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

Upward Refinement Operators for Conceptual Blending in the Description Logic $\mathcal{EL}^{++}$

Roberto Confalonieri · Manfred Eppe · Marco Schorlemmer · Oliver Kutz · Rafael Peñaloza · Enric Plaza

Received: date / Accepted: date

Abstract Conceptual blending is a mental process that serves a variety cognitive purposes, including human creativity. In this line of thinking, human creativity is modeled as a process that takes different mental spaces as input and combines them into a new mental space, called a *blend*. According to this form of *combinational creativity*, a blend is constructed by taking the commonalities among the input mental spaces into account, to form a so-called *generic space*, and by projecting the non-common structure of the input spaces in a selective way to the novel blended space. 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. The generalisations obtained by the ASP process are used in a conceptual blending algorithm that generates and evaluates possible combinations of blends. We exemplify our approach in the domain of computer icons.

Keywords Computational Creativity · Conceptual blending · Description logic · Answer Set Programming

PACS 07.05.Mh · 89.20.Ff

R. Confalonieri · M. Schorlemmer · E. Plaza
Artificial Intelligence Research Institute (IIIA-CSIC)
Campus Universitat Autònoma Barcelona
E-08193 Bellaterra, Catalonia, Spain
E-mail: {confalonieri,marco,enric}@iiia.csic.es

M. Eppe
International Computer Science Institute
Berkeley, USA
E-mail: eppe@icsi.berkeley.edu

O. Kutz · R. Peñaloza
Free University of Bozen-Bolzano
Bolzano, Italy
E-mail: {oliver.kutz,rafael.penaloza}@unibz.it

1 Introduction

The upward refinement—or generalisation—of concepts plays a crucial role in creative cognitive processes for analogical reasoning and concept invention. In this work we focus on its role in *conceptual blending* [20], where one combines two input concepts to invent a new one. A problem in blending is that the combination of two concepts may generate an unsatisfiable one due to contradiction, or may not satisfy certain properties. However, by generalising input concepts, we can remove inconsistencies to find a novel and useful combination of the input concepts. 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 called *generic space* in conceptual blending, which states the shared structure of both concepts.

This paper addresses the formalisation and implementation of such a generalisation process in the context of the description logic $\mathcal{EL}^{++}$ [4, 6]. The choice of $\mathcal{EL}^{++}$ as the knowledge representation language for a computational interpretation of the cognitive theory of conceptual blending is motivated by several reasons. First, $\mathcal{EL}^{++}$ is the underpinning logic of the OWL 2 EL Profile[1], a recommendation of the W3C, and, therefore, a well-understood and commonly used knowledge representation formalism. Second, $\mathcal{EL}^{++}$ offers a good tradeoff between expressiveness and efficiency of reasoning and is considered to be sufficiently expressive to model large real-world ontologies, specially in the bio-medical domains [14, 40]. Finally, subsumption of concepts w.r.t. an $\mathcal{EL}^{++}$ TBox is computable in polynomial time [4], and therefore of special interest for a tractable real-world implementation of conceptual blending. Indeed, a nontrivial problem of conceptual blending is that there usually exists a considerable number of possible combinations for the blend creation that are inconsistent or otherwise not interesting (see e.g. [19]). These combinations need to be evaluated. Our $\mathcal{EL}^{++}$-based formalisation of conceptual blending suggests that these combinations, leading to the blends, can be evaluated against the entailment of some properties, modelled as ontology consequence requirements. The nice computational properties of $\mathcal{EL}^{++}$ facilitate this kind of evaluation since the entailment in $\mathcal{EL}^{++}$ is not computationally hard.

The generalisation of $\mathcal{EL}^{++}$ concepts has been studied both in the Description Logic (DL) 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 [2, 5, 43], approaches in ILP are concerned on learning DL descriptions from examples [32].

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 properties are shared by the concepts in order to blend them, it is not necessary

[1] `http://www.w3.org/TR/owl2-profiles/`, accessed 26/11/2015

(though desirable) to find a *generic space* that is also a LGG. A minimally specific common subsumer w.r.t. the subconcepts that can be built using the axioms in a Tbox 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, local finiteness, properness and completeness [29].[2] Particularly, our operator is locally finite and proper, but it is not complete. As a consequence, it cannot generate all the possible generalisations of an $\mathcal{EL}^{++}$ concept. As we shall discuss, we sacrifice completeness for finiteness (since we do not need to compute a LGG, strictly speaking), but we need the applications of the operator to always terminate at each refinement step.

As far as the implementation of the operator is concerned, we state the problem of finding a generic space of $\mathcal{EL}^{++}$ concepts as a planning problem. This involves finding a sequence of generalisations with conditional effects to reach the generic space. This is natural, because modifying $\mathcal{EL}^{++}$ concepts underlies certain conditional rules. These rules are ultimately defined through the *upward cover set* which is generated within the generalisation operator definitions (see Definitions 6 and 7). It is well-known that planning problems are inherently non-monotonic because of the inertia assumption. That is, one assumes that world properties, in this case parts of $\mathcal{EL}^{++}$ concept descriptions, persist unless there is evidence that they changed. The 'unless there is evidence' condition implies the use of Negation as Failure (NaF). To this end, we adopt the nonmonotonic logic programming paradigm of Answer Set Programming (ASP) [23].[3]

To implement the the upward refinement operator and generic space search, we employ the incremental solving capabilities of *clingo* [21], an advanced ASP solver, to find a generic space among two $\mathcal{EL}^{++}$ input concepts. The ASP search is embedded in an amalgam-based process that models conceptual blending. We present a conceptual blending algorithm that uses the generalisations found by the ASP-based search process to create new blended concepts. New concepts are evaluated by means of ontology consequence requirements and a heuristics function. Throughout the paper, we use an example in the domain of computer icon design.

This paper is an extended and revised version of [13]. It now contains a formal definition and analysis of the refinement operator properties (Propositions 1-3 and Theorem 1), an extension of the operator definition to deal with infinite chain of generalisations, the complete implementation of the operator in ASP, and a blending algorithm.

The remainder of this paper is organised as follows: Section 2 provides the background knowledge to make this paper self-contained. Section 3 describes how conceptual blending can be used to design new computer icons modeled in $\mathcal{EL}^{++}$. Section 4 proposes the formalisation of a refinement operator for generalising $\mathcal{EL}^{++}$

---

[2] Briefly, a refinement operator is said to be locally finite when it generates a finite set of refinements at each step; proper, when its refinements are not equivalent to the original concept, and complete, when it produces all possible refinements of a given concept. These property are formally presented in Section 2.2.

[3] The planning problem could also have been encoded in SAT. There are many approaches to realize NaF and non-monotonicity for SAT, with circumscription [34] probably being the most prominent method. In this sense, the computational complexity is equivalent with the one of ASP. However, since NaF is already an inherent part of ASP, we found ASP more straightforward. This is also in-line with recent trends in Commonsense Reasoning about Action and Change, where ASP is commonly used to solve planning problems (see e.g., [16, 17, 30, 33]).

concepts. In Section 5, the implementation of the operator and the ASP incremental encoding, which models the generic space search, are presented. Section 6 describes an algorithm for conceptual blending. Section 7 outlines several works that relate to ours from different perspectives. Finally, Section 8 concludes the paper and envisions some future work.

2 Background

In this section we introduce the basic notions that will be used throughout the paper. After presenting the $\mathcal{EL}^{++}$ description logic, we introduce refinement operators. Then, we provide the definition of *amalgams* that provides a computational chacterisation of conceptual blending. We conclude the background with an overview of Answer Set Programming (ASP) and the incremental solving capabilities of *clingo*.

2.1 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. The upper part of 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 [4, 6].

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 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. The lower part of 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

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

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

$\mathcal{T}$. 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 $C \equiv_{\mathcal{T}} D$ is just an abbreviation for $C \sqsubseteq_{\mathcal{T}} D$ and $D \sqsubseteq_{\mathcal{T}} C$.

2.2 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.

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.[4]

Definition 1 A generalisation refinement operator γ over a quasi-ordered set $\langle \mathcal{S}, \preceq \rangle$ is a set-valued 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 [29]. 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;

[4] A deeper analysis of refinement operators can be found in [29].

– *complete*, if there are no generalisations that 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 zero or 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 [31]. In this paper, we define a generalisation refinement operator for $\mathcal{EL}^{++}$ and study its properties.

2.3 Computational Concept Blending by Amalgams

The process of conceptual blending can be characterised in terms of *amalgams* [36], a notion that has its root in case-based reasoning and focuses on the problem of combining solutions coming from multiple cases in search-based approaches to reuse and that has also been used to model analogy [8]. 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.

Formally, the notion of amalgams 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 $\mathcal{T}$.

Definition 2 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, $C_{lgg} \sqsubseteq_{\mathcal{T}} D$.

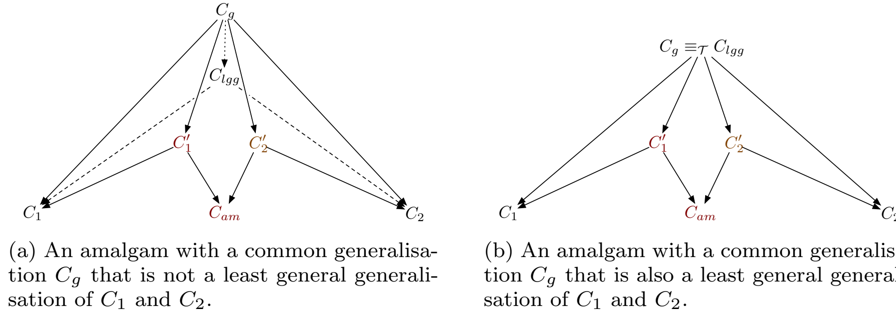
Intuitively, a MGS is a description that has all the information from both the original descriptions C_1 and C_2, while a LGG contains that which is common to them. Depending on the structure of $\mathcal{T}$, it is not always possible to find a least general generalisation. Thus, the definition of C_{lgg} is relaxed as follows.

Definition 3 Given two descriptions $C_1, C_2 \in \mathcal{L}(\mathcal{T})$, a *common generalisation* is a description C_g such that $C_1 \sqsubseteq_{\mathcal{T}} C_g$ and $C_2 \sqsubseteq_{\mathcal{T}} C_g$.

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' could be '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 4 (Amalgam) Let $\mathcal{T}$ be an $\mathcal{EL}^{++}$ TBox. A description $C_{am} \in \mathcal{L}(\mathcal{T})$ is an *amalgam* of two descriptions C_1 and C_2 (with common generalisation C_g) if there exist two descriptions C_1' and C_2' such that:

1. $C_1 \sqsubseteq_{\mathcal{T}} C_1' \sqsubseteq_{\mathcal{T}} C_g$,



(a) An amalgam with a common generalisation C_g that is not a least general generalisation of C_1 and C_2.

(b) An amalgam with a common generalisation C_g that is also a least general generalisation of C_1 and C_2.

Fig. 1: Two diagrams of an amalgam C_{am} from descriptions C_1 and C_2 with generalisations C'_1 and C'_2. Arrows indicate the subsumption of the target by the source of the arrow.

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

This definition is illustrated in Figure 1, where the common generalisation of the inputs is indicated as C_g, and the amalgam C_{am} is the MGS of two concrete generalisations C'_1 and C'_2 of the inputs. Notice that C_g used to define the amalgam does not need to be a least general generalisation. Although having a least general generalisation is desirable, a common generalisation of the inputs will suffice.

In Section 4, 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. We may generalise concepts C_1 and C_2 beyond the LGG but we need to do this to guarantee termination, as we shall explain. We implement the operator and the search for generalisation in Answer Set Programming (ASP) [23]. To this end, we provide some basic notions about ASP in the next section.

2.4 Answer Set Programming

Answer Set Programming (ASP) is a declarative approach to solve NP-hard search problems (see e.g. [7, 23]). An ASP program is similar to a PROLOG program in that it is non-monotonic, takes logic programming style Horn clauses as input, and uses negation-as-failure (NaF). However, instead of using Kowalski [27]'s SLDNF resolution semantics as in PROLOG, it employs Gelfond and Lifschitz [24]'s Stable Model Semantics, which makes it truly declarative, i.e., the order in which ASP rules appear in a logic program does not matter. Furthermore, the Stable Model Semantics has the advantage that Answer Set Programs always terminate, while PROLOG programs do not. For example, given a program $p \leftarrow \textit{not } q.$ and $q \leftarrow \textit{not } p.$, asking whether p holds results in an infinite loop for PROLOG, while ASP returns two stable models as solution, namely the sets $\{p\}$ and $\{q\}$.

An ASP program consists of a set of rules, facts and constraints. Its solutions are called *Stable Models* (SM). In this paper we only consider so-called *normal* rules [7], which are written as:

$$a_0 \leftarrow a_1, \ldots, a_j, \textit{not } a_{j+1}, \ldots, \textit{not } a_n \quad (1)$$

in which $a_1, ..., a_n$ are atoms and *not* is negation-as-failure. When $n = 0$ the rule $a_0 \leftarrow$ is known as a fact and the $\leftarrow$ is omitted. A constraint is a rule of the form $\leftarrow a_1, \ldots, a_j, not\ a_{j+1}, \ldots, not\ a_n$. Constraints are rules that are used to discard some models of a logic program.

The models of an ASP program are defined according to the *stable model semantics*. The stable semantics is defined in terms of the so-called *Gelfond-Lifschitz reduction* [24]. Let $\mathcal{L}_P$ be the set of atoms in the language of a normal logic program P, then for any set $M \subseteq \mathcal{L}_P$, the Gelfond-Lifschitz reduction P^M is the definite logic program obtained from P by deleting:

(i) each rule that has a formula *not a* in its body with $a \in M$, and
(ii) all formulæ of the form *not a* in the bodies of the remaining rules.

P^M does not contain *not* and M is called a *stable model* of P if and only if M is the minimal model of P^M. A stable model M of an ASP program P contains those atoms that satisfy all the rules in the program and, consequently, represent a solution of the problem that represents.

ASP is interesting not only because can capture complex knowledge representation problems, but also because efficient ASP implementations exists. In particular, the *clingo* solver [21] offers a step-oriented, incremental approach that allows us to control and modify an ASP program at run-time, without the need of restarting the grounding the solving process from scratch. To this end, a program is partitioned into a base part, describing the static knowledge independent of a step parameter t, a cumulative part, capturing knowledge accumulating with increasing t, and a volatile part specific for each value of t. The grounding and integration of these subprograms into the solving process is completely modular and controllable from a scripting language such as Python.

The ASP implementation in this paper follows this methodology of specifying and solving a problem incrementally. For further details about incremental solving, we refer to [22] in which several examples can be found.

3 Conceptual Blending of Computer Icons

To exemplify our approach, we take the domain of computer icons into account. We consider computer icons as combinations of signs, such as *Document*, *MagnifyingGlass*, *HardDisk* and *Pen* that are described in terms of meanings [12]. Meanings convey *actions-in-the-world* or *object-types*.

Figure 2 shows the concept names defined in the *ComputerIcon* ontology and their relations. In what follows, concept names are capitalised (e.g., Sign) and role names are not (e.g., hasMeaning). 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 $\mathcal{T}_{dk}$. $\mathcal{T}_{bk}$ contains the following axioms:

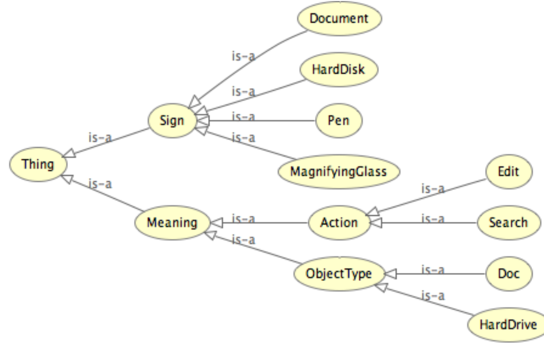


Fig. 2: The *ComputerIcon* ontology, showing the concept names and their relation.

α_{bk_1}: Action $\sqsubseteq$ Meaning
α_{bk_2}: ObjectType $\sqsubseteq$ Meaning
α_{bk_3}: Search $\sqsubseteq$ Action
α_{bk_4}: Edit $\sqsubseteq$ Action
α_{bk_5}: HardDrive $\sqsubseteq$ ObjectType
α_{bk_6}: Doc $\sqsubseteq$ ObjectType
α_{bk_7}: Action $\sqcap$ ObjectType $\sqsubseteq \bot$
α_{bk_8}: Search $\sqcap$ Edit $\sqsubseteq \bot$
... ...
$\alpha_{\mathsf{bk}_{14}}$: HardDrive $\sqcap$ Doc $\sqsubseteq \bot$

Axioms α_{bk_1}-α_{bk_6} capture the different meanings associated with signs; axioms α_{bk_7}-$\alpha_{\mathsf{bk}_{14}}$ model the disjointness among all Action and ObjectType concepts defined in the ontology. Signs are associated with a meaning. This is modeled by the hasMeaning role in the following axioms:

$\alpha_{\mathsf{bk}_{15}}$: MagnifyingGlass $\equiv$ Sign $\sqcap$ $\exists$hasMeaning.Search
$\alpha_{\mathsf{bk}_{16}}$: HardDisk $\equiv$ Sign $\sqcap$ $\exists$hasMeaning.HardDrive
$\alpha_{\mathsf{bk}_{17}}$: Pen $\equiv$ Sign $\sqcap$ $\exists$hasMeaning.Edit
$\alpha_{\mathsf{bk}_{18}}$: Document $\equiv$ Sign $\sqcap$ $\exists$hasMeaning.Doc
$\alpha_{\mathsf{bk}_{19}}$: MagnifyingGlass $\sqcap$ HardDisk $\sqsubseteq \bot$
... ...
$\alpha_{\mathsf{bk}_{25}}$: Pen $\sqcap$ Document $\sqsubseteq \bot$

A sign is associated with a meaning. For instance, MagnifyingGlass is associated with Search to describe that it conveys the action of looking for something. Sign concepts are disjoint ($\alpha_{\mathsf{bk}_{19}}$-$\alpha_{\mathsf{bk}_{25}}$). Signs are related by spatial relationships such as isAboveIn, isAboveInLeft, isAboveInRight, isUpIn, isUpLeft, isUpRight, isDownIn, isDownLeft, and isDownRight. Spatial relationships are modelled as roles.

$\alpha_{\mathsf{bk}_{26}}$: isAboveIn $\sqsubseteq$ isInSpatialRelation
$\alpha_{\mathsf{bk}_{27}}$: isAboveLeft $\sqsubseteq$ isInSpatialRelation
$\alpha_{\mathsf{bk}_{28}}$: isAboveRight $\sqsubseteq$ isInSpatialRelation
... ...
$\alpha_{\mathsf{bk}_{37}}$: isDownRight $\sqsubseteq$ isInSpatialRelation

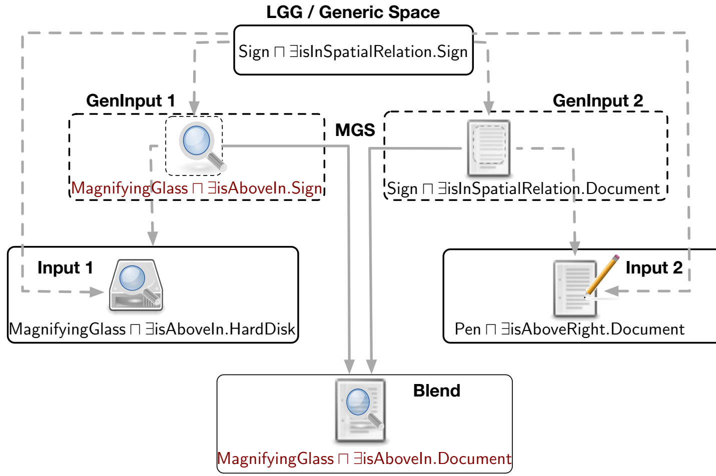


Fig. 3: Blending the SearchHardDisk and EditDocument icon concepts into a new concept representing a search-in-document icon. Sign's meanings are not represented.

For the sake of simplicity, we assume that icons are modelled according to a canonical form. Axioms describing icon concepts are of the form $\mathsf{IconName} \equiv \mathsf{C} \sqcap \exists \mathsf{r}.\mathsf{D}$, where r is a spatial relation and C, D are concepts that describe signs. Based on this canonical form and on the axioms above, we modeled some icons as 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 in the middle of the HardDisk sign. Another icon is EditDocument, where the Pen sign is above on the right of the Document sign. Both icons are shown in Figure 3.

$$\alpha_{\mathsf{dk}_1}: \quad \mathsf{SearchHardDisk} \equiv \mathsf{MagnifyingGlass} \sqcap \exists \mathsf{isAboveIn}.\mathsf{HardDisk}$$
$$\alpha_{\mathsf{dk}_2}: \quad \mathsf{EditDocument} \equiv \mathsf{Pen} \sqcap \exists \mathsf{isAboveRight}.\mathsf{Document}$$

We consider the above knowledge as a library of icons. We assume that the library is managed and used by a computer icon design tool. The tool accepts a query as input and retrieves those icons that satisfy certain properties. For instance, a query asking for an icon with the meaning of searching in a hard-disk will retrieve the SearchHardDisk concept. In contrast, a query asking for an icon with the meaning of searching in a document does not return any result. In such a case, the tool tries to answer the query by running a conceptual blending algorithm.

Intuitively, the conceptual blending algorithm works as follows. Given two input concepts, the algorithm tries to create new concepts that can satisfy the query. New concepts are created by taking the commonalities and some of their specifics into account (Figure 3). For instance, both SearchHardDisk and EditDocument are icons that consist of two signs related by a spatial relation (the generic space). Then, if we keep the MagnifyingGlass concept from SearchHardDisk and the Document concept from EditDocument, and we generalise the HardDisk and Pen concepts and

the role isAboveRight, we can blend the generalised input concepts of SearchHardDisk and EditDocument into a new concept representing an icon whose meaning is to search in a document.

$$\mathsf{MagnifyingGlass} \sqcap \exists \mathsf{isAboveIn}.\mathsf{Document}$$

In this paper, we show how the above concept generation description can be computationally realised by two processes. An ASP-based implementation that generalises $\mathcal{EL}^{++}$ concept descriptions and finds a generic space; and a procedural implementation that generates and evaluates the blended concepts. First, we introduce a refinement operator for generalising an $\mathcal{EL}^{++}$ concept.

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

In any description logic the set of 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, $\top$ as greatest element, and $\bot$ as least element. In order to define a generalisation refinement operator for $\mathcal{EL}^{++}$, we need some auxiliary definitions.

Definition 5 Let $\mathcal{T}$ be an $\mathcal{EL}^{++}$ TBox. The set of *subconcepts* of $\mathcal{T}$ is given as

$$\mathsf{sub}(\mathcal{T}) = \{\top, \bot\} \cup \bigcup_{C \sqsubseteq D \in \mathcal{T}} \mathsf{sub}(C) \cup \mathsf{sub}(D) \tag{2}$$

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.[5]

Definition 6 Let $\mathcal{T}$ be an $\mathcal{EL}^{++}$ TBox with concept names from N_C. The *upward cover set* 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:

$$\begin{aligned}
\mathsf{UpCov}(A) := \{C \in \mathsf{sub}(\mathcal{T}) \mid\ & A \sqsubseteq_{\mathcal{T}} C \\
& \text{and for all } C' \in \mathsf{sub}(\mathcal{T}) \text{ such that } A \sqsubseteq_{\mathcal{T}} C' \\
& \text{then } C \sqsubseteq_{\mathcal{T}} C'
\end{aligned} \tag{3}$$

$$\begin{aligned}
\mathsf{UpCov}(r) := \{s \in N_R \mid\ & r \sqsubseteq_{\mathcal{T}} s \\
& \text{for all } s' \in N_r \text{ such that } r \sqsubseteq_{\mathcal{T}} s' \\
& \text{then } s \sqsubseteq_{\mathcal{T}} s'\}
\end{aligned} \tag{4}$$

[5] We assume that $\mathcal{T}$ is finite.

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

Definition 7 Let $\mathcal{T}$ be an $\mathcal{EL}^{++}$ TBox. We define the *generalisation operator* γ inductively over the structure of concept descriptions as follows:

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

where γ_r and γ_C are defined as:

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

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

Definition 8 A finite sequence $C_1, \ldots, C_n$ of $\mathcal{EL}^{++}$ concepts is a *concept refinement path* $C_1 \xrightarrow{\gamma} C_n$ from C_1 to C_n of the generalisation operator γ iff $C_{i+1} \in \gamma(C_i)$ for all $i : 1 \leq i < n$. $\gamma^*(C)$ denotes the set of all concepts that can be reached from C by means of γ in zero or a finite number of steps.

Proposition 1 *The operator γ is a generalisation refinement operator over the set of all $\mathcal{EL}^{++}$ concepts with the order $\sqsubseteq$.*

Proof We need to prove that for every $\mathcal{EL}^{++}$ concept C and every $D \in \gamma(C)$, the subsumtion $C \sqsubseteq_{\mathcal{T}} D$ holds. We do this by induction on the structure of C. If C is a concept name, $\top$, or $\bot$, the subsumption holds directly by definition. If C is of the form $C_1 \sqcap C_2$, we can assume w.l.o.g. that D is $C' \sqcap C_2$ for some $C' \in \gamma(C_1)$. By induction hypothesis, $C_1 \sqsubseteq_{\mathcal{T}} C'$ and hence $C_1 \sqcap D \sqsubseteq_{\mathcal{T}} C' \sqcap D$. Finally, if C is of the form $\exists r.C_1$ we have three possible cases. If $\mathsf{UpCov}(r) \neq \emptyset$, and D is $\exists s.C_1$ for $s \in \mathsf{UpCov}(r)$ then by definition $\exists r.C_1 \sqsubseteq_{\mathcal{T}} \exists s.C_1$. If $\mathsf{UpCov}(C) \neq \emptyset$, $C \neq \top$ and D must be of the form $\exists r.C'$ with $C_1 \sqsubseteq_{\mathcal{T}} C'$, and hence the subsumption holds. In the last case, D is equivalent to $\top$, and hence the subsumption follows trivially. □

We now analyse the properties of the generalisation operator γ. Observe first that our definition of UpCov for basic concepts and roles only considers the set of subconcepts present in a TBox $\mathcal{T}$. This guarantees that γ is locally finite, since at each generalisation step, the set of possible generalisations is finite.

Proposition 2 *The generalisation refinement operator γ is locally finite.*

Proof We prove that for every $\mathcal{EL}^{++}$ concept C, $\gamma(C)$ is finite by induction on the structure of C.

For $A \in N_C \cup \{\top, \bot\}$, we have that $\gamma(A) \subseteq \mathsf{sub}(\mathcal{T})$. Since $\mathsf{sub}(\mathcal{T})$ is finite, the result immediately holds. For $C \sqcap D$, we have that $|\gamma(C \sqcap D)| \leq |\gamma(C)| + |\gamma(D)|$.

By induction hypothesis, the two sets on the right-hand side of this inequality are finite, and hence $\gamma(C \sqcap D)$ must be finite too. Finally, it holds that $|\gamma(\exists r.C)| \leq |\mathsf{UpCov}(r)| + |\gamma(C)|$. By the fact that $\mathsf{UpCov}(r) \subseteq N_R$, which is finite, and the induction hypothesis, the result follows. □

When generalising concept names and role names, we always ensure that the resulting concepts are more general (w.r.t. the TBox $\mathcal{T}$) than the original elements. Unfortunately, this does not guarantee that γ is proper.

Example 2 Let $\mathcal{T} := \{A \sqsubseteq B\}$. Then, following Definition 7, we have that generalising the concept $A \sqcap B$ yields $A \sqcap \top \in \gamma(A \sqcap B)$. However, both these concepts are equivalent to A w.r.t. $\mathcal{T}$. Therefore, γ is not proper.

One possible way to avoid this situation, and, therefore, to guarantee the properness of γ, is to redefine it with an additional semantic test. More precisely, let γ' be defined as:

$$\gamma'(C) := \gamma(C) \backslash \{D \in \gamma(C) \text{ such that } D \sqsubseteq_{\mathcal{T}} C\} \tag{5}$$

Essentially, γ' discards those generalisations that are equivalent to the concept being generalised. It is easy to see that γ' is still a finite generalisation operator and it is proper.

Proposition 3 *The generalisation refinement operator γ' is proper.*

Proof This proposition trivially follows from Eq. 5.

The repetitive application of the generalisation refinement operator allows one to find a description that represents the properties that two or more $\mathcal{EL}^{++}$ concepts have in common. This description is a common generalisation of $\mathcal{EL}^{++}$ concepts, the so-called generic space that is used in conceptual blending.

Definition 9 An $\mathcal{EL}^{++}$ concept description G is a generic space of the $\mathcal{EL}^{++}$ concept descriptions $C_1, \ldots, C_n$ if and only if $G \in \gamma'^*(C_i)$ for all $i = 1, \ldots, n$.

Example 3 Let us consider the $\mathcal{EL}^{++}$ concepts $\mathsf{EditDocument}$ and $\mathsf{SearchHardDisk}$ defined in Example 1. It can be checked that:

$$\{(\mathsf{Sign} \sqcap \exists \mathsf{hasMeaning.Action}) \sqcap \exists \mathsf{isInSpatialRelation.}(\mathsf{Sign} \sqcap \exists \mathsf{hasMeaning.ObjectType})\} \in \gamma'^*(\mathsf{EditDocument})$$
$$\{(\mathsf{Sign} \sqcap \exists \mathsf{hasMeaning.Action}) \sqcap \exists \mathsf{isInSpatialRelation.}(\mathsf{Sign} \sqcap \exists \mathsf{hasMeaning.ObjectType})\} \in \gamma'^*(\mathsf{SearchHardDisk})$$

$(\mathsf{Sign} \sqcap \exists \mathsf{hasMeaning.Action}) \sqcap \exists \mathsf{isInSpatialRelation.}(\mathsf{Sign} \sqcap \exists \mathsf{hasMeaning.ObjectType})$ is a generic space (Definition 9) of $\mathsf{EditDocument}$ and $\mathsf{SearchHardDisk}$.

Unfortunately, due to the fact that upward cover set we defined only takes subconcepts already present in the TBox into account, neither γ nor its refinement γ' are complete; that is, these operators may fail to compute some of the generalisations of a given $\mathcal{EL}^{++}$ concept.

Example 4 Let $\mathcal{T} := \{A \sqsubseteq B,\ A \sqsubseteq C\}$. Then, generalising the concept A yields $\gamma(A) = \{B, C\}$. However, $B \sqcap C$ is also a possible generalisation of A w.r.t. $\sqsubseteq_{\mathcal{T}}$.

More generally, as the following theorem shows, no generalisation refinement operator over $\mathcal{EL}^{++}$ concepts w.r.t. $\sqsubseteq_{\mathcal{T}}$ can be locally finite, proper, and complete.

Theorem 1 *There is no ideal generalisation refinement operator for $\mathcal{EL}^{++}$ concepts.*

Proof Consider the TBox $\mathcal{T} = \{A \sqsubseteq \exists r.A, \exists r.A \sqsubseteq A\}$, and define the concepts $G_0 := \top, G_{i+1} := \exists r.G_i$ for all $i \geq 0$. Notice first that these concepts form an infinite chain of generalisations $G_0 \sqsupset_{\mathcal{T}} G_1 \sqsupset_{\mathcal{T}} G_2 \sqsupset_{\mathcal{T}} \cdots \sqsupset_{\mathcal{T}} A$. Moreover, every $\mathcal{EL}^{++}$ concept C with $A \sqsubset_{\mathcal{T}} C$ is equivalent (w.r.t. $\mathcal{T}$) to one such G_i. Let now γ be a locally finite and proper generalisation refinement operator. Then $\gamma(A)$ is a finite set of concepts which, w.l.o.g. we can assume to be of the form $\{G_i \mid i \in I\}$, where I is a finite set of indices. In particular, I contains a maximum index n. Then G_{n+1} is strictly more specific than all elements of $\gamma(A)$ and cannot be derived by further applications of γ. Thus, γ is not complete. $\square$

Since the refinement operator is not complete, it cannot guarantee to find a generic space that is a *least* general generalisation. Although a least general generalisation, is desirable, finding a common description, which allows us creating new $\mathcal{EL}^{++}$ concepts from existing ones by conceptual blending, will suffice.

At this point, we should note, however, that the generalisation operator may even fail to find a generic space of a set of $\mathcal{EL}^{++}$ concepts. Indeed, as the following example shows, the generalisation operator can produce an infinite chain of generalisations.

Example 5 Let $\mathcal{T} := \{A \sqsubseteq \exists r.A, B \sqsubseteq \top\}$. Then, the generalisation of the concept description B can yield $\top$. The generalisation of the concept description A yields the infinite chain $\{\exists r.\exists r \ldots \exists r.A\}$. A common (trivial) generalisation for A, B is $\top$ but it is not found by γ.

Not finding a common generalisation of a set of $\mathcal{EL}^{++}$ concepts is a not a new problem in the DL literature. Different solutions have been proposed [1, 2, 5, 43, 44]. Typically, some assumptions are made over the structure of the TBox or a fixed role depth of concepts is considered. In the following, we adopt the latter view, and we restrict the number of nested quantifiers in a concept description to a fixed constant k. To this end, we introduce the definition of role depth of a concept as follows.

Definition 10 The role depth of an $\mathcal{EL}^{++}$ concept description C is defined as the maximum number of nested (existential) quantifiers in C:

$$\begin{aligned} \mathsf{roleDepth}(\top) &= \mathsf{roleDepth}(A) = 0, \\ \mathsf{roleDepth}(C \sqcap D) &= \max\{\mathsf{roleDepth}(C), \mathsf{roleDepth}(D)\}, \\ \mathsf{roleDepth}(\exists r.C) &= \mathsf{roleDepth}(\mathsf{C}) + 1 \end{aligned}$$

Based on the role depth of a concept we modify the definition of the generalisation operator γ' to take a fixed constant $k \in \mathbb{N}_{>0}$ of nested quantifiers into account. More precisely, let γ'_k be defined as γ', except that for the case of generalising a concept $\exists r.C$ we set:

$$\gamma'_k(\exists r.C) := \begin{cases} \gamma_r(\exists r.C) \cup \gamma_C(\exists r.C) & \text{if } (\mathsf{UpCov}(r) \neq \emptyset \text{ or } \mathsf{UpCov}(C) \neq \emptyset) \text{ and} \\ & \mathsf{roleDepth}(C) \leq k, \\ \{\top\} & \text{otherwise.} \end{cases}$$

The role depth prevents the generalisation operator from generating infinite chains of generalisations. Consequently, it can ensure that a generic space between $\mathcal{EL}^{++}$ concepts can always be found.

Predicates modeling $\mathcal{EL}^{++}$ concepts	**Description**
$dConcept(\mathcal{C})$	A reference to a domain knowledge concept $\mathcal{C}$
$concept(A)$	A concept A
$subConcept(A, B)$	A concept B subsumes A
$role(r)$	A role r
$subRole(r, s)$	A role r subsumes s
$hasConjunct(\mathcal{C}, ex, A, t)$	A concept A is an expression ex in $\mathcal{C}$ at step t
$hasRoleEx(\mathcal{C}, roleEx, r, depth, A, t)$	A concept A fills the role r in a role expression *roleEx* with depth *depth* in $\mathcal{C}$ at step t

Predicates modeling the refinement	**Description**
$notEqual(\mathcal{C}_1, \mathcal{C}_2, t)$	The domain concepts $\mathcal{C}_1$, $\mathcal{C}_2$ are not equivalent at step t
$conjunctNotEq(\mathcal{C}_1, \mathcal{C}_2, A, t)$	The concept A is not equivalent in $\mathcal{C}_1$ and $\mathcal{C}_2$ at step t
$hasRoleExNotEq(\mathcal{C}_1, \mathcal{C}_2, C, t)$	A conjunct C is not equivalent in the $\mathcal{C}_1$, $\mathcal{C}_2$ at step t
$roleInExpressionNotEq(\mathcal{C}_1, \mathcal{C}_2, C, r, t)$	A role r in a conjunct C is not equivalent in $\mathcal{C}_1$, $\mathcal{C}_2$ at step t
$app(a, \mathcal{C}, t)$	A refinement step a is applicable in $\mathcal{C}$ at step t
$poss(a, \mathcal{C}, t)$	A refinement step a is possible in $\mathcal{C}$ at step t
$exec(a, \mathcal{C}, t)$	A refinement step a is executed in $\mathcal{C}$ at step t

Table 2: Overview of the main predicates used to formalise the upward refinement process in ASP. The predicates in the top table are used to model $\mathcal{EL}^{++}$ concepts, whereas predicates in the table below are used to model the refinement operators.

Definition 11 An $\mathcal{EL}^{++}$ concept description G^k is a k-approximation of a generic space of the $\mathcal{EL}^{++}$ concept descriptions $C_1, \ldots, C_n$ if and only if $G^k \in \gamma_k'^*(C_i)$ for all $i = 1, \ldots, n$.

Proposition 4 *There always exists a k-approximation of a generic space G^k for any $\mathcal{EL}^{++}$ concept descriptions $C_1, \ldots, C_n$.*

Proof The proof of this proposition can be done by noticing that every concept can always be generalised to $\top$ in a finite number of applications of γ_k'. Therefore, $\top$ is always a generic space of any concept descriptions $C_1, \ldots, C_n$. □

The role depth not only avoids infinite chains of generalisations, but also provides a way to maintain the structure of the input concepts in conceptual blending. For instance, by choosing the value of k as the maximum role depth of the input concepts to be blended, the operator yields generalisations with a similar role structure.

5 Implementing Upward Refinement in ASP

We consider an $\mathcal{EL}^{++}$ TBox $\mathcal{T}$ that consists of a background knowledge $\mathcal{T}_{bk}$ and a domain knowledge $\mathcal{T}_{dk}$. A generic space between $\mathcal{EL}^{++}$ concepts in the domain knowledge is found by means of an ASP program that generalises $\mathcal{T}_{dk}$ in a step-wise transition process. Since finding a generic space of n concepts can be reduced to the problem of finding a generic space between pairs of concepts [3], the ASP program we devise takes two $\mathcal{EL}^{++}$ concepts into account.

In what follows, we describe how an $\mathcal{EL}^{++}$ TBox $\mathcal{T}$ is translated into an ASP representation needed for implementing the generic space search. Table 2 shows the main predicates used in the ASP implementation.[6]

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

For each concept name $A \in N_C$ in $\mathcal{T}_{bk}$, we state the fact:

$$concept(A) \tag{6}$$

For each role $r \in N_R$ in $\mathcal{T}_{bk}$ with $domain(r) \sqsubseteq C$ and $range(r) \sqsubseteq D$, we state the facts:

$$role(r) \tag{7a}$$
$$domain(r, C) \tag{7b}$$
$$range(r, D) \tag{7c}$$

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

$$subConcept(A, B) \tag{8}$$

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

$$subRole(r, s) \tag{9}$$

For each inclusion axiom $A \sqsubseteq C \in \mathcal{T}_{bk}$ 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 $\mathcal{C}$ within the following facts:

$$concept(\mathcal{C}) \tag{10a}$$
$$subConcept(A, \mathcal{C}) \tag{10b}$$

Then, $\mathcal{C}$ is translated to ASP facts by means of the following function:

$$\begin{aligned}
\mathsf{toASP}(\mathcal{C}, ex_{(k)}, C \sqcap D, depth) = \{&hasConjunct(\mathcal{C}, ex_{(k)}, subEx_{(k+1)}), \\
&hasConjunct(\mathcal{C}, ex_{(k)}, subEx_{(k+2)})\} \\
\cup \{&\mathsf{toASP}(\mathcal{C}, subEx_{(k+1)}, C, depth)\} \\
\cup \{&\mathsf{toASP}(\mathcal{C}, subEx_{(k+2)}, D, depth)\}
\end{aligned} \tag{11a}$$

$$\mathsf{toASP}(\mathcal{C}, ex_{(k)}, \top, depth) = \{hasConjunct(\mathcal{C}, ex_{(k)}, Thing)\} \tag{11b}$$

$$\mathsf{toASP}(\mathcal{C}, ex_{(k)}, B, depth) = \{hasConjunct(\mathcal{C}, ex_{(k)}, B)\} \tag{11c}$$

$$\begin{aligned}
\mathsf{toASP}(\mathcal{C}, ex_{(k)}, \exists r.B, depth) = \{&hasConjunct(\mathcal{C}, ex_{(k)}, roleEx_{(k)}), \\
&hasRoleEx(\mathcal{C}, roleEx_{(k)}, depth, r, B)\}
\end{aligned} \tag{11d}$$

$$\begin{aligned}
\mathsf{toASP}(\mathcal{C}, ex_{(k)}, \exists r.C, depth) = \{&hasConjunct(\mathcal{C}, ex_{(k)}, roleEx_{(k)}), \\
&hasRoleEx(\mathcal{C}, roleEx_{(k)}, depth, r, subEx_{(k+1)})\} \\
\cup \{&\mathsf{toASP}(\mathcal{C}, subEx_{(k+1)}, C, depth + 1)\}
\end{aligned} \tag{11e}$$

[6] Disjointness axioms are not translated to ASP because they are not used in the generalisation process.

toASP models a complex concept description as a set of *hasConjunct/3* and *hasRoleEx/5* predicates that are generated by recursively traversing its structure. $ex_{(k)}$ and $roleEx_{(k)}$ are atoms that are dynamically generated during the translation; k is a counter that let the predicates be identifiable in a unique way, and *depth* is used to count the depth of a role r. A conjunction in a concept description is modeled by means of *hasConjunct/3* predicates. For instance, if $\mathcal{C} = C \sqcap D$, then, the predicate $hasConjunct(\mathcal{C}, ex_{(1)}, subEx_{(3)})$ models that the complex concept $\mathcal{C}$ has a concept D as one of its conjunct (Eq. 11a). The translation of D is done by the recursive call $\mathsf{toASP}(\mathcal{C}, subEx_{(3)}, D, depth)$. Role expressions are modeled by means of a *hasConjunct/3* and *hasRoleEx/5* predicate. For instance, if we consider $D = \exists r.B$ then the role expression $\exists r.B$ is modeled by the predicates $hasConjunct(\mathcal{C}, subEx_{(2)}, roleEx_{(2)})$ and $hasRoleEx(\mathcal{C}, roleEx_{(2)}, 1, r, B)$. The former predicate states that the expression $subEx_{(2)}$ —referring to D— has a role expression $roleEx_{(2)}$. The latter predicate models that, in the the complex concept $\mathcal{C}$, the expression $roleEx_{(2)}$ has a concept B filling the role r, and that the depth of r is 1 (Eq. 11d). Cases 11b-11c-11e can be explained in a similar way.

While the background knowledge is static, the domain knowledge changes. To this end, we need to keep track of the generalisations applied to each domain concept. This is done by modeling a concept in the domain knowledge by means of the predicates *hasConjunct* and *hasRoleEx* with an extra atom, t, that is a step-counter representing the number of modifications made to the concept.

For each axiom $A \equiv C \in \mathcal{T}_{dk}$, in which A is a concept in the domain knowledge and C is its definition, we denote it by $\mathcal{C}$ and we add the following fact:

$$dConcept(\mathcal{C}) \tag{12}$$

Then, $\mathcal{C}$ is translated to ASP in the following way:

1. $\mathcal{C}$ is rewritten to $\mathcal{C}'$ by using all the axiom definitions in the background knowledge;
2. $\mathcal{C}$' is translated to ASP by means of the function toASP with the only difference that the predicates *hasConjunct* and *hasRoleEx* have an extra atom t, equal to 0.

To exemplify the translation process, we provide the following example.

Example 6 Let us consider the TBox and the domain concept EditDocument in Section 3. The background knowledge is translated to the following ASP facts:

Sign $\sqsubseteq$ Thing	$concept(Sign)$.	By Eq. 6
	$concept(Thing)$.	
	$subConcept(Document, Thing)$.	By Eq. 8
Document $\sqsubseteq$ Sign	$concept(Document)$.	By Eq. 6
	$subConcept(Document, Sign)$.	By Eq. 8
...	...	
$domain$(isAboveIn) $\sqsubseteq$ Sign	$role(isAboveIn)$.	By Eq. 7a
$range$(isAboveIn) $\sqsubseteq$ Sign	$domain(isAboveIn, Sign)$.	By Eq. 7b
	$range(isAboveIn, Sign)$.	By Eq. 7c
...	...	
isAboveIn $\sqsubseteq$ isInSpatialRelation	$subRole(isAboveIn, isInSpatialRelation)$.	By Eq. 9
...	...	

The concept EditDocument is translated to the following ASP facts:

Pen ⊓ ∃isRightIn.Document	
(Sign ⊓ ∃hasMeaning.Edit) ⊓ ∃isRightIn.(Sign ⊓ ∃hasMeaning.Doc)	
$dConcept(EditDocument)$.	By Eq. 12
$hasConjunct(EditDocument, ex_{(1)}, subEx_{(2)}, 0)$.	By Eq. 11a
$hasConjunct(EditDocument, ex_{(1)}, subEx_{(3)}, 0)$.	By Eq. 11a
$hasConjunct(EditDocument, subEx_{(2)}, subEx_{(3)}, 0)$.	By Eq. 11a
$hasConjunct(EditDocument, subEx_{(2)}, subEx_{(4)}, 0)$.	By Eq. 11a
$hasConjunct(EditDocument, subEx_{(3)}, Sign, 0)$.	By Eq. 11c
$hasConjunct(EditDocument, subEx_{(4)}, roleEx_{(4)}, 0)$.	By Eq. 11d
$hasRoleEx(EditDocument, roleEx_{(4)}, 1, hasMeaning, Edit, 0)$.	By Eq. 11d
$hasConjunct(EditDocument, subEx_{(3)}, roleEx_{(3)}, 0)$.	By Eq. 11e
$hasRoleEx(EditDocument, roleEx_{(3)}, 1, isRightOn, subEx_{(6)}, 0)$.	By Eq. 11e
$hasConjunct(EditDocument, subEx_{(4)}, subEx_{(5)}, 0)$.	By Eq. 11a
$hasConjunct(EditDocument, subEx_{(4)}, subEx_{(6)}, 0)$.	By Eq. 11a
$hasConjunct(EditDocument, subEx_{(5)}, Sign, 0)$.	By Eq. 11c
$hasConjunct(EditDocument, subEx_{(6)}, roleEx_{(6)}, 0)$.	By Eq. 11d
$hasRoleEx(EditDocument, roleEx_{(6)}, 1, hasMeaning, Doc, 0)$.	By Eq. 11d

Besides, we model the concept $\top$ as the fact *concept*(*Thing*), and for each concept name $A \in N_C$, which is not already subsumed by other concept names, we add a fact *subConcept*(A, *Thing*). We check for (in)equality of domain concepts $\mathcal{C}_1$ and $\mathcal{C}_2$ by a predicate *notEqual*($\mathcal{C}_1, \mathcal{C}_2, t$). The predicate is true whenever conjuncts, role expressions and roles are not equal in $\mathcal{C}_1$ and $\mathcal{C}_2$.

5.2 Formalising upward refinement in ASP

We consider each step of the refinement operator in Definition 7 as an operator type by itself. We consider five types of generalisation that can be applied to a concept in the domain knowledge at each step:

1. The generalisation of an atomic concept, and we denote it as γ_A;
2. The generalisation of a concept filling the range of a role up to a role depth k (γ_C);
3. The generalisation of a role (γ_r);
4. The removal of a role, and we denote it as γ_{r^-};
5. The removal of a concept, and we denote it as γ_{C^-}.

We treat each upward refinement operator type as 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 *app/3* that states when an operator type is *applicable*. Inertia is modelled with different non-inertial predicates that state when an element in a domain concept 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 an atom *exec*($\gamma_x, \mathcal{C}, t$). This atom denotes that a generalisation operator type $\gamma_x \in \{\gamma_A, \gamma_C, \gamma_r, \gamma_{r^-}, \gamma_{C^-}\}$ is applied to $\mathcal{C}$ at step t.

Upward refinement of atomic concepts. A fact *app*($genConcept(Ex, A, B), \mathcal{C}, t$) denotes the applicability of the generalisation of a concept A to a concept B in a

conjunct Ex of $\mathcal{C}$ at step t using γ_A:

$$\begin{aligned}
&\mathit{app}(\mathit{genConcept}(Ex, A, B), \mathcal{C}_1, t) \leftarrow \\
&\qquad \mathit{hasConjunct}(\mathcal{C}_1, Ex, A, t), \\
&\qquad \mathit{subConcept}(A, B), \\
&\qquad \text{not } \mathit{hasRoleEx}(\mathcal{C}_1, A, _, _, _, t), \\
&\qquad \text{not } \mathit{hasConjunct}(\mathcal{C}_1, A, _, t), \\
&\qquad \mathit{conjunctNotEq}(\mathcal{C}_1, \mathcal{C}_2, A, t), \\
&\qquad \text{not } \mathit{exec}(\mathit{genConcept}(Ex, A, B), \mathcal{C}_2, t), \mathit{dConcept}(\mathcal{C}_2)
\end{aligned} \tag{13}$$

There are several preconditions for generalising an atomic concept in a conjunct Ex. First, Ex involves a concept A that has a parent concept B in the subsumption hierarchy defined by the axioms of the TBox (first two EDB predicates). Second, Ex is neither a role expression nor a complex expression. Third, A is not equivalent in $\mathcal{C}_1$ and $\mathcal{C}_2$ (*conjunctNotEq/4*). This latter atom is true when either $\mathcal{C}_1$ or $\mathcal{C}_2$ does not contain A. Another condition is that A is not being generalised in $\mathcal{C}_2$, since we want to keep elements that are common in $\mathcal{C}_1$ and $\mathcal{C}_2$.
We also need a simple inertia rule for generalising a concept in a conjunct. This is as follows:

$$\begin{aligned}
\mathit{noninertialGenConcept}(\mathcal{C}, Ex, A, t) \leftarrow\ & \mathit{exec}(\mathit{genConcept}(Ex, A, _), \mathcal{C}, t), \\
& \mathit{hasConjunct}(\mathcal{C}, Ex, A, t)
\end{aligned} \tag{14}$$

noninertialGenConcept atoms will cause a concept A to remain in a conjunct Ex in $\mathcal{C}$, as defined via rule (23a).

Upward refinement of range concepts. A fact $\mathit{app}(\mathit{genConceptInRole}(Ex, r, A, B), \mathcal{C}, t)$ denotes the applicability of the generalisation of a concept A to a concept B when A fills the range of a role r in a role expression *RoleEx* of $\mathcal{C}$ at step t using γ_C:

$$\begin{aligned}
&\mathit{app}(\mathit{genConceptInRole}(\mathit{RoleEx}, r, A, B), \mathcal{C}_1, t) \leftarrow \\
&\qquad \mathit{hasRoleEx}(\mathcal{C}_1, \mathit{RoleEx}, \mathit{Depth}, r, A, t), \\
&\qquad \mathit{app}(\mathit{genConcept}(\mathit{RoleEx}, A, B), \mathcal{C}_1, t), \\
&\qquad \mathit{hasRoleExNotEq}(\mathcal{C}_1, \mathcal{C}_2, \mathit{RoleEx}, t), \mathit{Depth} \leq k, \\
&\qquad \text{not } \mathit{exec}(\mathit{genConceptInRole}(\mathit{RoleEx}, _, _, _), \mathcal{C}_2, t), \mathit{dConcept}(\mathcal{C}_2)
\end{aligned} \tag{15}$$

The preconditions for generalising a concept filling the role of a role expression *RoleEx* are similar to the case of the upward refinement of an atomic concept: *RoleEx* involves a concept A that is generalisable, the role expression is not equivalent in $\mathcal{C}_1$ and $\mathcal{C}_2$ (*hasRoleExNotEq/4*), and the concept to be generalised must not be under generalisation in $\mathcal{C}_2$. Please note how the maximum role depth of a concept k controls the applicability of this rule.
The inertia rule for generalising a concept that fills the range of a role in $\mathcal{C}$ is:

$$\begin{aligned}
&\mathit{noninertialGenConceptInRole}(\mathcal{C}, \mathit{RoleEx}, r, A, t) \leftarrow \\
&\qquad \mathit{exec}(\mathit{genConceptInRole}(\mathit{RoleEx}, r, A, _), \mathcal{C}, t), \\
&\qquad \mathit{hasRoleEx}(\mathcal{C}, \mathit{RoleEx}, _, r, A, t)
\end{aligned} \tag{16}$$

noninertialGenConceptInRole atoms will cause a concept A to remain in the range of a role as defined via rule (23b).

Upward refinement of roles. A fact $app(genRole(RoleEx, r, s), \mathcal{C}, t)$ denotes the applicability of the generalisation of a role r to a role s in a role expression $RoleEx$ of $\mathcal{C}$ at step t using γ_r:

$$\begin{aligned} app(genRole(RoleEx, r, s), \mathcal{C}_1, t) \leftarrow \\ & hasConjunct(\mathcal{C}_1, Ex, RoleEx, t), \\ & hasRoleEx(\mathcal{C}_1, RoleEx, _, r, A, t), \\ & subRole(r, s), \\ & roleInExpressionNotEq(\mathcal{C}_1, \mathcal{C}_2, RoleEx, r, t), \\ & \text{not } exec(genRole(RoleEx, r, _), \mathcal{C}_2, t), dConcept(\mathcal{C}_2) \end{aligned} \tag{17}$$

The main precondition for generalising a role r contained in a role expression $RoleEx$ is that r has a parent role s in the subsumption hierarchy defined by the axioms of the TBox. Other preconditions are that the role expression $RoleEx$ is not equivalent in $\mathcal{C}_1$ and $\mathcal{C}_2$ (*roleInExpressionNotEq/4*) and is not being generalised in $\mathcal{C}_2$.
The inertia rule for generalising a role in a role expression is:

$$\begin{aligned} noninertialGenRole(\mathcal{C}, RoleEx, r, t) \leftarrow\ & exec(genRole(RoleEx, r, _), \mathcal{C}, t), \\ & hasRoleEx(\mathcal{C}, RoleEx, _, r, A, t) \end{aligned} \tag{18}$$

noninertialGenRole atoms will cause a role r to remain in a role expression $RoleEx$ in $\mathcal{C}$, as defined via rule (23b).

Removal of a role. A fact $app(rmRole(RoleEx, r, A), \mathcal{C}, t)$ denotes the applicability of the removal of a role r from a role expression $RoleEx$ of $\mathcal{C}$ at step t using γ_{r^-}:

$$\begin{aligned} app(rmRole(RoleEx, r, A), \mathcal{C}_1, t) \leftarrow \\ & hasConjunct(\mathcal{C}_1, Ex, RoleEx, t), \\ & hasRoleEx(\mathcal{C}_1, RoleEx, _, r, A, t), \\ & \text{not } app(genRole(RoleEx, r, s), \mathcal{C}_1, t), \\ & \text{not } app(genConceptInRole(RoleEx, r, A, _), \mathcal{C}_1, t), \\ & hasRoleExNotEq(\mathcal{C}_1, \mathcal{C}_2, RoleEx, t), \\ & \text{not } exec(rmRole(RoleEx, r, _), \mathcal{C}_2, t), dConcept(\mathcal{C}_2) \end{aligned} \tag{19}$$

Essentially, a role r is removable from a role expression $RoleEx$ when neither itself nor the concept filling its range are generalisable. This is captured by the negated-by-failure predicates *app*/3. Other preconditions are that the role expression $RoleEx$ is not equivalent in $\mathcal{C}_1$ and $\mathcal{C}_2$ (*hasRoleExNotEq/4*) and is not being removed from $\mathcal{C}_2$.
The inertia rule for removing a role in a role expression is:

$$\begin{aligned} noninertialRmRole(\mathcal{C}, Ex, RoleEx, r, A, t) \leftarrow\ & exec(rmRole(RoleEx, r, A), \mathcal{C}, t), \\ & hasConjunct(\mathcal{C}, Ex, RoleEx, t), \\ & hasRoleEx(\mathcal{C}, RoleEx, _, r, A, t) \end{aligned} \tag{20}$$

noninertialRmRole atoms will cause a role r to remain in a role expression in $\mathcal{C}$, as defined via rules (23a-23b).

Removal of a concept. A fact $app(rmConcept(C, A), \mathcal{C}, t)$ denotes the applicability of the removal of a concept A from a conjunct Ex of $\mathcal{C}$ at step t using γ_{C^-}:

$$\begin{aligned} app(rmConcept(Ex, A), \mathcal{C}_1, t) \leftarrow \\ & hasConjunct(\mathcal{C}_1, Ex, A, t), \\ & \text{not } app(genConcept(Ex, A, _), \mathcal{C}_1, t), \\ & conjunctNotEq(\mathcal{C}_1, \mathcal{C}_2, A, t), \\ & \text{not } exec(rmConcept(Ex, A), \mathcal{C}_2, t), dConcept(\mathcal{C}_2) \end{aligned} \tag{21}$$

Essentially, a concept A is removable from a conjunct Ex when is not generalisable. This is captured by the negated-by-failure predicates *app*/3. Other preconditions are that the conjunct from where the concept will be removed is not equivalent in $\mathcal{C}_1$ and $\mathcal{C}_2$ (*conjunctNotEq/4*) and A is not being removed from $\mathcal{C}_2$.
The inertia rule for removing a concept is:

$$\begin{aligned} noninertialRmConcept(\mathcal{C}, Ex, A, t) \leftarrow & \; exec(rmConcept(Ex, A), \mathcal{C}, t), \\ & hasConjunct(\mathcal{C}, Ex, A, t) \end{aligned} \tag{22}$$

noninertialRmConcept atoms will cause a concept A to remain in a conjunct Ex in $\mathcal{C}$, as defined via rule (23a).

Inertia. The following rules state which concepts remain unchanged when they are inertial.

$$\begin{aligned} hasConjunct(\mathcal{C}, C, A, t+1) \leftarrow & \; hasConjunct(\mathcal{C}, C, A, t), \\ & \text{not } noninertialGenConcept(\mathcal{C}, C, A, t), \\ & \text{not } noninertialRmRole(\mathcal{C}, C, A, _, _, t), \\ & \text{not } noninertialRmConcept(\mathcal{C}, C, A, t) \end{aligned} \tag{23a}$$

$$\begin{aligned} hasRoleEx(\mathcal{C}, C, r, Depth, A, t+1) \leftarrow \\ & hasRoleEx(\mathcal{C}, C, r, Depth, A, t), \\ & \text{not } noninertialGenConceptInRole(\mathcal{C}, C, r, A, t), \\ & \text{not } noninertialGenRole(\mathcal{C}, C, r, A, t), \\ & \text{not } noninertialRmRole(\mathcal{C}, _, C, r, A, t) \end{aligned} \tag{23b}$$

Effects. Effect rules model how the knowledge changes when a concepts is generalised. The rule below shows an example of the effects of the generalisation of an atomic concept. Other two effect rules model the changes in the case of the generalisation of a role and of a concept in the range of a role.

$$\begin{aligned} hasConjunct(\mathcal{C}, C, B, t+1) \leftarrow \\ & hasConjunct(\mathcal{C}, C, A, t), \\ & exec(genConcept(C, A, B), \mathcal{C}, t) \end{aligned} \tag{24}$$

Additional rules handle the case in which the generalisation adds facts that model concept definitions (Eq. 11a-11e). In such a case, the number of roles *Depth* can be increased. To this end, the precondition $Depth \leq k$ in Eq. 15 prevents the applicability of further generalisations of a concept filling the range of a role when *Depth* reaches k, the maximum number of nested roles allowed.

Checking the equivalence between generalisations. As seen in the previous section, the upward refinement operator γ is proper when those generalisations, which are equivalent to the concept being generalised, are discarded (see Eq. 5). To this end, during the generic space search, we discard these generalisations. The *clingo* solver allows one to interleave the solving capabilities of ASP with a procedural language such as Python. This allowed us to check the equivalence between two generalisations in an external Python function and return the result to the ASP program. The rule below shows an example of how an external function *isGenEq* can be called from our ASP program.

$$\begin{aligned} &poss(genConcept(Ex, A, B), \mathcal{C}_1, t) \leftarrow \\ &\qquad app(genConcept(Ex, A, B), \mathcal{C}_1, t), \\ &\qquad EQ \neq 1, EQ = @isGenEq(`genConcept', \mathcal{C}, Ex, _, A, B, t) \end{aligned} \tag{25}$$

The *isGenEq* function internally does two things. First, it builds the concept description $\mathcal{C}$ based on the current generalisation. Since the incremental ASP solving process is controlled by a Python script, the Python function contains all the generalisations of a concept. Second, it checks whether the generalisation at step t is equivalent to the generalisation at step $t-1$. This is done by means of the *jcel* reasoner [35].[7] We test the equivalence between the current and the previous generalisation by checking the corresponding subsumptions. If the two generalisations are equivalent, then the function returns 1. In this case, the applicability of a generalisation operation is disabled by preventing the instantiation of the corresponding *poss*/3 predicate.

5.3 Upward refinement search

We use ASP for finding a generic space and the generalised versions of the concepts in the domain knowledge of an $\mathcal{EL}^{++}$ TBox $\mathcal{T}$, 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.

Given a concept description C in an $\mathcal{EL}^{++}$ TBox $\mathcal{T}$, the repetitive application of the generalisation operator types is a *refinement path*.

Definition 12 Let $\mathcal{C}$ be a domain concept in an $\mathcal{EL}^{++}$ TBox $\mathcal{T}$, let $\{\gamma_x^1, \ldots, \gamma_x^n\}$ be the set of generalisation steps for $\mathcal{C}$, $0 = t_1 < \cdots < t_n = n$ be refinement

[7] The *jcel* is a modular rule-based reasoner for description logics of the EL family implemented in Java. It uses a rule-based completion algorithm in which a set of completion rules are successively applied to saturate data structures that are used to model $\mathcal{EL}$ axioms. The algorithm is based on the CEL's algorithm [4] but is generalised with a change propagation approach. It implements reasoning tasks such as *classification*, *consistency*, *satisfability*, and *entailment*. The main advantage of the *jcel* reasoner is that these tasks are computable in polynomial time.'

steps and $\gamma_x \in \{\gamma_A, \gamma_C, \gamma_r, \gamma_{r^-}, \gamma_{C^-}\}$. The set of atoms $S = \{exec(\gamma_x^1, \mathcal{C}, t_1), \cdots, exec(\gamma_x^n, \mathcal{C}, t_n)\}$ is a refinement path of $\mathcal{C}$. A refinement path of $\mathcal{C}$ leads to the generalised concept $\mathcal{C}^n = \gamma_x^n(\cdots \gamma_x^2(\gamma_x^1(\mathcal{C})))$. We write $\mathcal{C}^j$ $(1 \leq j \leq n)$ to denote the concept $\mathcal{C}$ after j generalisation steps.

Refinement paths are generated by means of a choice rule, that allows one or zero refinement operators per $\mathcal{C}$ at each step t. The only generalisations that are executed are those whose preconditions are satisfied. Refinement paths lead from the domain concepts to a generic space. A generic space is reached, if the generalised domain concepts are equal. A constraint 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.

We should note at this point that the ASP implementation is sound and complete w.r.t. the upward refinement operator γ'^*. So, given two $\mathcal{EL}^{++}$ concepts C_1 and C_2, each stable model of the logic program P —encoding the generic space search and the two concepts— contains the refinement paths S_1 and S_2 through which C_1 and C_2 can be generalised to a concept G that is a generic space according to Definition 9. Proving this result can be done by induction over the structure of toASP and P, similar to the proof in [16, Appendix B]. On the other hand, if two concepts has a generic space G by applying γ'^*, this generic space is found by P, thus, the implementation is complete. However, the ASP implementation is neither sound nor complete w.r.t. the $\mathcal{EL}^{++}$ semantics ($\sqsubseteq_{\mathcal{T}}$) since the operator is not complete (see Theorem 1).

Example 7 Let us consider the SearchHardDisk and EditDocument concepts in Example 1 representing icons in the domain knowledge of the *ComputerIcon* ontology. Their refinement paths are:

$$
\begin{aligned}
S_{SearchHardDisk} = \{&exec(genConceptInRole(roleEx_{(6)}, hasMeaning, HardDrive,\\
&\qquad ObjectType), SearchHardDisk, 0),\\
&exec(genRole(roleEx_{(3)}, isAboveIn, isInSpatialRelation, subEx_{(4)}),\\
&\qquad SearchHardDisk, 1),\\
&exec(genConceptInRole(roleEx_{(4)}, hasMeaning, Search, Action),\\
&\qquad SearchHardDisk, 2)\}\\
S_{EditDocument} = \{&exec(genConceptInRole(roleEx_{(4)}, hasMeaning, Edit, Action),\\
&\qquad EditDocument, 0),\\
&exec(genRole(roleEx_{(3)}, isAboveInRight, isInSpatialRelation, subEx_{(4)}),\\
&\qquad EditDocument, 1),\\
&exec(genConceptInRole(roleEx_{(6)}, hasMeaning, Doc, ObjectType),\\
&\qquad EditDocument, 2)\}
\end{aligned}
$$

The refinement paths are parsed in order to translate the ASP encoding back to $\mathcal{EL}^{++}$ and apply the corresponding generalisation operators. The application of the refinement paths to the input concepts lead to the generalised concepts and to their a generic space. It is easy to check that this corresponds to the generic space in Example 3.

Algorithm 1 Conceptual blending of $\mathcal{EL}^{++}$ concepts

Input: $\begin{cases} \text{An } \mathcal{EL}^{++} \text{ TBox } \mathcal{T} \\ \text{Two domain concepts } C_1 \text{ and } C_2 \\ \text{A consequence requirement } CR \\ \text{A maximum role depth } k \end{cases}$

Output: A ranked list of blended concepts $\mathcal{B}$

{$\langle \mathcal{C}'_1, \mathcal{C}'_2 \rangle$ denotes a set of generalisations for C_1 and C_2 that lead to a generic space.}

1: **for all** $\langle \mathcal{C}'_1, \mathcal{C}'_2 \rangle \leftarrow \mathit{generalise}(C_1, C_2, k)$ **do**
2:     **for** $C'_1 \in \mathcal{C}'_1$ **do**
3:         **for** $C'_2 \in \mathcal{C}'_2$ **do**
4:             $C_{am} \leftarrow MGS(C'_1, C'_2)$
5:             **if** $C_{am} \notin \mathcal{B}$ and $\{\mathcal{T} \cup C_{am}\}$ entails CR **then**
6:                 $C'_{am} \leftarrow \mathit{completion}(C_{am})$
7:                 $\mathit{rankBlend}(C'_{am}, \mathit{compactness}(C'_{am}), \mathcal{B})$
8:             **end if**
9:         **end for**
10:     **end for**
11: **end for**
12: **return** $\mathcal{B}$

The output the generalisation search is then passed to a blending algorithm in order to create and evaluate $\mathcal{EL}^{++}$ blended concepts, as described next.

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

Conceptual blending by Fauconnier and Turner [20] is a cognitive theory that explains human creativity. According to this theory, humans create through a mental process that takes different mental spaces as input and combines them into a new mental space, called a *blend*. A blend is constructed by taking the commonalities among the input mental spaces into account, to form a so-called *generic space*, and by projecting the non-common structure of the input spaces in a selective way to the novel blended space. Since this theory focuses on the cognitive aspects of human creation, it is not a computational framework. It needs to be re-interpreted in a computational way, when one wants to use it in computational creativity.

In working towards this objective, we have characterised mental spaces in terms of $\mathcal{EL}^{++}$ concept descriptions and we have devised a generalisation algorithm to find the generic space between $\mathcal{EL}^{++}$ concepts. In this section, we provide an algorithm to find $\mathcal{EL}^{++}$ blended concepts. From a cognitive point view, conceptual blending involves the following aspects:

1. blend *generation*: it takes the generic space of two input spaces into account and combines their non-common structure in a selective way to a novel blended space;
2. blend *completion*: it constructs the *emergent structure* of a blend —a structure that is not directly copied from the inputs— by taking some background knowledge into account;
3. blend *evaluation*: it assesses the quality of a blend by means of certain optimality principles.

Our algorithm for blending $\mathcal{EL}^{++}$ concepts (Algorithm 1) implements these aspects as three phases, and re-interprets them in order to provide a computational account for conceptual blending. The implementation of the conceptual blending algorithm is available at: `https://bitbucket.org/rconfalonieri/ontolp-implementation`.

The blend generation is implemented according to the definition of an amalgam (Definition 4). To this end, first, a generic space is found by means of the ASP-based generalisation process described in the previous section. The method *generalise* finds different refinement paths of two (domain) $\mathcal{EL}^{++}$ concepts that lead to a generic space (Line 1). Then, a blend is created by computing the most general specialisation (MGS) of a pair of generalised concepts (Line 4). The MGS of two $\mathcal{EL}^{++}$ concepts corresponds to their conjunction.

Due to this combinational way of generating the blends, some of them might have already been found using some previous refinement paths, and they are simply not considered. Some other blends, on the other hand, may be not interesting. For instance, they might not have certain desirable properties.

In the algorithm, blend evaluation consists of two parts: a logical check and a heuristic function (Line 5 and 7).[8] The logical check discards those blends that do not satisfy certain properties. Desirable properties are modeled as an ontology consequence requirement CR that is given as input to the algorithm. For instance, a consequence requirement can ask that a blend should contain certain concepts and roles. For our purposes, it can require that a blended concepts contains a sign with meaning *search* above a sign with meaning *document*, which can be modeled in $\mathcal{EL}^{++}$ as $\mathsf{Sign} \sqcap \exists \mathsf{hasMeaning.Search} \sqcap \exists \mathsf{isAboveIn.Sign} \sqcap \exists \mathsf{hasMeaning.Doc}$, To verify whether a consequence requirement is satisfied or not, the algorithm makes use of the *jcel* reasoner. Consequence satisfaction is achieved by checking whether the ontology in the TBox $\mathcal{T}$ and the new blended concept entail the consequence requirement (Line 5).[9]

Then, those blends that satisfy the consequence requirement are *completed*. In conceptual blending, completion refers to the "*background knowledge that one brings into a blend*" [20]. Clearly, in a computational setting, there can be different interpretations of what background knowledge stands for. In our implementation, we interpreted it as structural properties that a blend should have. In blending computer icons, we expect new blended icon concepts to be defined by one spatial relation between signs in which each sign has only one meaning relation.[10] To this end, completion is an operation that transforms the structure of a blend by taking this background knowledge into account. In particular, completion consists of a set of transformation rules that aggregate roles and concepts by taking the axioms in the TBox into account.

[8] Blend evaluation is an open research topic in conceptual blending and it can be accomplished in different ways. For instance, evaluation could be achieved through an argumentative dialogue, in which users engage in order to decide which blend to keep and which one to discard. We refer the interested reader to [12] where a discussion about the use of Lakatosian reasoning to evaluate conceptual blends is presented.

[9] Consequence satisfaction can be checked by means of the 'entailment' option of the *jcel* reasoner as: `java -jar jcel.jar entailment ontology.owl --conclusion=conclusion.owl`, where `ontology.owl` is the ontology extended with the definition of a new blended concept Blend and `conclusion.owl` contains an axiom of the form $\mathsf{Blend} \sqsubseteq \mathsf{CR}$.

[10] This constraint is not expressable in $\mathcal{EL}^{++}$. We re-interpreted it as the background knowledge used to complete blended concepts.

To implement completion, we specified a simple rewriting system using Maude [11], a system that supports rewriting logic specification and programming. The transformation rules that we used for completing a blend are:

$$A \sqcap B \text{ is transformed to } A \text{ if } A \sqsubseteq_{\mathcal{T}} B \quad (26a)$$

$$\exists r.C \sqcap \exists s.C \text{ is transformed to } \exists r.(C \sqcap D) \text{ if } r \sqsubseteq_{\mathcal{T}} s \quad (26b)$$

Besides, we make use of concept definitions — equivalence axioms in the TBox— to rewrite a blend into a shorter equivalent form. It is worthy to notice that whilst this last rewriting preserves concept equivalence —therefore, it can be considered a simple instance of DL rewriting [3]— the above rules do not. Indeed, the rule in Eq. 26b is not invariant w.r.t. $\mathcal{EL}^{++}$ semantic equivalence, since it transforms a concept into a more specific one. Blends are completed before a heuristic function is applied (Line 6).

To decide which blends are better than others, the algorithm ranks them by means of a heuristic function (Line 7). The *compactness* heuristic counts the number of concepts and roles used in the definition of a blend B:

$$compactness(B) = \frac{1}{conceptsNr(B) + rolesNr(B)} \quad (27)$$

The algorithm considers as best blends those that have a higher compactness value. This heuristic can be considered as a computational interpretation of some of the optimality principles proposed by Fauconnier and Turner [20]. The *integration principle*, for instance, states that "*a blend must constitute a tightly integrated scene that can be manipulated as a unit*". The compactness of a blend captures the idea behind this principle in the sense that minimises the number of concepts and roles that are used to define a blend.

Example 8 Let us consider $C_1 = \mathsf{SearchHardDisk}$, $C_2 = \mathsf{EditDocument}$, G in Example 3 and the generalisation steps in $P_{SearchHardDisk}$ and in $P_{EditDocument}$ of Example 7. Given a consequence requirement expressing that a blended concept should contain a sign with meaning *search* above a sign with meaning *document*, modeled in $\mathcal{EL}^{++}$ as $\mathsf{Sign} \sqcap \exists \mathsf{hasMeaning.Search} \sqcap \exists \mathsf{isAboveIn.Sign} \sqcap \exists \mathsf{hasMeaning.Doc}$, and the maximum role depth $k = 2$, the algorithm returns the following ranked blends:[11]

Blend	Compactness
$MGS(C_1^1, C_2^2)$	0,33
$MGS(C_1, C_2^1)$	0,2
$MGS(C_1, C_2^2)$	0,16
$MGS(C_1^1, C_2^1)$	0,14
$MGS(C_1, C_2)$	0,13
$MGS(C_1^1, C_2)$	0,1

Each blend is obtained by combining different generalisations of the input concepts C_1 and C_2. The concepts C_1^1 and C_2^2 correspond to the generalised concepts 'GenConcept1' and 'GenConcept2' in Figure 3 respectively. They are obtained

[11] Recalling Definition 12, in the table, C_i^j stands for 'j generalisations have been applied to the concept i'. When j is omitted, C_i denotes the input concept i with no generalisations.

by applying the generalisations steps from Example 7. C_1^1 is obtained from C_1 in one generalisation step by generalising the concept HardDrive (filling the role hasMeaning) to ObjectType. C_2^2 is obtained from C_2 in two generalisation steps by generalising the concept Edit (filling the role hasMeaning) to Action and the role isAboveRightIn to isInSpatialRelation. $MGS(C_1^1, C_2^2)$ is completed and elaborated into MagnifyingGlass $\sqcap$isAbove.Document and its compactness value is 0,33. $MGS(C_1^1, C_2^2)$ is the best blend found by the algorithm. Other valid but less ranked blends are obtained by other combinations of generalised concepts.

7 Related Work

Conceptual blending in $\mathcal{EL}^{++}$ as described in this paper is a special case of the amalgam-based concept blending model described in [9], and implemented for CASL theories in [18] in order to invent cadences and chord progressions. This model has also been used to study the role of blending in mathematical invention [10]. This concept blending model, as the one presented here, is based on the notion of amalgam defined over a space of generalisations [36]. The space of generalisations is defined by refinement operators, that can be specialisation operators or generalisation operators, notions developed by the Inductive Logic Programming (ILP) community for inductive learning. These notions can be specified in any language where refinement operators define a generalisation space like ILP [29], description logics [39], or order-sorted feature terms [37].

Several approaches for generalising ontology concepts in the $\mathcal{EL}$ family exist in the DL 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 [1, 2, 5, 43, 44]. Generally speaking, since the LGG w.r.t. general TBoxes in the $\mathcal{EL}$ family does usually not exist, these approaches propose several solutions for computing it. For instance, Baader [1, 2] devises the exact conditions for the existence of the LGG for cyclic $\mathcal{EL}$-TBoxes based on graph-theoretic generalisations. Baader et al [5] propose an algorithm for computing good LGGs w.r.t. a background terminology. Turhan and Zarrieß [43], Zarrieß and Turhan [44] 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.

An approach in DL that does use refinement operators is [39], where the language chosen for representing the generalisation space is that of DL Conjunctive Queries. Here LGG between two inputs, translated to conjunctive queries, can be determined by searching over the generalisation space using downward specialisation operators.

On the other hand, studying the LGG in terms of generalisation and specialisation refinement operators has been used for order sorted-feature terms and Horn clauses in ILP. Anti-unification (or LGG) in order sorted-feature terms was studied in [37], which was conducive to later develop the notion of amalgam [36]. The notion of refinement operator, that originated in ILP, has been more studied in the space of Horn clauses [29], but LGG in particular has not been a topic intensively pursued in the context of inductive learning in ILP.

Finally, other approaches that combine ASP for reasoning over DL ontologies worthy to be mentioned are [15, 38, 41].

8 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. We showed that the operator is locally finite, proper, but it is not complete (Propositions 2-3 and Theorem 1). 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 an implementation of the refinement operator in ASP. We showed how to model the description of $\mathcal{EL}^{++}$ concepts in ASP and to implement a search process for generalising the domain knowledge of an $\mathcal{EL}^{++}$ 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 embedded the ASP-based search process in an amalgamation process to implement an algorithm for conceptual blending. The algorithm creates new $\mathcal{EL}^{++}$ concepts by combining pair of generalised $\mathcal{EL}^{++}$ concepts. The blends are logically evaluated and ranked by means of ontology consequence requirements and a heuristic function respectively. We exemplified our approach in the domain of computer icon design.

We envision some directions of future research. We aim at employing a richer DL, such as $\mathcal{SROIQ}$ [26] —the DL underlying the Web Ontology Language OWL 2[12]—, in our conceptual blending framework. This will allow us to capture more complex concept descriptions and consequence requirements. By doing this, however, we will have to sacrifice efficiency, since the reasoning tasks in this logic are computational more expensive than in $\mathcal{EL}^{++}$. A possible way to find a tradeoff between expressivity and efficiency is to employ a richer DL only in one of the phases of our conceptual blending framework, e.g., either in the generation or in the evaluation phase. For instance, $\mathcal{SROIQ}$ could be employed in the generation phase (to this end, we will need to extend the generalisation operator), while the blend evaluation could be realised through argumentation [12]. On the contrary, we can keep $\mathcal{EL}^{++}$ in the generation phase and use $\mathcal{SROIQ}$ in the evaluation. These options are perfectly justifiable from the conceptual blending point of view, since the blend generation and evaluation are separate processes that can use different languages and techniques. This is also what usually happens in data mining approaches to computational creativity [42].

Another extension of the framework that we wish to explore is the blending of ontologies rather only concepts. Blending ontologies has already been explored in an ontological blending framework [25, 28], where blends are computed as *colimits* of algebraic specifications. In this framework, the blending process is not characterised in terms of amalgamation, the input concepts are not generalised, and the

[12] `http://www.w3.org/TR/owl2-overview/`, accessed 04/12/2015

generic space is assumed to be given. Therefore, the results of this paper can be extended and applied in this framework.

We consider the work of this paper to be a fundamental step towards the challenging task of defining and implementing a computational framework for conceptual blending which uses DLs as its formal underpinning language.

Acknowledgements The authors wish to express their thanks to the reviewers of this paper for their helpful comments. The research presented in this article was partially supported by the COINVENT project (FET-Open grant number: 611553).

References

1. 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, Lex W (eds) Conceptual Structures for Knowledge Creation and Communication, Lecture Notes in Computer Science, vol 2746, Springer Berlin Heidelberg, pp 117–130
2. 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, Westfechtel B (eds) Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, vol 3353, Springer Berlin Heidelberg, pp 177–188
3. Baader F, Küsters R (2006) Non-standard Inferences in Description Logics: The Story So Far. In: Gabbay DM, Goncharov SS, Zakharyaschev M (eds) Mathematical Problems from Applied Logic I, International Mathematical Series, vol 4, Springer New York, pp 1–75
4. Baader F, Brandt S, Lutz C (2005) Pushing the EL Envelope. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 364–369
5. Baader F, Sertkaya B, Turhan AY (2007) Computing the least common subsumer w.r.t. a background terminology. Journal of Applied Logic 5(3):392 – 420
6. Baader F, Brandt S, Lutz C (2008) Pushing the EL Envelope Further. In: Clark K, Patel-Schneider PF (eds) In Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions
7. Baral C (2003) Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press
8. Besold TR, Plaza E (2015) Generalize and Blend: Concept Blending Based on Generalization, Analogy, and Amalgams. In: Proceedings of the 6th International Conference on Computational Creativity, ICCC15
9. Bou F, Eppe M, Plaza E, Schorlemmer M (2014) D2.1: Reasoning with Amalgams. Tech. rep., COINVENT Project, available at `http://www.coinvent-project.eu/fileadmin/publications/D2.1.pdf`
10. Bou F, Schorlemmer M, Corneli J, Gomez-Ramirez D, Maclean E, Smail A, Pease A (2015) The role of blending in mathematical invention. In: Proceedings of the 6th International Conference on Computational Creativity, ICCC15
11. Clavel M, Durán F, Eker S, Lincoln P, Martí-Oliet N, Meseguer J, Talcott C (2003) The Maude 2.0 System. In: Nieuwenhuis R (ed) Rewriting Techniques

and Applications (RTA 2003), Springer-Verlag, no. 2706 in Lecture Notes in Computer Science, pp 76–87

12. Confalonieri R, Corneli J, Pease A, Plaza E, Schorlemmer M (2015) Using Argumentation to Evaluate Concept Blends in Combinatorial Creativity. In: Proceedings of the 6th International Conference on Computational Creativity, ICCC15
13. Confalonieri R, Eppe M, Schorlemmer M, Kutz O, Peñaloza R, Plaza E (2015) Upward Refinement for Conceptual Blending in Description Logic —An ASP-based Approach and Case Study in $\mathcal{EL}^{++}$. In: Proceedings of 1st International workshop of Ontologies and Logic Programming for Query Answering, ONTOLP 2015, co-located with IJCAI-2015
14. Cornet R, de Keizer N (2008) Forty years of SNOMED: a literature review. BMC medical informatics and decision making 8 Suppl 1
15. Eiter T, Ianni G, Lukasiewicz T, Schindlauer R, Tompits H (2008) Combining answer set programming with description logics for the semantic web. Artificial Intelligence 172(12–13):1495–1539
16. Eppe M, Bhatt M (2015) Approximate Postdictive Reasoning with Answer Set Programming. Journal of Applied Logic 13(4, Part 3):676–719
17. Eppe M, Bhatt M, Dylla F (2013) Approximate epistemic planning with postdiction as answer-set programming. In: Cabalar P, Son TC (eds) Logic Programming and Nonmonotonic Reasoning: 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 290–303
18. Eppe M, Confalonieri R, Maclean E, Kaliakatsos-Papakostas MA, Cambouropoulos E, Schorlemmer WM, Codescu M, Kühnberger K (2015) Computational Invention of Cadences and Chord Progressions by Conceptual Chord-Blending. In: Yang Q, Wooldridge M (eds) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, AAAI Press, pp 2445–2451
19. Eppe M, Maclean E, Confalonieri R, Kutz O, Schorlemmer WM, Plaza E (2015) ASP, Amalgamation, and the Conceptual Blending Workflow. In: Calimeri F, Ianni G, Truszczynski M (eds) Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings, pp 309–316
20. Fauconnier G, Turner M (2002) The Way We Think: Conceptual Blending And The Mind's Hidden Complexities. Basic Books
21. Gebser M, Kaminski R, Kaufmann B, Schaub T (2014) Clingo = ASP + control: Preliminary report. CoRR abs/1405.3694
22. Gebser M, Kaminski R, Kaufmann B, Lindauer M, Ostrowski M, Romero J, Schaub T, Thiele S (2015) Potassco User Guide 2.0. Tech. rep., University of Potsdam
23. Gelfond M, Kahl Y (2014) Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach. Cambridge University Press, New York, NY, USA
24. Gelfond M, Lifschitz V (1988) The stable model semantics for logic programming. In: Proceedings of the Fifth International Conference on Logic Programming, (ICLP'88), The MIT Press, pp 1070–1080
25. Hois J, Kutz O, Mossakowski T, Bateman J (2010) Towards ontological blending. In: Dicheva D, Dochev D (eds) Artificial Intelligence: Methodology, Sys-

tems, and Applications, Lecture Notes in Computer Science, vol 6304, Springer Berlin Heidelberg, pp 263–264
26. Horrocks I, Kutz O, Sattler U (2006) The Even More Irresistible SROIQ. In: Doherty P, Mylopoulos J, Welty CA (eds) Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006, AAAI Press, pp 57–67
27. Kowalski R (1974) Predicate Logic as Programming Language. In: Proceedings of International Federation for Information Processing, pp 569– 574
28. Kutz O, Bateman J, Neuhaus F, Mossakowski T, 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
29. van der Laag PR, Nienhuys-Cheng SH (1998) Completeness and properness of refinement operators in inductive logic programming. The Journal of Logic Programming 34(3):201 – 225
30. Lee J, Palla R (2012) Reformulating the Situation Calculus and the Event Calculus in the General Theory of Stable Models and in Answer Set Programming. Journal of Artificial Intelligence Research 43:571–620
31. Lehmann J, Haase C (2010) Ideal Downward Refinement in the EL Description Logic. In: Proc. of the 19th Int. Conf. on Inductive Logic Programming, Springer-Verlag, Berlin, Heidelberg, ILP'09, pp 73–87
32. Lehmann J, Hitzler P (2010) Concept learning in description logics using refinement operators. Machine Learning 78(1-2):203–250
33. Ma J, Miller R, Morgenstern L, Patkos T (2013) An Epistemic Event Calculus for ASP-based Reasoning About Knowledge of the Past, Present and Future. In: International Conference on Logic for Programming, Artificial Intelligence and Reasoning
34. McCarthy J (1986) Applications of circumscription to formalizing commonsense knowledge. Artificial Intelligence 28(1):89–116
35. Mendez J (2012) jcel: A Modular Rule-based Reasoner. In Proceedings of the 1st International Workshop on OWL Reasoner Evaluation (ORE 2012) 858
36. Ontañón S, Plaza E (2010) Amalgams: A Formal Approach for Combining Multiple Case Solutions. In: Bichindaritz I, Montani S (eds) Proceedings of the International Conference on Case Base Reasoning, Springer, Lecture Notes in Computer Science, vol 6176, pp 257–271
37. Ontañón S, Plaza E (2012) Similarity measures over refinement graphs. Machine Learning Journal 87(1):57–92
38. Ricca F, Gallucci L, Schindlauer R, Dell'Armi T, Grasso G, Leone N (2009) OntoDLV: An ASP-based System for Enterprise Ontologies. Journal of Logic and Computation 19(4):643–670
39. Sánchez-Ruiz A, Ontañón S, González-Calero P, Plaza E (2013) Refinement-Based Similarity Measure over DL Conjunctive Queries. In: Delany S, Ontañón S (eds) Case-Based Reasoning Research and Development, Lecture Notes in Computer Science, vol 7969, Springer Berlin, pp 270–284
40. Spackman K, Campbell K, Cote R (1997) SNOMED RT: A reference terminology for health care. Journal of the American Medical Informatics Association
41. Swift T (2004) Deduction in Ontologies via ASP. In: Lifschitz V, Niemelä I (eds) Logic Programming and Nonmonotonic Reasoning, Lecture Notes in Computer Science, vol 2923, Springer Berlin, pp 275–288

42. Toivonen H, Gross O (2015) Data mining and machine learning in computational creativity. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 5(6):265–275
43. Turhan A, Zarrieß B (2013) Computing the lcs w.r.t. general $\mathcal{EL}^+$-TBoxes. In: Proceedings of the 26th International Workshop on Description Logics, pp 477–488
44. Zarrieß B, Turhan AY (2013) Most Specific Generalizations w.r.t. General $\mathcal{EL}$-TBoxes. In: Proceedings of the 23th International Joint Conference on Artificial Intelligence, AAAI Press, IJCAI '13, pp 1191–1197