Termination and Complexity in Higher-Order Term Rewriting

Part 1. Higher-order Rewriting: motivation and definitions

Cynthia Kop

ISR 2024

Download handout and slides from:
https://www.cs.ru.nl/~cynthiakop/2024_isr/

Common features from functional programming

$\texttt{map} :: (\alpha \Rightarrow \beta) \Rightarrow \text{list } \alpha \Rightarrow \text{list } \beta$
$\texttt{map } F \text{ } [] = []$
$\texttt{map } F \text{ } (h \text{ ; } t) = (F \text{ } h) \text{ ; } (\texttt{map } F \text{ } t)$

$\texttt{double\_all} :: \text{list int} \Rightarrow \text{list int}$
$\texttt{double\_all } lst = \texttt{map double } lst$

$\texttt{handle} :: \text{event} \Rightarrow \text{list (event} \Rightarrow \text{result)} \Rightarrow \text{list result}$
$\texttt{handle } e \text{ } callbacks = \texttt{map } (\lambda F.F \text{ } e) \text{ } callbacks$

Common features from functional programming

$$\texttt{init} :: (\text{int} \Rightarrow \alpha) \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{list}\ \alpha$$
$$\texttt{init}\ F\ x\ y = \text{if}\ (x \geq y)\ \text{then}\ [] \ \text{else}\ ((F\ x)\ ;\ (\texttt{init}\ F\ (x+1)\ y))$$

$$\texttt{filter} :: (\alpha \Rightarrow \text{bool}) \Rightarrow \text{list}\ \alpha \Rightarrow \text{list}\ \alpha$$
$$\texttt{filter}\ F\ [] = []$$
$$\texttt{filter}\ F\ (h\ ;\ t) = \text{if}\ (F\ h)\ \text{then}\ (h\ ;\ (\texttt{filter}\ F\ t))$$
$$\text{else}\ (\texttt{filter}\ F\ t)$$

$$\texttt{evens} :: \text{int} \Rightarrow \text{list int}$$
$$\texttt{evens}\ n = \texttt{filter}\ \texttt{is\_even}\ (\texttt{init}\ \texttt{id}\ 0\ n)$$

Common features from functional programming

$$\texttt{fold} :: (\alpha \Rightarrow \beta \Rightarrow \beta) \Rightarrow \beta \Rightarrow \text{list } \alpha \Rightarrow \beta$$
$$\texttt{fold}\ F\ x\ [] = x$$
$$\texttt{fold}\ F\ x\ (h\ ;\ t) = \texttt{fold}\ F\ (F\ h\ x)\ t$$

$$\texttt{rec} :: (\text{int} \Rightarrow \text{int} \Rightarrow \text{int}) \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{int}$$
$$\texttt{rec}\ F\ x\ n = \text{if } (n \leq 0) \text{ then } x \text{ else } (\texttt{rec}\ F\ (F\ x\ n)\ (n-1))$$

$$\texttt{exp} :: \text{int} \Rightarrow \text{int} \Rightarrow \text{int}$$
$$\texttt{exp}\ x\ y = \texttt{rec}\ ((*)\ x)\ 1\ y$$

$$\texttt{greatest} :: \text{list int} \Rightarrow \text{int}$$
$$\texttt{greatest}\ l = \texttt{fold}\ (\lambda y, z.\text{if } (z > y) \text{ then } z \text{ else } y)\ 0\ l$$

Shorthand using higher-order functions

$$\texttt{sumfun}\ F\ x = \textsf{if}\ (x \leq 0)\ \textsf{then}\ (F\ x)\ \textsf{else}\ ((F\ x) + (\texttt{sumfun}\ F\ (x-1)))$$

$$\texttt{sumfun}\ F\ x = \texttt{fold}\ (+)\ 0\ (\texttt{map}\ F\ (\texttt{init}\ \texttt{id}\ 0\ x))$$

$$\texttt{sumfun}\ F\ x = \texttt{rec}\ (\lambda y, z.y + F(z))\ (F\ 0)\ x$$

Higher-order functions appear naturally in. . .

- functional programming

Higher-order functions appear naturally in...

- functional programming
- object-oriented programming

Higher-order functions appear naturally in. . .

- functional programming
- object-oriented programming
- mathematics (e.g., quantifiers)

Higher-order functions appear naturally in...

- functional programming
- object-oriented programming
- mathematics (e.g., quantifiers)

$$\forall P :: \mathsf{nat} \Rightarrow \mathsf{bool}.(P(0) \wedge \forall x :: \mathsf{nat}.P(x) \Rightarrow P(x+1)) \Rightarrow \forall x :: \mathsf{nat}.P(x)$$

Higher-order functions appear naturally in...

- functional programming
- object-oriented programming
- mathematics (e.g., quantifiers)

$$\forall P :: \mathsf{nat} \Rightarrow \mathsf{bool}.(P(0) \wedge \forall x :: \mathsf{nat}.P(x) \Rightarrow P(x+1)) \Rightarrow \forall x :: \mathsf{nat}.P(x)$$

- theorem proving

$$\begin{array}{rcl} \mathtt{ack}\ \mathtt{0}\ n & \rightarrow & \mathtt{s}\ n \\ \mathtt{ack}\ (\mathtt{s}\ m)\ \mathtt{0} & \rightarrow & \mathtt{ack}\ m\ (\mathtt{s}\ \mathtt{0}) \\ \mathtt{ack}\ (\mathtt{s}\ m)\ (\mathtt{s}\ n) & \rightarrow & \mathtt{ack}\ m\ (\mathtt{ack}\ (\mathtt{s}\ m)\ n) \end{array}$$

Higher-order functions appear naturally in...

- functional programming
- object-oriented programming
- mathematics (e.g., quantifiers)

$$\forall P :: \mathsf{nat} \Rightarrow \mathsf{bool}.(P(0) \wedge \forall x :: \mathsf{nat}.P(x) \Rightarrow P(x+1)) \Rightarrow \forall x :: \mathsf{nat}.P(x)$$

- theorem proving

$$\begin{array}{rcl} \mathtt{ack}\ \mathtt{0}\ n & \rightarrow & \mathtt{s}\ n \\ \mathtt{ack}\ (\mathtt{s}\ m)\ \mathtt{0} & \rightarrow & \mathtt{ack}\ m\ (\mathtt{s}\ \mathtt{0}) \\ \mathtt{ack}\ (\mathtt{s}\ m)\ (\mathtt{s}\ n) & \rightarrow & \mathtt{ack}\ m\ (\mathtt{ack}\ (\mathtt{s}\ m)\ n) \end{array}$$

$$\Longrightarrow$$

$$\begin{array}{rcl} \mathtt{helper}\ F\ \mathtt{0} & \rightarrow & F\ (\mathtt{s}\ \mathtt{0}) \\ \mathtt{helper}\ F\ (\mathtt{s}\ n) & \rightarrow & F\ (\mathtt{helper}\ F\ n) \\ \mathtt{ack}\ \mathtt{0} & \rightarrow & \mathtt{s} \\ \mathtt{ack}\ (\mathtt{s}\ m) & \rightarrow & \mathtt{helper}\ (\mathtt{ack}\ m) \end{array}$$

A first idea

Question: do we need more than we already have?

A first idea

Question: do we need more than we already have?

Idea:

- application symbol @ with arity 2
- all other symbols: arity 0

A first idea

Question: do we need more than we already have?

Idea:

- application symbol @ with arity 2
- all other symbols: arity 0

Examples:

$$
\begin{array}{rcl}
@(@(\mathtt{map}, F), []) & \to & [] \\
@(@(\mathtt{map}, F), @(@(\mathtt{cons}, h), t)) & \to & @(@(\mathtt{cons}, @(F, h)), @(@(\mathtt{map}, F) \\
@(@(@(\mathtt{rec}, F), x), 0) & \to & x \\
@(@(@(\mathtt{rec}, F), x), @(\mathtt{s}\ y)) & \to & @(@(@(\mathtt{rec}, F), \\
 & & \quad @(@(F, x), @(\mathtt{s}, y))), \\
 & & \quad y)
\end{array}
$$

A first idea

Question: do we need more than we already have?

Idea:

- application symbol @ with arity 2
- all other symbols: arity 0

Examples:

$$\begin{aligned} \mathtt{map} \cdot F \cdot [] &\to [] \\ \mathtt{map} \cdot F \cdot (\mathtt{cons} \cdot h \cdot t) &\to \mathtt{cons} \cdot (F \cdot h) \cdot (\mathtt{map} \cdot F \cdot t) \\ \mathtt{rec} \cdot F \cdot x \cdot \mathtt{0} &\to x \\ \mathtt{rec} \cdot F \cdot x \cdot (\mathtt{s} \cdot y) &\to \mathtt{rec} \cdot F \cdot (F \cdot x \cdot (\mathtt{s} \cdot y)) \cdot y \end{aligned}$$

Applicative rewriting

Question: what about lambda?

$$\texttt{handle} \cdot e \cdot \mathit{callbacks} \quad \rightarrow \quad \texttt{map} \cdot \text{“}(\lambda F.F\ e)\text{”} \cdot \mathit{callbacks}$$

Applicative rewriting

Question: what about lambda?

$$\begin{aligned} \texttt{handle} \cdot e \cdot \mathit{callbacks} &\rightarrow \texttt{map} \cdot (\texttt{helper} \cdot e) \cdot \mathit{callbacks} \\ \texttt{helper} \cdot e \cdot F &\rightarrow F \cdot e \end{aligned}$$

Applicative rewriting

Question: what about lambda?

$$\begin{aligned} \texttt{handle} \cdot e \cdot \mathit{callbacks} &\rightarrow \texttt{map} \cdot (\texttt{helper} \cdot e) \cdot \mathit{callbacks} \\ \texttt{helper} \cdot e \cdot F &\rightarrow F \cdot e \end{aligned}$$

Downsides

- λ-expressions *in the input* cannot be represented (but: may not be needed!)

Applicative rewriting

Question: what about lambda?

$$\begin{aligned} \texttt{handle} \cdot e \cdot \mathit{callbacks} &\to \texttt{map} \cdot (\texttt{helper} \cdot e) \cdot \mathit{callbacks} \\ \texttt{helper} \cdot e \cdot F &\to F \cdot e \end{aligned}$$

Downsides

- λ-expressions *in the input* cannot be represented (but: may not be needed!)
- Having only one root symbol makes analysis harder

Exercise: playing with applicative systems

1.

$$\begin{aligned} \mathtt{rec} \cdot F \cdot x \cdot \mathtt{0} &\to x \\ \mathtt{rec} \cdot F \cdot x \cdot (\mathtt{s} \cdot y) &\to \mathtt{rec} \cdot F \cdot (F \cdot x \cdot (\mathtt{s} \cdot y)) \cdot y \end{aligned}$$

Use `rec` to define addition, multiplication and factorial. You are allowed to use lambda (or define helper functions), but don't write your own recursive functions.

2. Prove termination of

$$\mathtt{f} \cdot x \to \mathtt{g} \cdot x \cdot x$$

(There are no additional function symbols – so also no lambda)

Termination of `map`

Recall:

$$\begin{array}{rcl} \texttt{map} \cdot F \cdot [] & \to & [] \\ \texttt{map} \cdot F \cdot (\texttt{cons} \cdot h \cdot t) & \to & \texttt{cons} \cdot (F \cdot h) \cdot (\texttt{map} \cdot F \cdot t) \end{array}$$

Termination of `map`

Recall:

$$\begin{array}{rcl} \texttt{map} \cdot F \cdot [] & \to & [] \\ \texttt{map} \cdot F \cdot (\texttt{cons} \cdot h \cdot t) & \to & \texttt{cons} \cdot (F \cdot h) \cdot (\texttt{map} \cdot F \cdot t) \end{array}$$

Debate: is this TRS terminating?

Termination of `map`

Recall:

$$\begin{array}{rcl} \texttt{map} \cdot F \cdot [] & \to & [] \\ \texttt{map} \cdot F \cdot (\texttt{cons} \cdot h \cdot t) & \to & \texttt{cons} \cdot (F \cdot h) \cdot (\texttt{map} \cdot F \cdot t) \end{array}$$

Debate: is this TRS terminating?

Debate: what if we add the following rules?

$$\begin{array}{rcl} \texttt{g} \cdot x \cdot F & \to & F \cdot x \\ \texttt{tmp} \cdot x & \to & \texttt{map} \cdot (\texttt{g} \cdot x) \cdot x \end{array}$$

Termination of `map`

$$
\begin{array}{rcl}
\texttt{map} \cdot F \cdot [] & \to & [] \\
\texttt{map} \cdot F \cdot (\texttt{cons} \cdot h \cdot t) & \to & \texttt{cons} \cdot (F \cdot h) \cdot (\texttt{map} \cdot F \cdot t) \\
\texttt{g} \cdot x \cdot F & \to & F \cdot x \\
\texttt{tmp} \cdot x & \to & \texttt{map} \cdot (\texttt{g} \cdot x) \cdot x
\end{array}
$$

Answer: NO

Termination of map

$$
\begin{array}{rcl}
\mathtt{map} \cdot F \cdot [] & \to & [] \\
\mathtt{map} \cdot F \cdot (\mathtt{cons} \cdot h \cdot t) & \to & \mathtt{cons} \cdot (F \cdot h) \cdot (\mathtt{map} \cdot F \cdot t) \\
\mathtt{g} \cdot x \cdot F & \to & F \cdot x \\
\mathtt{tmp} \cdot x & \to & \mathtt{map} \cdot (\mathtt{g} \cdot x) \cdot x
\end{array}
$$

Answer: NO

Consider: Let $\omega := \mathtt{cons} \cdot \mathtt{tmp} \cdot []$

$$\mathtt{map} \cdot (\mathtt{g} \cdot \omega) \cdot \omega$$

Termination of `map`

$$\begin{aligned}
\texttt{map} \cdot F \cdot [] &\to [] \\
\texttt{map} \cdot F \cdot (\texttt{cons} \cdot h \cdot t) &\to \texttt{cons} \cdot (F \cdot h) \cdot (\texttt{map} \cdot F \cdot t) \\
\texttt{g} \cdot x \cdot F &\to F \cdot x \\
\texttt{tmp} \cdot x &\to \texttt{map} \cdot (\texttt{g} \cdot x) \cdot x
\end{aligned}$$

Answer: NO

Consider: Let $\omega := \texttt{cons} \cdot \texttt{tmp} \cdot []$

$$\begin{aligned}
& \texttt{map} \cdot (\texttt{g} \cdot \omega) \cdot \omega \\
= \quad & \texttt{map} \cdot (\texttt{g} \cdot \omega) \cdot (\texttt{cons} \cdot \texttt{tmp} \cdot [])
\end{aligned}$$

Termination of map

$$\begin{aligned}
\texttt{map} \cdot F \cdot [] &\to [] \\
\texttt{map} \cdot F \cdot (\texttt{cons} \cdot h \cdot t) &\to \texttt{cons} \cdot (F \cdot h) \cdot (\texttt{map} \cdot F \cdot t) \\
\texttt{g} \cdot x \cdot F &\to F \cdot x \\
\texttt{tmp} \cdot x &\to \texttt{map} \cdot (\texttt{g} \cdot x) \cdot x
\end{aligned}$$

Answer: NO

Consider: Let $\omega := \texttt{cons} \cdot \texttt{tmp} \cdot []$

$$\begin{aligned}
& \texttt{map} \cdot (\texttt{g} \cdot \omega) \cdot \omega \\
= \quad & \texttt{map} \cdot \underbrace{(\texttt{g} \cdot \omega)}_{F} \cdot (\texttt{cons} \cdot \underbrace{\texttt{tmp}}_{h} \cdot \underbrace{[]}_{t})
\end{aligned}$$

Termination of map

$$\begin{array}{rcl} \texttt{map} \cdot F \cdot [] & \to & [] \\ \texttt{map} \cdot F \cdot (\texttt{cons} \cdot h \cdot t) & \to & \texttt{cons} \cdot (F \cdot h) \cdot (\texttt{map} \cdot F \cdot t) \\ \texttt{g} \cdot x \cdot F & \to & F \cdot x \\ \texttt{tmp} \cdot x & \to & \texttt{map} \cdot (\texttt{g} \cdot x) \cdot x \end{array}$$

Answer: NO

Consider: Let $\omega := \texttt{cons} \cdot \texttt{tmp} \cdot []$

$$\begin{array}{rl} & \texttt{map} \cdot (\texttt{g} \cdot \omega) \cdot \omega \\ = & \texttt{map} \cdot \underbrace{(\texttt{g} \cdot \omega)}_{F} \cdot (\texttt{cons} \cdot \underbrace{\texttt{tmp}}_{h} \cdot \underbrace{[]}_{t}) \\ \to & \texttt{cons} \cdot (\texttt{g} \cdot \omega \cdot \texttt{tmp}) \cdot (\texttt{map} \cdot (\texttt{g} \cdot \omega) \cdot []) \end{array}$$

Termination of map

$$\begin{aligned}
\texttt{map} \cdot F \cdot [] &\to [] \\
\texttt{map} \cdot F \cdot (\texttt{cons} \cdot h \cdot t) &\to \texttt{cons} \cdot (F \cdot h) \cdot (\texttt{map} \cdot F \cdot t) \\
\texttt{g} \cdot x \cdot F &\to F \cdot x \\
\texttt{tmp} \cdot x &\to \texttt{map} \cdot (\texttt{g} \cdot x) \cdot x
\end{aligned}$$

Answer: NO

Consider: Let $\omega := \texttt{cons} \cdot \texttt{tmp} \cdot []$

$$\begin{aligned}
& \texttt{map} \cdot (\texttt{g} \cdot \omega) \cdot \omega \\
= \quad & \texttt{map} \cdot (\texttt{g} \cdot \omega) \cdot (\texttt{cons} \cdot \texttt{tmp} \cdot []) \\
\to \quad & \texttt{cons} \cdot (\texttt{g} \cdot \omega \cdot \texttt{tmp}) \cdot (\texttt{map} \cdot (\texttt{g} \cdot \omega) \cdot []) \\
= \quad & \texttt{cons} \cdot (\texttt{g} \cdot \omega \cdot \texttt{tmp}) \cdot [\cdots]
\end{aligned}$$

Termination of map

$$
\begin{array}{rcl}
\mathtt{map} \cdot F \cdot [] & \to & [] \\
\mathtt{map} \cdot F \cdot (\mathtt{cons} \cdot h \cdot t) & \to & \mathtt{cons} \cdot (F \cdot h) \cdot (\mathtt{map} \cdot F \cdot t) \\
\mathtt{g} \cdot x \cdot F & \to & F \cdot x \\
\mathtt{tmp} \cdot x & \to & \mathtt{map} \cdot (\mathtt{g} \cdot x) \cdot x
\end{array}
$$

Answer: NO

Consider: Let $\omega := \mathtt{cons} \cdot \mathtt{tmp} \cdot []$

$$
\begin{array}{rl}
 & \mathtt{map} \cdot (\mathtt{g} \cdot \omega) \cdot \omega \\
= & \mathtt{map} \cdot (\mathtt{g} \cdot \omega) \cdot (\mathtt{cons} \cdot \mathtt{tmp} \cdot []) \\
\to & \mathtt{cons} \cdot (\mathtt{g} \cdot \omega \cdot \mathtt{tmp}) \cdot (\mathtt{map} \cdot (\mathtt{g} \cdot \omega) \cdot []) \\
= & \mathtt{cons} \cdot (\mathtt{g} \cdot \omega \cdot \mathtt{tmp}) \cdot [\cdots] \\
\to & \mathtt{cons} \cdot (\mathtt{tmp} \cdot \omega) \cdot [\cdots]
\end{array}
$$

Termination of map

$$\begin{array}{rcl} \mathtt{map} \cdot F \cdot [] & \to & [] \\ \mathtt{map} \cdot F \cdot (\mathtt{cons} \cdot h \cdot t) & \to & \mathtt{cons} \cdot (F \cdot h) \cdot (\mathtt{map} \cdot F \cdot t) \\ \mathtt{g} \cdot x \cdot F & \to & F \cdot x \\ \mathtt{tmp} \cdot x & \to & \mathtt{map} \cdot (\mathtt{g} \cdot x) \cdot x \end{array}$$

Answer: NO

Consider: Let $\omega := \mathtt{cons} \cdot \mathtt{tmp} \cdot []$

$$\begin{array}{rl} & \mathtt{map} \cdot (\mathtt{g} \cdot \omega) \cdot \omega \\ = & \mathtt{map} \cdot (\mathtt{g} \cdot \omega) \cdot (\mathtt{cons} \cdot \mathtt{tmp} \cdot []) \\ \to & \mathtt{cons} \cdot (\mathtt{g} \cdot \omega \cdot \mathtt{tmp}) \cdot (\mathtt{map} \cdot (\mathtt{g} \cdot \omega) \cdot []) \\ = & \mathtt{cons} \cdot (\mathtt{g} \cdot \omega \cdot \mathtt{tmp}) \cdot [\cdots] \\ \to & \mathtt{cons} \cdot (\mathtt{tmp} \cdot \omega) \cdot [\cdots] \\ \to & \mathtt{cons} \cdot (\mathtt{map} \cdot (\mathtt{g} \cdot \omega) \cdot \omega) \cdot [\cdots] \end{array}$$

Termination of map

$$\begin{aligned} \texttt{map} \cdot F \cdot [] &\to [] \\ \texttt{map} \cdot F \cdot (\texttt{cons} \cdot h \cdot t) &\to \texttt{cons} \cdot (F \cdot h) \cdot (\texttt{map} \cdot F \cdot t) \\ \texttt{g} \cdot x \cdot F &\to F \cdot x \\ \texttt{tmp} \cdot x &\to \texttt{map} \cdot (\texttt{g} \cdot x) \cdot x \end{aligned}$$

Answer: NO

Consider: Let $\omega := \texttt{cons} \cdot \texttt{tmp} \cdot []$

$$\begin{aligned} & \underline{\texttt{map} \cdot (\texttt{g} \cdot \omega) \cdot \omega} \\ = \quad & \texttt{map} \cdot (\texttt{g} \cdot \omega) \cdot (\texttt{cons} \cdot \texttt{tmp} \cdot []) \\ \to \quad & \texttt{cons} \cdot (\texttt{g} \cdot \omega \cdot \texttt{tmp}) \cdot (\texttt{map} \cdot (\texttt{g} \cdot \omega) \cdot []) \\ = \quad & \texttt{cons} \cdot (\texttt{g} \cdot \omega \cdot \texttt{tmp}) \cdot [\cdots] \\ \to \quad & \texttt{cons} \cdot (\texttt{tmp} \cdot \omega) \cdot [\cdots] \\ \to \quad & \texttt{cons} \cdot (\underline{\texttt{map} \cdot (\texttt{g} \cdot \omega) \cdot \omega}) \cdot [\cdots] \end{aligned}$$

Problem: termination of `map`

Discussion: is there an inherent problem that makes systems like this non-terminating?

$$
\begin{aligned}
\texttt{map} \cdot F \cdot [] &\to [] \\
\texttt{map} \cdot F \cdot (\texttt{cons} \cdot h \cdot t) &\to \texttt{cons} \cdot (F \cdot h) \cdot (\texttt{map} \cdot F \cdot t) \\
\texttt{g} \cdot x \cdot F &\to F \cdot x \\
\texttt{tmp} \cdot x &\to \texttt{map} \cdot (\texttt{g} \cdot x) \cdot x
\end{aligned}
$$

Problem: termination of `map`

Discussion: is there an inherent problem that makes systems like this non-terminating?

$$\begin{aligned} \texttt{map} \cdot F \cdot [] &\to [] \\ \texttt{map} \cdot F \cdot (\texttt{cons} \cdot h \cdot t) &\to \texttt{cons} \cdot (F \cdot h) \cdot (\texttt{map} \cdot F \cdot t) \\ \texttt{g} \cdot x \cdot F &\to F \cdot x \\ \texttt{tmp} \cdot x &\to \texttt{map} \cdot (\texttt{g} \cdot x) \cdot x \end{aligned}$$

Idea: λ-calculus is non-terminating, so obviously we should not allow λ-terms!

Problem: termination of `map`

Discussion: is there an inherent problem that makes systems like this non-terminating?

$$\begin{aligned} \texttt{map} \cdot F \cdot [] &\to [] \\ \texttt{map} \cdot F \cdot (\texttt{cons} \cdot h \cdot t) &\to \texttt{cons} \cdot (F \cdot h) \cdot (\texttt{map} \cdot F \cdot t) \\ \texttt{g} \cdot x \cdot F &\to F \cdot x \\ \texttt{tmp} \cdot x &\to \texttt{map} \cdot (\texttt{g} \cdot x) \cdot x \end{aligned}$$

Idea: λ-calculus is non-terminating, so obviously we should not allow λ-terms!

Idea: the problem arises due to duplication; can we avoid that?

Problem: termination of $\mathtt{map}$

Discussion: is there an inherent problem that makes systems like this non-terminating?

$$\begin{aligned} \mathtt{map} \cdot F \cdot [] &\to [] \\ \mathtt{map} \cdot F \cdot (\mathtt{cons} \cdot h \cdot t) &\to \mathtt{cons} \cdot (F \cdot h) \cdot (\mathtt{map} \cdot F \cdot t) \\ \mathtt{g} \cdot x \cdot F &\to F \cdot x \\ \mathtt{tmp} \cdot x &\to \mathtt{map} \cdot (\mathtt{g} \cdot x) \cdot x \end{aligned}$$

Idea: λ-calculus is non-terminating, so obviously we should not allow λ-terms!

Idea: the problem arises due to duplication; can we avoid that?

Idea: the problem arises due to the construction of an untypable term $\mathtt{map} \cdot (\mathtt{g} \cdot \omega) \cdot \omega$

Types: avoiding undesirable terms

Without type restrictions:

$$\texttt{add}(\texttt{0}, \texttt{apple})$$

Types: avoiding undesirable terms

Without type restrictions:

$$\texttt{add}(\texttt{0}, \texttt{apple})$$

$$\texttt{map}(\texttt{map}(F, []), \texttt{s}(\texttt{pear}))$$

Types: avoiding undesirable terms

Without type restrictions:

$$\texttt{add}(\texttt{0}, \texttt{apple})$$

$$\texttt{map}(\texttt{map}(F, []), \texttt{s}(\texttt{pear})) \rightarrow \texttt{map}([], \texttt{s}(\texttt{pear}))$$

Problems of type-insensitive analysis

$$
\begin{aligned}
\mathtt{f}(\mathtt{0}) &\to \mathtt{0} \\
\mathtt{f}(\mathtt{s}(x)) &\to \mathtt{f}(x) \\
\mathtt{cost}(\mathtt{apple}) &\to \mathtt{s}(\mathtt{0}) \\
\mathtt{cost}(\mathtt{pear}) &\to \mathtt{s}(\mathtt{s}(\mathtt{0}))
\end{aligned}
$$

Problems of type-insensitive analysis

$$\begin{aligned} \mathtt{f}(\mathtt{0}) &\to \mathtt{0} \\ \mathtt{f}(\mathtt{s}(x)) &\to \mathtt{f}(x) \\ \mathtt{cost}(\mathtt{apple}) &\to \mathtt{s}(\mathtt{0}) \\ \mathtt{cost}(\mathtt{pear}) &\to \mathtt{s}(\mathtt{s}(\mathtt{0})) \end{aligned}$$

Question: are the following properties satisfied?

- $\mathtt{f}(t)$ reduces to $\mathtt{0}$ for all t
- every ground term reduces to a constructor normal form

Problems of type-insensitive analysis

$$\begin{aligned} \mathtt{f}(\mathtt{0}) &\to \mathtt{0} \\ \mathtt{f}(\mathtt{s}(x)) &\to \mathtt{f}(x) \\ \mathtt{cost}(\mathtt{apple}) &\to \mathtt{s}(\mathtt{0}) \\ \mathtt{cost}(\mathtt{pear}) &\to \mathtt{s}(\mathtt{s}(\mathtt{0})) \end{aligned}$$

Question: are the following properties satisfied?

- $\mathtt{f}(t)$ reduces to $\mathtt{0}$ for all t
- every ground term reduces to a constructor normal form

$$\begin{aligned} \mathtt{f}(x,\mathtt{1},\mathtt{2}) &\to \mathtt{f}(x,x,x) \\ \mathtt{chooselist}(x,y) &\to x \\ \mathtt{chooselist}(x,y) &\to y \end{aligned}$$

Typing a term rewriting system (intuition)

Idea: add types to function symbols

Typing a term rewriting system (intuition)

Idea: add types to function symbols

Example: `0` :: nat, `apple` :: fruit, `banana` :: fruit, `s` :: nat $\Rightarrow$ nat, `cost` :: fruit $\Rightarrow$ nat, `add` :: nat $\Rightarrow$ nat $\Rightarrow$ nat, [] :: list, `cons` :: nat $\Rightarrow$ list $\Rightarrow$ list

Typing a term rewriting system (intuition)

Idea: add types to function symbols

Example: `0` :: nat, `apple` :: fruit, `banana` :: fruit, `s` :: nat $\Rightarrow$ nat, `cost` :: fruit $\Rightarrow$ nat, `add` :: nat $\Rightarrow$ nat $\Rightarrow$ nat, [] :: list, `cons` :: nat $\Rightarrow$ list $\Rightarrow$ list

Requirement: terms must be well-typed!

Typing a term rewriting system (intuition)

Idea: add types to function symbols

Example: $\mathtt{0}$ :: nat, $\mathtt{apple}$ :: fruit, $\mathtt{banana}$ :: fruit, $\mathtt{s}$ :: nat $\Rightarrow$ nat, $\mathtt{cost}$ :: fruit $\Rightarrow$ nat, $\mathtt{add}$ :: nat $\Rightarrow$ nat $\Rightarrow$ nat, [] :: list, $\mathtt{cons}$ :: nat $\Rightarrow$ list $\Rightarrow$ list

Requirement: terms must be well-typed!

- Terms: $\mathtt{s}(\mathtt{add}(\mathtt{0},\mathtt{0}))$ and $\mathtt{cons}(\mathtt{0},\mathtt{cons}(\mathtt{add}(\mathtt{0},\mathtt{cost}(\mathtt{apple})),[]))$
- Not terms: $\mathtt{cons}(\mathtt{0},\mathtt{banana})$ and $\mathtt{s}([])$

Typing a term rewriting system (intuition)

Idea: add types to function symbols

Example: $\mathtt{0}$ :: nat, $\mathtt{apple}$ :: fruit, $\mathtt{banana}$ :: fruit, $\mathtt{s}$ :: nat $\Rightarrow$ nat, $\mathtt{cost}$ :: fruit $\Rightarrow$ nat, $\mathtt{add}$ :: nat $\Rightarrow$ nat $\Rightarrow$ nat, [] :: list, $\mathtt{cons}$ :: nat $\Rightarrow$ list $\Rightarrow$ list

Requirement: terms must be well-typed!

- Terms: $\mathtt{s}(\mathtt{add}(\mathtt{0},\mathtt{0}))$ and $\mathtt{cons}(\mathtt{0},\mathtt{cons}(\mathtt{add}(\mathtt{0},\mathtt{cost}(\mathtt{apple})),[]))$
- Not terms: $\mathtt{cons}(\mathtt{0},\mathtt{banana})$ and $\mathtt{s}([])$

Variables: must be typed consistently within the same term or rule.

- Allowed: $\mathtt{add}(x,x)$
- Not allowed: $\mathtt{cons}(x,x)$

Typing a term rewriting system (intuition)

Idea: add types to function symbols

Example: $\mathtt{0}$ :: nat, $\mathtt{apple}$ :: fruit, $\mathtt{banana}$ :: fruit, $\mathtt{s}$ :: nat $\Rightarrow$ nat, $\mathtt{cost}$ :: fruit $\Rightarrow$ nat, $\mathtt{add}$ :: nat $\Rightarrow$ nat $\Rightarrow$ nat, [] :: list, $\mathtt{cons}$ :: nat $\Rightarrow$ list $\Rightarrow$ list

Requirement: terms must be well-typed!

- Terms: $\mathtt{s}(\mathtt{add}(\mathtt{0},\mathtt{0}))$ and $\mathtt{cons}(\mathtt{0},\mathtt{cons}(\mathtt{add}(\mathtt{0},\mathtt{cost}(\mathtt{apple})),[]))$
- Not terms: $\mathtt{cons}(\mathtt{0},\mathtt{banana})$ and $\mathtt{s}([])$

Variables: must be typed consistently within the same term or rule.

- Allowed: $\mathtt{add}(x,x)$
- Not allowed: $\mathtt{cons}(x,x)$

Reduction: unchanged!

Simple types

Fix: a set of base types

Simple types

Fix: a set of base types

For example: nat, int, bool, fruit, list

Simple types

Fix: a set of base types

For example: nat, int, bool, fruit, list

Types are:

- all base types
- if σ and τ are types, then $\sigma \Rightarrow \tau$

Simple types

Fix: a set of base types

For example: nat, int, bool, fruit, list

Types are:

- all base types
- if σ and τ are types, then $\sigma \Rightarrow \tau$

Notation: the type arrow is right-associative

Simple types

Fix: a set of base types

For example: nat, int, bool, fruit, list

Types are:

- all base types
- if σ and τ are types, then $\sigma \Rightarrow \tau$

Notation: the type arrow is right-associative

So $\sigma \Rightarrow \tau \Rightarrow \rho$ is just $\sigma \Rightarrow (\tau \Rightarrow \rho)$

Simple types

Fix: a set of base types

For example: nat, int, bool, fruit, list

Types are:

- all base types
- if σ and τ are types, then $\sigma \Rightarrow \tau$

Notation: the type arrow is right-associative

So $\sigma \Rightarrow \tau \Rightarrow \rho$ is just $\sigma \Rightarrow (\tau \Rightarrow \rho)$

And: all types can be written as $\sigma_1 \Rightarrow \ldots \Rightarrow \sigma_m \Rightarrow \iota$

HTRSs

Signature: $\mathcal{F}$ of pairs $\mathtt{f} :: \sigma$ with $\mathtt{f}$ a function symbol and σ a simple type

HTRSs

Signature: $\mathcal{F}$ of pairs $\mathtt{f} :: \sigma$ with $\mathtt{f}$ a function symbol and σ a simple type

Example: $\mathcal{F} = \{\mathtt{0} :: \mathsf{nat}, \mathtt{s} :: \mathsf{nat} \Rightarrow \mathsf{nat}, \mathtt{add} :: \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat}\}$

HTRSs

Signature: $\mathcal{F}$ of pairs $\mathtt{f} :: \sigma$ with $\mathtt{f}$ a function symbol and σ a simple type

Example: $\mathcal{F} = \{\mathtt{0} :: \mathsf{nat}, \mathtt{s} :: \mathsf{nat} \Rightarrow \mathsf{nat}, \mathtt{add} :: \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat}\}$

Variables: we assume given a set $\mathcal{V}$ of variables

HTRSs

Signature: $\mathcal{F}$ of pairs $\mathtt{f} :: \sigma$ with $\mathtt{f}$ a function symbol and σ a simple type

Example: $\mathcal{F} = \{\mathtt{0} :: \text{nat}, \mathtt{s} :: \text{nat} \Rightarrow \text{nat}, \mathtt{add} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}\}$

Variables: we assume given a set $\mathcal{V}$ of variables

Terms:

- If $(\mathtt{f} :: \sigma) \in \mathcal{F}$ then $\mathtt{f} :: \sigma$
- If $x \in \mathcal{V}$ and σ a simple type then $x_\sigma :: \sigma$
- If $s :: \sigma \Rightarrow \tau$ and $t :: \sigma$ then $s \cdot t :: \tau$
- if $s :: \tau$, $x \in \mathcal{V}$ and σ a simple type, then $\lambda x_\sigma.s :: \sigma \Rightarrow \tau$

HTRSs

Signature: $\mathcal{F}$ of pairs $\mathtt{f} :: \sigma$ with $\mathtt{f}$ a function symbol and σ a simple type

Example: $\mathcal{F} = \{\mathtt{0} :: \text{nat}, \mathtt{s} :: \text{nat} \Rightarrow \text{nat}, \mathtt{add} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}\}$

Variables: we assume given a set $\mathcal{V}$ of variables

Terms:

- If $(\mathtt{f} :: \sigma) \in \mathcal{F}$ then $\mathtt{f} :: \sigma$
- If $x \in \mathcal{V}$ and σ a simple type then $x_\sigma :: \sigma$
- If $s :: \sigma \Rightarrow \tau$ and $t :: \sigma$ then $s \cdot t :: \tau$
- if $s :: \tau$, $x \in \mathcal{V}$ and σ a simple type, then $\lambda x_\sigma.s :: \sigma \Rightarrow \tau$

$\Longrightarrow$ the application operator is not a function symbol!

HTRSs

Signature: $\mathcal{F}$ of pairs $\mathtt{f} :: \sigma$ with $\mathtt{f}$ a function symbol and σ a simple type

Example: $\mathcal{F} = \{\mathtt{0} :: \text{nat}, \mathtt{s} :: \text{nat} \Rightarrow \text{nat}, \mathtt{add} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}\}$

Variables: we assume given a set $\mathcal{V}$ of variables

Terms:

- If $(\mathtt{f} :: \sigma) \in \mathcal{F}$ then $\mathtt{f} :: \sigma$
- If $x \in \mathcal{V}$ and σ a simple type then $x_\sigma :: \sigma$
- If $s :: \sigma \Rightarrow \tau$ and $t :: \sigma$ then $s \cdot t :: \tau$
- if $s :: \tau$, $x \in \mathcal{V}$ and σ a simple type, then $\lambda x_\sigma.s :: \sigma \Rightarrow \tau$

$\Longrightarrow$ the application operator is not a function symbol!

α-renaming: term equality is modulo renaming of bound variables

HTRSs (continued)

Rules: pairs of terms $\ell \to r$ with the same type

HTRSs (continued)

Rules: pairs of terms $\ell \to r$ with the same type

Reduction:

- $C[\ell\gamma] \to_{\mathcal{R}} C[r\gamma]$ if $\ell \to r$ is a rule

HTRSs (continued)

Rules: pairs of terms $\ell \to r$ with the same type

Reduction:

- $C[\ell\gamma] \to_{\mathcal{R}} C[r\gamma]$ if $\ell \to r$ is a rule
- $C[\ (\lambda x_\sigma.s) \cdot t\ ] \to_{\mathcal{R}} C[\ s[x := t]\ ]$ (β-reduction)

HTRSs (continued)

Rules: pairs of terms $\ell \to r$ with the same type

Reduction:

- $C[\ell\gamma] \to_{\mathcal{R}} C[r\gamma]$ if $\ell \to r$ is a rule
- $C[\ (\lambda x_\sigma.s) \cdot t\ ] \to_{\mathcal{R}} C[\ s[x := t]\ ]$ (β-reduction)

Notation:

- omit variable types when clear from context
- use uncurried notation $\mathtt{f}(s_1, \ldots, s_n)$ for $\mathtt{f} \cdot s_1 \cdots s_n$

Example HTRS

$$
\begin{aligned}
\texttt{map}(F, []) &\to [] \\
\texttt{map}(F, \texttt{cons}(x, y)) &\to \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\
\texttt{double}(x) &\to \texttt{map}(\lambda y.\texttt{add}(y, x))
\end{aligned}
$$

Example HTRS

$$
\begin{array}{rcl}
\texttt{map}(F, []) & \to & [] \\
\texttt{map}(F, \texttt{cons}(x, y)) & \to & \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\
\texttt{double}(x) & \to & \texttt{map}(\lambda y.\texttt{add}(y, x))
\end{array}
$$

Signature:

$$
\begin{array}{ll}
\texttt{add} & :: \\
\texttt{double} & :: \\
[] & :: \\
\texttt{cons} & :: \\
\texttt{map} & ::
\end{array}
$$

Example HTRS

$$
\begin{array}{rcl}
\texttt{map}(F, []) & \to & [] \\
\texttt{map}(F, \texttt{cons}(x, y)) & \to & \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\
\texttt{double}(x) & \to & \texttt{map}(\lambda y.\texttt{add}(y, x))
\end{array}
$$

Signature:

$$
\begin{array}{ll}
\texttt{add} & :: \quad \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat} \\
\texttt{double} & :: \\
[] & :: \\
\texttt{cons} & :: \\
\texttt{map} & ::
\end{array}
$$

Example HTRS

$$
\begin{array}{rcl}
\texttt{map}(F, []) & \to & [] \\
\texttt{map}(F, \texttt{cons}(x, y)) & \to & \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\
\texttt{double}(x) & \to & \texttt{map}(\lambda y.\texttt{add}(y, x))
\end{array}
$$

Signature:

$$
\begin{array}{lcl}
\texttt{add} & :: & \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat} \\
\texttt{double} & :: & \mathsf{nat} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \\
[] & :: & \\
\texttt{cons} & :: & \\
\texttt{map} & :: &
\end{array}
$$

Example HTRS

$$
\begin{array}{rcl}
\texttt{map}(F, []) & \to & [] \\
\texttt{map}(F, \texttt{cons}(x, y)) & \to & \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\
\texttt{double}(x) & \to & \texttt{map}(\lambda y.\texttt{add}(y, x))
\end{array}
$$

Signature:

$$
\begin{array}{lcl}
\texttt{add} & :: & \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat} \\
\texttt{double} & :: & \mathsf{nat} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \\
[] & :: & \mathsf{list} \\
\texttt{cons} & :: & \\
\texttt{map} & :: &
\end{array}
$$

Example HTRS

$$
\begin{array}{rcl}
\texttt{map}(F, []) & \to & [] \\
\texttt{map}(F, \texttt{cons}(x, y)) & \to & \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\
\texttt{double}(x) & \to & \texttt{map}(\lambda y.\texttt{add}(y, x))
\end{array}
$$

Signature:

$$
\begin{array}{lcl}
\texttt{add} & :: & \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat} \\
\texttt{double} & :: & \mathsf{nat} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \\
[] & :: & \mathsf{list} \\
\texttt{cons} & :: & \mathsf{nat} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \\
\texttt{map} & :: &
\end{array}
$$

Example HTRS

$$
\begin{array}{rcl}
\texttt{map}(F, []) & \to & [] \\
\texttt{map}(F, \texttt{cons}(x, y)) & \to & \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\
\texttt{double}(x) & \to & \texttt{map}(\lambda y.\texttt{add}(y, x))
\end{array}
$$

Signature:

$$
\begin{array}{lcl}
\texttt{add} & :: & \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat} \\
\texttt{double} & :: & \mathsf{nat} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \\
[] & :: & \mathsf{list} \\
\texttt{cons} & :: & \mathsf{nat} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \\
\texttt{map} & :: & (\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow \mathsf{list} \Rightarrow \mathsf{list}
\end{array}
$$

Example HTRS

$$\begin{aligned} \texttt{map}(F, []) &\to [] \\ \texttt{map}(F, \texttt{cons}(x, y)) &\to \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\ \texttt{double}(x) &\to \texttt{map}(\lambda y.\texttt{add}(y, x)) \end{aligned}$$

Signature:

$$\begin{aligned} \texttt{add} &:: \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat} \\ \texttt{double} &:: \mathsf{nat} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \\ [] &:: \mathsf{list} \\ \texttt{cons} &:: \mathsf{nat} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \\ \texttt{map} &:: (\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \end{aligned}$$

Syntactic sugar for:

$$\begin{aligned} \texttt{map} \cdot F_{\mathsf{nat}\Rightarrow\mathsf{nat}} \cdot [] &\to [] \\ \texttt{map} \cdot F_{\mathsf{nat}\Rightarrow\mathsf{nat}} \cdot (\texttt{cons} \cdot x_{\mathsf{nat}} \cdot y_{\mathsf{list}}) &\to \texttt{cons} \cdot (F_{\mathsf{nat}\Rightarrow\mathsf{nat}} \cdot x_{\mathsf{nat}}) \cdot (\texttt{map} \cdot \\ \texttt{double} \cdot x_{\mathsf{nat}} &\to \texttt{map} \cdot (\lambda y_{\mathsf{nat}}.\texttt{add} \cdot y_{\mathsf{nat}} \cdot x_{\mathsf{nat}}) \end{aligned}$$

How to find suitable types?

$$
\begin{aligned}
\texttt{map}(F, []) &\rightarrow [] \\
\texttt{map}(F, \texttt{cons}(x, y)) &\rightarrow \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\
\texttt{double}(x) &\rightarrow \texttt{map}(\lambda y.\texttt{add}(y, x))
\end{aligned}
$$

How to find suitable types?

$$
\begin{aligned}
\texttt{map}(F, []) &\rightarrow [] \\
\texttt{map}(F, \texttt{cons}(x, y)) &\rightarrow \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\
\texttt{double}(x) &\rightarrow \texttt{map}(\lambda y.\texttt{add}(y, x))
\end{aligned}
$$

Signature:

$$
\begin{aligned}
\texttt{add} &:: \\
\texttt{double} &:: \\
[] &:: \\
\texttt{cons} &:: \\
\texttt{map} &::
\end{aligned}
$$

How to find suitable types?

$$\begin{aligned} \texttt{map}(F, []) &\to [] \\ \texttt{map}(F, \texttt{cons}(x, y)) &\to \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\ \texttt{double}(x) &\to \texttt{map}(\lambda y.\texttt{add}(y, x)) \end{aligned}$$

Signature:

$$\begin{aligned} \texttt{add} &:: \\ \texttt{double} &:: \\ [] &:: \text{list} \\ \texttt{cons} &:: \text{nat} \Rightarrow \text{list} \Rightarrow \text{list} \\ \texttt{map} &:: \end{aligned}$$

How to find suitable types?

$$
\begin{array}{rcl}
\texttt{map}(F, []) & \to & [] \\
\texttt{map}(F, \texttt{cons}(x, y)) & \to & \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\
\texttt{double}(x) & \to & \texttt{map}(\lambda y.\texttt{add}(y, x))
\end{array}
$$

Signature:

$$
\begin{array}{lcl}
\texttt{add} & :: & \Rightarrow \quad \Rightarrow \\
\texttt{double} & :: & \Rightarrow \\
[] & :: & \mathsf{list} \\
\texttt{cons} & :: & \mathsf{nat} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \\
\texttt{map} & :: & \Rightarrow \quad \Rightarrow
\end{array}
$$

How to find suitable types?

$$
\begin{aligned}
\texttt{map}(F, []) &\rightarrow [] \\
\texttt{map}(F, \texttt{cons}(x, y)) &\rightarrow \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\
\texttt{double}(x) &\rightarrow \texttt{map}(\lambda y.\texttt{add}(y, x))
\end{aligned}
$$

Signature:

$$
\begin{aligned}
\texttt{add} &:: \quad \Rightarrow \quad \Rightarrow \\
\texttt{double} &:: \quad \Rightarrow \\
[] &:: \text{list} \\
\texttt{cons} &:: \text{nat} \Rightarrow \text{list} \Rightarrow \text{list} \\
\texttt{map} &:: \quad \Rightarrow \quad \Rightarrow \text{list}
\end{aligned}
$$

How to find suitable types?

$$
\begin{array}{rcl}
\texttt{map}(F, []) & \rightarrow & [] \\
\texttt{map}(F, \texttt{cons}(x, y)) & \rightarrow & \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\
\texttt{double}(x) & \rightarrow & \texttt{map}(\lambda y.\texttt{add}(y, x))
\end{array}
$$

Signature:

$$
\begin{array}{lcl}
\texttt{add} & :: & \Rightarrow \quad \Rightarrow \\
\texttt{double} & :: & \Rightarrow \\
[] & :: & \text{list} \\
\texttt{cons} & :: & \text{nat} \Rightarrow \text{list} \Rightarrow \text{list} \\
\texttt{map} & :: & \Rightarrow \text{list} \Rightarrow \text{list}
\end{array}
$$

How to find suitable types?

$$\begin{aligned} \texttt{map}(F, []) &\rightarrow [] \\ \texttt{map}(F, \texttt{cons}(x, y)) &\rightarrow \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\ \texttt{double}(x) &\rightarrow \texttt{map}(\lambda y.\texttt{add}(y, x)) \end{aligned}$$

Signature:

$$\begin{aligned} \texttt{add} &:: \quad \Rightarrow \quad \Rightarrow \\ \texttt{double} &:: \quad \Rightarrow \text{list} \Rightarrow \text{list} \\ [] &:: \text{list} \\ \texttt{cons} &:: \text{nat} \Rightarrow \text{list} \Rightarrow \text{list} \\ \texttt{map} &:: \quad \Rightarrow \text{list} \Rightarrow \text{list} \end{aligned}$$

How to find suitable types?

$$\begin{aligned} \texttt{map}(F, []) &\to [] \\ \texttt{map}(F, \texttt{cons}(x_{\text{nat}}, y)) &\to \texttt{cons}(F \cdot x_{\text{nat}}, \texttt{map}(F, y)) \\ \texttt{double}(x) &\to \texttt{map}(\lambda y.\texttt{add}(y, x)) \end{aligned}$$

Signature:

$$\begin{aligned} \texttt{add} &:: \quad \Rightarrow \quad \Rightarrow \\ \texttt{double} &:: \quad \Rightarrow \text{list} \Rightarrow \text{list} \\ [] &:: \text{list} \\ \texttt{cons} &:: \text{nat} \Rightarrow \text{list} \Rightarrow \text{list} \\ \texttt{map} &:: (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{list} \Rightarrow \text{list} \end{aligned}$$

How to find suitable types?

$$\begin{aligned} \texttt{map}(F, []) &\to [] \\ \texttt{map}(F, \texttt{cons}(x, y)) &\to \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\ \texttt{double}(x) &\to \texttt{map}(\lambda y_{\text{nat}}.\texttt{add}(y_{\text{nat}}, x)) \end{aligned}$$

Signature:

$$\begin{aligned} \texttt{add} &:: \text{nat} \Rightarrow \quad \Rightarrow \text{nat} \\ \texttt{double} &:: \quad \Rightarrow \text{list} \Rightarrow \text{list} \\ [] &:: \text{list} \\ \texttt{cons} &:: \text{nat} \Rightarrow \text{list} \Rightarrow \text{list} \\ \texttt{map} &:: (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{list} \Rightarrow \text{list} \end{aligned}$$

How to find suitable types?

$$\begin{aligned} \texttt{map}(F, []) &\rightarrow [] \\ \texttt{map}(F, \texttt{cons}(x, y)) &\rightarrow \texttt{cons}(F \cdot x, \texttt{map}(F, y)) \\ \texttt{double}(x) &\rightarrow \texttt{map}(\lambda y.\texttt{add}(y, x)) \end{aligned}$$

Signature:

$$\begin{aligned} \texttt{add} &:: \text{nat} \Rightarrow ?A \Rightarrow \text{nat} \\ \texttt{double} &:: ?A \Rightarrow \text{list} \Rightarrow \text{list} \\ [] &:: \text{list} \\ \texttt{cons} &:: \text{nat} \Rightarrow \text{list} \Rightarrow \text{list} \\ \texttt{map} &:: (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{list} \Rightarrow \text{list} \end{aligned}$$

Group exercise

Challenge: find the type of `rec`!

$$\begin{aligned} \texttt{rec}(\texttt{0}, Y, F) &\to Y \\ \texttt{rec}(\texttt{s}(X), Y, F) &\to F \cdot X \cdot \texttt{rec}(X, Y, F) \end{aligned}$$

Signature:

$$\begin{aligned} \texttt{0} &:: \\ \texttt{s} &:: \\ \texttt{rec} &:: \end{aligned}$$

Group exercise

Challenge: find the type of `rec`!

$$\begin{aligned} \texttt{rec}(\texttt{0}, Y, F) &\to Y \\ \texttt{rec}(\texttt{s}(X), Y, F) &\to F \cdot X \cdot \texttt{rec}(X, Y, F) \end{aligned}$$

Signature:

$$\begin{aligned} \texttt{0} &:: \text{nat} \\ \texttt{s} &:: \\ \texttt{rec} &:: \end{aligned}$$

Group exercise

Challenge: find the type of `rec`!

$$\begin{aligned} \texttt{rec}(\texttt{0}, Y, F) &\to Y \\ \texttt{rec}(\texttt{s}(X), Y, F) &\to F \cdot X \cdot \texttt{rec}(X, Y, F) \end{aligned}$$

Signature:

$$\begin{array}{lll} \texttt{0} & :: & \mathsf{nat} \\ \texttt{s} & :: & \mathsf{nat} \Rightarrow \mathsf{nat} \\ \texttt{rec} & :: & \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat} \end{array}$$

Group exercise

Challenge: find the type of `rec`!

$$\begin{aligned} \texttt{rec}(\texttt{0}, Y, F) &\to Y \\ \texttt{rec}(\texttt{s}(X), Y, F) &\to F \cdot X \cdot \texttt{rec}(X, Y, F) \end{aligned}$$

Signature:

$$\begin{aligned} \texttt{0} &:: \text{nat} \\ \texttt{s} &:: \text{nat} \Rightarrow \text{nat} \\ \texttt{rec} &:: \text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat} \end{aligned}$$

Group exercise

Challenge: find the type of `rec`!

$$\begin{aligned} \texttt{rec}(\texttt{0}, Y, F) &\to Y \\ \texttt{rec}(\texttt{s}(X), Y, F) &\to F \cdot X \cdot \texttt{rec}(X, Y, F) \end{aligned}$$

Signature:

$$\begin{aligned} \texttt{0} &:: \text{nat} \\ \texttt{s} &:: \Rightarrow \text{nat} \\ \texttt{rec} &:: \text{nat} \Rightarrow \Rightarrow (\Rightarrow \Rightarrow) \Rightarrow \end{aligned}$$

Group exercise

Challenge: find the type of `rec`!

$$\begin{aligned} \texttt{rec}(\texttt{0}, Y, F) &\to Y \\ \texttt{rec}(\texttt{s}(X), Y, F) &\to F \cdot X \cdot \texttt{rec}(X, Y, F) \end{aligned}$$

Signature:

$$\begin{aligned} \texttt{0} &:: \text{nat} \\ \texttt{s} &:: \text{nat} \Rightarrow \text{nat} \\ \texttt{rec} &:: \text{nat} \Rightarrow \quad \Rightarrow (\text{nat} \Rightarrow \quad \Rightarrow \quad) \Rightarrow \end{aligned}$$

Group exercise

Challenge: find the type of `rec`!

$$\begin{aligned} \texttt{rec}(\texttt{0}, Y, F) &\to Y \\ \texttt{rec}(\texttt{s}(X), Y, F) &\to F \cdot X \cdot \texttt{rec}(X, Y, F) \end{aligned}$$

Signature:

$$\begin{aligned} \texttt{0} &:: \mathsf{nat} \\ \texttt{s} &:: \mathsf{nat} \Rightarrow \mathsf{nat} \\ \texttt{rec} &:: \mathsf{nat} \Rightarrow A \Rightarrow (\mathsf{nat} \Rightarrow A \Rightarrow A) \Rightarrow A \end{aligned}$$

Exercise

Assign types to all symbols in the following rules. Use $\texttt{cons}$: nat $\Rightarrow$ list $\Rightarrow$ list and $\texttt{s}$: nat $\Rightarrow$ nat.

$$
\begin{aligned}
\texttt{filter}(F, []) &\to [] \\
\texttt{filter}(F, \texttt{cons}(h, t)) &\to \texttt{test}(F \cdot h, , F, h, t) \\
\texttt{test}(\texttt{true}, F, h, t) &\to \texttt{cons}(h, \texttt{filter}(F, t)) \\
\texttt{test}(\texttt{false}, F, h, t) &\to \texttt{filter}(F, t)
\end{aligned}
$$

$$
\begin{aligned}
\texttt{I}(\texttt{0}) &\to \texttt{0} \\
\texttt{I}(\texttt{s}(x)) &\to \texttt{s}(\texttt{twice}(\texttt{I}, x)) \\
\texttt{twice}(F) &\to \lambda x.F \cdot F \cdot x
\end{aligned}
$$

For those who like a bigger challenge!

Assign types to all symbols in the following rules. Use $\mathtt{s} : \text{ord} \Rightarrow \text{ord}$, and let the output type of $\mathtt{rec}$ be A.

$$\begin{aligned}
\mathtt{rec}(\mathtt{0}, Y, F, G) &\to Y \\
\mathtt{rec}(\mathtt{s}(X), Y, F, G) &\to F \cdot X \cdot \mathtt{rec}(X, Y, F, G) \\
\mathtt{rec}(\mathtt{lim}(H), Y, F, G) &\to G \cdot H \cdot (\lambda x_{\mathsf{nat}}.\mathtt{rec}(H \cdot x_{\mathsf{nat}}, Y, F, G))
\end{aligned}$$

The limitation of simple types

Question:

The limitation of simple types

Question: what if I want a list of numbers and a list of booleans and a list of functions?

The limitation of simple types

Question: what if I want a list of numbers and a list of booleans and a list of functions?

What should be the type of `cons`?

The limitation of simple types

Question: what if I want a list of numbers and a list of booleans and a list of functions?

What should be the type of `cons`?

Solution: have separate types intlist, boollist, inttoboollist, ...

The limitation of simple types

Question: what if I want a list of numbers and a list of booleans and a list of functions?

What should be the type of `cons`?

Solution: have separate types intlist, boollist, inttoboollist, ...

$\Longrightarrow$ also make copies of `map`, `filter`, `fold`, etc. for each

The limitation of simple types

Question: what if I want a list of numbers and a list of booleans and a list of functions?

What should be the type of `cons`?

Solution: have separate types intlist, boollist, inttoboollist, ...

$\Longrightarrow$ also make copies of `map`, `filter`, `fold`, etc. for each

The limitation of simple types

Question: what if I want a list of numbers and a list of booleans and a list of functions?

What should be the type of `cons`?

Solution: have separate types intlist, boollist, inttoboollist, ...

$\Longrightarrow$ also make copies of `map`, `filter`, `fold`, etc. for each

Alternative solution: use wrappers

The limitation of simple types

Question: what if I want a list of numbers and a list of booleans and a list of functions?

What should be the type of `cons`?

Solution: have separate types intlist, boollist, inttoboollist, ...

$\Longrightarrow$ also make copies of `map`, `filter`, `fold`, etc. for each

Alternative solution: use wrappers

$$\begin{array}{rcl} \texttt{cons} & :: & A \Rightarrow \text{list} \Rightarrow \text{list} \\ \texttt{wrapint} & :: & \text{int} \Rightarrow A \\ \texttt{wrapbool} & :: & \text{bool} \Rightarrow A \end{array}$$

The limitation of simple types

Question: what if I want a list of numbers and a list of booleans and a list of functions?

What should be the type of `cons`?

Solution: have separate types intlist, boollist, inttoboollist, ...

$\Longrightarrow$ also make copies of `map`, `filter`, `fold`, etc. for each

Alternative solution: use wrappers

$$\begin{aligned} \texttt{cons} &:: A \Rightarrow \text{list} \Rightarrow \text{list} \\ \texttt{wrapint} &:: \text{int} \Rightarrow A \\ \texttt{wrapbool} &:: \text{bool} \Rightarrow A \end{aligned}$$

Shallow polymorphism

Preferred type: $\texttt{map} :: (\alpha \Rightarrow \beta) \Rightarrow \mathsf{list}(\alpha) \Rightarrow \mathsf{list}(\beta)$

Shallow polymorphism

Preferred type: $\texttt{map} :: (\alpha \Rightarrow \beta) \Rightarrow \mathsf{list}(\alpha) \Rightarrow \mathsf{list}(\beta)$

Idea: polymorphically-typed rules correspond to a set of simply-typed rules!

Shallow polymorphism

Preferred type: $\texttt{map} :: (\alpha \Rightarrow \beta) \Rightarrow \mathsf{list}(\alpha) \Rightarrow \mathsf{list}(\beta)$

Idea: polymorphically-typed rules correspond to a set of simply-typed rules!

$$\texttt{map}(F, \texttt{cons}(h, t)) \rightarrow \texttt{cons}(F \cdot h, \texttt{map}(F, t))$$

$\Longrightarrow$

$$\begin{aligned}\texttt{map}_{\sigma,\tau}(F_{\sigma\Rightarrow\tau}, \texttt{cons}_{\sigma}(h_{\sigma}, t_{\mathsf{list}(\tau)})) \rightarrow \\ \texttt{cons}_{\tau}(F_{\sigma\Rightarrow\tau} \cdot h_{\sigma}, \texttt{map}_{\sigma,\tau}(F_{\sigma\Rightarrow\tau}, t_{\mathsf{list}(\tau)}))\end{aligned}$$

Shallow polymorphism

Preferred type: $\texttt{map} :: (\alpha \Rightarrow \beta) \Rightarrow \mathsf{list}(\alpha) \Rightarrow \mathsf{list}(\beta)$

Idea: polymorphically-typed rules correspond to a set of simply-typed rules!

$$\texttt{map}(F, \texttt{cons}(h, t)) \rightarrow \texttt{cons}(F \cdot h, \texttt{map}(F, t))$$

$\Longrightarrow$

$$\begin{aligned}\texttt{map}_{\sigma,\tau}(F_{\sigma\Rightarrow\tau}, \texttt{cons}_{\sigma}(h_{\sigma}, t_{\mathsf{list}(\tau)})) \rightarrow \\ \texttt{cons}_{\tau}(F_{\sigma\Rightarrow\tau} \cdot h_{\sigma}, \texttt{map}_{\sigma,\tau}(F_{\sigma\Rightarrow\tau}, t_{\mathsf{list}(\tau)}))\end{aligned}$$

For now: let's stick to simple types

More higher-order definitions

- CS [Aczel, 1978]
- CRS [Klop, 1980]
- ERS [Khasidashvili, 1990]
- HRS [Nipkow, 1991]
- AFS [Jouannaud, Okada, 1991]
- HORS [van Oostrom, 1994]
- IDTS [Blanqui, 2000]
- STRS [Kusakari, 2001]
- STTRS [Yamada, 2001]
- Nominal rewriting [Gabbay, Pitts, 2002]
- AFSM [Kop, 2012]
- ...

Eta-expansion

Example difference: is equality modulo η?

Eta-expansion

Example difference: is equality modulo η?

$$s =_\eta \lambda x_\sigma.(s \cdot x_\sigma)$$

if $s :: \sigma \Rightarrow \tau$ and x_σ does not occur in s, and s not an abstraction

Typical usage: expand $\mathtt{f}(s_1, \ldots, s_n)$ to
$\lambda x_1 \ldots x_m.\mathtt{f}(s_1, \ldots, s_n, x_1, \ldots, x_m)$ if $\mathtt{f} :: \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_m \Rightarrow \iota$

Eta-expansion

Example difference: is equality modulo η?

$$s =_\eta \lambda x_\sigma.(s \cdot x_\sigma)$$

if $s :: \sigma \Rightarrow \tau$ and x_σ does not occur in s, and s not an abstraction

Typical usage: expand $\mathtt{f}(s_1, \ldots, s_n)$ to
$\lambda x_1 \ldots x_m.\mathtt{f}(s_1, \ldots, s_n, x_1, \ldots, x_m)$ if $\mathtt{f} :: \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_m \Rightarrow \iota$

Downside: polymorphism

Eta-expansion

Example difference: is equality modulo η?

$$s =_\eta \lambda x_\sigma.(s \cdot x_\sigma)$$

if $s :: \sigma \Rightarrow \tau$ and x_σ does not occur in s, and s not an abstraction

Typical usage: expand $\mathtt{f}(s_1, \ldots, s_n)$ to
$\lambda x_1 \ldots x_m.\mathtt{f}(s_1, \ldots, s_n, x_1, \ldots, x_m)$ if $\mathtt{f} :: \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_m \Rightarrow \iota$

Downside: polymorphism

$\mathtt{head}(\mathtt{cons}(X, Y)) \to X$ with $\mathtt{head} :: \mathsf{list}(\alpha) \Rightarrow \alpha$

Eta-expansion

Example difference: is equality modulo η?

$$s =_\eta \lambda x_\sigma.(s \cdot x_\sigma)$$

if $s :: \sigma \Rightarrow \tau$ and x_σ does not occur in s, and s not an abstraction

Typical usage: expand $\mathtt{f}(s_1, \ldots, s_n)$ to
$\lambda x_1 \ldots x_m.\mathtt{f}(s_1, \ldots, s_n, x_1, \ldots, x_m)$ if $\mathtt{f} :: \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_m \Rightarrow \iota$

Downside: polymorphism

$\mathtt{head}(\mathtt{cons}(X, Y)) \to X$ with $\mathtt{head} :: \mathsf{list}(\alpha) \Rightarrow \alpha$

$\Longrightarrow$

$$\begin{aligned} \mathtt{head}_{\mathsf{int}}(\mathtt{cons}(X_{\mathsf{int}}, Y_{\mathsf{list(int)}})) &\to X_{\mathsf{int}} \\ \mathtt{head}_{\mathsf{int}\Rightarrow\mathsf{bool}}(\mathtt{cons}(X_{\mathsf{int}\Rightarrow\mathsf{bool}}, Y_{\mathsf{list(int}\Rightarrow\mathsf{bool)}}), Z_{\mathsf{int}}) &\to X_{\mathsf{int}\Rightarrow\mathsf{bool}} \cdot Z_{\mathsf{int}} \\ &\ldots \end{aligned}$$

Final exercise

Write a short HTRS (including signature!) that, given a list of natural numbers and a function mapping numbers to Booleans, finds the number of items in the list that satisfy the requirement.