The Semantics of Higher Order Algorithms
Lecture III
May 31 - 2013

Dag Normann
The University of Oslo
Department of Mathematics

Finite Sequential Procedures

- Yesterday we defined the *Finite Sequential Procedures*.
- This class is generated from all constant functionals of finite type with the use of finite branching *Case* constructions and oracle calls on previously defined FSP's.
- They are intimately related to the game semantics approach, mainly due to Abramsky, to the analysis of sequentiality due to Hyland and Ong, and to Nickau, and they have their origin in the thesis of Sazonov from the 70'ies.

Outline of today's lecture

- Today we will explain what we mean by evaluating in this context.
- This will turn the finite sequential procedures into an intensional typed structure.
- We will define the typed structure of extensional hereditarily sequential functionals.
- We will show that this class, naturally ordered by function extension, will not be sequentially closed.

The Reference

Further details can be found in

D. Normann and V.Yu. Sazonov

The extensional ordering of the sequential functionals

Annals of Pure and Applied Logic.163(2012) pp. 575-603.

The Composition Theorem

For the sake of notational simplicity, we let $\vec{\sigma}$ etc. denote a finite sequence of types.

Theorem

Let T be a finite sequential procedure of type $\alpha, \vec{\sigma}, \vec{\tau} \rightarrow \mathrm{Nat}$ and let S be a finite sequential procedure of type $\delta, \sigma \rightarrow \alpha$.
Then there is a finite sequential procedure $T \circ S$ such that for all $\vec{x}$, $\vec{y}$ and $\vec{z}$ of the appropriate types we have that

$$(T \circ S)(\vec{x}, \vec{y}, \vec{z}) = T(S(\vec{x}, \vec{y}), \vec{y}, \vec{z}) \ .$$

Here $\vec{x}$, $\vec{y}$ and $\vec{z}$ will vary over any typed structure where all FSP's have natural interpretations.

The Composition Theorem

- The proof will be in two steps
  1. Give a finite set of deterministic transformation rules for expressions of the form
     $$(T \circ (S_1, \ldots, S_k))(\vec{x}, \vec{y}, \vec{z})$$
  2. Prove that the sequence of transformations is finite and results in an FSP H satisfying
     $$H(\vec{x}, \vec{y}, \vec{z}) = T(S_1(\vec{x}, \vec{y}), \ldots, S_k(\vec{x}, \vec{y}), \vec{y}, \vec{z})$$
     for all $\vec{x}$, $\vec{y}$ and $\vec{z}$ in all relevant typed structures.
- The proof use methods from normalization of typed λ-calculus, and is related to the proof of Plotkin's adequacy theorem.

The Composition Theorem

- The significance is that we can evaluate any FSP with FSP-arguments in a sequential way, and the evaluation will conclude, after finitely many steps, with an element in $\mathbb{N}_\perp$.
- Another consequence is that an FSP of type $\delta \to \tau$ will map, through a rewriting procedure, each FSP of type τ to an FSP of type δ.

Sequential Procedures

- The set of *sequential procedures* will, in a precise sense, be the completion of the set of finite sequential procedures.
- We use the same definition, with two relaxations
  1. We allow K to be infinite, i. e. we allow case-constructions with infinite branching.
  2. We use co-induction instead of induction.

Sequential Procedures

Definition

The *Sequential procedures* of finite types is the largest class of infinitary syntactic-like entities such that every sequential procedure F of type $\sigma = \tau_1, \ldots, \tau_n \to \mathrm{Nat}$ is of one of two forms

1. C_a^σ where $a \in \mathbb{N}_\perp$.
2. $$F(\vec{x}) = F_k(\vec{x}) \text{ if } x_i(G_1(\vec{x}), \ldots, G_m(\vec{x})) = k \in K$$
   where $K \subseteq \mathbb{N}$ and each F_k and G_j are sequential procedures of the correct types.

Sequential Procedures

- The sequential procedures are intensional objects in the sense that the strategy for evaluating a given procedure on a set of inputs does not depend only on the function the procedure defines, but on the form of the procedure itself.
- An evaluation may now not terminate, as the deterministic strategy of evaluation may lead to an infinite evaluation branch.
- In order to be precise about which function a procedure defines, we must consider the *hereditarily extensional collapse* of the typed structure of sequential procedures.

The Sequential Functionals

Definition

Let F_1 and F_2 be sequential procedures of types $\sigma = \tau_1, \ldots, \tau_n \to \text{Nat}$.
We say that $F_1 \sqsubseteq F_2$ if we for all procedures $G_1, \ldots, G_n$ of types $\tau_1, \ldots, \tau_n$ and all $a \in \mathbb{N}$ have that $F_2(G_1, \ldots, G_n)$ evaluates to a whenever $F_1(G_1, \ldots, G_n)$ evaluates to a.

Theorem

Each sequential procedure is monotone with respect to $\sqsubseteq$.

The proof is trivial for finite sequential procedures, by structural induction. For sequential procedures in general, observe that a counterexample will manifest itself by finite sub-procedures.

The Sequential Functionals

- $\sqsubseteq$ is a partial ordering of each type σ, and will induce equivalence relations $\equiv_\sigma$.
- Each sequential procedure will, due to the monotonicity, respect equivalence between inputs.
- Thus the typed structure of equivalence classes can be viewed as a structure of typed functionals.
- It will be these functionals that we call *The sequential functionals*.

The semilattice

- It is not the case that two sequential functionals of the same type will be bounded by a sequential functional, even if they agree on all inputs where they both terminate.
- There will, however, always be a greatest lower bound of two sequential functionals F and G of type $\sigma = \tau_1, \ldots, \tau_n \to \mathrm{Nat}$:
- $H(\vec{x}) = H_a(\vec{x})$ if $F(\vec{x}) = a$

  where
- $H_a(\vec{x}) = C_a(\vec{x})$ if $G(\vec{x}) = a$

The semilattice

- We can use this to prove that if two finite sequential functionals are bounded by a sequential functional, they have a least upper bound among the sequential functionals.
- This does not need to be the least upper bound in the sense of the Scott model.

An important example

- Let $\sigma = \tau_1, \ldots, \tau_n \to \text{Nat}$ be a type, where $n \geq 0$, let x be a variable of type $\sigma \to \sigma$, and let $y_1, \ldots, y_n$ be variables of type $\tau_1, \ldots, \tau_n$ resp.
- Let Y_σ be the sequential procedure

$$Y_\sigma(x, y_1, \ldots, y_n) = C_a(x, y_1, \ldots, y_n) \text{ if } x(Y_\sigma(x), y_1, \ldots, y_n) = a\,.$$

- Inspection shows that this is a sequential procedure, requiring that one proves that projections are sequential.
- We need the construction by co-induction in order to make sense of this.

Sequential procedures and LCF

- We now have the tools needed to show that all LCF-terms are represented by sequential procedures.
- Our previous example provided us with the least fixed point operator.
- It is a trivial exercise to express all typed versions of the combinators K and S as sequential procedures.
- Successor and predecessor are directly sequential and definition by cases is taken care of by the inductive definition of these procedures.

The sequential functionals

- The converse is also, in some sense, true.
- PCF_Ω is PCF with constants $\underline{f}$ for each function $f : \mathbb{N} \to \mathbb{N}$ and with the rules
$$\underline{f}\ \underline{n} \to \underline{f(n)}\ .$$
- Then the sequential functionals will be the unique extensional typed structure of PCF_Ω-definable functionals.
- The question we now should ask is:

  Is the typed structure of sequential functionals a well behaved mathematical object?

Two negative results

- One might think that since the sequential procedures is the completion of the finite sequential procedures, the sequential functionals will be the completion of those represented by finite sequential procedures.
- In a sense, the finite sequential functionals form a dense subset of the set of all sequential functionals, but this is not so useful as it may seem because Loader proved that $\sqsubseteq$ restricted to FSP's is not a decidable ordering, it is only of complexity Π^0_2.

Two negative results

- We will prove another negative result.
- There is a sequence $\{F_n\}$ of finite sequential procedures of type
  $$(\mathrm{Nat}, \mathrm{Nat} \to \mathrm{Nat}) \to \mathrm{Nat}$$
  such that
  1. $F_n \sqsubseteq F_m$ whenever $n \leq m$.
  2. There is no sequential procedure F such that $F_n \sqsubseteq F$ for all $n \in \mathbb{N}$.

A counterexample

- We will construct a sequence $\{f_n\}_{n \in \mathbb{N}}$ of finite sequential procedures of type $\mathrm{Nat}, \mathrm{Nat} \to \mathrm{Nat}$.
- We will show that for each $n \in \mathbb{N}$ there is a $\sqsubseteq$-minimal finite sequential procedure F_n such that $F_n(f_0) = 0$, $F_n(f_k) = 1$ for $0 < k \leq n$ and $F_n(f_k)$ is undefined for $k > n$.
- We will prove that whenever F is a sequential procedure terminating on all f_n then F is constant.

A counterexample

- Let $f_0(a,b) = 0$ if $a = 0$, and let $f_0(a,b) = \perp$ otherwise.
- If $m > 0$, let $f_m(a,b) = 1$ if $b = m$ or if $b < m$ and $a = 1$, and let $f_m(a,b) = \perp$ otherwise.
- Let $G_0(f) = 0$ if $f(0,\perp) = 0$ and $\perp$ otherwise.
- For $n > 0$, let

$$G_n(f) = f(\cdots f(f(G_{n-1}(f), n), n-1), \ldots, 0)$$

provided all intermediate values are 0 or 1.

A counterexample

- Direct evaluation gives us that $G_0(f_0) = 0$ and that $G_0(f_m)$ is undefined for $m > 0$.
- Induction on n gives us that $G_n(f_0) = 1$ for $n \geq 0$.
- Use that $f_m(\bot, m) = 1$ and the definition of f_m for $m > 0$ to see that $G_n(f_m) = 1$ when $0 < m \leq n$.
- Use induction on n to show that $G_n(f_m) = \bot$ when $n < m$.
- Since there are only finitely many $F \sqsubseteq G_n$ satisfying the specifications, we can let F_n be the least one.
- Then clearly $F_n \sqsubset F_{n+1}$ for each n.

A counterexample

- Now assume that F is a sequential procedure such that $F(f_0) \neq \perp$ and such that $F(f_n) \neq \perp$ for infinitely many n. We will use induction on the number of steps in the evaluation of $F(f_0)$ and prove that F must be constant.
- The induction base will be when F is of the form C_a and the induction step will be when F is of the form

$$F(f) = F_k(f) \text{ if } f(H_1(f), H_2(f)) = k \in K \ .$$

- The induction base is trivial, so assume that we are in the case of the induction step.

A counterexample

- $H_1(f_0) = 0$ since $F(f_0)$ has a value in $\mathbb{N}$.
- Let $n > 0$ be such that $F(f_n) \in \mathbb{N}$. Then either $H_2(f_n) = n$ or $H_2(f_n) < n$ and $H_1(f_n) = 1$.
- Since f_n is bounded by the constant 1 for all $n > 0$, there is at most one $n > 0$ such that $H_2(f_n) = n$, all values must be the same.
- Thus $H_1(f_n) = 1$ for infinitely many n.
- By the induction hypothesis, H_1 is a constant.
- This contradicts that $H_1(f_0) = 0$ while $H_1(f_n) = 1$ for infinitely many n.
- So, assuming that we are in the case of the induction step leads, with the use of the induction hypothesis, to a contradiction, and the induction base is the only possible case.

Discussion

- There is an intimate connection between our sequential procedures and the intensional objects studied in game semantics.
- The extensional typed structures obtained by the two approaches are identical.
- We have proved that there are increasing sequences with no upper bound in this structure.
- Modifying this construction we may construct sequences with least upper bounds that differ from those in the domain interpretation.

Discussion

- As observed by N. and Sazonov, there will be sequential functionals that are not continuous with respect to least upper bounds.
- These phenomena beg for a modification of concepts from topology in order to adjust them to the sequential functionals.
- Pioneering work in this direction is carried out by Escardó and Ho in one paper and by Sazonov in two papers.
- There are still open problems concerning the order type of the sequential functionals, and room for further research.

Partial vs. hereditarily total functionals

- It is well known that a nondeterministic Turing machine can be simulated by a deterministic one, at an exponential cost of effectivity.
- We have also seen that *parallel or* cannot be defined in PCF.
- The problem is that any sequential algorithm for $\mathbf{POR}(x, y)$ will have to first ask for the value of x or for the value of y, and if this is undefined the algorithm aborts.
- There will be a sequential procedure $\mathbf{OR} \sqsubseteq \mathbf{POR}$ that works well *under the assumption* that both inputs are true boolean values.

Partial vs. hereditarily total functionals

- The same will hold for any Scott-continuous function $f : \mathbb{N}_\perp^n \to \mathbb{N}_\perp$, there will be a sequential $g \sqsubseteq f$ such that for all *total* inputs $\vec{x}$ we have that $g(\vec{x}) = f(\vec{x})$.
- In a sense, g is of higher complexity than f, it requires more resources in the form of more information about the input.

Partial vs. hereditarily total functionals

- This picture is even more transparent at type level 2:
- If $F : (\mathbb{N}_\perp \to \mathbb{N}_\perp) \to \mathbb{N}_\perp)$ is Scott continuous, there will be a sequential $G \sqsubseteq F$ such that $G(f) \simeq F(f)$ for all total f.
- The algorithm for computing $G(f)$ will be to compute $f(0), f(1), \ldots$ until we have enough information about f to know the value of $F(f)$.
- If F happened to be defined by $F(f) = 0$ if $f(17) = 0$ or $f(18) = 1$ for all partial f, we see that G is of higher complexity than F.

Partial vs. hereditarily total functionals

- If we move up in type level, we will still be able to prove a similar theorem, but now not within the time limits of this course.
- In the Scott model, we define by recursion on the type that a functional Φ is *hereditarily total* if it takes a value in $\mathbb{N}$ for all hereditarily total inputs.
- All sequential procedures will have interpretations in the Scott model.
- We then have the following:

Partial vs. hereditarily total functionals

Theorem (N.)

For every Scott continuous functional Φ there is a sequential $\Psi \sqsubseteq \Phi$ such that for all hereditarily total $\vec{\phi}$ of the appropriate types:

$$\Psi(\vec{\phi}) \simeq \Phi(\vec{\phi}) \; .$$

Corollary (Plotkin)

Each Kleene-Kreisel continuous functional is definable by a sequential procedure.

Speculations

- The relation $\Phi \sqsubseteq \Psi$ between functionals is defined by $\Phi(x) = \Psi(x)$ whenever $\Phi(x)$ is defined.
- Another way to put this is that Φ requires more from the input in order to give a value than Ψ does.
- If we consider available resources as part of the input material, this means that Φ just requires more resources than Ψ.
- So, another reading of $\sqsubseteq$ is *more complex than*.

Speculations

- This view only opens up for concepts of relative complexity of higher order algorithms.
- Concepts like *polynomial* make only artificial sense, and generalizing those can be considered as a *cul de sac* for higher order complexity theory.
- Still one might want measurements of improved or worsted complexities in this setting.
- These are mere speculations, but a further understanding of the semantics might offer the possibility of obtaining such measurements.

END