Nuprl-Light

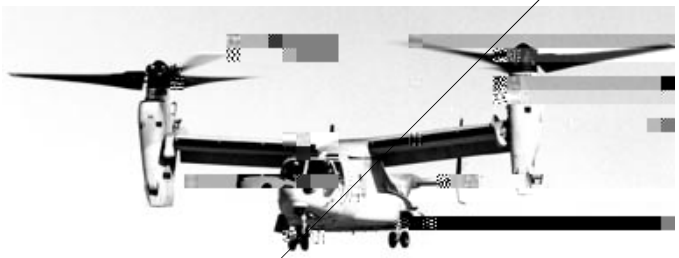
Outline

- **Intro to Nuprl**

Example theory

Engine

Plane



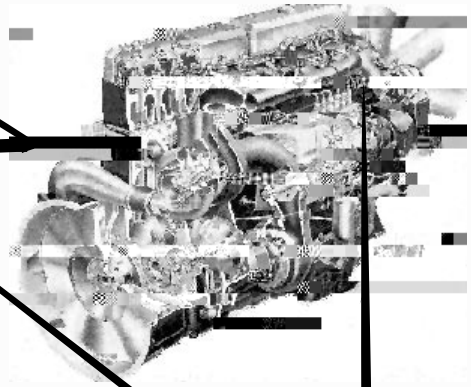
$\vdash \forall b{:}\mathbb{N}.\forall p{:}\mathit{Plane}.\forall$

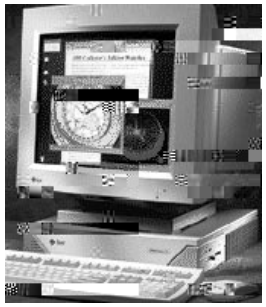
Overall Architecture

Library

Refiner

Logic Engine)





Problems

- **Want specialization**
-

Programming Languages

- **Type systems (weak)**

  Module systems

  - ute modules

4225 **Objects with:**

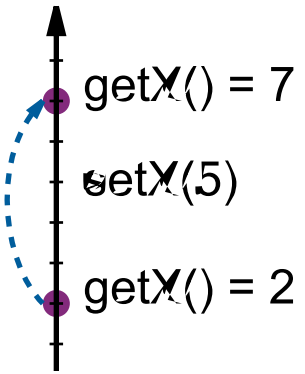
  - Subtyping
  - Inheritance
  - Reuse

Formal Modules

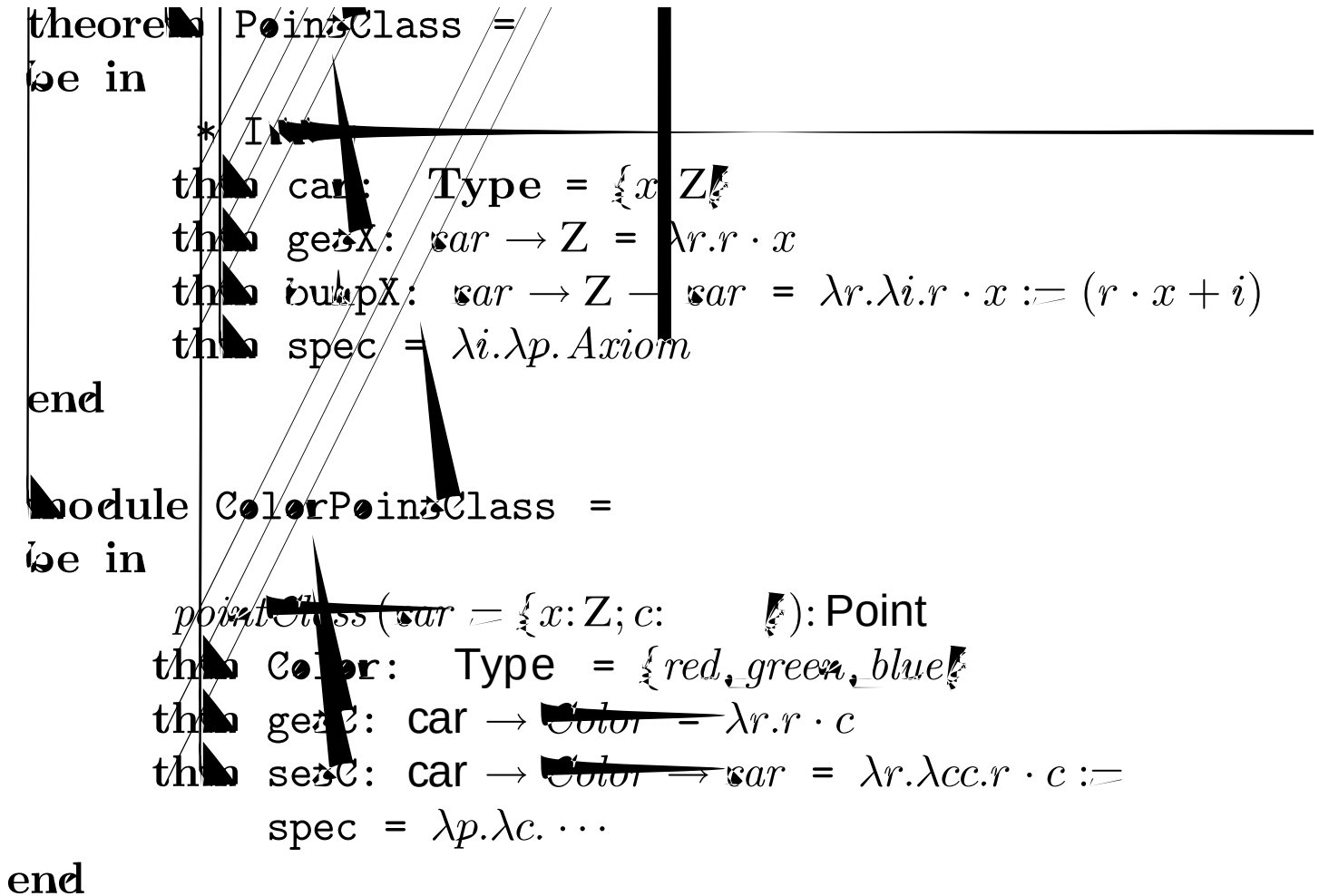
First class (

Point Example

```
module Point =
begin
        include
```



Point Implementation



Algebra

theory *LeftMonoid*

sort *car*

axiom $\oplus : car \rightarrow car$

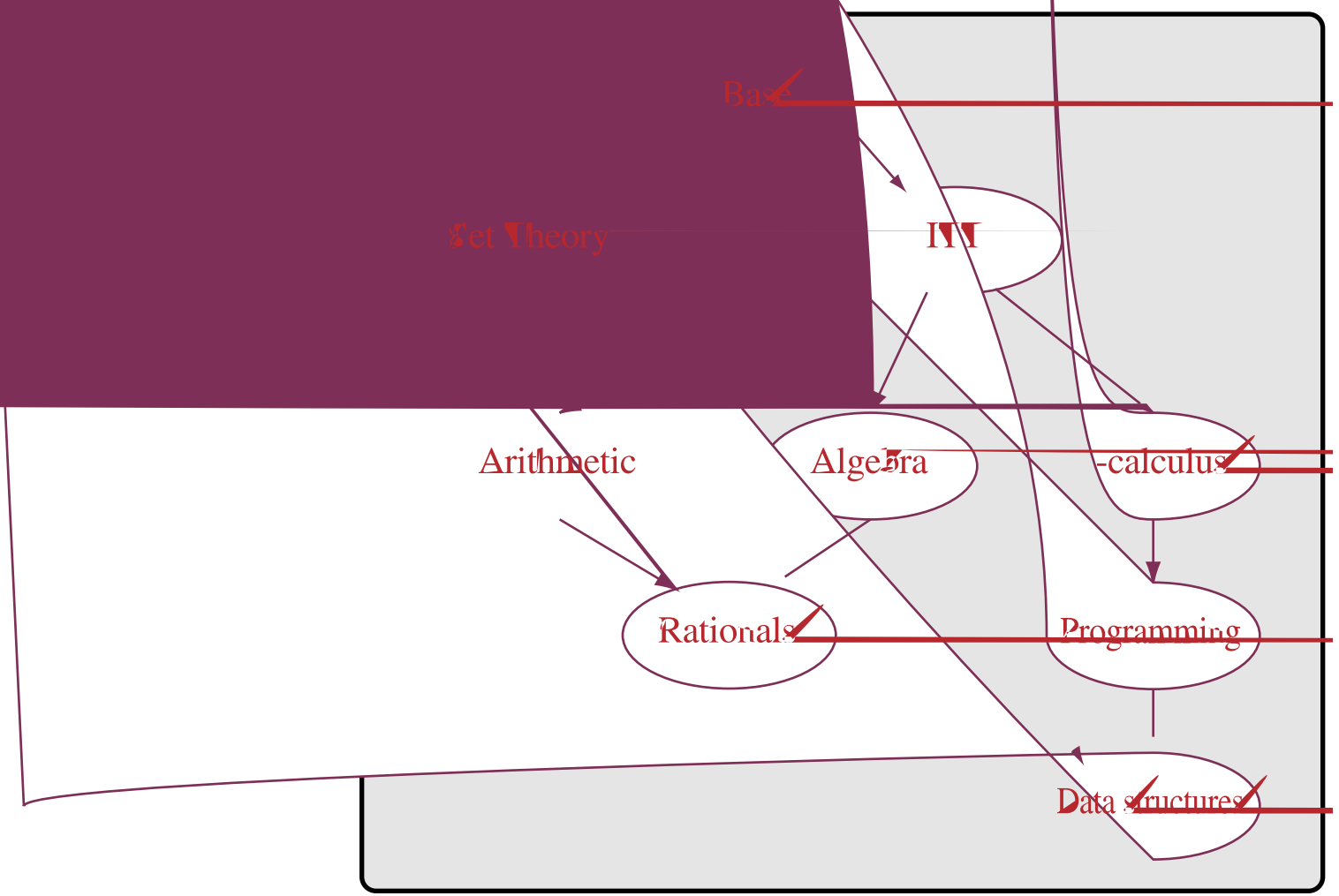
axiom $1 : car$

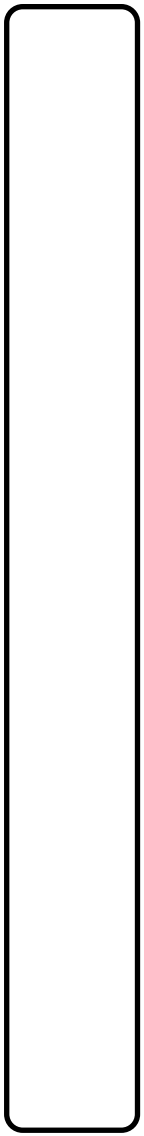
axiom $\forall m : car . m \oplus 1 \equiv m$

...

theory *Group* = *LeftMonoid*

Logical Framework





Features of the module system

- **Multiple inheritance**
- **Multiple display form collections**
- **Automatic tactic generation from rules**
- **Tactic joining for “D,” “Eq**
- **Automatic generation of**

Dependencies

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

Declarations

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

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

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

Rewrites

```
rewrite reduceAss "ass"{natural_number[$i n]; natural_number[$j n]} <-->
   natural_number[$i + $j];;
...

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

Rules

```
(*
 * 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 ass 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] };;
```

Primitives


axiom <name> <term>
rewrite <name>
declare <term>
define <name> <term> <--> <term>

Implementation

- **Refiner is implemented in Caml-Light**
- **“Refiner” is unit of truth**
  - Rule set
  - Tactic/rewrite validation
  - Proof validation

    Everythin

Editor

Future goals

Bring Caml and Nu