Origami Fold as Algebraic Graph Rewriting

Tetsuo Ida
ida@score.cs.tsukuba.ac.jp

Hidekazu Takahashi
hidekazu@score.cs.tsukuba.ac.jp

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

ABSTRACT

We formalize paper fold (origami) by graph rewriting. Origami construction is abstractly described by a rewriting system \((\mathcal{O}, \leftrightarrow)\), where \(\mathcal{O}\) is the set of abstract origami and \(\leftrightarrow\) is a binary relation on \(\mathcal{O}\), called fold. An abstract origami is a triplet \((\Pi, \prec, \succ)\), where \(\Pi\) is a set of faces constituting an origami, and \(\prec\) and \(\succ\) are binary relations on \(\Pi\), each representing adjacency and superposition relations between the faces.

We then address representation and transformation of abstract origami’s and further reasoning about the construction for computational purposes. We present a hypergraph abstract of origami and define origami 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 combinatoric domain and geometric domain \(\mathbb{R} \times \mathbb{R}\), and thus helps us to further tackle challenging problems in computational origami research.

Categories and Subject Descriptors

G.2.2 [Mathematics of Computing]: Discrete Mathematics—Graph Theory; I.3.5 [Computing Methodologies]: Computer Graphics—Computational Geometry and Object Modeling

Keywords

origami, geometric modeling, hypergraph, graph rewriting

1. INTRODUCTION

The art of paper folding, known as origami, provides the methodology of constructing a geometrical object out of a piece 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 geometric objects, e.g., finding intersection of lines and checking the overlap of two faces, on one hand, and purely combinatoric 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 [6]. Indeed the implementation of computational origami system Eos [7] 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 basic origami operations. In Section 3 we will explain the bases for graph-theoretic modeling of origami. In Section 4, we will show how origami fold is formalized in the algebraic graph-theoretic framework. In Section 5, we will give one result of the application of our formalism. In Section 6, we will summarize the results and point out the direction of the research.

2. FORMALIZING ORIGAMI

2.1 Origami at a glance

We start an origami construction with a single piece of paper, and repeat folding of 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 exhibit 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 observe 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.

**2.2 Abstract origami**

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 consider a finite set \(\Pi\) of faces to be the object of our study, and introduce two binary relations on \(\Pi\), expressing horizontal and vertical arrangements of faces rather than a single binary relation \(R\), 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.

**Definition 2.1 (Abstract origami).** An abstract origami is a structure \((\Pi, \prec, \succ)\) where

- \(\Pi\) is the finite set of faces, and
- \(\prec\) and \(\succ\) are binary relations on \(\Pi\), called adjacency and superposition relations, respectively.

Now we use the notion of abstract rewrite system (ARS) to define the abstract origami system.

**Definition 2.2 (Abstract origami system).** An abstract origami system is an abstract rewrite system \((\mathcal{O}, \Rightarrow)\) where

- \(\mathcal{O}\) is the set of abstract origami’s.
- \(\Rightarrow\) is the binary relation on \(\mathcal{O}\) called abstract fold.

When \(O, O’ \in \mathcal{O}\) are related by the abstract fold relation we write \(O \Rightarrow O’\).

Origami construction proceeds stepwise. We start with an initial origami \(O_0\) and perform folds along fold lines, repeatedly until we obtain a desired shape. Suppose that we are at the beginning of step \(i\) of the construction, having an origami \(O_{i-1} = (\Pi_{i-1}, \prec_{i-1}, \succ_{i-1})\). We make the next fold and obtain the next origami \(O_i = (\Pi_i, \prec_i, \succ_i)\). Thus we have the following:

**Definition 2.3 (Abstract origami construction).** An abstract origami construction is a finite sequence of abstract origami’s satisfying

\[O_0 \Rightarrow O_1 \Rightarrow \cdots \Rightarrow O_n, \text{ where } O_0, O_1, \ldots, O_n \in \mathcal{O}\]

**2.3 Geometric structure**

Although some properties of origami can be studied with necessary rigor at this level, more geometric information is needed to understand many of the properties of origami. We are thus lead to the definition of face given in Definition 2.4.

Before we proceed, we note the following definition of \(n\)-gon. An \(n\)-gon \((n \geq 3)\) is a polygon consisting of \(n\) edges none of which intersect each other. We further use the notion of overlapping. Let the expression \(p \sim q\) denote the interior of an \(n\)-gon \(p\). We identify the interior of an \(n\)-gon with the set of all the points in the interior. \(n\)-gons \(p\) and \(q\) are called overlapping if \(p \cap q \neq \emptyset\).

**Definition 2.4 (Face).** A face is a convex \(n\)-gon.

Then we can define the adjacency relation as follows:

**Definition 2.5 (Face adjacency).** Two faces are adjacent if they share an edge.

We can determine whether a face is adjacent to the other face if an adequate representation is used for edges and faces.

We now represent an \(n\)-gon as a sequence of points \((P_1, \ldots, P_n)\), where points \(P_1, \ldots, P_n\) are vertices of the \(n\)-gon. A face is thus represented as a sequence of points. When points \(P_1, \ldots, P_n\) are arranged counterclockwise, we say that the face is up, and when clockwise, it is down.

Concerning the superposition relation, we assume the existence of a decision procedure to determine above or below relation among the faces. Namely, for any two faces \(f\) and \(g\), we can determine one of three situations: \(f\) is above \(g\), \(g\) is above \(f\), and \(f\) and \(g\) are not related by the relation ‘above’.

Let ‘below’ be the inverse relation of ‘above’. Then we have the following definition of the superposition relation.

**Definition 2.6 (Face superposition).** Face \(f\) superposes face \(g\) iff \(f\) and \(g\) are overlapping, \(f\) is above \(g\) and no faces that are above \(g\) is below \(f\).

**2.4 From abstract fold to concrete fold**

When we fold an origami paper that does not have face overlapping, the operational meaning of fold is quite simple. Namely the fold is essentially a reflection along the fold line. Unfold is similarly understood. In a mathematical origami where we are interested in generating points of intersection of face edges, in-depth studies have been made [1, 5]. However, when the (abstract) origami consists of faces with non-empty superposition relations, it does not admit a simple algebraic interpretation. Origami fold is a complex operation consisting of the following sub-operations.

- Specify a basic fold operation together with the set \(\mathcal{C}\) of the faces of concern, i.e. the faces that the origamist wants to apply fold. We use one of Huzita’s basic folds [4] or classical fold methods such as mountain and valley folds.
In the figure, the face \( f \) right of \( r \) is made sense of. Through the ray, the notion of left and right of the fold line is made sense of.

For each face \( f \) in \( \mathcal{C} \), do the following until \( \mathcal{C} = \emptyset \).
- Divide \( f \) by the ray \( r \).
- Update \( \mathcal{C} \) by removing \( f \) from \( \mathcal{C} \) and adding to \( \mathcal{C} \) the faces that are affected by this division using the superposition and adjacency relations.

Obtain the new set \( \mathcal{F} \) of all the faces that constitutes the new origami.

Update adjacency relation on \( \mathcal{F} \) caused by the division.

Compute new superposition relation on \( \mathcal{F} \) caused by the division.

Rotate the relevant faces to the right of \( r \) along \( r \).

Compute new superposition relation on \( \mathcal{F} \) caused by the rotation.

Let us add a comment on our treatment of the division of a face. Suppose we divide \( f \) by the ray \( r \). When \( r \) intersects with the edges of \( f \) at two distinct points, the face \( f \) is divided into the pair of faces \( \langle f_1, f_2 \rangle \), where \( f_2 \) is to the right of \( r \). Otherwise, the face \( f \) is not split into two faces. Even in this latter case, we say that \( f \) is divided into \( \langle f_1, f_2 \rangle \), where either \( f_1 \) or \( f_2 \) (but not both) is \( f \) and the other is the empty face. The division of an up face is illustrated in Fig. 2. In the figure, the face \( f \) is divided by the ray \( r \) into \( \langle f_1, f_2 \rangle \). The face \( f_2 \) is to be rotated. This case is further investigated for the graph transformation in the next section.

3. GRAPH FORMALISM FOR ORIGAMI

3.1 Hypergraph and graph term

To make origami amenable to computation, we further concretize the abstract origami by graph-theoretic formalism. We use a labeled hypergraph for this purpose. Since we do not need algebraic graph theories in full generality such as discussed in [3], we work with hypergraphs defined as follows.

**Definition 3.1 (Hypergraph).** A hypergraph is a quadruple \((V, E, s, t)\), where
- \( V \) is the set of nodes\(^1\),
- \( E \) is the set of hyperedges, and
- \( s, t : E \rightarrow V^* \) are source and target functions.

**Definition 3.2 (Labeled hypergraph).** Given a pair \( \mathcal{L} = (\mathcal{L}_V, \mathcal{L}_E) \) of label alphabets together with functions \( \tau_s, \tau_t : \mathcal{L}_E \rightarrow \mathcal{L}_V^* \), 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
- \( l_V : V \rightarrow \mathcal{L}_V \) and \( l_E : E \rightarrow \mathcal{L}_E \) are functions satisfying \( \tau_s \circ l_E = l_V \circ s \) and \( \tau_t \circ l_E = l_V \circ t \) \((3.1)\).

Hereafter, we only consider hypergraphs, and hence the prefix “hyper” may be omitted.

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 admit easy-to-understand drawing in general. Furthermore, we are interested in graph transformation, as well as graphs themselves. Therefore, we need a yet another representation of graphs with which we can reason about the graph transformation easily.

We define symbolic representation of a hyperedge as follows. Let \( e \) be a hyperedge with \( s(e) = \langle v_1, \ldots, v_m \rangle \) and \( t(e) = \langle w_1, \ldots, w_n \rangle \). Since we work with labeled graphs, we will use labels in describing their hyperedges and nodes. We use labels as constructor symbols for terms. Then, we can represent the hyperedge \( e \) as \( e[v_1, \ldots, v_m, w_1, \ldots, w_n] \). We call the symbolic representation \( e[v_1, \ldots, v_m, w_1, \ldots, w_n] \) graph term (g-term for short) of \( e \). Let \( \hat{e} \) denote the g-term of \( e \). Likewise we define \( \hat{v} \) for \( v \in V \).

**Definition 3.3 (Graph term representation).** Given an \( \mathcal{L} \)-labeled graph \( G = (V, E, s, t, l_V, l_E) \), graph term representation \( \hat{G} \) of a graph \( G \) is defined as
\[
\{ \hat{e} \mid e \in E \} \cup \{ \hat{v} \mid v \in V \} \quad (3.2)
\]
In (3.2), \( \{ \hat{e} \mid e \in E \} \) 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 denotations \( \hat{e} \) and \( \hat{e'} \) are the same when their labels are the same.

**Example 1.** Let \( \mathcal{L}_V \) and \( \mathcal{L}_E \) be \{F\} and \( \{A, R, L\} \), respectively, and let \( G \) be an \( \mathcal{L} \)-labeled graph \((V, E, s, t, l_V, l_E)\), where
- \( V = \{f_1, f_2, v_1, \ldots, v_n\} \)
- \( E = \{e_1, e_2, e_3, e_4\} \)
- \( s = \{e_1 \mapsto f_1, e_2 \mapsto f_2, e_3 \mapsto \langle f_1, v_1, \ldots, v_i, f_2 \rangle, e_4 \mapsto \langle f_2, v_i, \ldots, v_n, v_1, f_1 \rangle\} \)
- \( t = \{e_1 \mapsto f_1, e_2 \mapsto f_2, e_3 \mapsto f_1, e_4 \mapsto f_2\} \)
- \( l_E = \{e_1 \mapsto L, e_2 \mapsto R, e_3 \mapsto A, e_4 \mapsto A\} \)
- \( l_V = \{f_1 \mapsto F, f_2 \mapsto F, f \mapsto F, v_1 \mapsto F, \ldots, v_n \mapsto F\} \).

The labels may appear ad-hoc at this point, but they are given meaning later.

The g-term representation \( \hat{G} \) is:
A sequence variable is a g-term possibly with a condition of the form \( s/\theta \), where \( s \) is a sequence variable. The sequence variable is indispensable in our language since it allows a hyperedge whose label is a sequence variable to occur at the argument positions, i.e.

\[
(g\text{-term}) := c[a, \ldots, a]
\]

A g-pattern \( c[x] \) matches with a g-term \( c[s_1, \ldots, s_n] \) of arbitrary \( n \geq 0 \) arguments. The sequence variable is indispensable in our language since a constructor symbol \( c \) in a g-term \( c[s_1, \ldots, s_n] \) represents a hyperedge whose label is \( c \) visiting \( n \) nodes, has a flexible arity.

A g-pattern is a g-term possibly with a condition \( t \):

\[
(g\text{-pattern}) := (g\text{-term}) \mid (g\text{-term}) \mid t
\]

The expression of the form \( s/\theta \) is called a conditional g-pattern. The conditional g-pattern is used in the context part of a graph rewrite rule. It is used for conditional pattern matching. During pattern matching with a subgraph by a substitution \( \theta \), if \( t \theta \) is matched by the evaluator, the host language to true, \((s/\theta) \theta \) reduces to \( s \theta \), and otherwise it reduces to \( \perp \). We use \( u \) to denote either a g-term or 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 3.4 (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

\[
L / : C \rightarrow R
\]

where:

- \( C := \{u_1, \ldots, u_m\} \) is a graph called a context graph,
- \( L \) is a subset of \( \{\tilde{u}_1, \ldots, \tilde{u}_m\} \), where \( \tilde{u}_i = s_i \), if \( u_i = s_i/\theta \); otherwise \( \tilde{u}_i = u_i \). \( L \) is called the left-hand side of the rewrite rule,
- \( R := \{t_1, \ldots, t_n\} \) 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 \( L \) by writing \( n : t \) in \( L \) and refer to it as \( n \) in \( C \). Then, we can write, for example, \( \{n : f[x]\} / \{g[x], n, h[x]\} \rightarrow \{f[x], f[x]\} \). The occurrence of \( n \) in the context graph refers to \( f[x] \) of the left-hand side of the rewrite rule.

**Definition 3.5 (Graph rewriting).** A graph \( \hat{G} \) is rewritten to \( \hat{G}' \) by a rewrite rule \( r := L / : C \rightarrow R \), denoted by

\[
\hat{G} \Rightarrow_r \hat{G}'
\]

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

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

\[
(C \leftarrow (C \setminus L) \rightarrow (C \setminus L) \cup R),
\]

where \( C \setminus L \) is an interface, as in [3]. 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.

### 4. FOLD AS GRAPH REWRITING

We are now ready to describe the fold explained in Subsection 2.4 in graph rewriting framework.

#### 4.1 Face division

We consider the division of a face \( f \) into \( f_1 \) and \( f_2 \) by a ray \( r \) as shown in Figs. 2 and 3. Figure 3 is the subgraph of the entire graph of the origami that we are working on.

The graph was transformed in the following steps from the graph of the previous step:

---

**Figure 3: Graph of an origami created by face division**

\( \hat{G} = \{ L[f, f_1], R[f, f_2], A[f, v_1, \ldots, v_n, v_1, f_1, f_2], F[f], F[f_1], F[f_2], F[v_1], \ldots, F[v_n], F[v_{n+1}] \} \)

To be precise, the two A labels have to be distinguished if the numbers of the arguments of A’s are different, in order for (3.1) to hold true. This could be done by properly indexing the labels, but for clarity we omitted the indexing.

The graph \( G \) is shown in Fig. 3. In the graph the node \( v \) with label Label is represented by a circled \( v:Label \), and hyperedge \( c \) with label Label by a boxed \( c:Label \). This graph represents the face division given in Fig. 2, where \( V \) is the set of the faces.

### 3.2 Graph rewriting

In this subsection we present a language, to be denoted as \( \mathcal{G} \), for graph rewriting. The language \( \mathcal{G} \) is embedded in a general purpose programming language, i.e. the host language of \( \mathcal{G} \), upon which we rely for controlling the application of graph rewrite rules as well as for evaluating functional expressions in a graph rewrite rule. To be more specific, we use Mathematica for the host language. The syntax of a functional expression is of the form \( f[t_1, \ldots, t_n] \), and we use this syntax throughout.

A g-pattern is a g-term possibly with a condition of the form \( s/\theta \); to allow a variable and a sequence variable to occur at the argument positions, i.e.

\[
(g\text{-term}) := c[a, \ldots, a]
\]

A g-pattern \( c[x] \) matches with a g-term \( c[s_1, \ldots, s_n] \) of arbitrary \( n \geq 0 \) arguments. The sequence variable is indispensable in our language since a constructor symbol \( c \) in a g-term \( c[s_1, \ldots, s_n] \) represents a hyperedge whose label is \( c \) visiting \( n \) nodes, has a flexible arity.

A g-pattern is a g-term possibly with a condition \( t \):

\[
(g\text{-pattern}) := (g\text{-term}) \mid (g\text{-term}) \mid t
\]

We use infix notation for commonly used functions, however.
1. Construct nodes $f_1$ and $f_2$.

2. Construct the hyperedges $e_1$ that connects $f$ with $f_1$ and $e_2$ that connects $f$ with $f_2$. The hyperedge $e_1$ is labeled $L$ (L for Left) since face $f_1$ is to the left of the ray $r$, and $e_2$ is labeled $R$ (R for Right).

3. Construct the hyperedges $e_3$ and $e_4$ issuing from $f_1$ and from $f_2$, respectively. We have $s(e_3) = (f_1, v_1, \ldots, v_n, f_2)$, $t(e_3) = f_1$, $s(e_4) = (f_2, v_1, \ldots, v_n, r_1, f_1)$ and $t(e_4) = f_2$. We label those hyperedges by $A$ (A for Adjacency) since the constructed hyperedges represent the adjacency relation.

In the case that either $f_1$ or $f_2$ is empty, the node corresponding to the empty node is not created. Suppose that $f_1$ is empty. Then we have $f_2 = f$ and we have the hyperedge $e_2$ only with $s(e_2) = t(e_2) = f$.

### 4.2 Update of adjacency relation

The graph constructed in the face division step has to be updated; some of other faces are also divided later in the face division process, but the hyperedges still connect to those nodes of the previous faces. At this step we perform this face update by the following rewrite rules:

$$
\{n: A[f_1, \mathcal{V}] \}/: \{ L[f_1, i], n, L[g, g_1] / (g \neq g_1 \land g \in \{\mathcal{V}\}) \} \\
\rightarrow A[f_1, \mathcal{V} (g \rightarrow g_1)]
$$

$$
\{n: A[f_1, \mathcal{V}] \}/: \{ R[f, f_1], n, R[g, g_1] / (g \neq g_1 \land g \in \{\mathcal{V}\}) \} \\
\rightarrow A[f_1, \mathcal{V} (g \rightarrow g_1)]
$$

Note that we omit the $g$-terms for the nodes in all the subgraphs involved. This is allowed in our language since the nodes are unchanged by the rewriting.

![Figure 4: Change of faces by division](image)

**Example 2.** Suppose we have a face $b$ surrounded by the faces $a_1, a_2, a_3$ as shown in Fig. 4 (left). The face $b$ is divided by the ray $r$ into $\{b_1, b_2, b_3\}$ as shown in Fig. 4 (right). At the time of the division, we have $A[b_1, a_3, a_1, b_2, b_1]$ and $A[b_2, a_1, a_3, b_1, b_2]$ that represent the hyperedges $e_1$ and $e_2$ satisfying

$$s(e_1) = (b_1, a_3, a_1, b_2), t(e_1) = b_1$$

$$s(e_2) = (b_2, a_1, a_3, b_1), t(e_2) = b_2.$$

Faces $a_1$ and $a_3$ are divided into $\{a_1, a_2\}$ and $\{a_3, a_3\}$, respectively. This transformation is achieved by the rewriting of the $g$-terms from

$A[b_1, a_3, a_1, b_2, b_1]$ to $A[b_1, a_3, a_1, b_2, b_1]$ and

$A[b_2, a_1, a_3, b_1, b_2]$ to $A[b_2, a_1, a_3, b_1, b_2]$.

The rule (4.1) performs the rewriting of the former, and the rule (4.2) performs the rewriting of the latter. The instantiated rules, i.e. the rules after applying the substitution formed during the pattern matching of the rewrite rule and the graph:

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

and

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

from rule (4.1) are used to update the $g$-term $A[b_1, a_3, a_1, b_2, b_1]$.

### 4.3 Update of superposition relation induced by division

Suppose that faces $f$ and $g$ satisfying $f \cong g$ are divided into $(f_1, f_2)$ and $(g_1, g_2)$, respectively. We should note that the relation $\cong$ is not preserved by the face division on the faces created by the division. Namely, we do not necessarily have $f_1 \cong g_i$ for $i = 1$ and $2$ even if $f \cong g$. The reason is that $f_i \cap g_i \neq \emptyset$ may not always hold true. See the example in Fig. 5. In the case of the fold along $r_1$, the relation $\cong$ is preserved for $f_1$ and $g_i$ for $i = 1, 2$, but in the case of the fold along $r_2$, neither $f_1 \cong g_1$ or $g_1 \cong f_1$ holds. Therefore, in general the check of overlap among the faces created by the division are necessary.

![Figure 5: Superposition relation on the divided faces](image)

When $f_1 \cong g_1$ and $f_2 \cong g_2$, the graph transformation is straightforward. In the graph of Fig. 6, the hyperedge $e_5$ is labeled $S$ (S for Superposition) since $f \cong g$. The hyperedges $e_6$ and $e_7$ with label $S$ are added. The A-labeled hyperedges are omitted in the figure.

This transformation is realized by the following rewrite rule:

$$\{ \} /: \{ S[f, g_1], L[f, f_1], R[f, f_2], L[g, g_1] / f_1 \cap g_1 \neq \emptyset, R[g, g_2] / f_2 \cap g_2 \neq \emptyset \} \\
\rightarrow \{ S[f, g_1], S[f, f_2] \}$$

### 4.4 Rotation of faces and update of superposition relation

The final step of fold is the rotation of faces. The rotation induces changes in the coordinates of the vertices of the faces that are rotated. This will further invoke numerical computation of the coordinates, on one hand, and symbolic computation of the reflection relation between the vertices before the rotation and the vertices after the rotation. These computations do not change the structure of the graph.
However, after these computations, the rotation process does require the check of the superposition relation for any pair of moved and non-moved faces, and the modification of the superposition relation accordingly.

The graph rewriting is performed as follows, by distinguishing the following three cases. Suppose we have a pair of faces \( f \) and \( g \) and a fold ray \( r \).

1. Faces \( f \) and \( g \) are to the right of and to the left of \( r \), respectively: If \( f \cap g \neq \emptyset \) and no other faces above \( g \) is below \( f \), then \( f \succ g \) is formed.

2. Faces \( f \) and \( g \) are to the right of \( r \) and moved: If \( f \succ g \), the relation \( f \succ g \) is deleted and \( g \succ f \) is formed.

3. Other cases: No new superposition is formed.

5. APPLICATION

The abstraction discussed in this paper is being applied to revise the engine of Eos. It enabled us to reconstruct the system code in a lucid and modularized way. The system is now configured to consist of the following modules: the constraint solver for obtaining fold lines, the reasoner for theorem proving, the graph transformer, visualizer and others for basic geometric computations.

The construction of the origami cicada shown in Fig. 1 is produced by the new engine of Eos. The graphs of the final origami are shown in Fig. 7. We show two separate subgraphs in order to perceive the structure of the graph: left one for \((II, \prec)\) and the right one for \((II, \succ)\), although internally we have only one hypergraph. Note further that the two graphs are shown as ordinary graph (not hypergraph).

By this abstraction, we are able to see the mathematical structure of the origami cicada more clearly.

6. CONCLUSION

We have presented an abstract model of origami. The abstraction leads to graph-theoretic modeling and transformation of an origami. We have formalized an origami as a hypergraph and define origami fold as algebraic graph transformations. The graph-theoretic formalism enables us to reason about origami in two separate domains of discourse, i.e. pure combinatoric domain, and geometric domain \( \mathbb{R} \times \mathbb{R} \), and thus will help us further tackle challenging problems such as of discovering a new construction given an origami shape, and of discovering a new origami method that has certain geometric properties.

Our formalism follows closely that of algebraic and categorical graph theories, and we anticipate the rich theories in this area will be applicable to our computational origami research.

7. ACKNOWLEDGMENTS

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.

8. REFERENCES


