Guarded and Mendler-Style (Co)Recursion in Circular Proofs

Tarmo Uustalu, Institute of Cybernetics

Joint work with
Varmo Vene, University of Tartu

Theory Days at Vanaõue, 28–30 Sept. 2007

Motivation: Total/terminating functional programming

- There are reasons to dream about *total/terminating* functional programming, where programs denote total functions and terminate.
  - In type-theoretic settings, where programs and proofs are identified, this is unavoidable (proofs must be total/terminating).
  - In simpler settings this can mean a simpler semantic discourse (eg, set-theoretic instead of domain-theoretic).
  - And why should it be necessary to support partiality in a language where we only want to program total functions?
- Some approaches and implementations exist, eg, D. A. Turner's strong functional programming, Cockett's CHARITY, the various type-theoretic languages, but none are fully satisfactory because...

General recursion

- . . . the challenge is to ban *general recursion*, ie, definitions

$$f(x) = \Phi(f)(x)$$

  without making programming impossible or unfeasible.
- General recursion is problematic because the totality/termination of a generally recursive definition is not decidable.
- A possibility would be to rely on some sufficient conditions.

Structured (co)recursion

- For most total/terminating programming however, general general recursion is not necessary, it is only a convenience.
- One can do with tamed forms of general recursion:
  - *structured recursion*, for defining functions consuming *inductive data*, eg, $\mathsf{exp} : \mathsf{Nat} \Rightarrow \mathsf{Int}$,

$$\begin{array}{rcl} \mathsf{exp}\,\mathsf{o} & = & 1 \\ \mathsf{exp}\,(\mathsf{s}\ x) & = & 2 * \mathsf{exp}\,x \end{array}$$

  - *structured corecursion*, for defining functions producing *coinductive data*, eg, $\mathsf{from} : \mathsf{Int} \Rightarrow \mathsf{Str}\,\mathsf{Int}$,

$$\mathsf{from}\,n \quad = \quad n : \mathsf{from}\,(n+1)$$

where the meaning of the word "structured" is flexible, ranging from *iteration and coiteration* and *primitive (co)recursion* to ...

Conventional vs guarded vs Mendler-style (co)recursion

- Structured (co)recursors (fold, unfold, primitive (co)recursion etc) in their *conventional* form are not an option, they are impractical, although semantically clean.
- Alternatives:
  - *Guarded* (co)recursors: general recursion like behaviour, but syntactic conditions ensure conformance to a structured recursion scheme, messy theoretically.
  - *Mendler-style* (co)recursors: similar, but conformance to a structured (co)recursion scheme is enforced by more restrictive typing, combine the benefits of conventional and guarded combinators.

This talk: Circular proofs

- A problem with structured (co)recursion is that nothing seems to tell us how far to go, the choice of the structured (co)recursion scheme to support is not canonical.
- *Circular proofs* are nonstandard kind of sequent calculi – with *rational* instead of wellfounded derivations – for classical predicate/modal logics with (co)inductive definitions.
- We show that two equivalent definitions of an analogous intuitionistic propositional sequent calculus lead to canonical term calculi for guarded resp Mendler-style (co)recursion.
- The known relation between Mendler-style and conventional structured recursion gives a syntactic embedding of the circular sequent calculus into the conventional, Park-style sequent calculus.

Conventional-style (co)recursion

- A combinator calculus for with (co)inductive types and conventional-style (co)recursion is obtained from simply typed lambda calculus like this:
- We introduce a type former $\mu : (* \Rightarrow *) \Rightarrow *$ which we only allow to be applied to positive type transformers.
- We also introduce a combinator $\mathrm{in} : F(\mu F) \Rightarrow \mu F$ as the constructor and a combinator

  $$\mathrm{iter} : (F\ C \Rightarrow C) \Rightarrow \mu F \Rightarrow C$$

  for iteration with the reduction rule

  $$\mathrm{iter}\, s\, (\mathrm{in}\, t) \quad \rhd \quad s\, (\mathrm{map}\, (\mathrm{iter}\, s)\, t)$$

  where $\mathrm{map} : (A \Rightarrow B) \Rightarrow F\ A \Rightarrow F\ B$ is the functoriality witness of F (as F is positive, it has a canonical such).
- Similarly we also introduce the former of inductive types ν, and the destructor and coiterator out, coit.

- Eg, $F = \lambda X.\, 1 + X$, $\mathsf{Nat} = \mu F$, $[\mathsf{o}, \mathsf{s}] = \mathsf{in}$.
- Suppose we want to define $\mathsf{exp} : \mathsf{Nat} \Rightarrow \mathsf{Int}$ by

$$
\begin{array}{rcl}
\mathsf{exp}\,\mathsf{o} & = & 1 \\
\mathsf{exp}\,(\mathsf{s}\,x) & = & 2 * \mathsf{exp}\,x
\end{array}
$$

- We can define

$$\mathsf{exp} = \mathsf{iter}(\lambda y^{:1+\mathsf{Int}}.\, \mathsf{case}(y, \lambda\langle\rangle.\, 1, \lambda y'.\, 2 * y'))$$

Guarded (co)recursion

- Alternatively, we could introduce a guarded combinator

$$\text{giter} :: ((\mu F \Rightarrow C) \Rightarrow F(\mu F) \Rightarrow C) \Rightarrow \mu F \Rightarrow C$$

  that can only be applied to abstractions $\lambda f \lambda x\, r$ where only the μF-components of x can be used as arguments of f in r and that is their only allowed usage.
  (This is informal and the formal condition is hard to state correctly.)

- giter would come with the reduction rule

$$\text{giter}\ s\,(\text{in}\ t) \quad \rhd \quad s\,(\text{giter}\ s)\ t$$

  ie, reduce as general recursion (modulo the constructor in).

- Eg, we should be able to define

$$\mathsf{exp} = \mathsf{giter}\,(\lambda f^{:\mathsf{Nat}\Rightarrow\mathsf{Int}}\lambda x^{:1+\mathsf{Nat}}.\,\mathsf{case}(x, \lambda\langle\rangle\,1, \lambda x'.\,2 * f\,x'))$$

- But then, why cannot we alternatively define

$$\mathsf{exp} = \mathsf{giter}\,(\lambda f^{:\mathsf{Nat}\Rightarrow\mathsf{Int}}\lambda x^{:1+\mathsf{Nat}}.\,\mathsf{case}(x, \lambda\langle\rangle\,1, \lambda x'.\,(\lambda g.\,2{*}g\,x')\,f))$$

etc?

Mendler-style (co)recursion

- N. P. Mendler realized that the flow of data from x into f in the abstraction body r in giter($\lambda f\,\lambda x\, r$) is better controlled by a tighter typing.
- The Mendler-style combinator miter is typed

$$\mathrm{miter} : (\forall Y.(Y \Rightarrow C) \Rightarrow F(Y) \Rightarrow C) \Rightarrow \mu(F) \Rightarrow C$$

rather than

$$: ((\mu F \Rightarrow C) \Rightarrow F(\mu F) \Rightarrow C) \Rightarrow \mu(F) \Rightarrow C$$

and the reduction rule remains

$$\mathrm{miter}\ s\,(\mathrm{in}\,t) \quad \triangleright \quad s\,(\mathrm{miter}\ s)\,t$$

- This has a clean semantic justification via the Yoneda lemma.
- Eg, $\exp = \mathrm{giter}\,(\lambda f^{:Y \Rightarrow \mathrm{Int}} \lambda x^{:1+Y}.\,\mathrm{case}(x, \lambda\langle\rangle.\,1, \lambda x'.\,2 * f\,x'))$.

Park-style sequent calculus for (co)inductive types with (co)iteration

- To the sequent calculus of IPL one adds the inference rules

$$\frac{\Gamma \longrightarrow F(\mu F)}{\Gamma \longrightarrow \mu F}\ \mu\mathcal{R} \qquad \frac{\Gamma, F(\prod \Gamma \Rightarrow C) \longrightarrow C}{\Gamma, \mu F \longrightarrow C}\ \mu\mathcal{L}$$

  (stating that μF is a prefixpoint and a least such).
- The term calculus has a conventional-style (co)iterator.
- A similar Park-style calculus is possible for, eg, primitive (co)recursion.

Calculus of circular proofs (guarded version)

- To the sequent calculus of IPL one adds the inference rules

$$\frac{\Gamma \longrightarrow F(\mu F)}{\Gamma \longrightarrow \mu F}\ \mu\mathcal{R} \qquad \frac{\Gamma, F(\mu F) \longrightarrow C}{\Gamma, \mu F \longrightarrow C}\ \mu\mathcal{L}^{\star}$$

  (stating only that μF is a pre- and postfixpoint of F) and redefines that a derivation is a *rational tree* (ie, an infinite tree with a finite number of distinct subtrees), subject to a wellformedness condition.
- These derivations are subject to a wellformedness (syntactic guardedness) condition: every infinite path in a derivation must contain a μ-subformula occurrence trace passing through infinitely many $\mu\mathcal{L}^{*}$ inferences with that subformula as the main formula.
- Intuition: Infinite paths satisfying the condition correspond to impossible cases, so they "don't matter", and infinite paths falsifying the condition are forbidden.

- With a standard, wellfounded notion of a derivation, we can achive the same with inference rules

$$\frac{\Gamma \longrightarrow F(\mu F)}{\Gamma \longrightarrow \mu F}\ \mu\mathcal{R} \qquad \frac{\begin{matrix}\Gamma, \mu F \longrightarrow C \\ \vdots \\ \Gamma, F(\mu F) \longrightarrow C\end{matrix}}{\Gamma, \mu F \longrightarrow C}\ \mu\mathcal{L}$$

  (where the $\mu\mathcal{L}$-rule is higher-order) and a modified wellformedness condition that, in any $\mu\mathcal{L}$-inference, the μ-formula occurrences of the conclusion and of any occurrence of the hypothesis are on the same trace.
- The path segments from the premise and to occurrences of the hypothesis represent cycles in the rational tree.
- The term calculus is with guarded (co)recursion.

Calculus of circular proofs (Mendler-style version)

- With Mendler's idea of tracking flow with quantified types we can reformulate the version with higher-order inference rules like this:

$$\frac{\Gamma \longrightarrow F(\mu F)}{\Gamma \longrightarrow \mu F}\ \mu\mathcal{R}$$

$$\dfrac{\begin{array}{c} \Gamma, Y_0 \longrightarrow C \quad \dfrac{\Gamma_0, \mu F \longrightarrow C_0}{\underset{[\Gamma_0, C_0]}{\Gamma_0, Y_0 \longrightarrow C_0}} \\ \vdots \\ \underset{[Y_0]}{\Gamma, FY_0 \longrightarrow C} \end{array}}{\Gamma, \mu F \longrightarrow C}\ \mu\mathcal{L}_0$$

- . . . except that this is not general enough, we also need this rule:

$$
\dfrac{
\Gamma, Y_0 \longrightarrow C \qquad
\begin{array}{c}
\dfrac{\Gamma_0, \mu F \longrightarrow C_0}{\underset{[\Gamma_0, C_0]}{\Gamma_0, Y_0 \longrightarrow C_0}} \\
\vdots \\
\underset{[Y_0]}{\Gamma, FY_0 \longrightarrow C}
\end{array}
\qquad
\dfrac{
\begin{array}{c}
\Gamma_0, Y_1 \longrightarrow C_0 \quad \dfrac{\Gamma_1, Y_0 \longrightarrow C_1}{\underset{[\Gamma_1, C_1]}{\Gamma_1, Y_1 \longrightarrow C_1}} \\
\vdots \\
\underset{[Y_1]}{\Gamma_0, FY_1 \longrightarrow C_0}
\end{array}
}{\underset{[\Gamma_0, C_0]}{\Gamma_0, Y_0 \longrightarrow C_0}}
}{\Gamma, \mu F \longrightarrow C}\ \mu\mathcal{L}_1
$$

and similarly also rules $\mu\mathcal{L}_2$, . . .

- This is primitive recursion with simultaneous subsidiary primitive recursion on structurally smaller arguments.
- To express the same we could define the combinators

$$
\begin{array}{lcl}
\mathsf{mxrec}_0 & : & (\forall Y_0.(Y_0 \Rightarrow C_0) \Rightarrow (Y_0 \Rightarrow \mu F) \Rightarrow F Y_0 \Rightarrow C_0) \\
 & & \Rightarrow \mu F \Rightarrow C_0 \\
\mathsf{mxrec}_1 & : & (\forall Y_0.(Y_0 \Rightarrow C_0) \Rightarrow (Y_0 \Rightarrow \mu F) \Rightarrow \forall C_1.(\\
 & & \quad ((\forall Y_1.(Y_1 \Rightarrow C_1) \Rightarrow (Y_1 \Rightarrow Y_0) \Rightarrow F Y_1 \Rightarrow C_1) \\
 & & \quad \Rightarrow Y_0 \Rightarrow C_1)) \\
 & &) \Rightarrow \mu F \Rightarrow C_0
\end{array}
$$

  etc with reduction rules

$$
\begin{array}{lcl}
\mathsf{mxrec}_0\, s\, (\mathsf{in}\, t) & \rhd & s\, (\mathsf{mxrec}_0\, s)\, \mathsf{id}\, t \\
\mathsf{mxrec}_1\, s\, (\mathsf{in}\, t) & \rhd & s\, (\mathsf{mxrec}_1\, s)\, \mathsf{id}\, \mathsf{mxrec}_0\, t
\end{array}
$$

  etc.
- These schemes are of greater direct expressive power than, eg, course-of-value primitive recursion. They have a semantic justification in terms of comonadic recursion.

Summary

- Circular proofs are an interesting kind of sequent calculi with rational derivations or with higher-order inference rules.
- The wellformedness condition can be stated as a syntactic guardedness condition, but also in a better way à la Mendler.
- This opens a novel avenue for the design of total functional programming languages, based on sequent calculi instead of Hilbert systems (combinatory logics) or natural deduction (lambda-calculi).