Grokking the Sequent Calculus

Functional Pearl

David Binder, Marco Tzschentke, Marius Müller, Klaus Ostermann

University of Tübingen

	Natural Deduction	Sequent Calculus
Terms	$t := x \mid \lambda x\,.\,t \mid t\ t$	$p := x \mid \textbf{cocase}\{\mathsf{ap}(x,\alpha) \Rightarrow s\} \mid \mu\alpha\,.\,s$ $c := \alpha \mid \mathsf{ap}(p,c) \mid \tilde{\mu}x\,.\,s$ $s := \langle p \mid c \rangle$
Logic	$\dfrac{\begin{matrix}[\phi]^1 \\ \mathcal{D} \\ \psi\end{matrix}}{\phi \to \psi} : 1$ $\qquad \dfrac{\phi \to \psi \qquad \phi}{\psi}$	$\dfrac{\Gamma, \phi \vdash \psi, \Delta}{\Gamma \vdash \phi \to \psi, \Delta}$ $\qquad \dfrac{\Gamma_1 \vdash \phi, \Delta_1 \qquad \Gamma_2, \psi \vdash \Delta_2}{\Gamma_1, \Gamma_2, \phi \to \psi \vdash \Delta_1, \Delta_2}$

$\mu\tilde{\mu}$-calculus (Curien & Herbelin, 2000)

Fun

Core

$t ::= x \mid \lambda x \,.\, t \mid t\ t$

compile

$p ::= x \mid \textbf{cocase}\{\mathsf{ap}(x, \alpha) \Rightarrow s\} \mid \mu\alpha \,.\, s$

$c ::= \alpha \mid \mathsf{ap}(p, c) \mid \tilde{\mu}x \,.\, s$

$s ::= \langle p \mid c \rangle$

- \+ Let Bindings
- \+ Data & Codata Types
- \+ Control Effects
- \+ Arithmetic Primitives
- \+ Toplevel Definitions

Core

$$p := x \mid \textbf{cocase}\{\mathsf{ap}(x, \alpha) \Rightarrow s\} \mid \mu\alpha\,.\,s$$

Producer / Proof

$$c := \alpha \mid \mathsf{ap}(p, c) \mid \tilde{\mu}x\,.\,s$$

Consumer / Refutation / Continuation

$$s := \langle p \mid c \rangle$$

Statement / Command / Contradiction

1. Arithmetic Expressions
2. Let Bindings
3. Control Effects

1. Arithmetic Expressions

Fun

Core

$$t := x \mid \ulcorner n \urcorner \mid \mathbf{ifz}(t, t, t) \xrightarrow{\text{compile}}$$

$$p := x \mid \ulcorner n \urcorner \mid \mu\alpha \,.\, s$$

$$c := \alpha$$

$$s := \mathbf{ifz}(p; s, s) \mid \langle p \mid c \rangle$$

$$[\![x]\!] := x$$

Terms are translated to producers!

$$[\![\ulcorner n \urcorner]\!] := \ulcorner n \urcorner$$

$$[\![\mathbf{ifz}(t_1, t_2, t_3)]\!] := \mu\alpha.\mathbf{ifz}([\![t_1]\!], \langle [\![t_2]\!] \mid \alpha \rangle, \langle [\![t_3]\!] \mid \alpha \rangle)$$

2. Let Bindings

2. Let Bindings

Fun

$t := \ldots \mid \textbf{let}\ x = t\ \textbf{in}\ t$

compile

Core

$p := \ldots$

$c := \ldots \mid \tilde{\mu}x\,.\,s$ Exactly dual to μ

$s := \ldots$

$$[\![\textbf{let}\ x = t_1\ \textbf{in}\ t_2]\!] := \mu\alpha.\langle [\![t_1]\!] \mid \tilde{\mu}x.\langle [\![t_2]\!] \mid \alpha\rangle\rangle \quad (\alpha\ \text{fresh})$$

3. Control Effects

3. Control Effects

Fun

$$t := \ldots \mid \textbf{label}\ \alpha\ \{t\} \mid \textbf{goto}(t; \alpha)$$

Similar to call/cc & let/cc

$$\xrightarrow{\text{compile}}$$

Core

$$p := \ldots$$

$$c := \ldots$$

$$s := \ldots$$

Nothing new!

$$[\![\textbf{label}\ \alpha\ \{t\}]\!] := \mu\alpha.\langle [\![t]\!] \mid \alpha \rangle$$

Capture the continuation α!

$$[\![\textbf{goto}(t; \alpha)]\!] := \mu\beta.\langle [\![t]\!] \mid \alpha \rangle \quad (\beta \text{ fresh})$$

Throw away continuation β!

Example

$$\textbf{def}\ \mathrm{mult}(l) \coloneqq \textbf{case}\ l\ \textbf{of}\ \{\, \mathrm{Nil} \Rightarrow 1, \mathrm{Cons}(x, xs) \Rightarrow x * \mathrm{mult}(xs) \,\}$$

$$\textbf{def } \mathrm{mult}(l) \coloneqq \textbf{case } l \textbf{ of } \{\, \mathrm{Nil} \Rightarrow 1, \mathrm{Cons}(x, xs) \Rightarrow x * \mathrm{mult}(xs) \,\}$$

Add label here

$$\textbf{def } \mathrm{mult}(l) \coloneqq \textbf{label } \alpha \,\{\, \mathrm{mult'}(l; \alpha) \,\}$$
$$\textbf{def } \mathrm{mult'}(l; \alpha) \coloneqq \textbf{case } l \textbf{ of } \{\, \mathrm{Nil} \Rightarrow 1, \mathrm{Cons}(x, xs) \Rightarrow \textbf{ifz}(x, \textbf{goto}(0; \alpha), x * \mathrm{mult'}(xs; \alpha)) \,\}$$

Jump with 0 to the label here

$$\textbf{def}\ \mathrm{mult}(l) := \textbf{label}\ \alpha\ \{\ \mathrm{mult'}(l; \alpha)\ \}$$

$$\textbf{def}\ \mathrm{mult'}(l; \alpha) := \textbf{case}\ l\ \textbf{of}\ \{\ \mathrm{Nil} \Rightarrow 1, \mathrm{Cons}(x, xs) \Rightarrow \textbf{ifz}(x, \textbf{goto}(0; \alpha), x * \mathrm{mult'}(xs; \alpha))\ \}$$

Continuation argument

$$\textbf{def}\ \mathrm{mult}(l; \alpha) := \mathrm{mult'}(l; \alpha, \alpha)$$

Short-circuiting continuation: α

Normal function return: β

$$\textbf{def}\ \mathrm{mult'}(l; \alpha, \beta) :=$$

$$\langle l \mid \textbf{case}\ \{\mathrm{Nil} \Rightarrow \langle 1 \mid \beta \rangle, \mathrm{Cons}(x, xs) \Rightarrow \textbf{ifz}(x, \langle 0 \mid \alpha \rangle, \mathrm{mult'}(xs; \alpha, \tilde{\mu} z. * (x, z; \beta)))\}\rangle$$

Jump to α with 0

Recursive call

What else will you find?

Typing Rules

$$\frac{x :^{\text{prd}} \tau \in \Gamma}{\Gamma \vdash x :^{\text{prd}} \tau}\ \textsc{Var}_1 \qquad \frac{\alpha :^{\text{cns}} \tau \in \Gamma}{\Gamma \vdash \alpha :^{\text{cns}} \tau}\ \textsc{Var}_2$$

Producer Typing

Consumer Typing

$$\frac{\Gamma, \alpha :^{\text{cns}} \tau \vdash s}{\Gamma \vdash \mu\alpha.s :^{\text{prd}} \tau}\ \mu \qquad \frac{\Gamma, x :^{\text{prd}} \tau \vdash s}{\Gamma \vdash \tilde{\mu}x.s :^{\text{cns}} \tau}\ \tilde{\mu}$$

$$\frac{\Gamma \vdash p :^{\text{prd}} \tau \qquad \Gamma \vdash c :^{\text{cns}} \tau}{\Gamma \vdash \langle p \mid c \rangle}\ \textsc{Cut}$$

Statement Typing

Operational Semantics

From statement to statement

$$\mathbf{ifz}(\ulcorner 0 \urcorner, s_1, s_2) \triangleright s_1$$

$$\mathbf{ifz}(\ulcorner n \urcorner, s_1, s_2) \triangleright s_2 \quad (\text{if } n \neq 0)$$

What about nested computation / congruence?

$$\mathcal{F}(\mathbf{ifz}(p, s_1, s_2)) \quad := \quad \langle \mathcal{F}(p) \mid \tilde{\mu} x.\mathbf{ifz}(x, s_1, s_2) \rangle \qquad (p \text{ not a value}), (x \text{ fresh})$$

Focusing to explicitly sequentialize computation

Approximately like an ANF-transformation

No need for evaluation contexts

Insights!

- Let-Bindings and Control Operators are dual
- Pattern matching on Data and copattern matching on Codata are dual **Bonus Slide**
- Commutative conversions (case-of-case) are μ-reductions **Bonus Slide**
- Duality of call-by-value and call-by-name

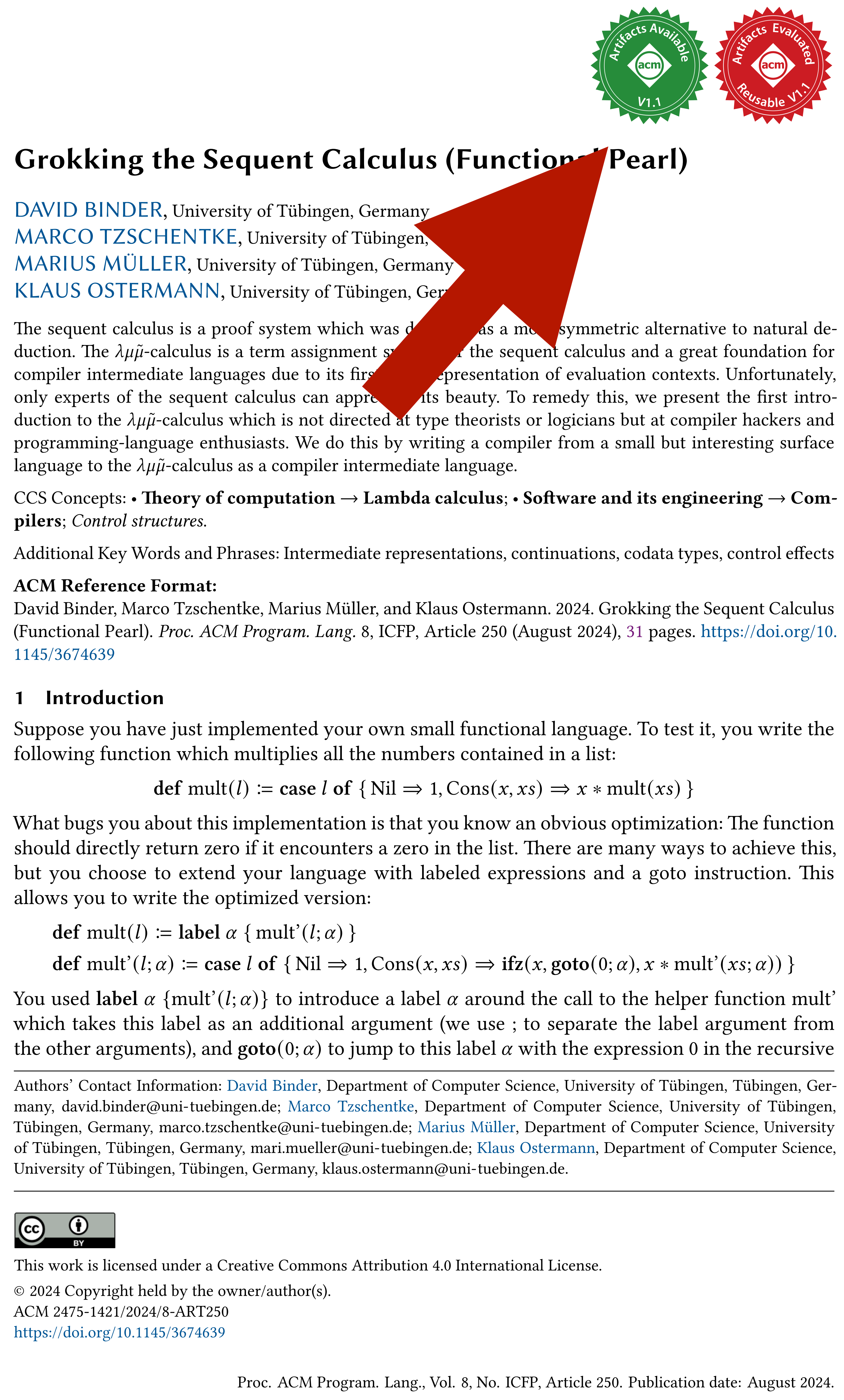
One Takeaway!

Expressive Power of CPS without Complicated CPS-Types

Thank you for listening

- Marco Tzschentke
- Marius Müller
- Klaus Ostermann

grokking-sc.github.io/grokking-sc/



Artifacts Available V1.1 · Artifacts Evaluated Reusable V1.1

Grokking the Sequent Calculus (Function Pearl)

DAVID BINDER, University of Tübingen, Germany
MARCO TZSCHENTKE, University of Tübingen,
MARIUS MÜLLER, University of Tübingen, Germany
KLAUS OSTERMANN, University of Tübingen, Ger

The sequent calculus is a proof system which was d as a mo symmetric alternative to natural deduction. The $\lambda\mu\tilde{\mu}$-calculus is a term assignment s the sequent calculus and a great foundation for compiler intermediate languages due to its firs epresentation of evaluation contexts. Unfortunately, only experts of the sequent calculus can appre its beauty. To remedy this, we present the first introduction to the $\lambda\mu\tilde{\mu}$-calculus which is not directed at type theorists or logicians but at compiler hackers and programming-language enthusiasts. We do this by writing a compiler from a small but interesting surface language to the $\lambda\mu\tilde{\mu}$-calculus as a compiler intermediate language.

CCS Concepts: • **Theory of computation** → **Lambda calculus**; • **Software and its engineering** → **Compilers**; *Control structures.*

Additional Key Words and Phrases: Intermediate representations, continuations, codata types, control effects

ACM Reference Format:
David Binder, Marco Tzschentke, Marius Müller, and Klaus Ostermann. 2024. Grokking the Sequent Calculus (Functional Pearl). *Proc. ACM Program. Lang.* 8, ICFP, Article 250 (August 2024), 31 pages. https://doi.org/10.1145/3674639

1 Introduction

Suppose you have just implemented your own small functional language. To test it, you write the following function which multiplies all the numbers contained in a list:

$$\textbf{def}\ \mathrm{mult}(l) \coloneqq \textbf{case}\ l\ \textbf{of}\ \{\ \mathrm{Nil} \Rightarrow 1, \mathrm{Cons}(x, xs) \Rightarrow x * \mathrm{mult}(xs)\ \}$$

What bugs you about this implementation is that you know an obvious optimization: The function should directly return zero if it encounters a zero in the list. There are many ways to achieve this, but you choose to extend your language with labeled expressions and a goto instruction. This allows you to write the optimized version:

$$\textbf{def}\ \mathrm{mult}(l) \coloneqq \textbf{label}\ \alpha\ \{\ \mathrm{mult'}(l;\alpha)\ \}$$
$$\textbf{def}\ \mathrm{mult'}(l;\alpha) \coloneqq \textbf{case}\ l\ \textbf{of}\ \{\ \mathrm{Nil} \Rightarrow 1, \mathrm{Cons}(x, xs) \Rightarrow \textbf{ifz}(x, \textbf{goto}(0;\alpha), x * \mathrm{mult'}(xs;\alpha))\ \}$$

You used **label** α $\{\mathrm{mult'}(l;\alpha)\}$ to introduce a label α around the call to the helper function mult' which takes this label as an additional argument (we use ; to separate the label argument from the other arguments), and **goto**$(0;\alpha)$ to jump to this label α with the expression 0 in the recursive

Authors' Contact Information: David Binder, Department of Computer Science, University of Tübingen, Tübingen, Germany, david.binder@uni-tuebingen.de; Marco Tzschentke, Department of Computer Science, University of Tübingen, Tübingen, Germany, marco.tzschentke@uni-tuebingen.de; Marius Müller, Department of Computer Science, University of Tübingen, Tübingen, Germany, mari.mueller@uni-tuebingen.de; Klaus Ostermann, Department of Computer Science, University of Tübingen, Tübingen, Germany, klaus.ostermann@uni-tuebingen.de.

This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2024 Copyright held by the owner/author(s).
ACM 2475-1421/2024/8-ART250
https://doi.org/10.1145/3674639

Proc. ACM Program. Lang., Vol. 8, No. ICFP, Article 250. Publication date: August 2024.

Bonus Slides

Commutative Conversions

Case-of-case

$$\mathbf{case}\ (\mathbf{case}\ e_1\ \mathbf{of}\ \{\mathtt{T} \Rightarrow e_2; \mathtt{F} \Rightarrow e_3\})\ \mathbf{of}\ \{\mathtt{T} \Rightarrow e_4; \mathtt{F} \Rightarrow e_5\}$$

$$\mathbf{case}\ e_1\ \mathbf{of}\ \{\mathtt{T} \Rightarrow \mathbf{case}\ e_2\ \mathbf{of}\ \{\mathtt{T} \Rightarrow e_4; \mathtt{F} \Rightarrow e_5\}; \mathtt{F} \Rightarrow \mathbf{case}\ e_3\ \mathbf{of}\ \{\mathtt{T} \Rightarrow e_4; \mathtt{F} \Rightarrow e_5\}\}$$

$$\mu\alpha.\underline{\langle \mu\beta.\langle [\![e_1]\!] \mid \mathbf{case}\ \{\mathtt{T} \Rightarrow \langle [\![e_2]\!] \mid \beta\rangle; \mathtt{F} \Rightarrow \langle e_3 \mid \beta\rangle\}\rangle \mid \mathbf{case}\ \{\mathtt{T} \Rightarrow \langle [\![e_4]\!] \mid \alpha\rangle, \mathtt{F} \Rightarrow \langle [\![e_5]\!] \mid \alpha\rangle\}\rangle}$$

$$\begin{aligned}
&\mu\alpha.\langle [\![e_1]\!] | \mathbf{case}\ \{\\
&\qquad \mathtt{T} \Rightarrow \underline{\langle \mu\beta.\langle [\![e_2]\!] \mid \mathbf{case}\ \{\mathtt{T} \Rightarrow \langle [\![e_4]\!] \mid \beta\rangle, \mathtt{F} \Rightarrow \langle [\![e_5]\!] \mid \beta\rangle\}\rangle \mid \alpha\rangle}\\
&\qquad \mathtt{F} \Rightarrow \underline{\langle \mu\beta.\langle [\![e_3]\!] \mid \mathbf{case}\ \{\mathtt{T} \Rightarrow \langle [\![e_4]\!] \mid \beta\rangle, \mathtt{F} \Rightarrow \langle [\![e_5]\!] \mid \beta\rangle\}\rangle \mid \alpha\rangle}\}\rangle
\end{aligned}$$

Duality of Data and Codata Types

Definition 2.5 (Algebraic Data and Codata Types).

Fun

$$
\begin{aligned}
t &::= \ldots \mid K(\overline{t}) \mid \textbf{case}\ t\ \textbf{of}\ \{\overline{K(\overline{x}) \Rightarrow t}\} \\
&\mid t.D(\overline{t}) \mid \textbf{cocase}\ \{\overline{D(\overline{x}) \Rightarrow t}\} \\
\mathfrak{t} &::= \ldots \mid K(\overline{\mathfrak{t}}) \mid \textbf{cocase}\ \{\overline{D(\overline{x}) \Rightarrow t}\}
\end{aligned}
$$

$$\textbf{case}\ K(\overline{\mathfrak{t}})\ \textbf{of}\ \{K(\overline{x}) \Rightarrow t, \ldots\} \triangleright t[\overline{\mathfrak{t}}/\overline{x}]$$

$$\textbf{cocase}\ \{D(\overline{x}) \Rightarrow t, \ldots\}.D(\overline{\mathfrak{t}}) \triangleright t[\overline{\mathfrak{t}}/\overline{x}]$$

Core

$$
\begin{aligned}
p &::= \ldots \mid K(\overline{p};\overline{c}) \mid \textbf{cocase}\ \{\overline{D(\overline{x};\overline{\alpha}) \Rightarrow s}\} \\
c &::= \ldots \mid D(\overline{p};\overline{c}) \mid \textbf{case}\ \{\overline{K(\overline{x};\overline{\alpha}) \Rightarrow s}\} \\
\mathfrak{p} &::= \ldots \mid K(\overline{\mathfrak{p}};\overline{c}) \mid \textbf{cocase}\ \{\overline{D(\overline{x};\overline{\alpha}) \Rightarrow s}\} \\
\mathfrak{c} &::= \ldots \mid D(\overline{p};\overline{c}) \mid \textbf{case}\ \{\overline{K(\overline{x};\overline{\alpha}) \Rightarrow s}\}
\end{aligned}
$$

$$\langle K(\overline{\mathfrak{p}};\overline{\mathfrak{c}}) \mid \textbf{case}\ \{K(\overline{x};\overline{\alpha}) \Rightarrow s, \ldots\}\rangle \triangleright s[\overline{\mathfrak{p}}/\overline{x};\overline{\mathfrak{c}}/\overline{\alpha}]$$

$$\langle \textbf{cocase}\ \{D(\overline{x};\overline{\alpha}) \Rightarrow s, \ldots\} \mid D(\overline{\mathfrak{p}};\overline{\mathfrak{c}})\rangle \triangleright s[\overline{\mathfrak{p}}/\overline{x};\overline{\mathfrak{c}}/\overline{\alpha}]$$

$$
\begin{aligned}
\llbracket K(t_1, \ldots, t_n) \rrbracket &\coloneqq K(\llbracket t_1 \rrbracket, \ldots, \llbracket t_n \rrbracket) \\
\llbracket \textbf{case}\ t\ \textbf{of}\ \{\overline{K_i(\overline{x_{i,j}}) \Rightarrow t_i}\} \rrbracket &\coloneqq \mu\alpha.\langle \llbracket t \rrbracket \mid \textbf{case}\ \{\overline{K_i(\overline{x_{i,j}}) \Rightarrow \langle \llbracket t_i \rrbracket \mid \alpha \rangle}\}\rangle && (\alpha \text{ fresh}) \\
\llbracket t.D(t_1, \ldots, t_n) \rrbracket &\coloneqq \mu\alpha.\langle \llbracket t \rrbracket \mid D(\llbracket t_1 \rrbracket, \ldots, \llbracket t_n \rrbracket; \alpha)\rangle && (\alpha \text{ fresh}) \\
\llbracket \textbf{cocase}\ \{\overline{D_i(\overline{x_{i,j}}) \Rightarrow t_i}\} \rrbracket &\coloneqq \textbf{cocase}\ \{\overline{D_i(\overline{x_{i,j}}; \alpha_i) \Rightarrow \langle \llbracket t_i \rrbracket \mid \alpha_i \rangle}\} && (\overline{\alpha_i} \text{ fresh})
\end{aligned}
$$