Functional Semantics of Parsing Actions, and Left Recursion Elimination as Continuation Passing

Hayo Thielecke
University of Birmingham, UK

PPDP 2012

Introduction

Contributions

1. categorical model of parsing actions $[\![-]\!]$
2. killer app of the model:
   left-recursion elimination as continuation passing style

Outline: first vertical $[\![-]\!]$, then horizonal transformations

$$
\begin{tikzcd}
\text{Grammar} \arrow[r, "\text{Left recursion elimation}"] \arrow[d, "{[\![-]\!]}"] & \text{Transformed grammar} \arrow[d, "{[\![-]\!]\ \text{(extended)}}"] \\
\text{Meaning} \arrow[r, "\text{Continuation passing}"] & \text{Transformed meaning}
\end{tikzcd}
$$

Compare MacLane: in order to define natural transformation, had to define functor.

Some notation, following Dragon Book where possible

A, B, C, E, L	:	nonterminal symbols of a grammar
$\alpha, \beta, \gamma, \delta, \varphi, \psi$	:	strings of terminal or nonterminals
ε	:	empty string
λ	:	lambda-abstraction; not used for strings
v, w, w_1, w_2	:	strings of terminal symbols
$A :- \alpha$	:	grammar rule for replacing A by α
$X \to Y$	:	morphism from X to Y including: functions for semantics and derivations for syntax
$[\![\alpha]\!]$	:	semantics of a string α as a type or set
$[\![d]\!]$	:	semantics of a derivation d as a function

Motivation: "Idealized ANTLR"

Left recursion is a problem for LL parsers

- hand-written recursive descent parsers
- LL parser generators, like ANTLR: widely used tools, not just in functional programming

Example of direct left-recursion and its elimination

$$E \;:\!-\; 1 \tag{1}$$
$$E \;:\!-\; E - E \tag{2}$$

New symbol E' and rules:

$$E \;:\!-\; 1\; E' \tag{3}$$
$$E' \;:\!-\; -\; E\; E' \tag{4}$$
$$E' \;:\!-\; \varepsilon \tag{5}$$

What is the meaning of things like $-\; E\; E'$, and *why*?

A category of leftmost derivations ($\cong$ LL parsing)

Objects: strings α

Morphisms: leftmost derivations

Two functors on derivations d:

$d \otimes \alpha$ for strings α

$w \otimes d$ for strings w of terminal symbols; $A \otimes d$ not allowed

The functors $\otimes$ generalize premonoidal categories (see also Arrows)

$$\frac{A :\!- \alpha}{[(A, \alpha)] : A \to \alpha}$$

$$\frac{}{[\,] : \alpha \to \alpha} \qquad \frac{d_1 : \alpha \to \beta \qquad d_2 : \beta \to \gamma}{d_1 \cdot d_2 : \alpha \to \gamma}$$

$$\frac{d : \beta \to \gamma}{d \otimes \alpha : \beta\,\alpha \to \gamma\,\alpha} \qquad \frac{d : \alpha \to \beta}{w \otimes d : w\,\alpha \to w\,\beta}$$

Functional semantics of parsing actions

Parsing actions ($\cong$ input to parser generator)

- For each non-terminal symbols A there is a type $[\![A]\!]$.
  Extended to strings:

$$[\![X_1 \dots X_n]\!] \stackrel{\text{def}}{=} [\![X_1]\!] \otimes \cdots \otimes [\![X_n]\!]$$

- For each grammar rule $A :- \alpha$, there is a function of type

$$[\![A :- \alpha]\!] : [\![\alpha]\!] \longrightarrow [\![A]\!]$$

Parsing actions extended to derivations ($\cong$ runs of the parser)

- Each run of the parser generates a leftmost derivation $d : \alpha \to \beta$.
- Its semantics is a function $[\![d]\!] : [\![\beta]\!] \to [\![\alpha]\!]$.

Left-recursion elimination: syntactic transformation

Original rules for L:

$$\begin{aligned} L &\coloneq \psi_1 \\ &\vdots \\ L &\coloneq \psi_m \\ L &\coloneq L\ \varphi_1 \\ &\vdots \\ L &\coloneq L\ \varphi_n \end{aligned}$$

Transformed rules with L':

$$\begin{aligned} L &\coloneq \psi_1\ L' \\ &\vdots \\ L &\coloneq \psi_m\ L' \\ L' &\coloneq \varphi_1\ L' \\ &\vdots \\ L' &\coloneq \varphi_n\ L' \\ L' &\coloneq \varepsilon \end{aligned}$$

Semantic transformation = continuation passing

Reminder: continuation passing style transformation

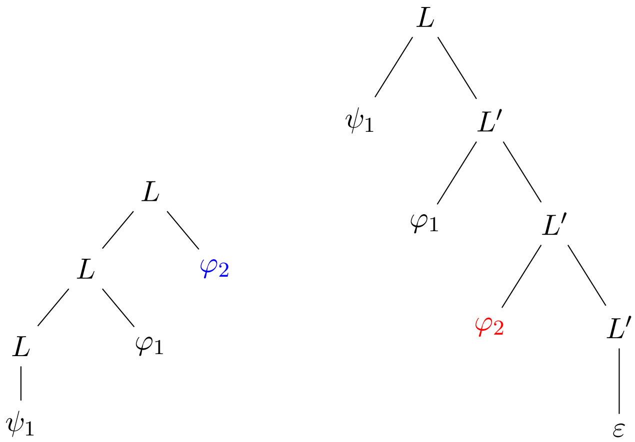
$$\begin{array}{rcl} \overline{1} & = & \lambda k.k\,1 \\ \overline{M_1 - M_2} & = & \lambda k.\overline{M_1}\,(\lambda x.\overline{M_2}(\lambda y.k(x-y))) \end{array}$$

Double negation on types: $\overline{M} : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$

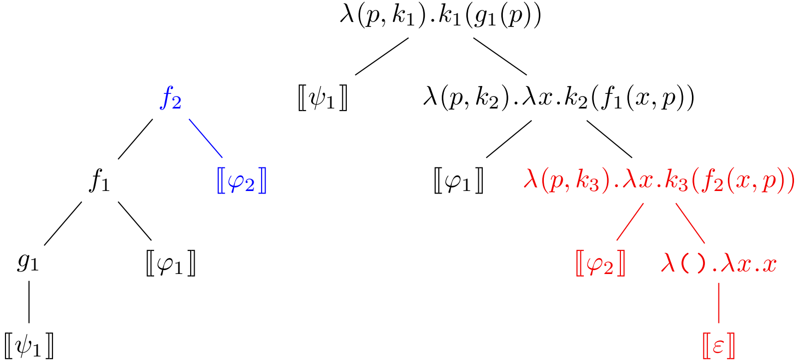
Transformation on types for left recursion elimation

$$\begin{array}{rcll} [\![L']\!] & \stackrel{\text{def}}{=} & [\![L]\!] \to [\![L]\!] & \\ L & :- & L\,\varphi_i & \text{original rule} \\ L' & :- & \varphi_i\,L' & \text{transformed rule} \\ ([\![L]\!] \to [\![L]\!]) & \leftarrow & ([\![\varphi_i]\!] \otimes ([\![L]\!] \to [\![L]\!])) & \end{array}$$

Example: parse tree for $\psi_1\ \varphi_1\ \varphi_2$ is turned inside out



Semantic actions for original and transformed parse trees



Linearly used continuations: $\lambda k. \ldots k(\ldots)$

Main result and proof technique

Simulation theorem

For each leftmost derivation in the transformed grammar

$$d : \alpha \to \beta \text{ where } L' \text{ does not occur in } \alpha \text{ or } \beta$$

there is a derivation $d^{\triangle}$ in the original grammar such that

$$[\![d^{\triangle}]\!] = [\![d]\!]$$

Proof technique by diagram chase

- The proof uses the contravariant functor $[\![-]\!]$ preserving $\otimes$.
- Need induction hypothesis: continuation k
- Direct style: compose with k
- Continuation passing style: pass $\lambda\, k$ as an argument

Simulation for a $L :\!- \psi_j L'$ step by a $L :\!- \psi_j$ step

$$L \xrightarrow{L \,:\!-\, \psi_j \, L'} \psi_j \otimes L' \xrightarrow{d_1 \;\otimes\; L'} w \otimes L'$$

Simulation for a $L :\!-\ \psi_j L'$ step by a $L :\!-\ \psi_j$ step

$$L \xrightarrow{L \,:\!-\, \psi_j\, L'} \psi_j \otimes L' \xrightarrow{d_1 \ \otimes\ L'} w \otimes L'$$

$$L \otimes \alpha \xrightarrow{L \,:\!-\, \psi_j \ \otimes\ \alpha} \psi_j \otimes \alpha \xrightarrow{d_1^{\triangle} \ \otimes\ \alpha} w \otimes \alpha$$

Simulation for a $L :- \psi_j L'$ step by a $L :- \psi_j$ step

$$\begin{tikzcd}[column sep=huge] [\![L]\!] & [\![\psi_j]\!] \otimes [\![L']\!] \arrow[l, "{[\![L :- \psi_j\, L']\!]}"'] & [\![w]\!] \otimes [\![L']\!] \arrow[l, "{[\![d_1]\!] \otimes [\![L']\!]}"'] \end{tikzcd}$$

$$\begin{tikzcd}[column sep=huge] [\![L]\!] \otimes [\![\alpha]\!] & [\![\psi_j]\!] \otimes [\![\alpha]\!] \arrow[l, "{[\![L :- \psi_j]\!] \otimes [\![\alpha]\!]}"'] & [\![w]\!] \otimes [\![\alpha]\!] \arrow[l, "{[\![d_1^{\triangle}]\!] \otimes [\![\alpha]\!]}"'] \end{tikzcd}$$

Simulation for a $L \mathrel{:-} \psi_j L'$ step by a $L \mathrel{:-} \psi_j$ step

$$
\begin{tikzcd}
\llbracket L \rrbracket & \llbracket \psi_j \rrbracket \otimes \llbracket L' \rrbracket \arrow[l, "\llbracket L \mathrel{:-} \psi_j\, L' \rrbracket"'] & \llbracket w \rrbracket \otimes \llbracket L' \rrbracket \arrow[l, "\llbracket d_1 \rrbracket \otimes \llbracket L' \rrbracket"'] \\
\llbracket L \rrbracket \otimes \llbracket \alpha \rrbracket \arrow[u, "k"'] & \llbracket \psi_j \rrbracket \otimes \llbracket \alpha \rrbracket \arrow[l, "\llbracket L \mathrel{:-} \psi_j \rrbracket \otimes \llbracket \alpha \rrbracket"'] \arrow[u, "\llbracket \psi_j \rrbracket \otimes \lambda k"'] & \llbracket w \rrbracket \otimes \llbracket \alpha \rrbracket \arrow[l, "\llbracket d_1^{\triangle} \rrbracket \otimes \llbracket \alpha \rrbracket"'] \arrow[u, "\llbracket w \rrbracket \otimes \lambda k"']
\end{tikzcd}
$$

Related work

- Attribute motion for left-recursion elimination (Lohmann, Riedewald & Stoy)
- Parsing combinators in Haskell and transformations (Swierstra et al)
- Premonoidal categories and Freyd categories (Power & Robinson; Power & Thielecke)
- Linearly used continuations $(A \to \mathbf{Ans}) \multimap \ldots$ (Berdine, O'Hearn, Reddy & Thielecke)
- Syntax as a non-commutative substructural logic (Lambek 1958)
- Reg exp and parsing with derivatives (Sulzman & Lu; Might et al)
- With Asiri Rathnayake: regular expressions using abstract machines

Conclusions

$$
\begin{tikzcd}
\text{Grammar} \arrow[r, "\text{Left recursion elimation}"] \arrow[d, "{[\![-]\!]}"] & \text{Transformed grammar} \arrow[d, "{[\![-]\!]\ \text{(extended)}}"] \\
\text{Meaning} \arrow[r, "\text{Continuation passing}"] & \text{Transformed meaning}
\end{tikzcd}
$$

- Syntax and semantics connected via functor
- Diagram chase as proof method
- Induction hypothesis on continuation k
- Possible applications: perform transform automatically for parser generators like ANTLR, correctness
- The ingredients are $\otimes$ and a contravariant negation, much like $\otimes\neg$-categories
- Continuations are everywhere, not just about `call/cc` in Scheme; discovered multiple times

The End

Thank you

Example grammar revisited

$$[\![E']\!] \stackrel{\text{def}}{=} \mathbb{N} \longrightarrow \mathbb{N}$$

The semantic actions for the transformed grammar rules can be written as lambda-terms as follows:

$$\begin{aligned}
[\![E :\!- \mathtt{1}\ E']\!] &\stackrel{\text{def}}{=} \lambda(\,()\,, k).k(1) \\
&: (\textbf{Unit} \otimes (\mathbb{N} \to \mathbb{N})) \to \mathbb{N} \\
[\![E' :\!- \mathtt{-}\ E\ E']\!] &\stackrel{\text{def}}{=} \lambda(\,()\,, x, k).\lambda y.k(y - x) \\
&: (\textbf{Unit} \otimes \mathbb{N} \otimes (\mathbb{N} \to \mathbb{N})) \to (\mathbb{N} \to \mathbb{N}) \\
[\![E' :\!- \varepsilon]\!] &\stackrel{\text{def}}{=} \lambda\,()\,.\lambda x.x \\
&: \textbf{Unit} \to (\mathbb{N} \to \mathbb{N})
\end{aligned}$$

Premonoidal $\otimes$ and central morphisms

f is central iff for all g the two compositions agree:

$$X_1 \xrightarrow{\quad f \quad} X_2$$

$$
\begin{tikzcd}
Y_1 \arrow[d, "g"] & & X_1 \otimes Y_1 \arrow[r, "f \otimes Y_1"] \arrow[d, "X_1 \otimes g"] & X_2 \otimes Y_1 \arrow[d, "X_2 \otimes g"] \\
Y_2 & & X_1 \otimes Y_2 \arrow[r, "f \otimes Y_2"] & X_2 \otimes Y_2
\end{tikzcd}
$$