Noname manuscript No.
(will be inserted by the editor)

Speedith

A reasoner for spider diagrams

Matej Urbas · Mateja Jamnik · Gem Stapleton

Received: date / Accepted: date

Abstract In this paper, we introduce Speedith which is an interactive diagrammatic theorem prover for the well-known language of spider diagrams. Speedith provides a way to input spider diagrams, transform them via the diagrammatic inference rules, and prove diagrammatic theorems. Speedith's inference rules are sound and complete, extending previous research by including all the classical logic connectives. In addition to being a stand-alone proof system, Speedith is also designed as a program that plugs into existing general purpose theorem provers. This allows for other systems to access diagrammatic reasoning via Speedith, as well as a formal verification of diagrammatic proof steps within standard sentential proof assistants. We describe the general structure of Speedith, the diagrammatic language, the automatic mechanism that draws the diagrams when inference rules are applied on them, and how formal diagrammatic proofs are constructed.

Keywords Interactive Theorem Proving · Diagrammatic Reasoning · Knowledge Representation · Automated Reasoning

1 Introduction

Diagrams are often employed as illustrations in pen-and-paper reasoning. In fact, since ancient times they frequently formed essential parts of proofs.[1]

Matej Urbas
Improbable, London, UK. E-mail: matej.urbas@gmail.com

Mateja Jamnik
University of Cambridge Computer Laboratory, UK. E-mail: mateja.jamnik@cl.cam.ac.uk

Gem Stapleton
University of Brighton, UK. E-mail: G.E.Stapleton@brighton.ac.uk

[1] The use of diagrams as evidence, or as a tool for constructing proofs, predates modern efforts of formalisation of logic. An early example of the use of diagrams in proofs is Euclid's Elements. In the past, diagrams were used in the context of geometry, or geometrical

One can argue that diagrams often provide compelling and intuitive solutions to problems. Despite this, and with the advent of proof theory, the role of diagrams became that of an *informal* visual aid – diagrams have rarely been formalised in proof tools to be used for reasoning. In this paper, we do just that: we present a new, formal diagrammatic theorem prover Speedith.[2] Speedith's domain is the language of spider diagrams. It allows us to apply diagrammatic inference rules on conjectures about spider diagrams, and thus construct a proof. The entire proof construction process is carried out visually. The derived proof is certified to be (logically) correct. The hypotheses that we aim to confirm in our work are:

– It is possible to design and implement a complete formal diagrammatic reasoner in the general domain of monadic first-order logic with equality (or MFOLE for short), expressed using the language of spider diagrams.
– The derived diagrammatic proofs can be guaranteed to be formally correct.
– A diagrammatic reasoner for spider diagrams can be standalone, yet also pluggable into external proof tools – thus providing alternative problem representations and proof construction methods for these tools.

The intuitive nature of diagrams recently motivated the design of some formal diagrammatic reasoning systems. Some examples include DIAMOND [13], Dr.Doodle [36], and Cinderella [17], but they target different, more restricted domains (e.g., a small subset of natural number arithmetic, a subset of real arithmetic), and are hence able to prove only a limited class and number of theorems. They do not provide a provably sound and complete set of inference rules. They are also not designed to be readily integrated into external proof tools.

There are theorem provers that were developed for spider diagrams, but they worked only for fragments of the logic in this paper: they did not include any logical connectives, or only a limited number of them [25]. In Speedith we formalise the whole spider diagram logic (Section 2), which includes the full range of classical logical connectives and is expressively equivalent to MFOLE. We also develop a set of sound and complete inference rules (Section 3), representing an extension of the system in [12].[3] Moreover, we argue that these inference rules allow the user to construct more intuitive proof steps.

Speedith is an interactive proof assistant for the language of spider diagrams that allows its users to interactively apply diagrammatic (visual) inference rules on spider-diagrammatic statements. It checks whether the inference rules are used correctly and verifies that a spider-diagrammatic statement expresses a true fact – it is a theorem. Thus, Speedith's diagrammatic proofs are

representations of concepts from algebra, number theory, analysis, topology and category theory.

[2] We first introduced Speedith in [33]. This paper gives a comprehensive account of Speedith, its design and theoretical properties, and also its further developments regarding the selection of inference rules, hand-drawing interface and pluggable infrastructure.

[3] The system in [12] is a proper fragment of that implemented in Speedith and was proved complete in the absence of $\longrightarrow$, $\longleftrightarrow$ and $\neg$. We enlarge the set of inference rules to obtain completeness in this syntactically richer logic.

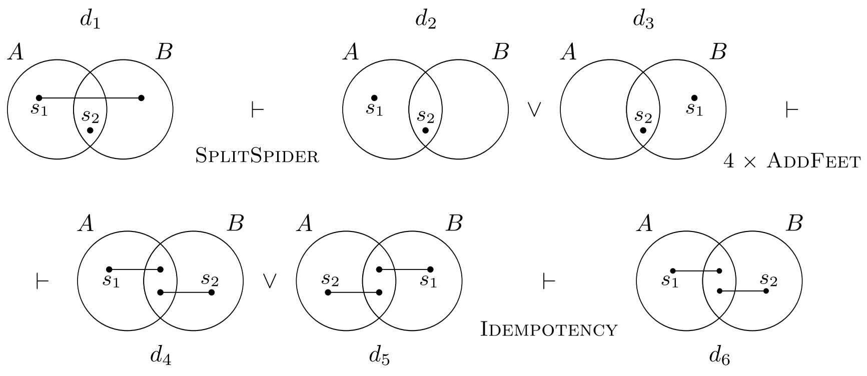


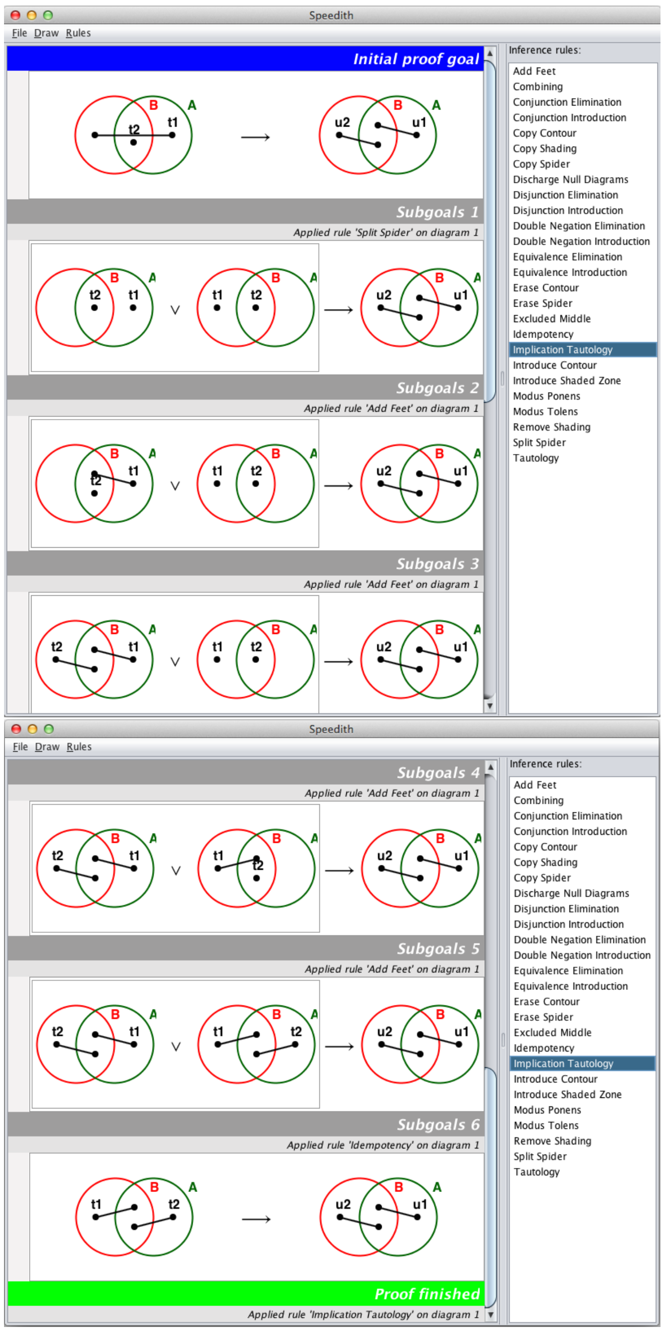
Fig. 1 A proof of a spider-diagrammatic statement. The proof establishes that given sets A and B, if there are two elements s_1 and s_2 and one is in both of A and B and the other is either in only A or only B, then we can deduce that one element is in A and the other is in B. In this proof, we applied the split spider on s_1, add feet, one to each of the four spiders, and idempotency inference rules. The rules are proved to be sound and their application in this proof is verified by Speedith to be correct. Hence, the proof is certified to be correct.

entirely formal and certified to be correct. Fig. 1 shows an example of Speedith's purely diagrammatic proof. Here, d_1 is a spider diagram which conveys some information about the relationships between two elements and two sets and proves that d_6 follows logically. In Section 4 we present the architecture of Speedith in detail, including a reasoning kernel that manages the state of the proofs, controls how inference steps are applied, and manages the communication with external general purpose theorem provers.

Speedith provides a graphical user interface through which all the diagrammatic proofs are constructed – we describe this in detail in Section 4.4. The user can input the theorem via hand-drawn diagrams or via a textual abstract representation of diagrams. Speedith visually displays spider-diagrammatic statements; allows the user to specify which inference rules should be applied on what parts of the spider diagram; and displays the result of this visually. Fig. 2 shows a screenshot of the proof presented above in Fig. 1 as it is constructed in Speedith.

Whilst Speedith is a standalone diagrammatic proof assistant, it is also designed to easily plug into external proof tools. This has the advantage that spider-diagrammatic proofs can be reconstructed in traditional logic, and thus certified with, for example, LCF-style general purpose theorem provers [8].

We evaluate Speedith in Section 5 by comparing it to other related work, assessing its generality and extensibility, and pointing out its limitations that indicate future directions. Finally, in Section 6 we conclude with some general observations.



Initial proof goal

Subgoals 1

Applied rule 'Split Spider' on diagram 1

Subgoals 2

Applied rule 'Add Feet' on diagram 1

Subgoals 3

Applied rule 'Add Feet' on diagram 1

Subgoals 4

Applied rule 'Add Feet' on diagram 1

Subgoals 5

Applied rule 'Add Feet' on diagram 1

Subgoals 6

Applied rule 'Idempotency' on diagram 1

Proof finished

Applied rule 'Implication Tautology' on diagram 1

Fig. 2 A screenshot of the proof from Fig. 1 constructed in Speedith - due to scrolling there are two screens.

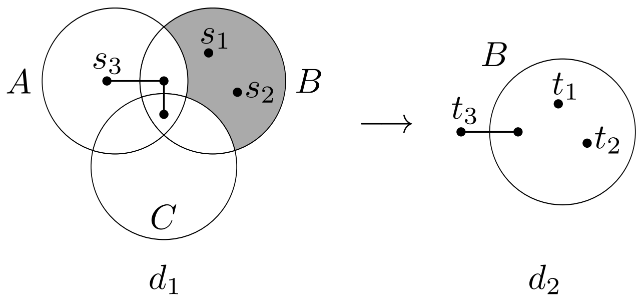
2 Spider diagrams

Spider diagrams have a formally defined syntax and semantics. The language of spider diagrams resembles Venn and Euler diagrams. It uses closed curves to denote sets, shading of areas to denote upper bounds on the cardinality of sets, and dots connected with lines to denote existentially quantified elements. The syntax and semantics of this language have been formally defined by Howse et al. [12]. The language of spider diagrams is also accompanied by inference rules, which results in the logic of spider diagrams. This logic is expressively equivalent to monadic first-order logic with equality (MFOLE) [23]. We also developed a new set of sound inference rules, which represent an extensions of the system in [12] and prove them to be complete – for details, see Section 3.

We firstly introduce the language and logic of spider diagrams, that is, its syntax and semantics (Section 2), and inference rules (Section 3). After, we introduce Speedith itself (Section 4).

2.1 Syntax

Sentences in the language of spider diagrams are capable of expressing assertions about sets and their elements. Fig. 3 contains an example spider-diagrammatic sentence, which is also a theorem.



$$\begin{aligned}&\exists\, s_1, s_2, s_3.\ (\text{distinct}\,[s_1, s_2, s_3] \wedge s_3 \in (A \setminus C) \cup (A \cap B) \wedge s_1 \in B \setminus (A \cup C)\ \wedge \\ &\wedge\ s_2 \in B \setminus (A \cup C) \wedge B \setminus (A \cup C) \subseteq \{s_1, s_2\}) \longrightarrow \\ &\exists\, t_1, t_2, t_3.\ (\text{distinct}\,[t_1, t_2, t_3] \wedge t_1 \in B \wedge t_2 \in B)\end{aligned}$$

Fig. 3 Top: An assertion in the language of spider diagrams. **Bottom**: an equivalent assertion as a sentential formula in MFOLE.

Spider diagrams use labelled closed curves to represent named sets. These curves are called *contours*. Their spatial and topological arrangement is used to assert relationships between the sets they represent. For instance, the enclosure of one contour by another corresponds to a subset (and consequently a superset) relationship between the represented sets. Contours are annotated with *labels* (in Fig. 3, the contour labels are A, B and C). The set of contour labels used in a diagram d is denoted by $L(d)$ (this set may also be empty).

All spider diagrams contain at least one *zone*. A zone is a region that is inside some or none of the contours. Formally, a zone is a pair of finite, disjoint sets of contour labels, $(in,\ out)$. Intuitively, $(in,\ out)$ is inside every contour of in, and outside every contour of out. So, in a diagram, the set of possible zones is formed by its contour labels. For example, in d_1 of Fig. 3, the zones are $(\emptyset, \{A, B, C\})$, $(\{A\},\{B, C\})$, $(\{B\}, \{A, C\})$, $(\{C\}, \{A, B\})$, $(\{A, B\}, \{C\})$, $(\{A, C\}, \{B\})$, $(\{B, C\}, \{A\})$, $(\{A, B, C\}, \emptyset)$. We denote the set of zones in a diagram d by $Z(d)$. Any collection of zones is called a *region*.

Spiders denote the existence of elements within a region. Spiders are connected acyclic graphs with at least one node. Each node of a spider, called a *spider foot*, resides in a distinct and unique zone. The nodes are visually connected with lines, called *spider legs*. The collection of zones, that is, the region in which all of a spider's nodes reside is called the *spider's habitat*.[4] In summary, a spider asserts that there exists an element in the set denoted by its habitat. Furthermore, spiders represent distinct elements. Consequently spiders place a lower bound on the cardinality of the set represented by the spider's habitat. The upper bound on the cardinality of a set is expressed with *shaded zones*. The set of shaded zones is denoted with $ShZ(d)$. The set of shaded zones is a subset of the set $Z(d)$. In a shaded zone, all elements are represented by spiders.

The set of spiders in a diagram d is denoted by $S(d)$. A spider's habitat is returned by the following function:

$$\eta_d : S(d) \to \mathcal{P}(Z(d)) \setminus \{\emptyset\}.$$

The range of the above function excludes the empty set – this reflects the fact that spiders cannot have empty habitats.[5] For example, Fig. 3 contains 6 spiders (denoted with indexed letters s and t). The spider s_3 has three feet and two legs. It's habitat consists of three zones: $(\{A\},\ \{B,\ C\})$, $(\{A,\ B\},\ \{C\})$ and $(\{A,\ C,\ B\},\ \emptyset)$.

A diagram consisting of only the above elements is called a *unitary spider diagram* (we typically use the symbol d_u to denote them). Unitary spider diagrams are atomic expressions in the language of spider diagrams.

Definition 1 A *unitary spider diagram* is an atomic element of the language of spider diagrams. It is defined as a tuple of the following form:

$$d_u = (L,\ Z,\ ShZ,\ S,\ \eta), \tag{1}$$

[4] Note that we use labels on spider feet in order to be able to refer to specific spiders. However, as can be observed from the definition of spiders above, these labels do not form part of the syntax of spider diagrams. They are a convenience, and can be arbitrarily and freshly chosen every time an inference rule is applied on a diagram. This means that, for example, two drawn spider diagrams that are identical apart from the spider labels have the same syntax.

[5] A spider with an empty habitat is a contradiction, as it would imply that there exists an element that does not belong to any set.

where L is the set of contour labels, Z is the set of zones in the diagram,[6] ShZ is the set of shaded zones (a subset of Z), S is the set of spiders and $\eta : S(d_u) \to \mathcal{P}(Z(d_u)) \setminus \{\emptyset\}$ is a function that returns the habitat of each spider.

Given a unitary spider diagram, d_u, we will write $L(d_u)$, $Z(d_u)$, $ShZ(d_u)$, $S(d_u)$ and η_{d_u} for L, Z, ShZ, S and η where necessary (e.g., when talking about more than one unitary spider diagram). A sentence in the language of spider diagrams may be a unitary spider diagram or a *compound spider diagram*. Compound spider diagrams connect multiple unitary spider diagrams through logical connectives.

Definition 2 A sentence d in the language of spider diagrams may be either a compound spider diagram or a unitary spider diagram. We define a *general spider diagram* as a compound diagrams with recursive nesting of unitary or other compound spider diagrams:

$$d =:: \begin{cases} d \longleftrightarrow d & \text{Logical equivalence} \\ d \longrightarrow d & \text{Implication} \\ d \vee d & \text{Disjunction} \\ d \wedge d & \text{Conjunction} \\ \neg d & \text{Negation} \\ d_u & \text{Unitary spider diagram} \end{cases} \tag{2}$$

Note that the connectives $\longleftrightarrow$, $\longrightarrow$, and $\neg$ were excluded from the sound and complete spider diagram logic studied by Stapleton et al. [23].

In Fig. 3, there are two unitary diagrams d_1 and d_2 which are connected with the implication operator $\longrightarrow$ into a compound spider diagram.

2.2 Semantics

We define the semantics of spider diagrams by *interpretations* and the *interpretation* tuple:

$$I = (U, \Phi), \tag{3}$$

where U is the universal set (containing all elements of a particular interpretation), and Φ is the function that maps contour labels to subsets of U:

Definition 3 Let C be a contour with label l in a spider diagram d (i.e., there exists a unitary spider diagram d_u within d such that $l \in L(d_u)$). Then, for a specific interpretation tuple (U, Φ), the function Φ maps the contour label l to a subset of U:

$$\Phi(l) \subseteq U. \tag{4}$$

[6] So, each zone (in, out) in Z ensures that $in \cup out = L$.

In addition to the interpretation tuple we also define a spider map function Σ. This function is analogous to the valuation function in Alfred Tarski's definition of formal semantic for first-order logic [30].

Definition 4 The *spider map function* $\Sigma_{d_u,U}$ maps spiders from a unitary spider diagram into the universal set U. Let $s \in S(d_u)$ be a spider living in the unitary spider diagram d_u, and U the universal set of a particular interpretation, then $\Sigma_{d_u,U}$ is defined by:

$$\Sigma_{d_u,U}(s) = x; \text{ where } x \in U, \tag{5}$$

for which we use the shorthand $\Sigma(s)$, where the universal set U and the unitary spider diagram d_u are implicitly given and understood from the context. Note that Σ is parametrised by both d_u and U, therefore $\Sigma_{d_u,U}$ may differ for different d_u and U.

2.2.1 Truth in spider diagrams

A sentence in the language of spider diagrams is an assertion which may or may not hold under a particular interpretation I and a particular spider map function Σ.

In order to formally define truth in the language of spider diagrams we firstly define the interpretation functions for zones and spider habitats. These are required to define the truth of a unitary spider diagram, which in turn is required in the definition of the truth of a compound sentence in the language of spider diagrams.

Definition 5 Let $z = (in, out)$ be a zone in the unitary spider diagram d_u, that is, $z \in Z(d_u)$, and let $I = (U, \Phi)$ be a particular interpretation. The set represented by the zone z is then defined as follows:

$$\zeta_I(z) \overset{\text{def}}{=} \left[\bigcap_{l \in in} \Phi(l) \right] \cap \left[\bigcap_{l \in out} U \setminus \Phi(l) \right]. \tag{6}$$

The interpretation of a region, or a spider's habitat, is the union of all zones within the region:

Definition 6 Let $h = \eta_{d_u}(s)$, where $h \subseteq \mathcal{P}(Z(d_u))$, be the habitat of the spider s in the unitary spider diagram d_u, and let $I = (U, \Phi)$ be a particular interpretation. Then the set represented by the habitat h is defined as follows:

$$\chi_I(h) \overset{\text{def}}{=} \bigcup_{z \in h} \zeta_I(z). \tag{7}$$

Unlike zones and regions, unitary spider diagrams represent assertions of truth. Therefore, the interpretation function of unitary spider diagrams maps to either the truth or falsehood:

Definition 7 We use the notation $\vDash_{I,\,\Sigma} d_u$ to denote that a unitary spider diagram $d_u = (L,\, Z,\, \mathit{ShZ},\, S,\, \eta)$ is true under the interpretation $I = (U, \Phi)$ and spider mapping Σ. Furthermore, let $S = \{s_1, s_2, \ldots, s_n\}$ be the set of all spiders in d_u.

We say that $\vDash_{I,\,\Sigma} d_u$ holds if $x_i = \Sigma(s_i)$, for all $i \in \{1, \ldots, n\}$, such that:

- distinct spiders map to distinct elements: $j \neq k \longrightarrow x_j \neq x_k$,
- spiders live in their respective habitats: $x_i \in \chi_I(\eta\,(s_i))$,
- the shaded zones form a subset of the spider elements: $\bigcup_{z \in \mathit{ShZ}} \zeta_I(z) \subseteq \{x_1, \ldots, x_n\}$, and
- the missing zones denote empty sets: $\forall z \in \mathit{MZ}(d_u).\zeta_I(z) = \emptyset$,

where missing zones $\mathit{MZ}(d_u)$ are the ones that are not in $Z(d_u)$ but may still be expressed with the labels in $L(d_u)$. In particular, the set of MZ is defined as follows:

$$\mathit{MZ}(d_u) = \{z \mid z = (\mathit{in},\, L(d_u) \setminus \mathit{in}) \wedge \mathit{in} \subseteq L(d_u) \wedge z \notin Z(d_u)\}\,.$$

Fig. 4 shows a unitary spider diagram with one missing zone. Intuitively, the disjoint spatial positioning of contours A and B indicates that the intersection of sets A and B is empty. Missing zones thus denote empty sets and are the result of spatially disjoint positioning of contours.

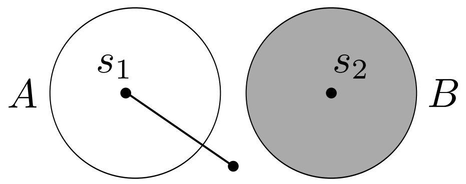


Fig. 4 A unitary spider diagram with the missing zone $(\{A, B\}, \emptyset)$.

Finally, the definition of truth for all sentences of the language of spider diagrams is given:

Definition 8 We use the notation $\vDash_I d$ to denote that the spider-diagrammatic sentence is true under the interpretation I. We define $\vDash_I d$ recursively by cases of the spider diagrammatic syntax (as defined in Definition 2):

- $\vDash_I d_1 \longleftrightarrow d_2$ is true iff $\vDash_I d_1$ and $\vDash_I d_2$ both are true or both are false.
- $\vDash_I d_1 \longrightarrow d_2$ is true iff $\vDash_I d_1$ is false or $\vDash_I d_2$ is true.
- $\vDash_I d_1 \vee d_2$ is true iff $\vDash_I d_1$ or $\vDash_I d_2$ are true.
- $\vDash_I d_1 \wedge d_2$ is true iff $\vDash_I d_1$ and $\vDash_I d_2$ are true.
- $\vDash_I \neg d_1$ is true iff $\vDash_I d_1$ is false.
- $\vDash_I d_u$ is true iff there exists a spider map Σ such that $\vDash_{I,\,\Sigma} d_u$ is true (as per Definition 7).

So far, we defined the truth of spider-diagrammatic sentences under a specific interpretation $I = (U, \Phi)$ and the existence (or otherwise) of appropriate spider mapping functions. Next, we define what it means for a spider-diagrammatic sentence to be a theorem:

Definition 9 A spider diagram d is a theorem if $\vDash_I d$ is true under all interpretations I. We use the following notation to denote that the spider diagram d is a theorem:

$$\vDash d \tag{8}$$

Definition 10 We say that diagram d' *logically entails* diagram d, if and only if the following holds:

$$\vDash d' \longrightarrow d. \tag{9}$$

To denote this, we use shorthand notation:

$$d' \vDash d. \tag{10}$$

Similarly, we say that d' and d are equivalent if they entail each other:

Definition 11 We say that diagram d is *logically equivalent to* diagram d', if and only if both $d \vDash d'$ and $d' \vDash d$ hold. To denote this, we use shorthand:

$$d' \equiv d. \tag{11}$$

3 Inference rules

Similarly to other traditional logical systems, spider diagrams are also equipped with inference rules. The central role of inference rules is to enable stepwise verification of the validity of a spider-diagrammatic sentence, that is, to determine whether a spider diagram is a theorem.

The inference rules in spider diagrams are of three basic types:

1. ***inference rules for logical connectives***: this category contains inference rules of propositional logic (which act purely on logical connectives of compound spider diagrams),
2. ***purely diagrammatic inference rules***: rules of this type transform unitary spider diagrams into logically entailed spider diagrams,
3. ***compound inference rules***: these rules act both on unitary spider diagrams and compound spider diagrams. Compound inference rules act on both the purely diagrammatic aspects of spider diagrams as well as the symbolic logical connectives that bind them.

The inference rules we present here extend those in [12] in three ways, motivated by the desire for completeness and for more intuitive, elegant proofs. With regard to completeness, all of the inference rules implemented in Speedith for the logical connectives $\longrightarrow$, $\longleftrightarrow$ and $\neg$ are new, as Howse et al. [12] did not include these connectives. We also introduce a further diagrammatic rule that is necessary for completeness: NegationElimination. This rule allows negation to be entirely eliminated from diagrams. With regard to intuitive and elegant proofs, we introduce new diagrammatic rules that allow shorter and more natural proofs to be constructed. These new rules are called CopyContours, CopyShading and CopySpider, and are introduced below. Each operates on two unitary diagrams joined by $\wedge$, copying information from one diagram into the other. Previously, in [12], information typically had to be copied to reduce the diagrams into a particular normal form, followed by applying the Combining rule (see below). The normal form would then need to be transformed back into the conjunction of the two modified initial diagrams. Clearly, this is a long, unnecessary and indirect process that reduces clarity of a proof – which is the reason that we introduced our new inference rules.

3.1 Inference rules for logical connectives

Rules for logical connectives in compound spider diagrams are based on the typical inference rules for propositional logic:

1. double negation elimination and introduction,
2. conjunction elimination and introduction,
3. disjunction elimination and introduction,
4. biconditional introduction and elimination,
5. modus ponens,
6. modus tollens,
7. tautologies and simplification rules, for example, $d \longrightarrow d \simeq \top$, $d \vee \neg d \simeq \top$, $d \wedge \neg d \simeq \bot$, and $d \wedge \bot \simeq \bot$; and
8. idempotency rules like $d \vee d \simeq d$ and $d \wedge d \simeq d$.

3.2 Purely diagrammatic inference rules

Purely diagrammatic inference rules transform unitary spider diagrams exclusively. We outline them here and provide some instances of their application on concrete spider diagrams. For more detail and their formal definitions, see Howse et al [12].

Add spider feet: Let $d = (L, Z, ShZ, S, \eta)$ be a unitary spider diagram, $s \in S$ be a spider with the habitat $\eta(s)$, and h be a region such that $h \subseteq \{z \mid z \in Z \wedge z \notin \eta(s)\}$. Then the AddFeet rule is applicable on d and produces a new unitary spider diagram d' such that $d \vDash d'$ and $d' = (L, Z, ShZ, S, \eta')$ where:

$$\eta'(x) = \begin{cases} \eta(x) & x \neq s, \\ \eta(x) \cup h & x = s. \end{cases}$$

This inference rule is not information-preserving (it may not be applied in the other direction). Fig. 5 shows an example application of the AddFeet inference rule.

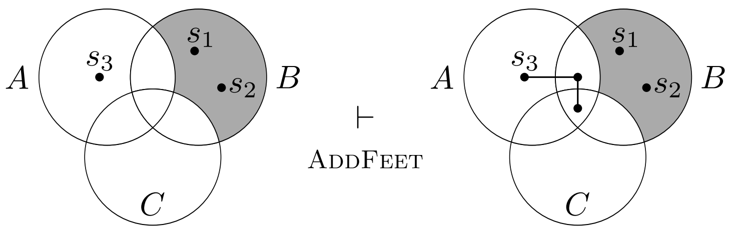


Fig. 5 An example application of the AddFeet inference rule. Two feet are added to the spider s_3. The new feet are inserted into the zones $(\{A, B\}, \{C\})$ and $(\{A, B, C\}, \emptyset)$.

Introduce a contour label: adds an additional contour (with a fresh label) to a unitary spider diagram d resulting in a new diagram d'. This rule introduces new zones, shaded zones and spider feet into d'. In particular, each zone (shaded or otherwise) is split into two (one is outside of the new contour and the other is within). Additionally, every spider is extended with new feet in the new zones that are the result of split zones within the spiders' original habitat. This inference rule may be applied in both directions, as both $d \vDash d'$ as well as $d' \vDash d$ hold. Fig. 6 shows an example application of IntroContour inference rule.[7]

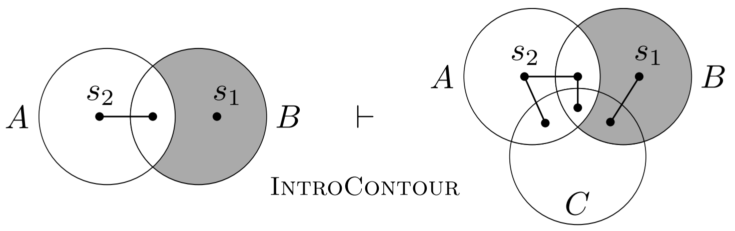
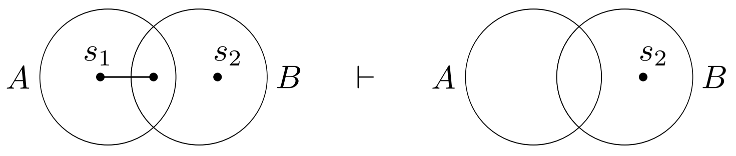


Fig. 6 An example application of the IntroContour inference rule. The contour C is introduced to the unitary spider diagram on the left-hand side.

Erasure of a spider: Let $d = (L, Z, ShZ, S, \eta)$ be a unitary spider diagram. A spider $s \in S$ with a habitat consisting of exclusively non-shaded zones (i.e., $\eta(s) \subseteq Z \setminus ShZ$) may be completely removed from the unitary spider diagram d. The result is a new unitary spider diagram d' which is an exact copy of the diagram d except that d' is missing the spider s and η' is undefined for this spider. The resulting diagram is thus: $d' = (L, Z, ShZ, S \setminus \{s\}, \eta')$. Fig. 7 is an example application of EraseSpider. This inference rule does not preserve information.

Erasure of a contour label: A contour label $l \in L$ may be removed from a unitary spider diagram $d = (L, Z, ShZ, S, \eta)$. This results in a new diagram with modified shading of zones and spider habitats $d' = (L \setminus \{l\}, Z', ShZ', S, \eta')$.

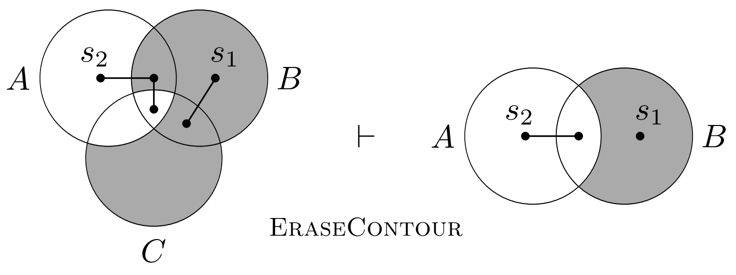
[7] Introducing a contour in abstract syntax is straightforward. However, drawing an additional contour may be more complex, for example, it may not be drawable as a single circle. We use iCircle algorithm for laying out spider diagrams – for details, see 4.4.2.



ERASESPIDER

Fig. 7 An example application of the ERASESPIDER inference rule. The spider s_1 is removed from the left-hand unitary spider diagram.

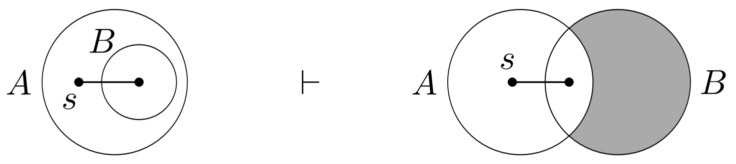
Zones $z_i = (\{l\} \cup in,\ out)$ and $z_o = (in,\ \{l\} \cup out)$, where at most one of them is shaded, collapse into non-shaded zones $z = (in,\ out)$. Otherwise, shading is preserved. Additionally, if a spider s has at least one foot in zones that collapse into one, the spider will have a foot in the collapsed zone in the new diagram. This rule is not an equivalence rule. Fig. 8 shows an example application of the ERASECONTOUR.



ERASECONTOUR

Fig. 8 An example application of the ERASECONTOUR inference rule. The contour label C is removed from the left-hand unitary spider diagram.

Introduce a shaded zone: If a zone z is missing from a unitary spider diagram $d = (L,\ Z,\ ShZ,\ S,\ \eta)$, that is, if $z \in MZ(d)$, then d can be replaced by $d' = (L,\ Z \cup \{z\},\ ShZ \cup \{z\},\ S,\ \eta)$. Diagrams d and d' are semantically equivalent, that is, $d \equiv d'$. This rule can thus be applied in both directions (i.e., d can be replaced by d' and vice versa). Fig. 9 shows an example application of INTROSHADEDZONE.



INTROSHADEDZONE

Fig. 9 An example application of the INTROSHADEDZONE inference rule. The shaded zone $(\{B\}, \{A\})$ is introduced into the right-hand unitary spider diagram.

Remove shading: Any region consisting of exclusively shaded zones, say $r \subseteq \mathit{ShZ}$, in the unitary spider diagram $d = (L, Z, \mathit{ShZ}, S, \eta)$ may be converted into a region consisting of exclusively non-shaded zones. This results in a new unitary spider diagram $d' = (L, Z, \mathit{ShZ} \setminus r, S, \eta)$. This rule is a weakening rule as it does not preserve information. Fig. 10 illustrates the application of REMOVE-SHADING with a concrete example.

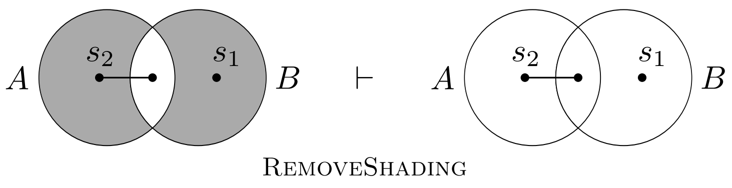


Fig. 10 An example application of the REMOVESHADING inference rule. Shading in the zone $(\{B\}, \{A\})$ in the unitary spider diagram on the left-hand side is removed.

3.3 Compound inference rules

Compound inference rules transform unitary spider diagrams that are connected through a logical connective in a compound diagram. We enumerate the following compound inference rules (others can be found in Howse et al. [12] and Urbas et al. [33]):

Splitting spiders: If a unitary diagram d contains a spider s with multiple feet, then the SPLITSPIDER rule can be applied to that spider. This rule takes as an argument a region r, which is a proper subset of the habitat of spider s: $r \subset \eta_d(s)$ and $|r| \geq 1$. The result of the application of this rule is two disjunctively connected unitary diagrams d_l and d_r that are identical to d except that the habitat of the spider s in diagram d_l equals $\eta_d(s) \setminus r$ and in d_r it equals r. This rule preserves information and is an equivalence rule. Fig. 11 shows an instance of the application of SPLITSPIDER.

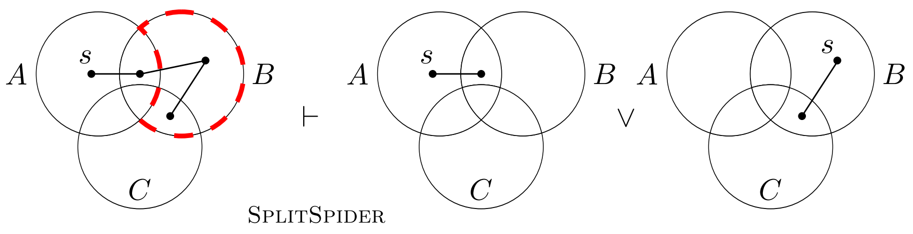


Fig. 11 An example application of the SPLITSPIDER inference rule. The SPLITSPIDER rule is applied to the spider s, which is split in the region marked with a dashed red outline.

Excluded middle: A unitary spider diagram d containing a non-shaded region r can be replaced by $d_1 \vee d_2$ where d_1 and d_2 differ from d only in region r being shaded in d_1 and region r containing an extra spider in d_2. The diagram d and $d_1 \vee d_2$ are semantically equivalent, therefore this rule can be applied in both directions (i.e., $d_1 \vee d_2$ may also be replaced by d). Fig. 12 illustrates the application of this rule.

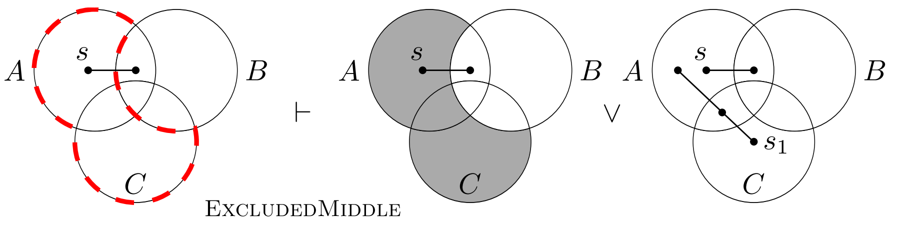


Fig. 12 An example application of the ExcludedMiddle inference rule. The ExcludedMiddle rule is applied to the region marked with a dashed red outline in the left-most unitary spider diagram.

Combining: This rule combines conjunctively connected unitary spider diagrams into a single unitary diagram. Combining is applicable under complex assumptions. For example, two conjunctively connected unitary diagrams may form a contradiction. A contradiction, however, cannot be expressed in a single unitary spider diagram. Therefore, in order to return a unitary diagram, combining needs to be carried out on two conjunctively connected unitary spider diagrams that do not contain conflicting information. Otherwise, the rule will return $\perp$.[8]

More specifically, the Combining rule can be performed on unitary diagrams that have the same sets of zones (and therefore missing zones) and all their spiders have single-zone habitats. The diagrams are non-contradictory iff no shaded zone has fewer spiders than its counterpart in the other diagram. In the non-contradictory case, Combining creates a new unitary spider diagram with the same set of zones as the two original unitary spider diagrams, but with shading in all zones that were shaded in at least one of the original diagrams. Also, spiders of a particular zone are copied from the zone of the original unitary diagram which contains the largest number of spiders. Otherwise, we are in the contradictory case, so there is a shaded zone in one diagram that contains more spiders in the other diagram and the rule returns $\perp$. Fig. 13 shows an example application of this rule in the non-contradictory case.

We now present four new inference rules that were not included in [12]. The formalisations for all four rules, and their proofs of soundness can be found in Appendix B. Firstly, we add a rule that allows the elimination of negation from spider diagrams.

[8] In Speedith, $\perp$ is equivalently represented by $\neg\top$.

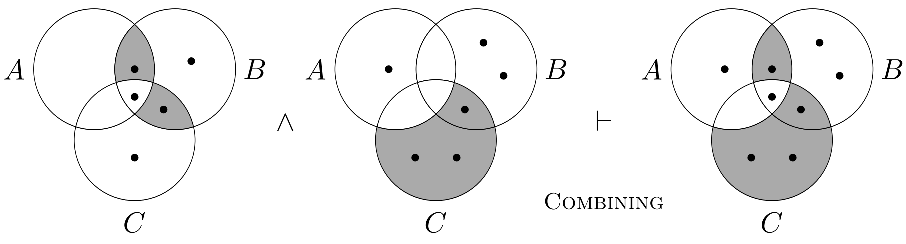


Fig. 13 An example application of the Combining inference rule.

Negation Elimination: The NegationElimination rule may be applied to a negated unitary diagram, $\neg d_n$, where d_n has no missing zones, contains one zone, z_n, with n spiders placed entirely within z_n, no other spiders and if shading is present then it also occurs only within z_n. Such a diagram asserts that there are at least n elements in the set represented by z_n and, should z_n be shaded, that there are no more elements. The rule creates n copies of d_n, giving diagrams d_0,..., d_{n-1}, where the zone z_n contains exactly i spiders in d_i along with shading. If z_n is shaded in d_n then a further copy of d_n is created, say d_{n+1}, where z_n contains $n+1$ spiders but no shading. The result of the rule NegationElimination is a disjunction[9] of unitary diagrams, $d_0 \vee d_1 \vee ... \vee d_{n-1}$ when z_n is not shaded in d_n, otherwise $d_0 \vee d_1 \vee ... \vee d_{n-1} \vee d_{n+1}$. This rule is a logical equivalence. Fig. 14 illustrates it with an example application.

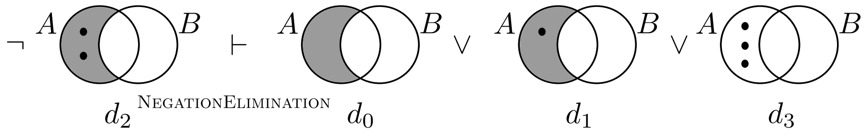


Fig. 14 An example application of the NegationElimination inference rule. In this example, $\neg d_2$ asserts that there are not exactly two elements in $(\{A\}, \{B\})$. This is equivalent to asserting that there are exactly 0, exactly 1 or at least three elements in $(\{A\}, \{B\})$, as seen in $d_0 \vee d_1 \vee d_3$.

Copy contours:[10] The CopyContours rule may be applied to a compound diagram of two conjunctively connected unitary diagrams $d_1 \wedge d_2$ with differing sets of contour labels. A contour label l that is present in only one unitary diagram, say d_1, may be added into the other, say d_2.

The result of this inference rule is a new compound diagram of two conjunctively connected unitary diagrams, $d_1 \wedge d_2'$. The unitary diagram d_2' is a modified version of d_2 with the additional label l, new zones and extended spider habitats. Fig. 15 illustrates this rule with an example application.

[9] Note that we assume an empty disjunction is $\perp$.

[10] This rule has been added in [33] and is not part of the original specification of spider diagrams in [12].

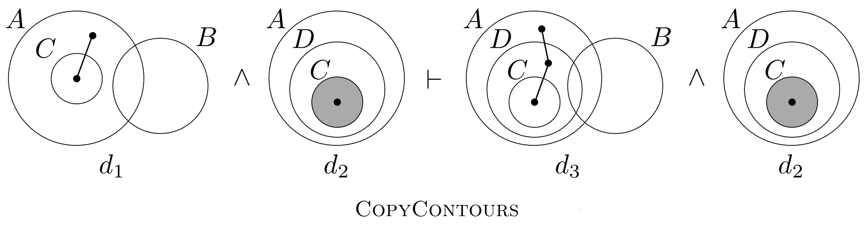


Fig. 15 An example application of the CopyContours inference rule. In this example, the contour D is being copied from the unitary diagram d_2 into the unitary diagram d_1 to produce d_3.

Copy shading:[10] The rule CopyShading may be applied to compound spider diagrams of the form $d_1 \wedge d_2$, where d_1 and d_2 are unitary diagrams. The unitary diagrams must contain regions r_1 and r_2 respectively, which must represent the same set.[11] One of the regions, say r_1, must be entirely shaded while the other must contain at least one non-shaded zone. In addition, diagrams d_1 and d_2 must share the same spiders in these two regions, all of which must have habitats that represent the same set.

The result of the application of the CopyShading rule on $d_1 \wedge d_2$ is the logically equivalent $d_1 \wedge d_2'$, where d_2' is an exact copy of d_2 except its region r_2 is entirely shaded. Fig. 16 contains an example application of this inference rule.

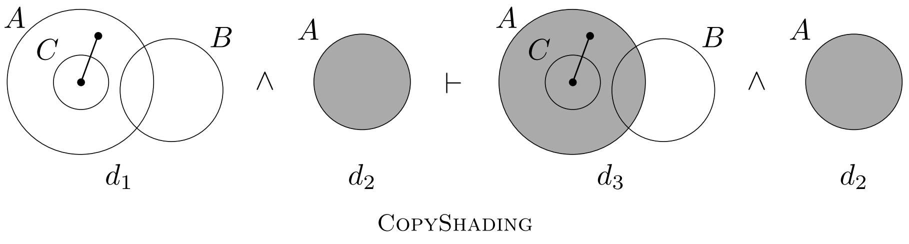


Fig. 16 An example application of the CopyShading inference rule. The shading in the region A is copied from d_2 to d_1 to produce d_3.

Copy a spider:[10] The rule CopySpider may be applied to $d_1 \wedge d_2$ if the unitary diagrams d_1 and d_2 respectively contain regions r_1 and r_2 representing the same set, however, r_1 contains no shaded zones. In addition, all spiders that have a foot in region r_1 must also be present in d_2 with habitats that represent the same set. Let there be a spider s which lives in r_2, but is not present in d_1. Then, $d_1' \wedge d_2$ is the result of the application of this rule on $d_1 \wedge d_2$ where

[11] Regions in two unitary spider diagrams that represent the same set are called *corresponding regions*. Corresponding regions can be identified syntactically and are therefore suitable as a proof-theoretic tool for defining inference rules. This was first seen in Howse et al. [11], where that work is generalized in Appendix A so that corresponding regions can be used for the formalization of inference rules.

d_1' now contains the new spider s in the region r_1. The compound diagram $d_1' \wedge d_2$ is logically equivalent to $d_1 \wedge d_2$. Fig. 17 illustrates the application of CopySpider on a concrete example.

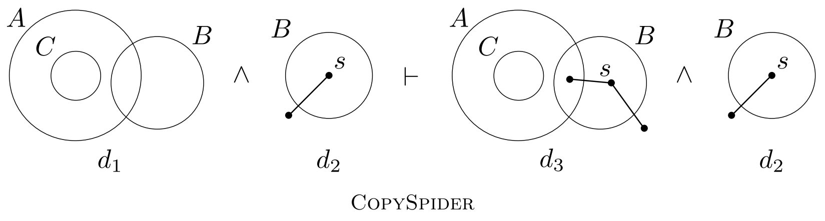


Fig. 17 An example application of the CopySpider inference rule where spider s is copied from d_2 to d_1 to produce d_3.

The above inference rules are used in spider-diagrammatic proofs such as the two examples in Fig. 1 on page 3 and Fig. 18. These two proofs demonstrate the use of all three different types of inference rules within a single proof: diagrammatic inference rules (AddFeet and RemoveContour), compound inference rules (SplitSpider, CopyContours, CopySpider and CopyShading), and inference rules for logical connectives (Idempotency and ConjunctionElimination).

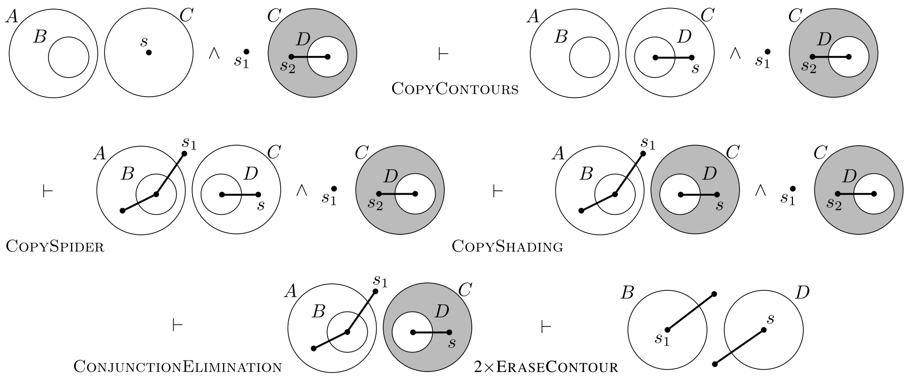


Fig. 18 A spider-diagrammatic proof employing inference rules that copy information from the unitary spider diagram to the right of the conjunction to the unitary spider diagram to the left of the conjunction. First, contour D is copied, followed by copying spider s_1. Next the shading in region $(\{C\}, \{D\})$ is copied over. This makes the right conjunct redundant, so it can be eliminated. Finally, obsolete contours A and C can be erased.

3.4 Properties

We now establish two desirable properties of the spider diagram logic. First, all our inference rules are sound:

Theorem 1 *The spider diagram logic is sound.*

Proof The proof relies on the individual inference rules being sound. Appendix B contains soundness proofs for the individual rules. Since proofs are constructed by repeated application of the inference rules, the logic is sound.

Second, we can establish that the spider diagram logic which we have extended to include $\longrightarrow$, $\longleftrightarrow$ and $\neg$ is complete:

Theorem 2 *The spider diagram logic is complete.*

Proof The proof is given in Appendix C.

4 Architecture and implementation

Speedith is our stand-alone interactive diagrammatic theorem prover for the logic of spider diagrams [33]. Moreover, it was also designed to be easily pluggable into other proof-assisting software. For example, statements and proofs in the language of spider diagrams can be exported to sentential first-order logic formulae which, in turn, may be imported into sentential theorem provers. It is also possible to import sentential formulae into Speedith by translating them into spider diagrams. This pluggable feature of Speedith was exploited via the MixR framework [32] where Speedith was integrated with a sentential theorem prover Isabelle to result in the Diabelli [31] heterogeneous reasoning system (i.e., a mixture of diagrammatic and sentential inference steps make the statements and the proof of a theorem) – for more information, see [32].

We now present the implementation of Speedith through the design of its architecture, the representations for spider diagrams that it uses, its reasoning engine and how it enables the construction of proofs, and finally its user interface.

4.1 Architecture

Speedith consists of four main components:

1. *abstract representation* of spider diagrams (Speedith's internal representation of spider-diagrammatic statements);
2. *reasoning kernel* that provides Speedith with its proof infrastructure (it contains a collection of spider-diagrammatic inference rules, handles the application of inference rules, and manages proofs);
3. *external communication system* which includes input and output mechanisms for spider-diagrammatic and sentential formulae – this system enables external verification through existing general-purpose theorem provers; and

4. *graphical user interface*, which includes spider diagram visualisation (using *iCircles visualisation algorithm* for unitary spider diagrams and *SpiderDrawer* for pen input of hand drawn spider diagrams), user interaction with spider-diagrammatic elements, graphical user interface panels for interactive proof management and interactive application of inference rules.

We separated these four components into four libraries: *Speedith Core*, *iCircles*, *SpiderDrawer* and *Speedith GUI*. Speedith Core contains the first three components (the abstract representation, the reasoning kernel, and the external communication system). The iCircles library[12] contains only unitary (but not compound) spider diagram visualisation. Therefore, we added support for compound spider diagrams in Speedith (on top of iCircles, rather than extending iCircles). Note that Speedith Core and iCircles may be used independently of each other. This enables the use of Speedith as a reasoning kernel without the user interface. The Speedith GUI library depends on both Speedith Core and iCircles. The SpiderDrawer library enables the user to input spider diagrams via a pen input interface. These four libraries together make up Speedith. Fig. 19 shows an outline of Speedith's architecture.

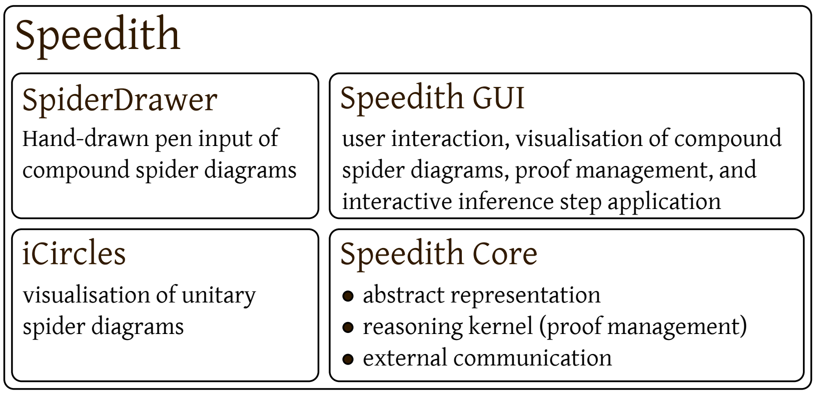

Speedith
SpiderDrawer
Hand-drawn pen input of compound spider diagrams
Speedith GUI
user interaction, visualisation of compound spider diagrams, proof management, and interactive inference step application
iCircles
visualisation of unitary spider diagrams
Speedith Core
● abstract representation
● reasoning kernel (proof management)
● external communication

Fig. 19 Speedith consists of four libraries: Speedith Core, iCircles, SpiderDrawer and Speedith GUI. Speedith Core consists of three components: the abstract representation of spider diagrams, the reasoning kernel and external communication. Visualisation of unitary spider diagrams is performed with iCircles. Visualisation of compound spider diagrams and user interaction is provided by Speedith GUI. Spider diagrams can be input by hand via a pen interface SpiderDrawer.

[12] iCircles was originally created by Stapleton et al. [22] to draw Euler diagrams (spider diagrams without any spiders in them). Flower then extended iCircles to include the visualisation of spiders, so iCircles supports the visualisation of unitary spider diagrams only, but not compound spider diagrams.

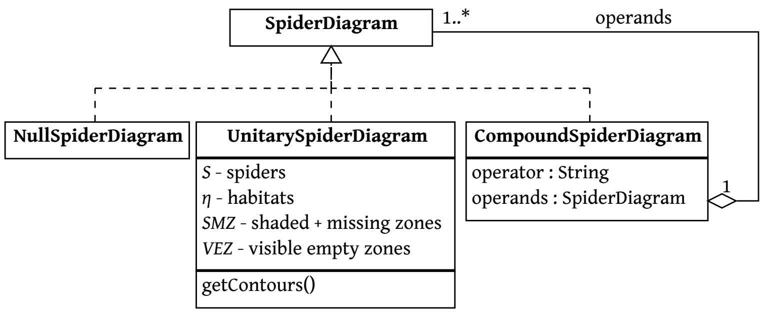


Fig. 20 The structure of the abstract representation for all spider diagrams in Speedith.

4.2 Abstract representation

Speedith uses an abstract representation, called SAR, to store and manipulate spider-diagrammatic sentences. The representation is of the form of an expression tree whose nodes are spider diagrams, which can be of the following three types (see Fig. 20):

1. ***Unitary spider diagram node***: contains a full description of a unitary spider diagram (as defined in Definition 1).
2. ***Compound spider diagram node***: connects one or two spider diagram nodes with a logical connective.
3. ***Null spider diagram node***: which denotes tautology and is also a shorthand for an empty unitary spider diagram.[13]

In Speedith, unitary spider diagrams are captured in a different, but equivalent, way to their presentation in Definition 1. In particular, Speedith modifies or omits some of the sets present in the unitary spider diagram tuple. Specifically, Speedith does not store the sets L and Z (the sets of all contour labels and the set of present zones). In addition, Speedith merges the sets ShZ and MZ (from Definition 7) into SMZ. The set SMZ thus contains zones that are either shaded or are missing in the unitary spider diagram. Finally, Speedith also uses the set VEZ which contains zones that are shaded and are not part of any spider's habitat (contain no spiders) – called *empty* zones – but are still visible in the unitary spider diagram. Speedith uses this structure in order to match the semantics of spider diagrams more closely. In fact, zones that convey no semantic information are not stored (i.e., zones with no spider feet or shading). This also removes data redundancy, optimises memory consumption, and lowers the complexity of spider diagram maintenance and manipulation as it removes the possibility of a spider having a foot in a missing zone.

Note that all sets from the tuple $d_u = (L, Z, ShZ, S, \eta)$ (see Definition 1) may still be obtained from Speedith's representation. The set of all contour

[13] An empty unitary spider diagram is the tuple $d = (\emptyset, \{(\emptyset, \emptyset)\}, \emptyset, \emptyset, \emptyset)$.

labels L is obtained via the `getContours()` method (which takes an arbitrary zone $(in,\ out)$ and returns the union $in \cup out$, which equals L). The set MZ equals $SMZ \setminus (VEZ \cup habitats)$, where *habitats* is the set of all zone where any spider has a foot. Lastly, the set of all zones Z equals:

$$\{(in,\ L - in) \mid in \subseteq L\ \} \setminus MZ.$$

The structure for compound spider diagrams in Speedith is an implementation of the compound spider diagram syntax as specified in Definition 2. Speedith provides support for compounding spider diagrams with all connectives, that is, logical equivalence, implication, disjunction, conjunction and negation of spider diagrams.

When creating unitary, compound, or null spider diagrams, Speedith will ensure that there is always only one instance of that diagram available (without duplicates). For example, the compound spider diagram $d_a \vee d_a$ connects the same instance of the spider diagram d_a through the logical connective $\vee$. In fact, Speedith makes it impossible that there are two distinct syntactically equal spider diagrams used anywhere in a spider-diagrammatic statement.

Definition 12 We say that two spider diagrams, say d_1 and d_2, are *syntactically equal* if they fall under one of the following:

1. Both d_1 and d_2 are null spider diagrams.
2. Both d_1 and d_2 are unitary spider diagrams and their sets S, SMZ, and VEZ are equal and so is their map of habitats η.
3. Both d_1 and d_2 are compound spider diagrams of the forms $d_1 = d_l \bigotimes d_r$ and $d_2 = d'_l \bigoplus d'_r$ where $\bigotimes$ and $\bigoplus$ are the same logical connective, d_l syntactically equals d'_l, and d_r syntactically equals d'_r.

To denote syntactical equality between two diagrams d_1 and d_2, we use the equality sign $d_1 = d_2$.

This method is used to preserve memory (by not creating multiple instances of any spider diagram) and, more importantly, for faster syntactical equality comparison. We ensure no two syntactically equal diagrams are stored by maintaining a pool of currently instantiated spider diagrams. Whenever a new spider diagram is created, it is checked whether the same spider diagram already exists. If one already exists the new one is deleted and the old one is returned.

An advantage of Speedith's abstract representation is the ease of transformation of spider diagrams into sentential first-order logic (for more information, see [32]). This representation is also designed with the aim for quick manipulations of spider diagrams through the application of inference rules (to make reasoning as efficient as possible).

4.2.1 Text format

Speedith specifies a textual format of the abstract representation of spider diagrams. This textual representation is called SDT (short for *spider diagrams text*), and is capable of expressing any valid spider diagram.

Speedith contains a parser capable of reading spider diagrams in the SDT format and producing the corresponding abstract representation. Fig. 21 shows an example spider diagram in the SDT format.

```
BinarySD {
    operator = "op -->",
    arg1 = PrimarySD {
        spiders = ["s1", "s2"],
        habitats = [
            ("s1", [(["A"], ["B"]), (["B"], ["A"])]),
            ("s2", [(["A", "B"], [])])
        ],
        sh_zones = []
    },
    arg2 = PrimarySD {
        spiders = ["s1", "s2"],
        habitats = [
            ("s1", [(["A"], ["B"]),    (["A", "B"], [])]),
            ("s2", [(["A", "B"], []), (["B"], ["A"])])
        ],
        sh_zones = []
    }
}
```

Fig. 21 A spider diagram expressed in the SDT format. This SDT example expresses the spider diagram $d_1 \longrightarrow d_6$ from Fig. 1 on page 3.

4.3 The reasoning kernel

Internally, reasoning in Speedith is performed via the reasoning kernel. The reasoning kernel checks whether the inference rules are used correctly. In case the user chooses an inference rule which is not applicable to the current spider diagram, the reasoning kernel will report this mistake and abort the application. Speedith applies only valid inference rules. Speedith thus produces proofs whose correctness relies on the soundness and completeness of spider diagrams, proved by Howse et. al. [12], and the correctness of our implementation.

4.3.1 Proofs in Speedith

The reasoning kernel manages the entire proof of a spider-diagrammatic theorem. Speedith's proof management infrastructure consists of the `Proof`, `Goals`, and `InferenceRule` data structures. Fig. 22 shows a class diagram of the proof-management infrastructure within Speedith. Fig. 22 uses the Unified Modelling

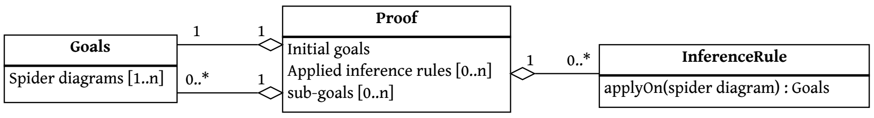


Fig. 22 A simplified class diagram of the Speedith's *proof management infrastructure.*

Language (UML) class diagram notation. For example, the line connecting `Proof` and `InferenceRule` indicates that a single `Proof` contains zero or more `InferenceRule` components. The `Goals` data structure contains a list of spider diagrams in their abstract representations. These are, for example, the spider-diagrammatic statements we set out to prove. A single goal is simply a spider diagram. The `InferenceRule` component identifies a Speedith's inference rule. It is responsible for performing the actual transformation on a spider-diagrammatic goal. Speedith contains a number of specific implementations of the `InferenceRule` component, each of which represents an inference rule outlined in Section 3. Finally, the `Proof` data structure stores the entire proof. It contains the initial goals (i.e., the statement that we set out to prove), a list of inference rules that were successfully applied to the initial goals and the resulting list of sub-goals. Proofs in Speedith are thus sequences of goals and inference rule applications. A proof starts with *initial goals*, here denoted with Δ, which is a set of spider diagrams that we want to prove are theorems. The proof then proceeds by applying an inference rule to a spider diagram D, where $D \in \Delta$. The result of the inference rule application is a spider diagram D', where D' logically entails D (i.e., $D' \vDash D$).

An application of an inference rule is called an *inference step*. We use the following notation to denoted an inference step where the inference rule Rule is applied to the set of goals Δ:

$$\frac{\Delta'}{\Delta}\ \textsc{Rule}, \tag{12}$$

where $\Delta' = (\Delta \setminus \{D\}) \cup \{D'\}$. A proof may consist of an arbitrary number of inference steps. Formula 13 outlines the structure of all Speedith's proofs (using the traditional inference step bar notation, where the proof is performed starting from the bottom and progressing upwards):

$$\cfrac{\cfrac{\cfrac{\top}{\vdots}\ \textsc{Rule}_n}{\Delta'}\ \textsc{Rule}_2}{\Delta}\ \textsc{Rule}_1 \tag{13}$$

The proof is finished once only null spider diagrams are left in the set of goals (in Formula 13, this is denoted by the symbol $\top$).

Speedith supports application of both forward-style and backward-style inference steps. Forward rules take a spider diagram d and produce a new spider diagram d' such that $d \vDash d'$. In Speedith forward rules are applied to goals of the following form: $d_1 \longrightarrow d_2$. Particularly, forward inference rules transform the left-hand side of the implication, here denoted with d_1. Thus, a forward inference step in Speedith takes the following form:

$$\frac{d_1' \longrightarrow d_2, \Delta}{d_1 \longrightarrow d_2, \Delta}\ \textsc{ForwardRule}. \tag{14}$$

On the other hand, backward inference steps in Speedith are performed directly with inference rules that take a diagram d and produce a new diagram d' such that $d' \vDash d$. In this case no implication is needed in a goal. Thus, a backward inference rule takes the following form:

$$\frac{d', \Delta}{d, \Delta}\ \textsc{BackwardRule}. \tag{15}$$

4.3.2 Targeting and transformation of spider diagrams

Every spider diagram in Speedith is a tree, called an *abstract syntax tree.* Every sub-tree in the abstract syntax tree is again a spider diagram. Unitary and null spider diagrams are leaves of the tree, while compound spider diagrams are inner nodes with one or two child-nodes. Speedith's inference rules perform transformations on these abstract syntax trees. For example, the Combining rule (see Fig. 13 on page 16) replaces a compound spider diagram node with a unitary spider diagram node. Speedith's inference rules are therefore tree-transformers, implemented with the visitor pattern. The visitor pattern traverses every node of the tree in a particular order until the target of the inference rule application is reached. Once the inference rule has visited the target node, it performs the transformation of the node. The transformation produces a new tree, that is, a new spider diagram, instead of applying the change on the visited tree directly.

Users can select highly specific elements of a spider diagram as the targets of the inference rules. This is performed via a graphical point-and-click mechanism (see Fig. 28 on page 30). For example, the SplitSpider rule (see Fig. 11 on page 14) acts on a sub-habitat of a specific spider that lives within a particular unitary spider diagram. This differs from inference rules in sentential theorem provers, where inferences are typically applied to the outermost connective or the inference automatically selects the first suitable target of the transformation. Therefore, Speedith requires an exact addressing mechanism.

Speedith defines an addressing mechanism at the level of the abstract syntax tree. It numbers the nodes in an abstract syntax tree with a left-to-right pre-order traversal. Fig. 23 shows a numbering example of a hypothetical spider-diagram abstract representation. The numbering starts with 0 at the

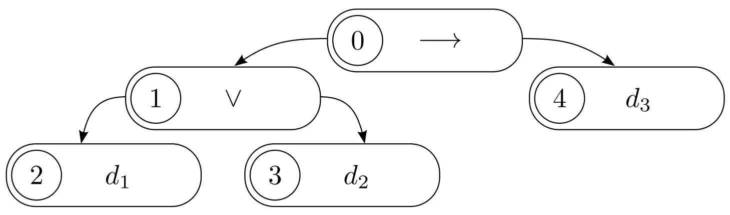


Fig. 23 Node numbering of a compound spider diagram.

root node and continues recursively, starting with the left sub-node and then the right sub-node. As a result, every sub-diagram has an associated number, called its *sub-diagram index*. This index is used to uniquely identify the sub-diagram on which an inference rule should be applied. However, this does not fully satisfy the targeting requirements of spider-diagrammatic inference rules. Speedith also allows to target arbitrary elements of a unitary spider diagram: sets of zones, sets of spiders, spider feet, and sets of contour labels. Thus, sub-diagram indices and the ability to target particular elements of unitary spider diagrams allow for exact targeting of any element within any unitary spider diagram, regardless of where it is nested within a surrounding compound spider diagram. Interactive selection of the target for a specific inference rule is covered in more detail next.

4.4 User interface

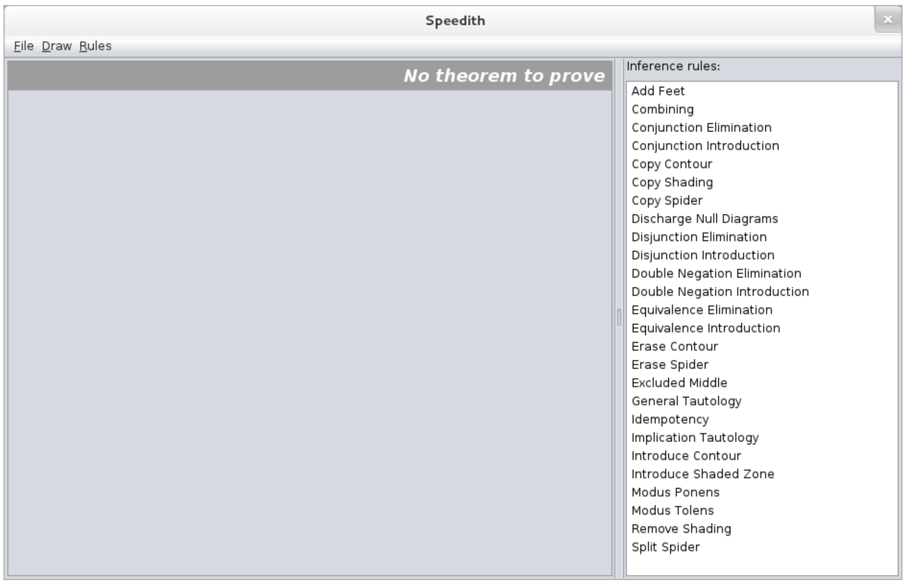
Speedith's user interface allows users to enter spider diagrams and perform spider-diagrammatic proofs interactively. For example, users can choose arbitrary elements of a spider diagram directly from Speedith's visualisation of a spider diagram. At start-up, Speedith's window contains a blank surface that is used to display and manage spider-diagrammatic proofs. Fig. 24 shows the initial state of the user interface.

4.4.1 Diagrams input

Speedith supports two modes of input of spider diagrams: the SDT textual input and the hand drawn spider diagrams via SpiderDrawer.

SDT textual input: The textual input method allows entry of any valid spider diagram: Fig. 25 shows an example. The dialogue in Fig. 25 can be activated with the key combination `Ctrl+T`.

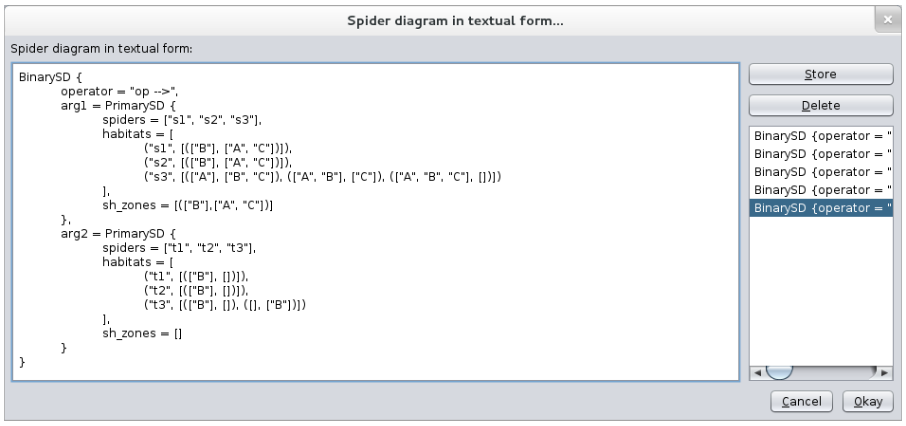
The user-entered SDT is loaded into the Speedith's parser. The parser converts the SDT representation into an abstract syntax tree. Finally, the abstract syntax tree becomes the initial goal of the current proof, which is immediately visualised in the proof panel.



Inference rules:

- Add Feet
- Combining
- Conjunction Elimination
- Conjunction Introduction
- Copy Contour
- Copy Shading
- Copy Spider
- Discharge Null Diagrams
- Disjunction Elimination
- Disjunction Introduction
- Double Negation Elimination
- Double Negation Introduction
- Equivalence Elimination
- Equivalence Introduction
- Erase Contour
- Erase Spider
- Excluded Middle
- General Tautology
- Idempotency
- Implication Tautology
- Introduce Contour
- Introduce Shaded Zone
- Modus Ponens
- Modus Tolens
- Remove Shading
- Split Spider

Fig. 24 Speedith's initial state. The complete set of inference rules is shown in the list on the right. The large grey area on the left is the proof management panel, which currently contains no goals.



Spider diagram in textual form:

```
BinarySD {
        operator = "op -->",
        arg1 = PrimarySD {
                spiders = ["s1", "s2", "s3"],
                habitats = [
                        ("s1", [(["B"], ["A", "C"])]),
                        ("s2", [(["B"], ["A", "C"])]),
                        ("s3", [(["A"], ["B", "C"]), (["A", "B"], ["C"]), (["A", "B", "C"], [])])
                ],
                sh_zones = [(["B"],["A", "C"])]
        },
        arg2 = PrimarySD {
                spiders = ["t1", "t2", "t3"],
                habitats = [
                        ("t1", [(["B"], [])]),
                        ("t2", [(["B"], [])]),
                        ("t3", [(["B"], []), ([], ["B"])])
                ],
                sh_zones = []
        }
}
```

```
BinarySD {operator = "
BinarySD {operator = "
BinarySD {operator = "
BinarySD {operator = "
BinarySD {operator = "
```

Fig. 25 The dialogue for entering spider diagrams in the textual form.

SpiderDrawer pen input: Instead of typing the SDT representation of spider diagrams, the user can alternatively quickly and easily draw spider diagrams via a pen input interface SpiderDrawer [2]. Speedith accesses SpiderDrawer via an input canvas window where the user draws the diagrams by hand with a stylus. SpiderDrawer is implemented in Java and is platform independent.[14] It

[14] Hand-drawn input support for spider diagrams similar to SpiderDrawer is currently being developed by Wang et al. [34] within the SketchSet tool. But unlike SpiderDrawer and

uses the RATA [3] library for shape recognition, and the Tesseract [21] library for text and connectives recognition.[15]

RATA is a shape recognition program that uses data mining analysis to recognise single stroke drawings. It is used to recognise 5 out of 7 shapes that make up spider diagrams. In particular, circles, rectangles, spider feet, spider legs, and shading are recognised. RATA first collects data (from a set of training examples) and then uses machine learning techniques to classify the shapes. The other two shapes, labels and logical connectives, are handled separately by Tesseract which is an open source optical character recognition program.

SpiderDrawer automatically coverts the recognised shapes, text and connectives from hand-drawn free-form and redraws them to precise formal drawings. SpiderDrawer recognises the relations between all the elements and checks them against the valid spider diagram representation. If the drawing is not a valid spider diagram, SpiderDrawer will not allow the user to proceed to the next, reasoning stage of the proof. If the drawing is a valid spider diagram, then SpiderDrawer allows the user to proceed with the proof and passes the drawn spider diagram's abstract representation to Speedith. Fig. 26 shows the final SpiderDrawer's pen drawing of the same spider diagram as in Fig. 25.

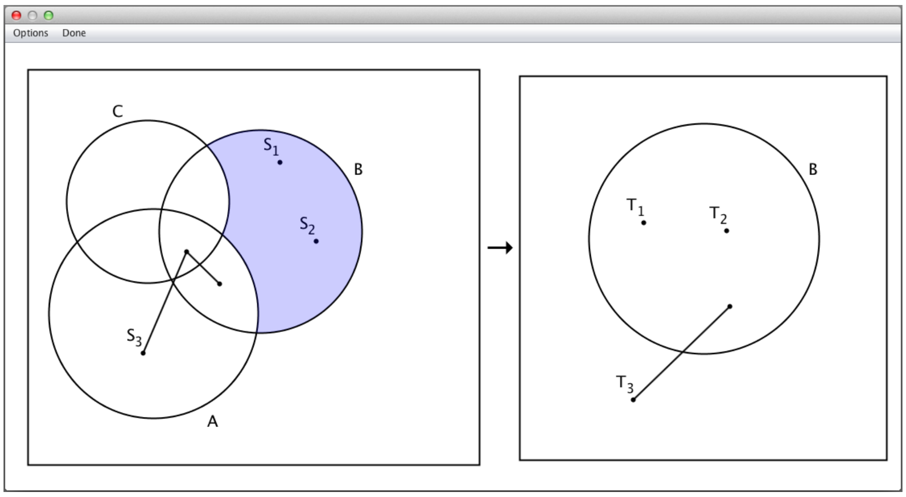


Fig. 26 The SpiderDrawer window for hand-drawing spider diagrams using a stylus. Each part of the free-form hand-drawn diagram is snapped into precise formal drawing for consistency and ease of reading.

Speedith, SketchSet is platform dependent and requires proprietary Windows libraries. Since this would seriously limit Speedith's reach to users, we instead developed SpiderDrawer.

[15] `https://code.google.com/p/tessaract-ocr/`

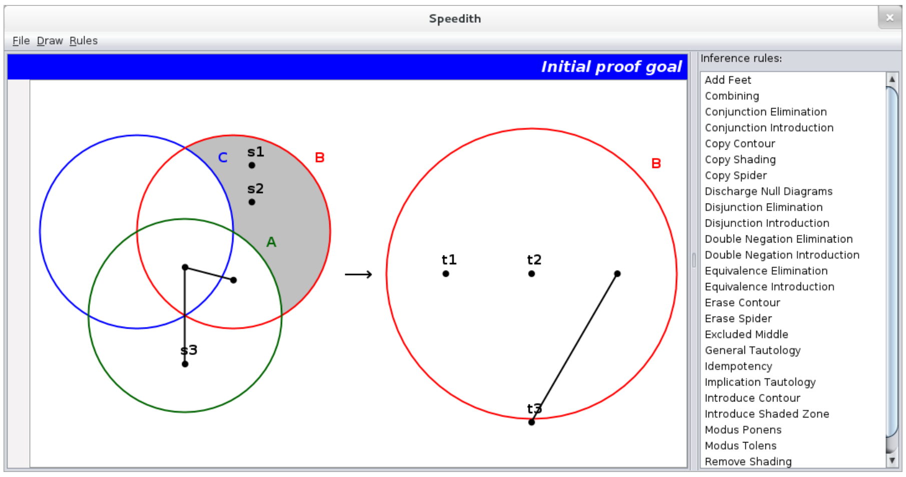


Fig. 27 The visualisation of the spider diagram as input via the text input dialogue in Fig. 25 or input via the SpiderDrawer pen input interface in Fig. 26. This is also Speedith's visualisation of the diagram in Fig. 3.

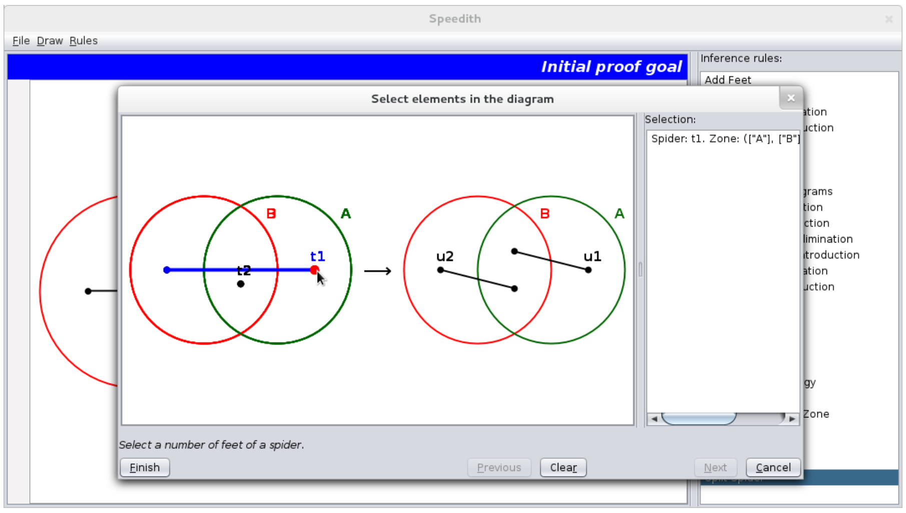
Speedith then makes this the initial goal of the current proof, which is immediately visualised[16] in the proof panel. Fig. 27 shows Speedith's proof panel with an initial goal. It shows the spider diagram from Fig. 25 and Fig. 26.

Now, the user may apply inference rules (enumerated in a list located to the right side of the proof panel). Double-clicking on an inference rule opens a window for selecting the target of the application: Fig. 28 shows an example. In this particular example the user has to select a set of spider feet for the SplitSpider inference rule. The proof is finished after all proof goals are reduced to null spider diagrams. Fig. 2 on page 4 shows an instance of a finished proof in Speedith.

4.4.2 Diagrams display

Speedith uses iCircles [22] library and algorithm for drawing unitary spider diagrams, and extends it with compound spider diagrams visualisation. Speedith also provides user interaction on top of the iCircles drawing surface. This allows the user to highlight and select specific parts of compound and unitary spider diagrams. Speedith uses this extended algorithm to display all spider-diagrammatic statements. Figs. 2, 27, 28, and 29 show Speedith's visualisations of compound spider diagrams. Fig. 28 also captures user interaction with the spider diagram. In this particular figure a spider's foot and leg are highlighted to indicate that the user may click on them and thereby select the foot as the target of an inference rule.

[16] Embedding SpiderDrawer's canvas directly within Speedith's proof panel (rather than using it as a separate pop-up window) is work that we plan for the future.



Spider: t1. Zone: (["A"], ["B"]

Select a number of feet of a spider.

Fig. 28 Speedith's window for interactively selecting the exact target for any inference rule. Speedith guides the user stepwise during the target selection. The label in the lower-left corner (above the "`Finish`" button) displays the instruction for every target selection step. This label also displays errors in the case when the user tries to select an invalid combination of targets.

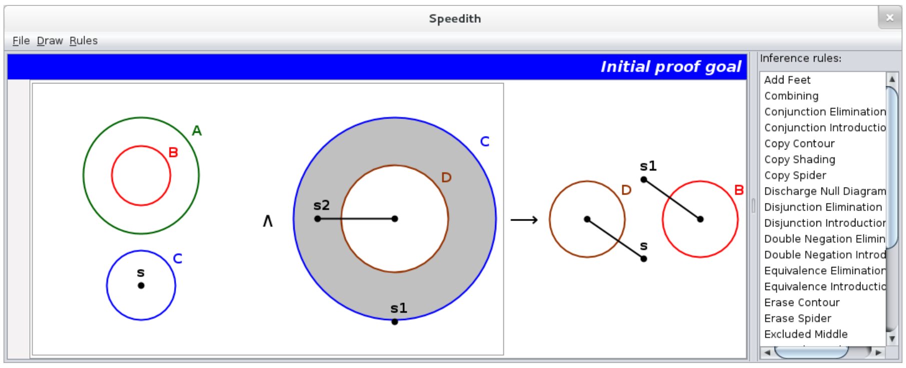


Fig. 29 The visualisation of the diagram that was proved in Fig. 18. This example demonstrates Speedith's and iCircles' capability of drawing diagrams with missing zones.

Algorithm Outline Here is the iCircles algorithm [22], extended by Flower, for visualising unitary spider diagrams:

1. The first step takes a set of visible zones and draws them by placing labelled contour circles onto the drawing panel. Note that iCircle only uses circles for contours. In complex diagrams with numerous contours and relations, it may not be possible to draw a contour using a single circle, so multiple circles are used. The algorithm then stores the *concrete zones* in an enumerable collection. The result of the first step is an Euler diagram. The diagram at this stage already contains shading, but it does not yet contain spiders. The algorithm will try to use missing zones wherever possible. If a

missing zone cannot be used to denote empty sets, then the algorithm will use shaded zones instead.
2. Shading is applied to the set of shaded zones. The algorithm finds the corresponding concrete zone (using the collection constructed in the first step) and fills it with grey colour.
3. In the last step, spider feet and legs are drawn. This step expects as input a set of spiders S with their habitats

$$\{h \mid h = \eta(s) \wedge s \in S\}\,.$$

For each spider and each zone z in its habitat a point p is found in the zone z. These points are locations near which the spider's feet will be drawn. The legs connect the points p by giving priority to points that are located within adjacent zones. The algorithm checks if any of the legs pass through any of the other spiders' feet. If so, the offending feet are nudged. Nudging is applied repeatedly in eight principal directions until a suitable position is found. Note that during the nudging step the legs are adjusted accordingly. As a consequence, all spider diagrams needed in any proof can be automatically displayed.

To visualise compound spider diagrams Speedith extends this algorithm in the standard way by drawing connectives and nested unitary spider diagrams.

5 Discussion

We evaluate our work in terms of how it compares to similar existing work; and also in terms of expressiveness, extensibility, and usability; finally we point out a few limitations of Speedith. Speedith is implemented in Java. Its sources are available from `https://github.com/urbas/speedith`.

5.1 Related work

Here we concentrate on relating aspects of Speedith to other diagrammatic reasoning systems, to similar diagrammatic logics, and to other sketching interfaces.

Diagrammatic systems Other diagrammatic theorem provers most related to Speedith are the prover by Flower et al. [7], Edith [25], DIAMOND [13], and Cinderella [17].

The system developed by Flower et al. [7] works, unlike Speedith, with unitary spider diagrams only, and is fully automated. Edith is an interactive diagrammatic theorem prover for Euler diagrams that finds the shortest and readable proof for only a subset of the spider diagrammatic language we are targeting. Whilst Edith is the closest to Speedith in terms of the domain it targets, it does not support spiders nor compound diagrams with logical connectives, and thus provides fewer inference rules and proves a much smaller class of theorems. These theorems only include diagrams that express subset and disjointness relationships with no information on set cardinality, except for when sets are empty.

Speedith differs from both, Flower et al.'s system and Edith, in that it works with the complete spider-diagrammatic language as defined in Section 2. Moreover, unlike these two systems, Speedith provides fully interactive proofs. Also, Speedith's proofs are guaranteed to be sound and correct. In addition, they can be verified with another external symbolic theorem prover, Isabelle, via a MixR heterogeneous reasoning framework – for details, see [32].

Diamond, on the other hand, supports external verification, but the class of problems it tackles is inductive theorems of natural numbers. By contrast, Speedith targets theorems about set constraints. Thus these two diagrammatic systems target different domains.

Cinderella targets the domain of geometry and uses a different approach to its diagrammatic proofs. The user gradually constructs the geometric model of the theorem, while in the background an automated theorem prover verifies that each construction step results in a valid geometric diagram. Thus, the steps in Cinderella are not guaranteed to be sound, and the proof process does not follow the standard inference rule application pattern.

Finally, Speedith was designed with language extensions in mind. Spider diagrams could be extended with non-monadic relations, functions, and universal quantification of elements. Designing meaningful and complete diagrammatic inference rules for such extended language is hard and remains work for the future.

Diagrammatic logics There is a variety of diagrammatic logics that are similar to spider diagrams. Of particular interest is the Euler diagram fragment of spider diagrams. Hammer was perhaps the first to devise a formal logic for unitary Euler diagrams [9]. This has since been extended to include the classical logical connectives $\wedge$, $\vee$ and $\neg$ for which soundness and completeness have been established [24]; it is a trivial matter to extend the inference rules in order to obtain completeness when the connectives $\longrightarrow$ and $\longleftrightarrow$ are added to the syntax of this Euler diagram logic. Thus, Speedith automatically provides theorem proving support for these systems – since they are fragments of the spider diagram system – and can be easily extended to include inference rules developed specifically for those logics.

There exist different formalisations of Euler diagram logics, such as [24, 29], and Shin's seminal work on Venn-I and Venn-II [20] extends Venn diagrams to include syntax to assert the non-emptiness of sets. Since Speedith allows for ready implementation of new rules, it would be possible to tailor Speedith to these other logics. Also related to spider diagrams are Swoboda and Allwein's Euler/Venn diagrams [28]. Euler/Venn diagrams incorporate constants to represent specific individuals, as opposed to the existence of elements in spider diagrams. Thus, Speedith also provides a basis for theorem proving technology implemented for Euler/Venn diagrams.

Sketching Interface Speedith builds on results on user interaction work that focuses on converting sketches into beautified diagrams. Numerous sketch tools have been proposed for visual languages including concept maps [14], graphs [18], UML class diagrams [4,10] and Euler diagrams [35]. Of particular relevance to Speedith is SketchSet which provides sketch recognition and conversion of some components of unitary spider diagrams [26]; SketchSet extends SketchNode which was developed for graphs in isolation [19]. Similarly to SketchSet, Speedith can recognise closed cures, their labels, and spiders. Moreover, Speedith takes drawn spider diagram recognition much further than SketchSet in that it can recognise shading, rectangles that form the boundaries of unitary diagrams, and the logical connectives, $\wedge$, $\vee$, $\longrightarrow$ and $\longleftrightarrow$. Thus, unlike any other sketch tool, Speedith is capable of recognising all of the syntax of spider diagrams.

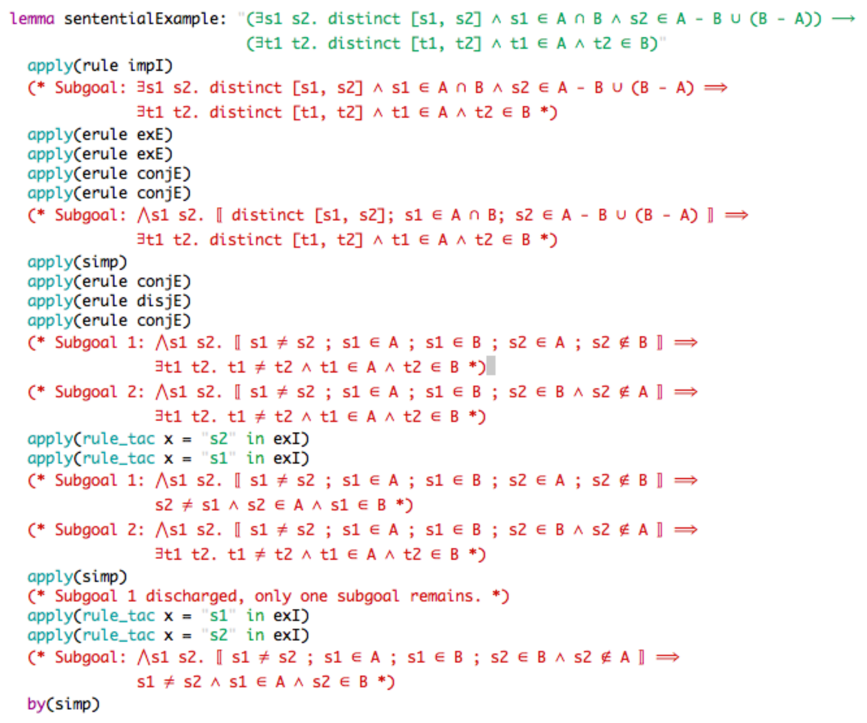
5.2 Properties

Expressiveness In terms of the theorems that can be proved using Speedith, spider diagrams have the expressiveness of MFOLE [27]. This means that spider diagrams can express theorems about set constraints [1]. These constraints include subset and disjointness relationships as well as both upper and lower (finite) bounds on cardinality. Since the logic is sound and complete, Speedith can also, therefore, prove all theorems about set constraints. That is, Speedith is able to prove all theorems of MFOLE, expressed using spider diagrams – this is a significant range and depth of theorems. The fact that spider diagrammatic logic is monadic means that with Speedith we cannot prove more complex theorems involving arbitrary n-ary relations, where $n > 1$.

Extensibility Extending Speedith with new inference rules is straightforward and only requires the addition of a single class. Implementation source code of inference rules is short and typically consist of about 100 lines of Java code (or 70 lines of Scala code). A significant part of that code is used for the preamble containing the name of the inference rule, its description, and instructions on how to use it. The remainder of the code is the actual logic of the inference rule. For example, the Combining rule and NegationElimination rule consist of 40 lines and 20 lines of logic code respectively. Speedith is also equipped with helpful libraries (e.g., the habitat builder, the region builder, set manipulation and spider manipulation libraries) that further simplify the implementation of the logic of new inference rules. Moreover, these libraries can also be used to write automated unit tests with which the implementer can improve the correctness of the implementation of the new inference rule. Clearly, soundness and completeness of the now new extended set of inference rules need to be proved again.

Usability One of Speedith's main contributions is its representation of formulae and proof steps. This differentiates it from interactive sentential theorem provers (such as Isabelle) in that it provides a domain-specific, visual, and thus perhaps more intuitive approach to proofs in MFOLE. Speedith's inference rules, which perform simple visual transformations of the diagrammatic statement are succinct and 'natural' – they capture the notion of truthfulness that humans find easy to understand. In contrast, proofs of the same theorems in sentential theorem provers consist of lower-level, more fine-grained proof steps which make them longer and arguably harder to "see" the intuition behind the proof.

Fig. 30 shows Isabelle's sentential proof of the same theorem that is proved diagrammatically in Fig. 1 on page 3 (the screenshot of its proof in Speedith is in Fig. 2). We suggest that it is perhaps clearer in the diagrammatic proof why the theorem holds and how the proof is constructed. However, psychological validity tests would have to be carried out on users to confirm this.



```
lemma sententialExample: "(∃s1 s2. distinct [s1, s2] ∧ s1 ∈ A ∩ B ∧ s2 ∈ A - B ∪ (B - A)) ⟶
                          (∃t1 t2. distinct [t1, t2] ∧ t1 ∈ A ∧ t2 ∈ B)"
  apply(rule impI)
  (* Subgoal: ∃s1 s2. distinct [s1, s2] ∧ s1 ∈ A ∩ B ∧ s2 ∈ A - B ∪ (B - A) ⟹
              ∃t1 t2. distinct [t1, t2] ∧ t1 ∈ A ∧ t2 ∈ B *)
  apply(erule exE)
  apply(erule exE)
  apply(erule conjE)
  apply(erule conjE)
  (* Subgoal: ⋀s1 s2. ⟦ distinct [s1, s2]; s1 ∈ A ∩ B; s2 ∈ A - B ∪ (B - A) ⟧ ⟹
              ∃t1 t2. distinct [t1, t2] ∧ t1 ∈ A ∧ t2 ∈ B *)
  apply(simp)
  apply(erule conjE)
  apply(erule disjE)
  apply(erule conjE)
  (* Subgoal 1: ⋀s1 s2. ⟦ s1 ≠ s2 ; s1 ∈ A ; s1 ∈ B ; s2 ∈ A ; s2 ∉ B ⟧ ⟹
                ∃t1 t2. t1 ≠ t2 ∧ t1 ∈ A ∧ t2 ∈ B *)
  (* Subgoal 2: ⋀s1 s2. ⟦ s1 ≠ s2 ; s1 ∈ A ; s1 ∈ B ; s2 ∈ B ∧ s2 ∉ A ⟧ ⟹
                ∃t1 t2. t1 ≠ t2 ∧ t1 ∈ A ∧ t2 ∈ B *)
  apply(rule_tac x = "s2" in exI)
  apply(rule_tac x = "s1" in exI)
  (* Subgoal 1: ⋀s1 s2. ⟦ s1 ≠ s2 ; s1 ∈ A ; s1 ∈ B ; s2 ∈ A ; s2 ∉ B ⟧ ⟹
                s2 ≠ s1 ∧ s2 ∈ A ∧ s1 ∈ B *)
  (* Subgoal 2: ⋀s1 s2. ⟦ s1 ≠ s2 ; s1 ∈ A ; s1 ∈ B ; s2 ∈ B ∧ s2 ∉ A ⟧ ⟹
                ∃t1 t2. t1 ≠ t2 ∧ t1 ∈ A ∧ t2 ∈ B *)
  apply(simp)
  (* Subgoal 1 discharged, only one subgoal remains. *)
  apply(rule_tac x = "s1" in exI)
  apply(rule_tac x = "s2" in exI)
  (* Subgoal: ⋀s1 s2. ⟦ s1 ≠ s2 ; s1 ∈ A ; s1 ∈ B ; s2 ∈ B ∧ s2 ∉ A ⟧ ⟹
              s1 ≠ s2 ∧ s1 ∈ A ∧ s2 ∈ B *)
  by(simp)
```

Fig. 30 The same theorem as the one proved diagrammatically in Fig. 1 on page 3 (the screenshot of its proof in Speedith is in Fig. 2) is proved here sententially with Isabelle. Which one is easier to understand?

5.3 Speedith's limitations and future directions

The layout and drawing mechanism of Speedith currently draws the diagrams of each step of the proof (after each inference step was applied) independently of the previous steps. For example, a proof step in Speedith may change relative positions of contours and zones. A proof step may also relocate spider labels, feet and legs without consideration for any other diagrams in the proof. Thus, diagrams in consecutive proof steps can look radically different from each other. For future work, we aim to improve layout heuristics to take entire sequences of diagrammatic statements into account.

In addition, Speedith and iCircles do not provide a way for the user to manually specify positions of contours or spider feet. The complete spider diagram (compound or unitary) is laid out entirely automatically, whether input using the abstract sentential representation or drawn via SpiderDrawer. Although the iCircles algorithm contains heuristics to improve diagram readability it does not always succeed. Therefore, a future direction of research is to provide a way for users to manually influence and manipulate the diagram layout, and develop better heuristics to improve the automated layout.

SpiderDrawer is currently used only as a hand-drawn diagram input mechanism. Inference steps are selected from the list in the side menu, rather than with pen interaction on the diagrams. The entire pen-input SpiderDrawer canvas needs to be integrated as Speedith's primary interaction input and display canvas.

Lastly, Speedith is an interactive proof assistant. In particular, it does not provide reasoning automation. Extending Speedith to include automated proof search techniques is part of our future tasks.

6 Conclusion

By developing Speedith, we demonstrated the feasibility of diagrammatic reasoning systems that utilise a rule-based deductive proof approach. This is similar to the approach employed by general purpose proof assistants like Isabelle.

We also showed how to utilise existing state-of-the-art theorem provers to verify diagrammatic inference steps. Whilst we focused on spider diagrams, the approach can be used for other diagrammatic logics, such as existential graphs [5] or constraint diagrams [15].

Part of our future directions for Speedith includes extending the abstract representation to better control how diagrams are drawn. Moreover, we also envision extensions to the language of spider diagrams, proof search automation, use of Speedith in practical settings [16,6], and a study of scalability of proofs and their visualisation in Speedith.

Speedith may be used on its own as a stand-alone spider-diagrammatic theorem prover. It is, as of yet, the only interactive theorem prover for the language of spider diagrams with our extensions (such as the new logical operators of implication and negation in compound diagrams, and new inference rules). We believe Speedith can contribute to the development of the language and logic of spider diagrams. A possible future direction of research could be to use Speedith in order to extend the language of spider diagrams with new language features or to implement related diagrammatic logics.

Acknowledgements We thank Jean Flower for her help with iCircles extension, and Charlie Bashford-Chuchla for his implementation of SpiderDrawer as part of his MPhil project in the University of Cambridge Computer Laboratory.

A Corresponding regions

This section sets out the theory required to compare syntactically different regions at a semantic level. So-called corresponding regions are not necessarily syntactically identical, but they do represent the same set under any interpretation. Similar notions of corresponding sub-regions and super-regions will also be defined in this section. To identify corresponding regions, we need access to the missing zones and the empty zones in a unitary diagram, d. To simplify notation, we generalise the notion of an empty zone from earlier in the paper (recall, the set VEZ contains the zones which are shaded in d yet contain no spider feet). We define the set of empty zones to be $MZ(d)$ together with VEZ:

Definition 13 Let d be a unitary diagram. The **empty** zones of d are elements of the set

$$EZ(d) = MZ(d) \cup \{(in, out) \in ShZ(d) : \forall s \in S(d)\, (in, out) \notin \eta_d(s)\}.$$

Lemma 1 *Let d be a unitary diagram and let $I = (U, \Phi)$ be a model for d. Then the empty zones represent the empty set, that is*

$$\forall z \in EZ(d)\ \zeta_I(z) = \emptyset.$$

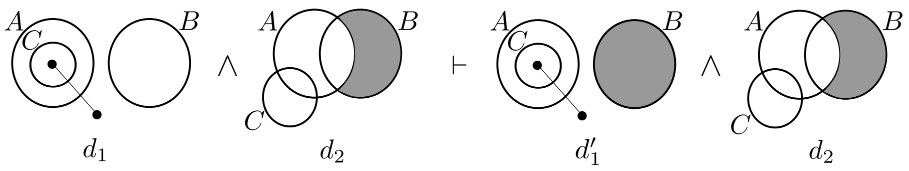


Fig. 31 Using empty zones to make deductions.

We use the concept of empty zones when defining inference rules: if we have two unitary diagrams taken in conjunction, and a zone, z, is empty in one of them, then we can use that information to determine how we apply inference rules on the other diagram, for example. To illustrate, in Fig. 31, in d_2 the zone $(\{B\}, \{A, C\})$ is empty so we can add shading to this zone in d_1, as shown in d_1'.

The notion of corresponding regions was introduced in [11] for Euler diagrams, where a syntactic definition was provided that established when two regions represented the same set. Here, we give a definition of corresponding regions that is effective for unitary spider diagrams taken in conjunction: we prove that our definition captures when two regions, one from d_1 and the other from d_2, necessarily represent the same set in all models for $d_1 \wedge d_2$. We also define the notion of a corresponding sub-region and a corresponding super-region, relating to subset and superset respectively.

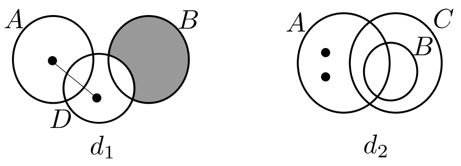


Fig. 32 Corresponding regions.

To illustrate, $r_1 = \{(\{A, D\}, \{B\}), (\{A\}, \{B, D\})\}$ and $r_2 = \{(\{A, C\}, \{B\}), (\{A\}, \{B, C\})\}$ both represent the same set and are corresponding; informally, they both represent the set $A \setminus B$. In this example, we can be confident that r_1 and r_2 represent the same set in any interpretation:

$$\begin{aligned}\chi_I(r_1) &= \zeta_I((\{A, D\}, \{B\})) \cup \zeta_I((\{A\}, \{B, D\}))\\ &= \zeta_I(\{A, D, C\}, \{B\})) \cup \zeta_I((\{A, C\}, \{B, D\})) \cup \zeta_I((\{A, D\}, \{B, C\})) \cup\\ &\quad \zeta_I((\{A\}, \{B, D, C\}))\\ &= \zeta_I((\{A, C\}, \{B\})) \cup \zeta_I((\{A\}, \{B, C\}))\\ &= \chi_I(r_2).\end{aligned}$$

Given d_1 and d_2 as in Fig. 32, the region

$$r_3 = \{(\{A, D\}, \{B\}), (\{A\}, \{B, D\}), (\{B\}, \{A, D\})\}$$

also represents the same set as r_2 (and r_1) in any model for $d_1 \wedge d_2$, since the zone $(\{B\}, \{A, D\})$ represents the empty set:

$$\begin{aligned}\chi_I(r_3) &= \zeta_I((\{A, D\}, \{B\})) \cup \zeta_I((\{A\}, \{B, D\})) \cup \zeta_I((\{B\}, \{A, D\}))\\ &= \zeta_I((\{A, D\}, \{B\})) \cup \zeta_I((\{A\}, \{B, D\}))\\ &= \chi_I(r_2).\end{aligned}$$

The region r_3 corresponds to r_2. In order to syntactically identify whether two regions, r and r', are corresponding, we need to transform them, altering the zones by adding labels. The transformation is based on the observation that given any zone, (in, out), and a label, l, not used in the zone,

$$\zeta_I((in, out)) = \zeta_I((in \cup \{l\}, out)) \cup \zeta_I((in, out \cup \{l\})).$$

The zone (in, out) can, thus, be transformed into the two zones $(in \cup \{l\}, out)$ and $(in, out \cup \{l\})$. We use this insight to define the notion of an *expansion* of a region, which given some set of labels iteratively 'splits' zones in this manner. In what follows, we denote the set of labels used in a region, r, by $L(r)$, so

$$L(r) = \bigcup_{(in, out) \in r} (in \cup out).$$

Definition 14 Let r be a region such that all of the zones, (in, out), in r ensure that $in \cup out = L(r)$. Let L' be a finite set of labels such that $L(r) \subseteq L'$. An **expansion** of r given L', denoted $exp(r, L')$, is the region defined as follows:

1. If $L' = L(r)$ then $exp(r, L') = r$.
2. If $|L' \setminus L(r)| = 1$ then

$$\begin{aligned}exp(r, L') = \{&(in \cup (L' \setminus L(r)), out) : (in, out) \in r\} \cup\\ \{&(in, out \cup (L' \setminus L(r))) : (in, out) \in r\}.\end{aligned}$$

3. If $|L' \setminus L(r)| > 1$ then
$$\mathit{exp}(r, L') = \mathit{exp}(r', L')$$
where
$$r' = \mathit{exp}(r, L'')$$
and $L'' = L(r) \cup \{\lambda\}$ for some label $\lambda \in L' \setminus L(r)$.

For example, given $r = \{(\{A\}, \{B\}), (\{B\}, \{A\})\}$ and $L' = \{A, B, C, D\}$, we have

$$\begin{aligned} \mathit{exp}(r, L') &= \mathit{exp}(\mathit{exp}(r, \{A, B, C\}), L') \\ &= \mathit{exp}(\{(\{A, C\}, \{B\}), (\{A\}, \{B, C\}), (\{B, C\}, \{A\}), (\{B\}, \{A, C\})\}, L') \\ &= \{(\{A, C, D\}, \{B\}), (\{A, C\}, \{B, D\}), (\{A, D\}, \{B, C\}), (\{A\}, \{B, C, D\}), \\ &\quad (\{B, C, D\}, \{A\}), (\{B, C\}, \{A, D\}), (\{B, D\}, \{A, C\}), (\{B\}, \{A, C, D\})\}. \end{aligned}$$

The order in which the labels are introduced during the expansion does not matter. Moreover, we do not change the represented set:

Lemma 2 *Let r be a region such that all of the zones, (in, out), in r ensure that $in \cup out = L(r)$. Let L' be a set of labels such that $L(r) \subseteq L'$. In any interpretation, $I = (U, \Phi)$,*
$$\chi_I(r) = \chi_I(\mathit{exp}(r, L')).$$

Proof (Sketch) The proof proceeds by induction on the cardinality of $L' \setminus L(r)$.

Definition 15 Let d_1 and d_2 be unitary diagrams. Let r_1 and r_2 be regions in $Z(d_1) \cup MZ(d_1)$ and $Z(d_2) \cup MZ(d_2)$ respectively. Then r_1 and r_2 are **corresponding**, denoted $r_1 \equiv_c r_2$, provided that

$$\begin{gathered} \mathit{exp}(r_1, L) \cup \mathit{exp}(EZ(d_1), L) \cup \mathit{exp}(EZ(d_2), L) \\ = \\ \mathit{exp}(r_2, L) \cup \mathit{exp}(EZ(d_1), L) \cup \mathit{exp}(EZ(d_2), L) \end{gathered}$$

where $L = L(d_1) \cup L(d_2)$. Furthermore, r_1 is a **corresponding sub-region** of r_2, denoted $r_1 \subseteq_c r_2$, provided that

$$\begin{gathered} \mathit{exp}(r_1, L) \cup \mathit{exp}(EZ(d_1), L) \cup \mathit{exp}(EZ(d_2), L) \\ \subseteq \\ \mathit{exp}(r_2, L) \cup \mathit{exp}(EZ(d_1), L) \cup \mathit{exp}(EZ(d_2), L). \end{gathered}$$

If r_1 is a corresponding sub-region of r_2 then r_2 is a **corresponding super-region** of r_1, denoted $r_2 \supseteq_c r_1$.

In Fig. 32, we have $r_4 \subseteq_c r_5$ where $r_4 = \{(\{A\}, \{B, D\})\}$ and $r_5 = \{(\{A\}, \{B, C\}), (\{A, C\}, \{B\}), (\{A, B, C\}, \emptyset)\}$. Intuitively, r_4 represents the set $A \setminus (B \cup D)$ and r_5 represents A, and we see that in any model, $I = (U, \Phi)$, for $d_1 \wedge d_2$ that $\chi_I(r_4) \subseteq \chi_I(r_5)$. The following theorem establishes that our syntactic correspondence relations respect the semantics as intended:

Theorem 3 *Let d_1 and d_2 be unitary diagrams and let r_1 and r_2 be regions in $Z(d_1) \cup MZ(d_1)$ and $Z(d_2) \cup MZ(d_2)$ respectively.*

1. *If $r_1 \equiv_c r_2$ then for all models $I = (U, \Phi)$ for $d_1 \wedge d_2$, $\chi_I(r_1) = \chi_I(r_2)$.*
2. *If $r_1 \subseteq_c r_2$ then for all models $I = (U, \Phi)$ for $d_1 \wedge d_2$, $\chi_I(r_1) \subseteq \chi_I(r_2)$.*
3. *If $r_1 \supseteq_c r_2$ then for all models $I = (U, \Phi)$ for $d_1 \wedge d_2$, $\chi_I(r_1) \supseteq \chi_I(r_2)$.*

Proof Suppose that $r_1 \equiv_c r_2$. Then, by definition,

$$\begin{gathered} exp(r_1, L) \cup exp(EZ(d_1), L) \cup exp(EZ(d_2), L) \\ = \\ exp(r_2, L) \cup exp(EZ(d_1), L) \cup exp(EZ(d_2), L) \end{gathered}$$

where $L = L(d_1) \cup L(d_2)$. By Lemma 2, given any interpretation, $I = (U, \Phi)$, we know that:

1. $\chi_I(r_i) = \chi_I(exp(r_i, L))$, and
2. $\chi_I(EZ(d_i)) = \chi_I(exp(EZ(d_i), L))$

for each $i \in \{1, 2\}$. Therefore, in any model for d_i, $\chi_I(exp(EZ(d_i), L)) = \emptyset$ since $\chi_I(EZ(d_i)) = \emptyset$ by Lemma 1. Thus, in any model for $d_1 \wedge d_2$,

$$\begin{aligned} \chi_I(r_1) &= \chi_I(exp(r_1, L)) \\ &= \chi_I(exp(r_1, L)) \cup \chi_I(exp(EZ(d_1), L)) \cup \chi_I(exp(EZ(d_2), L)) \\ &= \chi_I(exp(r_1, L) \cup exp(EZ(d_1), L) \cup exp(EZ(d_2), L)) \\ &= \chi_I(exp(r_2, L) \cup exp(EZ(d_1), L) \cup exp(EZ(d_2), L)) \qquad (*) \\ &= \chi_I(r_2) \end{aligned}$$

as required. The remainder of the proof is similar, noting that the line $(*)$ is, instead, a subset (superset) relation in the case of $\subseteq_c$ (resp. $\supseteq_c$).

B Formalised inference rules and proofs of soundness

First, we observe that all of the rules inherited from [12] and, trivially, all of the rules for logical connectives are sound.

Theorem 4 *The inference rules for the logical connectives are all sound, as are* AddFeet*,* IntroContour*,* EraseSpider*,* EraseContour*,* IntroShadedZone*,* RemoveShading*,* SplitSpider*,* ECludedMiddle*, and* Combining*.*

Here we include formalisations and soundness proofs for the diagrammatic inference rules that are new (i.e., those not included in [12]): NegationElimination, CopyContours, CopyShading and CopySpider. These rules are all equivalences, so we must show that their application preserves semantics. In what follows, we need to use the function $\Sigma_{d,U}$ that maps spiders to elements. Given an interpretation, $I = (U, \Phi)$, and a unitary diagram d, $\Sigma_{d,U}$ maps the spiders of d to the elements of U. Frequently, we will be considering a single interpretation and the spiders of more than one diagram. As such, rather than writing $\Sigma_{d,U}$, we more simply write Σ_d. We now formalise and prove the soundness of NegationElimination.

Negation Elimination Let d_n be a unitary diagram with exactly n spiders, no missing zones (so d_n is in Venn-form) where all spiders have single feet, and there is at most one zone, z_n, that contains spiders or shading. Let d_i, for $0 \le i < n$, be the unitary diagram where z_n contains exactly i spiders and shading. More precisely, d_i has components that are defined as follows:

1. the contour labels are $L(d_i) = L(d_n)$,
2. the zones are $Z(d_i) = Z(d_n)$,
3. the shaded zones are $ShZ(d_i) = \{z_n\}$,
4. the spiders are $S(d_i) = \{s_j : 1 \le j \le i\}$, and
5. the habitat of each spider, $s_j \in S(d_i)$, is $\eta_{d_i}(s_j) = \{z_n\}$.

Let d_{n+1} be a unitary diagram where z_n contains $n + 1$ spiders and no shading, that is:

1. the contour labels are $L(d_{n+1}) = L(d_n)$,
2. the zones are $Z(d_{n+1}) = Z(d_n)$,
3. the shaded zones are $ShZ(d_{n+1}) = \{z_n\}$,
4. the spiders are $S(d_{n+1}) = \{s_j : 1 \le j \le n + 1\}$, and
5. the habitat of each spider, $s_j \in S(d_{n+1})$, is $\eta_{d_{n+1}}(s_j) = \{z_n\}$.

The NegationElimination rule can be applied in the following way to $\neg d_1$:

1. if no zone in d_n contains spiders or shading then $\neg d_n$ is logically equivalent to $\perp$,
2. if z_n contains spiders but no shading in d_n then $\neg d_n$ is logically equivalent to $\bigvee\limits_{1 \le i < n} d_i$,
3. otherwise z_n contains spiders and shading in d_n and we have that $\neg d_n$ is logically equivalent to $\bigvee\limits_{1 \le i < n} d_i \vee d_{n+1}$.

Theorem 5 NegationElimination *is sound.*

Proof Given d_n as in the formalisation of the NegationElimination inference rule, we must show that

1. if no zone in d_n contains spiders or shading then $\neg d_n$ is logically equivalent to $\perp$,
2. if z_n contains spiders but no shading in d_n then $\neg d_n$ is logically equivalent to $\bigvee\limits_{1 \le i < n} d_i$,
3. otherwise z_n contains spiders and shading in d_n and we have that $\neg d_n$ is logically equivalent to $\bigvee\limits_{1 \le i < n} d_i \vee d_{n+1}$.

Let $I = (U, \Phi)$ be an interpretation. We consider the three cases in turn.

1. Case 1: if no zone in d_n contains spiders or shading then $\neg d_n$ is logically equivalent to $\perp$. We first show that d_n is modelled by I. Trivially, as d_n has no missing zones, $\chi_I(Z(d_n)) = U$. As there are no spiders, we also see that there exists a function, $\Sigma_{d_n} : S(d_n) \to U$, such that
   (a) $\forall s \in S(d_n)\Big(\Sigma_{d_n}(s) \in \chi_I(\eta_{d_n}(s))\Big)$, and
   (b) $\forall z \in ShZ(d_n)\Big(\zeta_I(z) \subseteq im(\Sigma_{d_n})\Big)$,

   where $im(\Sigma_{d_n})$ is the image of the function Σ_{d_n} (i.e., the set of elements in U to which Σ_{d_n} maps spiders). As there are no shaded zones in d_n, it is trivial that for all shaded zones in d_n, $\zeta_I(z) \subseteq im(\Sigma_{d_n})$. Hence I models d_n. As I was arbitrary, it follows that every interpretation models d_n. Therefore, $\neg d_n$ has no models and is logically equivalent to $\perp$, as required.
2. Case 2: if z_n contains spiders but no shading in d_n then $\neg d_n$ is logically equivalent to $\bigvee\limits_{1 \le i < n} d_i$. Suppose that I models $\neg d_n$. We show I models $\bigvee\limits_{1 \le i < n} d_i$. Now, given I models $\neg d_n$, I does not model d_n. The only way I can fail to model d_n, since there is no shading and there are no missing zones, is if there are insufficient elements in $\zeta_I(z_n)$ to which the spiders can map injectively. From this it follows that $|\zeta_I(z_n)| < n$. Therefore, $|\zeta_I(z_n)| = i$ for some $i < n$.
   We show I models d_i. Since d_i has no missing zones, again we see that $\chi_I(Z(d_n)) = U$. Choose the i elements in U that are in $\zeta_I(z_n)$, say $u_1, ..., u_i$, and further define $\Sigma_{d_i}(s_j) = u_j$. Then, by construction, Σ_{d_i} is injective and maps spiders to elements in their habitat (recall, there are no other spiders in d_i). Furthermore, there is only one shaded zone, namely z_n, in d_i and we know $\zeta_I(z_n) = \{u_1, ..., u_i\}$. Therefore,
   $$\zeta_I(z_n) \subseteq \{u_1, ..., u_i\},$$
   as required. Hence, I models d_i and, consequently, I models $\bigvee\limits_{1 \le i < n} d_i$.
   For the converse, suppose that I models $\bigvee\limits_{1 \le i < n} d_i$. Then I models one of the disjuncts, say d_j. We show that I does not model d_n. Since I models d_j, there is a spider map, Σ_{d_j}, which ensures that $\zeta_I(z_n) \subseteq \{u_1, ..., u_j\}$, where $u_1, ..., u_j$ are the elements mapped to by the j spiders in z_n in d_j. From this, it follows that $\zeta_I(z_n) < n$, since $j < n$. Therefore, as there are n spiders in z_n in d_n, there cannot exist an injective mapping of spiders in d_n, namely Σ_{d_n}, which ensures that all spiders represents elements in $\zeta_I(z_n)$. Hence I cannot model d_n. Therefore I models $\neg d_n$. Thus, we see that $\neg d_n$ is logically equivalent to $\bigvee\limits_{1 \le i < n} d_i$, as required.

3. Case 3: z_n contains spiders and shading in d_n and we have that $\neg d_n$ is logically equivalent to $\bigvee\limits_{1 \leq i < n} d_i \vee d_{n+1}$.
   Suppose that I models $\neg d_n$. We show I models $\bigvee\limits_{1 \leq i < n} d_i \vee d_{n+1}$. Now, given I models $\neg d_n$, I does not model d_n. The only way I can fail to be a model for d_n is if there are not exactly n elements in $\zeta_I(z_n)$. From this it follows that $|\zeta_I(z_n)| < n$ or $|\zeta_I(z_n)| > n$. Therefore, $|\zeta_I(z_n)| = i$ for some $i < n$ or $i > n$. In the former case, we have I models d_i for some $i < n$, as in Case 2. When $i > n$, choose $n + 1$ elements, say $u_1, ..., u_{n+1}$, in $\zeta_I(z_n)$, define $\Sigma_{d_{n+1}}$ by $\Sigma(s_j) = u_j$ and one can readily proceed to show I models d_{n+1} in much the same way, noting the details are more straightforward since z_n is not shaded in d_{n+1}. Therefore I models $\bigvee\limits_{1 \leq i < n} d_i \vee d_{n+1}$.
   For the converse, suppose I models $\bigvee\limits_{1 \leq i < n} d_i \vee d_{n+1}$. Then I models d_i for some $1 \leq i < n$ or I models d_{n+1}. If I models such a d_i then to show I models $\neg d_n$ the proof proceeds similarly to case 2. If I models d_{n+1} then it can readily be shown that there are at least $n + 1$ elements, say $u_1, ..., u_n, u_{n+1}$, in $\zeta_I(z_n)$. But then I does not model d_n, since d_n requires $|\zeta_I(z_n)| = n$,. Therefore, I models $\neg d_n$. Thus, we see that $\neg d_n$ is logically equivalent to $\bigvee\limits_{1 \leq i < n} d_i \vee d_{n+1}$, as required.

Hence NegationElimination is sound.

Recall that the CopyContours inference rule applies to $d_1 \wedge d_2$, copying a contour l_2 from d_2 into d_1, yielding $d_1' \wedge d_2$. In order to formalise CopyContours, we need to specify syntactically how the addition of the new contour, l_2, impacts on the existing zones in d_1. Zones can either be completely inside, completely outside or split by the new contour, for which we require three parameters. These parameters will be defined using $Z_i(l_2, d_2)$, $Z_o(l_2, d_2)$, and $Z_s(l_2, d_2)$ which we will shortly define. Zones that are in $Z_i(l_2, d_2)$ will necessarily represent subsets of $\Phi(l_2)$ in models for $d_1 \wedge d_2$; these zones will be inside l_2 in d_1'. Similarly, zones that are in $Z_o(l_2, d_2)$ will necessarily represent sets disjoint from $\Phi(l_2)$ and will be outside l_2. If zones are neither necessarily subsets of nor disjoint from $\Phi(l_2)$ then they will be split into two new zones by l_2, one inside and the other outside l_2.

To give further insight into the definition below, we observe that in any model for d_2, the following hold:

1. $\Phi(l_2) = \chi_I(\{(in_2, out_2) \in Z(d_2) : l_2 \in in_2\})$, and
2. $\Phi(l_2) \cap \chi_I(\{(in_2, out_2) \in Z(d_2) : l_2 \in out_2\}) = \emptyset$.

Definition 16 Let d_1 and d_2 be unitary diagrams and let l_2 be in $L(d_2) \setminus L(d_1)$. We define three subsets of $Z(d_1) \setminus EZ(d_1)$, namely $Z_i(l_2, d_2)$, $Z_o(l_2, d_2)$, and $Z_s(l_2, d_2)$, according to the following rules: let $(in_1, out_1) \in Z(d_1) \setminus EZ(d_1)$ such that $\{(in_1, out_1)\} \not\subseteq_c EZ(d_2)$, then

1. $(in_1, out_1) \in Z_i(l_2, d_2)$ provided
$$\{(in_1, out_1)\} \subseteq_c \{(in_2, out_2) \in Z(d_2) : l_2 \in in_2\},$$
2. $(in_1, out_1) \in Z_o(l_2, d_2)$ provided
$$\{(in_1, out_1)\} \subseteq_c \{(in_2, out_2) \in Z(d_2) : l_2 \in out_2\},$$
3. $(in_1, out_1) \in Z_s(l_2, d_2)$ provided
$$(in_1, out_1) \notin Z_i \cup Z_o.$$

We now establish some properties of the sets $Z_i(l_2, d_2)$, $Z_o(l_2, d_2)$, and $Z_s(l_2, d_2)$.

Lemma 3 *Let d_1 and d_2 be unitary diagrams and let l_2 be in $L(d_2) \setminus L(d_1)$. Then*

1. *$Z_i(l_2, d_2)$, $Z_o(l_2, d_2)$, and $Z_s(l_2, d_2)$ are pairwise disjoint, and*
2. *$Z(d_1) \setminus (Z_i(l_2, d_2) \cup Z_o(l_2, d_2) \cup Z_s(l_2, d_2) \cup EZ(d_1)) \equiv_c EZ(d_2)$.*

Proof First we show that $Z_i(l_2, d_2)$, $Z_o(l_2, d_2)$, and $Z_s(l_2, d_2)$ are pairwise disjoint. Trivially, $Z_s(l_2, d_2)$ is disjoint from both $Z_i(l_2, d_2)$ and $Z_o(l_2, d_2)$. Let (in_1, out_1) be a zone in $Z_i(l_2, d_2)$. We must show that (in_1, out_1) is not in $Z_o(l_2, d_2)$. Since (in_1, out_1) is a zone in $Z_i(l_2, d_2)$ we know that

$$\{(in_1, out_1)\} \subseteq_c \{(in_2, out_2) \in Z(d_2) : l_2 \in in_2\}.$$

By the definition of $\subseteq_c$,

$$\begin{aligned}&exp(\{(in_1, out_1), L\}) \cup exp(EZ(d_1), L) \cup exp(EZ(d_2), L) \subseteq \\ &exp(\{(in_2, out_2) \in Z(d_2) : l_2 \in in_2\}, L) \cup exp(EZ(d_1), L) \cup exp(EZ(d_2), L) \qquad (1)\end{aligned}$$

where $L = L(d_1) \cup L(d_2)$. If (in_1, out_1) was in $Z_o(l_2, d_2)$ then we would also have

$$\begin{aligned}&exp(\{(in_1, out_1), L\}) \cup exp(EZ(d_1), L) \cup exp(EZ(d_2), L) \subseteq \\ &exp(\{(in_2, out_2) \in Z(d_2) : l_2 \in out_2\}, L) \cup exp(EZ(d_1), L) \cup exp(EZ(d_2), L) \qquad (2)\end{aligned}$$

We show that there is a zone in the LHS of (1) and (2), namely

$$exp(\{(in_1, out_1), L\}) \cup exp(EZ(d_1), L) \cup exp(EZ(d_2), L)$$

that is not in the RHS of (2), namely

$$exp(\{(in_2, out_2) \in Z(d_2) : l_2 \in out_2\}, L) \cup exp(EZ(d_1), L) \cup exp(EZ(d_2), L)$$

Since (in_1, out_1) is in Z_i, we know that

$$(in_1, out_1) \not\subseteq_c EZ(d_2)$$

This implies, by the definition of $\subseteq_c$, that

$$exp(\{(in_1, out_1), L\}) \cup exp(EZ(d_1), L) \cup exp(EZ(d_2), L) \not\subseteq exp(EZ(d_1), L) \cup exp(EZ(d_2), L).$$

Choose a zone, (in, out), such that

$$(in, out) \in exp(\{(in_1, out_1), L\}) \setminus (exp(EZ(d_1), L) \cup exp(EZ(d_2), L)).$$

If $l_2 \notin out$ then (in, out) is not in the RHS of (2) but it is in the LHS of (1) and we are done. Alternatively, $l_2 \in out$, in which case $l_2 \notin in$. But then (in, out) is not in the RHS of (1) but it is in the LHS of (1), which is a contradiction. Hence, the LHS of (1) is not a subset of the RHS of (2), as required. Therefore, (in_1, out_1) is not in $Z_o(l_2, d_2)$. Thus, the sets $Z_i(l_2, d_2)$ and $Z_o(l_2, d_2)$ are also disjoint, completing the first part of the proof.

For the last part of the proof we need to establish that

$$Z(d_1) \setminus (Z_i(l_2, d_2) \cup Z_o(l_2, d_2) \cup Z_s(l_2, d_2) \cup EZ(d_1)) \equiv_c EZ(d_2).$$

Let (in_1, out_1) be a zone such that

$$(in_1, out_1) \in Z(d_1) \setminus (Z_i(l_2, d_2) \cup Z_o(l_2, d_2) \cup Z_s(l_2, d_2) \cup EZ(d_1)).$$

Trivially, since (in_1, out_1) is not in any one of $Z_i(l_2, d_2)$, $Z_o(l_2, d_2)$ $Z_s(l_2, d_2)$, we see that

$$\{(in_1, out_1)\} \subseteq_c EZ(d_2).$$

From this the result immediately follows.

We now formalise the CopyContours inference rule.

Copy contours Let d_1 and d_2 be unitary diagrams and let l_2 be in $L(d_2) \setminus L(d_1)$. Let Z_{IN}, Z_{OUT} and Z_{SPLIT} be a 3-way partition of $Z(d_1)$ such that

1. $Z_i(l_2, d_2) \subseteq Z_{IN}$,

2. $Z_o(l_2, d_2) \subseteq Z_{OUT}$, and
3. $Z_s(l_2, d_2) \subseteq Z_{SPLIT}$.

Let d_1' be the diagram whose components are defined as follows:

1. the contour labels are $L(d_1') = L(d_1) \cup \{l_2\}$,
2. the zones are
$$\begin{aligned} Z(d_1') = &\{(in \cup \{l_2\}, out) : (in, out) \in Z_{IN} \cup Z_{SPLIT}\} \cup \\ &\{(in, out \cup \{l_2\}) : (in, out) \in Z_{OUT} \cup Z_{SPLIT}\}, \end{aligned}$$
3. the shaded zones are
$$\begin{aligned} Z(d_1') = &\{(in \cup \{l_2\}, out) : (in, out) \in (Z_{IN} \cup Z_{SPLIT}) \cap ShZ(d_1)\} \cup \\ &\{(in, out \cup \{l_2\}) : (in, out) \in (Z_{OUT} \cup Z_{SPLIT}) \cap ShZ(d_1)\}, \end{aligned}$$
4. the spiders are $S(d_1') = S(d_1)$, and
5. the habitat of each spider, $s' \in S(d_1')$, is
$$\begin{aligned} \eta_{d_1'}(s') = &\{(in \cup \{l_2\}, out) : (in, out) \in (Z_{IN} \cup Z_{SPLIT}) \cap \eta_{d_1}(s)\} \cup \\ &\{(in, out \cup \{l_2\}) : (in, out) \in (Z_{OUT} \cup Z_{SPLIT}) \cap \eta_{d_1}(s)\}. \end{aligned}$$

The CopyContours rule can be applied to show $d_1 \wedge d_2$ is logically equivalent to $d_1' \wedge d_2$.

Theorem 6 CopyContours *is sound.*

Proof Let d_1, d_2 and d_1' be spider diagrams, let l_2 be a contour label and let Z_{IN}, Z_{OUT} and Z_{SPLIT} be a three way partition of $Z(d_1)$ as in the definition of the CopyContours inference rule. We must show that $d_1 \wedge d_2 \equiv d_1' \wedge d_2$. Let $I = (U, \Phi)$ be an interpretation and suppose that I models $d_1' \wedge d_2$. Trivially, $d_1' \vDash d_1$, by Theorem 4, since d_1 can be obtained from d_1' by applying the EraseContour inference rule (deleting the contour labelled l_2 from d_1'). Therefore, $d_1' \wedge d_2 \vDash d_1 \wedge d_2$.

For the converse, suppose that I models $d_1 \wedge d_2$. We must first show that $\chi_I(Z(d_1')) = U$. Trivially, $\chi_I(Z(d_1')) \subseteq U$. Let $e \in U$. We show that there exists a zone, $(in_1', out_1') \in Z(d_1')$, such that $e \in \zeta_I(in_1', out_1')$. We know that
$$e \in \zeta_I((in_1, out_1))$$
for some zone $(in_1, out_1) \in Z(d_1)$, since $\chi_I(Z(d_1)) = U$. There are three cases to consider, relating to the three-way partition, Z_{IN}, Z_{OUT} and Z_{SPLIT}, of $Z(d_1)$.

1. Case 1: $(in_1, out_1) \in Z_{IN}$. We show that $e \in \zeta_I(in_1 \cup \{l_2\}, out_1)$. Since $(in_1, out_1) \in Z_{IN}$, we know, by Lemma 3, that either $(in_1, out_1) \in Z_i$ or $(in_1, out_1) \in EZ(d_1)$, or $(in_1, out_1) \subseteq_c EZ(d_2)$. In the latter two subcases, $\zeta_I(in_1, out_1) = \emptyset$, by lemma 1 and so does not contain e. Thus, it can only be that $(in_1, out_1) \in Z_i$. We, therefore, know that
$$\{(in_1, out_1)\} \subseteq_c \{(in_2, out_2) \in Z(d_2) : l_2 \in in_2\}$$
by the definition of Z_i. This implies that
$$e \in \zeta_I(in_1, out_1) \subseteq \chi_I(\{(in_2, out_2) \in Z(d_2) : l_2 \in in_2\}) \qquad (1)$$
by Theorem 3. Since all zones, (in_2', out_2'), in $\{(in_2, out_2) \in Z(d_2) : l_2 \in in_2\}$ have the property that $l_2 \in in_2'$ it follows that
$$\chi_I(\{(in_2, out_2) \in Z(d_2) : l_2 \in in_2\}) \subseteq \Phi(l_2).$$
By (1), we deduce that
$$e \in \zeta_I((in_1, out_1)) \subseteq \Phi(l_2).$$
Therefore
$$e \in \zeta_I((in_1, out_1)) \cap \Phi(l_2) \subseteq \Phi(l_2)$$

and we know that

$$\zeta_I((in_1, out_1)) \cap \Phi(l_2) = \zeta_I((in_1 \cup \{l_2\}, out_1)).$$

Hence

$$e \in \zeta_I((in_1 \cup \{l_2\}, out_1)).$$

By the definition of d_1', the zone $(in_1 \cup \{l_2\}, out_1)$ is in $Z(d_1')$.

2. Case 2: $(in_1, out_1) \in Z_{OUT}$. We show that $e \in (in_1, out_1 \cup \{l_2\})$. This case is similar to Case 1, noting that all zones, (in_2', out_2'), in $\{(in_2, out_2) \in Z(d_2) : l_2 \in out_2\}$ have the property that $l_2 \in out_2'$. From this, it follows that

$$\chi_I(\{(in_2, out_2) \in Z(d_2) : l_2 \in out_2\}) \subseteq U \setminus \Phi(l_2).$$

3. Case 3: $(in_1, out_1) \in Z_{SPILT}$. Trivially,

$$e \in \zeta_I((in_1 \cup \{l_2\}, out_1)) \cup \zeta_I((in_1, out_1 \cup \{l_2\}))$$

and both the zones $(in_1 \cup \{l_2\}, out_1)$ and $(in_1, out_1) \cup \{l_2\})$ are in $Z(d_1')$ and we are done.

Hence for every element, e, in U there exists a zone, (in_1', out_1') in $Z(d_1')$ such that $e \in \zeta_I((in_1', out_1'))$. Thus $\chi_I(Z(d_1)) = U$, as required. That is, between them the zones of d_1' represent the universal set.

We must now show that the condition for I to model d_1' relating to spiders holds. For d_1 there exists a function, $\Sigma_{d_1} : S(d_1) \to U$, such that

1. $\forall s \in S(d_1)\Big(\Sigma_{d_1}(s) \in \chi_I(\eta_{d_1}(s))\Big)$, and
2. $\forall z \in ShZ(d_1)\Big(\zeta_I(z_n) \subseteq im(\Sigma_{d_1})\Big)$.

Choose such a Σ_{d_1}. We must show that a similar $\Sigma_{d_1'} : S(d_1') \to U$ exists for d_1'. We define $\Sigma_{d_1'} = \Sigma_{d_1}$.

We now show that $\Sigma_{d_1'}$ ensures that the spiders map to elements in the sets represented by their habitats. Let s' be a spider in $S(d_1')$. Then, by the definition of d_1',

$$\begin{aligned}\eta_{d_1'}(s') = \{&(in \cup \{l_2\}, out) : (in, out) \in (Z_{IN} \cup Z_{SPLIT}) \cap \eta_{d_1}(s)\} \cup \\ &\{(in, out \cup \{l_2\}) : (in, out) \in (Z_{OUT} \cup Z_{SPLIT}) \cap \eta_{d_1}(s\}.\end{aligned}$$

We know that $\Sigma_{d_1}(s) \in \chi_I(\eta_{d_1}(s))$. Choose the zone $(in_1, out_1) \in \eta_{d_1}(s)$ such that

$$\Sigma_{d_1}(s) \in \zeta_I((in_1, out_1)).$$

Then $(in_1, out_1) \notin EZ(d_1)$ and $(in_1, out_1) \not\subseteq_c EZ(d_2)$. This implies that either $(in_1, out_1) \in Z_i$, $(in_1, out_1) \in Z_o$, or $(in_1, out_1) \in Z_s$. Similarly to previous parts of the proof, we make the following three deductions.

1. If $(in_1, out_1) \in Z_i$ then

$$\Sigma_{d_1}(s) \in \zeta_I((in_1, out_1)) = \zeta_I((in_1 \cup \{l_2\}, out_1)).$$

The zone $(in_1 \cup \{l_2\}, out_1)$ is in $\eta_{d_1'}(s)$ and, since $\Sigma_{d_1}(s) = \Sigma_{d_1'}(s)$, we have

$$\Sigma_{d_1'}(s) \in \zeta_I((in_1 \cup \{l_2\}, out_1)).$$

2. If $(in_1, out_1) \in Z_o$ then

$$\Sigma_{d_1}(s) \in \zeta_I((in_1, out_1)) = \zeta_I((in_1, out_1 \cup \{l_2\})).$$

The zone $(in_1, out_1 \cup \{l_2\})$ is in $\eta_{d_1'}(s)$ and, since $\Sigma_{d_1}(s) = \Sigma_{d_1'}(s)$, we have

$$\Sigma_{d_1'}(s) \in \zeta_I((in_1, out_1 \cup \{l_2\})).$$

3. If $(in_1, out_1) \in Z_s$ then
$$\Sigma_{d_1}(s) \in \zeta_I((in_1, out_1)) = \zeta_I((in_1 \cup \{l_2\}, out_1)) \cup \zeta_I((in_1, out_1 \cup \{l_2\})).$$
The zones $(in_1 \cup \{l_2\}, out_1)$ and $(in_1, out_1 \cup \{l_2\})$ are both in $\eta_{d_1'}(s)$ and, since $\Sigma_{d_1}(s) = \Sigma_{d_1'}(s)$, we have
$$\Sigma_{d_1'}(s) \in \zeta_I((in_1 \cup \{l_2\}, out_1)) \cup \zeta_I((in_1, out_1 \cup \{l_2\})).$$

In all three cases, we have shown that $\Sigma_{d_1'}(s) \in \chi_I(\eta_{d_1'}(s))$, as required. That is, each spider in d_1' represents an element in the set represented by its habitat in d_1'.

Finally, we consider the shaded zones. Let z be a shaded zone in d_1', in which case
$$\begin{aligned} z \in \{&(in \cup \{l_2\}, out) : (in, out) \in (Z_{IN} \cup Z_{SPLIT}) \cap ShZ(d_1)\} \cup \\ &\{(in, out \cup \{l_2\}) : (in, out) \in (Z_{OUT} \cup Z_{SPLIT}) \cap ShZ(d_1)\}. \end{aligned}$$

Therefore, given that $z = (in \cup \{l_2\}, out)$ or $(in, out \cup \{l_2\}\}$ for some zone $(in, out) \in ShZ(d_1)$, we know that
$$\zeta_I(z) \subseteq im(\Sigma_{d_1}) = im(\Sigma_{d_1'})$$
since
$$\zeta_I(z) \subseteq \zeta_I((in, out)) \subseteq im(\Sigma_{d_1}).$$
Therefore, for all shaded zones, z, in d_1', $\zeta_I(z) \subseteq im(\Sigma_{d_1'})$ as required. That is, each shaded zone in d_1' represents a set containing only elements represented by spiders. Hence I is a model for d_1'. Since, by assumption, I models d_2 it follows that I models $d_1' \wedge d_2$. Thus, $d_1 \wedge d_2 \vDash d_1' \wedge d_2$. Hence $d_1 \wedge d_2 \equiv d_1' \wedge d_2$, that is, copyContour is sound.

We now formalise CopyShading and prove that it is sound. To formalise the rule, we need to identify spiders whose habitats have certain properties, given a diagram $d_1 \wedge d_2$. In particular, these spiders are in d_1 and have a foot in a particular region, say r_1. Moreover, all zones of the habitat outside of r_1 represent empty sets, which can be deduced from d_2.

Definition 17 Let d_1 and d_2 be unitary diagrams. We define
$$S(r_1, d_1, d_2) = \{s \in S(d_1) : \eta_{d_1}(s) \cap r_1 \neq \emptyset \wedge \eta_{d_1}(s) \setminus r_1 \subseteq_c EZ(d_2)\}.$$

Copy Shading Let d_1 and d_2 be unitary diagrams with regions, r_1 and r_2 respectively, such that:

1. $r_1 \equiv_c r_2$,
2. r_1 contains at least one non-shaded zone in d_1, that is $r_1 \setminus ShZ(d_1) \neq \emptyset$,
3. r_2 is entirely shaded in d_2, that is, $r_2 \subseteq ShZ(d_2)$,
4. in d_1, each spider, s, whose habitat includes a zone of r_1, that is, $\eta_{d_1}(s) \cap r_1 \neq \emptyset$, is also in $S(r_1, d_1, d_2)$,
5. in d_2, each spider, s, whose habitat includes a zone of r_2, that is, $\eta_{d_2}(s) \cap r_2 \neq \emptyset$, is also in $S(r_2, d_2, d_1)$, and
6. there is a bijection, $\sigma\colon S(r_1, d_1, d_2) \to S(r_2, d_2, d_1)$ such that for each spider, s, $\eta_{d_1}(s) \equiv_c \eta_{d_2}(\sigma(s))$.

Let d_1' be the diagram whose components are defined as follows:

1. the contour labels are $L(d_1') = L(d_1)$,
2. the zones are $Z(d_1') = Z(d_1)$,
3. the shaded zones are $ShZ(d_1') = ShZ(d_1) \cup r_1$,
4. the spiders are $S(d_1') = S(d_1)$,
5. the habitat of each spider, s, in $S(d_1')$ is $\eta_{d_1'}(s) = \eta_{d_1}(s)$.

The CopyShading rule can be applied to show $d_1 \wedge d_2$ is logically equivalent to $d_1' \wedge d_2$.

Theorem 7 CopyShading *is sound.*

Proof Let d_1, d_2 and d_1' be spider diagrams and let r_1 and r_2 be regions as in the definition of the CopyShading inference rule. We must show that $d_1 \wedge d_2 \equiv d_1' \wedge d_2'$. Let $I = (U, \Phi)$ be an interpretation and suppose that I models $d_1' \wedge d_2$. Trivially, $d_1' \vDash d_1$, by Theorem 4, since d_1 can be obtained from d_1' by applying the RemoveShading inference rule (deleting the shading from the region $r_1 \setminus ShZ(d_1)$). Therefore, $d_1' \wedge d_2 \vDash d_1 \wedge d_2$.

For the converse, suppose that I models $d_1 \wedge d_2$. First, since $Z(d_1') = Z(d_1)$ and since I models d_1, we immediately see that $\chi_I(Z(d_1')) = U$, because $\chi_I(Z(d_1)) = U$, as required. That is, between them the zones of d_1' represent the universal set.

We must now show that the condition for I to model d_1' relating to spiders holds. For d_1 there exists a function, $\Sigma_{d_1} : S(d_1) \to U$, such that

1. $\forall s \in S(d_1)\Big(\Sigma_{d_1}(s) \in \chi_I(\eta_{d_1}(s))\Big)$, and
2. $\forall z \in ShZ(d_1)\Big(\zeta_I(z) \subseteq im(\Sigma_{d_1})\Big)$.

Choose such a Σ_{d_1}. Similarly, choose such a Σ_{d_2} for d_2. We must show that a similar $\Sigma_{d_1'} : S(d_1') \to U$ exists for d_1'. We define $\Sigma_{d_1} : S(d_1') \to U$ by

$$\Sigma_{d_1'}(s) = \begin{cases} \Sigma_{d_1}(s) & \text{if } s \in S(d_1) \setminus S(r_1, d_1, d_2) \\ \Sigma_{d_2}(\sigma(s)) & \text{otherwise.} \end{cases}$$

Our first obligation is to show that $\Sigma_{d_1'}$ is injective. Clearly, $\Sigma_{d_1'}|_{S(d_1)\setminus S(r_1,d_1,d_2)}$ and $\Sigma_{d_1'}|_{S(r_1,d_1,d_2)}$ are both injective, since Σ_{d_1} and Σ_{d_2}, respectively, are injective. Let $s_1 \in S(d_1) \setminus S(r_1, d_1, d_2)$ and let $s_2 \in S(r_1, d_1, d_2)$ and suppose that $\Sigma_{d_1'}(s_1) = \Sigma_{d_1'}(s_2)$. Since

$$\Sigma_{d_1'}(s_1) = \Sigma_{d_1}(s_1) \in \chi_I(\eta_{d_1}(s_1)) = \chi_I(\eta_{d_1'}(s_1)) \qquad (\text{because } \eta_{d_1}(s_1) = \eta_{d_1'}(s_1))$$

and

$$\Sigma_{d_1'}(s_2) = \Sigma_{d_2}(\sigma(s_2))) \in \chi_I(\eta_{d_2}(\sigma(s_2))) = \chi_I(\eta_{d_1'}(s_2)) \ \ (\text{because } \eta_{d_2}(\sigma(s_2)) \equiv_c \eta_{d_1'}(s_2)),$$

we know that

$$\chi_I(\eta_{d_1'}(s_1)) \cap \chi_I(\eta_{d_1'}(s_2)) \neq \emptyset.$$

Since distinct zones in any unitary diagram represent disjoint sets, it follows that

$$\eta_{d_1'}(s_1)) \cap \eta_{d_1'}(s_2) \neq \emptyset,$$

that is, the spiders s_1 and s_2 have a common zone, z say, in their habitats in d_1'. Moreover, $s_2 \in S(r_1, d_1, d_2)$ implies $\chi_I(\eta_{d_1}(s_2) \setminus r_1) = \emptyset$, by Lemma 2 which, in turn, implies that $z \in r_1$. Therefore, since $z \in \eta_{d_1'}(s_1) = \eta_{d_1}(s_1)$, we see that s_1 is a spider in d_1 whose habitat includes a zone r_1. Hence $s_1 \in S(r_1, d_1, d_2)$, contradicting our assumption that $s_1 \in S(d_1) \setminus S(r_1, d_1, d_2)$. Hence $\Sigma_{d_1'}(s_1) \neq \Sigma_{d_1'}(s_2)$, so $\Sigma_{d_1'}$ is injective.

We now show that $\Sigma_{d_1'}$ ensures that the spiders map to elements in the sets represented by their habitats. Let s be a spider in $S(d_1')$. Then, by the definition of d_1', $\eta_{d_1'}(s) = \eta_d(s)$. If $s \in S(d_1) \setminus S(r_1, d_1, d_2)$ then $\Sigma_{d_1'}(s) = \Sigma_{d_1}(s) \in \chi_I(\eta_d(s)) = \chi_I(\eta_{d_1'}(s))$, as required. Otherwise, $s \in S(r_1, d_1, d_2)$. In this case,

$$\Sigma_{d_1'}(s) = \Sigma_{d_2}(\sigma(s)) \in \chi_I(\eta_{d_2}(\sigma(s))).$$

Since $\eta_{d_2}(\sigma(s)) \equiv_c \eta_{d_1}(s) = \eta_{d_1'}(s)$, by Theorem 3 we deduce

$$\chi_I(\eta_{d_2}(\sigma(s))) = \chi_I(\eta_{d_1}(s)) = \chi_I(\eta_{d_1'}(s)).$$

Hence

$$\Sigma_{d_1'}(s) \in \chi_I(\eta_{d_1'}(s)),$$

as required. That is, each spider in d_1' represents an element in the set represented by its habitat in d_1'.

Finally, we consider the shaded zones. Let z be a shaded zone in d_1'. There are two cases: $z \in ShZ(d_1') \setminus r_1$ and $z \in ShZ(d_1') \cap r_1$. If $z \in ShZ(d_1') \setminus r_1$ then $z \in ShZ(d_1)$, so

$$\zeta_I(z) \subseteq im(\Sigma_{d_1}) \qquad (1)$$

Let $e \in \zeta_I(z)$. We show that e is represented by a spider in $S(d_1)$ that is not in $S(r_1, d_1, d_2)$. Choose the spider, s, in $S(d_1)$ such that $\Sigma_{d_1}(s) = e$ (such a spider exists by (1)). Then $\Sigma_{d_1}(s) \in \chi_I(\eta_{d_1}(s))$ which implies that $z \in \eta_{d_1}(s)$. Since $z \notin r_1$, there is a zone in the habitat of s that is not in r_1. This implies that $s \notin S(r_1, d_1, d_2)$ because all of the spiders whose habitats includes a zone of r_1 represent elements in $\chi_I(r_1)$ (from the fact that for all $s' \in S(r_1, d_1, d_2)$, $\chi_I(\eta_{d_1}(s) \setminus r_1) = \emptyset$ in models for $d_1 \wedge d_2$). Since e was an arbitrary element in $\zeta_I(z)$ it follows that

$$\zeta_I(z) \subseteq im(\Sigma_{d_1}) \setminus \{\Sigma_{d_1}(s_1) : s_1 \in S(r_1, d_1, d_2)\} \subseteq im(\Sigma_{d_1'})$$

as required.

For the second case, $z \in ShZ(d_1') \cap r_1$. We show that $\zeta_I(z) \subseteq im(\Sigma_{d_1'}) \cap im(\Sigma_{d_2})$. Let $e \in \zeta_I(z)$. We show that e is represented by a spider in $S(d_2)$ that is also in $S(r_2, d_2, d_1)$. Choose the spider, s, in $S(d_2)$ such that $\Sigma_{d_2}(s) = e$; such a spider exists because $\zeta_I(z) \subseteq \chi_I(r_1) = \chi_I(r_2) \subseteq im(\Sigma_{d_2})$, by Theorem 3, since $r_1 \equiv_c r_2$. In particular, we see that

$$\Sigma_{d_2}(s) \in \chi_I(r_2).$$

Furthermore, we know that

$$\Sigma_{d_2}(s) \in \chi_I(\eta_{d_2}(s))$$

implying that $\chi_I(r_2) \cap \chi_I(\eta_{d_2}(s)) \neq \emptyset$. Since distinct zones in a unitary diagram represent disjoint sets, we deduce that $r_2 \cap \eta_{d_2}(s) \neq \emptyset$. That is, the habitat of s in d_2 includes a zone of r_2. Therefore, $s \in S(r_2, d_2, d_1)$, as required. From this, since e was an arbitrary element in $\zeta_I(z)$, it follows that

$$\zeta_I(z) \subseteq \{\Sigma_{d_2}(s_2)) : s_2 \in S(r_2, d_2, d_1)\} \qquad (2)$$

Since $\sigma\colon S(r_1, d_2, d_1) \to S(r_2, d_2, d_1)$ is a bijection and, for all spiders, $s \in S(r_1, d_2, d_1)$, $\Sigma_{d_1'}(s) = \Sigma_{d_2}(\sigma(s))$ we then see that

$$\{\Sigma_{d_2}(s_2) : s_2 \in S(r_2, d_2, d_1)\} = im(\Sigma_{d_1'}) \cap im(\Sigma_{d_2}).$$

Hence, by (2), $\zeta_I(z) \subseteq im(\Sigma_{d_1'})$. Therefore, for all shaded zones, z, in d_1', $\zeta_I(z) \subseteq im(\Sigma_{d_1'})$ as required. That is, each shaded zone in d_1' represents a set containing only elements represented by spiders. Hence I is a model for d_1'. Since, by assumption, I models d_2 it follows that I models $d_1' \wedge d_2$. Thus, $d_1 \wedge d_2 \vDash d_1' \wedge d_2$. Hence $d_1 \wedge d_2 \equiv d_1' \wedge d_2$, that is, CopyShading is sound.

Lastly, we formalise the CopySpider inference rule and establish its soundness.

Copy a spider Let d_1 and d_2 be unitary diagrams with regions r_1 and r_2 respectively, such that:

1. $r_1 \equiv_c r_2$,
2. r_1 contains no shaded zones in d_1, that is, $r_1 \cap ShZ(d_1) = \emptyset$,
3. in d_1, each spider, s, whose habitat includes a zone of r_1, that is, $\eta_{d_1}(s) \cap r_1 \neq \emptyset$, is also in $S(r_1, d_1, d_2)$,
4. there exists an injective, but not surjective, function $\sigma\colon S(r_1, d_1, d_2) \to S(r_2, d_2, d_1)$ such that
   (a) for each spider s, $\eta_{d_1}(s) \equiv_c \eta_{d_2}(\sigma(s))$, and
   (b) there exists a spider, s_2, that is in $S(r_2, d_2, d_1)$ but is not mapped to by σ, such that $\eta_{d_2}(s_2) \subseteq_c r_1$.

Let s_1 be a fresh spider. Let d_1' be the diagram whose components are defined as follows:

1. the contour labels are $L(d_1') = L(d_1)$,
2. the zones are $Z(d_1') = Z(d_1)$,
3. the shaded zones are $ShZ(d_1') = ShZ(d_1)$,
4. the spiders are $S(d_1') = S(d_1) \cup \{s_1\}$,
5. the habitat of each spider, s', in $S(d_1')$ is

$$\eta_{d_1'}(s') = \begin{cases} \eta_{d_1}(s') & \text{if } s' \in S(d_1) \\ r_1 & \text{otherwise.} \end{cases}$$

The CopySpider rule can be applied to show $d_1 \wedge d_2$ is logically equivalent to $d_1' \wedge d_2$.

Theorem 8 CopySpider *is sound.*

Proof Let d_1, d_2 and d_1' be spider diagrams, and let r_1 and r_2 be regions, and let s_1 be a spider as in the definition of the CopySpider inference rule. We must show that $d_1 \wedge d_2 \equiv d_1' \wedge d_2'$. Let $I = (U, \Phi)$ be an interpretation and suppose that I models $d_1' \wedge d_2$. Trivially, $d_1' \vDash d_1$, by Theorem 4, since d_1 can be obtained from d_1' by applying the EraseSpider inference rule (deleting the spider s_1, since its habitat is r_1 and this region contains no shaded zones). Therefore, $d_1' \wedge d_2 \vDash d_1 \wedge d_2$.

For the converse, suppose that I models $d_1 \wedge d_2$. First, since $Z(d_1') = Z(d_1)$ and since I models d_1, we immediately see that $\chi_I(Z(d_1')) = U$, because $\chi_I(Z(d_1)) = U$, as required. That is, between them the zones of d_1' represent the universal set.

We must now show that the condition for I to model d_1' relating to spiders holds. For d_1 there exists a function, $\Sigma_{d_1} : S(d_1) \to U$, such that

1. $\forall s \in S(d_1)\Big(\Sigma_{d_1}(s) \in \chi_I(\eta_{d_1}(s))\Big)$, and
2. $\forall z \in ShZ(d_1)\Big(\zeta_I(z) \subseteq im(\Sigma_{d_1})\Big)$.

Choose such a Σ_{d_1}. Similarly, choose such a Σ_{d_2} for d_2. We must show that a similar $\Sigma_{d_1'} : S(d_1') \to U$ exists for d_1'. Now, since $\sigma\colon S(r_1, d_1, d_2) \to S(r_2, d_2, d_1)$ ensures that there exists a spider, s_2, that is in $S(r_2, d_2, d_1)$ but is not mapped to by σ where $\eta_{d_2}(s_2) \subseteq_c r_1$, we extend σ to $\sigma\colon S(r_1, d_1', d_2) \to S(r_2, d_2, d_1')$ by defining $\sigma(s_1) = s_2$ (noting that $S(r_1, d_1', d_2) = S(r_1, d_1', d_2) \cup \{s_1\}$). We define $\Sigma_{d_1'} : S(d_1') \to U$ by

$$\Sigma_{d_1'}(s) = \begin{cases} \Sigma_{d_1}(s) & \text{if } s \in S(d_1) \setminus S(r_1, d_1, d_2) \\ \Sigma_{d_2}(\sigma(s)) & \text{otherwise.} \end{cases}$$

Our first obligation is to show that $\Sigma_{d_1'}$ is injective. Clearly, $\Sigma_{d_1'}|_{S(d_1)\setminus S(r_1,d_1,d_2)}$ and $\Sigma_{d_1'}|_{S(r_1,d_1,d_2)\cup\{s_1\}}$ are both injective, since Σ_{d_1} and Σ_{d_2}, respectively, are injective. Let $s_1' \in S(d_1) \setminus S(r_1, d_1, d_2)$ and let $s_2' \in S(r_1, d_1, d_1) \cup \{s_1\} = S(r_1, d_1', d_2)$ and suppose that $\Sigma_{d_1'}(s_1') = \Sigma_{d_1'}(s_2')$. Since

$$\Sigma_{d_1'}(s_1') \in \chi_I(\eta_{d_1}(s_1')) = \chi_I(\eta_{d_1'}(s_1')) \qquad \text{(because } \eta_{d_1}(s_1') = \eta_{d_1'}(s_1')\text{)}$$

and

$$\Sigma_{d_1'}(s_2') = \Sigma_{d_2}(s_2') \in \chi_I(\eta_{d_2}(\sigma(s_2'))) \subseteq \chi_I(\eta_{d_1'}(s_2')) \qquad \text{(because } \eta_{d_2}(\sigma(s_2')) \subseteq_c \eta_{d_1'}(s_2')\text{)}$$

we know that

$$\chi_I(\eta_{d_1'}(s_1')) \cap \chi_I(\eta_{d_1'}(s_2')) \neq \emptyset.$$

Since distinct zones in any unitary diagram represent disjoint sets, it follows that

$$\eta_{d_1'}(s_1')) \cap \eta_{d_1'}(s_2') \neq \emptyset,$$

that is, the spiders s_1' and s_2' have a common zone, z say, in their habitats in d_1' that represents a non-empty set. By definition, the only zones of $\eta_{d_1'}(s_2')$ that represent non-empty sets are in r_1. But then s_1' would be a spider in d_1 that includes a zone, z, of r_1 but is not in $S(r_1, d_2)$, which is a contradiction. Hence $\Sigma_{d_1'}(s_1') \neq \Sigma_{d_1'}(s_2')$, so $\Sigma_{d_1'}$ is injective.

We now show that $\Sigma_{d_1'}$ ensures that the spiders map to elements in the sets represented by their habitats. Let s' be a spider in $S(d_1')$. Then, by the definition of d_1', $\eta_{d_1'}(s') = \eta_d(s')$ or, when $s' = s_1$, $\eta_{d_1'}(s') = r_1$ (in this latter case, $s' \in S(r_1, d_1', d_2)$). If $s' \in S(d_1) \setminus S(r_1, d_1, d_2)$ then $\Sigma_{d_1'}(s') = \Sigma_{d_1}(s') \in \chi_I(\eta_{d_1}(s')) = \chi_I(\eta_{d_1'}(s'))$, as required. Otherwise, $s' \in S(r_1, d_1, d_2) \cup \{s_1\} = S(r_1, d_1', d_2)$. In this case,

$$\Sigma_{d_1'}(s') = \Sigma_{d_2}(s') \in \chi_I(\eta_{d_2}(\sigma(s'))).$$

Since, when $s' \neq s_1$, $\eta_{d_2}(\sigma(s')) \equiv_c \eta_{d_1}(s') = \eta_{d_1'}(s')$ and, when $s' = s_1$, $\eta_{d_2}(\sigma(s')) \subseteq_c r_1 = \eta_{d_1'}(s')$, by Theorem 3 we deduce

$$\chi_I(\eta_{d_2}(\sigma(s'))) \subseteq \chi_I(\eta_{d_1'}(s')).$$

Hence

$$\Sigma_{d_1'}(s') \in \chi_I(\eta_{d_1'}(s)),$$

as required. That is, each spider in d_1' represents an element in the set represented by its habitat in d_1'.

Finally, we consider the shaded zones. Let z be a shaded zone in d_1', in which case z is shaded in d_1. Let $e \in \zeta_I(z)$. We show that there is a spider, s, in d_1' that maps to e. Now, since z is shaded, $z \notin r_1$, by the definition of the inference rule. Then no spider in d_1 whose habitat includes a zone of r_1 maps to e. This is because any spider, s', whose habitat includes a zone of r_1, is in $S(r_1, d_1, d_2)$ and, thus, all zones in $\eta_{d_1}(s) \setminus r_1$ represent empty sets. Therefore, any spider that maps to e cannot include zones of r_1 in its habitat. Since z is shaded, there exists a spider s that maps to e. Therefore, s is not in $S(r_1, d_1, d_2)$, so $\Sigma_{d_1'}(s) = \Sigma_{d_1}(s)$. Since e is an arbitrary element in $\zeta_I(z)$ we deduce that $\zeta_I(z) \subseteq im(\Sigma_{d_1'})$. That is, each shaded zone in d_1' represents a set containing only elements represented by spiders. Hence I is a model for d_1'. Since, by assumption, I models d_2 it follows that I models $d_1' \wedge d_2$. Thus, $d_1 \wedge d_2 \vDash d_1' \wedge d_2$. Hence $d_1 \wedge d_2 \equiv d_1' \wedge d_2$, so CopySpider is sound.

C Proof of completeness

We now establish that the spider diagram logic, extended to include implication, bi-implication and negation, is complete. To achieve completeness, we added new logical rules for these connectives along with NegationElimination; the other new rules we introduced for constructing more readable proofs and whose soundness we just proved in Appendix B are not necessary for completeness. To prove completeness, we extend the proof given for spider diagrams in [12], which relies on the absence of $\longrightarrow$, $\longleftrightarrow$ and $\neg$. If we can establish that every spider diagram is syntactically equivalent to a diagram with no occurrences of $\longrightarrow$, $\longleftrightarrow$ and $\neg$ then we have established completeness for the extended spider diagram system implemented in Speedith.

It is trivial to eliminate $\longrightarrow$ and $\longleftrightarrow$ using standard logical inference rules. We now show how to eliminate negation. If we have a negated unitary diagram where all spiders have single feet then it is possible to eliminate the negation using three rules: IntroShadedZone, Combining and NegationElimination. The NegationElimination inference rule can only be applied to diagrams with information about at most one zone. Thus, it is useful to define this property:

Definition 18 Let d be a unitary diagram where all spiders have a single foot. Then d is in **zone-minimal** form if it has no missing zones and all zones except, perhaps a single zone, do not contain any spiders or shading.

Lemma 4 *Let d be a unitary diagram with zone set $Z = \{z_1, ..., z_n\}$ such that all spiders have single feet. Then d is syntactically equivalent to*

$$\bigwedge_{1 \leq i \leq n} d_i$$

where d_i is zone-minimal and has

1. *zone set Z, together with any missing zones,*
2. *the same number of spiders in z_i as in d,*
3. *shading in z_i, provided z_i was shaded in d,*
4. *no other spiders or shading.*

Proof We start by adding all missing zones to d, using IntroShadedZone. Noting that the Combining rule is an equivalence, it can be applied to turn d into $\bigwedge_{1\leq i\leq n} d_i$, as follows. First, turn d into $d_1 \wedge d'$, where d' is a copy of d except that it contains no spiders or shading in z_1. Repeat this process, iterating through all of the zones, to give $\bigwedge_{1\leq i\leq n} d_i$. Thus, d is syntactically equivalent to $\bigwedge_{1\leq i\leq n} d_i$.

Theorem 9 *Let d be a spider diagram. Then there exists a syntactically equivalent spider diagram, d', where d' does not contain any of $\longrightarrow$, $\longleftrightarrow$, and $\neg$.*

Proof We begin by using the logical inference rules to eliminate $\longrightarrow$, $\longleftrightarrow$. Next, apply SplitSpider until all spiders have single feet. Then replace each non-$\bot$ unitary part of the resulting diagram with $\bigwedge_{1\leq i\leq n} d_i$ as in Lemma 4. To eliminate negation, the next step is to push all negation symbols to the leaves, using standard logical inference rules. Since all non-$\bot$ unitary parts are zone-minimal and all spiders have single feet, the NegationElimination rule can be applied to eliminate all negation symbols. The resulting diagram does not contain any of $\longrightarrow$, $\longleftrightarrow$, and $\neg$. Since all rules used are equivalences, this completes the proof.

Since all diagrams can be reduced to syntactically equivalent diagrams without using any of $\longrightarrow$, $\longleftrightarrow$, and $\neg$, we can then use the completeness theorem from [12] to establish completeness for this extended system.

Theorem 10 (Completeness) *Let d_1 and d_2 be spider diagrams such that $d_1 \vDash d_2$. Then $d_1 \vdash d_2$.*

Proof Suppose that $d_1 \vDash d_2$. By Theorem 9, there exists d_1' and d_2' that are syntactically equivalent to d_1 and d_2 respectively, where d_1' and d_2' do not contain any of $\longrightarrow$, $\longleftrightarrow$, and $\neg$. Since the spider diagram logic in [12] did not include $\longrightarrow$, $\longleftrightarrow$, and $\neg$ and is complete, we have established completeness for our extended spider diagram logic.

References

1. Bachmair, L., Ganzinger, H., Waldmann, U.: Set Constraints are the Monadic Class (1992)
2. Bashford-Chuchla, C.T.: Pen input interface for a diagrammatic theorem prover. Mphil dissertation, University of Cambridge Computer Laboratory, Cambridge, UK (2014)
3. Chang, S.H., Blagojevic, R., Plimmer, B.: Rata.gesture: A gesture recognizer developed using data mining. Artificial Ingelligence for Engineering Design, Analysis and Manufacturing **26**(3), 351–366 (2012)
4. Damm, C., Hansen, K., Thomsen, M.: Tool support for cooperative object-oriented design: Gesture based modelling on an electronic whiteboard. In: SIGCHI conference on Human factors in computing systems, pp. 518–525. ACM (2000)
5. Dau, F.: Constants and functions in peirce's existential graphs. In: ICCS, *LNCS*, vol. 4604, pp. 429–442. Springer (2007)
6. De Chiara, R., Hammar, M., Scarano, V.: A System for Virtual Directories Using Euler Diagrams. ENTCS **134**, 33–53 (2005)

7. Flower, J., Masthoff, J., Stapleton, G.: Generating Readable Proofs: A Heuristic Approach to Theorem Proving With Spider Diagrams. In: A. Blackwell, K. Marriott, A. Shimojima (eds.) Diagrammatic Representation and Inference, *Lecture Notes in Computer Science*, vol. 2980, pp. 166–181. Springer (2004). DOI 10.1007/978-3-540-25931-2_17
8. Gordon, M.J., Milner, A.J., Wadsworth, C.P.: Edinburgh LCF: A Mechanised Logic of Computation, *Lecture Notes in Computer Science*, vol. 78. Springer (1979). DOI 10.1007/3-540-09724-4
9. Hammer, E.: Logic and Visual Information. CSLI Publications (1995)
10. Hammond, T., Davis, R.: Tahuti: A geometrical sketch recognition system for UML class diagrams. In: AAAI spring symposium on Sketch understanding (2002)
11. Howse, J., Stapleton, G., Flower, J., Taylor, J.: Corresponding Regions in Euler Diagrams. In: Diagrams, *Lecture Notes in Computer Science*, vol. 2317, pp. 76–90. Springer (2002)
12. Howse, J., Stapleton, G., Taylor, J.: Spider Diagrams. Journal of Computation and Mathematics **8**, 145–194 (2005)
13. Jamnik, M., Bundy, A., Green, I.: On Automating Diagrammatic Proofs of Arithmetic Arguments. Journal of Logic, Language and Information **8**(3), 297–321 (1999)
14. Jiang, Y., Tian, F., Zhang, X.L., Dai, G., Wang, H.: Understanding, Manipulating and Searching Hand-Drawn Concept Maps. ACM Transactions on Intelligent Systems and Technology **3**(11), 21 (2011)
15. Kent, S.: Constraint diagrams: Visualizing invariants in object oriented modelling. In: OOPSLA, *SIGPLAN*, vol. 32, pp. 327–341. ACM (1997)
16. Keslter, H., Muller, A., Kraus, J., Buchholz, M., Gress, T., abd D. Kane, H.L., Zeeberg, B., Weinstein, J.: Vennmaster: Area-proportional Euler diagrams for functional go analysis of microarrays. BMC Bioinformatics **9**(67) (2008)
17. Kortenkamp, U., Richter-Gebert, J.: Using automatic theorem proving to improve the usability of geometry software. In: Procedings of the Mathematical User-Interfaces Workshop, pp. 1–12 (2004)
18. Plimmer, B., Freeman, I.: A Toolkit Approach to Sketched Diagram Recognition. In: HCI, pp. 205–213. British Computer Society (2007)
19. Plimmer, B., Purchase, H., Yang, H.: SketchNode: Intelligent sketching support and formal diagramming. In: OZCHI 2010, pp. 136–143. ACM (2010)
20. Shin, S.J.: The Logical Status of Diagrams. Cambridge University Press (2009)
21. Smith, R.: An overview of the Tesseract OCR Engine. In: ICDAR '07: Proceedings of the Ninth International Conference on Document Analysis and Recognition, pp. 629–633. IEEE Computer Society (2007)
22. Stapleton, G., Flower, J., Rodgers, P., Howse, J.: Automatically drawing Euler diagrams with circles. Journal of Visual Languages and Computing **23**(3), 163–193 (2012)
23. Stapleton, G., Howse, J., Taylor, J., Thompson, S.: The Expressiveness of Spider Diagrams. Journal of Logic and Computation **14**(6), 857–880 (2004)
24. Stapleton, G., Masthoff, J.: Incorporating Negation into Visual Logics: A Case Study Using Euler Diagrams. In: Visual Languages and Computing 2007, pp. 187–194. Knowledge Systems Institute (2007)
25. Stapleton, G., Masthoff, J., Flower, J., Fish, A., Southern, J.: Automated Theorem Proving in Euler Diagram Systems. Journal of Automated Reasoning **39**(4), 431–470 (2007)
26. Stapleton, G., Plimmer, B., Delaney, A., Rodgers, P.: Combining Sketching and Traditional Diagram Editing Tools. Accepted for ACM Transactions on Intelligent Systems and Technology (2014)
27. Stapleton, G., Taylor, J., Howse, J., Thompson, S.: The Expressiveness of Spider Diagrams Augmented with Constants. Journal of Visual Languages and Computing **20**, 30–49 (2009)
28. Swoboda, N., Allwein, G.: Heterogeneous Reasoning with Euler/Venn Diagrams Containing Named Constants and FOL. In: Proceedings of Euler Diagrams 2004, *ENTCS*, vol. 134. Elsevier Science (2005)
29. Takemura, R.: Proof theory for reasoning with Euler diagrams: a Logic Translation and Normalization. Studia Logica **101**(1), 157–191 (2013)

30. Tarski, A.: The semantic conception of truth: and the foundations of semantics. Philosophy and phenomenological research **4**(3), 341–376 (1944)
31. Urbas, M., Jamnik, M.: Diabelli: A Heterogeneous Proof System. In: Proceedings of the International Joint Conference on Automated Reasoning, *Lecture Notes in Computer Science*, vol. 7364, pp. 559–566. Springer (2012)
32. Urbas, M., Jamnik, M.: A Framework for Heterogeneous Reasoning in Formal and Informal Domains. In: T. Dwyer, H. Purchase, A. Delaney (eds.) Diagrams, *Lecture Notes in Computer Science*, vol. 8578, pp. 277–292. Springer (2014)
33. Urbas, M., Jamnik, M., Stapleton, G., Flower, J.: Speedith: a diagrammatic reasoner for spider diagrams. In: Diagrams, *Lecture Notes in Computer Science*, vol. 7352, pp. 163–177. Springer (2012)
34. Wang, M., Plimmer, B., Schmieder, P., Stapleton, G., Rodgers, P., Delaney, A.: SketchSet: Creating Euler diagrams using pen or mouse. In: IEEE Symposium on Visual Languages and Human-Centric Computing, vol. 6898, pp. 75–82. IEEE Computer Society (2011)
35. Wang, M., Plimmer, B., Schmieder, P., Stapleton, G., Rodgers, P., Delaney, A.: SketchSet: Creating Euler Diagrams using Pen or Mouse. In: IEEE Symposium on Visual Languages and Computing, pp. 75–82. IEEE (2011)
36. Winterstein, D., Bundy, A., Gurr, C.: Dr.Doodle: A Diagrammatic Theorem Prover. In: Proceedings of the International Joint Conference on Automated Reasoning, *Lecture Notes in Artificial Intelligence*, vol. 3097, pp. 331–335 (2004)