Hokkaido Mathematical Journal Vol. 34 (2005) p. 103–134

A hypergraph rewriting language and its semantics*

Akira Higuchi

(Received December 26, 2002; Revised August 4, 2004)

Abstract. A resource conscious functional language is introduced based a hypergraph rewriting scheme described by higher dimensional hypergraphs. Its operational and denotational semantics are given.

Key words: hypergraph rewriting, hypercategory, functional programming.

1. Introduction

1.1. Hypergraphs

Higher dimensional hypergraphs [7] are generalization of directed graphs and multigraphs. Hypergraphs allow us to use cells of more general shapes in describing processes than trees. For example, consider the rewrite

$$s(x) + y \to x + s(y)$$

which is a part of the rewriting rules of the system of natural numbers and their basic functions. The both sides of the above rule consist of 1-cells s and $+$. To represent the rewrite as a 1-pasting diagram directly, we need non-restricted forms of hypergraphs.

The data of a 2-hypergraph defines a rewriting system, where 0-cells are types, 1-cells are operators, and 2-cells are rewriting rules. Since hypergraphs allow multiple targets (or codomains), the operators in the rewriting system can have multiple inputs, and multiple outputs. Another remarkable property of the rewriting system is its *resource consciousness*, an important aspect which is shared by linear term rewriting systems and Lafont's interaction net. For example, copying must be done explicitly, by using an operator such as

---

2000 Mathematics Subject Classification : 68Q10.

*partially supported by the Japan Society for the Promotion of Science Research Fellowships from 1997 to 1999

$$copy\colon X \to X, X$$

which takes an object of type X, and returns two copies of it.

1.2. Computer programs and noncopyable objects

Resource conscious languages are known to be useful for representing computer programs. Computer programs must have ability to manipulate noncopyable objects, so that it can interact with the outside of the process. On the other hand, mathematical objects (e.g., numbers) are stored on memory when they are represented on a computer program, and therefore they are copyable because we think as if a computer has virtually infinite amount of memory. Resource conscious languages can treat both noncopyable and copyable objects uniformly, and have ability to represent common algorithms that can be applied for both sorts of objects, without losing referential transparency.

1.3. Overview

The rest of the paper is organized as follows. In section two, we review basic concepts of higher dimensional hypergraphs [7], used to formulate our rewriting systems in section three. In section four, we introduce a set theoretical semantics for 1-hypergraphs and define the concept of models of 2-hypergraphs. Finally we single out a certain class of 2-hypergraphs, called functional, and prove the main result that functional 2-hypergraphs have set theoretical models. In the last section, we give a sketch of a resource conscious language based on the class of functional 2-hypergraphs developed here.

2. Hypergraphs

Higher dimensional hypergraphs [7] is a generalization of multigraphs. In this paper, we use those of dimension less than or equal to 2. The following definition of n-hypergraphs differs from that of [7] a little, so that we can introduce set-theoretic semantics for them easily.

2.1. Preliminaries

A list over a set X is either the empty list $\langle\rangle$ or a finite ordered sequence $\langle x_1, \ldots, x_n\rangle$ where $x_i \in X$ $(i = 1, \ldots, n)$ $(n \geq 1)$.

$\mathbf{List}(X)$ denotes the set of the lists over X. **length** is the function returning the length of a list, i.e., $\mathbf{length}(\langle\rangle) = 0$ and $\mathbf{length}(\langle x_1, \ldots, x_n\rangle) = n$. We sometimes write x_i as $x(i)$ for $x = \langle x_1, \ldots, x_n\rangle$.

2.2. 1-hypergraphs

(1-*hypergraphs*) A 1-hypergraph is a quadruple $(\Sigma_0, \Sigma_1, *, \delta)$, where

- Σ_0 is a set of *0-cells*,
- Σ_1 is a set of *1-cells*,
- $*\colon \Sigma_0 \sqcup \Sigma_1 \to \Sigma_0 \sqcup \Sigma_1$ is a bijection satisfying $*(\Sigma_i) \subset \Sigma_i$ $(i = 0, 1)$ and $x^{**} = x$ $(\forall x)$,
- $\delta\colon \Sigma_1 \to \mathbf{List}(\Sigma_0)$ is a map compatible with $*$, namely $\delta(x^*) = \langle x_1^*, x_2^*, \ldots, x_n^* \rangle$ when $\delta(x) = \langle x_1, x_2, \ldots, x_n \rangle$.

(1-*pasting diagrams*) *A 1-pasting diagram* x over a 1-hypergraph $\mathcal{H} = (\Sigma_0, \Sigma_1, *, \delta)$ is a pair (δ_x, ℓ_x), where δ_x is a member of $\mathbf{List}(\Sigma_1)$, and $\ell_x\colon \mathcal{N}_x^0 \to \mathcal{N}_x^0$ is an bijection satisfying

$$\ell_x(\ell_x(i)) = i \quad (\forall i \in \mathcal{N}_x^0)$$

and

$$\ell_x(i) \neq i \Longrightarrow \lambda(x, \ell_x(i)) = \lambda(x, i)^* \quad (\forall i \in \mathcal{N}_x^0),$$

where we use the following notations. $\mathcal{N}_x^0$ denotes the set of *0-node indexes* of x. A 0-node index is a pair of natural numbers $\langle i_1, i_2 \rangle$ satisfying $1 \leq i_1 \leq \mathbf{length}(\delta_x)$ and $1 \leq i_2 \leq \mathbf{length}(\delta_x(i_1))$. We denote by $\mathcal{N}_x^1$ the set of *1-node indexes* of x. A 1-node index is a natural number i satisfying $1 \leq i \leq \mathbf{length}(\delta_x)$. We define $\lambda(x, i) = \delta_x(i)$ for $i \in \mathcal{N}_x^1$, and $\lambda(x, i) = \delta(\delta_x(i_1))(i_2)$ for $i = \langle i_1, i_2 \rangle \in \mathcal{N}_x^0$. For simplicity, we call a 1-pasting diagram as *a diagram.* A diagram x is said to *be closed* if ℓ_x has no fixed point.

We call $\mathcal{N}_x^1$ and $\mathcal{N}_x^0$ *node indexes* because a 1-pasting diagram can be represented as a labeled tree.

Fig. 1 shows the *tree representation* [7] of a diagram $x = (\delta_x, \ell_x)$ over the 1-hypergraph $\mathcal{H} = (\Sigma_0, \Sigma_1, *, \delta)$, where $\Sigma_0 = \{A, A^*, B, B^*, C, C^*\}$ with the obvious involution $*$, $\Sigma_1 = \{\mathit{foo}, \mathit{bar}\}$, $\delta(\mathit{foo}) = \langle A^*, C \rangle$, $\delta(\mathit{bar}) = \langle A, B^*, A \rangle$, $\delta_x = \langle \mathit{foo}, \mathit{bar} \rangle$, $\ell_x(\langle 1, 1 \rangle) = \langle 2, 1 \rangle$, $\ell_x(\langle 1, 2 \rangle) = \langle 1, 2 \rangle$, and $\ell_x(\langle 2, 2 \rangle) = \langle 2, 3 \rangle$.

2.3. 2-hypergraphs

(2-*Hypergraphs*) A 2-hypergraph $\mathcal{H}$ is a 6-tuple $(\Sigma_0, \Sigma_1, \Sigma_2, *, \delta, \ell)$ satisfying the following conditions.

- Σ_i is a set of *i-cells* $(i = 0, 1, 2)$.
- $*$ is an bijection satisfying $x \in \Sigma_i \Rightarrow x^* \in \Sigma_i$ $(i = 0, 1, 2)$ and $x^{**} = x$.

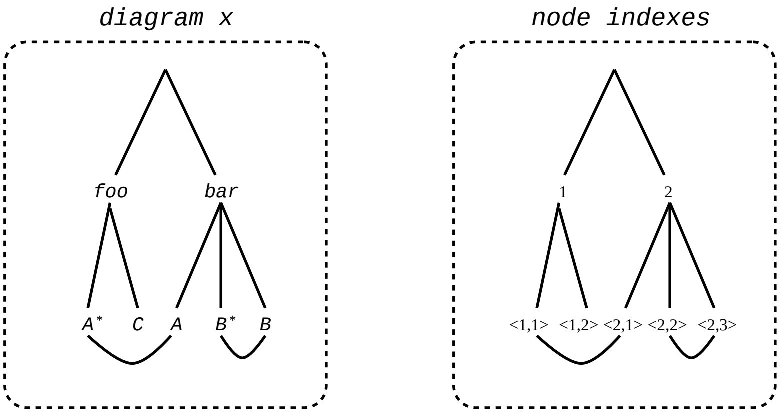

Fig. 1. tree representation of $x = (\delta_x, \ell_x)$ and its node indexes

- For $x \in \Sigma_i$ $(i = 1, 2)$, $\delta(x)$ is a finite list over the set Σ_{i-1}, and the map δ is compatible with $*$.
- For $x \in \Sigma_2$, $(\delta(x), \ell(x))$ is a closed 1-pasting diagram over $(\Sigma_0, \Sigma_1, *, \delta|_{\Sigma_1})$.

The diagram $(\delta(x), \ell(x))$ is called *the boundary* for x. We sometimes identify a 2-cell x with its diagram $(\delta(x), \ell(x))$, and write $\delta(x)$ and $\ell(x)$ by δ_x and ℓ_x respectively. This identification will not cause confusion when 2-cells with the same boundary coincide. We will hereafter assume that this condition is satisfied, so that a 2-hypergraph is just a collection of closed diagrams over a 1-hypergraph.

A 1-hypergraph $\mathcal{H} = (\Sigma_0, \Sigma_1, *, \delta)$ is said to *be with parity* if Σ_i $(i = 0, 1)$ is the disjoint union of two sets Σ_i^+ and Σ_i^- with $*(\Sigma_i^\pm) \subset \Sigma_i^\mp$. The elements of Σ_i^+ and Σ_i^- are called *positive* and *negative* respectively. A 2-hypergraph $\mathcal{H} = (\Sigma_0, \Sigma_1, \Sigma_2, *, \delta, \ell)$ is said to be with parity if the above condition holds for Σ_0, Σ_1, and Σ_2. For a 1-pasting diagram x over a 1-hypergraph $\mathcal{H} = (\Sigma_0, \Sigma_1, *, \delta)$, we use the following notations.

- $\mathcal{N}_x^{1\pm} = \left\{ i \in \mathcal{N}_x^1 \;\middle|\; \lambda(x, i) \in \Sigma_1^\pm \right\}$.
- $\mathcal{N}_x^{0\alpha\beta} = \left\{ \langle i_1, i_2 \rangle \in \mathcal{N}_x^0 \;\middle|\; i_1 \in \mathcal{N}_x^{1\alpha},\ \ell_x(\langle i_1, i_2 \rangle)(1) \in \mathcal{N}_x^{1\beta} \right\}$
  $(\alpha,\, \beta = +\ or\ -)$.

3. Rewriting logics

In this section, we define a rewriting system based on a 2-hypergraph with parity. *Rules* are the set of 2-cells of a 2-hypergraph. *Rewritings* are 1-pasting diagrams derived from the set of 2-cells. The main concern in this section is to define the rewritings derived from a given set of rules.

(*Identity diagrams*) Let $\mathcal{H} = (\Sigma_0, \Sigma_1, *, \delta)$ be a 1-hypergraph. A closed diagram $x = (\delta_x, \ell_x)$ over $\mathcal{H}$ is called *an identity diagram* if the following conditions are met.

- There is a natural number m such that $\delta_x = \langle a_1, \ldots, a_m, a_{m+1}, \ldots, a_{2m}\rangle$.
- $\lambda(x, i) \in \Sigma_1^-$ $\quad(\forall i \in \{1, \ldots, m\})$.
- $\lambda(x, m+i) = \lambda(x, i)^*$ $\quad(\forall i \in \{1, \ldots, m\})$.
- For $\langle i_1, i_2\rangle \in \mathcal{N}_x^0$ satisfying $i_1 \in \{1, \ldots, m\}$, $\ell_x(\langle i_1, i_2\rangle) = \langle i_1 + m, i_2\rangle$.

(*Disjoint sum*) Let x, y be closed diagrams over a 1-hypergraph $\mathcal{H} = (\Sigma_0, \Sigma_1, *, \delta)$, and $\delta_x = \langle a_1, \ldots, a_m\rangle$, $\delta_y = \langle b_1, \ldots, b_n\rangle$. We define a closed diagram $x \oplus y$, called *the disjoint sum of x and y*, as follows.

- $\delta_{x\oplus y} = \langle a_1, \ldots, a_m, b_1, \ldots, b_n\rangle$.
- $\ell_{x\oplus y}(\langle i_1, i_2\rangle) = \langle j_1, j_2\rangle$ holds iff $(i_1 \le m)$ and $(\ell_x(\langle i_1, i_2\rangle) = \langle j_1, j_2\rangle)$, or $(i_1 > m)$ and $(\ell_y(\langle i_1 - m, i_2\rangle) = \langle j_1 - m, j_2\rangle)$ holds.

(*Pasting maps, embeddings*) Let x, y be closed diagrams over a 1-hypergraph $\mathcal{H} = (\Sigma_0, \Sigma_1, *, \delta)$, and $\delta_x = \langle a_1, \ldots, a_m\rangle$, $\delta_y = \langle b_1, \ldots, b_n\rangle$. *A pasting map over $x \oplus y$* is a bijection $f\colon \mathcal{N}_{x\oplus y}^1 \to \mathcal{N}_{x\oplus y}^1$ satisfying the following conditions.

(PM-1) $f(f(i)) = i$ $\quad(\forall i \in \mathcal{N}_{x\oplus y}^1)$.

(PM-2) $\lambda(x \oplus y, f(i)) = \lambda(x \oplus y, i)^*$ $\quad(\forall i \in \mathcal{N}_{x\oplus y}^1)$.

(PM-3) If $i \in \mathcal{N}_{x\oplus y}^1$ and $f(i) \ne i$, then $(i \le m) \wedge (f(i) > m)$ or $(i > m) \wedge (f(i) \le m)$ holds.

(PM-4) If $\langle i_1, i_2\rangle, \langle j_1, j_2\rangle \in \mathcal{N}_{x\oplus y}^0$, $f(i_1) \ne i_1$, $f(j_1) \ne j_1$, and $\ell_{x\oplus y}(\langle i_1, i_2\rangle) = \langle j_1, j_2\rangle$, then $\ell_{x\oplus y}(\langle f(i_1), i_2\rangle) = \langle f(j_1), j_2\rangle$ holds.

A pasting map f over $x \oplus y$ is called *an embedding* if it satisfies the following conditions.

(E-1) $f(i) = i$ $\quad(\forall i \in \mathcal{N}_x^{1-})$.

(E-2) $f(m+i) = m+i$ $\quad(\forall i \in \mathcal{N}_y^{1+})$.

(E-3) $\forall i \in \mathcal{N}_x^{1+}(f(i) \ne i)$ or $\forall i \in \mathcal{N}_y^{1-}(f(m+i) \ne m+i)$ holds.

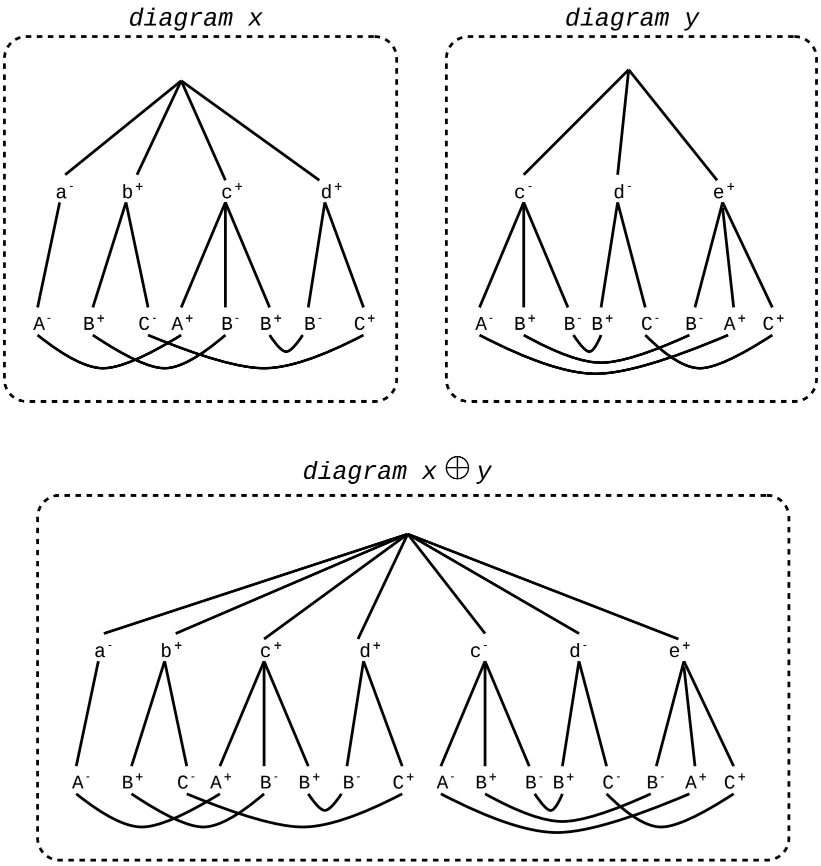

Fig. 2. disjoint sum

(*Binary composition*) Let x, y be closed diagrams over a 1-hypergraph $\mathcal{H} = (\Sigma_0, \Sigma_1, *, \delta)$, and f be a pasting map over $x \oplus y$. Let $f_0 \colon \mathcal{N}^0_{x\oplus y} \to \mathcal{N}^0_{x\oplus y}$ be the bijection defined by

$$f_0(\langle i_1, i_2\rangle) = \langle f(i_1), i_2\rangle.$$

Let $\mathbf{fix}(f_0)$ be the set of fixed points of f_0, and $\ell^f_{x\oplus y} \colon \mathbf{fix}(f_0) \to \mathbf{fix}(f_0)$ a map defined as

$$\ell^f_{x\oplus y}(i) = \begin{cases} \ell_{x\oplus y}(i) & \text{if } \ell_{x\oplus y}(i) \in \mathbf{fix}(f_0) \\ \ell^f_{x\oplus y}(f_0(\ell_{x\oplus y}(i))) & \text{otherwise} \end{cases}.$$

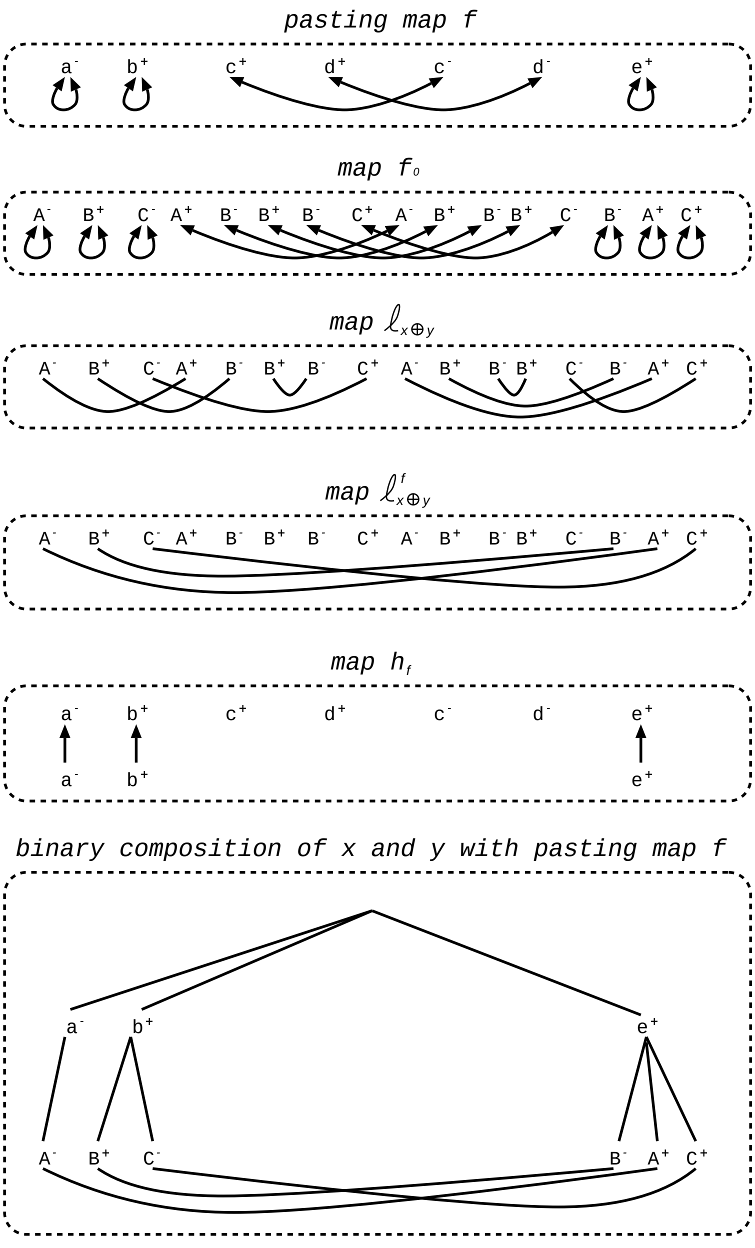

Fig. 3. binary composition

Note that $\ell^f_{x\oplus y}$ is well-defined. Let $h_f\colon \{1,\ldots,k\} \to \mathbf{fix}(f)$ be the monotone bijection, where $\mathbf{fix}(f)$ is the set of fixed points of f and $k = |\mathbf{fix}(f)|$. Then, we define the binary composition z of x and y with the pasting map f as the following closed diagram:

- $\delta_z = \langle \lambda(x \oplus y, h_f(1)), \ldots, \lambda(x \oplus y, h_f(k))\rangle$.
- $\ell_z(\langle i_1, i_2\rangle) = \langle j_1, j_2\rangle$ iff $\ell^f_{x\oplus y}(\langle h_f(i_1), i_2\rangle) = \langle h_f(j_1), j_2\rangle$.

(*Isomorphic diagrams*) Closed diagrams x and y over a 1-hypergraph $\mathcal{H}$ are said to *be isomorphic* if there is a pasting map over $x \oplus y$ which has no fixed point.

(*Rewrites*) Let $\mathcal{H}$ be a 2-hypergraph with parity. *The rewrites of* $\mathcal{H}$ are the 1-pasting diagrams defined as follows.

- The boundary of a positive 2-cell of $\mathcal{H}$ is a rewrite of $\mathcal{H}$.
- An identity diagram of $\mathcal{H}$ is a rewrite of $\mathcal{H}$.
- If x, y are rewrites of $\mathcal{H}$, and f is a embedding over $x \oplus y$, then the binary composition of x and y with pasting map f is also a rewrite of $\mathcal{H}$.
- If x is a rewrite of $\mathcal{H}$, y is a closed diagram over $\mathcal{H}$, and y is isomorphic to x, then y is also a rewrite of $\mathcal{H}$.

4. Set theoretic interpretations of 1-Hypergraphs

(*Set theoretic interpretation*) Let $\mathcal{H} = (\Sigma_0, \Sigma_1, *, \delta)$ be a 1-hypergraph with parity. A map $[\![\cdot]\!]$ which assigns a family of sets $\big\{[\![x]\!] \big| x \in \Sigma_0 \cup \Sigma_1\big\}$ is called *a set theoretic interpretation of* $\mathcal{H} = (\Sigma_0, \Sigma_1, *, \delta)$ if it satisfies $[\![x]\!] \subset [\![x_1]\!] \times \cdots \times [\![x_m]\!]$ with $\delta_x = \langle x_1, \ldots, x_m\rangle$ for $x \in \Sigma_1$, and $[\![x^*]\!] = [\![x]\!]$ for $x \in \Sigma_0 \cup \Sigma_1$.

(*Assignment*) Let $[\![\cdot]\!]$ be a set theoretic interpretation of $\mathcal{H} = (\Sigma_0, \Sigma_1, *, \delta)$, y a closed diagram over $\mathcal{H}$, and N a subset of $\mathcal{N}^0_y$. *An assignment to y over N with respect to* $[\![\cdot]\!]$ is a map $s\colon N \to \bigcup_{x\in\Sigma_0}[\![x]\!]$ satisfying $s(i) \in [\![\lambda(y, i)]\!]$ $(\forall i \in N)$, and $s(i) = s(\ell_y(i))$ if both sides are defined. $\mathbf{Asgn}(y, [\![\cdot]\!], N)$ denotes the set of assignments to y over N with respect to $[\![\cdot]\!]$.

(*Valid assignment*) Let N be a subset of $\mathcal{N}^0_y$, and M a subset of $\mathcal{N}^1_y$. An assignment s to y over N with respect to $[\![\cdot]\!]$ is said to *be valid over* M if it satisfies, for $k \in M$, $\big\langle s(\langle k, 1\rangle), \ldots, s(\langle k, m\rangle)\big\rangle \in [\![\lambda(y, k)]\!]$ if all the components are defined. $\mathbf{VAsgn}(y, [\![\cdot]\!], N, M)$ denotes the set of assignments to y over N with respect to $[\![\cdot]\!]$ which are valid over M.

(*Union of assignments*) Let N_1, $N_2 \subset \mathcal{N}_y^0$, and s_1, s_2 be assignments to y over N_1, N_2 respectively, with respect to $[\![\cdot]\!]$. Assume $s_1(i) = s_2(i)$ for $\forall i \in N_1 \cap N_2$. Then we define a map $s_1 \cup s_2$, *the union of* s_1 *and* s_2, as

$$(s_1 \cup s_2)(i) = \begin{cases} s_1(i) & \text{if } i \in N_1 \\ s_2(i) & \text{if } i \in N_2 \end{cases}.$$

Note that if both N_1 and N_2 are closed with respected to the map ℓ_y, the map $s_1 \cup s_2$ is an assignment to y over $N_1 \cup N_2$ with respect to $[\![\cdot]\!]$.

(*Interpretation satisfying an equation*) Let $[\![\cdot]\!]$ be a set theoretic interpretation of $\mathcal{H} = (\Sigma_0, \Sigma_1, *, \delta)$, y a closed diagram over $\mathcal{H}$. The interpretation $[\![\cdot]\!]$ is said to *satisfy the equation* y if, for any assignment s to y over $\mathcal{N}_y^{0-+} \cup \mathcal{N}_y^{0+-}$ with respect to $[\![\cdot]\!]$,

$$\begin{aligned} &\exists t_1 \in \mathbf{Asgn}(y, [\![\cdot]\!], \mathcal{N}_y^{0--})\Big((s \cup t_1) \in \mathbf{VAsgn}(y, [\![\cdot]\!], N_1, \mathcal{N}_y^{1-})\Big) \\ &\Longleftrightarrow \\ &\exists t_2 \in \mathbf{Asgn}(y, [\![\cdot]\!], \mathcal{N}_y^{0++})\Big((s \cup t_2) \in \mathbf{VAsgn}(y, [\![\cdot]\!], N_2, \mathcal{N}_y^{1+})\Big) \end{aligned}$$

holds, where $N_1 = \mathcal{N}_y^{0--} \cup \mathcal{N}_y^{0-+} \cup \mathcal{N}_y^{0+-}$, $N_2 = \mathcal{N}_y^{0++} \cup \mathcal{N}_y^{0-+} \cup \mathcal{N}_y^{0+-}$.

(*Models*) Let $\mathcal{H}_2 = (\Sigma_0, \Sigma_1, \Sigma_2, *, \delta, \ell)$ be a 2-hypergraph. A set theoretic interpretation $[\![\cdot]\!]$ is said to *be a model of* $\mathcal{H}_2$ if $[\![\cdot]\!]$ satisfies all the equations $\big\{(\delta_x, \ell_x) \bigm| x \in \Sigma_2\big\}$.

Lemma 1 Let x, y be closed diagrams over a 1-hypergraph $\mathcal{H} = (\Sigma_0, \Sigma_1, *, \delta)$, f an embedding over $x \oplus y$, and z the binary composition of x and y with pasting map f. If $[\![\cdot]\!]$ is a set theoretic interpretation of $\mathcal{H}$ satisfying equations x and y, then $[\![\cdot]\!]$ satisfies the equation z also.

Proof. We prove the lemma in the case that the latter condition of **(E-3)** holds. Let $M_A = \mathcal{N}_x^{1-}$, $M_B = \mathcal{N}_x^{1+} \cap \mathbf{fix}(f)$, $M_C = \mathcal{N}_x^{1+} \setminus M_B$, $M'_C = f(M_C)$, and $M_D = \big\{i + m \bigm| i \in \mathcal{N}_y^{1+}\big\}$. Because we assume that the latter condition of **(E-3)** holds, $M'_C = \big\{i + m \bigm| i \in \mathcal{N}_y^{1-}\big\}$, $M_A \subseteq \mathbf{fix}(f)$, and $M_D \subseteq \mathbf{fix}(f)$ holds (Fig. 4). Let

$$I_1 = \{\langle i_1, i_2\rangle \in \mathcal{N}_x^{0+-} \mid i_1 \in M_B\}$$

$$I_2 = \{\langle i_1, i_2\rangle \in \mathcal{N}_x^{0+-} \mid i_1 \in M_C\}$$
$$I_3 = \{\langle i_1, i_2\rangle \in \mathcal{N}_x^{0++} \mid i_1 \in M_C,\ \ell_{x\oplus y}(\langle i_1, i_2\rangle)(1) \in M_B\}$$
$$N_A = \mathcal{N}_x^{0--}$$
$$N_B = \{\langle i_1, i_2\rangle \in \mathcal{N}_x^{0++} \mid i_1 \in M_B,\ \ell_{x\oplus y}(\langle i_1, i_2\rangle)(1) \in M_B\}$$
$$N_C = \{\langle i_1, i_2\rangle \in \mathcal{N}_x^{0++} \mid i_1 \in M_C,\ \ell_{x\oplus y}(\langle i_1, i_2\rangle)(1) \in M_C\}.$$

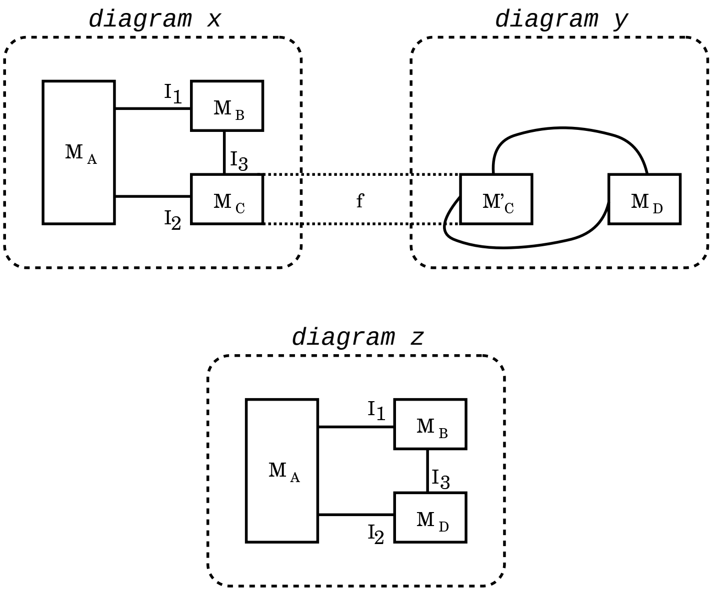

Fig. 4. binary composition, when the latter condition of **(E-3)** holds

Let P_A, P_B, P_C, P_D, and P'_C be predicates defined as follows. Let $V = \prod_{i\in I_1}[\![\lambda(x\oplus y, i)]\!]$, $W = \prod_{i\in I_2}[\![\lambda(x\oplus y, i)]\!]$, and $U = \prod_{i\in I_3}[\![\lambda(x\oplus y, i)]\!]$. For $v \in V$, $w \in W$, $u \in U$,

$$P_A(v, w)$$
$$\iff \exists t_1 \in \mathbf{Asgn}(x, [\![\cdot]\!], N_A)\big((s_1 \cup t_1) \in \mathbf{VAsgn}(x, [\![\cdot]\!], N_1, M_A)\big)$$

where $N_1 = \mathcal{N}_x^{0--} \cup \mathcal{N}_x^{0-+} \cup \mathcal{N}_x^{0+-}$, and s_1 is defined by $\langle s_1(i)\rangle_{i\in I_1} = \langle s_1(\ell_{x\oplus y}(i))\rangle_{i\in I_1} = v$, $\langle s_1(i)\rangle_{i\in I_2} = \langle s_1(\ell_{x\oplus y}(i))\rangle_{i\in I_2} = w$.

$$P_B(v, u)$$
$$\iff \exists t_2 \in \mathbf{Asgn}(x, [\![\cdot]\!], N_B)\big((s_2 \cup t_2) \in \mathbf{VAsgn}(x, [\![\cdot]\!], N_2, M_B)\big)$$

where $N_2 = N_B \cup I_1 \cup \ell_{x\oplus y}(I_1) \cup I_3 \cup \ell_{x\oplus y}(I_3)$, and s_2 is defined by $\langle s_2(i)\rangle_{i\in I_1} = \langle s_2(\ell_{x\oplus y}(i))\rangle_{i\in I_1} = v$, $\langle s_2(i)\rangle_{i\in I_3} = \langle s_2(\ell_{x\oplus y}(i))\rangle_{i\in I_3} = u$.

$$\begin{aligned} &P_C(w,u) \\ &\quad\Longleftrightarrow \exists t_3 \in \mathbf{Asgn}(x, [\![\cdot]\!], N_C)\big((s_3 \cup t_3) \in \mathbf{VAsgn}(x, [\![\cdot]\!], N_3, M_C)\big) \end{aligned}$$

where $N_3 = N_C \cup I_2 \cup \ell_{x\oplus y}(I_2) \cup I_3 \cup \ell_{x\oplus y}(I_3)$, and s_3 is defined by $\langle s_3(i)\rangle_{i\in I_2} = \langle s_3(\ell_{x\oplus y}(i))\rangle_{i\in I_2} = w$, $\langle s_3(i)\rangle_{i\in I_3} = \langle s_3(\ell_{x\oplus y}(i))\rangle_{i\in I_3} = u$.

$$\begin{aligned} &P_D(w,u) \\ &\quad\Longleftrightarrow \exists t_4 \in \mathbf{Asgn}(y, [\![\cdot]\!], \mathcal{N}_y^{0++})\big((s_4 \cup t_4) \in \mathbf{VAsgn}(y, [\![\cdot]\!], N_4, \mathcal{N}_y^{1+})\big) \end{aligned}$$

where $N_4 = \mathcal{N}_y^{0++} \cup \mathcal{N}_y^{0-+} \cup \mathcal{N}_y^{0+-}$, and s_4 is defined by
$\langle s_4(\langle f(i_1) - m, i_2\rangle)\rangle_{\langle i_1,i_2\rangle\in I_2} = \langle s_4(\ell_y(\langle f(i_1) - m, i_2\rangle))\rangle_{\langle i_1,i_2\rangle\in I_2} = w$,
$\langle s_4(\langle f(i_1) - m, i_2\rangle)\rangle_{\langle i_1,i_2\rangle\in I_3} = \langle s_4(\ell_y(\langle f(i_1) - m, i_2\rangle))\rangle_{\langle i_1,i_2\rangle\in I_3} = u$.

$$\begin{aligned} &P'_C(w,u) \\ &\quad\Longleftrightarrow \exists t_5 \in \mathbf{Asgn}(y, [\![\cdot]\!], \mathcal{N}_y^{0--})\big((s_5 \cup t_5) \in \mathbf{VAsgn}(y, [\![\cdot]\!], N_5, \mathcal{N}_y^{1-})\big) \end{aligned}$$

where $N_5 = \mathcal{N}_y^{0--} \cup \mathcal{N}_y^{0-+} \cup \mathcal{N}_y^{0+-}$, and s_5 is defined by
$\langle s_5(\langle f(i_1) - m, i_2\rangle)\rangle_{\langle i_1,i_2\rangle\in I_2} = \langle s_5(\ell_y(\langle f(i_1) - m, i_2\rangle))\rangle_{\langle i_1,i_2\rangle\in I_2} = w$,
$\langle s_5(\langle f(i_1) - m, i_2\rangle)\rangle_{\langle i_1,i_2\rangle\in I_3} = \langle s_5(\ell_y(\langle f(i_1) - m, i_2\rangle))\rangle_{\langle i_1,i_2\rangle\in I_3} = u$.

Because $[\![\cdot]\!]$ satisfies x and y,

$$\forall v \in V\ \forall w \in W\ \big(P_A(v,w) \Leftrightarrow \exists u \in U(P_B(v,u) \wedge P_C(w,u))\big)$$

and

$$\forall w \in W\ \forall u \in U\ \big(P_D(w,u) \Leftrightarrow P'_C(w,u)\big)$$

hold. Because f is an embedding, P_C is equivalent to P'_C. Therefore we have

$$\forall v \in V\ \forall w \in W\ \big(P_A(v,w) \Leftrightarrow \exists u \in U(P_B(v,u) \wedge P_D(w,u))\big),$$

which means that $[\![\cdot]\!]$ satisfies z.

Proposition 1 Let $\mathcal{H}$ be a 2-hypergraph, and $[\![\cdot]\!]$ a model of $\mathcal{H}$. Then $[\![\cdot]\!]$ satisfies the equation r for any rewrite r of $\mathcal{H}$.

5. Functional hypergraphs

In general, 2-hypergraphs can not be interpreted as systems of sets and functions. In this section, we introduce some conditions for 2-hypergraphs so that they can be interpreted as systems of sets and functions.

(*Functional hypergraphs*) A 2-hypergraph $\mathcal{H} = (\Sigma_0, \Sigma_1, \Sigma_2, *, \delta, \ell)$ is called *functional* if the following conditions are met.

(F-1) Σ_0, Σ_1, and Σ_2 are finite.

(F-2) $\mathcal{H}$ is with parity, and Σ_1^+ is the disjoint union of two sets Σ_c^+ and Σ_d^+, called positive *constructors* and *destructors*. Σ_1^- likewise, and $\Sigma_c^- = \{x^* \mid x \in \Sigma_c^+\}$, $\Sigma_d^- = \{x^* \mid x \in \Sigma_d^+\}$ hold.

(F-3) For $x \in \Sigma_c^+$, $\delta_x(1)$ is positive and $\delta_x(i)$ negative for $i \in \{2, 3, \dots\}$. When $\delta_x(1) = t$, x is called *a constructor for* t.

(F-4) For $x \in \Sigma_d^+$, $\delta_x(1)$ is negative (but $\delta_x(i)$ need not to be positive for $i \in \{2, 3, \dots\}$). When $\delta_x(1) = t$, x is called *a destructor for* t.

(F-5) If t is a positive 0-cell, x_1 a positive constructor for t, and x_2 a positive destructor for t, then there is exactly one 2-cell $y \in \Sigma_2^+$ such that $\delta_y(1) = x_1^*$, $\delta_y(2) = x_2^*$, $\delta_y(k)$ is positive for all $k \in \{3, \dots, \mathbf{length}(\delta_y)\}$, $\ell_y(\langle 1, 1\rangle) = \langle 2, 1\rangle$, and $\mathcal{N}_y^{0--} = \{\langle 1, 1\rangle, \langle 2, 1\rangle\}$.

(F-6) all the 2-cells in Σ_2^+ are of the form **(F-5)**.

(F-7) For $x \in \Sigma_2^+$, the shape graph of x has no cycle. The shape graph of x is a directed graph (E, V) defined as follows.
- V is the set of pairs $\langle i_1, i_2\rangle$ of 0-node indexes of the diagram (δ_x, ℓ_x), such that i_1 is positive, i_2 negative and $\ell_x(i_1) = i_2$.
- $\langle\langle i_1, i_2\rangle, \langle j_1, j_2\rangle\rangle \in E$ iff there is a 1-node index k such that $k = i_1(1) = j_2(1)$ and $\delta_x(k) \in \Sigma_1^-$, or $k = i_2(1) = j_1(1)$ and $\delta_x(k) \in \Sigma_1^+$.

Example 1 The following is a functional 2-hypergraph $\mathcal{H} = (\Sigma_0, \Sigma_1, \Sigma_2, *, \delta, \ell)$ which expresses the system of natural numbers. Fig. 5 is the tree representations of *zeroadd*$^+$, *succadd*$^+$, *zeroid*$^+$, and *succid*$^+$.

- $\Sigma_0^+ = \{Nat^+\}$, $\Sigma_0^- = \{Nat^-\}$.
- $\Sigma_c^+ = \{zero^+, succ^+\}$, $\Sigma_c^- = \{zero^-, succ^-\}$, $\Sigma_d^+ = \{add^+, id^+\}$, $\Sigma_d^- = \{add^-, id^-\}$.
- $\Sigma_2^+ = \{zeroadd^+, succadd^+, zeroid^+, succid^+\}$, $\Sigma_2^- = \{zeroadd^-, succadd^-, zeroid^+, succid^-\}$.

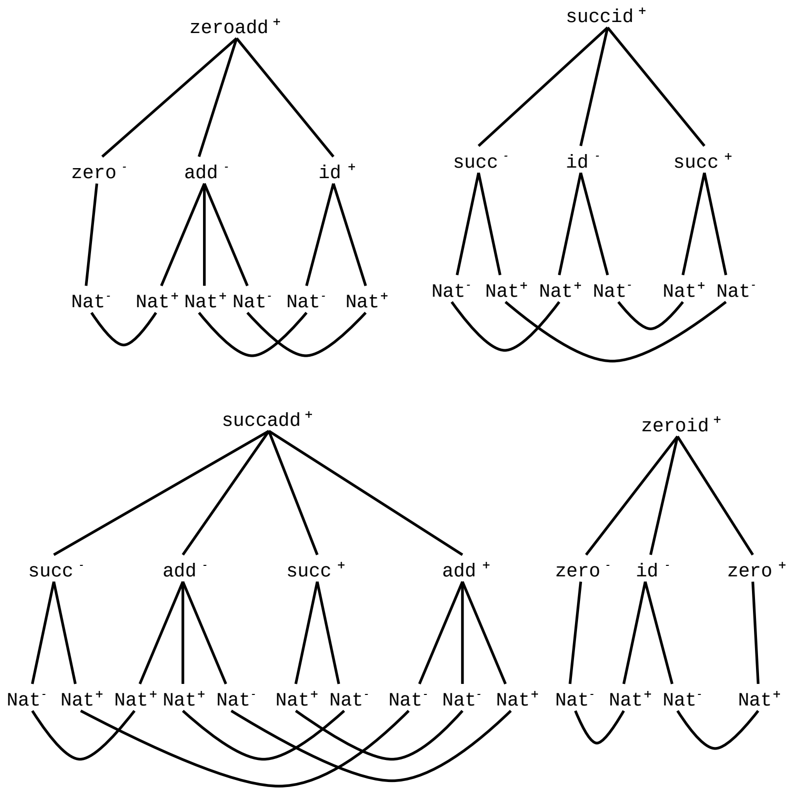

Fig. 5. $zeroadd^+$, $succadd^+$, $zeroid^+$, and $succid^+$

- $(Nat^+)^* = Nat^-$, $(zero^+)^* = zero^-$, $(succ^+)^* = succ^-$, $(add^+)^* = add^-$, $(id^+)^* = id^-$.
- $\delta_{zero^+} = \langle Nat^+\rangle$, $\delta_{succ^+} = \langle Nat^+, Nat^-\rangle$, $\delta_{add^+} = \langle Nat^-, Nat^-, Nat^+\rangle$, $\delta_{id^+} = \langle Nat^-, Nat^+\rangle$.
- $\delta_{zeroadd^+} = \langle zero^-, add^-, id^+\rangle$, $\delta_{succadd^+} = \langle succ^-, add^-, succ^+, add^+\rangle$, $\delta_{zeroid^+} = \langle zero^-, id^-, zero^+\rangle$, $\delta_{succid^+} = \langle succ^-, id^-, succ^+\rangle$.
- $\ell_{zeroadd^+} = \begin{cases} \langle 1,1\rangle \leftrightarrow \langle 2,1\rangle \\ \langle 2,2\rangle \leftrightarrow \langle 3,1\rangle \\ \langle 2,3\rangle \leftrightarrow \langle 3,2\rangle \end{cases}$

- $\ell_{succadd^+} = \begin{cases} \langle 1,1\rangle \leftrightarrow \langle 2,1\rangle \\ \langle 1,2\rangle \leftrightarrow \langle 4,1\rangle \\ \langle 2,2\rangle \leftrightarrow \langle 3,2\rangle \\ \langle 2,3\rangle \leftrightarrow \langle 4,3\rangle \\ \langle 3,1\rangle \leftrightarrow \langle 4,2\rangle \end{cases}$
- $\ell_{zeroid^+} = \begin{cases} \langle 1,1\rangle \leftrightarrow \langle 2,1\rangle \\ \langle 2,2\rangle \leftrightarrow \langle 3,1\rangle \end{cases}$
- $\ell_{succid^+} = \begin{cases} \langle 1,1\rangle \leftrightarrow \langle 2,1\rangle \\ \langle 1,2\rangle \leftrightarrow \langle 3,2\rangle \\ \langle 2,2\rangle \leftrightarrow \langle 3,1\rangle \end{cases}$

6. Models of functional hypergraphs

In this section, we give a set-theoretic model $\mathcal{D}[\![\cdot]\!]$ of a functional 2-hypergraph $\mathcal{H} = (\Sigma_0, \Sigma_1, \Sigma_2, *, \delta, \ell)$. At first, we define an interpretation $\mathcal{T}[\![\cdot]\!]$ for 0-cells and constructors, so that $(\mathcal{T}[\![\Sigma_0^+]\!], \mathcal{T}[\![\Sigma_c^+]\!])$ forms a term algebra. $\mathcal{D}[\![\cdot]\!]$ is defined as an extension to $\mathcal{T}[\![\cdot]\!]$, and defined for destructors also.

(*Definition of* $\mathcal{T}[\![\cdot]\!]$ *for* 0-*cells*) We define a sequence of interpretations $\mathcal{T}[\![\cdot]\!]_k$. $\mathcal{T}[\![x]\!]_0 = \emptyset$ for all $x \in \Sigma_0$. For $x \in \Sigma_0$, $\mathcal{T}[\![x]\!]_{k+1}$ is defined by

$$\begin{aligned} \mathcal{T}[\![x]\!]_{k+1} = \big\{ \langle c_i, v_1, \ldots, v_{m(i)} \rangle \bigm| & \; i \in \{1, \ldots, n\}, \\ & v_j \in \mathcal{T}[\![(y_i^j)^*]\!]_k \quad (\forall j \in \{1, \ldots, m(i)\}) \big\} \end{aligned}$$

where $c_1, \ldots, c_n$ are the constructors for x, and the natural number $m(i)$ and the sequence $\{y_i^j\}$ are defined by $\langle x, y_i^1, \ldots, y_i^{m(i)} \rangle = \delta(c_i)$ for $i \in \{1, \ldots, n\}$. When $m(i) = 0$, the list $\langle c_i, v_1, \ldots, v_{m(i)} \rangle$ means $\langle c_i \rangle$, and the condition $v_j \in \mathcal{T}[\![(y_i^j)^*]\!]_k$ ($\forall j \in \{1, \ldots, m(i)\}$) is always true. The interpretation $\mathcal{T}[\![x]\!]$ is defined by

$$\mathcal{T}[\![x]\!] = \bigcup_{k=0}^{\infty} \mathcal{T}[\![x]\!]_k$$

for all $x \in \Sigma_0^+$.

(*Definition of $\mathcal{T}[\![\cdot]\!]$ for constructors*) For $z \in \Sigma_c^+$, $\mathcal{T}[\![z]\!]$ is defined by

$$\mathcal{T}[\![z]\!] = \big\{ \langle \langle z, v_1, \ldots, v_m \rangle, v_1, \ldots, v_m \rangle \;\big|\; v_i \in [\![y_i]\!] \ (\forall i \in \{1, \ldots, m\}) \big\}$$

where m and the sequence $\{y_i\}$ are defined by $\langle x, y_1, \ldots, y_m \rangle = \delta(z)$.

Example 2 Suppose $\mathcal{H} = (\Sigma_0, \Sigma_1, \Sigma_2, *, \delta, \ell)$ is defined as the same as the previous example. The 0-cell Nat^+ is interpreted by $\mathcal{T}[\![\cdot]\!]_k$ $(k = 0, 1, 2, 3, \ldots)$ as

$$\begin{aligned}
\mathcal{T}[\![Nat^+]\!]_0 &= \emptyset \\
\mathcal{T}[\![Nat^+]\!]_1 &= \{\langle zero^+ \rangle\} \cup \{\langle succ^+, y \rangle \mid y \in \mathcal{T}[\![Nat^+]\!]_0\} \\
&= \{\langle zero^+ \rangle\} \\
\mathcal{T}[\![Nat^+]\!]_2 &= \{\langle zero^+ \rangle\} \cup \{\langle succ^+, y \rangle \mid y \in \mathcal{T}[\![Nat^+]\!]_1\} \\
&= \{\langle zero^+ \rangle,\ \langle succ^+, \langle zero^+ \rangle \rangle\} \\
\mathcal{T}[\![Nat^+]\!]_3 &= \{\langle zero^+ \rangle\} \cup \{\langle succ^+, y \rangle \mid y \in \mathcal{T}[\![Nat^+]\!]_2\} \\
&= \{\langle zero^+ \rangle,\ \langle succ^+, \langle zero^+ \rangle \rangle,\ \langle succ^+, \langle succ^+, \langle zero^+ \rangle \rangle \rangle\} \\
&\ \ \vdots
\end{aligned}$$

and so on, therefore

$$\begin{aligned}
\mathcal{T}[\![Nat^+]\!] &= \bigcup_{k=0}^{\infty} \mathcal{T}[\![Nat^+]\!]_k \\
&= \{\langle zero^+ \rangle,\ \langle succ^+, \langle zero^+ \rangle \rangle,\ \langle succ^+, \langle succ^+, \langle zero^+ \rangle \rangle \rangle, \ldots\}
\end{aligned}$$

holds. For the constructors $zero^+$ and $succ^+$,

$$\begin{aligned}
\mathcal{T}[\![zero^+]\!] &= \{\langle \langle zero^+ \rangle \rangle\} \\
\mathcal{T}[\![succ^+]\!] &= \{\langle \langle succ^+, v \rangle, v \rangle \mid v \in \mathcal{T}[\![Nat^+]\!]\}
\end{aligned}$$

holds. As a result, we have

$$\begin{aligned}
\mathcal{T}[\![Nat^+]\!] &= \overline{Nat} \\
\mathcal{T}[\![zero^+]\!] &= \{\langle \overline{zero} \rangle\} \\
\mathcal{T}[\![succ^+]\!] &= \big\{ \langle \overline{succ}(v), v \rangle \;\big|\; v \in \overline{Nat} \big\}
\end{aligned}$$

where $\overline{Nat} = \big\{ (\overbrace{\overline{succ} \circ \cdots \circ \overline{succ}}^{k})(\overline{zero}) \;\big|\; k \in \{0, 1, \ldots\} \big\}$, $\overline{zero} = \langle zero^+ \rangle$ and $\overline{succ}(x) = \langle succ^+, x \rangle$.

Proposition 2 For all $x \in \Sigma_c^+$, the relation $\mathcal{T}[\![x]\!]$ is a function from $\prod_{i \in I_x^-} \mathcal{T}[\![\delta_x(i)]\!]$ to $\prod_{i \in I_x^+} \mathcal{T}[\![\delta_x(i)]\!]$, where $I_x^\pm = \{ i \mid \delta_x(i) \in \Sigma_0^\pm \}$.

Proposition 3 $\left(\mathcal{T}[\![\Sigma_0^+]\!], \mathcal{T}[\![\Sigma_c^+]\!]\right)$ is a term algebra.

(*Definition of* $\mathcal{D}[\![\cdot]\!]$) We define a sequence of interpretations $\mathcal{D}[\![\cdot]\!]_k$ as

$$\mathcal{D}[\![x]\!]_0 = \begin{cases} \mathcal{T}[\![x]\!] & \text{if } x \in \Sigma_0^+ \cup \Sigma_c^+ \\ \mathcal{T}[\![x^*]\!] & \text{if } x \in \Sigma_0^- \cup \Sigma_c^- \\ \emptyset & \text{if } x \in \Sigma_d \end{cases}$$

and

$$\mathcal{D}[\![x]\!]_{k+1} = \begin{cases} \mathcal{D}[\![x]\!]_k & \text{if } x \in \Sigma_0 \cup \Sigma_c \\ \left\{ \left\langle s(\langle 2,1\rangle), \dots, s(\langle 2,m\rangle) \right\rangle \,\middle|\, y \in \Sigma_2,\ \delta_y(2) = x,\ s \in V_y^k \right\} & \\ & \text{if } x \in \Sigma_d \end{cases}$$

where $m = \mathbf{length}(\delta_x)$ and $V_y^k = \mathbf{VAsgn}(y, \mathcal{D}[\![\cdot]\!]_k, \mathcal{N}_y^0, \mathcal{N}_y^1 \setminus \{2\})$. $\mathcal{D}[\![\cdot]\!]$ is defined by

$$\mathcal{D}[\![x]\!] = \bigcup_{k=0}^{\infty} \mathcal{D}[\![x]\!]_k$$

for all $x \in \Sigma_0 \cup \Sigma_1$.

Example 3 Suppose $\mathcal{H} = (\Sigma_0, \Sigma_1, \Sigma_2, *, \delta, \ell)$ is defined as in the examples 1 and 2. By definition, we have

$$\begin{aligned} &\mathcal{D}[\![Nat^+]\!]_k = \mathcal{D}[\![Nat^-]\!]_k = \mathcal{T}[\![Nat^+]\!] = \overline{Nat} \\ &\mathcal{D}[\![zero^+]\!]_k = \mathcal{D}[\![zero^-]\!]_k = \mathcal{T}[\![zero^+]\!] = \{\langle \overline{zero} \rangle\} \\ &\mathcal{D}[\![succ^+]\!]_k = \mathcal{D}[\![succ^-]\!]_k = \mathcal{T}[\![succ^+]\!] = \left\{ \langle \overline{succ}(v), v \rangle \,\middle|\, v \in \mathcal{T}[\![Nat^+]\!] \right\} \end{aligned}$$

for all $k \in \{0, 1, 2, \dots\}$. By definition, $\mathcal{D}[\![id^+]\!]_0 = \emptyset$, and

$$\begin{aligned} \mathcal{D}[\![id^+]\!]_{k+1} = &\left\{ \langle s_1(\langle 2,1\rangle), s_1(\langle 2,2\rangle) \rangle \,\middle|\, s_1 \in V_{zeroid^+}^k \right\} \\ &\cup \left\{ \langle s_2(\langle 2,1\rangle), s_2(\langle 2,2\rangle) \rangle \,\middle|\, s_2 \in V_{succid^+}^k \right\} \end{aligned}$$

where $V_y^k = \mathbf{VAsgn}(y, \mathcal{D}[\![\cdot]\!]_k, \mathcal{N}_y^0, \mathcal{N}_y^1 \setminus \{2\})$. Because s_1 is an assignment to *zeroid*$^+$ over $\mathcal{N}_{zeroid^+}^0$, $s_1(\langle 1,1\rangle) = s_1(\langle 2,1\rangle)$ and $s_1(\langle 2,2\rangle) = s_1(\langle 3,1\rangle)$ hold. Because s_1 is valid over $\{1,3\}$, we have $s_1(\langle 1,1\rangle) \in \mathcal{D}[\![zero^-]\!]_k$ and

$s_1(\langle 3,1\rangle) \in \mathcal{D}[\![zero^+]\!]_k$. Therefore

$$\left\{\langle s_1(\langle 2,1\rangle), s_1(\langle 2,2\rangle)\rangle \;\middle|\; s_1 \in V^k_{zeroid^+}\right\} = \left\{\langle \overline{zero}, \overline{zero}\rangle\right\}.$$

Similarly, we have $s_2(\langle 1,1\rangle) = s_2(\langle 2,1\rangle)$, $s_2(\langle 1,2\rangle) = s_2(\langle 3,2\rangle)$, $s_2(\langle 2,2\rangle) = s_2(\langle 3,1\rangle)$,

$$\langle s_2(\langle 1,1\rangle), s_2(\langle 1,2\rangle)\rangle \in \left\{\langle \overline{succ}(v), v\rangle \;\middle|\; v \in \mathcal{D}[\![Nat^+]\!]\right\},$$

and

$$\langle s_2(\langle 3,1\rangle), s_2(\langle 3,2\rangle)\rangle \in \left\{\langle \overline{succ}(v), v\rangle \;\middle|\; v \in \mathcal{D}[\![Nat^+]\!]\right\},$$

whence

$$\begin{aligned}&\left\{\langle s_2(\langle 2,1\rangle), s_2(\langle 2,2\rangle)\rangle \;\middle|\; s_2 \in V^k_{succid^+}\right\}\\ &\quad = \left\{\langle \overline{succ}(v), \overline{succ}(v)\rangle \;\middle|\; v \in \overline{Nat}\right\}.\end{aligned}$$

We have

$$\mathcal{D}[\![id^+]\!]_{k+1} = \left\{\langle \overline{zero}, \overline{zero}\rangle\right\} \cup \left\{\langle \overline{succ}(v), \overline{succ}(v)\rangle \;\middle|\; v \in \mathcal{D}[\![Nat^+]\!]\right\}$$

for all $k \in \{0, 1, 2, \dots\}$, which implies

$$\mathcal{D}[\![id^+]\!] = \left\{\langle v, v\rangle \;\middle|\; v \in \overline{Nat}\right\}.$$

The interpretation for add^+ is defined similarly as follows. By definition, $\mathcal{D}[\![add^+]\!]_0 = \emptyset$, and

$$\begin{aligned}\mathcal{D}[\![add^+]\!]_{k+1} = &\left\{\left\langle s_3(\langle 2,1\rangle), s_3(\langle 2,2\rangle), s_3(\langle 2,3\rangle)\right\rangle \;\middle|\; s_3 \in V^k_{zeroadd^+}\right\}\\ &\cup \left\{\left\langle s_4(\langle 2,1\rangle), s_4(\langle 2,2\rangle), s_4(\langle 2,3\rangle)\right\rangle \;\middle|\; s_4 \in V^k_{succadd^+}\right\}\end{aligned}$$

where $V^k_y = \mathbf{VAsgn}(y, \mathcal{D}[\![\cdot]\!]_k, \mathcal{N}^0_y, \mathcal{N}^1_y \setminus \{2\})$. We have $s_3(\langle 1,1\rangle) = s_3(\langle 2,1\rangle)$, $s_3(\langle 2,2\rangle) = s_3(\langle 3,1\rangle)$, $s_3(\langle 2,3\rangle) = s_3(\langle 3,2\rangle)$,

$$\langle s_3(\langle 1,1\rangle)\rangle \in \mathcal{D}[\![zero^+]\!]_k = \left\{\langle \overline{zero}\rangle\right\},$$

and

$$\langle s_3(\langle 3,1\rangle), s_3(\langle 3,2\rangle)\rangle \in \mathcal{D}[\![id^+]\!]_k = \left\{\langle v, v\rangle \;\middle|\; v \in \overline{Nat}\right\},$$

whence

$$\begin{aligned}&\left\{\left\langle s_3(\langle 2,1\rangle), s_3(\langle 2,2\rangle), s_3(\langle 2,3\rangle)\right\rangle \;\middle|\; s_3 \in V^k_{zeroadd^+}\right\}\\ &\quad = \left\{\langle \overline{zero}, v, v\rangle \;\middle|\; v \in \overline{Nat}\right\}.\end{aligned}$$

Similarly, we have

$$\begin{aligned}&s_4(\langle 1,1\rangle)=s_4(\langle 2,1\rangle),\ s_4(\langle 1,2\rangle)=s_4(\langle 4,1\rangle),\\&s_4(\langle 2,2\rangle)=s_4(\langle 3,2\rangle),\ s_4(\langle 2,3\rangle)=s_4(\langle 4,3\rangle),\ s_4(\langle 3,1\rangle)=s_4(\langle 4,2\rangle),\\&\langle s_4(\langle 1,1\rangle),s_4(\langle 1,2\rangle)\rangle\in\big\{\langle\overline{succ}(v),v\rangle\ \big|\ v\in\overline{Nat}\big\},\\&\langle s_4(\langle 3,1\rangle),s_4(\langle 3,2\rangle)\rangle\in\big\{\langle\overline{succ}(v),v\rangle\ \big|\ v\in\overline{Nat}\big\},\end{aligned}$$

and

$$\big\langle s_4(\langle 4,1\rangle),s_4(\langle 4,2\rangle),s_4(\langle 4,3\rangle)\big\rangle\in\mathcal{D}[\![add^+]\!]_k,$$

whence

$$\begin{aligned}&\big\{\big\langle s_4(\langle 2,1\rangle),s_4(\langle 2,2\rangle),s_4(\langle 2,3\rangle)\big\rangle\ \big|\ s_4\in V^k_{succadd^+}\big\}\\&\quad=\big\{\langle\overline{succ}(v_1),v_2,v_3\rangle\ \big|\ \langle v_1,\overline{succ}(v_2),v_3\rangle\in\mathcal{D}[\![add^+]\!]_k\big\}.\end{aligned}$$

We have

$$\begin{aligned}\mathcal{D}[\![add^+]\!]_{k+1}&=\big\{\langle\overline{zero},v,v\rangle\ \big|\ v\in\overline{Nat}\big\}\\&\qquad\cup\big\{\langle\overline{succ}(v_1),v_2,v_3\rangle\ \big|\ \langle v_1,\overline{succ}(v_2),v_3\rangle\in\mathcal{D}[\![add^+]\!]_k\big\}\\&=\big\{\langle\overline{succ}^i(\overline{zero}),v,\overline{succ}^i(v)\rangle\ \big|\ i\in\{0,1,\dots,k\},\ v\in\overline{Nat}\big\}\end{aligned}$$

where $\overline{succ}^i=\overbrace{\overline{succ}\circ\cdots\circ\overline{succ}}^{i}$, which implies

$$\mathcal{D}[\![add^+]\!]=\big\{\langle\overline{succ}^i(\overline{zero}),v,\overline{succ}^i(v)\rangle\ \big|\ i\in\{0,1,\dots\},\ v\in\overline{Nat}\big\}.$$

Theorem 1 The interpretation $\mathcal{D}[\![\cdot]\!]$ is a model of the 2-hypergraph $\mathcal{H}$.

Proof. Let $y\in\Sigma_2^+$. We show that $\mathcal{D}[\![\cdot]\!]$ satisfies the equation y. Let s be an assignment to y over $\mathcal{N}_y^{0+-}\cup\mathcal{N}_y^{0-+}$ with respect to $\mathcal{D}[\![\cdot]\!]$.

Suppose $t_1\in\mathbf{Asgn}(y,\mathcal{D}[\![\cdot]\!],\mathcal{N}_y^{0--})$ satisfies

$$s\cup t_1\in\mathbf{VAsgn}(y,\mathcal{D}[\![\cdot]\!],N_1,\mathcal{N}_y^{1-})$$

where $N_1=\mathcal{N}_y^{0--}\cup\mathcal{N}_y^{0-+}\cup\mathcal{N}_y^{0+-}$. Let $x=\delta_y(2)$. Because t_1 is valid over $\{2\}$,

$$\big\langle t_1(\langle 2,1\rangle),\dots,t_1(\langle 2,m\rangle)\big\rangle\in\mathcal{D}[\![x]\!]$$

holds where $m=\mathbf{length}(\delta_x)$. There is a natural number $k\geq 1$ such that

$$\big\langle t_1(\langle 2,1\rangle),\dots,t_1(\langle 2,m\rangle)\big\rangle\in\mathcal{D}[\![x]\!]_k.$$

By the definition of $\mathcal{D}[\![\cdot]\!]_k$, there are $z \in \Sigma_2^+$ and

$$t_2 \in \mathbf{VAsgn}(z, \mathcal{D}[\![\cdot]\!]_{k-1}, \mathcal{N}_z^0, \mathcal{N}_z^1 \setminus \{2\})$$

such that $\delta_z(2) = x$ and $t_2(\langle 2, i\rangle) = t_1(\langle 2, i\rangle)$ for $1 \leq i \leq m$. By the definition of $\mathcal{D}[\![\cdot]\!]$,

$$t_2 \in \mathbf{VAsgn}(z, \mathcal{D}[\![\cdot]\!], \mathcal{N}_z^0, \mathcal{N}_z^1 \setminus \{2\})$$

holds. Because

$$t_2(\langle 1, 1\rangle) = t_2(\langle 2, 1\rangle) = t_1(\langle 2, 1\rangle) = t_1(\langle 1, 1\rangle)$$

and $\delta_y(1)$ is a constructor, we have $\delta_y(1) = \delta_z(1)$, which implies $y = z$ by **(F-5)**. Moreover, we have $t_2(\langle 1, i\rangle) = t_1(\langle 1, i\rangle)$ for $i \geq 2$ by Proposition 3. Because $t_2(j) = t_1(j)$ for $j \in \mathcal{N}_y^{0-+}$ and t_1 is an assignment over $\mathcal{N}_y^0$, $s(j) = t_2(j)$ holds for $j \in \mathcal{N}_y^{0-+} \cup \mathcal{N}_y^{0+-}$. Let $t_2' = t_2|_{\mathcal{N}_y^{0++}}$. We have $t_2' \in \mathbf{Asgn}(y, \mathcal{D}[\![\cdot]\!], \mathcal{N}_y^{0++})$ and $s \cup t_2' \in \mathbf{VAsgn}(y, \mathcal{D}[\![\cdot]\!], N_2, \mathcal{N}_y^{1+})$, where $N_2 = \mathcal{N}_y^{0++} \cup \mathcal{N}_y^{0-+} \cup \mathcal{N}_y^{0+-}$.

Conversely suppose $t_2 \in \mathbf{Asgn}(y, \mathcal{D}[\![\cdot]\!], \mathcal{N}_y^{0++})$ satisfies

$$s \cup t_2 \in \mathbf{VAsgn}(y, \mathcal{D}[\![\cdot]\!], N_2, \mathcal{N}_y^{1+})$$

where $N_2 = \mathcal{N}_y^{0++} \cup \mathcal{N}_y^{0-+} \cup \mathcal{N}_y^{0+-}$. There is a natural number $k \geq 1$ such that

$$s \cup t_2 \in \mathbf{VAsgn}(y, \mathcal{D}[\![\cdot]\!]_k, N_2, \mathcal{N}_y^{1+}).$$

Let t_1 be an assignment defined by

$$t_1(\langle 1, 1\rangle) = t_1(\langle 2, 1\rangle) = \big\langle \delta_y(1), s(\langle 1, 2\rangle), \ldots, s(\langle 1, m\rangle)\big\rangle$$

where $m = \mathbf{length}(\delta_z)$, $z = \delta_y(1)$. $s \cup t_1$ is a valid assignment over $\{1\}$ with respect to $\mathcal{D}[\![\cdot]\!]$. By the definition of $\mathcal{D}[\![\cdot]\!]_{k+1}$, $s \cup t_1$ is a valid assignment over $\{2\}$ with respect to $\mathcal{D}[\![\cdot]\!]_{k+1}$. Therefore we have $s \cup t_1 \in \mathbf{VAsgn}(y, \mathcal{D}[\![\cdot]\!], N_1, \mathcal{N}_y^{1-})$ where $N_1 = \mathcal{N}_y^{0--} \cup \mathcal{N}_y^{0-+} \cup \mathcal{N}_y^{0+-}$.

Lemma 2 For all $x \in \Sigma_d^+$ and $k \geq 0$, the relation $\mathcal{D}[\![x]\!]_k$ is a function from $\prod_{i \in I_x^-} \mathcal{D}[\![\delta_x(i)]\!]$ to $\prod_{i \in I_x^+} \mathcal{D}[\![\delta_x(i)]\!]$, where $I_x^\pm = \big\{i \bigm| \delta_x(i) \in \Sigma_0^\pm\big\}$.

Proof. By induction. When $k = 0$, $\mathcal{D}[\![x]\!]_k = \emptyset$. Suppose $k \geq 1$. Let $p, q \in \mathcal{D}[\![x]\!]_k$, $p = \langle p_1, \ldots, p_m\rangle$, $q = \langle q_1, \ldots, q_m\rangle$, where $m = \mathbf{length}(\delta_x)$. Suppose $p_i = q_i$ for $i \in I_x^-$. We show that $p_i = q_i$ for $i \in I_x^+$. Since $p, q \in \mathcal{D}[\![x]\!]_k$, there are $y_1, y_2 \in \Sigma_2^+$,

$$s_1 \in \mathbf{VAsgn}(y_1, \mathcal{D}[\![\cdot]\!]_{k-1}, \mathcal{N}_{y_1}^0, \mathcal{N}_{y_1}^1 \setminus \{2\}),$$

and

$$s_2 \in \mathbf{VAsgn}(y_2, \mathcal{D}[\![\cdot]\!]_{k-1}, \mathcal{N}_{y_2}^0, \mathcal{N}_{y_2}^1 \setminus \{2\})$$

satisfying

$$\delta_{y_1}(2) = x, \quad \delta_{y_2}(2) = x, \quad s_1(\langle 2, i\rangle) = p_i, \quad s_2(\langle 2, i\rangle) = q_i$$

for $1 \le i \le m$. Because $\left(\mathcal{T}[\![\Sigma_0^+]\!], \mathcal{T}[\![\Sigma_c^+]\!]\right)$ is a term algebra and $p_1 = q_1$ holds, we have $\delta_{y_1}(1) = \delta_{y_2}(1)$. By **(F-5)** and **(F-6)**, we have $y_1 = y_2$. From $s_1(\langle 2, 1\rangle) = s_2(\langle 2, 1\rangle)$ it follows $s_1(\langle 1, 1\rangle) = s_2(\langle 1, 1\rangle)$. Because $\mathcal{D}[\![\delta_{y_1}(1)]\!]$ is an injection by Proposition 3, we have $s_1(\langle 1, i\rangle) = s_2(\langle 1, i\rangle)$ for $2 \le i \le \mathbf{length}(\delta_{y_1})$. Let $N_1 = \{\ell_{y_1}(j) \mid j \in J_x^-\}$ and $N_2 = \{\ell_{y_1}(\langle 2, i\rangle) \mid i \in I_x^+\}$ where

$$J_x^- = \left\{\langle 1, i\rangle \;\middle|\; 2 \le i \le \mathbf{length}(\delta_{y_1})\right\} \cup \left\{\langle 2, i\rangle \mid i \in I_x^-\right\}.$$

Note that $\mathcal{N}_{y_1}^{0+-} = N_1 \cup N_2$ holds. By the induction hypothesis and **(F-7)**,

$$s_1(j) = s_2(j) \quad (\forall j \in N_1) \Longrightarrow s_1(j) = s_2(j) \quad (\forall j \in N_2)$$

holds. We already have $s_1(\langle 1, i\rangle) = s_2(\langle 1, i\rangle)$ for $2 \le i \le \mathbf{length}(\delta_{y_1})$ and $s_1(\langle 2, i\rangle) = s_2(\langle 2, i\rangle)$ for $i \in I_x^-$. Therefore we have $s_1(j) = s_2(j)$ for $j \in N_2$, which implies $p_i = q_i$ for $i \in I_x^+$.

Theorem 2 For all $x \in \Sigma_1^+$, the relation $\mathcal{D}[\![x]\!]$ is a function from $\prod_{i \in I_x^-} \mathcal{D}[\![\delta_x(i)]\!]$ to $\prod_{i \in I_x^+} \mathcal{D}[\![\delta_x(i)]\!]$, where $I_x^\pm = \left\{i \;\middle|\; \delta_x(i) \in \Sigma_0^\pm\right\}$.

7. Sample language

In this section, we show how a 2-hypergraph can represent interactions between noncopyable objects, using a realistic example. We define a simple language whose syntax elements correspond to 0, 1, and 2-cells of a functional 2-hypergraph.

7.1. Syntax summary

7.1.1. Types

A type definition has the following syntax. The following code defines a type *Nat* whose constructors are *zero* and *succ*.

```
.datatype Nat: zero, succ;
zero : -> Nat();
succ : Nat() -> Nat();
```

7.1.2. Type-parameterized types

A type can have type parameters. The following code defines a type *List* which has a type parameter t.

```
.datatype List : nil, cons;
nil : -> List(t);
cons : t, List(t) -> List(t);
```

7.1.3. Rewriting rules

Rewriting rules are defined with the following syntax.

```
add (x, y -> z)
{
    zero() = x;
    z = y;
|
    succ(xp) = x;
    ys = succ(y);
    z = add(xp, ys);
}
```

A function defined in this way corresponds to a destructor of a functional 2-hypergraph. The above code defines two unnamed rewriting rules. The former block defines $add(zero(), y) \to y$, and the latter defines $add(succ(xp), y) \to add(xp, succ(y))$. Variables such as x, y, z, xp, and ys correspond to pairs of node indexes connected by the bijection (Fig. 6). The first line of each block must be of the form

```
constructor_name(variable) = variable
```

where the right hand side is the first argument for *add*, so that the 2-hypergraph becomes functional. The expression `z = y` is a short for `z = id(y)` where id is the identity function.

7.1.4. Compositions

A function can be defined by a composition of functions. The following code defines a function *baz* by a composition of *foo* and *bar*.

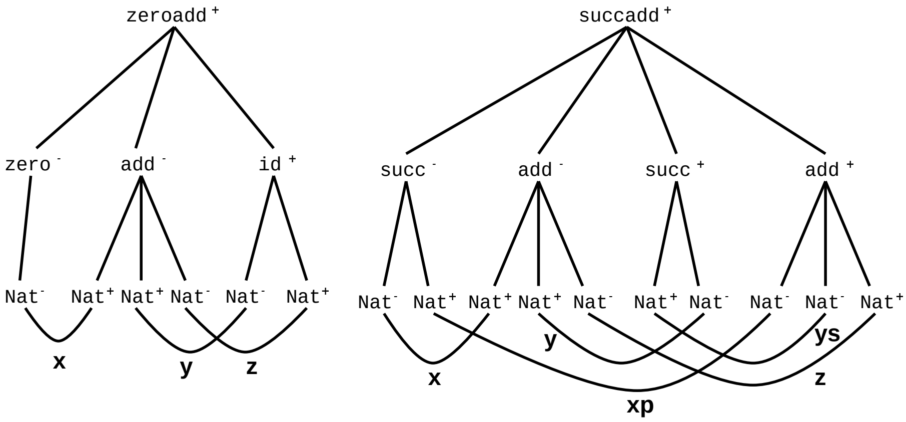

Fig. 6. variables correspond to connections between node indexes

```
baz(x, y -> z)
{
    w = foo(x);
    z = bar(w, y);
}
```

A function definition in this way does not correspond to a rewriting rule of a functional 2-hypergraph. Instead, in this example we think that such a function is just a macro expands to a composition of functions.

7.1.5. External abstract types and external functions

External abstract types and external functions can be used for importing types and functions without knowing their details.

```
.external_abstract_type Sys;
.external_abstract_type RFileDescriptor;
.external open_rfile;
open_rfile : Sys(), String()
    -> Sys(), Maybe(RFileDescriptor());
```

The above code declares two external abstract types *Sys* and *RFileDescriptor*, and an external function *open_rfile*. Because their details are defined by the operating system, we have no idea of what are constructors of *Sys* and *RFileDescriptor*, whether *open_rfile* is a constructor or not, and what are rewriting rules related to *open_rfile*.

7.1.6. Generic functions and type classes

The following code declares two *generic functions* ~ and !. The function ~ takes an object of type t and returns nothing. That is, the function ~ deletes an object of type t. Likewise, the function ! copies an object of type t.

```
~ : t -> : ~[t];
! : t -> t, t : ![t];
```

For each generic function g, there is a type class $g[t_1, \ldots, t_n]$ where $t_1, \ldots, t_n$ are the type variables occurred in the boundary of g. In the above code, there are two type classes $\tilde{}[t]$ of *deletable* types and $![t]$ of *copyable* types. An n-tuple $\langle T_1, \ldots, T_n \rangle$ of types belongs to $g[t_1, \ldots, t_n]$ iff there is an *instance* f of g such that the boundary of f matches the boundary of g. If a function f is an instance of a generic function g and g is called (invoked) with arguments whose types match the boundary of f, the function f is called instead.

```
.instance ~b8 : ~;
.instance !b8 : !;
```

The former line declares that the function $\tilde{}b8$ is an instance of the generic function g, that is, the type of the first argument for $\tilde{}b8$ belongs to the type class $\tilde{}[t]$. Likewise, the latter line declares that the type of the first argument for $!b8$ belongs to the type class $![t]$.

```
~list : List(t) -> : ~[t];
!list : List(t) -> List(t), List(t) : ![t];
```

The former line declares that the function $\tilde{}list$ takes an object of type $List(t)$ where t belongs to the type class $\tilde{}[t]$, i.e., the first argument for the function $\tilde{}list$ must be a list of deletable objects. Likewise, the latter line declares that the first argument for the function $!list$ must be a list of copyable ones.

7.2. Example

At first, we declare two generic functions ~ and !.

```
~ : t -> : ~[t];
! : t -> t, t : ![t];

.datatype B1 : false, true;
false : -> B1();
true : -> B1();
```

```
~b1 (x ->)
{
    false() = x;
|
    true() = x;
}

!b1 (x -> a, b)
{
    false() = x;
    a = false();
    b = false();
|
    true() = x;
    a = true();
    b = true();
}
```

The type $B1$ is a 2-valued boolean algebra[1]. It has two constructors *true* and *false*.

```
.datatype B2 : b2;
b2 : B1(), B1() -> B2();

~b2 (x -> )
{
    b2(h, l) = x;
    ~b1(h);
    ~b1(l);
}

!b2 (x -> a, b)
{
    b2(h, l) = x;
    h1, h2 = !b1(h);
    l1, l2 = !b1(l);
    a = b2(h1, l1);
```

[1]The boolean operations are not defined here because they are not used in the example.

```
    b = b2(h2, l2);
}

.datatype B4 : b4;
b4 : B2(), B2() -> B4();

~b4 (x ->)
{
    b4(h, l) = x;
    ~b2(h);
    ~b2(l);
}

!b4 (x -> a, b)
{
    b4(h, l) = x;
    h1, h2 = !b2(h);
    l1, l2 = !b2(l);
    a = b4(h1, l1);
    b = b4(h2, l2);
}

.datatype B8 : b8;
b8 : B4(), B4() -> B8();

.instance ~b8 : ~;
.instance !b8 : !;

~b8 (x ->)
{
    b8(h, l) = x;
    ~b4(h);
    ~b4(l);
}

!b8 (x -> a, b)
{
    b8(h, l) = x;
```

```
    h1, h2 = !b4(h);
    l1, l2 = !b4(l);
    a = b8(h1, l1);
    b = b8(h2, l2);
}
```

The types $B2$, $B4$, and $B8$ are 4-valued, 16-valued, and 256-valued boolean algebras. The functions ˜$b8$ and !$b8$ are declared as instances of the generic functions ˜ and !.

```
.datatype List : nil, cons;
nil : -> List(t);
cons : t, List(t) -> List(t);

~list : List(t) -> : ~[t];
!list : List(t) -> List(t), List(t) : ![t];

~list (a -> )
{
    nil() = a;
|
    cons(hd, tl) = a;
    ~(hd);
    ~list(tl);
}

!list (x -> a, b)
{
    nil() = x;
    a = nil();
    b = nil();
|
    cons(hd, tl) = x;
    h1, h2 = !(hd);
    t1, t2 = !list(tl);
    a = cons(h1, t1);
    b = cons(h2, t2);
}
```

The type $List(t)$ is the finite lists over t. The function $\tilde{}list$ deletes an object of type $List(t)$, and is available only if t is a deletable type. Likewise, the function $!list$ copies an object of type $List(t)$, and is available only if t is a copyable type.

```
.datatype String : string;

string : List(B8()) -> String();

.instance ~string : ~;
.instance !string : !;

~string (x -> )
{
    string(bl) = x;
    ~list(bl);
}

!string (x -> a, b)
{
    string(bl) = x;
    bl1, bl2 = !list(bl);
    a = string(bl1);
    b = string(bl2);
}
```

The type *String* is simply the finite lists over $B8$. Because the function $\tilde{}list$ calls the generic function $\tilde{}$, the expression $\tilde{}list(bl)$ requires that $B8$ is deletable. Likewise, the expression $!list(bl)$ requires $B8$ is copyable[2]. Both requirements are satisfied because $\tilde{}b8$ and $!b8$ are instances of $\tilde{}$ and ! respectively.

```
.datatype Maybe : just, nothing;

just : t -> Maybe(t);
nothing : -> Maybe(t);
```

[2]The function $!string$ is not used in this example.

```
~maybe : Maybe(t) -> : ~[t];

~maybe (mx -> )
{
    nothing() = mx;
|
    just(x) = mx;
    ~(x);
}
```

A value of type $Maybe(t)$ is either $nothing()$ or $just(x)$, where x is of type t. The type $Maybe(t)$ is used for error handling in this example.

```
.external_abstract_type Sys;
.external_abstract_type RFileDescriptor;
.external_abstract_type WFileDescriptor;
.external_abstract_type RFile;
.external_abstract_type WFile;

.external open_rfile, attach_rfile, detach_rfile, close_rfile;
.external read_b8;
.external open_wfile, attach_wfile, detach_wfile, close_wfile;
.external write_b8;

open_rfile : Sys(), String()
    -> Sys(), Maybe(RFileDescriptor());
attach_rfile : Sys(), RFileDescriptor() -> RFile();
detach_rfile : RFile() -> Sys(), RFileDescriptor();
close_rfile : Sys(), RFileDescriptor() -> Sys();
read_b8 : RFile() -> RFile(), Maybe(B8());
open_wfile : Sys(), String()
    -> Sys(), Maybe(WFileDescriptor());
attach_wfile : Sys(), WFileDescriptor() -> WFile();
detach_wfile : WFile() -> Sys(), WFileDescriptor();
close_wfile : Sys(), WFileDescriptor() -> Sys(), B1();
write_b8 : WFile(), B8() -> WFile();

write_string (str, wf -> wf_r)
```

```
{
    string(bl) = str;
    wf_r = write_b8list(bl, wf);
}

write_b8list (bl, wf -> wf_r)
{
    nil() = bl;
    wf_r = wf;
|
    cons(b, bl_tail) = bl;
    wf_r = write_b8list(bl_tail, write_b8(wf, b));
}
```

An object of type *Sys* holds the status of the operating system. Because the internals of the type *Sys* and related types/functions are defined by the operating system, they are declared as external abstract types and external functions (Section 7.1.5). The *open_rfile* function opens a file for reading[3]. It returns $just(x)$ on success, where x is a file descriptor which can be used for reading data from the file. The type *RFile* is simply the product of *Sys* and *RFileDescriptor*, and the *attach_file* function converts a pair of *Sys* and *RFileDescriptor* to a single object of type *RFile*. The *read_b8* function reads a byte from the file. It returns $just(x)$ on success, and $nothing()$ if an error occurs or the end of file is reached. The *detach_file* convert *RFile* back to *Sys* and *RFileDescriptor*. The *close_rfile* function closes a file descriptor. The type *WFile* and related functions are defined likewise.

```
main : Sys(), List(String()),
    RFileDescriptor(), WFileDescriptor(), WFileDescriptor()
    -> Sys(), B1();

main (sys, args, stdin, stdout, stderr -> sys_r, errflag)
{
    ~list(args);
    sys1, err_errfd = close_wfile(sys, stderr);
    ~b1(err_errfd);
```

[3] *open_rfile*, *open_wfile*, and *write_string* are not used in this example.

```
    sys_r, errflag = echo(sys1, stdin, stdout);
}

echo (sys, stdin, stdout -> sys_r, e)
{
    rf = attach_rfile(sys, stdin);
    rf1, mb8 = read_b8(rf);
    sys1, stdin1 = detach_rfile(rf1);
    sys_r, e = echo_cond(mb8, sys1, stdin1, stdout);
}

echo_cond (mb8, sys1, stdin1, stdout -> sys_r, e)
{
    nothing() = mb8;
    sys2 = close_rfile(sys1, stdin1);
    sys_r, e = close_wfile(sys2, stdout);
|
    just(c) = mb8;
    c1, c2 = !b8(c);
    wf = attach_wfile(sys1, stdout);
    wf1 = write_b8(wf, c1);
    wf2 = write_b8(wf1, c2);
    sys2, stdout1 = detach_wfile(wf2);
    sys_r, e = echo(sys2, stdin1, stdout1);
}
```

The *main* function is the entrance of a program. It receives a *Sys*, a list of argument strings, the standard input, the standard output, and the standard error. In this example, the *main* function simply calls the *echo* function after deleting some unnecessary objects[4]. The *echo* function reads a byte from the standard input, and calls the *echo_cond* function. The $mb8$ variable becomes $just(c)$ if the $read\_b8$ function successfully reads a byte from the standard input, and $nothing()$ if the end of file is reached. If the $mb8$ variable is $just(c)$, the *echo_cond* function creates two copies of the c object, writes them to the standard output, and calls *echo* recursively. This recursion ends when $mb8$ becomes $nothing()$, i.e., the standard input

[4] *main* and *echo* are defined by compositions of functions (Section 7.1.4).

reaches to the end of file. As a result, this sample program reads a byte from the standard input, writes it to the standard output twice, and repeats them until the end of file is reached.

8. Concluding remarks

Recently more "interaction" aspects occurring in the real world are taken into consideration in modeling computation especially in the study of parallel and distributed systems [6, 3, 10, 8]. Although the concept of graph rewriting has potential ability to describe intricate combinatorial structures of microscopic interaction and there are large research activities [4, 14] around graph rewriting, there are few which intend to focus on the interaction aspect, except for such frameworks as the interaction nets [9, 2, 5], Milner's π-calculus [11]. In the latter, the agents exchange link information during the interaction so that the topology of the net change globally, whereas in the former one, the net evolves asynchronously by succession of local interactions between two agents at specific ports of complementary type. The latter property is retained in our 2-hypergraph formulation. In fact one of our contributions is to formulate a mathematical theory which expresses atomic interactions explicitly as 2-cells of a 2-hypergraph, which opens a way to give direct semantics of such computational models as interaction nets via the theory of 1-hypercategories [7].

Our mathematical model of computation supports a functional programming language which has common features with the language Clean [1, 12, 13] in the referential transparency and the laziness of evaluation, although there exist radical differences. For example, in Clean, data which are not copyable must be declared so explicitly, whereas in our language, the procedure of copying must be given explicitly for copyable data. This aspect is useful in synthesizing actual systems reactive with the real world, where most objects are not copyable.

Acknowledgements

I would like to thank people in graduate school of Mathematics, Hokkaido University for their support during the preparation of this paper and Toru Tsujishita for his help and advice.

I am deeply indebted to the Japan Society for the Promotion of Science for the JSPS Research Fellowships from 1997 to 1999 which supported my research activity in that period.

References

[1] Achten P.M., et al., *High level specification of I/O in functional languages.* In Launchbury J. et al., editor, Proceedings Glasgow Workshop on Functional Programming, Springer Verlag, New York, NY, 1993.

[2] Banach R., *The algebraic theory of interaction nets.* 1995.

[3] Berry G. and Boudol G., *The chemical abstract machine.* Theoretical Computer Science, **96** (1992), 217–248.

[4] Ehrig H. and Taentzer G., *Computing by graph transformation, A survey and annotated bibliography.* Bulletin of the European Association for Theoretical Computer Science, **59** (1996), 182–226. Bibliographies.

[5] Fernandez M. and Mackie I., *A calculus for interaction nets.* In Principles and Practice of Declarative Programming, 1999, pp. 170–187.

[6] Girard J.-Y., *Towards a geometry of interaction.* In Categories in Computer Science, vol. 92 of *Contemporary Mathematics*, AMS, 1987, pp. 69–108.

[7] Higuchi A., Miyoshi H., and Tsujishita T., *Strict n-hypercategories.* Hokkaido Math. J. **31** (2002), 469–511.

[8] König B., *A general framework for types in graph rewriting.* Lecture Notes in Computer Science, vol. 1974, 2000, p. 373.

[9] Lafont Y., *From proof nets to interaction nets.* In Girard J.-Y., Lafont Y., and Regnier L., editors, Advances in Linear Logic, Cambridge University Press, 1995, pp. 225–247. London Mathematical Society Lecture Note Series 222, Proceedings of the 1993 Workshop on Linear Logic, Cornell University, Ithaca.

[10] Milner R., *Calculi for interaction.* Acta Informatica, **33** (1996), 707–737.

[11] Milner R., *Communicating and Mobile Systems: The π-Calculus.* Cambridge University Press, 1999.

[12] Nocker E., Smetsers S., van Eekelen M. and Plasmeijer R., *Concurrent clean.* In Leeuwen Aarts and Rem, editors, Proc. of Parallel Architectures and Languages Europe (PARLE '91), vol. 505, Springer-Verlag, 1991, pp. 202–219.

[13] Plasmeijer M. and van Eekelen M., *Language report concurrent clean*, 1998.

[14] van Eekelen M.C.J.D., Smetsers S. and Plasmeijer M.J., *Graph rewriting semantics for functional programming languages.* In CSL, 1996, pp. 106–128.

Kita 12 Nishi 3-5-217
Kita-ku, Sapporo 001-0012, Japan