The Semantics of Higher Order Algorithms
Lecture II
May 30 - 2013

Dag Normann
The University of Oslo
Department of Mathematics
The finitary total functionals

- Last Monday we defined
  i) The finite sets $D_n(\sigma)$ for each type $\sigma$.
  ii) The set $D_\omega(\sigma)$ of finitary partial functionals of type $\sigma$.
  iii) The Scott domains $D(\sigma)$ as the completion of $D_\omega(\sigma)$.

- Today we will look at typed structures in general.
- We will start with defining the hereditarily finitary total functionals.
Finite types revisited

- We defined the finite types by closing Nat under \( \tau, \delta \mapsto (\tau \to \delta) \).
- We have taken the liberty to let \( \sigma, \tau \to \delta \) be short for \((\sigma \to (\tau \to \delta))\).
- Pushing this further, we use the convention that

\[
\tau_1, \ldots, \tau_n \to \delta
\]

is another way of writing

\[
(\tau_1 \to (\tau_2 \to \cdots \to (\tau_n \to \delta) \cdots)).
\]
Finite types revisited

By an easy proof by induction we see that any type \( \sigma \) can be described on a normal form as

\[
\sigma = \tau_1, \ldots, \tau_n \rightarrow \text{Nat}.
\]

This also means that if we define a class of objects of types

\[
\tau_1, \ldots, \tau_n \rightarrow \text{Nat},
\]

we implicitly define a class of objects of all types.
The Prime Structure

Definition

1. Let $\mathcal{F}_n(\text{Nat}) = \{0, \ldots, n\}$.
2. When $\sigma = \tau \rightarrow \delta$, let $\mathcal{F}_n(\sigma)$ be the set of all functions from $\mathcal{F}_n(\tau)$ to $\mathcal{F}_n(\delta)$. 
The Prime Structure

Definition

If $\sigma$ is a type and $n \leq m$ we define, by recursion on $\sigma$

- $\eta_{n,m}^\sigma : \mathcal{F}_n(\sigma) \rightarrow \mathcal{F}_m(\sigma)$
- $\pi_{n,m}^\sigma : \mathcal{F}_m(\sigma) \rightarrow \mathcal{F}_n(\sigma)$

as follows:

i) $\eta_{n,m}^{\text{Nat}}(k) = k$ for $k \leq n$.

ii) $\pi_{n,m}^{\text{Nat}}(k) = \min\{k, n\}$ for $k \leq m$.

iii) $\eta_{n,m}^\sigma(\Phi)(\psi) = \eta_{n,m}^\delta(\Phi(\pi_{n,m}^\tau(\psi)))$ when $\sigma = \tau \rightarrow \delta$, $\Phi \in \mathcal{F}_n(\sigma)$ and $\psi \in \mathcal{F}_m(\tau)$.

iv) $\pi_{n,m}^\sigma(\Psi)(\phi) = \pi_{n,m}^\delta(\Psi(\eta_{n,m}^\tau(\phi)))$ when $\sigma$ is as above, $\Psi \in \mathcal{F}_m(\sigma)$ and $\phi \in \mathcal{F}_n(\tau)$. 
Lemma

a) For each $n$ and $\sigma$, both $\eta_{n,n}^\sigma$ and $\pi_{n,n}^\sigma$ will be the identity function on $\mathcal{F}_n(\sigma)$.

b) If $n \leq m \leq k$ then for all types $\sigma$ we have that

$$\eta_{n,k}^\sigma = \eta_{m,k}^\sigma \circ \eta_{n,m}^\sigma$$

and that

$$\eta_{n,m}^\sigma = \pi_{m,k}^\sigma \circ \eta_{n,k}^\sigma.$$ 

The proofs are easy, but instructive, by induction on $\sigma$. 

Nordic Spring School in Logic 2013
This lemma shows that there will be a total typed structure 
\( \mathcal{F}_\omega = \{ F_\omega(\sigma) \}_{\sigma \text{ type}} \) that essentially is the directed limit (co-limit) of the \( \mathcal{F}_n \)'s.

We will call this the *Prime Typed Structure*, indicating that it is related to the concept of *prime model* in model theory.
The Kernel

For a while, we let $T = \{ T(\sigma) \}_{\sigma \text{ type}}$ be a typed structure with $T(\text{Nat}) = \mathbb{N}$ and with the extra properties:

$\checkmark$ $T$ is a model for typed $\lambda$-calculus.
$\checkmark$ The functions

i) $\text{Suc}(n) = n + 1$
ii) $\text{Pred}(n) = \max\{0, n - 1\}$
iii)$\text{Ifzero}(a, b, c) = \begin{cases} b & \text{if } a = 0 \\ c & \text{if } a > 0 \end{cases}$

are in $T$. 

Nordic Spring School in Logic 2013  -  9
We may then define the corresponding maps $\eta_{n,T}^\sigma$ and $\pi_{n,T}^\sigma$, and the union of the images of the $\eta_{n,T}^\sigma$'s will form a typed substructure that is isomorphic, with respect to application, to $\mathcal{F}_\omega$.

This image will be called the \textit{kernel} of $T$. $T$ is \textit{rudimentary closed} if the definition of the kernel of $T$ is sound.

Any kernel will be isomorphic to the prime structure, provided $\mathbb{N}$ is a subset of the interpretation of the base type.

The elements of the kernel is also known as the \textit{hereditarily finitary elements} of $T$. 
The Kernel

- Alternatively we may describe $\pi_{n,T} \circ \eta_{n,T} : T(\sigma) \to T(\sigma)$ by a trivial induction on the type as the function $(\cdot)_n$ defined by
  - $(m)_n = m$ if $m \leq n$.
  - $(m)_n = n$ if $m > n$.
  - $(\Phi)_n(\psi) = (\Phi((\psi)_n))_n$ if $\Phi \in A(\tau \to \delta)$ and $\psi \in A(\tau)$.

The kernel will be exactly the union of the images of these operators.
A digression

- It is customary to study typed structures with some closure properties, but without too wild objects.
- One way to express this is to require that our structure is a model of Gödel’s $\mathcal{T}$ but does not contain the functional $^2E$.
- Gödel’s $\mathcal{T}$ essentially is a language for higher order primitive recursion.

\[
^2E(f) = \begin{cases} 
0 & \text{if } \forall n(f(n) = 0) \\
1 & \text{if } \exists n(f(n) > 0) 
\end{cases}
\]
A digression

If $T$ is a total typed structure satisfying these conditions we can express and prove the following theorem:

If $\Phi = \lim_{n \to \infty} \Phi_n$ then $\Phi = \lim_{n \to \infty} (\Phi_n)_n$.

A sequence in $T(\sigma)$ will here just be an element of $T(\text{Nat} \to \sigma)$.

We say that $\Phi = \lim_{n \to \infty} \Phi_n$ in $T(\sigma)$ if there is a $\Psi \in T(\sigma)$ such that

$$\forall \vec{x} \in T(\vec{\tau}) \forall n \geq \Psi(\vec{x})(\Phi(\vec{x}) = \Phi_n(\vec{x})).$$
Kleene’s first model

- We defined a typed structure as a hierarchy of functionals.
- However, in providing our examples of $D_\omega$, $D$ and $F_\omega$ we swept a lot of simple, but tedious, details under the carpet while claiming that we actually did construct typed structures.
We will now introduce the concept of *intensional typed structures*, and with these, we will also obtain the tools needed to lift up our carpet and handle the details that are under it.

We will start with one important example.

It is based on *Kleene’s first model*. 
Kleene’s first model

Let $\phi_e$ be the partial function from $\mathbb{N}$ to $\mathbb{N}$ defined by algorithm no. $e$, e. g. via a natural enumeration of the Turing Machines.

Kleene’s first model $\mathcal{K}_1$ consists of the set $\mathbb{N}$ and the partial application operator

$$e \cdot d \simeq \phi_e(d).$$
Kleene’s first model

- There will be a number $k$ such that $\phi_k(e)$ is an index for the constant function with value $e$.
- This actually means that $k \cdot e \cdot d = e$ for all $e$ and $d$ (recall how to insert the left-out brackets).
- Thus $k$ satisfies the property of the combinator $K$ introduced on Monday: $KNM \rightarrow N$. 
Kleene’s first model

- In order to find a number \( s \) serving as an interpretation of the combinator \( S \), let us recall the property of this combinator:

\[
SNML \rightarrow (NL)(ML)
\]

- Thus our specification for \( s \) is that

\[
s \cdot n \cdot m \cdot l = (n \cdot l) \cdot (m \cdot l).
\]
Spelled out, this actually requires a number $s$ such that we have for all $n, m$ and $l$ have

$$\phi_{\phi s(n)}(m)(l) \simeq \phi_{\phi n(l)}(\phi m(l)).$$

The right hand side is computable in the three variables $n, m$ and $s$, and then we use iteration of the $S_{n,m}$-theorem. (where the $n$’s and $m$’s are not the same in the two cases).
Kleene’s first model gives us an alternative model for PCF.

Each type $\sigma$ will be interpreted as a partial equivalence relation $\equiv_\sigma$ on $\mathbb{N}$.

A *partial equivalence relation* (per) is a relation $\equiv$ that is symmetric and transitive, but not necessarily reflexive.

Note that if $a \equiv b$ then $a \equiv a$ and $b \equiv b$.

$\{ a \mid a \equiv a \}$ is called the *domain* of $\equiv$, and $\equiv$ will be an equivalence relation on its domain.
The *per*-model

- We let $e \equiv_{\text{Nat}} d$ if $\phi_e(0) \simeq \phi_d(0)$.
- If $\sigma = \tau \rightarrow \delta$ we let
  \[ e \equiv_\sigma d \iff \forall a, b (a \equiv_\tau b \rightarrow (e \cdot a \equiv_\delta d \cdot b)) , \]
- We let $K(\sigma)$ be the domain of $\equiv_\sigma$ for each $\sigma$. 

The *per*-model

- Our first example of an intensional typed structure will be \( \{K(\sigma)\}_{\sigma} \) together with the restrictions \( App_{\tau,\delta} \) of the Kleene operator \( \cdot \) to each \( K(\tau \rightarrow \delta) \times K(\tau) \).

- \( App_{\tau,\delta} \) will be total by definition, and what we have constructed is an example of the more general *typed combinatory algebra*.

- By restricting ourselves to each \( K(\sigma) \), we see that \( \equiv_{\sigma} \) is definable from the application operators.

- With this property, we see that every intensional object actually defines an extensional one:
The *per*-model

- By recursion on $\sigma$ we define an extensional typed structure \{\(EP(\sigma)\)\}_{\sigma\text{ type}} by

- \(EP(\text{Nat}) = \mathbb{N} \cup \{\bot\}\), and for \(n \in \mathbb{N} = K(\text{Nat})\) we let \(\rho_{\text{Nat}}(n) \simeq \phi_n(0)\).

- If \(\sigma = \tau \to \delta\) and \(e \in K(\sigma)\), we let \(\rho_\sigma(e)\) be the one and only function \(F : EP(\tau) \to EP(\delta)\) that satisfies

\[
F(\rho_\tau(d)) = \rho_\delta(e \cdot d)
\]

for all \(d \in K(\tau)\).

- \(EP(\sigma)\) will be the image of \(\rho_\sigma\).
The \textit{per}-model

\textbf{Lemma}

Let $k$ and $s$ be the indices of the interpretations of the combiners $K$ and $S$ respectively. For all types $\sigma$, $\delta$ and $\tau$ we have

\begin{enumerate}
\item $k \in K(\sigma \rightarrow (\tau \rightarrow \sigma))$.
\item $s \in K(\sigma \rightarrow (\tau \rightarrow \delta), (\sigma \rightarrow \tau), \sigma \rightarrow \delta)$.
\end{enumerate}

The proofs are not hard, actually trivial, but are better worked out as exercises than via a slide.
The \textit{per}-model

- The interpretation of the typed combinators ensure that the \textit{per}-model is a model of pure typed $\lambda$-calculus.
- The terms for fixed point operators in untyped $\lambda$-calculus cannot be typed, so in order to have a model for PCF we need sound interpretations of each $Y_\sigma$.
- In order to obtain this, we actually need a generalized version of the Myhill-Shepherdson theorem:
The Myhill-Shepherdson Theorem

Given two expressions $t$ and $t'$ for partial numbers, $t \simeq t'$ will mean that they either both are defined and equal, or both are undefined. We write $t = t'$ to mean that they are both defined and equal.

**Theorem (Myhill-Shepherdson)**

Let $f$ be a partial computable function such that

$$\phi_e = \phi_d \Rightarrow f(e) \simeq f(d).$$

Then there is a partial computable functional $F$ of type 2 such that

$$F(\phi_e) = f(e)$$

for all $e$. This $F$ will be monotone, and finitely based.
The \textit{per}-model

- It is easy to see that the \textit{per}-interpretation of $\text{Nat} \to \text{Nat}$ will correspond to the set of partial computable functions on $\mathbb{N}$.
- The Myhill-Shepherdson theorem actually tells us that the \textit{per} interpretation of $(\text{Nat} \to \text{Nat}) \to \text{Nat}$ corresponds to the effectively continuous functionals of type 2.
The *per*-model

- In fact, the *per*-model corresponds to the effective version of the Scott model.
- This is an application of the proof of the Myhill-Shepherdson theorem.
- Since the typed least fixed point operators are effective elements of the Scott model, they “exist” in the *per*-model as well.
- Thus the *per*-model is a possible model for higher order computability.
In what we have done so far, there are several levels of abstraction.

Kleene’s first model is an example of a partial combinatory algebra, which will in general consist of a set $A$, a partial application operator $\cdot$ and two elements $K$ and $S$ obeying the axioms of the combinators.
Intensional typed structures

▶ If we interpret the base type Nat as a partial equivalence relation \( \equiv_{\text{Nat}} \) on \( A \), we implicitly interpret each type \( \sigma \) as a partial equivalence relation \( \equiv_{\sigma} \) on \( A \).

▶ If \( \equiv_{\text{Nat}} \) is induced from a partial function \( \rho_{\text{Nat}} \) into \( \mathbb{N} \cup \{\bot\} \), we can carry out our construction of an extensional typed structure.

▶ This is called the extensional collapse of \( (A, \cdot, \rho_{\text{Nat}}) \)
Intensional typed structures

- We do not have to start with a partial combinatory algebra in order to construct an extensional collapse.
- There is an intermediate concept of typed partial combinatory algebras, where we postulate typed application operators and typed versions of the combinators $K$ and $S$. 
In our next key example, the sequential procedures and the sequential functionals, we will be in the situation where \( \text{Nat} \) is interpreted directly as \( \mathbb{N} \cup \{ \perp \} \), but \( \tau \rightarrow \delta \) will be interpreted as a class of certain typed algorithms.

The application operators will then be interpreted via an observational semantics.

This will be another example of what we will call an \textit{intensional typed structure}. 
Intensional typed structures

Definition
An *intensional typed structure* will consist of

- A set $T(\sigma)$ for each type $\sigma$ such that $T(\text{Nat}) \subseteq \mathbb{N} \cup \bot$.
- An application operator $\text{App}_\sigma : T(\sigma) \times T(\tau) \rightarrow T(\delta)$ whenever $\sigma = \tau \rightarrow \delta$.
- Objects $K_{\tau,\delta}$ and $S_{\tau,\delta,\xi}$ obeying the typing and rules of typed combinators.
- Objects $\text{Case}_a^\sigma$ of type Nat, $\sigma, \sigma \rightarrow \sigma$ for each $a \in \mathbb{N} \cap T(\text{Nat})$ obeying:
Intensional typed structures

$$\text{Case}_a^\sigma(b, N, M) = \begin{cases} 
N & \text{if } b = a \\
M & \text{if } b \in \mathbb{N} \land b \neq a \\
C^\sigma_\bot & \text{if } b = \bot 
\end{cases}$$

(reformulated using $\text{App}$).
Intensional typed structures

- Let $T$ be an intensional typed structure.
- If $\sigma = \tau \rightarrow \delta$, $a \in T(\sigma)$ and $f : T(\tau) \rightarrow T(\delta)$, we say that $a$ tracks $f$ if $f(b) = \text{App}_\sigma(a, b)$ for all $b \in T(\delta)$.
- Each constant function of type $\tau \rightarrow \delta$ will be tracked: We use the properly typed version of $K$:
- The identity function of type $\sigma \rightarrow \sigma$ will be tracked: We use the properly typed version of $SKK$. (The two $K$’s are of different types.)
- The composition of two tracked functions $f : T(\delta) \rightarrow T(\tau)$ and $g : T(\tau) \rightarrow T(\xi)$ is tracked: If $a$ tracks $f$ and $b$ tracks $g$, then $S(Kb)a$, properly typed, tracks $g \circ f$. 
The Karoubi envelope

- Each intensional typed structure $T$ may be viewed as a category.
- The objects will be the interpretations $T(\sigma)$ when $\sigma$ varies over the types.
- The morphisms will be the set of functions that are tracked by elements of the structure.
The Karoubi envelope

- A morphism $e : \sigma \rightarrow \sigma$ is *idempotent* if $ee = e$.
- From the perspective of category theory, an idempotent automorphism may be viewed as a recognizable substructure.
- The objects of the Karoubi envelope will be pairs $(T(\sigma), f)$ where $f$ is a trackable idempotent map on $T(\sigma)$. 
We think of an object \((T(\sigma), f)\) as representing the isomorphism type of the set of fixed points of \(f\), and we think of the Karoubi envelope as representing all datatypes that are \textit{implicit} in our typed structure.

The morphisms will be morphisms in \(T\) commuting with the idempotents in question.
The Karoubi envelope

- For most important examples, the Karoubi envelope is richer than the simply typed structure, being closed under finite products, finite disjoint sums and often strictly positive induction and/or co-induction.

- However, questions related to the computational power of a calculus of higher order algorithms may often be solved for the full envelope just by studying the core types.

- Thus, even though a rich typed structure is an advantage when useful programs are in need, poor typed structures suffice for, and simplify, foundational research.
Towards another example

- Our next example, an example that still offers challenges for research, will be the *sequential operators*.
- In order to motivate this construction, let us see why the Scott model is, in some respects, unsatisfactory.
We have already seen that there are finite elements in the Scott model that are not the interpretation of any PCF-term.

We may extend the language, and introduce a constant with evaluation rules for objects with this property.

One possibility will be the non-sequential conditional $p$ of type Nat, Nat, Nat $\rightarrow$ Nat with the following rules:

1. $p \ 0 MN \rightarrow M.$
2. $p \ k + 1 MN \rightarrow N.$
3. $p \ MNN \rightarrow N.$
Plotkin proved that $\upiota_p$ is not PCF-definable, but that all finitary elements in the Scott model are PCF $+$ $\upiota_p$-definable.

This calculus is known as PCF$^+$

Even this new constant is not sufficient for defining all elements of the *per*-model.

For this we need the continuous existential quantifier $\exists_\omega$ over $\mathbb{N}$.
Full abstraction

For any typed calculus, we have the following

**Definition**

If $M$ and $N$ are closed terms of type $\sigma$, we say that $M$ is *observationally below* $N$, $M \sqsubseteq_{\text{obs}} N$, if whenever $K$ is a term of type Nat with one free variable $x$ of type $\sigma$, we have for every $k$

$$K^x_M \rightarrow^* k \Rightarrow K^x_N \rightarrow^* k \ .$$

This means in popular terms that in any program $K$ we can replace the subroutine $M$ with the subroutine $N$ and get an improved result.
Two terms are *operationally equivalent* if they are observationally below each other.

A model is *fully abstract* if observationally equivalent closed terms are interpreted as the same object.

An early observed problem with the Scott model is that it is **not** fully abstract.

Robin Milner produced a fully abstract model for PCF based on Scott domains.

Our question will be if domain theory is the best tool for defining models at all.
We will now define the finite sequential procedures as a piece of syntactic entities.

These procedures will have canonical interpretations in any typed structure with a minimum of closure properties.

The definition is by recursion.
Definition

Let $\sigma = \tau_1, \ldots, \tau_n \rightarrow \text{Nat.}$

a) If $a \in \mathbb{N}_\bot$ we let $C_\sigma^a$ be an FSP of type $\sigma$.

We write this definition as

$$C_\sigma^a(x_1, \ldots, x_n) = a$$

where $x_i$ is a variable of type $\tau_i$.

(We will normally drop the upper index when it is clear from the context.)
Finite Sequential Procedures

Definition (continued)

b) If

- $K \subset \mathbb{N}$ is finite
- $F_k(x_1, \ldots, x_n)$ is an FSP of type $\sigma$ for each $k \in K$
- $\tau_i = \delta_1, \ldots, \delta_m \rightarrow \text{Nat}$
- $G_j$ is an FSP of type $\tau_1, \ldots, \tau_n \rightarrow \delta_j$ for each $j = 1, \ldots, m$

Then

$$F(x_1, \ldots, x_n) = F_k(x_1, \ldots, x_n)$$

if $x_i(G_1(x_1, \ldots, x_n), \ldots, G_m(x_1, \ldots, x_n)) = k \in K$

is an FSP of type $\sigma$. 
There are a few questions that we will address tomorrow:

- Can we evaluate an expression like
  \[ F(H_1, \ldots, H_n) \]
  where \( F \) and \( H_1, \ldots, H_n \) are FSP’s, and in what sense is that evaluation sequential?

- If \( F \) is an FSP of type \( \tau \rightarrow \delta \), does \( F \) map FSP’s of type \( \tau \) to FSP’s of type \( \delta \)?

- What is the nature of the observational ordering of FSP’s?

- How can we move from FSP’s to sequential procedures?
We will end today’s lecture by considering a few examples.
Examples

Let

\[ F_L : (\mathbb{N}_\bot \times \mathbb{N}_\bot \to \mathbb{N}_\bot) \to \mathbb{N}_\bot \]

be defined by

\[ F_L(f) = 0 \text{ if } f(0, \bot) = 0. \]

Then \( F_L \) can be defined by an FSP as follows:

\[ F_L(f) = C_0(f) \text{ if } f(C_0(f), C_\bot(f)) = 0. \]

We can define \( F_R \) in a similar way.
Examples

With the same types as on the previous slide, let

\[ F(f) = f(F_L(f), F_R(f)) \]

This is defined by an FSP as follows

\[ F(f) = C_0(f) \text{ if } f(F_L(f), F_R(f)) = 0. \]

Why will there be a sequential evaluation here?
We only obtain a sequential evaluation when \( f \) is given in a sequential way, e.g. as

\[ f(x, y) = 0 \text{ if } y = 0. \]
Examples

The importance of this example is as follows:
Even if $F$ is clearly PCF-definable, there is no deterministic “interrogation tree" of oracle calls we can make to $f$ in order to compute $F(f)$.

Thus, the naïve belief that computable functionals at level 2 can be computed via a deterministic sequence of oracle calls is misleading when the types are not pure.

We will return to the failure of some folklore results based on this false intuition tomorrow.