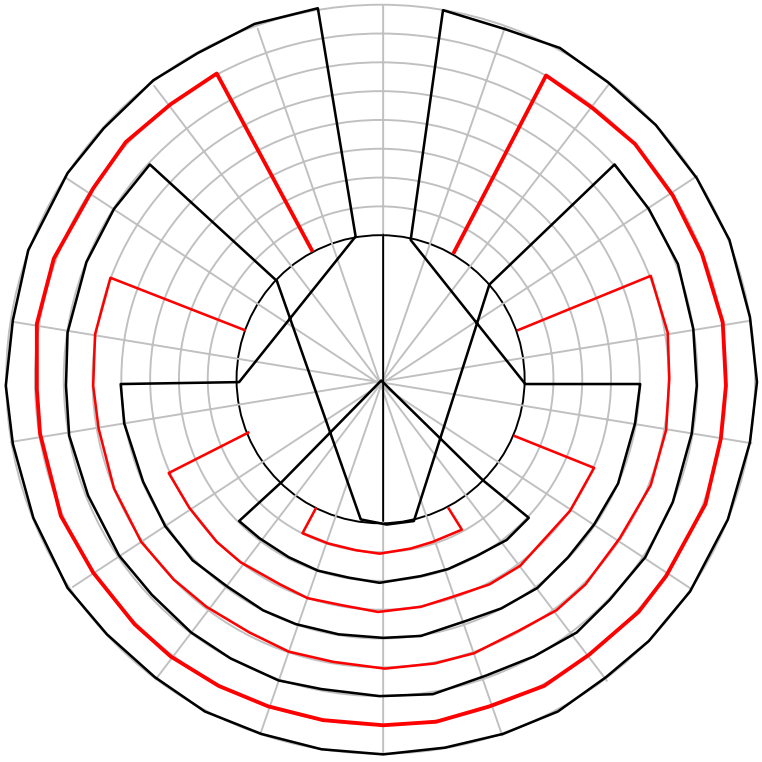
Generating constraint diagrams
Client documentation

MSc Software Engineering
Brighton University



Student: Jean Flower

Supervisors: Stuart Kent, University of Kent
John Howse, University of Brighton

Contents

Introduction

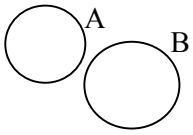
Constraint diagrams are used to specify constraints about a system. For detailed information about the definitions and purposes of constraint diagrams, see the literature (eg. [CD1]-[CD5], [SKCD]).

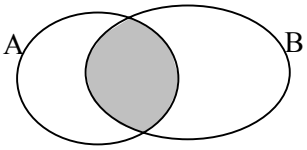
> ..a visual notation originally developed for writing constraints on UML models, hence can be used as an alternative for the OCL, the textual constraint language that is part of the UML standard. However, its application is more general: it is essentially a visual syntax for writing a significant subset of first order predicate logic.
>
> A sub-notation of constraint diagrams are spider diagrams, which combine and extend Venn and Euler diagrams, and are appropriate for relating sets and elements. Constraint diagrams add arrows which allows relations between sets to be defined and constrained.
>
> [SK]

> Spider diagrams combine and extend Venn diagrams and Euler circles to express constraints on sets and their relationships with other sets. They arose from the work on constraint diagrams which are diagrams that can be used in conjunction with the Unified Modelling Language (UML) and the Object Constraint Language (OCL) which are the Object Management Group's (OMG) standard for modeling in object-oriented software development
>
> [JT]

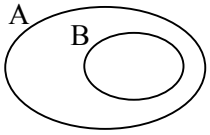
Here are some examples to briefly introduce constraint diagrams.

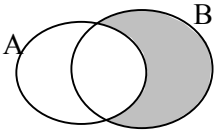
Contours are used to show membership of a set (the set of objects which satisfy some membership criterion). A zone is a minimal intersection between contours, and shading indicates that a zone is empty.





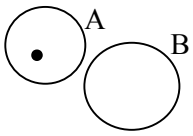
Both diagrams state that no object belongs to both sets A and B.

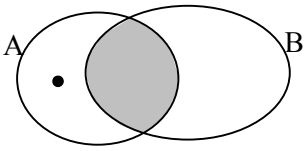




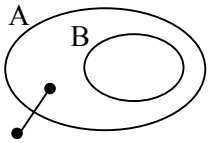
These diagrams are equivalent and both state that all objects in B are also objects in A.

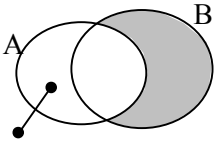
Spiders are drawn as a set of points in different zones, joined by a tree. The points (spiders' feet) assert exsitence of an object.



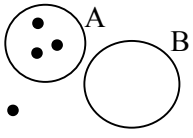


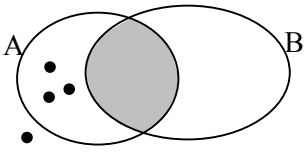
Both diagrams state that no object belongs to both sets A and B, and A has at least one object.



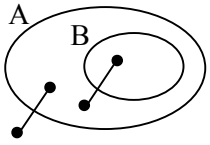


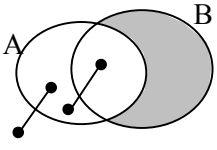
These diagrams are equivalent and both state that all objects in B are also objects in A.
Also, there is an object which is not in B.





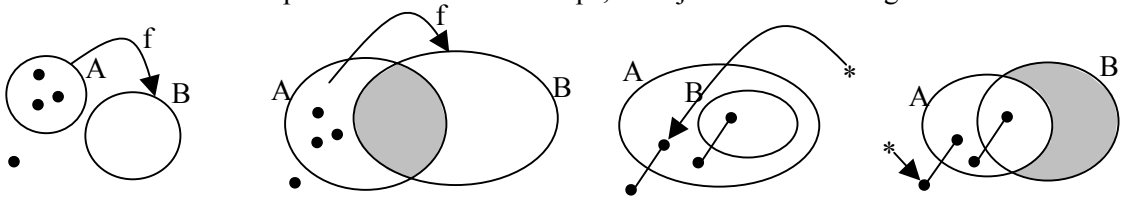
Both diagrams state that no object belongs to both sets A and B. A has at least three objects and there is some object in neither A nor B.





These diagrams are equivalent and both state that all objects in B are also objects in A.
There is an object which is not in B.
There is a (distinct) object which is in A.

Arrows can be used to represent mathematical maps, or object-oriented navigation.



B is the image of A under the operation f.

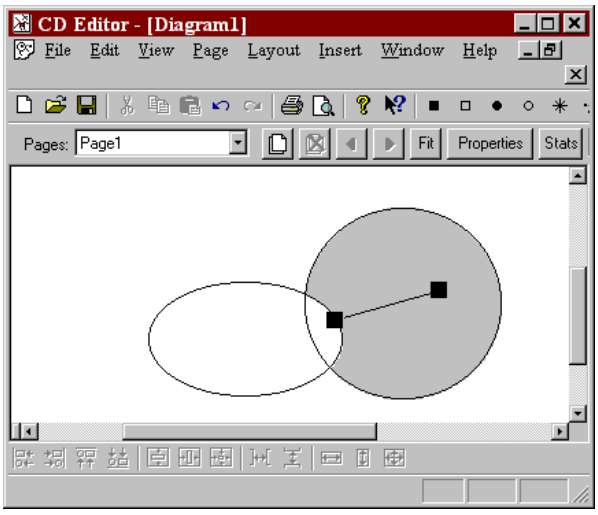
The image of an object outside A is outside B.

Such descriptions about a system are unabiguous and can play a part in modelling for software engineering as a communication tool and for system specification.

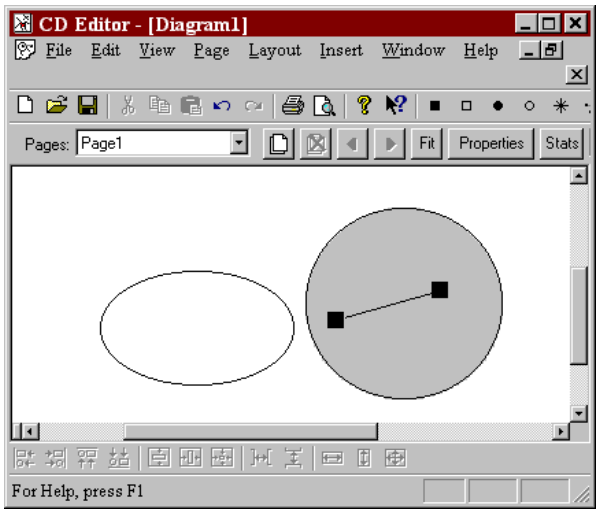
Propositional calculus in mathematics and object constraint language [OCL] in object-oriented contexts are formally defined, so free from inconsistencies and complete in their descriptive powers. The advantage of using constraint diagrams lies in their usability and approachability. If system designers find OCL difficult to use as a communication tool, there is a risk that they will fall back on English to communicate with other designers and domain experts. Such descriptions can be interpreted differently by different people, especially in an international context. Constraint diagrams have the formal underpinning of a mathematical system, but aim to be more usable than textual approaches.

Before constraint diagrams can be widely adopted by software enigineers, software tools need to be available to manipulate them. Ideally, these diagrams will gain standing as a part of the unified modelling language [FS], and CASE tools from Rational or TogetherSoft could include them to complement other UML diagrams (class diagram, sequence diagrams and so on).

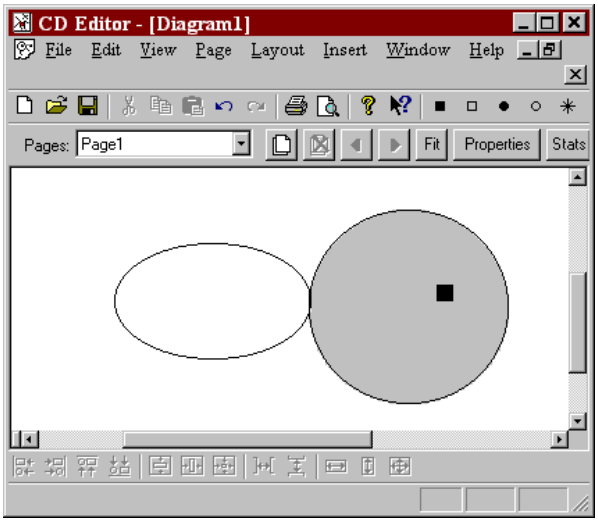
A constraint diagram drawing tool exists [CDE]. It allows the user to draw constraint diagrams and export the diagrams as pictures or as a textual description. The program draws all contours as ellipses, which places some constraint diagrams out of reach. It allows the user to manipulate the diagrams as pictures, without any reaction when the meaning changes or if the diagram becomes invalid.

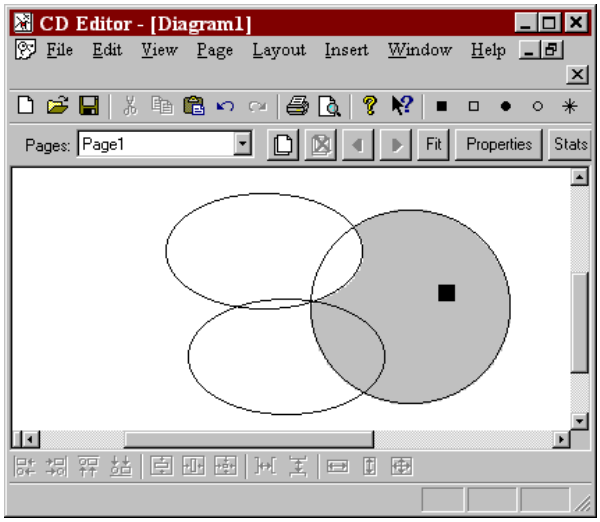


This is an invalid constraint diagram:
the foot of a spider does not belong to a zone.

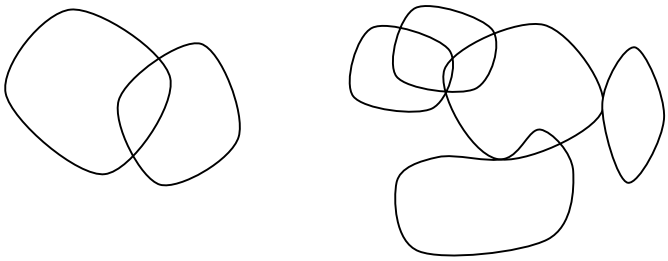


This is an invalid constraint diagram:
a spider has multiple feet in a single zone.

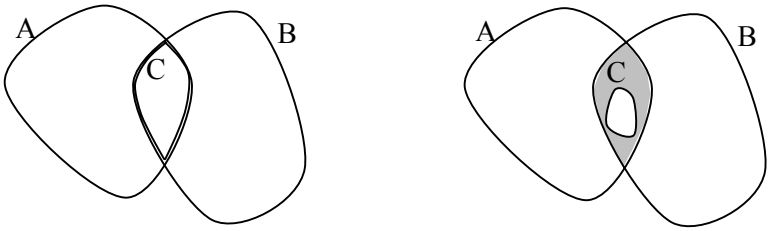




These diagrams are not invalid as constraint diagrams, but an improved tool will prevent contours meeting at tangential points or triple points, for visual clarity. For the purposes of this report, contours of constraint diagrams will only meet at transverse crossing points.

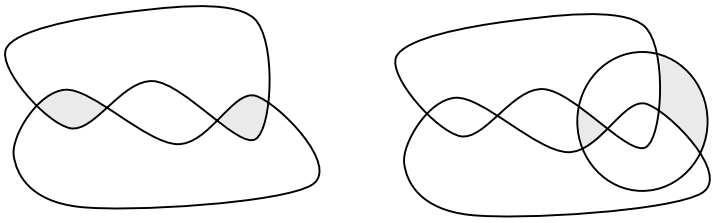


The first diagram has transverse crossing points and will be allowed. The second diagram has a triple point, and two tangential crossing points, which are not allowed. Some diagrams become undrawable under these rules:



This diagram has coincident contours, the boundary of the C-set follows along part of the A-contour and part of the B-contour. There is an equivalent diagram using just transverse contour crossings, but it introduces a new, shaded, zone.

Another condition placed on constraint diagrams, for this report, is that zones should be represented as connected areas. These diagrams would not be allowed:



A sophisticated constraint diagram tool would understand the link between a concrete constraint diagram, an abstract constraint diagram, and an OCL expression. It would allow the user to "synchronise" between constraint representations of these different forms.

Such a project in its entirety is beyond reach of an MSc project. It splits into:

- an editor user interface – similar to the existing diagram editor, but with an OCL pane and opportunity to "synchronise" between the two panes.
- the transformation from a given diagram to an object model for the constraint diagram and vice versa
- the transformation from an object model for the constraint diagram to an OCL expression and vice versa

The focus of this MSc project has been to

- create an object model for an abstract constraint diagram
- create an object model for a concrete constraint diagram representation
- and implement, in java, the transformation from the abstract constraint diagram to a concrete representation.

The writing up falls into four chapters – the first describes the algorithm to transform an abstract diagram into a concrete one. This original algorithm has been implemented in java and does the required job on a broad selection of examples.

The second chapter is about the design of the object models. It includes UML diagrams, patterns and constraints. Java code samples are included alongside descriptions of some important methods. The classes as presented in design form are also illustrated by including test programs. The test programs were a critical step in developing the implementation, and serve to show how to build objects of different classes and how they respond to different method calls. Complete code listsings and java documentation are submitted on a CD-ROM.

The third chapter shows a range of screendumps showing program output for different examples.

The fourth chapter consists of reflection on the algorithm and the design, with suggestions for future work. There is ongoing reflection and suggestions for improvement in chapter twom as each package is introduced. The improvements in chapter 4 are of a more general nature.

Chapter 1 An algorithm for creating constraint diagrams

This chapter describes a new algorithm for producing a diagram for the contours and zones of an abstract constraint diagram; details about spiders and arrows have been omitted. Constraint diagrams without spiders or arrows are referred to as Euler diagrams. A particular kind of Euler diagram, where all zones are visible, is called a Venn diagram. There is a good resource of Venn diagrams and their representations at [VENN].

An "abstract" constraint diagram consists of information about a constraint diagram without any visual representation. For example, "two disjoint contours called A and B, and an existential spider with a single foot inside A" is a description of an abstract constraint diagram.

A "concrete" constraint diagram is a visual image of a constraint diagram.

An "abstract Euler diagram" is information about a set of contours. It's an abstract constraint diagram with no shading, spiders or arrows.

A "concrete" Euler diagram is a visual representation of an abstract Euler diagram.

A "combinatorial graph" or "abstract graph" has a set of nodes and edges, but no information about positioning.

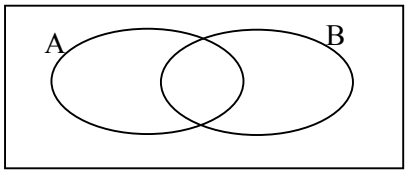
A "graph diagram", "graph representation", or "concrete graph" is a visual representation of a combinatorial graph.

The algorithm focuses on the task of producing a concrete Euler diagram representation of a given abstract Euler diagram. It uses concepts from Graph Theory (eg. [BB]) and topology (eg. [ES]).

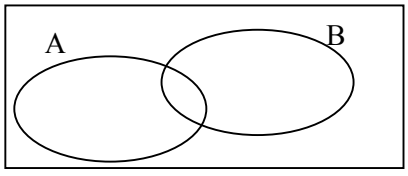
1.1 A sketch of the algorithm

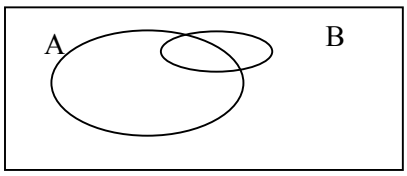
An abstract Euler diagram can be defined by its zones. For the purposes of this chapter, I will use a convention that all contours have names which are a single character, and zones are defined by a list of contours to which the zone belongs. So the zone "AB" is inside contours A and B and no others.

A description such as A, AB, B can be represented by this concrete diagram:



There are other concrete representations of the abstract Euler diagram, for example:





An algorithm to generate a concrete representation of an abstract diagram has to make a "choice" amongst an infinite range of possibilities.

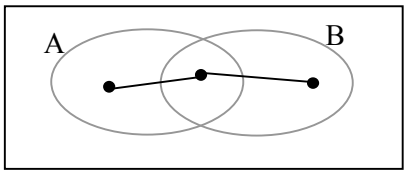
There are three key steps in generating a diagram of an abstract constraint diagram:

- generate a combinatorial dual graph (just connectivity).
- generate a planar dual graph diagram.
- generate the contours from the planar dual graph.

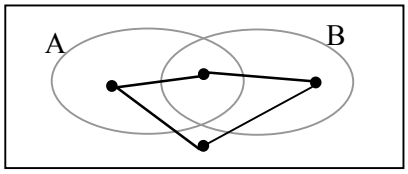
These three steps are described in detail in the next three sections.

1.1.1 Generate a combinatorial dual graph

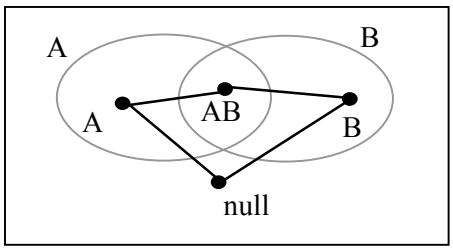
Given an Euler diagram, generate a dual graph diagram. Create a dual node in each zone of the Euler diagram, and join two dual nodes by a dual edge if there is a single contour line separating the dual nodes.



Include a node for the zone outside all the contours.



Label the vertices of the dual graph diagram with the zone information. Label the outside node null.

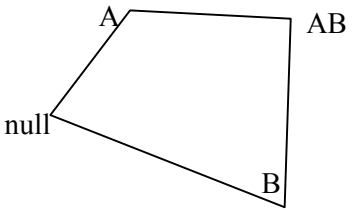


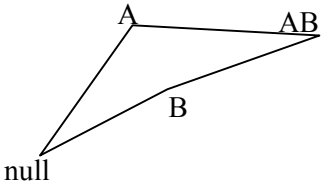
Given an abstract Euler diagram, studying the contour membership of zones always allows us to generate an unique abstract dual graph.
For example, the abstract diagram "zones: null, A, B, and AB" corresponds to the abstract dual "nodes null, A, B, and AB with edges null-A, null-B, A-AB and B-AB".

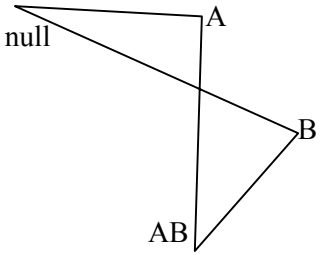
Given a concrete Euler diagram, we can visually create a concrete dual graph. All edges in the concrete dual correspond to edges in the abstract dual graph. The converse isn't always true (see section 1.3.1).

1.1.2 Generate a planar dual graph diagram

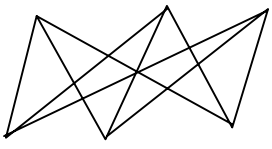
From a combinatorial dual graph, seek a planar representation as a concrete dual graph.
A combinatorial graph has many concrete representations.







All these concrete graphs represent the dual graph of the abstract diagram {null, A, B, AB}. The first two are planar representations, and the third is non-planar (has edge-crossings). Not all graphs have planar representations:

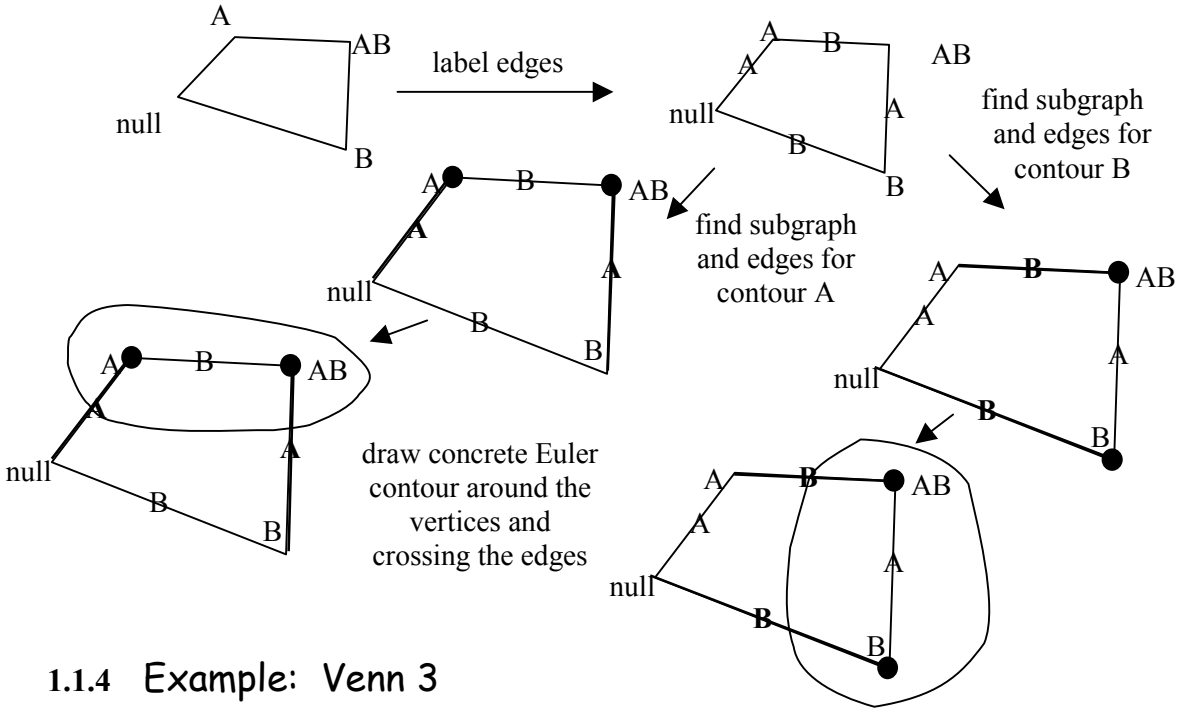


In this case, wherever the nodes are placed, there will be an edge-crossing. Such a graph is called a non-planar graph (as opposed to a non-planar representation of a graph). In 1. 3 , I have shown some steps required to generate a planar graph representation from a combinatorial graph, and what to do if the dual combinatorial graph is non planar.

1.1.3 Generate the Euler contours

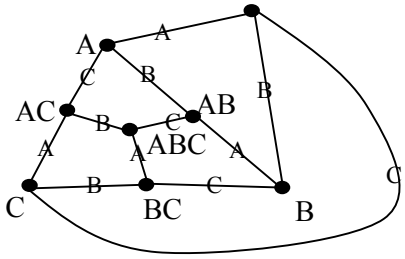
Assume we have a concrete planar representation of the dual graph. The nodes of the dual are labelled with zone descriptions, and the edges can be labelled with contour names for the change in membership from a zone at one end to the zone at the other.

For each contour of the abstract Euler diagram, the concrete Euler diagram contour is drawn around the nodes which contain that letter in their label. For each contour name, find a subgraph of nodes which include that name in their label and draw a contour around them. The drawn contour will cross all the edges labelled by that contour name and no other edges.

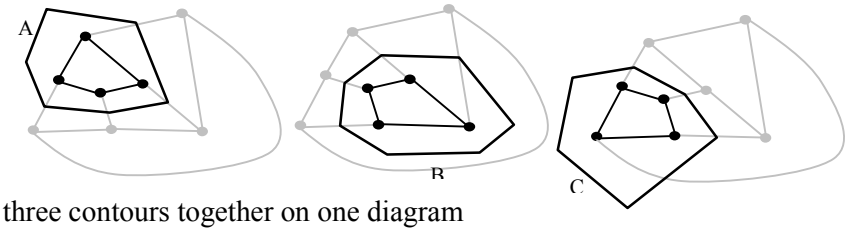


1.1.4 Example: Venn 3

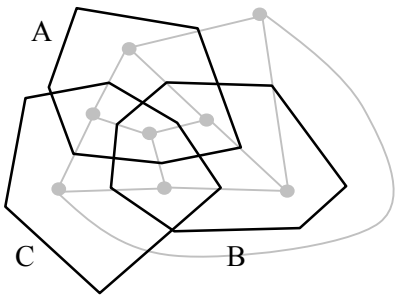
From zone information A, B, C, AB, AC, BC, ABC (the complete Euler diagram on three contours), the dual graph could be represented as follows:



Draw contours around each of subgraphs filtered with A, B, C, labelled by "A", "B", "C":



Bring the three contours together on one diagram



1. 2 Connectivity obstructions to drawing Euler diagrams

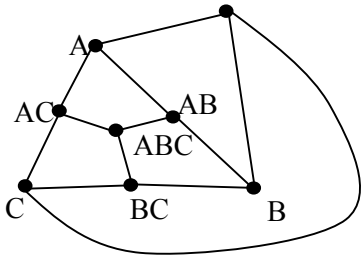
The algorithm described above can't always work – because some Euler diagrams are undrawable (the abstract diagram "null and AB", for example, has no direct concrete representation).

1.2.1 Disconnected dual graphs

It must possible to traverse a concrete Euler diagram from any zone to any other zone, crossing contour lines. In the dual graph, this corresponds to traversing paths along edges from any node to any other node. The dual graph must be connected.

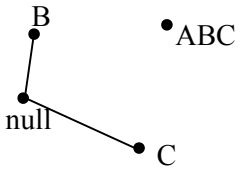
If the abstract dual graph of an abstract Euler diagram is disconnected, then the abstract Euler diagram has no concrete Euler diagram representative.

Example 1:
For zone information A, B, C, AB, AC, BC, ABC (the complete Euler diagram on three contours), the dual graph could be represented as follows:



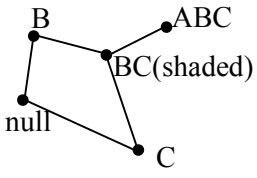
The dual graph is connected – a necessary condition for proceeding with the algorithm.

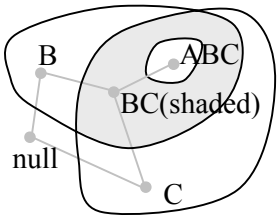
Example 2:
Given zone information B, C, ABC,
the dual graph we generate is disconnected.
The Euler diagram is undrawable.



1.2.2 Resolving disconnected duals

One way to resolve this obstacle to drawability would be to insert extra nodes and edges to the dual graph (e.g. "BC" in the last example), to make the graph connected, and remember to shade in the new zones in the resulting Euler diagram. This would create a diagram *equivalent* to the required diagram, not *equal* to the required diagram.





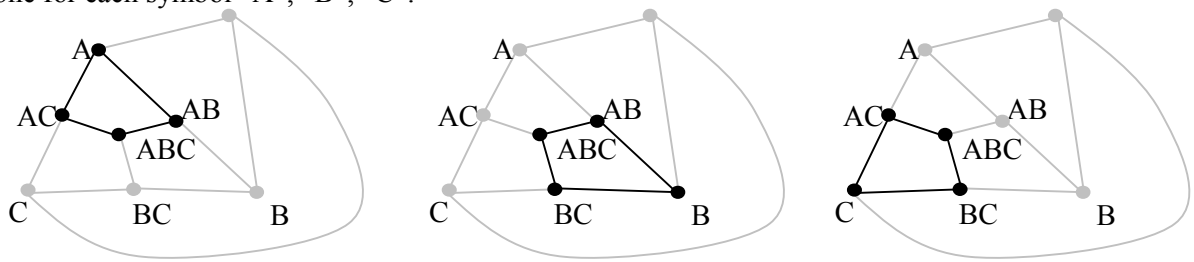
The concrete Euler diagram on the right probably represents the intended constraint, but has more zones than the abstract Euler diagram, for example, so it's an unequal Euler diagram.

1.2.3 Disconnected inclusive subgraphs

In a concrete Euler diagram, given two zones in the same contour, it is possible to navigate from one to the other without leaving the contour. In the abstract dual graph, this corresponds to navigating along edges between nodes sharing a contour-label. Subgraphs of the dual graph corresponding to the inside of each contour must be connected.

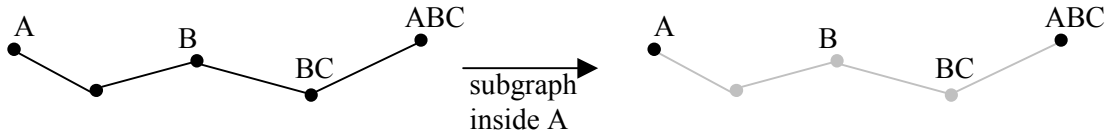
If an abstract Euler diagram generates an abstract dual graph and there is some contour with the property that the subgraph restricted to that contour is disconnected, the abstract Euler diagram has no concrete representation.

Example 1:
For the abstract diagram A, B, C, AB, AC, BC, ABC, consider three subdiagrams of the dual graph, one for each symbol "A", "B", "C".



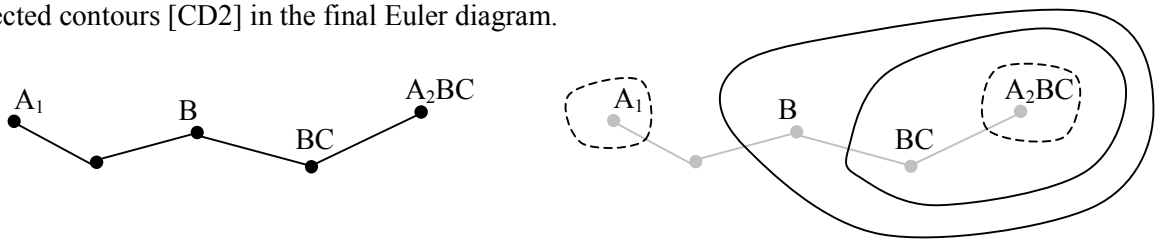
These subgraphs are all connected – a necessary condition for drawing the Euler contours.

Example 2:
{A, B, BC, ABC} gives a connected dual graph, but the subgraph obtained by filtering for A is disconnected – revealing an undrawable diagram.



1.2.4 Resolving disconected inclusive subgraphs

This could be resolved by labelling each component of the A-subgraph as A_1, A_2, and drawing these as projected contours [CD2] in the final Euler diagram.

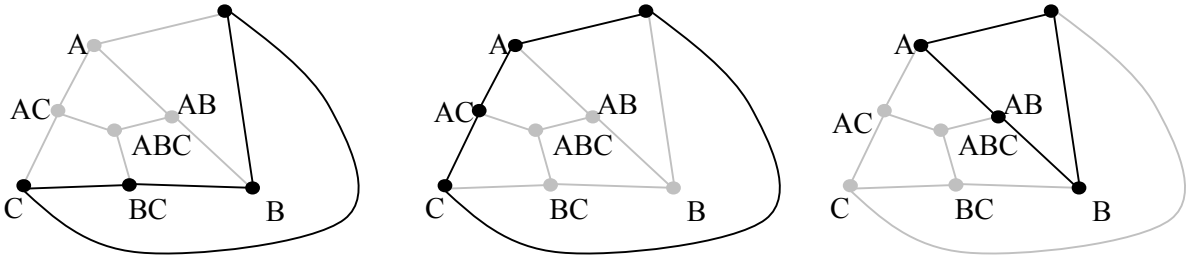


1.2.5 Disconnected exclusive subgraphs

In a concrete Euler diagram, for any contour, given two zones outside that contour, it is possible to navigate between them, remaining outside the contour. In the abstract dual graph, this corresponds to navigating from node to node, only through nodes which don't include a given contour name. Subgraphs of the dual graph obtained by eliminating zones from a given contour must be connected.

If an abstract Euler diagram generates an abstract dual graph and one contour name leaves a disconnected subgraph when all nodes in that contour are removed, then the abstract Euler diagram has no concrete representation.

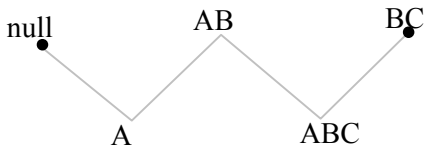
Example 1:
For abstract information A, B, C, AB, AC, BC, ABC, consider three subgraphs obtained by excluding zones within each contour – "filtered less A" etc.



These are all connected.

Example 2: An undrawable Euler diagram:

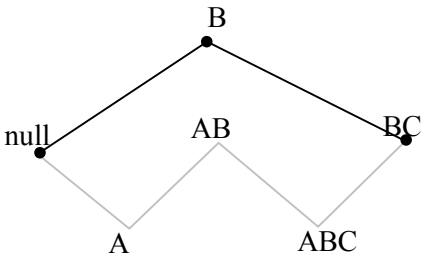
This combinatorial dual gives a disconnected subgraph upon removal of nodes inside contour A.

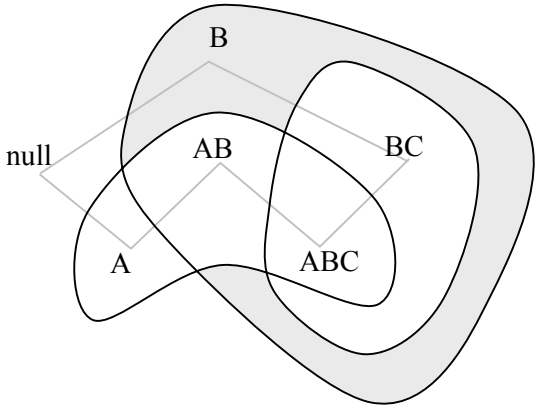


There can be no direct concrete represetation of the abstract diagram.

1.2.6 Resolving disconected exclusive subgraphs

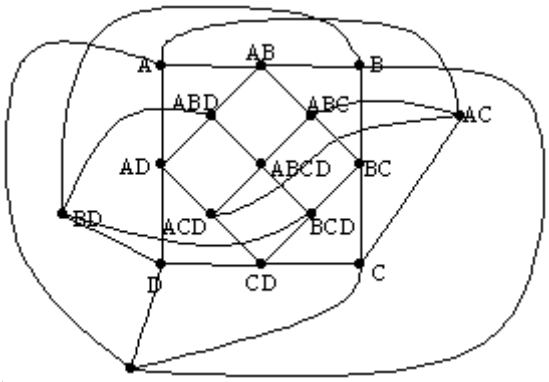
To make an exclusive disconnected subgraph connected, add new nodes (shaded zones) and edges:





1. 3 Planarising obstructions to drawing Euler diagrams

Say we take instructions to draw a Venn diagram with four contours. The zones are given by A, B, C, D, AB, AC, AD, BC, BD, CD, ABC, ABD, ACD, BCD, and ABCD. Here's a non-planar representation of the abstract dual graph:



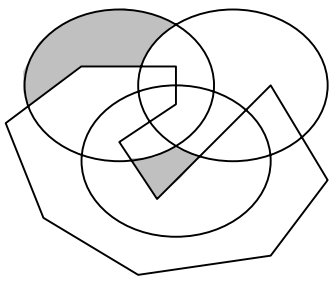
This graph is non-planar. No manipulation of the node positions will eliminate all edge-crossings.

1.3.1 Resolving planarising obstructions

A Venn diagram on 4 contours *can* be drawn – so how come the abstract dual graph is non-planar?

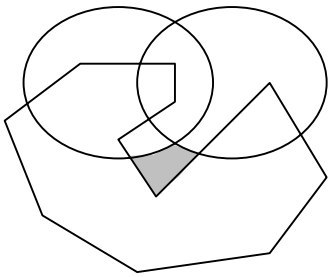
It's because abstract dual graphs constructed from zone information may have more edges than necessary. Concrete representations of Venn4 will have dual graphs which are planar subgraphs of this non planar dual graph. To create a concrete Euler diagram from a non-planar abstract dual graph, (if the constraint diagram is drawable) remove some edges, maintaining all connectivity conditions. Different representations of Venn4 correspond to different choices for edge-removal.

The removal of dual edges can cause some surprising results in the Euler diagram:



The shaded zones in this version of Venn4 could be adjacent, but the layout has separated them.

When removing contours from constraint diagrams, the abstract zones lose membership information about the missing contour. If the contour is removed from the diagram, some zones may be left disconnected (not a valid constraint diagram).



The shaded area should be part of the outside zone.

There is scope for more work on the removal of contours from diagrams:

Q: Is the removal of a contour always a legitimate move for abstract constraint diagrams?

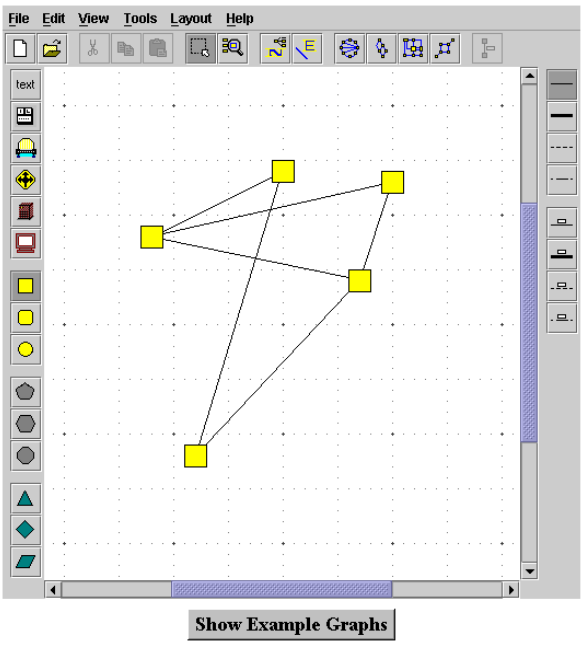
Q: What effect does contour removal have on the dual graph of a constraint diagram?

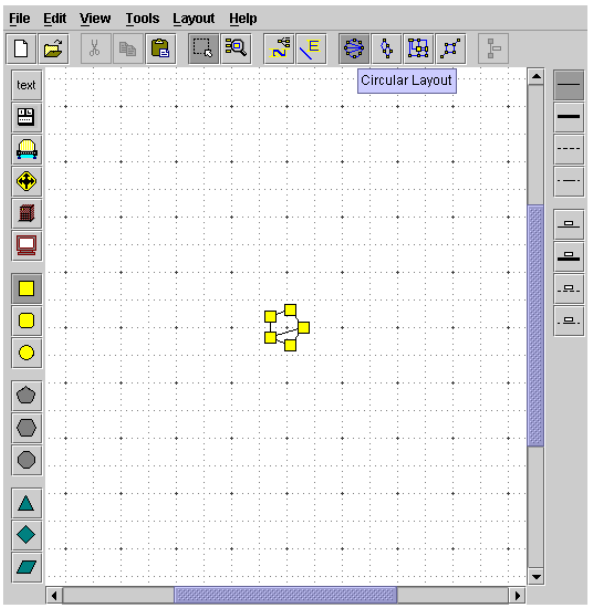
1.3.2 Planarising algorithms

After some investigation of existing planarising algorithms in the literature [G1]-[G3] I found that they all placed some condition on the outcome – spacing the nodes in a certain way, or maximising the symmetry of the graph.

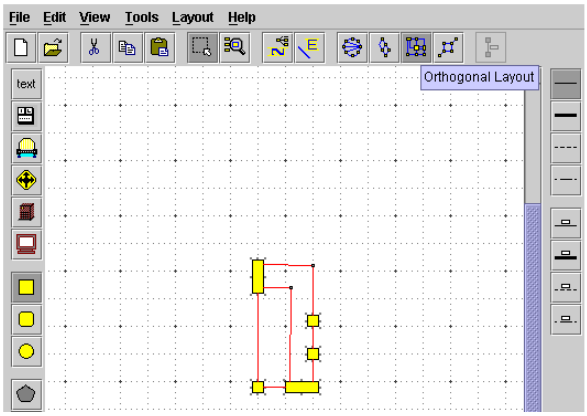
One example of a Graphing Layout tool comes from Tom Sawyer software:
http://www.tomsawyer.com:

> The Java Technologies Demonstration is a 100% Pure Java interactive graphical editor built with our Graph Editor Toolkit for Java. The palette items, menus, dialogs, and user interface are completely customizable to meet your application needs. This editor incorporates our scalable and incremental graph layout algorithms, so with a press of a button you'll clearly see the answers you need.

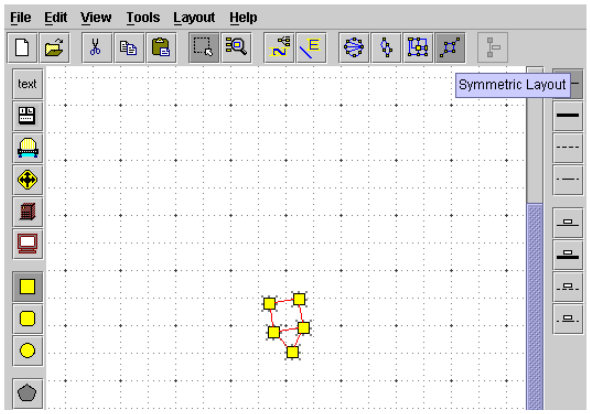




Automatic generation of a circular layout

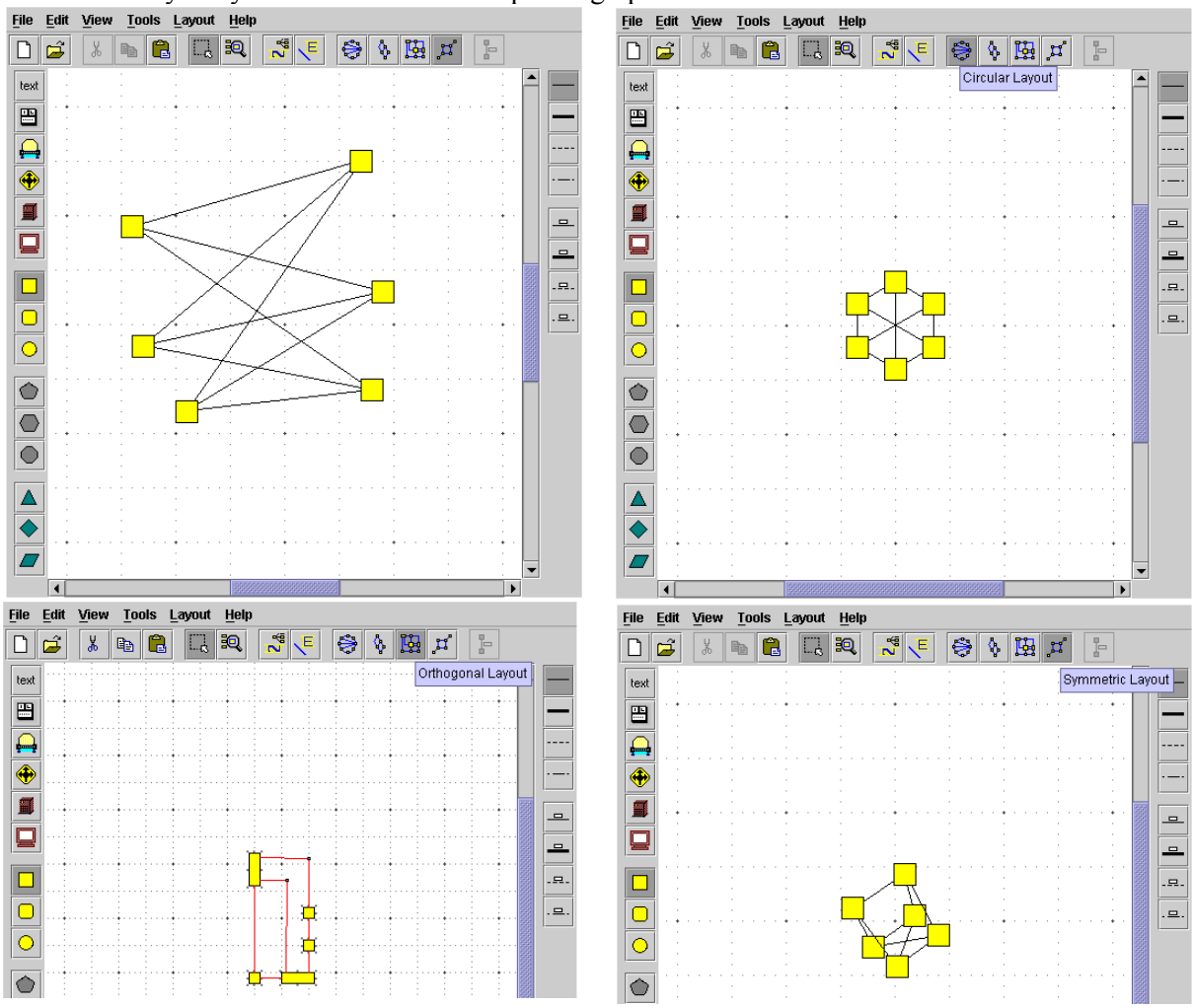


Automatic generation of an orthogonal layout



Automatic generation of a symmetric layout

The Tom Sawyer layout tools work for non planar graphs.

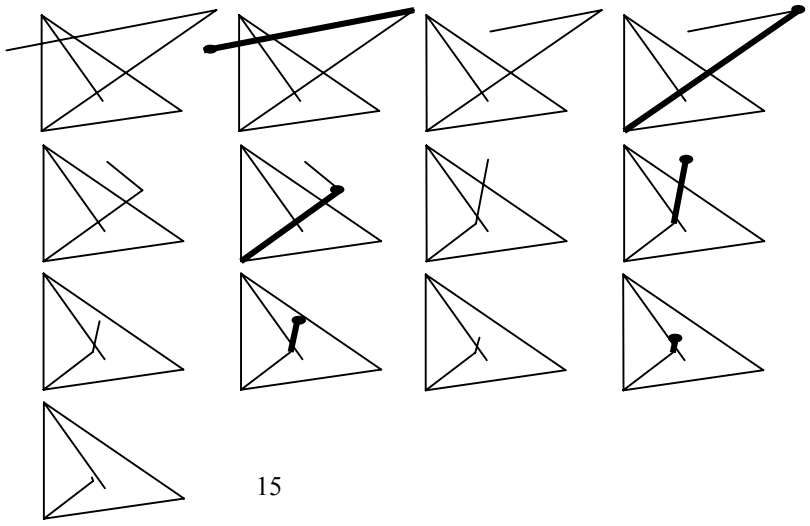


These algorithms are more sophisticated than I need. I wrote my own, simpler algorithm, which tries to create a planar representation of an abstract graph. For the purposes of this algorithm for drawing constraint diagrams, the aesthetics of the planar dual graph are immaterial (see the work on "circularising" a dual graph).

Take any concrete graph. For each edge, count the number of times it meets another edge. If all of these calculations give zero, the graph is planar and the planarisation can stop.
Otherwise, choose an edge which meets other edges most often. Take a node at one end of lowest degree, and move that node along the edge, halfway towards the other end. Iterate this step a given number of times. Either the graph will become planar or it won't.

In an example where the nodes have integer co-ordinates instead of real co-ordinates, it would also be useful to rescale the picture after each iteration to reduce the risk of nodes becoming co-incident.

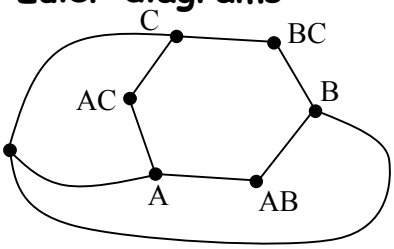
Example: Planarise this:



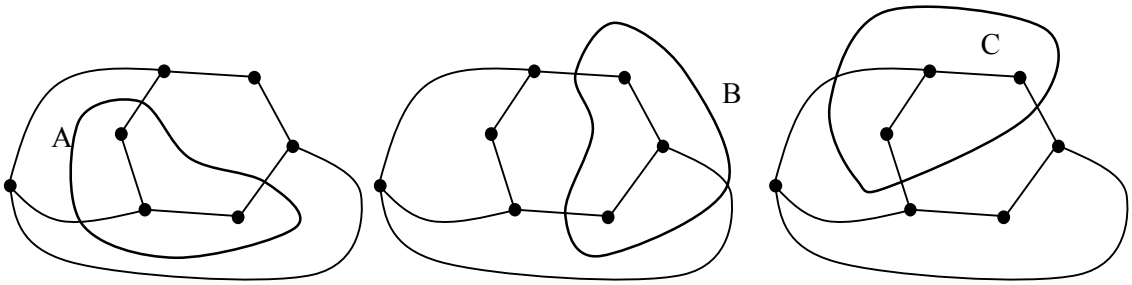
This algorithm is inefficient, but it does a good enough job for the time being, and a method which takes a graph object, an integer number of iterations, and tries to produce another, planar graph, can easily be overridden by a better algorithm. A great feature of OO design: as long as the signature and pre/post conditions of an operation are maintained, it's easy to substitute one algorithm for another.

1.4 Word obstructions to drawing Euler diagrams

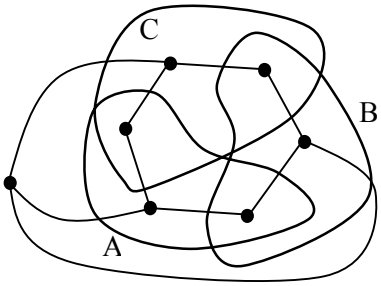
Given zone information
{A, B, C, AB, BC, AB}
this is a planar representation of the dual graph:



Draw each of the three contours:

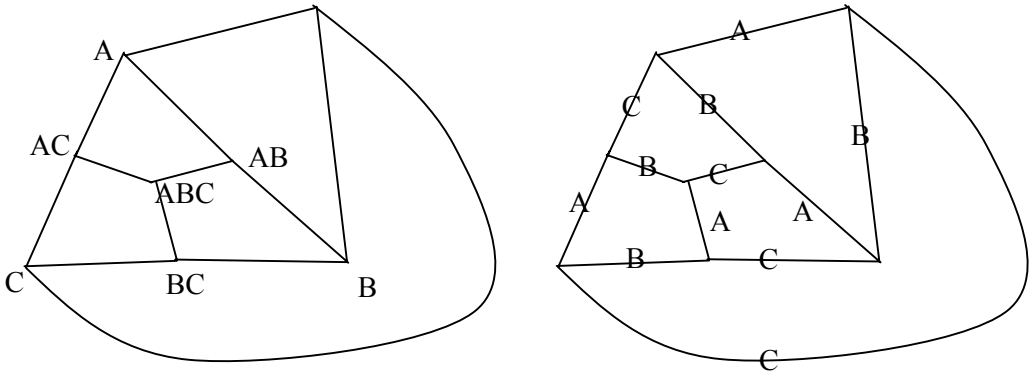


When these three contours are placing on a single diagram, a triple point could appear:



For the purposes of this project, constraint diagrams don't include triple points. There's a risk, if the contours are nudged, that the triple point will reduce to three transverse double points (allowed in constraint diagrams), but at the cost of introducing an unwanted new zone. This problem could have been recognised from *face-reading* – a property of the planar graph embedding. Face reading is explained below:

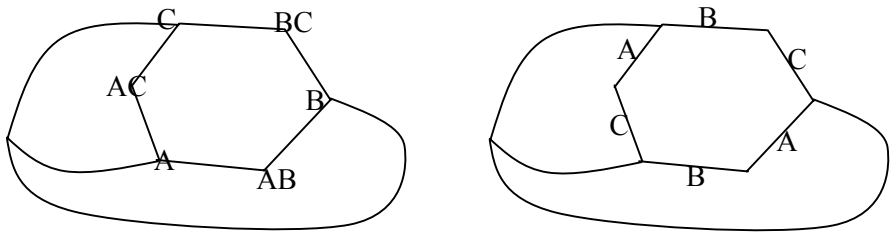
Given a planar dual graph representation, label the edges with contour names A, B, C.



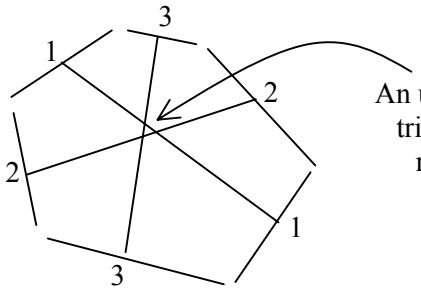
There are six faces of the diagram on the right (include the outside one). Read around these faces, starting at any vertex, following either direction, giving six words.

ABAB CBCB ACAC
ABAB BCBC ACAC

Notice that each word has an even number of occurrences of each symbol. Equivalent words can be obtained by cycling the order of symbols, or reversing the order. In this example, the words all have four letters, because the faces were all quadrilaterals.

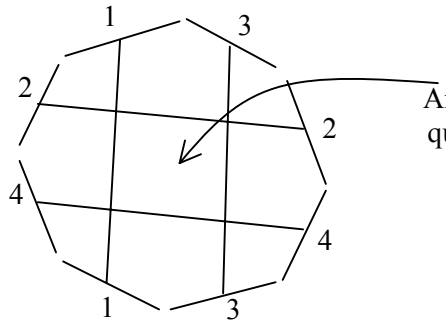


The hexagonal face in this example reads as CABCAB. This word is one of a family of words, recognisable using a template "XYZXYZ". To find out whether the internal arcs in a dual graph will create an unwanted zone, generate the contour-word from each face. A new zone will be a polygon:



An unwanted triangular region

Whatever other lines are present in the dual graph face, if three pairs of labels form the pattern 132132, then there will be at least one newly created, unwanted region between the contours. The Euler contours are undrawable from this planar representation of the dual graph.



An unwanted quadrilateral region

Whatever other lines are present in the dual graph face, if four pairs of labels form the pattern 13243142, then there will be at leastr one newly created, unwanted region between the contours.
The Euler contours are undrawable from this planar representation of the dual graph.

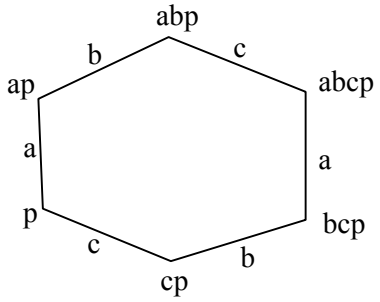
Such anomalies can be spotted by searching for (sub)words in which every letter is three from its partner. To summarise the word conditions: given a planar representation, we have a word-obstruction if

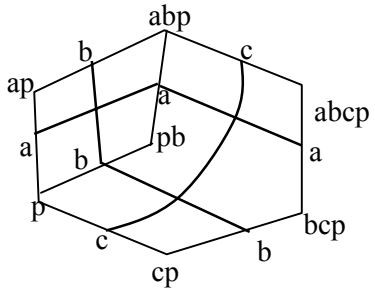
(i) any face includes a contour name as an edge label more than twice

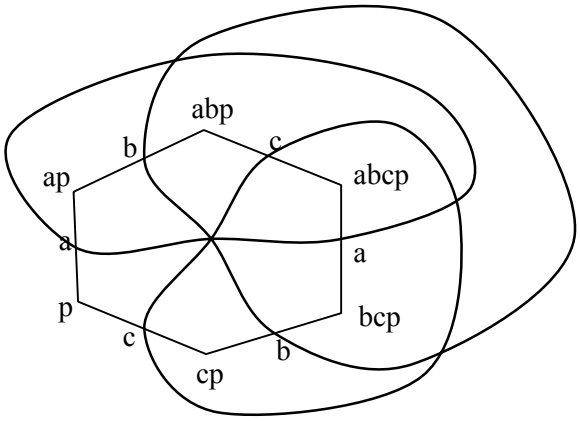
(ii) the word of some face, restricted to some subset of contour names, features each name three places from its partner.

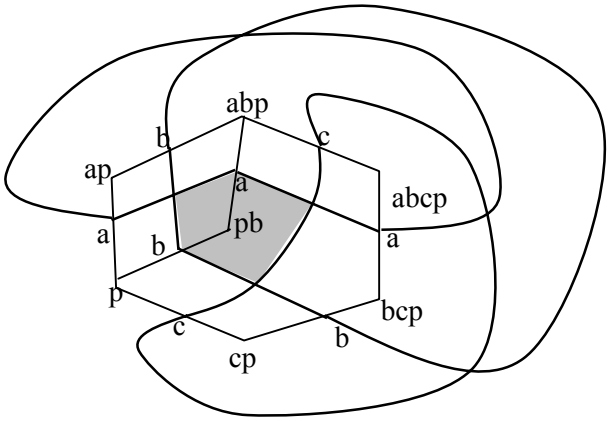
1.4.1 Resolving word-obstructions

If a face fails the word conditions, it can be resolved by adding a new zone.
For example, the word abcabc fails the word conditions, but adding a new zone splits up the hexagon into a square and a hexagon, essentially reversing two letters in the word, giving a drawable bacabc

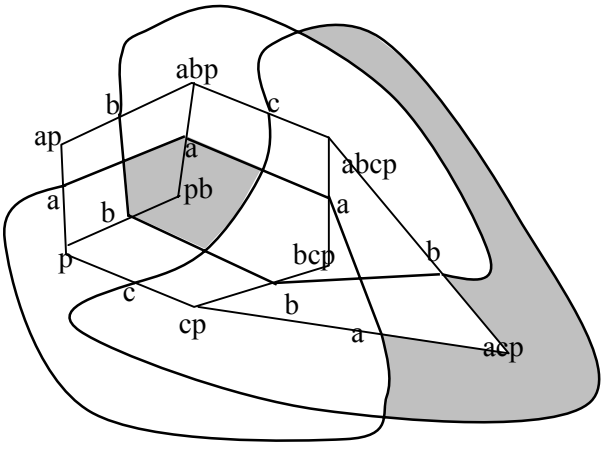








note that there is a triple point problem outside this hexagon, too:



Questions remain like:

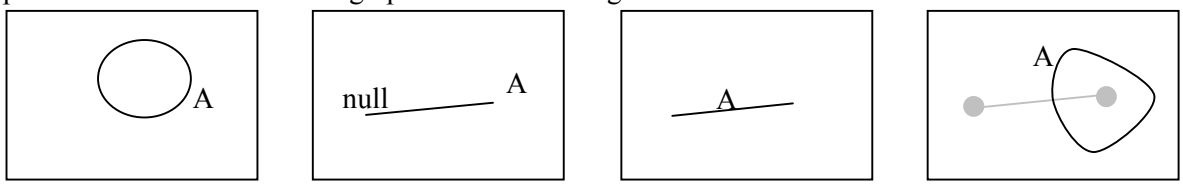
- can every word failure be resolved by introducing new nodes with swapped adjacent letters?
- what if the zones “pb” or “acp” existed already elsewhere in the dual graph?
- can a word failure exist with all potentially new resolving zones existing outside the face?

1. 5 First worked examples

1.5.1 Singleton set

Zone description: {A}

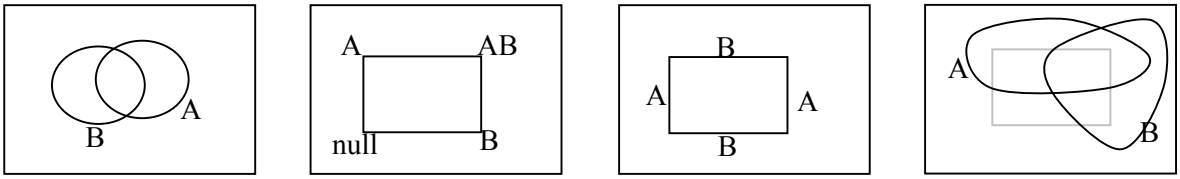
preview dual graph edge labels constructed contours



1.5.2 Two sets

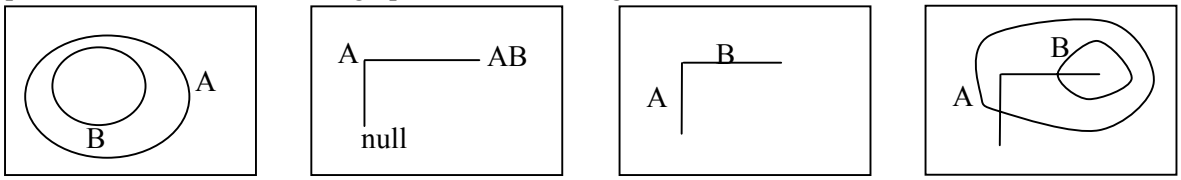
Zone descriptions: {A, B, AB}

preview dual graph edge labels constructed contours



Zone descriptions: {A, AB}

preview dual graph edge labels constructed contours



Other non-empty subsets of {A, AB, B} are

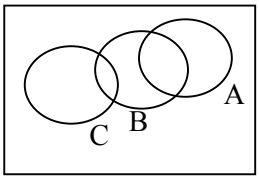
{A, B} (a disjoint union of two singleton diagrams)
{B, AB} (isomorphic to {A, AB})
{A} done as a singleton set
{B} done as a singleton set

and {AB} – here the dual graph is disconnected, so either introduce shaded zones or stop.

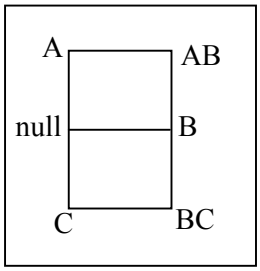
1.5.3 Three sets

Zone descriptors: {A, B, C, AB, AC, BC}

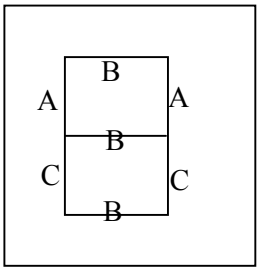
preview



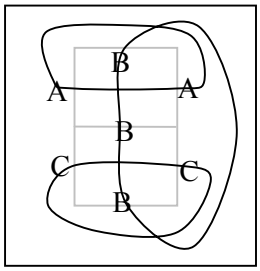
dual graph



edge labels

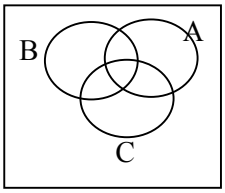


constructed contours

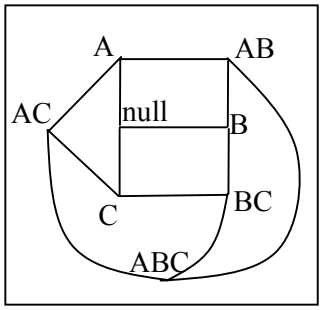


Zone descriptors: {A, B, C, AB, AC, ABC}

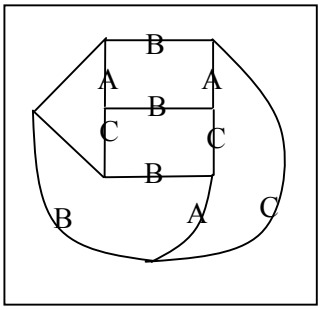
preview



dual graph

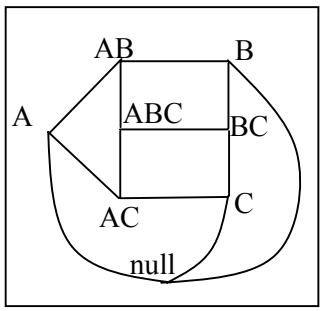


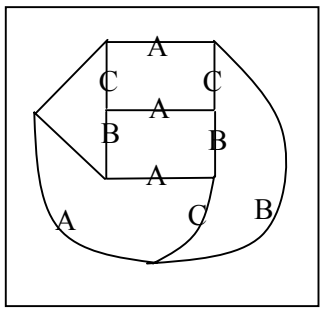
edge labels

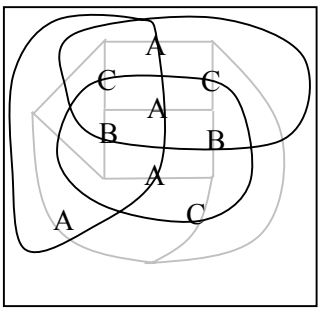


The contour for symbol "A" isn't drawable – there is a problem with the "placing of infinity".

The null-labelled node should be on the outside of the graph diagram. Redraw the graph diagram.







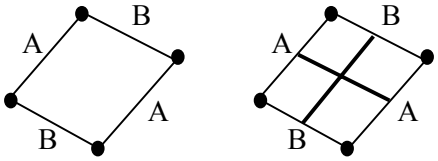
There are 127 ways to write down nonempty subsets of {A, B, C, AB, AC, BC, ABC}.
There is one drawable diagram from the singleton sets.
There are three types of diagram for subsets including only two contours.
There are eleven types of diagram for zone descriptions involving all three contour s.

The next section is more specific about the act of drawing contours. In small examples, it's easy to see how the contours must be drawn to pass through the required edges of the dual graph, and to enclose the required vertices. But an algorithm is required for the process of constructing contours from line segments.

1. 6 Convex faces and circularising

The mechanism for drawing contours in the last section was mainly intuitive –we can *see* a collar round a subgraph. This section specifies construction of "internal arcs" which can cross and "external arcs" which never meet any other line segments. Together, the internal and external arcs for closed contours.

1.6.1 Drawing contours through convex faces

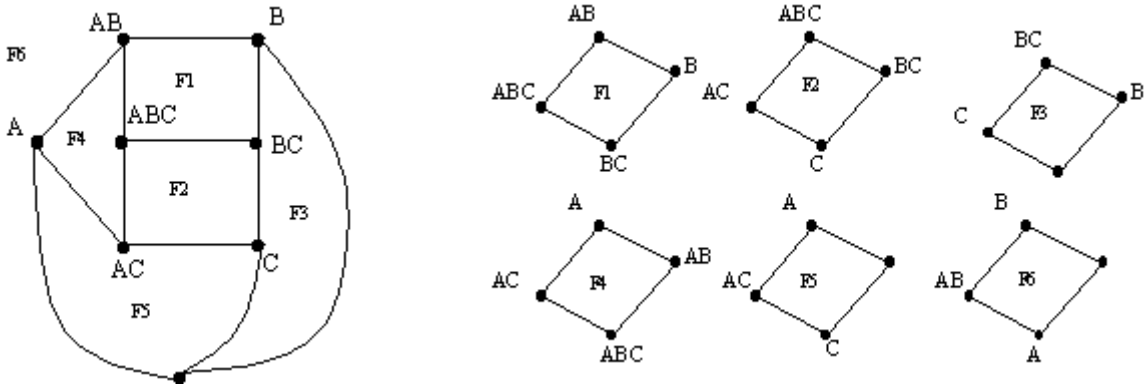


Once we have faces, with edges labelled, we want to be able to join up matching pairs of edge-labels. If the face is convex, this can be done with line segments joining the midpoints of the face edges.

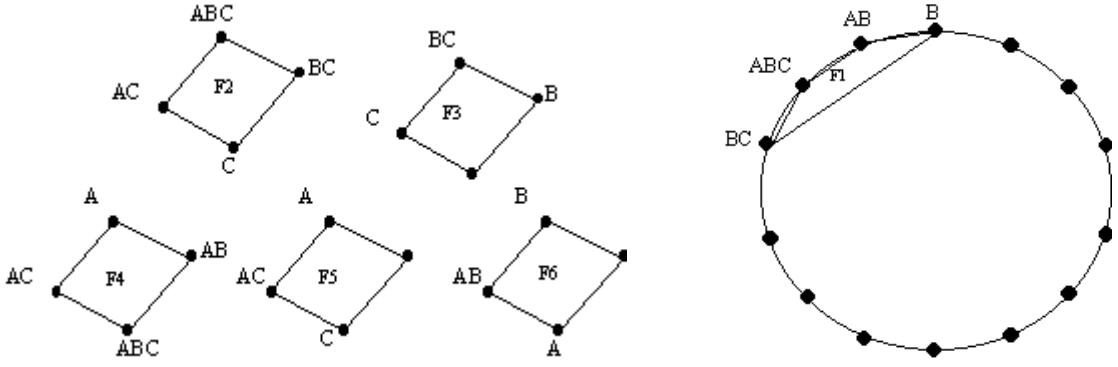
It would be desirable to have all faces convex.

1.6.2 Drawing a planar graph with convex faces

Given a planar representation of a graph, there is an algorithm using a disc which creates a representation with straight edges and convex faces. Take a planar representation, and decompose it into disjoint faces with identified edges:

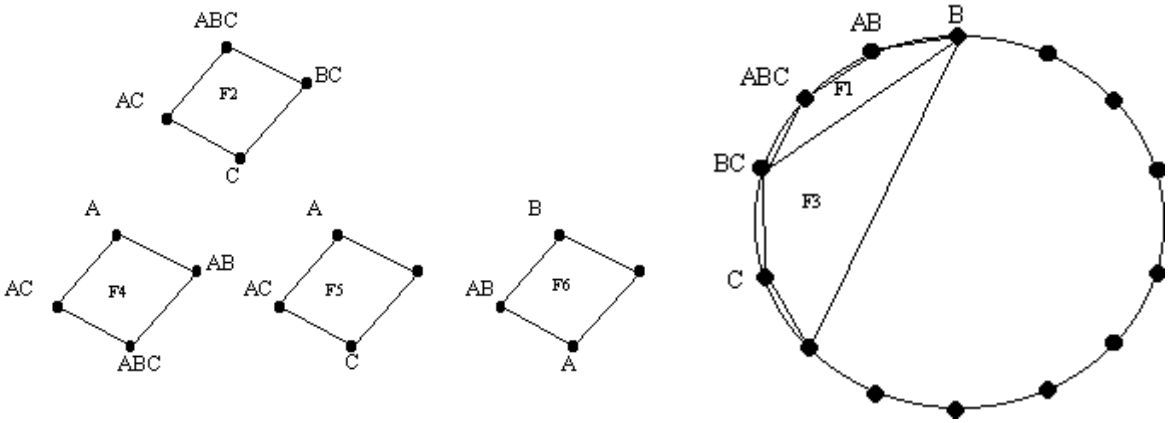


This decomposition into faces includes the outside face –contours have to be drawn there as well as in the inside faces. Each of these faces can be inserted into a disc. To explain the process, insert each face in turn. Choose any first face, for example F1.

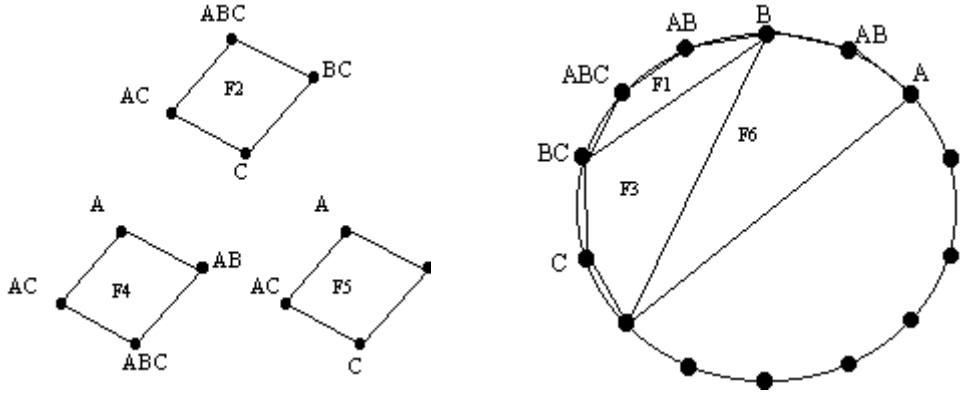


Face F1 in the disc has straight line edges and is convex.

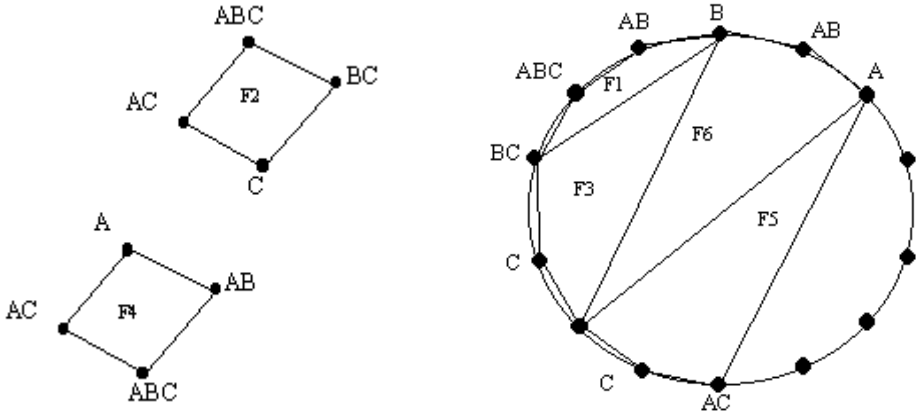
Face F1 into the disc has an "internal edge" in the disc : B–BC. There is another edge B–BC in face F3. This face will be placed in the disc, identifying the two occurrence of B–BC. There are three ways to place F3 in the disc – make sure that the new internal edge has a partner in the faces which are still to be put in the disc. Look at the three faces of F3 that aren't B–BC and find one whose partner isn't yet in the circle. For example null–B .
Draw F3 in the disc so that null–B is internal.



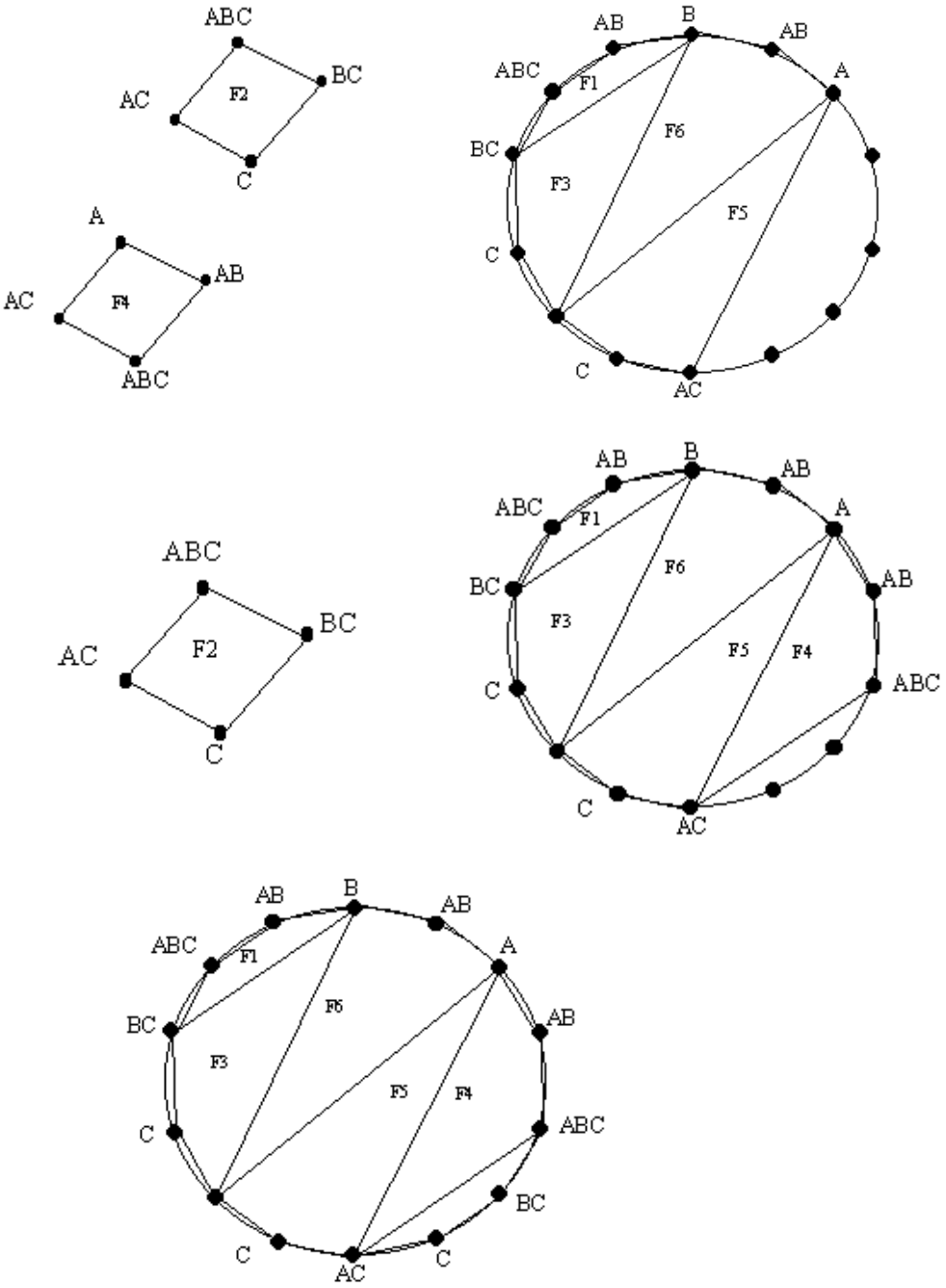
Face F3 in the disc has an internal edge: null–B. Find its partner in a face not yet in the disc. It's in face F6. Choose another edge of F6 which isn't in the disc: for example null–A.
Place F6 with null–A an internal edge.



Place face F5 in the disc adjoining edge null–A with edge C–AC internal.

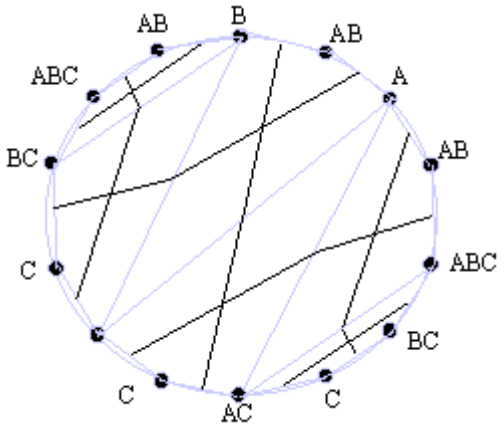


Place F4 in the disc with edge A–AC adjoined, and new internal edge AC–ABC.



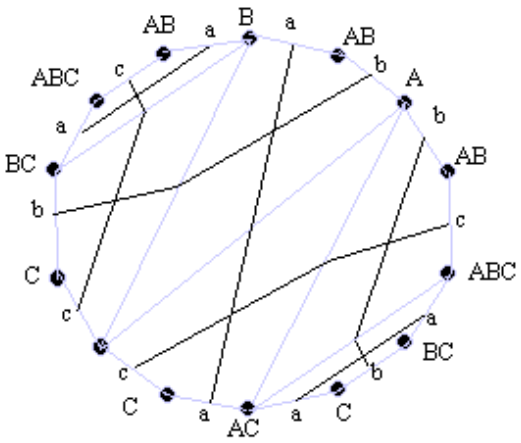
1.7 Building internal arcs and external arcs from a circularised graph

In a circularised dual graph, all faces are convex, so the Euler diagram contours can be built up with straight line segments.
The "internal" (inside the disc) line segments don't form closed contours until outer edge identifications are made. We can resolve this in two ways
- visualise identified outer edges coming together to recover our original graph, with the drawn linear contour segments distorting to follow the change
or
- join up the contours by further linear segments outside the disc.



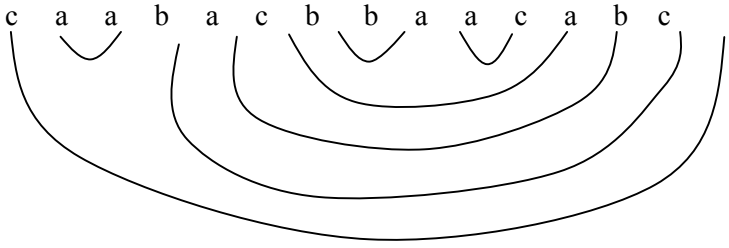
The first, topological, approach is impractical for computer implementation, but pleasing to see how the original graph relates to the graph in the disc.

The second approach will complete the algorithm for constructing polygonal Euler contours.



1.7.1 Topological reconstruction of the dual graph

The circularised graph may be transformed back into the original dual by identifying pairs of outside edges of the disc.

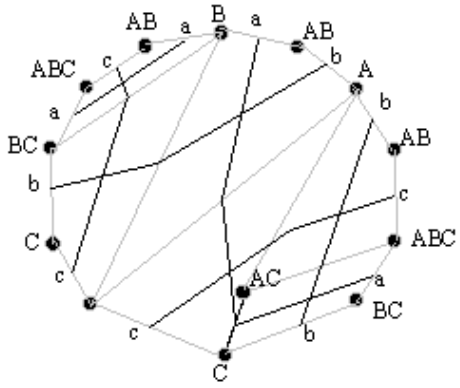


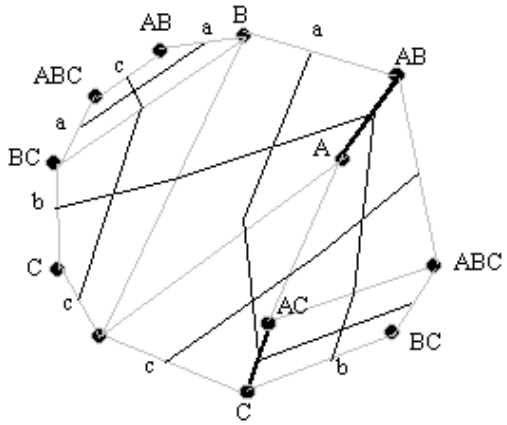
Construct a word (reading round the outside from the null node). Pair up equal letters without arc crossings. There's only one way to do this (like determining bracket-pairs).

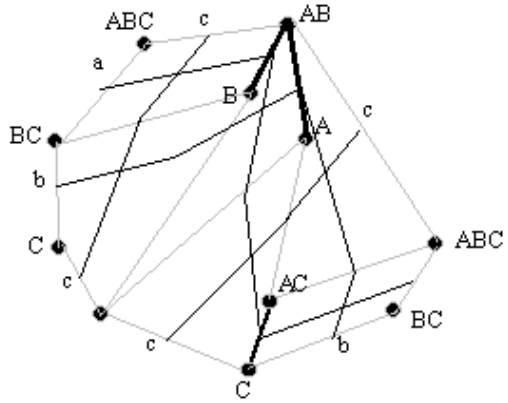
The adjacent pairs labelled a,a, b,b and a,a are matched first. Then, if these letters are removed from the word, another adjacent pairs c,c appears, and so on.

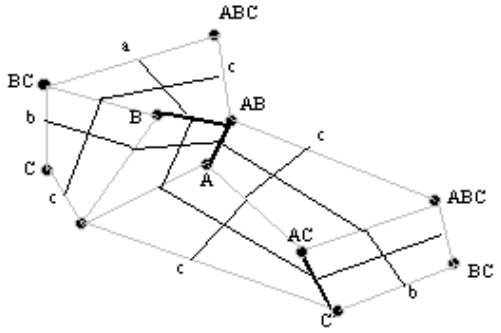
This letter-pairing determines how external arcs are constructed to make closed contours. The next pages sketch a topological argument for the connection between such a bracketing pairing on the outside word and contour construction. Readers in a hurry may wish to skip to the next section : "Constructing contours as polygons".

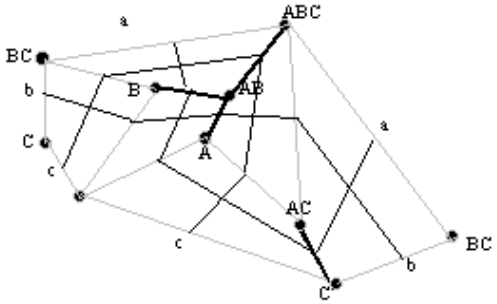
Outside edges corresponding to a,a, b,b, a,a can be folded together, distorting the disc, to identify the pairs edges. Edges c,c then become adjacent and can also be folded together.

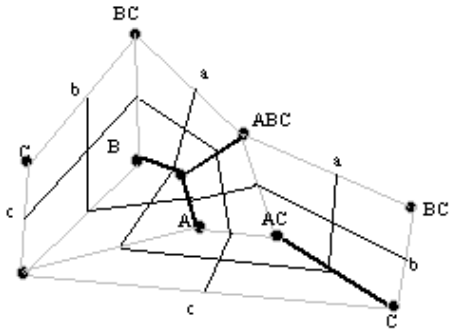


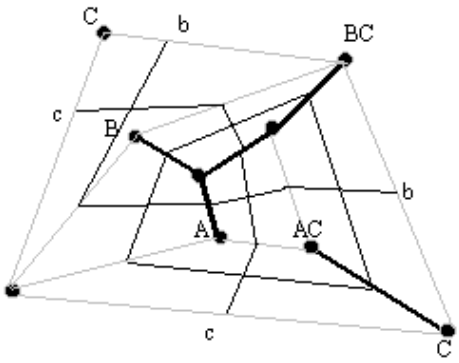


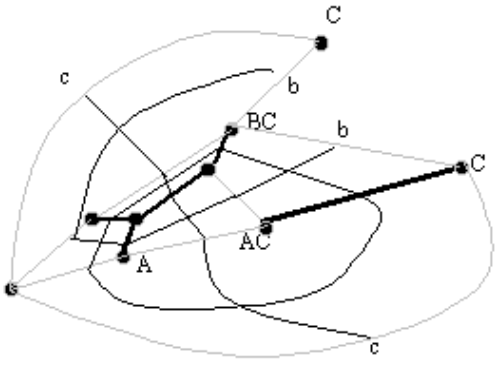


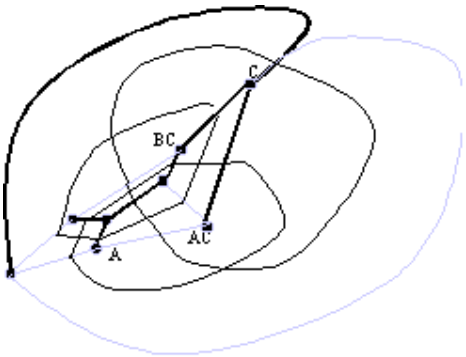






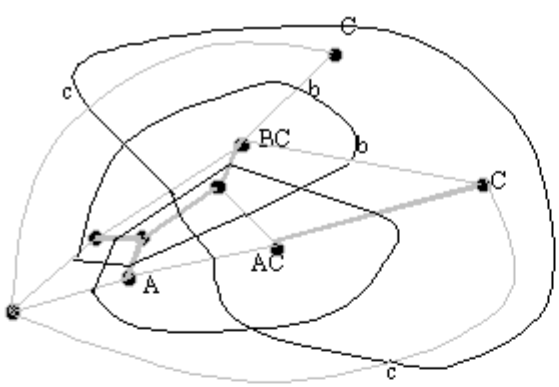


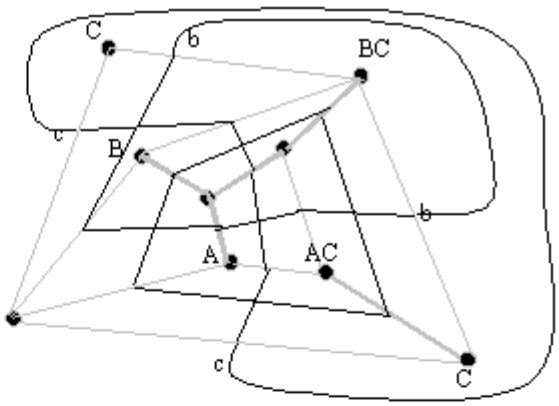


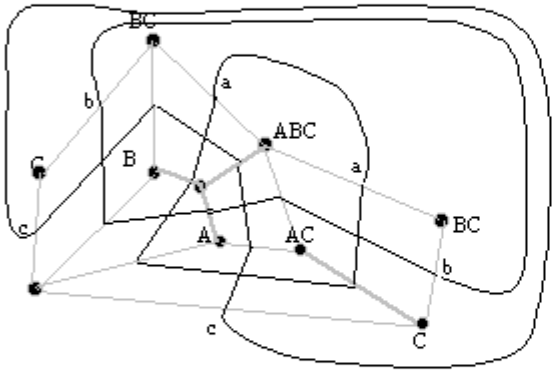


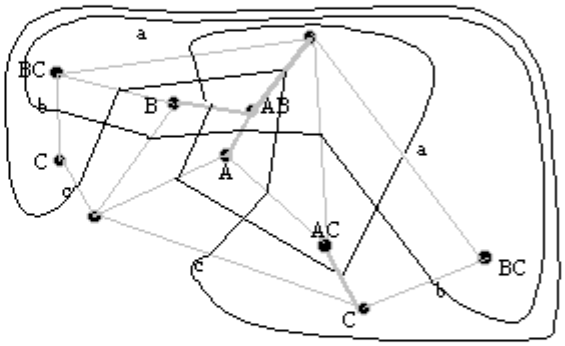
The identified edges (draw as thick black lines) form a spanning tree of the dual graph. The choices which were made in placing the faces inside the disc are equivalent to choices made in the construction of a spanning tree. The contours are no longer polygonal, but the have become closed contours which enclose the appropriate vertices.

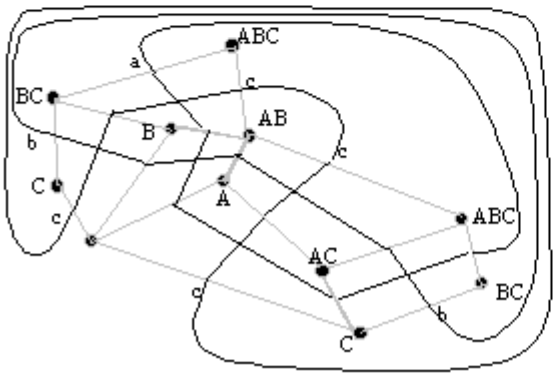
Un-wrap this diagram, splitting identified edges again, but keep track of how the contours are identified.

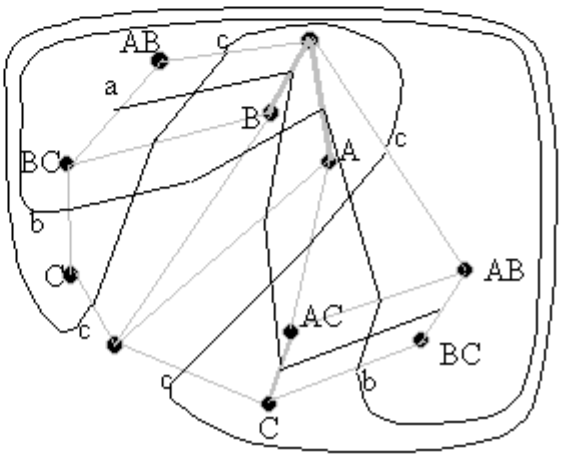


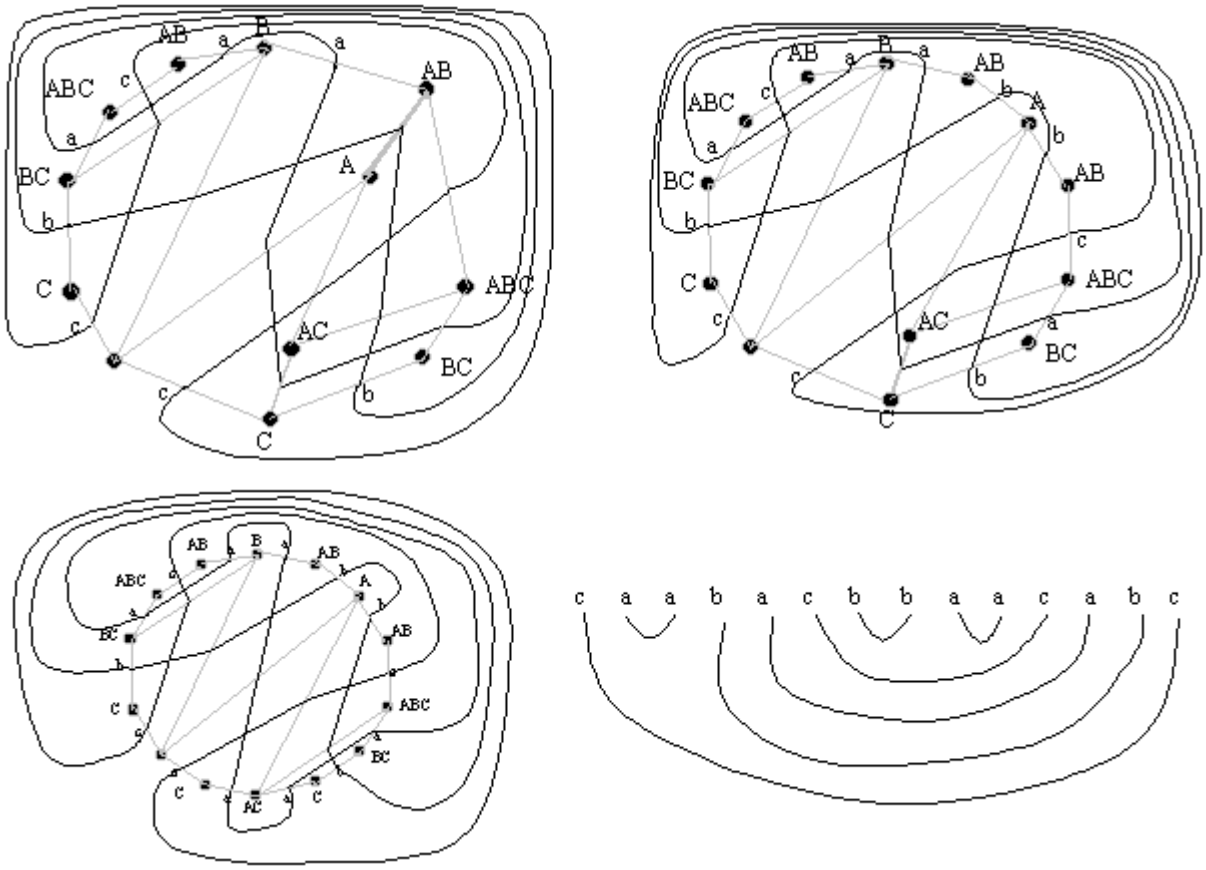






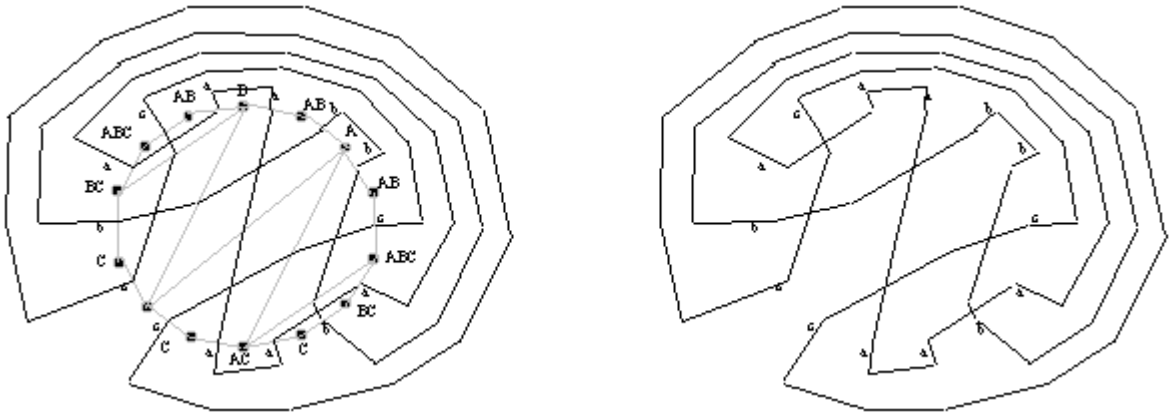






1.7.2 Constructing contours as polygons

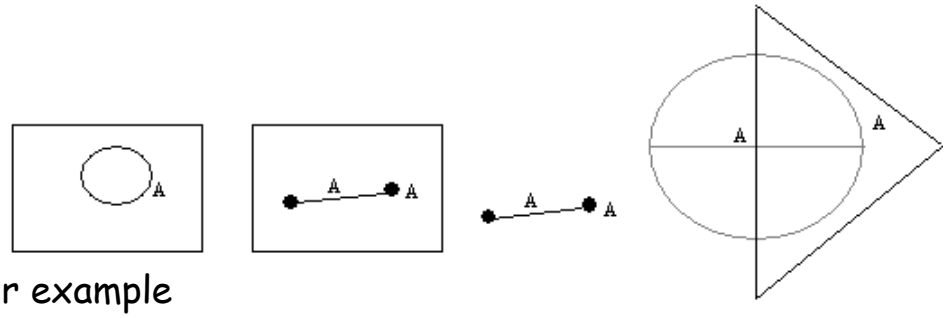
External arcs contributing to the contours can be constructed using line segments as shown below.



1. 8 Examples revisited

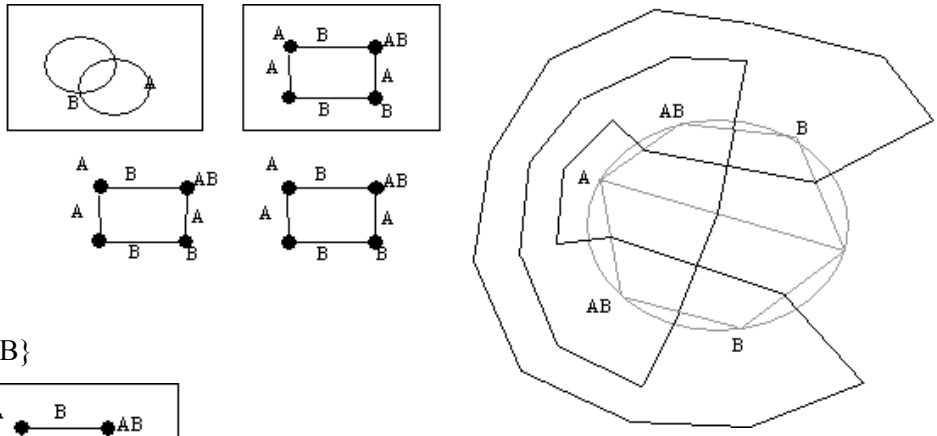
1.8.1 Singleton contour

Zone description: {A}

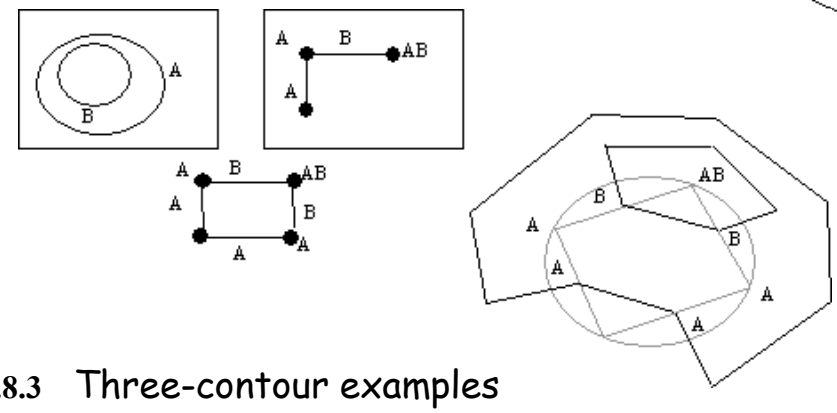


1.8.2 Two-contour example

Zone descriptions:
{A, B, AB}

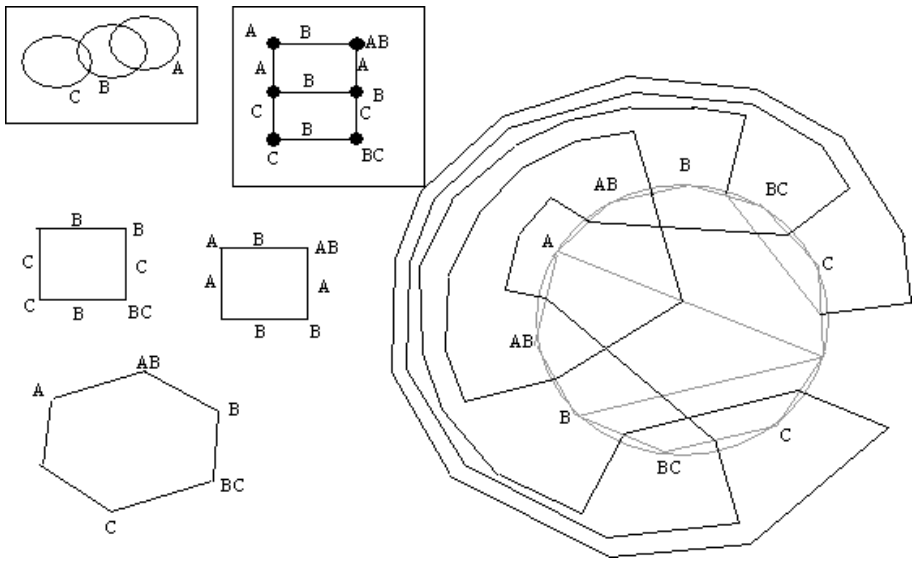


Zone descriptions: {A, AB}



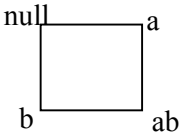
1.8.3 Three-contour examples

Zone descriptors: {A, B, C, AB, AC, BC}



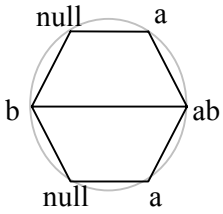
1.9 Why use null as a dual vertex?

Take an example with zones [], [a], [b] and [a,b]

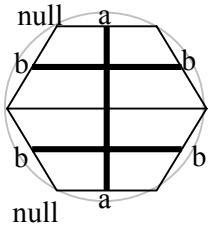


Including null, the dual graph looks like this:

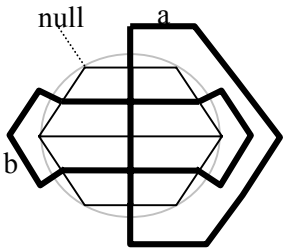
Circularise the dual graph



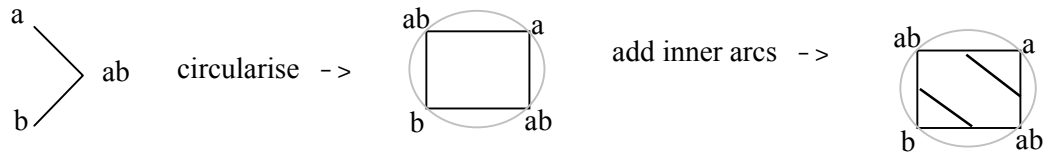
Construct the inner arcs of the constraint diagram:



The external arcs must connect the b's and a's at the ends of the internal arcs. Use the null (or one of them) to read round the outside word, giving something like abbabb. Join up the letters which pair off like brackets, avoiding crossing a "line to infinity" from null.

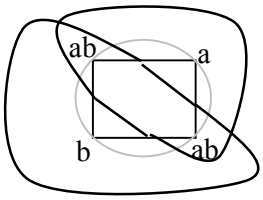


If this process is followed through without the null point included, we get this sequence of diagrams:

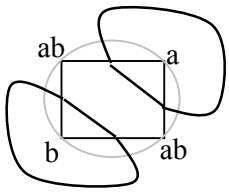


The question remains: how to join up the ends of the inner arcs with external arcs?

The correct result would be this:



not this:



It's not easy to see how to decide between these options, given only the abstract information.

A more serious issue is that the correct result has edge crossings outside the circularised graph. One reason for circularising was to allow the graph to be drawn with straight line segments. Another was to confine all edge-crossing to inside faces which had passed word-checks. Allowing crossings outside the circularised dual graph creates the risk of triple-point drawing obstrcutions outside, even though we have done contour-word checks to eliminate triple crossings inside.