

First-class open and closed code fragments

TFP, Tallinn 2005

Morten Rhiger

Roskilde University, Denmark

Goal

- Extend

a typed, higher-order language

with support for

run-time code generation
and execution

Motivation for run-time code generation

- Enable run-time program specialization:
 - May improve time and space usage
 - Applies in situations where "static specialization" does not

Motivation for run-time code execution

- Enable execution of the generated programs

Challenge of run-time code generation

- To **statically** guarantee that generated programs are well typed
- (To provide an efficient implementation)

Related work

- (Quasiquotations in Lisp and Scheme)
- Calculi based on modal and temporal logics, mid-1990's (Davies, Pfenning)
- MetaML, since 1997 (Sheard, Taha, Moggi, Calcagno, Benaissa, Nielsen, ...)
- Name-based calculi, since 2002 (Nanevski, Pfenning, Moggi, Ancona, Pientka)
- ...

Plan

- Motivation
- Presentation of $\lambda[]$
 - What is it?
 - How does it achieve its goals?
 - Is it relevant and useful?
 - Is it sound?
 - Is it part of a new trend?
- Conclusion

What is $\lambda^{\llbracket \cdot \rrbracket}$, part 1?

- A monomorphic, type-safe multi-stage language which
 - has a **first-class run operation**
 - supports **mutable cells**
 - provides a type for **open and closed code** (which is **first-class data**)

What is λ^{\perp} , part 2?

- A conservative extension of the simply-typed lambda calculus with
 - only **one new type** (of code)
 - only **4 new kinds of expressions**
 - only **4 new typing rules**

Plan

- Motivation
- Presentation of $\lambda[]$
 - What is it?
 - How does it achieve its goals?
 - Is it relevant and useful?
 - Is it sound?
 - Is it part of a new trend?
- Conclusion

How does $\lambda^[]$ achieve its goals?

- $\lambda^[]$ is **not hygienic**:
 - Dynamic variables **are not renamed** during substitutions at run time
 - (Static variables **are renamed**)

Consequences of non-hygiene for the semantics

- What do we lose?
 - Alpha conversion for code fragments
 - (A good language design principle)
 - The ability to implement certain applications

Consequences of non-hygiene for the type system

- What do we gain?
 - The type system can easily “track” variables defined and used in code fragments

The syntax of $\lambda[]$: Functional fragment

- Expressions E are
 - Constants: $1, 2, 3, \dots$
 - Variables: x, y, z, \dots
 - Abstractions: $\lambda x. E$
 - Applications: $E_1 E_2$

The syntax of $\lambda[\]$: Multi-stage fragment

(continued)

- Code introduction: $\uparrow E$
- Code elimination: $\downarrow E$
- Lifting values: $\text{lift } E$
- Evaluation: $\text{run } E$

The semantics of $\lambda[]$

- Scheme's quasiquote macro system without gensym

The semantics of $\lambda[\]$:

Free (static) variables

- Free static variables in stage- n term E :

$$FV_n(E)$$

Free (static) variables: Variables

- $FV_0(x) = \{ x \}$

- $FV_{n+1}(x) = \{ \}$

Free (static) variables: Abstractions

- $FV_0(\lambda x. E) = FV_0(E) \setminus \{x\}$

- $FV_{n+1}(\lambda x. E) = FV_{n+1}(E)$

Free (static) variables: Stage transitions

- $FV_n(\uparrow E) = FV_{n+1}(E)$
- $FV_{n+1}(\downarrow E) = FV_n(E)$

The semantics of $\lambda[\]$: Substituting (static) variables

- Replacing a static variable in stage- n term E :

$$E\{E' / x\}_n$$

Substituting (static) variables: Variables

- $x \{E / x\}_0 = E$
- $y \{E / x\}_0 = y, \quad \text{not } x=y$
- $y \{E / x\}_{n+1} = y$

Substituting (static) variables: Abstractions

- $(\lambda x. E) \{E' / x\}_0 = \lambda x. E$
- $(\lambda z. E) \{E' / x\}_0 = \lambda y. E \{y / z\}_0 \{E' / x\}_0$
not $z=x$ and where y is "fresh"
- $(\lambda z. E) \{E' / x\}_{n+1} = \lambda z. E \{E' / x\}_{n+1}$

Substituting (static) variables: Stage transitions

- $(\uparrow E) \{E' / x\}_n = \uparrow E \{E' / x\}_{n+1}$

- $(\downarrow E) \{E' / x\}_{n+1} = \downarrow E \{E' / x\}_n$

The semantics of $\lambda[\]$: (Static) reductions

- Statically reducing a stage- n term E :

$$E \rightarrow_n E'$$

(Static) reductions

- $(\lambda x. E) V \rightarrow_0 E\{V/x\}_0$
- $\downarrow(\uparrow V) \rightarrow_1 V$
- $\text{lift } V \rightarrow_0 \uparrow V$
- $\text{run } (\uparrow V) \rightarrow_0 V$

The type system of $\lambda[]$

- Types T
 - Base types: int, \dots
 - Function types: $T_1 \rightarrow T_2$
 - Code type: $[\gamma]T$
- Type environments: γ
- Stacks of environments: $\Gamma = \gamma_0, \gamma_1, \dots, \gamma_n$

The code type of $\lambda[]$

- New type of code: $[\gamma]T$
 - γ is a (completely standard) type environment
 - T is a (completely standard) type

The code type of $\lambda[\]$

- Intuition
 - $[\gamma]T$ is the type of code fragments that have type T in type environment γ

The code type of $\lambda[\]$:

Examples

- Open code

$$\uparrow x : [x:\text{int}]\text{int}$$
$$\uparrow (f\ x) : [f:A \rightarrow B, x:A]B$$

- Closed code

$$\uparrow (\lambda f. f\ 4) : []((\text{int} \rightarrow A) \rightarrow A)$$

The code type of $\lambda[\]$:

Principal types

- Open code

$$\uparrow x : [x:A; \gamma]A$$

$$\uparrow(f\ x) : [f:A \rightarrow B, x:A; \gamma]B$$

- Closed code

$$\uparrow(\lambda f. f\ 4) : [\gamma]((\text{int} \rightarrow A) \rightarrow A)$$

The typing judgment of $\lambda[]$

$$\Gamma \vdash E : \tau$$

The typing rules of $\lambda^{\llbracket \cdot \rrbracket}$:

Variables

$$\frac{\gamma(x) = T}{\Gamma, \gamma \vdash x : T}$$

The typing rules of $\lambda^{\llbracket \cdot \rrbracket}$:

Abstractions

$$\frac{\Gamma, \gamma[x:T] \vdash E:T_2}{\Gamma, \gamma \vdash \lambda x. E: T_1 \rightarrow T_2}$$

The typing rules of $\lambda[\]$:

Applications

$$\frac{\Gamma, \gamma \vdash E1 : T2 \rightarrow T \quad \Gamma, \gamma \vdash E2 : T2}{\Gamma, \gamma \vdash E1 E2 : T}$$

The typing rules of $\lambda[\]$:

Code introduction

$$\frac{\Gamma, \gamma, \gamma' \vdash E : \tau}{\Gamma, \gamma \vdash \uparrow E : [\gamma']\tau}$$

The typing rules of $\lambda[\]$:

Code elimination

$$\frac{\Gamma, \gamma \vdash E : [\gamma']\tau}{\Gamma, \gamma, \gamma' \vdash \downarrow E : \tau}$$

The typing rules of $\lambda[\]$: Lifting

$$\frac{\Gamma, \gamma \vdash E : T}{\Gamma, \gamma \vdash \text{lift } E : [\gamma']T}$$

The typing rules of $\lambda[]$: Evaluation

$$\frac{\Gamma, \gamma \vdash E : []T}{\Gamma, \gamma \vdash \text{run } E : T}$$

Plan

- Motivation
- Presentation of $\lambda[\]$
 - What is it?
 - How does it achieve its goals?
 - Is it relevant and useful?
 - Is it sound?
 - Is it part of a new trend?
- Conclusion

Standard example (1)

- Staging the power function

genpow : int × [γ]int → [γ]int

genpow(n,x) =

if n=0 then ↑1

else ↑(↓x * ↓(pow(n-1, x)))

Standard example (2)

- Using the staged power function

$\text{genpow} : \text{int} \times [\gamma]\text{int} \rightarrow [\gamma]\text{int}$

$\uparrow z : [z:\text{int}]\text{int}$

Standard example (3)

- Using the staged power function

$\text{genpow}(3, \uparrow z) : [z:\text{int}]\text{int}$

$\text{genpow}(3, \uparrow z) = \uparrow(z * z * z * 1)$

Standard example (4)

- Using the staged power function

$\uparrow(\lambda z. \downarrow(\text{genpow}(3, \uparrow z))) : [](\text{int} \rightarrow \text{int})$

$\uparrow(\lambda z. \downarrow(\text{genpow}(3, \uparrow z))) = \uparrow(\lambda z. z * z * z * 1)$

Standard example (5)

- Using the staged power function

run $\uparrow(\lambda z. \downarrow(\text{genpow}(3, \uparrow z))) : \text{int} \rightarrow \text{int}$

run $\uparrow(\lambda z. \downarrow(\text{genpow}(3, \uparrow z))) = \textit{function}$

Plan

- Motivation
- Presentation of $\lambda[\]$
 - What is it?
 - How does it achieve its goals?
 - Is it relevant and useful?
 - Is it sound?
 - Is it part of a new trend?
- Conclusion

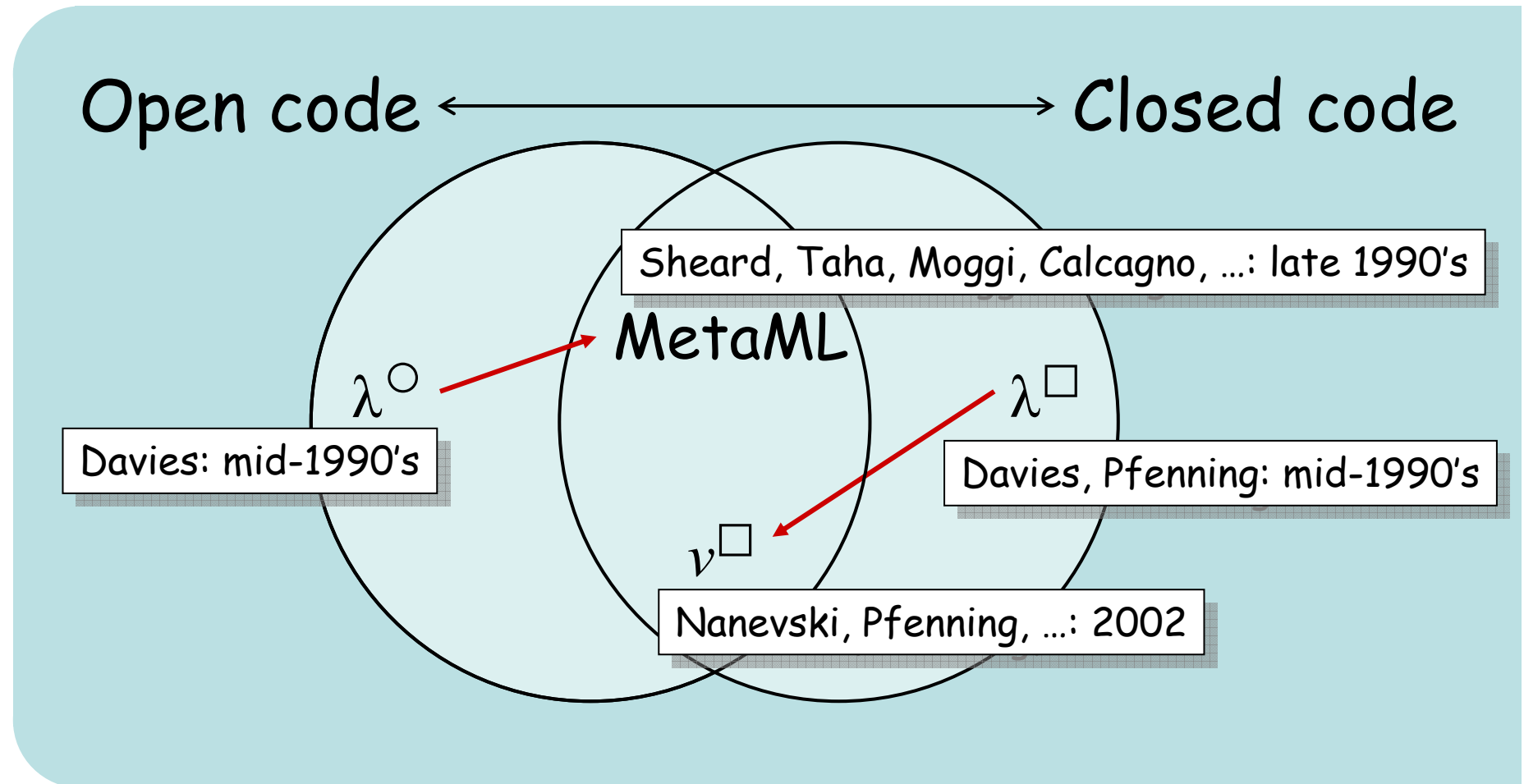
Is λ sound?

- Yes
- (See the paper)
- (See also my talk at the MetaOCaml Workshop on Wednesday, September 28, 2005)

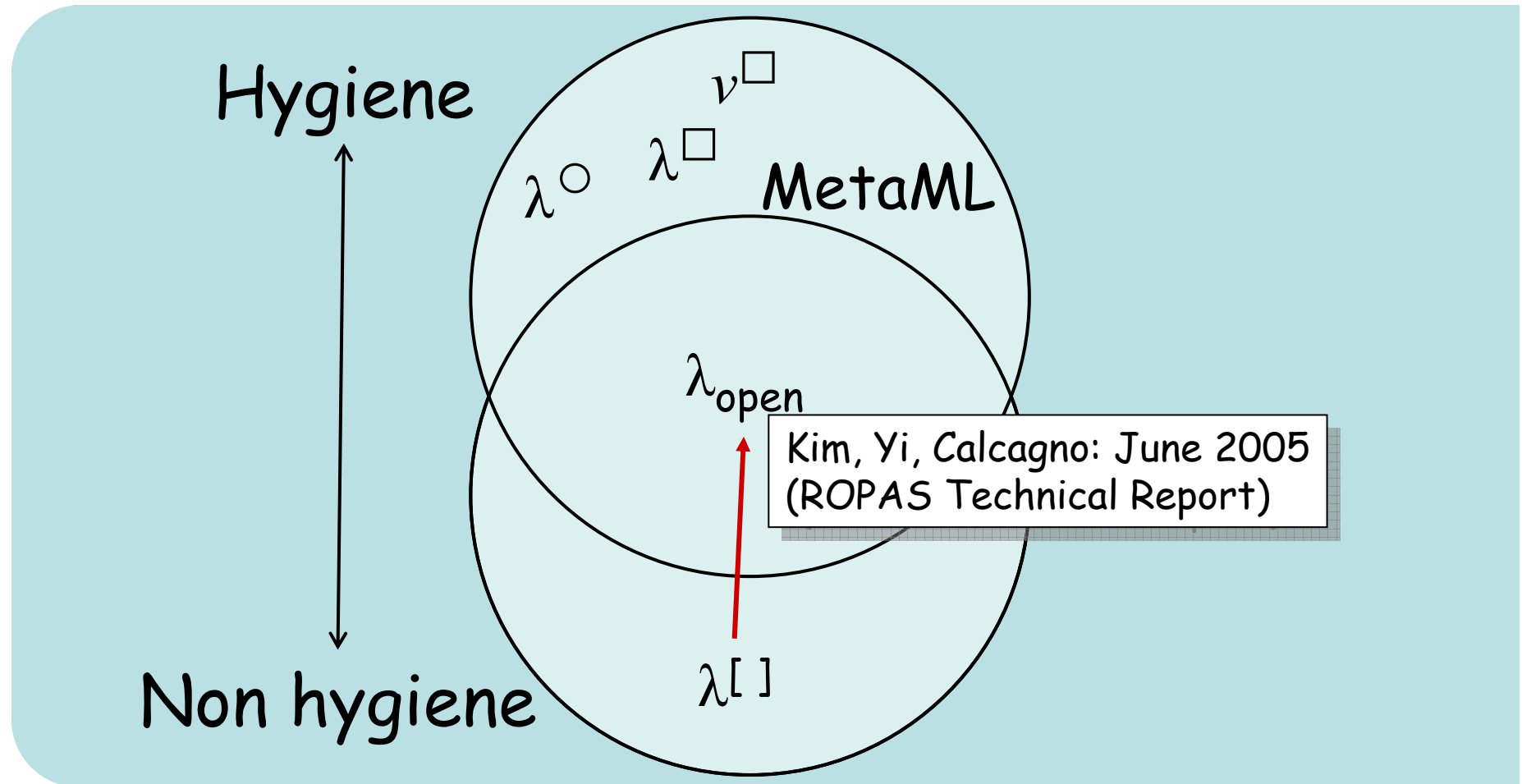
Plan

- Motivation
- Presentation of $\lambda[\]$
 - What is it?
 - How does it achieve its goals?
 - Is it relevant and useful?
 - Is it sound?
 - Is it part of a new trend?
- Conclusion

The interplay between open and closed code



The interplay between hygiene and non hygiene



Plan

- Motivation
- Presentation of $\lambda[\]$
 - What is it?
 - How does it achieve its goals?
 - Is it relevant and useful?
 - Is it sound?
 - Is it part of a new trend?
- Conclusion

Conclusion

- $\lambda^[]$ is a non-hygienic staged language
 - It is type safe
 - It has a run operation
 - It supports mutable cells
 - It builds on known concepts