# Dependent Type Systems as Macros

Stephen Chang, Michael Ballantyne, Milo Turner, William J. Bowman*
Northeastern University, University of British Columbia*

POPL 2020
New Orleans, LA, USA

$$\frac{\Gamma, x\!:\!\tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x\!:\!\tau_1.e : \Pi x\!:\!\tau_1.\tau_2}$$

## A two-part talk:

Create typed DSLs easily!

Prototype new type theories!

TURNSTILE+: A Framework for Implementing (Dependently) Typed Languages

CUR: An Extensible Proof Assistant (built with TURNSTILE+)

Explore new features, modularly!

Add a new tactic DSL!

# A Dependent Core Implementation with TURNSTILE+

```
(define-type Type : Type)
(define-type Π [#:bind Type] Type : Type)

(define-typerule (typed-λ [x : τ] e) »
  [⊢ τ » τ⁺ ⇐ Type]
  [x » x⁺ : τ⁺ ⊢ e » e⁺ ⇒ τ_out⁺]
  ------------------------------
  [⊢ (λ x⁺ e⁺) ⇒ (Π [x⁺ : τ⁺] τ_out⁺)])

(define-typerule (typed-app f e) »
  [⊢ f » f⁺ ⇒ (Π [X : τ] τ_out)]
  [⊢ e » e⁺ ⇐ τ]
  ------------------------------
  [⊢ (β f⁺ e⁺) ⇒ ($subst e⁺ X τ_out)])

(define-red β (app (λ x e) arg) ~>
              ($subst arg x e))
```

Implement complicated types concisely

## A Dependent Core Implementation with TURNSTILE+

```
(define-type Type : Type)
(define-type Π [#:bind Type] Type : Type)

(define-typerule (typed-λ [x : τ] e) »
  [⊢ τ » τ⁺ ⇐ Type]
  [x » x⁺ : τ⁺ ⊢ e » e⁺ ⇒ τ_out⁺]
  ------------------------------
  [⊢ (λ x⁺ e⁺) ⇒ (Π [x⁺ : τ⁺] τ_out⁺)])

(define-typerule (typed-app f e) »
  [⊢ f » f⁺ ⇒ (Π [X : τ] τ_out)]
  [⊢ e » e⁺ ⇐ τ]
  ------------------------------
  [⊢ (β f⁺ e⁺) ⇒ ($subst e⁺ X τ_out)])

(define-red β (app (λ x e) arg) ~>
              ($subst arg x e))
```

Implement complicated types concisely

Typecheck-and-elaborate inference rule syntax

# A Dependent Core Implementation with TURNSTILE+

```
(define-type Type : Type)
(define-type Π [#:bind Type] Type : Type)
```

Implement complicated types concisely

```
(define-typerule (typed-λ [x : τ] e) »
  [⊢ τ » τ⁺ ⇐ Type]
  [x » x⁺ : τ⁺ ⊢ e » e⁺ ⇒ τout⁺]
  ------------------------------
  [⊢ (λ x⁺ e⁺) ⇒ (Π [x⁺ : τ⁺] τout⁺)])

(define-typerule (typed-app f e) »
  [⊢ f » f⁺ ⇒ (Π [X : τ] τout)]
  [⊢ e » e⁺ ⇐ τ]
  ------------------------------
  [⊢ (β f⁺ e⁺) ⇒ ($subst e⁺ X τout)])

(define-red β (app (λ x e) arg) ~>
              ($subst arg x e))
```

Built-in support for:
- Binders (no deBruijn indices)
- Type contexts

Typecheck-and-elaborate inference rule syntax

Errors reported with surface syntax

# A Dependent Core Implementation with TURNSTILE+

```
(define-type Type : Type)
(define-type Π [#:bind Type] Type : Type)

(define-typerule (typed-λ [x : τ] e) »
  [⊢ τ » τ⁺ ⇐ Type]
  [x » x⁺ : τ⁺ ⊢ e » e⁺ ⇒ τout⁺]
  --------------------------------
  [⊢ (λ x⁺ e⁺) ⇒ (Π [x⁺ : τ⁺] τout⁺)])

(define-typerule (typed-app f e) »
  [⊢ f » f⁺ ⇒ (Π [X : τ] τout)]
  [⊢ e » e⁺ ⇐ τ]
  --------------------------------
  [⊢ (β f⁺ e⁺) ⇒ ($subst e⁺ X τout)])

(define-red β (app (λ x e) arg) ~>
              ($subst arg x e))
```

Implement complicated types concisely

Built-in support for:
- Binders (no deBruijn indices)
- Type contexts

Typecheck-and-elaborate inference rule syntax

Errors reported with surface syntax

Reduction-rule-style implementation of type-level computation

# A few final steps to make a language ...

```
(define-type Type : Type)
(define-type Π [#:bind Type] Type : Type)

(define-typerule (typed-λ [x : τ] e) »
  [⊦ τ » τ⁺ ⇐ Type]
  [x » x⁺ : τ⁺ ⊦ e » e⁺ ⇒ τ_out⁺]
  ------------------------------
  [⊦ (λ x⁺ e⁺) ⇒ (Π [x⁺ : τ⁺] τ_out⁺)])

(define-typerule (typed-app f e) »
  [⊦ f » f⁺ ⇒ (Π [X : τ] τ_out)]
  [⊦ e » e⁺ ⇐ τ]
  ------------------------------
  [⊦ (β f⁺ e⁺) ⇒ ($subst e⁺ X τ_out)])

(define-red β (app (λ x e) arg) ~>
              ($subst arg x e))
```

Language of this module's code

Module name

DEP

```
#lang TURNSTILE+
(provide Π Type
   (rename [typed-λ λ] [typed-app #%app])
(define-type Type : Type)
(define-type Π [#:bind Type] Type : Type)

(define-typerule (typed-λ [x : τ] e) »
  [⊢ τ » τ^+ ⇐ Type]
  [x » x^+ : τ^+ ⊢ e » e^+ ⇒ τ_out^+]
  ------------------------------
  [⊢ (λ x^+ e^+) ⇒ (Π [x^+ : τ^+] τ_out^+)])

(define-typerule (typed-app f e) »
  [⊢ f » f^+ ⇒ (Π [X : τ] τ_out)]
  [⊢ e » e^+ ⇐ τ]
  ------------------------------
  [⊢ (β f^+ e^+) ⇒ ($subst e^+ X τ_out)])

(define-red β (app (λ x e) arg) ~>
              ($subst arg x e))
```

Language of
this module's code

Exports define
a new language

DEP

Module name

… and language name

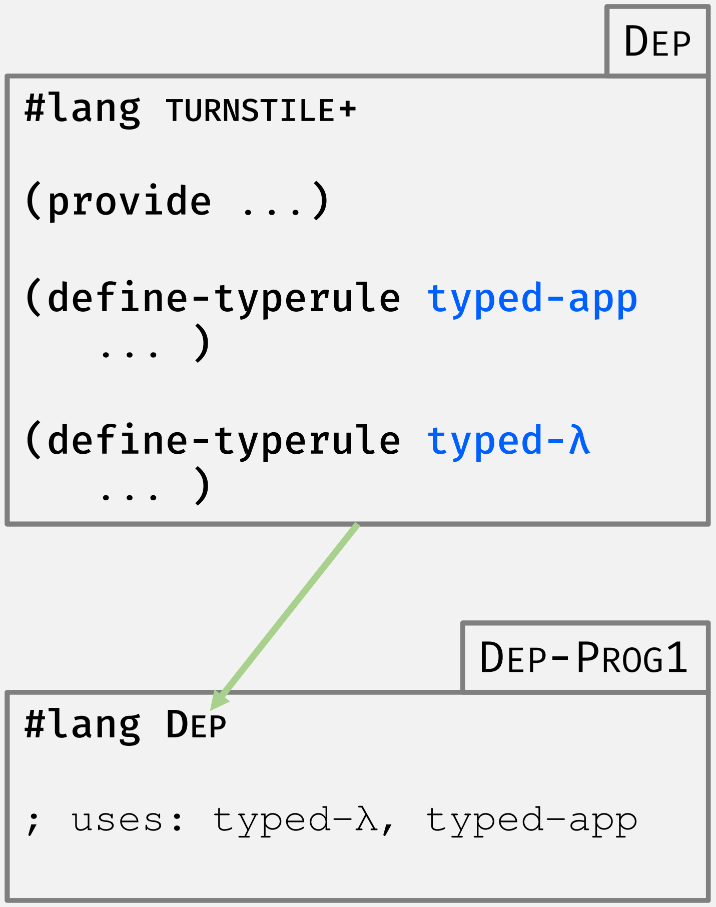
Implicit function app

```
#lang TURNSTILE+
(provide Π Type
   (rename [typed-λ λ] [typed-app #%app]))
(define-type Type : Type)
(define-type Π [#:bind Type] Type : Type)

(define-typerule (typed-λ [x : τ] e) »
  [⊢ τ » τ^+ ⇐ Type]
  [x » x^+ : τ^+ ⊢ e » e^+ ⇒ τ_out^+]
  ------------------------------
  [⊢ (λ x^+ e^+) ⇒ (Π [x^+ : τ^+] τ_out^+)])

(define-typerule (typed-app f e) »
  [⊢ f » f^+ ⇒ (Π [X : τ] τ_out)]
  [⊢ e » e^+ ⇐ τ]
  ------------------------------
  [⊢ (β f^+ e^+) ⇒ ($subst e^+ X τ_out)])

(define-red β (app (λ x e) arg) ~>
              ($subst arg x e))
```

DEP

```
#lang TURNSTILE+
(provide Π Type
   (rename [typed-λ λ] [typed-app #%app])
(define-type Type : Type)
(define-type Π [#:bind Type] Type : Type)

(define-typerule (typed-λ [x : τ] e) »
  [⊢ τ » τ⁺ ⇐ Type]
  [x » x⁺ : τ⁺ ⊢ e » e⁺ ⇒ τ_out⁺]
  ------------------------------
  [⊢ (λ x⁺ e⁺) ⇒ (Π [x⁺ : τ⁺] τ_out⁺)])

(define-typerule (typed-app f e) »
  [⊢ f » f⁺ ⇒ (Π [X : τ] τ_out)]
  [⊢ e » e⁺ ⇐ τ]
  ------------------------------
  [⊢ (β f⁺ e⁺) ⇒ ($subst e⁺ X τ_out)])

(define-red β (app (λ x e) arg) ~>
              ($subst arg x e))
```

## Using a TURNSTILE+ Created Language …


DEP

```
#lang TURNSTILE+

(provide ...)

(define-typerule typed-app
   ... )

(define-typerule typed-λ
   ... )
```

DEP-PROG1

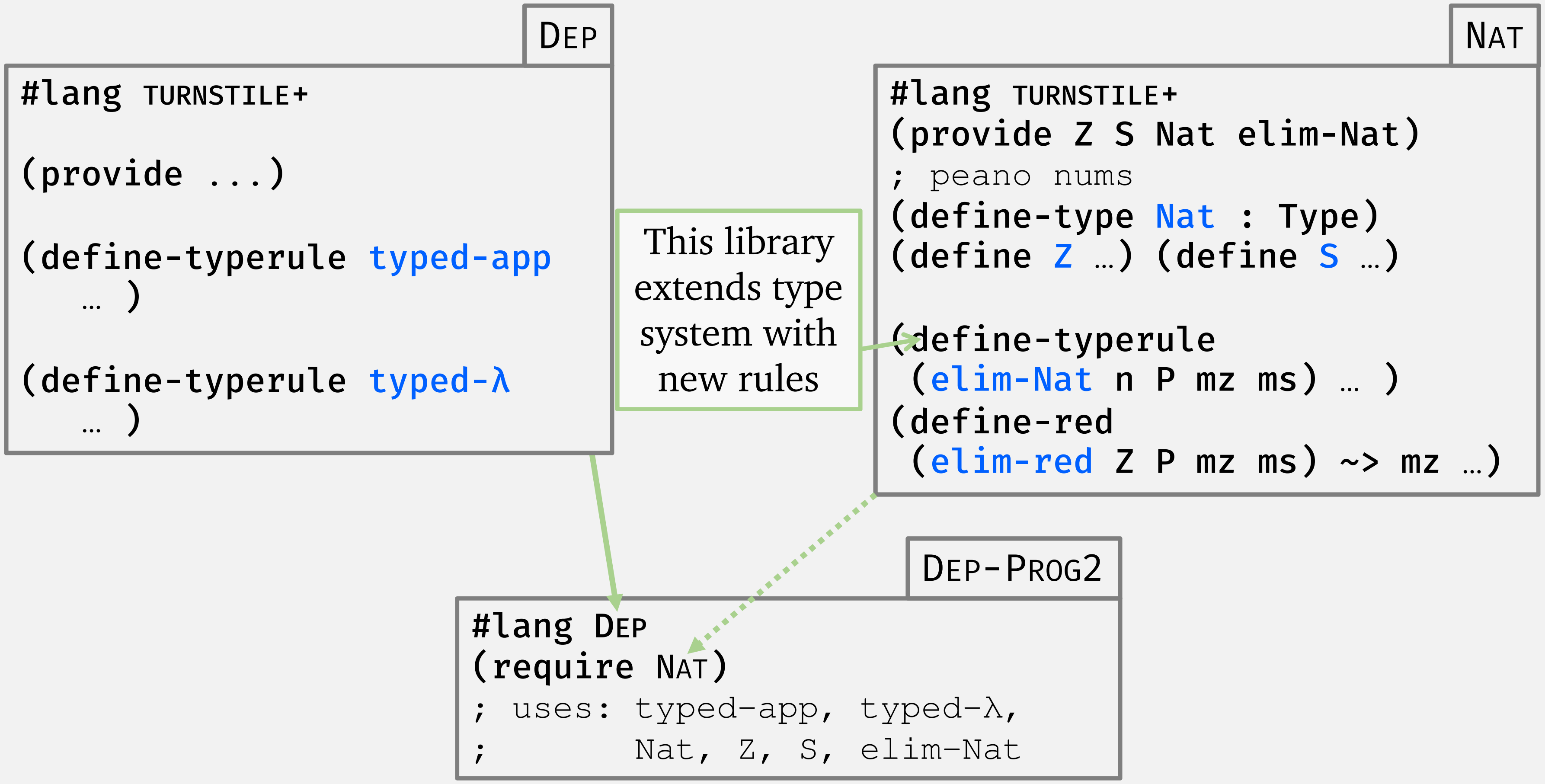
```
#lang DEP

; uses: typed-λ, typed-app
```

But wait! There's more!

## Do More with TURNSTILE+ Created Languages

- Extend: Add new type rules, modularly

# Extensible Languages: Type Rules as Libraries



DEP

```
#lang TURNSTILE+

(provide ...)

(define-typerule typed-app
   … )

(define-typerule typed-λ
   … )
```

NAT

```
#lang TURNSTILE+
(provide Z S Nat elim-Nat)
; peano nums
(define-type Nat : Type)
(define Z …) (define S …)

(define-typerule
 (elim-Nat n P mz ms) … )
(define-red
 (elim-red Z P mz ms) ~> mz …)
```

This library extends type system with new rules

DEP-PROG2

```
#lang DEP
(require NAT)
; uses: typed-app, typed-λ,
;       Nat, Z, S, elim-Nat
```

## Do More with TURNSTILE+ Created Languages

- Extend: Add new type rules, modularly

## Do More with TURNSTILE+ Created Languages

- Extend: Add new type rules, modularly
- Reuse: Type rules are linguistic constructs

# Modular, Reusable Languages

Reuse entire **DEP** lang to create *new* lang

DEP

```
#lang TURNSTILE+

(provide ...)

(define-typerule typed-app
   … )

(define-typerule typed-λ
   … )
```

DEP+IND

```
#lang TURNSTILE+
(require+provide DEP) ; reuse Dep rules
(provide def-datatype)
(define-typerule def-datatype
   (define-type …) ; inductive datatypes
   (define-typerule …)
   (define-red …) )
```

DEP-PROG3

```
#lang DEP+IND
; uses: typed-app, typed-λ, def-datatype
(def-datatype Nat : Type [Z : Nat] …)
; e.g., length-indexed list
(def-datatype Vec : [A : *] : [i : Nat] -> *
  [nil : (Vec A 0)]
  [cons [k : Nat] [x : A] [xs : (Vec A k)]
        : (Vec A (S k))])
```

## Do More with TURNSTILE+ Created Languages

- Extend: Add new type rules, modularly
- Reuse: Type rules are linguistic constructs

## Do More with TURNSTILE+ Created Languages

- Extend: Add new type rules, modularly
- Reuse: Type rules are linguistic constructs
- Interact: Created languages share a common substrate

## Under the TURNSTILE+ Hood: Macros!

```
(define-red β (app (λ x e) arg) ~> ($subst arg x e))
```

## Under the TURNSTILE+ Hood: Macros!

```
(define-red β (app (λ x e) arg) ~> ($subst arg x e))
```

Redex = macro input

desugars to

Contractum = macro output

```
(define-macro β
 [(app (λ x e) arg)
  (reflect ($subst arg x e))])
```

Macro system enables convenient syntax manipulations

"Normalization by Macro Expansion"

## Under the TURNSTILE+ Hood: Macros!

```
(define-red β (app (λ x e) arg) ~> ($subst arg x e))
```

Redex = macro input

desugars to

Contractum = macro output

```
(define-macro β
 [(app (λ x e) arg)
  (reflect ($subst arg x e))]
 [(placeholder . neutral-term)
  ((attach-red placeholder β) . neutral-term)])
```

Replaces placeholders with reduction macros

Macro system enables convenient syntax manipulations

Macro system enables tagging syntax with meta-information

Marks “placeholder” as potential β redex

“Normalization by Macro Expansion”

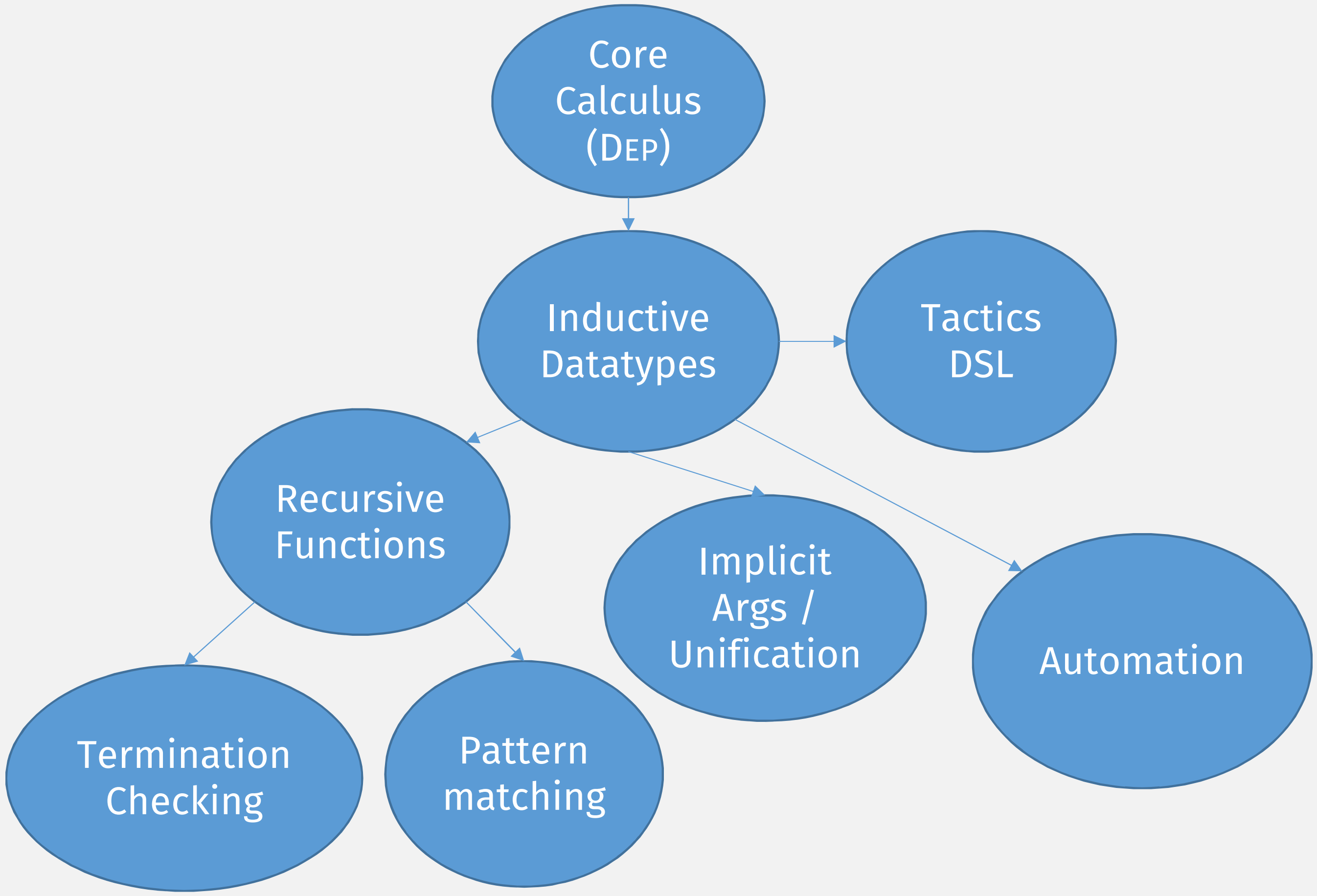
## Do More with TURNSTILE+ Created Languages

- Extend: Add new type rules, modularly
- Reuse: Type rules are linguistic constructs
- Interact: Created languages share a common substrate

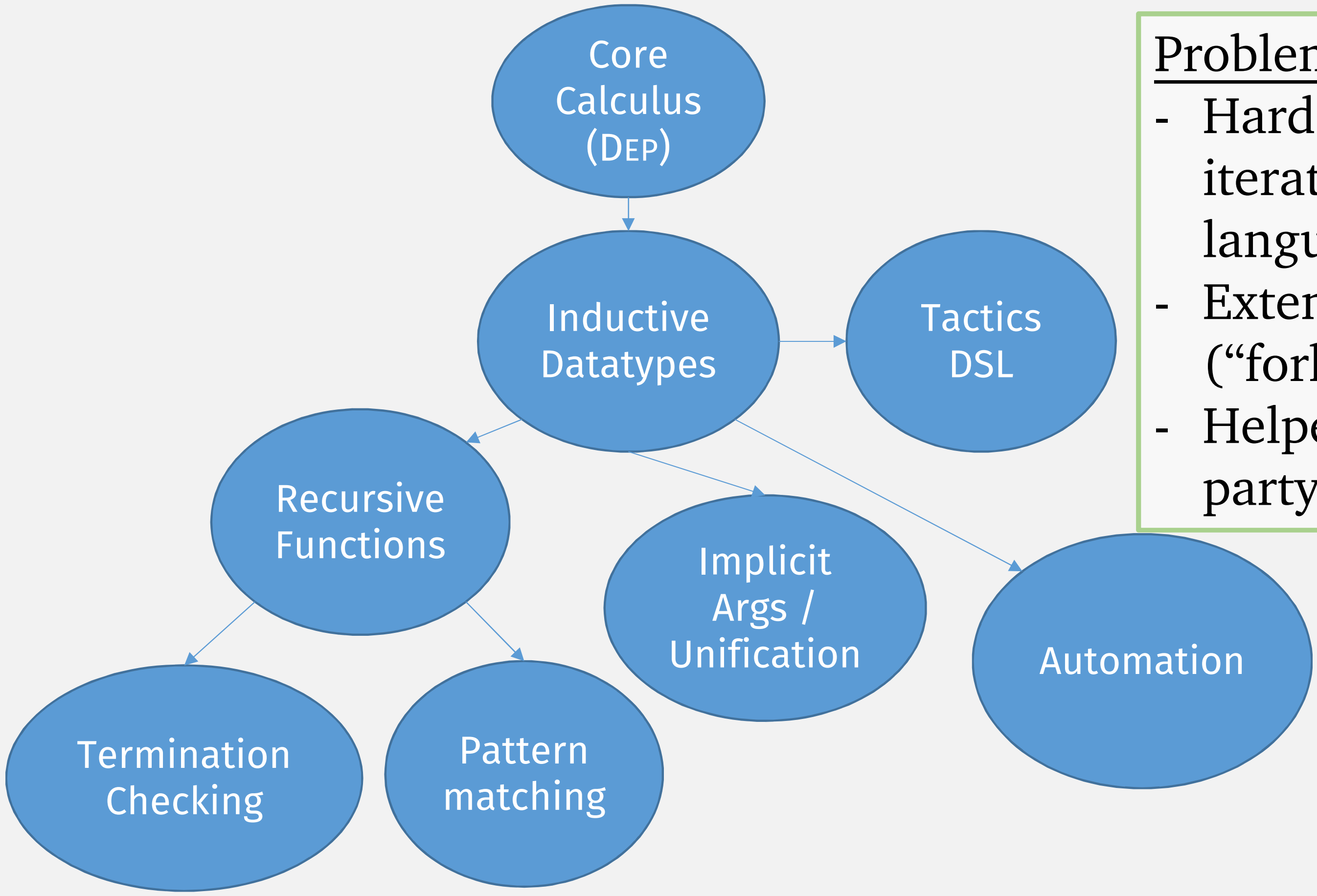
## Do More with TURNSTILE+ Created Languages

- Extend: Add new type rules, modularly
- Reuse: Type rules are linguistic constructs
- Interact: Languages share a **macro system toolbox**, enabling:
    - Transformation and convenient manipulation of syntax
    - Propagation of syntax meta-information
    - Overloading of features

# A Proof Assistant is:



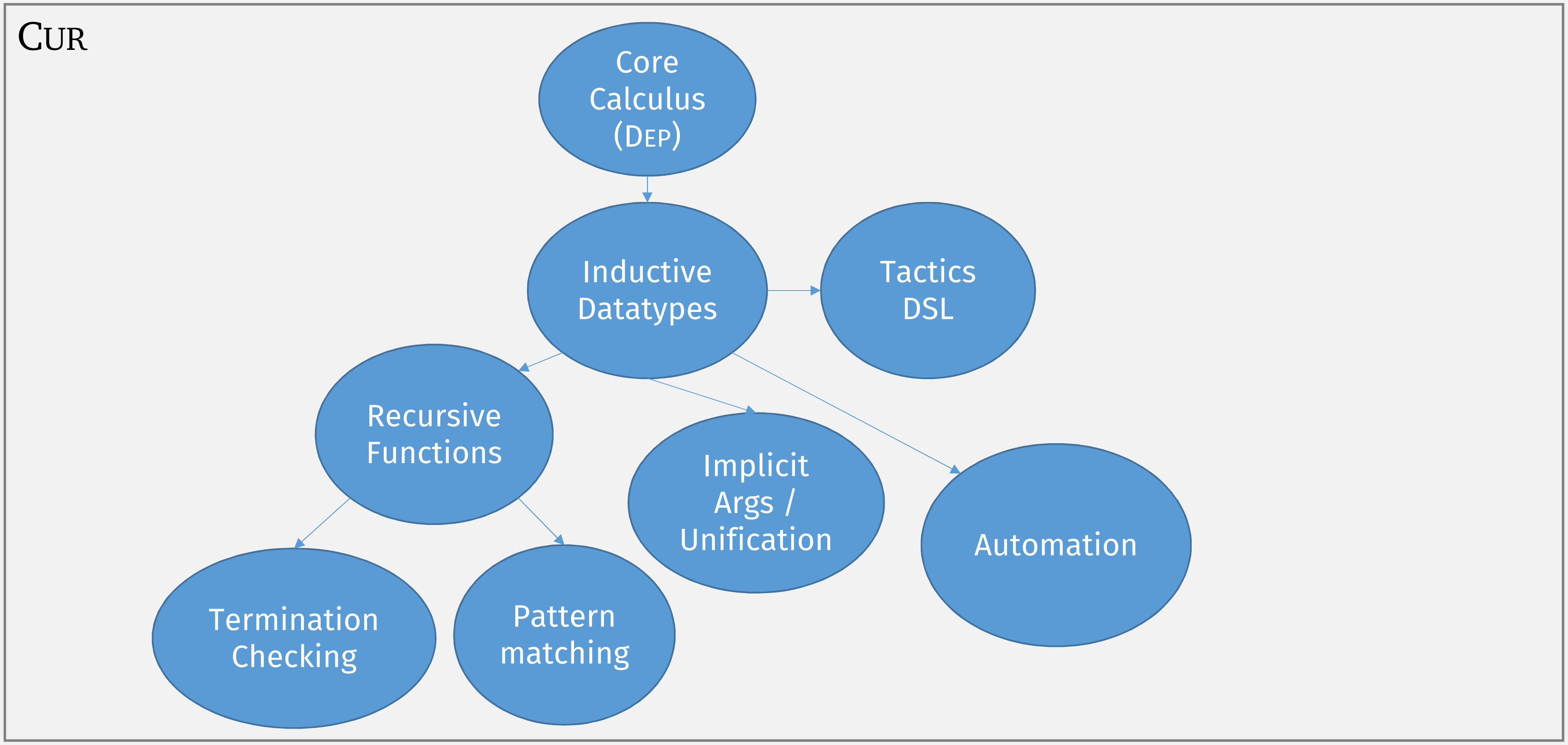
# A Proof Assistant is: a Collection of Interacting Extensions and DSLs



Problems:

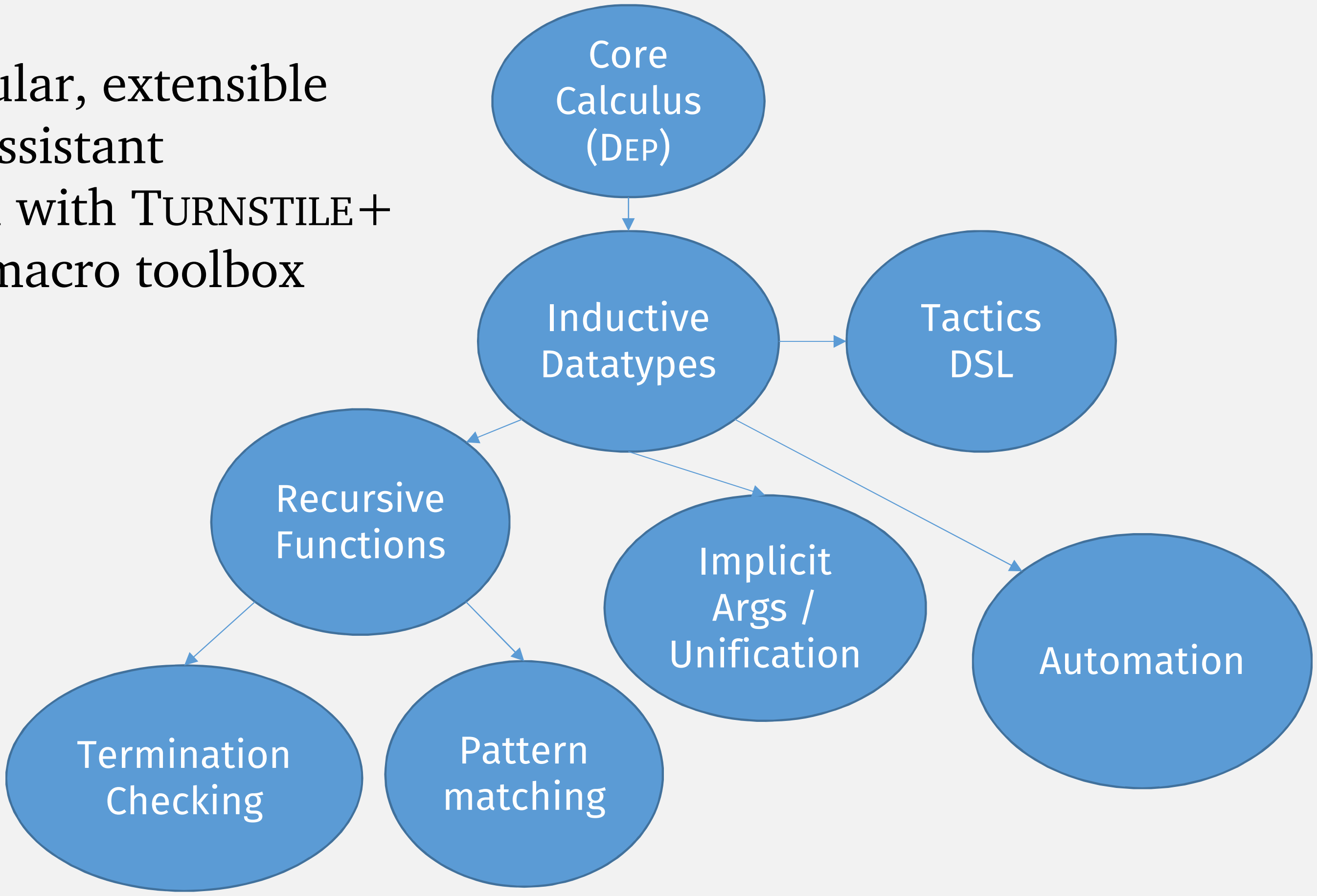
- Hard to experiment, iterate, and improve on language design
- Extensions not modular ("fork and modify")
- Helper DSLs are 3rd party tools

# The Cur Proof Assistant



# The Cur Proof Assistant

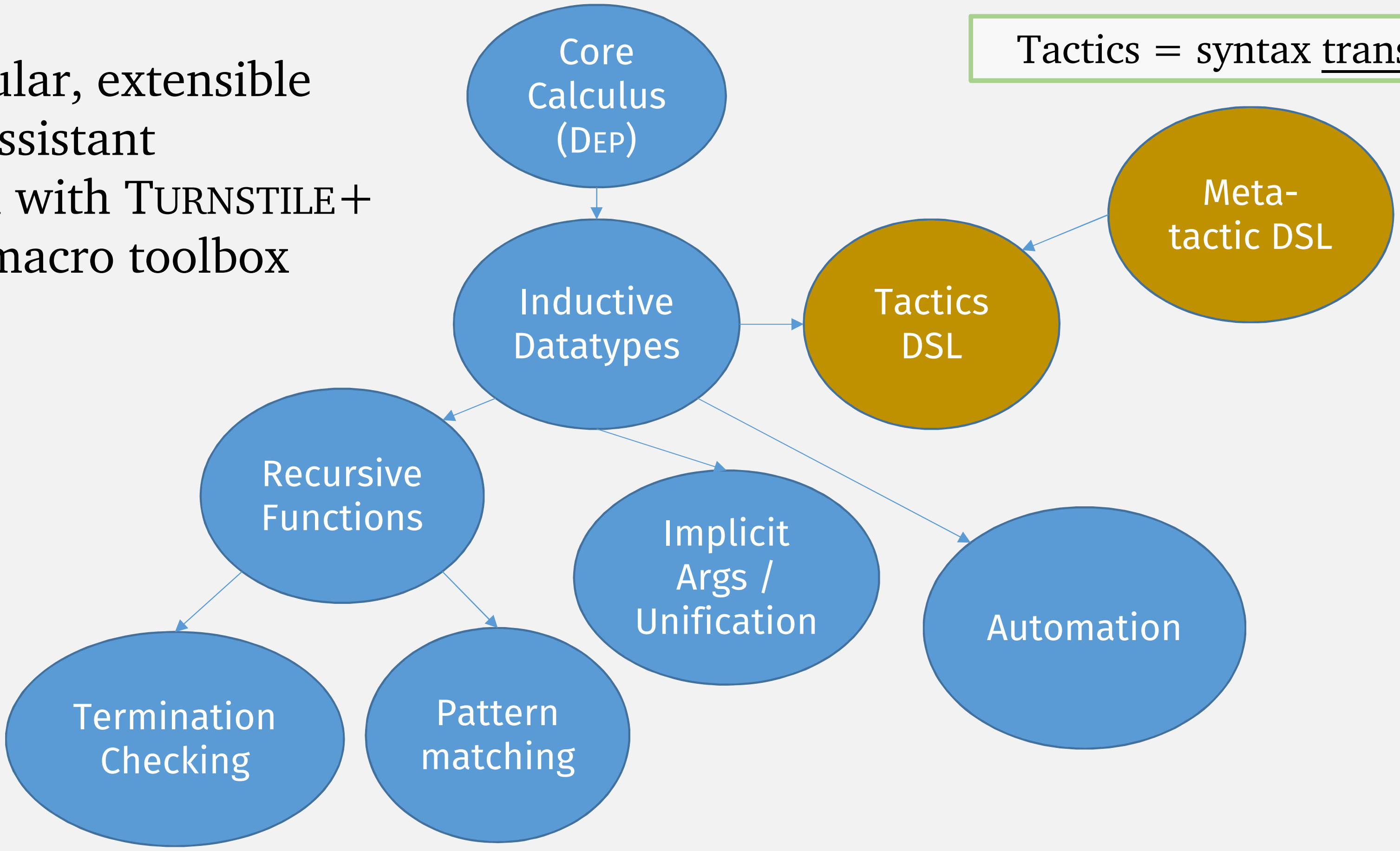
Cur:
A modular, extensible proof assistant created with Turnstile+ **and** a macro toolbox



# The CUR Proof Assistant

CUR:
A modular, extensible proof assistant created with TURNSTILE+ **and** a macro toolbox

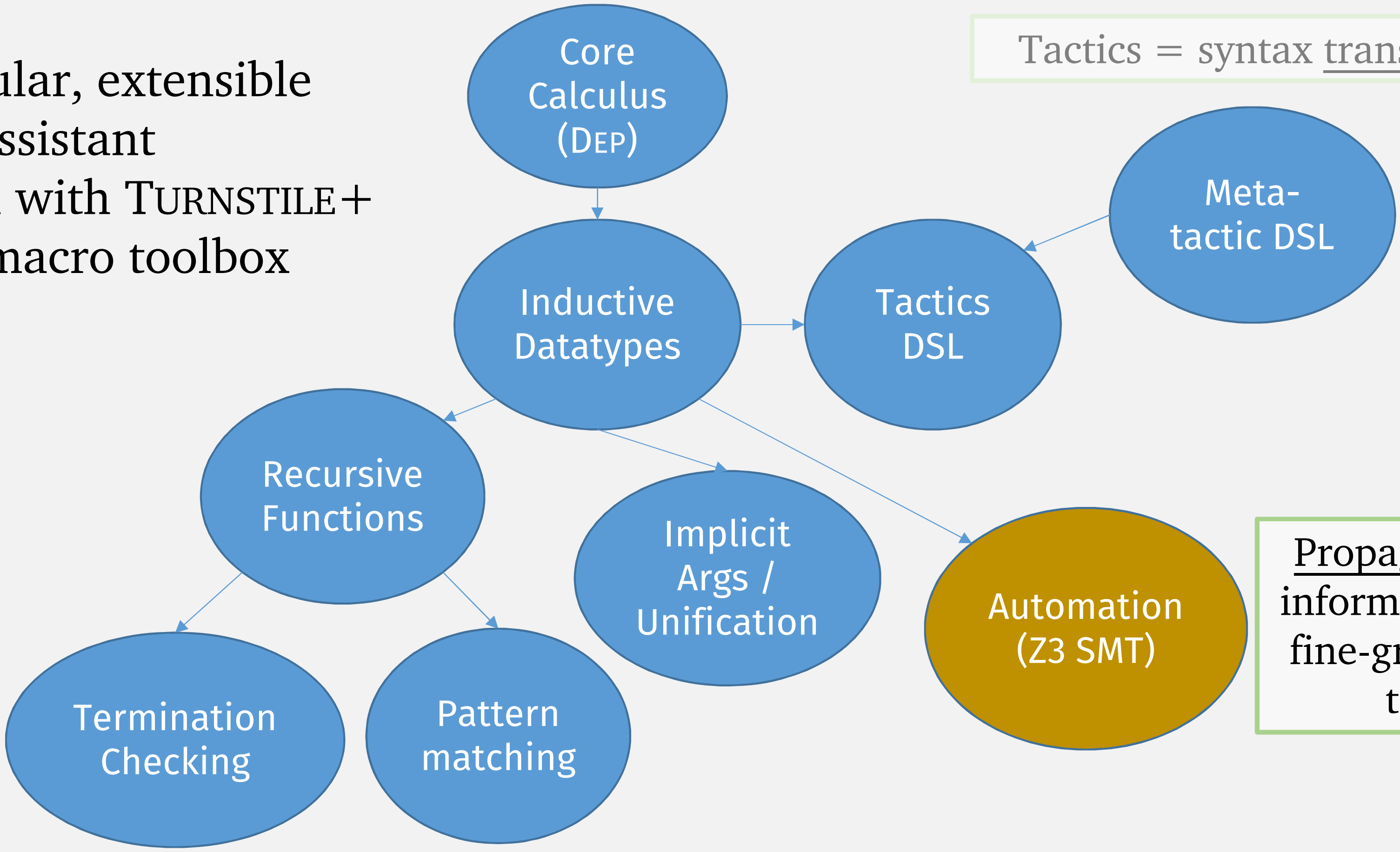
Tactics = syntax <u>transformers</u>



## The Cur Proof Assistant

Cur:
A modular, extensible proof assistant created with Turnstile+ **and** a macro toolbox

Tactics = syntax transformers



Propagating meta-information enables fine-grained axiom tracking

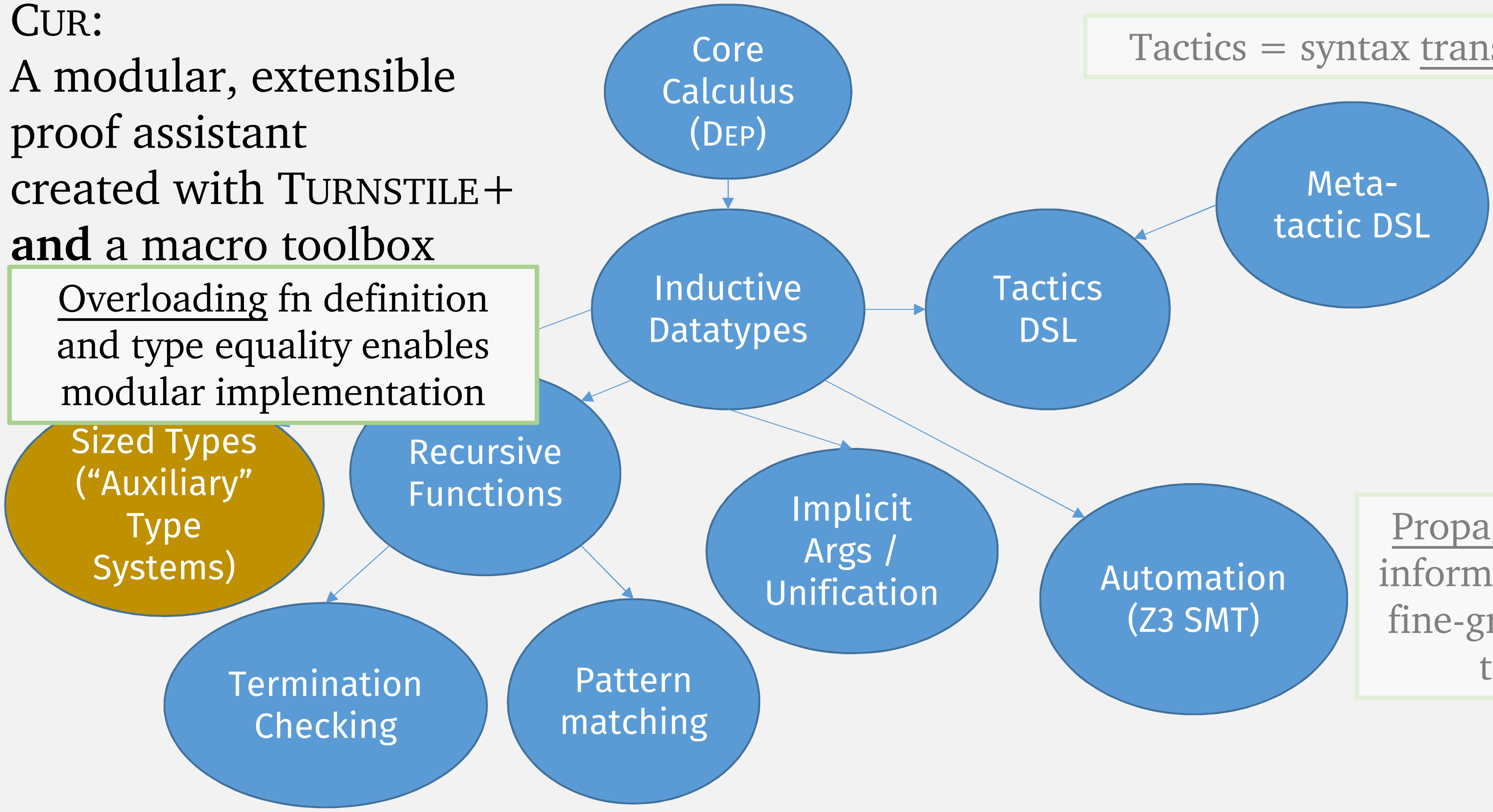
# The CUR Proof Assistant

CUR:
A modular, extensible proof assistant created with TURNSTILE+ **and** a macro toolbox

Overloading fn definition and type equality enables modular implementation

Tactics = syntax transformers

Propagating meta-information enables fine-grained axiom tracking



## Thank you!

Create typed DSLs easily!

Prototype new type theories!

extensible, reusable, and interacting

TURNSTILE+: A Framework for Implementing ^ (Dependently) Typed Languages

CUR: An Extensible Proof Assistant (built with TURNSTILE+ v)

*and* a macro toolbox

Add a new tactic DSL!

Explore new features, modularly!

https://github.com/stchang/macrotypes/tree/popl2020-artifact
https://github.com/stchang/cur/tree/popl2020-artifact
(requires Racket v7.5)