Effectful composition in natural language semantics

From Applicatives to Monads

Simon Charlow (Rutgers) Dylan Bumford (UCLA)

ESSLLI 2022, NUI Galway

Recap

Applicatives

F is applicative if it supports η and $\circledast$ with these types…

$$\eta : a \to F\,a \qquad\qquad \circledast : F\,(a \to b) \to F\,a \to F\,b$$

…Where η is a trivial way to inject something into the richer type characterized by F, and $\circledast$ is function application lifted into F.

McBride & Paterson 2008, Kiselyov 2015, Charlow 2019.

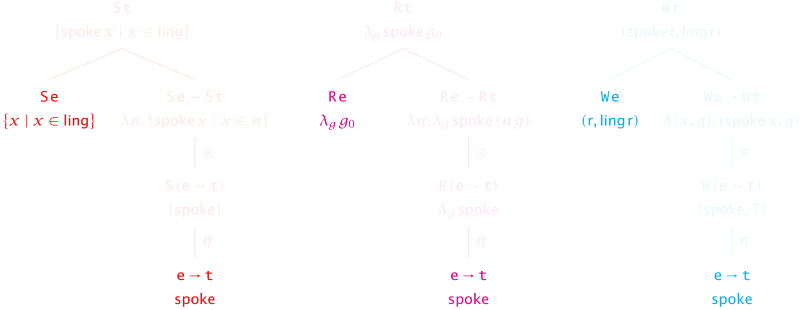
Three applicatives

$$\begin{aligned} \mathtt{S}\,a &::= \{a\} \\ \eta\, x &:= \{x\} \\ m \circledast n &:= \{f\,x \mid f \in m, x \in n\} \end{aligned}$$

$$\begin{aligned} \mathtt{R}\,a &::= \mathtt{g} \to a \\ \eta\, x &:= \lambda_g\, x \\ m \circledast n &:= \lambda_g\, m\, g\, (n\, g) \end{aligned}$$

$$\begin{aligned} \mathtt{W}\,a &::= a \times \mathtt{t} \\ \eta\, x &:= (a, \mathbb{T}) \\ (f, p) \circledast (x, q) &:= (f\,x, p \wedge q) \end{aligned}$$

Derivations: a linguist spoke/she$_0$ spoke/Roger, a linguist, spoke



St

$\{\mathsf{spoke}\,x \mid x \in \mathsf{ling}\}$

Se

$\{x \mid x \in \mathsf{ling}\}$

Se → St

$\lambda n.\{\mathsf{spoke}\,x \mid x \in n\}$

⊛

S(e → t)

$\{\mathsf{spoke}\}$

η

e → t

spoke

Rt

$\lambda_g\,\mathsf{spoke}\,g_0$

Re

$\lambda_g\,g_0$

Re → Rt

$\lambda n.\lambda_g\,\mathsf{spoke}\,(n\,g)$

⊛

R(e → t)

$\lambda_g\,\mathsf{spoke}$

η

e → t

spoke

Wt

$(\mathsf{spoke}\,\mathsf{r}, \mathsf{ling}\,\mathsf{r})$

We

$(\mathsf{r}, \mathsf{ling}\,\mathsf{r})$

We → Wt

$\lambda(x, q).(\mathsf{spoke}\,x, q)$

⊛

W(e → t)

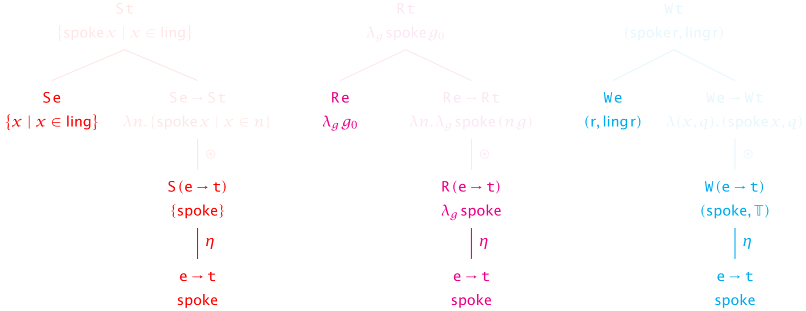
$(\mathsf{spoke}, \mathbb{T})$

η

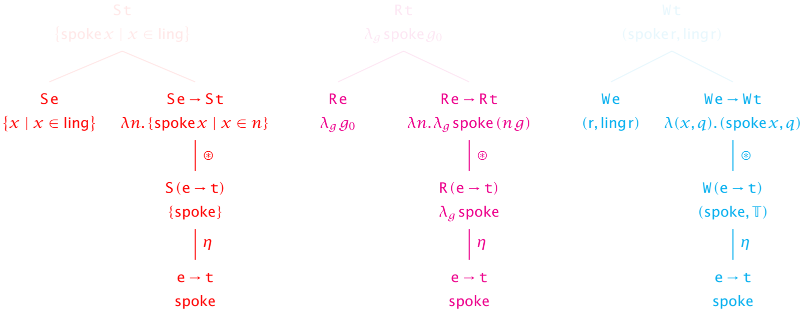
e → t

spoke

Derivations: a linguist spoke/she$_0$ spoke/Roger, a linguist, spoke



Derivations: a linguist spoke/she$_0$ spoke/Roger, a linguist, spoke



$\mathtt{S\,t}$

$\{\mathsf{spoke}\,x \mid x \in \mathsf{ling}\}$

$\mathtt{S\,e}$ — $\{x \mid x \in \mathsf{ling}\}$

$\mathtt{S\,e} \to \mathtt{S\,t}$ — $\lambda n.\{\mathsf{spoke}\,x \mid x \in n\}$

$\circledast$

$\mathtt{S\,(e \to t)}$ — $\{\mathsf{spoke}\}$

η

$\mathtt{e \to t}$ — spoke

$\mathtt{R\,t}$

$\lambda_g\, \mathsf{spoke}\, g_0$

$\mathtt{R\,e}$ — $\lambda_g\, g_0$

$\mathtt{R\,e} \to \mathtt{R\,t}$ — $\lambda n.\lambda_g\, \mathsf{spoke}\,(n\, g)$

$\circledast$

$\mathtt{R\,(e \to t)}$ — $\lambda_g\, \mathsf{spoke}$

η

$\mathtt{e \to t}$ — spoke

$\mathtt{W\,t}$

$(\mathsf{spoke}\,\mathsf{r}, \mathsf{ling}\,\mathsf{r})$

$\mathtt{W\,e}$ — $(\mathsf{r}, \mathsf{ling}\,\mathsf{r})$

$\mathtt{W\,e} \to \mathtt{W\,t}$ — $\lambda(x, q).(\mathsf{spoke}\,x, q)$

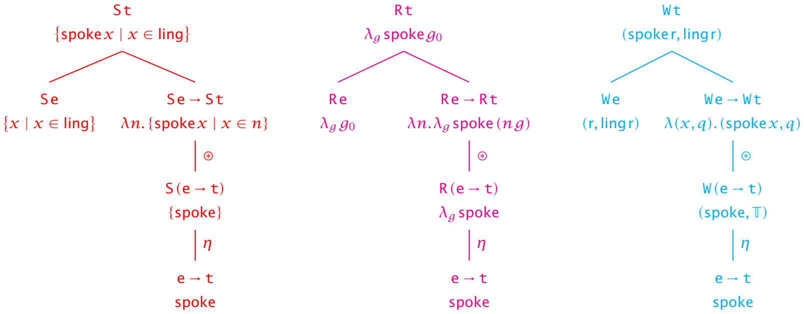
$\circledast$

$\mathtt{W\,(e \to t)}$ — $(\mathsf{spoke}, \mathbb{T})$

η

$\mathtt{e \to t}$ — spoke

Derivations: a linguist spoke/she$_0$ spoke/Roger, a linguist, spoke



$\mathtt{S\,t}$

$\{\mathsf{spoke}\,x \mid x \in \mathsf{ling}\}$

$\mathtt{S\,e}$

$\{x \mid x \in \mathsf{ling}\}$

$\mathtt{S\,e} \to \mathtt{S\,t}$

$\lambda n.\{\mathsf{spoke}\,x \mid x \in n\}$

$\circledast$

$\mathtt{S}\,(\mathtt{e} \to \mathtt{t})$

$\{\mathsf{spoke}\}$

η

$\mathtt{e} \to \mathtt{t}$

spoke

$\mathtt{R\,t}$

$\lambda_g\,\mathsf{spoke}\,g_0$

$\mathtt{R\,e}$

$\lambda_g\,g_0$

$\mathtt{R\,e} \to \mathtt{R\,t}$

$\lambda n.\lambda_g\,\mathsf{spoke}\,(n\,g)$

$\circledast$

$\mathtt{R}\,(\mathtt{e} \to \mathtt{t})$

$\lambda_g\,\mathsf{spoke}$

η

$\mathtt{e} \to \mathtt{t}$

spoke

$\mathtt{W\,t}$

$(\mathsf{spoke}\,\mathsf{r}, \mathsf{ling}\,\mathsf{r})$

$\mathtt{W\,e}$

$(\mathsf{r}, \mathsf{ling}\,\mathsf{r})$

$\mathtt{W\,e} \to \mathtt{W\,t}$

$\lambda(x, q).(\mathsf{spoke}\,x, q)$

$\circledast$

$\mathtt{W}\,(\mathtt{e} \to \mathtt{t})$

$(\mathsf{spoke}, \mathbb{T})$

η

$\mathtt{e} \to \mathtt{t}$

spoke

Monads

Indefinites and pronouns

Indefinite noun phrases can host pronouns:

1. Mary submitted a paper she wrote

Given what we have said so far, the type of a pronoun-harboring indefinite should include at least a Reading effect and a Set effect:

$$\mathsf{RS\,e} = \mathsf{r} \to \{\mathsf{e}\} \qquad \mathsf{SR\,e} = \{\mathsf{r} \to \mathsf{e}\}$$

With a little thought, you can convince yourself that only one of these makes any sense

$$[\![\text{a paper she}_0 \text{ wrote}]\!] =$$

Indefinites and pronouns

Indefinite noun phrases can host pronouns:

1. Mary submitted a paper she wrote

Given what we have said so far, the type of a pronoun-harboring indefinite should include at least a Reading effect and a Set effect:

$$\mathsf{RS}\,\mathsf{e} = \mathsf{r} \to \{\mathsf{e}\} \qquad \mathsf{SR}\,\mathsf{e} = \{\mathsf{r} \to \mathsf{e}\}$$

With a little thought, you can convince yourself that only one of these makes any sense

$$[\![\text{a paper she}_0 \text{ wrote}]\!] = \lambda_g \{x \mid \mathsf{paper}\, x, \mathsf{write}\, x\, g_0\}$$

Indefinites and binding

Indefinite noun phrases can also bind pronouns

2. A linguist submitted a paper she wrote.

Intuitively, (2) is ambiguous between these two meanings:

3. a. $\underbrace{\lambda_g \{\mathsf{submit}\, y\, x \mid \mathsf{ling}\, x, \mathsf{paper}\, y, \mathsf{wrote}\, y\, x\}}_{\mathtt{RS\,t}}$

   b. $\underbrace{\lambda_g \{\mathsf{submit}\, y\, x \mid \mathsf{ling}\, x, \mathsf{paper}\, y, \mathsf{wrote}\, y\, g_0\}}_{\mathtt{RS\,t}}$

How can these meanings be composed?

Modifying environments

Remember that we are treating pronouns as triggering a `Read` effect on the environment

So to accomplish the "bound" reading of (2), we need some mechanism to allow expressions to **modify** the environment that other expressions are evaluated in:

$$\triangleright_n := \lambda_m \lambda_x (\lambda_g\, m\, g^{n \to x}) \circledast \eta\, x$$

Note that this operation is Effect-polymorphic; it will work for any composition of Functors beginning with `R`

$$\triangleright_n :: \mathsf{R}\,(\mathsf{e} \to \sigma) \to \mathsf{e} \to \mathsf{R}\,\sigma$$

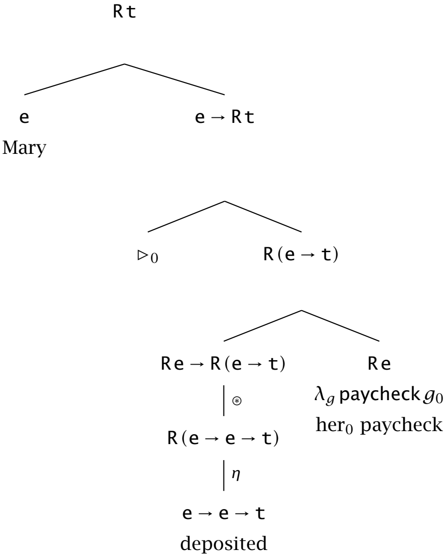
$$\triangleright_n :: \mathsf{RS}\,(\mathsf{e} \to \sigma) \to \mathsf{e} \to \mathsf{RS}\,\sigma$$

$$\triangleright_n :: \mathsf{RW}\,(\mathsf{e} \to \sigma) \to \mathsf{e} \to \mathsf{RW}\,\sigma$$

$$\triangleright_n :: \mathsf{RWS}\,(\mathsf{e} \to \sigma) \to \mathsf{e} \to \mathsf{RSW}\,\sigma$$

......

… binding …



$\mathtt{R\,t}$

$\mathtt{e}$

Mary

$\mathtt{e \to R\,t}$

$\rhd_0$

$\mathtt{R\,(e \to t)}$

$\mathtt{R\,e \to R\,(e \to t)}$

$\circledast$

$\mathtt{R\,(e \to e \to t)}$

η

$\mathtt{e \to e \to t}$

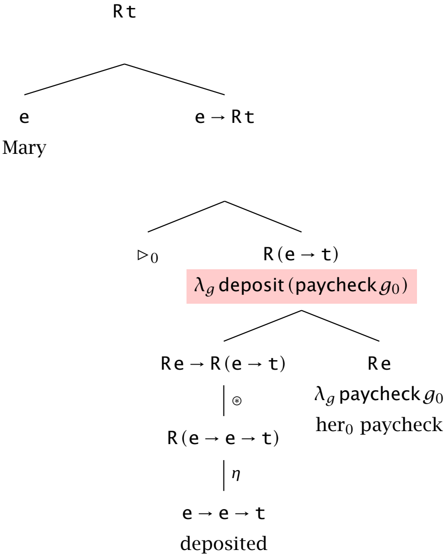
deposited

$\mathtt{R\,e}$

$\lambda_g\,\mathsf{paycheck}\,g_0$

her_0 paycheck

… binding …



$\mathtt{R\,t}$

$\mathtt{e}$

Mary

$\mathtt{e \to R\,t}$

$\triangleright_0$

$\mathtt{R\,(e \to t)}$

$\lambda_g\, \mathsf{deposit}\, (\mathsf{paycheck}\, g_0)$

$\mathtt{R\,e \to R\,(e \to t)}$

$\circledast$

$\mathtt{R\,(e \to e \to t)}$

η

$\mathtt{e \to e \to t}$

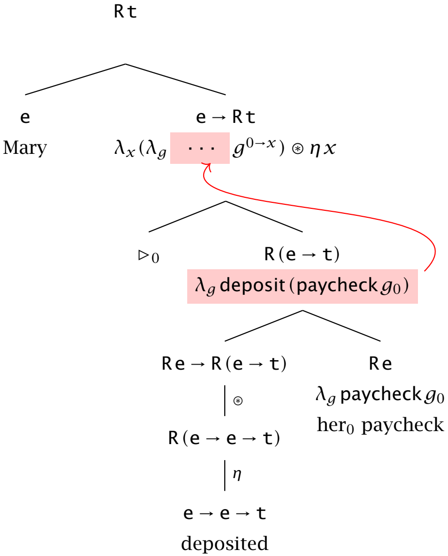
deposited

$\mathtt{R\,e}$

$\lambda_g\, \mathsf{paycheck}\, g_0$

her$_0$ paycheck

… binding …



$\mathtt{R\,t}$

$\mathtt{e}$

Mary

$\mathtt{e \to R\,t}$

$\lambda_x (\lambda_g \; \cdots \; g^{0 \to x}) \circledast \eta\, x$

$\triangleright_0$

$\mathtt{R\,(e \to t)}$

$\lambda_g\, \mathsf{deposit}\,(\mathsf{paycheck}\, g_0)$

$\mathtt{R\,e \to R\,(e \to t)}$

$\circledast$

$\mathtt{R\,(e \to e \to t)}$

η

$\mathtt{e \to e \to t}$

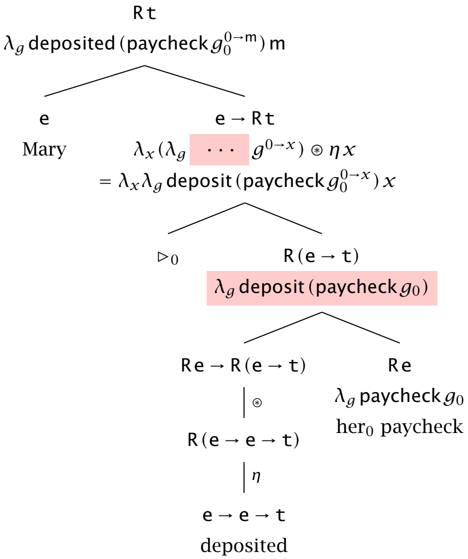
deposited

$\mathtt{R\,e}$

$\lambda_g\, \mathsf{paycheck}\, g_0$

her_0 paycheck

… binding …



$\mathtt{R\,t}$

$\lambda_g\, \mathsf{deposited}\,(\mathsf{paycheck}\, g_0^{0\to\mathsf{m}})\, \mathsf{m}$

$\mathtt{e}$

Mary

$\mathtt{e} \to \mathtt{R\,t}$

$\lambda_x(\lambda_g \; \boxed{\cdots} \; g^{0\to x}) \circledast \eta\, x$

$= \lambda_x \lambda_g\, \mathsf{deposit}\,(\mathsf{paycheck}\, g_0^{0\to x})\, x$

$\triangleright_0$

$\mathtt{R\,(e \to t)}$

$\boxed{\lambda_g\, \mathsf{deposit}\,(\mathsf{paycheck}\, g_0)}$

$\mathtt{R\,e \to R\,(e \to t)}$

$\circledast$

$\mathtt{R\,(e \to e \to t)}$

η

$\mathtt{e \to e \to t}$

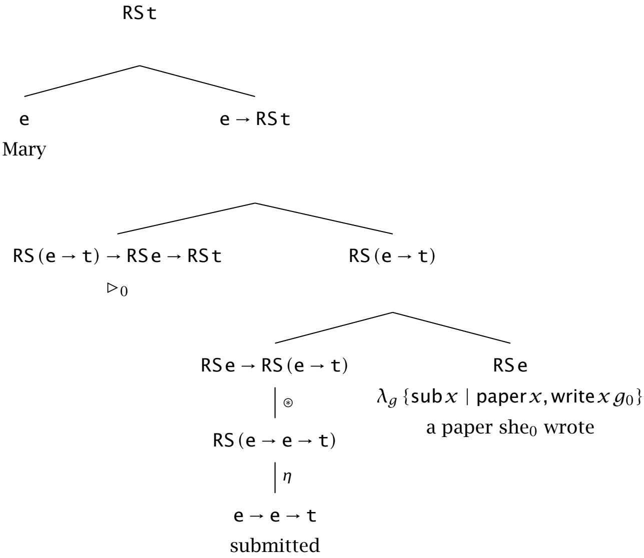
deposited

$\mathtt{R\,e}$

$\lambda_g\, \mathsf{paycheck}\, g_0$

her_0 paycheck

… binding into indefinites



$\mathsf{RS\,t}$

e

Mary

$\mathsf{e \to RS\,t}$

$\mathsf{RS\,(e \to t) \to RS\,e \to RS\,t}$

$\rhd_0$

$\mathsf{RS\,(e \to t)}$

$\mathsf{RS\,e \to RS\,(e \to t)}$

$\circledast$

$\mathsf{RS\,(e \to e \to t)}$

η

$\mathsf{e \to e \to t}$

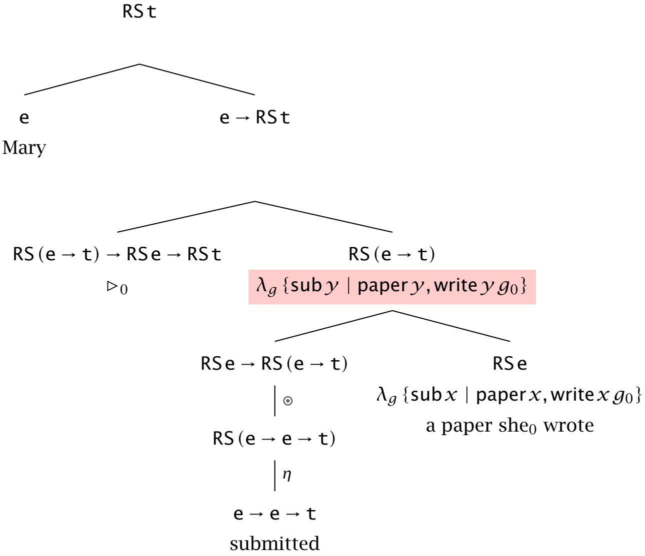
submitted

$\mathsf{RS\,e}$

$\lambda_g\,\{\mathsf{sub}\,x \mid \mathsf{paper}\,x, \mathsf{write}\,x\,g_0\}$

a paper she_0 wrote

… binding into indefinites



$\mathtt{RS\,t}$

$\mathtt{e}$

Mary

$\mathtt{e \to RS\,t}$

$\mathtt{RS\,(e \to t) \to RS\,e \to RS\,t}$

$\triangleright_0$

$\mathtt{RS\,(e \to t)}$

$\lambda_g\,\{\mathsf{sub}\,y \mid \mathsf{paper}\,y, \mathsf{write}\,y\,g_0\}$

$\mathtt{RS\,e \to RS\,(e \to t)}$

$\circledast$

$\mathtt{RS\,(e \to e \to t)}$

η

$\mathtt{e \to e \to t}$

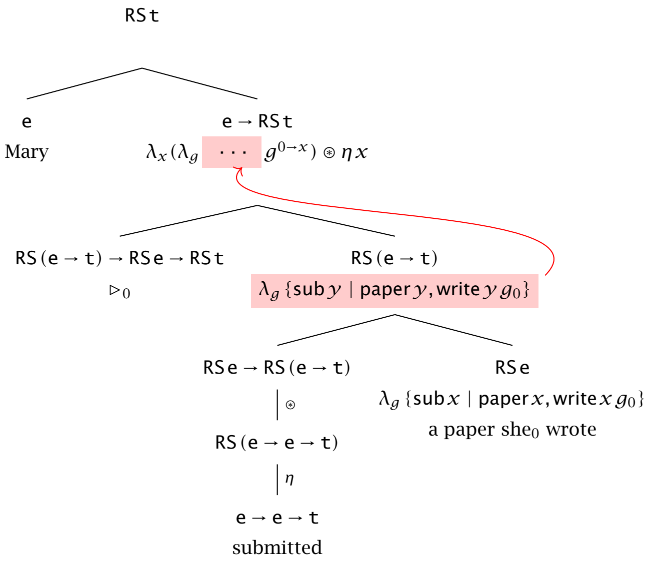
submitted

$\mathtt{RS\,e}$

$\lambda_g\,\{\mathsf{sub}\,x \mid \mathsf{paper}\,x, \mathsf{write}\,x\,g_0\}$

a paper she$_0$ wrote

… binding into indefinites



$\mathsf{RS\,t}$

e

Mary

$\mathsf{e \to RS\,t}$

$\lambda_x (\lambda_g \; \cdots \; g^{0\to x}) \circledast \eta\, x$

$\mathsf{RS\,(e \to t) \to RS\,e \to RS\,t}$

$\triangleright_0$

$\mathsf{RS\,(e \to t)}$

$\lambda_g \{\mathsf{sub}\, y \mid \mathsf{paper}\, y, \mathsf{write}\, y\, g_0\}$

$\mathsf{RS\,e \to RS\,(e \to t)}$

$\circledast$

$\mathsf{RS\,(e \to e \to t)}$

η

$\mathsf{e \to e \to t}$

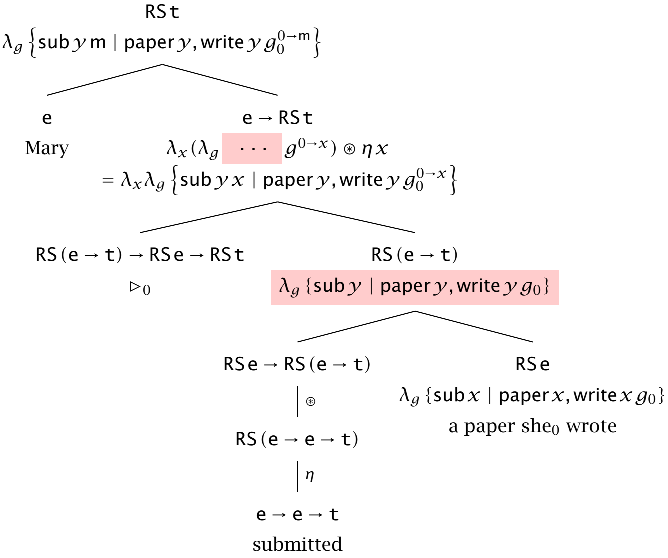
submitted

$\mathsf{RS\,e}$

$\lambda_g \{\mathsf{sub}\, x \mid \mathsf{paper}\, x, \mathsf{write}\, x\, g_0\}$

a paper she$_0$ wrote

… binding into indefinites



RS t

$\lambda_g \left\{\mathsf{sub}\,y\,\mathsf{m} \mid \mathsf{paper}\,y, \mathsf{write}\,y\,g_0^{0\to\mathsf{m}}\right\}$

e

Mary

e → RS t

$\lambda_x (\lambda_g \ \cdots \ g^{0\to x}) \circledast \eta\, x$

$= \lambda_x \lambda_g \left\{\mathsf{sub}\,y\,x \mid \mathsf{paper}\,y, \mathsf{write}\,y\,g_0^{0\to x}\right\}$

RS (e → t) → RS e → RS t

$\triangleright_0$

RS (e → t)

$\lambda_g \{\mathsf{sub}\,y \mid \mathsf{paper}\,y, \mathsf{write}\,y\,g_0\}$

RS e → RS (e → t)

$\circledast$

RS (e → e → t)

η

e → e → t

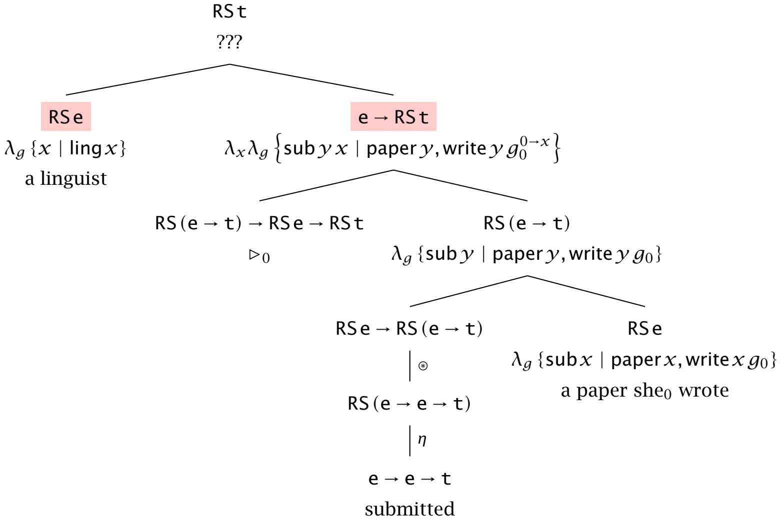
submitted

RS e

$\lambda_g \{\mathsf{sub}\,x \mid \mathsf{paper}\,x, \mathsf{write}\,x\,g_0\}$

a paper she$_0$ wrote

Indefinites binding into indefinites



$\mathtt{RS\,t}$

???

$\mathtt{RS\,e}$

$\lambda_g\,\{x \mid \mathsf{ling}\,x\}$

a linguist

$\mathtt{e \to RS\,t}$

$\lambda_x\lambda_g\left\{\mathsf{sub}\,y\,x \mid \mathsf{paper}\,y, \mathsf{write}\,y\,g_0^{0\to x}\right\}$

$\mathtt{RS\,(e \to t) \to RS\,e \to RS\,t}$

$\rhd_0$

$\mathtt{RS\,(e \to t)}$

$\lambda_g\,\{\mathsf{sub}\,y \mid \mathsf{paper}\,y, \mathsf{write}\,y\,g_0\}$

$\mathtt{RS\,e \to RS\,(e \to t)}$

$\circledast$

$\mathtt{RS\,(e \to e \to t)}$

η

$\mathtt{e \to e \to t}$

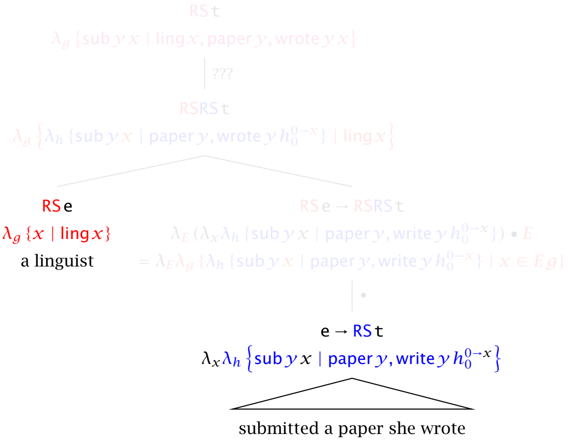
submitted

$\mathtt{RS\,e}$

$\lambda_g\,\{\mathsf{sub}\,x \mid \mathsf{paper}\,x, \mathsf{write}\,x\,g_0\}$

a paper she$_0$ wrote

Getting closer



$\mathsf{RS}\,\mathtt{t}$

$\lambda_g\,\{\mathsf{sub}\,y\,x \mid \mathsf{ling}\,x, \mathsf{paper}\,y, \mathsf{wrote}\,y\,x\}$

???

$\mathsf{RSRS}\,\mathtt{t}$

$\lambda_g\left\{\lambda_h\,\{\mathsf{sub}\,y\,x \mid \mathsf{paper}\,y, \mathsf{wrote}\,y\,h_0^{0\to x}\} \mid \mathsf{ling}\,x\right\}$

$\mathsf{RS}\,\mathtt{e}$

$\lambda_g\,\{x \mid \mathsf{ling}\,x\}$

a linguist

$\mathsf{RS}\,\mathtt{e} \to \mathsf{RSRS}\,\mathtt{t}$

$\lambda_E\,(\lambda_x\lambda_h\,\{\mathsf{sub}\,y\,x \mid \mathsf{paper}\,y, \mathsf{write}\,y\,h_0^{0\to x}\}) \bullet E$

$= \lambda_E\lambda_g\,\{\lambda_h\,\{\mathsf{sub}\,y\,x \mid \mathsf{paper}\,y, \mathsf{write}\,y\,h_0^{0\to x}\} \mid x \in E\,g\}$

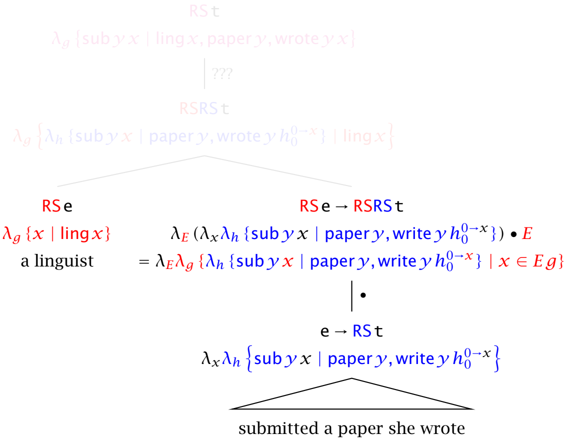
$\bullet$

$\mathtt{e} \to \mathsf{RS}\,\mathtt{t}$

$\lambda_x\lambda_h\left\{\mathsf{sub}\,y\,x \mid \mathsf{paper}\,y, \mathsf{write}\,y\,h_0^{0\to x}\right\}$

submitted a paper she wrote

Getting closer



$\mathsf{RS}\,\mathtt{t}$

$\lambda_g\,\{\mathsf{sub}\,y\,x \mid \mathsf{ling}\,x, \mathsf{paper}\,y, \mathsf{wrote}\,y\,x\}$

???

$\mathsf{RSRS}\,\mathtt{t}$

$\lambda_g\left\{\lambda_h\,\{\mathsf{sub}\,y\,x \mid \mathsf{paper}\,y, \mathsf{wrote}\,y\,h_0^{0\to x}\} \mid \mathsf{ling}\,x\right\}$

$\mathsf{RS}\,\mathtt{e}$

$\lambda_g\,\{x \mid \mathsf{ling}\,x\}$

a linguist

$\mathsf{RS}\,\mathtt{e} \to \mathsf{RSRS}\,\mathtt{t}$

$\lambda_E\,(\lambda_x\lambda_h\,\{\mathsf{sub}\,y\,x \mid \mathsf{paper}\,y, \mathsf{write}\,y\,h_0^{0\to x}\}) \bullet E$

$= \lambda_E\lambda_g\,\{\lambda_h\,\{\mathsf{sub}\,y\,x \mid \mathsf{paper}\,y, \mathsf{write}\,y\,h_0^{0\to x}\} \mid x \in E\,g\}$

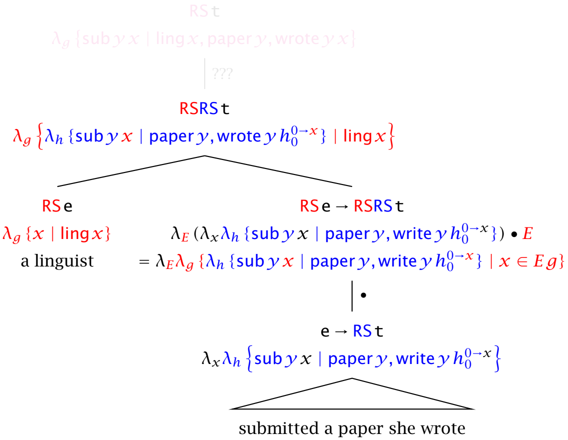
$\bullet$

$\mathtt{e} \to \mathsf{RS}\,\mathtt{t}$

$\lambda_x\lambda_h\left\{\mathsf{sub}\,y\,x \mid \mathsf{paper}\,y, \mathsf{write}\,y\,h_0^{0\to x}\right\}$

submitted a paper she wrote

Getting closer



$\mathsf{RS}\,\mathtt{t}$

$\lambda_g \{\mathsf{sub}\, y\, x \mid \mathsf{ling}\, x, \mathsf{paper}\, y, \mathsf{wrote}\, y\, x\}$

???

$\mathsf{RS}\mathsf{RS}\,\mathtt{t}$

$\lambda_g \left\{\lambda_h \{\mathsf{sub}\, y\, x \mid \mathsf{paper}\, y, \mathsf{wrote}\, y\, h_0^{0\to x}\} \mid \mathsf{ling}\, x\right\}$

$\mathsf{RS}\,\mathtt{e}$

$\lambda_g \{x \mid \mathsf{ling}\, x\}$

a linguist

$\mathsf{RS}\,\mathtt{e} \to \mathsf{RS}\mathsf{RS}\,\mathtt{t}$

$\lambda_E\, (\lambda_x \lambda_h \{\mathsf{sub}\, y\, x \mid \mathsf{paper}\, y, \mathsf{write}\, y\, h_0^{0\to x}\}) \bullet E$

$= \lambda_E \lambda_g \{\lambda_h \{\mathsf{sub}\, y\, x \mid \mathsf{paper}\, y, \mathsf{write}\, y\, h_0^{0\to x}\} \mid x \in E\, g\}$

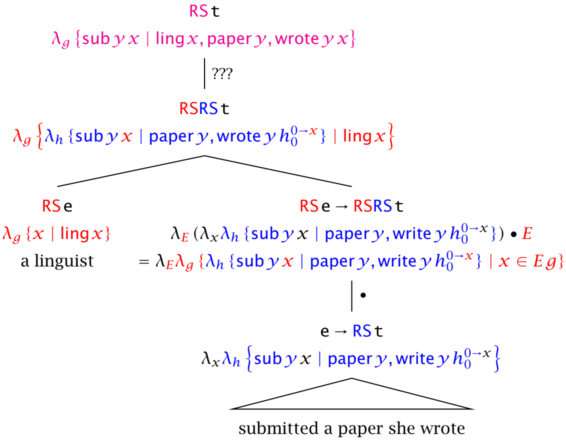
$\bullet$

$\mathtt{e} \to \mathsf{RS}\,\mathtt{t}$

$\lambda_x \lambda_h \left\{\mathsf{sub}\, y\, x \mid \mathsf{paper}\, y, \mathsf{write}\, y\, h_0^{0\to x}\right\}$

submitted a paper she wrote

Getting closer



$\mathsf{RS}\,\mathsf{t}$

$$\lambda_g \{\mathsf{sub}\, y\, x \mid \mathsf{ling}\, x, \mathsf{paper}\, y, \mathsf{wrote}\, y\, x\}$$

???

$\mathsf{RS}\mathsf{RS}\,\mathsf{t}$

$$\lambda_g \left\{\lambda_h \{\mathsf{sub}\, y\, x \mid \mathsf{paper}\, y, \mathsf{wrote}\, y\, h_0^{0\to x}\} \mid \mathsf{ling}\, x\right\}$$

$\mathsf{RS}\,\mathsf{e}$

$$\lambda_g \{x \mid \mathsf{ling}\, x\}$$

a linguist

$\mathsf{RS}\,\mathsf{e} \to \mathsf{RS}\mathsf{RS}\,\mathsf{t}$

$$\lambda_E \left(\lambda_x \lambda_h \{\mathsf{sub}\, y\, x \mid \mathsf{paper}\, y, \mathsf{write}\, y\, h_0^{0\to x}\}\right) \bullet E$$

$$= \lambda_E \lambda_g \{\lambda_h \{\mathsf{sub}\, y\, x \mid \mathsf{paper}\, y, \mathsf{write}\, y\, h_0^{0\to x}\} \mid x \in E\, g\}$$

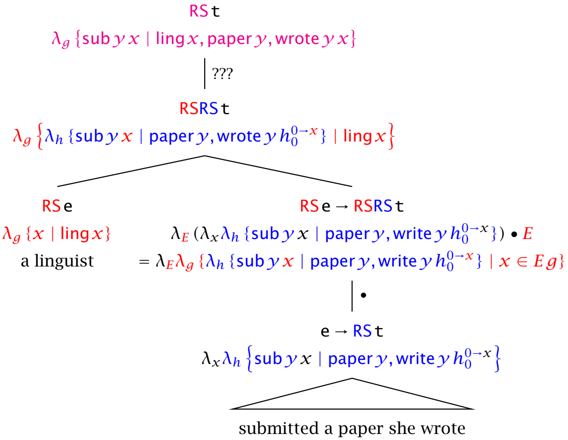
$\bullet$

$\mathsf{e} \to \mathsf{RS}\,\mathsf{t}$

$$\lambda_x \lambda_h \left\{\mathsf{sub}\, y\, x \mid \mathsf{paper}\, y, \mathsf{write}\, y\, h_0^{0\to x}\right\}$$

submitted a paper she wrote

Getting closer



$\mathsf{RS}\,\mathtt{t}$

$\lambda_g\,\{\mathsf{sub}\,y\,x \mid \mathsf{ling}\,x, \mathsf{paper}\,y, \mathsf{wrote}\,y\,x\}$

???

$\mathsf{RSRS}\,\mathtt{t}$

$\lambda_g\left\{\lambda_h\,\{\mathsf{sub}\,y\,x \mid \mathsf{paper}\,y, \mathsf{wrote}\,y\,h_0^{0\to x}\} \mid \mathsf{ling}\,x\right\}$

$\mathsf{RS}\,\mathtt{e}$

$\lambda_g\,\{x \mid \mathsf{ling}\,x\}$

a linguist

$\mathsf{RS}\,\mathtt{e} \to \mathsf{RSRS}\,\mathtt{t}$

$\lambda_E\,(\lambda_x\lambda_h\,\{\mathsf{sub}\,y\,x \mid \mathsf{paper}\,y, \mathsf{write}\,y\,h_0^{0\to x}\}) \bullet E$

$= \lambda_E\lambda_g\,\{\lambda_h\,\{\mathsf{sub}\,y\,x \mid \mathsf{paper}\,y, \mathsf{write}\,y\,h_0^{0\to x}\} \mid x \in E\,g\}$

$\bullet$

$\mathtt{e} \to \mathsf{RS}\,\mathtt{t}$

$\lambda_x\lambda_h\left\{\mathsf{sub}\,y\,x \mid \mathsf{paper}\,y, \mathsf{write}\,y\,h_0^{0\to x}\right\}$

submitted a paper she wrote

The meaning we can get has two layers of independent RS structure

But to get the meaning we want, we'll need a way to **flatten** them somehow

$$\mu :: \mathsf{RSRS}\,\mathtt{a} \to \mathsf{RS}\,\mathtt{a}$$

R flattener

Let's warm up by finding a function with the following type:

$$\mu :: \mathsf{R}\mathsf{R}a \to \mathsf{R}a$$

R flattener

Let's warm up by finding a function with the following type:

$$\mu :: \mathsf{R}\mathsf{R}a \to \mathsf{R}a$$

The obvious candidate duplicates an assignment:

$$\mu M := \lambda_g M g g$$

S flattener

Let's warm up by finding a function with the following type:

$$\mu :: \mathsf{S}\mathsf{S}a \to \mathsf{S}a$$

S flattener

Let's warm up by finding a function with the following type:

$$\mu :: \mathsf{SS}a \to \mathsf{S}a$$

The obvious candidate takes the grand union:

$$\begin{aligned} \mu M &:= \bigcup M \\ &= \{a \mid m \in M, a \in m\} \end{aligned}$$

RS flattener

So can we define a flattener function for RS?

$$\mu :: \mathsf{RSRS}\,a \to \mathsf{RS}\,a$$

RS flattener

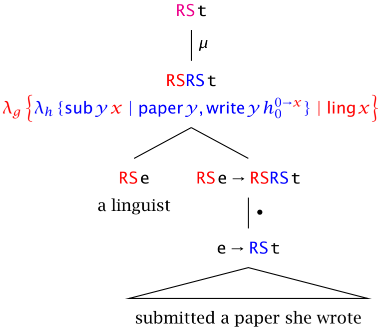
So can we define a flattener function for RS?

$$\mu :: \mathsf{RSRS}\,a \to \mathsf{RS}\,a$$

The obvious candidate mixes R's and S's μ operations:

$$\begin{aligned} \mu M &:= \lambda_g \bigcup \{m\,g \mid m \in M\,g\} \\ &= \lambda_g \{a \mid m \in M\,g, a \in m\,g\} \end{aligned}$$

Flattening in action



$\mathsf{RS}\,\mathsf{t}$

μ

$\mathsf{RS}\mathsf{RS}\,\mathsf{t}$

$\lambda_g \left\{\lambda_h \left\{\mathsf{sub}\,y\,x \mid \mathsf{paper}\,y, \mathsf{write}\,y\,h_0^{0\to x}\right\} \mid \mathsf{ling}\,x\right\}$

$\mathsf{RS}\,\mathsf{e}$

a linguist

$\mathsf{RS}\,\mathsf{e} \to \mathsf{RS}\mathsf{RS}\,\mathsf{t}$

$\bullet$

$\mathsf{e} \to \mathsf{RS}\,\mathsf{t}$

submitted a paper she wrote

$$\begin{aligned}
\mu\left(\lambda_g \left\{\lambda_h \left\{\ldots h_0^{0\to x} \ldots\right\} \mid \mathsf{ling}\,x\right\}\right) &= \lambda_g \left\{a \mid m \in \left(\lambda_g \left\{\lambda_h \left\{\ldots h_0^{0\to x} \ldots\right\} \mid \mathsf{ling}\,x\right\}\right) g,\ a \in m\,g\right\} \\
&= \lambda_g \left\{a \mid m \in \left\{\lambda_h \left\{\ldots h_0^{0\to x} \ldots\right\} \mid \mathsf{ling}\,x\right\},\ a \in m\,g\right\} \\
&= \lambda_g \left\{a \mid \mathsf{ling}\,x,\ a \in \left(\lambda_h \left\{\ldots h_0^{0\to x} \ldots\right\}\right) g\right\} \\
&= \lambda_g \left\{a \mid \mathsf{ling}\,x,\ a \in \left\{\ldots g_0^{0\to x} \ldots\right\}\right\} \\
&= \lambda_g \left\{\mathsf{submit}\,y\,x \mid \mathsf{ling}\,x, \mathsf{paper}\,y, \mathsf{wrote}\,y\,x\right\}
\end{aligned}$$

More on μ

$$\begin{aligned}\mu M &:= \lambda_g \bigcup_{m \in M g} m g \\ &= \lambda_g \{a \mid m \in M g,\ a \in m g\}\end{aligned}$$

This μ was cooked specifically to make composition possible in this particular structure

A natural question to ask is how specific it is to the task at hand, centered around a particular derivation of binding

Relating μ to η

The grammars we've considered so far are built from functorial operations: η, $\bullet$, $\circledast$

One thing we can readily observe is that for all of R, S, and RS, lifting a value with η and then lowering the result wit μ is a no-op

$$\begin{aligned} \mu_{\mathsf{R}}\,(\eta_{\mathsf{R}}\,\phi) &= \mu_{\mathsf{R}}\,(\lambda_g\,\phi) \\ &= \lambda_g\,(\lambda_g\,\phi)\,g\,g \\ &= \lambda_g\,\phi\,g \\ &= \phi \end{aligned}$$

$$\begin{aligned} \mu_{\mathsf{S}}\,(\eta_{\mathsf{S}}\,\phi) &= \mu_{\mathsf{S}}\,\{\phi\} \\ &= \bigcup\{\phi\} \\ &= \{x \mid x \in \phi\} \\ &= \phi \end{aligned}$$

$$\begin{aligned} \mu_{\mathsf{RS}}\,(\eta_{\mathsf{RS}}\,\phi) &= \mu_{\mathsf{RS}}\,(\lambda_g\,\{\phi\}) \\ &= \lambda_g \bigcup\{m\,g \mid m \in (\lambda_g\,\{\phi\})\,g\} \\ &= \lambda_g \bigcup\{m\,g \mid m \in \{\phi\}\} \\ &= \lambda_g \bigcup\{\phi\,g\} \\ &= \lambda_g\,\phi\,g \\ &= \phi \end{aligned}$$

Relating μ to $\bullet$: R

Even more to the point, given any higher-order structure, it doesn't matter whether we flatten the outer structure first or the inner one

$$\underbrace{\lambda_i \lambda_j \lambda_k \ldots i \ldots j \ldots k \ldots}_{\mu_R}$$

$$\underbrace{\lambda_h \lambda_k \ldots h \ldots h \ldots k \ldots}_{\mu_R}$$

$$\lambda_g \ldots g \ldots g \ldots g \ldots$$

$$\lambda_i \underbrace{\lambda_j \lambda_k \ldots i \ldots j \ldots k \ldots}_{\mu_R}$$

$$\underbrace{\lambda_i \lambda_k \ldots i \ldots k \ldots k \ldots}_{\mu_R}$$

$$\lambda_g \ldots g \ldots g \ldots g \ldots$$

Relating μ to $\bullet$: S

Even more to the point, given any higher-order structure, it doesn't matter whether we flatten the outer structure first or the inner one

$$\underbrace{\{\{\{\ldots\},\ldots\},\{\{\ldots\},\ldots\},\ldots\}}_{\mu_{\mathsf{S}}}$$

$$\underbrace{\{m' \mid m \in \{\ldots\}, m' \in m\}}_{\mu_{\mathsf{S}}}$$

$$\{a \mid m \in \{\ldots\}, m' \in m, a \in m'\}$$

$$\{\underbrace{\{\{\ldots\},\ldots\}}_{\mu_{\mathsf{S}}},\underbrace{\{\{\ldots\},\ldots\}}_{\mu_{\mathsf{S}}},\underbrace{\ldots}_{\mu_{\mathsf{S}}}\}$$

$$\underbrace{\{\{a \mid m' \in \{\ldots\}, a \in m'\},\{a \mid m' \in \{\ldots\}, a \in m'\},\ldots\}}_{\mu_{\mathsf{S}}}$$

$$\{a \mid m \in \{\ldots\}, m' \in m, a \in m'\}$$

Relating μ to $\bullet$: RS

The same is true of RS, though it is a little more tedious to work out

$$\begin{aligned}
\mu_{\mathsf{R}}\,(\mu_{\mathsf{R}}\,M) &= \mu_{\mathsf{R}}\,(\mu_{\mathsf{R}} \bullet M) &&= \lambda_g\, M\,g\,g\,g \\
\mu_{\mathsf{S}}\,(\mu_{\mathsf{S}}\,M) &= \mu_{\mathsf{S}}\,(\mu_{\mathsf{S}} \bullet M) &&= \{a \mid m \in M, m' \in m, a \in m'\} \\
\mu_{\mathsf{RS}}\,(\mu_{\mathsf{RS}}\,M) &= \mu_{\mathsf{RS}}\,(\mu_{\mathsf{RS}} \bullet M) &&= \lambda_g\, \{a \mid m \in M\,g, m' \in m\,g, a \in m'\,g\}
\end{aligned}$$

The set-flattening and environment-sharing are simply interleaved all the way down.

Monads

Indeed, any functor for which there is a μ satisfying these equations is known as a **Monad**

Left Identity	$\mu\,(\eta\,M) = M$
Right Identity	$\mu\,(\eta \bullet M) = M$
Associativity	$\mu\,(\mu\,M) = \mu\,(\mu \bullet M)$

For historical reasons, in Haskell the η of a Monad is called its `return`, and the μ called its `join`

```
return :: Monad f => a -> f a
join :: Monad f => f (f a) -> f a
```

Parser interlude

Again, stretching the parser is no more complicated than composing our existing **modes of combination** with `join`

```
combine' :: Type -> Type -> [(Mode, Type)]
combine' l r = addJ $ combine l r ++
  [ (LR op, Eff f c) | Eff f a <- [l]
                     , functor f, (op, c) <- combine' a r ] ++
  [ (LL op, Eff f c) | Eff f b <- [r]
                     , functor f, (op, c) <- combine' l b ] ++
  [ (A  op, Eff f c) | Eff f a <- [l], Eff g b <- [r], f == g
                     , applicative f, (op, c) <- combine' l b ]

addJ :: [(Mode, Type)] -> [(Mode, Type)]
addJ e = e ++
  [ (J op, Eff f a)
  | (op, Eff f (Eff g a)) <- e
  , monad f
  , f == g ]
```

Refactoring to $\star$

It turns out, an equivalent way to state a monad uses $\star$ and η in place of $\bullet$ and μ

$$\bullet :: (\alpha \to \beta) \to F\alpha \to F\beta \qquad \eta :: \alpha \to F\alpha$$

$$\mu :: FF\alpha \to F\alpha \qquad \star :: F\alpha \to (\alpha \to F\beta) \to F\beta$$

The Haskell name for $\star$ is `>>=`, pronounced, tellingly, as `bind`

```
class Monad f where
  return :: a -> f a
  (>>=) :: f a -> (a -> f b) -> f b
```

The monad laws governing η and $\star$ take the forms:

Left Identity	$\eta\, a \star k = k\, a$
Right Identity	$m \star \eta = m$
Associativity	$(m \star \lambda_a\, n\, a) \star o = m \star (\lambda_a\, n\, a \star o)$

Defining binds

The sense in which the $\bullet$ / μ construction and the η / $\star$ construction are equivalent is that they are interdefinable in a law-preserving way

$$\mu M =$$

Defining binds

The sense in which the $\bullet$ / μ construction and the η / $\star$ construction are equivalent is that they are interdefinable in a law-preserving way

$$
\begin{aligned}
\mu M &= M \star \mathbf{id} \\
m \star k &=
\end{aligned}
$$

Defining binds

The sense in which the $\bullet$ / μ construction and the η / $\star$ construction are equivalent is that they are interdefinable in a law-preserving way

$$\mu M = M \star \mathbf{id}$$

$$m \star k = \mu\,(k \bullet m)$$

Let's work out the $\star$ ('bind') operations for R, S, and RS:

$m \star f :=$

Defining binds

The sense in which the $\bullet$ / μ construction and the η / $\star$ construction are equivalent is that they are interdefinable in a law-preserving way

$$\mu M = M \star \textbf{id}$$

$$m \star k = \mu\,(k \bullet m)$$

Let's work out the $\star$ ('bind') operations for R, S, and RS:

$$m \star f := \lambda_g\, f\,(m\, g)\, g$$

Defining binds

The sense in which the $\bullet$ / μ construction and the η / $\star$ construction are equivalent is that they are interdefinable in a law-preserving way

$$\mu M = M \star \mathbf{id}$$

$$m \star k = \mu\,(k \bullet m)$$

Let's work out the $\star$ ('bind') operations for R, S, and RS:

$m \star f := \lambda_g\, f\,(m\,g)\,g$ $\qquad$ $m \star f :=$

Defining binds

The sense in which the $\bullet$ / μ construction and the η / $\star$ construction are equivalent is that they are interdefinable in a law-preserving way

$$\mu M = M \star \textbf{id}$$

$$m \star k = \mu (k \bullet m)$$

Let's work out the $\star$ ('bind') operations for R, S, and RS:

$m \star f := \lambda_g f (m g) g$ $\qquad$ $m \star f := \bigcup_{x \in m} f x$

Defining binds

The sense in which the $\bullet$ / μ construction and the η / $\star$ construction are equivalent is that they are interdefinable in a law-preserving way

$$\mu M = M \star \textbf{id}$$

$$m \star k = \mu (k \bullet m)$$

Let's work out the $\star$ ('bind') operations for R, S, and RS:

$m \star f := \lambda_g f\,(m\,g)\,g$ $\qquad$ $m \star f := \bigcup_{x \in m} f\,x$ $\qquad$ $m \star f :=$

Defining binds

The sense in which the $\bullet$ / μ construction and the η / $\star$ construction are equivalent is that they are interdefinable in a law-preserving way

$$\mu M = M \star \textbf{id}$$

$$m \star k = \mu\,(k \bullet m)$$

Let's work out the $\star$ ('bind') operations for R, S, and RS:

$$m \star f := \lambda_g\, f\,(m\,g)\,g \qquad m \star f := \bigcup_{x \in m} f\,x \qquad m \star f := \lambda_g \bigcup_{x \in m\,g} f\,x\,g$$

Monads are Applicative

If we harmlessly swap the order of $\star$'s arguments, you can see an interesting progression:

$$\bullet :: (\mathsf{a} \rightarrow \mathsf{b}) \rightarrow \mathsf{F\,a} \rightarrow \mathsf{F\,b}$$

$$\circledast :: \mathsf{F}\,(\mathsf{a} \rightarrow \mathsf{b}) \rightarrow \mathsf{F\,a} \rightarrow \mathsf{F\,b}$$

$$\lambda_k \lambda_m\, m \star k :: (\mathsf{a} \rightarrow \mathsf{F\,b}) \rightarrow \mathsf{F\,a} \rightarrow \mathsf{F\,b}$$

It's not hard to see that $\circledast$ can be defined in terms of $\star$:

$$\circledast :: \mathsf{F}\,(\mathsf{a} \rightarrow \mathsf{b}) \rightarrow \mathsf{F\,a} \rightarrow \mathsf{F\,b}$$

$$F \circledast A =$$

Monads are Applicative

If we harmlessly swap the order of $\star$'s arguments, you can see an interesting progression:

$$\bullet :: (\mathsf{a} \to \mathsf{b}) \to \mathsf{F\,a} \to \mathsf{F\,b}$$

$$\circledast :: \mathsf{F}\,(\mathsf{a} \to \mathsf{b}) \to \mathsf{F\,a} \to \mathsf{F\,b}$$

$$\lambda_k \lambda_m\, m \star k :: (\mathsf{a} \to \mathsf{F\,b}) \to \mathsf{F\,a} \to \mathsf{F\,b}$$

It's not hard to see that $\circledast$ can be defined in terms of $\star$:

$$\circledast :: \mathsf{F}\,(\mathsf{a} \to \mathsf{b}) \to \mathsf{F\,a} \to \mathsf{F\,b}$$

$$F \circledast A = F \star \lambda_f\, A \star \lambda_a\, \eta\,(f\,a)$$

And as long as $\star$ satisfies the Monad laws, the $\circledast$ defined above will be guaranteed to satisfy the Applicative laws

So every Monad is an Applicative

Monads are Functors

If we harmlessly swap the order of $\star$'s arguments, you can see an interesting progression:

$$\bullet :: (\mathsf{a} \to \mathsf{b}) \to \mathsf{F\,a} \to \mathsf{F\,b}$$

$$\circledast :: \mathsf{F}(\mathsf{a} \to \mathsf{b}) \to \mathsf{F\,a} \to \mathsf{F\,b}$$

$$\lambda_k \lambda_m\, m \star k :: (\mathsf{a} \to \mathsf{F\,b}) \to \mathsf{F\,a} \to \mathsf{F\,b}$$

It's not hard to see that $\bullet$ can be defined in terms of $\star$:

$$\bullet :: (\mathsf{a} \to \mathsf{b}) \to \mathsf{F\,a} \to \mathsf{F\,b}$$

$$k \bullet A =$$

Monads are Functors

If we harmlessly swap the order of $\star$'s arguments, you can see an interesting progression:

$$\bullet :: (a \to b) \to F\,a \to F\,b$$

$$\circledast :: F\,(a \to b) \to F\,a \to F\,b$$

$$\lambda_k \lambda_m\, m \star k :: (a \to F\,b) \to F\,a \to F\,b$$

It's not hard to see that $\bullet$ can be defined in terms of $\star$:

$$\bullet :: (a \to b) \to F\,a \to F\,b$$

$$k \bullet A = A \star \lambda_a\, \eta\,(k\,a)$$

And as long as $\star$ satisfies the Monad laws, the $\circledast$ defined above will be guaranteed to satisfy the Functor laws

So every Monad is a Functor

Compared

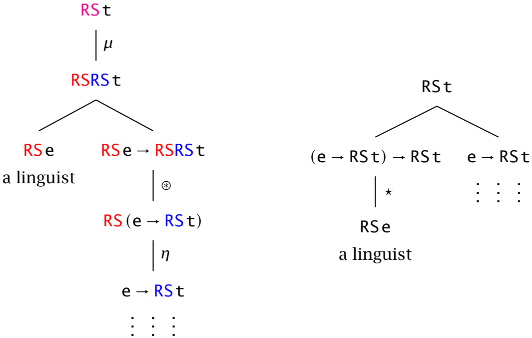
So in general, we have:

$$m \star k = \mu\,(k \bullet m)$$

And given that also:

$$k \bullet m = \eta\, k \circledast m$$

You can see the $\star$ hiding in the chain of type shifts from our binding derivation:



$\mathtt{RS\,t}$

μ

$\mathtt{RSRS\,t}$

$\mathtt{RS\,e}$

a linguist

$\mathtt{RS\,e \to RSRS\,t}$

$\circledast$

$\mathtt{RS\,(e \to RS\,t)}$

η

$\mathtt{e \to RS\,t}$

$\vdots \;\; \vdots \;\; \vdots$

$\mathtt{RS\,t}$

$\mathtt{(e \to RS\,t) \to RS\,t}$

$\star$

$\mathtt{RS\,e}$

a linguist

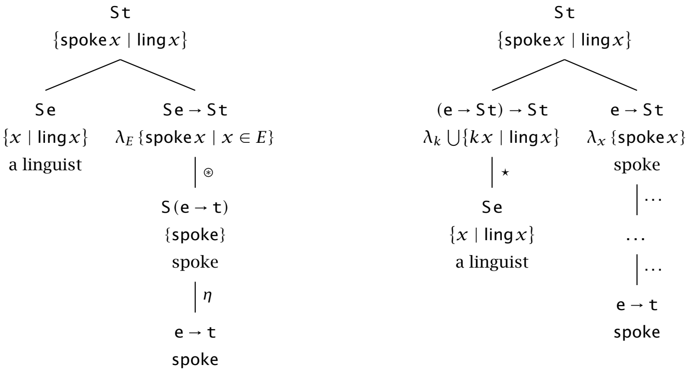
$\mathtt{e \to RS\,t}$

$\vdots \;\; \vdots \;\; \vdots$

How to use $\star$

Adding $\star$ to the grammar isn't as obviously immediately useful as adding $\bullet$ and $\circledast$ because functions of type $(\mathtt{a} \to \mathtt{F}\,\mathtt{b})$ don't occur very naturally in the wild

And moreover, with just the combinators we have, there's no way to pull an $(\mathtt{a} \to \mathtt{F}\,\mathtt{b})$ out of an $(\mathtt{a} \to \mathtt{b})$



$\mathtt{S}\,\mathtt{t}$
$\{\mathsf{spoke}\,x \mid \mathsf{ling}\,x\}$

$\mathtt{S}\,\mathtt{e}$
$\{x \mid \mathsf{ling}\,x\}$
a linguist

$\mathtt{S}\,\mathtt{e} \to \mathtt{S}\,\mathtt{t}$
$\lambda_E\,\{\mathsf{spoke}\,x \mid x \in E\}$
$\circledast$
$\mathtt{S}\,(\mathtt{e} \to \mathtt{t})$
$\{\mathsf{spoke}\}$
spoke
η
$\mathtt{e} \to \mathtt{t}$
spoke

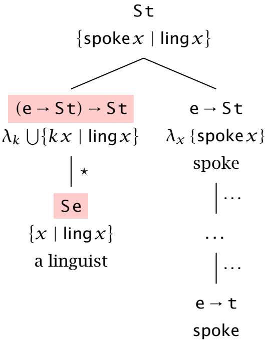
$\mathtt{S}\,\mathtt{t}$
$\{\mathsf{spoke}\,x \mid \mathsf{ling}\,x\}$

$(\mathtt{e} \to \mathtt{S}\,\mathtt{t}) \to \mathtt{S}\,\mathtt{t}$
$\lambda_k \bigcup\{k\,x \mid \mathsf{ling}\,x\}$
$\star$
$\mathtt{S}\,\mathtt{e}$
$\{x \mid \mathsf{ling}\,x\}$
a linguist

$\mathtt{e} \to \mathtt{S}\,\mathtt{t}$
$\lambda_x\,\{\mathsf{spoke}\,x\}$
spoke
$\dots$
$\dots$
$\dots$
$\mathtt{e} \to \mathtt{t}$
spoke

How to use $\star$

But the type signature of the $\star$-shifted subject might set alarm bells ringing if you're a linguist



$\mathsf{S\,t}$

$\{\mathsf{spoke}\,x \mid \mathsf{ling}\,x\}$

$(\mathsf{e} \to \mathsf{S\,t}) \to \mathsf{S\,t}$

$\lambda_k \bigcup\{k\,x \mid \mathsf{ling}\,x\}$

$\star$

$\mathsf{S\,e}$

$\{x \mid \mathsf{ling}\,x\}$

a linguist

$\mathsf{e} \to \mathsf{S\,t}$

$\lambda_x\,\{\mathsf{spoke}\,x\}$

spoke

$\dots$

$\dots$

$\dots$

$\mathsf{e} \to \mathsf{t}$

spoke

It looks an awful lot like good old LIFT-ing

$$\text{LIFT} :: \mathsf{e} \to (\mathsf{e} \to \mathsf{t}) \to \mathsf{t}$$

$$\star :: \mathsf{F\,e} \to (\mathsf{e} \to \mathsf{F\,t}) \to \mathsf{F\,t}$$

$\star$ and scope

In fact, if your fancy individual is not actually fancy, then the first Monad law

$$\textbf{Left Identity} \quad \eta\, a \star k = k\, a$$

just says that

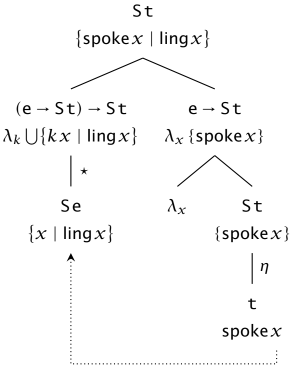
$$(\eta\, a)^{\star} = \lambda_k\, k\, a = \textsc{lift}\, a$$

This makes you wonder if you can use any of the techniques invented to deal with Generalized Quantifiers to facilitate composition

In particular, it calls for a theory of scope

Scope via “Q”R

This might take the form of re-introducing raising and abstraction into the syntax:



$\mathtt{St}$

$\{\mathsf{spoke}\,x \mid \mathsf{ling}\,x\}$

$(\mathtt{e} \to \mathtt{St}) \to \mathtt{St}$

$\lambda_k \bigcup \{k\,x \mid \mathsf{ling}\,x\}$

$\star$

$\mathtt{Se}$

$\{x \mid \mathsf{ling}\,x\}$

$\mathtt{e} \to \mathtt{St}$

$\lambda_x \{\mathsf{spoke}\,x\}$

λ_x

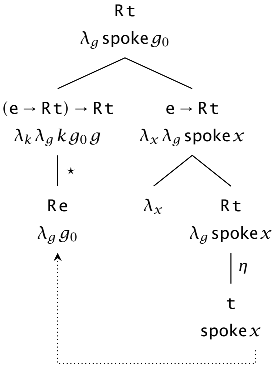
$\mathtt{St}$

$\{\mathsf{spoke}\,x\}$

η

$\mathtt{t}$

$\mathsf{spoke}\,x$



$\mathtt{Rt}$

$\lambda_g\,\mathsf{spoke}\,g_0$

$(\mathtt{e} \to \mathtt{Rt}) \to \mathtt{Rt}$

$\lambda_k \lambda_g\,k\,g_0\,g$

$\star$

$\mathtt{Re}$

$\lambda_g\,g_0$

$\mathtt{e} \to \mathtt{Rt}$

$\lambda_x \lambda_g\,\mathsf{spoke}\,x$

λ_x

$\mathtt{Rt}$

$\lambda_g\,\mathsf{spoke}\,x$

η

$\mathtt{t}$

$\mathsf{spoke}\,x$

(These are guaranteed to deliver the same results as the derivations with $\circledast$)

Scope via C

Or alternatively, you might recall that scope-taking itself is a kind of effect

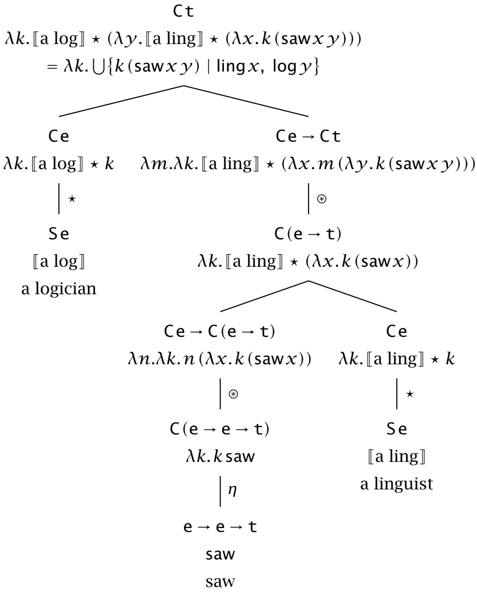
$$\begin{aligned} \mathtt{C\,e} &::= (\mathtt{e} \to \mathtt{F\,t}) \to \mathtt{F\,t} \\ \eta\, x &= \lambda_k\, k\, x \\ m \circledast n &= \lambda_k\, m\, (\lambda_f\, n\, (\lambda_x\, k\, (f\, x))) \end{aligned}$$

From this perspective, $\star$ looks like a kind of **Natural Transformation** from one effect to another

$$\star :: \mathtt{F\,a} \to \mathtt{C\,a}$$

In which case, we should be able to use C's $\circledast$ to handle composition

Scope via C in action



$\mathtt{C\,t}$

$\lambda k.[\![\text{a log}]\!] \star (\lambda y.[\![\text{a ling}]\!] \star (\lambda x.k\,(\mathsf{saw}\,x\,y)))$

$= \lambda k.\bigcup\{k\,(\mathsf{saw}\,x\,y) \mid \mathsf{ling}\,x,\ \mathsf{log}\,y\}$

$\mathtt{C\,e}$

$\lambda k.[\![\text{a log}]\!] \star k$

$\star$

$\mathtt{S\,e}$

$[\![\text{a log}]\!]$

a logician

$\mathtt{C\,e \to C\,t}$

$\lambda m.\lambda k.[\![\text{a ling}]\!] \star (\lambda x.m\,(\lambda y.k\,(\mathsf{saw}\,x\,y)))$

$\circledast$

$\mathtt{C\,(e \to t)}$

$\lambda k.[\![\text{a ling}]\!] \star (\lambda x.k\,(\mathsf{saw}\,x))$

$\mathtt{C\,e \to C\,(e \to t)}$

$\lambda n.\lambda k.n\,(\lambda x.k\,(\mathsf{saw}\,x))$

$\circledast$

$\mathtt{C\,(e \to e \to t)}$

$\lambda k.k\,\mathsf{saw}$

η

$\mathtt{e \to e \to t}$

saw

saw

$\mathtt{C\,e}$

$\lambda k.[\![\text{a ling}]\!] \star k$

$\star$

$\mathtt{S\,e}$

$[\![\text{a ling}]\!]$

a linguist

More Monads

As it happens, nearly all of the basic Functors we've introduced as case studies turn out to be Monads

For instance,

```
data Maybe a = Just a | Nothing
instance Monad Maybe where
  return a = Just a
  join m = case m of
    Just (Just a) -> a
    _             -> Nothing
```

More Monads

As it happens, nearly all of the basic Functors we've introduced as case studies turn out to be Monads

For instance,

```
data Maybe a = Just a | Nothing
instance Monad Maybe where
  return a = Just a
  join m = case m of
    Just (Just a) -> a
    _             -> Nothing
```

```
data Writer w a = Writer (a, w)

instance Monad (Writer [E]) where
  return a = ...
```

More Monads

As it happens, nearly all of the basic Functors we've introduced as case studies turn out to be Monads

For instance,

```
data Maybe a = Just a | Nothing
instance Monad Maybe where
  return a = Just a
  join m = case m of
    Just (Just a) -> a
    _             -> Nothing
```

```
data Writer w a = Writer (a, w)

instance Monad (Writer [E]) where
  return a = Writer (a, [])
  join (Writer (Writer (a, xs), ys)) = ...
```

More Monads

As it happens, nearly all of the basic Functors we've introduced as case studies turn out to be Monads

For instance,

```
data Maybe a = Just a | Nothing
instance Monad Maybe where
  return a = Just a
  join m = case m of
    Just (Just a) -> a
    _             -> Nothing
```

```
data Writer w a = Writer (a, w)

instance Monad (Writer [E]) where
  return a = Writer (a, [])
  join (Writer (Writer (a, xs), ys)) = Writer (a, xs ++ ys)
```

Fewer Monads

At the same time, many of the Functors we've seen are not (obviously) Monads

$$\begin{aligned} \mathtt{W}\,\mathtt{R}\,\alpha &= \langle \mathtt{r} \to \alpha, [\mathtt{t}] \rangle \\ \eta\, a &= \ldots \\ \langle f, p \rangle \star k &= \ldots \end{aligned}$$

There's no obvious way to define this even though `W` and `R` are themselves Monads

This means that while…

- the composition of two Functors is a Functor
- the composition of two Applicatives is an Applicative
- ☠ the composition of two Monads is not necesarrily a Monad

Layer with caution

Notably, though RS is a monad (as we've seen), SR is (probably?) not!

$$\begin{aligned} \mathsf{RS}\,\alpha &= \mathsf{r} \to \{\mathsf{a}\} \\ \eta\,a &= \lambda_g\,\{a\} \\ m \star k &= \lambda_g \bigcup\{k\,a \mid a \in m\,g\} \end{aligned}$$

$$\begin{aligned} \mathsf{SR}\,\alpha &= \{\mathsf{r} \to \mathsf{a}\} \\ \eta\,a &= \ldots \\ m \star k &= \ldots \end{aligned}$$

Distributive transformations

So how can you tell when a composition of Monads FG is a Monad? That is, how can you know whether there is a (law-abiding) function

$$\mu_{\mathsf{FG}} :: \mathsf{FGFG}\,\alpha \to \mathsf{FG}\,\alpha$$

One thing to notice is that since F and G are Monads, we are guaranteed functions

$$\mu_{\mathsf{F}} :: \mathsf{FF}\,\alpha \to \mathsf{F}\,\alpha$$

$$\mu_{\mathsf{G}} :: \mathsf{GG}\,\alpha \to \mathsf{G}\,\alpha$$

If we just had a function

$$\Upsilon :: \mathsf{GF}\,\alpha \to \mathsf{FG}\,\alpha,$$

then it seems like we'd be golden, since we could build the following pipeline:

$$\mu_{\mathsf{FG}} = \mathsf{FGFG} \xrightarrow[\Upsilon]{} \mathsf{FFGG} \xrightarrow[\mu_{\mathsf{F}}]{} \mathsf{FGG} \xrightarrow[\mu_{\mathsf{G}}]{} \mathsf{FG}$$

Distributing S over R

And indeed, for the composition RS, there's a natural way to get home when the effects are inverted

$$\Upsilon :: \mathsf{SR}\,\alpha \to \mathsf{RS}\,\alpha$$

$$\Upsilon = \ldots$$

It is so natural in fact, it is called a **Distributive Natural Transformation**, which means it satisfies these laws (and a few others)

$$\begin{aligned} \Upsilon\,(\eta_{\mathsf{R}} \bullet_{\mathsf{S}} S) &= \eta_{\mathsf{R}}\,S \\ \Upsilon\,(\eta_{\mathsf{S}}\,R) &= \eta_{\mathsf{S}} \bullet_{\mathsf{R}} R \\ f \bullet_{\mathsf{RS}} \Upsilon M &= \Upsilon\,(f \bullet_{\mathsf{SR}} M) \end{aligned}$$

As suspected, any time there is a Distributive $\Upsilon :: \mathsf{GF} \to \mathsf{FG}$ with these properties, you can be sure FG is a Monad[1]

---

[1] Beck 1969

Distributing S over R

And indeed, for the composition RS, there's a natural way to get home when the effects are inverted

$$\Upsilon :: \mathsf{SR}\,\alpha \rightarrow \mathsf{RS}\,\alpha$$

$$\Upsilon = \lambda_M \lambda_g \, \{ f\, g \mid f \in M \}$$

It is so natural in fact, it is called a **Distributive Natural Transformation**, which means it satisfies these laws (and a few others)

$$\Upsilon\,(\eta_{\mathsf{R}} \bullet_{\mathsf{S}} S) = \eta_{\mathsf{R}}\, S$$

$$\Upsilon\,(\eta_{\mathsf{S}}\, R) = \eta_{\mathsf{S}} \bullet_{\mathsf{R}} R$$

$$f \bullet_{\mathsf{RS}} \Upsilon\, M = \Upsilon\,(f \bullet_{\mathsf{SR}} M)$$

As suspected, any time there is a Distributive $\Upsilon :: \mathsf{GF} \rightarrow \mathsf{FG}$ with these properties, you can be sure FG is a Monad[1]

---

[1] Beck 1969

Distributing R over S

But for SR, we'd need to define a function in the opposite direction

$$\Upsilon :: \mathsf{RS}\,\alpha \to \mathsf{SR}\,\alpha$$
$$\Upsilon = \ldots$$

[2] Bumford 2022

Distributing R over S

But for SR, we'd need to define a function in the opposite direction

$$\Upsilon :: \mathsf{R}\mathsf{S}\alpha \rightarrow \mathsf{S}\mathsf{R}\alpha$$
$$\Upsilon = \ldots$$

It turns out that no such function can ever satisfy the Distributive laws[2]

---

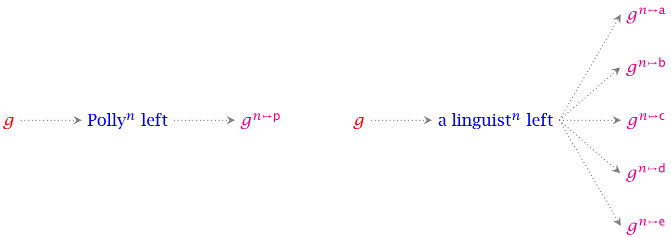
[2] Bumford 2022

Dynamics

Dynamic binding

4. Polly left. She was tired.
5. A linguist left. She was tired.
6. Every linguist left. ??She was tired.

The basic idea[3]



g Polly^n left $g^{n\mapsto \mathsf{p}}$

g a linguist^n left $g^{n\mapsto \mathsf{a}}$ $g^{n\mapsto \mathsf{b}}$ $g^{n\mapsto \mathsf{c}}$ $g^{n\mapsto \mathsf{d}}$ $g^{n\mapsto \mathsf{e}}$

- Dref introduction is assignment modification.
- Indefinites introduce drefs non-deterministically.
- New drefs may (not) pan out downstream (cf. Stalnaker 1978).

---

[3] Heim (1982), Barwise (1987), Groenendijk & Stokhof (1991), and Muskens (1996), etc.

Dynamics

Consider this (standard, DPL-ish) dynamic system:

$$\begin{aligned} [\![\exists x]\!] &:= \lambda_g \left\{ g^{x \mapsto d} \mid d \in D \right\} \\ [\![\phi \wedge \psi]\!] &:= \lambda_g \{ h \in k[\psi] \mid k \in g[\phi] \} \end{aligned}$$

Consider what ‘effects’ are embodied in this system (in what ways is it ‘richer’ than the basic grammar we began with?).

A dynamic monad

Can you devise an applicative type constructor F that does justice to these effects?

Can you devise a trivial way to 'lift' an a into an $\mathtt{D}\,a$, and an $\star$ recipe for composing a $a \to \mathtt{D}\,b$ and an $\mathtt{D}\,a$ to give an $\mathtt{D}\,b$?

$$\begin{aligned} \mathtt{RS}\,a &::= \mathtt{g} \to \mathtt{S}\,a & \qquad \mathtt{D}\,a &::= \\ \eta\,x &:= \lambda_g \{x\} & \eta\,x &:= \\ m \star f &:= \lambda_g \textstyle\bigcup_{x \in m\,g} f\,x\,g & m \star f &:= \end{aligned}$$

Bonus food for thought: is $\mathtt{D}$ commutative? Does that seem important?

A dynamic monad

Can you devise an applicative type constructor F that does justice to these effects?

Can you devise a trivial way to 'lift' an a into an $\mathtt{D}\,a$, and an $\star$ recipe for composing a $a \to \mathtt{D}\,b$ and an $\mathtt{D}\,a$ to give an $\mathtt{D}\,b$?

$$\mathtt{RS}\,a ::= \mathtt{g} \to \mathtt{S}\,a \qquad\qquad \mathtt{D}\,a ::= \mathtt{g} \to \mathtt{S}\,(a \times \mathtt{g})$$

$$\eta\,x := \lambda_g\{x\} \qquad\qquad \eta\,x :=$$

$$m \star f := \lambda_g \bigcup_{x \in m\,g} f\,x\,g \qquad\qquad m \star f :=$$

Bonus food for thought: is $\mathtt{D}$ commutative? Does that seem important?

A dynamic monad

Can you devise an applicative type constructor F that does justice to these effects?

Can you devise a trivial way to 'lift' an a into an $\mathtt{D}\,a$, and an $\star$ recipe for composing a $a \to \mathtt{D}\,b$ and an $\mathtt{D}\,a$ to give an $\mathtt{D}\,b$?

$$\begin{aligned} \mathtt{RS}\,a &::= \mathtt{g} \to \mathtt{S}\,a & \qquad \mathtt{D}\,a &::= \mathtt{g} \to \mathtt{S}\,(a \times \mathtt{g}) \\ \eta\, x &:= \lambda_g \{x\} & \eta\, x &:= \lambda_g \{(x, g)\} \\ m \star f &:= \lambda_g \bigcup_{x \in m\,g} f\,x\,g & m \star f &:= \end{aligned}$$

Bonus food for thought: is $\mathtt{D}$ commutative? Does that seem important?

A dynamic monad

Can you devise an applicative type constructor F that does justice to these effects?

Can you devise a trivial way to 'lift' an a into an $\mathtt{D}\,a$, and an $\star$ recipe for composing a $a \to \mathtt{D}\,b$ and an $\mathtt{D}\,a$ to give an $\mathtt{D}\,b$?

$$\mathtt{RS}\,a ::= \mathtt{g} \to \mathtt{S}\,a \qquad\qquad \mathtt{D}\,a ::= \mathtt{g} \to \mathtt{S}\,(a \times \mathtt{g})$$

$$\eta\,x := \lambda_g \{x\} \qquad\qquad \eta\,x := \lambda_g \{(x, g)\}$$

$$m \star f := \lambda_g \bigcup\nolimits_{x \in m\,g} f\,x\,g \qquad\qquad m \star f := \lambda_g \bigcup\nolimits_{(x,h) \in m\,g} f\,x\,h$$

Bonus food for thought: is $\mathtt{D}$ commutative? Does that seem important?

Binding

Remember the binding operator defined earlier

$$\triangleright_n :: \mathsf{R}\,(\mathsf{e} \to \sigma) \to \mathsf{e} \to \mathsf{R}\,\sigma$$

$$\triangleright_n \coloneqq \lambda_m\, \lambda_x\, (\lambda_g\, m\, g^{n \to x}) \circledast \eta\, x$$

With scope-taking in the grammar, we can make this a little simpler, since we can build $(\mathsf{e} \to \mathsf{RF}\,\sigma)$ functions directly[^bur]

$$\triangleright_n \coloneqq \lambda_f \lambda_x\, \lambda_g\, f\, x\, g^{n \to x}$$

Again, this operation is Effect-polymorphic; any Effect beginning with R, including $\mathsf{D} \equiv \mathsf{RSW}$

$$\triangleright_n :: (\mathsf{e} \to \mathsf{R}\,\sigma) \to \mathsf{e} \to \mathsf{R}\,\sigma$$

$$\triangleright_n :: (\mathsf{e} \to \mathsf{RS}\,\sigma) \to \mathsf{e} \to \mathsf{RS}\,\sigma$$

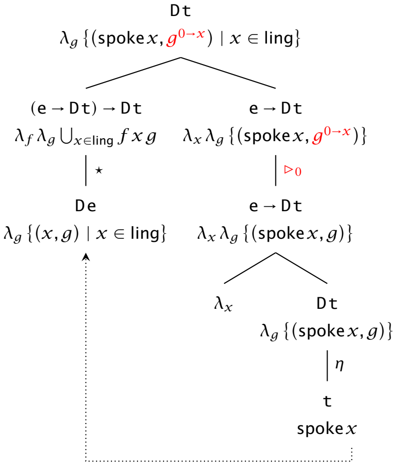
$$\triangleright_n :: (\mathsf{e} \to \mathsf{RW}\,\sigma) \to \mathsf{e} \to \mathsf{RW}\,\sigma$$

$$\triangleright_n :: (\mathsf{e} \to \mathsf{RSW}\,\sigma) \to \mathsf{e} \to \mathsf{RSW}\,\sigma$$

......

Dynamic binding

In D, ▷-referents are stored for later



$\mathtt{D\,t}$

$\lambda_g\,\{(\mathsf{spoke}\,x, g^{0\to x}) \mid x \in \mathsf{ling}\}$

$(\mathtt{e} \to \mathtt{D\,t}) \to \mathtt{D\,t}$

$\lambda_f\,\lambda_g \bigcup_{x\in\mathsf{ling}} f\,x\,g$

$\star$

$\mathtt{D\,e}$

$\lambda_g\,\{(x, g) \mid x \in \mathsf{ling}\}$

$\mathtt{e} \to \mathtt{D\,t}$

$\lambda_x\,\lambda_g\,\{(\mathsf{spoke}\,x, g^{0\to x})\}$

$\triangleright_0$

$\mathtt{e} \to \mathtt{D\,t}$

$\lambda_x\,\lambda_g\,\{(\mathsf{spoke}\,x, g)\}$

λ_x

$\mathtt{D\,t}$

$\lambda_g\,\{(\mathsf{spoke}\,x, g)\}$

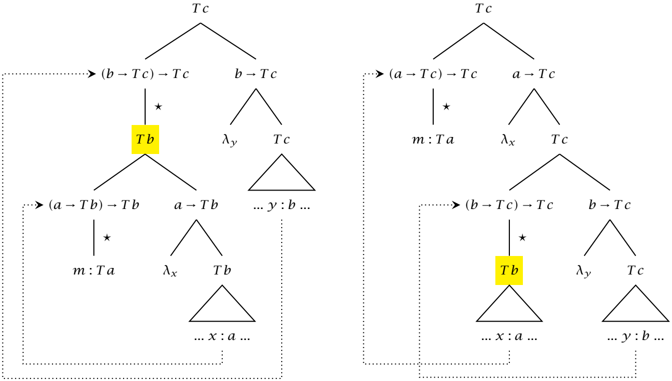
η

$\mathtt{t}$

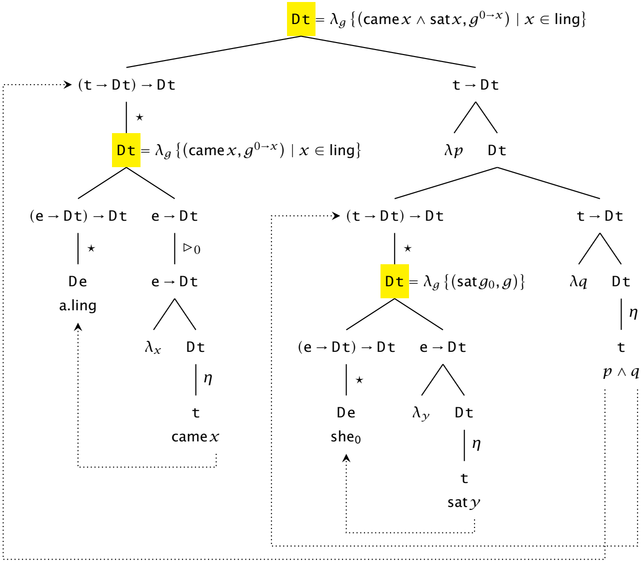
$\mathsf{spoke}\,x$

Associativity

For any monadic T, the **Associativity** law guarantees that the two derivations below are equivalent. It's as if m had scoped out of Tb, without actually doing so



Dynamic binding via static conjunction



Reassociating

Associativity $(m \star \lambda_a\, n\, a) \star o = m \star (\lambda_a\, n\, a \star o)$

$$(\ \text{a.ling}^0 \star \lambda_x\, \eta\,(\text{came}\, x)\ ) \star \lambda_p\, (\ \text{she}_0 \star \lambda_y\, \eta\,(\text{sat}\, y)\ ) \star \lambda_q\, \eta\,(p \wedge q)$$

$$\text{a.ling}^0 \star \lambda_x\, (\ \eta\,(\text{came}\, x) \star \lambda_p\, (\ \text{she}_0 \star \lambda_y\, \eta\,(\text{sat}\, y)\ ) \star \lambda_q\, \eta\,(p \wedge q)\ )$$

Barwise, Jon. 1987. Noun phrases, generalized quantifiers, and anaphora. In Peter Gärdenfors (ed.), *Generalized Quantifiers*, 1–29. Dordrecht: Reidel. `https://doi.org/10.1007/978-94-009-3381-1_1`.

Beck, Jon. 1969. Distributive laws. In *Seminar on triples and categorical homology theory*, 119–140.

Bumford, Dylan. 2022. Composition under distributive natural transformations: or, when predicate abstraction is impossible. *Journal of Logic, Language and Information.* 1–21.

Charlow, Simon. 2019. A modular theory of pronouns and binding. Unpublished ms., Rutgers University. `https://ling.auf.net/lingbuzz/003720`.

Groenendijk, Jeroen & Martin Stokhof. 1991. Dynamic predicate logic. *Linguistics and Philosophy* 14(1). 39–100. `https://doi.org/10.1007/BF00628304`.

Heim, Irene. 1982. *The semantics of definite and indefinite noun phrases*. University of Massachusetts, Amherst Ph.D. thesis. `https://semanticsarchive.net/Archive/Tk0ZmYyY/`.

Kiselyov, Oleg. 2015. Applicative abstract categorial grammars. In Makoto Kanazawa, Lawrence S. Moss & Valeria de Paiva (eds.), *NLCS'15. Third workshop on natural language and computer science*, vol. 32 (EPiC Series), 29–38.

McBride, Conor & Ross Paterson. 2008. Applicative programming with effects. *Journal of Functional Programming* 18(1). 1–13. `https://doi.org/10.1017/S0956796807006326`.

Muskens, Reinhard. 1996. Combining Montague semantics and discourse representation. *Linguistics and Philosophy* 19(2). 143–186. `https://doi.org/10.1007/BF00635836`.

Stalnaker, Robert. 1978. Assertion. In Peter Cole (ed.), *Pragmatics*, vol. 9 (Syntax and Semantics), 315–332. New York: Academic Press.