An Interpretation of Cyclic Structure Into Type Theory

Takeuti Izumi
Department of Information Science, Kyoto Univ., 606-8501, JAPAN
E-mail: takeuti@kuis.kyoto-u.ac.jp

Abstract:

This work proposes an interpretation of cyclic structure into System F, a second order typed lambda calculus. This interpretation is a translation which translates types of cyclic structure into types of F, terms which denote cyclic structure into terms of F, and the equality of cyclic structure into a partial equivalence relation in F. By using this interpretation, we can write down objects of cyclic structure as terms of System F. Moreover, we can write down functions over objects of cyclic structure as terms of System F.

In order to translate cyclic structure into F, we propose three characteristic points. The first is that we distinguish the type of already constructed terms of cyclic structures from the type of terms of cyclic structures under construction. The second is that the domain of cyclic structure is reguarded as a quotient domain made of the domain of cyclic graphs. The third is that the equality of cyclic structure is defined by induction, and functions over objects of cyclic structure are defined by induction. This induction is provided by the axiom of parametricity.

1 Introduction

1.1 Syntax of rational binary tree

We often see cyclic structures in informatics research. One of the simplest cyclic structures is the data type of rational binary trees. Rational binary trees are binary trees with cyclic sharing. In order to express cycles, we use variables and a constructor **rec**. The syntax of rational binary trees is

$$T ::= \mathit{Var} \mid \mathbf{leaf} \mid \mathbf{branch}TT \mid \mathbf{rec}\,\mathit{Var}.T$$

where $\mathbf{rec}x.M$ can be constructed if x occurs only in the scope of **branch**. This restriction prohibits constructing terms such as $\mathbf{rec}x.x$, $\mathbf{rec}x.\mathbf{rec}y.x$ and so forth.

1.2 Semantics of rational binary tree

There are two interpretations of terms with the constructor **rec**. One is to regard a term as a directed graph with cycles. This interpretation distinguishes

$\mathbf{rec}x.\mathbf{branch}\,\mathbf{leaf}\,x$ from $\mathbf{rec}x.\mathbf{branch}\,\mathbf{leaf}\,(\mathbf{branch}\,\mathbf{leaf}\,x)$. The other is to embed into infinite trees.

The formal definition of the latter interpretation is given below. For a rational binary tree M and a number n, we define a finite tree $M|_n$ as follows.

$M|_0 := \mathbf{leaf}$,
$\mathbf{leaf}|_{n+1} := \mathbf{leaf}$,
$x|_{n+1} := x$ for a variable x,
$\mathbf{branch}MN|_{n+1} := \mathbf{branch}(M|_n)(N|_n)$,
$(\mathbf{rec}x.M)|_{n+1} := M[\mathbf{rec}x.M/x]|_{n+1}$.

If M is closed, then $M|_n$ has no variables. As n increases, $M|_n$ grows at the leaves. We obtain an infinite tree as the limit. It is regarded as the denotation of the rational tree M. We define $\models M = N$ as $\forall n.\ M|_n = N|_n$. We interpret both $\mathbf{rec}x.M$ and $M[\mathbf{rec}x.M/x]$ as the same rational tree in this interpretation, although the two are different expressions.

Hereafter in this paper, we regard the latter interpretation as the standard semantics of rational trees. When we say a rational tree, we talk about an equivalence class of terms which are interpreted into the same infinite tree. When we interpret a term of cyclic structure in the former interpretation, we call it a graph.

1.3 Axiomatization of equality

There is an axiomatization of the equality of rational binary trees [2]. In the followings, L, M and N are expressions of rational binary trees. The rules of the axiomatization are the followings.

$$\frac{}{M = M}\ (\text{reflexivity}) \qquad \frac{M = N}{N = M}\ (\text{symmetry})$$

$$\frac{L = M \quad M = N}{L = N}\ (\text{transitivity}) \qquad \frac{M = N}{L[M/x] = L[N/x]}\ (\text{congruence})$$

$$\frac{}{\mathbf{rec}x.M = M[\mathbf{rec}x.M/x]} \quad \frac{N = M[N/x]}{N = \mathbf{rec}x.M}\ (\text{cyclicity})$$

The semantics by infinite trees is sound and complete for this axiomatization, that is, $\vdash\ M = N$ by these rules iff $\models M = N$ [2].

Although the semantics of rational trees is given by interpretating into infinite trees, the interpretation is not satisfactory. The equality of rational trees is decidable [1]. However, the equality of infinite trees in general is only semidecidable, that is, one can tell that two are different, but one cannot tell that two are equal.

This work based upon the equation system in which equations are derived by finite rules. We want to extend this equation system into more rich theory

which is based on the inductive equation, and which can deal with functions over cyclic structures.

1.4 Cyclic structure

We propose a system which interprets cyclic structure. Our system deal not only with rational binary trees but also with rational trees with finite branching, labelled rational trees, or any other type of structure with cyclic sharing.

In this paper we call each type of data with cyclic sharing a type of cyclic structure. Namely, the type of rational trees is a type of cyclic structure. The elements of the types of cyclic structure are called objects of cyclic structure.

1.5 Terms and Types of F

We propose an interpretation of cyclic structures into System F. Why do we choose System F? That is because System F has a strong expressive power and that System F satisfies strong normalizability. We regard objects of cyclic structure in the semantics of expanded infinite structure. Thus, some careless calculation may not halt. It is not clear that each term has a unique value. If a term is interpreted into System F, then the strong normalizability and confluency of F guarantee that the term has exactly one value.

System F is a system with types and terms. Therefore, we have to tell what are interpreted into types, and what are interpreted into terms.

This interpretation interprets types of cyclic structure as inductive types in F. The inductive types in F are the types which are used for encoding inductive data, such as natural numbers, finite lists, finite trees, and so forth. For example, $\mu X.\mathbf{1} + X \equiv \Pi X.(\mathbf{1} + X \to X) \to X$ is an inductive type in F which is used for encoding natural numbers, and $\mu X.\mathbf{1}+X \times X \equiv \Pi.(\mathbf{1}+X \times X \to X) \to X$ is for finite binary trees, where $\mathbf{1}, +$ and $\times$ are some suitable encodings of unit type, some type, and pairing type in System F. The types of cyclic structure are also interpreted into this kind of types with some modification. Namely, the type of rational binary trees are encoded as $\Pi X.X \to \mu Y.X + (1 + (X + Y) \times (X + Y))$.

Our interpretation interprets objects of cyclic structure as terms of types of cyclic structure, that is, the interpretation translates terms which are constructed in the syntax of cyclic structure into terms of the types which denotes cyclic structure in System F. The objects of cyclic structure and cyclic graphs have the same syntax, therefore both their types are interpreted into the same types. Namely, the types of rational binary trees and directed graphs with binary branching are interpreted into the same type.

1.6 Function over cyclic structure

We propose an interpretation which deals not only with cyclic structure, but also with functions of cyclic structures into some other domain. One of our

purposes is to describe such a function in F by using our interpretation.

The way of describing functions over cyclic structures is as follows. First we describe a term which has a type of functions over cyclic graphs. Next we check whether the function is regarded as a function also over cyclic structure, that is, we check whether if two graphs denote the same object of cyclic structure, then the return values of the function applied to the two graphs are the same. We do this check by using the theory of parametricity, which is studied in [3, 6].

Cyclic structure often appears in the studies of graph rewriting [5]. In theories of graph rewriting, it depends only on the local condition whether a rewriting happens or not. In those studies, one looks at only the local structure, not the whole structure. One does not care whether the whole structure would be finite or infinite. Thus, one defines the equality by co-induction principle, and that one does not define functions over cyclic structure which depend upon their whole structure.

In contrast to this, our interpretation looks at the whole structures of cyclic structures. The functions over cyclic structures are calculated by the recursion on the structures. Our system has a destructor, which is a formal term denoting the recursion.

1.7 Typing

In our interpretation, cyclic structure is interpreted into type theory. We will define an interpretation of not only cyclic structure, but also functions over the objects of cyclic structure. Typing is necessary to describe functions of cyclic structures for the following two reasons.

One of the reasons is to prohibit constructing terms with no denotations. We want to have terms *left* and *right* which are functions of rational trees into rational trees such that

$$\mathit{left}(\mathbf{branch}MN) = M, \quad \mathit{left}\,\mathbf{leaf} = \mathbf{leaf},$$

$$\mathit{right}(\mathbf{branch}MN) = N, \quad \mathit{left}\,\mathbf{leaf} = \mathbf{leaf}.$$

Thus these functions works as destructors of **branch**. By using these functions, we can write down this term $\mathbf{rec}\,x.\mathbf{branch}\,\mathbf{leaf}(\mathit{left}\,x)$, which is calculated into $\mathbf{branch}\,\mathbf{leaf}\,\mathbf{leaf}$. However, we do not want to have this term $\mathbf{rec}\,x.\mathbf{branch}\,\mathbf{leaf}(\mathit{right}(\mathit{right}\,x))$, because this term has no rational binary tree as its value. Thus, we have to restrict the occurrences of destructors. We will define two type constructors σ and σ^X for cyclic structures in order to satisfy this restriction. The type σ is used for already constructed terms, and σ^X is used for terms under construction. We restrict the destructors to operate only upon already constructed terms.

The other reason is to tell whether a term denotes a step of recursion or not. We consider the case of natural numbers. When the type of natural numbers is encoded as $\mathbb{N} \equiv \mu X.1 + X$, The destructors of the type $\mathbb{N}$ are the terms which

denote primitive recursive operations. The construction of a destructor is to make a term $\boldsymbol{\rho} B : \mathbb{N} \to T$ from $B : 1 + T \to T$. In general, the construction of destructor of a finite structure $\mu X.FX$ is to make a term $\boldsymbol{\rho} B : (\mu X.FX) \to T$ from $B : FT \to T$. In $\boldsymbol{\rho} B$, the term B denotes a step of recursion. We can tell whether a term B denotes a step of recursion or not by looking at its type. Similarly, the construction of a destructor for cyclic structures demands a term of some type which denotes a step of recursion.

1.8 Outline

In Section 2, we give preliminary treatments of cyclic structures, and encoding cyclic structures by basic constructors. Especially, we give a simple syntax of cyclic structures, which we call the syntax of normal form, in Section 2.1. In Section 3, we give a translation of the terms of cyclic structure into terms of System F. In Section 4, we define partial equivalence relations which are the interpretation of domains of cyclic structure, and show that the left branch function *left* and right branch function *right* are functions over cyclic structure. In order to show that, we use the theory of parametricity.

2 Preliminary discussion

2.1 Syntax of Normal Forms

The rational tree $\mathbf{rec}x.\mathbf{rec}.y.M[x,y]$ is equal to the rational tree $\mathbf{rec}x.M[x,x]$. Also we have $\mathbf{rec}x.\mathbf{leaf} = \mathbf{leaf}$ and $\mathbf{rec}x.y = y$. Moreover, $\mathbf{branch}MN = \mathbf{rec}x.\mathbf{branch}MN$ for a fresh variable x. Thus we give another syntax of rational binary trees as follows:

$$\mathit{Tree}' ::= \mathit{Var} \mid \mathbf{leaf} \mid \mathbf{rec}\,\mathit{Var}.\mathbf{branch}\,\mathit{Tree}'\,\mathit{Tree}'$$

We write T' for the set of terms constructed by this light syntax. On the other hand, we write T for the set of terms constructed by the original syntax, that is

$$\mathit{Tree} ::= \mathit{Var} \mid \mathbf{leaf} \mid \mathbf{branch}\,\mathit{Tree}\,\mathit{Tree} \mid \mathbf{rec}\,\mathit{Var}.\,\mathit{Tree}$$

with the restriction that the variable x occurs only in the scope of **rec** in $\mathbf{rec}x.M$. We have $\mathit{Tree}' \subset \mathit{Tree}$ and $\forall M \in \mathit{Tree}.\, \exists N \in \mathit{Tree}'.\ \vdash M = N$. Thus we regard terms of Tree' as normal forms of the following rewriting:

$\mathbf{rec}x.\mathbf{rec}y.M[x,y]$	is rewritten into	$\mathbf{rec}x.M[x,x]$
$\mathbf{rec}x.\mathbf{leaf}$	is rewritten into	$\mathbf{leaf}$
$\mathbf{rec}x.y$	is rewritten into	y for different variables x and y
$\mathbf{branch}MN$	is rewritten into	$\mathbf{rec}x.\mathbf{branch}MN$ for a fresh variables x

if the occurrence of $\mathbf{branch}MN$ is not in the form of $\mathbf{rec}x.\mathbf{branch}MN$.

The syntax of *Tree*$'$ is a little simpler than that of *Tree*, although the terms in *Tree*$'$ may have many redundant **rec**'s.

Hereafter, we say the syntax of normal terms of rational binary trees only for this *Tree*$'$. Similarly, we use the syntax of normal terms in the above meaning for other types of cyclic structures.

2.2 Naïve encoding with basic type constructors

We will define an interpretation of cyclic structure into typed system F. Before defining the whole interpretation, We explain an encoding of terms of cyclic structure by basic type constructors. The basic type constructors are the unit type $\mathbf{1}$, $\to$ for functional types, $\times$ for pairing types, $+$ for sum types, μ for inductive types, and σ for types of cyclic structures. The basic type constructors can encode various data types.

For example, the type of natural numbers is encoded as $\mathbb{N} :\equiv \mu X.\mathbf{1} + X$. Zero and successor are encoded as $0 :\equiv \mathbf{f}(\mathbf{inl}\,\bullet^{\mathbf{1}})$ and $sn :\equiv \mathbf{f}(\mathbf{inr}n)$, where $\bullet^{\mathbf{1}}$ is a unique element of $\mathbf{1}$, and $\mathbf{f}$ is the fold operator which makes the term $\mathbf{f}M : \mathbb{N}$ from a term $M : 1 + \mathbb{N}$. We show one more example. The type of finite binary trees is encoded as $T :\equiv \mu X.\mathbf{1} + X \times X$. The leaf and branching operator are encoded as $\mathbf{leaf} :\equiv \mathbf{f}(\mathbf{inl}\,\bullet^{\mathbf{1}})$ and $\mathbf{branch}MN :\equiv \mathbf{f}(\mathbf{inr}\langle M, N\rangle)$, where $\mathbf{f}$ is the fold operator which makes the term $\mathbf{f}M : T$ from a term $M : 1 + T \times T$. The introduction rule of μ is

$$\frac{M : F\mu X.FX}{\mathbf{f}M : \mu X.FX}\ .$$

Now we encode rational trees. We encode the type of rational binary trees as $T :\equiv \sigma X.1 + X \times X$. First we try to make a naïve encoding. It is straightforward type matching that $\mathbf{leaf}$ and $\mathbf{rec}x.\mathbf{branch}MN$ are encoded as $\mathbf{leaf} :\equiv \mathbf{f}(\mathbf{rec}x.\mathbf{inl}\,\bullet^{\mathbf{1}})$ and $\mathbf{rec}x.\mathbf{branch}MN :\equiv \mathbf{rec}x.\mathbf{inr}\langle M, N\rangle$. The type inference of rational binary trees is constructed with the rules such as:

$$\frac{x : T \vdash M : 1 + T \times T}{\vdash \mathbf{rec}x.M : T}\ .$$

The type inference tree of $\mathbf{rec}x.\mathbf{branch}\,\mathbf{leaf}x$ is written as:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \bullet^{\mathbf{1}} : \mathbf{1}}}{\vdash \mathbf{inl}\,\bullet^{\mathbf{1}} : \mathbf{1} + T \times T}}{\vdash \mathbf{leaf} \equiv \mathbf{rec}y.(\mathbf{inl}\,\bullet^{\mathbf{1}}) : T} \qquad \dfrac{}{x : T \vdash x : T}}{x : T \vdash \langle \mathbf{leaf}, x\rangle : T \times T}}{x : T \vdash \mathbf{inr}\langle \mathbf{leaf}, x\rangle : \mathbf{1} + T \times T}}{\vdash \mathbf{rec}x.\mathbf{branch}\,\mathbf{leaf}x \equiv \mathbf{rec}x.\mathbf{inr}\langle \mathbf{leaf}, x\rangle : T}$$

In general, if $\mu X.FX$ denotes inductive structure generated by F, then $\sigma X.FX$ denotes a type of terms of cyclic structures generated by F in analogy. The introduction rule of σ is:

$$\frac{x : \sigma X.FX \vdash M : F\sigma X.FX}{\vdash \mathbf{rec}x.M : \sigma X.FX}.$$

This encoding is just naïve and is not available to encode functions over cyclic structure, because this encoding does not distinguish types of cyclic structure for terms under construction from those for already constructed terms. Such a distinction is necessary to introduce destructors such as *left* and *right*. Our interpretation uses type variables to indicate types of cyclic structure for terms under construction. Thus, our interpretation has three kinds of types for denoting cyclic structure. One is types for already constructed terms, which is $\sigma X.FX$. Another is types for terms under construction indicated by type variables, which is $\sigma^X Y.FY$. The last one is a type variable X which is used for this indication.

3 Interpretation of terms

First of all we define the interpretation of objects cyclic structure. Hereafter we often use the notation $M \circ N :\equiv \lambda x.M(Nx)$.

3.1 Basic constructors

Before defining the interpretation of cyclic structure, we describe a standard interpretation of basic type constructors $\times, +$ and μ into F.

3.1.1 Unit type

We will define abbreviations $\mathbf{1}$ and $\bullet^{\mathbf{1}}$ as follows:

$$\begin{array}{ll} \mathbf{1} & :\equiv \Pi X.X \to X, \\ \bullet^{\mathbf{1}} & :\equiv \Lambda X.\lambda x^X.x \quad \text{which is of type } \mathbf{1}. \end{array}$$

3.1.2 Pairing type

We will define abbreviations $\times, \langle,\rangle, \mathbf{left}$ and $\mathbf{right}$ as follows:

$$\begin{array}{lll} T \times U & :\equiv \Pi X.(T \to U \to X) \to X & \text{for types } T \text{ and } U, \\ \langle M, N\rangle & :\equiv \Lambda X.\lambda x.xMN & \text{which is of type } T \times U, \\ & & \text{for terms } M^T \text{ and } N^U, \\ \mathbf{left} & :\equiv \lambda x.xT\lambda yz.y & \text{which is of type } T \times U \to T, \\ \mathbf{right} & :\equiv \lambda x.xU\lambda yz.z & \text{which is of type } T \times U \to U. \end{array}$$

These serve as an encoding of paring. Thus, we have $\mathbf{left}\langle M, N\rangle =_\beta M$ and $\mathbf{right}\langle M, N\rangle =_\beta N$.

3.1.3 Sum type

We will define abbreviations $+, \mathbf{inl}, \mathbf{inr}$ and $\mathbf{cas}$ as follows:

$$\begin{array}{lll} T+U & :\equiv \Pi X.(T \to X) \to (U \to X) \to X & \text{for types } T \text{ and } U, \\ \mathbf{inl} & :\equiv \lambda x.\Lambda X.\lambda yz.yx & \text{which is of type } T \to T+U \text{ for terms } M^T, \\ \mathbf{inr} & :\equiv \lambda x.\Lambda X\lambda yz.zx & \text{which is of type } U \to T+U \text{ for terms } M^U, \\ \mathbf{cas} & :\equiv \lambda xyz.zVxy & \text{which is of type} \\ & & (T \to V) \to (U \to V) \to T+U \to V. \end{array}$$

These serve as an encoding of disjoint union. Thus, we have $\mathbf{cas}LM(\mathbf{inl}N) =_\beta LN$ and $\mathbf{cas}LM(\mathbf{inr}N) =_\beta MN$.

3.1.4 Functor

In order to define basic terms about inductive types, we introduce the notion of functor. Functors are not expressions in System F.

If T is a type, and a type variable X occurs only positively in T, then $\Lambda X.T$ is called a functor. Similarly, a type variable X occurs only negatively in T, then $\Lambda X.T$ is called a contravariant functor. $\Lambda X.T[X]$ is also written as $T[-]$.

A functor or a contravariant functor $\Lambda X.T$ operates upon types and terms of functional types, which have the form $T \to U$. The operation on types is defined as: $(\Lambda X.T)U := T[U/X]$. If F is a functor, and M is a term of type $T \to U$, then $\bar{F}(M)$ is a term of type $\bar{F}(T) \to \bar{F}(U)$. If F is a contravariant functor, and M is a term of type $T \to U$, then $\bar{F}(M)$ is a term of type $\bar{F}(U) \to \bar{F}(T)$. The operation on terms of functional types is defined inductively as below.

The case that X does not occur in T:

$\overline{\Lambda X.T}(M) :\equiv \lambda x^T.x.$

Otherwise:

$\overline{\Lambda X.X}(M) :\equiv M,$

$\overline{\Lambda X.T \to U}(M) :\equiv \lambda x.\left(\overline{\Lambda X.U}(M) \circ x \circ \overline{\Lambda X.T}(M)\right),$

$\overline{\Lambda X.\Pi Y.T}(M) :\equiv \lambda x.\Lambda Y.\overline{\Lambda X.T}(M)(xY).$

Latter we will define functors' operation upon predicates.

3.1.5 Inductive type

We will define abbreviations $\mu, \mathbf{f}, \mathbf{g}$ and $\boldsymbol{\rho}$ as follows:

$$
\begin{array}{ll}
\mu X.T & :\equiv \Pi X.(T \to X) \to X \\
 & \quad \text{as long as the type variable } X \text{ occurs} \\
 & \quad \text{only positively in the type } T, \\
\mathbf{f}_{\mu X.FX} & :\equiv \Lambda X.\lambda xy.y(\bar{F}(y)x) \\
 & \quad \text{which is of type } F(\mu X.FX) \to \mu X.FX, \\
\boldsymbol{\rho}_{\mu X.FX,T} & :\equiv \lambda xy.yTx \\
 & \quad \text{which is of type } (FT \to T) \to \mu X.FX \to T, \\
\mathbf{g}_{\mu X.FX} & :\equiv \boldsymbol{\rho}_{\mu X.FX,F\mu X.FX}\bar{F}(\mathbf{f}_{\mu X.FX}) \\
 & \quad \text{which is of type } (\mu X.FX) \to F\mu X.FX.
\end{array}
$$

We write simply μF for $\mu X.FX$ when F is a functor. We often omit the type index, that is, we often write $\boldsymbol{\rho}$ for $\boldsymbol{\rho}_{\mu X.FX,T}$. These serve as an encode of inductive types. The following diagram commutes up to β-equality. Thus, we have $\boldsymbol{\rho}M \circ \mathbf{f} =_\beta M \circ \bar{F}(\boldsymbol{\rho}M)$.

$$
\begin{array}{ccc}
 & \overset{\mathbf{f}}{\longrightarrow} & \\
F\mu F & \underset{\mathbf{g}}{\longleftarrow} & \mu F \\
\bar{F}(\boldsymbol{\rho}B)\downarrow & & \downarrow\boldsymbol{\rho}B \\
FT & \underset{B}{\longrightarrow} & T
\end{array}
$$

3.1.6 List

The type of lists of elements of T can be encoded as $\mu X.\mathbf{1} + T \times X$. We will define some terms which handle lists.

$$
\begin{array}{lll}
\mathsf{List} & :\equiv \Lambda Y.\mu X.\mathbf{1} + Y \times X & \text{which is a functor,} \\
\mathsf{(\,)} & :\equiv \mathbf{f}(\mathbf{inl}\bullet^{\mathbf{1}}) & \text{which is of type } \mathsf{List}\,T, \\
\mathsf{cons} & :\equiv \lambda xy.\mathbf{f}(\mathbf{inr}\langle x, y\rangle) & \text{which is of type } T \to \mathsf{List}\,T \to \mathsf{List}\,T, \\
\mathsf{head} & :\equiv \lambda x.\boldsymbol{\rho}(\mathbf{cas}(\lambda y.x)(\mathbf{left} \circ \mathbf{g})) & \text{which is of type } T \to \mathsf{List}\,T \to T, \\
\mathsf{tail} & :\equiv \boldsymbol{\rho}(\mathbf{cas}(\lambda y.\mathsf{(\,)})\mathbf{right}) & \text{which is of type } \mathsf{List}\,T \to \mathsf{List}\,T.
\end{array}
$$

$\mathsf{(\,)}$ is the empty list, or so called *nil*, cons is the addition of an element to a list on its head, or so called *cons-pair*. The term $\mathsf{head}ML$ returns the first element of $L^{\mathsf{List}\,T}$ if it exists, or if not, it returns the default value M^T. The term $\mathsf{tail}L^{\mathsf{List}\,T}$ returns the list of elements after the second element in L. If L is empty, then $\mathsf{tail}L$ returns $\mathsf{(\,)}$.

Thus the followings hold:

$$
\mathsf{head}N\mathsf{(\,)} =_\beta N,\ \mathsf{head}N(\mathsf{cons}ML) =_\beta M,
$$

$$
\mathsf{tail}\mathsf{(\,)} =_\beta \mathsf{(\,)},\ \mathsf{tail}(\mathsf{cons}ML) =_\beta L,
$$

$$
\overline{\mathsf{List}}(N)\mathsf{(\,)} =_\beta \mathsf{(\,)},\ \overline{\mathsf{List}}(N)(\mathsf{cons}ML) =_\beta \mathsf{cons}(NM)(\overline{\mathsf{List}}(N)L).
$$

3.2 Objects of cyclic structure

We will define abbreviations $\sigma, \sigma^X, \mathbf{rec}, \mathbf{rec}', \mathbf{var}, \lceil - \rceil, \lfloor - \rfloor$, and $\boldsymbol{\rho}$. These serve an encoding of cyclic structure in F.

$\sigma^X Y.FY \quad :\equiv \mu Y.X \to F(X+Y)$
for a functor F and fresh type variables X, Y,
$\sigma Y.FY \quad :\equiv \Pi X.\sigma^X Y.FY$ where X does not occur in FY,
$\mathbf{rec}\, x.M \quad :\equiv \mathbf{f}\lambda x.M$
which is of type $\sigma^X Y.FY$ for a variable x of type X
and a term M of type $F(X + \sigma^X Y.FY)$,
$\mathbf{rec}'\, x.M \quad :\equiv \mathbf{inr}(\mathbf{f}\lambda x.M)$ which is of type $X + \sigma^X Y.FY$,
$\mathbf{var}\, x \quad :\equiv \mathbf{inl}\, x$ which is of type $X + \sigma^X Y.FY$ for a variable x^X,
$\lceil M \rceil_X^{\sigma Y.FY} \quad :\equiv \Pi X.M$
which is of type $\sigma Y.FY$ for a term M of type $\sigma^X Y.FY$
where X does not occur freely in the types of free variables of M,
$\lfloor M \rfloor_X^{\sigma^X Y.FY} :\equiv MX$
which is of type $\sigma^X Y.FY$ for a term M of type $\sigma Y.FY$,
$\boldsymbol{\rho}_{\sigma X.FX,S,T} :\equiv \lambda xy.\boldsymbol{\rho}_{\mu X.S\to F(S+X),T} x(yS)$
which is of type $((S \to F(S+T)) \to T) \to \sigma Y.FY \to T$.

We will explain the intuitive meanings of these types and terms. We have pointed out that we need to distinguish the type of already constructed terms of cyclic structure from that of terms under construction. That is because the former terms can be destroyed freely by any destructor, but there are some restriction to apply some destructors to the latter terms. Therefore, we have two kinds of types for cyclic structure, one is the family of $\sigma Y.FY$, and the other is that of $\sigma^X Y.FY$.

As for $\sigma^X Y.FY$, it has a free variable X. We use this X for a type of variables x which is bound by **rec** in $\mathbf{rec}x.M$. Thus we prohibit this x from being destroyed by any destructor.

The term $\lfloor M \rfloor$ is used when one uses an already constructed term M as a component in construction of another term. The term $\lceil M \rceil$ is used when one completes the construction of a term. If $\lceil M \rceil \equiv \Pi X.M$ is constructed, then M has no free variable of type X. Thus M is ready to complete the construction.

3.2.1 Example: Rational Binary Tree

We define a translation of rational binary trees in the syntax of normal forms in Section 2.1 into terms of type $\sigma^X Y.1 + Y \times Y$ in System F. As an auxiliary, we also define a translation into $X + \sigma^X Y.1 + Y \times Y$.

First we show the encoding of finite binary trees by the type $\mu X.1 + X \times X$ in F. The syntax of finite binary trees is $T ::= \mathbf{leaf} \mid \mathbf{branch}TT$. A translation $(-)_{\mathrm{F}}$ translates terms in this syntax into terms of type $\mu X.1 + X \times X$.

$$(\mathbf{leaf})_{\mathrm{F}} \quad :\equiv \quad \mathbf{f}(\mathbf{inl}\, \bullet^{\mathbf{1}}),$$

$$(\mathbf{branch}MN)_{\mathrm{F}} \quad :\equiv \quad \mathbf{f}(\mathbf{inr}\langle (M)_{\mathrm{F}}, (N)_{\mathrm{F}}\rangle).$$

Let F be $\Lambda Y.1 + Y \times Y$, and R_X be $\sigma^X Y.FY$, which is type of rational binary trees in our interpretation. A translation $(-)_{\mathrm{F}}$ translates terms in the syntax of normal forms in Section 2.1 into terms of type R_X, and a translation $(-)'_{\mathrm{F}}$ translates terms in the syntax of normal forms in Section 2.1 into terms of type $X + R_X$.

$$\begin{aligned}
(x)'_{\mathrm{F}} &:\equiv \mathbf{var}x \\
(\mathbf{leaf})'_{\mathrm{F}} &:\equiv \mathbf{rec}'x.\mathbf{inl}\bullet^{\mathbf{1}}, \\
(\mathbf{rec}x.\mathbf{branch}MN)'_{\mathrm{F}} &:\equiv \mathbf{rec}'x.\mathbf{inr}\langle (M)'_{\mathrm{F}}, (N)'_{\mathrm{F}}\rangle, \\
(x)_{\mathrm{F}} &:\equiv \text{undefined}, \\
(\mathbf{leaf})_{\mathrm{F}} &:\equiv \mathbf{rec}x.\mathbf{inl}\bullet^{\mathbf{1}}, \\
(\mathbf{rec}x.\mathbf{branch}MN)_{\mathrm{F}} &:\equiv \mathbf{rec}x.\mathbf{inr}\langle (M)'_{\mathrm{F}}, (N)'_{\mathrm{F}}\rangle.
\end{aligned}$$

For example,

$$(\ \mathbf{rec}x.\mathbf{branch}x(\mathbf{rec}y.\mathbf{branch}x\mathbf{leaf})\ )_{\mathrm{F}} \equiv \mathbf{rec}\lambda x.\mathbf{inr}\langle \mathbf{rec}'\lambda y.\mathbf{inr}\langle x, \mathbf{rec}'\lambda y.\mathbf{inl}\bullet^{\mathbf{1}}\rangle\rangle$$

It is easy to show the following theorem.

Theorem 1 *For each closed $\beta\eta$-normal term M of type R_X, there exists a closed term N in the syntax of Section 2.1 such that $M =_\beta (N)'_{\mathrm{F}}$.*

3.2.2 Expansion of cyclic sharing

Let R be a type for cyclic structure. Many studies of cyclic structure give some denotations to self referential terms, namely in [5]. One can make a term $\mathbf{rec}x.M[x]$ from a term $M[x]$ with free variable x^R, The meaning of $\mathbf{rec}x.M[x]$ is $M[x]$ with x denoting $M[x]$ again. In other words, such studies have a construction which makes a term $\mathbf{rec}x.M[x]$ of type R from a term $\lambda x.M[x]$ of type $R \to R$. Hasegawa's work settles that by using traced category which can make an arrow of $A \to B$ from an arrow of $A \times X \to B \times X$.

In our interpretation, it is impossible to make a term of type R from a term of type $R \to R$, Because the term of type $R \to R$ may be made with a destructor such as *left* and *right* for binary tree. We do not regard the terms $\mathbf{rec}x.\mathit{left}x$ as a term of rational binary tree. On the other hand, in studies of graph rewriting, *left* and *right* are regarded as nodes, not destructors.

Instead of constructing terms of type R from terms of type $R \to R$, our interpretation supports an expansion of cyclic sharings which expands a term of R into a term of $R \to R$, that is, it expands $\mathbf{rec}x.M$ into $x \mapsto M$. This expansion is necessary to consider the equation rule of cyclicity such as $\mathbf{rec}x.M[x] = \mathbf{rec}y.M[\mathbf{rec}x.M[x]]$, where y is a fresh variable for void binding.

Now we define a term **expn** in F. For a functor F and a type T, we use the notation of a new functor:

$$F_T :\equiv \Lambda Y.T \to F(T + Y).$$

Let F be an arbitrary functor, a type R be $\sigma Y.FY$ and a type R_X be $\sigma^X Y.FY \equiv \mu F_X$. Our aim is to define a term of type $R \to R \to R$ which makes an object $\mathbf{rec}x.M[x]$ into a function $\lambda x^R.\mathbf{rec}y.M[x]$. But it is not enough, because the equation $\mathbf{rec}x.M[x] = \mathbf{rec}y.M[\mathbf{rec}x.M[x]]$ is instantiated not only when the term $\mathbf{rec}x.M[x]$ is closed, but also when it has some free variables. Hence we define also the term $\mathbf{expn}'$ of type $(\Pi X.\mathsf{List}X \to R_X) \to \mathsf{List}X \to R_X \to R_X$. In this type, the argument of type$\mathsf{List}X$ holds the list of free variables.

The terms $\mathbf{expn}$ and $\mathbf{expn}'$ are defined as follows:

$$
\begin{aligned}
&\mathbf{expn} :\equiv \lambda xy.\lceil \mathbf{expn}'(\Lambda X.\lambda z.\lfloor x \rfloor)\,()\,\lfloor y \rfloor_X \rceil_X,\\
&\mathbf{expn}' :\equiv \lambda xy.(\mathbf{f} \circ \lambda vw.w \circ \bar{F}(\\
&\qquad\qquad \mathbf{cas}(\mathbf{cas}\,\mathbf{inl}\,\mathbf{inr})\\
&\qquad\qquad (\mathbf{inr}(\boldsymbol{\rho}\lambda z.\mathbf{f}(\mathbf{inl} \circ z \circ \bar{F}(\mathbf{cas}(\mathbf{cas}\,\mathbf{inl}\,\mathbf{inr})\mathbf{inr})\\
&\qquad)\circ \mathbf{g}(x(X+\mu F_X)(\overline{\mathsf{List}}y)) \circ \mathbf{inr}).
\end{aligned}
$$

The term $\mathbf{expn}'$ is made along the following consturuction.

$$
\begin{array}{rcccc}
y & : & \mathsf{List}X & \cdots & X\\
 & & \Big\downarrow \overline{\mathsf{List}}(\mathbf{inl}) & & \Big\downarrow \mathbf{inl}\\
\overline{\mathsf{List}}(\mathbf{inl})y & : & \mathsf{List}(X+\mu F_X) & \cdots & X+\mu F_X
\end{array}
$$

Thus,

$$
\begin{array}{rcl}
x & : & \Pi X.\mathsf{List}X \to \mu F_X\\
 & \downarrow & \\
x(X+\mu F_X) & : & \mathsf{List}(X+\mu F_X) \to \mu F_{X+\mu F_X}\\
 & \downarrow & \\
x(X+\mu F_X)(\overline{\mathsf{List}}(\mathbf{inl})y) & : & \mu F_{X+\mu F_X}\\
 & \downarrow & \\
\mathbf{g}(x(X+\mu F_X)(\overline{\mathsf{List}}(\mathbf{inl})y)) & : & F_{X+\mu F_X}\mu F_{X+\mu F_X}\\
 & \equiv & \underline{X+\mu F_X} \to F((X+\mu F_X)+\mu F_{X+\mu F_X})\\
 & \downarrow & \\
\mathbf{g}(x(X+\mu F_X)(\overline{\mathsf{List}}(\mathbf{inl})y)) \circ \mathbf{inr} & : & \mu F_X \to F((X+\mu F_X)+\mu F_{X+\mu F_X})
\end{array}
$$

On the other hand,

$$
\begin{array}{rcccc}
\mathbf{inl} & : & X & \to & X+\mu F_X\\
\mathbf{inr} & : & \mu F_X & \to & X+\mu F_X\\
\mathbf{inr}(*) & : & \mu F_{X+\mu F_X} & \to & X+\mu F_X\\
\hline
\mathbf{cas}(\mathbf{cas}\,\mathbf{inl}\,\mathbf{inr})(\mathbf{inr}(*)) & : & (X+\mu F_X)+\mu F_{X+\mu F_X} & \to & X+\mu F_X\\
\downarrow & & \downarrow & & \downarrow\\
\bar{F}(\mathbf{cas}(\mathbf{cas}\,\mathbf{inl}\,\mathbf{inr})(\mathbf{inr}(*))) & : & F((X+\mu F_X)+\mu F_{X+\mu F_X}) & \to & F(X+\mu F_X)
\end{array}
$$

And also,

$$\begin{array}{rcl}
\lambda vw.w & : & F(X+\mu F_X) \to X \to F(X+\mu F_X) \\
 & \equiv & F(X+\mu F_X) \to F_X(\mu F_X) \\
\mathbf{f} & : & F_X(\mu F_X) \to \mu F_X \\
\hline
\mathbf{f} \circ \lambda vw.w & : & F(X+\mu F_X) \to \mu F_X
\end{array}$$

Thus,

$$\mathbf{f} \circ \lambda vw.w \circ \bar{F}(\mathbf{cas}(\mathbf{cas\ inl\ inr})(\mathbf{inr}(*))) \circ \mathbf{g}(x(X+\mu F_X)(\overline{\mathsf{List}}(\mathbf{inl})y)) \circ \mathbf{inr} \ : \ \mu F_X \to \mu F_X$$

In the diagram, the term $(*)$ is

$$\boldsymbol{\rho}\lambda z.\mathbf{f}(\mathbf{inl} \circ z \circ \bar{F}(\mathbf{cas}(\mathbf{cas\ inl\ inr})\mathbf{inr})) \ : \ \mu F_{X+\mu F} \to \mu F_X,$$

which is made from the arrow of $F_{X+\mu F_X}\ \mu F_X \to \mu F_X$ given by the following construction.

$$\begin{array}{rcl}
\mathbf{cas}(\mathbf{cas\ inl\ inr})\mathbf{inr} & : & (X+\mu F_X) \to X+\mu F_X \\
 & \downarrow & \\
\bar{F}(\mathbf{cas}(\mathbf{cas\ inl\ inr})\mathbf{inr}) & : & F((X+\mu F_X)+\mu F_X) \to F(X+\mu F_X) \\
z & : & F_{X+\mu F_X}\ \mu F_X \\
 & \equiv & X+\mu F_X \to F((X+\mu F_X)+\mu F_X) \\
\mathbf{inl} & : & X \to X+\mu F_X \\
\hline
\mathbf{inl} \circ z \circ \bar{F}(\mathbf{cas}(\mathbf{cas\ inl\ inr})\mathbf{inr}) & : & X \to F(X+\mu F_X) \\
 & \equiv & F_X \mu F_X
\end{array}$$

$$\Big\downarrow$$

$$\begin{array}{rcl}
\mathbf{f}(\mathbf{inl} \circ z \circ \bar{F}(\mathbf{cas}(\mathbf{cas\ inl\ inr})\mathbf{inr})) & : & \mu F_X \\
 & \downarrow & \\
\lambda z.\mathbf{f}(\mathbf{inl} \circ z \circ \bar{F}(\mathbf{cas}(\mathbf{cas\ inl\ inr})\mathbf{inr})) & : & F_{X+\mu F_X}\ \mu F_X \to \mu F_X \\
 & \downarrow & \\
\boldsymbol{\rho}\lambda z.\mathbf{f}(\mathbf{inl} \circ z \circ \bar{F}(\mathbf{cas}(\mathbf{cas\ inl\ inr})\mathbf{inr})) & : & \mu F_{X+\mu F_X} \to \mu F_X
\end{array}$$

We consider rational binary trees as an example. Let F be a functor $\Lambda Y.1 + Y \times Y$. Let $\mathbf{rec}x.M[x]$ and N be closed terms in the syntax of normal forms in Section 2.1. Then, we have the following equation:

$$\mathbf{expn}\ulcorner(\mathbf{rec}x.M[x])_{\mathrm{F}}\urcorner\ulcorner(N)_F\urcorner =_\beta \ulcorner(\mathbf{rec}y.M[N])_{\mathrm{F}}\urcorner$$

where y is a fresh variable for void binding.

We have an equation not only for closed terms, but also for open terms. Let $\mathbf{rec}x.M[x]$ and N be open terms in the syntax of normal forms in Section 2.1, and $\langle z_1, z_2, ..., z_n \rangle$ be the list of all the free variables of $\mathbf{rec}x.M[x]$ and N. Put Z as $Z \equiv \mathsf{cons}z_1(\mathsf{cons}z_2...(\mathsf{cons}z_n\,(\,))...)$, which is an encode of the list $\langle z_1, z_2, ..., z_n \rangle$ in F. Let M' and N' be closed terms of type $\Pi X.\mathsf{List}X \to \sigma^X Y.FY$ in F such that

$$M'XZ =_\beta (\mathbf{rec}x.M[x])'_F,\ N'XZ =_\beta (\mathbf{rec}x.N[x])'_F.$$

Then, we have the following equation:

$$\mathbf{expn}M'Z(N'XZ) =_\beta (\mathbf{rec}y.M[N])'_{\mathrm{F}}$$

where y is a fresh variable for void binding.

3.3 Destructor of graphs

The functions over inductive structure are often defined as recursive calculation. In type theory, a term denoting recursive calculation works as a destructor of the type for the inductive structure.

For example, the inductive type is encoded as $\mu X.FX$ in System F. The construction of recursive calculation over natural numbers is:

$$\frac{B : FT \to T}{\boldsymbol{\rho}_{\mu X.FX,T}B : (\mu X.FX) \to T}\ .$$

This construction is often called structural recursion. In this rule, the term B of type $FT \to T$ denotes an recursion step.

Similarly, destructors of cyclic structure will be defined by the following construction:

$$\frac{\dfrac{B : (S \to F(S+T)) \to T}{\boldsymbol{\rho}_{\mu Y.F_S Y,T}B : (\mu F_S) \to T} \quad \dfrac{m : \sigma Y.FY}{MS : \mu F_S}}{\boldsymbol{\rho}_{\mu Y.F_S Y,T}B(MS) : T}$$

In this rule, T is a type of codomain of the constructed function, but there are no mention to S, so one can take an arbitrary type as S. We write $\boldsymbol{\rho}_{\sigma Y.FY,S,T}BM$ for $\boldsymbol{\rho}_{\mu Y.F_S Y,T}B(MS)$ by its definition.

We show some examples about rational binary trees. In the following examples, let F be $\Lambda X.1 + X \times X$, thus $\sigma Y.FX \equiv \Pi X.\mu F_X$ is a type of rational binary trees.

Example 3.3.1 we define the function which counts the number of leaves. The type $\mathbb{N}$ is a standard encoding of natural numbers in F, The terms $\bar{0}, \bar{1}, \bar{2}, ...$ denotes zero, one two, and so on in the encoding. The term s denote successor, and plus denotes plus function in the encoding. Put b_1 as:

$b_1 :\equiv \lambda w.\mathbf{cas}(\lambda y.\bar{1})$

$$(\lambda x.\mathsf{plus}(\mathbf{cas}(\lambda w.\bar{0})(\lambda z.z)(\mathbf{left}x))(\mathbf{cas}(\lambda w.\bar{0})(\lambda z.z)(\mathbf{right}x)))$$

of type $(\mathbf{1} \to F(\mathbf{1} + \mathbb{N})) \to \mathbb{N}$

Then the following equations hold.

$\boldsymbol{\rho} b_1 \lceil \mathbf{leaf} \rceil =_\beta \bar{1}$

$\boldsymbol{\rho} b_1 \lceil (\mathbf{rec}x.\mathbf{branch}xx)_{\mathrm{F}} \rceil =_\beta \bar{0}$

$\boldsymbol{\rho} b_1 \lceil (\mathbf{rec}x.\mathbf{branch}\,\mathbf{leaf}\,x)_{\mathrm{F}} \rceil =_\beta \bar{1}$

$\boldsymbol{\rho} b_1 \lceil (\mathbf{rec}x.\mathbf{branch}\,\mathbf{leaf}\,(\mathbf{branch}\,\mathbf{leaf}\,x))_{\mathrm{F}} \rceil =_\beta \bar{2}$

Example 3.3.2 Next we will define the term which calculates the depth of the shallowest leaf. Let min be a term of type $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$ such that

$\mathsf{min}\,\bar{m}\bar{n} =_\beta \bar{m}$ iff $m \leq n$ or $n = 0$,

$\mathsf{min}\,\bar{m}\bar{n} =_\beta \bar{n}$ iff $1 \leq n < m$.

Put b_2 as:

$$b_2 :\equiv \lambda w.\mathbf{cas}(\lambda y.\bar{1})(\lambda x.\mathsf{min}(\mathbf{cas}(\lambda w.\bar{0})\mathsf{s}(\mathbf{left}x))(\mathbf{cas}(\lambda w.\bar{0})\mathsf{s}(\mathbf{right}x)))$$

of type $(\mathbf{1} \to F(\mathbf{1} + \mathbb{N})) \to \mathbb{N}$

Then the following equations hold.

$\boldsymbol{\rho} b_2 \lceil \mathbf{leaf} \rceil =_\beta \bar{1}$

$\boldsymbol{\rho} b_2 \lceil (\mathbf{rec}x.\mathbf{branch}xx)_{\mathrm{F}} \rceil =_\beta \bar{0}$

$\boldsymbol{\rho} b_2 \lceil (\mathbf{rec}x.\mathbf{branch}\,\mathbf{leaf}\,x)_{\mathrm{F}} \rceil =_\beta \bar{1}$

$\boldsymbol{\rho} b_2 \lceil (\mathbf{rec}x.\mathbf{branch}x(\mathbf{branch}x\mathbf{leaf}))_{\mathrm{F}} \rceil =_\beta \bar{1}$

Example 3.3.3 Next we will define the terms *left* and *right*. which returns the left branch and the right branch of a term given as an argument. We write R for the type $\sigma Y.FY$ and R_T for the type $\mu Y.T \to F(T + Y)$, or μF_T. Let N be some term of type $\sigma Y.FY$. Put b_3 as:

$$b_3 :\equiv \lambda w.\mathbf{f}(\bar{F}(\mathbf{cas}(\mathbf{cas}\,\mathbf{inl}\,\mathbf{inr})\mathbf{inr}) \circ \mathbf{g}w \circ \mathbf{inr})$$

of type $(F_{X+\mu F_X}\ \mu F_X) \to \mu F_X$.

This term b_3 also appears in the construction of **expn**. Then we have a term $\boldsymbol{\rho} b_3$ of type $(\mu F_{X+\mu F_X}) \to \mu F_X$. Put $E[N]$ as

$$E[N] :\equiv \lambda x.\bar{F}(\mathbf{cas}(\lambda z.NX)(\boldsymbol{\rho} b_3))(\mathbf{g}(x(X + \mu F_X)(\mathbf{inr}(NX))))$$

of type $R \to F\mu F_X \equiv 1 + R_X \times R_X$. Then we have these equations:

$E[(N)_{\mathrm{F}}]\lceil \mathbf{leaf} \rceil =_\beta \mathbf{inl}\bullet^{\mathbf{1}}$,

$E[(N)_{\mathrm{F}}]\lceil (\mathbf{rec}x.\mathbf{branch}L[x]M[x])_{\mathrm{F}} \rceil =_\beta \mathbf{inr}\langle (L[N]), (M[N]) \rangle$.

Now we define *left* and *right* as the following.

$$\begin{aligned} \mathit{left} &:\equiv \lambda x.\mathbf{cas}(\lambda z.\mathbf{leaf})\mathbf{left}(E[x]x) \\ &\equiv \lambda x.\mathbf{cas}(\lambda z.\mathbf{leaf})\mathbf{left} \\ &\qquad \lambda y.\bar{F}(\mathbf{cas}(\lambda z.xX)(\boldsymbol{\rho} b_3))(\mathbf{g}(y(X + \mu F_X)(\mathbf{inr}(xX)))) \\ \mathit{right} &:\equiv \lambda x.\mathbf{cas}(\lambda z.\mathbf{leaf})\mathbf{right}(E[x]x) \\ &\equiv \lambda x.\mathbf{cas}(\lambda z.\mathbf{leaf})\mathbf{right} \\ &\qquad \lambda y.\bar{F}(\mathbf{cas}(\lambda z.xX)(\boldsymbol{\rho} b_3))(\mathbf{g}(y(X + \mu F_X)(\mathbf{inr}(xX)))) \end{aligned}$$

Then the following equations hold:

$$\begin{array}{l} \mathit{left}\mathbf{leaf} =_\beta \mathit{right}\mathbf{leaf} =_\beta \mathbf{leaf}, \\ \mathit{left}\lceil(\mathbf{rec}x.\mathbf{branch}L[x]M[x])_{\mathrm{F}}\rceil =_\beta (L[\mathbf{rec}x.\mathbf{branch}L[x]M[x]])_{\mathrm{F}}, \\ \mathit{right}\lceil(\mathbf{rec}x.\mathbf{branch}L[x]M[x])_{\mathrm{F}}\rceil =_\beta (M[\mathbf{rec}x.\mathbf{branch}L[x]M[x]])_{\mathrm{F}}. \end{array}$$

We have got four functions $\boldsymbol{\rho}b_1, \boldsymbol{\rho}b_2, \mathit{left}$ and right. The function $\boldsymbol{\rho}b_1$ is only a function over the graphs of rational binary trees, not a function over rational binary trees, although other tree are functions over rational binary trees. The two terms $\mathbf{rec}x.\mathbf{branch}\,\mathbf{leaf}x$ and $\mathbf{rec}x.\mathbf{branch}\,\mathbf{leaf}(\mathbf{branch}\,\mathbf{leaf})$ are different as graphs, but equal one as a rational tree. Hence a function of rational binary trees returns the same value for the two terms $\mathbf{rec}x.\mathbf{branch}\,\mathbf{leaf}x$ and $\mathbf{rec}x.\mathbf{branch}\,\mathbf{leaf}(\mathbf{branch}\,\mathbf{leaf})$. The function $\boldsymbol{\rho}b_1$ returns 1 as the return value for the argument $\mathbf{rec}x.\mathbf{branch}\,\mathbf{leaf}x$, and returns 2 for the argument $\mathbf{rec}x.\mathbf{branch}\,\mathbf{leaf}(\mathbf{branch}\,\mathbf{leaf}\,x)$. Therefore $\boldsymbol{\rho}b_1$ is not a function of rational trees.

In the next section, we introduce a logical system in order to prove a function over graphs is also a function over cyclic structure.

4 Function over cyclic structure

One of our aims is to describe functions over cyclic structure. In the last section, we have written down functions over graphs denoting objects of cyclic structure.

4.1 Quotient domain and functors' operations

We define a domain of objects of cyclic structure as a quotient domain, which is expressed by a partial equivalence relation in predicate logic. In order to deal with these relations, we use the theory of parametricity, which is first studied by Plotkin and Abadi in [3], and later by Takeuti in [6]. We sometimes write binary relations as infix in the theory, namely $x\Phi y$, as in [3].

Let Φ be a binary predicate over type T. Then, Φ is a partial equivalence relation iff its symmetricity $\forall xy.\, x\Phi y \supset y\Phi x$ and its transitivity $\forall xyz.\, x\Phi y \supset y\Phi z \supset y\Phi z$ hold.

Let Φ be a partial equivalence relation over type T. Then Φ determines a quotient domain in the domain of T. We write $\mathcal{D}_\Phi$ for the quotient domain. Let M and N be a term of type T. Then, $M \in \mathcal{D}_\Phi$ iff $M\Phi M$, and M and N are equal as the element of $\mathcal{D}_\Phi$ iff $M\Phi N$.

We write $\mathcal{D}_\Phi \to \mathcal{D}_\Psi$ for the domain of functions from $\mathcal{D}_\Phi$ into $\mathcal{D}_\Psi$. The domain $\mathcal{D}_\Phi \to \mathcal{D}_\Psi$ is also a quotient domain. We will define a predicate $\Phi \Rightarrow \Psi$ as follows in order that $\Phi \Rightarrow \Psi$ determines the domain $\mathcal{D}_\Phi \to \mathcal{D}_\Psi$. Let Φ be a binary predicate over type T, and Ψ be a binary predicate over type U. Then $\Phi \Rightarrow \Psi$ is a binary predicate over type $T \to U$ such that:

$$\Phi \Rightarrow \Psi :\equiv \lambda x^{T\to U} y^{T\to U}.\forall v^T w^U.\, v\Phi w \supset (xv)\Psi(yw)$$

It is easily shown that if Φ and Ψ are partial equivalence relations, then so is $\Phi \Rightarrow \Psi$.

Later we will sometimes use functors operating upon formulae. Here we will define that. Let F be a functor, and Φ be a predicate, then functors' operation upon formulae is defined as $\bar{F}\Phi :\equiv \langle FX\rangle\{\Phi/X\}$ in the notation of [6].

4.2 Partial Equivalence relation for cyclic structure

We will define a predicate $Q_{\sigma Y.FY}$ over type $\sigma Y.FX$, which is an equivalence relation denoting equality of cyclic structure. We write R for $\sigma Y.FY$, and R_X for $\sigma^X Y.FY$. Our aim is to define a predicate Q_R over R which determines the domain of cyclic structure. In order to do that, we define a predicate $Q'_{\Pi.\mathsf{List}X\to R_X}$ over $\Pi.\mathsf{List}X \to R_X$, because the axioms of equality in Section 1.3 consists of equations of open terms. We often omit the type subscript in $Q'_{\Pi.\mathsf{List}X\to R_X}$, as Q'. The predicate Q' satisfies the following formulae:

$$
\begin{array}{l}
\forall x.\, xQ'x,\\
\forall xy.\, xQ'y \supset yQ'x,\\
\forall xyz.\, xQ'y \supset yQ'z \supset xQ'z,\\
\forall xy.\, xQ'(\Lambda X.\lambda z.\mathbf{expn}'yz(yXz)) \supset xQ'y,\\
\forall xy.\, xQ'(\Lambda X.\lambda z.\mathbf{expn}'yz(xXz)) \supset xQ'y,\\
\forall xy.\, (\forall X.\forall z^{\mathsf{List}X} w^X.\, (\mathbf{g}(xXz)w)\ \bar{F}(\mathcal{E}q_X \bar{+} \Phi[Q', X, w, z])\ (\mathbf{g}(yXz)w))\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \supset xQ'y
\end{array}
$$

where $\Phi[\phi, X, w, z] \equiv \lambda x^{R_X} y^{R_X}.\exists u^{\Pi.\mathsf{List}X\to R_X} v^{\Pi.\mathsf{List}X\to R_X}.$

$$x = uX(\mathsf{cons}wz) \wedge y = vY(\mathsf{cons}wz) \wedge v\phi w.$$

The last formula means congruency, and the forth and fifth formulae mean cyclicity. The predicate Q' is defined as the least relation which satisfies the formulae above, that is

$$
\begin{array}{l}
Q' :\equiv \mu\phi.\lambda xy.\\
\qquad x = y \vee y\phi x \vee (\exists z.\, x\phi z \wedge z\phi y)\\
\qquad \vee\, x\phi(\Lambda X.\lambda z.\mathbf{expn}'yz(yXz)) \vee x\phi(\Lambda X.\lambda z.\mathbf{expn}'yz(xXz))\\
\qquad \vee \forall X.\forall z^{\mathsf{List}X} w^X.\, (\mathbf{g}(xXz)w)\ \bar{F}(\mathcal{E}q_X \bar{+} \Phi[\phi, X, w, z])\ (\mathbf{g}(yXz)w)
\end{array}
$$

where $\Phi[\phi, X, w, z]$ is defined in the previous formula.

Now we define the predicate Q_R as:

$$Q_R :\equiv \lambda x^R y^R.\, (\Lambda X.\lambda z^{\mathsf{List}X}.\lfloor x\rfloor_X)\, Q'\, (\Lambda X.\lambda z^{\mathsf{List}X}.\lfloor y\rfloor_X).$$

The next theorem shows the soundness of this interpretation for rational binary trees.

Theorem 2 *Let F be the functor $\Lambda Y.1 + Y \times Y$, R be $\sigma Y.FY$. Let M and N be closed terms in the syntax of Section 2.1. If $\vdash M = N$ by the axioms of 1.3, then the theory of parametricity proves $(M)_{\mathrm{F}} Q_R (N)_{\mathrm{F}}$.*

4.3 Example: left branch and right branch

Now we discuss the examples of left branch function and right branch function again. We will prove that the two functions are defined over cyclic structure, not only over graphs. That is to derive the formulae $\mathit{left}(Q_R \Rightarrow Q_R)\mathit{left}$ and $\mathit{right}(Q_R \Rightarrow Q_R)\mathit{right}$.

The theory of parametricity proves the initiality of μF_X. Hence we can derive $\mathbf{f} \circ \mathbf{g} = \lambda x.x$. By using this equation, we have the following formula in the formal theory:

$$\mathit{left}\, M = \mathit{left}(\mathbf{expn} MM),\ \mathit{right}\, M = \mathit{right}(\mathbf{expn} MM).$$

Thus, we have

$$\forall xy.(\mathit{left}\, x)Q_R(\mathit{left}(\mathbf{expn} yy)) \supset (\mathit{left}\, x)Q_R(\mathit{left}\, y),$$

$$\forall xy.(\mathit{right}\, x)Q_R(\mathit{right}(\mathbf{expn} yy)) \supset (\mathit{right}\, x)Q_R(\mathit{right}\, y).$$

Put Φ as

$$\begin{array}{l}\Phi :\equiv \lambda x^{\Pi X.\mathsf{List}X \to R_X}\, y^{\Pi X.\mathsf{List}X \to RX}.\\ \quad (\mathit{left}\Lambda X.xX(\,))\ Q_R\ (\mathit{left}\Lambda X.yX(\,))\\ \quad \wedge\ (\mathit{right}\Lambda X.xX(\,))\ Q_R\ (\mathit{right}\Lambda X.yX(\,)).\end{array}$$

Then the above two formulae are equivalent to:

$$\forall xy.\, x\Phi(\Lambda X.\lambda z.\mathbf{expn}' yz(yXz)) \supset x\Phi y$$

Other cases listed below are also proved in the theory of parametricity

$$\begin{array}{l}\forall xy.\ x = y \supset x\Phi y,\\ \forall xy.\ y\Phi x \supset x\Phi y,\\ \forall xyz.\ x\Phi z \supset z\Phi y \supset x\Phi y,\\ \forall xy.\ x\Phi(\Lambda X.\lambda z.\mathbf{expn}' yz(xXz)) \supset x\Phi y,\\ \forall xy.\forall X.\forall z^{\mathsf{List}X}\, w^X.\ ,\\ \qquad (\mathbf{g}(xXz)w)\ \bar{F}(\mathcal{E}q_X \bar{+} \Phi[\phi, X, w, z])\ (\mathbf{g}(yXz)w) \supset \mathit{left} x\Phi y.\end{array}$$

Thus, by the inference rule of μ, we prove $\forall xy.\ xQ'y \supset x\Phi y$, which is equivalent to

$$\forall x^R y^R.\ xQ_R y \supset (\mathit{left}\ x)Q_R(\mathit{left}\ y) \wedge (\mathit{right}\ x)Q_R(\mathit{right}\ y).$$

Thus we have $\mathit{left} \in \mathcal{D}_{Q_R} \to \mathcal{D}_{Q_R}$ and $\mathit{right} \in \mathcal{D}_{Q_R} \to \mathcal{D}_{Q_R}$.

5 Conclusion

We have proposed an interpretation of cyclic structure into System F. This interpretation deals not only with objects of cyclic structure, but also with functions over cyclic structure.

For this purpose, this interpretation has two characteristic points. First, this interpretation interprets domains of cyclic structure as inductive type. Second, this interpretation distinct types of already constructed terms of cyclic structure from types of terms of cyclic structure under construction.

Acknowledgement

I am greatly indebted to Masahiko Sato for his total support for this work. I am deeply indebted to Yukiyoshi Kameyama for reading the draft of this paper. I am very indebted to Masako Hourai Takahashi, Ryu Hasegawa and Masahito Hasegawa for many previous discussions and comments. I would like to thank the anonymous referees very much for the comments. The conference version of this paper is submitted to the 3rd Fuji International Symposium on Functional and Logic Programming. I would like to thank very much also the anonymous referees of the symposium.

References

[1] Thatcher, J. W. & Wright, J. B., General finite automata theory with an application to a decision problem of second order logic, *Math. Syst. Theor.*, 2, pp57–81, (1968).

[2] Amadio, R. M. & Cardelli, L., Subtyping recursive types, *18th Annual ACM Symposium on Principle of Programming Languages*, pages 104–118 (1991), & *ACM Trans. on Prog. Lang. & Syst.*, 15(4), pp575–631, (1993).

[3] Plotkin, G. D., & Abadi, M., A Logic for Parametric Polymorphism, *Typed Lambda Calculi and Applications, 1993,* Lecture Notes in Computer Science, 664, pages 361–375 (1993).

[4] Brandt, M., & Henglein, F.: Coinductive axiomatization of recursive type equality and subtyping, *Typed Lambda Calculi and Applications, 1997,* Lecture Notes in Computer Science, 1210, pages 63–81 (1997). & *Fundamenta Informatica* , 33, pp309–338, (1998).

[5] Hasegawa, M.: Recursion from cyclic sharing: traced monoidal categories and models of cyclic lambda calculi. *Typed Lambda Calculi and Applications, 1997,* Lecture Notes in Computer Science, 1210, pages 196–213 (1997).

[6] Takeuti, I., An Axiomatic System of Parametricity *Typed Lambda Calculi and Applications, 1997,* Lecture Notes in Computer Science, 1210, pages 354–372 (1997). & *Fundamenta Informatica* , 33, pp397–432, (1998).