Polymorphic Higher-order Termination

Łukasz Czajka, TU Dortmund University
Cynthia Kop, Radboud University Nijmegen

June 2019

Polynomial interpretations: first-order

Question: Does the following TRS terminate?

$$\begin{array}{rcl} \mathtt{append}(\mathtt{nil}, ys) & \longrightarrow & ys \\ \mathtt{append}(\mathtt{cons}(x, xs), ys) & \longrightarrow & \mathtt{cons}(x, \mathtt{append}(xs, ys)) \end{array}$$

Polynomial interpretations: first-order

Question: Does the following TRS terminate?

$$\begin{array}{rcl}\texttt{append}(\texttt{nil}, ys) & \longrightarrow & ys \\ \texttt{append}(\texttt{cons}(x, xs), ys) & \longrightarrow & \texttt{cons}(x, \texttt{append}(xs, ys))\end{array}$$

Answer: Yes, it does.

Polynomial interpretations: first-order

Question: Does the following TRS terminate?

$$\begin{array}{rcl} \mathtt{append}(\mathtt{nil}, ys) & \longrightarrow & ys \\ \mathtt{append}(\mathtt{cons}(x, xs), ys) & \longrightarrow & \mathtt{cons}(x, \mathtt{append}(xs, ys)) \end{array}$$

Answer: Yes, it does. Let:

$$\begin{array}{rcl} [\![\mathtt{nil}]\!] & = & 1 \\ [\![\mathtt{cons}]\!](s, t) & = & 1 + [\![s]\!] + [\![t]\!] \\ [\![\mathtt{append}]\!](s, t) & = & 2 * [\![s]\!] + [\![t]\!] \end{array}$$

Polynomial interpretations: first-order

Question: Does the following TRS terminate?

$$\begin{array}{rcl} \texttt{append}(\texttt{nil}, ys) & \longrightarrow & ys \\ \texttt{append}(\texttt{cons}(x, xs), ys) & \longrightarrow & \texttt{cons}(x, \texttt{append}(xs, ys)) \end{array}$$

Answer: Yes, it does. Let:

$$\begin{array}{rcl} [\![\texttt{nil}]\!] & = & 1 \\ [\![\texttt{cons}]\!](s, t) & = & 1 + [\![s]\!] + [\![t]\!] \\ [\![\texttt{append}]\!](s, t) & = & 2 * [\![s]\!] + [\![t]\!] \end{array}$$

Then:

$$\begin{array}{rcl} [\![\texttt{append}(\texttt{nil}, ys)]\!] & = & 2 + ys \\ & > & ys \\ & = & [\![ys]\!] \\ [\![\texttt{append}(\texttt{cons}(x, xs), ys)]\!] & = & 2 + 2x + 2xs + ys \\ & > & 1 + x + 2xs + ys \\ & = & [\![\texttt{cons}(x, \texttt{append}(xs, ys))]\!] \end{array}$$

Polynomial interpretations: higher-order

Question: Does the following higher-order TRS terminate?

$$\begin{array}{rcl} \mathtt{map}(F, \mathtt{nil}) & \longrightarrow & \mathtt{nil} \\ \mathtt{map}(F, \mathtt{cons}(x, xs)) & \longrightarrow & \mathtt{cons}(F \cdot x, \mathtt{map}(F, xs)) \end{array}$$

Polynomial interpretations: higher-order

Question: Does the following higher-order TRS terminate?

$$\begin{array}{rcl} \mathtt{map}(F, \mathtt{nil}) & \longrightarrow & \mathtt{nil} \\ \mathtt{map}(F, \mathtt{cons}(x, xs)) & \longrightarrow & \mathtt{cons}(F \cdot x, \mathtt{map}(F, xs)) \end{array}$$

Answer: Depends.

Polynomial interpretations: higher-order

Question: Does the following higher-order TRS terminate?

$$\begin{array}{rcl} \mathtt{map}(F, \mathtt{nil}) & \longrightarrow & \mathtt{nil} \\ \mathtt{map}(F, \mathtt{cons}(x, xs)) & \longrightarrow & \mathtt{cons}(F \cdot x, \mathtt{map}(F, xs)) \end{array}$$

Answer: Depends. Not terminating if:

$$\begin{array}{rcl} \mathtt{nil} & : & \mathtt{o} \\ \mathtt{cons} & : & (\mathtt{o} \to \mathtt{o}) \to \mathtt{o} \to \mathtt{o} \\ \mathtt{map} & : & ((\mathtt{o} \to \mathtt{o}) \to \mathtt{o} \to \mathtt{o}) \to \mathtt{o} \to \mathtt{o} \end{array}$$

Polynomial interpretations: higher-order

Question: Does the following higher-order TRS terminate?

$$\begin{array}{rcl} \mathtt{map}(F, \mathtt{nil}) & \longrightarrow & \mathtt{nil} \\ \mathtt{map}(F, \mathtt{cons}(x, xs)) & \longrightarrow & \mathtt{cons}(F \cdot x, \mathtt{map}(F, xs)) \end{array}$$

Answer: Depends. Not terminating if:

$$\begin{array}{rcl} \mathtt{nil} & : & \mathsf{o} \\ \mathtt{cons} & : & (\mathsf{o} \to \mathsf{o}) \to \mathsf{o} \to \mathsf{o} \\ \mathtt{map} & : & ((\mathsf{o} \to \mathsf{o}) \to \mathsf{o} \to \mathsf{o}) \to \mathsf{o} \to \mathsf{o} \end{array}$$

Let $\omega := \mathtt{cons}(\lambda x : o.\mathtt{map}(\lambda y : \mathsf{o} \to \mathsf{o}.\lambda z : \mathsf{o}.y\ x, x), \mathtt{nil})$. Then:

$$\begin{array}{ll} & \mathtt{map}(\lambda y : \mathsf{o} \to \mathsf{o}.\lambda z : \mathsf{o}.y\ \omega, \omega) \\ \longrightarrow & \mathtt{cons}(\ (\lambda y.\lambda z.y\ \omega)\ (\lambda x : o.\mathtt{map}(\lambda y : \mathsf{o} \to \mathsf{o}.\lambda z : \mathsf{o}.y\ x, x))\ , \mathtt{map}(\dots)) \\ \longrightarrow_\beta & \mathtt{cons}(\ \lambda z : \mathsf{o}.(\lambda x : o.\mathtt{map}(\lambda y : \mathsf{o} \to \mathsf{o}.\lambda z : \mathsf{o}.y\ x, x))\ \omega\ , \mathtt{map}(\dots)) \\ \longrightarrow_\beta & \mathtt{cons}(\ \lambda z : \mathsf{o}.\underline{\mathtt{map}(\lambda y : \mathsf{o} \to \mathsf{o}.\lambda z : \mathsf{o}.y\ \omega, \omega)}\ , \mathtt{map}(\dots)) \end{array}$$

Polynomial interpretations: higher-order

Question: Does the following higher-order TRS terminate?

$$\begin{array}{rcl} \mathtt{map}(F, \mathtt{nil}) & \longrightarrow & \mathtt{nil} \\ \mathtt{map}(F, \mathtt{cons}(x, xs)) & \longrightarrow & \mathtt{cons}(F \cdot x, \mathtt{map}(F, xs)) \end{array}$$

Answer: Depends.

Polynomial interpretations: higher-order

Question: Does the following higher-order TRS terminate?

$$\begin{array}{rcl} \mathtt{map}(F, \mathtt{nil}) & \longrightarrow & \mathtt{nil} \\ \mathtt{map}(F, \mathtt{cons}(x, xs)) & \longrightarrow & \mathtt{cons}(F \cdot x, \mathtt{map}(F, xs)) \end{array}$$

Answer: Depends. It does terminate if:

$$\begin{array}{rcl} \mathtt{nil} & : & \mathtt{List} \\ \mathtt{cons} & : & \mathtt{Nat} \to \mathtt{List} \to \mathtt{List} \\ \mathtt{map} & : & (\mathtt{Nat} \to \mathtt{Nat}) \to \mathtt{List} \to \mathtt{List} \end{array}$$

Polynomial interpretations: higher-order

Question: Does the following higher-order TRS terminate?

$$\begin{array}{rcl} \mathtt{map}(F, \mathtt{nil}) & \longrightarrow & \mathtt{nil} \\ \mathtt{map}(F, \mathtt{cons}(x, xs)) & \longrightarrow & \mathtt{cons}(F \cdot x, \mathtt{map}(F, xs)) \end{array}$$

Answer: Depends. It does terminate if:

$$\begin{array}{rcl} \mathtt{nil} & : & \mathtt{List} \\ \mathtt{cons} & : & \mathtt{Nat} \rightarrow \mathtt{List} \rightarrow \mathtt{List} \\ \mathtt{map} & : & (\mathtt{Nat} \rightarrow \mathtt{Nat}) \rightarrow \mathtt{List} \rightarrow \mathtt{List} \end{array}$$

Let:

$$\begin{array}{rcl} [\![\mathtt{nil}]\!] & = & 0 \\ [\![\mathtt{cons}(s, t)]\!] & = & 1 + [\![s]\!] + [\![t]\!] \\ [\![\mathtt{map}(s, t)]\!] & = & (2 + [\![t]\!]) * (1 + [\![s]\!]([\![t]\!])) \\ [\![s\ t]\!] & = & [\![s]\!]([\![t]\!]) + [\![t]\!] \quad \text{if } s : \mathtt{Nat} \rightarrow \mathtt{Nat} \end{array}$$

Polynomial interpretations: higher-order

Question: Does the following higher-order TRS terminate?

$$\begin{array}{rcl} \mathtt{map}(F, \mathtt{nil}) & \longrightarrow & \mathtt{nil} \\ \mathtt{map}(F, \mathtt{cons}(x, xs)) & \longrightarrow & \mathtt{cons}(F \cdot x, \mathtt{map}(F, xs)) \end{array}$$

Answer: Depends. It does terminate if:

$$\begin{array}{rcl} \mathtt{nil} & : & \mathtt{List} \\ \mathtt{cons} & : & \mathtt{Nat} \to \mathtt{List} \to \mathtt{List} \\ \mathtt{map} & : & (\mathtt{Nat} \to \mathtt{Nat}) \to \mathtt{List} \to \mathtt{List} \end{array}$$

Let:

$$\begin{array}{rcl} [\![\mathtt{nil}]\!] & = & 0 \\ [\![\mathtt{cons}(s, t)]\!] & = & 1 + [\![s]\!] + [\![t]\!] \\ [\![\mathtt{map}(s, t)]\!] & = & (2 + [\![t]\!]) * (1 + [\![s]\!]([\![t]\!])) \\ [\![s\ t]\!] & = & [\![s]\!]([\![t]\!]) + [\![t]\!] \quad \text{if } s : \mathtt{Nat} \to \mathtt{Nat} \end{array}$$

Then:

$$\begin{array}{rcl} [\![\mathtt{map}(F, \mathtt{nil})]\!] & = & F(0) + 1 \\ & > & 0 = [\![\mathtt{nil}]\!] \\ [\![\mathtt{map}(F, \mathtt{cons}(x, xs))]\!] & = & (2 + F(1 + x + xs)) * (2 + x + xs) \\ & > & 1 + F(x) + x + (2 + xs) * (1 + F(xs)) \\ & = & [\![\mathtt{cons}(F \cdot x, \mathtt{map}(F, xs))]\!] \end{array}$$

Polynomial interpretations: polymorphic higher-order

Question: Does the following polymorphic HO-TRS terminate?

$$\begin{array}{rcl} \mathtt{map}(F, \mathtt{nil}) & \longrightarrow & \mathtt{nil} \\ \mathtt{map}(F, \mathtt{cons}(x, xs)) & \longrightarrow & \mathtt{cons}(F \cdot x, \mathtt{map}(F, xs)) \end{array}$$

$$\begin{array}{rcll} \mathtt{nil} & : & \forall \alpha. & \mathtt{List}(\alpha) \\ \mathtt{cons} & : & \forall \alpha. & \alpha \to \mathtt{List}(\alpha) \to \mathtt{List}(\alpha) \\ \mathtt{map} & : & \forall \alpha \beta. & (\alpha \to \beta) \to \mathtt{List}(\alpha) \to \mathtt{List}(\beta) \end{array}$$

Polynomial interpretations: polymorphic higher-order

Question: Does the following polymorphic HO-TRS terminate?

$$\begin{array}{rcl} \mathtt{map}(F, \mathtt{nil}) & \longrightarrow & \mathtt{nil} \\ \mathtt{map}(F, \mathtt{cons}(x, xs)) & \longrightarrow & \mathtt{cons}(F \cdot x, \mathtt{map}(F, xs)) \end{array}$$

$$\begin{array}{rcll} \mathtt{nil} & : & \forall\alpha. & \mathtt{List}(\alpha) \\ \mathtt{cons} & : & \forall\alpha. & \alpha \to \mathtt{List}(\alpha) \to \mathtt{List}(\alpha) \\ \mathtt{map} & : & \forall\alpha\beta. & (\alpha \to \beta) \to \mathtt{List}(\alpha) \to \mathtt{List}(\beta) \end{array}$$

Question: What about the following?

$$\begin{array}{rcl} \mathtt{fold}(F, a, \mathtt{nil}) & \longrightarrow & a \\ \mathtt{fold}(F, a, \mathtt{cons}(x, xs)) & \longrightarrow & \mathtt{fold}(F, F\ a\ x, xs) \end{array}$$

$$\mathtt{fold} \quad : \quad \forall\alpha\beta. \quad (\beta \to \alpha \to \beta) \to \beta \to \mathtt{List}(\alpha) \to \beta$$

Shallow vs. Higher-rank polymorphism

Shallow polymorphism:

$$
\begin{array}{rcll}
\texttt{nil} & : & \forall\alpha. & \texttt{List}(\alpha) \\
\texttt{cons} & : & \forall\alpha. & \alpha \to \texttt{List}(\alpha) \to \texttt{List}(\alpha) \\
\texttt{fold} & : & \forall\alpha\beta. & (\beta \to \alpha \to \beta) \to \beta \to \texttt{List}(\alpha) \to \beta
\end{array}
$$

$$
\begin{array}{rcl}
\texttt{fold}(F, a, \texttt{nil}) & \longrightarrow & a \\
\texttt{fold}(F, a, \texttt{cons}(x, xs)) & \longrightarrow & \texttt{fold}(F, F\ a\ x, xs)
\end{array}
$$

Shallow vs. Higher-rank polymorphism

Shallow polymorphism:

$$\begin{array}{rcll} \mathtt{List} & : & * \Rightarrow * & \Longleftarrow \\ \mathtt{nil} & : & \forall\alpha. & \mathtt{List}(\alpha) \\ \mathtt{cons} & : & \forall\alpha. & \alpha \to \mathtt{List}(\alpha) \to \mathtt{List}(\alpha) \\ \mathtt{fold} & : & \forall\alpha\beta. & (\beta \to \alpha \to \beta) \to \beta \to \mathtt{List}(\alpha) \to \beta \end{array}$$

$$\begin{array}{rcll} \mathtt{fold}_{\tau,\sigma}(F, a, \mathtt{nil}_\tau) & \longrightarrow & a & \Longleftarrow \\ \mathtt{fold}_{\tau,\sigma}(F, a, \mathtt{cons}_\tau(x, xs)) & \longrightarrow & \mathtt{fold}_{\tau,\sigma}(F, F\ a\ x, xs) & \Longleftarrow \end{array}$$

Shallow vs. Higher-rank polymorphism

Shallow polymorphism:

$$
\begin{array}{rcll}
\mathtt{List} & : & * \Rightarrow * & \\
\mathtt{nil} & : & \forall\alpha. & \mathtt{List}(\alpha) \\
\mathtt{cons} & : & \forall\alpha. & \alpha \rightarrow \mathtt{List}(\alpha) \rightarrow \mathtt{List}(\alpha) \\
\mathtt{fold} & : & \forall\alpha\beta. & (\beta \rightarrow \alpha \rightarrow \beta) \rightarrow \beta \rightarrow \mathtt{List}(\alpha) \rightarrow \beta
\end{array}
$$

$$
\begin{array}{rcl}
\mathtt{fold}_{\tau,\sigma}(F, a, \mathtt{nil}_\tau) & \longrightarrow & a \\
\mathtt{fold}_{\tau,\sigma}(F, a, \mathtt{cons}_\tau(x, xs)) & \longrightarrow & \mathtt{fold}_{\tau,\sigma}(F, F\ a\ x, xs)
\end{array}
$$

Higher-rank polymorphism:

$$
\begin{array}{rcll}
\mathtt{List} & : & & * \\
\mathtt{nil} & : & & \mathtt{List} \\
\mathtt{cons} & : & \forall\alpha. & \alpha \rightarrow \mathtt{List} \rightarrow \mathtt{List} \\
\mathtt{fold} & : & \forall\beta. & (\forall\alpha.\beta \rightarrow \alpha \rightarrow \beta) \rightarrow \beta \rightarrow \mathtt{List} \rightarrow \beta
\end{array}
$$

$$
\begin{array}{rcl}
\mathtt{fold}_{\sigma}(F, a, \mathtt{nil}) & \longrightarrow & a \\
\mathtt{fold}_{\sigma}(F, a, \mathtt{cons}_\tau(x, xs)) & \longrightarrow & \mathtt{fold}_{\sigma}(F, F\ \tau\ a\ x, xs)
\end{array}
$$

Polymorphism

$$\begin{array}{rcl}\mathtt{fold}_\sigma(F, a, \mathtt{nil}) & \longrightarrow & a \\ \mathtt{fold}_\sigma(F, a, \mathtt{cons}_\tau(x, xs)) & \longrightarrow & \mathtt{fold}_\sigma(F, F\ \tau\ x\ a, xs)\end{array}$$

Polymorphism

$$\begin{array}{rcl} \mathtt{fold}_\sigma(F, a, \mathtt{nil}) & \longrightarrow & a \\ \mathtt{fold}_\sigma(F, a, \mathtt{cons}_\tau(x, xs)) & \longrightarrow & \mathtt{fold}_\sigma(F, F\ \tau\ x\ a, xs) \end{array}$$

$$\mathtt{fold}_{\mathtt{List}}(\mathtt{cons}, \mathtt{nil}, \mathtt{cons}_{\tau_1}(a_1, \mathtt{cons}_{\tau_2}(a_2, \mathtt{nil})))$$

Polymorphism

$$
\begin{array}{rcl}
\mathtt{fold}_\sigma(F, a, \mathtt{nil}) & \longrightarrow & a \\
\mathtt{fold}_\sigma(F, a, \mathtt{cons}_\tau(x, xs)) & \longrightarrow & \mathtt{fold}_\sigma(F, F\ \tau\ x\ a, xs)
\end{array}
$$

$$
\begin{array}{l}
\mathtt{fold}_{\mathtt{List}}(\mathtt{cons}, \mathtt{nil}, \mathtt{cons}_{\tau_1}(a_1, \mathtt{cons}_{\tau_2}(a_2, \mathtt{nil}))) \quad \longrightarrow \\
\mathtt{fold}_{\mathtt{List}}(\mathtt{cons}, \mathtt{cons}_{\tau_1}(a_1, \mathtt{nil}), \mathtt{cons}_{\tau_2}(a_2, \mathtt{nil}))
\end{array}
$$

Polymorphism

$$\begin{array}{rcl} \mathtt{fold}_\sigma(F, a, \mathtt{nil}) & \longrightarrow & a \\ \mathtt{fold}_\sigma(F, a, \mathtt{cons}_\tau(x, xs)) & \longrightarrow & \mathtt{fold}_\sigma(F, F\ \tau\ x\ a, xs) \end{array}$$

$$\begin{array}{ll} \mathtt{fold}_{\mathtt{List}}(\mathtt{cons}, \mathtt{nil}, \mathtt{cons}_{\tau_1}(a_1, \mathtt{cons}_{\tau_2}(a_2, \mathtt{nil}))) & \longrightarrow \\ \mathtt{fold}_{\mathtt{List}}(\mathtt{cons}, \mathtt{cons}_{\tau_1}(a_1, \mathtt{nil}), \mathtt{cons}_{\tau_2}(a_2, \mathtt{nil})) & \longrightarrow \\ \mathtt{fold}_{\mathtt{List}}(\mathtt{cons}, \mathtt{cons}_{\tau_2}(a_2, \mathtt{cons}_{\tau_1}(a_1, \mathtt{nil})), \mathtt{nil}) & \end{array}$$

Polymorphism

$$\begin{aligned} \mathtt{fold}_\sigma(F, a, \mathtt{nil}) &\longrightarrow a \\ \mathtt{fold}_\sigma(F, a, \mathtt{cons}_\tau(x, xs)) &\longrightarrow \mathtt{fold}_\sigma(F, F\ \tau\ x\ a, xs) \end{aligned}$$

$$\begin{aligned} \mathtt{fold}_{\mathtt{List}}(\mathtt{cons}, \mathtt{nil}, \mathtt{cons}_{\tau_1}(a_1, \mathtt{cons}_{\tau_2}(a_2, \mathtt{nil}))) &\longrightarrow \\ \mathtt{fold}_{\mathtt{List}}(\mathtt{cons}, \mathtt{cons}_{\tau_1}(a_1, \mathtt{nil}), \mathtt{cons}_{\tau_2}(a_2, \mathtt{nil})) &\longrightarrow \\ \mathtt{fold}_{\mathtt{List}}(\mathtt{cons}, \mathtt{cons}_{\tau_2}(a_2, \mathtt{cons}_{\tau_1}(a_1, \mathtt{nil})), \mathtt{nil}) &\longrightarrow \\ \mathtt{cons}_{\tau_2}(a_2, \mathtt{cons}_{\tau_1}(a_1, \mathtt{nil})) & \end{aligned}$$

Our goal: handle systems like this!

Polymorphic functional systems

- Function symbols have a type: $\forall \alpha_1 \dots \alpha_n . \sigma_1 \rightarrow \dots \rightarrow \sigma_k \rightarrow \tau$
- Terms are variables, abstractions, type abstractions and: $\mathtt{f}_{\pi_1,\dots,\pi_n}(s_1, \dots, s_k) : \tau[\alpha_1 := \pi_1, \dots, \alpha_n := \pi_n]$.

Our goal: handle systems like this!

Polymorphic functional systems

- Function symbols have a type: $\forall \alpha_1 \dots \alpha_n . \sigma_1 \to \dots \to \sigma_k \to \tau$
- Terms are variables, abstractions, type abstractions and: $\mathtt{f}_{\pi_1,\dots,\pi_n}(s_1,\dots,s_k) : \tau[\alpha_1 := \pi_1,\dots,\alpha_n := \pi_n]$.

$$
\begin{array}{rcll}
\mathtt{List} & : & & * \\
\mathtt{nil} & : & & \mathtt{List} \\
\mathtt{cons} & : & \forall \alpha. & \alpha \to \mathtt{List} \to \mathtt{List} \\
\mathtt{fold} & : & \forall \beta. & (\forall \alpha.\beta \to \alpha \to \beta) \to \beta \to \mathtt{List} \to \beta
\end{array}
$$

$$
\begin{array}{rcl}
\mathtt{fold}_\sigma(F, a, \mathtt{nil}) & \longrightarrow & a \\
\mathtt{fold}_\sigma(F, a, \mathtt{cons}_\tau(x, xs)) & \longrightarrow & \mathtt{fold}_\sigma(F, F\ \tau\ a\ x, xs)
\end{array}
$$

Our goal: handle systems like this!

Polymorphic functional systems

- Function symbols have a type: $\forall \alpha_1 \dots \alpha_n . \sigma_1 \to \dots \to \sigma_k \to \tau$
- Terms are variables, abstractions, type abstractions and: $\mathtt{f}_{\pi_1,\dots,\pi_n}(s_1,\dots,s_k) : \tau[\alpha_1 := \pi_1,\dots,\alpha_n := \pi_n]$.

$$\begin{array}{rcll}
\mathtt{List} & : & & * \\
\mathtt{nil} & : & & \mathtt{List} \\
\mathtt{cons} & : & \forall \alpha . & \alpha \to \mathtt{List} \to \mathtt{List} \\
\mathtt{fold} & : & \forall \beta . & (\forall \alpha . \beta \to \alpha \to \beta) \to \beta \to \mathtt{List} \to \beta \\
@ & : & \forall \alpha \forall \beta . & (\alpha \to \beta) \to (\alpha \to \beta) \quad \Longleftarrow \\
\mathtt{tapp} & : & \forall \alpha : * \Rightarrow * . \forall \beta . & (\forall \gamma . \alpha \gamma) \to \alpha \beta \quad \Longleftarrow
\end{array}$$

$$\begin{array}{rcl}
\mathtt{fold}_\sigma(F, a, \mathtt{nil}) & \longrightarrow & a \\
\mathtt{fold}_\sigma(F, a, \mathtt{cons}_\tau(x, xs)) & \longrightarrow & \mathtt{fold}_\sigma(F, F\ \tau\ a\ x, xs)
\end{array}$$

$$\begin{array}{rcll}
@_{\sigma,\tau}(\lambda x : \sigma . s., t) & \longrightarrow & s[x := t] & \Longleftarrow \\
\mathtt{tapp}_{\lambda \alpha . \sigma, \tau}(\Lambda \alpha . s) & \longrightarrow & s[\alpha := \tau] & \Longleftarrow
\end{array}$$

Our goal: handle systems like this!

Polymorphic functional systems

- Function symbols have a type: $\forall \alpha_1 \dots \alpha_n . \sigma_1 \to \dots \to \sigma_k \to \tau$
- Terms are variables, abstractions, type abstractions and: $\mathtt{f}_{\pi_1,\dots,\pi_n}(s_1, \dots, s_k) : \tau[\alpha_1 := \pi_1, \dots, \alpha_n := \pi_n]$.

$$
\begin{array}{rcll}
\mathtt{List} & : & & * \\
\mathtt{nil} & : & & \mathtt{List} \\
\mathtt{cons} & : & \forall \alpha. & \alpha \to \mathtt{List} \to \mathtt{List} \\
\mathtt{fold} & : & \forall \beta. & (\forall \alpha . \beta \to \alpha \to \beta) \to \beta \to \mathtt{List} \to \beta \\
@ & : & \forall \alpha \forall \beta. & (\alpha \to \beta) \to (\alpha \to \beta) \\
\mathtt{tapp} & : & \forall \alpha : * \Rightarrow * . \forall \beta. & (\forall \gamma . \alpha \gamma) \to \alpha \beta
\end{array}
$$

$$
\begin{array}{rcl}
\mathtt{fold}_\sigma(F, a, \mathtt{nil}) & \longrightarrow & a \\
\mathtt{fold}_\sigma(F, a, \mathtt{cons}_\tau(x, xs)) & \longrightarrow & \\
\multicolumn{3}{r}{\mathtt{fold}_\sigma(F, @_{\tau,\sigma}(@_{\sigma,\tau\to\sigma}(\mathtt{tapp}_{\lambda\alpha.\sigma\to\alpha\to\sigma,\tau}(F), a), x), xs)} \\
@_{\sigma,\tau}(\lambda x : \sigma . s., t) & \longrightarrow & s[x := t] \\
\mathtt{tapp}_{\lambda\alpha.\sigma,\tau}(\Lambda \alpha . s) & \longrightarrow & s[\alpha := \tau]
\end{array}
$$

Our goal: handle systems like this!

Polymorphic functional systems

- Function symbols have a type: $\forall \alpha_1 \dots \alpha_n . \sigma_1 \rightarrow \dots \rightarrow \sigma_k \rightarrow \tau$
- Terms are variables, abstractions, type abstractions and: $\mathtt{f}_{\pi_1,\dots,\pi_n}(s_1,\dots,s_k) : \tau[\alpha_1 := \pi_1, \dots, \alpha_n := \pi_n]$.

$$
\begin{array}{rcll}
\mathtt{List} & : & & * \\
\mathtt{nil} & : & & \mathtt{List} \\
\mathtt{cons} & : & \forall \alpha. & \alpha \rightarrow \mathtt{List} \rightarrow \mathtt{List} \\
\mathtt{fold} & : & \forall \beta. & (\forall \alpha.\beta \rightarrow \alpha \rightarrow \beta) \rightarrow \beta \rightarrow \mathtt{List} \rightarrow \beta \\
@ & : & \forall \alpha \forall \beta. & (\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \beta) \\
\mathtt{tapp} & : & \forall \alpha : * \Rightarrow *. \forall \beta. & (\forall \gamma . \alpha\gamma) \rightarrow \alpha\beta
\end{array}
$$

$$
\begin{array}{rcl}
\mathtt{fold}_\sigma(F, a, \mathtt{nil}) & \longrightarrow & a \\
\mathtt{fold}_\sigma(F, a, \mathtt{cons}_\tau(x, xs)) & \longrightarrow & \\
\multicolumn{3}{r}{\mathtt{fold}_\sigma(F, @_{\tau,\sigma}(@_{\sigma,\tau\rightarrow\sigma}(\mathtt{tapp}_{\lambda\alpha.\sigma\rightarrow\alpha\rightarrow\sigma,\tau}(F), a), x), xs)} \\
@_{\sigma,\tau}(\lambda x : \sigma.s., t) & \longrightarrow & s[x := t] \\
\mathtt{tapp}_{\lambda\alpha.\sigma,\tau}(\Lambda\alpha.s) & \longrightarrow & s[\alpha := \tau]
\end{array}
$$

- **Note: not meant as a formalism of interest by itself, but only as a tool to analyse polymorphic systems.**

Monomorphic higher-order algebras [Pol96]

Idea:

- Choose a set $\mathcal{A}$ with well-founded ordering $\succ$.
- Define:
  - $\mathcal{WM}_\kappa = \mathcal{A}$ if κ is a base type
  - $\mathcal{WM}_{\sigma \to \tau} = \{f \in \mathcal{WM}_\sigma \Longrightarrow \mathcal{WM}_\tau \mid f \text{ is weakly monotonic}\}$.
- Terms of type σ are mapped to $\mathcal{WM}_\Sigma$.

Natural choice: natural numbers

Monomorphic higher-order algebras [Pol96]

$$
\begin{array}{rcl}
\mathtt{map}(F, \mathtt{nil}) & \longrightarrow & \mathtt{nil} \\
\mathtt{map}(F, \mathtt{cons}(x, xs)) & \longrightarrow & \mathtt{cons}(F \cdot x, \mathtt{map}(F, xs))
\end{array}
$$

$$
\begin{array}{rcl}
\mathtt{nil} & : & \mathtt{List} \\
\mathtt{cons} & : & \mathtt{Nat} \rightarrow \mathtt{List} \rightarrow \mathtt{List} \\
\mathtt{map} & : & (\mathtt{Nat} \rightarrow \mathtt{Nat}) \rightarrow \mathtt{List} \rightarrow \mathtt{List}
\end{array}
$$

Interpretation to $\mathbb{N}$:

Monomorphic higher-order algebras [Pol96]

$$\begin{array}{rcl} \texttt{map}(F, \texttt{nil}) & \longrightarrow & \texttt{nil} \\ \texttt{map}(F, \texttt{cons}(x, xs)) & \longrightarrow & \texttt{cons}(F \cdot x, \texttt{map}(F, xs)) \end{array}$$

$$\begin{array}{rcl} \texttt{nil} & : & \texttt{List} \\ \texttt{cons} & : & \texttt{Nat} \rightarrow \texttt{List} \rightarrow \texttt{List} \\ \texttt{map} & : & (\texttt{Nat} \rightarrow \texttt{Nat}) \rightarrow \texttt{List} \rightarrow \texttt{List} \end{array}$$

Interpretation to $\mathbb{N}$:

- $\mathcal{WM}_{\texttt{List}} = \mathcal{WM}_{\texttt{Nat}} = \mathbb{N}$

Monomorphic higher-order algebras [Pol96]

$$\begin{array}{rcl} \mathtt{map}(F, \mathtt{nil}) & \longrightarrow & \mathtt{nil} \\ \mathtt{map}(F, \mathtt{cons}(x, xs)) & \longrightarrow & \mathtt{cons}(F \cdot x, \mathtt{map}(F, xs)) \end{array}$$

$$\begin{array}{rcl} \mathtt{nil} & : & \mathtt{List} \\ \mathtt{cons} & : & \mathtt{Nat} \to \mathtt{List} \to \mathtt{List} \\ \mathtt{map} & : & (\mathtt{Nat} \to \mathtt{Nat}) \to \mathtt{List} \to \mathtt{List} \end{array}$$

Interpretation to $\mathbb{N}$:

- $\mathcal{WM}_{\mathtt{List}} = \mathcal{WM}_{\mathtt{Nat}} = \mathbb{N}$
- $\mathcal{WM}_{\mathtt{Nat} \to \mathtt{Nat}} = \{$weakly monotonic functions from $\mathbb{N}$ to $\mathbb{N}\}$

Monomorphic higher-order algebras [Pol96]

$$
\begin{array}{rcl}
\mathtt{map}(F, \mathtt{nil}) & \longrightarrow & \mathtt{nil} \\
\mathtt{map}(F, \mathtt{cons}(x, xs)) & \longrightarrow & \mathtt{cons}(F \cdot x, \mathtt{map}(F, xs))
\end{array}
$$

$$
\begin{array}{rcl}
\mathtt{nil} & : & \mathtt{List} \\
\mathtt{cons} & : & \mathtt{Nat} \to \mathtt{List} \to \mathtt{List} \\
\mathtt{map} & : & (\mathtt{Nat} \to \mathtt{Nat}) \to \mathtt{List} \to \mathtt{List}
\end{array}
$$

Interpretation to $\mathbb{N}$:

- $\mathcal{WM}_{\mathtt{List}} = \mathcal{WM}_{\mathtt{Nat}} = \mathbb{N}$
- $\mathcal{WM}_{\mathtt{Nat} \to \mathtt{Nat}} = \{$weakly monotonic functions from $\mathbb{N}$ to $\mathbb{N}\}$
- $\mathcal{WM}_{\mathtt{Nat} \to \mathtt{List} \to \mathtt{List}} = \{$weakly monotonic functions from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}\}$

Monomorphic higher-order algebras [Pol96]

$$\begin{aligned} \mathtt{map}(F, \mathtt{nil}) &\longrightarrow \mathtt{nil} \\ \mathtt{map}(F, \mathtt{cons}(x, xs)) &\longrightarrow \mathtt{cons}(F \cdot x, \mathtt{map}(F, xs)) \end{aligned}$$

$$\begin{aligned} \mathtt{nil} &: \mathtt{List} \\ \mathtt{cons} &: \mathtt{Nat} \rightarrow \mathtt{List} \rightarrow \mathtt{List} \\ \mathtt{map} &: (\mathtt{Nat} \rightarrow \mathtt{Nat}) \rightarrow \mathtt{List} \rightarrow \mathtt{List} \end{aligned}$$

Interpretation to $\mathbb{N}$:

- $\mathcal{WM}_{\mathtt{List}} = \mathcal{WM}_{\mathtt{Nat}} = \mathbb{N}$
- $\mathcal{WM}_{\mathtt{Nat}\rightarrow\mathtt{Nat}} = \{$weakly monotonic functions from $\mathbb{N}$ to $\mathbb{N}\}$
- $\mathcal{WM}_{\mathtt{Nat}\rightarrow\mathtt{List}\rightarrow\mathtt{List}} = \{$weakly monotonic functions from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}\}$
- $\mathcal{WM}_{(\mathtt{Nat}\rightarrow\mathtt{Nat})\rightarrow\mathtt{List}\rightarrow\mathtt{List}} = \{$weakly monotonic functionals from $\mathcal{WM}_{\mathtt{Nat}\rightarrow\mathtt{Nat}} \times \mathbb{N}$ to $\mathbb{N}\}$

Monomorphic higher-order polynomial interpretations [FuhKop12]

Interpretation:

$$
\begin{array}{rcl}
[\![\texttt{nil}]\!] & = & 0 \\
[\![\texttt{cons}(s,t)]\!] & = & 1 + [\![s]\!] + [\![t]\!] \\
[\![\texttt{map}(s,t)]\!] & = & (2 + [\![t]\!]) * (1 + [\![s]\!]([\![t]\!])) \\
[\![s\ t]\!] & = & [\![s]\!]([\![t]\!]) + [\![t]\!] \quad \text{if } s : \texttt{Nat} \rightarrow \texttt{Nat}
\end{array}
$$

Monomorphic higher-order polynomial interpretations [FuhKop12]

Interpretation:

$$
\begin{array}{rcl}
[\![\texttt{nil}]\!] & = & 0 \\
[\![\texttt{cons}(s,t)]\!] & = & 1 + [\![s]\!] + [\![t]\!] \\
[\![\texttt{map}(s,t)]\!] & = & (2 + [\![t]\!]) * (1 + [\![s]\!]([\![t]\!])) \\
[\![s\ t]\!] & = & [\![s]\!]([\![t]\!]) + [\![t]\!] \ \ \text{if } s : \texttt{Nat} \rightarrow \texttt{Nat}
\end{array}
$$

Put differently:

$$
\begin{array}{rcl}
[\![\texttt{nil}]\!] & = & 0 \\
[\![\texttt{cons}]\!] & = & \lambda xy.1 + x + y \\
[\![\texttt{map}]\!] & = & \lambda fx.(2 + x) * (1 + f(x)) \\
[\![@_{\texttt{Nat},\texttt{Nat}}]\!] & = & \lambda fx.f(x) + x
\end{array}
$$

Polymorphic higher-order interpretations (our work)

- Problem: set-theoretic interpretation **fails**.

  (How to interpret $\Lambda\alpha.\lambda x : \alpha.x$?)

Polymorphic higher-order interpretations (our work)

- Problem: set-theoretic interpretation **fails**.

  (How to interpret $\Lambda\alpha.\lambda x : \alpha.x$?)

- Alternative: use polynomial interpretations to a set of terms, not functions!

Polymorphic higher-order interpretations (our work)

- Problem: set-theoretic interpretation **fails**.

  (How to interpret $\Lambda\alpha.\lambda x : \alpha.x$?)
- Alternative: use polynomial interpretations to a set of terms, not functions!

Example:

$$\begin{array}{rcll} \texttt{nil} & : & \forall\alpha. & \texttt{List}(\alpha) \\ \texttt{cons} & : & \forall\alpha. & \alpha \to \texttt{List}(\alpha) \to \texttt{List}(\alpha) \\ \texttt{map} & : & \forall\alpha\beta. & (\alpha \to \beta) \to \texttt{List}(\alpha) \to \texttt{List}(\beta) \end{array}$$

$$\begin{array}{rcll} [\![\texttt{List}(\alpha)]\!] & = & \alpha & \\ [\![\texttt{nil}]\!] & = & \Lambda\alpha. & \texttt{lift}_\alpha(0) \\ [\![\texttt{cons}]\!] & = & \Lambda\alpha.\lambda xy. & \texttt{lift}_\alpha(1) \oplus x \oplus y \\ [\![\texttt{map}]\!] & = & \Lambda\alpha\beta.\lambda fx. & (\texttt{lift}_\beta(2) \oplus \texttt{lift}_\beta(\texttt{flatten}_\alpha(x))) \\ & & & \otimes\ (\texttt{lift}_\beta(1) \oplus f(x)) \end{array}$$

Overview

Step 1. Define a set $\mathcal{I}$ of interpretation terms.

Step 2. Define an ordering relation $\succ$ on the set $\mathcal{I}$.

Step 3. Systematically map terms s to interpretation terms $[\![s]\!]$.

Step 4. Prove that if $s \longrightarrow_{\mathcal{R}} t$ then $[\![s]\!] \succ [\![t]\!]$.

A set of interpretation terms

An extension of system $\mathtt{F}_\omega$

One type constant `nat`.

A set of interpretation terms

An extension of system $\mathtt{F}_\omega$

One type constant `nat`.

Terms: variables, abstractions, type (constructor) abstractions, natural number constants, and function symbols

A set of interpretation terms

An extension of system $\mathtt{F}_\omega$

One type constant `nat`.

Terms: variables, abstractions, type (constructor) abstractions, natural number constants, and function symbols:

- $\oplus : \forall\alpha.\alpha \to \alpha \to \alpha$

A set of interpretation terms

An extension of system $\mathtt{F}_\omega$

One type constant $\mathtt{nat}$.

Terms: variables, abstractions, type (constructor) abstractions, natural number constants, and function symbols:

- $\oplus : \forall\alpha.\alpha \to \alpha \to \alpha$

$$\begin{array}{rcl} n \oplus_{\mathtt{nat}} m & \rightsquigarrow & n + m \\ s \oplus_{\sigma \to \tau} t & \rightsquigarrow & \lambda x : \sigma.(s \cdot x) \oplus_\tau (t \cdot x) \\ s \oplus_{\forall\alpha.\sigma} t & \rightsquigarrow & \Lambda\alpha.(s * \alpha) \oplus_\sigma (t * \alpha) \end{array}$$

$$\Lambda\alpha.x \oplus_\alpha y$$

$$\Lambda\alpha.x \oplus_\alpha y$$

$$(\Lambda\alpha.x \oplus_\alpha y)\tau \rightsquigarrow x \oplus_\tau y$$

A set of interpretation terms

An extension of system $\mathtt{F}_\omega$

One type constant `nat`.

Terms: variables, abstractions, type (constructor) abstractions, natural number constants, and function symbols:

- $\oplus : \forall\alpha.\alpha \to \alpha \to \alpha$

$$\begin{array}{rcl} n \oplus_{\mathtt{nat}} m & \leadsto & n + m \\ s \oplus_{\sigma\to\tau} t & \leadsto & \lambda x : \sigma.(s \cdot x) \oplus_\tau (t \cdot x) \\ s \oplus_{\forall\alpha.\sigma} t & \leadsto & \Lambda\alpha.(s * \alpha) \oplus_\sigma (t * \alpha) \end{array}$$

- $\otimes : \forall\alpha.\alpha \to \alpha \to \alpha$

A set of interpretation terms

An extension of system $\mathtt{F}_\omega$

One type constant `nat`.

Terms: variables, abstractions, type (constructor) abstractions, natural number constants, and function symbols:

- $\oplus : \forall\alpha.\alpha \to \alpha \to \alpha$

$$\begin{array}{rcl} n \oplus_{\mathtt{nat}} m & \rightsquigarrow & n + m \\ s \oplus_{\sigma\to\tau} t & \rightsquigarrow & \lambda x : \sigma.(s \cdot x) \oplus_\tau (t \cdot x) \\ s \oplus_{\forall\alpha.\sigma} t & \rightsquigarrow & \Lambda\alpha.(s * \alpha) \oplus_\sigma (t * \alpha) \end{array}$$

- $\otimes : \forall\alpha.\alpha \to \alpha \to \alpha$
- $\mathtt{flatten} : \forall\alpha.\alpha \to \mathtt{nat}$

A set of interpretation terms

An extension of system $\mathtt{F}_\omega$

One type constant $\mathtt{nat}$.

Terms: variables, abstractions, type (constructor) abstractions, natural number constants, and function symbols:

- $\oplus : \forall\alpha.\alpha \to \alpha \to \alpha$

$$
\begin{array}{rcl}
n \oplus_{\mathtt{nat}} m & \rightsquigarrow & n + m \\
s \oplus_{\sigma\to\tau} t & \rightsquigarrow & \lambda x : \sigma.(s \cdot x) \oplus_\tau (t \cdot x) \\
s \oplus_{\forall\alpha.\sigma} t & \rightsquigarrow & \Lambda\alpha.(s * \alpha) \oplus_\sigma (t * \alpha)
\end{array}
$$

- $\otimes : \forall\alpha.\alpha \to \alpha \to \alpha$
- $\mathtt{flatten} : \forall\alpha.\alpha \to \mathtt{nat}$

$$
\begin{array}{rcl}
\mathtt{flatten}_{\mathtt{nat}} \cdot s & \rightsquigarrow & s \\
\mathtt{flatten}_{\sigma\to\tau} \cdot s & \rightsquigarrow & \mathtt{flatten}_\tau \cdot (s \cdot (\mathtt{lift}_\sigma \cdot 0)) \\
\mathtt{flatten}_{\forall\alpha.\sigma} \cdot s & \rightsquigarrow & \mathtt{flatten}_{\sigma[\alpha:=\mathtt{nat}]} \cdot (s * \mathtt{nat})
\end{array}
$$

A set of interpretation terms

An extension of system $\mathtt{F}_\omega$

One type constant $\mathtt{nat}$.

Terms: variables, abstractions, type (constructor) abstractions, natural number constants, and function symbols:

- $\oplus : \forall \alpha . \alpha \to \alpha \to \alpha$

$$\begin{array}{rcl} n \oplus_{\mathtt{nat}} m & \rightsquigarrow & n + m \\ s \oplus_{\sigma \to \tau} t & \rightsquigarrow & \lambda x : \sigma . (s \cdot x) \oplus_\tau (t \cdot x) \\ s \oplus_{\forall \alpha . \sigma} t & \rightsquigarrow & \Lambda \alpha . (s * \alpha) \oplus_\sigma (t * \alpha) \end{array}$$

- $\otimes : \forall \alpha . \alpha \to \alpha \to \alpha$
- $\mathtt{flatten} : \forall \alpha . \alpha \to \mathtt{nat}$

$$\begin{array}{rcl} \mathtt{flatten}_{\mathtt{nat}} \cdot s & \rightsquigarrow & s \\ \mathtt{flatten}_{\sigma \to \tau} \cdot s & \rightsquigarrow & \mathtt{flatten}_\tau \cdot (s \cdot (\mathtt{lift}_\sigma \cdot 0)) \\ \mathtt{flatten}_{\forall \alpha . \sigma} \cdot s & \rightsquigarrow & \mathtt{flatten}_{\sigma[\alpha := \mathtt{nat}]} \cdot (s * \mathtt{nat}) \end{array}$$

- $\mathtt{lift} : \forall \alpha . \mathtt{nat} \to \alpha$

A set of interpretation terms

An extension of system $\mathtt{F}_\omega$

One type constant $\mathtt{nat}$.

Terms: variables, abstractions, type (constructor) abstractions, natural number constants, and function symbols:

- $\oplus : \forall\alpha.\alpha \to \alpha \to \alpha$

$$\begin{array}{rcl} n \oplus_{\mathtt{nat}} m & \rightsquigarrow & n + m \\ s \oplus_{\sigma\to\tau} t & \rightsquigarrow & \lambda x : \sigma.(s \cdot x) \oplus_\tau (t \cdot x) \\ s \oplus_{\forall\alpha.\sigma} t & \rightsquigarrow & \Lambda\alpha.(s * \alpha) \oplus_\sigma (t * \alpha) \end{array}$$

- $\otimes : \forall\alpha.\alpha \to \alpha \to \alpha$
- $\mathtt{flatten} : \forall\alpha.\alpha \to \mathtt{nat}$

$$\begin{array}{rcl} \mathtt{flatten}_{\mathtt{nat}} \cdot s & \rightsquigarrow & s \\ \mathtt{flatten}_{\sigma\to\tau} \cdot s & \rightsquigarrow & \mathtt{flatten}_\tau \cdot (s \cdot (\mathtt{lift}_\sigma \cdot 0)) \\ \mathtt{flatten}_{\forall\alpha.\sigma} \cdot s & \rightsquigarrow & \mathtt{flatten}_{\sigma[\alpha:=\mathtt{nat}]} \cdot (s * \mathtt{nat}) \end{array}$$

- $\mathtt{lift} : \forall\alpha.\mathtt{nat} \to \alpha$

$$\begin{array}{rcl} \mathtt{lift}_{\mathtt{nat}} \cdot s & \rightsquigarrow & s \\ \mathtt{lift}_{\sigma\to\tau} \cdot s & \rightsquigarrow & \lambda x : \sigma.\mathtt{lift}_\tau \cdot s \\ \mathtt{lift}_{\forall\alpha.\sigma} \cdot s & \rightsquigarrow & \Lambda\alpha.\mathtt{lift}_\sigma \cdot s \end{array}$$

A set of interpretation terms

An extension of system $\mathtt{F}_\omega$

Theorem

The reduction relation $\leadsto$ is terminating.

A set of interpretation terms

An extension of system $\mathtt{F}_\omega$

Theorem

The reduction relation $\rightsquigarrow$ is terminating.

Corollary

All closed normal interpretation terms of type `nat` *are natural numbers.*

The well-founded order

The relation $s \succ_\sigma t$ is defined coinductively by:

$$\frac{s\downarrow > t\downarrow \text{ in } \mathbb{N}}{s \succ_{\mathtt{nat}} t}$$

$$\frac{s \cdot q \succ_\tau t \cdot q \text{ for all } q \in \mathcal{I}^f_\sigma}{s \succ_{\sigma\to\tau} t}$$

$$\frac{s * \tau \succ_{\mathrm{nf}_\beta(\sigma[\alpha:=\tau])} t * \tau \text{ for all closed } \tau \in \mathcal{T}_\kappa}{s \succ_{\forall(\alpha:\kappa).\sigma} t}$$

The well-founded order

Example derivation

$$\dfrac{\dfrac{((s \oplus \texttt{lift}_{\texttt{nat}\to\texttt{nat}}(1))u){\downarrow} > (su){\downarrow} \text{ for any } u \in \mathcal{I}^f_{\texttt{nat}}}{(s \oplus \texttt{lift}_{\texttt{nat}\to\texttt{nat}}(1))u \succ_{\texttt{nat}} su \text{ for any } u \in \mathcal{I}^f_{\texttt{nat}}}}{s \oplus \texttt{lift}_{\texttt{nat}\to\texttt{nat}}(1) \succ_{\texttt{nat}\to\texttt{nat}} s}$$

The well-founded order

Example derivation

$$\cfrac{\cfrac{((s \oplus \mathtt{lift}_{\mathtt{nat}\to\mathtt{nat}}(1))u)\!\downarrow\, > (su)\!\downarrow \text{ for any } u \in \mathcal{I}^f_{\mathtt{nat}}}{(s \oplus \mathtt{lift}_{\mathtt{nat}\to\mathtt{nat}}(1))u \succ_{\mathtt{nat}} su \text{ for any } u \in \mathcal{I}^f_{\mathtt{nat}}}}{s \oplus \mathtt{lift}_{\mathtt{nat}\to\mathtt{nat}}(1) \succ_{\mathtt{nat}\to\mathtt{nat}} s}$$

$((s \oplus \mathtt{lift}_{\mathtt{nat}\to\mathtt{nat}}(1))u)\!\downarrow\, > (su)\!\downarrow$ holds because

$$(s \oplus \mathtt{lift}_{\mathtt{nat}\to\mathtt{nat}}(1))u \rightsquigarrow^{+} su \oplus \mathtt{lift}_{\mathtt{nat}\to\mathtt{nat}}(1)u \rightsquigarrow^{+} su \oplus 1$$

The well-founded order

Infinite derivations

In any derivation of $s \succ_{\forall\alpha.\alpha} t$ there is an infinite branch.

$$
\cfrac{\cfrac{\cfrac{\vdots}{s * \forall\alpha.\alpha * \forall\alpha.\alpha \succ_{\forall\alpha.\alpha} t * \forall\alpha.\alpha * \forall\alpha.\alpha} \quad \dots}{s * \forall\alpha.\alpha \succ_{\forall\alpha.\alpha} t * \forall\alpha.\alpha} \qquad \dots}{s \succ_{\forall\alpha.\alpha} t}
$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$
\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}
$$

Girard, “Proofs and Types”, Chapter 11

How to interpret terms?

Type and function symbol mapping

First idea:

$$\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta\to\alpha\to\beta)\to\beta\to\beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f:\forall\alpha.\beta\to\alpha\to\beta.\lambda x:\beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha xh)\oplus\texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta fx\oplus\texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & fx \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}$$

Girard, "Proofs and Types", Chapter 11

$$[\![\texttt{nil}]\!][\![\sigma]\!][\![F]\!][\![a]\!]\oplus\texttt{lift}_{[\![\sigma]\!]}(1) \quad \succ^? \quad [\![a]\!]$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$
\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}
$$

Girard, “Proofs and Types”, Chapter 11

$$[\![a]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(1) \quad \succ^? \quad [\![a]\!]$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$
\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}
$$

Girard, "Proofs and Types", Chapter 11

$$
[\![a]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(1) \quad \succ \quad [\![a]\!]
$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$
\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}
$$

Girard, "Proofs and Types", Chapter 11

$$
\begin{array}{rcl}
[\![a]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(1) & \succ & [\![a]\!] \\
[\![\texttt{fold}_\sigma(F, a, \texttt{cons}_\tau(x, xs))]\!] & \succ^? & \\
\multicolumn{3}{r}{[\![\texttt{fold}_\sigma(F, @_{\tau,\sigma}(@_{\sigma,\tau\to\sigma}(\texttt{tapp}_{\lambda\alpha.\sigma\to\alpha\to\sigma,\tau}(F), a), x), xs)]\!]}
\end{array}
$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}$$

Girard, "Proofs and Types", Chapter 11

$$\begin{array}{rcl}
[\![a]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(1) & \succ & [\![a]\!] \\
[\![\texttt{cons}_\tau(x, xs)]\!][\![\sigma]\!][\![F]\!][\![a]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(1) & \succ^? & \\
\multicolumn{3}{c}{[\![\texttt{fold}_\sigma(F, @_{\tau,\sigma}(@_{\sigma,\tau\to\sigma}(\texttt{tapp}_{\lambda\alpha.\sigma\to\alpha\to\sigma,\tau}(F), a), x), xs)]\!]}
\end{array}$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}$$

Girard, “Proofs and Types”, Chapter 11

$$\begin{array}{rcl}
[\![a]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(1) & \succ & [\![a]\!] \\
[\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(1) \oplus \texttt{lift}_{[\![\sigma]\!]}(1) & \succ^{?} & \\
\multicolumn{3}{r}{[\![\texttt{fold}_\sigma(F, @_{\tau,\sigma}(@_{\sigma,\tau\to\sigma}(\texttt{tapp}_{\lambda\alpha.\sigma\to\alpha\to\sigma,\tau}(F), a), x), xs)]\!]}
\end{array}$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$
\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}
$$

Girard, "Proofs and Types", Chapter 11

$$
\begin{array}{rcl}
[\![a]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(1) & \succ & [\![a]\!] \\
[\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(2) & \succ^{?} & \\
[\![\texttt{fold}_\sigma(F, @_{\tau,\sigma}(@_{\sigma,\tau\to\sigma}(\texttt{tapp}_{\lambda\alpha.\sigma\to\alpha\to\sigma,\tau}(F), a), x), xs)]\!] & &
\end{array}
$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$
\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}
$$

Girard, "Proofs and Types", Chapter 11

$$
\begin{array}{rcl}
[\![a]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(1) & \succ & [\![a]\!] \\
[\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(2) & \succ^? & \\
& & [\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(1)
\end{array}
$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}$$

Girard, “Proofs and Types”, Chapter 11

$$\begin{array}{rcl}
[\![a]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(1) & \succ & [\![a]\!] \\
[\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(2) & \succ & \\
 & & [\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(1)
\end{array}$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$
\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta\to\alpha\to\beta)\to\beta\to\beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f:\forall\alpha.\beta\to\alpha\to\beta.\lambda x:\beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h)\oplus\texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x\oplus\texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}
$$

Girard, "Proofs and Types", Chapter 11

$$
\begin{array}{rcl}
[\![a]\!]\oplus\texttt{lift}_{[\![\sigma]\!]}(1) & \succ & [\![a]\!] \\
[\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!])\oplus\texttt{lift}_{[\![\sigma]\!]}(2) & \succ & \\
\multicolumn{3}{r}{[\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!])\oplus\texttt{lift}_{[\![\sigma]\!]}(1)} \\
[\![@_{\sigma,\tau}(\lambda x:\sigma.s.,t)]\!] & \succ^? & [\![s[x:=t]]\!]
\end{array}
$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$\begin{array}{rcll} \mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\ \mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\ \mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\ \mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\ \mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\ \mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta \end{array}$$

Girard, "Proofs and Types", Chapter 11

$$\begin{array}{rcl} [\![a]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(1) & \succ & [\![a]\!] \\ [\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(2) & \succ & \\ \multicolumn{3}{r}{[\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(1)} \\ [\![s]\!][x := [\![t]\!]] & \succ^? & [\![s[x := t]]\!] \end{array}$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}$$

Girard, "Proofs and Types", Chapter 11

$$\begin{array}{rcl}
[\![a]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(1) & \succ & [\![a]\!] \\
[\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(2) & \succ & \\
\multicolumn{3}{r}{[\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(1)} \\
[\![s]\!][x := [\![t]\!]] & \succeq & [\![s[x := t]]\!]
\end{array}$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & fx \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}$$

Girard, "Proofs and Types", Chapter 11

$$\begin{array}{rcl}
[\![a]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(1) & \succ & [\![a]\!] \\
[\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(2) & \succ & \\
\multicolumn{3}{r}{[\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(1)} \\
[\![s]\!][x := [\![t]\!]] & \succeq & [\![s[x := t]]\!] \\
[\![\texttt{tapp}_{\lambda\alpha.\sigma,\tau}(\Lambda\alpha.s)]\!] & \succ^? & [\![s[\alpha := \tau]]\!]
\end{array}$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$
\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}
$$

Girard, "Proofs and Types", Chapter 11

$$
\begin{array}{rcl}
[\![a]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(1) & \succ & [\![a]\!] \\
[\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(2) & \succ & \\
\multicolumn{3}{r}{[\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(1)} \\
[\![s]\!][x := [\![t]\!]] & \succeq & [\![s[x := t]]\!] \\
[\![s]\!][\alpha := [\![\tau]\!]] & \succ^? & [\![s[\alpha := \tau]]\!]
\end{array}
$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}$$

Girard, "Proofs and Types", Chapter 11

$$\begin{array}{rcl}
[\![a]\!] \oplus \texttt{lift}_{[\![\sigma]\!]}(1) & \succ & [\![a]\!] \\
[\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(2) & \succ & \\
\multicolumn{3}{r}{[\![xs]\!][\![\sigma]\!][\![F]\!]([\![F]\!][\![\tau]\!][\![a]\!][\![x]\!]) \oplus \texttt{lift}_{[\![\sigma]\!]}(1)} \\
[\![s]\!][x := [\![t]\!]] & \succeq & [\![s[x := t]]\!] \\
[\![s]\!][\alpha := [\![\tau]\!]] & \succeq & [\![s[\alpha := \tau]]\!]
\end{array}$$

How to interpret terms?

Type and function symbol mapping

First idea:

$$
\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}
$$

Failure: this mapping is not monotonic!

How to interpret terms?

Type and function symbol mapping

First idea:

$$
\begin{array}{rcll}
\mathcal{TM}(\mathtt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\mathtt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\mathtt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \mathtt{lift}_\beta(1) \\
\mathcal{J}(\mathtt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \mathtt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & f x \\
\mathcal{J}(\mathtt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}
$$

Failure: this mapping is not monotonic!

$[\![\mathtt{foldl}_\sigma(\lambda x.s, t, \mathtt{nil})]\!] = [\![\mathtt{fold}_\sigma(\lambda x.w, t, \mathtt{nil})]\!]$ regardless of s and w.

How to interpret terms?

Type and function symbol mapping

First idea:

$$
\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h) \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(1) \\
\mathcal{J}(@) & = & \Lambda\alpha\beta.\lambda f.\lambda x. & fx \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}
$$

Failure: this mapping is not monotonic!

$[\![\texttt{foldl}_\sigma(\lambda x.s, t, \texttt{nil})]\!] = [\![\texttt{fold}_\sigma(\lambda x.w, t, \texttt{nil})]\!]$ regardless of s and w.

So if $\texttt{foldl}_\sigma(\lambda x.s_1, t, \texttt{nil}) \longrightarrow \texttt{foldl}_\sigma(\lambda x.s_2, t, \texttt{nil}) \longrightarrow \dots$
we do not have an infinite decrease in $\succ$.

How to interpret terms?

Type and function symbol mapping

Second idea:

$$
\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h \\
& & & \quad \oplus\, \texttt{lift}_\beta(\texttt{flatten}_\beta(x) \oplus \texttt{flatten}_\alpha(h))) \\
& & & \quad \oplus\, \texttt{lift}_\beta(\texttt{flatten}_\beta(f\alpha x h) \oplus \texttt{flatten}_\alpha(h) \oplus 1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(\texttt{flatten}_{\forall\alpha.\beta\to\alpha\to\beta}(f) \oplus \texttt{flatten}_\beta(x) \oplus 1) \\
\mathcal{J}(@) & = & \Lambda\alpha.\Lambda\beta.\lambda f.\lambda x. & f x \oplus \texttt{lift}_\beta(\texttt{flatten}_\alpha(x)) \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}
$$

How to interpret terms?

Type and function symbol mapping

Second idea:

$$
\begin{array}{rcll}
\mathcal{TM}(\texttt{List}) & = & \forall\beta.(\forall\alpha.\beta \to \alpha \to \beta) \to \beta \to \beta & \\
\mathcal{J}(\texttt{nil}) & = & & \Lambda\beta.\lambda f : \forall\alpha.\beta \to \alpha \to \beta.\lambda x : \beta.x \\
\mathcal{J}(\texttt{cons}) & = & \Lambda\alpha.\lambda h.\lambda t. & \Lambda\beta.\lambda f.\lambda x.t\beta f(f\alpha x h \\
 & & & \quad \oplus\, \texttt{lift}_\beta(\texttt{flatten}_\beta(x) \oplus \texttt{flatten}_\alpha(h))) \\
 & & & \quad \oplus\, \texttt{lift}_\beta(\texttt{flatten}_\beta(f\alpha x h) \oplus \texttt{flatten}_\alpha(h) \oplus 1) \\
\mathcal{J}(\texttt{foldl}) & = & \Lambda\beta.\lambda f.\lambda x.\lambda l. & l\beta f x \oplus \texttt{lift}_\beta(\texttt{flatten}_{\forall\alpha.\beta\to\alpha\to\beta}(f) \oplus \texttt{flatten}_\beta(x) \oplus 1) \\
\mathcal{J}(@) & = & \Lambda\alpha.\Lambda\beta.\lambda f.\lambda x. & f x \oplus \texttt{lift}_\beta(\texttt{flatten}_\alpha(x)) \\
\mathcal{J}(\texttt{tapp}) & = & \Lambda\alpha.\Lambda\beta.\lambda x. & x\beta
\end{array}
$$

This works!

$$
\begin{array}{rcl}
[\![\texttt{fold}_\sigma(F, a, \texttt{nil})]\!] & \succ & [\![a]\!] \\
[\![\texttt{fold}_\sigma(F, a, \texttt{cons}_\tau(x, xs))]\!] & \succ & \\
\multicolumn{3}{c}{[\![\texttt{fold}_\sigma(F, @_{\tau,\sigma}(@_{\sigma,\tau\to\sigma}(\texttt{tapp}_{\lambda\alpha.\sigma\to\alpha\to\sigma,\tau}(F), a), x), xs)]\!]} \\
[\![@_{\sigma,\tau}(\lambda x : \sigma.s., t)]\!] & \succeq & [\![s[x := t]]\!] \\
[\![\texttt{tapp}_{\lambda\alpha.\sigma,\tau}(\Lambda\alpha.s)]\!] & \succeq & [\![s[\alpha := \tau]]\!]
\end{array}
$$

Larger applications

IPC2

The system IPC2 can be seen as a PFS with type constructors:

$$\Sigma_\kappa^T = \{ \quad \bot : *, \quad \texttt{or} : * \Rightarrow * \Rightarrow *, \quad \texttt{and} : * \Rightarrow * \Rightarrow *, \quad \exists : (* \Rightarrow *) \Rightarrow * \}$$

and function symbols:

$$\begin{array}{rcl}
@ & : & \forall\alpha\forall\beta.(\alpha \to \beta) \to \alpha \to \beta \\
\texttt{tapp} & : & \forall\alpha : * \Rightarrow *.\forall\beta.(\forall\beta[\alpha\beta]) \to \alpha\beta \\
\texttt{pair} & : & \forall\alpha\forall\beta.\alpha \to \beta \to \texttt{and}\,\alpha\,\beta \\
\texttt{case} & : & \forall\alpha\forall\beta\forall\gamma.\texttt{or}\,\alpha\,\beta \to (\alpha \to \gamma) \to (\beta \to \gamma) \to \gamma \\
\texttt{let} & : & \forall\alpha : * \Rightarrow *.\forall\beta.(\exists(\alpha)) \to (\forall\gamma.\alpha\gamma \to \beta) \to \beta \\
\texttt{ext} & : & \forall\alpha : * \Rightarrow *.\forall\beta.\alpha\beta \to \exists(\alpha) \\
\epsilon & : & \forall\alpha.\bot \to \alpha \\
\texttt{pr}^1 & : & \forall\alpha\forall\beta.\texttt{and}\,\alpha\,\beta \to \alpha \\
\texttt{pr}^2 & : & \forall\alpha\forall\beta.\texttt{and}\,\alpha\,\beta \to \beta \\
\texttt{in}^1 & : & \forall\alpha\forall\beta.\alpha \to \texttt{or}\,\alpha\,\beta \\
\texttt{in}^2 & : & \forall\alpha\forall\beta.\beta \to \texttt{or}\,\alpha\,\beta
\end{array}$$

Larger applications

IPC2

The system ICP2 has 28 reduction rules, including:

$$
\begin{array}{rcl}
@_{\sigma,\tau}(\lambda x.s, t) & \longrightarrow & s[x := t] \\
\mathtt{tapp}_{\lambda\alpha.\sigma,\tau}(\Lambda\alpha.s) & \longrightarrow & s[\alpha := \tau] \\
\mathtt{let}_{\varphi,\rho}(\mathtt{ext}_{\varphi,\tau}(s), \Lambda\alpha.\lambda x.t) & \longrightarrow & t[\alpha := \tau][x := s] \\
\mathtt{pr}^1_{\sigma,\tau}(\mathtt{pair}_{\sigma,\tau}(s,t)) & \longrightarrow & s \\
\mathtt{case}_{\sigma,\tau,\rho}(\mathtt{in}^1_{\sigma,\tau}(u), \lambda x.s, \lambda y.t) & \longrightarrow & s[x := u] \\
\mathtt{pr}^2_{\sigma,\tau}(\mathtt{pair}_{\sigma,\tau}(s,t)) & \longrightarrow & t \\
\mathtt{case}_{\sigma,\tau,\rho}(\mathtt{in}^2_{\sigma,\tau}(u), \lambda x.s, \lambda y.t) & \longrightarrow & t[x := u] \\
@_{\sigma,\tau}(\epsilon_{\sigma\to\tau}(s), t) & \longrightarrow & \epsilon_\tau(s) \\
\epsilon_\rho(\mathtt{case}_{\sigma,\tau,\bot}(u, \lambda x.s, \lambda y.t)) & \longrightarrow & \mathtt{case}_{\sigma,\tau,\rho}(u, \lambda x.\epsilon_\rho(s), \lambda y.\epsilon_\rho(t))
\end{array}
$$

$$
\begin{array}{l}
\mathtt{case}_{\rho,\pi,\xi}(\mathtt{case}_{\sigma,\tau,\mathtt{or}\,\rho\,\pi}(u, \lambda x.s, \lambda y.t), \lambda z.v, \lambda a.w) \;\longrightarrow \\
\qquad \mathtt{case}_{\sigma,\tau,\xi}(u, \lambda x.\mathtt{case}_{\rho,\pi,\xi}(s, \lambda z.v, \lambda a.w), \lambda y.\mathtt{case}_{\rho,\pi,\xi}(t, \lambda z.v, \lambda a.w))
\end{array}
$$

Larger applications

IPC2

The system ICP2 has 28 reduction rules, including:

$$
\begin{array}{rcl}
@_{\sigma,\tau}(\lambda x.s, t) & \longrightarrow & s[x := t] \\
\mathtt{tapp}_{\lambda\alpha.\sigma,\tau}(\Lambda\alpha.s) & \longrightarrow & s[\alpha := \tau] \\
\mathtt{let}_{\varphi,\rho}(\mathtt{ext}_{\varphi,\tau}(s), \Lambda\alpha.\lambda x.t) & \longrightarrow & t[\alpha := \tau][x := s] \\
\mathtt{pr}^1_{\sigma,\tau}(\mathtt{pair}_{\sigma,\tau}(s,t)) & \longrightarrow & s \\
\mathtt{case}_{\sigma,\tau,\rho}(\mathtt{in}^1_{\sigma,\tau}(u), \lambda x.s, \lambda y.t) & \longrightarrow & s[x := u] \\
\mathtt{pr}^2_{\sigma,\tau}(\mathtt{pair}_{\sigma,\tau}(s,t)) & \longrightarrow & t \\
\mathtt{case}_{\sigma,\tau,\rho}(\mathtt{in}^2_{\sigma,\tau}(u), \lambda x.s, \lambda y.t) & \longrightarrow & t[x := u] \\
@_{\sigma,\tau}(\epsilon_{\sigma\to\tau}(s), t) & \longrightarrow & \epsilon_\tau(s) \\
\epsilon_\rho(\mathtt{case}_{\sigma,\tau,\perp}(u, \lambda x.s, \lambda y.t)) & \longrightarrow & \mathtt{case}_{\sigma,\tau,\rho}(u, \lambda x.\epsilon_\rho(s), \lambda y.\epsilon_\rho(t))
\end{array}
$$

$$
\begin{array}{l}
\mathtt{case}_{\rho,\pi,\xi}(\mathtt{case}_{\sigma,\tau,\mathtt{or}\,\rho\,\pi}(u, \lambda x.s, \lambda y.t), \lambda z.v, \lambda a.w) \longrightarrow \\
\qquad \mathtt{case}_{\sigma,\tau,\xi}(u, \lambda x.\mathtt{case}_{\rho,\pi,\xi}(s, \lambda z.v, \lambda a.w), \lambda y.\mathtt{case}_{\rho,\pi,\xi}(t, \lambda z.v, \lambda a.w))
\end{array}
$$

Most of this system can be handled by our method. Problem:

$$
\mathtt{let}_{\psi,\rho}(\mathtt{let}_{\varphi,\exists\psi}(s, \Lambda\alpha.\lambda x : \varphi\alpha.t), u) \longrightarrow \mathtt{let}_{\varphi,\rho}(s, \Lambda\alpha.\lambda x : \varphi\alpha.\mathtt{let}_{\psi,\rho}(t,u))
$$

Conclusions and future work

A powerful and systematic methodology to prove termination of higher-order rewriting with full impredicative polymorphism.

Conclusions and future work

A powerful and systematic methodology to prove termination of higher-order rewriting with full impredicative polymorphism.

Future work:

- automation

Conclusions and future work

A powerful and systematic methodology to prove termination of higher-order rewriting with full impredicative polymorphism.

Future work:

- automation
- handle the remaining rules of IPC2

Conclusions and future work

A powerful and systematic methodology to prove termination of higher-order rewriting with full impredicative polymorphism.

Future work:

- automation
- handle the remaining rules of IPC2
- extend other techniques for higher-order termination: orderings, dependency pairs