I217E: Functional Programming

Nao Hirokawa

JAIST

Term 2-1, 2022

http://www.jaist.ac.jp/~hirokawa/lectures/fp/

Schedule

10/12	introduction	11/9	interpreters
10/14	algebraic data types I	11/11	compilers
10/19	algebraic data types II	11/16	termination
10/21	applications	11/18	confluence
10/26	program reasoning	11/25	verification
10/28	data structures I	11/30	review
11/2	data structures II	12/5	exam
11/4	computational models		

Evaluation

exam (60) + reports (40)

Orthogonality

Orthogonality

Definition

t is linear if every variable in t occurs exactly once

Example

$\mathsf{eq}(x, y)$ and $\mathsf{eq}(\mathsf{s}(x), 0)$ are linear, but $\mathsf{eq}(\mathsf{s}(x), \mathsf{s}(x))$ and $x + (-x)$ are not

Definition

- TRS $\mathcal{R}$ is left-linear if ℓ is linear for all rules $\ell \to r \in \mathcal{R}$
- TRS $\mathcal{R}$ is **orthogonal** if $\mathcal{R}$ is left-linear and $\mathsf{CP}(\mathcal{R}) = \varnothing$

Theorem (Rozen, 1973)

every orthogonal TRS is confluent

Exhaustiveness of Patterns

data Nat = Z | S Nat

eq :: Nat → Nat → Bool
eq Z Z = True
eq (S x) (S y) = eq x y

is this well-defined? what is result of following term?

eq (S Z) (S (S Z))

Haskell Programs are Orthogonal TRSs

data Nat = Z | S Nat

eq :: Nat → Nat → Bool
eq Z Z = True
eq (S x) (S y) = eq x y
eq x y = False

Exercise

instantiate eq x y = False to fulfill orthogonality

Note

- every Haskell program is virtually orthogonal
- hence confluence is guaranteed

Type System

Type Inference (Type Reconstruction Problem)

Question

what is type of f in next Haskell program?

```
data List a = Nil | Cons a (List a)
f (Cons x y) z = Cons x (f y z)
f Nil        z = z
```

Note

corresponding system is applicative TRS $\mathcal{R}$ over type environment Γ

$$\mathcal{R} = \left\{ \begin{array}{c} \mathsf{f}\ (\mathsf{c}\ x\ y)\ z \to \mathsf{c}\ x\ (\mathsf{f}\ y\ z) \\ \mathsf{f}\ \mathsf{nil}\ z \to z \end{array} \right\} \qquad \Gamma = \left\{ \begin{array}{c} \mathsf{c} : a \to \mathsf{List}\ a \to \mathsf{List}\ a \\ \mathsf{nil} : \mathsf{List}\ a\} \end{array} \right\}$$

Typing

Definition

- **polymorphic type** τ is term of form:

$$\tau ::= \underbrace{a}_{\text{type variable}} \mid \underbrace{c}_{\text{type constructor}} \mid \underbrace{\tau\ \tau}_{\text{type application}} \mid \underbrace{\tau \to \tau}_{\text{function type}}$$

- **type environment** is partial function from symbols to types

Example

$$\left\{\begin{array}{l} x : \mathsf{Nat} \\ 0 : \mathsf{Nat} \\ \mathsf{f} : a \to (a \to \mathsf{Bool}) \end{array}\right\} \text{ is type environment}$$

Definition (type judgement)

given type environment Γ

$$\frac{\Gamma(x) = \tau}{x : \tau\sigma} \qquad \frac{t : \tau_1 \to \tau_2 \quad u : \tau_1}{t\ u : \tau_2}$$

where σ is type version of substitution

Example

$$\Gamma = \left\{\begin{array}{l} x : \mathsf{Nat} \\ 0 : \mathsf{Nat} \\ \mathsf{f} : a \to a \to \mathsf{Bool} \end{array}\right\} \quad \dfrac{\dfrac{\dfrac{\Gamma(\mathsf{f}) = a \to a \to \mathsf{Bool}}{\Gamma \vdash \mathsf{f} : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Bool}} \quad \dfrac{\Gamma(x) = \mathsf{Nat}}{\Gamma \vdash x : \mathsf{Nat}}}{\Gamma \vdash \mathsf{f}\ x : \mathsf{Nat} \to \mathsf{Bool}} \quad \dfrac{\Gamma(0) = \mathsf{Nat}}{\Gamma \vdash 0 : \mathsf{Nat}}}{\Gamma \vdash \mathsf{f}\ x\ 0 : \mathsf{Bool}}$$

Definition

- term t is typable under Γ if $\Gamma \vdash t : \tau$ for some τ
- rule $\ell \to r$ is typable under Γ if $\Gamma \vdash \ell : \tau$ and $\Gamma \vdash r : \tau$ for some type τ
- applicative TRS is typable under Γ if all rules are typable under Γ

Example

- $\mathsf{g}\ \mathsf{g}$ is typeable under $\{\mathsf{g} : a \to a\}$ but not under $\{\mathsf{g} : \mathsf{List}\ a \to \mathsf{List}\ a\}$
- $\{\mathsf{map}\ f\ (x : xs) \to f\ x : \mathsf{map}\ f\ xs\}$ is typable under

$$\{\mathsf{map} : (a \to b) \to \mathsf{List}\ a \to \mathsf{List}\ b, \ldots\}$$

Fact (type preservation)

for every applicative TRS $\mathcal{R}$ typable under Γ

$$\Gamma \vdash s : \tau \ \ \&\ \ s \to_{\mathcal{R}} t \implies \Gamma \vdash t : \tau$$

Definition (constraint typing)

let Γ be type environment and a, d_x, d_f type variables

$$\frac{}{\Gamma \vdash_{\mathcal{C}} x : a}[a \approx d_x] \qquad \frac{\Gamma(f) = \tau}{\Gamma \vdash_{\mathcal{C}} f : a}[a \approx \tau'] \qquad \frac{\Gamma(f) = \bot}{\Gamma \vdash_{\mathcal{C}} f : a}[a \approx d_f]$$

$$\frac{\Gamma \vdash_{\mathcal{C}} t : b \qquad \Gamma \vdash_{\mathcal{C}} u : c}{\Gamma \vdash_{\mathcal{C}} t\ u : a}[b \approx c \to a]$$

where b, c are fresh variables, and τ' is renamed version of τ with fresh variables

Notation

- $\mathcal{C}_\Gamma(t, a)$ is set of constraints in derivation of $\Gamma \vdash_{\mathcal{C}} t : a$
- $\mathcal{C}_\Gamma(\mathcal{R}) = \{e \mid e \in \mathcal{C}_\Gamma(\ell \to r) \text{ for some } \ell \to r \in \mathcal{R}\}$

Example of (Monomorphic) Type Inference

let $\Gamma = \{\mathsf{c} : a \to \mathsf{L}\ a \to \mathsf{L}\ a,\ \mathsf{nil} : \mathsf{L}\ a\}$ (below, $\Gamma \vdash_{\mathcal{C}}$ is omitted)

$$\dfrac{\dfrac{\dfrac{\Gamma(\mathsf{f}) = \bot}{\mathsf{f} : a_2}\,[3] \quad \dfrac{\dfrac{\dfrac{\dfrac{\Gamma(\mathsf{c}) = a_6 \to \mathsf{L}\ a_6 \to \mathsf{L}\ a_6}{\mathsf{c} : a_5}\,[6] \quad \dfrac{\Gamma(x) = \bot}{x : a_7}\,[7]}{\mathsf{c}\ x : a_4}\,[5] \quad \dfrac{\Gamma(y) = \bot}{y : a_8}\,[8]}{\mathsf{c}\ x\ y : a_3}\,[4]}{\mathsf{f}\ (\mathsf{c}\ x\ y) : a_1}\,[2] \quad \dfrac{\Gamma(z) = \bot}{z : a_9}\,[9]}{\mathsf{f}\ (\mathsf{c}\ x\ y)\ z : a_0}\,[1]$$

$\mathcal{C}_\Gamma(\mathsf{f}\ (\mathsf{c}\ x\ y)\ z, a_0)$ consists of following equations:

$1:\ a_1 \approx a_9 \to a_0$	$4:\ a_4 \approx a_8 \to a_3$	$7:\ a_7 \approx d_x$
$2:\ a_2 \approx a_3 \to a_1$	$5:\ a_5 \approx a_7 \to a_4$	$8:\ a_8 \approx d_y$
$3:\ a_2 \approx d_\mathsf{f}$	$6:\ a_5 \approx a_6 \to \mathsf{L}\ a_6 \to \mathsf{L}\ a_6$	$9:\ a_9 \approx d_z$

variable-renamed applicative TRS $\mathcal{R}$ over $\Gamma = \{\mathsf{c} : a \to \mathsf{L}\ a \to \mathsf{L}\ a,\ \mathsf{nil} : \mathsf{L}\ a\}$:

$$\begin{aligned} \ell_1 &= \mathsf{f}\ (\mathsf{c}\ x\ y)\ z \to \mathsf{c}\ x\ (\mathsf{f}\ y\ z) &= r_1 \\ \ell_2 &= \mathsf{f}\ \mathsf{nil}\ w \to w &= r_2 \end{aligned}$$

$\mathcal{C}_\Gamma(\mathcal{R}) = \mathcal{C}_\Gamma(\ell_1, a_0) \cup \mathcal{C}_\Gamma(r_1, a_{10}) \cup \{a_0 \approx a_{10}\} \cup \mathcal{C}_\Gamma(\ell_2, a_{20}) \cup \mathcal{C}_\Gamma(r_2, a_{25}) \cup \{a_{20} \approx a_{25}\}$

$$\left\{\begin{array}{lll} 1:\ a_1 \approx a_9 \to a_0 & 4:\ a_4 \approx a_8 \to a_3 & 7:\ a_7 \approx d_x \\ 2:\ a_2 \approx a_3 \to a_1 & 5:\ a_5 \approx a_7 \to a_4 & 8:\ a_8 \approx d_y \\ 3:\ a_2 \approx d_\mathsf{f} & 6:\ a_5 \approx a_6 \to \mathsf{L}\ a_6 \to \mathsf{L}\ a_6 & 9:\ a_9 \approx d_z \\ 10:\ a_{11} \approx a_{15} \to a_{10} & 13:\ a_{14} \approx d_x & 16:\ a_{17} \approx d_\mathsf{f} \\ 11:\ a_{12} \approx a_{14} \to a_{11} & 14:\ a_{16} \approx a_{19} \to a_{15} & 17:\ a_{18} \approx d_y \\ 15:\ a_{12} \approx a_{13} \to \mathsf{L}\ a_{13} \to \mathsf{L}\ a_{13} & 12:\ a_{17} \approx a_{18} \to a_{16} & 18:\ a_{19} \approx d_z \\ 19:\ a_0 \approx a_{10} & & \\ 20:\ a_{21} \approx a_{24} \to a_{20} & 22:\ a_{22} \approx d_\mathsf{f} & 24:\ a_{24} \approx d_w \\ 21:\ a_{22} \approx a_{23} \to a_{21} & 23:\ a_{23} \approx d_\mathsf{nil} & 25:\ a_{25} \approx d_w \\ 26:\ a_{20} \approx a_{25} & & \end{array}\right\}$$

since $\mu(d_\mathsf{f}) = \mathsf{L}\ d_x \to \mathsf{L}\ d_x \to \mathsf{L}\ d_x$ for mgu μ of $\mathcal{C}_\Gamma(\mathcal{R})$, type of f is $\mathsf{L}\ a \to \mathsf{L}\ a \to \mathsf{L}\ a$

Review

1: Trees

Consider trees defined by the following type:

data Tree = Leaf | Node Tree Int Tree

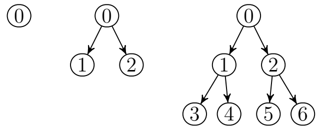
Implement the postorder traversal function `post` :: Tree → [Int].

$$\mathsf{post}\left(\begin{array}{lll} \mathsf{Node} & (\mathsf{Node} & \mathsf{Leaf}\ 1\ \mathsf{Leaf})\ 2 \\ & (\mathsf{Node} & (\mathsf{Node\ Leaf}\ 4\ \mathsf{Leaf})\ 3 \\ & & (\mathsf{Node\ Leaf}\ 5\ \mathsf{Leaf})\end{array}\right)$$
$$= [1, 4, 5, 3, 2]$$

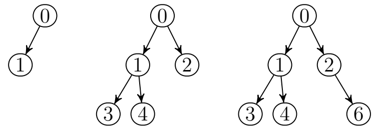
2: Trees

A binary tree t is perfect if for each node in t, its left and right subtrees have the same number of nodes.

Implement `perfect` :: Tree $\to$ Bool that checks if a binary tree is perfect.



Perfect binary trees.



Non-perfect binary trees.

3: Proof

Consider the following code:

```
len :: [a] -> Int .
len []        = 0 .
len (x : xs) = len xs + 1 .

f :: [a] -> [a] -> [a]
f []        ys = ys
f (x : xs) ys = f xs (x : ys) .

rev :: [a] -> [a]
rev xs          = f xs []
```

Show that len $($rev $xs) =$ len xs for all lists xs.

4: Inifinite List

Let H be the smallest subset of $\mathbb{N}$ that satisfies the next two conditions:

- $1 \in H$.
- If $n \in H$ then $2n, 3n, 5n \in H$.

Implement the infinite list `h` that enumerates all elements in H in ascending order:

$$\texttt{h} = 1 : 2 : 3 : 4 : 5 : 6 : 8 : 9 : 10 : 12 : 15 : 16 : \cdots$$

5: Termination

Prove or disprove termination of the TRS $\mathcal{R}$:

$$\mathsf{d}([\,]) \to [\,]$$
$$\mathsf{d}(x : xs) \to x : (x : \mathsf{d}(xs))$$

6: Confluence

Prove or disprove confluence of the TRS $\mathcal{R}$:

$$\begin{aligned} \mathsf{p}(x) + y &\to \mathsf{p}(x+y) \\ x + (y+z) &\to (x+y)+z \end{aligned}$$

7: Combinatorial Problem

Implement the function $\mathsf{f} : [a] \to \mathsf{Int} \to [[a]]$

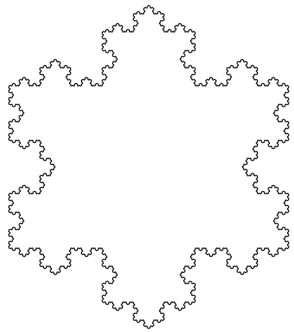
$$\begin{aligned} &\mathsf{f}\ [a_1, a_2, \ldots, a_n]\ k \\ &= [\, [a_{i_1}, a_{i_2}, \ldots, a_{i_k}] \mid 1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n \,] \end{aligned}$$

For instance, we have:

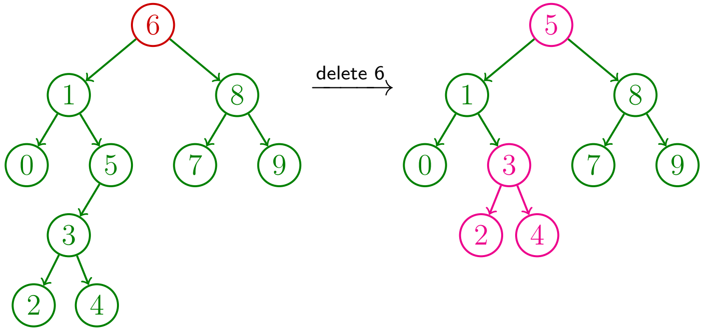
$$\begin{aligned} &\mathsf{f}\ [1,2,3,4]\ 3 = [\, [1,2,3], [1,2,4], [1,3,4], [2,3,4] \,] \\ &\mathsf{f}\ [1,2,3,4]\ 5 = [\,] \end{aligned}$$

To Conclude...

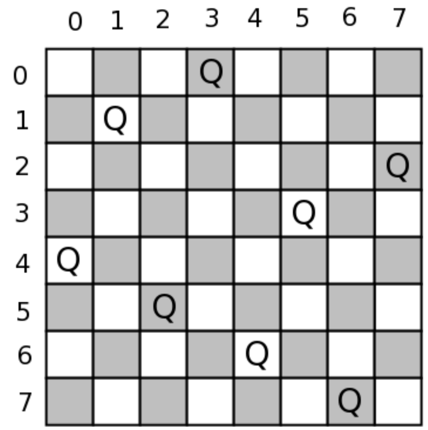
Drawing Fractals



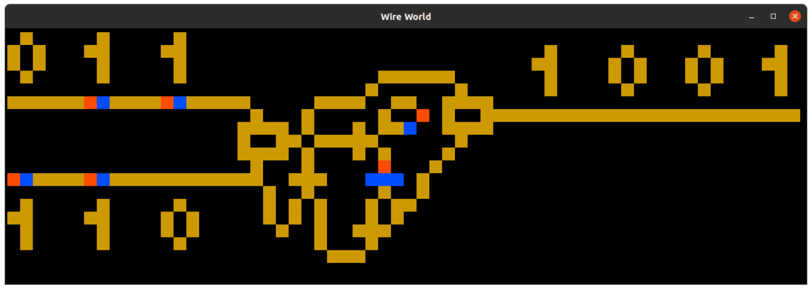
Data Structures and Algorithms



N-Queen Problem Solver



Wire World



Mini-Haskell Interpreter

program → **MiniHaskell** → computation result

sample input:

```
main = qsort [5,1,4,2,3]

qsort []        = []
qsort (x : xs)  =
  qsort (filter (<= x) xs) ++ [x] ++
  qsort (filter (>  x) xs)
                  :
```

output:

```
[1,2,3,4,5]
```

Conclusion

Goal

to become familiar with function definitions, learning

- **functional programming**
- **program reasoning**
- **computational model** (programming language theory)

Ultimate Goal

to understand that math is not your enemy

Thank You for Your Active Participation

Good Luck in Exam!