Formal Objects in Type Theory

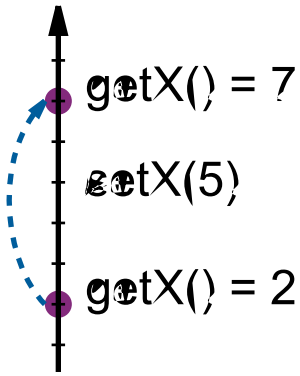
Jason Hickey

Outline

- **Object Oriented Programming**
- **Object Calculi**
- **Examples**
- **Interpretation in Nuprl**

Object Oriented Programming

Example class




```
class Point {
private
    int x;
public
    Point(int
    int getX
    Point setX(int i) {
        return new Point(i);
    }
};
```

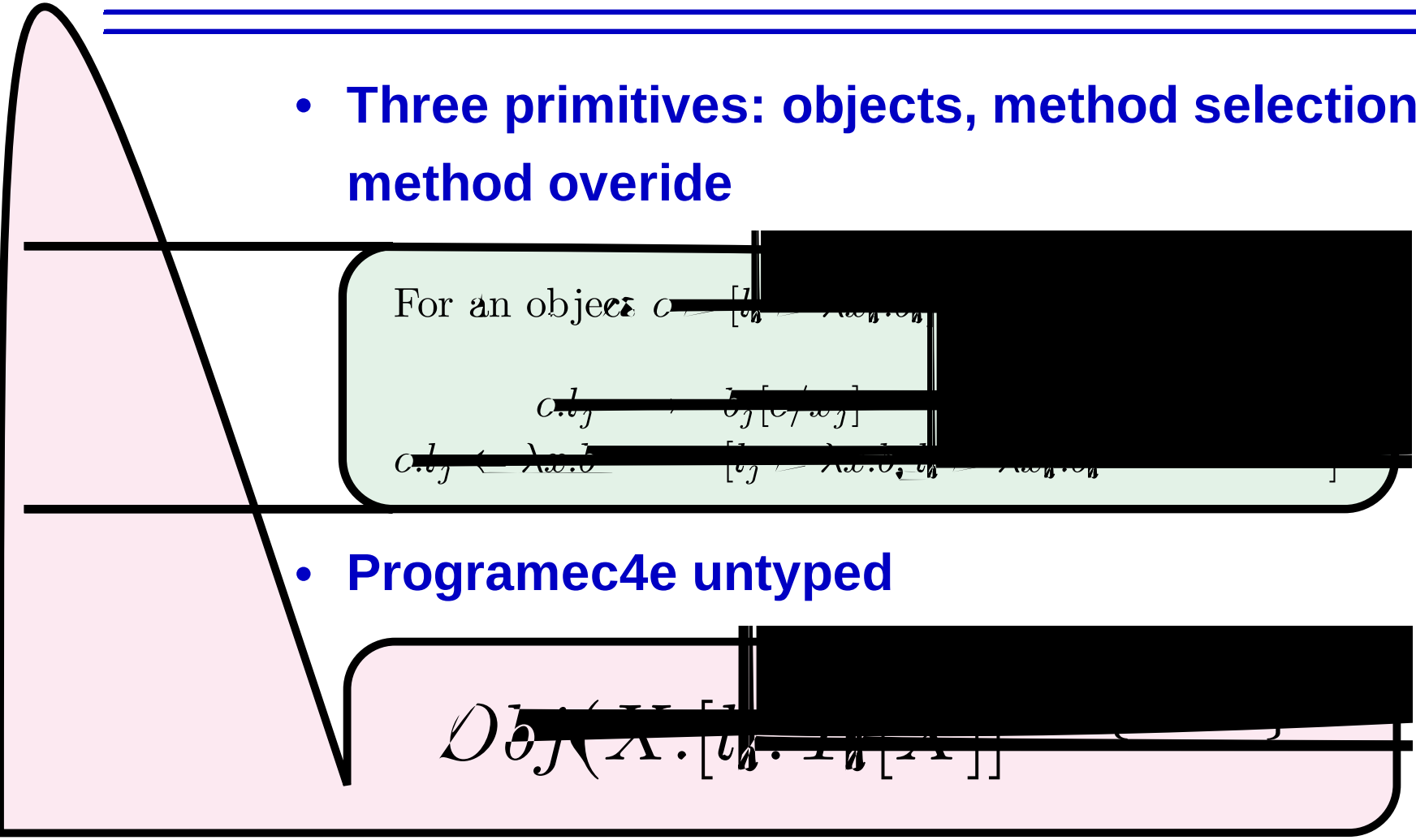
Inheritance

What type is setX()?

```
class ColorPoint : inherits Point {
priva e:
    Color color;
public:
    ColorPoint(
    in  getColor(
    ColorPoint setColor(Color c) {
        re urn new
```

Object calculus (Cardelli, Abadi)

- **Three primitives: objects, method selection, method overide**



- **Programec4e untyped**

...t example

$$p_5 = [\quad getx = \lambda c.5;$$
$$\quad setx = \lambda c.\lambda i.\ c.getx \Leftarrow \lambda c.i\ ]$$

$$p_5 = [\quad x = ;5$$
$$\quad getx =$$
$$\quad = \qquad \Leftarrow$$

Point $= Obj$

λ-calculus

$[\![\lambda x.b]\!] = [\ \ = \varsigma x.x; \text{body} = \varsigma x.[\![b]\!]]$

$[\![f\ a$

$[\![$

$[\![(\quad + x)\ 1]\!] =$

$([\text{arg} = \quad; \text{body} = \quad \varsigma x.x.\text{arg} + x: arg \quad 1)\ body$

$[\text{arg} = 1; \text{body} = \ \&\quad \text{arg} + x: \text{arg}]: \text{body}$

$[arg = 1; \text{body} = \ \varsigma x.x.arg \quad \text{arg} = \cdots]: arg$

$$c \equiv [\ new = \varsigma z.\ [get = \varsigma s.\ z.get(s); set = \varsigma s.\ z.set(s)];\ get\varsigma s:$$

Interpretations

- **Primitive, axiomatic (Cardelli, Abadi)**
  - Fuly functional
  - Syntactic
  - F ω, < μ
- **Existential (Hoffman, Pierce, Trner)**
  - Updates onields ony
  - Type-speci interpretation
  - F ω, <:

Type Theory

- **Advantages**
  - Many models:
    - Set theoretic [Howe]
    - PER [Allen, Mendler]
    - Denotational [Rezus]
    - Recursive Realizibility [Aczel]
    - Categorical [Palmgren]
    - •••
  - Predicativity
  - Large, expressive, formal foundation

  - Predicativity

Formal Objects in Type Theory

- **Scale in formal systems**
  - Jackson, large formalization of abstract algebra

Results

- **One new type constructor for defining object types**
  - Formal interpretation of objects
- **New theory structure for verification systems**
  - Correspondences (propositions-as-types)

Object Types

- **Existential interpretation [Hoffman, Pierce, Turner]**

  “state-application” semantics

  - complex interpretation

  Canonical

Weak existential

may

____ have different

Objects

- **Object types specified with method descriptions**

$$Point\ M \triangleq \lambda Rep.\{get_x : Rep \to \mathbb{Z}, set_x : Rep \to \mathbb{Z} \to Rep\}$$

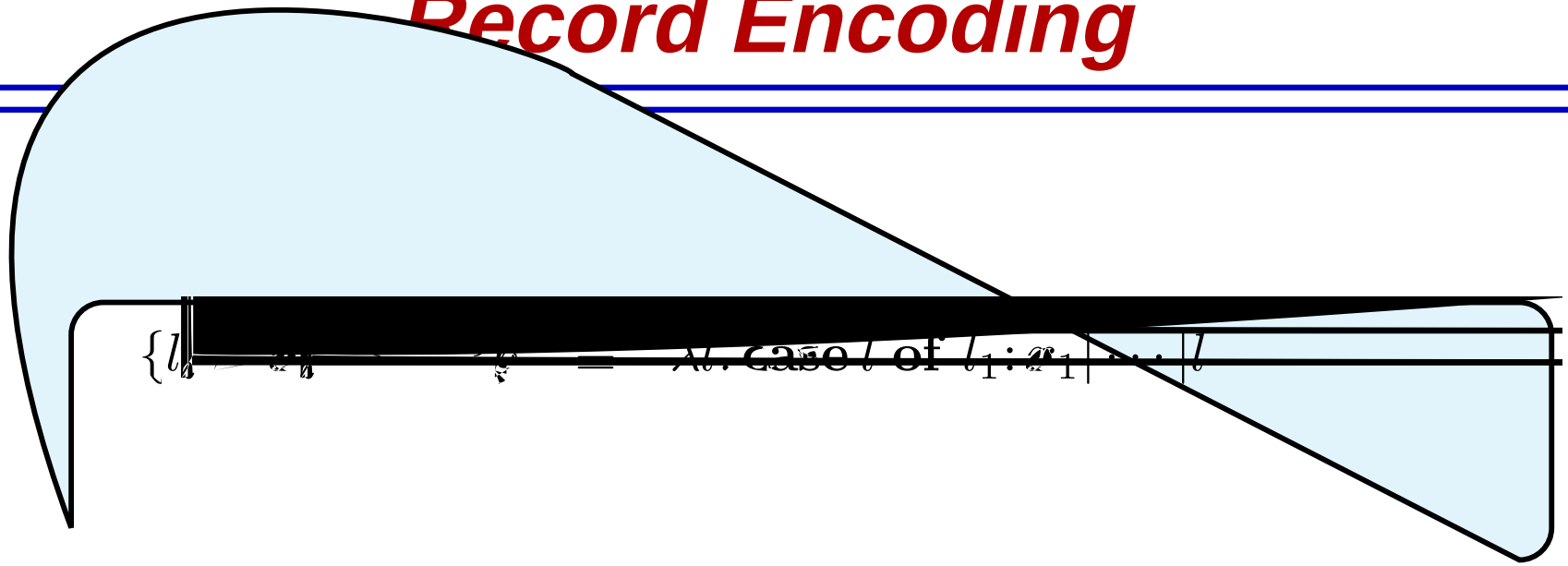
- **Generic type constructor**

$$\{\ldots, methods: M(Rep)\}$$

- **Point example:**

$$\ldots, set_x : Rep \to \mathbb{Z} \to Rep\}\}$$

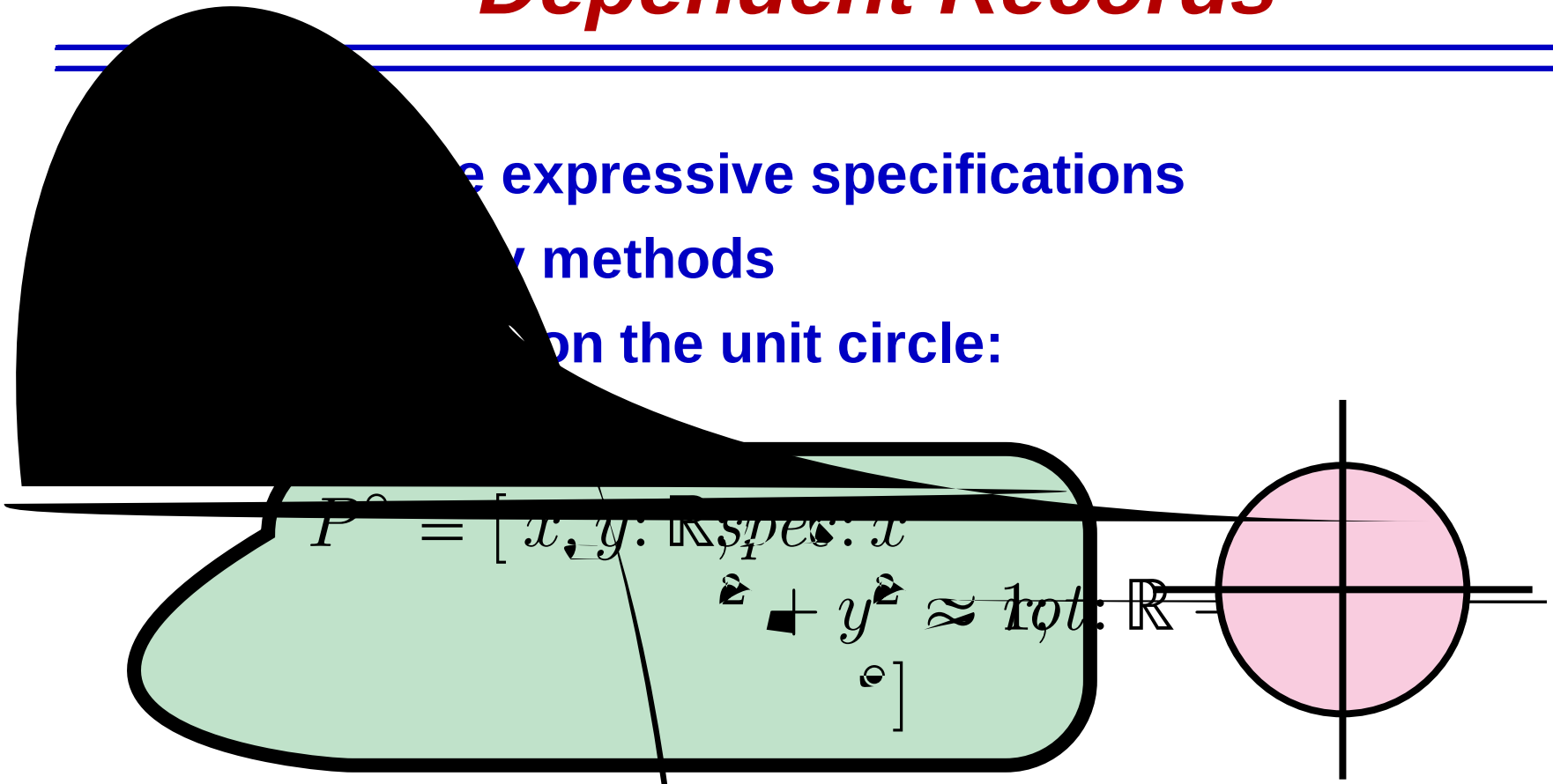
Record Encoding



- **Type: dependent function type**

Dependent Records

- e expressive specifications
- y methods
- on the unit circle:



Very-dependent function encoding

- **Dependent function** $a\colon A \rightharpoonup B_a$ $\quad (\Pi a\colon A.B_a)$
- **Very-dependent function allows calls in the range:**

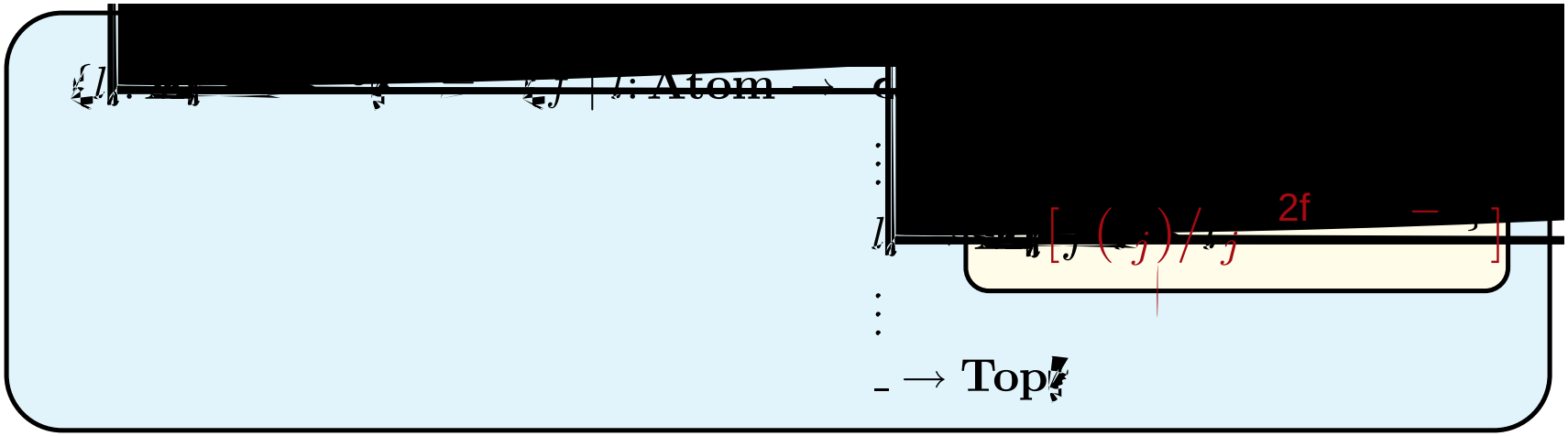
$$\{f \mid a\colon A \rightharpoonup B_{f,a}\}$$

- **Circle points:**

$$\{f \mid l\colon \mathbf{Atom} \rightharpoonup \begin{array}{l} \mathbf{case}\ l\ \mathbf{of} \\ \text{“}get X\text{”} \rightharpoonup \mathbb{R} \\ \text{“}get Y\text{”} \rightharpoonup \mathbb{R} \\ \\ \text{“}rot\text{”} \rightharpoonup \cdots \\ \_ \rightharpoonup \mathbf{Top}\} \end{array}$$

Dependent Record Encoding

- **Map function *f* over occurrences of labels**



Point object

- **Point is untyped**

$$0 \equiv \texttt{pack}\{\, state = 0;$$

Interfaces

- **unpack object, call method, repack object**

Point`setX Υ**01jı´0 24.24 5.101 356.508 302 Tmı´0.369 0 0**

$$\lambda p, x.\ \mathtt{open}\ p\ \mathtt{in}$$

$$r\quad state := (r \cdot methods \cdot$$

Subsumption

- **Untyped interfaces are polymorphic**

- **Interface typing**

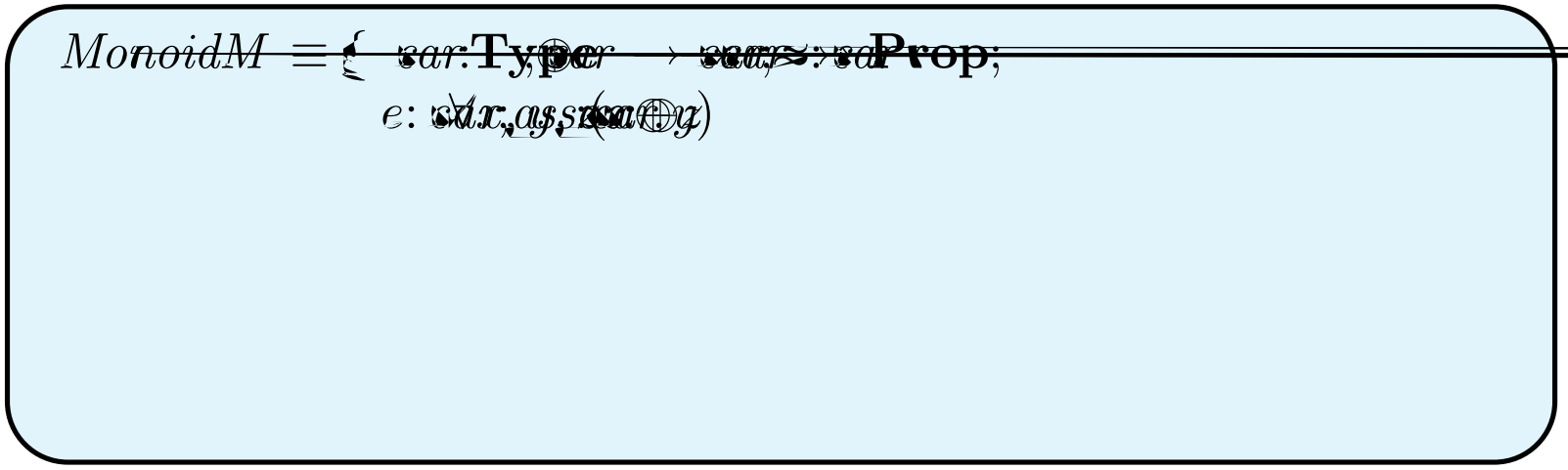
Propositions-as-types

- **Proof of an object type is an object**
- **Theory construction:**
  - state axioms
  - derive theorems
- **Relationships of theories are formalizable**
- **Add rules to object types**

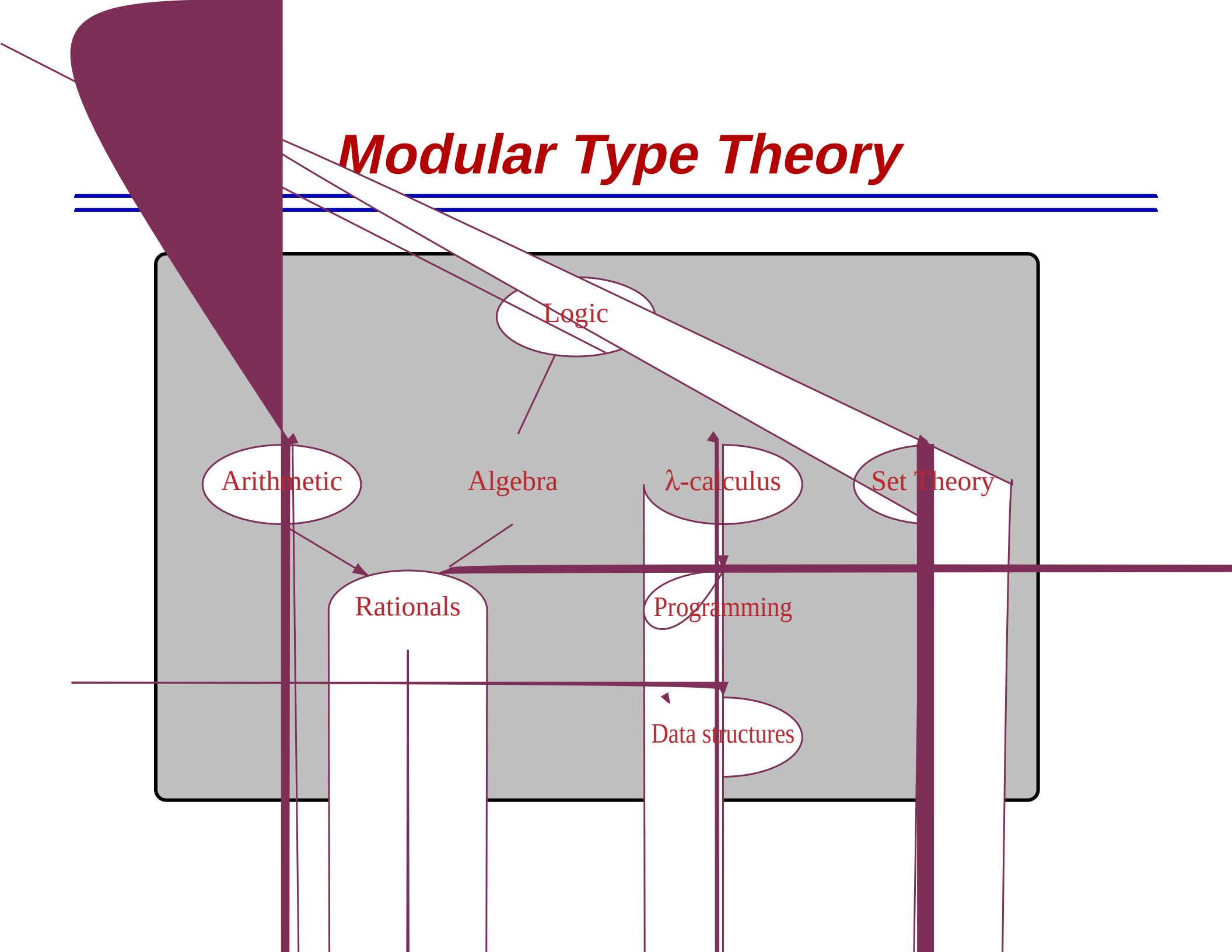
II. Theories as Objects

- **Expressive, dependent, object signatures**
- **Binary operators**

  Expected subtyping properties GroupM 9



Modular Type Theory



Conclusion

- **Type theoretic interpretation provides mathematical understanding**
- **New contributions:**
  - Dependent object types
  - New form of “binary” methods
- **Feed prmcmples of object-ormented programing back into formal systems**
  - Modularity (multiple, possmbly inconsistj•(t, dom)-22es)
  - Formal re-use
  - Enables collaboration