

2nd International Conference on
Mathematical Methods & Computational Techniques in Science & Engineering

The Systems Engineering of Consistent Pure Language
with Effect Type System for Certified Applications and Higher Languages

PTS-infinity

Murray Edwards College, University of Cambridge, UK, February 16-18, 2018
MMCTSE 2018 144 Maksym Sokhatskyi, Pavlo Maslianko

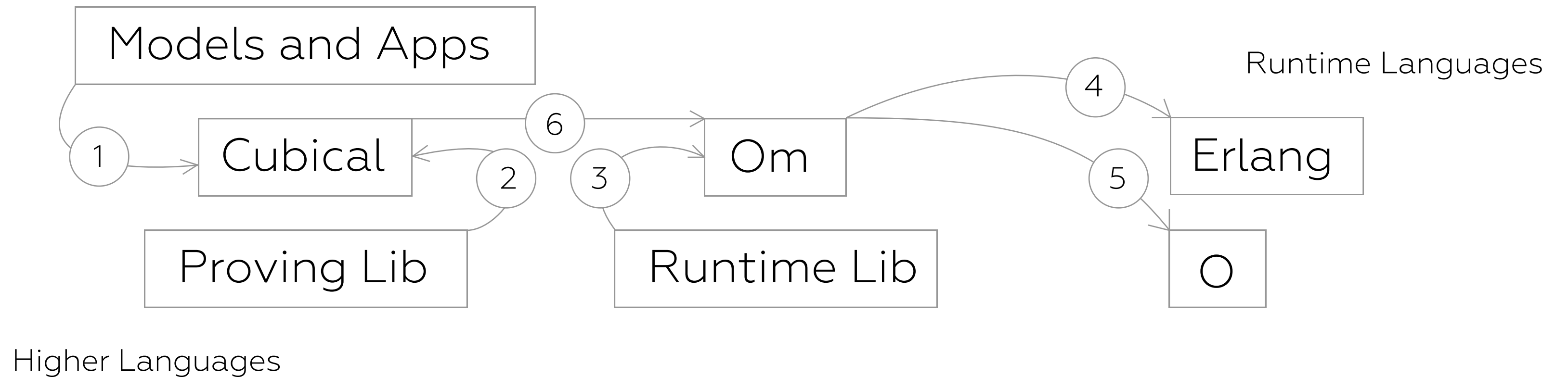
Abstract

PTS Language for Erlang virtual machines

Om — pure type system language as minimal core for evaluating Erlang programs for BEAM and LING virtual machines. The main motivation is trusted clean implementation in 260 lines of code that easily can be replicated by the authority site. The engineered system is built compatible with most lambda evaluators syntactically (Morte, Caramel) and semantically (Henk, CoC, PTS). Om language is member of languages family for proving and running verified applications.

Structure

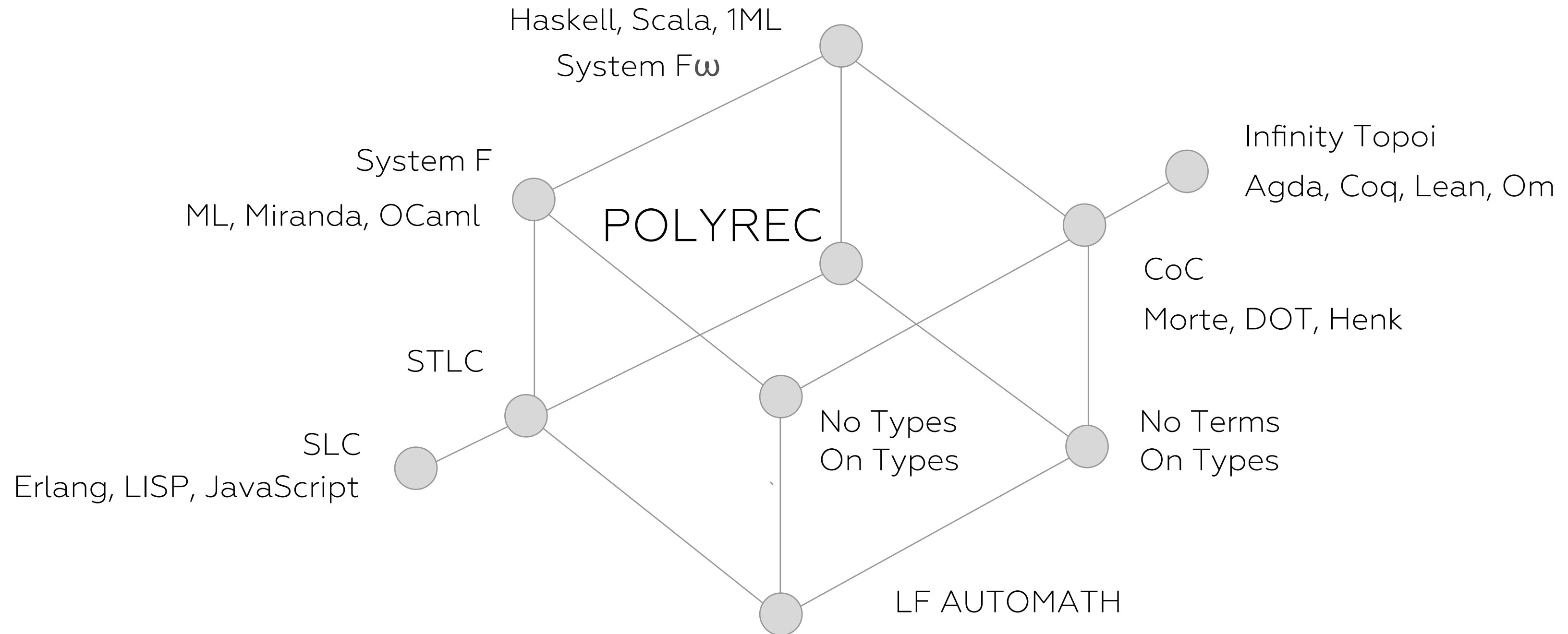
Models, Languages, Libraries, Applications



[3,4] cover the presented work, [1,2,5,6] cover the future works.

Programming Languages

in Extended Lambda Cube



Model Verification Process

From proving to Extraction, Linking and Running

1) Models

- IR/II
- Bohm
- HoTT

3) Extraction

- LLVM
- Interpreters
- Detying
- Optimization
- Linking

2) Core – Infinity Language

- Model Verification
- Normalization
- Bidirectional Checking
- Pure Type System (Om)
- Identity
- Induction
- Homotopy Interval [0,1]

4) Runtimes

- O
- Erlang
- V8
- JVM

Runtime Languages

Through a Prism of Engineering

JIT	Interpreters	LLVM	Non-LLVM
LuaJIT	K	Rust	OCaml
V8	LING/Erlang	Julia	GHC
SpiderMonkey	O	C/C++	Spiral
EDGE			
JVM/HotSpot			
CLR			

Higher Languages

for Proving and Model Checking

Target	Class	Higher Language	Type Theory
CPU	Non-LLVM	Spiral	System F
JVM	JIT	Scala	System F-omega
GHC	Non-LLVM	Morte	CoC
Erlang	Interpreter	Om	PTS-infinity
O	Interpreter	Om	PTS-infinity
Haskell	Extract	Coq/Agda	CiC

MLTT — CT — Proof Theory

Syntax — Semantics — Logic

$x : A$ — x is a object of type A

$y = [x : A]$ — x and y are definitionally
equal objects of type A

$\text{List } A = 1 \longrightarrow \text{List } A \longrightarrow \text{List } A \longleftarrow A : U$

$\text{nil} : 1 \longrightarrow \text{List } A$

$\text{cons} : A \longrightarrow \text{List } A \longrightarrow \text{List } A$

$\text{Nat} : U \rightarrow U$ — constant functor

$\text{List } (A : U \rightarrow U) : U \rightarrow U$ — functor

$\text{List } \text{Nat} : U \rightarrow U$ — constant functor

$\text{Nat} = 1 \longrightarrow N \longrightarrow N : U$

$\text{zero} : 1 \longrightarrow N$

$\text{succ} : N \longrightarrow N$

Pure Type System

Infinity Topoi

$U_0 : U_1 : U_2 : U_3 : \dots \infty$

U_0 — propositions

U_1 — sets

U_2 — types

U_3 — sorts

$S (n : \text{nat}) = U \ n$

$A_1 (n \ m : \text{nat}) = U \ n : U \ m \text{ where } m > n$ — cumulative

$R_1 (m \ n : \text{nat}) = U \ m \longrightarrow U \ n : U \ (\max \ m \ n)$ — predicative

$A_2 (n : \text{nat}) = U \ n : U \ (n + 1)$ — non-cumulative

$R_2 (m \ n : \text{nat}) = U \ m \longrightarrow U \ n : U \ n$ — impredicative

$\text{Prop} = \text{Large } \Omega_0 = U_0$

$\Sigma = \text{Large } \Omega_2 = U_2$

Pi Type

Inductive Type, AST, Logical Framework

```
data O1 := U : nat → O1
  | Var: Ident → O1
  | App: O1 → O1 → O1
  | Lambda: Binder → O1 → O1 → O1
  | Arrow: O1 → O1 → O1
  | Pi: name → O1 → O1 → O1.
```

$\langle \rangle ::= \#option$

$V ::= \#identifier$

$S ::= * < \#number >$

$O ::= S \mid V \mid (O) \mid O O \mid O \rightarrow O$

$\mid \lambda (l: O) \rightarrow O \mid \forall (l: O) \rightarrow O$

```
record Pi (A: Type) :=
  intro: (A → Type) → Type :=
  fun: (B: A → Type) →  $\forall$  (a: A) → B a → intro B
  app: (B: A → Type) → intro B →  $\forall$  (a: A) → B a
  app-fun (B: A → Type) (f:  $\forall$  (a: A) → B a):  $\forall$  (a: A) → app (fun f) a = f a
  fun-app (B: A → Type) (p: intro B): fun ( $\lambda$  (a: A) → app p a) = p
```

Shifting

Modified version of De Bruin indeces

sh (:star, X)	$N\ P \rightarrow (:star, X)$
(:var, N, I)	$N\ P \rightarrow (:var, N, I+1)$ when $I \geq P$ $\rightarrow (:var, N, I)$
(:remote, X)	$N\ P \rightarrow (:remote, X)$
(:pi, N, 0, I, O)	$N\ P \rightarrow (:pi, N, 0, sh\ I\ N\ P, sh\ O\ N\ (P+1))$
(:fn, N, 0, I, O)	$N\ P \rightarrow (:fn, N, 0, sh\ I\ N\ P, sh\ O\ N\ (P+1))$
(:app, L, R)	$N\ P \rightarrow (:app, L, R)$

Substitution

Replacing variable occurrence in terms

sub (:star, X)	$N \ V \ L \rightarrow (:star, X)$
(:var, N, L)	$N \ V \ L \rightarrow V$
(:var, N, I)	$N \ V \ L \rightarrow (:var, N, I-1)$ when $I > L$
(:remote, X)	$N \ V \ L \rightarrow (:remote, X)$
(:pi, N, O, I, O)	$N \ V \ L \rightarrow (:pi, N, O, \text{sub } I \ N \ V \ L, \text{sub } O \ N \ (\text{sh } V \ N \ O) \ L+1)$
(:pi, F, X, I, O)	$N \ V \ L \rightarrow (:pi, F, X, \text{sub } I \ N \ V \ L, \text{sub } O \ N \ (\text{sh } V \ F \ O) \ L)$
(:fn, N, O, I, O)	$N \ V \ L \rightarrow (:fn, N, O, \text{sub } I \ N \ V \ L, \text{sub } O \ N \ (\text{sh } V \ N \ O) \ L+1)$
(:fn, F, X, I, O)	$N \ V \ L \rightarrow (:fn, F, X, \text{sub } I \ N \ V \ L, \text{sub } O \ N \ (\text{sh } V \ F \ O) \ L)$
(:app, F, A)	$N \ V \ L \rightarrow (:app, \text{sub } F \ N \ V \ L, \text{sub } A \ N \ V \ L)$

Normalization

Replacing variable occurrence in terms

type (:star, N)	$D \rightarrow (:star, N+1)$
(:var, N, I)	$D \rightarrow :true = \text{proplists:defined } N \ B, \text{om:keyget } N \ D \ I$
(:remote, N)	$D \rightarrow \text{om:cache}(\text{type } ND)$
(:pi, N, O, I, O)	$D \rightarrow (:star, h(\text{star}(\text{type } I \ D)), \text{star}(\text{type } O \ [(N, \text{norm } I) D]))$
(:fn, N, O, I, O)	$D \rightarrow \text{let } \text{star}(\text{type } I \ D), \ NI = \text{norm } I$ $\quad \text{in } (:pi, N, O, NI, \text{type}(O, [(N, NI) D]))$
(:app, F, A)	$D \rightarrow \text{let } T = \text{type}(F, D), \ (:pi, N, O, I, O) = T, \ :true = \text{eq } I \ (\text{type } AD)$ $\quad \text{in } \text{norm}(\text{subst } O \ N \ A)$

Type Inference

Type Checker

type (:star, N)	$D \rightarrow (:star, N+1)$
(:var, N, I)	$D \rightarrow :true = \text{proplists:defined } N \text{ B, om:keyget } N \text{ D I}$
(:remote, N)	$D \rightarrow \text{om:cache}(\text{type } N \text{ D})$
(:pi, N, O, I, O)	$D \rightarrow (:star, h(\text{star}(\text{type } I \text{ D})), \text{star}(\text{type } O [(N, \text{norm } I) D]))$
(:fn, N, O, I, O)	$D \rightarrow \text{let star (type } I \text{ D), NI = norm I}$ $\quad \text{in } (:pi, N, O, NI, \text{type}(O, [(N, NI) D]))$
(:app, F, A)	$D \rightarrow \text{let } T = \text{type } (F, D), (:pi, N, O, I, O) = T, :true = \text{eq } I (\text{type } A \text{ D})$ $\quad \text{in norm (subst } O \text{ N A)}$

Equality

Definitional, built into Type Checker

eq (:star ,N)	(:star ,N)	→ true
(:var,N,l)	(:var,(N,l))	→ true
(:remote ,N)	(:remote ,N)	→ true
(:pi,N1,O,l1,O1)	(:pi,N2,O,l2,O2)	→ let :true = eq l1 l2 in eq O1 (subst (shift O2 N1 O) N2 (:var,N1,O) O)
(:fn,N1,O,l1,O1)	(:fn,N2,O,l2,O2)	→ let :true = eq l1 l2 in eq O1 (subst (shift O2 N1 O) N2 (:var,N1,O) O)
(:app,F1,A1)	(:app,F2,A2)	→ let :true = eq F1 F2 in eq A1 A2
(A,	B)	→ (:error ,(:eq,A,B))

Language Usage

Erlang or UNIX shell

Om implemented as escript file accessible from UNIX command line, resembling the Erlang API for manipulating Om terms and contexts.

```
$ ./om help me
[{a,[expr],"to parse. Returns { , } or {error , }."},
 {type,[term],"typechecks and returns type."},
 {erase ,[ term ] ," to untyped term . Returns { , }."},
 {norm,[term],"normalize term. Returns term's normal form."},
 {file ,[name],"load file as binary."},
 {str ,[binary],"lexical tokenizer."},
 {parse ,[ tokens ] ," parse given tokens into { , } term ."},
 {fst ,[{x,y}],"returns first element of a pair."},
 {snd ,[{x,y}] ," returns second element of a pair ."},
 {debug ,[ bool ] ," enable / disable debug output ."}, {mode,[name],"select metaverse folder."},
 {modes ,[ ] ," list all metaverses ."}]
```


Language Usage

Erlang or UNIX shell

```
$ ./om print fst erase norm a "#List/Cons"
```

```
    \ Head
```

```
-> \ Tail
```

```
-> \ Cons
```

```
-> \ Nil
```

```
-> Cons Head (Tail Cons Nil)
```

```
ok
```

Applications: Sigma Types

Syntax and Model

data $O_2 := O_1$

| Sigma: name $\rightarrow O_2 \rightarrow O_2 \rightarrow O_2$

| Pair: $O_2 \rightarrow O_2 \rightarrow O_2$

| Fst: $O_2 \rightarrow O_2$

| Snd: $O_2 \rightarrow O_2$.

$\Sigma x: A, B x : U$ — formation rule

pair (x : A) (y : B x) — introduction

pr1 s : A — elimination

pr2 s : B x — elimination

data Sigma (A: Type) (P: A \rightarrow Type) (x: A): Type =
intro: P x \rightarrow Sigma A P

Sigma Types

Typing and Introduction Rules

```
-- Sigma/@ \ (A: *)  
-> \ (P: A -> *)  
-> \ (n: A)  
-> \ (Exists: *)  
-> \ (Intro: A -> P n -> Exists)  
-> Exists
```

```
-- Sigma/Intro \ (A: *)  
-> \ (P: A -> *)  
-> \ (x: A)  
-> \ (y: P x)  
-> \ (Exists: *)  
-> \ (Intro: \ (x:A) -> P x -> Exists)  
-> Intro x y
```

Sigma Types

Dependent Eliminators

```
-- Sigma/fst \ (A: *)  
-> \ (B: A -> *)  
-> \ (n: A)  
-> \ (S: #Sigma/@ABn)  
-> S A ( \ (x: A) -> \ (y: B n) -> x )
```

```
-- Sigma/snd \ (A: *)  
-> \ (B: A -> *)  
-> \ (n: A)  
-> \ (S: #Sigma/@ABn)  
-> S (B n) ( \ ( : A) -> \ (y: B n) -> y )
```

Equ Type a la Martin-Löf

record Id (A: Type): Type :=

Id: A → A → Type

refl (a: A): Id a a

Predicate: $\forall (x,y: A) \rightarrow \text{Id } x \ y \rightarrow \text{Type}$

Forall (C: Predicate): $\forall (x,y: A) \rightarrow \forall (p: \text{Id } x \ y) \rightarrow C \ x \ y \ p$

Δ (C: Predicate): $\forall (x: A) \rightarrow C \ x \ x \ (\text{refl } x)$

axiom-J (C: Predicate): $\Delta \ C \rightarrow \text{Forall } C$

computation (C: Predicate) (t: $\Delta \ C$): $\forall (x: A) \rightarrow (J \ C \ t \ x \ x \ (\text{refl } x)) ==> (t \ x) \)$

record Subst (A: Type): Type :=

intro (P (a: A): Type) (a1,a2: A) : Id a1 a2 → P a1 → P a2 :=

Id.axiom-J ($\lambda a1 \ a2 \ p12 \rightarrow P \ a1 \rightarrow P \ a2$))

Equ Type in Om

```
-- Equ/Refl \ (A: *)
-> \ (x: A)
-> \ (Equ: A -> A -> *)
-> \ (Refl: \ (z: A) -> Equ z z)
-> Refl x

-- Equ/@
\ (A: *)
-> \ (x: A)
-> \ (y: A)
-> \ (Equ: A -> A -> *)
-> \ (Refl: \ (z: A) -> Equ z z) -> Equ x y
```

Effects Protocol

Type Spec

String: Type = List Nat

```
data IO: Type =  
  getLine: (String -> IO) -> IO  
  putLine: String -> IO  
  pure: () -> IO
```

```
-- IO/@  
  \ (a : *)  
-> \ (IO : *)  
-> \ (GetLine : (#IO/data -> IO) -> IO)  
-> \ (PutLine : #IO/data -> IO -> IO)  
-> \ (Pure : a -> IO) -> IO
```

Replication

Church Encoded Identity Folding

```
-- IO/replicateM
  \ (n: #Nat/@)
-> \ (io: #IO/@ #Unit/@)
-> #Nat/fold n (#IO/@ #Unit/@)
              (#IO[>>] io)
              (#IO/pure #Unit/@ #Unit /Make)

— Nat/fold
#id #Nat/@
```


Runtime Recursion Sample

Recursion Elimination

```
-- Recursion
((#IO/replicateM #Nat/Five)
  ((((#IO/[>>=] #IO/data) #Unit/@) #IO/getLine)
    #IO/putLine))
```

Infinity I/O

Corecursion Fixpoint

```
-- IOI/@  
  \ (r : *)  
-> \ (x : *)  
-> (\ (s : *)  
-> s  
-> (s -> #IOI/F r s) -> x)  
-> x
```

```
-- IOI/F  
  \ (a : *)  
-> \ (State : *)  
-> \ (IOF : *)  
-> \ (PutLine : #IOI/data -> State -> IOF)  
-> \ (GetLine : (#IOI/data -> State) -> IOF)  
-> \ (Pure : a -> IOF)  
-> IOF
```

Infinity I/O Construction

Corecursion Introduction

```
-- IOI/MkIO
  \ (r : *)
-> \ (s : *)
-> \ (seed: s)
-> \ (step: s -> #IOI/F r s)
-> \ (x : *)
-> \ (k : forall (s : *) -> s -> (s -> #IOI/F r s) -> x)
-> k s seed step
```

Infinity I/O Process

Corecursion Elimination

-- Corecursion

```
( \ (r: *1) -> ( (((#IOI/MkIO r) (#Maybe/@ #IOI/data))
  (#Maybe/Nothing #IOI/data))
  ( \ (m: (#Maybe/@ #IOI/data))
    -> (((((#Maybe/maybe #IOI/data) m)
      ((#IOI/F r) (#Maybe/@ #IOI/data)))
      ( \ (str: #IOI/data)
        -> ((((#IOI/putLine r ) (#Maybe/@ #IOI/data)) str)
          (#Maybe/Nothing #IOI/data))))
      (((#IOI/getLine r ) (#Maybe/@ #IOI / data ))
        (#Maybe/Just #IOI/data))))))
```

Application 1. Logic

`data` Proper (A: Type) (R: A → A → Prop) (m: A): Prop := intro: R m m

`data` Inhabited (A: Type): Prop := intro: A → Inhabited A

`data` True: Prop := intro: () → True

`data` False: Prop := ()

`data` Eq (A : Type): A → A → Type := refl: $\forall (x: A) \rightarrow \text{Eq } A \ x \ x$

`data` Exists (A: Type): A → Type → Type := intro: $\forall (P: A \rightarrow \text{Type}) \rightarrow$
 $\forall (x: A) \rightarrow P \ x \rightarrow \text{Exists } A \ P$

`record` SKI := S : (p → q → r) → (p → q) → p → r
 K : p → (q → p)
 I : p → p

Application 2. Control

record pure (P: Type → Type) (A: Type): Type := return: P A

record functor (F: Type → Type) (A,B: Type): Type := map: (A → B) → F A → F B

record applicative (F: Type → Type) (A,B: Type): Type :=

pure: pure F A

functor: functor F A B

ap: F (A → B) → F A → F B

record monad (F: Type → Type) (A,B: Type): Type :=

pure: pure F A

functor: functor F A B

join: F (F A) → F B

Benchmarks

Operation	Type	Time
Pack/Unpack 1M	Inductive Nat	776407 us
Pack/Unpack 1M	Inductive List	1036461 us
Pack/Unpack 1M	Erlang/OTP	148084 us
Type Checking	Om ShadowTrans	4.972s
Type Checking	Morte ShadowTrans	57.867s

Components

Erlang/OTP Modules

File	LOC	Description
om_tok	54	Handcoded Tokenizer
om_parse	81	Inductive AST Parser
om_type	60	Term normalization and typechecking
om_erase	36	Delete information about types
om_extract	34	Extract Erlang Code

Runtime Library

Simple Types shipped with Compiler

Bool

Cmd

Equ

Either

Frege

IO

IOI

Lazy

Leibnitz

List

Maybe

Mon

Monad

Monoid

Exec

Nat

Path

Prod

Prop

Sigma

Simple

String

Sum

Unit

Vector

Higher Language

$\langle \rangle ::= \# \text{option}$

$[] ::= \# \text{list}$

$| ::= \# \text{sum}$

$1 ::= \# \text{unit}$

$l ::= \# \text{identifier}$

$U ::= \text{Type} \langle \# \text{nat} \rangle$

$T ::= 1 \mid (l : O) T$

$F ::= 1 \mid l : O = O, F$

$B ::= 1 \mid [\mid [l] \rightarrow O]$

$O ::= l \mid (O) \mid U$

$\mid O \rightarrow O$

$\mid \text{fun} (l : O) \rightarrow O$

$\mid \text{snd } O$

$\mid J \ O \ O \ O \ O \ O$

$\mid (l : O) * O$

$\mid \text{data } l \ T : O := T$

$\mid O \ O$

$\mid \text{fst } O$

$\mid \text{id } O \ O \ O$

$\mid \text{let } F \text{ in } O$

$\mid (l : O) \rightarrow O$

$\mid \text{record } l \ T : O := T$

$\mid \text{case } O \ B$

Conclusion

Results and Future development

- Fixpoint as Induction essence
- Fixpoint separation and absence Paradoxes
- Refined Type Checker 265 LOC for Erlang/OTP
- Base Library for Certified Applications
- IO Samples and Effect Type System
- Higher Language AST Proposal

Maksym Sokhatskyi: maxim@synrc.com

Pavlo Maslianko: mppdom@i.ua

Bibliography

1. Barendregt. The Lambda Calculus with Types <http://5ht.co/pts.pdf>
2. Martin-Löf. Intuitionistic Type Theory <http://5ht.co/mltt.pdf>
3. Awodey. Category Theory <http://5ht.co/cat.pdf>
4. Hermida, Jacobs. Fibrations with indeterminates <http://5ht.co/completeness.pdf>
5. Jacobs. Categorical Logic <http://5ht.co/fibrations.pdf>
6. Streicher. The groupoid interpretation of type theory <http://5ht.co/groupoid.pdf>
7. Voevodsky et al. Homotopy Type Theory <http://5ht.co/hott.pdf>
8. Huber, Coquand. Cubical Type Theory <http://5ht.co/cubicaltt.pdf>