Type Systems as Macros

Stephen Chang, Alex Knauth, Ben Greenman
“Racket is a programming language for creating new programming languages.”
--- The Racket Manifesto (2015)
The Racket Way: Linguistic Reuse

- #%app
- lambda
- #%module-begin
- #%datum
- ...
IT'S DANGEROUS TO BUILD LANGUAGES FROM SCRATCH! TAKE THIS.
Racket-included languages
User-created #langs

- afl
- agile
- arc
- ckanren
- dssl
- ecmascript
- envy
- forth
- heresy
- honu
- lazy
- lux
- marketplace
- pollen
- pop-pl
- python
- r7rs
- rackjure
- rash
- remix
- rosette
- scratchy
- shill
- superc
- syndicate
- tulip
- urlang
- whalesong

All untyped!
“Racket is a programming language for creating new programming languages.”

--- The Racket Manifesto (2015)
Typed Racket

- Typecheck
- Expand
- Execute
Typed Racket

Typecheck
- Traverse program
- Compute program properties
- Check program properties
- Rewrite program

Expand

Execute

typed program
Typed Racket

Can we reuse the expander for typechecking? 
 i.e., the Racket Way!
Our approach to creating typed languages:

1. **Expand / Typecheck**
2. **Execute**
```
#lang LC

(define-simple-macro (lm x e) (λ (x) e))
(define-simple-macro (app e1 e1) (#%app e1 e2))
(provide (rename-out [lm λ] [app #%app]))
```
#lang LC + types = #lang STLC

#lang racket
(define-simple-macro (checked-λ [x : τ] e) #:when ???
  (λ (x) e))
(define-simple-macro (checked-app e1 e2) #:when ???
  (#%app e1 e2))
Type checking function application

Why not write type rules directly?
You can, with `#lang turnstile`