Types, Abstraction, and Parametric Polymorphism

John C. Reynolds
Presented by Dietrich Geisler
Abstraction

Dietrich Geisler
(With apologies to John C. Reynolds)
What is Abstraction?

1. Define complex numbers
2. When are they equal?

1. Pairs of real numbers
2. Equality of components

1. Pairs of real numbers; first component is nonnegative
2. Equality of first component AND second component differs by multiple of $2\pi$
### Published in 1983

<table>
<thead>
<tr>
<th>Previous Papers:</th>
<th>Intel 80286 Processor:</th>
<th>Higher-level languages:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recursive Functions (1960)</td>
<td>10 MHz clock rate</td>
<td>Scheme</td>
</tr>
<tr>
<td>Axiomatic Basis (1969)</td>
<td>No memory cache</td>
<td>ML</td>
</tr>
<tr>
<td>CBN and CBV (1975)</td>
<td></td>
<td>C++</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1975</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1980</td>
</tr>
</tbody>
</table>
Sets and Types

If \( e_1 \in E_{\pi, \omega \rightarrow \omega'} \) and \( e_2 \in E_{\pi \omega} \) then

\[
e_1(e_2) \in E_{\pi \omega'}.,
\]

If \( e_1 \) has type \( \omega \rightarrow \omega' \) and \( e_2 \) has type \( \omega \)
Then the result of applying \( e_1 \) to \( e_2 \) has type \( \omega' \)
Some Notation

Extension to constants, pairs, and functions

e.g. \( S^# (\omega \times \omega') = S^#\omega \times S^#\omega' \)

Set Assignment
(e.g. \( S(\tau) = \{0, 1, 2\} \))

Extension to a context
(Works pointwise over the map)

\[
S^{#*}_{\pi} = \prod_{v \in \text{dom } \pi} S^#(\pi v).
\]
Some Semantics

If $k \in K_{\omega}$
then $\mu_{\pi \omega} \{ k \} S \eta = \alpha_{\omega} k$

If $v \in \text{dom } \pi$
then $\mu_{\pi, \pi v} \{ v \} S \eta = \eta v$

$\eta \vdash k : \omega$

$\eta \vdash \alpha_{\omega}(k)$

$\eta \vdash v : \omega$

$\eta \vdash \eta(v)$
Semantics of Pairs

If \( e \in E_{\pi \omega} \) and \( e' \in E_{\pi \omega} \), then

\[
\mu_{\pi, \omega \times \omega}, [<e, e'>] S \eta =
\langle \mu_{\pi \omega}[e] S \eta, \mu_{\pi \omega}[e'] S \eta \rangle
\]

where

\( \eta \vdash e_1 : \omega \quad \eta \vdash e_2 : \omega' \)

\[
\eta \vdash <e_1, e_2> : \omega \times \omega'
\]
How to compare set assignments?

Sets are related using pairs of set elements under $\text{Rel}(s_1, s_2)$

Functions and pairs are related if each component is related

$R$ is the pointwise relation between two set interpretations of types $S_1, S_2$
What is an Abstraction? (Formally)

**Abstraction Theorem** Let $R$ be a relation assignment between set assignments $S_1$ and $S_2$. For all $\pi \in \Omega^*$, $\omega \in \Omega$, $e \in E_{\pi\omega}$, and $<\eta_1, \eta_2> \in R#^*\pi$, 

$$<\mu_{\pi\omega}[e] S_1\eta_1, \mu_{\pi\omega}[e] S_2\eta_2> \in R#^\omega.$$ 

Evaluating expressions maps related arguments to related results
Extending this to a Typing Theorem

Pure Type Definition Theorem Let $S$ be a set assignment, $\omega_1, \omega_2 \in \Omega$, and $r$ be a relation between $S^{#\omega_1}$ and $S^{#\omega_2}$. For all $\pi \in \Omega^*, \tau \in T$, $\omega' \in \Omega$, $e \in E_{\pi-\tau, \omega'}$, and $\eta \in S^{#^*\pi}$,

\[
\begin{align*}
<\mu_{\pi}, (\omega'/\tau \rightarrow \omega_1) & \left[\text{lettype } \tau = \omega_1 \text{ in } e\right] S \eta, \\
\mu_{\pi}, (\omega'/\tau \rightarrow \omega_2) & \left[\text{lettype } \tau = \omega_2 \text{ in } e\right] S \eta > \\
\in [IA \mid \tau : r]^# \omega', 
\end{align*}
\]

where $IA$ is the relation assignment such that $IA \tau = I(S \tau)$ for all $\tau \in T$. 
What Happened to this work?

Some was folded into System F

Rust is starting to use some relational proofs

Ideas behind free theorems (e.g. properties $\lambda f : \alpha \to \alpha$?)