Witnessing Purity, Constancy and Mutability

SAPLING 2009/10/2

Ben Lippmeier

Australian National University

Haskell lacks mutability polymorphism

$$\begin{aligned}&\textbf{data}\ \mathit{List}\ a\\&\qquad = \mathit{Nil}\\&\qquad \mid\ \mathit{Cons}\ a\ (\mathit{List}\ a)\end{aligned}$$

$$\begin{aligned}&\textbf{data}\ \mathit{MutableList}\ a\\&\qquad = \mathit{MNil}\\&\qquad \mid\ \mathit{MCons}\ a\ (\mathit{IORef}\ (\mathit{MutableList}\ a))\end{aligned}$$

- These are incompatible data types.
- The first is good for optimisation, but can't be destructively updated.
- I'm tired of refactoring code to use one or the other.

Adding region parameters

$$\begin{aligned}
&\textbf{data}\ \mathit{List}\ r\ a \\
&\qquad = \mathit{Nil} \\
&\qquad \mid\ \mathit{Cons}\ a\ (\mathit{List}\ r\ a)
\end{aligned}$$

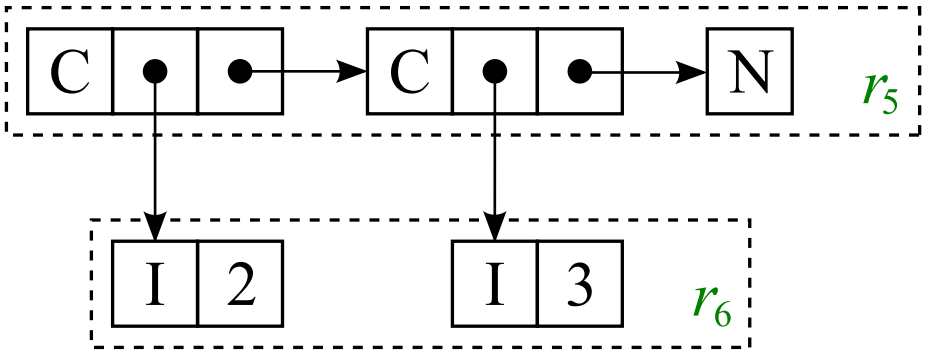
$$\begin{aligned}
\mathit{Nil} &:: \forall (r : \textbf{region})\ (a : \textbf{type}).\ \mathit{List}\ r\ a \\
\mathit{Cons} &:: \forall (r : \textbf{region})\ (a : \textbf{type}).\ a \to \mathit{List}\ r\ a \to \mathit{List}\ r\ a
\end{aligned}$$

- The region parameter represents *where* the data is stored.
- *Nil* is a function that can allocate an object into any region.

All list cells are in the same region

$$Nil \quad :: \forall(r : \textbf{region})\ (a : \textbf{type}).\ List\ r\ a$$
$$Cons \quad :: \forall(r : \textbf{region})\ (a : \textbf{type}).\ a \to List\ r\ a \to List\ r\ a$$

$$list :: List\ r_5\ (Int\ r_6)$$
$$list = Cons\ r_5\ \ (Int\ r_6)\ (2\ r_6)$$
$$(Cons\ r_5\ \ (Int\ r_6)\ (3\ r_6)$$
$$(Nil\ r_5\ (Int\ r_6)))$$



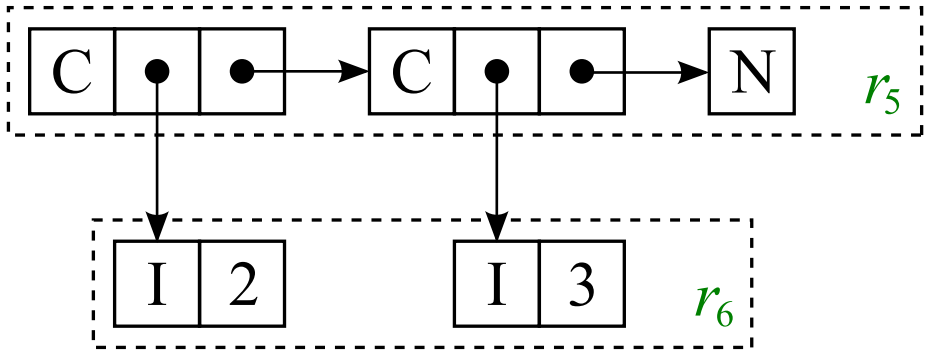
Region constraints provide mutability polymorphism

$$list_m :: Mutable\ r_5 \Rightarrow List\ r_5\ (Int\ r_6)$$

$$list_c :: Const\ r_5 \Rightarrow List\ r_5\ (Int\ r_6)$$

$$list_{cm} :: Const\ r_5 \Rightarrow Mutable\ r_6 \Rightarrow List\ r_5\ (Int\ r_6)$$

$$list_{mc} :: Mutable\ r_5 \Rightarrow Const\ r_6 \Rightarrow List\ r_5\ (Int\ r_6)$$



letregion introduces new regions

letregion r_1 **in**
printInt r_1 $(5\ r_1)$

$$printInt \quad :: \forall(r_1 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1 \vee Console} ()$$

Evaluation

letregion r_1 **in**
printInt r_1 $(5\ r_1)$

HEAP

$$printInt \quad :: \forall(r_1 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1 \vee Console} ()$$

Evaluation (allocate region $r_1 \sim \rho_1$)

$\longrightarrow$ *printInt* $\underline{\rho_1}$ $(5\ \underline{\rho_1})$

HEAP

$\rho_1 : \emptyset$

printInt $:: \forall(r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} ()$

Evaluation (allocate object at location l_1)

$\longrightarrow$ *printInt* $\underline{\rho_1}$ $\underline{l_1}$

HEAP

$\rho_1 : \{\ l_1 \mapsto 5\ \}$

printInt $:: \forall(r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} ()$

Evaluation (print)

$\longrightarrow\ ()$

HEAP

$\rho_1 : \{\ l_1 \mapsto 5\ \}$

"5"

$printInt \quad :: \forall(r_1 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1 \vee Console} ()$

Updating Integers

$$
\begin{array}{l}
\textbf{letregion}\ r_1\ \textbf{in} \\
\textbf{letregion}\ r_2\ \textbf{in} \\
\textbf{do}\ \ x = 5\ r_1 \\
\qquad \mathit{updateInt}\ r_1\ r_2\ \ \ldots\ \ x\ (23\ r_2) \\
\qquad \mathit{printInt}\ r_1\ x
\end{array}
$$

$$
\begin{array}{ll}
\mathit{printInt} & :: \forall (r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} () \\
\mathit{updateInt} & :: \forall (r_1, r_2 : \textbf{region}).\ \mathit{Mutable}\ r_1 \Rightarrow \mathit{Int}\ r_1 \rightarrow \mathit{Int}\ r_2 \xrightarrow{\mathit{Read}\ r_2 \vee \mathit{Write}\ r_1} ()
\end{array}
$$

The mutability of r_1 is witnessed by w_1

$$
\begin{array}{l}
\textbf{letregion } r_1 \textbf{ where } w_1 = \mathit{MkMutable}\ r_1 \textbf{ in} \\
\textbf{letregion } r_2 \textbf{ in} \\
\textbf{do } \ x = 5\ r_1 \\
\qquad \mathit{updateInt}\ r_1\ r_2\ w_1\ x\ (23\ r_2) \\
\qquad \mathit{printInt}\ r_1\ x
\end{array}
$$

$$
\begin{array}{ll}
\mathit{printInt} & :: \forall (r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} () \\
\mathit{updateInt} & :: \forall (r_1, r_2 : \textbf{region}).\ \mathit{Mutable}\ r_1 \Rightarrow \mathit{Int}\ r_1 \rightarrow \mathit{Int}\ r_2 \xrightarrow{\mathit{Read}\ r_2 \vee \mathit{Write}\ r_1} ()
\end{array}
$$

MkMutable has a dependent kind

$$
\begin{array}{l}
\textbf{letregion } r_1 \textbf{ where } w_1 = \mathit{MkMutable}\ r_1 \textbf{ in} \\
\textbf{letregion } r_2 \textbf{ in} \\
\textbf{do } \ x = 5\ r_1 \\
\qquad \mathit{updateInt}\ r_1\ r_2\ w_1\ x\ (23\ r_2) \\
\qquad \mathit{printInt}\ r_1\ x
\end{array}
$$

$$
\begin{array}{ll}
\mathit{printInt} & :: \forall (r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} () \\
\mathit{updateInt} & :: \forall (r_1, r_2 : \textbf{region}).\ \mathit{Mutable}\ r_1 \Rightarrow \mathit{Int}\ r_1 \rightarrow \mathit{Int}\ r_2 \xrightarrow{\mathit{Read}\ r_2 \vee \mathit{Write}\ r_1} () \\
\mathit{MkMutable} & :: \Pi (r_1 : \textbf{region}).\ \mathit{Mutable}\ r_1
\end{array}
$$

Evaluation

$$\begin{array}{l}
\textbf{letregion } r_1 \textbf{ where } w_1 = \mathit{MkMutable}\ r_1 \textbf{ in} \qquad \underline{\text{HEAP}} \\
\textbf{letregion } r_2 \textbf{ in} \\
\textbf{do } \ x = 5\ r_1 \\
\qquad \mathit{updateInt}\ r_1\ r_2\ w_1\ x\ (23\ r_2) \\
\qquad \mathit{printInt}\ r_1\ x
\end{array}$$

$$\begin{array}{ll}
\mathit{printInt} & :: \forall(r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} () \\
\mathit{updateInt} & :: \forall(r_1, r_2 : \textbf{region}).\ \mathit{Mutable}\ r_1 \Rightarrow \mathit{Int}\ r_1 \rightarrow \mathit{Int}\ r_2 \xrightarrow{\mathit{Read}\ r_2 \vee \mathit{Write}\ r_1} () \\
\mathit{MkMutable} & :: \Pi(r_1 : \textbf{region}).\ \mathit{Mutable}\ r_1
\end{array}$$

Evaluation (allocate region $r_1 \sim \rho_1$)

HEAP

$\rho_1 : \emptyset$

mutable ρ_1

$\longrightarrow$ **letregion** r_2 **in**
do $x = 5\ \rho_1$
updateInt $\rho_1\ r_2$ *mutable* $\rho_1\ x\ (23\ r_2)$
printInt $\rho_1\ x$

$$\begin{array}{ll} \mathit{printInt} & :: \forall(r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} () \\ \mathit{updateInt} & :: \forall(r_1, r_2 : \textbf{region}).\ \mathit{Mutable}\ r_1 \Rightarrow \mathit{Int}\ r_1 \rightarrow \mathit{Int}\ r_2 \xrightarrow{\mathit{Read}\ r_2 \vee \mathit{Write}\ r_1} () \\ \mathit{MkMutable} & :: \Pi(r_1 : \textbf{region}).\ \mathit{Mutable}\ r_1 \end{array}$$

Evaluation (allocate region $r_2 \sim \rho_2$)

HEAP

$\rho_1 : \emptyset$

mutable ρ_1

$\rho_2 : \emptyset$

$\longrightarrow$ **do** $x = 5\ \underline{\rho_1}$

updateInt $\underline{\rho_1}\ \underline{\rho_2}\ \underline{\textit{mutable}\ \rho_1}\ x\ (23\ \underline{\rho_2})$

printInt $\underline{\rho_1}\ x$

$$\begin{array}{ll}
\textit{printInt} & :: \forall(r_1 : \textbf{region}).\ \textit{Int}\ r_1 \xrightarrow{\textit{Read}\ r_1 \vee \textit{Console}} () \\
\textit{updateInt} & :: \forall(r_1, r_2 : \textbf{region}).\ \textit{Mutable}\ r_1 \Rightarrow \textit{Int}\ r_1 \rightarrow \textit{Int}\ r_2\ \xrightarrow{\textit{Read}\ r_2 \vee \textit{Write}\ r_1} () \\
\textit{MkMutable} & :: \Pi(r_1 : \textbf{region}).\ \textit{Mutable}\ r_1
\end{array}$$

Evaluation (allocate object at location l_1)

$\longrightarrow$ **do** $x = l_1$
updateInt $\rho_1\ \rho_2$ *mutable* $\rho_1\ x\ (23\ \rho_2)$
printInt $\rho_1\ x$

HEAP
$\rho_1 : \{\ l_1 \mapsto 5\ \}$
mutable ρ_1
$\rho_2 : \emptyset$

printInt $:: \forall(r_1 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1 \vee Console} ()$
updateInt $:: \forall(r_1, r_2 : \textbf{region}).\ Mutable\ r_1 \Rightarrow Int\ r_1 \rightarrow Int\ r_2\ \xrightarrow{Read\ r_2 \vee Write\ r_1} ()$
MkMutable $:: \Pi(r_1 : \textbf{region}).\ Mutable\ r_1$

Evaluation (substitute for x)

$\longrightarrow$ **do** *updateInt* $\rho_1\ \rho_2$ *mutable* $\rho_1\ l_1\ (23\ \rho_2)$
printInt $\rho_1\ l_1$

HEAP
$\rho_1 : \{\ l_1 \mapsto 5\ \}$
mutable ρ_1
$\rho_2 : \emptyset$

printInt $:: \forall(r_1 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1 \vee Console} ()$
updateInt $:: \forall(r_1, r_2 : \textbf{region}).\ Mutable\ r_1 \Rightarrow Int\ r_1 \rightarrow Int\ r_2\ ^{Read\ r_2 \vee Write\ r_1}\!\!\longrightarrow ()$
MkMutable $:: \Pi(r_1 : \textbf{region}).\ Mutable\ r_1$

Evaluation (allocate object at location l_2)

HEAP

$\rho_1 : \{\ l_1 \mapsto 5\ \}$

$mutable\ \rho_1$

$\rho_2 : \{\ l_2 \mapsto 23\ \}$

$\longrightarrow$ **do** *updateInt* $\rho_1\ \rho_2\ mutable\ \rho_1\ l_1\ l_2$
printInt $\rho_1\ l_1$

$$printInt :: \forall(r_1 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1 \vee Console} ()$$

$$updateInt :: \forall(r_1, r_2 : \textbf{region}).\ Mutable\ r_1 \Rightarrow Int\ r_1 \rightarrow Int\ r_2 \xrightarrow{Read\ r_2 \vee Write\ r_1} ()$$

$$MkMutable :: \Pi(r_1 : \textbf{region}).\ Mutable\ r_1$$

Evaluation (update object at l_1)

HEAP

$\rho_1 : \{\ l_1 \mapsto 23\ \}$

mutable ρ_1

$\rho_2 : \{\ l_2 \mapsto 23\ \}$

$\longrightarrow$ *printInt* $\underline{\rho_1}\ \underline{l_1}$

$$
\begin{array}{ll}
\mathit{printInt} & :: \forall(r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} () \\
\mathit{updateInt} & :: \forall(r_1, r_2 : \textbf{region}).\ \mathit{Mutable}\ r_1 \Rightarrow \mathit{Int}\ r_1 \rightarrow \mathit{Int}\ r_2 \xrightarrow{\mathit{Read}\ r_2 \vee \mathit{Write}\ r_1} () \\
\mathit{MkMutable} & :: \Pi(r_1 : \textbf{region}).\ \mathit{Mutable}\ r_1
\end{array}
$$

Evaluation (print result)

HEAP

$\rho_1 : \{\ l_1 \mapsto 23\ \}$

$mutable\ \rho_1$

$\rho_2 : \{\ l_2 \mapsto 23\ \}$

$\longrightarrow \qquad ()$

"23"

$printInt \quad :: \forall(r_1 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1 \vee Console} ()$

$updateInt \quad :: \forall(r_1, r_2 : \textbf{region}).\ Mutable\ r_1 \Rightarrow Int\ r_1 \rightarrow Int\ r_2 \xrightarrow{Read\ r_2 \vee Write\ r_1} ()$

$MkMutable \quad :: \Pi(r_1 : \textbf{region}).\ Mutable\ r_1$

Introducing Laziness

$$
\begin{array}{l}
\textbf{letregion}\ r_1\ \textbf{in} \\
\textbf{letregion}\ r_2\ \textbf{in} \\
\textbf{do}\ \ x = 5\ r_1 \\
\qquad y = \mathit{suspend}\ (\mathit{Int}\ r_1)\ (\mathit{Int}\ r_2)\ (\mathit{Read}\ r_1) \\
\qquad\qquad\qquad \ldots\ (\mathit{succ}\ r_1\ r_2)\ x \\
\qquad \ldots \\
\qquad \ldots \\
\qquad \mathit{printInt}\ r_1\ y
\end{array}
$$

$$
\begin{array}{ll}
\mathit{printInt} & :: \forall (r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} () \\
\mathit{succ} & :: \forall (r_1, r_2 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1} \mathit{Int}\ r_2 \\
\mathit{suspend} & :: \forall (a,\ b : \textbf{type})\ (e_1 : \textbf{effect}).\ \mathit{Pure}\ e_1 \Rightarrow (a \xrightarrow{e_1} b) \rightarrow a \rightarrow b
\end{array}
$$

Don't update objects read by suspended applications

$$
\begin{array}{l}
\textbf{letregion}\ r_1\ \textbf{in} \\
\textbf{letregion}\ r_2\ \textbf{in} \\
\textbf{do}\ \ x = 5\ r_1 \\
\qquad y = \mathit{suspend}\ (\mathit{Int}\ r_1)\ (\mathit{Int}\ r_2)\ (\mathit{Read}\ r_1) \\
\qquad\qquad\qquad \ldots\ (\mathit{succ}\ r_1\ r_2)\ x \\
\qquad \mathit{updateInt}\ r_1\ r_2\ \ldots\ x\ (42\ r_2) \qquad\qquad \text{NO!} \\
\qquad \ldots \\
\qquad \mathit{printInt}\ r_1\ y
\end{array}
$$

$$
\begin{array}{ll}
\mathit{printInt} & :: \forall (r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} () \\
\mathit{succ} & :: \forall (r_1, r_2 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1} \mathit{Int}\ r_2 \\
\mathit{suspend} & :: \forall (a,\ b : \textbf{type})\ (e_1 : \textbf{effect}).\ \mathit{Pure}\ e_1 \Rightarrow (a \xrightarrow{e_1} b) \rightarrow a \rightarrow b
\end{array}
$$

Introducing Laziness

$$
\begin{array}{l}
\textbf{letregion}\ r_1\ \textbf{in} \\
\textbf{letregion}\ r_2\ \textbf{in} \\
\textbf{do}\ \ x = 5\ r_1 \\
\qquad y = \mathit{suspend}\ (\mathit{Int}\ r_1)\ (\mathit{Int}\ r_2)\ (\mathit{Read}\ r_1) \\
\qquad\qquad\qquad \ldots\ (\mathit{succ}\ r_1\ r_2)\ x \\
\qquad \ldots \\
\qquad \ldots \\
\qquad \mathit{printInt}\ r_1\ y
\end{array}
$$

$$
\begin{array}{ll}
\mathit{printInt} & :: \forall (r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} () \\
\mathit{succ} & :: \forall (r_1, r_2 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1} \mathit{Int}\ r_2 \\
\mathit{suspend} & :: \forall (a,\ b : \textbf{type})\ (e_1 : \textbf{effect}).\ \mathit{Pure}\ e_1 \Rightarrow (a \xrightarrow{e_1} b) \rightarrow a \rightarrow b
\end{array}
$$

Objects read by suspended applications must be constant

$$
\begin{array}{l}
\textbf{letregion } r_1 \textbf{ where } w_1 = \mathit{Const}\ r_1 \textbf{ in} \\
\textbf{letregion } r_2 \textbf{ in} \\
\textbf{do } \ x = 5\ r_1 \\
\qquad y = \mathit{suspend}\ (\mathit{Int}\ r_1)\ (\mathit{Int}\ r_2)\ (\mathit{Read}\ r_1) \\
\qquad\qquad\qquad \ldots\ (\mathit{succ}\ r_1\ r_2)\ x \\
\qquad \ldots \\
\qquad \ldots \\
\qquad \mathit{printInt}\ r_1\ y
\end{array}
$$

$$
\begin{array}{ll}
\mathit{printInt} & :: \forall(r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} () \\
\mathit{succ} & :: \forall(r_1, r_2 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1} \mathit{Int}\ r_2 \\
\mathit{suspend} & :: \forall(a,\ b : \textbf{type})\ (e_1 : \textbf{effect}).\ \mathit{Pure}\ e_1 \Rightarrow (a \xrightarrow{e_1} b) \to a \to b \\
\mathit{MkConst} & :: \Pi(r_1 : \textbf{region}).\ \mathit{Const}\ r_1
\end{array}
$$

Reads from constant regions are always pure

$$
\begin{array}{l}
\textbf{letregion}\ r_1\ \textbf{where}\ w_1 = \mathit{Const}\ r_1\ \textbf{in} \\
\textbf{letregion}\ r_2\ \textbf{in} \\
\textbf{do}\ \ x = 5\ r_1 \\
\qquad y = \mathit{suspend}\ (\mathit{Int}\ r_1)\ (\mathit{Int}\ r_2)\ (\mathit{Read}\ r_1) \\
\qquad\qquad\qquad (\mathit{MkPurify}\ r_1\ w_1)\ (\mathit{succ}\ r_1\ r_2)\ x \\
\qquad \ldots \\
\qquad \ldots \\
\qquad \mathit{printInt}\ r_1\ y
\end{array}
$$

$$
\begin{array}{ll}
\mathit{printInt} & :: \forall(r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} () \\
\mathit{succ} & :: \forall(r_1, r_2 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1} \mathit{Int}\ r_2 \\
\mathit{suspend} & :: \forall(a,\ b : \textbf{type})\ (e_1 : \textbf{effect}).\ \mathit{Pure}\ e_1 \Rightarrow (a \xrightarrow{e_1} b) \to a \to b \\
\mathit{MkConst} & :: \Pi(r_1 : \textbf{region}).\ \mathit{Const}\ r_1 \\
\mathit{MkPurify} & :: \Pi(r_1 : \textbf{region}).\ \mathit{Const}\ r_1 \to \mathit{Pure}\ (\mathit{Read}\ r_1)
\end{array}
$$

Evaluation

$$
\begin{array}{l}
\textbf{letregion } r_1 \textbf{ where } w_1 = \mathit{Const}\ r_1 \textbf{ in} \\
\textbf{letregion } r_2 \textbf{ in} \\
\textbf{do } \ x = 5\ r_1 \\
\qquad y = \mathit{suspend}\ (\mathit{Int}\ r_1)\ (\mathit{Int}\ r_2)\ (\mathit{Read}\ r_1) \\
\qquad\qquad\qquad (\mathit{MkPurify}\ r_1\ w_1)\ (\mathit{succ}\ r_1\ r_2)\ x \\
\qquad \ldots \\
\qquad \ldots \\
\qquad \mathit{printInt}\ r_1\ y
\end{array}
$$

HEAP

$$
\begin{array}{ll}
\mathit{printInt} & :: \forall(r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} () \\
\mathit{succ} & :: \forall(r_1, r_2 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1} \mathit{Int}\ r_2 \\
\mathit{suspend} & :: \forall(a,\ b : \textbf{type})\ (e_1 : \textbf{effect}).\ \mathit{Pure}\ e_1 \Rightarrow (a \xrightarrow{e_1} b) \to a \to b \\
\mathit{MkConst} & :: \Pi(r_1 : \textbf{region}).\ \mathit{Const}\ r_1 \\
\mathit{MkPurify} & :: \Pi(r_1 : \textbf{region}).\ \mathit{Const}\ r_1 \to \mathit{Pure}\ (\mathit{Read}\ r_1)
\end{array}
$$

Evaluation (allocate $r_1 \sim \rho_1$)

HEAP

$\rho_1 : \emptyset$

$const\ \rho_1$

$\longrightarrow$ **letregion** r_2 **in**

do $x = 5\ \underline{\rho_1}$

$y = suspend\ (Int\ \underline{\rho_1})\ (Int\ r_2)\ (Read\ \underline{\rho_1})$

$(MkPurify\ \underline{\rho_1}\ \underline{const\ \rho_1})\ (succ\ \underline{\rho_1}\ r_2)\ x$

...

...

$printInt\ \underline{\rho_1}\ y$

$printInt :: \forall(r_1 : \mathbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1 \vee Console} ()$

$succ :: \forall(r_1, r_2 : \mathbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1} Int\ r_2$

$suspend :: \forall(a, b : \mathbf{type})\ (e_1 : \mathbf{effect}).\ Pure\ e_1 \Rightarrow (a \xrightarrow{e_1} b) \rightarrow a \rightarrow b$

$MkConst :: \Pi(r_1 : \mathbf{region}).\ Const\ r_1$

$MkPurify :: \Pi(r_1 : \mathbf{region}).\ Const\ r_1 \rightarrow Pure\ (Read\ r_1)$

Evaluation (allocate $r_2 \sim \rho_2$)

HEAP

$\rho_1 : \emptyset$

$const\ \rho_1$

$\rho_2 : \emptyset$

$\longrightarrow$ **do** $x = 5\ \underline{\rho_1}$

$y = suspend\ (Int\ \underline{\rho_1})\ (Int\ \underline{\rho_2})\ (Read\ \underline{\rho_1})$

$(MkPurify\ \underline{\rho_1}\ \underline{const\ \rho_1})\ (succ\ \underline{\rho_1}\ \underline{\rho_2})\ x$

...

...

$printInt\ \underline{\rho_1}\ y$

$printInt :: \forall(r_1 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1 \vee Console} ()$

$succ :: \forall(r_1, r_2 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1} Int\ r_2$

$suspend :: \forall(a, b : \textbf{type})\ (e_1 : \textbf{effect}).\ Pure\ e_1 \Rightarrow (a \xrightarrow{e_1} b) \rightarrow a \rightarrow b$

$MkConst :: \Pi(r_1 : \textbf{region}).\ Const\ r_1$

$MkPurify :: \Pi(r_1 : \textbf{region}).\ Const\ r_1 \rightarrow Pure\ (Read\ r_1)$

Evaluation (allocate object at location l_2)

HEAP

$\rho_1 : \{\ l_1 \mapsto 5\ \}$

const ρ_1

$\rho_2 : \emptyset$

$\longrightarrow$ **do** $x = l_1$

$y = suspend\ (Int\ \rho_1)\ (Int\ \rho_2)\ (Read\ \rho_1)$
$(MkPurify\ \rho_1\ const\ \rho_1)\ (succ\ \rho_1\ \rho_2)\ x$

...

...

$printInt\ \rho_1\ y$

$printInt \quad :: \forall(r_1 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1 \vee Console} ()$

$succ \quad :: \forall(r_1, r_2 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1} Int\ r_2$

$suspend \quad :: \forall(a,\ b : \textbf{type})\ (e_1 : \textbf{effect}).\ Pure\ e_1 \Rightarrow (a \xrightarrow{e_1} b) \rightarrow a \rightarrow b$

$MkConst \quad :: \Pi(r_1 : \textbf{region}).\ Const\ r_1$

$MkPurify \quad :: \Pi(r_1 : \textbf{region}).\ Const\ r_1 \rightarrow Pure\ (Read\ r_1)$

Evaluation (substitute for x)

HEAP

$\rho_1 : \{\ l_1 \mapsto 5\ \}$

$const\ \rho_1$

$\rho_2 : \emptyset$

$\longrightarrow$ **do** $y = suspend\ (Int\ \rho_1)\ (Int\ \rho_2)\ (Read\ \rho_1)$
$(MkPurify\ \rho_1\ const\ \rho_1)\ (succ\ \rho_1\ \rho_2)\ l_1$

...

...

$printInt\ \rho_1\ y$

$printInt \quad :: \forall(r_1 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1 \vee Console} ()$

$succ \quad :: \forall(r_1, r_2 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1} Int\ r_2$

$suspend \quad :: \forall(a,\ b : \textbf{type})\ (e_1 : \textbf{effect}).\ Pure\ e_1 \Rightarrow (a \xrightarrow{e_1} b) \rightarrow a \rightarrow b$

$MkConst \quad :: \Pi(r_1 : \textbf{region}).\ Const\ r_1$

$MkPurify \quad :: \Pi(r_1 : \textbf{region}).\ Const\ r_1 \rightarrow Pure\ (Read\ r_1)$

Evaluation (substitute for y)

HEAP

$\rho_1 : \{\ l_1 \mapsto 5\ \}$

$const\ \rho_1$

$\rho_2 : \emptyset$

$\longrightarrow \quad printInt\ \rho_1\ (suspend\ \ldots\ (MkPurify\ \rho_1\ const\ \rho_1)\ (succ\ \rho_1\ \rho_2)\ l_1)$

$printInt \quad :: \forall(r_1 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1 \vee Console} ()$

$succ \quad :: \forall(r_1, r_2 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1} Int\ r_2$

$suspend \quad :: \forall(a,\ b : \textbf{type})\ (e_1 : \textbf{effect}).\ Pure\ e_1 \Rightarrow (a \xrightarrow{e_1} b) \rightarrow a \rightarrow b$

$MkConst \quad :: \Pi(r_1 : \textbf{region}).\ Const\ r_1$

$MkPurify \quad :: \Pi(r_1 : \textbf{region}).\ Const\ r_1 \rightarrow Pure\ (Read\ r_1)$

Evaluation (make witness of purity)

HEAP

$\rho_1 : \{\ l_1 \mapsto 5\ \}$

$const\ \rho_1$

$\rho_2 : \emptyset$

$\longrightarrow \quad printInt\ \underline{\rho_1}\ (suspend\ \ldots\ (\underline{pure\ (Read\ \rho_1)})\ (succ\ \underline{\rho_1}\ \underline{\rho_2})\ \underline{l_1})$

$$
\begin{array}{ll}
printInt & :: \forall(r_1 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1 \vee Console} () \\
succ & :: \forall(r_1, r_2 : \textbf{region}).\ Int\ r_1 \xrightarrow{Read\ r_1} Int\ r_2 \\
suspend & :: \forall(a,\ b : \textbf{type})\ (e_1 : \textbf{effect}).\ Pure\ e_1 \Rightarrow (a \xrightarrow{e_1} b) \rightarrow a \rightarrow b \\
MkConst & :: \Pi(r_1 : \textbf{region}).\ Const\ r_1 \\
MkPurify & :: \Pi(r_1 : \textbf{region}).\ Const\ r_1 \rightarrow Pure\ (Read\ r_1)
\end{array}
$$

Evaluation (force suspension of *succ*)

HEAP

$\rho_1 : \{\ l_1 \mapsto 5\ \}$

const ρ_1

$\rho_2 : \{\ l_2 \mapsto 6\ \}$

$\longrightarrow$ *printInt* ρ_1 l_2

printInt	$:: \forall(r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} ()$
succ	$:: \forall(r_1, r_2 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1} \mathit{Int}\ r_2$
suspend	$:: \forall(a,\ b : \textbf{type})\ (e_1 : \textbf{effect}).\ \mathit{Pure}\ e_1 \Rightarrow (a \xrightarrow{e_1} b) \rightarrow a \rightarrow b$
MkConst	$:: \Pi(r_1 : \textbf{region}).\ \mathit{Const}\ r_1$
MkPurify	$:: \Pi(r_1 : \textbf{region}).\ \mathit{Const}\ r_1 \rightarrow \mathit{Pure}\ (\mathit{Read}\ r_1)$

Evaluation (print result)

HEAP

$\rho_1 : \{\ l_1 \mapsto 5\ \}$

$\mathit{const}\ \rho_1$

$\rho_2 : \{\ l_2 \mapsto 6\ \}$

$\longrightarrow \quad () \quad$ “6”

$$
\begin{array}{ll}
\mathit{printInt} & :: \forall(r_1 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1 \vee \mathit{Console}} () \\
\mathit{succ} & :: \forall(r_1, r_2 : \textbf{region}).\ \mathit{Int}\ r_1 \xrightarrow{\mathit{Read}\ r_1} \mathit{Int}\ r_2 \\
\mathit{suspend} & :: \forall(a,\ b : \textbf{type})\ (e_1 : \textbf{effect}).\ \mathit{Pure}\ e_1 \Rightarrow (a \xrightarrow{e_1} b) \to a \to b \\
\mathit{MkConst} & :: \Pi(r_1 : \textbf{region}).\ \mathit{Const}\ r_1 \\
\mathit{MkPurify} & :: \Pi(r_1 : \textbf{region}).\ \mathit{Const}\ r_1 \to \mathit{Pure}\ (\mathit{Read}\ r_1)
\end{array}
$$

Summary

- Use region constraints to support mutability polymorphism.
- Dependently kinded witnesses encode mutability/constancy of regions, and purity of effects.
- The type system ensures that suspended function applications don't have observable side effects.
- Lots of room for type directed compiler optimisations.
- Implementation at `http://www.haskell.org/haskellwiki/DDC`

Questions?

Reads from constant regions are always pure

$$\frac{\Gamma \mid \Sigma \vdash_{\kappa} w :: \mathit{Pure}\ e_1}{\Gamma \mid \Sigma \vdash e_1 \sqsubseteq \bot}$$