Origami Fold as Algebraic Graph Rewriting

Tetsuo Ida

Department of Computer Science, University of Tsukuba
Tsukuba, 305-8573, Japan

Hidekazu Takahashi [1]

Shiga Prefectural Yokaichi High School, Higashiomi, 527-0022, Japan

Abstract

We formalize paper fold (origami) by graph rewriting. Origami construction is abstractly described by a rewriting system $(\mathbb{O}, \looparrowright)$, where $\mathbb{O}$ is the set of abstract origamis and $\looparrowright$ is a binary relation on $\mathbb{O}$, that models *fold*. An abstract origami is a structure $(\Pi, \backsim, \succ)$, where Π is a set of faces constituting an origami, and $\backsim$ and $\succ$ are binary relations on Π, each representing adjacency and superposition relations between the faces.

We then address representation and transformation of abstract origamis and further reasoning about the construction for computational purposes. We present a labeled hypergraph of origami and define fold as algebraic graph transformation. The algebraic graph-theoretic formalism enables us to reason about origami in two separate domains of discourse, i.e. pure combinatorial domain where symbolic computation plays the main role and geometrical domain $\mathbb{R} \times \mathbb{R}$. We detail the program language for the algebraic graph rewriting and graph rewriting algorithms for the fold, and show how fold is expressed by a set of graph rewrite rules.

Key words: computational origami, geometrical modeling, graph rewriting, rewrite system

[1] The second author's work was conducted during his Doctoral Program of Computer Science at University of Tsukuba.

⋆ This research is supported by the JSPS Grants-in-Aid for Exploratory Research No. 19650001, Scientific Research (B) No. 20300001 and by Japan-Austria Research Cooperative Program of JSPS and FWF.

⋆⋆The earlier and abridged version of this paper appeared in the Proceedings of the 24th Annual ACM Symposium on Applied Computing, March 2009.

Email addresses: ida@cs.tsukuba.ac.jp (Tetsuo Ida), t.hidekazu@gmail.com (Hidekazu Takahashi).
URL: www.score.cs.tsukuba.ac.jp/ ida/ (Tetsuo Ida).

1. Introduction

The art of paper folding, known as *origami* [2], provides the methodology of constructing a geometrical object out of a sheet of paper solely by means of folding by hands. Computational origami studies the mathematical and computational aspects of origami, including visualization by a computer [2]. By the assistance of a computer we will be able to formalize origami with rigor and capability that are beyond the methods performed by hands.

In this paper we give graph-theoretic formalization of origami. Our motivation of this study is to give more abstract view of fold used in origami. Although origami fold appears to be an easy operation to humans, even a naïve anatomy of origami reveals that it is not the case from computational point of view. There are two distinct operations in paper fold, i.e. division and reflection of origami faces. These operations lend themselves to distinct modes of computations: algebraic and numeric computation on geometrical objects, e.g., finding intersection of lines and checking the overlap of two faces, on one hand, and purely combinatorial computation on discrete objects, e.g., computing transitive closure of the adjacency relation on faces, on the other.

These computations tend to be mixed when origami is analyzed mathematically [8]. Indeed the implementation of computational origami system Eos [9] relies very much on algorithms which resort to mixtures of algebraic, numeric and symbolic computing. Sometimes algorithms are hard to describe mathematically because of this complication. There should be clearer separation of computations of discrete and continuous objects in origami. When this has been done, we not only clarify the algorithms developed for the implementation of Eos, but also are in a position to extend the capability of Eos to allow for more complex origami constructions such as of 3D and modular origami, and to reason about their geometrical and algebraic properties.

The rest of the paper is organized as follows. In Section 2 we will formalize origami as an abstract origami system, which can be regarded as an abstract rewrite system. In Section 3 we will elaborate the abstract origami system by giving to it more algebraic and geometrical structures, and then we will analyze the structures. Section 4 is a short passage from the abstract description of origami to more algorithmic description of origami. In Sections 5 and 6, we will explain the bases for graph-theoretic modeling of origami. In Section 7, we show fold as a set of graph rewrite rules. In Section 8, we will summarize the results and point out the direction of further research.

2. Formalizing origami

2.1. Preliminaries

In this subsection, we summarize the basic mathematical notations that we will use throughout this paper. Given an arbitrary set A, a sequence of elements $a_1, \ldots, a_n \in A$ is denoted by $\langle a_1, \ldots, a_n \rangle$. If $n = 1$, the sequence is written as a_1. In other words, $\langle a \rangle$ and a denote the same mathematical object. If A is an alphabet, the sequence of elements $a_1, \ldots, a_n (\in A)$ is denoted by the juxtaposition of the elements, i.e. $a_1 \cdots a_n$.

[2] The word *ori-gami*, meaning ori (fold) and gami (paper), is used in two ways in this paper as is customary in Japanese. The former is the art of paper fold (used as an uncountable noun) and the other is a sheet of paper (used as a countable noun).

The set of all the sequences of the elements in A is denoted by A^*. We use * in the superscript position in the following ways, too. Namely, f^* for a function f is defined as: $f^*(\langle a_1, \ldots, a_n \rangle) = \langle f(a_1), \ldots, f(a_n) \rangle$. R^* for a binary relation R is the reflexive and transitive closure of R. The transitive closure of R is denoted by R^+. A sequence $\langle a_1, \ldots, a_n \rangle$ is called an R-sequence if $a_1\ R\ a_2\ \wedge\ \cdots\ \wedge\ a_{n-1}\ R\ a_n$. For an irreflexive R, an R-sequence $\langle a_1, \ldots, a_n \rangle$ is called *acyclic* if $\forall i < j\ \ \neg(a_j\ R\ a_i)$. The cross product of sets A_1, ..., A_n is denoted by $A_1 \times \ldots \times A_n$. The element of $A_1 \times \ldots \times A_n$, i.e. n-tuple, is denoted by $(a_1, \ldots, a_n)$, where $a_i \in A_i$ for $i = 1, \ldots, n$.

2.2. Origami at a glance

We begin an origami construction with a single sheet of paper, and repeat folding the paper until it becomes a desired shape. We can observe that an origami can be modeled as a set of faces. During the construction, some of the faces are divided by a fold line, reflected along the fold line and become above or below the others. The faces form a stack of layers. The stack of layers of faces exhibits a remarkable shape, which may be regarded as a piece of art such as illustrated in Fig. 1.

The left origami in Fig. 1 is the top view of the constructed object. We see the faces in two different colors in the figure. This is because the initial origami has two sides, each colored differently. During the construction, some faces become up and the others become down, resulting in the two colored object. We can imagine that this origami models a cicada. The right is a 3D view of the same origami after stretching it vertically and making superposing faces slightly far apart. From the shapes in Fig. 1, we will be able to see that an origami can be formalized as a set of faces together with the relations that express relative positions, horizontally and vertically, among the faces.

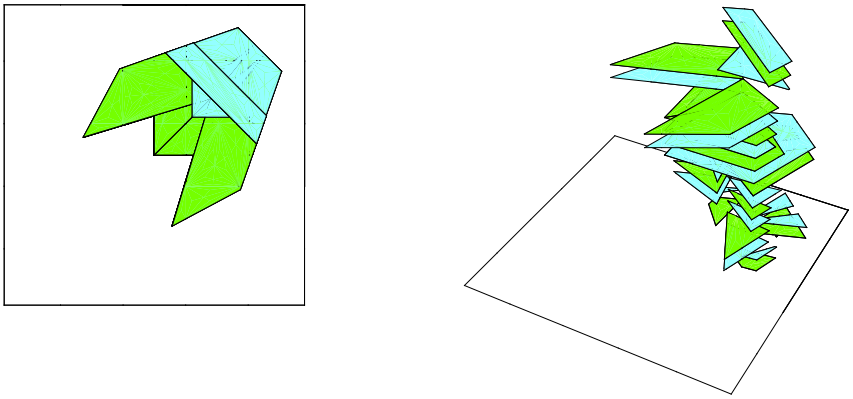


Fig. 1. Origami cicada: art piece (left) and stack of face layers (right)

2.3. Abstract origami rewrite system

An origami can be modeled at several abstraction levels. A most abstract view is to take an origami as an algebra (A, R), where A is a set and R is a binary relation on A, where we identify a set of faces that constitute an origami with A, and a geometrical relation on the faces with R. The origami construction is then a transformation of the algebras viewed as an abstract rewrite system. We begin with this abstract view of origami and gradually make our modeling concrete.

Our first attempt is as follows. We take a finite set Π of faces as the object of our study, and introduce two binary relations on Π, expressing horizontal and vertical arrangements of faces rather than a single binary relation R mentioned above. Then, we have the following definition of an origami.

Definition 1 (Abstract origami)**.** An abstract origami is a structure $(\Pi, \backsim, \succ)$, where Π is the finite set of (origami) faces, $\backsim$ is a symmetric binary relation on Π, called adjacency relation, and $\succ$ is a binary relation on Π, called superposition relation.

An abstract origami is abbreviated to AO, hereafter.

We next present a view that an origami construction is a rewrite sequence of an abstract rewrite system. This view bridges the set theoretic treatment of origami and graph rewriting of origami.

Definition 2 (Abstract origami system)**.** An abstract origami system is an abstract rewrite system $(\mathbb{O}, \looparrowright)$, where $\mathbb{O}$ is the set of AOs, and $\looparrowright$ is a rewrite relation on $\mathbb{O}$, called *abstract fold*.

When $O, O' \in \mathbb{O}$ are related by $\looparrowright$, we write $O \looparrowright O'$ and say that O is rewritten or abstractly folded to O'.

An origami construction proceeds as follows. We start with an initial origami and perform folds repeatedly until we obtain a desired shape of the origami. Usually we begin an origami construction with a square sheet of paper, but the sheet can be in any shape having convexity. This initial sheet of paper is abstracted as a structure having a single distinguished face to be denoted by f_0. Then, we define the initial AO, denoted by I, as follows.

Definition 3 (Initial abstract origami)**.** The initial AO I is a structure $(\{f_0\}, \emptyset, \emptyset)$.

Suppose that we are at the beginning of step i of the construction, having an origami $O_{i-1} = (\Pi_{i-1}, \backsim_{i-1}, \succ_{i-1})$. We perform a fold and obtain the next origami $O_i = (\Pi_i, \backsim_i, \succ_i)$. Thus we have the following:

Definition 4 (Abstract origami construction)**.** An abstract origami construction is a finite sequence of AOs satisfying

$$O_0 (= I) \looparrowright O_1 \looparrowright \cdots \looparrowright O_n, \quad \text{where } O_0, O_1, \ldots, O_n \in \mathbb{O}$$

In this paper, we decompose an abstract fold into two finer operations in order to make our formalism general enough to model both mathematical origami and art origami. Namely, we decompose a fold into the following operations: (a) making a crease along

which the sheet is bended, and then (b) actually bending the sheet. We call the operation (a) *crease* and operation (b) *basic fold*.

In mathematical origami, we are primarily interested in constructing points on the origami. Those points form the vertices of the shape that we want to construct. In such cases we only need creases and points of intersections of the creases and the sides of the initial origami. In art origami, we are interested in artistic shapes that are composed of layers of faces. In art origami, bending the sheets are equally important. Bending outwards is commonly called *mountain fold*, and inwards *valley fold*. In summary, we have the relation of the abstract fold as the composition of the relations *crease* ($\overset{\rightsquigarrow}{\rightarrow}$), and *basic fold* ($\overset{\frown}{\rightarrow}$). The basic fold is a union of *mountain fold* ($\overset{\frown}{\rightarrow}$) and *valley fold* ($\overset{\smile}{\rightarrow}$).

$$\looparrowright = \overset{\frown}{\rightarrow} \circ \overset{\rightsquigarrow}{\rightarrow}, \text{ where } \overset{\frown}{\rightarrow} = \overset{\frown}{\rightarrow} \cup \overset{\smile}{\rightarrow}$$

In the case of an unfold, no crease operation precedes a valley fold or a mountain fold. In this case, the crease operation is vacuous, i.e. $\overset{\rightsquigarrow}{\rightarrow}$ is the identity relation. In some mathematical folds, creasing is important and basic folds play no role. In such a case, $\overset{\smile}{\rightarrow}$ (or $\overset{\frown}{\rightarrow}$) is the identity relation.

Let us consider an example of the abstract origami construction. We are interested in the construction that starts with $O_0 = I$ and simulates the origami construction by hands.

Example 1. Construction from the initial abstract origami We are given the initial AO I. We define O'_1 and O_1 as follows: $O'_1 = (\Pi'_1, \backsim'_1, \succ'_1)$, where $\Pi'_1 = \{f_1, f_2\}$, $\backsim'_1 = \{(f_1, f_2)\}$, $\succ'_1 = \emptyset$, $O_1 = (\Pi_1, \backsim_1, \succ_1)$, where $\Pi_1 = \Pi'_1$, $\backsim_1 = \backsim'_1$, $\succ_1 = \{(f_2, f_1)\}$. Figure 2 shows the rewrites of the initial AO. O_1 is the abstraction of the origami resulting from folding the initial origami once. O_1 is obtained first by creasing I, thus constructing O'_1, and then by making a valley fold.

We also allow a fold along the sides of the faces. Then the fold along one of the four sides of the initial origami is simply turning over the initial origami. I and the abstraction of the upside-down ones of the initial origami are the same. Therefore, we have abstract origami constructions such as:

$$I \looparrowright O_1 (= I),$$
$$I \looparrowright O_1, (I \overset{\rightsquigarrow}{\rightarrow} O'_1 \text{ and } O'_1 \overset{\smile}{\rightarrow} O_1).$$

Figure 2 shows that some faces are colored differently from the others. This property is abstracted away during the process of the abstraction that leads to the abstract rewriting system. The property of "sides" appear when we give more geometrical structures in the next section.

3. Geometrical and algebraic structures of origami

The notion of fold is central in origami and in abstract origami systems. The relation of $\looparrowright$ is given a priori in defining the abstract rewrite system. There is no clue in the above formalism as to how an AO is computationally related via $\looparrowright$ to the other. We now give more geometrical structures to the constituents of the abstract origami system to relate AOs computationally.

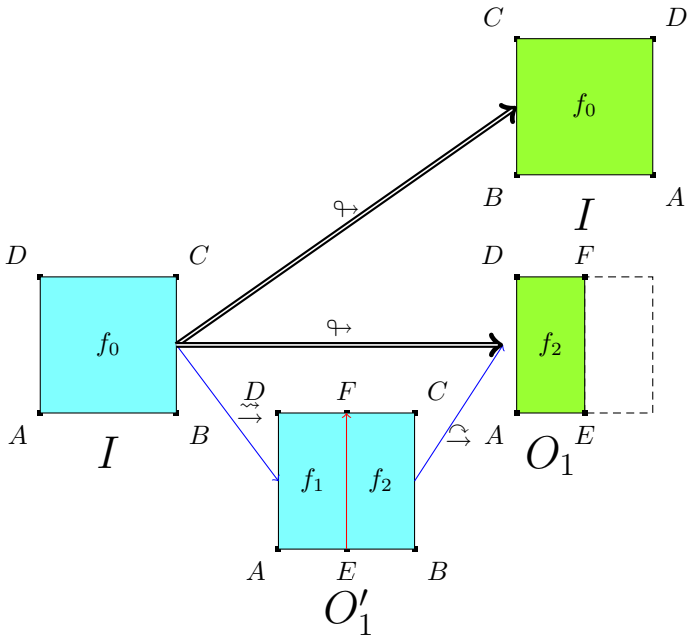
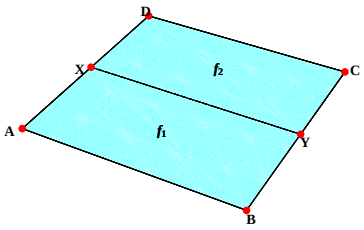


Fig. 2. Abstract folds from the initial AO with geometrical interpretation

3.1. Geometry of origami

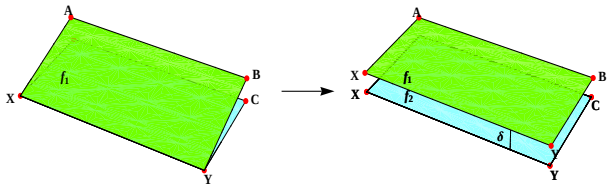
We rely on the following geometrical intuition.

(I1) A face is a plane surface in the sense of Euclid's Elements.
(I2) A face can share the boundary with other faces. See Figs. 3 and 4.
(I3) A face can be above other faces, and all the faces form a stack of face layers. See Fig. 5.
(I4) A point belonging to a face can be overlaid on the other points belonging to other faces. See Fig. 6.



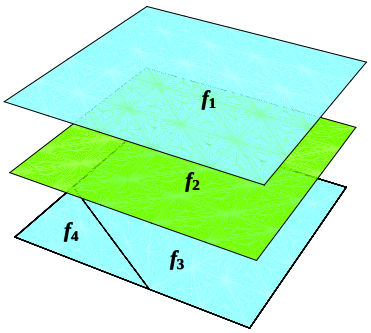
The two faces f_1 and f_2 are created by the crease XY. The two faces share the same edge XY.

Fig. 3. Adjacent faces created by crease



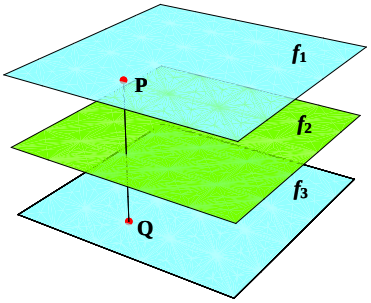
On the left, the two faces f_1 and f_2 share the same edge XY. On the right, the faces f_1 and f_2 are modeled as two plane surfaces with infinitesimally small vertical distance. The two surfaces have the same edge XY and are considered to share the same edge.

Fig. 4. Adjacent faces created by fold along crease



The faces f_1, f_2, f_3 and f_4 form a stack of three layers (f_1), (f_2) and (f_3, f_4).

Fig. 5. Stack of face layers



The point P belonging to the face f_1 overlays the point Q belonging to the face f_3.

Fig. 6. Overlay of points

Then we give more geometrical structures to the notion of a face. We appeal to the readers for the intuition (I1) and define a face as follows.

Definition 5 (Face)**.** A face is a convex n-gon.

Recall that an n-gon ($n \geqslant 3$) is a polygon consisting of n edges none of which intersects each other. We represent an n-gon as a sequence $\langle P_1, \ldots, P_n \rangle$ of pairwise distinct points, where points $P_1, \ldots, P_n$ are vertices of the n-gon. Any cyclic permutations of a given sequence could represent the same face, but we choose one particular sequence among them in order to represent a face uniquely. The convexity does not play a role in this paper. It is abstracted away in our discussion, but is used crucially in the design of the algorithms for face division and for checking face overlap. When points $P_1, \ldots, P_n$ are arranged counterclockwise, we say that the face is up, and when clockwise, down. In the figures we distinguish the sides of the faces by the shading (colored if color print is possible).

We can now define the adjacency relation as follows.

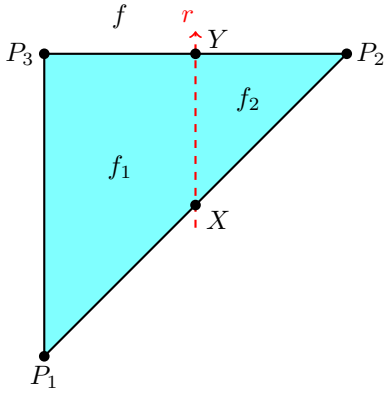
Definition 6 (Face adjacency)**.** Two distinct faces are adjacent if they share an edge.

The adjacency relation is generated between the two faces that have been constructed by the division of a face. Our construction ensures the following properties, and we take them as an axiom of our construction by folds. In describing the axiom, we use the following notation. When a face f is divided by a ray r into (f_1, f_2), where f_1 is to the left of the ray r and f_2 is to the right of r, we write $f \rightarrowtail_r (f_1, f_2)$. Later we also write $f \rightarrowtail_r f'$, where f' is one of f_1 and f_2, whenever we need only to specify one of the divided faces. The subscript r in $\rightarrowtail_r$ may be omitted if intended r is clear from the context. When $f \rightarrowtail (f_1, f_2)$, f is called the *immediate ancestor* of f_1(and f_2). Face g_1 is called the *ancestor* of face g_k if there exists g_2, …, g_k such that g_j is the immediate ancestor of g_{j+1} for $j = 1, \ldots, k-1$.

Axiom 7. *Suppose $f \rightarrowtail_r (f_1, f_2)$. Then we have the following:*
(1) $f_1 \backsim f_2$

(2) Let X and Y be points of intersection with the ray r and XY be one of the edges of f_1 such that $f_1 = \langle X, Y, \ldots \rangle$. Then $f_2 = \langle Y, X, \ldots \rangle$.
(3) The sides of the faces f, f_1 and f_2 are the same.

Figure 7 depicts the adjacency relation between the divided faces.



The faces $f_1 = \langle X, Y, P_3, P_1 \rangle$ and $f_2 = \langle Y, X, P_2 \rangle$ are created by the division of face $f = \langle P_1, P_2, P_3 \rangle$ by the ray r.

Fig. 7. Faces and the adjacency relation as determined by Axiom 7

Note that the n-gon $\langle X, Y, \ldots \rangle$ can be denoted by any cyclic permutation of the sequence $\langle X, Y, \ldots \rangle$. However, we fix the denotation as in Axiom 7. The following lemma is an easy consequence of our construction.

By $\langle\!\langle X, Y \rangle\!\rangle$ we denote the set of all the finite sequences that have a subsequence $\langle X, Y \rangle$ and their cyclic permutations.

Lemma 8. *For any $O = (\Pi, \backsim, \succ)$ such that $I \not\hookrightarrow^* O$,*

$$\forall f, g \in \Pi \ \ (f, g \in \langle\!\langle X, Y \rangle\!\rangle \text{ for some points } X \text{ and } Y \Leftrightarrow f = g)$$

Lemma 8 states that a directed edge of a face in a given Π uniquely identifies the face. As we have decided the representation of a face, we have the following equivalence.

Proposition 9. *For any $O = (\Pi, \backsim, \succ)$ such that $I \not\hookrightarrow^* O$,*
$\forall f, g \in \Pi$

$$f \backsim g \Leftrightarrow \exists \text{ points } X \text{ and } Y \text{ such that } f \in \langle\!\langle X, Y \rangle\!\rangle \wedge g \in \langle\!\langle Y, X \rangle\!\rangle.$$

Proof: ($\Leftarrow$) The edge XY are shared by f and g. By definition of the adjacency, $f \backsim g$. ($\Rightarrow$) We have either $f \in \langle\!\langle X, Y \rangle\!\rangle \wedge g \in \langle\!\langle Y, X \rangle\!\rangle$ for some points X and Y, or $f \in \langle\!\langle X, Y \rangle\!\rangle \wedge g \in \langle\!\langle X, Y \rangle\!\rangle$ for some points X and Y. The latter is impossible by Lemma 8. □

Example 2. Adjacency relation We make use of Example 1. The faces of f_1 and f_2 in Fig. 2 are adjacent by the following reasoning. We have $f_1 = \langle E, F, D, A \rangle$ and $f_2 = \langle F, E, B, C \rangle$. Since f_1 and f_2 share the edge EF, we can see that $f_1 \backsim f_2$ and $f_2 \backsim f_1$.

The superposition relation gives the vertical arrangement of faces. We first appeal to the readers for intuition about the notion of *above* and *below* among faces (cf. Intuition (I3)), where *below* is the inverse relation of *above*. Roughly speaking, face f superposes face g iff f is above g, and moreover f and g "face" each other. Let us take a look at Fig. 2 again. We consider making a valley fold on the face f_0. The face f_0 is divided into (f_1, f_2). The face f_2 is rotated along the axis EF by an angle π. The face f_2 is clearly above f_1 in O_2 and f_1 faces f_2. Hence f_2 superposes f_1.

We are now ready to formalize the notion of superposition. We proceed as follows. First we formalize the notion of overlap based on the intuition (I4). Then we define the relation *over* on the set of faces. Using the notion of over, we define *above* and *superposition*.

Definition 10 (Overlap)**.** Faces f and g *overlap*, denoted by $f \asymp g$, iff there exist points P and Q, each belonging to faces f and g respectively, such that P overlays Q (or vice versa).

Note that we allow a point to be overlaid on itself, Hence, a face can overlap with itself. One might imagine that there is a through-hole that penetrates the points P and Q.

To define the relation over, we will use the following notations. Let $\overset{=}{\rightarrowtail}_r$ be $\rightarrowtail_r \cup =$, where $=$ is the equality on faces. The relation $\overset{=}{\rightarrowtail}_r$ is introduced more for convenience; By $f \overset{=}{\rightarrowtail}_r f'$ we want to describe the situation where r may not divide f.

Let $\backsim_{\mathrm{on}(r)}$ denote the adjacency relation between two faces that share an edge that lies on the ray r and let $\backsim_{\neg\mathrm{on}(r)} = \backsim \setminus \backsim_{\mathrm{on}(r)}$. Let $\backsim_{\mathrm{left}(r)}$ denote the adjacency relation between two faces that share an edge that is to the left of the ray r and let $\backsim_{\neg\mathrm{left}(r)} = \backsim \setminus \backsim_{\mathrm{left}(r)}$.

To make a crease on an origami, we need to specify the ray r and the set C of faces to which we want to apply the face division and on which we want to make a crease afterwards. The set C is called a *candidate set of face division*. Origamists give a set F of faces that they want to make a crease. Then, C can be determined as

$$C = \bigcup_{f \in F} \left\{ g \mid g \backsim^*_{\neg\mathrm{left}(r)} f \right\}.$$

Later we will discuss how to compute the set C. For now, it is sufficient to note that whenever we discuss the fold $\looparrowright$, we assume that r and C are implicitly given.

To make a basic fold $O' \overset{\curvearrowright}{\rightarrow} O''$ after $O \overset{\rightrightarrows}{\rightarrow} O' (= (\Pi', \backsim', \succ'))$, we need to determine the set Π'_M of faces that are to be moved and the set Π'_N of faces that are not moved. We define Π'_M and Π'_N as follows:

$$\Pi'_M = \bigcup_{f \in D} \left\{ g \mid g \, (\backsim'_{\neg\mathrm{on}(r)} \cup \succ')^* \, f \right\}, \text{ and } \Pi'_N = \Pi' \setminus \Pi'_M.$$

where D is the set of faces that have been divided in $O \overset{\rightrightarrows}{\rightarrow} O'$ and are to the right of the ray r.

Definition 11 (Over)**.** Let $O = (\Pi, \backsim, \succ)$ be an AO. We define the relation *over*, denoted by $\succcurlyeq$, on Π inductively on the construction of AOs.

Suppose $I \looparrowright^* O$.
$\forall f, g \in \Pi$

- Base case: $O = I$

$$\gtrdot = \emptyset$$

- Inductive case 1: $I \looparrowright^* O \overset{\rightsquigarrow}{\rightarrow} O'(\overset{\frown}{\rightarrow} O'')$ where $O = (\Pi, \backsim, \succ)$, and $\gtrdot$ is defined on Π. We will define $\gtrdot'$ on Π' as follows.

$$f' \gtrdot' g' \text{ iff } \exists f, g \in \Pi \ (f \gtrdot g) \wedge (f \overset{=}{\rightarrowtail} f') \wedge (g \overset{=}{\rightarrowtail} g') \wedge (f' \asymp g')$$

- Inductive case 2.1: $I \looparrowright^* O \overset{\rightsquigarrow}{\rightarrow} O' \overset{\frown}{\rightarrow} O''$ where $O' = (\Pi', \backsim', \succ')$, and $\gtrdot'$ is defined on Π'. We will define $\gtrdot''$ on Π'' as follows. Note that in both cases 2.1 and 2.2 we have $\Pi'' = \Pi'$ and that $f \asymp g$ is geometrically interpreted for the case that $f, g \in \Pi''$.

$$f \gtrdot'' g \text{ iff}$$
$$(g \gtrdot' f \wedge f, g \in \Pi'_M) \vee (f \gtrdot' g \wedge f, g \in \Pi'_N) \vee (f \asymp g \wedge f \in \Pi'_M \wedge g \in \Pi'_N)$$

- Inductive case 2.2: $I \looparrowright^* O \overset{\rightsquigarrow}{\rightarrow} O' \overset{\frown}{\rightarrow} O''$ where $O' = (\Pi', \backsim', \succ')$, and $\gtrdot'$ is defined on Π'. We will define $\gtrdot''$ on Π'' as follows.

$$f \gtrdot'' g \text{ iff}$$
$$(g \gtrdot' f \wedge f, g \in \Pi'_M) \vee (f \gtrdot' g \wedge f, g \in \Pi'_N) \vee (f \asymp g \wedge g \in \Pi'_M \wedge f \in \Pi'_N)$$

Definition 12 (Above)**.** The relation *above*, denoted by $\gg$, is the transitive closure of $\gtrdot$.

The notion of *below* becomes rigorous now since it is the inverse relation of *above*. Note that the relation $\gtrdot$ does not have the transitivity, as the following example shows.

Example 3. Over and above Figure 8 illustrates the relations *over* and *above*. Face f_1 is *over* f_2. Face f_2 is *over* f_3. Hence, face f_1 is *above* f_2 and f_3, and face f_2 is *above* f_3. However, face f_1 is not *over* f_3 since f_1 does not overlap with f_3.

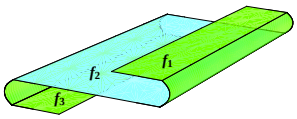


Fig. 8. Relations: over and above

Finally, we define the relation of superposition.

Definition 13 (Superposition)**.** Let $O = (\Pi, \backsim, \succ)$ be an AO such that $I \looparrowright^* O$. We define the relation *superposition*, denoted by $\succ$, on Π as follows.

$$\forall\, f, g \in \Pi \ \ f \succ g \text{ iff } (f \gtrdot g) \wedge (\forall h \in \Pi \ (f \gtrdot h \gtrdot^* g \implies h = g))$$

3.2. Analysis of relations

We will derive basic mathematical properties of the relations defined in the previous subsection. The following propositions are immediate from the definitions of the relations.

Proposition 14 (Asymmetry of $\succcurlyeq$)**.**

$$\forall\, O = (\Pi, \rightsquigarrow, \succ) \in \mathbb{O} \text{ such that } I \twoheadrightarrow^* O, \forall\, f, g \in \Pi \;\; f \succcurlyeq g \implies \neg(g \succcurlyeq f).$$

In the following, all the propositions in this subsection hold for O under the context "$\forall\, O = (\Pi, \rightsquigarrow, \succ) \in \mathbb{O}$ such that $I \twoheadrightarrow^* O$". We omit this context for brevity.

Corollary 15 (Irreflexivity of $\succcurlyeq$)**.**

$$\forall\, f \in \Pi \;\; \neg(f \succcurlyeq f).$$

Proposition 16.

$$\forall\, f, g \in \Pi \;\; f \succcurlyeq g \implies f \asymp g.$$

Proposition 17.

$$\gg \supset \;\; \succcurlyeq \supset \succ .$$

Lemma 18.

$$\neg\, (\succcurlyeq \supset \succ^+).$$

Proof: There exist faces f, g and h such that $f \succ h$ and $h \succ g$ and $f \not\asymp g$. By Proposition 16, $\neg(f \succcurlyeq g)$ if $f \not\asymp g$. □

Proposition 19. *An $\succcurlyeq$-sequence is acyclic.*
Proof: By induction on the construction of AOs.

- *Base case: $O = I$*
  Immediate, since we have $\neg\, (f_0 \succcurlyeq f_0)$.
- *Inductive case: $I \twoheadrightarrow^* O \twoheadrightarrow O'$ and $O = (\Pi, \rightsquigarrow, \succ)$*
  We assume the non-existence of a cyclic $\succcurlyeq$-sequence. We prove by contradiction. Suppose there exists a cyclic $\succcurlyeq'$-sequence. Taking the irreflexivity of $\succcurlyeq'$ into account, we can write the sequence as

$$\left\langle g'_0(= g'), g'_1, \ldots, g'_n, g'_{n+1}(= g') \right\rangle, \text{ where } n \geq 1.$$

We distinguish the following three cases:

(1) $I \twoheadrightarrow^* O \overset{\rightsquigarrow}{\rightarrow} O'$ ($\overset{\frown}{\rightarrow} O''$)
Let g and g'_i be the faces such that $g \overset{=}{\rightarrowtail} g'$, $g_i \overset{=}{\rightarrowtail} g'_i$ for all $i \in \{1, \ldots, n\}$.
By the definition of $\succcurlyeq'$, we have on O the $\succcurlyeq$-sequence:

$$\langle g_0(= g), g_1, \ldots, g_n, g_{n+1}(= g) \rangle, \text{ where } n \geq 1. \tag{1}$$

The sequence (1) is cyclic. This contradicts the induction hypothesis.

(2) $I \twoheadrightarrow^* O \overset{\rightsquigarrow}{\rightarrow} O' \overset{\frown}{\rightarrow} O''$
We further distinguish the following two cases.

(a) $g \in \Pi'_M$

We show that $g_i \in \Pi'_M$ *for all* $i \in \{1, \ldots, n\}$. *Suppose the contrary, i.e., there exists* $i \in \{1, \ldots, n\}$ *such that* $g_i \in \Pi'_N$. *Let* j *be the maximum index among such i's, i.e.,*

$$g_j \gtrdot'' g_{j+1},\ g_j \in \Pi'_N,\ g_{j+1} \in \Pi'_M.$$

By the definition of $\gtrdot''$, *if* g_j *and* g_{j+1} *overlap, then we have* $g_{j+1} \gtrdot'' g_j$, *else we have* $\neg\,(g_{j+1} \gtrdot'' g_j) \wedge \neg\,(g_j \gtrdot'' g_{j+1})$. *Using the asymmetry of* $\gtrdot''$, *in both cases, we have* $\neg\,(g_j \gtrdot'' g_{j+1})$, *which is contradictory. Therefore, we have for all* $i \in \{1, \ldots, n\}$, $g_i \in \Pi'_M$. *This implies that we have the* $\gtrdot'$*-sequence:*

$$\langle g_{n+1}(= g), g_n, \ldots, g_1, g_0(= g)\rangle \ \text{ where } n \geq 1. \tag{2}$$

The sequence (2) is cyclic. This contradicts the induction hypothesis.

(b) $g \in \Pi'_N$

We can show that $g_i \in \Pi'_N$ *for all* $i \in \{1, \ldots, n\}$. *We take the minimum* j *instead. We then derive the same contradiction.*

(3) $I \leftarrow\!\!\!\rightarrow^* O \overset{\rightsquigarrow}{\rightarrow} O' \overset{\curvearrowright}{\rightarrow} O''$

This case is treated similarly to the previous case.

In summary, we can conclude that $\gtrdot$ *is acyclic.* □

Noting Proposition 17, we have the following corollaries.

Corollary 20. *A* $\succ$*-sequence is acyclic.*

Corollary 21. *Relation* $\gg$ *is asymmetric.*

Since a strict partial order is a relation that is asymmetric and transitive, we have the following.

Proposition 22. *Relation* $\gg$ *is a strict partial order.*

Lemma 23.

$$\forall f, g \in \Pi\ \ f \gtrdot g \implies \exists h \in \Pi\ \ (f \succ h \gtrdot^* g)$$

Proof: Let f *and* g *be arbitrary but fixed elements of* Π, *satisfying* $f \gtrdot g$. *Let* $K_f = \{k \in \Pi \mid f \gtrdot k\}$. $K_f \neq \emptyset$ *since* $g \in K_f$. *We prove* $\exists h \in \Pi\ \ (f \succ h \gtrdot^* g)$ *by the induction on the cardinality of* K_f.

If the cardinality of K_f *is 2, i.e. the base case, we are done. Let* $H = \{h \in K_f \mid \forall k \in K_f\ \neg\,(k \gtrdot h)\}$. *Obviously,* $H \neq \emptyset$.

If $g \in H$, *then* $f \succ g$ *by definition of* $\succ$, *and we are done.*

If $g \notin H$, *there exists* $h \in K_f$ *such that* $h \gtrdot g$. *We consider* $K'_f = \{k \in \Pi \setminus \{g\} \mid f \gtrdot k\}$ *and* $f \gtrdot h$ *for our inductive argument. The cardinality of* K'_f *is one less than the cardinality of* K_f. *By the induction hypothesis,* $\exists h' \in \Pi\ \ f \succ h' \gtrdot^* h$. *Therefore, we have* $f \succ h' \gtrdot^* h \gtrdot g$, *and we are done.* □

Proposition 24.

$$\succ^+ \supset \gtrdot^+.$$

Proof: Since Π *is finite and an* $\gtrdot$*-sequence is acyclic by Proposition 19, the relation* $\gtrdot$ *on* Π *is a well-founded order. We therefore prove the claim by the well-founded induction. Let* f *and* g *be arbitrary but fixed elements in* Π *such that* $f \gtrdot^+ g$. *By Lemma 23,*

$\exists h \in \Pi \;\; f \succ h \gtrdot^* g$. *If* $h = g$ *then we are done. This constitutes the base case of the induction.*

We consider next the case $f \succ \;\; h \;\; \gtrdot^* g$ *and* $h \neq g$. *Since* $\gtrdot \supset \succ$, $f \succ h \gtrdot^* g$ *implies* $f \gtrdot h \gtrdot^* \; g$. *By the induction hypothesis* $h \succ^+ g$. *Hence,* $f \succ h \succ^+ g$, *and then* $f \succ^+ g$. $\square$

Theorem 25.

$$\ggg = \succ^+ .$$

Proof: By Proposition 24, we have $\succ^+ \supset \;\; \gtrdot^+ (= \ggg)$. *The relation* $\gtrdot^+ \;\; \supset \succ^+$ *holds by Proposition 17.* $\square$

Theorem 25 tells the following. Although the relation *above*, i.e. $\ggg$, is defined via $\gtrdot$, which is less intuitive notion introduced for the sake of formalization and ease of computation, $\ggg$ is the right notion on which we base our reasoning about the vertical arrangements of faces. The relation $\succ$ is the finest relation to compose the *above* relation. Taking Theorem 25 and Corollary 20 together, we have the following. Given faces f and g in an AO constructed from the initial origami, $f \ggg g$ can be expanded by the finite number of faces $g_1, \ldots, g_n$ such that $g_1 (= f) \succ g_2 \succ \cdots \succ g_n (= g)$.

4. Abstract fold from computational viewpoint

This section is devoted to the algorithmic description of fold. It is a prelude to our argument for the necessity of graph rewriting. When we construct an origami that does not have face overlapping, the operational meaning of folds can be quite simple. The fold is essentially intended to construct a reflection in the fold line. Unfold is similarly understood. In mathematical origami, which is the case in point, some studies have been made in [1, 7]. When an (abstract) origami consists of faces with non-empty superposition relations, the situation is more complex and does not admit a simple algebraic interpretation. Geometrical properties of folds are expounded in [3]. However, the fold performed during the construction of an origami, as has been abstracted as an operation on discrete objects, has not been studied systematically. Having this in mind, let us analyze the abstract fold from computational viewpoint.

Computationally, an origami fold is a composite operation consisting of the following operations.

(1) Specify a fold method and the set F of the faces of concern, i.e. the faces on which the origamist wants to make a fold. We can specify the fold method by one of Huzita's axioms [6] or classical fold methods such as mountain and valley folds with constructed points and lines as arguments.
(2) Compute a fold line and define the associated directed line *ray* r. Through the ray, the notion of left and right of the fold line is made sense of.
(3) Compute the set C of the faces that are the candidates for the face division.
(4) Divide all the faces in C by the ray r and classify the divided and non-divided faces according to the locations relative to r.
(5) Obtain the new set Π of all the faces that constitutes the new origami.
(6) Compute the adjacency relation $\backsim$ on Π.
(7) Compute the superposition relation $\succ$ on Π.

(8) Rotate the relevant faces to the right of r along r.
(9) Compute the new superposition relation $\succ'$ on Π caused by the rotation.
(10) $(\Pi, \backsim, \succ')$ is the new origami created by the fold.

As we saw in Section 2.3, functionally a fold is decomposed into two operations. Steps (1) to (7) constitute one operation which corresponds to the crease relation, and steps (8) and (9) constitutes the other operation corresponding to the basic fold relation. At the completion of step (7), we have a well-defined origami.

More algorithmic description in a procedural program style is given in Algorithms 2 and 3. In both Algorithms 2 and 3, we use a common algorithm **RefTransR**. Given a set $A \subset X$ and a binary relation R on X, it computes the set B of the elements in X that are related by R^* to the elements of the set A. In our case the set X is taken to be Π.

Algorithm 1 RefTransR

Input: A, R
Output: B
1: $B \leftarrow A$
2: **while** $A \neq \emptyset$ **do**
3: Take $f \in A$
4: $W \leftarrow \{g \mid g\ R\ f\} \setminus B$
5: $B \leftarrow B \cup W$
6: $A \leftarrow (A \setminus \{f\}) \cup W$
7: Return B

Algorithm 2 Crease

Input: $(\Pi, \backsim, \succ)$, F, r
Output: $(\Pi', \backsim', \succ')$, D
1: $C \leftarrow$ **RefTransR**$(F, \backsim_{\neg \mathrm{left}(r)})$
2: {**comment** C is the candidate set of the face division}
3: **for all** $f \in C$ **do**
4: Divide f and classify the divided faces
5: {**comment** As the result of the divide-and-classify, we obtain the set D that contains the faces to the right of the ray r}
6: Compute $\backsim'$ on Π'
7: Compute $\succ'$ on Π'
8: Return $(\Pi', \backsim', \succ')$ and D

Algorithm 3 Basic fold

Input: $(\Pi, \backsim, \succ)$, D, r, θ
Output: $(\Pi', \backsim', \succ')$
1: $M \leftarrow$ **RefTransR**$(D, \backsim_{\neg \mathrm{on}(r)} \cup \succ)$
2: {**Comment** M is Π_M}
3: Rotate the faces in M by angle θ along r
4: Compute $\succ'$
5: Return $(\Pi, \backsim, \succ')$

5. Graph formalism for origami

The origami fold as explained in the previous sections shows that the fold is a complex operation. We need to compute the relations $\backsim$ and $\succ$. Furthermore,we need geometrical information that is abstracted away in the formulation of the abstract origami system. One of the promising approach for compromising abstraction and concretization towards the understanding of the geometrical structures of an origami and its implementation is to use graph-theoretic formalism [4, 10]. In particular, we see that a labeled hypergraph is applicable to model our origami.

5.1. Hypergraph

Definition 26 (Hypergraph)**.** A hypergraph is a quadruple (V, E, s, t), where

- V is the set of nodes,
- E is the set of hyperedges, and
- $s, t : E \to V^*$ are source and target functions.

Let $\mathcal{L}_V$ and $\mathcal{L}_E$ be the label alphabets for V and E, respectively, and $\mathbb{L}$ be the set of all the regular expressions over $\mathcal{L}_V$.

Definition 27 (Labeled hypergraph)**.** Given a pair $\mathcal{L} = (\mathcal{L}_V, \mathcal{L}_E)$ of label alphabets together with label constraints $\tau_s, \tau_t : \mathcal{L}_E \to \mathbb{L}$ that constrain labeling of the source and the target nodes of the hypergraph, respectively, an $\mathcal{L}$-labeled hypergraph is a 6-tuple (V, E, s, t, l_V, l_E), where

- (V, E, s, t) is a hypergraph and
- $l_V : V \to \mathcal{L}_V$ and $l_E : E \to \mathcal{L}_E$ are labeling functions that satisfy the following labeling constraint:

$$\forall e \in E \;\; l_V^*(s(e)) \in \tau_s(l_E(e)) \wedge l_V^*(t(e)) \in \tau_t(l_E(e)) \tag{3}$$

The condition (3) ensures the consistency of labeling. To see how it works, let us consider the following simple example.

Example 4. Simple labeled hypergraph Let c be a label of a hyperedge e and $\tau_s(c) = c_1c_2$. Suppose that $s(e) = \langle v_1, v_2 \rangle$. Then, we must have $l_V(v_1) = c_1$ and $l_V(v_2) = c_2$. We can only assign to the nodes v_1 and v_2 a label that satisfies the condition (3).

With another $\tau_s(c) = (c_1 \mid d)c_2$, for $l_E(e') = c$ and $s(e') = \langle v_1', v_2' \rangle$, the labeling function l_V may be such that $l_V(v_1') = d$ and $l_V(v_2') = c_2$. Note that $(c_1 \mid d)c_2$ as well as c_1c_2 are regular expressions denoting the set $\{c_1c_2, dc_2\}$ and the set $\{c_1c_2\}$, respectively.

Example 5. Labeled hypergraph of the initial AO We are given the label alphabets together with label constraints τ_s and τ_t as follows:

$$\mathrm{F} \in \mathcal{L}_V, \mathrm{A} \in \mathcal{L}_E, \tau_s = \{\mathrm{A} \mapsto \mathrm{F}^*\}, \tau_t = \{\mathrm{A} \mapsto \mathrm{F}\}$$

We define a graph G of the initial AO I as an $\mathcal{L}$-labeled graph (V, E, s, t, l_V, l_E), where

$$V = \{f_0\}, E = \{e\}, s = \{e \mapsto f_0\}, t = \{e \mapsto f_0\}, l_E = \{e \mapsto \mathrm{A}\}, l_V = \{f_0 \mapsto \mathrm{F}\}.$$

Example 6. Graph immediately after face division We are given the label alphabets and the label constraints τ_s, τ_t as follows:

$$\mathcal{L}_V = \{\mathrm{F}\}, \mathcal{L}_E = \{\mathrm{A}, \mathrm{R}, \mathrm{L}\}$$

$$\tau_s = \{\mathrm{A} \mapsto \mathrm{F}^*, \mathrm{R} \mapsto \mathrm{F}, \mathrm{L} \mapsto \mathrm{F}\}, \tau_t = \{\mathrm{A} \mapsto \mathrm{F}, \mathrm{R} \mapsto \mathrm{F}, \mathrm{L} \mapsto \mathrm{F}\}.$$

We have an $\mathcal{L}$-labeled graph $G = (V, E, s, t, l_V, l_E)$, where

$$\begin{aligned} V &= \{f_0, f_1, f_2\} \\ E &= \{e_1, e_2, e_3, e_4\} \\ s &= \{e_1 \mapsto f_0, e_2 \mapsto f_0, e_3 \mapsto \langle f_1, f_2\rangle,\ e_4 \mapsto \langle f_2, f_1\rangle\} \\ t &= \{e_1 \mapsto f_1, e_2 \mapsto f_2, e_3 \mapsto f_1, e_4 \mapsto f_2\} \\ l_E &= \{e_1 \mapsto \mathrm{L}, e_2 \mapsto \mathrm{R}, e_3 \mapsto \mathrm{A}, e_4 \mapsto \mathrm{A}\} \\ l_V &= \{f_0 \mapsto \mathrm{F}, f_1 \mapsto \mathrm{F}, f_2 \mapsto \mathrm{F}\} \end{aligned}$$

The hyperedge e_3 forms a cycle from f_1 to f_1, visiting the node f_2 on the way. The hyperedge e_4 also forms a cycle. The graph G is shown in Fig. 9. The graph represents AO O_1' in Fig. 2. In the graph, the circles, rectangles and symbols after colons denote nodes, hyperedges and labels, respectively.

5.2. Graph term

Often graphs are drawn using diagrams. The diagrammatic representation of graphs helps perceive many of properties of graphs, and is indeed effective as long as they are fit into a manageably small space. Graphs for origami become complicated as the construction of an origami proceeds and they do not lend themselves to easy-to-understand drawing in general. Since we are interested in graph rewriting, in addition to the diagrammatic representation that we have just seen, we need a good symbolic representation of graphs for reasoning about the graph transformation, and efficiently manipulating graphs by programs. Thus we are guided to the following symbolic representation of nodes and edges.

Definition 28 (Edge term of a hyperedge)**.** Let e be a hyperedge with $s(e) = \{v_1, \ldots, v_m\}$, $t(e) = \langle w_1, \ldots, w_n\rangle$, and $l_E(e) = c$. The edge term representation of e, denoted by $\widehat{e}$, is a term $c[v_1, \ldots, v_m, w_1, \ldots, w_n]$.

Definition 29 (Node term of a node)**.** Let v be a node with $l_V(v) = c$. The node term representation of v, denoted by $\widehat{v}$, is a term $c[v]$.

The edge term and node term are called *graph terms*, *g-terms* for short. We will later extend the definition of graph term to allow variables and sequence variables to occur as subterms.

Definition 30 (Graph term of a hypergraph)**.** Given an $\mathcal{L}$-labeled hypergraph $G = (V, E, s, t, l_V, l_E)$, graph term representation $\widehat{G}$ of G is a multi-set

$$\{\widehat{e} \mid e \in E\} \cup \{\widehat{v} \mid v \in V\}.$$

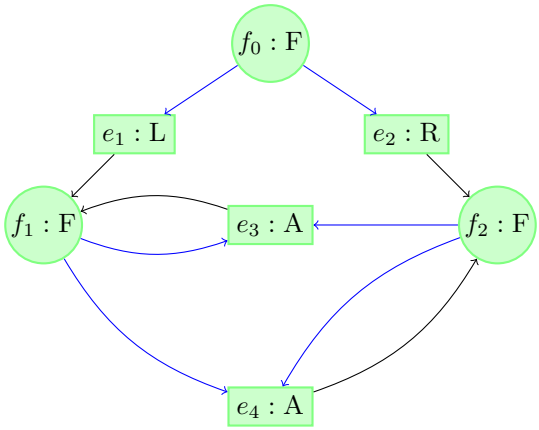


Fig. 9. Hypergraph of an origami created by face division

Note that $\{\widehat{e} \mid e \in E\}$ is a multi-set and hence $\cup$ is the multi-set union; For different e and e' with $s(e) = s(e')$ and $t(e) = t(e')$, the representations $\widehat{e}$ and $\widehat{e'}$ are the same when their labels are the same.

Example 7. G-term representation of a hypergraph Let G be a hypergraph given in Example 6. The g-term representation $\widehat{G}$ is

$$\widehat{G} = \{\mathrm{L}[f_0, f_1], \mathrm{R}[f_0, f_2], \mathrm{A}[f_1, f_2, f_1], \mathrm{A}[f_2, f_1, f_2], \mathrm{F}[f_0], \mathrm{F}[f_1], \mathrm{F}[f_2]\}.$$

In our application, V is implicitly given, and $\mathcal{L}_V$ is a singleton. In such a case we describe $\widehat{G}$ simply as $\{\widehat{e} \mid e \in E\}$, since the affected node terms can be inferred from the edge terms.

6. Graph rewriting

In this section we present a graph rewriting system for origami. We first introduce a language $\mathcal{G}$ for graph rewriting informally. Language $\mathcal{G}$ is embedded in the host language of $\mathcal{G}$, on which we rely for controlling the application of graph rewrite rules as well as for evaluating host functional expressions during the graph rewriting [3]. The syntax of the host expression is of the form $f[t_1, \ldots, t_n]$, and we use this syntax throughout in $\mathcal{G}$ [4]. A basic expression t in $\mathcal{G}$, called *term*, is defined by the following grammar together with the auxiliary notion of an atomic term a:

$$\begin{aligned} t &::= a \mid \langle \text{g-term} \rangle \mid \langle \text{host expr} \rangle \\ a &::= v \mid x \mid \underline{x} \\ \langle \text{host expr} \rangle &::= f[t, \ldots, t] \end{aligned}$$

Here, x denotes a variable, $\underline{x}$ a sequence variable and v a graph node. In this paper we do not give any syntactic specification of v apart from the fact that v denotes a node

[3] To be more specific, in our implementation we use *Mathematica* for the host language.

[4] We use infix notation for commonly used functions, however.

of a graph. A sequence variable is used for a g-pattern (see below for the definition) such as $c[\underline{x}]$. ⟨host expr⟩ means an expression of the host language. It is used during the application of graph rewrite rules, and gives fine control over pattern matching.

A g-term is extended to have variables and sequence variables as subterms, i.e.

$$\langle \text{g-term} \rangle ::= c[a, \ldots, a]$$

A g-pattern $c[\underline{x}]$ matches a g-term $c[s_1, \ldots, s_n]$ of arbitrary $n\,(\geqslant 0)$ arguments. Sequence variables are indispensable constituents in our language since the constructor symbol c in g-term $c[s_1, \ldots, s_n]$, representing a hyperedge whose label is c visiting n nodes, has a flexible arity.

A g-pattern is a g-term possibly with a condition written after "/;".

$$\langle \text{g-pattern} \rangle ::= \langle \text{g-term} \rangle \mid \langle \text{g-term} \rangle \,/; \langle \text{host expr} \rangle$$

The expression of the form $s/;t$ is a *conditional g-pattern*. The conditional g-pattern is used in the context graph part of a graph rewrite rule. During pattern matching with a subgraph by a substitution θ, if $t\,\theta$ is evaluated by the evaluator of the host language to true, $(s/;t)\theta$ reduces to $s\,\theta$, and otherwise it reduces to $\bot$. We use u to denote a g-pattern.

Finally, a graph in $\mathcal{G}$ is a multi-set of g-terms subjected to the conditions for defining a graph.

Definition 31 (Graph rewrite rule)**.** A graph rewrite rule (rewrite rule for short) is a triplet (C, L, R) of multi-sets of g-terms, written as

$$L \,/ : C \to R$$

where

- $C := \{u_1, \ldots, u_m\}$ is a graph called a *context graph*,
- L is a subset of $\{\widetilde{u_1}, \ldots, \widetilde{u_m}\}$, where $\widetilde{u_i} = s_i$, if $u_i = s_i/;t_i$, otherwise $\widetilde{u_i} = u_i$. L is called the left-hand side of the rewrite rule, and
- $R := \{t_1, \ldots, t_n\}$ is a subgraph called the right-hand side of the rewrite rule.

In order to identify the same g-terms in L and C, we can give a name to a g-term. For example, a name n is given to the g-term t in C by writing $n : t$ in C and refer to it as n in L:

$$\{n\} \,/ : \{g[x], n : f[x], h[x]\} \to \{f[x], f[x]\}$$

The occurrence of n in the left-hand side of the rewrite rule refers to $f[x]$ of the context graph of the rewrite rule.

Definition 32 (Graph rewriting)**.** A graph $\widehat{G}$ is rewritten to $\widehat{G}'$ by a rewrite rule $r := L \,/ : C \to R$, denoted by

$$\widehat{G} \Rightarrow_r \widehat{G}'$$

if there exist g-terms $s_1, \ldots, s_m$, and a substitution θ such that $\{s_1, \ldots, s_m\} \subseteq \widehat{G}$, $C\,\theta = \{s_1, \ldots, s_m\}$ after the evaluation of the conditions, if any, and $\widehat{G}' = (\widehat{G} \setminus L\theta) \cup R\theta$. The set-related notations are taken to be those for multi-sets.

The graph rewriting can be formalized as the double push out using graph production

$$\langle C \leftarrow (C \setminus L) \rightarrow (C \setminus L) \cup R \rangle ,$$

where $C \setminus L$ is an interface, as in [4]. However, we prefer our definition of the rewrite rule from the programming language point of view. Thanks to g-terms, it makes clear the parts of the graph involved for rewriting and the graph rewriting becomes a simple multi-set rewriting.

Rewrite rules are combined using the following combinators (i.e. alternative, non-strict sequencing, strict sequencing and repeat) to form a composite rewrite rule. The combinators control the application of the rewrite rules. Hereafter, rewrite rule and composite rewrite rule are collectively called rewrite rules.

Rewrite rule ρ is defined by the following grammar.

$$\rho ::= r \mid \rho_1 \dashv \rho_2 \mid \rho_1; \rho_2 \mid \rho_1 * \rho_2 \mid (\text{Cond } \rho_1\ \rho_2\ \rho_3) \mid (\text{Repeat } \rho)$$

- r is a basic rewrite rule as defined in Definition 31.
- $\rho_1 \dashv \rho_2$ is an alternative.
- $\rho_1; \rho_2$ is a non-strict sequencing.
- $\rho_1 * \rho_2$ is a strict sequencing.
- (Cond ρ_1 ρ_2 ρ_3) is a conditional.
- (Repeat ρ) is a repetition.

We apply a rewrite rule ρ to a graph $\widehat{G}$ to obtain a new graph $\widehat{G}'$. The graph $\widehat{G}'$ is the result of one of the following operations.

(1) The rewrite rule ρ is applicable to some subgraph of $\widehat{G}$, which then is rewritten by ρ, and returns the new graph $\widehat{G}'$. We denote the new graph $\widehat{G}'$ by $\rho[\widehat{G}]$. We call this case successful rewrite.

(2) The rewrite rule ρ is applicable to no subgraph of $\widehat{G}$, and it returns the new graph $\widehat{G}' = \widehat{G}$. We call this case unsuccessful rewrite.

To describe the semantics of the graph rewriting, we need to distinguish the above two cases since by mere observation of the resulting graph we cannot tell whether the graph rewriting has been successful or not. We, therefore, adopt continuation passing style of semantic treatment. We use two continuations as the parameters to the graph rewriting: continuation ν to be obeyed when the rewrite is successful and the continuation κ to be obeyed when the rewrite is unsuccessful.

Then we have the following semantic equations to describe the graph rewriting.

$$r\ G\ \kappa\ \nu = \begin{cases} \nu\ (r[G]) & \text{if the rewrite is successful} \\ \kappa\ G & \text{if the rewrite is unsuccessful} \end{cases} \tag{4}$$

$$(\rho_1 \dashv \rho_2)\ G\ \kappa\ \nu = \rho_1\ G\ (\lambda G'.\ \rho_2\ G\ \kappa\ \nu)\ \nu \tag{5}$$

$$(\rho_1; \rho_2)\ G\ \kappa\ \nu = \rho_1\ G\ \kappa\ (\lambda G'.\ \rho_2\ G'\ (\lambda G'.\ \kappa\ G)\ \nu) \tag{6}$$

$$(\rho_1 * \rho_2)\ G\ \kappa\ \nu = \rho_1\ G\ (\lambda G'.\ \rho_2\ G\ \kappa\ \nu)\ (\lambda G.\ \rho_2\ G\ \nu\ \nu) \tag{7}$$

$$(\text{Cond } \rho_1\ \rho_2\ \rho_3)\ G\ \kappa\ \nu = \rho_1\ G\ (\lambda G'.\ \rho_3\ G\ \kappa\ \nu)\ (\lambda G'.\ \rho_2\ G\ \kappa\ \nu) \tag{8}$$

$$(\text{Repeat } \rho)\ G\ \kappa\ \nu = \rho\ G\ \kappa\ (\lambda G.\ (\text{Repeat}\rho)\ G\ \nu\ \nu) \tag{9}$$

The semantic equation (5) for the alternative states the following. We first apply rule ρ_1 to $\widehat{G}$. If the rewrite is successful, the continuation ν is applied to the result $\rho_1[\widehat{G}]$ of rewriting of $\widehat{G}$ by ρ_1. If the rewrite is unsuccessful, we apply rule ρ_2 to $\widehat{G}$ with the

continuations κ and ν. The semantic equations (6) and (7) for the sequencing rules show that the applications of the rules ρ_1 and ρ_2 are performed sequentially from left to right. In the non-strict sequencing, ρ_2 is not applied if the rewrite by ρ_1 is unsuccessful, and it immediately completes the rewrite unsuccessfully. In the strict sequencing, even if the rewrite by ρ_1 is unsuccessful, the rewrites continue with $\widehat{G}$, and the rewrite of $\widehat{G}$ by ρ_2 is performed. The rewrite of $\widehat{G}$ by $\rho_1 \ * \ \rho_2$ completes successfully either (including both) of rewrites by ρ_1 and ρ_2 are successful. Otherwise the rewrite is unsuccessful. Thus $(r_1 \ * \ r_2)\ \widehat{G}\ \kappa\ \nu$ evaluates to one of the following: $\nu\ (r_2[r_1[\widehat{G}]])$, $\nu(r_2[\widehat{G}])$, $\nu(r_1[\widehat{G}])$ and $\kappa\ \widehat{G}$, whereas $(r_1\ ;\ r_2)\ \widehat{G}\ \kappa\ \nu$ evaluates to one of the following: $\nu\ (r_2[r_1[\widehat{G}]])$ and $\kappa\ \widehat{G}$. The semantic equation (8) states that if ρ_1 rewrites $\widehat{G}$ successfully then the resulting graph is $\rho_2[\widehat{G}]$ else it is $\rho_3[\widehat{G}]$. The semantic equation (9) states the following: (Repeat ρ) rewrites $\widehat{G}$ successfully if there exists an $n \geqslant 1$ such that ρ^n rewrites $\widehat{G}$ successfully and ρ^{n+1} rewrites $\widehat{G}$ unsuccessfully, where ρ^n denotes $\underbrace{\rho\,;\ldots;\,\rho}_{n}$. (Repeat ρ) rewrites $\widehat{G}$ unsuccessfully if ρ rewrites $\widehat{G}$ unsuccessfully.

7. Fold as graph rewriting

We are now ready to describe the fold explained in Section 4 in graph rewriting framework. We follow closely Algorithms **Crease** (Algorithm 2) and **Basic fold** (Algorithm 3) and describe how the graph representing an origami is transformed. In the following, Subsection 7.1 explains step 4, Subsection 7.2 step 6, and Subsection 7.3 step 7 of Algorithm **Crease**. Subsection 7.4 explains steps 3 and 4 of Algorithm **Basic fold**.

7.1. Face division

We consider face division $f \rightarrowtail_r (f_1, f_2)$. An example of the graph obtained by the face division was given in Fig. 9. We will discuss the face division in general cases. Suppose that face f is surrounded by faces v_1, ..., v_n in this order, and that ray r divides the faces v_1, f and v_i. We transform the graph as follows.

(1) Construct nodes f_1 and f_2.
(2) Construct the hyperedge e_1 that connects f with f_1 and e_2 that connects f with f_2. We label the hyperedge e_1 by L (L for Left) and e_2 by R (R for Right).
(3) Construct the hyperedges e_3 and e_4 starting from f_1 and from f_2, respectively. We have $s(e_3) = \langle f_1, v_1, \ldots, v_i, f_2\rangle$, $t(e_3) = f_1$, $s(e_4) = \langle f_2, v_i, \ldots, v_n, v_1, f_1\rangle$ and $t(e_4) = f_2$. We label those hyperedges A (A for Adjacency).

Let $\widehat{G}$ be the g-term of a graph before the division of the face f. The g-term $\widehat{G'}$ of the graph after the division is

$$\{\mathrm{L}[f, f_1], \mathrm{R}[f, f_2], \mathrm{A}[f_1, v_1, \ldots, v_i, f_2, f_1], \mathrm{A}[f_2, v_i, \ldots, v_n, v_1, f_1, f_2], \mathrm{F}[f_1], \mathrm{F}[f_2]\} \ \cup \ \widehat{G}.$$

The label A in $\mathrm{A}[x_1, \ldots, x_n]$ is the realization of the adjacency relation. The intended meaning of $\mathrm{A}[x_1, \ldots, x_n]$ is that $x_1 \backsim x_2, \ldots, x_1 \backsim x_{n-1}$. But at this point, the relations is transient since some of the faces in $\{x_2, \ldots, x_{n-1}\}$ may be divided later during the face division. We will discuss about this in the next subsection. The label R in $\mathrm{R}[x_1, x_2]$ indicates the relation that x_1 is the right (w.r.t. r) part sub-face of x_2. This relation is introduced temporarily during the graph rewriting, and the R-labeled hyperedges will

be deleted (garbage collected) after the graph construction of each step of the origami construction is completed. The label L is used similarly.

We have two special cases that we need to consider besides the above. The first case is that either f_1 or f_2 is empty. In this case, the node corresponding to the empty face is not constructed. For example, if f_1 is empty, we have $f_2 = f$ and the hyperedge e_2 only with $s(e_2) = t(e_2) = f$. The other case is that the ray passes through the vertex (vertices) of the face f. In both cases, the construction of $\widehat{G'}$ is straightforward.

7.2. Computation of adjacency relation

In order to complete the computation of the adjacency relation, the subgraph constructed at the face division step has to be traversed again to update the nodes on the hyperedges. Some of the other faces have been divided later during the face division. The constructed hyperedges still connect to those nodes of the previous non-divided faces. We need to update those hyperedges by the traversal of the graph using the following rewrite rules:

$$\rho_1 := \{n\} \,/\, : \{\mathrm{L}[f, f_1], n : \mathrm{A}[f_1, \underline{x}], \mathrm{L}[g, g_1] / ; (g \neq g_1 \wedge g \in \{\underline{x}\})\} \mapsto \mathrm{A}[f_1, \underline{x}]\{g \mapsto g_1\} \tag{10}$$

$$\rho_2 := \{n\} \,/\, : \{\mathrm{R}[f, f_1], n : \mathrm{A}[f_1, \underline{x}], \mathrm{R}[g, g_1] / ; (g \neq g_1 \wedge g \in \{\underline{x}\})\} \mapsto \mathrm{A}[f_1, \underline{x}]\{g \mapsto g_1\} \tag{11}$$

$$\rho := (\text{Repeat } \rho_1) * (\text{Repeat } \rho_2)$$

The term $\mathrm{A}[f_1, \underline{x}]\{g \mapsto g_1\}$ is the application of the substitution $\{g \mapsto g_1\}$ to a term $\mathrm{A}[f_1, \underline{x}]$. Note that we omit the g-terms for the nodes in all the subgraphs involved.

Example 8. Update of hyperedges after face division Suppose we have a face b surrounded by the faces a_1, a_2, a_3 as shown in Fig. 10 (left). The face division $b \rightarrowtail_r (b_1, b_2)$ is shown in Fig. 10 (right).

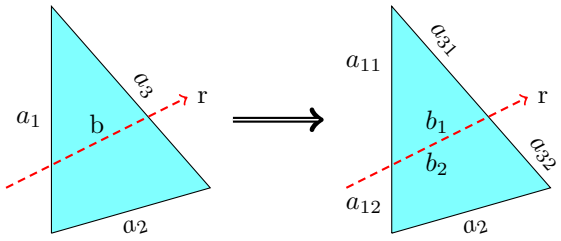


Fig. 10. Node changes after face divisions

At the time of the face division, we have $\mathrm{A}[b_1, a_3, a_1, b_2, b_1]$ and $\mathrm{A}[b_2, a_1, a_2, a_3, b_1, b_2]$ that represent the hyperedges e_1 and e_2 satisfying $s(e_1) = \langle b_1, a_3, a_1, b_2 \rangle, t(e_1) = b_1$, and $s(e_2) = \langle b_2, a_1, a_2, a_3, b_1 \rangle, t(e_2) = b_2$. Later we have $a_1 \rightarrowtail (a_{11}, a_{12})$ and $a_3 \rightarrowtail (a_{31}, a_{32})$. The update of the hyperedges is achieved by rewriting the g-terms $\mathrm{A}[b_1, a_3, a_1, b_2, b_1]$ and $\mathrm{A}[b_2, a_1, a_2, a_3, b_1, b_2]$, to $\mathrm{A}[b_1, a_{31}, a_{11}, b_2, b_1]$ and $\mathrm{A}[b_2, a_{12}, a_2, a_{32}, b_1, b_2]$, respectively.

The instantiated rule from rule (10), i.e. the rule after applying the substitution formed during the pattern matching of the rewrite rule with the graph,

$$\{n\} \,/\, : \{\mathrm{L}[b, b_1], n : \mathrm{A}[b_1, a_3, a_1, b_2, b_1], \mathrm{L}[a_3, a_{31}]\} \rightarrow \{\mathrm{A}[b_1, a_{31}, a_1, b_2, b_1]\}$$

is used to update the g-term $\mathrm{A}[b_1, a_3, a_1, b_2, b_1]$ to $\mathrm{A}[b_1, a_{31}, a_{11}, b_2, b_1]$. Similarly, $\mathrm{A}[b_2, a_1, a_2, a_3, b_1, b_2]$ is rewritten to $\mathrm{A}[b_2, a_{12}, a_2, a_{32}, b_1, b_2]$ by the rewrite rule (11).

7.3. Computation of superposition relation

The superposition relation is computed twice during the fold operation: one induced by the face division and the other by the face rotation. Although both computations may appear different, they are based on the same algorithmic pattern. Namely, in both computations the basic operations are the insertion of a new face to the peaks (to be defined shortly) of the graph and the transitive reduction of the graph.

In order to describe fully these operations, we use the following definitions and notations. Let V be a set and R be a binary relation on V. A pair (V, R) is identified as a directed graph [5]. Let us further stipulate that $G = (V, R)$ is a directed acyclic graph. A graph $G^T = (V, R^+)$ is called a *transitive closure* of the graph G. Let G^- be a directed acyclic graph with a minimum number of edges satisfying $G^T = (G^-)^T$. The graph G^- is called a *transitive reduction* of the graph G. In the following, we consider $G = (\Pi, \succcurlyeq)$. It is easy to see that $G^T = (\Pi, \gg)$ and $G^- = (\Pi, \succ)$ and is unique. We call a node f in Π of G a *peak* iff there exists no node g in Π such that $g \succcurlyeq f$. The set of peaks of a graph G is denoted by $\mathcal{K}_G$.

From a given $O_i = (\Pi_i, \backsim_i, \succ_i)$ of an AO construction $I \looparrowright^* O_i$, we can define a graph $(\Pi_i, \succcurlyeq_i)$. As $\succcurlyeq$ is defined inductively on the construction of AOs, we can construct $\succcurlyeq_{i+1}$ from $\succcurlyeq_i$. Then we take the transitive reduction of $(\Pi_{i+1}, \succcurlyeq_{i+1})$ and finally we extract $\succ_{i+1}$ from the transitive reduction. The known algorithm for the transitive reduction of a directed acyclic graph $G = (V, R)$ can be as low as $O(n^3)$, where n is the cardinality of V [5]. Actually, it suffices to construct a graph (Π_{i+1}, R) with a relation R such that $\succ_{i+1} \subset R \subset \succcurlyeq_{i+1}$. The crucial observation here is that if $f \succcurlyeq g$ then for any h such that $g \succcurlyeq^+ h$, we have $\neg(f \succ h)$. Therefore, once the edge representing $f \succcurlyeq g$ is established, we do not have to consider the possible edges representing $f \succcurlyeq h$ because those edges would be unnecessary to construct the transitive reduction.

We define the above algorithm with the set of graph rewrite rules in the following way. Suppose we have a unique g-term NewNode[f] in $\widehat{G}$. We define the graph rewrite rule:

$$\mathrm{AddS}(h) := \{\}/ : \{\mathrm{NewNode}[f]/; f \asymp h\} \rightarrow \{\mathrm{S}[f, h]\}.$$

The rewrite rule states that we can create an S-labeled edge from f to h when $f \asymp h$. Note, however, that some of the S-labeled edges thus constructed may be deleted during the subsequent transitive reduction. The remaining ones after the transitive reduction are the edges representing the superposition relation. The node f is to be added to the peaks of G. To find the node to which we construct an edge from f is a simple graph traversal. For this purpose we incorporated the graph traversal strategy Trav in our language $\mathcal{G}$. The semantics of Trav can be described by the following semantic equation.

[5] In this subsection, in order to explain the essential idea of the computation of the superposition, we employ an ordinary graph. Actual implementation is done in the labeled hypergraphs that have been discussed so far.

$$\text{Trav } \rho\ \{\}\ G\ \kappa\ \nu = \nu\ G$$
$$\text{Trav } \rho\ (\{e\} \cup es)\ G\ \kappa\ \nu = \rho\ e\ G\ (\lambda G.\ \text{Trav } \rho\ (\text{Succ } e\ G)\ G\ \nu'\ \nu')\ \nu'$$
where $\nu' = \lambda G.\ \text{Trav } \rho\ es\ G\ \nu\ \nu$
and (Succ e G) computes the set of nodes x of G
such that $e \succ x$

Note that (Succ e G) finds the set of successor nodes using $\succ$ rather than $\succcurlyeq$ since any node connected by $\succcurlyeq$ from e can be reached by multiple edges of $\succ$ from e by Proposition 24. Let $\text{Tr}(G)$ be a function to compute the transitive reduction of a graph G. Then the following program, receiving a graph G, returns the desired graph.

$$\text{Trav AddS } \mathcal{K}_G\ G\ \kappa\ \text{Tr}$$

Example 9. Superposition after making a crease
We consider $O = (\Pi, \backsim, \succ) \overset{\rightsquigarrow}{\to} O' = (\Pi', \backsim', \succ')$ as shown in Fig. 11. We see that $\Pi = \{f, g\}$, $\backsim = \{f, g\}$ and $\succ = \{f, g\}$. By the face division, we have $f \rightarrowtail (f_1, f_2)$ and $g \rightarrowtail (g_1, g_2)$. Whether or not $f_1 \succcurlyeq' g_1$ holds in O' depends on whether faces f_1 and g_1 overlap. In the case of the fold along r_1, we have $f_i \succcurlyeq' g_i$ for $i = 1, 2$, but in the case of the fold along r_2, neither $f_1 \succcurlyeq' g_1$ or $g_1 \succ \succcurlyeq' f_1$ holds. Let us consider the latter case further. Figure. 12 depicts a subgraph of the graph representing O'. In $O' = (\Pi', \backsim', \succ')$,

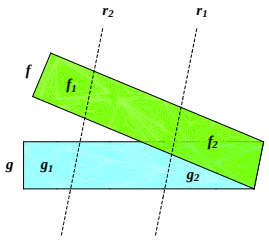


Fig. 11. Superposition relation after face division

$\Pi' = \{f_1, f_2, g_1, g_2\}$, $\backsim = \{(f_1, f_2), (g_1, g_2), (f_2, g_2)\}$ and $\succ = \{(f_2, g_2)\}$. The hyperedge e_1 is labeled L since the f_1 is to the left of f, and the e_2 labeled R. Similarly, e_3 and e_4 are labeled L and R, respectively. The hyperedge e_5 is labeled S since $f \succ g$. The hyperedge e_6 is added. This hyperedge is labeled S since they realize the superposition relation after the face division. In the figure the other incoming edges of the A-labeled hyperedges are omitted.

7.4. Computation of superposition relation induced by the rotation

The final stage of the fold is the rotation of faces. It consists of steps 3 and 4 of Algorithm **Basic fold** (Algorithm 3). Prior to the face rotation, we check the foldability. Namely, we check whether the fold line passes through the interior of a face. If true, we cannot make a fold. The rotation at step 3 induces the changes in the coordinates of the vertices of the moved faces. This invokes numerical computation of the coordinates, on

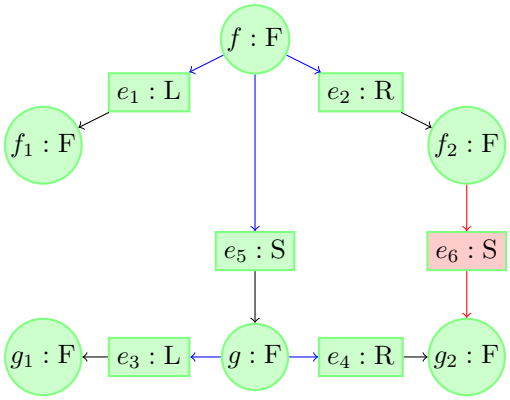


Fig. 12. Insertion of a superposition hyperedge

one hand, and symbolic computation of the reflection relation between the vertices before and after the rotation, on the other. These computations do not change the structure of the graph. We construct S-labeled hyperedges using the rewrite rule Trav explained in Subsection 7.3 for finding the the pair of faces that should be connected by S-labeled hyperedges. The algorithm is similar to the one for step 7 of Algorithm **Crease**. Finally, we take the transitive reduction of the constructing graph.

Example 10. Superposition by rotation Figure 13 shows the origami after making creases. The ray r corresponding to the fold line runs from lower right to upper left. The faces on the base layer of the origami have been divided by the ray r into faces f and a. At this step, the origami consists of faces a, b, c, d, f and g. We see that the faces a, b, c and d are to the right of r, and the faces f and g to the left of r. Figure 14 shows the result of the rotation along r. Face g superposes face f. Faces b and c superpose a. Face d superposes b.

Figure 15 shows the graph of the origami of Fig. 13. The rotation induces the graph transformation from the graph of Fig. 15 to that of Fig. 16. The newly added S-labeled hyperedges are e_5, e_6, e_7, e_8 and e_9. For simplicity we omit the hyperedges of the other labels.

8. Conclusions

We have presented an abstract model for origami. Central to the modeling is the abstraction of fold. The abstraction of fold led to graph-theoretic modeling of an origami and transformation of the graph representing the origami. By this formalism we are able to rigorously construct an origami and reason about the process of construction of the origami as well as the geometrical properties of constructed origami. Furthermore, it has been shown that the graph-theoretic formalism has the advantage of separating domains of discourse into pure combinatorial domain and geometrical domain $\mathcal{R} \times \mathcal{R}$.

We have also presented a language of graph rewriting. The language enables us to describe the process of graph transformation algebraically. Our formalism follows closely that of algebraic and categorical graph theories.

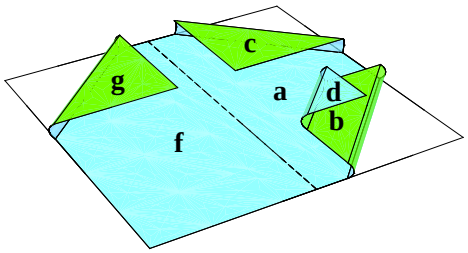


Fig. 13. Origami after crease

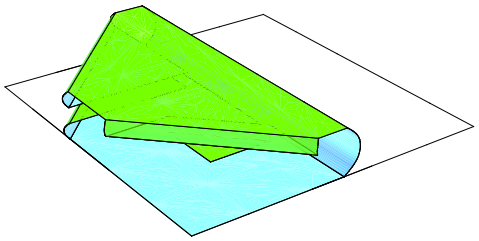


Fig. 14. Origami after basic fold

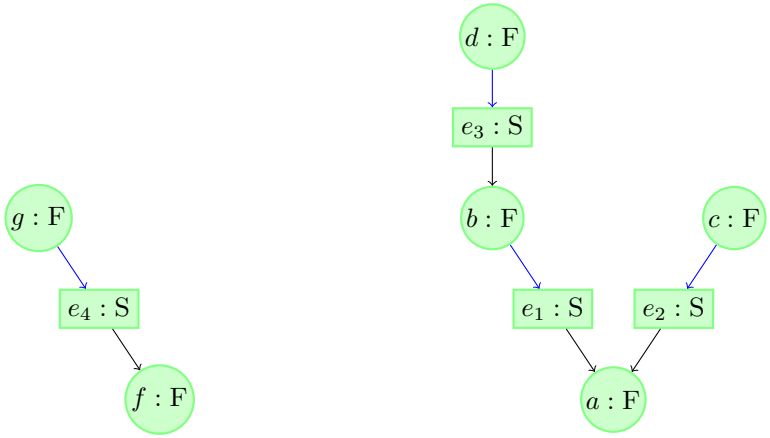


Fig. 15. Graph of origami of Fig. 13

The graph contains necessary information to explore other possible methods of origami constructions. We are thus in a position to tackle challenging problems such as of discovering a new construction given an origami shape, and of discovering a new origami method that has certain geometrical properties employing various AI techniques.

References

[1] R. C. Alperin. A Mathematical Theory of Origami Constructions and Numbers. *New York Journal of Mathematics*, 6:119–133, 2000.

[2] E. D. Demaine and M. L. Demaine. Recent Results in Computational Origami. In *Proceedings of the Third International Meeting of Origami Science, Mathematics and Education*, pages 3–16. A K Peters, Ltd., 2002.

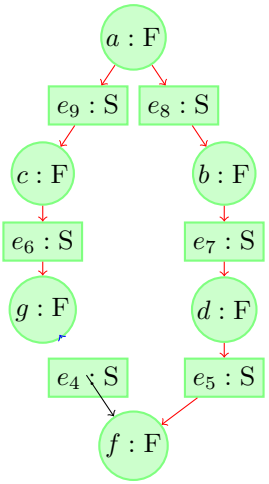


Fig. 16. Graph of origami of Fig. 14

[3] E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, New York, 2007.

[4] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer-Verlag, 2006.

[5] D. Gries, A. J. Martin, J. L. A. van de Snepscheut, and J. T. Udding. An Algorithm for Transitive Reduction of an Acyclic Graph. *Science of Computer Programming*, 12(2):151–155, 1989.

[6] H. Huzita. Axiomatic Development of Origami Geometry. In H. Huzita, editor, *Proceedings of the First International Meeting of Origami Science and Technology*, pages 143–158, 1989.

[7] T. Ida, M. Marin, H. Takahashi, and F. Ghourabi. Computational Origami Construction as Constraint Solving and Rewriting. In *Proceedings of the 16th International Workshop on Functional and (Constraint) Logic Programming*, volume 216 of *Electronic Notes in Theoretical Computer Science*, pages 31–44. Elsevier B.V., 2008.

[8] T. Ida, H. Takahashi, M. Marin, and F. Ghourabi. Modeling Origami for Computational Construction and Beyond. In *Proceedings of the 2007 International Conference on Computational Science and Its Applications*, volume 4706 of *Lecture Notes in Computer Science*, pages 653 – 665. Springer-Verlag, 2007.

[9] T. Ida, H. Takahashi, M. Marin, F. Ghourabi, and A. Kasem. Computational Construction of a Maximal Equilateral Triangle Inscribed in an Origami. In *Proceedings of the Second International Congress on Mathematical Software*, volume 4151 of *Lecture Notes in Computer Science*, pages 361–372. Springer-Verlag, 2006.

[10] H. J. Schneider. *Graph Transformations - An Introduction to the Categorical Approach* - . 2008. published on the web as a preliminary text of the planned book.