David Binder, Marco Tzschentke, Marius Müller, Klaus Ostermann University of Tübingen

Grokking the Sequent Calculus Functional Pearl

Natural Deduction Sequent Calculus

Terms $t := x \mid \lambda x \cdot t \mid t \mid t$

 \hat{D} $\mu\mu$ valuulus luul len σ liel bell

Logic
\n
$$
\frac{\varphi}{\psi}
$$

\n $\frac{\psi}{\phi \to \psi} : 1$
\n ψ
\n ψ <

 $p := x \mid \textbf{cocase} \{ \textbf{ap}(x, \alpha) \Rightarrow s \} \mid \mu\alpha \text{ . } s$ $c := \alpha \mid \text{ap}(p, c) \mid \tilde{\mu}x \text{ . } s$ *s* := ⟨*p* ∣ *c*⟩ *D* $s := (p \mid$ *D* $C \cdot S$! \boldsymbol{J} *μμ* ˜-calculus (Curien & Herbelin, 2000)

$$
\frac{\Gamma, \phi \vdash \psi, \Delta}{\Gamma \vdash \phi \rightarrow \psi, \Delta} \qquad \frac{\Gamma_1 \vdash \phi, \Delta_1 \qquad \Gamma_2, \psi \vdash \Delta_2}{\Gamma_1, \Gamma_2, \phi \rightarrow \psi \vdash \Delta_1, \Delta_2}
$$

Fun Core

 $p := x \mid \text{cocase} \{ \text{ap}(x, \alpha) \Rightarrow s \} \mid \mu \alpha \cdot s$ $c := \alpha | \text{ ap}(p, c) | \tilde{\mu}x$.*s* $s := \langle p | c \rangle$

compile

 $t := x \mid \lambda x \cdot t \mid t \mid t$

+ Let Bindings

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

- $p := x \mid \text{cocase} \{ \text{ap}(x, \alpha) \Rightarrow s \} \mid \mu \alpha$.*s*
	- Producer / Proof
- $c := \alpha | \text{ ap}(p, c) | \tilde{\mu}x$.*s*
	- Consumer / Refutation / Continuation

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

Core

-
-

Statement / Command / Contradiction

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

Fun Core

 $t := x \mid \lceil n \rceil \mid \textbf{if} z(t, t, t)$ *p* := *x* | $\lceil n \rceil \mid \mu \alpha$.*s c* := *α s* := **ifz**(*p*;*s*,*s*) ∣ ⟨*p* ∣ *c*⟩ compile $compile$ $\llbracket x \rrbracket := x$ Ferms are translated to producers! $\llbracket \ulcorner n \urcorner \rrbracket \coloneqq \ulcorner n \urcorner$ $\begin{array}{ccc} & & \cdot & & \cdot \\ & & \cdot & & \cdot \\ & & & \cdot & \cdot \\ & & & & \cdot & \cdot \end{array}$ $\alpha := \alpha$
 $\alpha := \frac{1}{2} \int_{-\infty}^{\infty} f(x) \, dx \, dx$ $s := \mathbf{IZ}(p; s, s) \mid \langle p \mid c \rangle$

\mathbf{r} ! | "# | ⊙ | ifz(,,)

-
-
- $\[\begin{bmatrix} \textbf{ifz}(t_1, t_2, t_3) \end{bmatrix} := \mu \alpha.\textbf{ifz}(\begin{bmatrix} t_1 \end{bmatrix}, \langle \begin{bmatrix} t_2 \end{bmatrix} \mid \alpha \rangle, \langle \begin{bmatrix} t_3 \end{bmatrix} \mid \alpha \rangle)\]$

,, ,... ∈ VaRiables ⋆, , ,,... ∈ CovaRiables ⊙ ∈ {∗, +, −} \blacksquare ,, ,... ∈ VaRiables ⋆, , ,,... ∈ CovaRiables ⊙ ∈ {∗, +, −} 1. Arithmetic Expressions

2. Let Bindings

2. Let Bindings Defnition 2.3 (Let-Bindings and ˜-abstractions).

Fun

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

3. Control Effects

3. Control Effects

$[\![\text{label }\alpha \{t\}]] \coloneqq \mu\alpha.\langle [\![t]\!] \mid \alpha \rangle$ Capture the continuation $\alpha!$ $\{ \pmb{\text{goto}}(t; \alpha) \} \coloneqq \mu \beta. \langle \lfloor t \rfloor \mid \alpha \rangle \quad (\beta \text{ fresh})$ intow away continuation provides a location to which and the set of which are β $[\textsf{goto}(t;\alpha)] \coloneqq \mu\beta.\langle \llbracket t \rrbracket \mid \alpha \rangle \quad (\beta \text{ fresh})$ Throw away continuation $\beta!$

$t := ...$ | **label** $\alpha \{t\}$ | **goto**(t ; α) *p* := Defnition 2.7 (Control Operators). 250:12 David Binder, Marco Tzschentke, Marius Müller, and Klaus Ostermann

Example

def mult(l) $:=$ case l of {Nil \Rightarrow 1, Cons(x, xs) \Rightarrow x * mult(xs) }

but you choose to extend your label with labeled expressions and a goto instructions and a goto instruction. The contract of t der mult(*l*) $:=$ label $\alpha \in$ mult $(l; \alpha)$ $\}$ $\frac{1}{\sqrt{2}} \frac{1}{\sqrt{2}} \left(\frac{1}{\sqrt{2}} \right) \right) \right) \right) \right) - \frac{1}{\sqrt{2}} \right) \right)}{\sqrt{2}}$ $Jump$ with 0 to the label here $\hbox{---}$ def mult(l) $:=$ label $\alpha \{ \text{mult}'(l; \alpha) \}$ def mult'(l; α) := case l of { Nil \Rightarrow 1, Cons(x, xs) \Rightarrow ifz(x, goto(0; α), x * mult'(xs; α)) } J_l Jump with 0 to the label here Jump with 0 to the label here

should directly return zero if it encounters a zero in the list. Tere are many ways to achieve this, Add label here

which takes the label as an additional as an additional argument (we use ; to see also separate the label argument from α

def mult(l) $:=$ case l of {Nil \Rightarrow 1, Cons(x, xs) \Rightarrow x * mult(xs) }

allows you to write the optimized version:

def mult(l) $:=$ label $\alpha \{ \text{mult}'(l; \alpha) \}$ def mult'($l; \alpha$) = case l of {Nil \Rightarrow 1, Cons(x, xs) \Rightarrow ifz(x , goto(0; α), $x *$ mult'($xs; \alpha$)) } helper function. But since $\mathcal{L}(\mathbf{z})$ and $\mathcal{L}(\mathbf{z})$ and $\mathcal{L}(\mathbf{z})$ now has control effects, you need to reconsider how to reconsider to reconsider $\mathcal{L}(\mathbf{z})$ **def** mult(*l*) $:=$ label $\alpha \in$ mult $(i; \alpha)$ $\}$

which takes this label as an additional argument (we use ; to separate the label argument from def $mult(l; \alpha) := mult'(l; \alpha, \alpha)$ Continuation aroumont def mult($l; \alpha$) $:=$ mult'($l; \alpha, \alpha$) Continuation argument

 $\det \text{mult}'(l; \alpha, \beta) := \text{Partivity continuous.}$ a Nurversity of Tübingen, Tübingen, Tübingen, Tübingen, P $\ell l \log \rho$ (Nil \rightarrow ℓ 1 R) Cons(x xs) \rightarrow if α (x lo α) mult (xs: α iiz \ast (x z: R))))) $T_{\text{max}} = \frac{1 - 1}{\sqrt{2}}$ S_{UWAN} to α with 0 U_{maxima} $\langle l \mid \text{case } \{ Nil \Rightarrow \langle 1 \mid \beta \rangle, \text{Cons}(x, xs) \Rightarrow \text{ifz}(x, \langle 0 \mid \alpha \rangle, \text{mult}'(xs; \alpha, \tilde{\mu}z. * (x, z; \beta))) \} \rangle$ Jump to α with 0 the argument indicates $\mathsf{Recursively}$ call Short-circuiting continuation: α Normal function return: β Recursive cal

What else will you find?

$$
\frac{p \cdot^{pred} \tau}{\Gamma \vdash \langle p \mid c \rangle} \frac{\Gamma \vdash c \cdot^{cns} \tau}{\Gamma \vdash \langle p \mid c \rangle}
$$

$x : \text{prd } \tau \in \Gamma$	VAR_1	$\alpha : \text{cns } \tau \in \Gamma$	VAR_2
Product Typing	$\frac{\Gamma, \alpha : \text{cns } \tau \vdash s}{\Gamma \vdash \mu \alpha.s : \text{prd } \tau} \mu$	$\frac{\Gamma, x : \text{prd } \tau \vdash s}{\Gamma \vdash \tilde{\mu} x.s : \text{rns } \tau} \tilde{\mu}$	Consumer Typing

Operational Semantics Dperational Semantions Example 1988 $ifz("0", s_1, s_2) > s_1$ ifz($\lceil n \rceil, s_1, s_2 \rceil \rceil s_2$ (if $n \neq 0$) What about nested computation / congruence? $\mathcal{F}(ifz(p, s_1, s_2)) \coloneqq \langle \mathcal{F}(p) | \tilde{\mu}x. ifz(x, s_1, s_2) \rangle$ (*p* not a value), (*x* fresh) Tugusling to explicitly sequeritalize contractors in the language ℓ cappi values play into an calue in answer. of the terms that we have a variable. And since we have both variable. And since we have both variables which .⟨ |F((, , ;))⟩⟩ (not a value) From statement to statement Γ Γ Γ Γ $\mathbf{H} \mathbf{Z}$ (n, s_1, s_2) $\triangleright s_2$ ($\mathbf{H} \mathbf{n} \neq$ $F_{\rm c}$ (\bigcup) F (12) QDUU LIESTEU GUINPUTATIUM $\mathcal{F}(\mathcal{G}) = \mathcal{F}(\mathcal{G})$ **Example 10 Focusing to explicitly sequentialize computation** Approximately like an ANF-transformation T_{V} is the focusing transformation described in D is not ideal since it can lot of a lot of ad-What about nested computation / congruence? 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 u-reductions **Bonus Slide**
- Duality of call-by-value and call-by-name

Expressive Power of CPS without Complicated CPS-Types

- Marco Tzschentke
- Marius Müller
- Klaus Ostermann

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

Thank you for listening

The sequent calculus is a proof system which was d as a more symmetric alternative to natural deduction. The $\lambda \mu \tilde{\mu}$ -calculus is a term assignment system for the sequent calculus and a great foundation for compiler intermediate languages due to its first-class representation of evaluation contexts. Unfortunately, only experts of the sequent calculus can appreciate 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 \rightarrow Lambda calculus; • Software and its engineering \rightarrow Compilers; Control structures.

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:

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

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

You used label α {mult'(l; α)} 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 $\text{goto}(0; \alpha)$ to jump to this label α with the expression 0 in the recursive

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

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:

def mult(l) $:=$ case l of { Nil \Rightarrow 1, Cons(x, xs) \Rightarrow x $*$ mult(xs) }

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.

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

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

Grokking the Sequent Calculus (Function Pearl)

Bonus Slides

.⟨.⟨!1" | case {T ⇒ ⟨!2" | ⟩; F ⇒ ⟨³ | ⟩}⟩ | case {T ⇒ ⟨!4" | ⟩, F ⇒ ⟨!5" | ⟩}⟩ $\mu\alpha.\mu\rho.\{\text{exp}\left(1 \rightarrow \text{exp}\left(1 \rightarrow \$ $\mu\alpha.\langle \mu p. \langle \lbrack \lbrack e_1 \rbrack \rbrack$ | case { $\bot \Rightarrow \langle \lbrack \lbrack e_2 \rbrack \rbrack$ | p); $\Gamma \Rightarrow \langle e_3 \rbrack p$ | case { $\bot \Rightarrow \langle \lbrack \lbrack e_4 \rbrack \rbrack$ | α), $\Gamma \Rightarrow \langle \lbrack \lbrack e_5 \rbrack \rbrack$ | α) }

 $\mu\alpha$. $\langle \lbrack \lbrack e_1 \rbrack \rbrack$ case { $\Gamma \implies \langle \mu \beta. \langle \ell_2 \rangle \mid \text{case } \{1 \implies \langle \ell_4 \rangle \mid \beta \rangle, \Gamma \implies \langle \ell_5 \rangle \mid \beta \rangle \}$ $F \implies \langle \mu p. \langle \ell_3 | \rangle | \text{ case } \{1 \implies \langle \ell_4 | \rangle | p \rangle, F \implies \langle \ell_5 | \rangle | p \rangle \} | \alpha \rangle$ $\sqrt{1}$ $\sqrt{1}$ T ⇒ ⟨.⟨!2" | case {T ⇒ ⟨!4" | ⟩, F ⇒ ⟨!5" | ⟩}⟩ | ⟩ $\frac{\sqrt{r^2 - 11^2 - 211^2 - 12^2 - 11^2}}{r^2 - 11^2}$ $\mu\alpha$. $\langle [e_1] | \text{case } \{$ $T \Rightarrow \langle \mu \beta. \langle [\![e_2]\!] \mid \text{case } \{T \Rightarrow \langle [\![e_4]\!] \mid \beta \rangle, F \Rightarrow \langle [\![e_5]\!] \mid \beta \rangle \} \rangle \mid \alpha \rangle$ $F \Rightarrow \langle \mu \beta. \langle [\![e_3]\!] \mid \text{case } \{ \texttt{T} \Rightarrow \langle [\![e_4]\!] \mid \beta \rangle, F \Rightarrow \langle [\![e_5]\!] \mid \beta \rangle \} \rangle \mid \alpha \rangle \} \rangle$

- case (case e_1 of $\{T \Rightarrow e_2; F \Rightarrow e_3\}$) of $\{T \Rightarrow e_4; F \Rightarrow e_5\}$
- case e_1 of $\{T \Rightarrow \text{case } e_2 \text{ of } \{T \Rightarrow e_4; F \Rightarrow e_5\}; F \Rightarrow \text{case } e_3 \text{ of } \{T \Rightarrow e_4; F \Rightarrow e_5\}\}$
- $\mu\alpha.\langle \mu\beta.\langle [\![e_{1}]\!] \mid \text{case } \{T \Rightarrow \langle [\![e_{2}]\!] \mid \beta \rangle; F \Rightarrow \langle e_{3} \mid \beta \rangle \} \rangle \mid \text{case } \{T \Rightarrow \langle [\![e_{4}]\!] \mid \alpha \rangle, F \Rightarrow \langle [\![e_{5}]\!] \mid \alpha \rangle \} \rangle$

$$
\frac{| \beta \rangle, F \implies \langle [e_5] | \beta \rangle \rangle | \alpha \rangle}{| \beta \rangle, F \implies \langle [e_5] | \beta \rangle \rangle \rangle | \alpha \rangle}
$$

Commutative Conversions Case-of-case Commutative Conversions translating Maurer et al.'s example into the \sim calculus. First, let us translate the two examples the two examples the two examples the two examples the
The two examples the two udsc-ul-lasc
China **Commutative Conversions** pass. The special instance of -reductions instance of -reductions instance of -reductions by instance the point by instance of -reductions in the special instance of -reductions in the special instance of -reductions in th In the ˜-calculus, commuting conversions don't have to be implemented as a special compiler $\textbf{Case-of-case}$ for compiler writers who want to generate efcient code. Commutative Conversions dominative Case-of-case translating Maurer et al.'s example into the set al.'s example into the set al.'s example into the set al.'s e
's example into the set al.'s example in the set al. 's example in the set al. 's example in the set al.'s exa

using patern-matching syntax: case (case e₁ o

case e_1 of $\{1 \Rightarrow \text{case } e_2 \text{ of } \{1 \Rightarrow e_4; F\}$

Duality of Data and Codata Types Stream is defned by two destructors, hd yields the head of the stream and tl yields the remaining IITY OF Data and Codata Tynes the cocase {...} construct.

$$
p ::= ... | K(\overline{p}; \overline{c}) | \text{cocase } \{ \overline{D(\overline{x}; \overline{\alpha})} \Rightarrow s \}
$$

\n
$$
c ::= ... | D(\overline{p}; \overline{c}) | \text{case } \{ \overline{K(\overline{x}; \overline{\alpha})} \Rightarrow s \}
$$

\n
$$
p ::= ... | K(\overline{p}; \overline{c}) | \text{cocase } \{ \overline{D(\overline{x}; \overline{\alpha})} \Rightarrow s \}
$$

\n
$$
c ::= ... | D(\overline{p}; \overline{c}) | \text{case } \{ \overline{K(\overline{x}; \overline{\alpha})} \Rightarrow s \}
$$

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

Definition 2.5 (Algebraic Data and Codata Types).
\n**Fun**
\n
$$
t := ... | K(\overline{t}) | \csc \theta \{ \overline{K(\overline{x})} \Rightarrow t \}
$$

\n $| t.D(\overline{t}) | \csc \theta \{ \overline{D(\overline{x})} \Rightarrow t \}$
\n $| t.D(\overline{t}) | \csc \theta \{ \overline{D(\overline{x})} \Rightarrow t \}$
\n $| t ::= ... | K(\overline{t}) | \csc \theta \{ \overline{D(\overline{x})} \Rightarrow t \}$
\n $| t ::= ... | K(\overline{t}) | \csc \theta \{ \overline{D(\overline{x})} \Rightarrow t \}$
\n $| t ::= ... | D(\overline{p}; \overline{c}) | \csc \theta \{ \overline{K(\overline{x}; \overline{\alpha})} \Rightarrow s \}$
\n $| t ::= ... | D(\overline{p}; \overline{c}) | \csc \theta \{ \overline{K(\overline{x}; \overline{\alpha})} \Rightarrow s \}$
\n $| t ::= ... | D(\overline{p}; \overline{c}) | \csc \theta \{ \overline{K(\overline{x}; \overline{\alpha})} \Rightarrow s \}$
\n $| t ::= ... | D(\overline{p}; \overline{c}) | \csc \theta \{ \overline{K(\overline{x}; \overline{\alpha})} \Rightarrow s \}$
\n $| t ::= ... | D(\overline{p}; \overline{c}) | \csc \theta \{ \overline{K(\overline{x}; \overline{\alpha})} \Rightarrow s \}$
\n $| \csc \theta \{ D(\overline{x}; \overline{\alpha}) \Rightarrow t, ... \} D(\overline{t}) \triangleright t [\overline{t} / \overline{x}]$
\n $| [K(t_1, ..., t_n)]] := K([[t_1], ..., [[t_n]])$
\n $[[\cos \theta \{ D_t(\overline{x}_{i,j}) \Rightarrow t_i \}]] := \mu \alpha. ([[t]] | D([t_1], ..., [[t_n]], \alpha)) \qquad (\alpha \text{ fresh})$
\n $[[\cos \theta \{ D_t(\overline{x}_{i,j}) \Rightarrow t_i \}]] := \csc \theta \{ \overline{D_t(\overline{x}_{i,j}; \alpha_i) \Rightarrow (\| t_i \| \alpha \})}$
\n $[\cos \alpha \{ D_t(\overline{x}_{i,j}) \Rightarrow t_i$