Refined Environment Classifiers

Type- and Scope-safe Code Generation with Mutable Cells

Oleg Kiselyov Yukiyoshi Kameyama Yuto Sudo

Tohoku University

University of Tsukuba

APLAS 2016

November 22, 2016

Region Memory Management for Free Variables

Type- and Scope-safe Code Generation with Mutable Cells

Oleg Kiselyov   Yukiyoshi Kameyama   Yuto Sudo

Tohoku University

University of Tsukuba

APLAS 2016
November 22, 2016

Summary

First stage calculus `<NJ>` for imperative code generators without ad hoc restrictions

- Store open code and retrieve in a different binding environment
- Proven sound type system: generated code is always well-typed and *well-scoped*
- Distillation of StagedHaskell

Practical

- Justification of (of a part of) StagedHaskell
- Easily embeddable (in OCaml) and can actually be used

Insightful

Insights

- Region-based memory management
- Contextual Modal Type Theory
- ML^F
- Overcoming the bureaucracy of syntax for names
- What is lexical scope, after all

Why code generation

Program like

$$y_k = \sum_{j=0}^{N-1} x_j e^{-2\pi ijk/N} \quad k = 0..N-1$$

but run like

```
fun x_35  →
  let t_36 = x_35.(0) +. x_35.(4) in
  let t_37 = x_35.(1) +. x_35.(5) in
  let t_38 = x_35.(0) −. x_35.(4) in
  let t_39 = x_35.(1) −. x_35.(5) in
  let t_40 = x_35.(2) +. x_35.(6) in
  let t_41 = x_35.(3) +. x_35.(7) in
  let t_42 = x_35.(2) −. x_35.(6) in
  let t_43 = x_35.(3) −. x_35.(7) in
  let t_44 = t_36 +. t_40 in
  let t_45 = t_37 +. t_41 in
  let t_46 = t_36 −. t_40 in
  let t_47 = t_37 −. t_41 in
...
```

Why code generation

```
fun x_35  →
  let t_36 = x_35.(0) +. x_35.(4) in
  let t_37 = x_35.(1) +. x_35.(5) in
  let t_38 = x_35.(0) −. x_35.(4) in
  let t_39 = x_35.(1) −. x_35.(5) in
  let t_40 = x_35.(2) +. x_35.(6) in
  let t_41 = x_35.(3) +. x_35.(7) in
  let t_42 = x_35.(2) −. x_35.(6) in
  let t_43 = x_35.(3) −. x_35.(7) in
  let t_44 = t_36 +. t_40 in
  let t_45 = t_37 +. t_41 in
  let t_46 = t_36 −. t_40 in
  let t_47 = t_37 −. t_41 in
...
```

- write – and re-write, and re-write,... generators
- some degree of correctness is needed: well-typedness and well-boundness

Is well-scopedness so important?

> *The re-factor solved performance issues in our use case of LMS which appeared due to the huge size of code we tend to generate. While we were able to resolve the performance issues, we introduced new bugs . . . [that] would manifest in errors such as:*
>
> ```
> forward reference extends over definition of value
> x1620 [error] val x1343 = x1232(x1123, x1124, x1180,
> x1181, x1223, x1224, x1223, x1229, x1216, x1120, x1122,
> x1121)
> ```
>
> *Note that variables are indexed in ascending order starting at zero, meaning that a large piece of code is processed before we hit this error. The root cause of bugs such as this one often proved to be very simple but heavily obfuscated in the code it manifested in. The concrete example was triggered by the code motion. . .*

RandIR: Differential Testing for Embedded Compilers.
Ofenbeck, Rompf, Püschel. Scala Symposium 2016.

Calculi for Code generation

- λ° (1996), λ^α (2003),...
- $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ (2000), Mint (2010),...
- pure freshML (2007),...

Calculi for Code generation

- λ° (1996), λ^{α} (2003),...
- $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ (2000), Mint (2010),...
- pure freshML (2007),...

- No effects

Calculi for Code generation

- λ° (1996), λ^α (2003),...
- $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ (2000), Mint (2010),...
- pure freshML (2007),...

- Only closed code can be stored

Calculi for Code generation

- λ° (1996), λ^α (2003),...
- $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ (2000), Mint (2010),...
- pure freshML (2007),...

- So complex it is not even implemented

Calculi for Code generation

- λ° (1996), λ^{α} (2003),...
- $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ (2000), Mint (2010),...
- pure freshML (2007),...

- Can emulate mutation/control effects with state-passing, CPS?

Calculi for Code generation

- λ° (1996), λ^{α} (2003),...
- $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ (2000), Mint (2010),...
- pure freshML (2007),...

- Can emulate mutation/control effects with state-passing, CPS?
- Yes, but we can't do code movement across binders

<NJ> by Example

<NJ> is the standard CBV λ-calculus with constants for code-generation

$$\text{power } x\ n = x^n$$

let body = λf n x. **if** n=0 **then** $\underline{\text{cint}}$ 1 **else** x $\underline{*}$ f (n−1) x **in**
let power = λn. $\underline{\lambda}$x. (fix body) n x **in** power 2

<NJ> by Example

$\textbf{let}\ \mathsf{body} = \lambda \mathsf{f\ n\ x.}\ \textbf{if}\ \mathsf{n{=}0}\ \textbf{then}\ \underline{\mathsf{cint}}\ \mathsf{1}\ \textbf{else}\ \mathsf{x}\ \underline{*}\ \mathsf{f\ (n{-}1)\ x}\ \textbf{in}$
$\textbf{let}\ \mathsf{power} = \lambda \mathsf{n.}\ \underline{\lambda} \mathsf{x.\ (fix\ body)\ n\ x}\ \textbf{in}\ \mathsf{power\ 2}$

$\rightsquigarrow^*\ \textbf{let}\ \mathsf{body} = \ldots\ \textbf{in}\ \underline{\lambda}\mathsf{x.\ (fix\ body)\ 2\ x}$

<NJ> by Example

$\textbf{let}\ \text{body} = \lambda \text{f n x}.\ \textbf{if}\ \text{n}=0\ \textbf{then}\ \underline{\text{cint}}\ 1\ \textbf{else}\ \text{x} \mathbin{\underline{*}} \text{f}\ (\text{n}-1)\ \text{x}\ \textbf{in}$
$\textbf{let}\ \text{power} = \lambda \text{n}.\ \underline{\lambda}\text{x}.\ (\text{fix}\ \text{body})\ \text{n x}\ \textbf{in}\ \text{power}\ 2$

$\rightsquigarrow^* \textbf{let}\ \text{body} = \ldots\ \textbf{in}\ \underline{\lambda}\text{x}.\ (\text{fix body})\ 2\ \text{x}$

$\rightsquigarrow\ \textbf{let}\ \text{body} = \ldots\ \textbf{in}\ \underline{\underline{\lambda}}\text{y}.(\text{fix}\ \text{body})\ 2\ \langle \text{y} \rangle$

<NJ> by Example

$\textbf{let}$ body $= \lambda$f n x. $\textbf{if}$ n=0 $\textbf{then}$ $\underline{\text{cint}}$ 1 $\textbf{else}$ x $\underline{*}$ f (n−1) x $\textbf{in}$
$\textbf{let}$ power $= \lambda$n. $\underline{\lambda}$x. (fix body) n x $\textbf{in}$ power 2

$\rightsquigarrow^*$ $\textbf{let}$ body $= \ldots$ $\textbf{in}$ $\underline{\lambda}$x. (fix body) 2 x

$\rightsquigarrow$ $\textbf{let}$ body $= \ldots$ $\textbf{in}$ $\underline{\underline{\lambda}}$y.(fix body) 2 $\langle$y$\rangle$

$\rightsquigarrow$ $\textbf{let}$ body $= \ldots$ $\textbf{in}$
$\underline{\underline{\lambda}}$y. $\textbf{if}$ 2=0 $\textbf{then}$ $\underline{\text{cint}}$ 1 $\textbf{else}$ $\langle$y$\rangle$ $\underline{*}$ (fix body) 1 $\langle$y$\rangle$

<NJ> by Example

$\textbf{let}\ \ \mathsf{body} = \lambda \mathsf{f\ n\ x.}\ \ \textbf{if}\ \ \mathsf{n{=}0}\ \textbf{then}\ \underline{\mathsf{cint}}\ 1\ \textbf{else}\ \ \mathsf{x}\ \underline{*}\ \ \mathsf{f}\ (\mathsf{n}-1)\ \mathsf{x}\ \textbf{in}$
$\textbf{let}\ \ \mathsf{power} = \lambda \mathsf{n.}\ \underline{\lambda} \mathsf{x.}\ (\,\mathsf{fix}\ \ \mathsf{body})\ \mathsf{n\ x}\ \textbf{in}\ \ \mathsf{power}\ 2$

$\rightsquigarrow^*\ \textbf{let}\ \ \mathsf{body} = \ldots\ \textbf{in}\ \underline{\lambda}\mathsf{x.}\ (\mathsf{fix\ body})\ 2\ \mathsf{x}$

$\rightsquigarrow\ \ \textbf{let}\ \ \mathsf{body} = \ldots\ \textbf{in}\ \underline{\underline{\lambda}}\mathsf{y.}(\,\mathsf{fix}\ \ \mathsf{body})\ 2\ \langle \mathsf{y} \rangle$

$\rightsquigarrow\ \ \textbf{let}\ \ \mathsf{body} = \ldots\ \textbf{in}$
$\underline{\underline{\lambda}}\mathsf{y.}\ \ \textbf{if}\ \ 2{=}0\ \textbf{then}\ \underline{\mathsf{cint}}\ 1\ \textbf{else}\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ (\,\mathsf{fix}\ \ \mathsf{body})\ 1\ \langle \mathsf{y} \rangle$

$\rightsquigarrow^*\ \underline{\underline{\lambda}}\mathsf{y.}\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \underline{\mathsf{cint}}\ \ 1$

<NJ> by Example

$\textbf{let}\ \ \mathsf{body} = \lambda \mathsf{f\ n\ x.}\ \ \textbf{if}\ \ \mathsf{n{=}0}\ \textbf{then}\ \underline{\mathsf{cint}}\ 1\ \textbf{else}\ \ \mathsf{x}\ \underline{*}\ \ \mathsf{f}\ (\mathsf{n}-1)\ \mathsf{x}\ \textbf{in}$
$\textbf{let}\ \ \mathsf{power} = \lambda \mathsf{n}.\ \underline{\lambda} \mathsf{x}.\ (\,\mathsf{fix}\ \ \mathsf{body})\ \mathsf{n\ x}\ \textbf{in}\ \ \mathsf{power}\ 2$

$\rightsquigarrow^*\ \textbf{let}\ \ \mathsf{body} = \ldots\ \textbf{in}\ \underline{\lambda} \mathsf{x}.\ (\mathsf{fix\ body})\ 2\ \mathsf{x}$

$\rightsquigarrow\ \ \textbf{let}\ \ \mathsf{body} = \ldots\ \textbf{in}\ \underline{\underline{\lambda}} \mathsf{y}.(\,\mathsf{fix}\ \ \mathsf{body})\ 2\ \langle \mathsf{y} \rangle$

$\rightsquigarrow\ \ \textbf{let}\ \ \mathsf{body} = \ldots\ \textbf{in}$
$\underline{\underline{\lambda}} \mathsf{y}.\ \ \textbf{if}\ \ 2{=}0\ \textbf{then}\ \underline{\mathsf{cint}}\ 1\ \textbf{else}\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ (\,\mathsf{fix}\ \ \mathsf{body})\ 1\ \langle \mathsf{y} \rangle$

$\rightsquigarrow^*\ \underline{\underline{\lambda}} \mathsf{y}.\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \underline{\mathsf{cint}}\ \ 1$

$\rightsquigarrow\ \ \underline{\underline{\lambda}} \mathsf{y}.\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \langle 1 \rangle$

<NJ> by Example

$$\textbf{let}\ \ \mathsf{body} = \lambda \mathsf{f\ n\ x}.\ \ \textbf{if}\ \ \mathsf{n{=}0}\ \textbf{then}\ \underline{\mathsf{cint}}\ 1\ \textbf{else}\ \ \mathsf{x}\ \underline{*}\ \ \mathsf{f}\ \ (\mathsf{n}-1)\ \mathsf{x}\ \textbf{in}$$
$$\textbf{let}\ \ \mathsf{power} = \lambda \mathsf{n}.\ \underline{\lambda} \mathsf{x}.\ (\ \mathsf{fix}\ \ \mathsf{body})\ \mathsf{n\ x}\ \textbf{in}\ \ \mathsf{power}\ 2$$

$$\rightsquigarrow^*\ \textbf{let}\ \ \mathsf{body} = \ldots\ \textbf{in}\ \underline{\lambda} \mathsf{x}.\ (\mathsf{fix\ body})\ 2\ \mathsf{x}$$

$$\rightsquigarrow\ \ \textbf{let}\ \ \mathsf{body} = \ldots\ \textbf{in}\ \underline{\underline{\lambda}} \mathsf{y}.(\ \mathsf{fix}\ \ \mathsf{body})\ 2\ \langle \mathsf{y} \rangle$$

$$\rightsquigarrow\ \ \textbf{let}\ \ \mathsf{body} = \ldots\ \textbf{in}$$
$$\underline{\underline{\lambda}} \mathsf{y}.\ \ \textbf{if}\ \ 2{=}0\ \textbf{then}\ \underline{\mathsf{cint}}\ 1\ \textbf{else}\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ (\ \mathsf{fix}\ \ \mathsf{body})\ 1\ \langle \mathsf{y} \rangle$$

$$\rightsquigarrow^*\ \underline{\underline{\lambda}} \mathsf{y}.\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \underline{\mathsf{cint}}\ \ 1$$

$$\rightsquigarrow\ \ \underline{\underline{\lambda}} \mathsf{y}.\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \langle 1 \rangle$$

$$\rightsquigarrow\ \ \underline{\underline{\lambda}} \mathsf{y}.\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \langle \mathsf{y}\ *\ 1 \rangle$$
$$\rightsquigarrow^*\ \underline{\underline{\lambda}} \mathsf{y}.\ \ \langle \mathsf{y}\ *\ \mathsf{y}\ *\ 1 \rangle$$

<NJ> by Example

$$\textbf{let}\ \ \mathsf{body} = \lambda \mathsf{f\ n\ x.}\ \ \textbf{if}\ \ \mathsf{n{=}0}\ \textbf{then}\ \underline{\mathsf{cint}}\ 1\ \textbf{else}\ \ \mathsf{x}\ \underline{*}\ \ \mathsf{f}\ (\mathsf{n}-1)\ \mathsf{x}\ \textbf{in}$$
$$\textbf{let}\ \ \mathsf{power} = \lambda \mathsf{n.}\ \underline{\lambda}\mathsf{x.}\ (\,\mathsf{fix\ \ body})\ \mathsf{n\ x}\ \textbf{in}\ \ \mathsf{power}\ 2$$

$$\leadsto^{*}\ \textbf{let}\ \ \mathsf{body} = \ldots\ \textbf{in}\ \underline{\lambda}\mathsf{x.}\ (\mathsf{fix\ body})\ 2\ \mathsf{x}$$

$$\leadsto\ \ \textbf{let}\ \ \mathsf{body} = \ldots\ \textbf{in}\ \underline{\underline{\lambda}}\mathsf{y.}(\,\mathsf{fix\ \ body})\ 2\ \langle \mathsf{y} \rangle$$

$$\leadsto\ \ \textbf{let}\ \ \mathsf{body} = \ldots\ \textbf{in}$$
$$\underline{\underline{\lambda}}\mathsf{y.}\ \ \textbf{if}\ \ 2{=}0\ \textbf{then}\ \underline{\mathsf{cint}}\ 1\ \textbf{else}\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ (\,\mathsf{fix\ \ body})\ 1\ \langle \mathsf{y} \rangle$$

$$\leadsto^{*}\ \underline{\underline{\lambda}}\mathsf{y.}\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \underline{\mathsf{cint}}\ \ 1$$

$$\leadsto\ \ \underline{\underline{\lambda}}\mathsf{y.}\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \langle 1 \rangle$$

$$\leadsto\ \ \underline{\underline{\lambda}}\mathsf{y.}\ \ \langle \mathsf{y} \rangle\ \ \underline{*}\ \ \langle \mathsf{y} * 1 \rangle$$
$$\leadsto^{*}\ \underline{\underline{\lambda}}\mathsf{y.}\ \ \langle \mathsf{y} * \mathsf{y} * 1 \rangle$$

$$\leadsto\ \ \langle \lambda \mathsf{y.}\ \ \mathsf{y} * \mathsf{y} * 1 \rangle$$

Generating a function takes two steps:

- Generate a variable name
- eventually, generate a binder for it

State Power

$\textbf{let}\ \mathsf{body} = \lambda \mathsf{n}.\lambda \mathsf{x}.\ \textbf{let}\ \mathsf{r} = \textbf{ref}\ (\underline{\mathsf{cint}}\ 1)\ \textbf{in}$
$\quad \mathsf{fix}\ (\lambda \mathsf{f}.\lambda \mathsf{n}.\ \textbf{if}\ \mathsf{n} = 0\ \textbf{then}\ 0\ \textbf{else}\ (\mathsf{r} := !\mathsf{r}\ \underline{*}\ \mathsf{x};\ \mathsf{f}\ (\mathsf{n}-1)))\ \mathsf{n};\ !\mathsf{r}\ \textbf{in}$
$\textbf{let}\ \mathsf{power} = \lambda \mathsf{n}.\ \underline{\lambda} \mathsf{x}.\ \mathsf{body}\ \mathsf{n}\ \mathsf{x}\ \textbf{in}\ \mathsf{power}\ 2$

State Power

$$\textbf{let}\ \ \mathsf{body} = \lambda \mathsf{n}.\lambda \mathsf{x}.\ \textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ (\underline{\mathsf{cint}}\ \ 1)\ \ \textbf{in}$$
$$\mathsf{fix}\ \ (\lambda \mathsf{f}.\lambda \mathsf{n}.\ \ \textbf{if}\ \ \mathsf{n} = 0\ \textbf{then}\ 0\ \textbf{else}\ (\mathsf{r}\ := !\mathsf{r}\ \underline{*}\ \ \mathsf{x};\ \ \mathsf{f}\ \ (\mathsf{n}-1)))\ \mathsf{n};\ !\mathsf{r}\ \ \textbf{in}$$
$$\textbf{let}\ \ \mathsf{power} = \lambda \mathsf{n}.\ \underline{\lambda} \mathsf{x}.\ \mathsf{body}\ \mathsf{n}\ \mathsf{x}\ \textbf{in}\ \ \mathsf{power}\ 2$$

$$\rightsquigarrow^*\ \textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \langle 1 \rangle\ \ \textbf{in}$$
$$\underline{\underline{\lambda}} \mathsf{y}.\ (\ \textbf{if}\ \ 2 = 0\ \textbf{then}\ 0\ \textbf{else}\ (\mathsf{r}\ := !\mathsf{r}\ \underline{*}\ \langle \mathsf{y} \rangle\ \ ;\ \ \mathsf{fix}\ \ \mathsf{f}\ \ 1);\ \ !\mathsf{r})$$

State Power

$\textbf{let}\ \ \mathsf{body} = \lambda \mathsf{n}.\lambda \mathsf{x}.\ \textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ (\underline{\mathsf{cint}}\ \ 1)\ \ \textbf{in}$
$\mathsf{fix}\ \ (\lambda \mathsf{f}.\lambda \mathsf{n}.\ \ \textbf{if}\ \ \mathsf{n} = 0\ \textbf{then}\ 0\ \textbf{else}\ (\mathsf{r}\ := !\mathsf{r}\ \underline{*}\ \ \mathsf{x};\ \ \mathsf{f}\ \ (\mathsf{n}-1)))\ \mathsf{n};\ !\mathsf{r}\ \ \textbf{in}$
$\textbf{let}\ \ \mathsf{power} = \lambda \mathsf{n}.\ \underline{\lambda} \mathsf{x}.\ \mathsf{body}\ \mathsf{n}\ \mathsf{x}\ \textbf{in}\ \ \mathsf{power}\ 2$

$\rightsquigarrow^*\ \textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \langle 1 \rangle\ \ \textbf{in}$
$\underline{\underline{\lambda}} \mathsf{y}.\ (\textbf{if}\ \ 2 = 0\ \textbf{then}\ 0\ \textbf{else}\ (\mathsf{r}\ := !\mathsf{r}\ \underline{*}\ \langle \mathsf{y} \rangle\ \ ;\ \ \mathsf{fix}\ \ \mathsf{f}\ \ 1);\ \ !\mathsf{r})$

$\rightsquigarrow^*\ \textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \langle 1 * \mathsf{y} \rangle\ \ \ \textbf{in}$
$\underline{\underline{\lambda}} \mathsf{y}.\ (\textbf{if}\ \ 1 = 0\ \textbf{then}\ 0\ \textbf{else}\ (\mathsf{r}\ := !\mathsf{r}\ \underline{*}\ \langle \mathsf{y} \rangle;\ \ \mathsf{fix}\ \ \mathsf{f}\ \ 1);\ \ !\mathsf{r})$

State Power

$\textbf{let}\ \ \mathsf{body} = \lambda \mathsf{n}.\lambda \mathsf{x}.\ \textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ (\underline{\mathsf{cint}}\ \ 1)\ \ \textbf{in}$
$\quad \mathsf{fix}\ \ (\lambda \mathsf{f}.\lambda \mathsf{n}.\ \ \textbf{if}\ \ \mathsf{n} = 0\ \textbf{then}\ 0\ \textbf{else}\ (\mathsf{r}\ := !\mathsf{r}\ \underline{*}\ \ \mathsf{x};\ \ \mathsf{f}\ \ (\mathsf{n}-1)))\ \mathsf{n};\ !\mathsf{r}\ \ \textbf{in}$
$\textbf{let}\ \ \mathsf{power} = \lambda \mathsf{n}.\ \underline{\lambda} \mathsf{x}.\ \mathsf{body}\ \mathsf{n}\ \mathsf{x}\ \textbf{in}\ \ \mathsf{power}\ 2$

$\rightsquigarrow^*\ \textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \langle 1 \rangle\ \ \textbf{in}$
$\underline{\underline{\lambda}}\mathsf{y}.\ (\,\textbf{if}\ \ 2 = 0\ \textbf{then}\ 0\ \textbf{else}\ (\mathsf{r}\ := !\mathsf{r}\ \underline{*}\ \langle \mathsf{y} \rangle\ \ ;\ \ \mathsf{fix}\ \ \mathsf{f}\ \ 1);\ \ !\mathsf{r})$

$\rightsquigarrow^*\ \textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \langle 1 * \mathsf{y} \rangle\ \ \ \textbf{in}$
$\underline{\underline{\lambda}}\mathsf{y}.\ (\,\textbf{if}\ \ 1 = 0\ \textbf{then}\ 0\ \textbf{else}\ (\mathsf{r}\ := !\mathsf{r}\ \underline{*}\ \ \langle \mathsf{y} \rangle;\ \ \mathsf{fix}\ \ \mathsf{f}\ \ 1);\ \ !\mathsf{r})$

$\rightsquigarrow^*\ \textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \langle 1 * \mathsf{y} * \mathsf{y} \rangle\ \ \ \textbf{in}\ \underline{\underline{\lambda}}\mathsf{y}.\ \ !\mathsf{r}$

State Power

$$\textbf{let}\ \mathsf{body} = \lambda n.\lambda x.\ \textbf{let}\ r = \textbf{ref}\ (\underline{\mathsf{cint}}\ 1)\ \textbf{in}$$
$$\mathsf{fix}\ (\lambda f.\lambda n.\ \textbf{if}\ n = 0\ \textbf{then}\ 0\ \textbf{else}\ (r := !r \mathbin{\underline{*}} x;\ f\ (n-1)))\ n;\ !r\ \textbf{in}$$
$$\textbf{let}\ \mathsf{power} = \lambda n.\ \underline{\lambda} x.\ \mathsf{body}\ n\ x\ \textbf{in}\ \mathsf{power}\ 2$$

$$\rightsquigarrow^{*}\ \textbf{let}\ r = \textbf{ref}\ \langle 1 \rangle\ \textbf{in}$$
$$\underline{\underline{\lambda}} y.\ (\textbf{if}\ 2 = 0\ \textbf{then}\ 0\ \textbf{else}\ (r := !r \mathbin{\underline{*}} \langle y \rangle\ ;\ \mathsf{fix}\ f\ 1);\ !r)$$

$$\rightsquigarrow^{*}\ \textbf{let}\ r = \textbf{ref}\ \langle 1 * y \rangle\ \textbf{in}$$
$$\underline{\underline{\lambda}} y.\ (\textbf{if}\ 1 = 0\ \textbf{then}\ 0\ \textbf{else}\ (r := !r \mathbin{\underline{*}} \langle y \rangle;\ \mathsf{fix}\ f\ 1);\ !r)$$

$$\rightsquigarrow^{*}\ \textbf{let}\ r = \textbf{ref}\ \langle 1 * y * y \rangle\ \textbf{in}\ \underline{\underline{\lambda}} y.\ !r$$

$$\rightsquigarrow\ \underline{\underline{\lambda}} y.\ \langle 1 * y * y \rangle\ \rightsquigarrow\ \langle \lambda y.\ 1 * y * y \rangle$$

Too much of the State Power

$$\textbf{let}\ \ \mathsf{r}\ =\textbf{ref}\ \underline{\mathsf{cint}}\ \ 0\ \textbf{in}\ \ (\underline{\lambda}\mathsf{x}.\ \ \mathsf{r}\ :=\mathsf{x});\ \ !\,\mathsf{r}$$

Too much of the State Power

$$\textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \underline{\mathsf{cint}}\ \ 0\ \textbf{in}\ (\underline{\lambda}\mathsf{x}.\ \mathsf{r}\ := \mathsf{x});\ \ !\,\mathsf{r}$$

$$\rightsquigarrow\ \ \textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \underline{\mathsf{cint}}\ \ 0\ \textbf{in}\ (\underline{\underline{\lambda}}\mathsf{y}.\ \mathsf{r}\ := \langle \mathsf{y} \rangle\,);\ \ !\,\mathsf{r}$$

Too much of the State Power

$$\textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \underline{\mathsf{cint}}\ \ 0\ \textbf{in}\ \ (\underline{\lambda}\mathsf{x}.\ \ \mathsf{r}\ := \mathsf{x});\ \ !\,\mathsf{r}$$

$$\rightsquigarrow\ \ \textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \underline{\mathsf{cint}}\ \ 0\ \textbf{in}\ \ (\underline{\underline{\lambda}}\mathsf{y}.\ \ \mathsf{r}\ := \langle \mathsf{y} \rangle\,);\ \ !\,\mathsf{r}$$

$$\rightsquigarrow\ \ \textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \ \ \langle \mathsf{y} \rangle\ \ \ \textbf{in}\ \ (\underline{\underline{\lambda}}\mathsf{y}.\ \langle \mathsf{y} \rangle\,);\ \ !\,\mathsf{r}$$

Too much of the State Power

$$\textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \underline{\mathsf{cint}}\ \ 0\ \textbf{in}\ \ (\underline{\lambda}\mathsf{x}.\ \ \mathsf{r}\ := \mathsf{x});\ \ !\,\mathsf{r}$$

$$\leadsto\ \ \textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \underline{\mathsf{cint}}\ \ 0\ \textbf{in}\ \ (\underline{\underline{\lambda}}\mathsf{y}.\ \ \mathsf{r}\ := \langle \mathsf{y}\rangle\,);\ \ !\,\mathsf{r}$$

$$\leadsto\ \ \textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \ \langle \mathsf{y}\rangle\ \ \textbf{in}\ \ (\underline{\underline{\lambda}}\mathsf{y}.\ \langle \mathsf{y}\rangle\,);\ \ !\,\mathsf{r}$$

$$\leadsto\ \ \textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \ \langle \mathsf{y}\rangle\ \ \textbf{in}\ \ !\,\mathsf{r}$$

Too much of the State Power

$$\textbf{let } \mathsf{r} = \textbf{ref } \underline{\mathsf{cint}}\ 0 \textbf{ in } (\underline{\lambda}\mathsf{x}.\ \mathsf{r} := \mathsf{x});\ !\mathsf{r}$$

$$\rightsquigarrow \textbf{let } \mathsf{r} = \textbf{ref } \underline{\mathsf{cint}}\ 0 \textbf{ in } (\underline{\underline{\lambda}}\mathsf{y}.\ \mathsf{r} := \langle \mathsf{y} \rangle);\ !\mathsf{r}$$

$$\rightsquigarrow \textbf{let } \mathsf{r} = \textbf{ref } \langle \mathsf{y} \rangle \textbf{ in } (\underline{\underline{\lambda}}\mathsf{y}.\ \langle \mathsf{y} \rangle);\ !\mathsf{r}$$

$$\rightsquigarrow \textbf{let } \mathsf{r} = \textbf{ref } \langle \mathsf{y} \rangle \textbf{ in } !\mathsf{r}$$

$$\rightsquigarrow \textbf{let } \mathsf{r} = \textbf{ref } \langle \mathsf{y} \rangle \textbf{ in } \langle \mathsf{y} \rangle$$

Type System (outline)

$$\vdash \langle \mathsf{y} \rangle : \; ??$$

Type System (outline)

$\Gamma \vdash \langle \mathsf{y} \rangle : \langle \mathsf{int} \rangle^{??}$ where $\mathsf{y{:}int} \in \Gamma$

We are manipulating open code: cf. "Open CBV" yesterday

Annotate the type of a code value with *some form* of Γ

- size of Γ
- Γ itself (CMTT)
- Γ (without names, just a sequence of types)

Annotated code types by example

$\textbf{let}\ \mathsf{r}\ = \textbf{ref}\ \underline{\mathsf{cint}}\ 0\ \textbf{in}\ (\underline{\lambda}\mathsf{z}.\ \underline{\lambda}\mathsf{x}.\ \mathsf{r} := \mathsf{x});\ !\mathsf{r}\ :\ <\!\mathsf{int}\!>^{(int,(bool,()))}$

$\Gamma \vdash \mathsf{z}\!:\ <\!\mathsf{bool}\!>^{(bool,())}$
$\Gamma \vdash \mathsf{x}\!:\ <\!\mathsf{int}\!>^{(int,(bool,()))}$
$\Gamma \vdash \mathsf{r}\!:\ <\!\mathsf{int}\!>^{(int,(bool,()))}\ \textbf{ref}$

Annotated code types by example

$\textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \underline{\mathsf{cint}}\ \ 0\ \textbf{in}\ (\underline{\lambda}\mathsf{z}.\ \underline{\lambda}\mathsf{x}.\ \mathsf{r} := \mathsf{x});\ !\mathsf{r}\ :\ <\!\mathsf{int}\!>^{(int,(bool,()))}$

$\Gamma \vdash \mathsf{z}\!:\ <\!\mathsf{bool}\!>^{(bool,())}$
$\Gamma \vdash \mathsf{x}\!:\ <\!\mathsf{int}\!>^{(int,(bool,()))}$
$\Gamma \vdash \mathsf{r}\!:\ <\!\mathsf{int}\!>^{(int,(bool,()))}\ \textbf{ref}$

However,
$\textbf{let}\ \ \mathsf{r}\ = \textbf{ref}\ \underline{\mathsf{cint}}\ \ 0\ \textbf{in}\ (\underline{\lambda}\mathsf{z}.\ \underline{\lambda}\mathsf{x}.\ \mathsf{r} := \mathsf{x});\ (\underline{\lambda}\mathsf{y}.\ \underline{\lambda}\mathsf{u}.\ !\mathsf{r})$
$\leadsto^{*}\ \langle \lambda\mathsf{y}.\ \lambda\mathsf{u}.\ \mathsf{x}\rangle\ :\ <\!\mathsf{int}\!\to\!\mathsf{int}\!\to\!\mathsf{int}\!>^{()}$
According to the type system, the result is closed.

Taste of a Type System

$$\frac{\gamma \in \Gamma \qquad \gamma_1 \notin \Gamma \qquad \Gamma,\ \gamma_1,\ (\gamma_1 \succ \gamma),\ (\mathsf{x}{:}\langle \mathsf{t}_1 \rangle^{\gamma_1}) \vdash \mathsf{e}{:}\ \langle \mathsf{t}_2 \rangle^{\gamma_1}}{\Gamma \vdash \underline{\lambda}\mathsf{x.e}{:}\ \langle \mathsf{t}_1 {\rightarrow} \mathsf{t}_2 \rangle^{\gamma}}\ \text{CAbs}$$

Taste of a Type System

$$
\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{\Gamma_2 \vdash \mathsf{x}_1\!:\ \langle \mathsf{int}\rangle^{\gamma_1}} \quad \Gamma_2 \models \gamma_2 \succ \gamma_1}{\Gamma_2 \vdash \mathsf{x}_1\!:\ \langle \mathsf{int}\rangle^{\gamma_2}} \qquad \cfrac{}{\Gamma_2 \vdash \mathsf{x}_2\!:\ \langle \mathsf{int}\rangle^{\gamma_2}}}{\Gamma_2 \vdash \mathsf{x}_1 \ \underline{+}\ \mathsf{x}_2\!:\ \langle \mathsf{int}\rangle^{\gamma_2}}}{\gamma_1,\ (\gamma_1 \succ \gamma_0),\ (\mathsf{x}_1\!:\!\langle \mathsf{int}\rangle^{\gamma_1}) \vdash \underline{\lambda}\mathsf{x}_2.\ \mathsf{x}_1 \ \underline{+}\ \mathsf{x}_2\!:\ \langle \mathsf{int}\!\to\!\mathsf{int}\rangle^{\gamma_1}}}{[]\ \vdash \underline{\lambda}\mathsf{x}_1.\underline{\lambda}\mathsf{x}_2.\ \mathsf{x}_1 \ \underline{+}\ \mathsf{x}_2 \ :\ \langle \mathsf{int}\!\to\!\mathsf{int}\!\to\!\mathsf{int}\rangle^{\gamma_0}}
$$

Taste of a Type System

$$\dfrac{\dfrac{\mathsf{r} : \langle \mathsf{int} \rangle^{\gamma_1},\ \gamma_2, \gamma_2 \succ \gamma,\ \mathsf{x}{:}\langle \mathsf{int} \rangle^{\gamma_2} \vdash \mathsf{r} := \mathsf{x} : \langle \mathsf{int} \rangle^{\gamma_2}}{\mathsf{r} : \langle \mathsf{int} \rangle^{\gamma_1}\ \textbf{ref} \vdash (\underline{\lambda}\mathsf{x}.\ \mathsf{r} := \mathsf{x}) : \langle \mathsf{int} \to \mathsf{int} \rangle^{\gamma}}}{[\,] \vdash \textbf{let}\ \mathsf{r} = \textbf{ref}\ \underline{\mathsf{cint}}\ 0\ \textbf{in}\ (\underline{\lambda}\mathsf{x}.\ \mathsf{r} := \mathsf{x}) : \langle \mathsf{int} \to \mathsf{int} \rangle^{\gamma}}$$

Region memory management

Summary

First stage calculus `<NJ>` for imperative code generators without ad hoc restrictions

- Store open code and retrieve in a different binding environment
- Sound type system: generated code is always well-typed and *well-scoped*
- Distillation of StagedHaskell

Practical

- Justification of (of a part of) StagedHaskell
- Easily embeddable (in OCaml) and can actually be used

Insightful