Generating Euler Diagrams

Jean Flower and John Howse

School of Computing and Mathematical Sciences, University of Brighton,
Lewes Road, Brighton BN2 4GJ,UK
{J.A.Flower,John.Howse}@bton.ac.uk fax +44 1273 642405,
http:\\www.it.bton.ac.uk\research\vmg\VisualModellingGroup.html

Abstract. This article describes an algorithm for the automated generation of any Euler diagram starting with an abstract description of the diagram. An automated generation mechanism for Euler diagrams forms the foundations of a generation algorithm for notations such as Harel's higraphs, constraint diagrams and some of the UML notation. An algorithm to generate diagrams is an essential component of a diagram tool for users to generate, edit and reason with diagrams.
The work makes use of properties of the dual graph of an abstract diagram to identify which abstract diagrams are "drawable" within given wellformedness rules on concrete diagrams. A Java program has been written to implement the algorithm and sample output is included.

1 Introduction and background

Euler diagrams consist of contours, simple closed curves, which split the plane into zones. A concrete Euler diagram is a drawing which represents information about sets and their intersections. This information can be encapsulated by an abstract diagram. An abstract (Euler) diagram consists of contours, which are just abstract notions, and information about how those contours are used to give a set of zones. An abstract diagram has zero or many concrete representations, and this paper is primarily concerned with the construction of concrete representations from an abstract description. An important consequence is the identification of a complete set of drawable abstract diagrams, with tests for drawability made at the abstract level. This work supports the development of a constraint diagram tool, and this application is discussed in subsection 1.1. Definitions of the terms "concrete (Euler) diagram", "abstract (Euler) diagram" are in subsection 1.3, and the algorithm for construction of concrete diagrams is in section 2.

1.1 An application to motivate the work

The construction of a concrete representation of an abstract diagram is useful in many environments. In this section we describe one application for the work. *Constraint diagrams* [6] convey a subset of first-order predicate calculus concerning sets and their elements. The simplest kind of constraint diagrams are

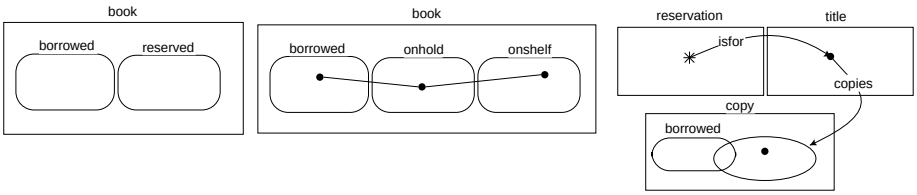


Fig. 1. An Euler diagram, a spider diagram and a constraint diagram

Euler diagrams [4], which make statements about the enclosure and intersection of sets in the system. The first of three diagrams in figure 1 is an Euler diagram which conveys the fact no book is both reserved and borrowed.

The second diagram in figure 1 is a *spider diagram* [5] showing three states for a book: 'borrowed', 'on the shelf', or 'on hold'. No book may be in two states, and there is a book in one of the states. A further extension of the notation adds universal elements, and relational navigation [6], giving the full notation for constraint diagrams. The third diagram in figure 1 is a constraint diagram which states that reservations are only held on titles for which there is a copy in stock. Furthermore, there is a copy that is in stock and not in the borrowed state.

Textual notations (eg. *OCL* [17]) and diagrammatic notations (eg. constraint diagrams) can be used for the same purpose. Many statements can be more clearly expressed in diagrammatic form. Diagrams can also be used fruitfully for reasoning ([7], [8]), with animation showing the logical steps involved in an argument.

Reasoning by transforming diagrams holds potential for the future, especially if are tools available to assist. Such a tool would be an editor (be able to assist the user with creation and editing of diagrams) but also needs an understanding of the diagram. This understanding determines whether two constraints are logically related and enables animated reasoning with diagrams. Flexible and combined use of textual and diagrammatic notations requires easy transformation from one notation to the other, in particular generating diagrams from textual notation.

1.2 Related work

There is much work on graph-drawing algorithms, see e.g. [1].

Euler diagrams are similar to the hypergraphs discussed in [2]. A hypergraph is given by a set of contours, and a set of points. Each contour is specified as containing some points and excluding others. This is similar to the problem addressed here, except the description of an Euler diagram places conditions on the make-up of the *regions* of the diagram, and not just on a set of specific points. Our Euler diagrams could be thought of as examples of hypergraphs.

There is also work on string graphs in e.g. [14]. The authors cast the problem of the drawability of Euler diagrams as equivalent in complexity to the problem of recognising string graphs.

In the above references the authors use weaker definitions of a well-formed diagram.

1.3 Key definitions

An *abstract Euler diagram* comprises a set of *contours* and a set of *zones* which are subsets of the set of contours. This corresponds to *type-syntax* in [9].

Definition 1. *An* abstract (Euler) diagram *is a pair:* $d = \langle \mathcal{C}(d), \mathcal{Z}(d) \rangle$ *where*
(i) $\mathcal{C}(d)$ *is a finite set whose members are called* contours.
(ii) $\mathcal{Z}(d) \subseteq \mathcal{PC}(d)$ *is the set of* zones *of* d*, so* $z \in \mathcal{Z}(d)$ *is* $z \subseteq \mathcal{C}(d)$.
(iii) $\bigcup_{z \in \mathcal{Z}(d)} z = \mathcal{C}(d)$
(iv) The empty set $\{\} \in \mathcal{Z}(d)$.
The set of abstract diagrams is denoted $\mathcal{D}$.

Example 1 (An abstract diagram). $\langle \{a, b, c\}, \{\{\}, \{a\}, \{a, b\}, \{b\}, \{c\}\} \rangle \in \mathcal{D}$.

The condition that the empty set is included as a zone could be omitted, and this work on converting abstract to concrete diagrams would hold for the subset of abstract diagrams which did include the empty set as a zone. In a concrete diagram, the zone corresponding to the empty set will be the zone outside all contours of the diagram.

A *concrete Euler diagram* is a set of labelled *contours* (simple closed curves) in the plane, each with a unique label. A *zone* is a connected component of the complement of the contour set. Each zone is contained in a set of contours. This corresponds to *token-syntax* in [9].

Definition 2. *A* concrete (Euler) diagram *is a triple* $\hat{d} = \langle \widehat{\mathcal{L}}(\hat{d}), \widehat{\mathcal{C}}(\hat{d}), \widehat{\mathcal{Z}}(\hat{d}) \rangle$ *whose components are defined as follows:*
(i) $\widehat{\mathcal{C}}(\hat{d})$ *is a finite set of simple closed curves,* contours*, in the plane* $\mathbb{R}^2$*. Each contour has a unique label from the set* $\widehat{\mathcal{L}}(\hat{d})$.
1. Contours meet transversely and without triple points.
2. Each component $\hat{z} \in \mathbb{R}^2 - \bigcup_{\hat{c} \in \widehat{\mathcal{C}}(\hat{d})} \hat{c}$ *is uniquely identified by a set of contours* $X \subset \widehat{\mathcal{C}}(\hat{d})$ *with* $\hat{z} = \bigcap_{\hat{c} \in X} \mathit{interior}\,(\hat{c}) \cap \bigcap_{\hat{c} \in \widehat{\mathcal{C}}(\hat{d}) - X} \mathit{exterior}\,(\hat{c})$.
(ii) A zone *is a connected component of* $\mathbb{R}^2 - \bigcup_{\hat{c} \in \widehat{\mathcal{C}}(\hat{d})} \hat{c}$*. The set of zones of* $\hat{d}$ *is denoted* $\widehat{\mathcal{Z}}(\hat{d})$.
(iii) A zone $\hat{z}$ *is uniquely determined by the set of contour labels* $\widehat{\mathcal{L}}(\hat{z})$ *for the contours which contain the zone.*
The set of concrete diagrams is denoted $\widehat{\mathcal{D}}$.

Example 2 (A concrete diagram). Let $\hat{d}$ be the concrete diagram given in figure 2. $\widehat{\mathcal{C}}(\hat{d})$ has three elements (the three contours shown) $\widehat{\mathcal{L}}(\hat{d}) = \{a, b, c\}$ and $\widehat{\mathcal{Z}}(\hat{d})$ has five elements, uniquely determined by the label sets $\{\}$, $\{a\}$, $\{a, b\}$, $\{b\}$ and $\{c\}$.

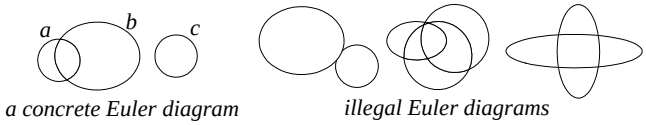


Fig. 2. Well-formed and not well-formed concrete diagrams

The rules about transverse crossing, absence of triple points and connectedness of zones are the chosen well-formedness rules for this paper. Figure 2 shows a well-formed concrete diagram and some which are not well-formed. In future work, it is intended to extend this work to accommodate different definitions of "well-formed" concrete diagrams.

Definition 3. *The mapping* $ab : \widehat{\mathcal{D}} \to \mathcal{D}$ *("ab" for "abstractify") forgets positioning of the contours. It is defined by*

$$ab\left(\langle \widehat{\mathcal{L}}(\hat{d}), \widehat{\mathcal{C}}(\hat{d}), \widehat{\mathcal{Z}}(\hat{d})\rangle\right) = \langle \widehat{\mathcal{L}}(\hat{d}), \left\{ \widehat{\mathcal{L}}(\hat{z}) : \hat{z} \in \widehat{\mathcal{Z}}(\hat{d}) \right\}\rangle$$

Example 3 (Abstractification). Let $\hat{d}$ be the concrete diagram given in figure 2. $\mathcal{C}(ab(\hat{d})) = \{a, b, c\}$ and $\mathcal{Z}(ab(\hat{d})) = \{\{\}, \{a\}, \{a, b\}, \{b\}, \{c\}\}$.

Definition 4. *A concrete diagram* $\hat{d}$ represents *or* complies with *an abstract diagram* d *if and only if* $d = ab(\hat{d})$*. An abstract diagram which has a compliant concrete representation is* drawable*.*

Example 4 (Drawability). The abstract diagram $\langle \{a, b\}, \{\{\}, \{a, b\}\} \rangle$ is undrawable. This can be investigated by futile attempts to produce a diagram (the temptation to draw two coincident concrete contours violates the condition that contours meet transversely). Later in the paper, corollary 1 is used to show that there is no concrete representation for the abstract diagram.

This paper is about the construction of concrete representations of abstract diagrams. We seek an inverse of the map $ab : \widehat{\mathcal{D}} \to \mathcal{D}$.

2 A sketch of the algorithm

We use the concept of a *plane dual graph* of a concrete diagram.

Definition 5. *A* concrete labelled graph $\hat{G}$ *is a triple* $\langle \widehat{\mathcal{L}}(\hat{G}), \widehat{\mathcal{V}}(\hat{G}), \widehat{\mathcal{E}}(\hat{G})\rangle$ *where the components are defined as follows:*

(i) $\widehat{\mathcal{L}}(\hat{G})$ *is a set of labels for the graph*

(ii) $\widehat{\mathcal{V}}(\hat{G})$ *is a set of vertices. Each vertex* $\hat{v}$ *is labelled with* $\widehat{\mathcal{L}}(\hat{v}) \subseteq \widehat{\mathcal{L}}(\hat{G})$ *and each vertex has a position in the plane* $\mathbb{R}^2$*.*

(iii) $\widehat{\mathcal{E}}(\hat{G})$ *is a set of edges. Each edge* $\hat{e}$ *is a pair of vertices from* $\widehat{\mathcal{V}}(\hat{G})$*. The label sets on adjacent vertices must have singleton symmetric difference (one set is the other with a single additional element). The edges can be associated with the label which distinguishes the labels of the end-vertices.*

The set of concrete labelled graphs is denoted $\widehat{\mathcal{LG}}$.

Note that although the vertices of a concrete labelled graph have a position in the plane, the edges are simply pairs of vertices. The edges are not associated with curves in the plane ie. we have a *geometric graph* in [1].

Definition 6. *The map* $pdual : \widehat{\mathcal{D}} \to \mathcal{P}\widehat{\mathcal{LG}}$ *("pdual" for "plane dual") is defined by*

$$\hat{G} \in pdual\langle \widehat{\mathcal{L}}(\hat{d}), \widehat{\mathcal{C}}(\hat{d}), \widehat{\mathcal{Z}}(\hat{d}) \rangle$$

if and only if $\widehat{\mathcal{L}}(\hat{G}) = \widehat{\mathcal{L}}(\hat{d})$ *and there is a bijection* $\widehat{\mathcal{V}}(\hat{G}) \to \widehat{\mathcal{Z}}(\hat{d}); v \mapsto \hat{z}$ *if and only if* v *is inside the part of the plane specified by* $\hat{z}$ *and the labelling matches:* $\widehat{\mathcal{L}}(\hat{v}) = \widehat{\mathcal{L}}(\hat{z})$. *Finally,* $e \in \widehat{\mathcal{E}}(\hat{G})$ *if and only if the corresponding zones are topologically adjacent in the plane.*

Any concrete labelled graph gives an *abstract labelled graph*, by forgetting vertex positions.

Definition 7. *An* abstract labelled graph *is a triple* $\langle \mathcal{L}(G), \mathcal{V}(G), \mathcal{E}(G) \rangle$ *where the components are defined as follows:*

(i) $\mathcal{L}(G)$ *is a set of labels for the graph*

(ii) $\mathcal{V}(G)$ *is a set of vertices. Each vertex* $\hat{v}$ *is labelled with* $\mathcal{L}(v) \subseteq \mathcal{L}(G)$.

(iii) $\mathcal{E}(G)$ *is a set of edges. Each edge* $\hat{e}$ *is a pair of vertices in* $\mathcal{V}(G)$, *where the vertex labels must have a singleton symmetric difference (one vertex set exceeds the other by a single additional element). The label which distinguishes the end vertices can be used to label the edge.*

The set of abstract labelled graphs is denoted $\mathcal{LG}$.

Example 5 (An abstract labelled graph). The abstract labelled graph G has two labels: $\mathcal{L}(G) = \{a, b\}$, three vertices: $\mathcal{V}(G) = \{v_1, v_2, v_3\}$, the vertices labelled as follows: $\mathcal{L}(v_1) = \{\}$, $\mathcal{L}(v_2) = \{a\}$, $\mathcal{L}(v_3) = \{a, b\}$ and one edge: $\mathcal{E}(G) = \{\{v_1, v_2\}\}$. Note that a second edge $\{v_2, v_3\}$ could have been added to this example, but the pair $\{v_1, v_3\}$ would not be admitted as an edge because of the labels on the vertices v_1 and v_3.

Definition 8. *The map* $f : \widehat{\mathcal{LG}} \to \mathcal{LG}$ *('f' for 'forgetful') is defined by forgetting positional information in the vertex set* $\widehat{\mathcal{V}}(\hat{G})$.

Proposition 1. *Given a concrete diagram* $\hat{d} \in \widehat{\mathcal{D}}$, *and two plane duals* $\hat{G}_1, \hat{G}_2 \in pdual(\hat{d})$, *their abstract labelled graphs are equal:* $f(\hat{G}_1) = f(\hat{G}_2)$.

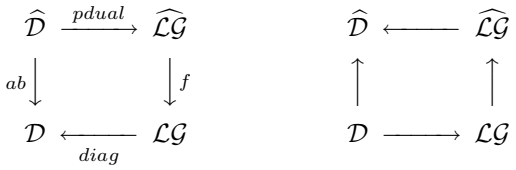
Definition 9. *Although a concrete diagram* $\hat{d}$ *has many plane dual graphs given by the set* $pdual(\hat{d})$, *we can refer to "the" abstract dual graph of a concrete diagram,* $abG(\hat{d}) \in \mathcal{LG}$.

Definition 10. *The map* $diag : \mathcal{LG} \to \mathcal{D}$ *("diag" for "diagrammise") is defined by* $\langle \mathcal{L}(G), \mathcal{V}(G), \mathcal{E}(G) \rangle \mapsto \langle \mathcal{C}(d), \mathcal{Z}(d) \rangle$ *where the abstract contour set is the label set* $\mathcal{L}(G) = \mathcal{C}(d)$ *and the zones are the vertices* $\mathcal{V}(G) = \mathcal{Z}(d)$.

This map can be thought of as forgetting the edge information in an abstract labelled graph.

Example 6 (The diag mapping). Let G be the abstract labelled graph given in example 5. The abstract diagram $diag(G) = d$ has contours $\mathcal{C}(d) = \{a, b, c\}$ and zones $\{\{\}, \{a\}, \{a, b\}\}$

What we have so far is illustrated in the following commutative diagram (shown on the left) which shows "forgetful" mappings as information is lost moving from a concrete environment to the abstract level.



The strategy of the algorithm is to attempt to find inverses of the functions *pdual*, f and *diag* (inverses shown on the right). A mapping from abstract diagrams to concrete diagrams will be found which factors through abstract dual graphs and plane dual graphs. Factoring the problem through dual graphs reduces one task to three steps, and allows the use of existing knowledge from graph theory and graph drawing. As the work progresses, we will see that some inverse functions cannot be defined on the whole domain, and in this way, some abstract diagrams become classified as undrawable. The following three sections of the paper cover three key steps:

(i) map from abstract diagrams to abstract labelled graphs: $\mathcal{D} \to \mathcal{LG}$

(ii) map from abstract labelled graphs to concrete labelled graphs: $\mathcal{LG} \to \widehat{\mathcal{LG}}$

(iii) map from concrete labelled graphs to concrete diagrams: $\widehat{\mathcal{LG}} \to \widehat{\mathcal{D}}$

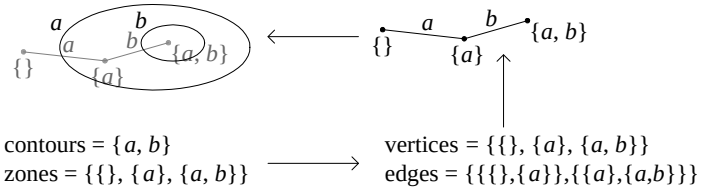


Fig. 3. The steps of the algorithm

2.1 Creation of an abstract labelled graph from an abstract diagram

Definition 11. *The map superDual* : $\mathcal{D} \to \mathcal{LG}$ *is defined by* $\langle \mathcal{C}(d), \mathcal{Z}(d) \rangle \mapsto \langle \mathcal{C}(d), \mathcal{Z}(d), \mathcal{E}(G) \rangle$ *where the edges include all possible* $e = (v_1, v_2)$ *where* v_1 *and* v_2 *have singleton symmetric difference.*

The superdual of an abstract diagram uses the contours for labels and the zones for vertices. Deriving the superdual of an abstract diagram d and mapping back to an abstract diagram recovers d, having constructed edges between some vertices and then forgotten the edges again.

Proposition 2. *If $d \in \mathcal{D}$ then $diag(superDual(d)) = d$.*

Figure 3 illustrates vertex and edge-labelling. In small cases (all examples with three or fewer contours), the abstract labelled dual graph of a concrete diagram $abG(\hat{d})$ is exactly the super-dual of its abstract diagram $superDual(ab(\hat{d}))$. However, figure 4 shows that vertex labels can differ by a single contour label even when the zones are not adjacent in the concrete diagram. The vertices $\{s\}$ and $\{p, s\}$ in $superDual(ab(\hat{d}))$ are adjacent, but they are not adjacent in $abG(\hat{d})$.

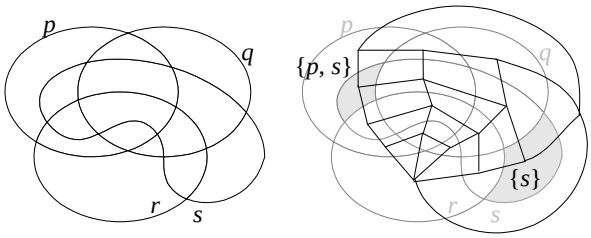


Fig. 4. The dual graph of the Venn diagram on four contours

Proposition 3. *Let $\hat{d}$ be a concrete diagram. Then $abG(\hat{d}) \in \mathcal{LG}$ is a subgraph of $superDual(ab(\hat{d}))$ which includes all vertices (sometimes called a* wide *subgraph).*

The abstract dual of a concrete diagram satisfies *connectivity conditions*:

Definition 12 (The connectivity conditions). *An abstract labelled graph $\langle \mathcal{L}(G), \mathcal{V}(G), \mathcal{E}(G) \rangle$ satisfies the* connectivity conditions *if it is connected and, for all labels $l \in \mathcal{L}(G)$, the subgraphs $G^+(l)$ generated by vertices whose labels include l, and $G^-(l)$ generated by vertices whose labels exclude l are connected.*

Theorem 1 (The connectivity theorem). *Let $\hat{d}$ be a concrete diagram. Then $abG(\hat{d})$ passes the connectivity conditions.*

The proof of this result uses some results from topology concerning paths in the plane.

Corollary 1 (The connectivity test). *If d is an abstract diagram whose super-dual fails the connectivity conditions then d is undrawable.*

This corollary provides a practical condition for drawability at the abstract diagram level.

Example 7. The abstract diagram of example 4 $\langle \{a, b\}, \{\{\}, \{a, b\}\} \rangle$ has super-dual with two vertices, labelled $\{\}$ and $\{a, b\}$ and no edges. The super-dual is disconnected, and so the abstract diagram is undrawable.

2.2 Creation of a plane dual graph from an abstract labelled graph

If we are given an abstract labelled graph which passes the connectivity conditions, it is potentially the abstract graph of a concrete diagram. To begin the construction of such a concrete diagram (if it exists), assign points in the plane to vertices of the graph to give a concrete labelled graph. Recall that the vertices of a concrete labelled graph have positions in the plane, but the edges are simply pairs of vertices, rather than curves in the plane.

The dual of a concrete diagram can be drawn as a plane graph. Given an abstract labelled graph G, it is potentially the abstract graph of a concrete diagram only if it is planar. If G is non-planar, it may still be possible to remove edges from it to leave a reduced planar graph which still passes the connectivity conditions. For example, edge removal is necessary to produce a plane dual for the Venn diagram on four contours. Different choices of edge-removal lead to different concrete representations.

For non-planar abstract labelled graphs, further work is needed to ascertain a sound strategy for edge-removal (maintaining connectivity conditions) to give a planar graph where possible. Strategies exist in the literature for removing edges to gain planarity, but in this context, attention must be paid to the connectivity conditions.

Example 8 (Edge-removal and connectivity). The following abstract diagram has a superdual which passes the connectivity conditions. However, it is non-planar and if any edge is removed, the connectivity conditions fail. It has five contours $\{a, b, c, d, e\}$ and fifteen zones. $\{\{\}, \{b\}, \{c\}, \{d\}, \{e\}, \{a, b\}, \{a, c\}, \{a, d\}, \{a, e\}, \{a, b, c\}, \{a, b, d\}, \{a, b, e\}, \{a, c, d\}, \{a, c, e\}, \{a, d, e\}\}$ (inspired by an example in [14]). The superdual is homeomorphic to the complete graph K_5.

In the current implementation of the algorithm, an iterative planarising step is used which is not optimal but works in small cases. An improved planarising step should take account of the rich structure of the abstract graph given by the labelling on the vertices. Connectivity conditions reveal the graph as connected and bipartite (by cardinality of vertex labels). The fact that edges only join vertices with singleton symmetric difference provides more structure within the dual graph.

Whatever planarising step is used, the aesthetics of the result are unimportant at this stage. The placing of the vertices determines faces of the plane graph. The next step extracts the faces as as combinatorial constructions and makes a new embedding in the plane.

2.3 Creation of concrete contours from a plane labelled graph

Given a concrete labelled graph with labelled vertices and edges, the faces of the graph are the starting point for constructing the contours for a concrete diagram. We would like to be able to construct concrete contours which give one zone for each graph vertex, and no other zones. The labels on the zones should match the labels on the graph vertices. The strategy is to draw the edges of

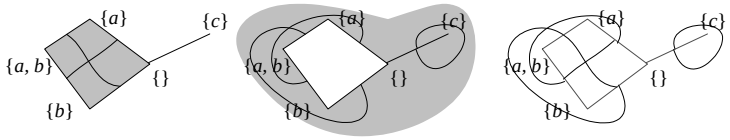


Fig. 5. Arcs across dual faces

the concrete labelled graph, and to draw arcs across faces to join the midpoints of edges. These arcs will combine to create the closed contours of a concrete diagram (see figure 5). The labels on the edges around each face determine how the arcs will intersect.

Definition 13. *A* face-cycle *of face f in concrete labelled graph G is a cycle of edges in the boundary of f which make up a cycle in G. The contour labels in a face-cycle can be cycled, or reversed, giving another representative of the same cycle.*

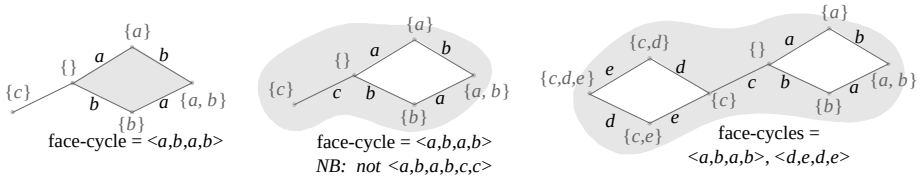


Fig. 6. Face-cycles of a concrete labelled graph

For internal (bounded) faces of a concrete labelled graph G, there is a single face-cycle made up of a subset of the boundary edges. However, an outside (unbounded) face of G may have multiple face-cycles as shown in figure 6.

Lemma 1. *Let f be a face of a concrete labelled graph. For any contour label c, there is an even number of occurrences of c in a face-cycle of f.*

The proof uses a count of symbols in vertex-labels.

Lemma 2. *Let G be a concrete labelled graph whose abstract labelled graph passes the connectivity conditions. Take a face f and a contour label c. There are zero or two occurrences of c in the face-cycle of f.*

The proof of this result is similar to a proof of the five-colouring theorem which uses *Kempe chains* [3]. Use connectivity and lemma 1.

When constructing the concrete contours, one zone will be constructed around each plane labelled vertex. It is important to ensure that the arcs across each face intersect so that no additional unwanted zone(s) appear in the face. Problematic cases with unwanted concrete zones are shown in figure 7. The unwanted zones could be either zones which aren't specified by the vertex set, or a second component of an existing zone, giving disconnected zones. The first diagram in

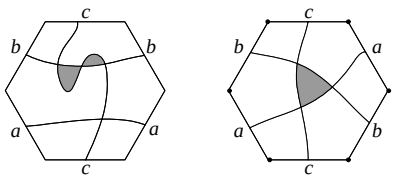


Fig. 7. Introducing unwanted zones

figure 7 can be resolved by rendering the face convex and using linear arcs across the face.

In the second diagram, however the arcs are placed without triple points, an unwanted zone will appear because of the sequence of edges around the face. The potential introduction of unwanted zones in this way provides another set of conditions for construction of a concrete diagram. The following definition and theorem set up enough notation to determine whether or not a concrete labelled graph can be used to construct a compliant concrete diagram.

Definition 14. *Define the* crossing index *of a face-cycle. For each pair of contour labels which occur in the face-cycle, determine whether the pairs are nested. If the letters are not nested, the pair contributes 1 to the crossing index, otherwise the pair contributes 0 to the crossing index. Symbols a and b are* nested *in abba, but not nested in abab.*

Example 9 (Crossing index). The cycle $\langle a, b, c, a, b, c\rangle$ has crossing index equal to 3, because all pairs $\{a, b\}$, $\{b, c\}$, and $\{a, c\}$ give non-nested sub-cycles $\langle a, b, a, b\rangle$, $\langle b, c, b, c\rangle$ and $\langle a, c, a, c\rangle$. Another example, $\langle a, b, c, b, a, c\rangle$, has crossing index equal to 2.

If a face-cycle has n symbols, each occurring twice, then its crossing index x is bounded by $0 \leq x \leq \frac{n(n-1)}{2}$.

Theorem 2 (The face-conditions). *Let d be a concrete diagram, and P a plane dual graph. For each face-cycle of a plane dual graph, with crossing index x and length $2n$, the crossing index is $x = n - 1$.*

The proof uses Euler's formula for plane graphs and the handshaking lemma.

Corollary 2 (The face-cycle conditions). *If P is a concrete labelled graph with a face-cycle whose crossing index is x and number of edges is $2n$ with $x \neq n - 1$, then P cannot be used to construct concrete contours with one zone containing each vertex.*

Example 10 (Face-conditions). In the left-hand diagram of figure 8, there are three faces and two face-cycles. The four-sided faces have crossing index $x = 1$ and cycle length $n = 2$, and the six-sided face has crossing index $x = 2$ and cycle length $n = 3$. The concrete labelled graph passes the face-conditions.

The abstract diagram with zones $\{\{\}, \{a\}, \{a, b\}, \{a, b, c\}, \{b, c\}, \{c\}\}$ is undrawable (see the second graph in figure 8). Its plane dual graph is a cycle of six

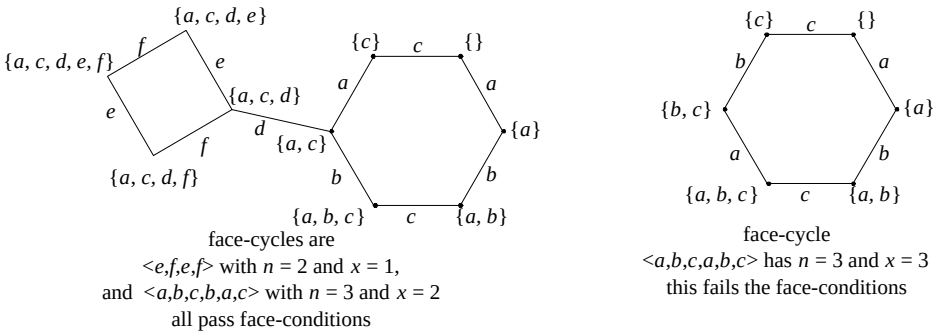


Fig. 8. Face-conditions

edges. The faces both have face-cycles $\langle a, b, c, a, b, c\rangle$ with crossing indices $x = 3$, and the lengths given by $n = 3$. It is not the case that $x = n - 1$.

Proposition 4. *If a concrete labelled graph passes the face-conditions and connectivity conditions, and removal of an edge maintains the connectivity conditions, then removal of that edge also maintains the face-conditions.*

Removal of edges (e.g. to ease planarisation) cannot jeopardise the existence of a plane representation which passes the face-conditions.

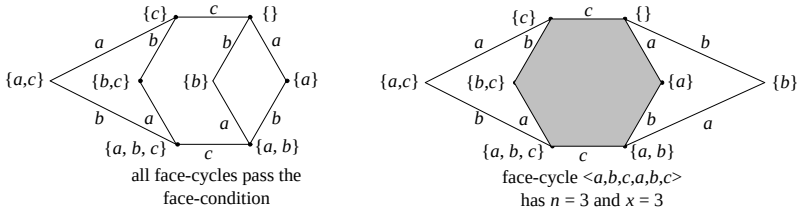


Fig. 9. Face-conditions and layout

One might hope that, given an abstract dual graph, all plane representations of it either pass or fail the face-conditions. However, this is not the case. Figure 9 shows a dual of the Venn diagram on three contours, after removal of the edges $\{\{a\}, \{a, c\}\}$ and $\{\{b\}, \{b, c\}\}$. The absence of these two edges does not cause failure of the seven connectivity conditions. The super-dual has a plane representation which passes the face-conditions (in fact *all* plane representations of the super-dual will pass the face conditions), so proposition 4 says that after removing two edges, there must also be a plane representation which passes the face conditions. This is shown on the left of the figure. But removal of these edges also introduces the existence of a plane representation which fails the face-conditions, shown on the right. This example shows that failure of the face-conditions in a single plane subgraph of $superDual(d)$ does not necessarily render the underlying abstract diagram d undrawable. To complete the argument for example 10 we have to say that not only does the presented plane graph fail the face-conditions,

but *all* plane representations of the abstract dual will fail the face conditions. For the example shown, this is easy, but for larger graphs, this kind of argument poses a problem.

Future work on improving the planarising step in this algorithm should seek plane representations which pass the face-conditions, if they exist.

Given an abstract labelled graph which passes the connectivity conditions, and a plane graph representation which passes the face-conditions, to complete the construction of the contours we draw the plane graph with all faces convex and draw arcs linear across faces. Of course, it is impossible to draw a graph with *all* faces convex, but the faces can be placed as convex faces in a disc with one new non-convex outside face. This step, called *circularisation* of the dual, is discussed in the next section.

2.4 Circularisation

Given a concrete labelled graph which passes the connectivity and face conditions, we seek a plane representation with convex faces. The barycentric approach to graph drawing [1] gives all but one face convex. Circularisation reproduces all the faces of a graph inside a disc, introducing a new (non-convex) face outside the disc. Some vertices of the original graph may appear more than once, and some edges will be duplicated. Circularisation can be achieved by taking faces

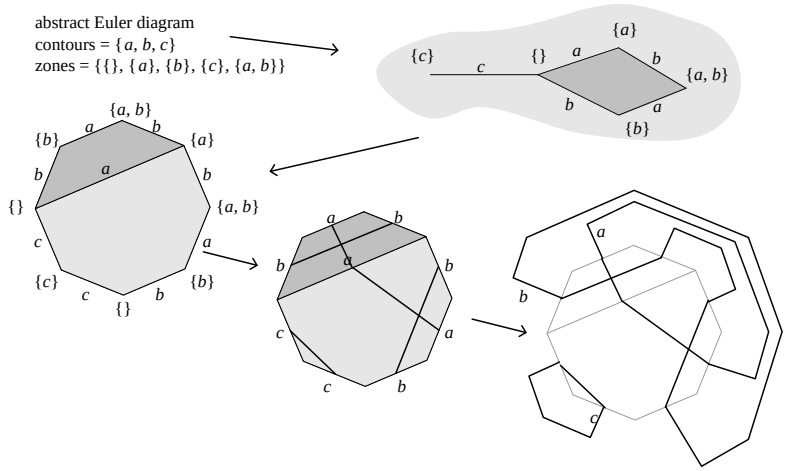


Fig. 10. The circularisation process and addition of arcs

in turn, inserting them into a disc. After the first face, further faces are chosen and inserted by identifying common edges which are already present in the disc. The edges which remain un-identified after all faces are inserted are the edges around the edge of the disc (polygon). These correspond to edges making up a spanning tree of the original graph.

An alternative view of the same process begins by choosing a spanning tree of the original graph. Split the edges into pairs, and fatten the spanning tree into

a new polygonal face. Move the infinity point into this new face and squeeze the original faces so that all vertices lie around a circle (the boundary of the new face). Edges in the chosen spanning tree are duplicated around the outside of the circularised graph whereas edges not in the spanning tree appear once each, across the disc.

The process of *circularisation* is illustrated in figure 10 as the second step (top right to bottom left diagram): a dual graph with two faces is circularised to give two faces in a disc and one additional non-convex face outside the disc.

The rewards of circularisation come from the construction of the concrete Euler contours. As in figure 5, arcs will be constructed across faces of the concrete labelled graph, but now the arcs can be drawn as straight lines. A $2n$-sided face will have n arcs drawn across it. The arcs join the midpoints of edges with shared labels. They are guaranteed to only cross other arcs in the same face (because the faces are convex), and they are guaranteed to meet transversely (because they are linear). The face-conditions guarantee that no new zones will appear as the linear arcs are added.

The added arcs contribute to piecewise-linear contours of the required concrete diagram. The contours need to be completed by drawing arcs outside the disc (the last step shown in figure 10). Starting at a vertex labelled {}, read the labels from the edges around the circle, giving a word of contour labels. This word is made up of nested pairs of contour labels. In figure 10, the nested word is *babbabcc*. Use the nesting to determine which labels are pairs. The labels which pair identify edges which will be joined by arcs outside the disc. Draw arcs outside the disc joining the innermost pairs: *bb* and *cc*. Join the two edges labelled a and join the remaining two edges labelled b.

The algorithm to this point has been implemented in the Java programming language, with outcomes shown in figure 12. The results can be difficult to interpret by eye. The final proposition suggests one resolution of this problem. The resulting concrete diagram can appear somewhat convoluted. Measure

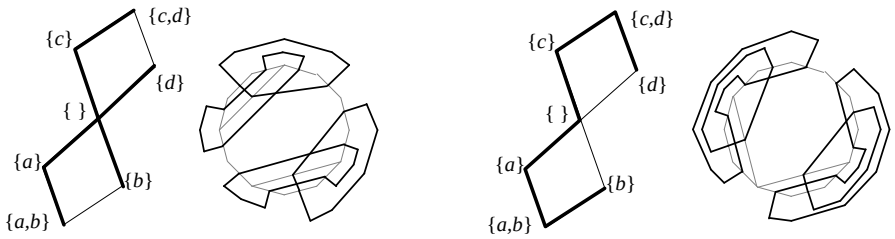


Fig. 11. Spanning trees and circularisation

the lengths of arcs drawn outside the disc, and take this to be a measure of convolution of the concrete diagram. Recall that the decisions made during circularisation give a spanning tree of the dual.

Proposition 5. *Convolution of the concrete diagram is minimised by choosing a spanning tree with the vertex labelled* {} *as its centre, with vertices as close as possible to the centre (for* centre *of a tree, see [3]).*

3 Conclusions

Figure 12 shows the outcome for all drawable diagrams with two or three contours. Some diagrams could be made less convoluted by application of the span-

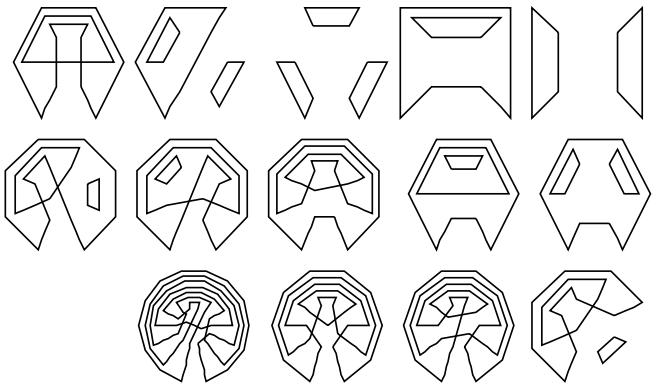


Fig. 12. Program output

ning tree with null centre (see prop 5). All other abstract diagrams with two or three contours were determined to be undrawable by testing for connectivity conditions and face-conditions.

The algorithm has been proved to be practical in small cases, as it has been implemented in the Java programming language. The program accepts a string description of a set of sets as zone descriptors. It first constructs the super-dual graph (def 11) before checking the connectivity conditions (thm 1). If the connectivity conditions pass, then edges are removed to assist with an iterative planarising step (sec 2.2). If the face-cycle conditions (cor 2) fail, then alternative planarisations are sought. If a planar representation of the dual is found which passes the face conditions, then the circularisation process (sec 2.4) is applied to construct contours and resulting concrete diagram is drawn.

Remaining questions include:

(i) If an abstract diagram has a non-planar super-dual (e.g. the Venn diagram on four contours), what is a good strategy for selecting edges (maintaining connectivity conditions) to get a planar dual graph?
(ii) What is the most effective planarising algorithm for a labelled dual graph?
(iii) Is it possible to adapt the algorithm to allow inclusion of triple points, or new zones, in the concrete diagrams? This would make more abstract diagrams drawable.

(iv) How can the resulting concrete diagram be manipulated to maintain the topological properties of contour intersection, and enhance the clarity of the concrete diagram?(prettification)

One intended application of this work is to enable a constraint diagram tool to construct appropriate diagrams which are equivalent to textual constraints in software modelling. The kinds of constraints which occur normally have few contours. If a diagram is considered as a set of "nested" components - diagrams within diagrams - then the number of contours will be further reduced. The algorithm as it is presented here is effective for such small examples, and after more work on the planarising step, would be practical for larger examples too.

Current work includes the study of existing Venn diagram algorithms to address edge-removal and smart planarising steps for Euler duals. We are also looking at "nesting" diagrams and the impact this has on layout algorithms.

Thanks to John Taylor, Gem Stapleton, and the conference referees for constructive comments on drafts of this article. This work was partially supported by UK EPSRC grant GR/R63516.

References

1. G. Di Battista, P. Eades, R. Tamassia, I. G. Tollis. *Graph Drawing: algorithms for the visualization of graphs.* Prentice Hall, 1999.
2. F. Bertault, P. Eades. *Drawing hypergraphs in the subset standard.* Springer Verlag, Graph Drawing proceedings, LNCS 1984, 2000.
3. N. Biggs, E. K. Lloyd, R. J. Wilson. *Graph Theory 1736-1936* OUP, 1976.
4. L. Euler. *Lettres a Une Princesse d'Allemagne*, vol 2. 1761. Letters No. 102–108.
5. J. Gil, J. Howse, S. Kent. Formalising Spider Diagrams. *Proc. IEEE Symposium on Visual Languages (VL99)*, Tokyo, Sept 1999. IEEE Comp Soc Press, 130-137.
6. J. Gil, J. Howse, S. Kent, J. Gil. Towards a formalisation of constraint diagrams. Proc IEEE Symp on Human-Centric computing (HCC'01). Stresa, Sept 2001, p.72-79.
7. E. Hammer. *Logic and Visual Information.* CSLI Publications, Stanford, 1995.
8. J. Howse, F. Molina, J. Taylor. On the completeness and expressiveness of spider diagram systems. *Proc. Diagrams 2000*, Edinburgh, Sept 2000. LNAI 1889, Springer-Verlag, 26-41.
9. J. Howse, F. Molina, S.-J. Shin, J. Taylor. On diagram tokens and types. Accepted for Diagrams 2002.
10. S. Kent. Constraint diagrams: Visualising invariants in object oriented models. *proceedings of OOPSLA97*, ACM SIGPLAN Notices 32, 1997.
11. F. Molina. Reasoning with extended Venn-Peirce diagrammatic Systems. PhD Thesis, University of Brighton, 2001.
12. Object Management Group. UML Specification, Version 1.3: www.omg.org.
13. C. Peirce. *Collected Papers.* Harvard University Press, 1933.
14. M. Schaefer and D. Štefankovič. *Decidability of string graphs.* Proc 33rd ACM Symposium on the Thy of Comp, p.241-246, 2001.
15. S.-J. Shin. *The Logical Status of Diagrams.* CUP, 1994.
16. J. Venn. On the diagrammatic and mechanical representation of propositions and reasonings. *Phil.Mag.*, 1880. 123.
17. J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling with UML.* Addison-Wesley, 1998.