Nuprl-Light

Outline

- **Intro to Nuprl**
- **Intro to PL**
- **Formal Modules**
- **System Architecture**
- **Relation (PL & Formal Math)**

Nuprl

- **Formal mathematics (& programming)**
- **Foundational**
  - type theory
  - rules, semantics
- **Theories**
  - theorems
  - definitions
  - proofs-as-programs
- **Tactics**
  - Algorithms, heuristics for theorem proving

Example theory

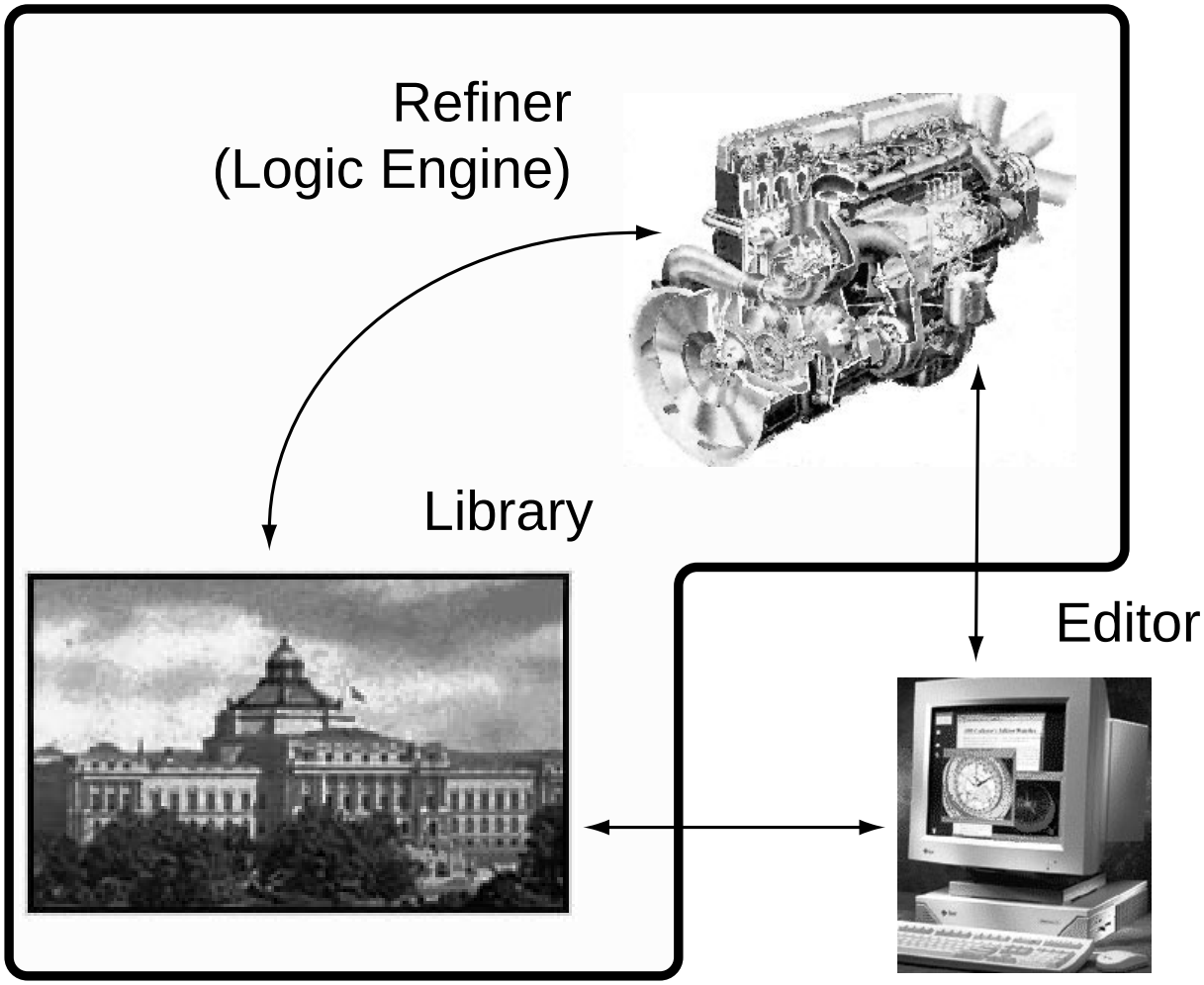
Engine

$\text{Piston}^{12} \times$
Crankshaft $\times$
$\ldots$

Plane

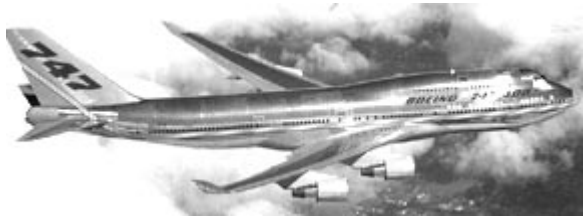
$\vdash \forall t\colon \mathbb{N}.\forall p\colon \mathit{Plane}.\forall o\colon \mathit{Ocean}.\mathit{pos}(p,t) \notin o$
BY `D 0 THENM NatInd`
$\vdash \forall p\colon \mathit{Plane}.\forall o\colon \mathit{Ocean}.\mathit{pos}(p,0) \notin o$
$t\colon \mathbf{N}^{+}; G(t-1) \vdash G(t)$

Overall Architecture



Problems

- **Want specialization**



- **Want modularity**
- **Want inheritance & subtyping**

$$\forall P\colon Plane \rightarrow \mathbf{Prop}.P \in B52 \rightarrow \mathbf{Prop}$$

- **Want formal theories (theories should have types)**

Programming Languages

- **Type systems (weak)**
- **Module systems**
  - Second class types
  - Functors (functions that compute modules)
- **Objects with:**
  - Subtyping
  - Inheritance
  - Reuse

Formal Modules

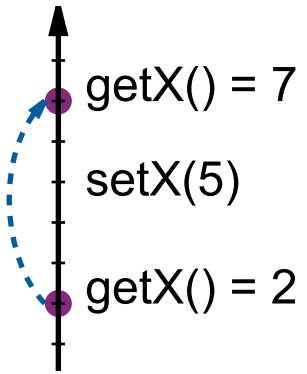
First class (no “functors”)

- **Bring formal system and language closer**
- **Signatures/declarations**
  - `val x : int    (* type *)`
  - Axiom/Rule - an assertion/type
- **Implementations**
  - `let x = 1      (* value *)`
  - Theorem/program
- **Rewrite - definition of computation**
- **Inclusion/inheritance**

Point Example

```
module Point =
begin
        include ITT
        axiom car:   Type
        axiom getX:  car → ℤ
        axiom bumpX:  car → ℤ → car
        axiom spec:   ∀i: ℤ.∀p: car.getX(setX(p, i)) = getX(p) + i
end

module ColorPoint =
begin
        include Point
        axiom Color:   Type
        axiom getC:  car → Color
        axiom setC:  car → Color → car
        axiom spec:   ∀c: Color.∀p: car.getC(setC(p, c)) = c
end
```



Point Implementation

theorem `PointClass` =
begin
`/* ITT */`
thm `car:` **Type** = $\{x{:}\,\mathbf{Z}\}$
thm `getX:` $car \to \mathbf{Z}$ = $\lambda r.r \cdot x$
thm `bumpX:` $car \to \mathbf{Z} \to car$ = $\lambda r.\lambda i.r \cdot x := (r \cdot x + i)$
thm `spec` = $\lambda i.\lambda p.Axiom$
end

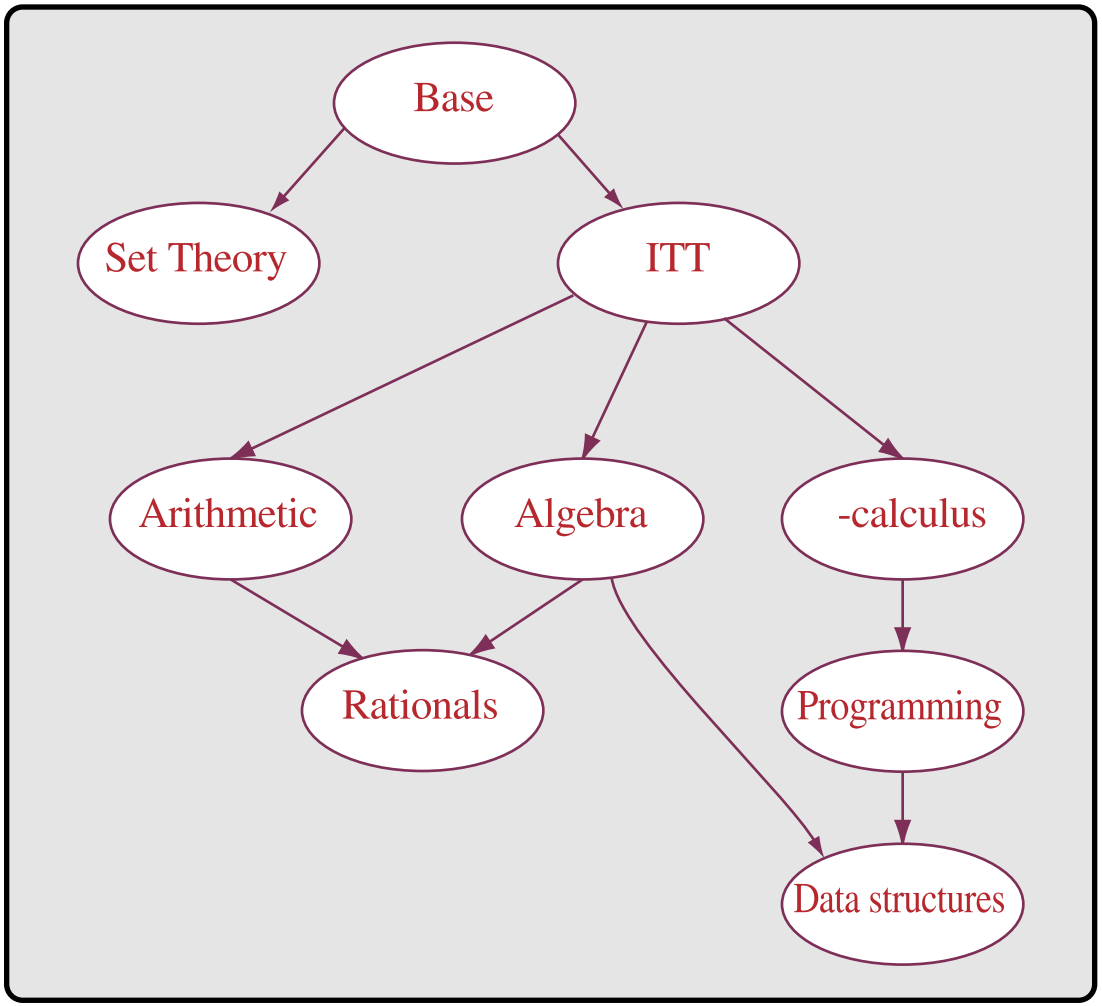
module `ColorPointClass` =
begin
$pointClass\,(car = \{x{:}\,\mathbf{Z}; c{:}\,Color\}){:}\,Point$
thm `Color:` **Type** = $\{red, green, blue\}$
thm `getC:` $car \to Color$ = $\lambda r.r \cdot c$
thm `setC:` $car \to Color \to car$ = $\lambda r.\lambda cc.r \cdot c := cc$
thm `spec` = $\lambda p.\lambda c.\cdots$
end

Algebra

$$
\begin{array}{l}
\textbf{theory}\ \ \mathit{LeftMonoid}\ \{\mathit{ITT}\} = \{ \\
\qquad \textbf{axiom}\ \ \mathit{car}\colon U_i \\
\qquad \textbf{axiom}\ \ \oplus\colon \mathit{car} \to \mathit{car} \to \mathit{car} \\
\qquad \textbf{axiom}\ \ \equiv\colon \mathit{car} \to \mathit{car} \to P_i \\
\qquad \textbf{axiom}\ \ 1\colon \mathit{car} \\
\qquad \textbf{axiom}\ \ \forall m\colon \mathit{car}.\ m \oplus 1 \equiv m \\
\qquad \ldots \\
\}
\end{array}
$$

$$
\begin{array}{l}
\textbf{theory}\ \ \mathit{Group}\ \{\mathit{LeftMonoid}\} = \{ \\
\qquad \textbf{axiom}\ \ \forall m\colon \mathit{car}.\ \exists n\colon \mathit{car}.\ m \oplus n \equiv 1 \\
\qquad \textbf{theorem}\ \forall m\colon \mathit{car}.\ \exists n\colon \mathit{car}.\ n \oplus m \equiv 1 \\
\}
\end{array}
$$

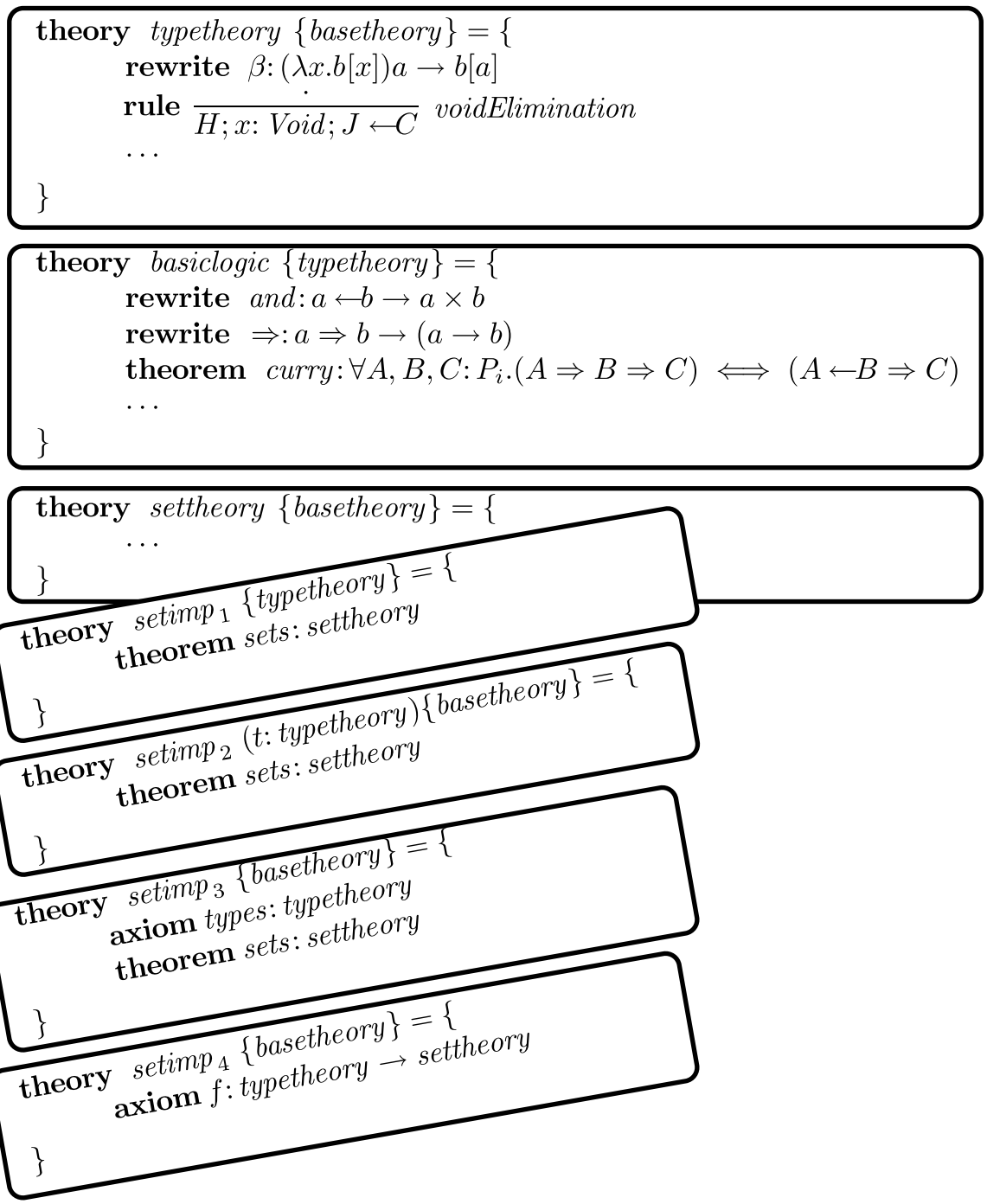
Logical Framework



Theory design

A “compilation unit” contains:

- **dependencies**
- **axioms/rules**
- **theorems/justifications/derived rules**
- **rewrite axioms/justifications/derived rewrites**
- **other theories as axioms/rules**
- **tactics**
- **comments: display forms, real comments, etc.**


theory typetheory {basetheory} = {
rewrite β: (λx.b[x])a → b[a]
rule H; x: Void; J ←C voidElimination
...
}
theory basiclogic {typetheory} = {
rewrite and: a ←b → a × b
rewrite ⇒: a ⇒ b → (a → b)
theorem curry: ∀A, B, C: P_i.(A ⇒ B ⇒ C) ⟺ (A ←B ⇒ C)
...
}
theory settheory {basetheory} = {
...
}
theory setimp 1 {typetheory} = {
theorem sets: settheory
}
theory setimp 2 (t: typetheory){basetheory} = {
theorem sets: settheory
}
theory setimp 3 {basetheory} = {
axiom types: typetheory
theorem sets: settheory
}
theory setimp 4 {basetheory} = {
axiom f: typetheory → settheory
}

Features of the module system

- **Multiple inheritance**
- **Multiple display form collections**
- **Automatic tactic generation from rules**
- **Tactic joining for “D,” “EqCD,” “Auto,” etc.**
- **Automatic generation of signatures**
- **Theorems as “default” values**

```
(* Integers *)
include Itt_equal;;
include Itt_rfun;;
include Itt_logic;;
```

Dependencies

```
declare int;;
declare natural_number[$n:n];;
...

val int_term : term;;
val zero : term;;

declare "add"{'a; 'b};;
...
```

Declarations

```
rewrite reduceAdd : "add"{natural_number[$i:n]; natural_number[$j:n]} <-->
   natural_number[$i + $j];;
...

rewrite indReduceDown :
   'x < 0 -->
   ((ind{'x; i, j. 'down['i; 'j]; 'base; k, l. 'up['k; 'l]}) <-->
    'down['x; ind{'x add 1; i, j. 'down['i; 'j]; 'base; k, l. 'up['k; 'l]}]);;
```

Rewrites

```
(*
 * Induction:
 * H, n:Z, J[n] >> C[n] ext ind(i; m, z. down[n, m, it, z]; base[n]; m, z. up[n, m, it, z])
 * by intElimination [m; v; z]
 *
 * H, n:Z, J[n], m:Z, v: m < 0, z: C[m + 1] >> C[m] ext down[n, m, v, z]
 * H, n:Z, J[n] >> C[0] ext base[n]
 * H, n:Z, J[n], m:Z, v: 0 < m, z: C[m - 1] >> C[m] ext up[n, m, v, z]
 *)
axiom intElimination 'H 'J 'n 'm 'v 'z :
   sequent { 'H; n. int; 'J['n]; m. int; v. 'm < 0; z. 'C['m add 1] >> 'C['m] } -->
   sequent { 'H; n. int; 'J['n] >> 'C[0] } -->
   sequent { 'H; n. int; 'J['n]; m. int; v. 0 < 'm; z. 'C['m sub 1] >> 'C['m] } -->
   sequent { 'H; n. int; 'J['n] >> 'C['n] };;
```

Rules

Primitives

```
axiom <name> : <term> [--> <term>]*
rewrite <name> : [<term> -->]* <term> <--> <term>
declare <term>
define <name> : <term> <--> <term>
include <name>
thm <name> = <tactic>
mlterm <term> [= begin <code> end]
dform <term> = <def>
prec <name> [> <name>]
```

Implementation

- **Refiner is implemented in Caml-Light**
- **“Refiner” is unit of truth**
  - Rule set
  - Tactic/rewrite validation
  - Proof validation
  - Everything is a rewrite
- **Library on filesystem**
  - Modular, object storage
- **Editor in emacs**

Editor

Future goals

Bring Caml and Nuprl closer

- **Ability to extract ML code from theorems**
- **Apply proof techniques to code generation**
- **Assertions**
- **Caml semantics library**
- **Nuprl programming library**