On the Expressiveness of Second-Order Spider Diagrams

Peter Chapman, Gem Stapleton, Aidan Delaney
Visual Modelling Group, University of Brighton, UK

Abstract
Existing diagrammatic notations based on Euler diagrams are mostly limited in expressiveness to monadic first-order logic with an order predicate. The most expressive monadic diagrammatic notation is known as spider diagrams of order. A primary contribution of this paper is to develop and formalise a second-order diagrammatic logic, called second-order spider diagrams, extending spider diagrams of order. A motivation for this lies in the limited expressiveness of first-order logics. They are incapable of defining a variety common properties, like ‘is even’, which are second-order definable. We show that second-order spider diagrams are at least as expressive as monadic second-order logic. This result is proved by giving a method for constructing a second-order spider diagram for any regular expression. Since monadic second-order logic sentences and regular expressions are equivalent in expressive power, this shows second-order spider diagrams can express any sentence of monadic second-order logic.

Keywords:
expressiveness, spider diagrams, diagrammatic logic, second-order logic

1. Introduction

The investigation of diagrammatic systems for information representation has shed light on a variety of areas and the benefits of using diagrammatic and visual representations are widely acknowledged. There have also been numerous theoretical studies of diagrammatic logics aimed at establishing important properties like their soundness, completeness, decidability and, where this paper sits, expressiveness. Perhaps one of the simplest diagrammatic logics is that formed from Euler diagrams. An example Euler diagram, $d_1$, can be seen in figure 1 tells us that $C$ is a subset of $B$ and that $B$ is a subset of $A$. For free, we can see that $C$ is a subset of $A$.

Figure 1: An Euler diagram and a spider diagram.

Various diagrammatic logics extend Euler diagrams by augmenting them with more syntactic elements to increase expressiveness, such as [14, 21], or focus on just the pure Euler diagrams [10, 18]. The example most related to this paper is the spider diagram logic, introduced in [9, 12] and extended in [13]. To give a brief illustration, the spider diagram, $d_2$, in figure 1 asserts, using two dots called spiders, that there is an element in the set $A - B$ and a further element in $A \cap B$; we have summarized the syntax and semantics of spider diagrams in an appendix. We go in to more detail on these logics in section 3 on related work.

There are two primary contributions made in this paper. First, we develop and formalise second-order spider diagrams by extending spider diagrams of order [6]. Second, we show that second-order spider diagrams are capable of defining all regular languages and, by extension, they are capable of defining all sentences of monadic second-order logic (MSOL), building on [7, 3, 4]. The first paper showed that spider diagrams can encode a very restricted fragment of regular languages, the commutative star-free regular languages. The second paper introduced, informally, second-order spider diagrams, and posed the question of what class of languages could be encoded by them. The third paper showed, again informally, that all second-order spider diagrams could encode regular languages with at most one instance of the Kleene star. Thus, the cumulative knowledge of the papers was that second-order spider diagrams could encode a small subset of regular languages. In this paper, we prove formally that second-order spider diagrams can encode at least all the regular languages, significantly extending the existing state-of-the-art.

In particular, to spider diagrams we add unlabelled curves to denote the existence of sets and we incorporate arrows to denote properties of a successor function. More precisely, an arrow between two curves asserts that the successors of the elements in the set denoted by the arrow’s source are precisely the elements in the set denoted by the arrow’s target. These extensions allow us to assert the universe has even cardinality and to define languages like $(aa)^*$, where we need to assert that there are an even number of $a$s, neither of which can be done using first-order languages. Whilst our extensions to the syntax of spider diagrams of order are relatively simple, we demonstrate that the in-
crease in expressiveness that they bring is substantial. In particular, we prove
that second-order spider diagrams are at least as expressive as MSOL (over an
ordered universe). It is known that a language can be defined in MSOL iff
it is regular. Our proof strategy takes any regular expression and produces a
diagram defining the same language.

Section 2 provides motivation for the work in this paper. In section 3 we
provide an overview of directly related research on which this paper builds. In
section 4 we define the syntax and semantics of second-order spider diagrams.
We define the language of a second-order spider diagram in section 5. Section 6
provides some basic results for regular expressions, allowing us to transform
them into a normal form that can be readily translated into a second-order
spider diagram. Section 7 establishes that all regular languages can be defined
by second-order spider diagrams. We also include appendices that detail various
basic definitions from formal language theory, concerning regular expressions,
and an overview basic diagrammatic notations, for the reader who is less familiar
with this branch of computer science.

2. Motivation

There are three primary motivations for the research in this paper and are
broadly categorized as: expressiveness limitations, expressive power, and links
with other theories.

First, existing diagrammatic logics are somewhat limited in expressiveness.
Many of them are limited to being monadic first-order logics, which cannot
therefore express numerous constraints on systems. In terms of practical ap-
plication, therefore, the expressive power of diagrammatic logics needs to be
increased. First-order logics themselves are also very limited in expressiveness.
Example application areas, where higher levels of expressiveness are required,
arise in software development and ontology modelling, to name just two. It
would therefore seem natural to develop more highly expressive diagrammatic
logics. This paper takes us a step closer to gaining more highly expressive
diagrammatic logics by introducing second-order spider diagrams.

Second, we need to establish the expressive power of logics in order to know
their capabilities and limitations. For example, it is known that first-order logics
cannot define properties like ‘is finite’ or construct the transitive closure of a
binary relation, whereas these are second-order definable. By establishing that
second-order spider diagrams can express statements arising in monadic second-
order logic, we provide insight into how far we have pushed the boundaries of
what can be expressed diagrammatically.

Thirdly, the approach we take to proving our expressiveness result allows us
to make natural connections with other symbolic notations. There are fre-
quently benefits arising from the development of links between what may (at
first) be seemingly disparate theories. In the case of formal languages, these
benefits have also manifested through their study via algebraic formalisms; for
example, each language has an associated syntactic monoid, some properties of
which correspond to properties of the language [? ]. Other research, by Büchi,
has shown that regular languages can be defined using second-order symbolic logic [2] and their star-free subclass can be defined by Monadic First-Order Logic of Order, MFOL[<], [22]. Thomas demonstrates that symbolic logic can provide insight into formal languages by establishing that the level at which a star-free language $L$ first appears in the dot-depth hierarchy [5] is the same as the minimum number of blocks of alternating quantifiers in an MFOL[<] sentence, in prenex normal form, which defines $L$ [22]; determining the level at which a language first appears is a long-standing open problem and the development, by Thomas, of this link has provided an additional angle of attack. We can summarise the above by saying that the different characteristics of the syntax of notations (such as diagrammatic logics, finite automata, symbolic logics, and regular expressions) imply that the study of each can provide unique insight into properties of the others.

Our work on second-order spider diagrams builds on recent work which lined spider diagrams with formal language theory that demonstrated that spider diagrams can define precisely the star-free languages [7]. In section 3, we will go in to more detail on the relationship between spider diagrams and regular languages. For now, it suffices to say that a kind of normal form that naturally arises in the spider diagram logic allowed a characterisation of commutative star-free regular languages to be identified. Thus there have been demonstrable benefits from studying spider diagrams in terms of enhancing our understanding of computation.

The spider diagram $d_2$ in figure 1 can be considered to define a language whose words contain at least two letters, each one arising from one of the two spiders. The actual letter that arises from each spider is determined by an assignment of sets of letters to curves. For instance, if $A$ is assigned the set $\{a, b, c\}$ and $B$ is assigned $\{b, c, d\}$ over alphabet $\Sigma = \{a, b, c, d, e\}$ then $d_2$ defines the language whose words contain

1. at least one letter from the set $\{a, b, c\} - \{b, c, d\} = \{a\}$ (from the spider in $A - B$), and
2. at least one letter from the set $\{a, b, c\} \cap \{b, c, d\} = \{b, c\}$ (from the spider in $A \cap B$).

So, the language of $d_1$ is

$$\mathcal{L}(d_1) = \{w \in \Sigma^* : w \text{ contains an } a \text{ and, in addition, a } b \text{ or a } c\}.$$ 

Thus, spider diagrams have helped us gain insight in to the theory of computation.

By associating second-order spider diagrams with regular languages, we are able to import standard results about regular languages over to spider diagrams. As an example, in logics one often wants to establish semantic equivalence (i.e. determine when two diagrams – in our case – represent the same information). By determining the language defined by a spider diagram, we are able to use results from regular languages to determine semantic equivalence as follows.

Two second-order spider diagrams are semantically equivalent whenever they
3. Related Work

As well as spider diagrams, other extensions of Euler or Venn diagrams exist. Initially, Venn-II diagrams were introduced by Shin [17]. These diagrams contained contours and shading, as do spider diagrams, but the contours were restricted to forming Venn diagrams. The shading represented emptiness, for instance a shaded region was interpreted as having no individuals in it. Where Venn-II diagrams further differ from spider diagrams, however, is in their representation of elements. Venn-II diagrams use $\otimes$-sequences rather than spiders, and the presence of more than one $\otimes$-sequence in a region provides no more information than a single $\otimes$-sequence in that region. In other words, Venn-II cannot represent inequality. Furthermore, whereas a single spider placed in a shaded zone in a spider diagram tells us that the interpretation of that zone has exactly one element, a $\otimes$-sequence in a shaded zone in a Venn-II diagram tells us that the zone is both empty and non-empty, i.e. is a contradiction. Shin shows that Venn-II is equivalent in expressive power to monadic first-order logic [17].

Swoboda and Allwein [20] introduced Euler/Venn diagrams, which, unlike Venn-II are based upon Euler diagrams, rather than Venn diagrams. They are thus less restrictive. Instead of $\otimes$-sequences, they use constant sequences, which give names to elements. However, distinct constant sequences in the diagram with distinct labels do not necessarily represent distinct elements in the semantics. In the case of spider diagrams, distinct spiders represent distinct elements.

We now summarise key results by Thomas [22], which establish a strong relationship between regular languages and Monadic First-Order Logic of Order (MFOL[$<$]), in which the only binary predicate is $<$. The syntax of MFOL[$<$] also includes constants min and max, interpreted as the least and greatest elements of $<$ respectively, a successor function and a predecessor function. As with the two constants, the interpretation of the successor and predecessor functions are completely determined by that of $<$. To illustrate Thomas’ work and its extension to spider diagrams or order, we take an example alphabet to be $\Sigma = \{a, b, c, d\}$ and we assign the monadic predicate symbols $Q_1$ and $Q_2$ to the sets $\{a, b\}$ and $\{b, c\}$ respectively. The sentence $\forall xQ_1(x)$ therefore defines the language consisting of all words whose letters are chosen from $\{a, b\}$, the set assigned to $Q_1$. That is, this sentence defines the language whose words contain only as or bs (such as abba and bbb).
The sentence $\forall x (Q_1(x) \land \neg Q_2(x))$ defines the language consisting of all words containing only $a$s since $\{a\} = \{a, b\} \cap \{b, c\}$.

More precisely, to define the words in a language of a sentence, $S$, we are required to consider the notion of satisfaction in a structure, $(U, \Psi, <)$, where $U$ is the universal set, $\Psi$ interprets the monadic predicate symbols as subsets of $U$, and $<$ is a strict, total order on $U$. To illustrate, $w = ab$ is satisfied by $U = \{1, 2\}$, $\Psi(Q_1) = \{1, 2\}$, $\Psi(Q_2) = \{2\}$ and $<$ is the natural order on $U$. However, $w$ is not satisfied by the structure with $U = \{1, 2, 3\}$, $\Psi(Q_1) = \{1, 2\}$, $\Psi(Q_2) = \{2\}$ and with $<$ interpreted as the natural order on $U$ since the length of $w$ is 2 but $|U| = 3$. The words in the language of $S$ are precisely those which are satisfied by the models of $S$ [22]. We return to this notion in section 5.

Spider diagrams can also define languages. First, we introduce their syntax and semantics. Spider diagrams use closed curves to denote particular sets and spiders to denote the existence of elements [13]. The topological relationships between the curves correspond to set theoretic relationships between the represented sets. For example, the diagram in figure 2 states the intersection of the two sets $B$ and $C$ lies within the set $A$. Furthermore, there are no elements in common between the sets $D$ and $A$, or between $D$ and $B$. In addition, spider diagrams use shading to place upper bounds on set cardinality; by contrast, spiders place lower bounds on set cardinality. Consider again the diagram in figure 2. The shading states that the set $A \cap B \cap C \cap D = \emptyset$, whereas the spider (the tree with two nodes, called feet) says that there is some element in $A \cup B \cup C$.

Linking back with Thomas’s work, with spider diagrams there are directly analogous concepts of structures (called interpretations) and of satisfaction. Thus, Thomas’ definition of a language definable by a sentence immediately extends to the notion of a language definable by a spider diagram.

In terms of languages, the spider diagram in figure 3 defines the language containing precisely the words that contain at least two $a$s and at least one $b$, with the assignment of letters to curve labels (analogous to monadic predicate symbols) shown by the placement of the letters. A finite state machine that accepts the same language is shown on the right. Since this language is star-free, it may also be defined by a star-free generalised regular expression such as

$$(\bar{a}ba\bar{a}ab\bar{b})|(\bar{a}ba\bar{a}ab\bar{b})|(\bar{b}ba\bar{a}ab\bar{b})$$
or an MFOL[<] sentence\(^1\), such as:

$$\exists x \exists y \exists z (Q_1(x) \land \neg Q_2(x) \land Q_1(y) \land \neg Q_2(y) \land Q_1(z) \land Q_2(z) \land x \neq y).$$

Spider diagrams cannot define any ordering on the universal set. In the context of regular languages, this limitation means that spider diagrams can only define commutative languages (a language closed under permutation of its constituent words). This observation motivated the development of spider diagrams of order, which come equipped with syntax for specifying ordering information [6]. To illustrate, figure 4 shows a spider diagram of order that asserts that an \(a\) must occur before a \(b\). This ordering information is achieved by use of the product operator, \(<\). The same language may also be defined by a generalised star-free regular expression

$$\emptyset a \emptyset b \emptyset$$

or an MFOL[<] sentence, such as:

$$\exists x \exists y (Q_1(x) \land \neg Q_2(x) \land Q_1(y) \land Q_2(y) \land x < y).$$

However, the ability to express this kind of ordering information is still insufficient to define arbitrary regular languages. It has been shown that the

\(^1\)This is also an MFOL with equality sentence.
languages definable by spider diagrams of order are precisely those that are definable by generalised star-free regular expressions [8]. The language defined by, for example, \((aa)^*\) is not such a language. In terms of a logical definition, it amounts to being able to specify that the cardinality of the universal set is even, which is a properly second-order concept.

To summarise, spider diagrams of order are capable of defining precisely the star-free regular languages [8] and are, therefore, equivalent in expressiveness to MFOL[\(<\)]. The spider diagram fragment, which does not use \(<\), is capable of defining precisely the commutative star-free languages [7] and are known to be equivalent in expressiveness to Monadic First-Order Logic with Equality (MFOL[=]) [19], and thus are more expressive than Shih’s Venn-II system, which we recall is equivalent to MFOL.

To establish a lower bound on expressiveness for second-order spider diagrams, we follow a similar method as that described for spider diagrams of order: we show how to encode a class of regular expressions, and then use a known result to establish expressiveness in terms of symbolic logic. In this case, we encode all regular expressions, and the known result is:

**Theorem 1** ([? ]). A language is regular iff it is definable in monadic second-order logic.

4. Second-Order Spider Diagrams

We now proceed to define second-order spider diagrams, extending [6]. Our additional syntax includes unlabelled curves to represent existential quantification over subsets, and the use of arrows to represent properties of a successor function. The formal definition is definition 1 on page 9: for now we just describe the various elements in a diagram.

Diagram \(d_1\) in figure 5 depicts a second-order spider diagram. The unlabelled curve denotes the existence of a set, \(E\), in this case a subset of \(A - B\), and the arrow asserts that the image of the successor function with its domain restricted to \(E\) is the element, \(e\), denoted by the spider in \(B - A\). Although there are no spiders explicitly represented in \(E\), we know it must, therefore, represent a non-empty set and moreover be a singleton set: the elements of this set have, between them, a single successor. The sets represented by the source and target of a successor arrow have the same cardinality.

We formalise the abstract syntax of unitary diagrams which are combined in various ways to create compound diagrams. We have a finite set of fixed contours, \(\mathcal{FC}\), and a countably infinite set of existential contours, \(\mathcal{EC}\). Each contour in \(\mathcal{FC}\) corresponds to a labelled curve in a diagram, whereas the contours in \(\mathcal{EC}\) correspond to unlabelled curves. A zone is a pair of finite, disjoint sets of contours. Given a diagram, \(d\), with a finite set of contours \(C \subset \mathcal{FC} \cup \mathcal{EC}\), one defines the zones of \(d\) as pairs of sets of contours \((in, out)\), such that \(in \cup out = C\). Intuitively, the zone thus defined is inside every contour of \(in\), and outside every contour of \(out\). The set of all zones is \(\mathcal{Z}\), and the set of zones of a diagram
is denoted $Z$. Zones may be shaded: the set of shaded zones of a diagram is denoted $Sh$, where $Sh \subseteq Z$.

In addition, we have a set $\mathcal{ES}$ whose elements are called existential spiders or simply, spiders. In diagrams, spiders are placed in a set of zones, called their habitat. The sets $\mathcal{FC}, \mathcal{EC}$ and $\mathcal{ES}$ are assumed to be pairwise disjoint. Properties of the successor function will be denoted by arrows, formally defined as a set of pairs, $(source, target)$. The source and target for each pair is drawn from $\mathcal{FC} \cup \mathcal{EC} \cup \mathcal{ES}$. As an example, the arrow in $d_2$ in figure 5 is $(A, B)$.

In $d_1$, figure 5, there are two fixed contours, $A$ and $B$, and one existential contour, unlabelled in the diagram but given the name $E$ in the abstract syntax. There are 5 zones in this diagram, such as $(\{A\}, \{E, B\})$ and one shaded zone: $(\{A, B\}, \{E\})$. The spider, $s$, is placed in the region $(\{(B), \{E, A\})$. The set $SucA(d)$ contains one element, namely $(E, s)$.

In order to simplify many definitions in this paper, we do not record the zones of a diagram explicitly in the abstract syntax. Instead, for each zone $(in, out)$, we record only the set of contours that contain the zone, in, as a set of zone identifiers, $ZI$. So, $in \in ZI$, iff $(in, C − in) \in Z$. The following definition builds on that for spider diagrams of order from [6]:

**Definition 1** (Unitary second-order spider diagram). A unitary second-order spider diagram is a tuple:

$$d = (C, ZI, ShZI, S, \eta, SucA)$$

which satisfies the following:

1. $C = C(d) \subset FC \cup EC$ is a finite set of contours.

---

2 Note that our definition of a second-order spider diagram provides an abstract syntax, sometimes called a type syntax. This is typical in the literature: visual languages are often defined using an abstract syntax. This can then be linked with a concrete syntax, sometimes called a token syntax, that captures geometric properties of the diagrams. For the purposes of this paper, we only need access to the abstract syntax and, thus, omit the definition of the concrete syntax. For related work on abstract and concrete syntax see [11]. Further, it is easy to extend results in [16], concerning Euler diagram drawability, to demonstrate that every (abstract) second-order spider diagram can be realised as a concrete drawing.
2. $ZI = ZI(d)$ is a finite set of zone identifiers, such that for each zone identifier $in \in ZI(d)$, $in \subseteq C(d)$.
3. $\text{Sh}ZI = \text{Sh}ZI(d) \subseteq ZI(d)$ is a set of shaded zone identifiers.
4. $S = S(d) \subseteq ES$ is a finite set of spiders.
5. The function $\eta_d : S(d) \rightarrow \mathcal{P}(ZI(d))$ returns the habitat of every spider.
6. $\text{Suc}A = \text{Suc}A(d) \subseteq ZI(d)$ is a set of successor arrows such that for each $(s, t) \in \text{Suc}A(d)$, $s$ and $t$ are each either a contour in $C(d)$ or a spider in $S(d)$.

We define the **zones** of $d$ to be $Z(d) = \{(\text{in}, C(d) - \text{in}) : \text{in} \in ZI(d)\}$, and the set of **shaded** zones of $d$ to be $\text{Sh}(d) = \{(\text{in}, C(d) - \text{in}) : \text{in} \in \text{Sh}ZI(d)\}$. The symbol $\bot$ is also a unitary second-order spider diagram.

We now extend the definition of unitary second-order spider diagrams to **compound** second-order spider diagrams:

**Definition 2** (Second-Order Spider Diagram). A **second-order spider diagram** is defined inductively:

1. Any unitary second-order spider diagram is a second-order spider diagram,
2. If $d_1$ is a second-order spider diagram, then $\neg d_1$ is a second-order spider diagram,
3. If $d_1$ and $d_2$ are second-order spider diagrams, then:
   (a) $(d_1 \land d_2)$,
   (b) $(d_1 \lor d_2)$,
   (c) $(d_1 \triangleleft d_2)$ (read $d_1$ product $d_2$)
are second-order spider diagrams.

Any second-order spider diagram which is not a unitary diagram is a **compound** diagram.

We have a further concept of a **missing zone**. Intuitively, a zone is missing if such a zone is possible given the contour set of a diagram, but not present in the zones of that diagram. The diagram $d_1$ in figure 5 has 3 contours, and so 8 possible zones. Thus, there are 3 missing zones: $\{(\{E\}, \{A, B\})\}$, $\{(B, \emptyset), \{A\}\}$ and $\{(A, B, \emptyset), \emptyset\}$.

**Definition 3.** Given a unitary second-order diagram $d$ ($\neq \bot$), a zone $(\text{in}, \text{out})$ is **missing** from $d$ if it is in the set $MZ(d) = \{(\text{in}, C(d) - \text{in}) : \text{in} \subseteq C(d)\} - Z(d)$.

The semantics extend those in [6]. Informally, we map contours and spiders to subsets of a totally ordered universe, $U$, using a function denoted $\Psi$. For instance, in $d_2$ in figure 5, since the spider $s_1$ is contained in the contour $A$, we must have $\Psi(s_1) \in \Psi(A)$. Furthermore, the interpretation of $s_1$ under $\Psi$ should precede the interpretation of $s_2$ under $\Psi$, with respect to the total ordering on $U$. Since the successor arrow connects $A$ and $B$, we further require that the interpretation of $B$ is the successor set of the interpretation of $A$. The shading and spider in $B$ mean that there should be exactly one element in the
interpretation of $B$. Thus, a possible model for $d_2$ would be to take $U = \{1, 2\}$ with the natural order $<$, and $\Psi(A) = \{1\}, \Psi(B) = \{2\}, \Psi(s_1) = \{1\}$, and $\Psi(s_2) = \{2\}$. Since spiders represent the existence of elements, not constants, and existential contours represent the existence of sets, we break down the definition of the mapping $\Psi$ into two parts. First, we map the fixed contours to subsets of the universal set, $U$, to give us an interpretation. For an interpretation to be a model it must be possible to extend $\Psi$ in an appropriate way to the spiders and existential contours.

**Definition 4 (Interpretation).** An **interpretation** is a 4-tuple, $(U, \Psi, <, \text{Suc})$, where

1. $U$ is some finite set,
2. $\Psi : \mathcal{FC} \to \mathcal{P}U$ is a function which maps fixed contours to subsets of $U$,
3. $<$ is a strict total order on $U$, and
4. $\text{Suc}$ is a function such that for each $a, b$ if $a < b$ and there exists no $c$ such that $a < c < b$ then we have $b = \text{Suc}(a)$.

In the above definition, $\text{Suc}$ is an injective, partial function; the maximal element of $<$ has no successor but all other elements have unique successors. Given an interpretation, we wish to know when it agrees with the intended meaning of the diagram; an ‘agreeing’ interpretation is called a **model**. As an example, consider again $d_1$ in figure 5. A model for this diagram is $U = \{1, \ldots, 5\}$, with $<$ being the natural order on $U$, $\text{Suc}$ being the natural successor function on $U$, and $\Psi(A) = \{1, 2\}$ and $\Psi(B) = \{3\}$. In order to identify this interpretation as a model, we need to interpret the existentially quantified elements of the diagram, namely the existential contour $E$ and the spider $s$. To do so, we extend $\Psi$ to $\Psi'$, mapping these elements in an appropriate manner to subsets of $U$: $\Psi'(E) = \{2\}$ and $\Psi'(s) = \{3\}$.

**Definition 5 (Extended Interpretation).** Let $I = (U, \Psi, <, \text{Suc})$ be an interpretation, and let $EC$ be a set of existential contours, and let $S$ be a set of spiders. An **extension** of $I$ to $EC$ and $S$ is $J = (U, \Psi', <, \text{Suc})$ where $\Psi' : \mathcal{FC} \cup EC \cup S \to \mathcal{P}U$ and:

1. for each $c \in \mathcal{FC}$, $\Psi'(c) = \Psi(c)$,
2. for each spider $s \in S$, we have $|\Psi'(s)| = 1$
3. we further extend $\Psi'$ to interpret zones and regions:
   (a) for each zone $(\text{in, out}) \in \mathcal{P}(\mathcal{FC} \cup EC) \times \mathcal{P}(\mathcal{FC} \cup EC)$, we define:
   $$\Psi'(\text{in, out}) = \bigcap_{c \in \text{in}} \Psi'(c) \cap \bigcup_{c \notin \text{out}} (U - \Psi'(c))$$
   (b) for each such set of zones $Z$, we define:
   $$\Psi'(Z) = \bigcup_{z \in Z} \Psi'(z)$$
We can then give conditions as to when an interpretation, \( I \), is a model for a diagram \( d \), by extending \( I \) to interpret the existential contours of \( d \), denoted \( EC(d) \), and the spiders of \( d \), denoted \( S(d) \). So, \( EC(d) = C(d) \cap EC \).

**Definition 6 (Model).** Let \( d \) be a unitary second-order spider diagram and let \( I = (U, \Psi, <, Suc) \) be an interpretation. If \( d \neq \bot \) and there exists an extension \( J = (U, \Psi', <, Suc) \) of \( I \) to \( EC(d) \cup S(d) \) where the following conditions hold, then \( I \) is a model for \( d \):

1. **The missing zones condition.** The missing zones represent the empty set, i.e.
   \[
   \bigcup_{z \in MZ(d)} \Psi'(d) = \emptyset.
   \]

2. **The spiders’ distinctness condition.** If two spiders have the same interpretation then they are the same spider:
   \[
   \forall s_1, s_2 \in S(d). (\Psi'(s_1) = \Psi'(s_2) \Rightarrow s_1 = s_2).
   \]

3. **The shaded zones condition.** All elements in the sets represented by shaded zones are represented by spiders:
   \[
   \forall z \in Sh(d). \Psi'(z) \subseteq \bigcup_{s \in S(d)} \Psi'(s).
   \]

4. **The spiders’ location condition.** The elements represented by the spiders are in the sets represented by their habitats:
   \[
   \forall s \in S(d). \Psi'(s) \subseteq \bigcup_{in \in \Psi(s)} \Psi'(in, C(d) - in).
   \]

5. **The successor condition.** Successor arrows indicate that the successor function induced by \( < \) gives rise to a bijection between the sets represented by the source and target:
   \[
   \forall (s, t) \in SucA(d). Suc|^\Psi(s) \text{ is a bijection with image } \Psi'(t).
   \]

If \( J \) makes the above conditions hold then \( J \) is a valid extension of \( I \). If \( d = \bot \) then no interpretation is a model for \( d \).

Consider again \( d_1 \) from figure 5. Taking \( U = \{1, 2, 3\} \), if we had \( \Psi(A) = \{1, 2\} \) and \( \Psi(B) = \{2, 3\} \), then we would have \( \Psi'([A, B], \emptyset) = \{2\} \). To satisfy the spiders’ location condition, the interpretation of the spider, \( s \), must be \( \Psi'(s) = \{3\} \). The diagram only contains one spider, so we have \( \bigcup_{s \in S(d)} \Psi'(s) = \{3\} \). The zone \( ([A, B], \emptyset) \) is shaded, but \( \{2\} = \Psi'([A, B], \emptyset) \not\subseteq \bigcup_{s \in S(d)} \Psi'(s) = \{3\} \). Thus, this interpretation is not a model for \( d_1 \).

We now extend the concept of a model to compound diagrams. In the case of the connectives \( \land, \lor \) and \( \neg \), the extension is obvious. However, the product
operator \( \triangleleft \) is more subtle. Consider figure 6. A model for this diagram is \( U = \{1, 2, 3, 4\} \), with \( \Psi(A) = \{1\} \) and \( \Psi(B) = \{4\} \), with \( \triangleleft \) and \( \text{Suc} \) defined naturally. This can be split into two models, \( I_1 \) and \( I_2 \), where

\[
I_1 = (\{1, 2\}, \Psi_1, \triangleleft_1, \text{Suc}_1) \quad \text{and} \quad I_2 = (\{3, 4\}, \Psi_2, \triangleleft_2, \text{Suc}_2),
\]

where \( \Psi_i, \triangleleft_i \) and \( \text{Suc}_i \) are the relation \( \triangleleft \) and the functions \( \Psi \) and \( \text{Suc} \) restricted in the appropriate way. Here \( I_1 \) models \( d_1 \) and \( I_2 \) models \( d_2 \). We say \( I \) is the ordered sum of \( I_1 \) and \( I_2 \), denoted \( I_1 + I_2 \).

Formally, we extend the following definition from [8]:

**Definition 7 (Ordered Sum).** Let \( I_1 = (U_1, \Psi_1, \triangleleft_1, \text{Suc}_1) \) and \( I_2 = (U_2, \Psi_2, \triangleleft_2, \text{Suc}_2) \) be interpretations where \( U_1 \) and \( U_2 \) are disjoint. The ordered sum of \( I_1 \) and \( I_2 \), denoted \( I_1 + I_2 \), is defined to be the interpretation \( I = (U, \Psi, \triangleleft, \text{Suc}) \) such that:

1. \( U = U_1 \cup U_2 \),
2. for each \( c \in FC \), \( \Psi(c) = \Psi_1(c) \cup \Psi_2(c) \), and
3. \( \triangleleft = \triangleleft_1 \cup \triangleleft_2 \cup \{ (u_1, u_2) : u_1 \in U_1 \land u_2 \in U_2 \} \).

**Definition 8 (Model).** Let \( I = (U, \Psi, \triangleleft, \text{Suc}) \) be an interpretation and let \( d \) be a compound diagram. Then \( I \) is a model for \( d \) provided:

1. if \( d = \neg d_1 \) then \( I \) models \( d \) whenever \( I \) does not model \( d_1 \),
2. if \( d = d_1 \lor d_2 \) then \( I \) models \( d \) whenever \( I \) models \( d_1 \) or \( I \) models \( d_2 \),
3. if \( d = d_1 \land d_2 \) then \( I \) models \( d \) whenever \( I \) models \( d_1 \) and \( I \) models \( d_2 \), and
4. if \( d = d_1 \triangleleft d_2 \) then \( I \) models \( d \) whenever there exist interpretations \( I_1 \) and \( I_2 \) such that \( I = I_1 + I_2 \) and \( I_1 \) models \( d_1 \) and \( I_2 \) models \( d_2 \).

Second-order spider diagrams are a direct extension of spider diagrams of order as in [8], augmenting them with arrows to talk about successors, and existential contours. Therefore:

**Theorem 2.** Second-order spider diagrams are at least as expressive as spider diagrams of order.

Previous work has shown that spider diagrams of order are equivalent in expressive power to MFOL\[\triangleleft\] [8]. Thus:

**Corollary 3.** Second-order spider diagrams are at least as expressive as MFOL\[\triangleleft\].
5. Defining Languages using Diagrams

Consider the diagram \( d_1 \) in figure 7. There is one spider in the contour \( A \), and one spider outside the contour \( A \). Suppose we assign the letter \( a \) to the contour \( A \), and the letter \( b \) to the rest of the diagram. What language does \( d_1 \) then represent? Because there is no shading in the diagram, the number of letters in a word is not restricted. However, owing to the spiders and the successor arrow, we know any word in the language of \( d_1 \) must contain the subword \( ab \). This diagram provides no further restriction on words and, therefore, contains all words with \( ab \) as a subword. Now, the shading in \( d_2 \) means that there is exactly one element in the contour \( A \), so every word contains exactly one \( a \). In terms of regular expressions, \( d_2 \) defines the same language as \( b^* (ab)b^* \).

![Figure 7: Languages for diagrams](image)

Any language that is definable by a spider diagram of order will be definable by a second-order spider diagram. It is known that the language \((aa)^+\) is not first-order definable and, therefore, is not definable by a spider diagram of order. However, it is second-order definable. Consider \( d_1 \) in figure 8. Given an alphabet \( \Sigma = \{a, b\} \), we assign \( a \) to the given contour \( A \). Elements in the existential subset containing the spider \( s_2 \) (call this subset \( A_2 \)) are successors of elements of the existential subset containing the spider \( s_1 \) (call this subset \( A_1 \)). Furthermore, the shading elsewhere in the diagram shows that elements in \( A_2 \) can only have successors in \( A_1 \). Since we have a bijection between the disjoint subsets \( A_1 \) and \( A_2 \), and \( \Psi(A_1) \cup \Psi(A_2) = U \), we deduce \( U \) has even cardinality. Thus, since both are assigned the letter \( a \), any word in the language defined by \( d_1 \) must consist of an even number of \( a \); in other words, the language defined is \((aa)^+\). Omitting both spiders from the diagram would change the defined language to \((aa)^*\), whereas omitting exactly one spider would lead the defined language unchanged. Similar reasoning gives us that \( d_2 \) in figure 8 defines the language \((aba)^+\). Further, the universal set of any model of this diagram must have cardinality which is divisible by 3. Again, omitting both spiders would mean that the language defined is \((aba)^*\).

We now formalise the notion of a language of a diagram. To do so, we assume that the sets \( FC \) and \( \Sigma \) are fixed. Moreover, we assume that there is a specified mapping from given contours to sets of letters:

**Definition 9.** A function \( l: FC \rightarrow P\Sigma \) is a letter assignment. We extend \( l \)
to assign sets of letters to zones, such that for each zone \((\text{in}, \text{out})\)

\[
\begin{align*}
  l(\text{in}, \text{out}) &= \bigcap_{c \in \text{in} \cap FC} l(c) \bigcap_{c \in \text{out} \cap FC} (\Sigma - l(c)).
\end{align*}
\]

Using \(l\), we are able to associate letters with zones. First, we define for each letter \(a \in \Sigma\), the set of fixed contours \((fc)\) that ‘contain’ that letter:

\[
fc(a) = \{c \in FC : a \in l(c)\}.
\]

So, \(fc\) is a function with domain \(\Sigma\) and codomain \(PFC\), that returns a zone identifier. We can think of the zone \((fc(a), FC - fc(a))\) as containing \(a\). For instance, if we take the mapping \(l(A) = \{a, b\}\) and \(l(B) = \{b, c\}\) over the alphabet \(\Sigma = \{a, b, c\}\) and with \(FC = \{A, B\}\) then the letter \(a\) has \(fc(a) = \{A\}\) and is therefore contained by the zone \((\{A\}, \{B\})\). Thus, a unitary diagram containing the zone \((\{A\}, \{B\})\) with a spider placed in that zone would assert the existence of a letter \(a\) in each word of the language it defines. A unitary diagram containing the zone \((\{A\}, \emptyset)\) with a spider placed in that zone would assert the existence of a letter \(a\) or a letter \(b\) in each word of the language it defines; see figure 9, where the diagram defines \((a|b|c)^*(a|b)(a|b|c)^*\) with the sub-expression \((a|b)\) arising from the spider. By contrast, a diagram containing both the fixed contours \(A\) and \(B\) can distinguish the letters \(a\) and \(b\) using spiders.

It is important that we are able to distinguish each letter using some diagram: if two letters, \(a\) and \(b\) say, have the same image under \(fc\) then no diagram can
define the language $a^*$ for example. We require that no two letters are placed in any zone that arises from the function $fc$.

**Definition 10** (Valid letter assignment). Given $FC$, $\Sigma$ and a letter assignment $l$, we say that $l$ is **valid** if the induced function $fc$ is injective.

From this point forward, we are assuming that a valid letter assignment, $l$, has been specified. We are now able to identify word models, extending and adapting [22]. That is, given an interpretation, we can identify whether it models a word. This notion can then be extended to languages.

**Definition 11** (Word model). Let $I = (U, \Psi, <, Suc)$ be an interpretation and let $w = w_1...w_n$ be a word drawn from $\Sigma^*$, where each $w_i \in \Sigma$. Then $I$ is a **model** for $w$ iff there exists a bijection $f$, from the multi-set $\{w_1,...,w_n\}$ to $U$ such that for each $w_i$:

1. $f(w_i)$ is the $i$th element of the total ordering induced by $<$
2. $f(w_i) \in \Psi(fc(w_i), FC - fc(w_i))$.

Furthermore, $I$ is a **model** for language $L$ iff there exists a word, $w$, in $L$ such that $I$ models $w$.

**Definition 12** (Language of a diagram). Let $d$ be a second-order spider diagram and let $L$ be a language. We say that $L$ is the **language of** $d$, denoted $\mathcal{L}(d)$, iff the models of $d$ are the same as the models of $L$. If $L$ is the language of $d$ then we say that $d$ **defines** $L$.

For example, under the just defined function $l$ with alphabet $\Sigma = \{a, b, c\}$, if the diagrams $d_1$ and $d_2$ from figure 10 are combined using $\triangleleft$, then this compound diagram defines the same language as the regular expression $r_1 \cdot r_2$ where

$$r_1 = ((a|b|c)^*a(a|b|c)^*a(a|b|c)^*b(a|b|c)^*)$$
$$((a|b|c)^*a(a|b|c)^*b(a|b|c)^*a(a|b|c)^*)$$
$$((a|b|c)^*b(a|b|c)^*a(a|b|c)^*a(a|b|c)^*)$$

and $r_2 = ce^*$. Note that, for $r_1$, each of the three disjuncts arises from an ordering of the letters arising from the spiders. Later in the paper, we provide a method for constructing a diagram, $d$, given a regular expression, $r$, such that $r$ and $d$ define the same language. The construction allows us to prove that all regular languages are definable by second-order spider diagrams.

Under the same $l$, the diagram $d_3$ in figure 10 does not define any language. This is because the zone $(\emptyset, \{A, B\})$ is (a) not shaded, and (b) is not assigned any letters under $l$. More formally, $d_3$ has models that are not models for any words and, therefore, not models for any language. Such a model for $d_3$ is given by $I = (U, \Psi, <, Suc)$ where $U = \{1, 2\}$, $\Psi(A) = \{1\}$, $\Psi(B) = \emptyset$ and $<$ is the natural order over $U$; here, we have $\Psi(\emptyset, \{A, B\}) = \{2\}$. Any word, $w = w_1w_2$, that is modelled by $I$ would need to have exactly two letters, in order that a bijection, $f$, exists from $\{w_1, w_2\}$ to $U$. Suppose that $I$ does indeed model $w$. 

16
For the ordering of letters in $w$ to respect $<$, we therefore have $f(w_1) = 1$ and $f(w_2) = 2$. Since $I$ models $w$,

$$f(w_2) \in \Psi(fc(w_2), FC - fc(w_2)).$$

(1)

Noting that, for any letter, $w_j$ in $\Sigma$ $fc(w_j) \neq \emptyset$,

$$(fc(w_2), FC - fc(w_2)) \neq (\emptyset, \{A, B\}).$$

(2)

It is relatively easy to show that distinct zones, $(in_1, out_1)$ and $(in_2, out_2)$ where $in_1 \cup out_1 = in_2 \cup out_2$ represent disjoint sets in any interpretations. Thus, from (1) and (2) it cannot be the case that

$$2 = f(w_2) \in \Psi(\emptyset, \{A, B\}) = \{2\}.$$

We reach a contradiction and, hence, the supposition that $I$ models $w$ is false. Therefore, $d_3$ has a model that does not model any word. Thus, under $l$, no language has the same model set as $d_3$. In conclusion, $d_3$ does not define any language.

This reasoning yields the following result, given without proof:

**Lemma 4.** Let $I = (U, \Psi, <, Suc)$ be an interpretation, and $\Sigma$ be some alphabet. Then, $I$ models some word $w \in \Sigma^*$ iff for each zone $z$, if $\Psi(z) \neq \emptyset$ then $\exists a \in \Sigma$ $z = (fc(a), FC - fc(a))$.

In other words, this lemma states that every zone which is interpreted in the model must have a letter assigned to it, if that diagram is to represent a word.

### 6. Regular Expressions

An aim of this paper is to show that every regular language can be defined using a second-order spider diagram, and thus that every sentence in MSOL can be defined in a second-order spider diagram. Our strategy to prove that second-order spider diagrams can define all regular languages is: given such a language, $L$, identify a diagram that defines $L$. Now, it is known that $L$ is defined by a regular expression, $r$, with the typical definition of such outlined in
the appendix. Of note is that regular expressions do not include the complement operator; the inclusion of this operator yields generalised regular expressions.

Regular expressions contain some components which are troublesome to encode in a diagram. For instance, take the regular expression \((a^*(b\lambda)c)\). A word in the language of this diagram may start with an \(a\), a \(b\), or a \(c\). Furthermore, we may have a \(b\) following an \(a\), or a \(c\) following an \(a\). This will mean that many arrows are needed in a diagram that defines a language and, since arrows are bijective, their source and target contours need to be specified correctly. In order to simplify the construction of a diagram defining languages like this one, we instead work with a different, but equivalent, formulation of regular expressions. We call these \(d\)-regular expressions, with the \(d\) standing for diagram.

**Definition 13** (\(d\)-regular expressions). Given an alphabet \(\Sigma\), a \(d\)-regular expression over \(\Sigma\) is defined as:

\[
r = \emptyset \mid \lambda \mid a \quad \text{where } a \in \Sigma
\]

\[
r = r_1 \cdot r_2 \quad \text{where } r_1, r_2 \text{ contain no instances of } \emptyset \text{ or } \lambda
\]

\[
r = r_1 | r_2 \quad \text{where } r_1, r_2 \text{ contain no instances of } \emptyset \text{ or } \lambda
\]

\[
r = r_1^+ \quad \text{where } r_1 \text{ contains no instances } \lambda \text{ or } \emptyset, \text{ and if } r_1 \text{ is a disjunction, say } (r_1' | \cdots | r_n'), \text{ then none of its disjuncts are single letters, that is } r_i' \not\in \Sigma \text{ for each } i.
\]

The last side condition for \(r_1^+\) is for technical reasons which will be explained later: in essence it is not possible to draw a single letter raised to a power using arrows and only one contour. We show any regular expression is essentially equivalent to some \(d\)-regular expression. We use the notation \(r =_L r'\) to stand for \(r\) defines the same language as \(r'\). In the following proof, \(\equiv\) means literal identity.

**Lemma 5.** Given any regular expression, \(r\), there exists a \(d\)-regular expression, \(r'\) such that either:

1. \(r =_L r'\), or
2. \(r =_L \lambda r'\)

**Proof.** Let a regular expression, \(r\), over an alphabet, \(\Sigma\), be given. The proof proceeds by induction on the structure of \(r\). If \(r \equiv \lambda\), \(r \equiv a \in \Sigma\) or \(r \equiv \emptyset\), (i.e. the base cases of the induction) then the lemma is trivially verified, since \(r\) is also a \(d\)-regular expression.

1. \(r \equiv r_1 \cdot r_2\). The induction hypothesis gives us two \(d\)-regular expressions \(r_1'\) and \(r_2'\) such that \(r_1\) defines the same language as \(r_1'\) or \(\lambda | r_1\) (similarly for \(r_2\)). Thus, there are four subcases, depending on whether each \(r_i'\) appears in a disjunction with \(\lambda\). We show two cases, the other two are similar:
(a) \( r_1 = L r'_1 \) and \( r_2 = L r'_2 \). Observe that \( s \cdot \emptyset = \emptyset \cdot s = \emptyset \), and \( s \cdot \lambda = \lambda \cdot s = s \) for any expression \( s \). Thus, if either \( r'_1 \) or \( r'_2 \) is \( \lambda \) or \( \emptyset \), we can simplify \( r'_1 r'_2 \) appropriately to a \( d \)-regular expression. If neither is \( \lambda \) nor \( \emptyset \), then \( r'_1 \cdot r'_2 \) is trivially \( d \)-regular, hence \( r_1 r_2 = L r'_1 \cdot r'_2 \), as required.

(b) \( r_1 = L \lambda | r'_1 \) and \( r_2 = L \lambda | r'_2 \). Then:

\[
\begin{align*}
r_1 \cdot r_2 &= L (\lambda | r'_1) \cdot (\lambda | r'_2) \\
&= L (\lambda | r'_1 | r'_2) \\
&= L \lambda | (r'_1 | r'_2).
\end{align*}
\]

Observe that \( s | \lambda | \lambda = s | \emptyset \) and \( \emptyset | s = s | \emptyset = s \) for any expression \( s \).

Thus, by case analysis of the form of \( r'_1 \) and \( r'_2 \), we can transform the last line into \( \lambda \) or \( \lambda | r' \), where \( r' \) is \( d \)-regular, as required.

2. \( r \equiv r_1 | r_2 \). Then the induction hypothesis gives us \( r_1 \) and \( r_2 \) define the same language as some \( d \)-regular expressions \( r'_1 \) and \( r'_2 \) (or \( \lambda | r'_1 \) and \( \lambda | r'_2 \)) respectively, giving four subcases. Again, we show only one: the rest are similar. Suppose \( r_1 = L \lambda | r'_1 \) and \( r_2 = L \lambda | r'_2 \). Then:

\[
r_1 r_2 = L (\lambda | r'_1) | (\lambda | r'_2) = L \lambda | (r'_1 | r'_2).
\]

Using the same transformations from the previous case to eliminate \( \lambda \) and \( \emptyset \), it is straightforward to transform the last expression into a \( d \)-regular expression.

3. \( r \equiv r^+_1 \). We have \( r^+_1 = L \lambda | r^+_1 \). There are two subcases:

(a) \( r_1 = L r'_1 \). Then, \( r^+_1 = L (\lambda | r^+_1) \). Observe that \( \lambda^+ = \lambda \) and \( \emptyset^+ = \emptyset \).

This observation, along with the other methods for eliminating \( \emptyset \) and \( \lambda \) from case 1, lets us transform \( (\lambda | r^+_1) \) into a \( d \)-regular expression, if \( r'_1 \) is anything but a disjunction containing a single letter:

i. if \( r'_1 \equiv \emptyset \), then \( \lambda | r^+_1 = L \lambda | \emptyset = L \lambda \).

ii. if \( r'_1 \equiv \lambda \), then \( \lambda | r^+_1 = L \lambda | \lambda = L \lambda \), and

iii. if \( r'_1 \not\equiv \emptyset \) and \( r'_1 \not\equiv \lambda \) and \( r'_1 \) is not a disjunction containing a single letter, then \( r^+_1 \) is \( d \)-regular, and hence \( \lambda | r^+_1 \) is of the required form.

If \( r'_1 \) is a disjunction containing a single letter as a disjunct, we have to perform one more transformation. It is easy to show that:

\[
(r'_1 | \cdots | r'_i | \cdots | r'_n)^+ = L (r'_1 | \cdots | r^+_i | \cdots | r'_n)
\]

where \( r'_i \in \Sigma \). Thus, any single letter, say \( a \), which appears as a disjunct within \( r'_i \) can be replaced with \( a^+ \), and the language defined remains the same, as required. The proof then proceeds as for the previous cases.

(b) \( r_1 = L (\lambda | r'_1) \). Then \( r^+_1 = L (\lambda | r^+_1) \). Observe that \( (\lambda | s)^+ = L \lambda | s^+ \), and so:

\[
r^+_1 = L \lambda | (r^+_1)^+ = L \lambda | r^+_1 = L \lambda | r^+_1
\]

and we can then proceed as in the previous subcase.
Why is this result useful? It is because instances of \( \lambda \) and starred expressions within words cause problems for diagrams: they represent possibly empty expressions, and since arrows typically force the existence of elements, the combining of arrows and starred expressions or \( \lambda \) becomes troublesome. The expression \( r = (a|\lambda) \cdot (b|\lambda) \cdot d^* \cdot e \) is very simple, however there are lots of words in the language of this expression which are difficult to capture in a diagram. For instance, the word \( a \cdot e \) is in the language of \( r \), but a diagram would have difficulty expressing this. However, when \( r \) is converted to a \( d \)-regular expression, it becomes:

\[
\text{abd}^*e|ae|bd^*e|d^*e|e
\]

In this form, we do not have any concatenations where one of the concatenands is \( \lambda \). Thus, the diagrams are easier to construct, although they may be larger.

7. Definability of Regular Languages

A primary goal of this paper is to show that second-order spider diagrams are at least as expressive as MSOL. We know, by theorem 1, that any sentence is MSOL is equivalent to some regular expression. Using lemma 5, we know for any regular expression, \( r \), there exists a \( d \)-regular expression \( r' \), such that \( r \) encodes the same language as \( r' \) or \( \lambda | r' \). We complete the chain by describing a method for constructing a diagram for any \( d \)-regular expression.

We adopt the notation \( \mathcal{L}(r) \) and \( \mathcal{L}(d) \) to denote the languages defined by \( r \) and \( d \) respectively; note that the appendix formally defines \( \mathcal{L}(r) \). This section concludes with a proof that the construction is correct in that we show \( \mathcal{L}(d) = \mathcal{L}(r) \), where \( d \) is constructed from \( r \).

To build a diagram for a \( d \)-regular expression, we mimic the construction of the \( d \)-regular expression itself. Thus, we have three base cases (for \( \emptyset \), \( \lambda \) and letters in \( \Sigma \)) along with an extra base case for \( a^+ \) where \( a \in \Sigma \), and three inductive cases (for |, \( \cdot \) and \( ^+ \)). To illustrate the process, consider figure 11 which shows the construction of a diagram for the simple \( d \)-regular expression \( r = (ab)^+ \). For the purposes of this example, we take \( \Sigma = \{a,b\}, FC = \{A\}, \) and the valid letter assignment \( l(A) = \{a\} \). The process starts with an entirely shaded unitary diagram, \( d_1 \), containing all of the fixed contours (\( C(d_1) = FC \)) and no missing zones. To \( d_1 \), we add first add an existential contour, \( EC_a \), for \( a \), and place a spider within this contour, giving \( d_2 \). To construct \( a \cdot b \), we add a further existential contour, \( EC_b \), for \( b \), and join the two contours by an arrow, giving \( d_3 \). Next, to simulate the plus operation, we add further contours to \( EC_a \) and \( EC_b \), which intersect \( EC_a \) and \( EC_b \), and add another arrow, to give \( d_4 \). We say that the diagram \( d_4 \) is constructed for \( r \) and we write \( d_4 = d_1 + r \).

We now discuss why \( d_4 \) defines the appropriate language. Now,

\[
\mathcal{L}(r^+) = \{ab, abab, ababab, \ldots\}
\]
Any model, $I = (U, \Psi, <, \text{Suc})$, for the word $ab$, is isomorphic to that where $U = \{1, 2\}$ and $\Psi(A) = \{1\}$, with the interpretations of $<$ and $\text{Suc}$ naturally defined. The element 1 can be assigned to the existential contour, $EC_a$, because it is the only contour that can represent a non-empty set, and, since this contour is not the target of an arrow, it can contain the minimal element of $U$. This means that the element 2 is assigned to the contour $EC_b$, since this set must contain precisely the successors of $EC_a$. The contour $EC_b$ contains a zone that is not within the source of an arrow, so 2 can be the maximal element, as required. The extension of $\Psi$ to existential contours as described (with the remaining existential contours representing appropriate sets) can be shown to be a valid extension and, thus, $I$ models $d_4$. Hence $ab \in \mathcal{L}(d_4)$. Working through similar reasoning gives $abab \in \mathcal{L}(d_4)$, where $U = \{1, 2, 3, 4\}$, $\Psi(A) = \{1, 3\}$, $\Psi(EC_a) = \{1, 3\}$, $\Psi(EC_b) = \{2, 4\}$, $\Psi(EC_s) = \{2\}$, and $\Psi(EC_t) = \{3\}$.

There are various considerations that arise during the construction process that were not exemplified by this simple example. These arise from the presence of $\emptyset$ and $\lambda$: as noted earlier, $\lambda$ and $\emptyset$ will be dealt with in their own unitary diagrams. Moreover, we must also consider how to deal with subexpressions of the form $a^+$, where $a \in \Sigma$. We will give more examples throughout this section that illustrate the difficulties surrounding these cases.

Figure 11: Constructing a diagram for $r = (ab)^+$.

7.1. Diagram Transformations

For now, we set up a series of diagram transformations, akin to as seen in [?] for constraint diagrams [1]. The first transformation places an existential contour inside a zone, as seen for $EC_b$ added to $d_2$ in the zone identifier $\emptyset$, in
figure 11. For what we need, we take the zone to be inside some specified set of fixed contours but outside all other contours in the diagram, although our transformation is set up more generally. We use this transformation to add contours for letters, a base case of our inductive construction. So, a contour to be added for the letter $a_i$ will be placed in the zone $(fc(a_i), FC - fc(a_i))$.

**Definition 14.** Let $d$ be a unitary diagram and let $zi' \in ZI(d)$. Let $EC \in EC - C(d)$. We define $d +_i (EC, zi') = (C, ZI, ShZI, S, \eta, SucA)$ to be the diagram obtained by inserting $EC$ into the zone $(zi', C(d) - zi')$, specified by

1. $C = C(d) \cup \{EC\}$
2. $ZI = ZI(d) \cup \{zi' \cup \{EC\}\}$
3. $ShZI = ShZI(d)$
4. $S = S(d)$
5. for each $s \in S$, $\eta(s) = \eta_d(s)$
6. $SucA = SucA(d)$.

In figure 12, $d_1$ is a Venn diagram. The diagram $d_2$ is $d_1 +_i (EC, \{A\})$ with an added spider inside the new curve; a transformation for adding spiders is introduced later (definition 19). In figure 11, $d_3$ is obtained from $d_2 +_i (EC_b, fc(b))$ (by adding the arrow $(EC_b, EC_b)$).

![Figure 12: The construction $+_i$.](image)

We now describe the construction for a single letter raised to a power. If an arrow is interpreted in a model (i.e. is sourced on a contour which necessarily represents a non-empty set), then any word defined by such a diagram must have at least two letters. Thus, we would not be able to define the language containing a single letter word if such a contour was the source of an arrow: every element in the set for that contour would have a successor. We see one solution to this as diagram $d_1$ in figure 13. We require a two footed spider: one foot should be in a zone which is not the source of an arrow (and so can represent the single letter word $a$, for instance), and one foot should be in a zone which is the source of an arrow (from which we build up longer words consisting of $as$). Moreover, this arrow should have, as its target, another contour which has two zones, one of which is not inside the source of an arrow, and another which is inside an arrow source. In effect, we require a set of three contours which forms a Venn-3 diagram, two of which are connected by an arrow. Formally:
**Definition 15.** Let \( d \) be a unitary diagram and let \( zi' \in ZI(d) \). Let \( EC = \{EC_1, EC_2, EC_3\} \subseteq EC - C(d) \). We define

\[
d +_i (EC_1, EC_2, EC_3, zi') = (C, ZI, ShZI, S, \eta, SucA)
\]

to be the diagram obtained by inserting \( EC_1, EC_2, \) and \( EC_3 \) into the zone identifier \( zi' \), and inserting the two-footed spider \( s \) into the diagram, specified by

1. \( C = C(d) \cup EC \)
2. \( ZI = ZI(d) \cup \{zi' \cup in : in \subseteq EC\} \)
3. \( ShZI = ShZI(d) \cup \{zi' \cup EC, zi' \cup \{EC_1, EC_2\}, zi' \cup \{EC_1\}, zi' \cup \{EC_2\}, zi' \cup \{EC_3\}\} \)
4. \( S = S(d) \cup \{s\} \)
5. \( \eta = \eta(d) \cup \{(s, \{zi' \cup \{EC_1\}, zi' \cup \{EC_2\}\})\} \)
6. \( SucA = SucA(d) \cup \{(EC_2, EC_3)\} \).

This is a complicated definition. In reality, it is simply the diagram \( d_1 \) in figure 13 inserted into the zone contained by the contours in \( zi' \).

![Figure 13: The construction +_i.](image)

So, we now have diagram transformations that allows us to add contours for letters and letters raised to a power, by inserting contours into appropriate zones. The next transformation (definition 17) adds an existential contour that ‘splits’ all zones inside a specified set of contours. We need this to create diagrams for \( d \)-regular expressions of the form \( r^+ \): we need to be able to go from any possible “finishing” letter to any possible “starting” letter. For example, we need to be able to go from an \( a \) or a \( b \) to a \( c \) or a \( d \) in the expression \((c^+d^+) \cdot (a^+b^+)^+\). Adding a contour that appropriately splits zone identifiers for \( a \) or \( b \) thus provides a source contour for a successor arrow to a similar contour for \( c \) or \( d \). This type of transformation was seen in figure 11, when going from \( d_3 \) to
The two new existential contours each split two zones in $d_4$, those contained by $EC_a$ and $EC_b$ respectively.

The transformation of definition 17 has a further argument, $i$, which specifies which of the newly created zones are to be shaded. Any zone contained in two source contours potentially causes a contradiction. For example, suppose there exists an unshaded zone, $z$, which is contained in two source contours, $S_1$ and $S_2$, whose arrows target different contours, $T_1$ and $T_2$, respectively, and further assume that $T_1$ and $T_2$ are disjoint. Any element in the set represented by $z$ would thus be the immediate predecessor of an element in $T_1$, whilst simultaneously being the immediate predecessor of an element in $T_2$. Since $T_1$ and $T_2$ are disjoint, this represents a contradiction.

GS: I can’t understand this reworded paragraph. For instance, what does it mean to introduce an ambiguity? I really don’t get it. Sorry.

Similarly, any zone which is contained in two target contours needs to be shaded. When we define the transformation, we shade new zones that are inside just a single source (resp. target) since we will be adding a new arrow sourced (targeted) on this new contour when constructing a diagram for a $d$-regular expression. We define the following:

**Definition 16** (Source and Target sets). Given a diagram $d$, the **source set** for $d$, denoted $\text{sour}(d)$, is:

$$\text{sour}(d) = \{ s \in C(d) : (s, t) \in \text{SucA}(d) \text{ for some } t \}.$$ 

Similarly, the **target set** for $d$, denoted $\text{tar}(d)$, is:

$$\text{tar}(d) = \{ t \in C(d) : (s, t) \in \text{SucA}(d) \text{ for some } s \}.$$ 

For example, in figure 11, we have $\text{sour}(d_4) = \{ EC_s, EC_a \}$, and $\text{tar}(d_4) = \{ EC_t, EC_b \}$.

**Definition 17.** Let $d$ be a unitary diagram and let $ZI' \subseteq ZI(d)$. Let $EC \in EC - C(d)$, and $i \in \{0, 1\}$. We define $d_{+,i}(EC, ZI', i) = (C, ZI, ShZI, S, \eta, \text{SucA})$ to be the diagram obtained by adding $EC$ to $d$ so that it splits all zone identifiers in $ZI'$, specified by

1. $C = C(d) \cup \{ EC \}$
2. $ZI = ZI(d) \cup \{ z \cup \{ EC \} : z \in ZI' \}$
3. • if $i = 0$, then $\text{ShZI} = \text{ShZI}(d) \cup \{ z \cup \{ EC \} : z \in ZI' \land z \cap \text{tar}(d) \neq \emptyset \}$
   • if $i = 1$, then $\text{ShZI} = \text{ShZI}(d) \cup \{ z \cup \{ EC \} : z \in ZI' \land z \cap \text{sour}(d) \neq \emptyset \}$
4. $S = S(d)$
5. $\eta = \eta(d)$
6. $\text{SucA} = \text{SucA}(d)$.

Consider the diagram $d_1$ in figure 14. Assuming the two unlabelled contours are $EC_1$ and $EC_2$, the diagram has the following zone identifiers: $\{ A \}$,
\[ \{B\}, \{A, B\}, \{A, EC_1, EC_2\}, \{A, EC_1\}, \{A, EC_2\} \text{ and } \emptyset. \] Then, adding the contour \( EC_3 \) to split any zone identifier which contains \( EC_1 \) or \( EC_2 \) (so \( ZI' = \{\{A, EC_1\}, \{A, EC_2\}, \{A, EC_1, EC_2\}\} \)) as an interior contour creates the following zone identifiers:

- \( \{A, EC_1, EC_2, EC_3\} \)
- \( \{A, EC_1, EC_3\} \)
- \( \{A, EC_2, EC_3\} \).

In \( d_2 \) in figure 14, let \( EC_1 \) be the source contour and \( EC_2 \) be the target contour. Then, adding \( EC_3 \) with \( ZI' = \{EC_2, B\} \) and \( i = 0 \) gives the diagram \( d_2 +_s (EC_3, \{EC_2, B\}, 0) \). Because \( i = 0 \), we shade the newly created zone.

Next, we need a transformation to add arrows.

**Definition 18.** Let \( d \) be a unitary diagram and let \( (s, t) \in (C(d) \cup S(d))^2 \). We define \( d +_a (s, t) \) to be the diagram obtained by adding the arrow \( (s, t) \) to \( d \), that is

\[ d +_a (s, t) = (C(d), ZI(d), Sh(d), S(d), \eta_d, SucA(d) \cup \{(s, t)\}). \]
Figure 15 demonstrates the transformation $+a$: the two existential contours are joined by an arrow.

Finally, we need a transformation to place a spider in a specific region. Specifically, we give a transformation which ensures every diagram contains exactly one spider, and that the spider has a given (shaded) location. We say a spider is in a zone identifier, $z$, if its habitat contains $z$.

**Definition 19.** Let $d$ be a unitary diagram, and let $ZI' \subseteq ZI(d)$ be a set of zone identifiers. The diagram with a unique spider, $s$, in $ZI'$, denoted $d +_{sp} (s, ZI')$, is defined as:

$$d +_{sp} (s, ZI') = (C(d), ZI(d), Sh(d) \cup ZI', \{s\}, \{(s, ZI')\}, SucA(d))$$

This transformation deletes all spiders from $d$, and adds the spider $s$, simultaneously shading the habitat of the spider. The diagrams in figure 16 shows adding a two footed spider to the specified zones.
7.2. Constructing diagrams for \(d\)-regular expressions

In this section, we first present the construction algorithm, then provide diagrammatic instances of the various stages in the construction, and finally show a non-trivial example of how the algorithm works. The following definition uses the previously described transformations to construct a diagram for \(d\)-regular expressions.

**Definition 20.** Let \(d\) be a fully shaded unitary diagram with contour set \(C(d) = FC\), with no missing zones, no spiders, and no arrows. Let \(r\) be a \(d\)-regular expression. We define the diagram constructed for \(r\), denoted \(d + r\), the start-zone identifiers for \(d + r\), denoted \(\alpha(d + r)\), and the end-zone identifiers for \(d + r\), denoted \(\omega(d + r)\), inductively as follows:

1. Base cases:
   (a) If \(r = \emptyset \) then \(d + r = \bot\).
   (b) If \(r = \lambda \) then \(d + r = d\), and \(\alpha(d + r) = \omega(d + r) = \emptyset\).
   (c) If \(r = a\) for some \(a \in \Sigma\) then \(d + r = d + i \cdot (EC, fc(a)) + sp(s, \{EC\})\), and \(\alpha(d + a) = \omega(d + a) = \{\{EC\} \cup fc(a)\}\).
   (d) If \(r = a^+\) for some \(a \in \Sigma\) then \(d + r = d + i^+ \cdot (EC_1, EC_2, EC_3, fc(a))\), and

\[
\begin{align*}
\alpha(d + a^+) &= \{\{EC_1\} \cup fc(a), \{EC_2\} \cup fc(a)\} \text{ and } \\
\omega(d + a^+) &= \{\{EC_1\} \cup fc(a), \{EC_3\} \cup fc(a)\}.
\end{align*}
\]

2. Inductive cases:
   (a) If \(r = r_1|r_2\) then

\[
\begin{align*}
d + r &= d + r_1 + r_2 + sp(s, \alpha(d + r)),
\end{align*}
\]

and \(\alpha(d+r) = \alpha(d+r_1) \cup \alpha(d+r_2)\) and \(\omega(d+r) = \omega(d+r_1) \cup \omega(d+r_2)\).
   (b) If \(r = r_1 \cdot r_2\) then:

\[
\begin{align*}
d + r &= (d + r_1 + r_2) + s(\{EC_1, \omega(d + r_1), 0\} \\
&\quad + s(\alpha(d + r_2) + a, EC_1, EC_2) \\
&\quad + sp(s, \alpha(d + r_1))\}
\end{align*}
\]

where \(EC_1\) and \(EC_2\) are existential contours that are not in \(d + r_1 + r_2\), \(s\) is a spider, and

\[
\begin{align*}
\alpha(d + r) &= \alpha(d + r_1), \text{ and } \omega(d + r) = \omega(d + r_2).
\end{align*}
\]
(c) If $r = r_1^+$, then:

$$d + r = (d + r_1) + s(EC_1, \omega(d + r_1), 0) + s(EC_2, \alpha(d + r_1), 1) + (EC_1, EC_2)$$

where $EC_1$ and $EC_2$ are existential contours not in $d + r_1$, and

$$\alpha(d + r) = \alpha(d + r_1) \text{ and } \omega(d + r) = \omega(d + r_1)$$

Diagrammatically, this creates the instances such as those in figures 17, 18, 19 and 20, where $\mathcal{FC} = \{A, B\}$, $\Sigma = \{a, b, c, d\}$, and the letter assignment is given by $l(A) = \{a, b\}, l(B) = \{b, c\}$. In the latter three cases, the instances of the construction rules are displayed in a natural deduction style.

![Diagram](image1.png)

Figure 17: The base constructions

![Diagram](image2.png)

Figure 18: The inductive constructions

This construction creates a lot of redundancy, however the resulting diagrams are not designed to be particularly usable, rather it makes the proof of correctness straightforward. We are showing that second-order spider diagrams
Interesting properties of diagrams arise when we use the construction for $a^+$: it may be difficult to present concrete diagrams. Consider the following example: we wish to create a diagram for the expression $(a^+bc)^+$. Starting with a Venn-2 diagram with two contours $A$ and $B$, we assign letters to the zones as $fc(a) = \{A\}$, $fc(b) = \{A, B\}$ and $fc(c) = \{B\}$. The first two steps of the construction are straightforward: figure 21 shows the construction of $(a^+bc)$. 
However, when we come to ‘add the plus’, the construction stipulates that we add a contour to the end-zone identifiers. The end-zone identifiers are not topologically adjacent in this concrete diagram, thus any contour added to this diagram in the required manner will have some self-concurrency. An example is given in figure 22: there is a self-concurrent contour labelled $A$ (self-concurrent means that the contour intersects itself at a non-discrete set of points: here the self-concurrency is the line between the two circles). We will, to reduce clutter in diagrams, draw such a contour as several disconnected curves with a common label. In this example, the effect would be to remove the line between the two $A$ curves.
Using the assumption that we only need one arrow to join disconnected contours, we can then draw a diagram, without self-concurrent contours, for \((a^+|bc)^+\). This is shown in figure 23, where the new contours which will be a source are labelled \(s\), and the new contours which will be targets are labelled \(t\). So, the new arrow is \((s,t)\), as required. The diagram in figure 23 again highlights that we do not claim that the diagrams defined by the construction are readable. In fact, this can be typical of other processes that are devised to obtain meta-level results about a system of logic (in our case, we are deriving an expressiveness result). For instance, one often reasons about normal forms, which can often be unreadable, or far from ‘naturally constructed’ statements.

![Figure 23: A diagram for \((a^+|bc)^+\)](image)

### 7.3. Constructing diagrams for regular expressions

We have seen how to create a unitary diagram for a \(d\)-regular expression. As a subcase of this, we have seen how to construct a diagram for \(\lambda\). We showed in lemma 5 that any regular expression, \(r\), was equivalent to some \(d\)-regular expression, \(r'\), or equivalent to \(\lambda r'\). Thus, we can give a diagram for any regular expression: in the case where we only need a \(d\)-regular expression, we use the construction in definition 20, and in the case where we require a disjunction with \(\lambda\), we use the construction for \(\lambda\), the construction in definition 20, and the diagram connective \(\lor\) from definition 2.
7.4. Correctness of the construction

Given a \( d \)-regular expression \( r \), and the diagram, \( d + r \), two questions arise. Firstly, given any model, \( M \), for \( d + r \), is \( M \) also a model for some word in the language of \( r \)? In other words, is it the case that the language of \( d + r \) is a subset of that defined by \( r \)? Secondly, given a word in the language of \( r \), is it the case that the language of \( r \) is a subset of that defined by \( d + r \)? If both questions can be answered in the affirmative, then we can say that the diagram \( d + r \) defines the same language as the expression \( r \). We just give sketches for the proofs of these two theorems: the full proofs can be found in Appendix C.

**Theorem 6.** Let \( d \) be a fully shaded Euler diagram with contour set \( C(d) = FC \), with no missing zones, no spiders and no arrows. For any \( d \)-regular expression \( r \) and for all interpretations \( M \) such that \( M \models d + r \), there exists a word, \( w \), such that \( w \in L(r) \) and \( M \models w \). That is, \( d + r \) defines a language and \( L(d + r) \subseteq L(r) \).

**Proof Sketch.** Let \( r \) be given, and let \( M = (U, \Psi', <, Suc) \) be given such that \( M \models d + r \). We prove the theorem by induction on the structure of \( r \).

1. If \( r \equiv \emptyset \), then the construction gives \( d + r = \bot \). The theorem is trivially satisfied in this case.
2. If \( r \equiv \lambda \), then the construction gives a fully shaded Venn diagram for \( d + r \). Again, the theorem is trivially satisfied in this case.
3. If \( r \equiv a \), where \( a \in \Sigma \), then we place a spider and existential contour into the zone \( (fc(a), FC - fc(a)) \), meaning \( |U| = 1 \) and, moreover, the spider will represent the single-letter word \( a \), which is the only word in \( L(a) \), as required.
4. If \( r \equiv a^+ \), where \( a \in \Sigma \), then we place a spider and three existential contours into the zone \( (fc(a), FC - fc(a)) \). Thus, the only word which results from this diagram will consist entirely of \( a \)s, and will be at least one letter long (owing to the presence of the spider). Thus, the word defined will be \( a^+ \), which is a word in the language \( L(a^+) \), as required.
5. If \( r \equiv r_1 | r_2 \), then the construction puts both individual constructions in the same diagram, and connects the spider for \( r_1 \) with the spider for \( r_2 \). Since only one foot of the spider will be interpreted in \( M \), the rest of the spider’s habitat will then represent the empty set. Without loss of generality, assume that spider foot which is interpreted is the spider from \( d + r_1 \). Then, all of the existential contours from \( d + r_2 \) will have an empty interpretation. Thus, the model for \( d + r_1 | r_2 \) will be a model for \( d + r_1 \), and the induction hypothesis tells us that there exists a word \( w_1 \in L(r_1) \subseteq L(r_1 | r_2) \), as required.
6. If \( r \equiv r_1 \cdot r_2 \), then the construction places both individual constructions in the same diagram, and connects the end of the construction for \( r_1 \) to the beginning of the construction for \( r_2 \) using an arrow. Thus, we can split the model \( M \) into two smaller models \( M_1 \) and \( M_2 \) for \( d + r_1 \) and \( d + r_2 \).
respectively, where the universes \((U_1, U_2\) respectively) of these models partition the universe \(U\) according to \(<\): i.e. \(\forall u \in U_1, \forall v \in U_2, u < v\). Then, the induction hypothesis gives us two words, \(w_1 \in \mathcal{L}(r_1)\) and \(w_2 \in \mathcal{L}(r_2)\), so that when we glue the models back together again, we have \(M \models w_1 \cdot w_2\), and \(w_1 \cdot w_2 \in \mathcal{L}(r_1) \cdot \mathcal{L}(r_2) = \mathcal{L}(r_1 \cdot r_2)\), as required.

7. If \(r \equiv r_1^+\), then we perform a similar proof to that for concatenation. The universe \(U\) will be finite. Therefore, we take the set of elements which are assigned to the start zone identifiers of \(d + r_1\), and partition \(U\) according to these elements and \(<\). Taking each of these sub-sets, and restricting \(<\) appropriately, we will create sub-models for \(d + r_1\). The induction hypothesis will then give us a sequence of words \(w_i \in \mathcal{L}(r_1)\). When we glue these sub-models back together using \(M_1 + M_2 + \ldots + M_n\), we will obtain a word \(w_1 \cdot w_2 \cdot \ldots \cdot w_n \in \mathcal{L}(r_1) \cdot \mathcal{L}(r_1) \cdot \ldots \cdot \mathcal{L}(r_1) \subseteq \mathcal{L}(r_1^+),\) as required.

\[ \Box \]

**Theorem 7.** Let \(d\) be a fully shaded Euler diagram with no missing zones, no spiders, and no arrows. For any \(d\)-regular expression \(r\), any word \(w\) in \(L(r)\), and any interpretation, \(M \models w\), we have \(M \models d + r\). That is, \(L(r) \subseteq L(d + r)\).

**Proof Sketch.** Let \(r\) and \(w \in L(r)\) be given. Let \(M = (U, \Psi', <, Suc)\) be any model for \(w\). We show that \(M \models d + r\), by induction on the structure of \(r\) and then by verifying the conditions from definition 6. This is more straighforward than the proof for theorem 6: the induction hypothesis does most of the work in the inductive cases, and in the base cases there is at most one spider and one arrow, which means the spiders’ location and successor conditions are easy to verify.

\[ \Box \]

8. **Expressiveness**

Any regular expression can be encoded as a \(d\)-regular expression, or a disjunction of \(\lambda\) and a \(d\)-regular expression. It is easy to show that a disjunction of a fully shaded, spiderless and arrowless diagram (for \(\lambda\)) and a diagram for a \(d\)-regular expression \(r'\) models \(\lambda | r'\). Thus, using the two theorems, we have:

**Theorem 8 (Definability of regular languages).** Every regular language is definable by a second-order spider diagram.

Furthermore, we have seen from theorem 1 that regular expressions and MSOL are equivalent, and so:

**Corollary 9.** Second-order spider diagrams are at least as expressive as monadic second-order logic.
With the exception of diagrams involving $\lambda$, only the unitary fragment of second-order spider diagrams is needed for defining regular expressions. We have not used the connectives $\wedge$, $\exists$ and $\neg$ for any constructs. We have no results to say whether second-order spider diagrams using all the connectives can define statements which do not correspond to regular expressions. In some formalisms, such as classical first-order logic, there is some set of minimal connectives needed for expressiveness: all other connectives can be defined in terms of this minimal set ($\forall$, $\neg$ and $\wedge$ form such a set for classical first-order logic), and are thus syntactic sugar. It could be that the remaining connectives for second-order spider diagrams are simply syntactic sugar, or it could be that adding negation, for example, strictly increases expressiveness. This is an open question, and a problem for further study.

9. Discussion

In this paper, two choices were made: to use a minimal syntax for regular expressions, and to use spider diagrams of order as a basis for our second-order spider diagrams. In this section, we will justify those decisions.

Regular expressions can be written in a form which renders them inelegant (as was shown in §6). By translating inelegant expressions directly into diagrams, we therefore create diagrams which may be unintuitive for understanding. In other words, the method of proof of theorem 8 creates, as a side effect, diagrams which may be aesthetically unappealing. The notation itself, however, is more than capable of producing elegant diagrams. In particular, we can produce concise diagrams for some regular expressions which are syntactically complex when written in sentential form.

Consider figure 24. Diagram $d_1$ (where the lower case letters give the assignment of letters to zones) represents the regular expression $(a|b|c|d)^*ac(a|b|c|d)^*$. In an abuse of notation, we will write $\Sigma^*$ for $(a|b|c|d)^*$. If we add a pair of spiders joined by an arrow to $d_1$ to create $d_2$, the corresponding regular expression becomes much more syntactically complex. The regular expression for $d_2$ is $(\Sigma^*b\Sigma^*ac\Sigma^*)|(\Sigma^*ac\Sigma^*b\Sigma^*)$. If we remove syntax from $d_2$ we make it syntactically less complex. Removing the arrow, however, increases the syntactic complexity of the corresponding regular expression: we have removed order from the language. In this case, the regular expression encoded by $d_3$ is:

![Figure 24: Elegant diagrams for inelegant regular expressions](image-url)
Informally, call a series of spiders connected by (possibly 0) arrows a spider chain. Then, a single spider which is neither the source nor target of an arrow is a spider chain. A diagram for a regular expression \( r \) containing \( n \) spider chains will impose no order on the appearance of subwords represented by each spider chain in a word \( w \in L(r) \). Using a naïve, but allowable, form for \( r \) (like that in \( \ast \)) would then require \( n! \) disjunctive branches. Moreover, each disjunctive branch will become more complex, requiring \( n + 1 \) instances of a starred expression (in the example \( \Sigma^* \)) to separate the subwords represented by each spider chain.

We now have an explanation as to why removing an arrow from a diagram increases the complexity of the encoded regular expression: removing an arrow creates 2 spider chains exist where before there was 1, thus increasing the number of spider chains in the diagram.

Regular expressions impose an order on the appearance of subwords in a word, whereas second-order spider diagrams can display both ordered, and unordered information. The informal description of a regular language as “any word in the language contains at least 2 a’s, 2 b’s, and a c, in any order” will therefore be problematic to encode as a regular expression. It requires \( 5! = 120 \) disjunctive branches, each of which contains 6 instances of a starred word (here \( \Sigma^* \)). By contrast, the second-order spider diagram for the language would be visually simple, being an unshaded diagram with 5 spiders in it. Thus we have shown that second-order spider diagrams are much more visually efficient for some regular languages than the corresponding sentential representation.

To create second-order spider diagrams, we extended spider diagrams of order. These diagrams provided a natural starting point: they included a product operator \( \triangleleft \), and have been shown to be as expressive as star-free regular languages. Of course, the operator \( \triangleleft \) is not used during our constructions of diagrams from regular expressions but \( \triangleleft \) is still available for use in diagrams as are the other operators. One could instead use, therefore, spider diagrams as the starting point for extension. It could be argued that second-order spider diagrams are siblings, rather than children, of spider diagrams of order. However, whilst arrows can simulate \( \triangleleft \), there are cases where diagrams that do not use \( \triangleleft \) are not as appealing as those created using \( \triangleleft \). For instance, consider the regular expression \((ab)^+ \cdot (bc)^+\). Two encodings are given in figure 25. In \( d_2 \) we can clearly see the structure of the regular expression: there is a subword \((ab)^+\) which is concatenated with the subword \((bc)^+\), corresponding to the first diagram being concatenated with the second diagram. Being able to split the diagram into two sub-diagrams not only mirrors the regular expression, but alleviates the clutter in each individual sub-diagram. By contrast, in \( d_1 \), we
have many more curves, and no clear relationship between the diagram and the regular expression.

That we do not use \(\triangleleft\), or \(\land\), in proofs, raises an interesting question. Are they syntactic sugar? If that is the case then we can conclude that the language encoded by any diagram \(d\) would be regular, owing to the results of this paper. We strongly suspect that \(\triangleleft\) and \(\land\) are, in fact, syntactic sugar: we conjecture any second-order spider diagram will encode a regular language, and nothing more. Figure 25 demonstrates that having extra syntactic elements in a diagram, whilst they may not increase expressiveness, certainly makes it easier to encode regular languages using second-order spider diagrams.

10. Conclusion and Further Work

We have given a method for converting any regular expression into an equivalent \(d\)-regular expression. Furthermore, we have given a method for constructing a diagram for any \(d\)-regular expression. Thus, we have given a method for constructing a diagram for any regular expression. Since regular expressions and MSOL sentences are equally expressive, we have thus shown that second-order spider diagrams are at least as expressive as MSOL. There exists a translation between MSOL sentences and regular expressions. Thus, using our constructions, we could encode any MSOL sentence directly. However, we do not claim that this is the best way to encode MSOL sentences: we have given examples where relatively simple regular expressions are translated into quite complicated diagrams. However, we claim that the syntax of second-order spider diagrams can encode any MSOL sentence, but usability studies should be performed to find the best way of doing this.

Consider the example \(r = ((aa)|(bb))^+\). A construction for \(r\) can be seen as \(d_1\) in figure 26, where we have the disconnected contours \(s\) and \(t\). Note that the diagram \(d_2\) in figure 26 also defines \(((aa)|(bb))^+\), and moreover is a simpler diagram.

This demonstrates that our construction certainly does not produce ‘syntactically minimal’ diagrams or necessarily the most elegant or accessible diagrams. Rather, it produces diagrams that are essentially in a normal form that reflects the inductive manner in which \(d\)-regular expressions are built. Given a language
class that one would want to define, it is perhaps unlikely that the regular expression of choice would be $d$-regular. Equally, given a class of models that one wished to axiomatise, it would be equally unlikely that one would naturally construct a diagram like those in this kind of normal form. This is a common feature of normal forms that arise in other logics. For instance, whilst it is often an advantage to consider such forms when proving meta-level results about logics (such as restricting consideration to only sentences drawn from a syntactically minimal first-order logic), these are not typically in the most human-readable form. Future work would include investigating what features of these diagrams correspond with readability.

The trade-off which must be addressed in any usability study is inferential load versus readability. For instance, consider the diagrams $d_1$ and $d_2$ in figure 26. The diagram $d_1$ requires little inference on the part of the reader: every successor relationship is explicitly represented by an arrow, and thus any information needed can be read off the diagram. There is redundancy in this diagram, however: having two intersecting contours with shading to represent an empty intersection does not necessarily aid readability. Thus, it is more complicated than it needs to be. However, the diagram $d_2$ requires the reader to infer that there is an implicit arrow from the target contours back to the source contours. So, whilst the diagram is uncluttered, it is not as easy to reason about as $d_1$. For example, a response to the question “can this diagram encode the word $aabbaa$?” may be formulated more quickly for $d_1$ than $d_2$.

Another avenue of investigation is the use of reasoning rules for second-order spider diagrams. These could stipulate when it is allowable to remove a contour, for instance, or when it is allowable to move contours. As an example, if there were two contours $EC_1$ and $EC_2$ in a diagram, and the zone identifier $\{EC_1, EC_2\}$ was shaded and contained no spiders, then we could make $EC_1$ and $EC_2$ disjoint. In this example, any model set for the conclusion diagram (disjoint contours) will be a model set for the premiss diagram (shaded, spiderless intersection). Any such rule would require that the model sets of the premisses

Figure 26: Constructing $r = ((aa)((bb))^+$. 

\[ r = ((aa)((bb))^+ \]
of a rule are subsets of the model set of the conclusion. Such systems of rules have been developed for spider diagrams [19].

Acknowledgement This research is supported by EPSRC grants for the Defining Regular Languages with Diagrams project [EP/H012311/1] and the Sketching Euler Diagrams project [EP/H048480/1].

Appendix A. An Overview of Spider Diagrams

Here we briefly introduce some of the core concepts from spider diagrams on which this paper builds. For a full account see [13], although [7] gives an account that more closely matches that presented here. For spider diagrams of order see [8]. We focus here on the abstract syntax, rather than the concrete syntax, since this is what is necessary for this paper. Before we can define the syntax of a spider diagram, we need a countably infinite set, \( \mathcal{L} \), of labels which will be used for the curves. Using \( \mathcal{L} \), we can define a set, \( \mathcal{Z} \), of zones, where each zone, \((\text{in}, \text{out})\) is a pair of finite, disjoint sets of labels. Additionally, we also have a set, \( \mathcal{S} \), whose elements are called spiders.

Definition 21. A unitary spider diagram is a tuple, \( d = (L, Z, ShZ, S, \eta) \) where

1. \( L \subseteq \mathcal{L} \) is a finite set of labels.
2. \( Z \) is a set of zones such that for each \((\text{in}, \text{out}) \in Z\), \( \text{in} \cup \text{out} = L \).
3. \( ShZ \) is a subset of \( Z \), whose elements are called shaded zones.
4. \( S \) is a finite set of spiders.
5. \( \eta: S \rightarrow \mathcal{P}Z - \{\emptyset\} \) is a function that returns the habitat of each spider.

The symbol \( \bot \) is also a unitary spider diagram.

For example, in figure 1, the spider diagram \( d_2 \) has abstract syntax as follows:

1. \( L = \{A, B\} \),
2. \( Z = \{\emptyset, \{A, B\}\}, \{\{A\}, \{B\}\}, \{\{B\}, \{A\}\}, \{\{A, B\}, \emptyset\}\}, \)
3. \( ShZ = \emptyset \),
4. \( S = \{s_1, s_2\} \), and
5. \( \eta(s_1) = \{\{A\}, \{B\}\} \) and \( \eta(s_2) = \{\{A, B\}, \emptyset\} \).

At the concrete level, the spiders are represented by trees, with one node placed in each zone of the spider’s habitat. The nodes are called feet and the edges are called legs.

Unitary diagrams form the building blocks of more complex diagrams, formed using standard logical operators.

Definition 22. A spider diagram is defined inductively:

1. Any unitary spider diagram is a spider diagram.
2. If \( d_1 \) is a spider diagram then \( \neg d_1 \) is a spider diagram.
3. If \( d_1 \) and \( d_2 \) are spider diagrams then:
(a) \(d_1 \land d_1\), and
(b) \(d_1 \lor d_2\)
are spider diagrams.

In spider diagrams of order, the operator \(<\) also joins to spider diagrams together, in the same fashion as \(\land\) and \(\lor\).

In order to define the semantics of spider diagrams, we need to identify the so-called missing zones of unitary diagrams. Zones represent sets and missing zones are required to represent the empty set.

**Definition 23.** Given a unitary spider diagram, \(d = (L, Z, SHR, S, \eta)\), a zone, \((in, out)\), is missing from \(d\) if it is in the set \(MZ(d) = \{(in \in L - in) : in \subseteq L\} - Z\).

The semantics of spider diagrams are model based. To begin, we define an interpretation that maps labels to subsets of some universal set.

**Definition 24.** An interpretation is a pair, \(I = (U, \Psi)\) where \(U\) is a set and \(\Psi: L \rightarrow \mathcal{P}U\) maps labels to subsets of \(U\). The function \(\Psi\) extends to zones and sets of zones as follows:

1. for each zone, \((in, out)\), we define
   \[
   \Psi(in, out) = \bigcap_{i \in in} \Psi(l) \cap \bigcap_{i \in out} (U - \Psi(l)),
   \]
2. for each set of zones, \(Z\), we define
   \[
   \Psi(Z) \cup \bigcup_{z \in Z} \Psi(z).
   \]

For example, taking \(U = \{1, 2, 3\}\) we can define an interpretation, \(I\), by mapping labels to subsets of \(U\). Assuming \(\{A, B, C\} \subseteq L\), we can define \(\Psi\) as follows:

1. \(\Psi(A) = \{1, 2, 3\}\),
2. \(\Psi(B) = \{2\}\),
3. \(\Psi(C) = \{2, 3\}\), and
4. for all \(l \in L - \{A, B, C\}\), \(\Psi(l) = \emptyset\)

To illustrate the extension of \(\Psi\) to zones, we have

\[
\Psi(\{A\}, \{B\}) = \Psi(A) \cap (U - \Psi(B)) = \{1, 2, 3\} \cap \{1, 3\} = \{1, 3\}.
\]

We now define when an interpretation agrees with the intended meaning of a spider diagram. Such ‘agreeing’ interpretations are called models.

**Definition 25.** Let \(d\) be a spider diagram and let \(I = (U, \Psi)\) be an interpretation. If \(d\) is a unitary diagram and \(d \neq \bot\) then \(I\) is a model for \(d\) provided the following conditions hold.

39
1. **The missing zones condition.** The missing zones represent the empty set, i.e.
\[ \bigcup_{z \in MZ(d)} \Psi(z) = \emptyset. \]

2. There exists an extension of \( \Psi \) to spiders in \( S \) such that for each \( s \in S \), 
\[ |\Psi(s)| = 1 \] and the following conditions hold:

   (a) **The spiders’ distinctness condition.** If two spiders have the same interpretation then they are the same spider:
   \[ \forall s_1, s_2 \in S(d), (\Psi(s_1) = \Psi(s_2) \Rightarrow s_1 = s_2). \]

   (b) **The shaded zones condition.** All elements in the sets represented by shaded zones are represented by spiders:
   \[ \forall z \in Sh(d). \Psi'(z) \subseteq \bigcup_{s \in S(d)} \Psi(s). \]

   (c) **The spiders’ location condition.** The elements represented by the spiders are in the sets represented by their habitats:
   \[ \forall s \in S(d). \Psi(s) \subseteq \bigcup_{z \subseteq \eta(s)} \Psi(z). \]

If \( d = \bot \) then \( I \) is not a model for \( d \). The definition of a model extends in the obvious inductive way for non-unitary spider diagrams.

The interpretation just defined, where we took \( U = \{1, 2, 3\} \) is a model for \( d_2 \) in figure 1, which can be shown by defining \( \Psi(s_1) = \{1\} \) and \( \Psi(s_2) = \{2\} \). In this example, the spiders in \( d_2 \) both inhabit single zones. If a spider inhabits more than one zone then that spider provides disjunctive information. For instance, if a spider \( s_3 \) was added to \( d_2 \) such that \( \eta(s_3) = \{\{(\{A\}, \{B\}), \{\{B\}, \{A\}\}\} \) then the conditions on \( I \) to be a model for the resulting diagram requires there to be a choice of element, \( e \), in \( U \) such that \( \Psi(s_3) = \{e\} \) and
\[ \Psi(s_3) = \{e\} \subseteq \Psi(\{A\}, \{B\}) \cup \Psi(\{B\}, \{A\}). \]

That is,
\[ \Psi(s_3) \subseteq \Psi(\{A\}, \{B\}) \lor \Psi(s_3) \subseteq \Psi(\{B\}, \{A\}). \]

**Appendix B. Basic Definitions: Regular Languages**

Here, we provide a brief overview of the basic concepts from formal language theory that have been used throughout the paper. Recall a (not necessarily regular) language over an alphabet \( \Sigma \) is a subset of \( \Sigma^* \) (all the words formed from letters in \( \Sigma \), where a word is simply a sequence of letters drawn from the alphabet). A language is **regular** if it can be defined by a regular expression [15]. **Regular expressions** are formed using letters in \( \Sigma \), \( \lambda \) to denote the empty word,
∅ to denote the empty language, disjunction (denoted |) concatenation (denoted ·) and the Kleene star, *.
In addition, generalised regular expressions make use of ¯ to denote language complements, but for our purposes we do not include the complement operator. Formally, we have the following definitions.

**Definition 26 (Regular Expression).** Given an alphabet Σ, a regular expression over Σ is defined as:

\[
 r = \begin{cases} 
 \emptyset \\
 \lambda \\
 a \in \Sigma \\
 r \cdot r \\
 r|r \\
 r^*.
\end{cases}
\]

For example, given \( \Sigma = \{a, b, c, d\} \), \( r = ab^* \) is a regular expression. It defines the regular language comprising all words that begin with the letter a, contain no further as, no cs and no ds. So the language defined by \( r \) includes the word abbbb but not the word aba. We call a regular expression star-free if it contains no instances of the Kleene star.

Given an expression, we build the language defined by the expression. First, given two languages \( L_1 \) and \( L_2 \), we define \( L_1 \cdot L_2 \) to be the set of words

\[
 L_1 \cdot L_2 = \{w_1w_2 : w_1 \in L_1 \land w_2 \in L_2\}.
\]

Given a language \( L \), we define \( L^* \) to be

\[
 L^* = \bigcup_{i \in \mathbb{N}-\{0\}} L_i
\]

where \( L_1 = L \cup \{\lambda\} \), and for each \( i \),

\[
 L_{i+1} = L_i \cup \{w_1...w_{i+1} : \forall j, 1 \leq j \leq i + 1, w_j \in L_i\}.
\]

Clearly, then, \( \Sigma^* \) comprises all the words over the alphabet Σ. Equivalently, we can think of \( L^* \) as the free monoid whose generators are drawn from \( L \), under the binary operation of concatenation and with identity element \( \lambda \).

**Definition 27 (Language of a regular expression).** The language of a regular expression, \( r \), is defined by induction on the structure of \( r \):

\[
\begin{align*}
\mathcal{L}(\emptyset) &= \emptyset \\
\mathcal{L}(\lambda) &= \{\lambda\} \\
\mathcal{L}(a) &= \{a\} \\
\mathcal{L}(r_1 \cdot r_2) &= \mathcal{L}(r_1) \cdot \mathcal{L}(r_2) \\
\mathcal{L}(r_1|r_2) &= \mathcal{L}(r_1) \cup \mathcal{L}(r_2) \\
\mathcal{L}(r^*) &= \mathcal{L}(r)^*.
\end{align*}
\]

**Definition 28 (Regular Language).** A language, \( L \), is regular iff there exists a regular expression, \( r \), such that \( \mathcal{L}(r) = L \).
Appendix C. Proofs

Here we give the proofs of theorems 6 and 7.

**Theorem.** Let $d$ be a fully shaded Euler diagram with contour set $C(d) = FC$, with no missing zones, no spiders and no arrows. For any $d$-regular expression $r$ and for all interpretations $M$ such that $M \models d + r$, there exists a word, $w$, such that $w \in L(r)$ and $M \models w$. That is, $d + r$ defines a language and $L(d + r) \subseteq L(r)$.

**Proof.** Let $r$ be given, and let $M = (U, \Psi, <, Suc)$ be given such that $M \models d + r$. Choose a valid extension of $M$, say $(U, \Psi', <, Suc)$. We prove the theorem by induction on the structure of $r$.

1. If $r \equiv \lambda$, then from the construction $d + \lambda$ we conclude that $U = \emptyset$. The only possible word which $M$ can model, therefore, is $\lambda$. Indeed, $M$ does model $\lambda$ and, trivially, $\lambda \in L(\lambda)$, so $L(d + \lambda) \subseteq L(\lambda)$.

2. If $r \equiv \emptyset$, then the construction $d + r = \bot$, and this has no models. Hence $L(d + \emptyset) \subseteq L(\emptyset)$.

3. If $r \equiv a$, for some $a \in \Sigma$, then the diagram $d + a$ is fully shaded, with a single spider, $s$, in the zone identifier $\{fc(a) \cup \{EC_1\}\}$, for some existential contour $EC_1$. We prove that the model, $M$, for $d + a$ is also a model for the word $a$, which is clearly an element of $L(a) = \{a\}$. Now, since $d + a$ is fully shaded and contains a single spider, we deduce from the shaded zones condition that $U = \{a\}$, for some element $u$. In particular, we know that

$$\Psi'(s) = \{u\} \subseteq \Psi'(fc(a) \cup \{EC_1\}, FC - fc(a)) \tag{C.1}$$

by the spiders’ location condition. We define $f: \{a\} \rightarrow \{u\}$, trivially, by $f(a) = u$. Clearly, $u$ is the first (and only) element of the total ordering induced by $<$. In addition, we have

$$f(a) \in \Psi'(fc(a) \cup \{EC_1\}, FC - fc(a)) \subseteq \Psi(fc(a), FC - fc(a)).$$

Hence $M \models a$ and therefore the only word in the language of $d + a$ is $a$. That is, $L(d + a) = L(a)$, whence $L(d + a) \subseteq L(a)$.

4. If $r \equiv a^+$ for some $a \in \Sigma$, then the diagram $d + a^+$ is fully shaded except for two zone identifiers, $zi_1 = \{fc(a) \cup \{EC_1, EC_2\}\}$ and $zi_2 = \{fc(a) \cup \{EC_2, EC_3\}\}$ for some existential contours $EC_1, EC_2$ and $EC_3$. Using lemma 4, we know there exists some word $w$ such that $M \models w$.

Let $w = w_1 \ldots w_n \in L(d + a^+)$ be given. We show this word consists only of $a$. We know $M \models w$ and $M \models d + a^+$. From the construction, let $zi_1 = fc(a) \cup \{EC_1\}, zi_2 = fc(a) \cup \{EC_2\}, zi_3 = fc(a) \cup \{EC_2, EC_3\}$ and $zi_4 = fc(a) \cup \{EC_1, EC_3\}$. Then, the spider $s$ in $d + a^+$ has habitat $\eta(s) = \{zi_1, zi_2\}$. By the spiders’ location condition, this means:
If $\Psi$ consider the former, the latter is symmetrical. Because $\Psi'$ hence, every letter $M$ where $z$ where $\{\psi_1, \ldots, \psi_n\}$ is singleton, and the two sets of start zone identifiers are disjoint, that $\Psi(\psi_1) \subseteq \Psi(\psi_2)$ we can show $\Psi(z \subseteq \Psi(\psi_1) \subseteq \Psi(\psi_2)$ and the zones corresponding to $z_1$ and $z_2$ respectively: $d + a^+$ contains precisely one spider which inhabits $\{z_1, z_2\}$, and the shaded zones condition says every element in a shaded zone is represented by a spider. Then, since $\Psi'(s) \subseteq \Psi'(z_1, z_2)$, we deduce $\Psi'(z_1, z_2) = \Psi'(s)$. We have $\psi(\psi_i(\psi_i) \subseteq \Psi'(z_1, z_2)$, so $\{\psi(\psi_i)\} = \Psi'(s)$. Using (C.2), we have $\psi(\psi_i(\psi_i) \in \Psi(\psi(\psi_i), \mathcal{FC} - \psi(\psi_i))$. This means, using (C.3):

$\Psi(\psi(\psi_i), \mathcal{FC} - \psi(\psi_i)) \cap \Psi(\psi(\psi_i), \mathcal{FC} - \psi(\psi_i)) \neq \emptyset$

Distinct zones over $\mathcal{FC}$ represent disjoint sets; this implies $\psi(\psi_i) = \psi(\psi_i)$. We know $\psi(\psi_i)$ is injective, thus $a = \psi_i$, as required.

(b) If $z_{\psi_i}$ is unshaded, then $z_{\psi_i} = z_3$ or $z_{\psi_i} = z_4$. We can show $\Psi'(z_3) \subseteq \Psi(\psi(\psi_i), \mathcal{FC} - \psi(\psi_i))$ and $\Psi'(z_4) \subseteq \Psi(\psi(\psi_i), \mathcal{FC} - \psi(\psi_i))$. But, $\psi(\psi_i(\psi_i) \in \Psi'(z_{\psi_i})$, and so $\psi_i$ is an $a$, as required.

Hence, every letter $\psi_i$ of $w$ is an $a$, so $w \in \mathcal{L}(a^+)$, as required.

5. If $r \equiv r_1 | r_2$, then $M \models d + (r_1 | r_2)$. From the construction, we have a spider, $s$, in the start zone identifiers of $d + r_1$ and the start zone identifiers of $d + r_2$. Thus, $\Psi'(s) \subseteq \Psi(\alpha(d + r_1))$ or $\Psi'(s) \subseteq \Psi(\alpha(d + r_2))$. Since $\Psi(s)$ is singleton, and the two sets of start zone identifiers are disjoint, and by construction the start zone identifiers are shaded, then:

(a) $\Psi'(s) = \Psi(\alpha(d + r_1))$ and $\Psi'(\alpha(d + r_2)) = \emptyset$, or

(b) $\Psi'(s) = \Psi(\alpha(d + r_2))$ and $\Psi'(\alpha(d + r_1)) = \emptyset$.

Consider the former, the latter is symmetrical. Because $\Psi'(\alpha(d + r_2)) = \emptyset$, we can deduce that every zone identifier constructed for $d + r_2$ has an empty interpretation. This is shown by induction on the structure of $r_2$ (the cases for $r_2 \equiv \lambda$ and $r_2 \equiv \emptyset$ are trivial):
(a) \( r_2 \equiv a \). Then, \( d + r_2 \) adds only a single existential contour \( EC \), which is fully shaded, thus \( \alpha(d + r_2) = \{ EC \} \). We know, from (a), that \( \Psi(\alpha(d + r_2)) = \emptyset \), as required.

(b) \( r_2 \equiv a^+ \). Then, every zone identifier in \( d + r_2 \) is fully shaded, except for a pair of unshaded zone identifiers \( zi_1 \) and \( zi_2 \) (as in case 4). We need to show \( \Psi'(zi_1) = \Psi'(zi_2) = \emptyset \). By the successor condition, we know \( \text{Suc}[\Psi'(zi_1)] = \Psi'(zi_1) \cup \Psi'(zi_2) \), and thus \( |\Psi'(zi_1)| = |\Psi'(zi_1)| + |\Psi'(zi_2)| \). This can only be satisfied if \( \Psi'(zi_2) = \emptyset \). However, this gives \( \text{Suc}[\Psi'(zi_1)] = \Psi'(zi_1) \), or rather states that every element in \( \Psi'(zi_1) \) has a successor. Since we use a finite universe, this is impossible unless \( \Psi'(zi_1) = \emptyset \), as required.

(c) \( r_2 \equiv r'_2 \)\( r'_2 \). The induction hypothesis deals with this case: \( d + r'_2 \) and \( d + r'_2 \) both have empty interpretations, so \( d + r_2 \) has an empty interpretation.

(d) \( r_2 \equiv r'_2 \cdot r''_2 \). Again, the induction hypothesis deals with this case: it yields that \( d + r'_2 \) and \( d + r''_2 \) have empty interpretations, so \( d + r_2 \) has an empty interpretation.

(e) \( r_2 \equiv r''_2 \). The start zone identifiers, by construction, will consist of a contour \( EC_1 \) added to the start zone identifiers with the feet of \( s \). Since these feet of \( s \) are not interpreted, then \( \Psi'(\alpha(d + r_2)) = \Psi'(EC_1) \). Consider the end zone identifiers of \( d + r_2 \). By construction, an existential contour \( EC_2 \) was added to these zone identifiers, and the successor condition gives us \( \text{Suc}[\Psi'(EC_1)] = \Psi'(EC_1) \). However, by the bijectivity of other arrows in the construction, we have \( |\Psi'(EC_1)| = |\Psi'(\omega d + r_2)| \), whence \( \Psi'(EC_1) = \Psi'(\omega d + r_2) \). Thus, every element has a successor, which is again not possible unless \( \Psi'(EC_1) = \emptyset \), and every other zone identifier has empty interpretation, as required.

From this, we deduce \( M \models d + r_1 \): it is the only part of the diagram with a non-empty interpretation. Then, the induction hypothesis gives us \( M \models w_1 \) for some \( w_1 \in \mathcal{L}(r_1) \). However, \( w_1 \in \mathcal{L}(r_1) \cup \mathcal{L}(r_2) = \mathcal{L}(r_1 \cdot r_2) = \mathcal{L}(r) \), as required.

6. If \( r \equiv r_1 \cdot r_2 \), then \( M \models d + r_1 \cdot r_2 \). Any universe \( U \) can be partitioned into two components, \( U_1 \) and \( U_2 \), such that the maximum element of \( U_1 \) has as its successor the minimum element of \( U_2 \) and both \( U_1 \) and \( U_2 \) are closed under successor for every other element. We select the partition by taking \( U_1 \) to be the union of the interpretations of existential contours created to construct \( r_1 \), and \( U_2 \) will then be the union of the interpretations of existential contours created to construct \( d + r_2 \). We know this forms a partition of \( U \) because \( M \) is a model for \( d + r_1 \cdot r_2 \). However, restricting \( < \) and \( \text{Suc} \) to \( U_1 \), we have a model \( M_1 = (U_1, \Psi', <, \text{Suc}) \) such that \( M_1 \models d + r_1 \): it is a model because it inherits the conditions from \( M \) being a model for \( d + r_1 \cdot r_2 \) and the observation that the constructions for \( r_1 \) and \( r_2 \) are disjoint. Similarly, we obtain a model \( M_2 \) such that \( M_2 \models d + r_2 \). From the induction hypothesis, we obtain \( w_1 \in \mathcal{L}(r_1) \) and \( w_2 \in \mathcal{L}(r_2) \) such that \( M_1 \models w_1 \) and \( M_2 \models w_2 \). Since we split \( M \) to form two smaller
models, i.e. $M = M_1 + M_2$, we conclude $M = M_1 + M_2 \models w_1 \cdot w_2$, where $w_1 \cdot w_2 \in \mathcal{L}(r_1) \cdot \mathcal{L}(r_2) = \mathcal{L}(r_1 \cdot r_2) = \mathcal{L}(r)$, as required.

7. If $r \equiv r_1^*$, then a similar argument to that for concatenation applies, except that we form many submodels of $M$, which will each correspond to models of $d + r_1$. From the construction for $d + r_1^*$, let $EC_s$ be the contour added to the end zone-identifiers of $d + r_1$, and let $EC_t$ be the contour added to the start zone-identifiers of $d + r_1$. If $M \models d + r_1^*$, then $M \models d'$, where $d'$ is the diagram obtained by removing the arrow $(EC_s, EC_t)$ from $d + r_1^*$. Let $\{u_1, \ldots, u_n\}$ be the subset of elements of $U$ which is the interpretation of $EC_s$, i.e. $\{u_1, \ldots, u_n\} = \Psi(EC_s)$. Let $U_1 = \{u : u \leq u_1\}$, and $U_i = \{u : u_{i-1} < u \leq u_i\}$ for $i = 2, \ldots, n$. Since $M \models d'$, we know that the successor of each $u_i$ for $i = 1, \ldots, n - 1$ must be in $\Psi(EC_t)$. Then, since $EC_t$ was added to the start zone identifiers of $r_1$, we have that $M_i = (U_i, \Psi, Suc, <)$ (with Suc and $<$ restricted in the appropriate way) will model $d + r_1$: it will inherit the satisfiability of the conditions of definition 6 from $M \models d + r_1^*$. From the induction hypothesis, this means each $M_i \models w_i$, where $w_i \in \mathcal{L}(r_1)$. Then, $M_1 + \ldots + M_n \models w_1, \ldots, w_n$. But, $w_1 \cdot \ldots \cdot w_n \in \mathcal{L}(r_1)^+ = \mathcal{L}(r_1^+)$, and so $\mathcal{L}(d + r_1^*) \subseteq \mathcal{L}(r_1^+)$, as required.

\[\square\]

Theorem. Let $d$ be a fully shaded Euler diagram with no missing zones, no spiders, and no arrows. For any $d$-regular expression $r$, any word $w$ in $\mathcal{L}(r)$, and any interpretation $M \models w$, we have $M \models d + r$. That is, $\mathcal{L}(r) \subseteq \mathcal{L}(d + r)$.

Proof. Let $r$ and $w \in \mathcal{L}(r)$ be given. Construct the diagram $d + r$. Let $M = (U, \Psi', <, Suc)$ be any model for $w$. We show that $M \models d + r$, by induction on the structure of $r$ and then by verifying the conditions from definition 6.

1. If $r \equiv \lambda$, then we have $U = \emptyset$. The diagram is simply a fully shaded Venn diagram, so has no missing zones, no spiders and no arrows. Thus, the model $M$ is trivially a model for the diagram $d + \lambda$.
2. If $r \equiv \emptyset$, then $d + r = \perp$ and there are no models for this diagram. Since no word in $\emptyset$ has a model, we are done.
3. If $r \equiv a$, for some $a \in \Sigma$, then we have $w = a$ (this is the only word in $\mathcal{L}(a)$). The only model for this word has a singleton universe, say $\{a\}$, and $<$ and $Suc$ are undefined. The diagram for this expression is fully shaded, with a single spider in the zone identifier $fc(a) \cup \{EC_1\}$. The successor condition is trivially satisfied: the diagram contains no arrows. Likewise, every other condition is straightforward to verify. Thus, $M \models d + a$.
4. If $r \equiv a^+$, for some $a \in \Sigma$, then we have $w = a \cdot a$ (the any word in $\mathcal{L}(a)$). The only model for this word has a singleton universe, say $\{a\}$, and $<$ and $Suc$ are undefined. The diagram for this expression is fully shaded, with a single spider in the zone identifier $fc(a) \cup \{EC_1\}$. The successor condition is trivially satisfied: the diagram contains no arrows. Likewise, every other condition is straightforward to verify. Thus, $M \models d + a$.

\[\square\]

45
(a) If $n = 1$, the situation is the same as in the previous case: $M$ is a model for the diagram if we interpret the spider as being in a contour without a successor arrow. We can then deduce the interpretation of the zone with a successor arrow is empty.

(b) If $n > 1$, we must verify the successor condition of definition 6. The universe $U$ has the same number of elements as there are instances of $a$ in $w$. Let $U = \{u_1, \ldots, u_n\}$. Let $\Psi(s) = \Psi'(z_1) = \{u_1\}$. Let $\Psi'(z_1) = \{u_n\}$, and $\Psi'(z_2) = \bigcup_{i<1<n} u_i$. The only arrow in $d+a$ is $(EC_2, EC_3)$, thus we need to check whether $Suc[\Psi'(EC_2)] = \Psi'(EC_3)$. Since $\Psi'(EC_2) = \{u_1, \ldots, u_{n-1}\}$, and $\Psi'(EC_3) = \{u_2, \ldots, u_n\}$, we have verified the condition.

5. If $r \equiv r_1 \cdot r_2$, then $w = w_1 \cdot w_2$ for some $w_1 \in \mathcal{L}(r_1)$ and $w_2 \in \mathcal{L}(r_2)$. Construct the diagram $d + (r_1 \cdot r_2)$. Again, we only need to check what happens at the boundaries of the construction for $r_1$ and $r_2$. Let $U_1$ be the universe for the model for the word $w_1$, and let $U_2$ be the universe for the model of the word $w_2$, where we restrict $< \text{ and } Suc$ appropriately. Call these models $M_1$ and $M_2$. Then, the induction hypothesis tells us $M_1 \models d + r_1$ and $M_2 \models d + r_2$.

We know the construction places a spider in the start zone-identifiers of $r_1$. Since there was a spider in the start zone-identifiers of $r_1$ before we combined it with $r_2$, this means the induction hypothesis tells us the spiders' location condition is satisfied for $d + r_1 \cdot r_2$. There is only one new arrow in the diagram. Owing to the shading created when adding an arrow (i.e. shading of zones which are both targets of arrows), we have that the successor condition is satisfied too.

6. If $r \equiv r_1[r_2]$, then $w \in \mathcal{L}(r_1)$ or $w \in \mathcal{L}(r_2)$. Consider the former: the latter is symmetrical. If $w \in \mathcal{L}(r_1)$, then by the induction hypothesis, $M \models d + r_1$. Then, $M \models d + (r_1[r_2])$: we simply give an empty interpretation for every contour constructed for $r_2$. Any arrows in the construction for $r_2$ will not violate the successor condition. Similarly, because the spider present in $d_1$ will still be present (it may just have more feet), the spiders' location condition is not violated. For, we know $\Psi(s) \subseteq \Psi'(\eta(s))$ for $d + r_1$, and $\eta(s)$ for $d + (r_1[r_2])$ contains $\eta(s)$ for $d + r_1$. Thus, the spiders' location condition is satisfied. Then, $M \models d + r$, as required.

7. If $r \equiv r_1^+$, then $w = w_1^+$, where $w_1 \in \mathcal{L}(r_1)$. Suppose $w_1$ has length $n$, and $w$ consists of $m$ copies of $w_1$. Thus, $U = \{u_1, \ldots, u_n, u_{n+1}, \ldots, u_{mn}\}$. Take the first $n$ elements of $U$; call this set $U_1$. Restricting $<$ and $Suc$ to $U_1$, creating the model $M_1$, we have $M_1 \models w_1$. By the induction hypothesis, $M \models d + r_1$. The source contour, $EC_1$, for the added arrow is added to the end zone identifiers of $d + r_1$, similarly the target contour, $EC_2$, is added to the start zone identifiers of $d + r_1$. Let the interpretation of the new contours be $\{u_n, u_{2n}, \ldots, u_{(m-1)n}\}$ and $\{u_{n+1}, u_{2n+1}, \ldots, u_{(m-1)n+1}\}$. Clearly, $Suc[\Psi'(EC_1)] = \Psi'(EC_2)$, and thus the successor condition is verified. The spiders' location condition is verified as well: there is only one spider in the diagram, and its location has not changed (as in the previous case). Then, we have $M \models d + r_1^+$, or $M \models d + r$, as required.
References


