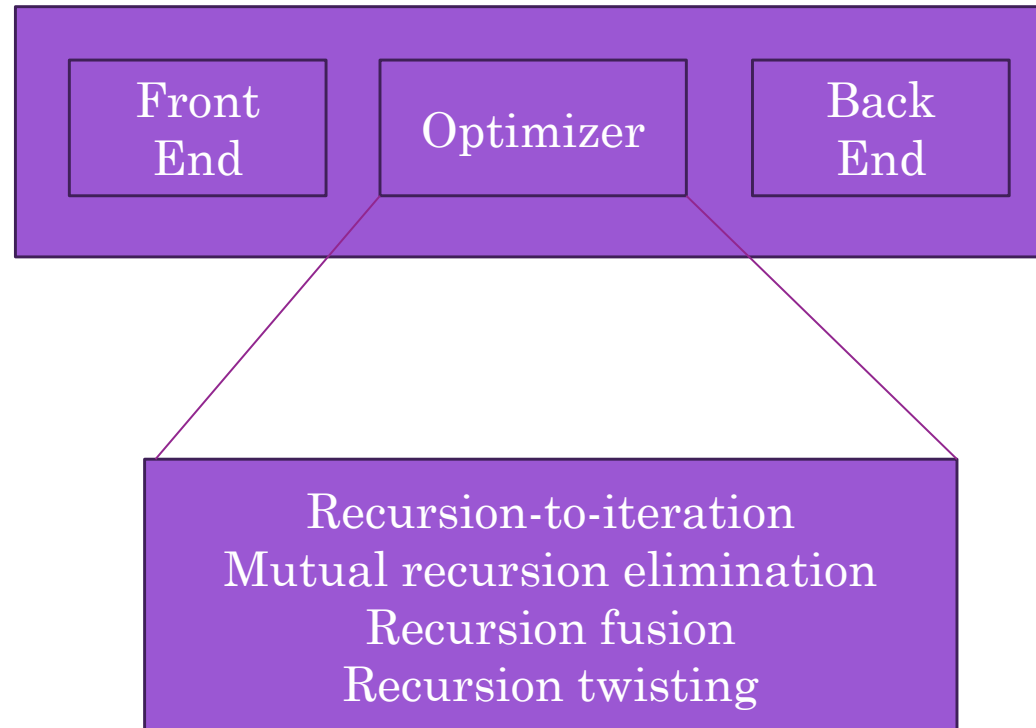
Compiler Testing with Sized Types



Optimization testing



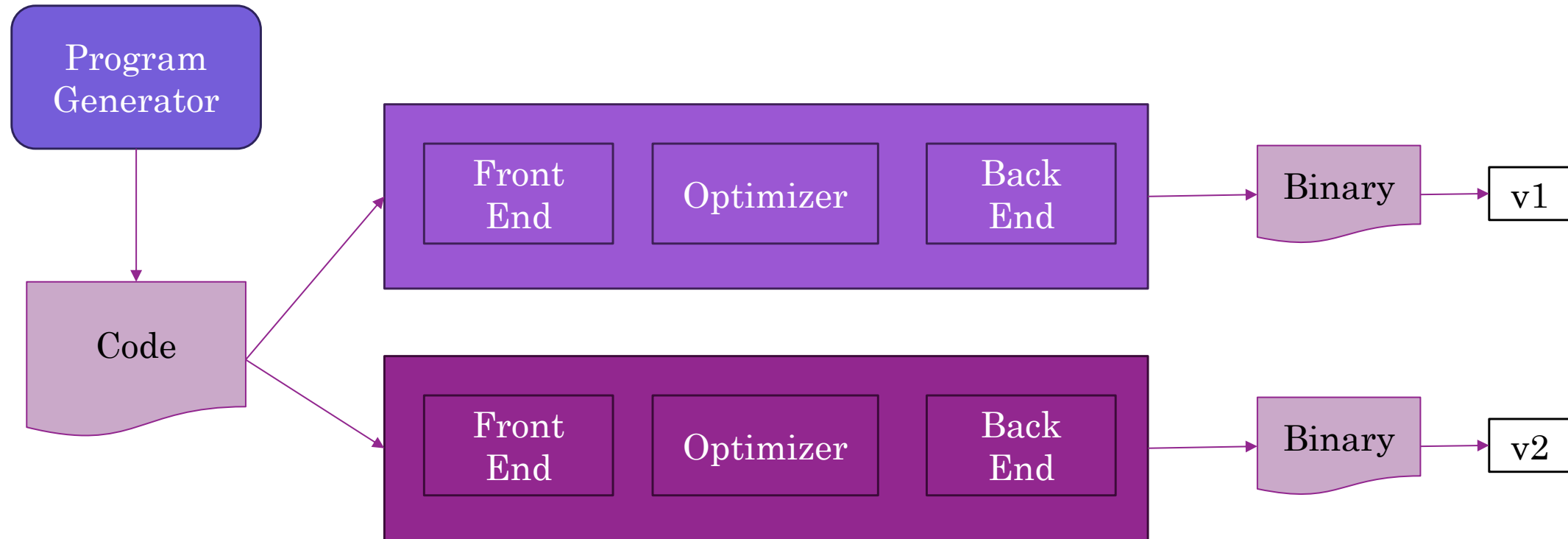
Recursion Optimizations



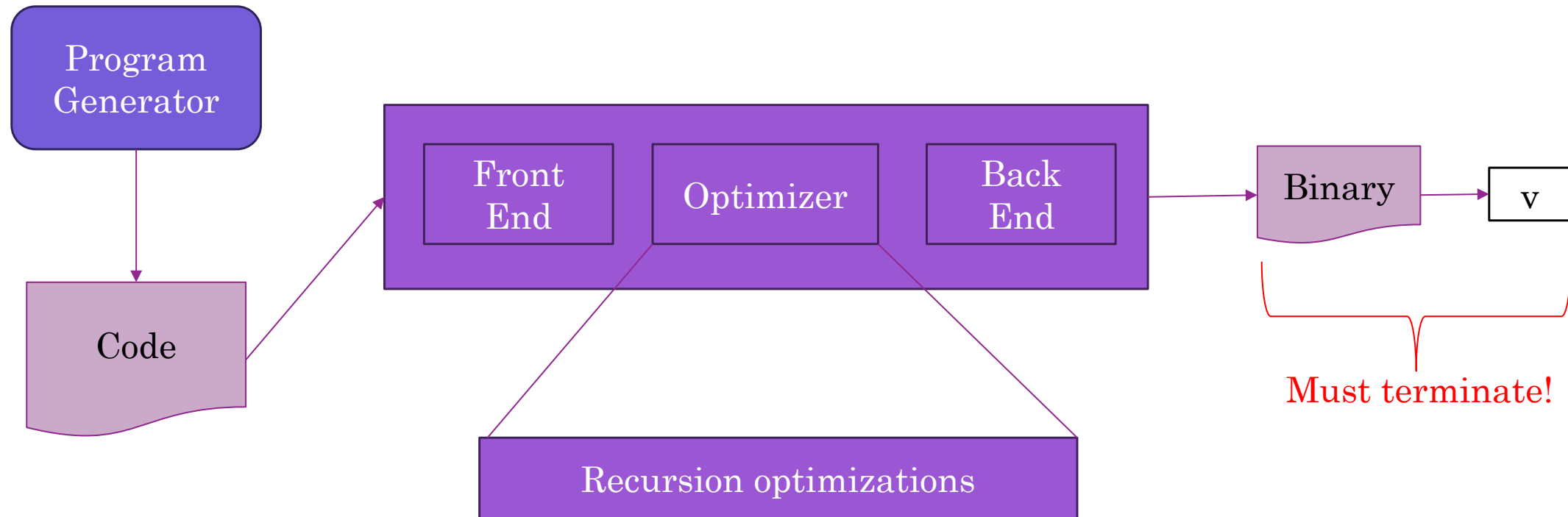
broken inlining
heuristic
(Racket#3027)



Why do we need termination?



Challenge: Termination



Program Generation

$$\frac{\Gamma \vdash \square : \text{Nat} \rightsquigarrow e_1 : \text{Nat} \quad \Gamma \vdash \square : \text{Nat} \rightsquigarrow e_2 : \text{Nat}}{\Gamma \vdash \boxed{\square : \text{Nat}} \rightsquigarrow (e_1 + e_2)} \text{ ADD}$$

$$\square : \text{Nat} \rightsquigarrow (1 + 2)$$

Program Generation

$$\frac{\Gamma, x : \tau_1, f : \tau_1 \rightarrow \tau_2 \vdash \Box : \tau_2 \rightsquigarrow e}{\Gamma \vdash \Box : \tau_1 \rightarrow \tau_2 \rightsquigarrow (\text{let rec } f \text{ } x = e)} \text{REC}$$

$\Box : \text{Nat} \rightarrow \text{Nat} \rightsquigarrow$

```
let rec f x =  
  match x with  
  | 0 -> f x  
  | S x' -> f x'
```

Inspiration from termination checking

- Sized types → **type theoretic and compositional** 😊
- Guard predicate → used in Rocq; structural/syntactic condition, not compositional
- Well-founded relations → used in recursive program synthesis (Miltner 2024, Polikarpova 2024, Choi 2023); often requires proofs

Sized Types to the rescue

$$\begin{array}{l} \text{Nat} := \text{o} : \text{Nat}^{i+1} \\ \quad | \text{s} : \text{Nat}^i \rightarrow \text{Nat}^{i+1} \end{array}$$

Size annotations on
inductive datatypes

$$\begin{array}{l} \text{add} : \text{Nat}^i \rightarrow \text{Nat} \rightarrow \text{Nat} \\ \text{minus} : \text{Nat}^i \rightarrow \text{Nat} \rightarrow \text{Nat}^i \\ \text{div} : \text{Nat}^i \rightarrow \text{Nat} \rightarrow \text{Nat}^i \end{array}$$

Size annotations on
functions

Sized Types to the rescue

$$\frac{\Gamma, x : \text{Nat}^{i+1}, f : \text{Nat}^i \rightarrow \theta \vdash \square : \theta[i := i + 1] \rightsquigarrow e}{\Gamma \vdash \square : \forall i. \text{Nat}^i \rightarrow \theta \rightsquigarrow (\text{let rec } f \text{ } x = e)} \quad \text{REC}$$

$\square : \text{Nat} \rightarrow \text{Nat} \rightsquigarrow$

$\text{let rec } f \text{ } x =$
 $\quad f \text{ } x$

Error! Does not terminate

$\frac{i \leq i + 1}{\text{Nat}^i \sqsubseteq \text{Nat}^{i+1}} \quad \text{SUBTYPE}$

Sized Types to the rescue

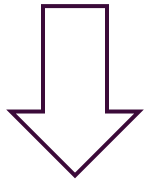
$$\frac{\Gamma \vdash \square : \boxed{\text{Nat}^{i+1}} \rightsquigarrow e_1 \quad \Gamma \vdash \square : \theta \rightsquigarrow e_2 \quad \Gamma, \boxed{x' : \text{Nat}^i} \vdash \square : \theta \rightsquigarrow e_3}{\Gamma \vdash \square : \theta \rightsquigarrow (\text{match } e_1 \text{ with } | o \rightarrow e_2 | s \ x' \rightarrow e_3)} \text{MATCH}$$

$\square : \text{Nat} \rightarrow \text{Nat} \rightsquigarrow$

```
let rec f x =  
  match x with  
  | 0 -> e2  
  | S x' -> f x'
```

Sized Types & composition

```
let rec div n m =  
  match n with  
  | 0 -> 0  
  | S n' -> S (div (□ : Nati) m)
```



```
let rec div n m =  
  match n with  
  | 0 -> 0  
  | S n' -> S (div (minus n' m) m)
```

$\Gamma = \{\text{div} : \text{Nat}^i \rightarrow \text{Nat} \rightarrow \text{Nat}^i,$
 $\text{minus} : \forall k. \text{Nat}^k \rightarrow \text{Nat} \rightarrow \text{Nat}^k,$
 $n : \text{Nat}^{i+1},$
 $n' : \text{Nat}^i\}$

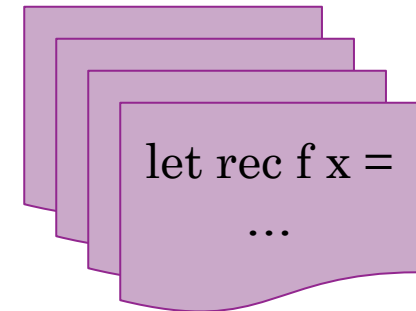
can be tricky with guard
predicates & well-founded
relations

Ongoing Work

Implemented in OCaml

Remaining challenges:

- Mutual recursion generation
- Evaluation



(Planned) Evaluation

- Toy compiler with manually injected bugs (CMSC 430)
- Differential testing across Racket languages
- Differential testing across ML compilers



Compiler Testing with Sized Types

