A diagrammatic reasoning system for the description logic ALC

Peter W. Eklund  
*University of Wollongong*, peklund@uow.edu.au

Frithjof Dau  
dau@uow.edu.au

Follow this and additional works at: https://ro.uow.edu.au/infopapers

Part of the Physical Sciences and Mathematics Commons

Recommended Citation
Eklund, Peter W. and Dau, Frithjof: A diagrammatic reasoning system for the description logic ALC 2008, 539-573.  
https://ro.uow.edu.au/infopapers/1823

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au
Abstract
Diagrammatic reasoning is a tradition of visual logic that allows sentences that are equivalent to first
order logic to be written in a visual or structural form: usually for improved usability. A calculus for the
diagram can then be defined that allows well-formed formulas to be derived. This calculus is intended in
the analog of logical inference. Description logics (DLs) have become a popular knowledge
representation and processing language. DLs correspond to decidable fragments of first order logic; their
notation is in the style of symbolic, variable-free formulas. Moreover, DLs are equipped with table au
theorem provers that are proven to be sound and complete. Although DLs have roots in diagrammatic
languages (such as semantic networks), they are elaborated in a purely symbolic manner. This paper
discusses how DLs can be equivalently represented in terms of a diagrammatic reasoning system. First,
existing diagrammatic reasoning systems, namely spider- and constraint diagrams, as well as existential
and conceptual graphs, are investigated to determine if they are compatible with DLs. It turns out that
Peirce's existential graphs are better suited for this purpose than the alternatives we examine. The paper
then redevelops the DL ALC, which is the smallest propositional DL, by means of labeled trees, and
provides a diagrammatic representation for these trees in the style of Peircean graphs. We provide a
calculus based on C.S. Peirce's calculus for existential graphs and prove the soundness and
completeness of the calculus. The calculus acts on labeled trees, but can be best understood as a
diagrammatic calculus whose rules modify the Peircean-style representation of ALC.

Keywords
diagrammatic, reasoning, system, for, description, logic, ALC

Disciplines
Physical Sciences and Mathematics

Publication Details
Visual Languages and Computing, 19 (5), 539-573.

This journal article is available at Research Online: https://ro.uow.edu.au/infopapers/1823
A Diagrammatic Reasoning System for the Description Logic $\mathcal{ALC}$

Frithjof Dau, Peter Eklund

University of Wollongong, Wollongong, Australia

Abstract

Diagrammatic reasoning is a tradition of visual logic that allows sentences that are equivalent to first order logic to be written in a visual or structural form: usually for improved usability. A calculus for the diagram can then be defined that allows well-formed formulas to be derived. This calculus is intended is the analog of logical inference.

Description logics (DLs) have become a popular knowledge representation and processing language. DLs correspond to decidable fragments of first order logic; their notation is in the style of symbolic, variable-free formulas. Moreover, DLs are equipped with tableau theorem provers that are proven to be sound and complete.

Although DLs have roots in diagrammatic languages (such as semantic networks), they are elaborated in a purely symbolic manner. This paper discusses how DLs can be equivalently represented in terms of a diagrammatic reasoning system.

First, existing diagrammatic reasoning systems, namely spider- and constraint diagrams, as well as existential and conceptual graphs, are investigated to determine if they are compatible with DLs. It turns out that Peirce’s existential graphs are better suited for this purpose than the alternatives we examine.

The paper then redevelops the DL $\mathcal{ALC}$, which is the smallest propositional DL, by means of labeled trees, and provides a diagrammatic representation for these trees in the style of Peircean graphs. We provide a calculus based on C.S. Peirce’s calculus for existential graphs and prove the soundness and completeness of the calculus. The calculus acts on labeled trees, but can be best understood as a diagrammatic calculus whose rules modify the Peircean-style representation of $\mathcal{ALC}$.

Key words:
Description Logics, Spider Diagrams, Constraint Diagrams, Conceptual Graphs, Existential Graphs, Logic, Syntax, Semantics, Calculus

Preprint submitted to Elsevier 9 January 2008
1. Introduction

Description logics (DLs) are a common family of knowledge representation languages tailored to express knowledge about concepts and concept hierarchies. They include sound and complete decision procedures for reasoning about such knowledge. One of the main applications of DLs is their use as the basis for an ontology language, especially popular for the Semantic Web. In particular, the Ontology Web Language (OWL) — a W3C recommendation for the knowledge language of the Semantic Web — is based on a specific and expressive DL termed $SHOIN(D)$.

The basic building blocks of DLs are concepts (unary predicates), roles (binary relations) and sometimes individuals, which can be composed by language constructs such as intersection, union, value or number restrictions to build more complex well-formed formulas that themselves represent complex concepts and roles. For example, when MAN, FEMALE, MALE, RICH, HAPPY are predefined concepts and if hasChild is a predefined role, then

$$\text{MAN} \sqcap \exists \text{hasChild}.\text{FEMALE} \sqcap \exists \text{hasChild}.\text{MALE} \sqcap \forall \text{hasChild}.(\text{RICH} \sqcup \text{HAPPY})$$

(1)

describes the concepts of men who have both male and female children, and where all the children are rich or happy. Let us call a concept defined in this way as HAPPYMAN.

The formal notation of DLs has the flavour of a variable-free first order predicate logic (FOL). In fact, DLs correspond to (decidable) fragments of FOL, and like FOL, DLs have a well-defined, formal syntax, a semantics in the form of Tarski-style models, and a sound and complete calculi (based on Tableaux-algorithms). It is often emphasised that DLs offer, in contrast to other knowledge representation languages, sound, complete and (empirically) tractable reasoning services. A comprehensive overview on DLs is given in the Description Logic Handbook [4].

The notation of DLs is in the style of the usual linear and symbolic notations of FOL. The fact that the notation of DLs is variable-free makes them easier to comprehend than the common FOL formulas which include free variables. Nonetheless, for untrained users, the symbolic notation of DLs can be hard to learn and comprehend.

A main alternative to the symbolic notation is the development of a diagrammatic representation of DLs. It is well accepted that diagrams are in many cases easier to comprehend than symbolic notations (see for example [48,65,8,67]), and in particular it has been argued that they are useful for knowledge representation systems [35,46]. This has been acknowledged by the DL community and is a common view among the broader knowledge representation community [53]. In [52], the introduction to the Description Logic Handbook, Nardi and Brachman write that besides the possibility of “providing a syntax that resembles more closely natural language”, a “major alternative for increasing the usability of Description Logics as a modeling language” is to “implement interfaces where the user can specify the representation structures through graphical operations.”

---

1 In $SHOIN(D)$, $S$ stands for the basic DL $ALC$ (equivalent to the propositional modal logic extended with transitive roles), $H$ stands for role hierarchies, $O$ stands for nominals (classes whose extension is a single individual), $N$ stands for unqualified number restrictions and $D$ stands for datatypes)[42]

2 We use the term ‘symbolic’ according to C.S. Peirce’s classification of signs into ‘icons’, ‘indices’ and ‘symbols’. See Shin [67] for an introduction.
A first attempt at a diagrammatic representation for DL is can be found in [35], where Gaines elaborates a graph-based representation for the textual DL CLASSIC, part of the KL-One-framework. More recently, the focus has shifted from the development of proprietary diagrammatic representations to representations within the framework of UML (Unified Modeling Language). In 2003, the Object Management Group requested a metamodel for the purpose of defining ontologies. Following this proposal, [12] provides a UML-based, diagrammatic representation for the OWL DL. In these approaches, the focus is on a graphical representation of DL, however, as emphasized in many works on DL (see for example [4]), reasoning is seen as a distinguishing feature of DL and such reasoning is not supported diagrammatically by that treatment. Correspondences between graphical representation of the DL and the DL reasoning system are therefore important but remain largely unelaborated to date. Similar arguments hold for other popular diagrammatic languages like UML and ORM\(^3\) the difference being that, that unlike DLs, these diagrammatic modeling languages provide no extensional, mathematical semantics, nor any automated reasoning facilities.

On the other hand, there are some candidate diagrammatic reasoning systems that have the expressiveness of fragments of FOL or even full FOL. In this paper, we will evaluate two families of contemporary mathematical diagrammatic reasoning systems, which have the following two (historical) origins:

(i) The system of Euler Circles and Venn-Diagrams, the latter enriched by C.S. Peirce to Venn-Peirce-Diagrams (see [66,38]).
(ii) The system of Peirce’s Existential Graphs\(^4\).

The first system is the background to the contemporary development of spider- and constraint diagrams, the latter is background to a contemporary interpretation as conceptual graphs [70]. Why is it worth considering these diagrammatic reasoning systems as a starting point for a diagrammatic version of DL? Firstly, for all these systems mathematical elaborations in the general style of mathematical logic exists. These include:

- A well-defined syntax: usually, the syntax is defined at an abstract level (for example in terms of graph theory), such that the well-formed formulas have diagrammatic representations. Moreover, the syntax is – like the syntax for DL – variable-free.
- Extensional – Tarski-style semantics – and/or translations to formulas of FOL.
- Sound and complete calculi: usually the rules are defined on the abstract syntax but mostly they can be understood as manipulations of the represented diagrams.

Therefore, in contrast to data modeling languages like UML and ORM, we have a well-defined syntax and semantics for the candidate diagrammatic representations. Moreover, if we adopt one of these systems, we can adopt (partly or completely) the existing calculi as a diagrammatic reasoning service for DL. Finally, over and above the general arguments for diagrammatic systems, some of the systems investigated in this paper – constraint and spider diagrams and conceptual graphs – have had their diagrammatic benefits investigated in user-evaluations [83,64,31,30].

As a specific DL is the corresponding logic behind the OWL, the results of this paper benefit not only DL, but the Semantic Web more generally. Developing a Semantic Web language as a mathematical diagrammatic reasoning system has already been carried

---

\(^3\) [http://www.orm.net/index.html]

\(^4\) [http://www.existentialgraphs.com/] or (a different site) [http://www.existential-graphs.net/]
out for a much simpler language: namely RDF. For RDF, mathematical elaborations based on graph theory, including a Tarski-style semantics and a sound and complete calculus based on “projections” (see [5,6]) or via diagrammatic rules (see [24]) have been elaborated.

In the next three Sections, 2, 3 and 4, the basic notions of description logics, spider- and constraint diagrams, and existential- and conceptual graphs, are introduced. In the subsequent Sections 5 and 6, we investigate whether spider- and constraint diagrams and existential- and conceptual graphs respectively are suited as a starting point for developing DL as a diagrammatic reasoning system. It will turn out that there are prospects to use spider- and constraint diagrams for this purpose. However, conceptual graphs and, as we will show, existential graphs are a much better match. In the three Sections that follow, $\mathcal{ALC}$ is developed as a diagrammatic reasoning system based on Peirce’s existential graphs. In Section 7, a formalization of $\mathcal{ALC}$ by means of labeled trees is given. Section 8 provides the Peirce style representation of such trees. In Section 9, a calculus for $\mathcal{ALC}$-trees, based on Peirce’s calculus for existential graphs, is given, and soundness and completeness proven. Finally, in Section 10, further research is discussed. As we use several abbreviations in this paper, these are presented in Section 11.

2. Introduction into Description Logics

The vocabulary of any DL consists of CONCEPTS, which denote sets of individuals, and ROLES, which denote binary relationships between individuals. The starting points are ATOMIC CONCEPTS AND ROLES – CONCEPT AND ROLE NAMES – from which more complex concepts and roles are built with constructs such as intersection, union, value or number restrictions, and so on. Moreover, we usually consider concept vocabularies where we have a universal concept $\top$. The tables of Figures 1 and 2 provide an overview of the most important entities and constructors. In both tables, the first column provides the common name for the entity or constructor, in the second column the syntax is presented.

In DLs concept descriptions gain meaning when they are interpreted in a model. An INTERPRETATION is a pair $(\Delta^I, I)$, consisting of an nonempty DOMAIN of the interpretation $\Delta$ and INTERPRETATION FUNCTION $I$ which assigns to every atomic concept $A$ a set $A^I \subseteq \Delta^I$ and to every atomic role $R$ a binary relation $R^I \subseteq \Delta^I \times \Delta^I$. For each atomic role, let $I(R^-) = \{(b, a) \mid (a, b) \in I(R)\}$, we require $I(\top) = \Delta^I$. The interpretation function is then extended to arbitrary roles and concept descriptions, as depicted in the third column of the tables of Figures 1 and 2. In the third column of Figure 2 we use the following notation which will be used in the next section as well: if $U$ is a set, $S \subseteq U \times U$ is a relation, and if $A \subseteq U$ and $a \in U$, then let $A.R$ denotes the set $\{u \in U \mid \exists a \in A.a.Ru\}$ (the image of $A$ under $R$). The set $\{a\}.R$ is abbreviated $a.R$. The sets $R.A$ and $R.a$ are defined analogously.

Depending on the choice of constructors we get different DLs with different levels of expressiveness. The last column of the table in Figure 2 provides the common letters which are used to abbreviate the constructors. In this paper, we focus on the description logic $\mathcal{ALC}$. Literally, this logic has conjunction, negation and value restriction as constructors, but as $C_1 \sqcup C_2$ is equivalent to $\neg(\neg C_1 \cap \neg C_2)$, and as $\exists R.C$ can be replaced by $\neg\forall R.\neg C$, it is also assumed that $\mathcal{ALC}$ contains disjunction and existential quantification, i.e. $\mathcal{ALC}$
is the smallest propositionally closed description logic.

<table>
<thead>
<tr>
<th>Construct Name</th>
<th>Syntax</th>
<th>Semantics</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic concept A</td>
<td>$A$</td>
<td>$A^I \subseteq \Delta^I$</td>
<td>$AL$</td>
</tr>
<tr>
<td>universal concept $\top$</td>
<td>$\Delta^I$</td>
<td>$\Delta^I$</td>
<td></td>
</tr>
<tr>
<td>atomic role R</td>
<td>$R^I \subseteq \Delta^I \times \Delta^I$</td>
<td>$\mathcal{R}$</td>
<td></td>
</tr>
<tr>
<td>individuals a</td>
<td>$a^I \in \Delta^I$</td>
<td>$a^I \in \Delta^I$</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 1. Entities in Description Logics

### Class Constructors

<table>
<thead>
<tr>
<th>Construct Name</th>
<th>Syntax</th>
<th>Semantics</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>conjunction $C_1 \cap C_2$</td>
<td>$C^I \cap D^I$</td>
<td>$C_{\cap}^I$</td>
<td>$\cap$</td>
</tr>
<tr>
<td>disjunction $C_1 \cup C_2$</td>
<td>$C^I \cup D^I$</td>
<td>$C_{\cup}^I$</td>
<td>$\cup$</td>
</tr>
<tr>
<td>negation $\neg C_1$</td>
<td>$\Delta^I - C^I$</td>
<td>$C_{\neg}^I$</td>
<td></td>
</tr>
<tr>
<td>value restriction $\forall R.C$</td>
<td>${x \in \Delta^I \mid \forall y \in xR^I : y \in C^I}$</td>
<td>$\mathcal{A}$</td>
<td></td>
</tr>
<tr>
<td>exists restriction $\exists R.C$</td>
<td>${x \in \Delta^I \mid \exists y \in xR^I : y \in C^I}$</td>
<td>$\mathcal{E}$</td>
<td></td>
</tr>
<tr>
<td>nominals ${o_1, \ldots, o_n}$</td>
<td>${o_1^I, \ldots, o_n^I}$</td>
<td>$\mathcal{O}$</td>
<td></td>
</tr>
<tr>
<td>unqualified number restriction $\geq nR$</td>
<td>${x \in \Delta^I \mid</td>
<td>xR^I</td>
<td>\geq n}$</td>
</tr>
<tr>
<td>unqualified number restriction $\leq nR$</td>
<td>${x \in \Delta^I \mid</td>
<td>xR^I</td>
<td>\leq n}$</td>
</tr>
<tr>
<td>qualified number restriction $\geq nR.C$</td>
<td>${x \in \Delta^I \mid</td>
<td>xR^I \cap C^I</td>
<td>\geq n}$</td>
</tr>
<tr>
<td>qualified number restriction $\leq nR.C$</td>
<td>${x \in \Delta^I \mid</td>
<td>xR^I \cap C^I</td>
<td>\leq n}$</td>
</tr>
</tbody>
</table>

### Role Constructors

<table>
<thead>
<tr>
<th>Construct Name</th>
<th>Syntax</th>
<th>Semantics</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>inverse role $R^-$</td>
<td>${(y, x) \mid (x, y) \in R^I}$</td>
<td>$\mathcal{I}$</td>
<td></td>
</tr>
<tr>
<td>role intersection $R_1 \cap R_2$</td>
<td>$R_1^I \cap R_2^I$</td>
<td>$\mathcal{R}$</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 2. Constructors in Description Logics

3. Spider and Constraint Diagrams

Spider diagrams (SDs) and constraint diagrams (CDs) are a diagrammatic reasoning system based on Euler circles and Venn-Peirce diagrams. They are inspired by a fragment of UML which is used for software specifications as an alternative to UML’s textual Object Constraint Language (OCL). They are elaborated as abstract mathematical structures, including an extensional semantics and inference rules. In this section, SDs and CDs are briefly and informally introduced. Particularly, only the diagrams of SDs and CDs are provided, not the underlying mathematical structures. For the formal definitions we refer the reader to [77,79].
3.1. Spider Diagrams

Since the first paper on SDs [37], several elaborations of the ideas have been published [84]. Most variations of SDs are based on Venn diagrams as a foundation, which can be propositionally combined in conjunctive normal form. It turns out however that SDs foundation in Venn-diagrams renders many of them difficult to read: the restriction of combining these diagrams in conjunctive normal form yields a number of technical problems that result from the rendering of four or more sets. The final version of SDs has been published in the PhD thesis of Stapleton [77] – a useful 20 page overview can be found at [84]. Another source is [79], which provides the definitive SDs paper. In that treatment SDs are a combination of Venn diagrams and the more user friendly Euler diagrams and the restriction to conjunctive normal form is removed. Below, two examples of so-called unary SDs are depicted.

![Fig. 3. Two Spider Diagrams](image)

As in Venn diagrams the bounding rectangles depict the universe (of discourse), i.e., the ground set of the respective models in which the diagrams are evaluated. We use the letter \( U \) to denote the universe of discourse. The first diagram to the left has two contours: these are the circles labeled \( A \) and \( B \), representing the sets \( A \) and \( B \). The diagram has four zones which represent all combinations of \( A \) and \( B \), i.e. \( A \cap B \), \( A - B \), \( B - A \), and \( U - (A \cup B) \). As all possible combinations of \( A \) and \( B \) are represented, this SD is a Venn-Peirce diagram as well. The diagram contains two spiders: The shaded zone representing \( A - B \) is the habitat of the first spider, the region composed of the zones \( A - B \) and \( B - A \), i.e. the region which represents \( B \), is the habitat of the second spider. Each spider denotes a uniquely given object (i.e., different spiders necessarily denote different objects) which is a member of its habitat. In Venn-Peirce diagrams, shading a region means that the corresponding set is empty. Thus, when read as a Venn-Peirce diagram, the diagram left contains the information that – due to the shading – \( A - B \) is empty, and – due to the spider – there is an element in \( A - B \), so the diagram is contradictory. In SDs, the semantics for shading is slightly different from that of Venn-Peirce diagrams: a region does not contain more elements than the elements represented by some spiders. So the SDs reads as follows: there are two sets \( A \) and \( B \), the set \( A - B \) contains exactly one element, and the set \( B \) contains at least one element.

In the second diagram to the right, a third contour representing a set \( C \) is involved. This diagram uses the notions of Euler circles: as the contour labeled \( C \) does not overlap with the \( A \) and \( B \)-contours, \( C \) and \( A \cup B \) are disjoint. Moreover, there are three elements \( u \),

---

5 A survey of Euler diagram-based reasoning systems and notations is given in [78], or in the introduction of [77].

6 The usage of disjoint features in Euler circles has a drawback: not all abstract SDs are drawable (see [79]).
v and w (represented by the three spiders) such that \( u, v \in A \) and \( w \in U - A \). Further, due to its shading, the set \( A - B \) must not contain any other elements than \( u \) and \( v \), i.e. it contains exactly two elements, and \( A \cap B \) must not contain any other elements than \( w \), i.e., it contains no element or at most one element.

The above diagrams are **UNARY** SDs. They can be propositionally combined with the logical operators \( \cap \) (‘and’) and \( \cup \) (‘or’). Below is an example of an SD which uses these conjunctors is shown.

In SDs, the spiders can be understood to correspond to existentially quantified objects. Later in our presentation, a different kind of spider, which corresponds to universally quantified objects in constraint diagrams, is introduced; this then allows one to speak about **existential** and **universal** spiders. Nonetheless, since zones can be shaded, we can express universal statements as well. In fact, Stapleton [77] has shown that SDs are equivalent in expressive power to monadic FOL, *with equality*, MFOLe\(^7\). Therefore the system of SDs is semantically an extension of Shin’s Venn-II-system [66] which is equivalent in expressive power to monadic FOL *without equality*. This feature is obtained by the change of the semantics that includes the use of shading.

The Stapleton system of SDs [77] is equipped with a sound and complete calculus which is a combination of five diagrammatic rules and of eight ‘propositional logic style’-rules that encompass the fact that unary diagrams can be assembled with the propositional conjunctors \( \cap \) and \( \cup \). Furthermore, Stapleton proved that the system is decidable\(^8\), which makes SDs attractive for application to DLs.

Finally, there have been attempts to develop automated theorem provers for SDs. Flower in [32,33] investigates a heuristic approach to generating proofs in the SD system and further in [34] presents a theorem proving tool: available from [84]. Moreover, a tableaux system for SDs has also been implemented by Patrascoiu [56].

### 3.2. Constraint Diagrams

SDs only allow reasoning about sets, namely unary predicates, and provide no possibility to represent or reason about any sort of relations. Moreover, as already mentioned, the

---

\(^7\) The proof is sketched as follows: in one direction, it is easy to assign to each diagram a corresponding MFOLe-formula. The other direction is more challenging: first, to each MFOLe-formula, a finite set of finite models which are prototypic for all models of the formula are assigned, and to each of these models a unary SD is assigned.

\(^8\) It has to be emphasised that in the semantics for SDs (and CDs), in contrast to FOL, allows *empty* models. For this reason, one cannot immediately transfer decidability results for fragments of FOL – like the well-known Bernays-Schönfinkel-fragment of FOL – to SDs.
spiders in SDs are ‘existential spiders’ as they can be read as existentially quantified objects. On the other hand, constraint diagrams are an extension of SDs with **universal spiders** (quantifiers) and **arrows** which represent binary relations (to be precise: the arrow labels stand for binary relations, thus an arrow represents a property of the relation denoted by its label). A full constraint notation was introduced by Kent [45] in an informal manner. Since then, several papers attempt to elaborate a full mathematical treatment of Kent’s vision, including syntax, semantics, and a sound and complete calculus for constraint diagrams. Let us first depict a constraint diagram (taken from [77]):

![Constraint Diagram](image)

**Fig. 4. A Constraint Diagram**

The core information expressed in this diagram is that “every person can only borrow books that are in the collections of libraries they have joined”.

In Fish et al [29], the syntax and semantics of full constraint diagrams is developed but a sound and complete calculus is elusive. The first ever constraint reasoning system (i.e., including a sound and complete calculus) was developed by Stapleton [77] but compared to Kent’s approach it has several limitations. For example, universal spiders inhabit single zones only, with at most one spider in any given zone (in Kent’s full system, universal spiders are allowed to inhabit more than one zone, and a zone could contain more than one universal spider), and each universal spider in a unitary diagram must be the source of at least one arrow. In Kent’s full constraint diagram systems there are unlabeled contours, termed derived contours. For example, the targets of the arrows labeled with ‘canBorrow’ and ‘collection’ in Figure 4 are derived contours. In Stapleton’s system, there is at most one derived contour, which must be the target of at least one arrow. Moreover, the inside of a derived contour is always shaded, and must therefore necessarily represent the empty set.

In Stapleton’s constraint diagrams, the source of an arrow is an (existential or universal) spider, and its target can either be an existential spider or a contour. Depending on the source, we distinguish existential and universal arrows. As each universal spider is the source of at least one arrow, in order to clarify the semantics of Stapleton’s constraint diagrams, we have to make clear how arrows are read. As already stated however, the label of an arrow denotes a binary relation on the universe $U$.

Let $l$ be an existential arrow with source $s$ (an existential spider) and target $t$ (an existential spider or a contour). Let $R$ be the relation represented by the label of $l$. Let $S$ be the object denoted by $s$ and let $T$ be the set denoted by $t$ (for simplicity reasons we agree that an existential spider denotes a singleton set). Then $l$ stands for the condition $S.R = T$. If $l$ is an universal spider, the source now stands for the set denoted by the contour of the spider’s habitat, i.e., if the habitat of $l$ is the contour $c$ which represents the set $C$, then each element $x \in C$ has to satisfy the condition $x.R = T$. In the semantics...
existential quantifiers take precedence over universal quantifiers (i.e., existential spiders are read first). For example, we consider the following constraint diagram:

The reading of the diagram is as follows:

\[
C \cap (A \cup B) = \emptyset \wedge \exists u \in U - (A \cup B \cup C). \exists b \in B. \exists c_1, c_2 \in C : (c_1 \neq c_2 \wedge uR = A \wedge c_1S = b \wedge \forall x \in A - B : xT = \{c_2\})
\]

Similar to SDs, Stapleton [77] developed a sound and complete calculus for constraint diagrams. It contains 25 rules, 22 are needed in the completeness proof (the other three rules are useful shortcuts). In contrast however to SDs, there are unary constraint diagrams which are not satisfiable. To compensate, Stapleton provided a compatibility relation on the arrows that can be used to classify satisfiable constraint diagrams.

Finally, after discussing the syntax and semantics of CDs, we need to mention that Fish et al [31,30] provide empirical studies on how human users read CDs.

After this brief introduction into SDs and CDs, we introduce in the next section the system of existential and conceptual graphs.

4. Existential and Conceptual Graphs

In this section, we introduce Peirce’s existential graphs (EGs), Sowa’s conceptual graphs (CGs), particularly its most common fragment of simple conceptual graphs (SCGs) and the system of concept graphs with cuts (CGwCs) introduced by one of the authors.

Similar to spider and constraint diagrams, there exist mathematically precise elaborations for each of these systems. Again, like spider and constraint diagrams, the diagrammatic benefits of EGs or CGs (used in teaching) has been investigated by Schäfe [64] and also by Pollant [83]. Furthermore, a theorem prover for existential graphs by developed by Stewart [80].

4.1. Existential Graphs

Existential graphs [39] are a diagrammatic logic invented by C.S. Peirce (1839-1914) in the last two decades of his life. Existential graphs are divided into three parts called Alpha, Beta and Gamma. The three parts build on one another, Beta builds upon Alpha, and Gamma builds on both Alpha and Beta. Alpha corresponds to propositional logic, Beta corresponds to FOL (to be precise: first order logic with predicates and equality, but without functions or constants). Gamma encompasses features of higher order logic,
including modal logic, self-reference and more. In contrast to Alpha and Beta, Gamma was never finished by Peirce, and even now, only fragments of Gamma (mainly the modal logic part) are elaborated to contemporary mathematical standards. In this section, only Alpha and Beta are introduced. For further readings we recommend Roberts [62], which is the standard book on EGs and which describes in a more informal manner the complete system; (see also Roberts [63]); Zeman’s formal elaboration in [89] (including a fragment of Gamma); [67], a book where Shin provides her interpretation of Alpha and Beta which focuses on its iconic features; [58], which is a tutorial of Peirce himself with commentary by Sowa, or a short introduction by Sowa in [73]. A mathematical elaboration of Alpha and Beta has been produced but is currently unpublished.

We start with the description of Alpha. The EGs of Alpha consist only of predicate names of arity zero, which are called medads, and of closed, doublepoint-free curves which are called cuts (or sometimes seps) and used to negate the enclosed subgraph. Medads can be considered as propositions. Propositions can be written down on an area (the term Peirce uses instead of ‘writing’ is ‘scribing’), and writing down a proposition is to assert it. The area where the proposition is written or drawn (scribed) is what Peirce called the sheet of assertion. It may be a sheet of paper, a blackboard or any other surface. Scribing several propositions next to each other (this operation is called a juxtaposition) asserts the truth of each proposition, i.e. the juxtaposition corresponds to the conjunction of the juxtaposed propositions. For example, scribing the propositions ‘it rains’, ‘it is cold’ and ‘it is stormy’ next to each other yields the graph

\[
\text{it rains} \quad \text{it is cold} \quad \text{it is stormy}
\]

which means ‘it rains, it is cold, and it is stormy’.

The location of the proposition on the sheet of assertion has no significance. Below, another diagram with the same meaning is given.

\[
\text{it rains} \\
\text{it is stormy} \\
\text{it is cold}
\]

The possibility that an existential graph can have different representations is addressed by Peirce at the beginning of [57], where he writes:

Convention No. Zero. Any feature of these diagrams that is not expressly or by previous conventions of languages required by the conventions to have a given character may be varied at will. This “convention” is numbered zero, because it is understood in all agreements.

To explain, the two diagrams above are different representations of the same graph. In order to distinguish graphs from their diagrams, Peirce coined the term graph and graph replica, i.e., the above diagrams are different graph replicas of the same graph. Similar

---

9 The reader is encourage to refer and correspond to Dau[23] for his forthcoming habilitation thesis on this subject.

10 An asserted proposition is called a judgement.
distinctions are made between types and tokens, known from philosophy, or abstract and concrete syntax, used widely in Computer Science. As discussed in [43,21], for a formally precise elaboration of any logic by means of diagrams, this distinction is vital. This approach is adopted in this paper as well. After evaluating different diagrammatic reasoning systems for using them for DLs, in Section 7, a fragment of Peirce’s graphs is used as a diagrammatic system for the DL $\mathcal{ALC}$. In this section, the syntax of this fragment is defined on an abstract level which prescinds from the topological properties of the diagrammatic representations.

So far, we have only seen how conjunction is expressed in the system of Peirce’s graphs. Next, negating a graph is depicted by encirling it. The space within a cut is called its close or area. For example, the graph

![it rains](image)

has the meaning ‘it is not true that it rains’, i.e. ‘it does not rain’. The graph

![it rains it is cold](image)

has the meaning ‘it is not true that it rains and that it is cold’, i.e. ‘it does not rain or it is not cold’. Cuts must not overlap, but they may be nested. The next graph has two nested cuts.

![it rains it is cold](image)

This graph has the meaning ‘it is not true that it rains and that it is not cold’, i.e. ‘if it rains, then it is cold’. The device of two nested cuts is called a scroll. From the last example we learn that a scroll can be read as an implication. A scroll with nothing on its first area is called double cut (it corresponds to a double negation). As mentioned before, the space within a cut is called the area of the cut. In the example above, we therefore have three distinct areas: all the space outside the outer cut, i.e. the sheet of assertion, the space between the outer and the inner cut, which is the area of the outer cut, and the space inside the inner cut, which is the area of the inner cut. An area is oddly enclosed if it is enclosed by an odd number of cuts, and it is evenly enclosed if it is enclosed by an even number of cuts. As we have the possibility to express conjunction and negation of propositions, we see that Alpha has the expressiveness of propositional logic.

If we go from the Alpha part of EGs to the Beta part, predicate names of arbitrary arity may be used, and a new symbol, the line of identity, is introduced. Lines of identity are used to denote both the existence of objects and the identity between objects. Lines of identity are attached to predicate names. Peirce drew them in bold. Consider the following graph:

![cat on mat](image)

This contains two lines of identity, hence it denotes two (not necessarily different) objects. The first line of identity is attached to the unary predicate ‘cat’, hence the first object denotes a cat. Analogously, the second line of identity denotes a mat. Both lines are attached to the dyadic predicate ‘on’, i.e. the first object (the cat) stands in the relation
‘on’ to the second object (the mat). The meaning of the graph is therefore ‘there is a cat and a mat such that the cat is on the mat’, or in short: A cat is on a mat.

Consider the following graphs, where cuts are involved.

\[
\begin{align*}
& \text{man} \\
& \text{man} \\
& \text{man}
\end{align*}
\]

The meaning of the first graph is clear: it is ‘there is a man’. The second graph is built from the first graph by drawing a cut around it, i.e. the first graph is denied. Hence the meaning of the second graph is ‘it is not true that there is a man’, i.e. ‘there is no man’. In the third graph, the heavily drawn line (it is not a line of identity, which will be discussed shortly) begins on the sheet of assertion. Hence, the existence of the object is asserted, not denied. For this reason the meaning of the third graph is ‘there is something which is not a man’.

Peirce writes in 4.116 (we adopt the usual convention to refer to his collected papers [39]), a “line of identity is [. . .] a heavy line with two ends and without other topical singularity (such as a point of branching or a node), not in contact with any other sign except at its extremities.” So lines of identity do not have any branching points, nor are they allowed to cross cuts. However, by connecting them at their endpoints, we can obtain networks of lines of identity, which are termed ligatures. Peirce allows only two or three lines of identity to be connected. If three lines of identity are connected, the point where they meet is called a branching point. Moreover, lines of identity are allowed where they connect directly on a cut. Due to this possibility, ligatures are permitted which cross a cut.

Let us first consider the three graphs of Fig. 5.

\[
\begin{align*}
& \text{male} \\
& \text{human} \\
& \text{African} \\
& \text{man} \quad \text{will die}
\end{align*}
\]

Fig. 5. Four Peirce graphs with so-called single-object-ligatures

In the first graph, the ligature consists of three lines of identity, which meet in a branching point, in the second graph, the ligature consists of two lines of identity meeting on the cut, and the ligature in the third graph is composed of seven lines of identity. Nonetheless, in all these graphs, a ligature can, similar to a line of identity, be understood to denote a single object. The meaning of the graphs of Figure 5 ‘there exists a male, human african’, ‘there exists a man who will not die’, and ‘it is not true that there is a pet cat such that it is not true that it is not lonely and owned by somebody’, i.e., ‘every pet cat is owned by someone and is not lonely’.

Nonetheless, other examples show that this interpretation of ligatures is not so simple in every case: namely a ligature may stand for more than one object. Let us consider the three graphs of Figure 6. These graphs have the meanings ‘there are at least two suns’, ‘there are (not necessarily distinct) objects which are blue, red, large and small, respectively’, and ‘the blue and large or the red and small object are distinct’, and ‘there are objects \( o_1, o_2, o_3 \) with the properties \( S, P, \) and \( T \) resp, and these objects are not all identical’ (i.e., \( o_1 = o_2 = o_3 \) does not hold). In every graphs, there is not a single ligature that can be understood to denote a single object.
In every graph in Figure 6 a part of a ligature traverses a cut (i.e., there is a cut \( c \) and a heavily drawn line \( l \) which is part of the ligature such that both endpoints of \( l \) are placed on \( c \) and the remainder of \( l \) is enclosed by \( c \)). Such a device denotes non-identity of the endpoints of \( l \) (for example, Peirce writes (in 4.459) that “a sep [sep is another word Peirce used for cut] which is vacant, except for a line of identity traversing it, expresses with its contents the non-identity of the extremities of that line.”), thus a ligature containing such an element \( l \) usually denotes different objects. But if such an element does not occur, it can be shown that the ligature denotes a single object. For this reason, a ligature \( L \), such that no part of \( L \) traverses any cut, will be called single-object-ligature (SO-ligature).

A complete discussion existential graphs with non-single-object-ligatures goes beyond the scope of this paper, see [22] for a more detailed discussion. It will turn out that the ligatures we have to deal with in this paper are all single-object-ligatures, so we will not run into problems caused by non-single-object-ligatures.

We now have all the necessary elements to express existential quantification, predicates of arbitrary arities, conjunction and negation. As such we see that the Beta part of existential graphs corresponds to FOL (without object names and without function names). Moreover, Peirce equipped EGs with a set of five sound and complete inference rules.

4.2. Conceptual Graphs

Sowa [70] developed CGs in the 1970s on the back of EGs (Sowa [71] provides an abbreviated overview). “Conceptual graphs are an extension of existential graphs with features adopted from linguistics and AI” [73], and the purpose of the system is to express meaning in a form that is “logically precise, humanly readable, and computationally tractable.” The term ‘extension’ in the above is to be understood semantically, not syntactically: Sowa adopted the ideas of EGs, but CGs have a different and richer syntax, “Besides Peirce’s primitives, CGs provide a means of representing case relations, generalized quantifiers, indexicals and other aspects of natural languages.” [73]. Another fundamental aspect of CGs is that different kinds of referents exist, in particular names for objects. Moreover, conceptual graphs may serve as referents themselves (which is similar to a feature of Peirce’s Gamma EGs). This is called nesting in CGs. Both aspects will be explained in the rest of this section.

In order to illustrate the broad range of CGs, we will provide several examples. The first two follow:

\[
\text{CAT: } * \quad \text{on} \quad \text{MAT: } * \\
\text{CAT: Yoyo} \quad \text{on} \quad \text{MAT: } *
\]

The meanings of these graphs are ‘a cat is on a mat’ and ‘the cat Yoyo is on a mat’ respectively. In each concept box, we have a type label \( t \) and a referent \( r \). Sowa
calls the boxes CONCEPTS, but in this paper the term CONCEPT BOX will be used to avoid confusion with the use of the word concept in DLs. In the CG above, we have the types CAT and MAT. The referents are ‘Yoyo’ and the GENERIC MARKER ‘∗’. Type labels are ordered in a sub-type-relation, according to their level of generality. For example, CAT is a subtype of ANIMAL. An ordered set of types is often called SUPPORT, TYPE HIERARCHY, TAXONOMY or even ONTOLOGY. Type hierarchies usually contain a greatest type ⊤, the UNIVERSAL TYPE, containing every object (of the respective universe of discourse) in its extension. In the graph above, the two concept boxes contain two different kinds of referents. The referent ‘Yoyo’ of the first box is a name for an object. The referent ‘∗’ of the second box is a fixed symbol which does not denote a particular individual, but it denotes an individual which is not further specified. So the star ‘∗’ can be read as an existential quantifier. It is called GENERIC MARKER, hence the second box is called GENERIC CONCEPT BOX. The ovals are (CONCEPTUAL) RELATIONS between the referents of the concept boxes which are linked to the oval.

Sowa uses various kinds of referents. For example, the following graphs have the meanings ‘all men married a certain woman’ and ‘she is eating four bones’.

\[
\text{MAN: } \forall \text{ marry } \text{WOMAN: } @\text{certain} \text{ marry } \text{FEMALE: } \# \text{ eats } \text{BONE: } \{∗\}4
\]

It is clear that these various kinds of referents go beyond the expressiveness of first order logic. In particular they are not adapted from EGs because they use universal quantifiers and numbered restriction. Moreover, even from the first example of this section (i.e., the graph with the meaning ‘a cat is on a mat’) a crucial difference between existential graphs and conceptual graphs is apparent. The lines of identity in EGs serve different purposes: first, they are used for existential quantification, second, they are used to connect arguments to relations, and third they are used to show identity between arguments. On the other hand in CGs these functions are separated. Existential quantification is expressed by generic concept boxes. The referents of concept boxes serve as arguments for relations, and the connection between the arguments and the relations is drawn by relation ovals.

We need moreover to clarify how identity is expressed in CGs. For this, a new syntactical element is used. In [71], Sowa says “Two concepts that refer to the same individual are coreferent. […] are used to show that they are coreferent, i.e. concepts are connected with a dotted line, called a coreference link.” This can be illustrated using the graph of Fig. 7. This says that ‘Mary is a person and there is a teacher who is the same as Mary’, or ‘Mary is a teacher’.

\[
\text{PERSON: Mary } \text{TEACHER: } ∗
\]

Fig. 7. conceptual graph for ‘Mary is a teacher’

The second fundamental difference between EGs and CGs is the nesting of CGs. When graphs are nested, some CGs serve as referents in concept boxes of other CGs themselves. In this manner, some CGs make statements and assertions about other CGs, i.e., the system of CGs therefore offers the possibility of meta-level statements. Concept boxes whose referents are CGs are called CONTEXTS. Peirce’s sheet of assertion or the surface
where a CG is scribed can equally be understood as a context, the outermost context. In Figure 8 we present a well-known example of a nested CG: It contains two contexts, namely the concept boxes of type PROPOSITION and SITUATION (which are common types of contexts). The graph can be read as follows: ‘The person Tom believes a proposition, which is described by a graph itself’. The proposition says that ‘the person Mary wants a situation, which again is described by a graph’. In this situation we have a concept box PERSON: Mary that is connected with a coreference link to the concept box PERSON: Mary in the context above. So the situation is that ‘Mary marries a sailor’. The formal understanding of the whole graph is now: ‘The person Tom believes the proposition that the person Mary wants the situation that Mary marries a sailor’. In short: ‘Tom believes that Mary wants the situation in which she marries a sailor’, even more succinctly: ‘Tom believes that Mary wants to marry a sailor’.

![Fig. 8. A nested conceptual graph](image)

Even negation is modeled with a specific context. As Sowa writes in [73]: “The EG negative contexts are a special case of the CG contexts. They are represented by a context of type Negation whose referent field contains a conceptual graph that states the proposition which is negated.” Concept boxes of type NEGATION are introduced by Sowa as abbreviations for contexts of type proposition with an unary relation ‘NEG’ attached (see Sowa [71], “Negation (NEG) is one of the most common relations attached to contexts”), and Sowa often abbreviates these contexts by drawing a simple rectangle with the mathematical negation symbol ¬ (see Sowa [75]).

Two examples for CGs with negations are presented in Figure 9. The left graph states ‘there are two suns, but it is not true that there is a thing which is identical to the two suns’, i.e. ‘there are at least two suns’ (compare this to Figure 6). In the right graph, the device of two nested negation contexts corresponds to scrolls in EGs and can be understood as an implication. Hence, the meaning of the graph is ‘if a farmer owns a donkey, then he beats it’.

![Fig. 9. Conceptual graphs with negation](image)

introduction and discussion of contexts in CGs. Also recommended are the works of McCarthy [51], and Barwise and Perry [7]).
There is presently no mathematical elaboration for the complete system of CGs [85,19]. Particularly troublesome is the handling of negation as specific contexts lead to several problems [19]. In the following, we will first discuss the most prominent fragment of CGs, namely simple conceptual graphs (SCGs). Then the system of conceptual graphs with cuts (CGwCs) is described. Both systems are elaborated in a formally precise manner.

4.2.1. Simple Conceptual Graphs

The most prominent and best investigated fragment of CGs is the system of simple conceptual graphs, SCGs. In this system, no contexts are allowed, so SCGs correspond to the conjunctive, positive and existential fragment of FOL. This fragment is decidable, nonetheless, a decision procedure to determine whether one SCG implies another is NP-complete [14].

There are two different approaches to a mathematical elaboration of SCGs. In both approaches, mathematical graphs are used to formalize SCGs. In the first, which follows the work of Chein and Mugnier [14,15], bipartite labeled graphs are used: both concept boxes and relation ovals are vertices in the graphs, and edges correspond to the lines connecting concept boxes and relation nodes. The other approach follows Wille [86] where concept boxes are modeled as vertices of a graph, whereas a relation oval, including all lines from it to concept boxes, is modeled as a directed hyper-edge. In this case, labeled directed multi-hypergraphs form the mathematical basis of SCGs. Nonetheless, both formalizations can easily be translated to the other [44].

Chein and Mugnier provide a translation of SCGs to FOL as semantics for SCGs [14]. Moreover, the authors provide a sound and complete syntactical entailment relation between SCGs based on graph homomorphisms, called projections. Briefly put, a SCG \( G \) implies a SCG \( H \) if there is a graph homomorphism \( f : H \rightarrow G \) which respects the underlying type hierarchy. In their first elaborations of projections [14], a further restriction on \( G \) was considered: namely it had to be in normal form, i.e., no referent was allowed to occur in distinct concept boxes. In Chein and Mugnier [16], this restriction is dismissed by extended projection to so-called coref-projections which are graph-homomorphisms that do not map vertices to vertices but rather maps sets of coreferent vertices to sets of coreferent vertices.

In Dau [18], one of the authors provides an extensional semantics for SCGs based on Wille’s formal concept analysis ([36]). Moreover, a sound and complete calculus, consisting of fairly simple manipulations of the diagrams, is also provided. Thus, in both of the popular mathematical elaborations of SCGs, a calculus is defined that is equivalent to reasoning in a restricted form of FOL corresponding to the conjunctive, positive and existential fragment of FOL.

4.2.2. Conceptual Graphs with Negation

As mentioned, employing negation in CGs, by means of introducing a special context, yields problems. In order to overcome these difficulties, and to provide a mathematical elaboration of that fragment of CGs which corresponds to full FOL, one of the authors has extended in [19] the Wille-style formalization of SCGs by adding the cuts of Peirce’s EGs. The resulting system is called the system of CONCEPTUAL GRAPHS WITH CUTS
(CGwCs). Similar to Peirce’s EGs, the cuts are in CGwCs are drawn as ovals, but in order to distinguish them from the relation ovals, they are drawn in bold. Besides the addition of cuts, the coreference-links in CGwCs are replaced by relation ovals, labeled with the relation sign ‘=’. For example, the CGs of Fig. 9 can now be represented with the additional of cuts as follows:

![Conceptual graphs with negations](image)

CGwCs are equipped with an extensional semantics, a sound and complete diagrammatic calculus (based on the calculus for Peirce’s EGs), and meaning-preserving translation to and from FOL. Particularly, they have the expressiveness of FOL, including equality and constants, but absent are function names.

5. Spider or Constraint Diagrams for DL

At a first glance, there are some striking similarities between DL and the system of SDs and CDs. Most importantly, both correspond to decidable fragments of FOL where only unary predicates and binary relations are used. Both systems have sound and complete calculi which are implemented on tableaux-based algorithms. It is therefore worth investigating whether spider and constraint diagrams can be used as a diagrammatic representation for DL.

In our scrutiny of SDs and CDs, we first restrict ourselves to a discussion of SDs only. That is, on the DL side, we consider class constructors where only concepts, but no roles are used. This is an unreasonable restriction for DLs, but it serves well to identify some early differences between SDs and DL and test their compatibility.

Firstly, an obvious difference between DL and SDs is that DLs are used to express concepts, which can be translated into FOL-formulas with one free variable, whereas SDs are translated into sentences, i.e. FOL-formulas without free variables. So we have to extend the syntax and semantics of SDs slightly to accommodate this. This could be done by labeling spiders with variables. Let us consider a conjunction of two concepts $A$ and $B$, i.e. $A \cap B$, as a very simple example for a class constructor. With a SD, we can express the formula $\exists x.(A(x) \land B(x))$. The variable $x$ will be replaced by a spider. By labeling the corresponding spider with $x$, we provide a simple SD-like representation for the formula $A(x) \land B(x)$, where $x$ is now free.
A more complex example is the concept description $A \cap (B \cup C)$. Again, we can express $\exists x.(A(x) \land (B(x) \lor C(x)))$ with a SD. Each formula in propositional logic can be converted into its disjunctive normal form, where in each of the conjuncts, all concepts of the formula appear as literals. For example, instead of $A \cap (B \cup C)$, we consider the equivalent formula $(A \cap B \cap \neg C) \cup (A \cap B \cap C) \cup (A \cap B \cap C)$. As each conjunct corresponds to one minimal region in the diagram, we therefore have the opportunity to express each concept by means of a single spider in SDs. Similar to the last example, we provide the concept description, the SD and the corresponding DL-SD, including a free variable $x$.

The purpose of SDs is to express relationships between predicates, not to express the construction of new predicates from those given. For example, we can express the disjointness of two predicates $A$ and $B$ by either using a shading in the corresponding Venn-diagram, or by using the features of Euler circles. Both possibilities are presented below.

Atomar DL-concepts are represented in SDs by contours. The Euler-features of SDs can be used to represent entailment or disjointness of the atomar DL-concepts represented by contours. A definition of a single DL-concept does not express such set-theoretical relationships. Thus if we model a single DL-concept by means of SDs, we cannot draw benefit from the Euler-features of SDs: We need only the Venn-diagram part of SDs without shadings. On the other hand, it is possible to express relationships between sets in DL-knowledge-bases: The axiom $\bot \equiv A \cap B$ expresses that $A$ and $B$ are disjoint, and $A \subseteq B$ expresses that $A$ is a subset of $B$. That is, when we attempt to diagrammatically represent a single DL-concept of a set of DL-concepts and DL-axioms, some of the axioms can incorporated in the diagram, expressed by means of Euler-features or shadings. An example for this is given below.

In any event, the contours of mutually independent atomar DL-concepts give rise to a Venn-diagram. It is possible to draw Venn-diagrams for an arbitrary number of predicates, but if more than three predicates are used, the diagrams become difficult to read. Below is an example of a concept definition where four concepts are involved, and the corresponding SD-diagram. A more complex example will be given below.
Besides the disadvantage that Venn-diagrams with more than three predicates are hard to read, SDs present a possible diagrammatic representation for DL when no roles are used. In the following, roles are taken into account.

Let us consider two very simple concept description with a role involved: $A \sqcap \exists R.B$ and $A \sqcap \forall R.B$. For the first concept, it seems self-suggesting to use an existential arrow to express the $\exists R.B$-part of this description. Similarly, for the second concept, using a universal spider seems appropriate. Therefore, the following CDs seem to be the canonical CD-style representation of the concept descriptions:

$$
A \sqcap \exists R.B
$$

$$
A \sqcap \forall R.B
$$

Fig. 11. A first, failed attempt to express DL-restrictions

However the semantics of these CDs with a free variable and the intended DL-semantics of the concept description differ.

<table>
<thead>
<tr>
<th></th>
<th>Left Diagram</th>
<th>Right Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD-semantics</td>
<td>${x \mid x \in A \land \exists b \in B : xR^T = {b}}$</td>
<td>${x \mid x \in A \land \forall b \in B : xR^T = {b}}$</td>
</tr>
<tr>
<td>DL-semantics</td>
<td>${x \mid x \in A \land \exists b \in xR^T : x \in B}$</td>
<td>${x \mid x \in A \land \forall b \in xR^T : x \in B}$</td>
</tr>
<tr>
<td></td>
<td>$= {x \mid x \in A \land xR^T \cap B \neq \emptyset}$</td>
<td>$= {x \mid x \in A \land xR^T \subseteq B}$</td>
</tr>
</tbody>
</table>

We see that arrows having existential spiders as targets are not suited to express the ‘exists restriction’ constructor of DLs, and similarly, arrows having universal spiders as targets are not suited to express the ‘value restriction’ constructor of DLs. Moreover, note that the right diagram is not a well-formed diagram according to CDs as defined by Stapleton [77] – because the universal spider inhabits not a single zone, and the universal spider is not the source (but the target) of an arrow. On the other hand, both diagrams are well-formed CDs according to the full CD system developed by Fish et al. [29] (but recall that Fish et al. [29] do not provide a calculus for the full system).

In the CDs of Fish et al. [29], we have arbitrary many derived contours. Derived contours are basically unlabeled contours which are targets of arrows. An arrow represents a relation, so a target of arrows denotes the range of this relation. Thus derived contours provide a means to express how the range of a relation is set-theoretically related to other sets. In fact, instead of spiders, derived contours are the right entities to express
the ‘exists restriction’ and ‘value restriction’ constructors of DLs. Let us consider the two 
DL-concepts again. The concept $A$ is of course expressed by $A \sqcap \exists R.B$.

For the first concept, i.e. $A \sqcap \exists R.B$, we have to add a derived contour to this diagram, 
being the target of an arrow labeled with $R$. We do not know any subset-relationships 
between $xR^2$ and $A$ or $B$, so the diagram we obtain is similar to a Venn-diagram with 
three sets. However we know that the intersection of $xR^2$ and $B$ is not empty, so we 
add an additional existential spider to the region that depicts $xR^2 \cap B$. For the second 
concept, i.e. $A \sqcap \forall R.B$, we have again to add a derived contour to this diagram. Now this 
contour has to express that $xR^2 \subseteq B$ holds. We can use the Euler-features of CDs to 
express this. In Figure 12 the two CDs that express the two DL-concepts are depicted.

There remains a slight flaw in the CD with the exists restriction: in the system of CDs 
distinct spiders denote distinct objects so this diagram is not an appropriate translation 
of the exists restriction with CDs. In [37], the notation of strands has been introduced: 
In the diagrams, strands are wavy lines between different spiders, and they are used to 
indicate that these spiders do not necessarily denote different objects. So we could add 
a strand to our diagram. Despite such repair, strands are not considered in [29], where 
the syntax and semantics of full CDs is developed. Another possibility is to slightly alter 
the semantics of [29] by dismissing the condition that different spiders denote different 
objects. Let us assume this approach in the following.

With the discussion so far, we can now provide a CD that corresponds to the DL concept 
HAPPYMAN of equation (1). First of all, this concept contains five atomic concepts, 
namely MAN, FEMALE, MALE, RICH, and HAPPY. The definition of HAPPYMAN does 
not provide information on how these concepts are related, so a CD which corresponds 
to equation (1) is based on a Venn-Diagram with five contours. The diagram to the left 
of Figure 13 is such a diagram. As mentioned earlier, Venn-diagrams with more than 
three contours become difficult to read. To make the example easier to comprehend, 
let us assume that besides the definition of HAPPYMAN, we have moreover DL-axioms 
$\bot \equiv \text{MALE} \sqcap \text{FEMALE}$ and $\text{MAN} \sqsubseteq \text{MALE}$. Then we can start with a more simpler 
Euler-diagram, as depicted right in Figure 13.
The CD corresponding to HappyMan will be based on the right diagram. To make this CD more comprehensible, it will be build inductively. The next two diagrams in Figure 14 depict the DL concepts Man and Man \( \sqcap \forall \text{hasChild.}(\text{Rich} \sqcup \text{Happy}) \), respectively.

Note that in the second step, we had to add a derived contour, so we split some spiders as well. This contour depicts in an iconic, Euler-style way, that we have \( x^R \subseteq B \) for each \( x \) in the concept extension. On the other hand, we already have six contours, which are not easy to comprehend. In the next two steps, which are depicted in Figure 15, we add the conjuncts \( \exists \text{hasChild.} \text{Female} \) and \( \exists \text{hasChild.} \text{Male} \) to the concept definition.

In each step, another derived contour is added to the diagram. In contrast to the second step, these new contours depict sets that have a set-theoretical relationship to the other sets: we have to add the contours in a way that they separate each minimal region into two parts. This yields a diagram which unfortunately lacks readability.

We haven’t discussed a complete translation of DL-concepts into CDs, but it seems there is some prospect that a general translation can be provided. However, we have to deal with the following problems: first, we have slightly altered the semantics of CDs by dismissing the condition that different spiders denote different objects. Second, neither for the system of CDs with the original semantics, nor for our system with the slightly altered semantics, do a sound and complete calculus exists. Finally, as the previous example shows, each atomic concept and each (value or exists) restriction in DL-concepts needs a contour in the corresponding CD which quickly yields CDs that, by any reasonable measure, are less readable than the originating DL-concepts.

6. Conceptual or Existential Graphs for DL

Both CGs and DLs have some common background: they are both developed and intended as knowledge representation systems to include reasoning facilities, they were developed at nearly the same time and they have an important common ancestor, namely the semantic networks of AI (and Minski’s frame systems). In fact, in 1979, the well-known system Kl-One was developed on the basis of informal use of semantic networks in AI (see [10,11]). Kl-One was itself a diagrammatic formalism and in 1987 a well-defined, extensional, Tarski-style semantics was developed for it. Thus, in contrast to semantic networks, the diagrammatic entities in Kl-One were equipped with precise meaning. This was the starting point for the first DL [49].

Thanks to this common background, there had been several attempts to identify how and where DL and CGs were related: namely specifying where DLs are related to corresponding fragments of CGs. Coupey and Faron [17] focus on SCGs where a rather restricted DL, namely a restricted version of ALEOI, is considered. In this case only the following class constructors are considered: inverse roles (i.e., \( R^- \)), a restricted version of conjunction \( (C_1 \sqcap C_2) \), and existential restriction \( \exists R.C \). It is argued that nominals can easily be added. More importantly, due to the lack of negation in SCGs, the authors cannot express the negation of concepts \( \neg C \), disjunction \( C_1 \sqcup C_2 \), or value restriction \( \forall R.C \). Baader et al. [3] extend the results of Coupey and Faron by allowing the existential intersection of concept descriptions, intersection of roles and unary ‘one-of’ concepts, i.e. they determine a fragment of conceptual graphs that corresponds to the DL ɛLTRO₁. Again, due to the lack of negation, this DL is not propositionally closed.
Fig. 13. A Venn Diagram and an Euler Diagram for the Atomar Concepts of HAPPYMAN

Fig. 14. CDs for MAN and MAN $\cap \forall$hasChild. (RICH $\cup$ HAPPY)

Fig. 15. CDs for $\forall$x hasChild. FEMALE $\cap \forall$x hasChild. (RICH $\cup$ HAPPY) and HAPPYMAN
As well similarities between DL and CGs, there are important differences. Firstly, similar to SDs and CGs, all graphs (i.e., both CGs and EGs) correspond to closed FOL formulas. Next, in the graphs, relations of arbitrary arities are allowed. There is no correspondence to the type-hierarchy of CGs in DL. Finally, the syntactical possibilities of the graphs, including identity, graphs that contain circles, graphs that are not connected etc, allow graphs to be constructed that do not have counterparts in DL. However, our intention with this paper is not to explore direct correspondences between graphs and DL, but rather using the graphs as a diagrammatic version of DL. Therefore we are free to dismiss the features of CGs not needed (for example, it is reasonable to consider only graphs where binary relations are allowed and where we have only a flat type hierarchy). On the other hand, the lack of free variables in the graphs is a absent feature which we need for DLs. As such, a first task is to discuss how free variables can be added to graphs.

Adding free variables to graphs is relatively straight forward. For existential graphs, Burch [13] focuses on an algebraic elaboration of EGs. Inspired by Burch, we find other approaches in the works of Pollandt [59,60], Wille [87], Hereth-Correia and Pöschel [40,41], and Hereth-Correia and Dau [27]. The resulting graphs are usually called relation graphs (RG), as they describe relations instead of propositions.

The diagrammatic rendering of free variables is via numbered question markers. In Figure 16 a relation graph, and the corresponding CGwC with free variables, are depicted.

![Diagram](image)

**Fig. 16. An relation graph and the CGwCs with free variables**

In the following, we focus on the logic $\mathcal{ALC}$, i.e. the DL which encompasses inverse roles, conjunction, disjunction, negation, value restriction and exists restriction. The main reason for this choice is that $\mathcal{ALC}$ is the smallest propositionally closed DL $\mathcal{ALC}$.

As we go for a propositionally closed DL, we do not consider SCGs, but only RGs and CGwCs.

Considering other DL-constructs is the subject of further research. As briefly discussed in the outlook, first unary ‘one-of’ concepts, inverse roles, and the intersection of roles will be targeted, as is it likely that adding these constructs will not be too problematic. Once this is done, we obtain a strict extension of the the results of [17,3].

We consider again the DL-concept of equation (1) as an example. First, it should be noted that in the graph-based systems of RGs and CGwCs we have no genuine opportunity to express disjunction or universal quantification. However this does not present any difficulty since both can be alternatively expressed by means of conjunction, negation, and existential quantification (the basic operators in RGs and CGwCs). On the DL side,

---

12 Another source is the appendix of [23], where the calculus of EGs is extended to a a sound and complete calculus for RGs.

13 For CGwCs, free variables have been added to them in [20].
$C_1 \sqcup C_2$ can be replaced by $\neg(\neg C_1 \sqcap \neg C_2)$, and $\forall R.C$ can be replaced by $\neg\exists R.\neg C$. Therefore, we consider the equivalent formula,

\[
\text{MAN} \sqcap \exists \text{hasChild} \text{.FEMALE} \sqcap \exists \text{hasChild} \text{.MALE} \sqcap \neg\exists \text{hasChild} \text{.}(\neg \text{RICH} \sqcap \neg \text{HAPPY})
\]

and provide “naive” translations of it to CGwCs and an RG.

Baader et al. [3] consider SCGs that have one distinguished node termed root. We adopt this idea by providing CGwCs where a distinguished concept box $\text{T:}?$, placed on the sheet of assertion, is used to describe the concept description (note that as concept descriptions of DL correspond to FOL-formulas with only one free variable, there is no need to use numbered question markers).

There are two possibilities to transform this concept description into a CGwC. On the one hand, we can translate the atomic concepts of DL to types in CGwCs, i.e. they appear in concept boxes. On the other hand, we can both translate atomic concepts and roles to relations in CGwCs, i.e. roles appear in relation ovals. The CGwCs corresponding to the given concept description are depicted in Figure 17.

Note that for the left graph of Figure 17, we need a generic marker for each concept of the concept description, as well as for each subformula $\exists R.C$ or $\forall R.C$, where $C$ is not an atomic concept. In the graph to the right, we need a generic marker for each role of the concept description, which now appears in concept boxes $\text{T:}?$ . In the graph left, we need additional identity links\(^{14}\).

Next in Figure 18 an RG for the concept description is depicted. In EGs and RGs both existential quantification and the identity of objects is modeled with ligatures. We can rid ourselves of identity links in CGwCs (see left example in Figure 17) and concept boxes $\text{T:}?$ (see right example in Figure 17).

These graphs give rise to the following conclusions: first, it is possible to provide fragments of CGwCs or RGs which correspond to $\mathcal{ALC}$. Second, as both identity links and concept boxes $\text{T:}?$ are encompassed by lines of identity in RGs, the RG is easier to comprehend than the two CGwCs.

Compared to SDs and CDs we see a more natural translation from $\mathcal{ALC}$-concepts to existential graphs than to CDs. Moreover, for RGs, there exists a sound and complete

\(^{14}\)A short discussion on the need for identity links in translating DLs to CGs is provided in [17].
calculus, whereas such a calculus is absent for CDs. Although CDs might still be elaborated as a diagrammatic reasoning system for ALC, the discussion of the last sections leads to the conclusion that RGs seem the most promising system to formalize the DL ALC as a diagrammatic reasoning system. For this reason, in the next sections we will determine a fragment of RGs corresponding to ALC.

7. A Tree-based Formalization of The Description Logic ALC

In this section, the syntax and semantics for the DL ALC is provided. First of all, as mentioned earlier in Section 2, the DL ALC has conjunction, negation and value restriction as constructors. As the RGs of Peirce have conjunction, negation and existential quantification as constituents, we can consider ALC made up from conjunction, negation and existential restriction (instead of value restriction).

Let RG^{ALC} denote the class of RGs that will be used to formalize ALC. The graphs of RG^{ALC} will be called ALC-graphs. The class RG^{ALC} is a subclass of all RGs and the graphs of RG^{ALC} exhibit significant additional constraints. First, for a graph in RG^{ALC}, only unary predicates and binary relations are used. The graph has exactly one pending edge labeled with ‘?’ and this is placed on the sheet of assertion. Moreover, each cut is crossed exactly once by a heavily drawn line (particularly, we must have no empty cuts).

Most importantly, the underlying graph structure without the cuts is a tree. The existing formalizations of the full system of RGs have to capture more complex structures. For the purpose of this paper, these formalizations are technically overloaded, thus we provide a new formalization by means of labeled trees. Using trees is, on the one hand, close to the usual approach to inductively define the formulas of ALC. On the other hand, we can easily provide for each tree the corresponding RG-diagrams (which will be done in the next section), and labeled trees are better suited than formulas when we finally employ a Peirce-style calculus for ALC.

Trees can be formalized either as rooted and acyclic graphs or as special posets. We adopt the second approach, i.e., a tree is a poset \((T, \geq)\), where \(t \geq s\) can be understood as ‘s is an ancestor of t’. A labelled tree is a structure \(T := (T, \leq, \nu)\), where \((T, \leq)\) is a tree and \(\nu : T \to L\) is a mapping from the set of nodes to some set \(L\) of labels. The greatest element of \(T\) is the root of the tree. As usual, each node \(v\) gives rise to a subtree \(T_v\) (formally, \(T_v = (T_v, \geq \mid_{T_v \times T_v}, \nu|_{T_v})\) with \(T_v := \{w \in T \mid v \geq w\}\)). We write \(T' \subseteq T\), if \(T'\) is a subtree of \(T\). Isomorphic labeled trees are implicitly identified.

Next, we introduce operations to inductively construct labelled trees. These operations
will be used to define the syntax of $\mathcal{ALC}$. We assume to have a set $L$ of labels.

**Chain:** Let $l_1, \ldots, l_n \in L$. With $l_1 l_2 \ldots l_n$ we denote the labelled tree $T := (T, \geq, \nu)$ with $T := \{v_1, \ldots, v_n\}$, $v_1 > v_2 > \ldots > v_n$ and $\nu(v_1) = l_1, \ldots, \nu(v_n) = l_n$. That is, $l_1 l_2 \ldots l_n$ denotes a chain, where the nodes are labelled with $l_1, l_2, \ldots, l_n$, respectively.

We extend this notation by allowing the last element to be a tree: If $l_1 l_2 \ldots l_n \in L$ and if $T'$ is a labelled tree, then $l_1 l_2 \ldots l_n T'$ denotes the labelled tree $T := (T, \geq, \nu)$ with $T := T' \cup \{v_1, \ldots, v_n\}$, $v_1 > v_2 > \ldots > v_n$ and $v_1 > v$ for each $i = 1, \ldots, n$ and $v \in T'$, and $\nu := \nu' \cup \{(v_1, l_1), \ldots, (v_n, l_n)\}$, i.e., $T$ is obtained by placing the chain $l_1 l_2 \ldots l_n$ above $T'$.

**Substitution:** Let $T_1, T_2$ be labelled trees and $S := (S, \geq, \nu_S)$ a subtree of $T_1$. Then $T := T_1[S / T_2]$ or $T := \left[\frac{S}{T_2}\right]$ denotes the labelled tree obtained from $T_1$ when $S$ is substituted by $T_2$. Formally, we define $T := (T, \geq, \nu)$ where we set $T := (T_1-S) \cup T_2$, $\geq := \geq_1 \cup \geq_2 \cup \{(w_1, w_2) \mid w_1 > v, w_1 \in T_1-S, w_2 \in T_2\}$, and $\nu := \nu_1 \cup \nu_2 \cup \{(v, l)\}$. That is, $T$ is the tree having a root labelled with $l$ and which has $T_1$ and $T_2$ as (direct) subtrees.

Strictly speaking, in the above operations we have sometimes to consider trees with disjoint sets of nodes (for example, we have to assume in $T_1[S / T_2]$ that $T_1$ and $T_2$ are disjoint). As we consider trees only up to isomorphism, this can always easily be achieved and is usually not explicitly mentioned.

Before the syntax and semantics of $\mathcal{ALC}$ based on labelled trees is introduced, let us more clearly define the vocabulary we will use. As we focus on $\mathcal{ALC}$ we only need to consider atomic concepts and atomic roles but no individuals. Therefore a vocabulary is defined to be a pair $(A, R)$ where $A$ is a set of (atomic) concepts and $R$ is a set (atomic) roles. We also assume to have a top-concept $\top \in A$. In an interpretation $(\Delta^T, I)$, the interpretation function $I$ assigns to every atomic concept $A \in A$ a set $A^T \subseteq \Delta^T$ and to every atomic role $R \in R$ a binary relation $R^T \subseteq \Delta^T \times \Delta^T$, and we have $I(\top) = \Delta^T$.

Now the tree-based syntax and semantics of $\mathcal{ALC}$ is defined. To distinguish this syntax from the usual notation of $\mathcal{ALC}$ by means of formulas, this system is called $\mathcal{ALC}^{\text{Tree}}$.

**Definition 7.1 (Syntax and Semantics of $\mathcal{ALC}^{\text{Tree}}$)** Let $(A, R)$ be a vocabulary, where $A$ is a set of atomic concept names and where $R$ is a set of atomic role names. We assume we have a further atomic name $\top$. Moreover, let `\land` and `\neg` be two further signs, denoting conjunction and negation. Let $(\Delta^T, I)$ be an interpretation for the vocabulary $(A, R)$. We inductively define the elements of $\mathcal{ALC}^{\text{Tree}}$ as labelled trees $T := (T, \geq, \nu)$, as well as the interpretation $I(T)$ of $T$ in $(\Delta^T, I)$.

**Atomic Tree for $\top$:** The labelled tree $T := (\{v, \top\})$ is in $\mathcal{ALC}^{\text{Tree}}$. We set $I(T) = \Delta^T$.

**Atomic Tree for $A$:** If $A$ is an atomic concept, then the tree $T_A := (\{v, A\})$ is in $\mathcal{ALC}^{\text{Tree}}$. We set $I(T) = A^T$.

**Negation:** Let $T \in \mathcal{ALC}^{\text{Tree}}$, let $v$ be a fresh vertex. Then $T' := \neg T := \neg T$ is in $\mathcal{ALC}^{\text{Tree}}$. We set $I(T') = \Delta^T - I(T)$.

**Conjunction:** Let $T_1, T_2 \in \mathcal{ALC}^{\text{Tree}}$. Then the tree $T := (T_1 \land T_2)$ is in $\mathcal{ALC}^{\text{Tree}}$. We set $I(T) = I(T_1) \cap I(T_2)$. 

26
**Exists Restriction:** Let $T \in \text{ALC}^{\text{Tree}}$, let $R$ be a role name. Then $T' := RT := RT$ is in $\text{ALC}^{\text{Tree}}$. We set $\mathcal{I}(T') = \{ x \in \Delta^T \mid \exists y \in \Delta^T : xR^Ty \land y \in \mathcal{I}(T) \}$. The labeled trees of $\text{ALC}^{\text{Tree}}$ are called $\text{ALC}$-trees.

Let $T := (T, \geq, \nu) \in \text{ALC}^{\text{Tree}}$. An element $v \in T$ respectively the corresponding subtree $T_v$ is said to be evenly enclosed, iff $|\{w \in T \mid w > v \land \nu(w) = \neg\}|$ is even. The notation to be oddly enclosed is defined accordingly.

Of course, $\text{ALC}$-trees correspond to the formulas of $\text{ALC}$, as they are defined in the usual linear fashion. For this reason, we will sometimes mix the notation of $\text{ALC}$-formulas and $\text{ALC}$-trees. Particularly, we sometimes write $T_1 \sqcap T_2$ instead of $\sqcap(T_1, T_2)$. Moreover, the conjunction of trees can be extended to an arbitrary number of conjuncts, i.e. if $T_1, \ldots, T_n$ are $\text{ALC}$-trees, we are free to write $T_1 \sqcap \ldots \sqcap T_n$. We agree that for $n = 0$, we set $T_1 \sqcap \ldots \sqcap T_n := \top$.

Now we define semantic entailment between $\text{ALC}$-trees.

**Definition 7.2 (Semantical Entailment)** Let $\{T_i \mid i \in I\}$ be a set of $\text{ALC}$-Trees and let $T$ be an $\text{ALC}$-Tree. We set

$$\{T_i \mid i \in I\} \models T :\iff \bigcap_{i \in I} \mathcal{I}(T_i) \subseteq \mathcal{I}(T)$$

for each interpretation $(\Delta^T, \mathcal{I})$. For $I = \emptyset$, we set $\bigcap_{i \in I} \mathcal{I}(T_i) := \Delta^T$ for the respective model, and write $\models T$. For $|I| = 1$, we write $T' \models T$.

8. A Diagrammatic Representation for $\text{ALC}^{\text{Tree}}$

In Section 4.1, we briefly discussed that C.S. Peirce distinguished between graphs and graph replicas. As it is more generally argued in [43,21], any formalization of logic by means of diagrams has to distinguish these two levels, usually called types or abstract syntax (which correspond to Peirce’s graphs), and tokens or concrete syntax (corresponding to Peirce’s replicas). The $\text{ALC}$-trees, as they have been defined in the last section, are obviously the abstract syntax. In this section we explain how each $\text{ALC}$-tree can be diagrammatically represented in the style of Peirce’s graphs, i.e., the token-level is introduced.

First, we will use the letter $G$, sometimes with indices, to denote the graphs (i.e., diagrams) of $\text{RG}^{\text{ALC}}$. As stated in the last section we will only consider RGs that have a single query marker placed on the sheet of assertion, i.e. there will be one heavily drawn line on the sheet of assertion with a “?” attached to it. We will call this the pending edge of the graph. In order to emphasize this pending edge of a graph $G$ we will often represent it in the following manner: $\text{---}G$.

To each $\text{ALC}$-tree $T$, we now assign corresponding relation graph diagrams $\Psi(T)$ with one query marker. Let $A$ be an atomic concept, $R$ be a role name, let $T, T_1, T_2$ be $\text{ALC}$-trees where we already have defined $\Psi(T) = \text{---}G$, $\Psi(T_1) = \text{---}G_1$, and $\Psi(T_2) = \text{---}G_2$, respectively. Now $\Psi$ is defined inductively as follows:
In the following, the term ‘\(\mathcal{ALC}\)-graph’ will be used to denote the diagrammatic, Peirce-style representation of \(\mathcal{ALC}\)-trees, i.e. with ‘\(\mathcal{ALC}\)-tree’ we refer to the type-level and with ‘\(\mathcal{ALC}\)-graph’ we refer to the token-level.

In this definition we implicitly used diagrammatic conventions to denote some operations on RGs. For example, in the 3rd step (corresponding to conjunction), the pending edges of \(\mathcal{G}_1\) and \(\mathcal{G}_2\) have been joined in a newly generated branching point; and in the 4th step, the pending edge of \(\mathcal{G}\) is extended outwards through the newly generated cut. According to Peirce's convention \textit{No. Zero}, we may vary certain features of the diagram, like the shape of the cut or the arrangement of the entities on the plane. Considering our \(\mathcal{ALC}\)-formula example given in the introduction, the corresponding \(\mathcal{ALC}\)-tree (the type), and two corresponding Peircean diagrams (two tokens), are provided in Figure 19. Finally, in the full system of RGs, we need to label the edges adjacent to a relation name \(R\) with numbers indicating the order of the arguments of \(R\). As the RGs of \(\mathcal{RG}^{\mathcal{ALC}}\) have a tree-like structure, we can omit this labeling.

We have to this point provided a translation of \(\mathcal{ALC}\)-trees into diagrams of Peirce’s graphs. If we have on the other hand a graph \(\mathcal{G}\), the first diagrammatic entity we reach from the question mark is either (if non-empty) an atomic concept, an atomic role, a branching point, or a cut, and from this entity we see the last step in the construction.
of the $\text{ALC}$-tree that corresponds to the diagram\textsuperscript{15}. Therefore, for any given diagram of $\text{RG}^{\text{ALC}}$ we can reconstruct the corresponding $\text{ALC}$-tree with ambiguity.

We have established a one-to-one correspondence between $\text{ALC}$-trees and their diagrammatic representations in the form of graphs of $\text{RG}^{\text{ALC}}$. The rules of the forthcoming calculus can be best understood to be carried out on the diagrammatic representations. Further, the ongoing formal proofs with $\text{ALC}$-trees will be depicted this way.

9. The Calculus for $\text{ALC}^{\text{Tree}}$

C.S. Peirce provided a set of five rules for the system of existential graphs termed erasure, insertion, iteration, deiteration, double cut. It is possible to elaborate Peirce’s graphical calculus, including a proof of its soundness and completeness, in a mathematical precise manner, and to extend the calculus for the system of RGs, but this goes beyond the scope of this paper.

$\text{RG}^{\text{ALC}}$ is a fragment of the full system of RGs. As one would expect, the rules for RGs are also sound rules for $\text{RG}^{\text{ALC}}$. However it is not clear whether these rules are also complete. For two graphs $G_1, G_2 \in \text{RG}^{\text{ALC}}$ with $G_1 \models G_2$, we have a proof for $G_1 \vdash G_2$ within the full system of RGs, i.e., a sequence of graphs starting with $G_1$, ending with $G_2$, where each graph in the sequence is derived from its predecessor by one of Peirce’s five rules. Despite this it might happen that we do not have a proof that consists only of graphs of $\text{RG}^{\text{ALC}}$. In fact, in the calculus we provide, we require more rules. Besides some trivial rules, like rules which capture the associativity of conjunction, we need special rules for handling roles. The rules iteration of roles into even and deiteration of roles from odd (see below) are the most important examples of this.

Next, the Peirce style rules for $\text{ALC}^{\text{Tree}}$ are provided. These rules transform a given $\text{ALC}$-tree into a new $\text{ALC}$-tree. In order to make the calculus more understandable, examples and diagrams that illustrate the operation of the rules are provide within their definitions. For each rule name, we provide an abbreviation that will subsequently be used in the proofs.

**Definition 9.1** The calculus for $\text{ALC}$-Trees over a given vocabulary $(A, R)$ consists of the following rules:

**Addition and Removal of $\top$ ($\top$-add. and $\top$-rem.)** Let $T := (T, \geq, \nu)$ be an $\text{ALC}$-tree, let $S \subseteq T$ be a subtree.

\[
T' := T \square \left[ \frac{S}{\top\left(\left[S, \top\right]\right)} \right]
\]

we set $T \vdash T'$.

($T \vdash T'$ abbreviates $T \vdash T'$ and $T' \vdash T$). We say that $T'$ is derived from $T$ by adding a $\top$-NODE, and $T$ is derived from $T'$ by removing a $\top$-NODE. For the $\text{ALC}$-graphs,

\textsuperscript{15}In fact each $\text{ALC}$-tree, in turn each graph of $\text{RG}^{\text{ALC}}$, has a unique derivational history. This is a common feature of linear and symbolic notations of logic. However, for inductively defined diagrammatic versions of logic, this feature is often absent. Having a unique derivational history provides a unique reading of a formula but diagrams often provide multiple readings, namely a given diagram can be read in different although semantically equivalent ways ([67]). The point is that in diagrammatic systems it is not necessarily desirable to provide a unique derivational history, see [21,67] for a deeper discussion on this issue.
this rule corresponds to adding a branch to a heavily drawn line, or removing it. A simple example is given below.

\[ \vdash \neg R \rightarrow \top \]

\[ \vdash \neg \neg \top \rightarrow R \]

These rules are "technical 'helper' rules that will be often combined with other rules that add or remove subtrees. Examples will be given below.

**Addition and Removal of Roles (R-add. and R-rem.):** Let \( T \) be an ALC-tree having \( \top \) as a subtree. Let \( R \) be a role name.

For \( T \vdash \neg R \rightarrow \top \)

We say that \( T \) is derived from \( T' \) by adding the role \( R \), and \( T' \) is derived from \( T \) by removing the role \( R \). A simple example for this rule with ALC-graphs is given below.

Due to the symmetry of the rules, the inverse direction of this proof is a proof as well.

**Associativity of Conjunction (conj.):** Let \( T \) be an ALC-tree let \( S_1 \sqcap (S_2 \sqcap S_3) \subseteq T \) be a subtree.

For \( T' := T \sqcap (S_1 \sqcap (S_2 \sqcap S_3)) \)

We say that \( T' \) is derived from \( T \) by using the associativity of conjunction.

**Addition and Removal of a Double Negation (dn):** Let \( T := (T, \geq, \nu) \) be an ALC-tree, let \( S \subseteq T \) be a subtree.

For \( T' := T \sqcup S \)

We say that \( T' \) is derived from \( T \) by adding a double negation and \( T \) is derived from \( T' \) by removing a double negation.

To see an example, consider the four ALC-graphs below. The second and third graph can be derived from the first by adding a double negation. The opposite direction can be applied symmetrically. The fourth graph is a result of adding a double negation in the general theory of RGs, but in the system of ALC-trees, this is even not a diagram of an ALC-tree, as the cuts cross more than one heavily drawn line.

**Erasure from even, Insertion into odd (era. and ins.):** Let \( T := \) be an ALC-tree with a positively enclosed subtree \( S \subseteq T \).

For \( T' := T[S \rightarrow \top] \)

We say that \( T' \) is derived from \( T \) by erasing \( S \) from even. In the opposite direction, let \( T := \) be an ALC-tree with an negatively enclosed subtree \( \top \subseteq T \). Let \( S \in ALC^{Tree} \).

For \( T' := T[S \rightarrow \top] \)

We say that \( T' \) is derived from \( T \) by inserting \( S \) into odd. In the opposite direction, let \( T := \) be an ALC-tree with a positively enclosed subtree \( \top \subseteq T \). Let \( S \in ALC^{Tree} \).
We say that $T'$ is derived from $T$ by inserting $S$ into odd.

This is another set of rules that often go together with the addition and removal of $\top$. Examples will be given later.

**Iteration and Deiteration (it. and deit.):** Let $T := (T, \geq, \nu) \subseteq T$. Let $s$ be the root of $S$, let $t$ be the parent node of $s$ in $T$. Let $\nu(t) = \sqcap$, let $v \in T$ be a node with $v < t$, $v \notin S$, $\nu(v) = \top$, such that for each node $w$ with $t > w > v$ we have $\nu(w) = \neg$ or $\nu(w) = \sqcup$.

For $T':= T \{S', \top\}$ we set $T \vdash T'$.

(More precisely and according to our convention, we set $T':= T \{S'/T\}$, where $S'$ is an isomorphic copy of $S$, having only fresh nodes.) We say that $T'$ is derived from $T$ by iterating $S$ and $T$ is derived from $T'$ by deiterating $S$.

Iteration and Deiteration often combine with the addition and removal of $\top$, and they are probably the most complex rules. To exemplify them, we consider the following six $\mathcal{ALC}$-graphs. The second and the third graph can be derived from the first graph by iterating the subgraph $\square$ (preceded by the $\top$-addition rule). The next three graphs are not results from the iteration rule. In the fourth graph, the condition that $\nu(w) = \neg$ or $\nu(w) = \sqcup$ holds for each node $w$ with $t > w > v$ is violated. The fifth graph violates the condition $v < t$. Finally, the sixth graph violates the condition $v \notin S$.

**Iteration of Roles into even, Deiteration of Roles from odd (R-it., R-deit.):** Let $T$ be an $\mathcal{ALC}$-tree. Let $S_a, S_b, S_1, S_2$ be $\mathcal{ALC}$-trees with $S_a := \sqcap\{RS_1, \neg RS_2\}$ and $S_b := R \sqcap \{S_1, \neg S_2\}$. Moreover, let $S_a \subseteq T$ be a positively enclosed subtree.

For $T':= T[S_b/S_a]$ we set $T \vdash T'$, and we say that $T'$ is derived from $T$ by deiterating the role $R$ from odd. Vice versa, let $S_b \subseteq T$ be a negatively enclosed subtree.

For $T':= T[S_a/S_b]$ we set $T \vdash T'$, and we say that $T'$ is derived from $T$ by iterating the role $R$ into even.

Below, a simple example for the rule with Peirce’s graphs is provided.
Based on these rules, we can now define formal proofs.

**Definition 9.2** Let $T_a, T_b$ be two $\mathcal{ALC}$-Trees. A proof for $T_a \vdash T_b$ is a finite sequence $(T_1, T_2, \ldots, T_n)$ with $T_a = T_1$, $T_b = T_n$, where each $T_{i+1}$ is obtained from $T_i$ by applying one of the rules of the calculus.

Now let $\{T_i \mid i \in I\}$ be a set of $\mathcal{ALC}$-Trees and let $T$ be an $\mathcal{ALC}$-Tree. We set

$$\{T_i \mid i \in I\} \vdash T \iff \text{there are } \mathcal{ALC}\text{-Trees } T_1, \ldots, T_n \in \{T_i \mid i \in I\}
\text{with } T_1 \sqcap \ldots \sqcap T_n \vdash T$$

Before the soundness and completeness of the calculus is presented, we first present an example of a proof, using the Peirce-style diagrams, and then derive some useful meta-rules. The example and the meta-rules will give some insights in how the calculus works.

A popular toy example for reasoning within $\mathcal{ALC}$ is the mad cow ontology. Consider the following $\mathcal{ALC}$-definitions:

- $Cow \equiv Animal \sqcap Vegetarian$
- $Sheep \equiv Animal \sqcap \text{hasWool}$
- $Vegetarian \equiv \forall\text{eats.}\neg\text{Animal}$
- $MadCow \equiv Cow \sqcap \exists\text{eats.Sheep}$

The question to answer is whether this ontology is consistent. This question can be reduced to rewriting the ontology to a single concept

$$MadCow \equiv Animal \sqcap \forall\text{eats.}\neg\text{Animal} \sqcap \exists\text{eats.}(Animal \sqcap \text{hasWool})$$

and investigating whether this concept is satisfiable, i.e., whether there exists at least one interpretation where this concept is a non-empty set. We will show that this is not the case by proving with our calculus that the concept entails the absurd concept. The proof is given below.

We started with the $\mathcal{ALC}$-graph for the given concept and derived the absurd concept, thus the ontology is not satisfiable.
Next, we provide the above mentioned metarules, some of them will be used in the completeness proof.

Each rule of the calculus is basically the substitution of a subtree of a given \(\mathcal{ALC}\)-tree by another subtree. Each rule can be applied to arbitrarily deeply nested subtrees, although we need to keep track in order to distinguish whether a subtree is positively and negatively enclosed. Moreover, if we have a rule that can be applied to positively enclosed subtrees, then we always have a rule in the converse direction that can be applied to negatively enclosed subtrees (and visa versa). Due to these structural properties of rules, we immediately obtain the following helpful lemma (adopted from [70]).

**Lemma 9.1 (Cut-and-Paste Theorem)** Let \(S_1, S_2\) be two \(\mathcal{ALC}\)-trees with \(S_1 \vdash S_2\). Let \(T\) be an \(\mathcal{ALC}\)-tree. Then if \(S_1 \subseteq T\) is a positively enclosed subtree of \(T\), we have \(T \vdash T[S_2 / S_1]\). Visa versa, if \(S_2 \subseteq T\) is a negatively enclosed subtree of \(T\), we have \(T \vdash T[S_1 / S_2]\).

The next lemma corresponds to the equivalence of the \(\mathcal{ALC}\)-concepts \(\forall R. (C \sqcap D)\) and \(\forall R. C \sqcap \forall R. D\).

**Lemma 9.2 (Splitting Roles)** Let \(S_1, S_2\) be \(\mathcal{ALC}\)-trees, let \(R \in \mathcal{R}\) be a role name. Then the trees

\[
\begin{align*}
\text{and }
\begin{array}{c}
\text{?} \quad R \quad S_2 \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quarter
For $\mathcal{ALC}$, the full deduction theorem holds.

**Theorem 9.3 (Deduction Theorem)** Let $T$ be a set of $\mathcal{ALC}$-trees, let $T_1, T_2$ be two $\mathcal{ALC}$-trees. Then we have

$$T \cup \{T_1\} \vdash T_2 \iff T \vdash \neg (T_1 \cap \neg T_2)$$

**Proof:** In the ongoing proofs, again applications of the $\top$-removal/addition rule are not explicitly mentioned. We start with $\Rightarrow$. Let $S_1, \ldots, S_n \in T$ with $S_1 \cap \ldots \cap S_n \cap T_1 \vdash T_2$. We have:

$$\vdash S_1 \ldots S_n \vdash \neg S_1 \ldots S_n T_1 \vdash \neg S_1 \ldots S_n$$

For $\Leftarrow$, we have $T \vdash \neg (T_1 \cap \neg T_2)$ and $T \cup \{T_1\} \vdash T_1$, thus $T \cup \{T_1\} \vdash T_1 \cap \neg (T_1 \cap \neg T_2)$. We proceed as follows:

$$\vdash T_1 \vdash \neg T_2$$

The next lemma corresponds to the rule of necessitation in modal logics.

**Lemma 9.3** If $T$ is an $\mathcal{ALC}$-tree with $\vdash T$, then we have $\vdash \neg R \cdot T$ as well.
Proof: All rules of the calculus modify subtrees $S$ of a given $\mathcal{ALC}$-tree $T$, and their application depends only on whether $S$ is positively or negatively enclosed. So if $(T_1, T_2, \ldots, T_n)$ with $T_1 = \top$ and $T_n = T$ is a proof for $T$, then $(\top, \neg R \cdot T_1, \neg R \cdot T_2, \ldots, \neg R \cdot T_n)$ is a proof for $\top \vdash \neg R \cdot \top$. The additional first step is an application of the rule ‘addition of roles’.

Please note that for this lemma, it is vital that $T$ is derived from the empty set (the empty sheet of assertion in Peirce’s terminology). The proof of the lemma does not work if $T$ is derived from some set $\mathcal{I}$, and it can easily be seen that we generally cannot conclude $\mathcal{I} \vdash \neg R \cdot \top$ from $\mathcal{I} \vdash T$.

In the following, the soundness and completeness of the calculus is proven. Common calculi (like Hilbert-style calculi, natural deduction, or sequent calculi) are shallow in the understanding that the rules of the calculus modify formulas only at their top-level. In constrast, the rules presented in this paper are deep rules, as they modify deeply nested subtrees. For this reason, the following lemma is helpful for proving the soundness of the rules.

**Lemma 9.4** Let $T_1$, $T_2$, $S_1$, $S_2$ be $\mathcal{ALC}$-trees with $T_2 = T_1[S_2/S_1]$.

(i) If $S_1 \models S_2$ and the substitution takes place in an even, then $T_1 \models T_2$.

(ii) If $S_2 \models S_1$ and the substitution takes place in an odd, then $T_1 \models T_2$.

(iii) If $S_1$ and $S_2$ are semantically equivalent, then so are $T_1$ and $T_2$.

**Proof:** The proof of this lemma is a straight-forward induction on $\mathcal{ALC}$-trees.

We are now prepared to prove the soundness of the calculus.

**Theorem 9.4 (Soundness of the Calculus)** If $(\Delta^T, \mathcal{I})$ is a model and if $T'$ is obtained from $T$ by one of the rules of the calculus, we have $\mathcal{I}(T) \subseteq \mathcal{I}(T')$.

**Proof:** The $\mathcal{ALC}$-trees $S$ and $\top(S, \top)$ are obviously equivalent. So the soundness of the rule ‘Addition or Removal of $\top$’ follows immediately from Lemma 9.4(3). The rules ‘Addition or Removal of Roles’, ‘Associativity of Conjunction’ and ‘Addition or Removal of a Double Negation’ are handled similarly.

Next, as we have $T \models \top$, Lemma 9.4(1) yields the soundness of the erasure of a positively enclosed subtree, and Lemma 9.4(2) yields the soundness of the insertion negatively enclosed subtree.

Next we consider the iteration and deiteration of roles. Let $S_a, S_b$ be defined as in the rule. We will show $S_a \models S_b$. Let $a \in I(S_a)$. Then it follows $a \in I(RS_1)$ and $a \in I(\neg R(S_2))$. Therefore there exists $b \in \Delta^T$ with $aRb$ and $b \in I(S_1)$, but there exists no $c \in \Delta^T$ with $aRc$ and $c \in I(S_2)$. Particularly, we have $b \notin I(S_2)$. We conclude $b \in I(\neg S_2)$, so we have $b \in I(\neg(S_1, \neg S_2))$ as well. Due to $aRb$, we finally obtain $a \in I(S_b)$. As we now have $S_a \models S_b$, Lemma 9.4(1) yields the soundness of deiterating a role $R$ from an odd, and Lemma 9.4(2) yields the soundness of iterating a role into an even.

Finally, we have to prove the soundness of the iteration and deiteration rule. First note the iteration rule removes $v$ from $T$ and adds the fresh nodes of $S'$ to $T$, i.e., we have $T - \{v\} \subseteq T'$. To ease the technical presentation, let us assume that the greatest element of $S'$ is $v$ (instead of a fresh node), so that we have $T \subseteq T'$.

For a node $w \in T$, let $T_w$ be the corresponding subtree of $T$, and let $T'_w$ be the corresponding subtree of $T'$ (particularly, due to our assumption, we have $T_v = \top$ and $T'_v = \top$).

35
We will prove that $T_t$ and $T'_t$ are semantically equivalent. So let $a \in \Delta^T$. We have to show that,

$$a \in \mathcal{I}(T_w) \iff a \in \mathcal{I}(T'_w)$$

(2)

holds for $w = t$. We have $T' = T[\mathcal{T}_t / T_t]$, so once Equation (2) is proven for $w = t$, we can now apply Lemma 9.4(3) and obtain that $T$ and $T'$ are semantically equivalent, which yields the soundness of the iteration and deiteration rule. So it remains to show Equation (2).

In either $T$ and $T'$, the node $t$ has two branches, one of which is $S$. If we have $a \notin \mathcal{I}(S)$, we have $a \notin \mathcal{I}(T_t)$ and $a \notin \mathcal{I}(T'_t)$ as well, so Equation (2) holds. Now let us assume the alternate case, that we have $a \in \mathcal{I}(S)$. We will prove that Equation (2) holds for each $w$ with $t \geq w \geq v$ by induction over $w$.

We have $T_v = \top$, $T'_v = S$, $a \in \mathcal{I}(\top)$ and $a \in \mathcal{I}(S')$, so Equation (2) holds for $w = v$.

For the induction step, let $w$ be such that the induction hypothesis is proven for the child $u$ of $w$ with $u \geq v$. There are two cases to consider: $\nu(w) = \neg$ or $\nu(w) = \top$.

For $\nu(w) = \neg$, we have $T_w = \neg T_u$ and $T'_w = (w, \neg) \oplus T'_u$. As we have $a \in \mathcal{I}(T_u) \iff a \in \mathcal{I}(T'_u)$ due to the induction hypothesis, we obtain that Equation (2) holds for $w$.

For $\nu(w) = \top$, $w$ has two children, one of them being $u$. Let $u'$ be the child of $w$ which is different from $u$. Then we have $T_w = \sqcap(T_u, T_{u'})$ and $T'_w = \sqcap(T'_u, T'_{u'})$. Moreover, we have $T'_{u'} = T'_{u'}$, and $a \in \mathcal{I}(T_u)$ holds due to the induction hypothesis. From this we conclude that Equation (2) holds.

This finishes the induction, so we conclude Equation (2) for $w = t$, which in turn finishes the proof for the soundness of the iteration and deiteration rule.

We are now prepared to prove the completeness of the calculus. The following proof is a standard proof for the completeness of a Hilbert-style calculus for the multi-modal logic $K$, adapted to our system of $\mathcal{ALC}$-trees.

**Theorem 9.5 (Completeness of the Calculus)** Let $\mathcal{T} := \{T_i \mid i \in I\}$ be a set of $\mathcal{ALC}$-Trees and let $T$ be an $\mathcal{ALC}$-Tree. Then we have:

$$\{T_i \mid i \in I\} \models T \implies \{T_i \mid i \in I\} \vdash T$$

**Proof:** We assume that there is no derivation of $T$ from $\mathcal{T}$, and we show that $\mathcal{T} \not\models T$.

We call a set $\mathcal{S}$ of $\mathcal{ALC}$-Trees inconsistent, if we have $\mathcal{S} \vdash \neg \top$. Due to the deduction theorem, $\mathcal{S}$ is inconsistent if and only if there are $S_1, \ldots, S_n \in \mathcal{S}$ with $\vdash \neg (S_1 \sqcap \ldots \sqcap S_n)$.

We assume that there is no proof of $T$ from $\mathcal{T}$. Then $\mathcal{T} \cup \{\neg T\}$ is consistent.

For a set $\mathcal{S}$ of $\mathcal{ALC}$-Trees and a role name $R$, let $\mathcal{S}^R := \{S \mid R \neg S \in \mathcal{S}\}$. We first prove the following property:

If $\mathcal{S}$ is consistent, where $R \neg S \in \mathcal{S}$, then $\mathcal{S}^R \cup \neg \mathcal{S}$ is also consistent. (3)

It is easier to show the contraposition of Eqn. (3), so we assume that $\mathcal{S}^R \cup \neg \mathcal{S}$ is not consistent. Then there exists finitely many elements $S_1, \ldots, S_n$ of $\mathcal{S}^R$ such that there is a proof of $\neg (S_1 \sqcap \ldots \sqcap S_n \sqcap \neg S)$ (from the empty set). Now Lemma 9.3 yields that we have a proof of $\neg R \neg (S_1 \sqcap \ldots \sqcap S_n \sqcap \neg S)$ as well. We proceed as follows:

36
So \( \mathfrak{S} \) is also \( \mathcal{T} \)-inconsistent, thus Eqn. (3) holds.

Now, using a standard argument based on the axiom of choice, every consistent set can be extended to a maximal consistent set, i.e., a consistent set which cannot be properly extended to another consistent set.

Next, for a maximal consistent set \( \mathfrak{S}_m \), we have

\[
S \in \mathfrak{S}_m \iff \neg S \notin \mathfrak{S}_m \quad (4)
\]

\[
S \sqcap S' \in \mathfrak{S}_m \iff S \in \mathfrak{S}_m \text{ and } S' \in \mathfrak{S}_m \quad (5)
\]

for arbitrary \( \mathcal{ALC} \)-Trees \( S, S' \). We only prove Eqn. (4), the proof of Eqn. (5) is done similarly.

Due to \( S \sqcap \neg S \not\vdash \neg \top \) and \( \neg \top \not\vdash \neg \top \), and as we can infer any tree from \( \neg \top \), we cannot have both \( S \in \mathfrak{S}_m \) and \( \neg S \in \mathfrak{S}_m \). On the other hand, let us suppose we have \( \neg S \notin \mathfrak{S}_m \). Then we have \( \mathfrak{S}_m \not\vdash \neg S \). Assume \( \mathfrak{S}_m \cup \{S\} \) is inconsistent. Then we have \( S_1, \ldots, S_n \in \mathfrak{S}_m \) with \( \vdash \neg (S_1 \sqcap \ldots \sqcap S_n \sqcap S) \). Thm. 9.3 yields \( S_1 \sqcap \ldots \sqcap S_n \not\vdash \neg S \), i.e., we have \( \mathfrak{S}_m \not\vdash \neg S \). This contradicts \( \mathfrak{S}_m \not\vdash \neg S \). So if \( \neg S \notin \mathfrak{S}_m \), then \( \mathfrak{S}_m \cup \{S\} \) is consistent, thus \( S \in \mathfrak{S}_m \) due to the maximality of \( \mathfrak{S}_m \). Hence Equation (4) is proven.

Now we construct a model \((\Delta^T, \mathcal{I})\) as follows. We set

\[
\Delta^T := \{ \mathfrak{S}_m \mid \mathfrak{S}_m \text{ is maximal consistent} \}
\]

\[
\mathcal{I}(A) := \{ \mathfrak{S}_m \mid T_A \in \mathfrak{S}_m \}
\]

\[
\mathcal{I}(R) := \{ (\mathfrak{S}_m, T_m) \mid \mathfrak{S}_m \sqsubseteq T_m \}\}
\]

We prove by induction over the construction of \( \mathcal{ALC} \)-trees that for \( S \in \mathcal{ALC}^{\mathcal{T}\text{ree}} \) and \( \mathfrak{S}_m \in \Delta^T \), we have

\[
\mathfrak{S}_m \in \mathcal{I}(S) \iff S \in \mathfrak{S}_m \quad (6)
\]

For a concept name \( A \), Eqn. (6) holds by the definition of the model. For a tree \( \neg S \), we have \( \mathfrak{S}_m \in \mathcal{I}(\neg S) \iff \mathfrak{S}_m \notin \mathcal{I}(S) \iff S \notin \mathfrak{S}_m \iff S \notin \mathfrak{S}_m \). For a tree \( S \sqcap S' \), Eqn. (6) is similarly proven using Eqn. (5). It remains to consider role names.

Let \( R \in R \). We first prove Eqn. (6) for \( \mathcal{ALC} \)-trees \( S' := \neg R \cdot S \) (instead of \( S' := R S \)). Due to our induction, we can assume that Eqn. (6) is proven for all \( \mathcal{ALC} \)-trees which have less occurrences of \( R \) than \( S' \). Def. 7.1 yields

\[
\mathfrak{S}_m \in \mathcal{I}(\neg R \cdot S) \iff \text{ for all } \mathfrak{T}_m \text{ with } (\mathfrak{S}_m, \mathfrak{T}_m) \in \mathcal{I}(R) \text{ we have } \mathfrak{T}_m \in \mathcal{I}(S) \quad (7)
\]
Suppose first we have $S' \in G_m$. If $T_m \in \Delta^I$ is arbitrary with $(G_m, T_m) \in I(R)$, then $S \in T_m$ by the definition of the model. The induction hypothesis yields $S_m \in I(S)$, so Eqn. (7) yields $S_m \in I(S')$. Next suppose $S' \notin G_m$. Eqn. (4) yields $R-S \in G_m$. Let $T' := G^R_m \cup \{\neg S\}$. Then $T'$ is consistent due to Eqn. (3). Let $T_m \supseteq T'$ be a maximal consistent set. Then $T_m \in \Delta^I$, and $(G_m, T_m) \in I(R)$. Since $\neg S \in T' \subseteq T_m$, we have $S \notin T_m$. The induction hypothesis yields $S_m \notin I(S)$, thus $G_m \notin I(S')$ due to Eqn. (7).

Hence Eqn. (6) is proven for $S' = \neg R \neg S$.

To finish the proof of Eqn. (6), let us finally observe that for $ALC$-trees of the form $RS$, we have $G_m \in I(RS) \iff G_m \notin I(\neg R \neg S) \iff \neg R \neg S \notin G_m \iff RS \in G_m$. So Eqn. (6) is proven.

Now let $G_m \in \Delta^I$. We have: $G_m \in \bigcap_{S \in T} I(S) \iff G_m \in I(S)$ for all $S \in T$. Eqn. (6) yields $\bigcap_{S \in T} I(S) = \{G_m \in \Delta^I \mid G_m \supseteq T\}$. As $T \supseteq \{\neg T\}$ is consistent, there exist a maximal consistent set $T_m \supseteq T \cup \{\neg T\}$. On the one hand, we now have $T_m \in \bigcap_{S \in T} I(S)$. On the other hand, we have $T \notin T_m$, thus $T_m \notin I(T)$ by Eqn. (6). So we obtain $\bigcap_{S \in T} I(S) \subset I(T)$, which means $T \not\models T$.

10. Conclusion and Further Research

This paper has discussed different diagrammatic reasoning systems – spider and constraint diagrams on the one hand, existential and conceptual graphs on the other – as candidates for a diagrammatic version of DL. It has been argued that spider and constraint diagrams are not as well suited to a diagrammatic version of DLs as conceptual graphs and existential graphs. More specifically, a version of conceptual graphs including negation and free variables, and relation graphs – existential graphs with free variables – are suited as a diagrammatic version of DLs. It has been argued that spider and constraint diagrams are not as well suited to a diagrammatic version of DLs as conceptual graphs and existential graphs. More specifically, a version of conceptual graphs including negation and free variables, and relation graphs – existential graphs with free variables – are suited as a diagrammatic version of the smallest propositionally closed DL $ALC$.

We claim further that relation graphs are slightly easier to comprehend. We then elaborated a fragment $RG^{ACC}$ of relation graphs that correspond to the $ALC$, and provided a diagrammatic, sound and complete calculus for $RG^{ACC}$, based on Peirce’s rules for existential graphs.

The results presented in this paper provide promising steps toward further investigating RGs as diagrammatic versions of DLs. There are several open problems that have to be addressed by extension research. First, our approach could be extended to other constructs of DL. For example, to investigate how the intersection of roles, inverse of roles, and unary (one-of) concepts can be built into the diagrammatic reasoning frameworks (this would obtain a strict extension of the results of [17,3]).

From the syntactical and semantical point of view this will not yield any more significant problems than we have already encountered. We can extend the graphs of $RG^{ACC}$ such that we allow objects at the end of heavily drawn lines (adding constants and function names to RGs is elaborated in [26]) or cycles, where only two branching points are involved. For example, the concept description

$$\text{MAN} \sqcap \exists(\text{hasFriend} \sqcap \text{hasColleague}).\{\text{John}\}$$

In [3] the authors allow the intersection of roles and write that the corresponding SCGs are trees, where they say that “a SCG $t$ is called a tree if $t$ contains no cycles of length greater than 2.”
describing ‘some man having John as friend and colleague’, could be depicted as follows:

Similarly, from a syntactic and semantic point of view, allowing concepts of the form \{o_1, \ldots, o_n\} is not problematic either.

More importantly, the approach we present can be extended to number restriction. In this case we encounter the \(SH\)-family of DLs and in so doing have the capacity to provide diagrammatic representations of the underlying DLs used in OWL. From a representational point of view a commonly used notation for number restrictions could be used: such as the (min, max)-notation used widely in both entity-relationship-diagrams (EER) and UML. For example, the concept

\[
\text{Man} \sqcap \geq 10 \text{hasFriend}\cdot \text{Man} \sqcap \leq 5 \text{hasColleague}\cdot \text{Woman}
\]

of men who have at least 10 men as friends and not more than 5 female colleagues, could be depicted as follows:

Further the syntactical restrictions of DL yield corresponding syntactical restrictions in \(RG^{ALC}\). It is possible to benefit from these restrictions by adding some “syntactic sugar” to the graphs, which is not possible for the general class of RGs. This applies for example to the universal quantifier of DL. Below is the general form of the value restriction of DL, its translation with \(\Psi\), and an modified version without cuts, where the heavily drawn line on the right is now marked with an universal quantifier.

\[
\forall R.C \quad \exists R.\Psi(C)
\]

Similar considerations are imposed by allowing an explicit notation for disjunction. This would allow cut-free representations for all formulas of \(ALC\). For example, Fig. 20 provides an \(ALC\)-concept and two possible improvements of the corresponding Peirce diagrams.

\[
\text{Man} \sqcap \exists \text{hasChild}\cdot \text{Female} \sqcap \exists \text{hasChild}\cdot \text{Male} \sqcap \forall \text{hasChild}(\text{Rich} \sqcup \text{Happy})
\]

Fig. 20. An \(ALC\)-concept descriptions and two RGs with abbreviation for \(\forall\) and disjunction

Finally, a main aspect of research in DLs focuses on complexity. At first glance, the complexity issues should be independent from the notation but there are some promising
results for Peirce’s existential alpha graphs (corresponding to propositional logic) in terms of computability that show that Peirce’s rules have some nice complexity results. In Dau [25], it is shown that the erasure-rule, which is the only rule for Peirce’s existential alpha graphs that does not satisfy the finite subformula property, is admissible, and in the remaining calculus, Statman’s formulas can still be proven in polynomial time.

Roughly speaking, Peirce’s calculus without the erasure rule has the nice computational properties of sequent calculi without the cut-rule\(^{17}\), while it is as fast as sequent calculi that includes the cut-rule\(^{18}\). It can be investigated whether similar results can be obtained for the calculus presented in this paper.

For the moment, the first key issue addressed in his paper is the question of what (if any) existing and well-known diagrammatic reasoning frameworks can be suitably adapted to DL. We are confident that existential and conceptual graphs can be used as a diagrammatic framework for DL and have presented our reasons for this conclusion. The second key issue was to elaborate a diagrammatic reasoning system for ALC based on existential graphs, including a diagrammatic, sound and complete calculus. In the long run, we intend to develop major variants of DLs as mathematically precise diagrammatic reasoning systems. The intention is to render DL more user-friendly. Such systems need to be evaluated against the traditional textual form of DL in order to measure any improvement to the readability of the DL in its diagrammatic form. Such evaluations involve usability experiments and substantial empirical testing and are beyond the scope of this paper which restricts itself to the technical feasibility of the candidate diagrammatic reasoning frameworks to DL.

11. Appendix

The following abbreviations are used:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL</td>
<td>description logic</td>
</tr>
<tr>
<td>SD</td>
<td>spider diagram</td>
</tr>
<tr>
<td>CG</td>
<td>conceptual graph</td>
</tr>
<tr>
<td>CGwC</td>
<td>concept(ual) graph with cuts</td>
</tr>
<tr>
<td>RG</td>
<td>relation graph</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>FOL</td>
<td>first order logic</td>
</tr>
<tr>
<td>CD</td>
<td>constraint diagram</td>
</tr>
<tr>
<td>SCG</td>
<td>simple conceptual graph</td>
</tr>
<tr>
<td>EG</td>
<td>existential graph</td>
</tr>
<tr>
<td>OWL</td>
<td>Ontology Web Language</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
</tbody>
</table>

12. Acknowledgements

We like to thank the reviewers for their helpful remarks and also Andrew Fish for his valuable comments on using CDs for ALC.

\(^{17}\)In sequent calculi the cut-rule does not satisfy the finite subformula property, which is one of the reasons that Gentzen’s famous cut-elimination-theorem is so important.

\(^{18}\)Statman’s formulas need proofs in exponential time in the cut-free sequent calculi.
References


[26] Frithjof Dau. Some notes on proofs with alpha graphs. In Øhrstrøm et al. [54], pages 172–188.


[41] Joachim Hereth Correia and Reinhard Pöschel. The teridentity and peircean algebraic logic. In Øhrstrøm et al. [54].


[69] John F. Sowa. Semantic foundations of contexts. This paper is a revised merger of [72] and [74]. Available at: http://www.jfsowa.com/ontology/contexts.htm.
[72] John F. Sowa. Syntax, semantics, and pragmatics of contexts. In Ellis et al. [28], pages 1–15. See also [69].