Upward Refinement for Conceptual Blending in Description Logic — An ASP-based Approach and Case Study in $\mathcal{EL}^{++}$—

Roberto Confalonieri, Marco Schorlemmer, Enric Plaza
Artificial Intelligence Research Institute, IIIA-CSIC, Spain
{confalonieri,marco,enric}@iiia.csic.es

Manfred Eppe
International Computer Science Institute, Berkeley, USA
eppe@icsi.berkeley.edu

Oliver Kutz, Rafael Peñaloza
Free University of Bozen-Bolzano, Italy
{oliver.kutz,rafael.penaloza}@unibz.it

Abstract

Conceptual blending is understood to be a process that serves a variety of cognitive purposes, including creativity, and has been highly influential in cognitive linguistics. In this line of thinking, human creativity is modeled as a blending process that takes different mental spaces as input and combines them into a new mental space, called a *blend*. According to this form of *combinatorial creativity*, a blend is constructed by taking the existing commonalities among the input mental spaces—known as the generic space—into account, and by projecting their structure in a selective way. Since input spaces for interesting blends are often initially incompatible, a generalisation step is needed before they can be blended. In this paper, we apply this idea to blend input spaces specified in the description logic $\mathcal{EL}^{++}$ and propose an upward refinement operator for generalising $\mathcal{EL}^{++}$ concepts. We show how the generalisation operator is translated to Answer Set Programming (ASP) in order to implement a search process that finds possible generalisations of input concepts. We exemplify our approach in the domain of computer icons.

Introduction

The upward refinement—or generalisation—of concepts plays a crucial role in creative processes for analogical reasoning and concept invention, in particular *conceptual blending* (Fauconnier and Turner 2002). In blending, one combines two input concepts to invent a new one. In general, however, this cannot be done because the combination of two concepts may generate an unsatisfiable one due to contradiction. However, by slightly generalising input concepts we might be able to find a novel and useful combination of both. For instance, a 'red French sedan' and a 'blue German minivan' can be blended to a 'red German sedan' by generalising the first concept to a 'red European sedan' and the second one to a 'coloured German car'. The least general generalisation of our input concepts—a 'coloured European car'—serves as an upper bound of the generalisation space to be explored, and, in a certain sense, plays the role of the so call *generic space* in conceptual blending, which states the shared structure of both concepts.

This paper addresses the formalisation of such a generalisation process and tackles the following question: *How can we define and implement a generalisation operator for description logics (DLs) and what are its desired or necessary properties in order to use it in the blending process?*

We focus on the particular case of the description logic $\mathcal{EL}^{++}$ (Baader, Brandt, and Lutz 2005; Baader, Brandt, and Lutz 2008). This is an excellent starting point for our investigation for several reasons. First, $\mathcal{EL}^{++}$ is the underpinning logic of the OWL 2 EL Profile, a recommendation of the W3C. Second, $\mathcal{EL}^{++}$ offers a good tradeoff between expressivity and efficiency of reasoning. Indeed, the $\mathcal{EL}^{++}$ syntax and axioms are considered to be sufficiently expressive to model large real-world ontologies. Finally, subsumption in an $\mathcal{EL}^{++}$ TBox is computable in polynomial time (Baader, Brandt, and Lutz 2005).

The generalisation of $\mathcal{EL}^{++}$ concepts has been studied both in the DLs and in the Inductive Logic Programming (ILP) literature, although from different perspectives. Whilst approaches in DL focus on formalising the computation of a least general generalisation (LGG) (also known as least common subsumer) among different concepts as a non-standard reasoning task (Baader 2005; Baader, Sertkaya, and Turhan 2007; Turhan and Zarrieß 2013), approaches in ILP are concerned on learning DL descriptions from examples (Lehmann and Hitzler 2010). In both cases, however, finding a LGG is a challenging task. Its computability depends on the type of DL adopted and on the assumptions made over the structure of concept definitions.

Our work relates to these approaches, but our main motivation for generalising DL concepts is intrinsically different. Although we do need to be aware of what concepts share in order to blend them, it is not necessary (though desirable) to find a *generic space* that is also a LGG. A sufficiently specific common subsumer will suffice. With this objective in mind, we propose an upward refinement operator for generalising $\mathcal{EL}^{++}$ concepts which is inductively defined over the structure of concept descriptions. We discuss some of the properties typically used to characterise refinement operators; namely, finiteness, properness and completeness (van der Laag and Nienhuys-Cheng 1998). Briefly, a refinement operator is said to be finite when it generates a finite set of refinements; proper, when its refinements are not equivalent to the original concept, and complete, when it produces all possible refinements of a given concept.

Particularly, the generalisations produced by our opera-

concept description	interpretation
A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
$\top$	$\Delta^{\mathcal{I}}$
$\bot$	$\emptyset$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}}.$ $(x,y) \in \wedge y \in C^{\mathcal{I}}\}$
axiom	**satisfaction**
$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
$r_1 \circ \cdots \circ r_n \sqsubseteq r$	$r_1^{\mathcal{I}}\ ;\ \cdots\ ;\ r_n^{\mathcal{I}} \subseteq r^{\mathcal{I}}$
$domain(r) \sqsubseteq C$	$r^{\mathcal{I}} \subseteq C^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
$range(r) \sqsubseteq C$	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C^{\mathcal{I}}$

Table 1: Syntax and semantics of some $\mathcal{EL}^{++}$ contructors and axioms. (Note:';' is the usual composition operator in relation algebra.)

tor are finite, but they can be sometimes equivalent to the concept being generalised (thus, the operator is not proper), and they are not all the possible ones (thus, the operator is not complete). As we shall discuss, we sacrifice completeness for finiteness (since we do not need to compute a LGG, strictly speaking), but we need the successive applications of the operator to always terminate. We point out, however, how the properness property can be achieved.

We show how part of the upward refinement operator is implemented in Answer Set Programming (ASP) (Gelfond and Kahl 2014) and how we employ the search capabilities of ASP to find a generic space among two $\mathcal{EL}^{++}$ input concepts. The ASP search is part of an amalgamation process (Ontañón and Plaza 2010) that models conceptual blending. Throughout the paper, we use an example in the domain of computer icon design.

Background and Running Example

The Description Logic $\mathcal{EL}^{++}$

In DLs, concept and role descriptions are defined inductively by means of concept and role constructors over a finite set N_C of concept names, a finite set N_r of role names, and (possibly) a finite set N_I of individual names. As is common practice, we shall write A, B for concept names, C, D for concept descriptions, r, s for role names, and a, b, for individual names.

The semantics of concept and role descriptions is defined in terms of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty domain and $\cdot^{\mathcal{I}}$ is an interpretation function assigning a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ to each concept name $A \in N_C$, a set $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each role name $r \in N_r$, and an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for each individual name $a \in N_I$, which is extended to general concept and role descriptions. Table 1 shows the constructors of the description logic $\mathcal{EL}^{++}$ that are relevant for this paper, together with their interpretation. For a complete presentation of $\mathcal{EL}^{++}$ we refer to (Baader, Brandt, and Lutz 2005; Baader, Brandt, and Lutz 2008).

A knowledge base usually consists of a finite set $\mathcal{T}$ of terminological axioms, called TBox, which contains intensional knowledge defining the main notions relevant to the domain of discourse; and a finite set $\mathcal{A}$ of assertional axioms, called ABox, which contains extensional knowledge about individual objects of the domain. In this paper, we shall focus only on terminological axioms of the form $C \sqsubseteq D$, i.e. general concept inclusions (GCIs), and $r_1 \circ \cdots \circ r_n \sqsubseteq r$, i.e. role inclusions (RIs), as well as axioms specifying domain and range restrictions for roles. Table 1 shows the form of these axioms, together with the condition for these to be satisfied by an interpretation $\mathcal{I}$. By $\mathcal{L}(\mathcal{T})$ we refer to the set of all $\mathcal{EL}^{++}$ concept descriptions we can form with the concept and role names occurring in $\mathcal{T}$.

RIs allow one to specify role hierarchies ($r \sqsubseteq s$) and role transitivity ($r \circ r \sqsubseteq r$). The bottom concept $\bot$, in combination with GCIs, allows one to express disjointness of concept descriptions, e.g. $C \sqcap D \sqsubseteq \bot$ tells that C and D are disjoint. An interpretation $\mathcal{I}$ is a model of a TBox $\mathcal{T}$ iff it satisfies all axioms in $\mathcal{T}$. The basic reasoning task in $\mathcal{EL}^{++}$ is subsumption. Given a TBox $\mathcal{T}$ and two concept descriptions C and D, we say that C is (strictly) subsumed by D w.r.t. $\mathcal{T}$, denoted as $C \sqsubseteq_{\mathcal{T}} D$ ($C \sqsubset_{\mathcal{T}} D$), iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ ($C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and $C^{\mathcal{I}} \neq D^{\mathcal{I}}$) for every model $\mathcal{I}$ of $\mathcal{T}$. Analogously, given two roles $r, s \in N_r$, we say that r is (strictly) subsumed by s w.r.t. $\mathcal{T}$, denoted as $r \sqsubseteq_{\mathcal{T}} s$ ($r \sqsubset_{\mathcal{T}} s$), iff $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ ($r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ and $r^{\mathcal{I}} \neq s^{\mathcal{I}}$) for every model $\mathcal{I}$ of $\mathcal{T}$. Finally, an equivalence axiom $C \equiv D$ is just an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$.

$\mathcal{EL}^{++}$ Running Example

To exemplify our approach, we take the domain of computer icons into account. We consider computer icons as combinations of signs (e.g., Document, MagnifyingGlass, HardDisk, Pen, etc.) (Confalonieri et al. 2015). Signs are related by qualitative spatial relations such as *above*, *behind*, etc.

In the ontology below, concept names are capitalised (e.g. Sign) and role names are written in lower-case (e.g. hasSign). We assume that a TBox $\mathcal{T}$ consists of two parts: one part that contains the background knowledge about the icon domain $\mathcal{T}_{bk}$, and another part that contains the domain knowledge about icon definitions $\mathcal{T}_{dk}$.

Generally, an icon is related to different signs by means of the hasSign role.

α_{bk_1}: Icon $\sqsubseteq$ Thing
α_{bk_2}: Sign $\sqsubseteq$ Thing
α_{bk_3}: Document $\sqsubseteq$ Sign
α_{bk_4}: HardDisk $\sqsubseteq$ Sign
α_{bk_5}: MagnifyingGlass $\sqsubseteq$ Sign
α_{bk_6}: Pen $\sqsubseteq$ Sign
α_{bk_7}: $domain$(hasSign) $\sqsubseteq$ Icon
α_{bk_8}: $range$(hasSign) $\sqsubseteq$ Sign
α_{bk_9}: Document $\sqcap$ HardDisk $\sqsubseteq \bot$
...
$\alpha_{\mathsf{bk}_{15}}$: MagnifyingGlass $\sqcap$ Pen $\sqsubseteq \bot$

Sign concepts are disjoint (α_{bk_9}-$\alpha_{\mathsf{bk}_{15}}$) and they are related by spatial relationships isAbove, isBehind, isLeft and isRight that are modelled as roles. These roles are subsumed by a more generic role that is isInSpatialRelation

whose domain and range is the Sign concept. Spatial relationships are transitive as expressed by axioms $\alpha_{\mathsf{bk}_{22}}$-$\alpha_{\mathsf{bk}_{26}}$.

$\alpha_{\mathsf{bk}_{16}}$: $domain(\mathsf{isInSpatialRelation}) \sqsubseteq \mathsf{Sign}$
$\alpha_{\mathsf{bk}_{17}}$: $range(\mathsf{isInSpatialRelation}) \sqsubseteq \mathsf{Sign}$
$\alpha_{\mathsf{bk}_{18}}$: $\mathsf{isAbove} \sqsubseteq \mathsf{isInSpatialRelation}$
$\alpha_{\mathsf{bk}_{19}}$: $\mathsf{isBehind} \sqsubseteq \mathsf{isInSpatialRelation}$
$\alpha_{\mathsf{bk}_{20}}$: $\mathsf{isLeft} \sqsubseteq \mathsf{isInSpatialRelation}$
$\alpha_{\mathsf{bk}_{21}}$: $\mathsf{isRight} \sqsubseteq \mathsf{isInSpatialRelation}$
$\alpha_{\mathsf{bk}_{22}}$: $\mathsf{isAbove} \circ \mathsf{isAbove} \sqsubseteq \mathsf{isAbove}$
...
$\alpha_{\mathsf{bk}_{26}}$: $\mathsf{isInSpatialRelation} \circ \mathsf{isInSpatialRelation} \sqsubseteq \mathsf{isInSpatialRelation}$

Given the axioms above, we can describe some icons in the domain knowledge of a TBox.

Example 1. SearchHardDisk *is an icon that consists of two signs* MagnifyingGlass *and* HardDisk*, where the* MagnifyingGlass *sign is above the* HardDisk *sign. Another icon is the* EditDocument *icon, where the* Pen *sign is above the* Document *sign. Both icons are shown in Figure 1:*

α_{dk_1}: $\mathsf{SearchHardDisk} \equiv \mathsf{Icon} \sqcap \exists \mathsf{hasSign.HardDisk} \sqcap \exists \mathsf{hasSign.}(\mathsf{MagnifyingGlass} \sqcap \exists \mathsf{isAbove.HardDisk})$
α_{dk_2}: $\mathsf{EditDocument} \equiv \mathsf{Icon} \sqcap \exists \mathsf{hasSign.Document} \sqcap \exists \mathsf{hasSign.}(\mathsf{Pen} \sqcap \exists \mathsf{isAbove.Document})$

In the paper, we will show how we can create new $\mathcal{EL}^{++}$ concepts by blending existing concepts in the domain knowledge of a TBox. Specifically, we will see how to generate the following concept representing a new blended icon:

$$\mathsf{Icon} \sqcap \exists \mathsf{hasSign.Document} \sqcap \exists \mathsf{hasSign.}(\mathsf{MagnifyingGlass} \sqcap \exists \mathsf{isAbove.Document})$$

Intuitively, given two input concepts, a new concept is created by taking the commonalities and some specific parts of the input concepts into account (Figure 1). For instance, both SearchHardDisk and EditDocument are icons that consist of two signs with one sign above the other one (the generic space). Then, if we keep the MagnifyingGlass sign from SearchHardDisk and the Document sign from EditDocument, and we 'relax' the HardDisk and Pen signs, we can blend the generalised input concepts of SearchHardDisk and EditDocument into a new concept representing a preview-document icon.[1]

The process of conceptual blending is characterised in terms of amalgamation (Ontañón and Plaza 2010), an approach that has its root in case-based reasoning and focuses on the problem of combining solutions coming from multiple cases. According to this approach, input concepts are generalised until a generic space is found, and pairs of generalised versions of the input concepts are 'combined' to create blends.

[1]Of course, there are some combinations of generalised input concepts that are not interesting. For instance, a concept such as Icon ⊓ ∃hasSign.MagnifyingGlass ⊓ ∃hasSign.(Pen ⊓ ∃isAbove.MagnifyingGlass) should be discarded. This issue relates to blend evaluation and it is not addressed in this paper. We refer the interested reader to (Confalonieri et al. 2015) where a discussion about the use of argumentation to evaluate conceptual blends is presented.

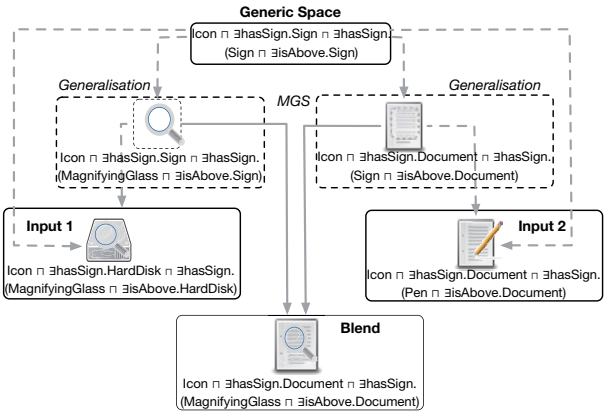

Figure 1: Blending the SearchHardDisk and EditDocument icon concepts into a new concept representing a preview-document icon.

Computational concept blending by amalgamation

Formally, the notion of amalgam can be defined in any representation language $\mathcal{L}$ for which a subsumption relation between formulas (or descriptions) of $\mathcal{L}$ can be defined, and therefore also in $\mathcal{L}(\mathcal{T})$ with the subsumption relation $\sqsubseteq_{\mathcal{T}}$ for a given $\mathcal{EL}^{++}$ Tbox $\mathcal{T}$.

Given two descriptions $C_1, C_2 \in \mathcal{L}(\mathcal{T})$, a *most general specialisation* (MGS) is a description C_{mgs} such that $C_{mgs} \sqsubseteq_{\mathcal{T}} C_1$ and $C_{mgs} \sqsubseteq_{\mathcal{T}} C_2$ and for any other description D satisfying these properties, $D \sqsubseteq_{\mathcal{T}} C_{mgs}$. A *least general generalisation* (LGG) is a description C_{lgg} such that $C_1 \sqsubseteq_{\mathcal{T}} C_{lgg}$ and $C_2 \sqsubseteq_{\mathcal{T}} C_{lgg}$ and for any other description D satisfying these properties, $C_{lgg} \sqsubseteq_{\mathcal{T}} D$.

Intuitively, a MGS is a description that has all the information in both the original descriptions C_1 and C_2, while a LGG contains that which is common to them.

An *amalgam* of two descriptions is a new description that contains *parts from these original descriptions*. For instance, an amalgam of 'a red French sedan' and 'a blue German minivan' is 'a red German sedan;' clearly there are always multiple possibilities for amalgams, like 'a blue French minivan'.

For the purposes of this paper we can define an amalgam of two descriptions as follows:

Definition 1 (Amalgam)**.** *Let $\mathcal{T}$ be a Tbox in $\mathcal{EL}^{++}$. A description $C_{am} \in \mathcal{L}(\mathcal{T})$ is an* amalgam *of two descriptions C_1 and C_2 (with LGG C_{lgg}) if there exist two descriptions C'_1 and C'_2 such that:*

1. $C_1 \sqsubseteq_{\mathcal{T}} C'_1 \sqsubseteq_{\mathcal{T}} C_{lgg}$,
2. $C_2 \sqsubseteq_{\mathcal{T}} C'_2 \sqsubseteq_{\mathcal{T}} C_{lgg}$, *and*
3. C_{am} *is a MGS of C'_1 and C'_2*

In the next section, we define an upward refinement operator that allows us to find generalisations of $\mathcal{EL}^{++}$ concept descriptions needed for computing the amalgams as described above, although we may generalise concepts C_1 and C_2 beyond the LGG C_{lgg}. We do this to guarantee termination, as we shall explain below.

Refinement Operators

Refinement operators are a well known notion in Inductive Logic Programming where they are used to structure a search process for learning concepts from examples. In this setting, two types of refinement operators exist: specialisation (or downward) refinement operators and generalisation (or upward) refinement operators. While the former constructs specialisations of hypotheses, the latter contructs generalisations.[2]

Generally speaking, refinement operators are defined over quasi-ordered sets. A quasi-ordered set is a pair $\langle \mathcal{S}, \preceq \rangle$ where $\mathcal{S}$ is a set and $\preceq$ is a binary relation among elements of $\mathcal{S}$ that is reflexive ($a \preceq a$) and transitive (if $a \preceq b$ and $b \preceq c$ then $a \preceq c$). If $a \preceq b$, we say that b is more general than a, and if also $b \preceq a$ we say that a and b are equivalent. A generalisation refinement operator is defined as follows.

Definition 2. *A generalisation refinement operator γ over a quasi-ordered set $\langle \mathcal{S}, \preceq \rangle$ is a function such that $\forall a \in \mathcal{S}$: $\gamma(a) \subseteq \{b \in \mathcal{S} \mid a \preceq b\}$.*

A refinement operator γ can be classified according to some desirable properties (van der Laag and Nienhuys-Cheng 1998). We say that γ is:

- *locally finite* if the number of generalisations generated for any given element by the operator is finite, that is, $\forall a \in \mathcal{S} : \gamma(a)$ is finite;
- *proper* if an element is not equivalent to any of its generalisations, i.e., $\forall a, b \in \mathcal{S}$, if $b \in \gamma(a)$, then a and b are not equivalent;
- *complete* if there are no generalisations which are not generated by the operator, i.e., $\forall a, b \in \mathcal{S}$ it holds that if $a \preceq b$, then $b \in \gamma^*(a)$ (where $\gamma^*(a)$ denotes the set of all elements which can be reached from a by means of γ in a finite number of steps).

When a refinement operator is locally finite, proper, and complete it is said to be *ideal*.

An ideal specialisation refinement operator for $\mathcal{EL}$ has been explored in (Lehmann and Haase 2010). In what follows, we will define a generalisation refinement operator for $\mathcal{EL}^{++}$ and discuss its properties.

A Generalisation Refinement Operator for $\mathcal{EL}^{++}$

In any description logic the set of (complex) concept descriptions are ordered under the subsumption relation forming a quasi-ordered set. For $\mathcal{EL}^{++}$ in particular they form a bounded meet-semilattice with conjunction as meet operation and $\top$ as greatest element as well as $\bot$ as least element.

In order to define a generalisation refinement operator for $\mathcal{EL}^{++}$ we will need some auxiliary definitions.

Definition 3. *Let $\mathcal{T}$ be a TBox in $\mathcal{EL}^{++}$. The set of* subconcepts *of $\mathcal{T}$ is given as*

$$\mathsf{sub}(\mathcal{T}) = \bigcup_{C \sqsubseteq D \in \mathcal{T}} \mathsf{sub}(C) \cup \mathsf{sub}(D)$$

where sub *is inductively defined over the structure of concept descriptions as follows:*

$$\begin{aligned} \mathsf{sub}(A) &= \{A\} \\ \mathsf{sub}(\bot) &= \{\bot\} \\ \mathsf{sub}(\top) &= \{\top\} \\ \mathsf{sub}(C \sqcap D) &= \{C \sqcap D\} \cup \mathsf{sub}(C) \cup \mathsf{sub}(D) \\ \mathsf{sub}(\exists r.C) &= \{\exists r.C\} \cup \mathsf{sub}(C) \end{aligned}$$

Based on $\mathsf{sub}(\mathcal{T})$, we define the upward cover set of atomic concepts and roles. $\mathsf{sub}(\mathcal{T})$ guarantees the following upward cover set to be finite.

Definition 4. *Let $\mathcal{T}$ be a Tbox in $\mathcal{EL}^{++}$ (with concept names from N_C). The set of* upward covers[3] *of an atomic concept $A \in N_C \cup \{\top, \bot\}$ and of a role $r \in N_r$ with respect to $\mathcal{T}$ is given as:*

$$\begin{aligned} \mathsf{UpCov}(A) = \{&C \in \mathsf{sub}(\mathcal{T}) \mid A \sqsubset_{\mathcal{T}} C \\ &\text{and there is no } C' \in \mathsf{sub}(\mathcal{T}) \\ &\text{such that } A \sqsubset_{\mathcal{T}} C' \sqsubset_{\mathcal{T}} C\} \end{aligned} \quad (1)$$

$$\begin{aligned} \mathsf{UpCov}(r) = \{&s \in N_r \mid r \sqsubset_{\mathcal{T}} s \\ &\text{and there is no } s' \in N_r \\ &\text{such that } r \sqsubset_{\mathcal{T}} s' \sqsubset_{\mathcal{T}} s\} \end{aligned} \quad (2)$$

We can now define our generalisation refinement operator for $\mathcal{EL}^{++}$ as follows:

Definition 5. *Let $\mathcal{T}$ be a TBox in $\mathcal{EL}^{++}$. We define the* generalisation operator γ *inductively over the structure of concept descriptions as shown in Figure 2.*

Given a refinement operator γ, $\mathcal{EL}^{++}$ concepts are related by refinement paths as follows:

Definition 6. *A finite sequence $C_1, \ldots, C_n$ of concepts is a* concept refinement path $C_1 \xrightarrow{\gamma} C_n$ *from C_1 to C_n of a generalisation operator γ if $C_2 \in \gamma(C_1), \ldots, C_n \in \gamma(C_{n-1})$. D can be reached from C by γ if there exists a refinement path from C to D. $\gamma^*(C)$ denotes the set of all concepts that can be reached from C by means of γ. Sometimes we write $C \rightsquigarrow_\gamma D$ instead of $D \in \gamma(C)$.*

That γ is indeed a generalisation refinement operator as defined in Definition 2 can be proven by applying structural induction on $\mathcal{EL}^{++}$ concepts to show that $C \rightsquigarrow_\gamma D$ implies $C \sqsubseteq D$, in a similar way as in the proof of Proposition 11 from (Lehmann and Hitzler 2010).

As far as the properties of γ are concerned, we can observe the following. Our definition of UpCov for concepts and roles only considers the set of subconcepts present in a Tbox $\mathcal{T}$. On the one hand, this guarantees that γ is finite, since at each generalisation step, the set of possible generalisations is finite. On the other hand, however, this implies

[2] A deeper analysis of refinement operators can be found in (van der Laag and Nienhuys-Cheng 1998).

[3] Notice that the set of upward covers we define only takes into accout subconcepts already present in the TBox. Therefore, strictly speaking, its elements may not be covers with respect to subsumption in $\mathcal{EL}^{++}$.

$$
\begin{array}{rcl}
\gamma(A) & = & \mathsf{UpCov}(A) \\
\gamma(\top) & = & \mathsf{UpCov}(\top) = \emptyset \\
\gamma(\bot) & = & \mathsf{UpCov}(\bot) \\
\gamma(C \sqcap D) & = & \{C' \sqcap D \mid C' \in \gamma(C)\} \cup \{C \sqcap D' \mid D' \in \gamma(D)\} \\
\gamma(\exists r.C) & = & \begin{cases} \gamma_r(\exists r.C) \cup \gamma_C(\exists r.C) & \text{whenever } \mathsf{UpCov}(r) \neq \emptyset \text{ or } C \neq \top \\ \{\top\} & \text{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{rcl}
\gamma_r(\exists r.C) & = & \{\exists s.C \mid s \in \mathsf{UpCov}(r)\} \\
\gamma_C(\exists r.C) & = & \{\exists r.C' \mid C' \in \gamma(C) \wedge C' \sqsubseteq \mathit{range}(r)\}
\end{array}
$$

Figure 2: A generalisation refinement operator for $\mathcal{EL}^{++}$. $\mathit{range}(r)$ is defined in Table 1.

that γ is not complete, since it cannot find all possible upward covers of a concept w.r.t. subsumption in $\mathcal{EL}^{++}$.[4]

Regarding the properness property, γ is not proper since there exist cases in which the generalisations produced by γ are equivalent to the concept being generalised. One way to guarantee the properness of γ is by rewriting the concept that we want to generalise into equivalent normal forms before and after each generalisation steps. We will investigate how normalisation can be done as future research.

For the blending, we are interested in finding a common generalisation G (generic space) between two concepts.

Example 2. *Let us consider the $\mathcal{EL}^{++}$ concepts* EditDocument *and* SearchHardDisk *defined in Example 1. It can be checked that:*

$$
\begin{array}{l}
\mathsf{EditDocument} \xrightarrow{\gamma} \mathsf{Icon} \sqcap \exists \mathsf{hasSign}.\mathsf{Sign} \sqcap \exists \mathsf{hasSign}.(\mathsf{Sign} \sqcap \exists \mathsf{isAbove}.\mathsf{Sign}) \\
\mathsf{SearchHardDisk} \xrightarrow{\gamma} \mathsf{Icon} \sqcap \exists \mathsf{hasSign}.\mathsf{Sign} \sqcap \exists \mathsf{hasSign}.(\mathsf{Sign} \sqcap \exists \mathsf{isAbove}.\mathsf{Sign})
\end{array}
$$

Therefore, $G = \mathsf{Icon} \sqcap \exists \mathsf{hasSign}.\mathsf{Sign} \sqcap \exists \mathsf{hasSign}.(\mathsf{Sign} \sqcap \exists \mathsf{isAbove}.\mathsf{Sign})$ *is a generic space for* EditDocument *and* SearchHardDisk.

It is worth to discuss that since γ is not complete, we cannot guarantee that the generic space between two $\mathcal{EL}^{++}$ concepts is a least general generalisation. Nevertheless, since the concepts and generalisations that γ finds form a bounded semilattice, we can ensure that we can always find a generic space between two concepts. We believe that not finding a least general generalisation is not a weakness of our approach since we are interested in finding a generic space that allow us to create new $\mathcal{EL}^{++}$ concepts from existing ones by conceptual blending.

Implementing Upward Refinement in ASP

We consider two TBoxes $\mathcal{T}_1$ and $\mathcal{T}_2$[5] and we assume that $\mathcal{T}_1$ and $\mathcal{T}_2$ have the same background knowledge about icon domain but different domain knowledge which, in our case, contain icon definitions.

In order to find a generic space between two icons, we generalise their definitions using the upward refinement operator in Figure 2 implemented in ASP.

The current status of the implementation considers two types of generalisation: $\gamma(A)$ that generalises an atomic concept by its upward cover and $\gamma_C(\exists r.C)$ that generalises a concept that fills the range of a role. We reserve the complete implementation of γ as future work. In this setting, the domain knowledge of a TBox $\mathcal{T}$ is generalised in a step-wise transition process. In the following, t denotes a step-counter that represents the number of modifications made to the domain knowledge part of $\mathcal{T}$. Table 2 shows the main EDB and IDB predicates used in the ASP implementation.

Modeling $\mathcal{EL}^{++}$ concepts in ASP

For each concept name $A \in N_C$ in a TBox $\mathcal{T}$, we state the facts:

$$\mathit{hasConcept}(\mathcal{T}, A, t) \tag{3a}$$
$$\mathit{isAtomicConcept}(\mathcal{T}, A, t) \tag{3b}$$
$$\mathit{isBackgroundConcept}(\mathcal{T}, A) \tag{3c}$$

For each role $r \in N_r$ in a TBox $\mathcal{T}$, with $\mathit{domain}(r) \sqsubseteq C$ and $\mathit{range}(r) \sqsubseteq D$, we state the facts:

$$\mathit{hasRole}(\mathcal{T}, r, t) \tag{4a}$$
$$\mathit{hasRoleRange}(\mathcal{T}, r, D, t) \tag{4b}$$
$$\mathit{hasRoleDomain}(\mathcal{T}, r, C, t) \tag{4c}$$
$$\mathit{isBackgroundRole}(\mathcal{T}, r) \tag{4d}$$

For each inclusion axiom $A \sqsubseteq B \in \mathcal{T}$ and A, B are atomic concepts, we state the fact:

$$\mathit{hasParentConcept}(\mathcal{T}, A, B, t) \tag{5}$$

Similarly, for each role inclusion axiom $r \sqsubseteq s \in \mathcal{T}$, we state the fact:

$$\mathit{hasParentRole}(\mathcal{T}, r, s, t) \tag{6}$$

For each inclusion axiom $A \sqsubseteq C \in \mathcal{T}$, in which A is an atomic concept and C is a complex concept, we call C the concept definition of A and denote it as $ADef$ within the following ASP facts:

$$\mathit{hasConcept}(\mathcal{T}, ADef, t) \tag{7a}$$
$$\mathit{isComplexConcept}(\mathcal{T}, ADef, t) \tag{7b}$$
$$\mathit{hasParentConcept}(\mathcal{T}, A, ADef, t) \tag{7c}$$
$$\mathit{isBackgroundConcept}(\mathcal{T}, ADef) \tag{7d}$$

For each C_i in the complex concept $C = C_1 \sqcap \ldots \sqcap C_n$, we make a case distinction and state the following facts:

[4] For instance, if $\mathcal{T}$ contains two axioms $A \sqsubseteq B$, $A \sqsubseteq C$, and we generalise A (in the domain knowledge), then $\gamma(A) = \{B, C\}$ while a possible generalisation of A w.r.t. $\sqsubseteq_{\mathcal{T}}$ is $B \sqcap C$.

[5] We assume to have two TBoxes because we want our implementation to be general, that is, to also work in a scenario in which we align concepts described using different background ontologies.

EDB predicates	Description
$spec(\mathcal{T})$	A reference to a TBox $\mathcal{T}$
$hasConcept(\mathcal{T}, C, t)$	A concept C belongs to a TBox $\mathcal{T}$ at step t
$hasParentConcept(\mathcal{T}, A, B, t)$	A concept B subsumes A in a TBox $\mathcal{T}$ at step t
$isComplexConcept(\mathcal{T}, C, t)$	A concept C is a complex concept in a TBox $\mathcal{T}$ at step t
$complexCInvolvesCon(\mathcal{T}, C, A, t)$	A concept C contains a concept A in a TBox $\mathcal{T}$ at step t
$complexCInvolvesRole(\mathcal{T}, C, r, A, t)$	A concept C contains a role r whose range is filled by A in a TBox $\mathcal{T}$ at step t
IDB predicates	**Description**
$notEqual(\mathcal{T}_1, \mathcal{T}_2, t)$	The TBoxes $\mathcal{T}_1$, $\mathcal{T}_2$ are not equivalent at step t
$conceptsNotEquivalent(\mathcal{T}_1, \mathcal{T}_2,, t)$	The concepts in the TBoxes $\mathcal{T}_1$, $\mathcal{T}_2$ are not equivalent at step t
$complexCConceptNotEq(\mathcal{T}_1, \mathcal{T}_2, C, A, t)$	A concept A in C is not equivalent in the TBoxes $\mathcal{T}_1$, $\mathcal{T}_2$ at step t
$complexCRoleConceptNotEq(\mathcal{T}_1, \mathcal{T}_2, C, r, A, t)$	A concept A filling the role r of C is not equivalent in the TBoxes $\mathcal{T}_1$, $\mathcal{T}_2$ at step t
$poss(a, \mathcal{T}, t)$	An upward refinement step a is executable in a TBox $\mathcal{T}$ at step t
$exec(a, \mathcal{T}, t)$	An upward refinement step a is executed in a TBox $\mathcal{T}$ at step t

Table 2: Overview of the main EDB and IDB predicates used to formalise the upward refinement process in ASP.

1. if $C_i = B$:

$$complexCInvolvesCon(\mathcal{T}, ADef, B, t) \tag{8}$$

2. if $C_i = \exists r.A$:

$$complexCInvolvesRole(\mathcal{T}, ADef, r, A, t) \tag{9}$$

3. if $C_i = \exists r.D$:

$$hasConcept(\mathcal{T}, DDef, t) \tag{10a}$$
$$isComplexConcept(\mathcal{T}, DDef, t) \tag{10b}$$
$$complexCInvolvesRole(\mathcal{T}, ADef, r, DDef, t) \tag{10c}$$
$$isBackgroundConcept(\mathcal{T}, DDef) \tag{10d}$$

The concepts belonging to the domain knowledge part $\mathcal{T}_{dk}$ of a TBox $\mathcal{T}$ are modeled in a similar way but without the $isBackgroundConcept/3$ facts. Besides, we model the concept $\top$ as the fact $hasConcept(\mathcal{T}, Thing, t)$, and for each concept name $A \in N_C$, which is not already subsumed by other concept names, we add a fact $hasParentConcept(\mathcal{T}, A, Thing, t)$. We check for (in)equality of TBoxes by a rule $notEqual(\mathcal{T}_1, \mathcal{T}_2, t) \leftarrow conceptsNotEquivalent(\mathcal{T}_1, \mathcal{T}_2,, t)$. The rule is triggered by additional rules if, for $\mathcal{T}_1$ and $\mathcal{T}_2$, at step t, atomic concepts, roles and complex concepts are not equal.

Formalising upward refinement in ASP

In what follows, we refer to the upward operator types we implemented as γ_A and γ_C. γ_A stands for the generalisation of an atomic concept (first row in Fig. 2) and γ_C for the generalisation of a concept filling the range of a role. Each upward refinement operator type is an action; to this end, we model each operator type via a *precondition* rule, an *inertia* rule, and an *effect* rule. Preconditions are modelled with a predicate $poss/3$ that states when it is possible to execute an operator type. Inertia is modelled with a predicate $noninertial/3$ that states when an element of a concept in $\mathcal{T}$ remains unchanged after the execution of a refinement operator type. Effect rules model how a refinement operator type changes a concept in the domain knowledge. We represent the execution of an upward refinement operator type with atoms $exec(\gamma_x, \mathcal{T}, t)$, to denote that a generalisation operator type $\gamma_x \in \{\gamma_A, \gamma_C\}$ is applied to $\mathcal{T}$ at a step t.

Upward refinement of atomic concepts. A fact $exec(genConcept(C, A, B), \mathcal{T}, t)$ denotes the generalisation of a concept A to a concept B in a complex concept C of a TBox $\mathcal{T}$ at step t using γ_A. The precondition rule for generalising A is:

$$\begin{aligned} poss(genConcept(C, A, B), \mathcal{T}_1, t) \leftarrow \\ &\text{not } isBackgroundConcept(\mathcal{T}_1, C), \\ &complexCInvolvesCon(\mathcal{T}_1, C, A, t), \\ &hasParentConcept(\mathcal{T}_1, A, B, t), \\ &complexCConceptNotEq(\mathcal{T}_1, \mathcal{T}_2, C, A, t), \\ &\text{not } exec(genConcept(C, A, B), \mathcal{T}_2, t), spec(\mathcal{T}_2) \end{aligned} \tag{11}$$

There are several preconditions for generalising an atomic concept in a complex concept C. First, C must not be a background concept since we do not want to modify the background knowledge of a TBox $\mathcal{T}$. Second, the concept C involves a concept A that has a parent concept B in the subsumption hierarchy defined by the axioms of the TBox. Third, the definition of C in the TBoxes is not equivalent ($complexCConceptNotEq/4$). The atom $complexCConceptNotEq/4$ is true either when one of the TBoxes does not contain C or when the definitions of (the structure of) C are different. Another condition is that C has not been generalised in the other TBox.

We also need a simple inertia rule for generalising a concept in a complex concept. This is as follows:

$$noninertial(\mathcal{T}, C, A, t) \leftarrow exec(genConcept(C, A, \_), \mathcal{T}, t) \tag{12}$$

$noninertial$ atoms will cause a concept A to remain in the complex concept C in a TBox $\mathcal{T}$, as defined via rule (15a).

Upward refinement of range concepts. A fact $exec(genConceptInRole(C, r, A, B), \mathcal{T}, t)$ denotes the generalisation of a concept A to a concept B when A fills the range of a role r in a complex concept C of a TBox $\mathcal{T}$ at step t using γ_C. The precondition rule for generalising

a concept A is:

$$poss(genConceptInRole(C, r, A, B), \mathcal{T}_1, t) \leftarrow \quad (13)$$
$$\text{not } isBackgroundConcept(\mathcal{T}_1, C),$$
$$complexCInvolvesRole(\mathcal{T}_1, C, r, A, t)$$
$$hasParentConcept(\mathcal{T}_1, A, B, t),$$
$$complexCRoleConceptNotEq(\mathcal{T}_1, \mathcal{T}_2, C, r, A, t),$$
$$\text{not } isNotInRoleRange(\mathcal{T}_1, r, B), spec(\mathcal{T}_2),$$
$$\text{not } exec(genConceptInRole(C, r, A, B), \mathcal{T}_2, t), spec(\mathcal{T}_2)$$

The preconditions for generalising a concept filling the role of a complex concept C are similar to the case of generalising an atomic concept. First, C must not be a background concept. Second, C involves a role in which the concept to be generalised has a parent concept in the subsumption hierarchy of the TBox. Then, the definitions of the concept to be generalised must not be equivalent in the TBoxes. ($complexCRoleConceptNotEq/4$). Another condition is needed in the case of this generalisation, that is, the concept that we want to use to generalise the range of a role, must be in the range of r. This is checked by means of the atom $isNotInRoleRange/3$. Finally, the concept to be generalised must have not been generalised in the other TBox.

The inertia rule for generalising a concept that fills the range of a role in a TBox is analogous to the inertial rule for generalising a concept:

$$noninertial(\mathcal{T}, C, A, t) \leftarrow \quad (14)$$
$$exec(genConceptInRole(C, A, \_), \mathcal{T}, t)$$

$noninertial$ atoms will cause a concept A to remain in the range of a role as defined via rule (15b).

Inertia. The following rules state which concepts remain in a TBox $\mathcal{T}$ when they are inertial.

$$complexCInvolvesCon(\mathcal{T}, C, A, t+1) \leftarrow \quad (15a)$$
$$complexCInvolvesCon(\mathcal{T}, C, A, t),$$
$$\text{not } noninertial(\mathcal{T}, C, A, t)$$

$$complexCInvolvesRole(\mathcal{T}, C, r, A, t+1) \leftarrow \quad (15b)$$
$$complexCInvolvesRole(\mathcal{T}, C, r, A, t),$$
$$\text{not } noninertial(\mathcal{T}, C, A, t)$$

Besides, other inertia rules are needed for expressing that all the concepts and roles of the background knowledge and their subsumption relations remain in a TBox $\mathcal{T}$. We omit them.

Effects. The following rules state which concepts change in a TBox $\mathcal{T}$ when they are generalised.

$$complexCInvolvesCon(\mathcal{T}, C, B, t+1) \leftarrow \quad (16a)$$
$$complexCInvolvesCon(\mathcal{T}, C, A, t),$$
$$exec(genConcept(C, A, B), \mathcal{T}, t)$$

$$complexCInvolvesRole(\mathcal{T}, C, r, B, t+1) \leftarrow \quad (16b)$$
$$complexCInvolvesRole(\mathcal{T}, C, r, A, t),$$
$$exec(genConceptInRole(C, r, A, B), \mathcal{T}, t)$$

Upward refinement search

We use ASP for finding a generic space and the generalised versions of the concepts in the domain knowledge of $\mathcal{T}$, which can lead to a blend. This is done by successively generalising the concepts in the domain knowledge by means of the upward operator steps we described in the previous subsection. Again, this is a first implementation and does not capture the recursive definition of the upward refinement operator that we leave as future work.

A sequence of generalisation operator types defines a *refinement path*.

Definition 7 (Refinement path)**.** *Let $\mathcal{T}$ be a TBox, let $\{\gamma_x^1, \ldots, \gamma_x^n\}$ be the set of generalisation steps for $\mathcal{T}$, $t_1 < \cdots < t_n$ be refinement steps and $\gamma_x \in \{\gamma_A, \gamma_C\}$. The set of atoms $P = \{exec(\gamma_x^1, \mathcal{T}, t_1), \cdots, exec(\gamma_x^n, \mathcal{T}, t_n)\}$ is a refinement path of $\mathcal{T}$.*

Refinement paths are generated with the following choice rule, that allows one or zero refinement operators per $\mathcal{T}$ at each step t:

$$0\{exec(a, \mathcal{T}, t) : poss(a, \mathcal{T}, t)\}1 \leftarrow \quad (17)$$
$$not\ genericReached(t), spec(\mathcal{T})$$

The only generalisations that are executed are those whose preconditions are satisfied. Refinement paths lead from the input TBoxes to a generic space, which is a generalised TBox that contains the commonalities of the concepts in the domain knowledge. A generic space is reached, if the generalised TBoxes are equals. The $notEqual$ predicate is used to determine if a generic space is reached.

$$notGenericReached(t) \leftarrow spec(\mathcal{T}_1), spec(\mathcal{T}_2), \quad (18a)$$
$$notEqual(\mathcal{T}_1, \mathcal{T}_2, t), \mathcal{T}_1 \neq \mathcal{T}_2$$
$$\leftarrow notGenericReached(t) \quad (18b)$$

The constraint (18b) ensures that the generic space is reached in all stable models. The ASP program generates one stable model for each combination of generalisation paths that lead to the generic space.

Example 3. *Let us consider the* SearchHardDisk *and* EditDocument *concepts in Example 1 representing icons in the domain knowledge of two TBoxes $SearchHD$ and $EditDoc$. Their refinement paths are:*

$$P_{SearchHD} = \{exec(genConceptInRole(SearchHDDef, hasSign, HardDisk, Sign, SearchHD, 0),$$
$$exec(genConceptInRole(SearchHDDefDef, isAbove, HardDisk, Sign), SearchHD, 1)\}$$
$$exec(genConcept(SearchHDDefDef, MagnifyingGlass, Sign), SearchHD, 2),$$
$$P_{EditDoc} = \{exec(genConcept(EditDocDefDef, Pen, Sign), EditDoc, 0),$$
$$exec(genConceptInRole(EditDocDef, hasSign, Document, Sign), EditDoc, 1),$$
$$exec(genConceptInRole(EditDocDefDef), isAbove, Document, Sign), EditDoc, 2)\}$$

After applying the respective generalisation operators a generic space is reached. It is easy to check that this corresponds to the generic space in Example 2.

Blending $\mathcal{EL}^{++}$ concepts

In Definition 1, we defined the blends of two $\mathcal{EL}^{++}$ concepts in terms of amalgams. Once a generic space between two concepts has been determined, blends are created by computing the MGS of pairs of generalised concepts. In $\mathcal{EL}^{++}$ the MGS of two $\mathcal{EL}^{++}$ concepts is just their conjunction. Then we will have to normalise this conjunction to obtain the most concise description of the newly created concept. This is just a rough description of conceptual blending in $\mathcal{EL}^{++}$, and normalisation for newly created blends still needs to be studied in detail (this also relates to the operator properness). Blending will also need to consider an interleaved evaluation and generation process in order to find the best pairs of generalised concepts for creating interesting blends.

Example 4. *Let us consider* C_1 = SearchHardDisk, C_2 = EditDocument, *G in Example 2 and the refinement paths $P_{SearchHD}$, $P_{EditDoc}$ in Example 3. The generalisations of C_1 and C_2 by applying the generalisation steps 0-1 in $P_{SearchHD}$ and step 0 in $P_{EditDoc}$ respectively are:*

C_1' = Icon ⊓ ∃hasSign.Sign ⊓ ∃hasSign.
(MagnifyingGlass ⊓ ∃isAbove.Sign)
C_2' = Icon ⊓ ∃hasSign.Document ⊓ ∃hasSign.
(Sign ⊓ ∃isAbove.Document)

Then, the conjunction $C_1' \sqcap C_2'$ = (Icon ⊓ ∃hasSign. Sign⊓ ∃hasSign. (MagnifyingGlass ⊓∃isAbove.Sign)) ⊓ (Icon⊓ ∃hasSign.Document⊓ ∃hasSign. (Sign ⊓∃isAbove.Document)). $C_1' \sqcap C_2'$ *is simplified to obtain the blend concept* Icon⊓ ∃hasSign. Document⊓ ∃hasSign.(MagnifyingGlass ⊓∃isAbove.Document) *by normalisation.*

Conceptual blending in $\mathcal{EL}^{++}$ as described in this paper is a special case of blending as modelled in (Bou et al. 2014) and implemented for CASL theories in (Eppe et al. 2015).

Related Work

There exist several works that relate to ours from different perspectives, that are, approaches to conceptual blending of ontologies, approaches for finding the LGG in the $\mathcal{EL}$ family, and approaches that uses ASP for reasoning over DL ontologies.

Conceptual blending of ontologies in DLs has been explored in (Hois et al. 2010; Kutz et al. 2014) where blends are computed as colimits of algebraic specifications. As such, the blending process is not characterised in terms of amalgamation, the input concepts are not generalised, and the generic space is assumed to be given.

Several approaches for generalising ontology concepts in the $\mathcal{EL}$ family exist in the DLs and ILP literature.

On the one hand, in DL approaches the LGG is defined in terms of a non-standard reasoning task over a TBox (Baader 2003; 2005; Baader, Sertkaya, and Turhan 2007; Zarrießand Turhan 2013; Turhan and Zarrieß 2013). Generally speaking, since the LGG w.r.t. general TBoxes in the $\mathcal{EL}$ family does usually not exist, these approaches propose several 'workarounds' for computing it. For instance, (Baader 2003; 2005) devises the exact conditions for the existence of the LGG for cyclic $\mathcal{EL}$-TBoxes based on graph-theoretic generalisations. (Baader, Sertkaya, and Turhan 2007) propose an algorithm for computing good LGGs w.r.t. a background terminology. (Zarrießand Turhan 2013; Turhan and Zarrieß 2013) specify the conditions for the existence of the LGG for general $\mathcal{EL}$- and $\mathcal{EL}^+$-TBoxes based on canonical models. As already commented in the introduction, our work relates to these approaches, but it is different in spirit, since we do not need to find the LGG between (two) $\mathcal{EL}^{++}$ concepts for the kind of application we are developing.

ILP approaches, on the other hand, study the LGG in terms of generalisation and specialisation refinement operators. Specialisation rsefinement operators have been defined for learning DL ontologies (Lehmann and Hitzler 2008; Lehmann and Haase 2010) and for measuring the similarity of $\mathcal{EL}$ concepts (Sánchez-Ruiz et al. 2011; 2013).

Finally, some of the approaches that combine ASP for reasoning over DL ontologies are (Swift 2004; Eiter et al. 2008; Ricca et al. 2009).

Conclusion and Future Works

In this paper, we defined an upward refinement operator for generalising $\mathcal{EL}^{++}$ concepts for conceptual blending. The operator works by recursively traversing their descriptions. We discussed the properties of the refinement operator. The operator is finite, can be proper (by allowing a normalisation before each refinement step), but it is not complete. We claimed, however, that completeness is not an essential property for our needs, since being able to find a generic space between two $\mathcal{EL}^{++}$ concepts, although not a LGG, is already a sufficient condition for conceptual blending.

We presented a first implementation of the refinement operator in ASP. We showed how to model the description of $\mathcal{EL}^{++}$ concepts in ASP and to formalise two refinement operator types for generalising the domain knowledge of a TBox. The stable models of the ASP program contain the generalisation steps needed to be applied in order to generalise two $\mathcal{EL}^{++}$ concepts until a generic space is reached. We discussed how the blend of two $\mathcal{EL}^{++}$ concepts, defined in terms of their most general specification, can be obtained by their conjunction. We exemplified our approach in the domain of computer icon design.

As future works, we plan to continue with the implementation of the $\mathcal{EL}^{++}$ generalisation operator, to investigate the normalisation rules needed for the operator to be proper and for normalising the blends, as well as to implement a conceptual blending algorithm. We also aim at studying the relationship between the category of CASL theories (Mosses 2004) and signature morphisms and the category of $\mathcal{EL}^{++}$ concept descriptions and subsumption relation.

We consider the work of this paper to be a fundamental step towards the challenging task of defining and implementing an upward refinement operator for more expressive DLs in the context of conceptual blending.

Acknowledgements

This work is partially supported by the COINVENT project (FET-Open grant number: 611553).

References

Baader, F.; Brandt, S.; and Lutz, C. 2005. Pushing the EL Envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 364–369. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Baader, F.; Brandt, S.; and Lutz, C. 2008. Pushing the EL Envelope Further. In Clark, K., and Patel-Schneider, P. F., eds., *In Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*.

Baader, F.; Sertkaya, B.; and Turhan, A.-Y. 2007. Computing the least common subsumer w.r.t. a background terminology. *Journal of Applied Logic* 5(3):392 – 420.

Baader, F. 2003. Computing the Least Common Subsumer in the Description Logic $\mathcal{EL}$ w.r.t. Terminological Cycles with Descriptive Semantics. In Ganter, B.; de Moor, A.; and Lex, W., eds., *Conceptual Structures for Knowledge Creation and Communication*, volume 2746 of *LNCS*. Springer Berlin Heidelberg. 117–130.

Baader, F. 2005. A Graph-Theoretic Generalization of the Least Common Subsumer and the Most Specific Concept in the Description Logic $\mathcal{EL}$. In Hromkovič, J.; Nagl, M.; and Westfechtel, B., eds., *Graph-Theoretic Concepts in Computer Science*, volume 3353 of *LNCS*. Springer Berlin Heidelberg. 177–188.

Bou, F.; Eppe, M.; Plaza, E.; and Schorlemmer, M. 2014. D2.1: Reasoning with Amalgams. Technical report, COINVENT Project. http://www.coinvent-project.eu/fileadmin/publications/D2.1.pdf.

Confalonieri, R.; Corneli, J.; Pease, A.; Plaza, E.; and Schorlemmer, M. 2015. Using Argumentation to Evaluate Concept Blends in Combinatorial Creativity. In *Proceedings of the 6th International Conference on Computational Creativity, ICCC15*. To Appear.

Eiter, T.; Ianni, G.; Lukasiewicz, T.; Schindlauer, R.; and Tompits, H. 2008. Combining answer set programming with description logics for the semantic web. *Artificial Intelligence* 172(1213):1495 – 1539.

Eppe, M.; Confalonieri, R.; MacLean, E.; Kaliakatsos, M.; Cambouropoulos, E.; Schorlemmer, M.; and Kühnberger, K.-U. 2015. Computational invention of cadences and chord progressions by conceptual chord-blending. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI 2015*. To Appear.

Fauconnier, G., and Turner, M. 2002. *The Way We Think: Conceptual Blending And The Mind's Hidden Complexities*. Basic Books.

Gelfond, M., and Kahl, Y. 2014. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. New York, NY, USA: Cambridge University Press.

Hois, J.; Kutz, O.; Mossakowski, T.; and Bateman, J. 2010. Towards ontological blending. In Dicheva, D., and Dochev, D., eds., *Artificial Intelligence: Methodology, Systems, and Applications*, volume 6304 of *LNCS*. Springer Berlin Heidelberg. 263–264.

Kutz, O.; Bateman, J.; Neuhaus, F.; Mossakowski, T.; and Bhatt, M. 2014. E pluribus unum: Formalisation, Use-Cases, and Computational Support for Conceptual Blending. In *Computational Creativity Research: Towards Creative Machines*, Thinking Machines. Atlantis/Springer.

Lehmann, J., and Haase, C. 2010. Ideal Downward Refinement in the EL Description Logic. In *Proc. of the 19th Int. Conf. on Inductive Logic Programming*, ILP'09, 73–87. Berlin, Heidelberg: Springer-Verlag.

Lehmann, J., and Hitzler, P. 2008. A Refinement Operator Based Learning Algorithm for the ALC Description Logic. In *Proceedings of the 17th Int. Conf. on Inductive Logic Programming*, 147–160. Berlin, Heidelberg: Springer-Verlag.

Lehmann, J., and Hitzler, P. 2010. Concept learning in description logics using refinement operators. *Machine Learning* 78(1-2):203–250.

Mosses, P. D. 2004. *CASL Reference Manual – The Complete Documentation of the Common Algebraic Specification Language*, volume 2960 of *LNCS*. Springer.

Ontañón, S., and Plaza, E. 2010. Amalgams: A Formal Approach for Combining Multiple Case Solutions. In Bichindaritz, I., and Montani, S., eds., *Proceedings of the International Conference on Case Base Reasoning*, volume 6176 of *LNCS*, 257–271. Springer.

Ricca, F.; Gallucci, L.; Schindlauer, R.; DellArmi, T.; Grasso, G.; and Leone, N. 2009. OntoDLV: An ASP-based System for Enterprise Ontologies. *Journal of Logic and Computation* 19(4):643–670.

Sánchez-Ruiz, A.; Ontañón, S.; González-Calero, P.; and Plaza, E. 2011. Measuring Similarity in Description Logics Using Refinement Operators. In Ram, A., and Wiratunga, N., eds., *Case-Based Reasoning Research and Development*, volume 6880 of *LNCS*. Springer Berlin. 289–303.

Sánchez-Ruiz, A.; Ontañón, S.; González-Calero, P.; and Plaza, E. 2013. Refinement-Based Similarity Measure over DL Conjunctive Queries. In Delany, S., and Ontañón, S., eds., *Case-Based Reasoning Research and Development*, volume 7969 of *LNCS*. Springer Berlin. 270–284.

Swift, T. 2004. Deduction in Ontologies via ASP. In Lifschitz, V., and Niemelä, I., eds., *Logic Programming and Nonmonotonic Reasoning*, volume 2923 of *LNCS*. Springer Berlin. 275–288.

Turhan, A., and Zarrieß, B. 2013. Computing the lcs w.r.t. general $\mathcal{EL}^+$-tboxes. In *Proceedings of the 26th International Workshop on Description Logics*, 477–488.

van der Laag, P. R., and Nienhuys-Cheng, S.-H. 1998. Completeness and properness of refinement operators in inductive logic programming. *The Journal of Logic Programming* 34(3):201 – 225.

Zarrieß, B., and Turhan, A.-Y. 2013. Most Specific Generalizations w.r.t. General EL-TBoxes. In *Proceedings of the 23th International Joint Conference on Artificial Intelligence*, IJCAI '13, 1191–1197. AAAI Press.