Lambda-calculi with Patterns

ISR '21

Luigi Liquori, UCA/INRIA, F & Vincent van Oostrom U. Inssbruck, A

λ-calculus with Patterns: in one slide

$$M \quad ::= \quad f \mid x \mid \lambda P.N \mid (M\,N) \quad \text{Terms}$$

- $P \subset M$ satisfying the RPC-condition = variables can be bound only if they occur linearly (i.e. at most once) and not actively (i.e. not in functional position). Otherwise Church-Rosser can be lost
- $(\lambda P.N)\,M \longrightarrow_{\beta\mathcal{T}} N\theta$ where $\theta = Sol(P \ll_{\mathcal{T}} M)$
- Different design choices can be found the very rich literature
  - "Pattern Syntax" (there are many notations in the literature)
  - "Shape of Good Patterns" (there are others CR-safe conditions)
  - "Matching Theories" (there are many matching theories, *e.g.* $T_{\mathcal{AC}}$)
  - "Type Theories" (there are many type theories offering SR and SN)
- Let's have an incomplete "panorama" on those topics together...

Unifying Functional and Algebraic Notations

This make the writing of patterns and of "λ-pattern code" more readable

$$(cons\ M\ N) \equiv cons(M, N) \qquad \text{Curryfication } \textit{vs.} \text{ Algebraic}$$
$$cons(M, N) \equiv (M\ ,\ N) \qquad \textit{cons IS the infix ","}$$

about lists : $(cons\ f\ (cons\ g\ nil)) \equiv cons(f\ , cons(g\ , nil)) \equiv (f\ , g)$

$$head \stackrel{def}{=} \lambda(x\ , y).x$$
$$tail \stackrel{def}{=} \lambda(x\ , y).y$$
$$merge \stackrel{def}{=} \lambda((x\ , y)\ , (z\ , w)).(x, z, merge(y, w))$$

Reminder: pair and list encoding in λ-calculus

$$
\begin{array}{rcl}
\mathit{true} & \stackrel{\mathit{def}}{=} & \lambda xy.x \\
\mathit{false} & \stackrel{\mathit{def}}{=} & \lambda xy.y \\
\mathit{pair} & \stackrel{\mathit{def}}{=} & \lambda xyz.((z\,x)\,y) \\
\mathit{first} & \stackrel{\mathit{def}}{=} & \lambda x.x\;\mathit{true} \\
\mathit{second} & \stackrel{\mathit{def}}{=} & \lambda x.x\;\mathit{false} \\
\mathit{list} & \stackrel{\mathit{def}}{=} & \lambda xy.(\mathit{pair}\;x\;y) \longrightarrow\!\!\!\!\!\rightarrow_{\beta} \lambda z.((z\,x)\,y) \\
\mathit{head} & \stackrel{\mathit{def}}{=} & (\mathit{list}\;M\;N\;\mathit{true}) \\
\mathit{tail} & \stackrel{\mathit{def}}{=} & (\mathit{list}\;M\;N\;\mathit{false}) \\
\mathit{merge} & \stackrel{\mathit{def}}{=} & \text{left as an exercise}
\end{array}
$$

Reduction Speed-up and the "Eating Metaphor"

- Lambda-calculus enriched with patterns "speed-up" reductions. There will be a bigger speed-up as patterns are bigger

$$\Lambda_P \quad (\lambda(x\,,y).x)\,(3,4) \quad \longrightarrow_{\beta_\mathcal{T}} \quad 3$$

$$\Lambda \qquad (\mathit{list}\ 3\ 4\ \mathit{true}) \quad \equiv \quad (\lambda x'y'.((\lambda xyz.((z\,x)\,y))\,x'\,y'))$$

$$\longrightarrow\!\!\!\!\!\rightarrow_{\beta} \quad 3$$

- In the application $(\lambda x.M)\,N$, the argument N can "always fit", and reduction always apply, with a β-reduction to $M[N/x]$
- In the application $(\lambda P.M)\,N$, the argument N should "fit the taste of P", and reduction apply only if there exist a substitution θ solution of the matching equation $P \ll_\mathcal{T} N$: if so then $\beta_\mathcal{T}$-reduce to $N\theta$ otherwise the application simply *get stuck*
- But stuck application can unstuck, ex: $(\lambda f(x).((\lambda g(3).3)\,g(x)))\,f(3)$

About the Shape of Patterns 1/2

- Pattern application of the shape

$$\lambda f(x).M$$

are Church-Rosser preserving

- Pattern application of the shape

$$\lambda(x\ y).M$$

with a variable x in *active position* make you loose Church-Rosser

- Ex:

$(\lambda(x\ y).x)\ ((\lambda z.z)\ a) \longrightarrow_\beta \lambda z.z$ but also $(\lambda(x\ y).x)\ a$

About the Shape of Patterns 2/2

- Pattern application *non left-linear* make you loose Church-Rosser

$$\lambda f(x, x).M$$

- Ex: Klop counter example, see Terese book, ex 2.7.20, p.57

$$\begin{aligned} d(x,x) &\to e \\ c(x) &\to d(x, c(x)) \\ a &\to c(a) \end{aligned}$$

- One informal translation in Lambda-calculi with Patterns:

$$\begin{aligned} M &\equiv \lambda(x,x).e \\ C &\equiv \lambda x.D(x, C(x)) \\ A &\equiv C(A) \end{aligned}$$

About Matching Theories 1/3: Syntactic $\mathcal{T}_\emptyset$

- It is a concrete research field that has really practical applications: exact string matching, `unix xgrep`, text processing, biological applications, telco applications, etc, just to mention a few
- Syntactic $\mathcal{T}_\emptyset$ Pattern Matching

$$\frac{\Vdash_\emptyset M = N \quad \Vdash_\emptyset N = P}{\Vdash_\emptyset M = P}\ (\mathit{Trans}) \qquad \frac{\Vdash_\emptyset N = M}{\Vdash_\emptyset M = N}\ (\mathit{Symm})$$

$$\frac{\Vdash_\emptyset M = N}{\Vdash_\emptyset P[M/x] = P[N/x]}\ (\mathit{Ctx}) \qquad \frac{}{\Vdash_\emptyset M = M}\ (\mathit{Refl})$$

- Ex: The matching problem

$$f(x, y) \ll_\emptyset f(3, 4)$$

has as unique solution the substitution

$$\theta = [3/x\,; 4/y]$$

About Matching Theories 2/3: Associative & Commutative $\mathcal{T}_{\mathcal{AC}}$

- Associative & Commutative $\mathcal{T}_{\mathcal{AC}}$ Pattern Matching

$$\frac{}{\Vdash_{AC} ((M, N), P) = (M, (N, P))} \ (Associative)$$

$$\frac{}{\Vdash_{AC} (M, N) = (N, M)} \ (Commutative)$$

$$\frac{\Vdash_{AC} M = N}{\Vdash_{AC} P[M/x] = P[N/x]} \ (Ctx)$$

- Ex: The matching problem $f(x, y) \ll_{AC} f((a, (b, c))$ has 12 sol:

$$\begin{aligned}\vec{\theta} \quad = \quad & [a/x\,;(b,c)/y] - [a/x\,;(c,b)/y] - [b/x\,;(a,c)/y] - [b/x\,;(c,a)/y] \\ & [c/x\,;(a,b)/y] - [c/x\,;(b,a)/y] - [(a,b)/x\,;c/y] - [(b,a)/x\,;c/y] \\ & [(a,c)/x\,;b/y] - [(c,a)/x\,;b/y] - [(c,b)/x\,;a/y] - [(b,c)/x\,;a/y]\end{aligned}$$

About Matching Theories 3/3: 0 as Unit Element & "," as Idempotent $\mathcal{T}_{UI}$

- Unit Element 0 and "," as Idempotent $\mathcal{T}_{UI}$ Pattern Matching

$$\frac{}{\Vdash_{UI} (0, N) = N}\ (Unit^0_L) \qquad \frac{}{\Vdash_{UI} (N, 0) = N}\ (Unit^0_R)$$

$$\frac{}{\Vdash_{UI} (M, M) = M}\ (Idempotency) \qquad \frac{\Vdash_{UI} M = N}{\Vdash_{UI} P[M/x] = P[N/x]}\ (Ctx)$$

- Ex: $f(x) \ll_{UI} f(3,3,3)$ and $f(x) \ll_{UI} f(1,0,2,0,3)$ have 3 sol (resp.)

$$\vec{\theta_U} = [(3,3,3)/x] - [(3,3)/x] - [3/x]$$

$$\vec{\theta_I} = [(1,0,2,0,3)/x] - [(1,2,0,3)/x] - [(1,2,3)/x]$$

Small Step Operational Semantic, reloaded

$$(\lambda P.M)\, N \longrightarrow_{\beta_{\mathcal{T}}} \{M\theta_1 \cdots M\theta_n\}$$

- where $\mathcal{FV}(P) = \{x_1 \ldots x_n\}$ and $\exists$ a (vector) substitution $\vec{\theta} = \theta_1 \ldots \theta_n$ result of solving the *pattern-matching modulo* $\mathcal{T}$ equation $P \ll_{\mathcal{T}} N$

- NB: in case of *matching failure* the term is *stuck* and does not reduce, *i.e.* it is in *normal form*

- NB: in case of multiple solutions of a pattern matching equation in a given theory $\mathcal{T}$ we say that we are working with *matching-modulo*

Ex: Clean Doubles

$$CD \stackrel{def}{=} \lambda(x, x, z).(x, z)$$

$$L \stackrel{def}{=} (Luigi\ ,\ Vincent\ ,\ Delia\ ,\ Luigi\ ,\ Carsten\ ,\ Femke\ ,\ Carsten)$$

$$(CD\ L) \rightarrow_{\beta\,ACI} (Luigi\ ,\ Vincent\ ,\ Delia\ ,\ Carsten\ ,\ Femke)$$

but also $(Vincent\ ,\ Delia\ ,\ Luigi\ ,\ Carsten\ ,\ Femke)$

but also $(Vincent\ ,\ Delia\ ,\ Luigi\ ,\ Carsten\ ,\ Femke)$

but also $(Luigi\ ,\ Vincent\ ,\ Delia\ ,\ Femke\ ,\ Carsten)$

Ex: Garbage Collector

$$GC \stackrel{def}{=} \lambda(x, Erase(y), z).(x, z)$$

$$L \stackrel{def}{=} (Luigi\ ,\ Vincent\ ,\ Erase(John)\ ,\ Femke\ ,\ Erase(Lea))$$

$$(GC\ L) \rightarrow_{\beta\, ACI} (Luigi\ ,\ Vincent\ ,\ Femke)$$

Ex: Super Merge

Merge two lists, clean doubles, and GC

$$merge \stackrel{def}{=} \lambda((x\,,y)\,,(z\,,w)).(x,z,merge(y,w)) \qquad \text{recursive in } T_{\emptyset}$$

$$SM \stackrel{def}{=} \text{left as an exercise: mailto: } \texttt{Luigi.Liquori@inria.fr}$$

$$L_1 \stackrel{def}{=} (Luigi\,,Vincent\,,Delia\,,Erase(John)\,,Femke\,,Silvia\,,nil)$$

$$L_2 \stackrel{def}{=} (Erase(Zak)\,,Silvia\,,Pierre\,,Luigi\,,Malika\,,nil)$$

$$(SM\,(L_1,L_2)) \rightarrow_{\beta\,ACIU} (Luigi\,,Vincent\,,Delia\,,Femke\,,Malika\,,Silvia\,,Pierre\,,nil)$$

but also $\rightarrow_{\beta\,ACIU} \cdots$

but also $\rightarrow_{\beta\,ACIU} \cdots$

Typed Pattern Matching Calculi

- Milner slogan: *well-typed program do not go wrong*
- A plætora of different type system: simple types, polymorphic-types, higher-order types, dependent-types
- Let's try to add types to the syntax presented above

$$
\begin{array}{lcll}
M & ::= & f \mid x \mid \lambda P{:}\Delta.N \mid (M\,N) & \text{Terms} \\
\sigma & ::= & \iota \mid \sigma \to \sigma & \text{Types} \\
\Sigma & ::= & \emptyset \mid \Sigma, f{:}\sigma & \text{Type Signatures} \\
\Gamma, \Delta & ::= & \emptyset \mid \Delta, x{:}\sigma & \text{Type Contexts}
\end{array}
$$

Reduction Semantics

$$\theta \quad = \quad Sol(P \ll_{\emptyset} N)$$

$$(\lambda P{:}\Delta.M)\, N \quad \longrightarrow_{\beta} \quad M\theta$$

As usual we can define a many step reduction $\longrightarrow\!\!\!\!\!\rightarrow_{\beta}$ and the congruence relation $=_{\beta}$

NB: We can define $\longrightarrow_{\beta}$, $\longrightarrow\!\!\!\!\!\rightarrow_{\beta}$, and $=_{\beta}$ also on substitutions θ

NB. We let $Dom(\Delta) = FV(P)$ (other choices: $Dom(\Delta) \subseteq FV(P)$)

The same syntactic conditions apply on typed patterns, namely:

- each free variable appears at most once (linearity condition)
- variables are not in functional position (non-activity condition)

Syntactic properties are preserved

- (Substitution) If $M \longrightarrow\!\!\!\!\!\rightarrow_\beta N$ and $\theta \longrightarrow\!\!\!\!\!\rightarrow_\beta \theta'$, then $M\theta \longrightarrow\!\!\!\!\!\rightarrow_\beta N\theta'$
- (Confluence) The relation $\longrightarrow_\beta$ is Confluent
- Both proofs are based on *parallel reduction* of Takahashi

Glance of the Signature and Type Rules

$$\frac{}{\emptyset \text{ sig}} \ (S{\cdot}Empty)$$

$$\frac{\Sigma \text{ sig} \quad f \notin Dom(\Sigma)}{\Sigma, f{:}\sigma \text{ sig}} \ (S{\cdot}Type)$$

$$\frac{\Sigma \text{ sig}}{\vdash_\Sigma \emptyset} \ (C{\cdot}Empty)$$

$$\frac{\vdash_\Sigma \Gamma \quad x \notin Dom(\Gamma)}{\vdash_\Sigma \Gamma, x{:}\sigma} \ (C{\cdot}Type)$$

Glance of the Term Rules

$$\frac{\vdash_\Sigma \Gamma \quad x{:}\sigma \in \Gamma}{\Gamma \vdash_\Sigma x : \sigma}\ (\mathit{Var})$$

$$\frac{\vdash_\Sigma \Gamma \quad f{:}\sigma \in \Sigma}{\Gamma \vdash_\Sigma f : \sigma}\ (\mathit{Const})$$

$$\frac{FV(P) = Dom(\Delta) \quad \vdash_\Sigma \Gamma, \Delta \qquad \Gamma, \Delta \vdash_\Sigma P : \sigma \qquad \Gamma, \Delta \vdash_\Sigma M : \tau}{\Gamma \vdash_\Sigma \lambda P{:}\Delta.M : \sigma \to \tau}\ (\mathit{Abs})$$

$$\frac{\Gamma \vdash_\Sigma M : \sigma \to \tau \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma M\,N : \tau}\ (\mathit{Appl})$$

Galleria of typed properties

- (Weakening) If $\Gamma \vdash_\Sigma M : \sigma$ and $\vdash_\Sigma \Gamma, \Sigma$, then $\Gamma, \Delta \vdash_\Sigma M : \sigma$
- (Substitution) If $\Gamma_1, x{:}\sigma, \Gamma_2 \vdash_\Sigma M : \tau$ and $\Gamma_2 \vdash_\Sigma N : \sigma$, then $\Gamma_1, \Gamma_2 \vdash_\Sigma M[N/x] : \tau$
- (Subject Reduction) If $\Gamma \vdash_\Sigma M : \sigma$ and $M \longrightarrow_\beta N$, then $\Gamma \vdash_\Sigma N : \sigma$
- (Strong Normalization) Let SN be the set of strongly normalizing terms. If $\Gamma \vdash_\Sigma M : \sigma$, then $M \in SN$
- All the proofs are fairly standard with the exception of the Strong Normalization that is based on a non-trivial extension of the standard Computability Argument to accommodate the presence of patterns in the syntax

Simple Typing Examples

$$\Sigma \stackrel{def}{=} \mathit{cons}{:}\sigma \to \sigma \to \sigma$$

$$\Gamma \stackrel{def}{=} v{:}\sigma, w{:}\sigma$$

$$\Delta \stackrel{def}{=} x{:}\sigma, y{:}\sigma$$

$$\Gamma \vdash_\Sigma (\lambda(\mathit{cons}\, x\, y){:}\Delta.x)\,(\mathit{cons}\, v\, w) : \sigma$$

$$\downarrow_\beta$$

$$\Gamma \vdash_\Sigma v : \sigma \text{ where } \theta \equiv [v/x; w/y] = \mathit{Sol}((\mathit{cons}\, x\, y) \ll (\mathit{cons}\, v\, w))$$

Advanced Glance: Edinburgh Logical Framework by Harper-Honsell-Plotkin

- *A framework to define logics, JSL'93* introduce a novel type discipline: DEPENDENT-TYPE THEORY
- Via the famous Curry-Howard isomorphism '80:
  $Theorems\ as\ Types\ and\ Proof\ as\ typed\ \lambda\text{-}terms$

  we can use LF as a metalanguage to encode logical systems, encoding calculi, certifying algorithms
- Dependent types look like $\Pi P{:}\Delta.\sigma$. Terms can occurs in types

$$\frac{\Gamma \vdash_\Sigma M : \Pi x{:}\sigma.\tau \quad \Gamma \vdash_\Sigma \Pi x{:}\sigma.\tau : s \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma M\,N : \tau[N/x]}\,(PLF{\cdot}Appl)$$

$$\frac{Dom(\Delta){=}FV(P) \quad \Gamma \vdash_\Sigma M : \Pi P{:}\Delta.\tau \quad \Gamma \vdash_\Sigma \Pi P{:}\Delta.\tau : s \quad \Gamma,\Delta \vdash_\Sigma P : \sigma \quad \Gamma \vdash_\Sigma{:} N : \sigma}{\Gamma \vdash_\Sigma M\,N : (\lambda P{:}\Delta.M)\,N}\,(LF{\cdot}Appl)$$

Advanced Glance: Pattern Logical Framework

$\Sigma \in \mathcal{S}$	$\Sigma ::= \emptyset \mid \Sigma, a{:}K \mid \Sigma, f{:}A$	*Signatures*
$\Gamma, \Delta \in \mathcal{C}$	$\Gamma ::= \emptyset \mid \Gamma, x{:}A$	*Contexts*
$K \in \mathcal{K}$	$K ::= \mathsf{Type} \mid \Pi P{:}\Delta.K \mid \lambda P{:}\Delta.K \mid K\ M$	*Kinds*
$A, B, C \in \mathcal{F}$	$A ::= a \mid \Pi P{:}\Delta.A \mid \lambda P{:}\Delta.A \mid A\ M$	*Families*
$M, N, Q \in \mathcal{O}$	$M ::= f \mid x \mid \lambda P{:}\Delta.M \mid M\ M$	*Objects*

Pattern Logical Framework Reduction

$$(\beta\text{-Obj}) \qquad (\lambda P{:}\Delta.M)\ N \mapsto_\beta M\theta$$

$$(\beta\text{-Fam}) \qquad (\lambda P{:}\Delta.A)\ N \mapsto_\beta A\theta$$

$$(\beta\text{-Kinds}) \qquad (\lambda P{:}\Delta.K)\ N \mapsto_\beta K\theta$$

Pattern Logical Framework Types

Object rules

$$\frac{\vdash_\Sigma \Gamma \quad x{:}A \in \Gamma}{\Gamma \vdash_\Sigma x : A}\ (\text{O·Var})$$

$$\frac{\vdash_\Sigma \Gamma \quad f{:}A \in \Sigma}{\Gamma \vdash_\Sigma f : A}\ (\text{O·Const})$$

$$\frac{\Gamma, \Delta \vdash_\Sigma P : B \quad \Gamma, \Delta \vdash_\Sigma M : A}{\Gamma \vdash_\Sigma \lambda P{:}\Delta.M : \Pi P{:}\Delta.A}\ (\text{O·Abs})$$

$$\frac{\Gamma \vdash_\Sigma M : \Pi P{:}\Delta.A \quad \Gamma, \Delta \vdash_\Sigma P : B \qquad \Gamma \vdash_\Sigma N : B}{\Gamma \vdash_\Sigma M\, N : (\lambda P{:}\Delta.A)\, N}\ (\text{O·Appl})$$

$$\frac{\Gamma \vdash_\Sigma M : A \quad \Gamma \vdash_\Sigma B : \mathsf{Type} \quad \Gamma \vdash_\Sigma A =_\beta B}{\Gamma \vdash_\Sigma M : B}\ (\text{O·Conv})$$

Pattern Logical Framework Ex: Plotkin λ_{CBV}

Syntactic Categories

o : Type

Constructors and Judgments

$! : o^2$ Lam $: \Pi f{:}[\Pi !x^o.o].\, o$ App $: o^3$ $= : o \to o \to$ Type

Axioms and Rules

$\mathsf{Eq}_{\mathsf{refl}} : \Pi x^o.\, x = x$

$\mathsf{Eq}_{\mathsf{symm}} : \Pi x^o.\, \Pi y^o.\, (x = y) \to (y = x)$

$\mathsf{Eq}_{\mathsf{trans}} : \Pi x^o.\, \Pi y^o.\, \Pi z^o.\, (x = y) \to (y = z) \to (x = z)$

$\mathsf{Eq}_{\mathsf{ctx}} : \Pi x^o.\, \Pi y^o.\, \Pi z^o.\, \Pi w^o.\, (x = y) \to (z = w) \to (\mathsf{App}\ x\ z = \mathsf{App}\ y\ w)$

$\mathsf{Betav} : \Pi f{:}[\Pi !x^o.o].\, \Pi y^o.\, \mathsf{App}\ (!(\mathsf{Lam}\ f))\ (!y) = f\ (!y)$

$\mathsf{Xiv} : \Pi f{:}[\Pi !x^o.o].\, \Pi g{:}[\Pi !x^o.o].$

$(\Pi z^o.\, f\ (!z) = g\ (!z) \to (!(\mathsf{Lam}\ f) = !(\mathsf{Lam}\ g))$

$\mathsf{Etav} : \Pi x^o.\, !(\mathsf{Lam}\ (\lambda(!y^o).\mathsf{App}\ (!x)\ (!y))) = !x$

Pattern Logical Framework Ex: S_4 Modal Logics

Propositional Connectives and Judgment

o : Type $\quad \supset : o^3 \quad \neg : o^2 \quad \Box : o^2 \quad$ True : $o \to$ Type

Propositional Axioms

$\mathsf{A}_1 \quad : \Pi\phi^o.\ \Pi\psi^o.\ \mathsf{True}\phi \supset (\psi \supset \phi)$

$\mathsf{A}_2 \quad : \Pi\phi^o.\ \Pi\psi^o.\ \Pi\theta^o.\ \mathsf{True}(\phi \supset (\psi \supset \theta)) \supset (\phi \supset \psi) \supset (\phi \supset \theta)$

$\mathsf{A}_3 \quad : \Pi\phi^o.\ \Pi\psi^o.\ \mathsf{True}(\neg\psi \supset \neg\phi) \supset ((\neg\psi \supset \phi) \supset \psi$

Modal Axioms

$\mathsf{K} \quad : \Pi\phi^o.\ \Pi\psi^o.\ \mathsf{True}\Box(\phi \supset \psi) \supset (\neg\phi \supset \neg\psi)$

$4 \quad : \Pi\phi^o.\ \mathsf{True}\Box\phi \supset \Box\Box\phi$

$\top \quad : \Pi\phi^o.\ \mathsf{True}\Box\phi \supset \phi$

Rules

$\mathsf{MP} \quad : \Pi\phi^o.\ \Pi\psi^o.\ \mathsf{True}\phi \supset \mathsf{True}\phi \supset \psi \to \mathsf{True}\psi$

$\mathsf{NEC} : \Pi\phi^o.\ \Pi x_\emptyset{:}\mathsf{True}\phi.\ \mathsf{True}\Box\phi$

Incomplete Bibliography

G. Barthe, H. Cirstea, C. Kirchner, and LL.
Pure Pattern Type Systems.
In *Proc. of POPL*, 2003.

F. Honsell, M. Lenisa, and LL.
A Framework for Defining Logical Frameworks.
In *Computation, Meaning and Logic. Articles dedicated to Gordon Plotkin. Electronic Notes in Theoretical Computer Science*, 2007.

H. Cirstea, C. Kirchner, and LL.
Matching Power.
In *Proc. of RTA*, 2001.

H. Cirstea, C. Kirchner, and LL.
The Rho Cube.
In *Proc. of FOSSACS*, 2001.

Incomplete Bibliography

D. J. Dougherty.
Adding Algebraic Rewriting to the Untyped Lambda Calculus.
Information and Computation, 1992.

M. Takahashi.
Parallel Reductions in λ-calculus.
Journal of Symbolic Computation, 1989.

V. van Oostrom.
Lambda Calculus with Patterns.
Technical Report IR-228, Faculteit der Wiskunde en Informatica, Vrije Universiteit Amsterdam, 1990.

Incomplete Bibliography


Val Tannen, Delia Kesner, Laurence Puel.
A Typed Pattern Calculus.
In *Proc. of LICS*, 1993.


C. Barry Jay, Delia Kesner:
Pure Pattern Calculus.
In *Proc. of ESOP*, 2006.


C. Barry Jay, Delia Kesner:
First-class patterns.
J. Funct. Program. 2009.

Pattern Matching in Programming Languages

- Pattern Matching occurs in many languages and paradigms: functional, imperative, logic, and OO
- Announcement: Feb 2021: Python adopt Pattern Matching
- PEP 622 – Structural Pattern Matching
- PEP 634 – Structural Pattern Matching: Specification
- PEP 635 – Structural Pattern Matching: Motivation and Rationale
- PEP 636 – Structural Pattern Matching: Tutorial