Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science 91 (2004) 246–263

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Automated Theorem Proving with Spider Diagrams

Jean Flower[1,3] and Gem Stapleton[2,4]

The Visual Modelling Group
University of Brighton
Brighton, UK

Abstract

Spider diagrams are a visual notation for expressing logical statements. In this paper we describe a tool that supports reasoning with a sound and complete spider diagram system. The tool allows the construction of diagrams and proofs by users. We present an algorithm which the tool uses to determine whether one diagram semantically entails another. If the premise diagram does semantically entail the conclusion diagram then a proof is presented to the user. Otherwise it gives a counterexample: a model for the premise that is not a model for the conclusion. The proof of completeness given in [8] can be used to create an alternative proof writing algorithm. The algorithm described here improves upon this by providing counterexamples and significantly shorter proofs.

Keywords: spider diagrams, diagrammatic reasoning, automated reasoning, proof writing.

1 Introduction

In this paper we present a theorem proving algorithm for a diagrammatic reasoning system. Such a system is comprised of three things. Firstly, one specifies the *syntax* of the diagrams under consideration. Second, one gives meaning to syntactically correct diagrams: the *semantics*. Finally, *reasoning*

[1] This work was done with the support of the EPSRC under grant GR/R63516. Thanks to Andrew Fish, Chris John, John Howse and John Taylor for their comments on earlier drafts of this paper.

[2] This work was done with the support of the EPSRC under grant 01800274.

[3] Email: j.a.flower@bton.ac.uk

[4] Email: g.e.stapleton@bton.ac.uk

doi:10.1016/j.entcs.2003.12.016

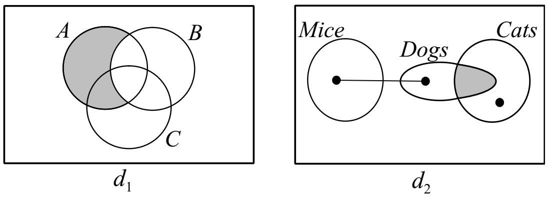


Fig. 1. A Venn diagram and a spider diagram.

rules are specified which transform one diagram into another. These rules must be sound and, ideally, form a complete set. Examples of simple diagrammatic systems are Venn and Euler diagrams. In Venn diagrams [18] all possible intersections between (regions in) contours must occur and shading is used to represent the empty set. Diagram d_1 in Fig. 1 is a Venn diagram. The Venn-II system formalized by Shin [11] is probably the best-known formalization of a diagrammatic reasoning system. Venn-II diagrams extend the Venn diagram notation, using additional syntax to represent non-empty sets.

Euler diagrams exploit topological properties of enclosure, exclusion and intersection to represent subsets, disjoint sets and set intersection respectively. Spider diagrams [6,7,8] are based on Euler diagrams. *Spiders* are used to represent the existence of elements and shading is used to place upper bounds on the cardinalities of sets. A spider is drawn as a collection of dots (the *feet*) joined by lines. The diagram d_2 in Fig. 1 is a spider diagram and expresses the statement "no mice are cats or dogs, no dogs are cats, there is a cat and there is something that is either a mouse or a dog". Sound and complete reasoning rules for spider diagram systems have been given [7,8].

Conceptual graphs [12,10] and Euler-based diagrams have both been used to visually express logical statements and for reasoning [1]. In both cases there is a textual form which can be used as an abstract representation of the diagram, which is especially useful for tool-building. Given a simple conceptual graph it is possible to draw a spider diagram to express the same information. Each type gives rise to a contour in the spider diagram. A concept drawn in a conceptual graph would give a spider (named or otherwise), and relations in conceptual graphs can be represented by contours. As the conceptual graph syntax becomes richer, with for example nested graphs, it gets more difficult to see how such statements could manifest themselves as spider diagrams. Indeed, the expressiveness of conceptual graphs [9] exceeds that of spider diagrams [14]. A more interesting comparison of expressiveness would be between conceptual graphs and constraint diagrams, which include a natural representation of relations between objects [2].

If diagrammatic reasoning is to be practical, then tool support is essential. Proof writing in diagrammatic systems without software support can be time-consuming and error prone. If we wish to find a proof that one diagram semantically entails another, one strategy could be to convert both diagrams into first order logic statements and use existing proving environments. However, we seek to create purely diagrammatic proofs, which can provide feedback to users who are modelling with diagrams. In [15] an approach towards implementing an Euler/Venn reasoning system is proposed, using directed acyclic graphs. These graphs are similar to our *abstract syntax* and they capture the "essential properties" of diagrams similar to Venn-II diagrams. This implementation is further discussed in [16] and the focus of the tool is on checking the correctness of user applied reasoning steps but the tool does not automate proof writing.

In this paper we discuss a proving environment for spider diagrams, implemented in java. By implementing this tool we show that it is possible to fully automate diagrammatic theorem proving. The tool we have implemented allows users to construct spider diagrams and write proofs, as well as automating proof construction. Given two diagrams, d_1 and d_2, if we wish to know whether d_1 semantically entails d_2, the tool will give one of two responses. If d_1 semantically entails d_2 then the tool will provide a proof, otherwise it provides a counterexample.

Spider diagrams form the basis of the much more expressive constraint diagram notation. Constraint diagrams include further syntactic elements, for example *universal spiders* and *arrows*. Universal spiders represent universal quantification (spiders in spider diagrams represent existential quantification). Arrows denote relational navigation. Semantics are given to constraint diagrams in [2] and a constraint diagram reasoning system (with restricted syntax and semantics) is introduced in [13]. Since the constraint diagram notation extends the spider diagram notation, developing this tool is a significant step towards the development of such a tool for constraint diagrams.

The implementation uses an algorithm which has some steps used in the completeness proof given in [8]. In [13], the authors state that the strategy used to prove completeness in spider diagram systems extends to a constraint diagram system. Thus it is likely that the proof writing algorithm we present here (and therefore our tool) can be extended to prove theorems with constraint diagrams.

One application for diagrammatic reasoning systems is for expressing, and reasoning about, constraints in object-oriented models. The Unified Modeling Language (UML) is a collection of mainly diagrammatic notations that are used by software engineers in the process of object-oriented modelling. The only non-diagrammatic notation in the UML is the Object Constraint Language (OCL). The OCL is, essentially, a stylized form of first order predicate logic and is used to convey formal statements. Spider diagrams and constraint diagrams complement the diagrammatic theme of the UML and provide an alternative, perhaps more intuitive, notation to the OCL.

2 Spider Diagrams

We now give an informal description of unitary spider diagrams. More details can be found in [8]. A **contour** is a labelled, simple closed plane curve. A **boundary rectangle** is a simple closed plane curve and is not labelled. A **basic region** is the set of points enclosed by a contour or the boundary rectangle. A **region** is defined recursively: any basic region is a region and any non-empty union, intersection or difference of regions is a region. A **zone** is a region having no other region contained within it. A region is **shaded** if each of its component zones is shaded. A **spider** is a tree with nodes, called **feet**, placed in different zones. A spider **touches** a zone if one of its feet appears in that zone. A spider, s, is said to **inhabit** the region which is the union of the zones it touches. This region is called the **habitat** of s. A **unitary diagram** is a finite collection of contours, shading and spiders properly contained by a boundary rectangle. A zone can be described by the set of labels of the contours that contain it (the containing label set) and the set of labels of the contours that exclude it (the excluding label set). We will define two zones to be equal if they have the same containing label set and excluding label set, even if they are in different diagrams.

The diagram d_2 in Fig. 1 (in section 1) contains three labelled contours and five zones, of which one is shaded. There are two spiders. The spider with one foot inhabits the zone inside (the contour labelled) $Cats$, but outside $Dogs$ and $Mice$. The other spider inhabits the region which consists of the zone inside $Mice$ and the zone inside $Dogs$ but outside $Cats$.

Unitary diagrams form the building blocks of **compound diagrams**. To enable us to present disjunctive and conjunctive information, we use connectives: $\sqcup$ and $\sqcap$. If D_1 and D_2 are spider diagrams then so are $D_1 \sqcup D_2$ ("D_1 or D_2") and $D_1 \sqcap D_2$ ("D_1 and D_2"). Fig. 2 shows a compound diagram $d_1 \sqcup d_2$. Our convention is to use lower case d for unitary diagrams, and upper case D for diagrams which may be unitary or compound.

Regions in spider diagrams represent sets. The region that comprises all the zones in a unitary diagram represents the universal set. A spider represents the existence of an element in the set represented by its habitat. Distinct spiders represent the existence of distinct elements. In the set represented by a shaded region, all of the elements are represented by spiders. Thus we can express lower and upper bounds on the cardinalities of sets. For the compound

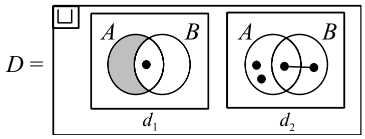


Fig. 2. A spider diagram $D = d_1 \sqcup d_2$.

diagram $D_1 \sqcup D_2$ ($D_1 \sqcap D_2$) the semantics are given by taking the disjunction (conjunction) of the semantics of D_1 and D_2. Given an **interpretation** m, that is, a universal set, U, and a mapping from zones to subsets of U, we can decide whether the semantics of D are true in m. If the semantics of D are true, we say m is a **model** for D, denoted $m \models D$, and say that m **satisfies** D. Every unitary diagram is satisfiable. Formal semantics can be found in [8].

The diagram d_1 in Fig. 2 expresses that (the set represented by) $A - B$ is empty and there is at least one element in $A \cap B$. Diagram d_2 expresses that there are at least two elements in $A - B$ and there is at least one element in B. Thus, if we define the universal set to be $U = \{1, 2\}$ and map the outside zone to $\emptyset$, the zone in just A to $\emptyset$, the zone in both A and B to $\{1\}$ and the zone in just B to $\{2\}$ then this interpretation is a model for d_1 but not for d_2. Since this interpretation is a model for d_1, it is also a model for $d_1 \sqcup d_2$.

3 Reasoning Rules

In this section we give informal descriptions of the syntactic reasoning rules for spider diagrams. Each rule is expressed as a transformation of one spider diagram into another. Formal descriptions of the rules can be found in [8]. Firstly, we consider rules that are applied to unitary diagrams.

Rule 1 ***Introduction of a contour (reversible).*** *A contour can be introduced to a unitary diagram provided the following occurs. The new contour has a label not present in the diagram. Each zone splits into two zones and shading is preserved. Each foot of each spider is replaced by a connected pair of feet, one in each new zone.*

Rule 2 ***Introduction of a shaded zone (reversible).*** *If unitary diagram d is not based on a Venn diagram then we can introduce a new, shaded, zone that is not part of the habitat of any spider, to d.*

In Fig. 3, diagram d_2 is obtained from d_1 by applying rule 1, adding the contour labelled B. To obtain d_3 from d_2 a zone is removed, using the reverse of rule 2.

Rule 3 ***Splitting spiders (reversible).*** *Let d be a unitary diagram con-*

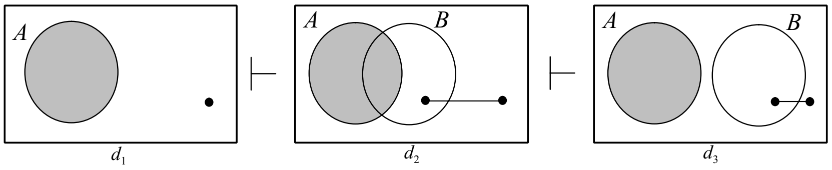


Fig. 3. Applications of rule 1 and (the reverse of) rule 2.

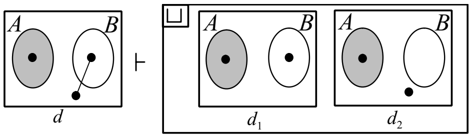


Fig. 4. An application of rule 3.

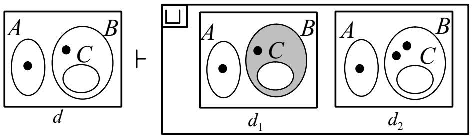


Fig. 5. An application of rule 4.

taining a spider, e, whose habitat has a partition into regions r_1 and r_2. We can replace d by $d_1 \sqcup d_2$, where d_1 and d_2 are copies of d except that the habitat of e is reduced to r_1 in d_1 and r_2 in d_2.

The diagram d in Fig. 4 has a spider with two feet. Its habitat has a partition into two zones, one inside B and the other inside $U - (A \cup B)$. We can split this spider into two parts, giving $d_1 \sqcup d_2$.

Rule 4 ***Excluded middle (reversible).*** *Let d be a unitary diagram with a non-shaded region, r. We can replace d by $d_1 \sqcup d_2$, where d_1 and d_2 are copies of d except that r is shaded in d_1 and there is an additional spider in d_2 with habitat r.*

The excluded middle rule is applied to diagram d in Fig. 5. We shade $B - C$ (giving d_1) and add a spider to $B - C$ (giving d_2), as shown in $d_1 \sqcup d_2$.

Rule 5 ***Erasure of shading.*** *If d is a unitary diagram with a shaded region r we may erase the shading from r.*

Rule 6 ***Erasure of a spider.*** *If d is a unitary diagram with a spider s whose habitat is a completely non-shaded region then we may erase s from d.*

The next rule we give replaces a conjunctive compound diagram all of whose components are unitary by a single unitary diagram. This process is performed on **α-diagrams** – diagrams where each spider has only one foot – with the same zone sets. For example, diagram $d_1 \sqcap d_2$ in Fig. 6 can be replaced by the unitary diagram d_3. The number of spiders in any zone in d_3

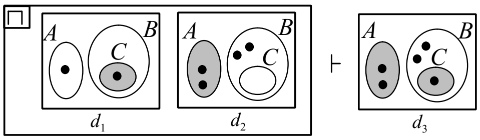


Fig. 6. Combining unitary α-diagrams.

is the maximum number in that zone in d_1 and d_2 and a zone is shaded in d_3 if that zone is shaded in either d_1 or d_2. The resulting diagram d_3 is called the *combined diagram* of $d_1 \sqcap d_2$. Sometimes the components of a conjunction represent contradictory information. The symbol $\perp$, called a **false diagram**, is defined to be a unitary diagram which has no models. This symbol can be introduced or removed using reasoning rules analogous to those in predicate logic. It can also be introduced using the *combining rule*, which we now define. For diagram $D = \prod_{d_i \in \mathcal{D}} d_i$, where each d_i is a unitary α-diagram and each pair d_i, d_j, have the same zone sets or one of the d_is is $\perp$, we define the **combined diagram** D^* (which is a unitary diagram) as follows:

(i) If, for any $d_i \in \mathcal{D}$, $d_i = \perp$ then $D^* = \perp$.

(ii) If a zone is shaded in one unitary component of D and contains more spiders in another then $D^* = \perp$.

(iii) Otherwise, D^* has the same zones as each $d_i \in \mathcal{D}$. The shaded zones of D^* are the zones that are shaded in at least one $d_i \in \mathcal{D}$. For each zone, z, in D^* if d_i has n_i spiders in z then D^* has $max\{n_i : d_i \in \mathcal{D}\}$ spiders in z.

Rule 7 ***Combining (reversible).*** *We may replace $D = \prod_{d_i \in \mathcal{D}} d_i$, where each d_i is a unitary α-diagram and each pair d_i, d_j, have the same zone sets or one of the d_is is $\perp$, by the combined diagram D^*.*

There are many rules (not all reversible), omitted for space reasons, that have analogies in propositional logic, for example inconsistency and associativity.

Let D_1 and D_2 be diagrams. We say D_2 is **obtainable** from D_1, denoted $D_1 \vdash D_2$, if and only if there is a sequence of diagrams $\langle D^1, D^2, ..., D^m \rangle$ such that $D^1 = D_1$, $D^m = D_2$ and, for each k where $1 \leq k < m$, D^k can be transformed into D^{k+1} by a single application of one of the reasoning rules. Such a sequence of diagrams is called a **proof** from **premise** D_1 to **conclusion** D_2. If every model for D_1 is also a model for D_2, then D_1 **semantically entails** D_2, denoted $D_1 \vDash D_2$. A reasoning rule, r, is **valid** if, whenever D_2 is obtained from D_1 by one application of r, then $D_1 \vDash D_2$. All the above reasoning rules are valid. Hence the system is sound.

Theorem 3.1 ***Soundness and Completeness.*** *Let D_1 and D_2 be spider* *diagrams. Then* $D_1 \vdash D_2$ *if and only if* $D_1 \vDash D_2$ [8].

4 A Proof Writing Algorithm

The completeness proof given in [8] has been directly converted into a very inefficient proof writing algorithm. In this paper we present a more efficient algorithm which produces much shorter proofs, see subsection 4.4. Given D_1 and D_2, our algorithm establishes a proof that $D_1 \vdash D_2$ (when $D_1 \vDash D_2$) or finds a counterexample (when $D_1 \nvDash D_2$). The first step in our algorithm is

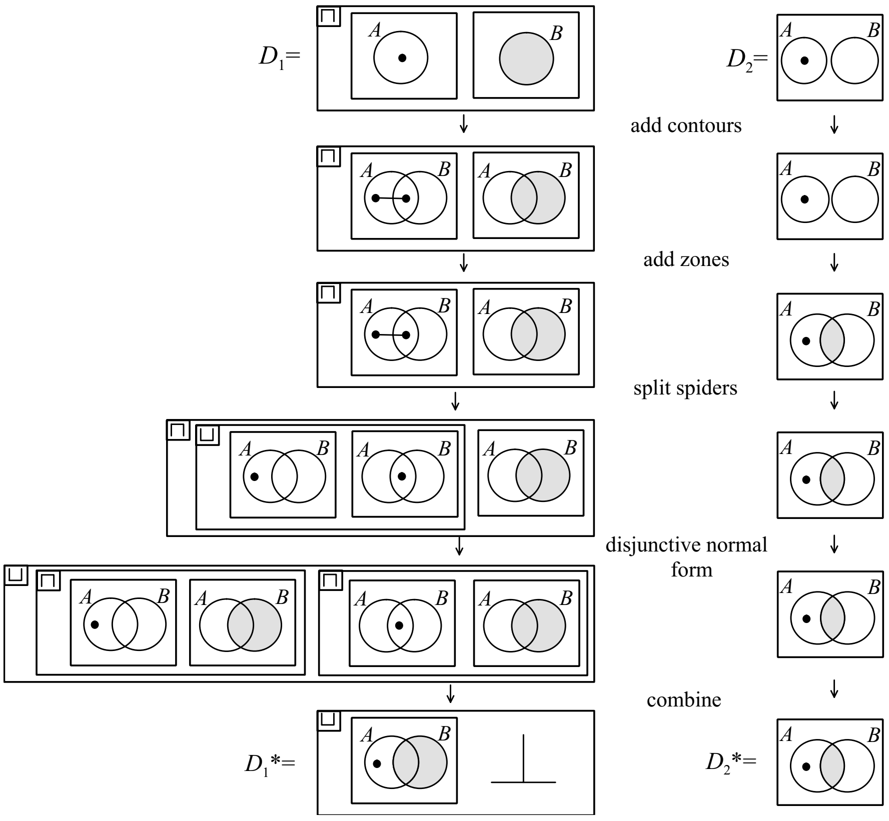


Fig. 7. Applying the algorithm.

to introduce contours (rule 1) to components of both D_1 and D_2 until each unitary component possesses the same label set. Secondly we introduce zones (rule 2) to each unitary component until all the unitary components have the same zone sets. Next, we split spiders (rule 3) to transform the diagrams into α-diagrams. At this stage, all unitary components have the same contour label sets, the same zones sets and all the spiders have exactly one foot. Next, we

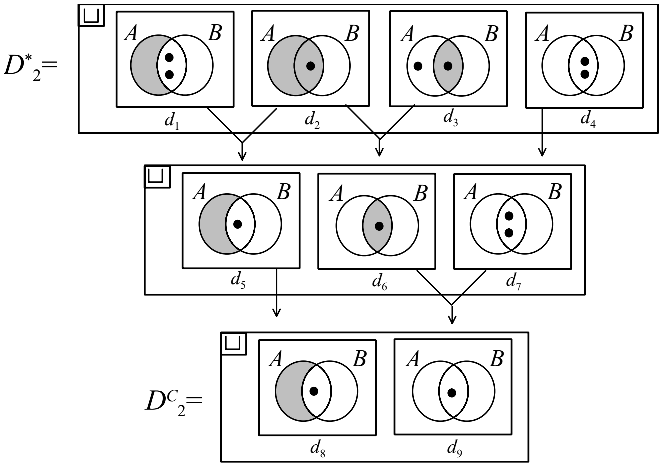


Fig. 8. Contracting D_2^*.

convert to disjunctive normal form. We remove all the conjuncts by combining (rule 7) and we denote the resulting diagrams D_1^* and D_2^*. An example of this process so far can be seen in Fig. 7.

If D_1^* has only $\perp$ as its unitary components then we are done. Suppose now that D_1^* contains non-false components. We remove all occurrences of $\perp$ and, for notational economy, we again call the resulting diagram D_1^*. The next step in the algorithm is to contract D_2^* excluded middle (rule 4) until no further applications of excluded middle are possible giving diagram D_2^C. For example, in Fig. 8 diagram $d_1 \sqcup d_2 \sqcup d_3 \sqcup d_4$ can be contracted (after duplicating d_2) to $d_8 \sqcup d_9$. Note here that $D_2^C \vdash D_2$, since all the rules used to obtain D_2^C are reversible. Also note that $D_1^* \vdash D_1$, so there is a proof from D_1 to D_2 if and only if there is a proof from D_1^* to D_2^C. Our task of seeking a proof from general D_1 to D_2 has reduced to seeking a proof from D_1^* to D_2^C where these are both (flat) disjunctions of unitary α-diagrams.

We compare each unitary component d_1 of D_1^* with D_2^C. For each comparison we check to see whether one of the following holds:

(i) d_1 is in *contradiction* with D_2^C (defined in subsection 4.1)

(ii) d_1 is a *super-diagram* of some component d_2 of D_2^C (defined in subsection 4.2)

The algorithm terminates when (i) holds for some d_1 or (ii) holds for all d_1.

If (i) holds for some d_1 then a contradiction can be generated from d_1 as outlined in subsection 4.1.

If (ii) holds for all d_1 then D_1^* can be transformed into D_2^C. To transform D_1^* into D_2^C, we erase shading and spiders from the unitary components of D_1^*, transforming each component into a sub-diagram that occurs in D_2^C, again we call the resulting diagram D_1^*. Next, we identify any unitary components in D_2^C that do not occur in D_1^*. We use the rule $D \vdash D \sqcup D'$ to join all such unitary components of D_2^C to D_1^*. To complete the transformation we change D_1^* by removing duplicated diagrams or by duplicating diagrams as necessary (using idempotency), until $D_1^* = D_2^C$.

Finally, if there is a d_1 for which neither (i) nor (ii) holds (that is, the algorithm has not terminated) then apply the excluded middle rule to d_1 according to subsection 4.3. Once the excluded middle rule has been applied, repeat the checking of components in D_1^* against D_2^C.

4.1 Diagrams in Contradiction

Let d_1 and d_2 be unitary α-diagrams with the same zone sets. We say d_1 and d_2 are in **contradiction** with one another if the combined diagram for $d_1 \sqcap d_2$ is $\perp$. If d_1 is in contradiction with all the unitary components of diagram D then we say d_1 is in **contradiction** with D.

Lemma 4.1 *Let d ($\neq\perp$) be a unitary α-diagram which is in contradiction with D, a disjunction of unitary α-diagrams. Assume also that every unitary component of D is $\perp$ or has the same zones as d. Then there exists a model for d that is not a model for D.*

Proof. We generate such a model, m, by taking the universal set to be the set of spiders in d. In m each zone in d represents the set of spiders inhabiting that zone.

If D contains only false components then m is not a model for D and we are finished.

Suppose instead that D contains at least one non-false component. Choose an arbitrary non-false component d'. Since the combined diagram for $d \sqcap d'$ is $\perp$ it follows that we can choose a zone, z, that has more spiders in one of d and d' and is shaded in the other. If z contains more spiders in d than in d' then in d', not all of the elements are represented by spiders in z, so m is not a model for d'. Alternatively, z is shaded in d and contains more spiders in d'. In this case, distinct spiders in z in d' do not represent distinct elements, so m is not a model for d'. Since d' was an arbitrary non-false component of D, it follows that m is not a model for any component of D. Therefore m is not a model for D. □

Thus, if there is a unitary component, d_1 say, in D_1^* that is in contradiction with D_2^C, then there is model for a D_1^* that is not a model for D_2^C (since D_1^* is a disjunction of unitary diagrams).

4.2 Super-diagrams and Sub-diagrams

Diagram d_1 is a **super-diagram** of d_2 and d_2 is a **sub-diagram** of d_1, if and only if the following hold.

(i) The diagrams d_1 and d_2 have the same zones.

(ii) All the shading in d_2 occurs in d_1.

(iii) All the spiders in d_2 occur in d_1.

(iv) If a zone z is shaded in d_2 then z has the same number of spiders in d_1 and d_2.

A diagram can be transformed into any sub-diagram by erasing first the extra shading (rule 5) then the extra spiders (rule 6).

4.3 Applying the Excluded Middle Rule

In this section we describe how to apply the excluded middle rule intelligently to a component d_1 of D_1^*. We will find, in D_2^C, the unitary diagram 'most like' d_1 that is not in contradiction with d_1.

Define the **excluded middle measure**, denoted $m(d_1, d_2)$, between two unitary diagrams as follows. Let d_1 and d_2 be two unitary α-diagrams with the same zone sets. Let $Sh(d_1, d_2)$ denote the number of zones that are shaded in d_2 but not shaded in d_1. Let Z^u denote the set of non-shaded zones in d_1. Let $Sp(z, d_i)$ denote the number of spiders inhabiting z in d_i. If d_1 and d_2 are not in contradiction then

$$m(d_1, d_2) = Sh(d_1, d_2) + \sum_{z \in Z^u} max\{Sp(z, d_2) - Sp(z, d_1), 0\}.$$

Roughly speaking, this means that the measure is large if d_2 has more spiders and shaded zones than d_1. Note that if $m(d_1, d_2) = 0$ then d_2 is a sub-diagram of d_1. For diagrams d_1 and d_2 which are in contradiction, the measure is defined to be ∞.

Given d_1 and D_2^C, not in contradiction, find $m(d_1, d_2)$ for each d_2 in D_2^C and choose a d_2 with $m(d_1, d_2)$ minimal. This minimal measure is finite. We define Z to be a set comprising all the zones of d_2 with extra shading or spiders. These are the zones which contributed to the sum $m = m(d_1, d_2)$. Apply the excluded middle rule to d_1, and its derivatives, in turn, m times, to generate a sequence

$d_1 = e_0,$

$d_1' \sqcup e_1,$

$d_1' \sqcup d_2' \sqcup e_2,$

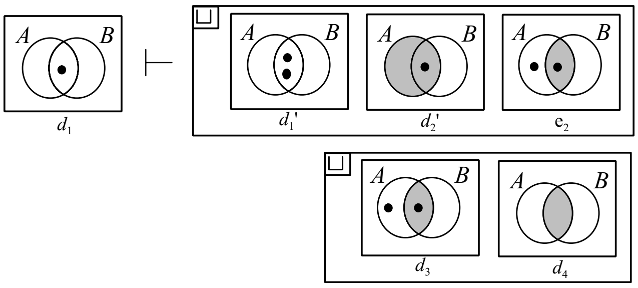


Fig. 9. Applying excluded middle to create a superdiagram.

$\vdots$

$d_1' \sqcup d_2' \sqcup d_3' ... \sqcup e_{m-1}$,

$d_1' \sqcup d_2' \sqcup d_3' ... d_m' \sqcup e_m$.

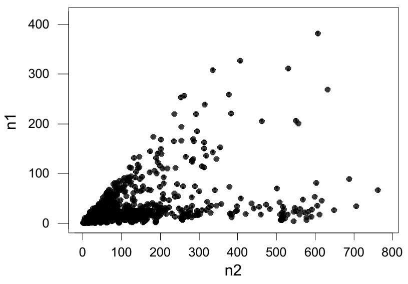
The zones in Z have matching spiders and shading in e_m and in the target component d_2. The other zones have at least as many spiders and shading in e_m as in d_2. Hence e_m is a super-diagram of d_2. In this sense, the compound diagram D_1^* is *closer* to D_2^C after application of excluded middle.

For example, in Fig. 9 diagrams d_1 and d_3 have $m(d_1, d_3) = 2$. To transform d_1 into d_3 we need to add a spider to the zone in A but not B, and shading to the zone in both A and B. Two applications of the excluded middle rule can be used to add these elements to d_1, resulting in $d_1' \sqcup d_2' \sqcup e_2$. Diagrams d_1' and d_2' are both in contradiction with d_3: d_1' contains more spiders in the zone that is shaded in d_3 and d_2' has a shaded zone that contains more spiders in d_3. Also, diagram d_1 is in contradiction with d_4. Because we have not erased elements when creating d_1' and d_2', it follows that d_1' and d_2' are also in contradiction with d_4.

In order to ensure termination of the algorithm, consideration must be given to each of the intermediate diagrams $d_1',..,d_m'$. Recall that we are changing D_1^* until either all its unitary components are a super-diagram of some component of D_2^C or some component of D_1^* is in contradiction with D_2^C. By applying the excluded middle rule to d_1 we have replaced d_1 by a disjunction of unitary diagrams. One of these new diagrams, e_m, is a super-diagram of d_2. The other components, $d_1', ..., d_m'$, are in contradiction with more unitary components of D_2^C than d_1, as we now show. If d_1 is in contradiction with d in D_2^C then all d_i' are in contradiction with d. Moreover, each d_i' is in contradiction with d_2. This can be seen by noting the existence of a zone, z, such that

(i) z, is shaded in d_i' and not shaded in d_2 or

(ii) z is inhabited by exactly one more spider in d_i' than in d_2.



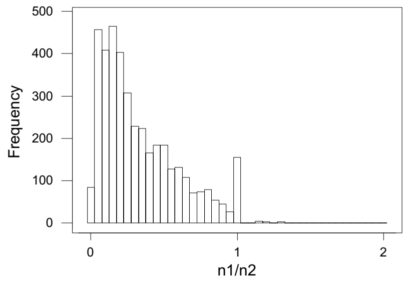


Fig. 10. The raw data and a histogram showing the frequencies for each ratio.

If z is shaded in d_i' then z contains more spiders in d_2. If z contains exactly one more spider in d_i' then z is shaded in d_2. Thus, for each d_i' there are more contradictory components in D_2^C than there were for d_1. Equivalently, for each d_i' there are fewer unitary components, d_j'', of D_2^C than there are for d_1 for which $m(d_i', d_j'')$ is finite. Since there are only finitely many unitary components in D_2^C the algorithm will terminate.

Theorem 4.2 ***Decidability.*** *Let D_1 and D_2 be spider diagrams. There is an algorithm which determines whether or not $D_1 \vdash D_2$.*

4.4 Empirical Results

In order to justify the claim that the proof writing algorithm we present in this paper is more efficient than the algorithm derivable from the completeness proof in [8] we generated a random sample (size $n = 4000$) of pairs of 'small' diagrams, D_1 and D_2 for which $D_1 \vDash D_2$. By small we mean with few contours ($\leq$ 3 in each unitary diagram) and few unitary parts ($\leq$ 4). Once we had randomly generated D_1 we (randomly) applied a sequence of reasoning rules to D_1 to give D_2. We then took each of these pairs of diagrams and constructed a proof that $D_1 \vdash D_2$ using each of the algorithms. For each pair of diagrams we calculated the ratio $\frac{n_1}{n_2}$ where n_1 is the length of the proof generated from the algorithm we present and n_2 is length of the proof generated by the algorithm arising from the completeness proof. A scatter plot of the raw data and a histogram showing the ratios obtained and their frequencies can be seen in Fig. 10. We found that the algorithm we present creates, on average, less than 35% of the number of proof steps that the algorithm arising from the completeness proof creates.

5 Implementation

The algorithm given above has been implemented in java and can be downloaded from [19]. The application allows users to create and edit diagrams. Diagrams are represented by their *abstract syntax* rather than their *concrete syntax* (see [8]). This means that a diagram is modelled solely in terms of the labels, zones, shading and spiders it possesses. Each zone is modelled in terms of the contours it is inside and excluded from. For unitary diagrams, users specify the contour set, the zone set, shading and spiders. A zone appears on the screen as a list of contours that it is inside. The diagram in Fig. 11 would appear to a user as

contours: $[A, B]$

zones: $[outside, A, B]$

shaded zones: $[B]$

spiders: $[[A], [A], [A, B]]$.

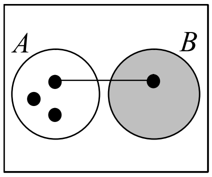


Fig. 11. A drawn diagram: from concrete to abstract.

Compound diagrams can be built from their unitary components. The composite pattern shown in Fig. 12, is used to build compound diagrams. Objects of the type `OrCompoundDiagram` and `AndCompoundDiagram` hold a collection of other diagram objects, referred to as their *children*. The diagram $d_1 \sqcap (d_2 \sqcup d_3)$ is of type `AndCompoundDiagram` with two children, one for the diagram d_1 and another which is of type `OrCompoundDiagram` with children d_2 and d_3.

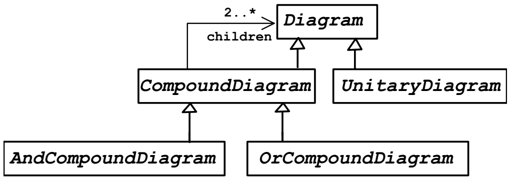
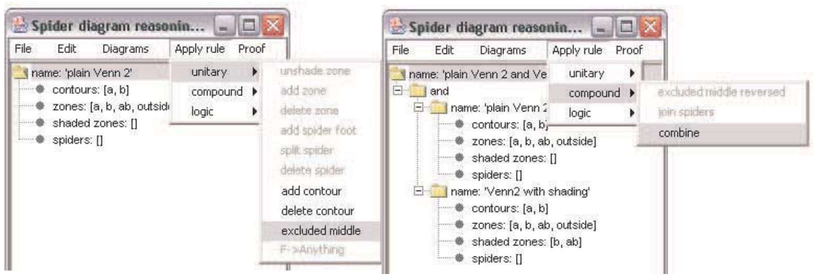


Fig. 12. The composite pattern for compound diagrams.

One decision made during the implementation was to *flatten* compound diagram structures as far as possible. The object structures built from the composite pattern include `AndCompoundDiagram` objects which have children comprising other `Diagram` objects. In principle, an `AndCompoundDiagram` could have a child which is another `AndCompoundDiagram` object, representing $D_1 \sqcap (D_2 \sqcap D_3)$. But in the implementation such structures are immediately flattened to a single `AndCompoundDiagram` object with the combined children set: $D_1 \sqcap D_2 \sqcap D_3$. As a result, we do not implement associative rules.

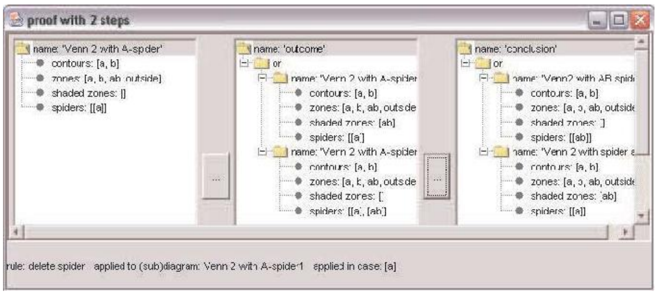
Finally, we consider the collection of children of a compound diagram as an unordered collection. This decision was made because a concrete diagrammatic presentation of a set of children might not show a clear ordering between the components, unlike a textual representation. As a result, we do not implement the commutative rules.



```
name: 'plain Venn 2'
    contours: [a, b]
    zones: [a, b, ab, outsid
    shaded zones: []
    spiders: []
```

```
name: 'plain Venn 2 and Ve
and
    name: 'plain Venn 2
        contours: [a, b]
        zones: [a, b, ab, outside]
        shaded zones: []
        spiders: []
    name: 'Venn2 with shading'
        contours: [a, b]
        zones: [a, b, ab, outside]
        shaded zones: [b, ab]
        spiders: []
```

Fig. 13. Context-sensitive menus.



```
name: 'Venn 2 with A-spider'
    contours: [a, b]
    zones: [a, b, ab, outside]
    shaded zones: []
    spiders: [[a]]
```

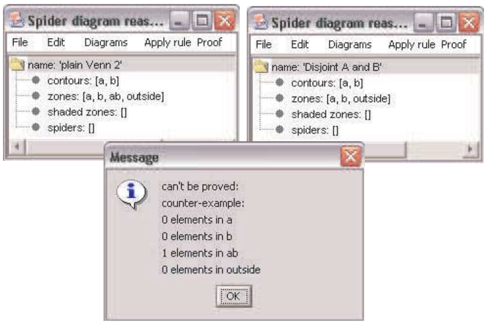
```
name: 'outcome'
or
    name: 'Venn 2 with A-spider
        contours: [a, b]
        zones: [a, b, ab, outside
        shaded zones: [ab]
        spiders: [[a]]
    name: 'Venn 2 with A-spider
        contours: [a, b]
        zones: [a, b, ab, outside
        shaded zones: [
        spiders: [[a], [ab]]
```

```
name: 'conclusion'
or
    name: 'Venn2 with AB spid
        contours: [a, b]
        zones: [a, b, ab, outside
        shaded zones: ]
        spiders: [[ab]]
    name: 'Venn 2 with spider
        contours: [a, b]
        zones: [a, b, ab, outside
        shaded zones: ab]
        spiders: [[a]]
```

```
rule: delete spider    applied to (sub)diagram: Venn 2 with A-spider    applied in case: [a]
```

Fig. 14. An automatically generated proof.

The tool allows the user to build their own proofs, preventing incorrect applications of rules. Incorrect application is prevented by using context sensitive menus on the user interface: only rules that can be applied to the diagram are offered, see Fig. 13. Also, the tool can automatically generate proofs, given a premise and a conclusion. If a proof exists, one is presented to the user, otherwise a counterexample is given (see Figs. 14 and 15). For many examples the automatically generated proofs are not the shortest and users may wish to write a proof themselves. When extending this tool to more expressive, possibly undecidable, diagrammatic languages an automated


Spider diagram reas...
File Edit Diagrams Apply rule Proof
Spider diagram reas...
File Edit Diagrams Apply rule Proof
Message
OK

name: 'plain Venn 2'
contours: [a, b]
zones: [a, b, ab, outside]
shaded zones: []
spiders: []

name: 'Disjoint A and B'
contours: [a, b]
zones: [a, b, outside]
shaded zones: []
spiders: []

can't be proved:
counter-example:
0 elements in a
0 elements in b
1 elements in ab
0 elements in outside

Fig. 15. An automatically generated counterexample.

decision procedure may not exist. Thus it may be even more useful for users to write their own proofs.

6 Conclusion and Further Work

In this paper we have described a tool that supports reasoning with spider diagrams. This tool allows users to construct their own proofs and, at any given proof step, offers only valid rule applications to the user. In addition, given D_1 and D_2, it can construct a proof that D_1 entails D_2 or a counterexample. The automatically generated proofs are significantly shorter than proofs which are generated using an algorithm directly derived from the completeness proof in [8].

Our plan is to extend the work in this paper to the considerably more expressive constraint diagram reasoning system. Ideally, we will be able to find, and implement, an algorithm to construct proofs or counterexamples for constraint diagrams but this is only possible for a decidable system. It is unknown whether constraint diagrams express a decidable fragment of first order predicate logic. It is known that the spider diagram language is equivalent in expressive power to monadic first order logic with equality [14]. Restricted forms of the constraint diagram notation, that include arrows and universal spiders, yield decidable systems [13]. When we extend the notation further, increasing expressiveness, there is a risk that we sacrifice decidability.

A heuristic approach to generate even shorter proofs has been developed for proofs between unitary diagrams [4]. The heuristic algorithm searches for a proof of length less than a given limit. If it fails to find a proof, it could be because more steps are required, or because no proof exists. The algorithm outlined here is essential when the heuristic approach fails to find a proof. In any case, the heuristic algorithm does not search for proofs between compound diagrams.

Currently the output from our tool appears in textual, rather than diagrammatic, form. In order to present proofs to users as a sequence of drawn diagrams we need to create diagrams from their abstract descriptions. In [3] the authors give an algorithm for drawing a class of spider diagrams from abstract descriptions. The quality of the diagram layout has been improved using iterative methods and layout metrics [5]. The work done so far on layout addresses the problem of drawing diagrams without consideration of their context in a proof. More research is required on drawing strategies for proof sequences so that the diagrams appear sufficiently similar after rule application, highlighting changes made by applying the rule.

Our ambition is to develop a suite of applications including a diagram viewer, editor and a proof writing environment, including automated proof writing. Such a tool could be integrated into a UML environment, like a diagrammatic version of the Key project at [17] which builds OCL proof obligations and automated proof generation into the Together Control Center CASE tool.

References

[1] Amati, G. and I. Ounis, *Conceptual graphs and first order logic,* The Computer Journal, 43(1), pages 1–12, 2000.

[2] Fish, A., J. Flower and J. Howse, *A reading algorithm for constraint diagrams,* In Proceedings of IEEE Symposium on Visual Languages and Formal Methods, Auckland, 2003.

[3] Flower, J., and J. Howse, *Generating Euler diagrams,* In Proceedings of Diagrams 2002, pages 61-75, Springer-Verlag, 2002.

[4] Flower, J., J. Masthoff and G. Stapleton, *Generating readable proofs: A heuristic approach to theorem proving with spier diagrams,* Submitted to 3rd International Conference, Diagrams 2004, Cambridge, 2004.

[5] Flower, J., P. Rodgers and P. Mutton, *Layout metrics for Euler diagrams,* In Proceedsings of 7th International Conference on Information Visualisation, pages 272-280, London, 2003.

[6] Gil, J., J. Howse and S. Kent, *Formalising spider diagrams,* In Proceedings of IEEE Symposium on Visual Languages (VL99), pages 130-137, IEEE Computer Society Press, 1999.

[7] Howse, J., F. Molina and J. Taylor, *SD2: A sound and complete diagrammatic reasoning system,* In Proceedings of IEEE Symposium on Visual Languages (VL2000), IEEE Computer Society Press, pages 127-136, 2000.

[8] Howse, J., G. Stapleton and J. Taylor, *Spider diagrams,* In preparation, to appear: www.cmis.brighton.ac.uk/research/vmg.

[9] Niinimäki, M., *Understanding the semantics of conceptual graphs,* University of Tampere, Department Of Computer Science, http://www.cs.uta.fi/reports/pdf/A-1999-4.pdf, 1999.

[10] Online course in knowledge representation using conceptual graphs, http://www.hum.auc.dk/cg/, produced by Humanistic Informatics at Department of Communication, Aalborg University, Denmark, and by the Flexnet Project in Information Science, University of Southern Denmark.

[11] Shin, S.-J., "The logical status of diagrams," Cambridge University Press, 1994.

[12] Sowa, J. F., *Conceptual graphs summary,* In Conceptual Structures: Current Research and Practice, P. Eklund, T. Nagle, J. Nagle, and L. Gerholz, eds., Ellis Horwood, pp. 3-52., 1992.

[13] Stapleton, G., J. Howse and J. Taylor, *A constraint diagram reasoning system,* In Proceedings of 9th International Conference on Distributed Multimedia Systems, International Conference on Visual Languages and Computing, pages 263-270, Miami, 2003.

[14] Stapleton, G., J. Howse, J. Taylor and S. Thompson, *What can spider diagrams say?,* Submitted to 3rd International Conference, Diagrams 2004, Cambridge, 2004.

[15] Swoboda, N., *Implementing Euler/Venn reasoning systems,* In Diagrammatic Representation and Reasoning, Anderson, M., Meyer, B., and Oliver, P., eds, Springer-Verlag, 2001.

[16] Swoboda, N. and G. Allwein, *A case study of the design and implementation of heterogeneous reasoning systems,* In Logical and Computational Aspects of Model-Based Reasoning, Manani, L., and Nersessian, N.J., eds, Kluwer Academic, 2002.

[17] University of Karlsruhe Institute of Logic, Complexity and Deductive Systems *The KeY tool.* http://i12www.ira.uka.de/ key.

[18] Venn, J., On the diagrammatic and mechanical representation of propositions and reasonings. *Phil.Mag*, 1880.

[19] The Visual Modelling Group Website, executable java program available from http://www.cmis.brighton.ac.uk/research/vmg.