A Functional Language with Graphs as First-Class Data

JSSST 2022

September 1, 2022

Waseda University, Tokyo, Japan

Jin SANO Kazunori UEDA

Overview

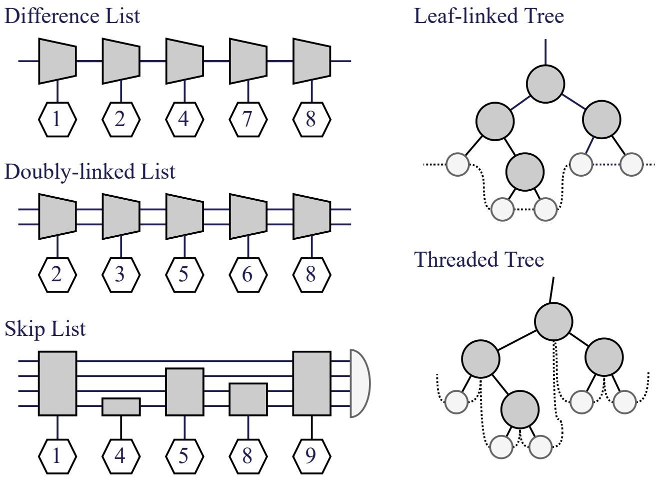
We propose a new purely functional language λ_{GT},
which handles graphs as immutable, first-class data
with pattern matching based on Graph Transformation.

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/*

Data structures more complex than trees



There are several important data structures (graphs) that are beyond trees.

How Programming Paradigms handle data

Imperative

- ! Heaps and pointers
- × Not Immutable

Purely Functional

- ✓ Algebraic Data Types (ADT)
- ✓ Immutable, First-class functions
- ✓ Type system
- × Complex data structures are difficult to handle

Graph Transformations[1]

- ✓ Graphs and pattern matching on them
- × Not Immutable, No First-class functions

[1] Hartmut Ehrig et al. **Fundamentals of Algebraic Graph Transformation**. Monographs in Theoretical Computer Science. 2006.

Our proposing language λ_{GT} is

a functional language with graphs as first-class data

- ✓ Graphs and pattern matching on them
- ✓ Immutable
- ✓ First-class functions
- ✓ Type system

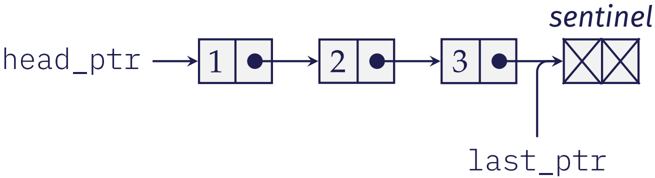
Contribution

0. We gave λ_{GT} a semantics based on HyperLMNtal[2]; a syntax-directed Graph Transformation formalism[3].
   - ✓ Simple and elegant and suitable for the type system but
   - × its implementation is not trivial. Therefore, ...
1. We build a reference implementation focusing on simplicity without regard to efficiency.

[2] Jin Sano and Kazunori Ueda. "Syntax-driven and compositional syntax and semantics of Hypergraph Transformation System". In: **Proc. 38nd JSSST Annual Conference**. 2021.

[3] Jin Sano, Naoki Yamamoto, and Kazunori Ueda. **Type checking data structures more complex than trees**. Presented at the 141th IPSJ Special Interest Group on Programming, Yamaguchi, Japan. 2022.

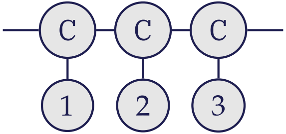
Queues with Lists in Imperative Style



Adding a new element needs

0. preparing a sentinel node and a pointer to the node,
1. creating a new last node,
2. destructive assignment to the previous last node, and
3. updating `last_ptr` ← forgettable!

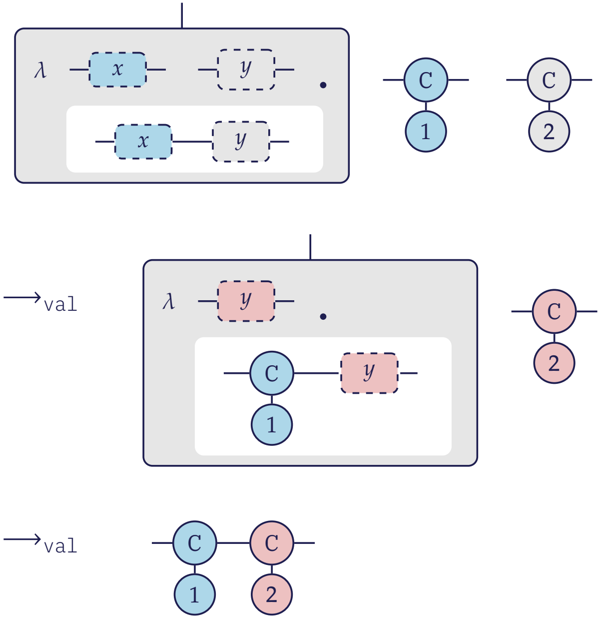
Queues with Lists in λ_{GT}



In λ_{GT}, such data structure can be abstracted to a **difference list**; a list with a link to the last node.

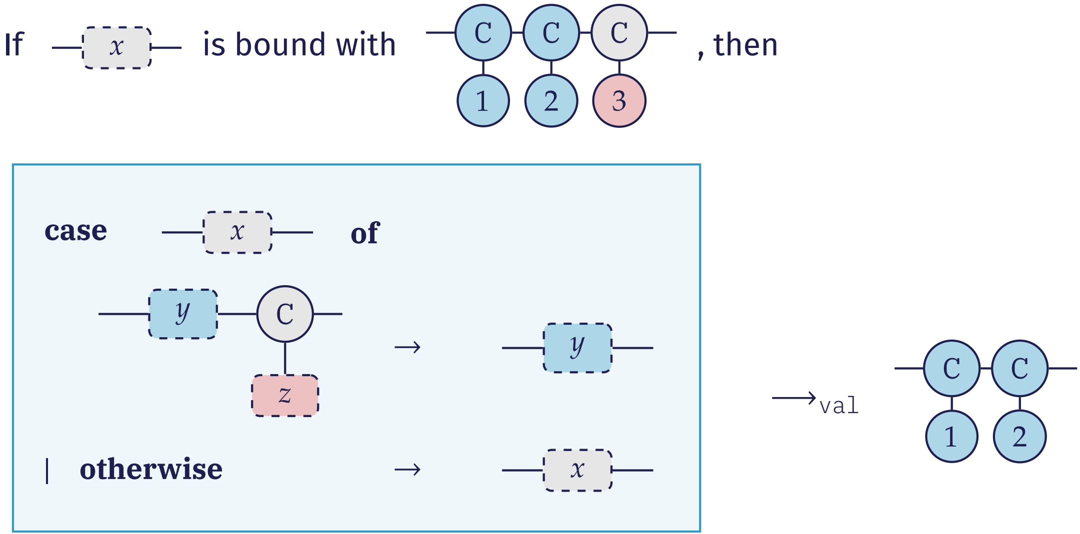
Adding a new element to the list can be understood as concatenating a singleton list.

Difference lists concatenation in λ_{GT}



...can be done with a type-safe, pure function.

Pattern matching graphs



Pop the last element of a difference list

Syntax of graphs in HyperLMNtal/λ_{GT}

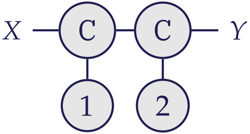
Graph

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

G	$::=$	$\mathbf{0}$	Null	*empty graph*
	$\mid$	$v\,(\vec{X})$	Atom	*vertex with name v and links $\vec{X}$*
	$\mid$	(G, G)	Molecule	*multiset of vertices*
	$\mid$	$\nu X.G$	Hyperlink Creation	*scope of link names*

For example, *Difference List* (*List Segment*) can be represented as

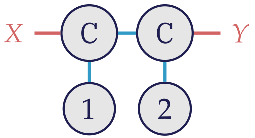
$$\begin{array}{l} \nu Z.(\\ \quad \nu Z_1.(\mathrm{Cons}(Z_1, Z, X), 1(Z_1)), \\ \quad \nu Z_2.(\mathrm{Cons}(Z_2, Y, Z), 2(Z_2)) \\ ) \end{array}$$



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

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

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.

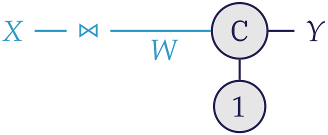
Fusion

Structural Congruence

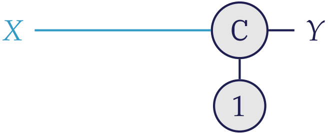
(E6) $\nu X.(X \bowtie Y, G) \quad \equiv \quad \nu X.G\langle Y/X\rangle$

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

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



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



Abbreviation schemes in HyperLMNtal

1. $\nu X_1. \dots \nu X_n.G$ can be abbreviated as $\nu X_1 \dots X_n.G$
2. A nullary atom $p()$ can be simply written as p
3. Term Notation:

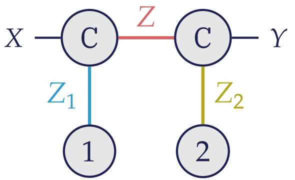
   $\nu X_n.(p(\dots, X_n, \dots), q(X_1, \dots, X_n))$ can be written as $p(\dots, q(X_1, \dots, X_{n-1}), \dots)$

For example,

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

can be abbreviated as

$\mathrm{Cons}(1, \mathrm{Cons}(2, Y), X)$



Syntax of λ_{GT}

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

wildcard

- ✓ λ_{GT} is designed to be a **small** language focusing on handling graphs.
- Value in λ_{GT} is a graph in HyperLMNtal
  - We allow $\bowtie$, *Constructor*, and λ-*abstraction* for the atoms' names

Syntax of λ_{GT}: Graph Template

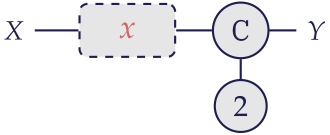
Graph Template

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

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

For example,

$$
\begin{array}{l}
\nu Z.(\\
\quad x[Z, X], \\
\quad \nu Z_2.(\mathrm{Cons}(Z_2, Y, Z), 2(Z_2)) \\
)
\end{array}
$$



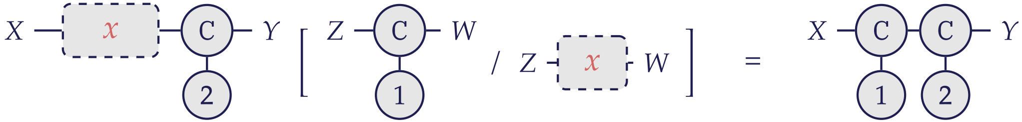
Graph Substitution

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.

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

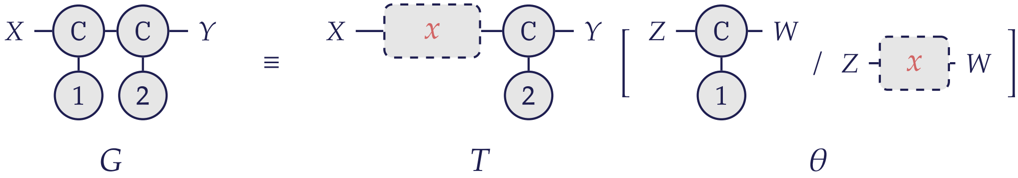
For example,



Graph Matching is defined 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,



Here, G can be matched to T with θ

Reduction of λ_{GT}

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

match succeeded

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

match failed

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

beta reduction

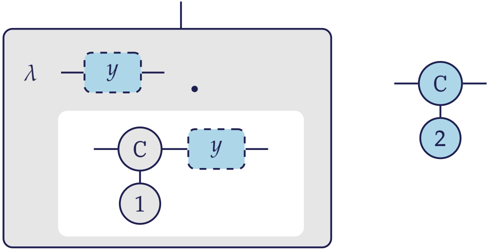
$$\frac{e \longrightarrow_{\texttt{val}} e'}{E[e] \longrightarrow_{\texttt{val}} E[e']}\ \text{Rd-Ctx}$$

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

Example of the β-reduction

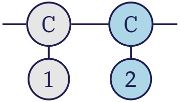
We can describe a program to append two singleton difference lists as follows.

$$\left(\begin{array}{l}\lambda\, y[Y,X].\\ \quad \mathrm{Cons}(1, y[Y], X)\end{array}\right)(Z) \quad \mathrm{Cons}(2, Y, X)$$



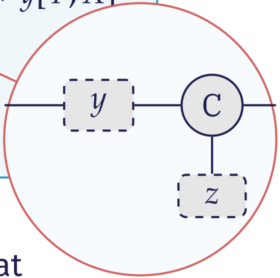
$$\begin{aligned} \longrightarrow_{\mathtt{val}} \;& \mathrm{Cons}(1, y[Y], X) \\ & \quad \left[\mathrm{Cons}(2, Y, X) / y[Y, X]\right] \\ = \;& \mathrm{Cons}(1, \mathrm{Cons}(2, Y), X) \end{aligned}$$

$\longrightarrow_{\mathtt{val}}$



Pop the last element of a difference list

$$
\begin{array}{l}
\textbf{let } \mathit{pop}[Z] = (\lambda\, x[Y,X].\\
\qquad \textbf{case } x[Y,X] \textbf{ of}\\
\qquad\quad \nu WZ.(y[W,X], \mathrm{Cons}(Z,Y,W), z[Z]) \to y[Y,X]\\
\qquad\ \mid \textbf{otherwise} \to x[Y,X]\\
\quad)(Z)\\
\textbf{in } \mathit{pop}[Z] \ \ G
\end{array}
$$



We need to find graph substitutions such that

$$
\begin{array}{l}
G\\
\equiv \ \ \nu WZ.(y[W,X], \mathrm{Cons}(Z,Y,W), z[Z])\\
\qquad [\ ?\ /y[W,X]][\ ?\ /z[Z]]
\end{array}
$$

Pattern matching graphs

If G is $\mathrm{Cons}(1, \mathrm{Cons}(2, Y), X)$, then it can be matched as follows.

$$
\begin{aligned}
&\nu WZ.(\,\mathrm{Cons}(1, W, X)\,, \mathrm{Cons}(Z, Y, W)\,, 2(Z)\,) \\
\equiv\ & \nu WZ.(\,y[W, X]\,, \mathrm{Cons}(Z, Y, W)\,, z[Z]\,) \\
& \quad \big[\mathrm{Cons}(1, W, X)/y[W, X]\big]\big[2(Z)/z[Z]\big]
\end{aligned}
$$

...seems not that difficult?

The corner case in pattern matching graphs

If G is $\mathrm{Cons}(1, Y, X)$,

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

$$\equiv\ \ \nu WZ.(\ y[W, X]\ ,\ \mathrm{Cons}(Z, Y, W)\ ,\ z[Z]\ )$$

$$[\ ?\ /y[W, X]][1(Z)/z[Z]]$$

...fails matching? $\rightarrow$ NO

Pattern matching with a supplying fusion

This time, we need to firstly **supply a fusion atom**.

Structural Congruence

$$\text{(E6)} \quad \nu X.(X \bowtie Y, G) \quad \equiv \quad \nu X.G\langle Y/X\rangle$$
$$\text{where } X \in \mathit{fn}(G) \vee Y \in \mathit{fn}(G)$$

The matching proceeds as follows.

$$\begin{aligned} & \nu WZ.(\mathrm{Cons}(Z,Y,X), 1(Z)) \\ \equiv \quad & \nu WZ.(W \bowtie X, \mathrm{Cons}(Z,Y,X), 1(Z)) \\ = \quad & \nu WZ.(y[W,X], \mathrm{Cons}(Z,Y,W), z[Z]) \\ & \quad \big[W \bowtie X / y[W,X]\big]\big[1(Z)/z[Z]\big] \end{aligned}$$

Implementation overview

The goal of this study is to implement as simple as possible, without regard to efficiency.

Our implementation consists of only 500 lines of OCaml code.

File	LOC
eval/match_ctxs.ml	79
parser/parser.mly	70
parser/lexer.mll	51
eval/syntax.ml	47
eval/eval.ml	43
eval/pushout.ml	42
eval/match_atoms.ml	36
eval/preprocess.ml	36
parser/syntax.ml	16
eval/match.ml	11
parser/parse.ml	4
bin/main.ml	3
SUM	438

Preprocessing

In the formal syntax		Data structure in the interpreter
$\nu Z.(\mathrm{Cons}(Z, Y, X), 1(Z))$	$\rightarrow$	$\left[\mathrm{Cons}(L_0, F_Y, F_X), 1(L_0)\right]$ *Host Graph*
$\nu W Z.(y[W, X], \mathrm{Cons}(Z, Y, W), z[Z])$	$\rightarrow$	$\left\langle \left[\mathrm{Cons}(L_{1000}, F_Y, L_{1001})\right], \left[y[L_{1001}, F_X], z[L_{1000}]\right] \right\rangle$ *Graph Template*

where $\begin{cases} L_i & \text{Local Link} \\ F_X & \text{Free Link} \end{cases}$

Matching atoms

Host Graph $\left[\mathrm{Cons}(L_0, F_Y, F_X), 1(L_0)\right]$

Graph Template $\left\langle \left[\mathrm{Cons}(L_{1000}, F_Y, F_{1001})\right], \left[y[L_{1001}, F_X], z[L_{1000}]\right] \right\rangle$

1. Match atoms with a mapping from the local links in the graph template to the links in the host graph.

$$\{L_{1000} \mapsto L_0, L_{1001} \mapsto F_X\}$$

2. Remove the matched atoms. Backtrack if fails.

Supplying a fusion

The rest host graph $\left[1(L_0)\right] \;\rightarrow\; \left[1(L_{1000}),\, L_{1001} \bowtie F_X\right]$

$$\{L_{1000} \mapsto L_0,\; L_{1001} \mapsto F_X\}$$

1. Substitute link names in the host graph with the inverse of the obtained link mapping.
2. Supply the fusion atom $L_i \bowtie F_X$ to the host graph if there exists a mapping $L_i \mapsto F_X$

Matching graph contexts

Host Graph $\left[\, 1(L_{1000}),\ L_{1001} \bowtie F_X \,\right]$

Graph Template $\left[\, y[L_{1001}, F_X],\ z[L_{1000}] \,\right]$

- Finally, we obtain the graph substitution

$$\left[L_{1001} \bowtie F_X \big/ y[L_{1001,F_X}],\ 1(L_{1000}) \big/ z[L_{1000}] \right]$$

What's next?

- × We did not focus on efficiency.
- → To improve performance, static analysis is necessary for
  1. efficient matching and
  2. ensuring safety over destructive rewriting and enabling it.
- → We are planning to extend the type system.

Summary

We propose a new purely functional language λ_{GT},
which handles graphs as immutable, first-class data
with pattern matching based on Graph Transformation.

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

→ Focused on simplicity without regard on efficiency.

We are developping static analysis and
planning to build an efficient compiler.

Related work

Comparison with Separation Logic

Related work

FUnCAL[4] is a functional language with Graph Transformation. The equality of graphs is defined with bisimulation. FUnCAL comes with its type system but does not support user-defined data types.

Initial algebra semantics for cyclic sharing tree structures[5] discusses how to express graphs by lambda expressions.

[4] Kazutaka Matsuda and Kazuyuki Asada. “A Functional Reformulation of UnCAL Graph-Transformations: Or, Graph Transformation as Graph Reduction”. In: **Proc. POPL’17**. 2017.

[5] Makoto Hamana. “Initial Algebra Semantics for Cyclic Sharing Tree Structures”. In: **Log. Methods Comput. Sci.** 6.3 (2010). URL: http://arxiv.org/abs/1007.4266.

Related work

There are several languages based on graph transformations. However, as far as we know, few published implementations have focused on simplicity.

HyperLMNtal, which is the language we have incorporated, has the compiler[6] and the runtime SLIM[7]. The compiler is written in Java in around 12,000 lines and the runtime is written in C++ in around 47,000 lines.

GP 2 has a reference interpreter[8]. This is written in around 1,000 lines of Haskell sources.

[6] LMNtal. https://github.com/lmntal/lmntal-compiler.

[7] SLIM. https://github.com/lmntal/slim; Masato Gocho, Taisuke Hori, and Kazunori Ueda. "Evolution of the LMNtal Runtime to a Parallel Model Checker". In: **Computer Software** (2011).

[8] Christopher Bak et al. "A Reference Interpreter for the Graph Programming Language GP 2". In: **Proceedings Graphs as Models**. 2015.

Related work

Comparison with Separation Logic

Comparison between Imperative Languages with λ_{GT}

Imperative Languages

- ✓ Heaps and pointers
- × Not Immutable
- ! **Verification techniques**
  Hoare triple, Separation Logic, Shape Analysis, ...

↔

Proposing language λ_{GT}

- ✓ Graphs and pattern matching on them
- ✓ Immutable
- ✓ First-class functions
- ✓ **Type system**
  simpler and automatic

Comparison between HyperLMNtal and Separation Logic

	Separation Logic	λ_{GT}/HyperLMNtal
Heap segment/Atom	$x \mapsto \vec{y}$	$C(\vec{X})$
Variable	x	$-$
Address/Hyperlink	$s(x)$	X
Separating Conjunction/Molecule	$*$	$,$
emp/null	$\mathbf{emp}$	$\mathbf{0}$
part of pure logic/fusion	$x = y$	$X \bowtie Y$
inductive predicate/non-terminal symbol	$P\vec{\mathrm{x}}$	$\alpha(\vec{X})$
existence quantifier/hyperlink creation	$\exists$	ν