Abstract Types and the Dot Notation

Luca Cardelli  
Xavier Leroy

Presented by: Mike Furr  
furr@cs.umd.edu
Abstract Types

• “different than every other type in the system when seen from outside the abstraction”

• Powerful programming technique

• existential types (Cyclone)

• Opaque types inside Modules (ML)
Notations

• existential quantifiers in logic lends the syntax
  \[ p : \exists A. f, g, \ldots \]
  open \( p \) as \( A', f', g', \ldots \) in \( \ldots A' \ldots f' \ldots g' \ldots \)

• PL developed the notation:
  \[ p : \exists A. f, g, \ldots \]
  \ldots p.A \ldots p.f \ldots p.g \ldots \]

• but what is the scoping in the PL case?
Motivation (circa 1990)

- two styles of notation exist from two different areas (logic & PL)
  - both need to be formalized
  - is one more expressive than the other?
Formalizing the Open Calculus

Lambda calculus with the “open” notation:

\[ A ::= \quad X \mid A \rightarrow B \mid \exists X.A \]
\[ a ::= \quad x \mid \lambda x : A.b \mid b(a) \mid \langle X = A, b : B \rangle \]
\[ \quad \mid \quad \text{open } a \text{ as } \langle X, y : B \rangle \text{ in } b \]

Note \( \langle X = A, b : B \rangle \) has type \( \exists X.B \).
\( X \) may occur in \( B \) (or \( b \)), but \( A \) may not
Type Checking

\[
\begin{align*}
E, X \vdash A \\
\hline
E \vdash \exists X.A
\end{align*}
\]

\[
\begin{align*}
E \vdash A & \quad E \vdash b\{X \leftarrow A\} : B\{X \leftarrow A\} \\
\hline
E \vdash \langle X = A, b : B\rangle : \exists X.B
\end{align*}
\]

\[
\begin{align*}
E \vdash a : \exists X.B & \quad E \vdash C & \quad E, X, y : B \vdash c : C \\
\hline
E \vdash \text{open } a \text{ as}\langle X, y : B\rangle \text{ in } c : C
\end{align*}
\]
Formalizing the Dot Calculus

lambda calculus with the “dot” notation:

\[ A ::= X \mid A \to B \mid \exists X.A \mid x.Fst \]
\[ a ::= x \mid \lambda x : A.b \mid b(a) \mid \langle X = A, b : B \rangle \mid x.snd \]

- \( x.Fst \) is the witness type, yet \( x \) is a term
- Note we are mixing terms with types
- However, only term variables are allowed
Type Checking

\[
\frac{E \vdash x : \exists X.A}{E \vdash x.Fst}
\]

\[
\frac{E \vdash x : \exists X.A}{E \vdash x.snd : A\{X \leftarrow x.Fst\}}
\]

\[
\frac{E \vdash A \quad E \vdash B \quad E, x : A \vdash b : B}{E \vdash \lambda x : A.b : A \to B}
\]

- Note that \(x\) may not appear free in \(B\).
From Dot to Open

• want to define a translation from the Dot calculus to the Open calculus

• In “open $x$ as $\langle Y, z : A \rangle$ in $b$”, $Y$ is similar to $x$.Fst and $z$ to $x$.snd

• One approach might be:

$$b[x.\text{Fst}, x.\text{snd}] \mapsto \text{open } x \text{ as } \langle Y, z : A \rangle \text{ in } b[Y, z]$$

• What if $x$.Fst appears in the type of $b$?
Translation Con’t

• Need to be careful not to create “open ... in” constructs where the term might escape its scope

• only include them for expressions that are “large enough”

• one safe way is to use the entire scope of the variable $x$
variables can only be introduced by \( \lambda \)-expressions

Thus, it is sufficient to use:

\[
\lambda(x : \exists X.A).b \quad \mapsto \quad \lambda(x : \exists X.A).\text{open } x \text{ as } \langle X, y : A \rangle \in b\{x.\text{Fst} \leftarrow X\}\{x.\text{snd} \leftarrow y\}
\]
Let $\mathcal{P}$ be the set of variables of existential type

$$x \in \mathcal{P} \Rightarrow x : \exists T_x. A_x$$

\[
\begin{align*}
[x] &= x \\
[x.snd] &= v_x \\
[\lambda(x : \exists X. A). b] &= \lambda x : [A]. \text{open } x \text{ as } \langle T_x, v_x : [A_x] \rangle \text{ in } [b] (\text{if } x \in \mathcal{P}) \\
[\lambda x : A. b] &= \lambda x : [A]. [b] (x \notin \mathcal{P}) \\
[b(a)] &= [b]([a]) \\
[\langle X = A, b : B \rangle] &= \langle X = [A], [b] : [B] \rangle
\end{align*}
\]
**Type Translation**

\[
\begin{align*}
\llbracket X \rrbracket &= X \\
\llbracket x. \text{Fst} \rrbracket &= T_x \\
\llbracket A \rightarrow B \rrbracket &= \llbracket A \rrbracket \rightarrow \llbracket B \rrbracket \\
\llbracket \exists X. A \rrbracket &= \exists X. \llbracket A \rrbracket
\end{align*}
\]

- Note this translation preserves typing
- Can also be shown to preserve semantics
Can easily go the other way:

\[
\text{open } a \text{ as } \langle X, y : A \rangle \text{ in } b \mapsto \\
(\lambda z : \exists X. A. b\{X \leftarrow z.\text{Fst}, y \leftarrow z.\text{snd}\})(a)
\]

both translations are type and semantic preserving

Thus, each notation is equally expressive
Expanding the Dot Calculus

- Recall ".Fst" and ".snd" could only be applied to term variables

- What about applying them to any generic term?

- combining types and terms in such a way could be dangerous(!)
Expanding the Dot Calculus II

• Most general form doesn’t model any real programming technique

• Some useful forms do exist, like paths: \( x.y.z \)

• Let’s analyze the most general case anyway :)
**Problems**

- However, consider the following in the dot calculus:

  \[
  \langle X = A, \langle Y = X, a : Y \rangle.\text{snd} : \langle Y = X, a : Y \rangle.\text{Fst} \rangle
  \]

- but in the open calculus, we get:

  \[
  \langle X = A, \text{open} \langle Y = X, a : Y \rangle \text{ as } \langle Z, z : Z \rangle \text{ in } z : Z \rangle
  \]

- \(Z\) escapes its scope!
Solution

• Turns out we can preserve typing and semantics if we add the following restriction

\[ \forall a \in P, \text{there are no type variables free in } a. \]

• Thus

\[ \langle X = A, \langle Y = X, a : Y \rangle. \text{snd} : \langle Y = X, a : Y \rangle. \text{Fst} \rangle \]

would not type-check
Extensions

• Thus, we can add support for paths in the grammar:

\[
\begin{align*}
  a & ::= \ldots \mid p.\text{snd} \mid \ldots \\
  A & ::= \ldots \mid p.\text{Fst} \mid \ldots \\
  p & ::= x \mid p.\text{snd}
\end{align*}
\]

• or even more elaborate abstractions like functors
Conclusions

• shown how to formalize an “open calculus” and a “dot calculus”

• both are equally expressive

• even with advanced abstractions like paths and functors
Questions?