February 1, 2023

A Functional Language with Hypergraphs as First-Class Data

Waseda University
Graduate School of Fundamental Science and Engineering

Master's Defence

Jin SANO
twitter@sano_jn

Overview

Imperative programmings with heaps and pointers are tedious and error-prone. E.g., `Rust`, `C++`, ...

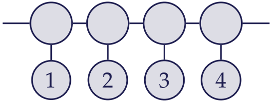
↓

We construct a new purely functional language λ_{GT}, which handles hypergraphs as immutable, first-class data with pattern matchings based on *Graph Transformation* and a new type system F_{GT}.

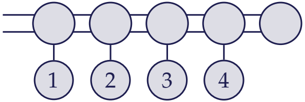
A Theoretical Foundation for
the Next Generation Programming Language.

Data Structures More Complex than Trees

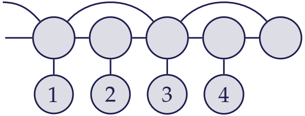
Difference List



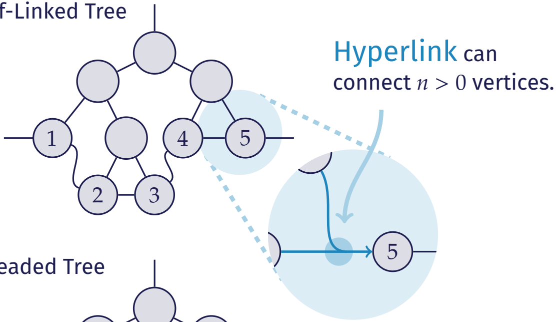
Doubly-Linked List



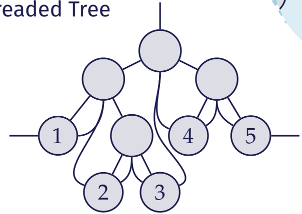
Skip List



Leaf-Linked Tree



Threaded Tree



There are several important data structures that are beyond trees:

hypergraphs (= vertices + hyperlinks).

How Programming Paradigms Handle Data Structures

Imperative

- ! Heaps and pointers
- × Impure
- × Tedious and error-prone, and easily leads to significant security issues

Purely Functional

- ✓ Algebraic Data Types (trees)
- ✓ Pure
- ✓ Type system
- × Complex data structures are difficult to handle

Graph Transformations[1]

- ✓ Graphs and pattern matchings on them
- × Impure

[1] H. Ehrig et al. *Fundamentals of Algebraic Graph Transformation.* Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2006.

Our Proposing Language λ_{GT} is ...

a functional language that extends
Algebraic Data Types (trees) to hypergraphs.

- ✓ Hypergraphs as first-class data
- ✓ Pattern matchings on hypergraphs
- ✓ Pure
- ✓ First-class functions
- ✓ Type system

Our Contributions

The main contributions of the thesis are threefold.

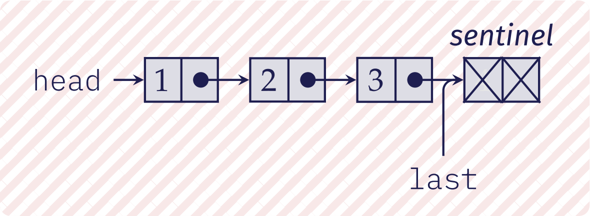
- **Chapter 2** We investigate the properties of the proposed hypergraph transformation formalism, HyperLMNtal[2].
- **Chapter 3** We propose λ_{GT}, a purely functional language that handles data structures beyond Algebraic Data Types, and implemented a PoC.
- **Chapter 4** We design a new type system F_{GT} for the λ_{GT} language.

Hereinafter, we may simply refer to hypergraphs as *graphs* and hyperlinks as *links*.

[2] J. Sano et al. “Syntax-driven and compositional syntax and semantics of Hypergraph Transformation System”. In: *Proc. 38th JSSST Annual Conference*. 2021.

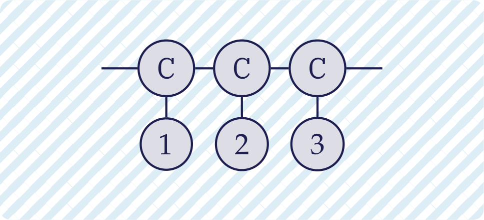
Example: Queues with Lists

Imperative



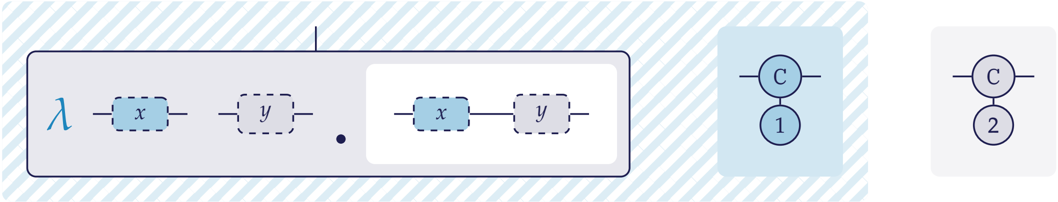
- × Requires a **dummy** sentinel node.
- × Low-level, tedious operations without a guarantee of the shape.

λ_{GT}

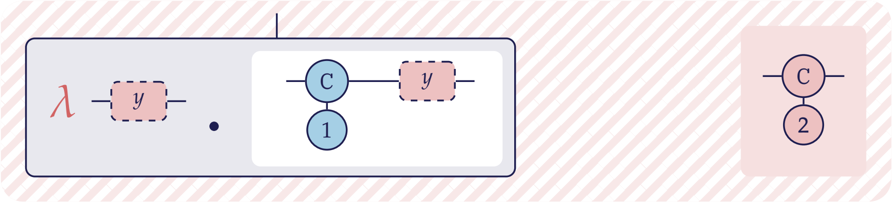


- ✓ Can define a *difference list*: a list with a link to the end.
- ✓ Declarative operations.
  → Next slides

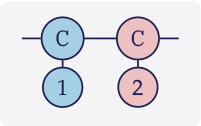
Difference Lists Concatenation in λ_{GT}



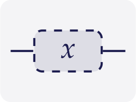
$\longrightarrow_{\texttt{val}}$

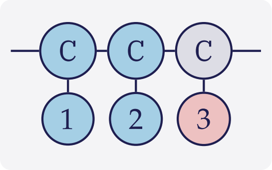


$\longrightarrow_{\texttt{val}}$

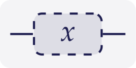


Pattern Matching Graphs

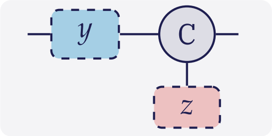
If 

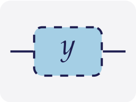
is bound with 

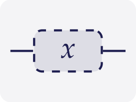
, then

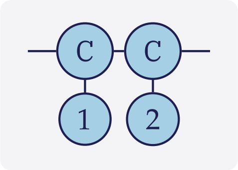
case 

of



$\rightarrow$ 

| **otherwise** $\rightarrow$ 

$\longrightarrow_{\texttt{val}}$ 

Pop the last element of a difference list

Chapter 2: Syntax of Graphs in HyperLMNtal

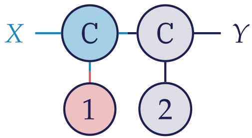
Graph

$\vec{X} = X_1, \dots, X_n$

G	$::=$	$\mathbf{0}$	Null	*empty graph*
	$\mid$	$p(\vec{X})$	Atom	*vertex with a name p and links $\vec{X}$*
	$\mid$	$X \bowtie Y$	Fusion	*link connection*
	$\mid$	(G, G)	Molecule	*composition of sub-graphs*
	$\mid$	$\nu X.G$	Hyperlink Creation	*scope of link names*

For example, a *difference list* can be represented as

$$
\begin{aligned}
&\nu Z.(\\
&\quad \nu Z_1.(\mathrm{Cons}\,(Z_1, Z, X), 1(Z_1)), \\
&\quad \nu Z_2.(\mathrm{Cons}\,(Z_2, Y, Z), 2(Z_2)) \\
&)
\end{aligned}
$$



Chapter 2: The Denoted Hypergraphs

We define the denoted hypergraphs and the mapping to them from HyperLMNtal terms, and prove that congruent terms are mapped to isomorphic graphs.

HyperLMNtal

$$\nu Z.(\; \nu Z_1.(\mathrm{Cons}\,(Z_1, Z, X), 1(Z_1)),\; \nu Z_2.(\mathrm{Cons}\,(Z_2, Y, Z), 2(Z_2))\;)$$

Equivalence is defined by Structural Congruence

$\rightarrow$

The Denoted Hypergraph

$$\langle \{v_1, v_2\}, \langle \{X\}, \{\{X\}, \{Y\}\}, \{X \mapsto \{\langle v_1, 3\rangle\}, Y \mapsto \{\langle v_2, 2\rangle\}\}\rangle, \{\{\langle v_1, 1\rangle, \langle v_2, 1\rangle\}, \{\langle v_1, 2\rangle, \langle v_3, 3\rangle\}, \{\langle v_3, 1\rangle, \langle v_4, 1\rangle\}\}, \{v_1 \mapsto \mathrm{Cons}, v_2 \mapsto 1, v_3 \mapsto \mathrm{Cons}, v_4 \mapsto 2\}\rangle$$

Equivalence is defined by Graph Isomorphism

✓ Required for implementation justification and advanced verifications.

$$
\begin{array}{rlcl}
\text{Value} & G & ::= & \mathbf{0} \mid p(\vec{X}) \mid X \bowtie Y \mid (G,G) \mid \nu X.G \\
\text{Atom Name} & p & ::= & C \mid \lambda\, x[\vec{X}].e \\
\text{Expression} & e & ::= & (\textbf{case}\ e\ \textbf{of}\ T \to e \mid \textbf{otherwise} \to e) \mid (e\ e) \mid T \\
\text{Graph Template} & T & ::= & \mathbf{0} \mid p(\vec{X}) \mid X \bowtie Y \mid (T,T) \mid \nu X.T \mid x[\vec{X}]
\end{array}
$$

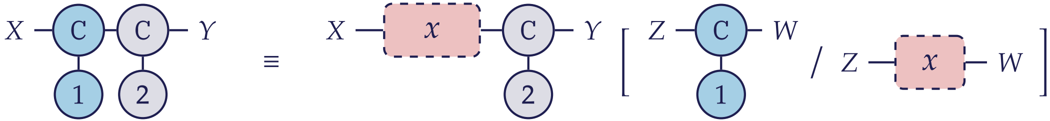
Wildcard

- Value in λ_{GT} is a graph in HyperLMNtal.
  We allow *Constructor* and λ-*abstraction* for the atoms' names.
- λ_{GT} program is an expression.
- We use Template = Value + Wildcards for pattern matching.

Chapter 3: Pattern Matching with Graph Substitution

$$\frac{G \equiv T\vec{\theta}}{(\textbf{case } G \textbf{ of } T \to e_2 \mid \textbf{otherwise} \to e_3) \longrightarrow_{\texttt{val}} e_2\vec{\theta}} \text{ Rd-Case1}$$

For example,



G T θ

Here, the graph G can be matched to the template T with a substitution θ.

$$\frac{G \equiv T\vec{\theta}}{(\textbf{case } G \textbf{ of } T \to e_2 \mid \textbf{otherwise} \to e_3) \longrightarrow_{\texttt{val}} e_2\vec{\theta}}\ \text{Rd-Case1}$$

Succeeded matching

$$\frac{\neg\exists\vec{\theta}.\left(G \equiv T\vec{\theta}\right)}{(\textbf{case } G \textbf{ of } T \to e_2 \mid \textbf{otherwise} \to e_3) \longrightarrow_{\texttt{val}} e_3}\ \text{Rd-Case2}$$

Failed matching

$$\frac{fn(G) = \{\vec{X}\}}{((\lambda\, x[\vec{X}].e)(\vec{Y})\ G) \longrightarrow_{\texttt{val}} e[G/x[\vec{X}]]}\ \text{Rd-}\beta$$

β-reduction

$$\frac{e \longrightarrow_{\texttt{val}} e'}{E[e] \longrightarrow_{\texttt{val}} E[e']}\ \text{Rd-Ctx}$$

Call-by-value

where $E ::= [\,] \mid (\textbf{case } E \textbf{ of } T \to e \mid \textbf{otherwise} \to e) \mid (E\ e) \mid (G\ E) \mid T$

Chapter 3: PoC Implementation

We build a reference implementation of the language
in only 500 lines of OCaml code.

Source `https://github.com/sano-jin/lambda-gt-alpha`
Try it at `https://sano-jin.github.io/lambda-gt-online`

We have run examples including ...

- ✓ Append two difference lists.
- ✓ Pop the last element of a difference list.
- ✓ Rotate a difference list (push an element to front from back).
- ✓ Pop all the elements from back of a difference list.
- ✓ Map a function on leaves of a leaf-linked tree.

We propose a new type system, F_{GT}, for the λ_{GT} language.

The type in F_{GT} is a *type atom* $\tau\,(\vec{X})$:

- ✓ intuitively, a type in functional languages τ + links $\vec{X}$.

The typing relation $(\Gamma, P) \vdash e : \tau\,(\vec{X})$ denotes that
e has the type $\tau\,(\vec{X})$ under the type environment $\Gamma \overset{\text{def}}{=} \{x[\vec{X}] : \tau\,(\vec{X}), \dots\}$
and a set P of production rules, a graph grammar:

- ✓ an extension of a regular tree grammar,
  on which Algebraic Data Types (trees) are based.

Chapter 4: Theorems of F_{GT}

We have proved some properties of F_{GT}.

Theorem 4.1 Soundness of F_{GT}.

Theorem 4.2 Correspondence between a typing relation in F_{GT} and a transitive closure of HyperLMNtal reduction.

- ✓ This allows us to take advantage of existing research of Graph Transformations[3].

[3] P. Fradet et al. “Structured Gamma”. In: *Science of Computer Programming* 31.2 (1998), pp. 263–289; P. Fradet et al. “Shape types”. In: *Proc. POPL’97*. ACM. 1997, pp. 27–39; H. Björklund et al. “Uniform parsing for hyperedge replacement grammars”. In: *J. Computer and System Sciences* 118 (2021), pp. 1–27.

Related Work includes ...

FUnCAL[4] is a functional language which supports graph-based database. They focus on a simple form of queries for the database, which is apart from our focus.

Structured Gamma[5] and Shape Types[6] provide a typing framework using graph grammar for graph transformation system applicable to verify imperative programs.

Separation Logic[7] is a verification framework for imperative programs with heaps and pointers, which includes Cyclic Proof[8] and SLRD[9].

[4] K. Matsuda et al. "A Functional Reformulation of UnCAL Graph-Transformations: Or, Graph Transformation as Graph Reduction". In: *Proc. POPL'97.* Paris, France: ACM, 2017, pp. 71–82.

[5] Fradet et al., "Structured Gamma".

[6] Fradet et al., "Shape types".

[7] J. Reynolds. "Separation logic: a logic for shared mutable data structures". In: *Proc. LICS 2002.* IEEE. 2002, pp. 55–74.

[8] J. Brotherston et al. "A Generic Cyclic Theorem Prover". In: *Proc. APLAS 2012.* Vol. 7705. Lecture Notes in Computer Science. Springer, 2012, pp. 350–367.

[9] R. Iosif et al. "The Tree Width of Separation Logic with Recursive Definitions". In: *Automated Deduction – CADE-24.* 2013, pp. 21–38.

Summary and Future Work

We introduced ...

- ✓ HyperLMNtal: a hypergraph transformation formalism,
- ✓ λ_{GT}: a new functional language with hypergraphs as first-class data, and
- ✓ F_{GT}: a new type system for the λ_{GT} language.

Artifact: `https://github.com/sano-jin/lambda-gt-alpha`

Future work includes ...

- ✓ more investigations on HyperLMNtal properties,
- ✓ the justification of the PoC, the construction of an efficient compiler, and
- ✓ more advanced verifications on F_{GT}.

Publications

1. J. Sano et al. "Syntax-driven and compositional syntax and semantics of Hypergraph Transformation System". In: *Proc. 38th JSSST Annual Conference.* 2021. Unrefereed. **Student encouragement award**.
2. J. Sano et al. "A functional language with graphs as first-class data". In: *Proc. 39th JSSST Annual Conference.* 2022. Unrefereed. **Presentation award**.
3. J. Sano et al. "Type Checking Data Structures More Complex Than Trees". In: *Journal of Information Processing and IPSJ Transactions on Programming* (2023). Accepted.
4. J. Sano et al. "Axiomatizing Hypergraph Isomorphism". In: *Special Interest Group on Programming and Programming Language.* 2023. Accepted.

Appendix

Related Work

HyperLMNtal

Lambda GT

FGT

Related Implementations

	Implemented with ...	LOC
λ_{GT} Reference Interpreter	OCaml	500
GP 2 Reference Interpreter[10]	Haskell	1,000
HyperLMNtal Compiler/Runtime[11]	Java/C++	12,000/47,000

[10] C. Bak et al. “A Reference Interpreter for the Graph Programming Language GP 2”. In: *Proceedings Graphs as Models, GaM@ETAPS 2015, London, UK, 11-12 April 2015.* Ed. by A. Rensink et al. Vol. 181. EPTCS. 2015, pp. 48–64.

[11] LMNtal. https://github.com/lmntal/lmntal-compiler. (Visited on 08/10/2022); SLIM. https://github.com/lmntal/slim. (Visited on 08/10/2022); M. Gocho et al. “Evolution of the LMNtal Runtime to a Parallel Model Checker”. In: *Computer Software* 28.4 (2011), 4_137–4_157.

Appendix

Related Work

HyperLMNtal

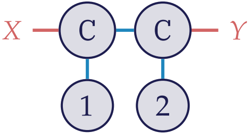
Lambda GT

FGT

Free Names and Substitutions of Hyperlinks

Links bound by ν are called *Local Links* and others are called *Free Links*

$$
\begin{aligned}
&\nu Z.(\\
&\quad \nu Z_1.(\mathrm{Cons}(Z_1, Z, X), 1(Z_1)),\\
&\quad \nu Z_2.(\mathrm{Cons}(Z_2, Y, Z), 2(Z_2))\\
&)
\end{aligned}
$$



- $fn(G)$ denotes the set of all free links in G
- $G\langle \vec{Y}/\vec{X} \rangle$ replaces all free occurrences of $\vec{X}$ with $\vec{Y}$.

The notion of locality of (link) names is NOT common in graph formalisms but in the formalisms for PLs; λ-calculus, π-calculus, ...

Structural Congruence: Axioms of Graph Equivalences

$$
\begin{array}{llcl}
\text{(E1)} & (\mathbf{0}, G) & \equiv & G \\
\text{(E2)} & (G_1, G_2) & \equiv & (G_2, G_1) \\
\text{(E3)} & (G_1, (G_2, G_3)) & \equiv & ((G_1, G_2), G_3) \\
\text{(E4)} & G_1 \equiv G_2 & \Rightarrow & (G_1, G_3) \equiv (G_2, G_3) \\
\text{(E5)} & G_1 \equiv G_2 & \Rightarrow & \nu X.G_1 \equiv \nu X.G_2 \\
\text{(E6)} & \nu X.(X \bowtie Y, G) & \equiv & \nu X.G\langle Y/X\rangle \\
 & \multicolumn{3}{l}{\text{where } X \in \mathit{fn}(G) \vee Y \in \mathit{fn}(G)} \\
\text{(E7)} & \nu X.\nu Y.X \bowtie Y & \equiv & \mathbf{0} \\
\text{(E8)} & \nu X.\mathbf{0} & \equiv & \mathbf{0} \\
\text{(E9)} & \nu X.\nu Y.G & \equiv & \nu Y.\nu X.G \\
\text{(E10)} & \nu X.(G_1, G_2) & \equiv & (\nu X.G_1, G_2) \\
 & \multicolumn{3}{l}{\text{where } X \notin \mathit{fn}(G_2)}
\end{array}
$$

For example,

$$
\begin{array}{l}
\nu Z.(\\
\quad \nu Z_1.(\mathrm{Cons}(Z_1, Z, X), 1(Z_1)), \\
\quad \nu Z_2.(\mathrm{Cons}(Z_2, Y, Z), 2(Z_2)) \\
) \\
\equiv \\
\nu Z.(\\
\quad \nu Z_1.(1(Z_1), \mathrm{Cons}(Z_1, Z, X)), \\
\quad \nu Z_2.(\mathrm{Cons}(Z_2, Y, Z), 2(Z_2)) \\
)
\end{array}
$$

by (E2), (E4) and (E5)

✓ Notice the rules are defined compositionally.

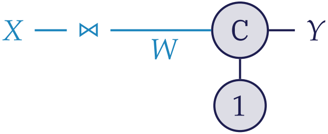
Fusion

Structural Congruence

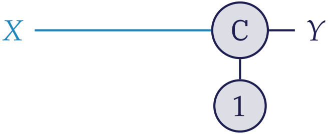
(E6) $\nu X.(X \bowtie Y, G) \quad \equiv \quad \nu X.G\langle Y/X\rangle$

where $X \in fn(G) \vee Y \in fn(G)$

$$\nu WZ.(W \bowtie X, \mathrm{Cons}(Z, Y, W), 1(Z))$$



$$\equiv \quad \nu WZ.(\mathrm{Cons}(Z, Y, X), 1(Z))$$



Appendix

Related Work

HyperLMNtal

Lambda GT

FGT

Graph Template

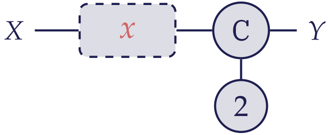
Graph Template

$$T \quad ::= \quad \mathbf{0} \ \mid \ p(\vec{X}) \ \mid \ X \bowtie Y \ \mid \ (T, T) \ \mid \ \nu X.T$$
$$\mid \quad x[\vec{X}]$$ Graph context *wildcard in pattern matching; variable*

Since the value in λ_{GT} is Graph, we use *Template* of graphs to represent data with variables.

For example,

$$\nu Z.($$
$$x[Z, X],$$
$$\nu Z_2.(\mathrm{Cons}(Z_2, Y, Z), 2(Z_2))$$
$$)$$



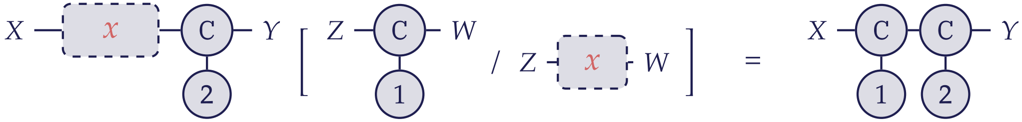
Graph Substitution

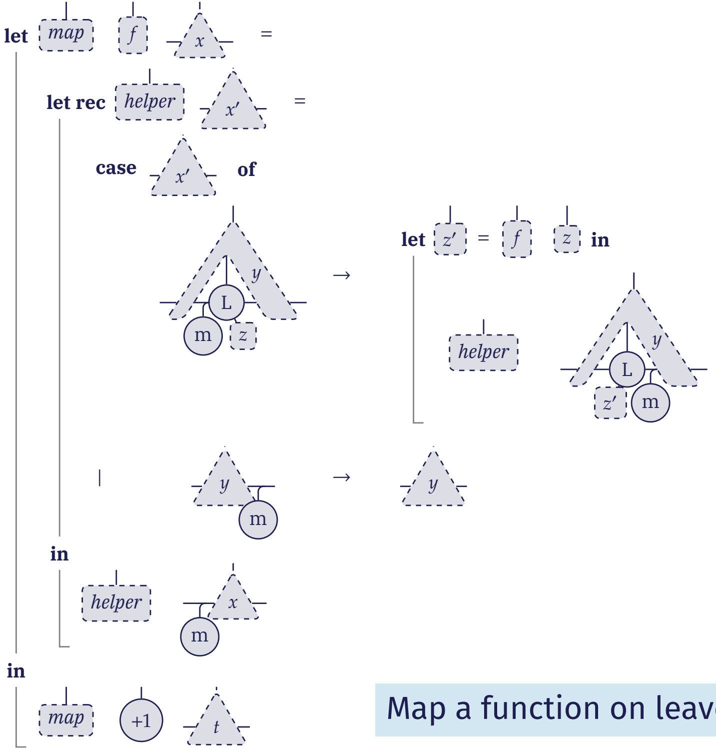
We define capture-avoiding substitution θ of a graph context $x[\vec{X}]$ with a template T in e, written $e[T/x[\vec{X}]]$.

- The definition is standard except for the graph contexts.

$$\begin{aligned}&(x[\vec{X}])[T/y[\vec{Y}]] \quad = \\ &\quad \text{if } x/|\vec{X}| = y/|\vec{Y}| \text{ then } T\langle \vec{X}/\vec{Y}\rangle \qquad \textit{reconnect free links} \\ &\quad \text{else } x[\vec{X}]\end{aligned}$$

For example,





Map a function on leaves of a leaf-linked tree.

Appendix

Chapter 4: Syntax of F_{GT}

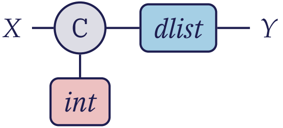
The type in F_{GT} is a type atom $\tau\,(\vec{X})$ where …

Atom Name for Types	τ	$::=$	$\alpha \mid \tau\,(\vec{X}) \rightarrow \tau\,(\vec{X})$
Production Rule	r	$::=$	$\alpha\,(\vec{X}) \longrightarrow \mathscr{T}$
Type Graph	$\mathscr{T}$	$::=$	$\tau\,(\vec{X}) \mid \mathrm{C}(\vec{X}) \mid X \bowtie Y \mid (\mathscr{T}, \mathscr{T}) \mid \nu X.\mathscr{T}$

For example, the production rules of difference lists are:

$$\mathit{dlist}\,(Y, X) \longrightarrow X \bowtie Y$$

$$\mid \quad \nu Z_1.\nu Z_2.((\mathrm{Cons}\,(Z_1, Z_2, X), \mathit{int}\,(Z_1)), \mathit{dlist}\,(Y, Z_2))$$



Rules of F_{GT} ⟨1/2⟩: Typing Rules as in Functional Languages

- These are basically the same as the type system of the other ordinary functional languages, except that **the type in $\mathbf{F_{GT}}$ is an atom**.

$$\frac{(\Gamma, P) \vdash e_1 : (\tau_1(\vec{X}) \to \tau_2(\vec{Y}))(\vec{Z}) \qquad (\Gamma, P) \vdash e_2 : \tau_1(\vec{X})}{(\Gamma, P) \vdash (e_1\ e_2) : \tau_2(\vec{Y})}\ \text{Ty-App}$$

$$\frac{((\Gamma, x[\vec{X}] : \tau_1(\vec{X})), P) \vdash e : \tau_2(\vec{Z})}{(\Gamma, P) \vdash (\lambda\, x[\vec{X}] : \tau_1(\vec{Y}).e)(\vec{W}) : (\tau_1(\vec{Y}) \to \tau_2(\vec{Z}))(\vec{W})}\ \text{Ty-Arrow}$$

$$\frac{}{(\Gamma\{x[\vec{X}] : \tau\,(\vec{Y})\}, P) \vdash x[\vec{X}] : \tau\,(\vec{Y})}\ \text{Ty-Var}$$

$$\frac{(\Gamma, P) \vdash e_1 : \tau_1(\vec{X}) \qquad ((\Gamma, \Gamma'^{*}), P) \vdash e_2 : \tau_2(\vec{Y}) \qquad (\Gamma, P) \vdash e_3 : \tau_2(\vec{Y})}{(\Gamma, P) \vdash (\textbf{case}\ e_1\ \textbf{of}\ T \to e_2 \mid \textbf{otherwise} \to e_3) : \tau_2(\vec{Y})}\ \text{Ty-Case}$$

* We gave a detailed explanation of Γ' in the paper.

Rules of F_{GT} ⟨2/2⟩: Typing Rules for Graphs

$$\frac{(\Gamma, P) \vdash T : \tau\,(\vec{X}) \qquad T \equiv T'}{(\Gamma, P) \vdash T' : \tau\,(\vec{X})}\ \text{Ty-Cong}$$

$$\frac{(\Gamma, P) \vdash T : \tau\,(\vec{X})}{(\Gamma, P) \vdash T\langle Z/Y\rangle : \tau\,(\vec{X})\langle Z/Y\rangle}\ \text{Ty-Alpha}$$

where $Z \notin fn(T)$

$$\frac{(\Gamma, P) \vdash T_1 : \tau_1(\overrightarrow{X_1}) \quad \dots \quad (\Gamma, P) \vdash T_n : \tau_n(\overrightarrow{X_n})}{(\Gamma, P\{\alpha(\vec{X}) \longrightarrow \mathscr{T}\}) \vdash \mathscr{T}\,[T_1/\tau_1(\overrightarrow{X_1}), \dots, T_n/\tau_n(\overrightarrow{X_n})] : \alpha(\vec{X})}\ \text{Ty-Prod}$$

where $\tau_i(\overrightarrow{X_i})$ *are all the type variable or arrow atoms appearing in* $\mathscr{T}$

Ty-Prod Example

$$\frac{(\Gamma, P) \vdash T_1 : \tau_1(\overrightarrow{X_1}) \quad \dots \quad (\Gamma, P) \vdash T_n : \tau_n(\overrightarrow{X_n})}{(\Gamma, P\{\alpha(\vec{X}) \longrightarrow \mathscr{T}\}) \vdash \mathscr{T}\,[T_1/\tau_1(\overrightarrow{X_1}), \dots, T_n/\tau_n(\overrightarrow{X_n})] : \alpha(\vec{X})}\ \text{Ty-Prod}$$

where $\tau_i(\overrightarrow{X_i})$ are all the type variable or arrow atoms appearing in $\mathscr{T}$

For example, for

$$nodes\,(Y, X) \longrightarrow \nu Z_1.\nu Z_2.(\mathrm{Cons}(Z_1, Z_2, X), nat\,(Z_1), nodes\,(Y, Z_2)) \qquad \cdots \qquad r_2$$

the Ty-Prod is

$$\frac{(\Gamma, P) \vdash T_1 : nat\,(Z_1) \qquad (\Gamma, P) \vdash T_2 : nodes\,(Y, Z_2)}{\begin{array}{l}(\Gamma, P\{P_2\}) \vdash \\ \quad \nu Z_1.\nu Z_2.(\mathrm{Cons}(Z_1, Z_2, X), nat\,(Z_1), nodes\,(Y, Z_2))[T_1/nat\,(Z_1), T_2/nodes\,(Y, Z_2)] \\ \quad = \nu Z_1.\nu Z_2.(\mathrm{Cons}(Z_1, Z_2, X), T_1, T_2) : nodes\,(Y, X)\end{array}}\ \text{Ty-Prod}$$

Example: Typing a Difference List

$$(\{n[Z_1] : \mathrm{nat}\,(Z_1)\}, \{r_1, r_2\}) \vdash \mathrm{Cons}(n, Y, X) : \mathit{nodes}\,(Y, X)$$

where r_1 and r_2 are the followings.

$$\begin{array}{rcl} \mathit{nodes}\,(Y, X) & \longrightarrow & X \bowtie Y \\ \mathit{nodes}\,(Y, X) & \longrightarrow & \mathrm{Cons}(\mathit{nat}, \mathit{nodes}\,(Y), X) \end{array}$$

can be shown as follows.

$$\dfrac{\dfrac{\dfrac{}{(\Gamma, P) \vdash n[Z_1] : \mathrm{n}\,(Z_1)}\,\text{Ty-Var} \quad \dfrac{\dfrac{}{(\Gamma, P\{r_1\}) \vdash X \bowtie Y : \mathit{nodes}\,(Y, X)}\,\text{Ty-Prod}}{(\Gamma, P) \vdash Z_2 \bowtie Y : \mathit{nodes}\,(Z_2, X)}\,\text{Ty-Alpha}}{(\Gamma, P\{r_2\}) \vdash T' : \mathit{nodes}\,(Y, X) \quad \text{where} \quad T' = \nu Z_1 Z_2.(\mathrm{Cons}(Z_1, Z_2, X), n[Z_1], Z_2 \bowtie Y)}\,\text{Ty-Prod} \quad T \equiv T'}{(\Gamma, P) \vdash T : \mathit{nodes}\,(Y, X) \quad \text{where} \quad T = \mathrm{Cons}(\mathit{succ}, Y, X)}\,\text{Ty-Cong}$$