Can you imagine... a language for combinatorial creativity?

Fabian M. Suchanek[1], Colette Menard[2], Meghyn Bienvenu[3], Cyril Chapellier

[1] Télécom ParisTech, [2] STIM, [3] LIRMM Montpellier; France

Abstract. Combinatorial creativity combines existing concepts in a novel way in order to produce new concepts. For example, we can imagine jewelry that measures blood pressure. For this, we would combine the concept of jewelry with the capabilities of medical devices. In this paper, we concentrate on creating new concepts in the description logic $\mathcal{EL}$. We propose a novel language to this effect, and study its properties and complexity. We show that our language can be used to model existing inventions and (to a limited degree) to generate new concepts.

1 Introduction

What if cars had wings? What if tables could serve as beds? What if spoons could talk? These questions may seem completely absurd. And yet, the following questions are much less absurd: What if phones could go online? What if cars could be used to sleep in them? What if tap water contained medicine? Much of what may seem absurd today may become reality in the future. The field that is concerned with combining components of existing concepts into new concepts is called *combinatorial creativity*[1]. This field serves different purposes. Most prominently, it serves to describe new inventions: a smartphone is a telephone that is connected to the Internet; a Segway is a vehicle with two wheels on the same axis; a Hyperloop is a train without wheels. But combinatorial creativity can also serve to develop new business ideas, to find plots for books or movies, to understand human creativity, to disrupt conventional assumptions, to find design alternatives, or to foster thinking outside the box.

A prominent current of research uses description logics (DLs) to model real-world concepts [11, 2], so it is natural to try to use DLs to capture the types of concept modification underlying combinatorial creativity. Suppose for example that cars are defined using the following (simplified) DL axiom:

$$Car \equiv Vehicle \sqcap \exists hasPart.Wheel$$

Now suppose that we wish to consider the concept obtained by *removing the wheels from car*. Any construction of the form $Car \sqcap \neg\exists hasPart.Wheel$ simply

[1] A more constrained subfield of computational creativity in general.

leads to a contradiction, i.e., the empty concept $\bot$. Thus, standard DL constructors do not provide any direct means of expressing modifications like taking the concept Car and removing the property $\exists hasPart.Wheel$ from it.

Much work has been done on conceptual blending [4, 7, 26, 11, 1, 20, 14, 6], in which two concepts from different thematic areas are blended to create a new concept. However, blending is concerned more with describing analogies and metaphors than with describing modifications of concepts. Blending can, e.g., explain how a human understands an expression such as "sign forest", but it cannot express an atomic operation such as removing the wheels from a car. Non-standard reasoning services for DLs, like semantic matchmaking, consider the problem of modifying concepts to achieve certain objectives, but do not provide a means of expressing explicit updates of concepts. What we would want is a language that allows writing *Remove the wheels from the car*, or: Car *"minus"* $\exists hasPart.Wheel$.

In this paper, we propose a formal language for concept modifications that can serve as a basis for combinatorial creativity. More precisely:

- We define a language that allows modeling the transition from one concept to another one explicitly – by adding, removing, or modifying its constituents.
- We explain the design rationale for our operators, discuss design alternatives, and prove their formal properties.
- We show that our language can be used to describe real-world inventions, and (in a limited manner) to *generate* new concepts.

This paper is structured as follows. We start with a discussion of related work in Section 2 and give the preliminaries in Section 3. The main part of our paper, Section 4, describes our language. Section 5 shows concrete applications of our language. We conclude in Section 6.

2 Related Work

Cognitive Sciences. Combinatorial creativity has first been studied in the cognitive sciences [5, 8, 24]. These analyses center on understanding human cognition and have not led to a formal theory based in logic. The COINVENT project [21] aims to develop a computationally feasible, cognitively-inspired, formal model of concept invention. The project, however, has started only recently, and the model is still in the process of development. Fictional ideation generates new concepts for narratives [16]. In a larger sense, computational creativity is concerned also with creative human-computer interaction, art, figurative language, humor, music, argumentation, generating narratives or poetry, and scientific discovery [25]. We concentrate here on works that come closest to a formal language for describing modifications of concepts.

Conceptual Blending. One of the areas that cognitive science investigates is amalgams and analogies [4]. An *amalgam* of two input concepts is any new concept that combines constraints from abstractions of each of the input concepts. In this way, "a red French sedan" and "a blue German minivan" can be

combined to "a red German sedan". *Analogies*, on the other hand, find commonalities between a combined concept (such as "sign forest") and source concepts (such as "forest"). The Structure Mapping Engine [7], likewise, is concerned with analogies. Analogies and amalgams fall into the broader field of conceptual blending [26, 11, 14]. Recent work in the area of linguistics [1] also discusses conceptual blending, as does the Heuristic-Driven Theory Projection [20]. Closest to our work in conceptual blending is work on upward refinement in the DL $\mathcal{EL}++$ [6]. All of these analyses are centered on describing the space between two concepts. However, they do not give us a language with operators to explicitly modify a single input concept. For example, none of the approaches can express the operation of taking a car, removing a plastic part, and replacing it by an aluminium part.

Modifying DL Concepts. Given a DL description of a product on offer and of a product in demand, semantic matchmaking is concerned with modifying the product in demand so that it matches the product on offer [17]. Work on identifying missing negative constraints also involves generation and manipulation of concepts [9]. However, while concept modification is central to these (and other) works on non-standard reasoning in DLs, they do not provide any means to explicitly express modifications of a concept, like *Remove the wheels from a car*. To the best of our knowledge, the only work that proposes such an operator is Teege [23]. We compare their subtraction operator with our own in Section 4.2. Also loosely related is work on belief change in DLs, which aims to modify a knowledge base to consistently incorporate new information, see e.g. [18, 10].

3 Preliminaries

Description logics (DLs) are a family of logical formalisms that describe semantic knowledge about a domain in terms of concepts (= classes, unary relations) and roles (=properties, binary relations). For example, the first line in Figure 1 says that the concept *PlasticRoof* is defined as the intersection of the concept *Roof* and the concept of all those things that are made of plastic. We concentrate here on one particular DL, $\mathcal{EL}$ [2]. We assume fixed sets $\mathsf{N_C}$ and $\mathsf{N_R}$ of *concept names* and *role names*, respectively. A *concept* is anything of the form

$$\top \mid A \mid C \sqcap D \mid \exists r.C$$

where C and D are concepts, A ranges over $\mathsf{N_C}$, and r ranges over $\mathsf{N_R} \cup \{u\}$, where u is the *universal role*. The set of these concepts will be denoted by $\mathcal{L}$.

A concept is *basic* if it is a named concept, $\top$, or an existential concept. By definition, every concept in $\mathcal{L}$ is a conjunction of one or more concepts. We will

$$\begin{array}{lcl} PlasticRoof & \equiv & Roof \sqcap \exists madeOf.Plastic \\ Car & \equiv & Vehicle \sqcap \exists hasPart.PlasticRoof \\ & & \sqcap \exists hasPart.Wheel \sqcap \exists usedFor.Travel \end{array}$$

Fig. 1. An example terminology $\mathcal{T}$.

therefore understand every concept C as a conjunction $C = C_1 \sqcap ... \sqcap C_n$. If any C_i is a conjunction, then C_i can be folded into the conjunction. In all of the following, we will therefore assume that every concept is a conjunction of basic concepts. To simplify notation, we will talk of "the conjunct C_i of C" to mean that $C = C_1 \sqcap ... \sqcap C_n$ and $1 \leq i \leq n$.

DL semantics relies on *interpretations* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty *domain*, which can be understood as a set of real-world entities, and $\cdot^{\mathcal{I}}$ is an *interpretation function* which maps each $A \in \mathsf{N_C}$ to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and each $r \in \mathsf{N_R}$ to a subset $r^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The universal role u is mapped to $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. This interpretation is extended to all concepts as follows: $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$, $(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$, and $(\exists r.C)^{\mathcal{I}} = \{x \mid \exists (x, y) \in r^{\mathcal{I}}$ such that $y \in C^{\mathcal{I}}\}$. We say that a concept C is *subsumed by* (or *implies*) a concept D, written $C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in all interpretations $\mathcal{I}$.

Subsumption between $\mathcal{EL}$ concepts can be decided in polynomial time [3] and adding the universal role does not increase the complexity. To test for subsumption between two concepts ($C \sqsubseteq D$), we can employ a polynomial algorithm based upon a syntactic characterization of subsumption [13] that works analogously to the well-known structural subsumption algorithm [2, Section 2.3.1] for the $\mathcal{FL}_0$ language. Details can be found in our technical report [22].

Concepts can contain redundant conjuncts, as in $\exists r.A \sqcap \exists r.\top$. We call a concept C *fully reduced* if there does not exist a concept C' such that (1) $C' \sqsubseteq C$, (2) every conjunct of C' appears in C, and (3) C' has less conjuncts than C. By removing redundant conjuncts from C, we can compute a fully reduced concept $red(C)$ equivalent to C. The *normal form* of a concept C, written $norm(C)$, is defined as follows:

- $norm(A) = A$, if A is a named concept or $\top$
- $norm(\exists r.C) = \exists r.norm(C)$
- $norm(C_1 \sqcap ... \sqcap C_n) = red(norm(C_1) \sqcap ... \sqcap norm(C_n))$

We show in our technical report [22] that the normal form of a concept is unique up to reordering of conjuncts and can be computed in polynomial time.

Ordering. We assume that the set of concept names $\mathsf{N_C}$ is ordered by a complete order $\prec_{\mathsf{N_C}}$, and the set of relation names $\mathsf{N_R}$ is ordered by a complete order $\prec_{\mathsf{N_R}}$. Based on these, it is easy to define a complete order $\prec$ on concepts, as follows.

Definition 1 (Order): Given a complete order $\prec_{\mathsf{N_C}}$ on concept names $\mathsf{N_C} \cup \{\top\}$ and a complete order $\prec_{\mathsf{N_C}}$ on relation names $\mathcal{R} \cup \{u\}$, the complete order $\prec$ on minimal concepts $\mathcal{L}$ is defined as follows:

- $A \prec B$ iff $A \prec_{\mathsf{N_C}} B$, for $A, B \in \mathsf{N_C}$
- $A \prec C$, for $A \in \mathsf{N_C}$ and $C \notin \mathsf{N_C}$
- $\exists r.C \prec D$ for conjunctions D
- $\exists r.C \prec \exists s.D$ iff $r \prec_{\mathsf{N_R}} s$
- $\exists r.C \prec \exists r.D$ iff $C \prec D$
- $C \prec D$ for conjunctions C, D is given by the lexicographical extension of $\prec$.

This order is purely syntactic; it does not relate to concept subsumption. If the conjuncts of a conjunction $C_1 \sqcap ... \sqcap C_n$ are written in increasing order $C_1 \prec ... \prec C_n$, we will talk of an *ordered conjunction*. In all of the following, we will assume conjunctions to be ordered. For example, from Figure 1, we will understand that $Vehicle \prec \exists hasPart.PlasticRoof \prec \exists hasPart.Wheel \prec \exists usedFor.Travel$, because the concepts are written in that order. Ordered conjunctions have an important property, which follows from the properties of the normal form:

Property 1 (Ordered conjunctions in normal form): Two ordered conjunctions in normal form are equivalent iff they are identical.

Terminologies. A *concept definition* takes the form $A \equiv C$, where $A \in \mathsf{N_C}$ and $C \in \mathcal{L}$. A *terminology* $\mathcal{T}$ is a set of concept definitions, in which no concept name occurs more than once on the left-hand side of a concept definition. Figure 1 shows an example terminology. We will see the terminology as a function $\mathcal{T} : \mathsf{N_C} \to \mathcal{L}$, which, given a named concept, replaces it by the right-hand-side of its definition in $\mathcal{T}$. We consider only *acyclic* terminologies, i.e., there are no cyclic dependencies between the concept definitions. This allows us to define the complete recursive *unfolding* $\mathcal{T}^*(\cdot)$, with $\mathcal{T}^*(A) = A$ for concept names that do not have a definition in $\mathcal{T}$, $\mathcal{T}^*(A) = \mathcal{T}^*(\mathcal{T}(A))$ for concept names that have a definition, $\mathcal{T}^*(\exists r.C) = \exists r.\mathcal{T}^*(C)$ and $\mathcal{T}^*(C_1 \sqcap ... \sqcap C_n) = \mathcal{T}^*(C_1) \sqcap ... \sqcap \mathcal{T}^*(C_n)$. We say that a concept name A is a *declared child* of a concept name B if B appears as a conjunct of the definition of A.

4 Operators for Concept Modification

We will now define the operators of our language for combinatorial creativity. Our operators will work directly on DL concepts. The background terminology $\mathcal{T}$ will play no role in defining the operators, but will serve instead to provide the DL concepts upon which we will apply the operators. This may seem unconventional at first, but perhaps this is only fitting for a paper that treats matters of creativity.

4.1 Addition

Definition 2 (Addition): For two concepts C and D, we define addition as $C + D := norm(C \sqcap D)$.

> *Example 2 (Addition):* In our example from Figure 1, the expression $\mathcal{T}(Car) + (\exists hasPart.Wing \sqcap \exists usedFor.Fly)$ denotes a car with wings that is used for flying. This yields $Vehicle \sqcap \exists hasPart.Wheel \sqcap \exists hasPart.PlasticRoof \sqcap \exists usedFor.Travel \sqcap \exists hasPart.Wing \sqcap \exists usedFor.Fly$.

Addition has the following properties, which follow directly from the properties of $\sqcap$ and the normalization.

Property 2 (Inclusion): If $C \sqsubseteq D$ for two concepts C and D, then $C + D = C$.

Property 3 (Commutative Monoid): For any three concepts C, D, E, the following hold: Addition is closed, $C+D \in \mathcal{L}$. Addition is monotone, $C+D \sqsubseteq C$. Addition is commutative, $C + D = D + C$. Addition is associative, $C + (D + E) = (C + D) + E$. $\top$ is its neutral element. $(+, \top)$ forms a commutative monoid.

4.2 Subtraction

Definition 3 (Subtraction): For a basic concept A, and a concept C that has an ordered normal form $norm(C) = C_1 \sqcap ... \sqcap C_n$, we define subtraction as $C - A := norm(C_1 \sqcap ... \sqcap C_{j-1} \sqcap C_{j+1} \sqcap ... \sqcap C_n)$, where $j = argmin\ _i\{C_i : C_i \sqsubseteq A\}$. If there is no such j, then $C - A = C$. Subtraction is left-associative. For a conjunction D with $norm(D) = D_1 \sqcap ... \sqcap D_m$, subtraction is defined as $C - D = C - D_1 - ... - D_m$.

In other words, the subtraction $C-A$ removes the first conjunct of the ordered conjunction C that implies A. If the subtrahent is a conjunction, subtraction removes each conjunct of the subtrahent.

> *Example 3 (Subtraction):* In our example from Figure 1, the expression $\mathcal{T}(Car) - (\exists hasPart.\top \sqcap \exists usedFor.Travel)$ removes the first $hasPart$ association and the $usedFor$ association. This yields $Vehicle \sqcap \exists hasPart.Wheel$.

The definition of subtraction offers several design choices. We could, e.g., define subtraction simply as the set difference of the conjuncts, without considering subsumption of concepts. However, the proposed definition has the advantage that we can remove a part of a car even if we do not fully specify it, as in $\mathcal{T}(Car) - \exists hasPart.\top$. Another design alternative for a subtraction $C-A$ would be to remove not just the first conjunct from C that implies A, but all conjuncts in C that imply A. However, we can easily express this design alternative in terms of the proposed definition by subtracting A several times, whereas it is not possible to express the subtraction of just one conjunct with an operator that subtracts all implied conjuncts simultaneously. We could define subtraction to remove not the first matching conjunct, but an arbitrary conjunct. This, however, would result in non-determinism. We could also define subtraction so as to return the set of all possible ways of subtracting the subtrahent. However, the result of this operation would be a set, not a concept. Thus, it would not be possible to use this result with the other operators.

Finally, the subtraction of a conjunction $(C - D)$ offers a design alternative. Instead of subtracting each conjunct of D separately, we could remove from C all those conjuncts that are subsumed by the complete concept term D. This, however, would violate the usual set semantics: subtracting a conjunction with more conjuncts would have less effect than subtracting a conjunction with only one conjunct. We could also make subtraction distributive, by defining $C - (A \sqcap B) = (C - A) \sqcap (C - B)$. This, however, would entail that $(A \sqcap B) - (A \sqcap B) = (A \sqcap B)$, which would defeat the purpose of subtraction.

The subtraction operator from Definition 3 has the following properties, which follow from the definition and the normalization.

Property 4 (Monotonicity): For any two concepts C and D, the following holds: Subtraction is closed, $C - D \in \mathcal{L}$. Subtraction is monotone, $C \sqsubseteq C - D$.

Property 5 (Destruction): If $C \sqsubseteq D$ for a basic concept C and a concept D, then $C - D = \top$. In particular, $C - \top = \top$.

Property 6 (Self-Destruction): For any concept C, $C - C = \top$.

Property 7 (Reversibility): If, for two conjunctions in normal form C and D, every conjunct of D appears in C, then $(C - D) + D = C$.
Proof: If C and D are in normal form, and every conjunct of D appears in C, then $C - D$ is obtained by removing every conjunct of D from C (the subsequent normalization will have no effect). Adding back these conjuncts yields C. □

We can generalize subtraction to remove an arbitrary conjunct among the implied conjuncts, as follows.

Definition 4 (Generalized Subtraction): For a concept C, a basic concept A, and a natural number i, we define

$$C -_1 A := C - A \qquad C -_i A := ((C - A) -_{i-1} A) + (C - (C - A))$$

Property 8 (Generalized Subtraction): The generalized subtraction $C -_i A$ removes the i^{th} conjunct of the normalized ordered conjunction of C that implies A. If no such conjunct exists, then $C -_i A = C$.
Proof: Assume that C has been transformed into an ordered conjunction in normal form. We proceed by induction. For $i = 1$, the claim follows from Definition 3. Now assume that $C -_{i-1} A$ removes the $(i-1)^{th}$ conjunct of C that implies A. We observe that $C - A$ removes the first such conjunct. Hence, the expression $((C - A) -_{i-1} A)$ removes the first such conjunct and the i^{th} such conjunct. The expression $(C - (C - A))$ yields the first such conjunct. Hence, $((C - A) -_{i-1} A) + (C - (C - A))$ is C without the i^{th} such conjunct. If the i^{th} such conjunct does not exist, then $((C - A) -_{i-1} A) = C - A$. Then, $C -_i A = (C - A) + (C - (C - A)) = C$. □

Example 4 (Generalized Subtraction): With generalized subtraction, we can remove, e.g., the second part of our example car from Figure 1, by saying $\mathcal{T}(Car) -_2 \exists hasPart.\top$. This yields $Vehicle \sqcap \exists hasPart.PlasticRoof \sqcap \exists usedFor.Travel$.

Comparison to Related Work. Teege [23, Definition 2.1] defines subtraction as $C - D := max_{\sqsupseteq}\{E \in \mathcal{L} : D \sqcap E \equiv C\}$. We first note that this operator is undefined if $C \not\sqsubseteq D$, while our operator is always defined. Let us now assume that $C \sqsubseteq D$ and that C and D are conjunctions in normal form. If every conjunct of D appears in C, then Property 7 tells us that our operator is equivalent to Teege's. In general, however, the operators are different. Our definition allows removing a conjunct that "matches" the subtrahent, as in $(A \sqcap \exists r.B) - \exists r.T = A$. Teege's operator has a different behavior: $(A \sqcap \exists r.B) - \exists r.T = (A \sqcap \exists r.B)$. This

allows our operator to remove the first, second, or n^{th} matching conjunct, while Teege's operator does not offer this functionality. We will show in Section 5 how this functionality can be used in practice.

4.3 Succession

Definition 5 (Succession): For an existential concept $\exists r.E$ and a concept C, we define succession as $C \rightarrow \exists r.E := E'$, where $\exists r'.E'$ is the first conjunct of the ordered normal form $norm(C)$ with $\exists r'.E' \sqsubseteq \exists r.E$. If there is no such conjunct, succession is undefined. For a conjunction D with $norm(D) = D_1 \sqcap ... \sqcap D_m$, succession is defined as $C \rightarrow D = norm(\sqcap_i (C \rightarrow D_i))$.

Succession finds the first existential conjunct in C that implies $\exists r.E$, and returns the inner concept in that existential conjunct. If succession is used with a conjunction, the operator joins all the target concepts in a conjunction.

> *Example 5 (Succession):* For our example from Figure 1, the expression $\mathcal{T}^*(Car) \rightarrow \exists hasPart.(\exists madeOf.Plastic)$ asks for a plastic part of a car. This yields the plastic roof, $Roof \sqcap \exists madeOf.Plastic$.

The definition of succession allows several design choices. The current definition picks the first target concept. We could equally well use the operator to pick one of them at random. This, however, would result in non-determinism. Another design alternative is to return not the first matching conjunct, but the set of all matching conjuncts. Then, however, the result would no longer be a concept, and could no longer be used with the other operators.

We could also combine all target concepts of all matching conjuncts into a conjunction. Then, however, the target concepts could no longer be manipulated individually. If we want to compute such a conjunction nonetheless, we can do so with the current definition of succession. It suffices to extract one concept after the other through generalized succession (see below), and join them by addition.

Succession has the following properties, which follow from Definition 5.

Property 9 (Closedness): For any concepts C and D, $C \rightarrow D$ is either undefined or a concept.

Property 10 (Inclusion): If $C \sqsubseteq D$, then $\exists r.C \rightarrow \exists r.D = C$, for any concepts C, D and role r.

Like for subtraction, we can define a succession for the i^{th} matching conjunct.

Definition 6 (Generalized Succession): For concepts C, an existential concept $\exists r.E$, and a natural number i, we define

$$(C \rightarrow_1 \exists r.E) := C \rightarrow \exists r.E$$
$$(C \rightarrow_i \exists r.E) := (C - \exists r.E) \rightarrow_{i-1} \exists r.E$$

Property 11 (Generalized Succession): The generalized subtraction $C \rightarrow_i A$ returns the inner concept of the i^{th} existential conjunct of the normalized ordered conjunction of C that implies A.

Proof: Assume that C has been transformed into an ordered conjunction in normal form. We proceed by induction. For $i = 1$, the claim follows from Definition 5. Now assume that $C \rightarrow_{i-1} A$ returns the $(i-1)^{th}$ existential conjunct of C that implies A. We observe that $(C - A)$ removes the first such conjunct. Hence, $(C - A) \rightarrow_{i-1} A$ returns the i^{th} such conjunct. □

Example 6 (Generalized Succession): In our example from Figure 1, the expression $\mathcal{T}(Car) \rightarrow_2 \exists hasPart.\top$ retrieves the second part of a car. This yields $Wheel$.

Comparison to Related Work. To the best of our knowledge, the succession operator has no analog in the formalisms of previous work [17, 6, 7, 23].

4.4 Selection and Replacement

We can define a selection operator by using multiple applications of subtraction:

Definition 7 (Selection): For a concept C, a basic concept A, and a natural number i, we define

$$C \uparrow_i A = (C \sqcap \chi) - (C \sqcap \chi -_i A)$$

where χ is a fresh concept name that comes last in the order $\prec_N$.

Property 12 (Selection): The selection $C \uparrow_i A$ returns the i^{th} conjunct of the ordered normal form of C that implies A, or else $\top$.
Proof: Assume that C has been transformed into an ordered conjunction in normal form. If there is no i^{th} conjunct of C that implies A, then $(C \sqcap \chi -_i A) = C \sqcap \chi$, and hence $C \uparrow_i A = \top$. Otherwise, $(C \sqcap \chi -_i A)$ will remove that i^{th} conjunct, and $(C \sqcap \chi) - (C \sqcap \chi -_i A)$ will deliver that conjunct. This holds in particular for the case where $i = 1, A = \top$ and C is basic. In this case, $C \uparrow_1 \top = (C \sqcap \chi) - (C \sqcap \chi - \top) = (C \sqcap \chi) - \chi = C$. Note that without the addition of χ, we would obtain $C \uparrow_1 \top = C - (C - \top) = C - \top = \top$. □

Example 7 (Selection): In our example from Figure 1, we can select the second $hasPart$-conjunct of a car by saying $\mathcal{T}(Car) \uparrow_2 \exists hasPart.\top$. This yields $\exists hasPart.Wheel$.

We can also combine addition and subtraction to define an operator that replaces a certain conjunct, as follows:

Definition 8 (Replacement): For concepts C, P, D with P taking the form $\exists r_1.\exists r_2.\ldots\exists r_n.A$ (with $n \geq 0$, all $r_i \neq u$, and A a concept name), we define

$$C.replace(P, D) := (C - P) + D \quad \text{for } P \text{ a named concept}$$
$$C.replace(\exists r.P, D) := (C - \exists r.P) + \exists r.((C \rightarrow \exists r.P).replace(P, D))$$

Example 8 (Replacement): In our example from Figure 1, we can replace the material of the roof of the car by aluminium by stating
$\mathcal{T}^*(Car).replace(\exists hasPart.(\exists madeOf.Plastic), Aluminium)$
This expression first unfolds Car completely w.r.t. $\mathcal{T}$. Then, the replacement operator descends into the $hasPart$ conjunct and into the $madeOf$ conjunct, where $Plastic$ is replaced by $Aluminium$. This yields:
$Vehicle \sqcap \exists hasPart.(Roof \sqcap \exists madeOf.Aluminium)$
$\sqcap \exists hasPart.Wheel \sqcap \exists usedFor.Travel$

Property 13 (Replacement): The replacement $C.replace(\exists r_1.\exists r_2....\exists r_n.A, D)$ descends inside the first existential conjunct of $norm(C)$ that implies $\exists r_1.\exists r_2.\ldots.\exists r_n.A$ by 'entering' $\exists r_1$, then enters $\exists r_2$ for the first conjunct that implies $\exists r_2.\ldots.\exists r_n.A$, continues in this manner until entering $\exists r_n$, and finally replaces the first occurrence of A in the resulting concept by D.
Proof: Let $P = \exists r_1.\exists r_2.\ldots.\exists r_n.A$, and assume that C has been transformed into an ordered conjunction in normal form. We proceed by induction. For $n = 0$, we have $C.replace(A, D)$. The first case of Definition 8 will then remove the first conjunct A in C, and then add D. Next assume that the claim holds when the chain of existentials has length at most $n - 1$. The second case of Definition 8 will first remove from C the first conjunct that implies P. Suppose this conjunct is $\exists r_1.F$. Then we will add back the concept $\exists r_1.(F.replace(P, D))$. As F has a chain of $n - 1$ existentials, we can apply the induction hypothesis to infer that $F.replace(P, D)$ is obtained by following the chain $\exists r_2 \ldots \exists r_n$ to reach a concept containing A (always choosing the first $\exists r_i$ conjunct that entails $\exists r_i \ldots \exists r_n.A$) and replacing there the first occurrence of A by D. By adding the prefix $\exists r_1$ to the resulting concept, we obtain a concept with the stated property. □

We define $C.replace^i(P, D)$ as the i-fold application of the operator.

4.5 Tractability and Generality

All of the above operations are defined on concepts without the use of a background theory (TBox). Subtraction, e.g., removes a conjunct that is subsumed by a given concept, but this subsumption is purely syntactic and independent of a terminology ($\sqsubseteq$ instead of $\sqsubseteq_{\mathcal{T}}$). If we want a terminology $\mathcal{T}$ to come into play, we have to explicitly unfold a concept by the operator $\mathcal{T}^*(\cdot)$. The reason for this design choice is that for the goal of modifying concepts, it can be counter-intuitive if concepts are automatically expanded. Consider again our example from Figure 1, and assume that we are looking for the first conjunct in the definition of Car: $\mathcal{T}(Car) \uparrow_1 \top = Vehicle$. If concepts were automatically expanded, then adding a definition of $Vehicle$ to $\mathcal{T}$ would change this result. Moreover, a full recursive concept expansion would be of exponential complexity [2, Section 2.2.4.2]. Our framework avoids this complexity.

All of our operations are defined for normal forms. This means that, before applying an operator, its arguments have to be reduced to their normal forms. This guarantees that, for any operator $\otimes$, $(C \otimes D) \equiv (C' \otimes D)$ if $C \equiv C'$. Consider for example the (redundant) concept $C = (\top \sqcap A)$. The term $C - \top$ yields A. Now

consider the equivalent concept $C' = A$. Now, $C' - \top$ yields $\top$. To avoid such artifacts, all concepts have to be brought to their normal form before applying the operators. Since two equivalent concepts have the same normal form, we can guarantee that $(C \otimes D) \equiv (C' \otimes D)$, if $C \equiv C'$. Bringing a concept to its normal form can be done by a simple polynomial algorithm [22].

All of our operators are polynomial-time operations. For addition, this is easy to see: it suffices to join the arguments in a conjunction, and to apply the normalization algorithm. For subtraction, we have to run through all conjuncts in the left argument, and perform a (polynomial) subsumption check for each of them. This yields a polynomial algorithm. The same is true for succession.

We conclude our discussion of operators by noting that our language can transform any concept into any target concept. This can always be achieved trivially by subtracting all conjuncts of the original concept and adding all conjuncts of the target concept.

5 Evaluation

We conducted three types of evaluations: a case study on a business case of a French start-up; a descriptive study, where we use our language to describe inventions; and a generative study, where we use it to generate new concepts.

5.1 Case Study

As a case study, we consider a business case by Stim[2], a French consulting firm founded in 2013. Stim studies the product or service of the client company, and then uses the C-K method [12] to rethink it and suggest alternatives. We look here at the case of the client Turbomeca[3], a French manufacturer of gas turbine engines for helicopters. The original technology that Turbomeca employed was a motor working with two symmetric turbines, which both run during the whole flight. The suggested alternative was a "hybrid motor", which works with two different turbines that can run and stop during the flight according to power needs. After a feasibility study, Turbomeca is now developing this new motor, which reduces the energy consumption by half.

We focus here on how the transition from the old motor to the new one can be written down explicitly. Assuming suitable sets of named concepts and relations, the original motor looks as follows:

$$Original \equiv Motor \sqcap \exists worksWith.(Turbine \sqcap \exists placed.Left \sqcap \exists runsDuring.Flight) \\ \sqcap \exists worksWith.(Turbine \sqcap \exists placed.Right \sqcap \exists runsDuring.Flight)$$

The motor is then modified as follows:

$$Original.replace^2(\exists worksWith.\exists runsDuring.Flight, (Duration \sqcap \exists partOf.Flight))$$

[2] http://www.wearestim.com/
[3] http://www.turbomeca.com/

The resulting motor is:

$$Motor \sqcap \exists worksWith.(Turbine \sqcap \exists placed.Right \sqcap \exists runsDuring.(Duration \sqcap \exists partOf.Flight))$$
$$\sqcap \exists worksWith.(Turbine \sqcap \exists placed.Left \sqcap \exists runsDuring.(Duration \sqcap \exists partOf.Flight))$$

This shows that it is possible to model the transition from the old motor to the new one in our language. However, it also shows that we cannot replace both turbines at the same time. We have to apply the *replace* operation twice (by help of the superscript "2", see Section 4.4). This can be traced back to a design choice in our language, which allows subtracting the first matching conjunct, but not *all* conjuncts. This choice was made because subtracting all conjuncts can be achieved by successively removing individual conjuncts. What is more, subtraction becomes idempotent after all matching conjuncts have been removed, so that one can simulate the removal of all conjuncts by iterating the removal of individual conjuncts a large number of times (the same is true for *replace*()). We leave such extensions for future work.

5.2 Descriptive Study

We wanted to analyze to what degree real-world inventions can be described in our language. For this purpose, we considered the top 25 inventions of 2015 according to Time Magazine[4]. We asked computer-science students in our department to model, for each invention, a pair of two concepts: the original concept (say, a toy that can talk) and the innovative concept (say, a toy that can engage in a dialog with the child). Each concept had to be drawn as a graph of concept nodes that are connected by role edges. The students were allowed to use any concept and role names. Figure 2 shows the talking toy as an example (in vectorized form).

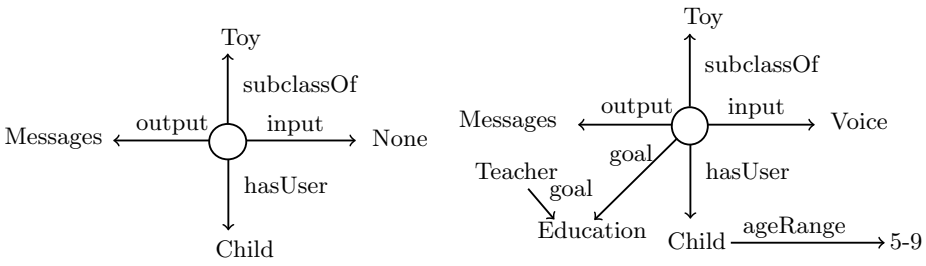

Fig. 2. A talking toy (left) and a toy that can engage in a dialog (right).

We translated the graphs to $\mathcal{EL}$, making modifications where necessary. For example, we translated numeric items such as the age range *5-9* into atomic

[4] `http://time.com/4115398/best-inventions-2015/`

concepts (*FiveNine*). We translated incoming edges (as in the *Education* node in Figure 2 on the right) by introducing new roles (e.g. *isGoalOf*).

Our goal was then to make the transition of the original concept to the new concept explicit, by modeling it in our language. Figure 3 exemplifies this process for the talking toy. In general, we found that the transitions could be modeled with a few operators in our language. The most useful operation proved to be *replace()*. It was used 29 times. The next most frequent operator was addition, with 16 cases. Subtraction was used in 7 cases. We discuss more details of this experiment in our technical report [22].

$$
\begin{array}{l}
(((\mathit{Toy} \sqcap \exists \mathit{input}.\mathit{None} \sqcap \exists \mathit{output}.\mathit{Messages} \sqcap \exists \mathit{hasUser}.\mathit{Child}) \\
.\mathit{replace}(\exists \mathit{input}.\mathit{None}, \mathit{Voice})).\mathit{replace}(\exists \mathit{hasUser}.\mathit{Child}, (\mathit{Child} \sqcap \exists \mathit{ageRange}.\mathit{FiveNine})) \\
+ \exists \mathit{goal}.(\mathit{Education} \sqcap \exists \mathit{isGoalOf}.\mathit{Teacher})) \\
\qquad\qquad\qquad\qquad \vdots \\
((((\mathit{Toy} \sqcap \exists \mathit{output}.\mathit{Messages} \sqcap \exists \mathit{hasUser}.\mathit{Child} \sqcap \exists \mathit{input}.\mathit{Voice}) - \exists \mathit{hasUser}.\mathit{Child}) \\
+ \exists \mathit{hasUser}.((((\mathit{Toy} \sqcap \exists \mathit{output}.\mathit{Messages} \sqcap \exists \mathit{hasUser}.\mathit{Child} \sqcap \exists \mathit{input}.\mathit{Voice}) \\
\rightarrow \exists \mathit{hasUser}.\mathit{Child})).\mathit{replace}(\mathit{Child}, (\mathit{Child} \sqcap \exists \mathit{ageRange}.\mathit{FiveNine})))) + \\
\exists \mathit{goal}.(\mathit{Education} \sqcap \exists \mathit{isGoalOf}.\mathit{Teacher})) \\
\qquad\qquad\qquad\qquad \vdots \\
(\mathit{Toy} \sqcap \exists \mathit{output}.\mathit{Messages} \sqcap \exists \mathit{input}.\mathit{Voice} \sqcap \exists \mathit{hasUser}.(\mathit{Child} \sqcap \exists \mathit{ageRange}.\mathit{FiveNine}) \\
\sqcap \exists \mathit{goal}.(\mathit{Education} \sqcap \exists \mathit{isGoalOf}.\mathit{Teacher}))
\end{array}
$$

Fig. 3. Transition from a talking toy to a toy that engages in dialog. Top line: original concept. Following lines: transition formula. In the middle: an example step from the actual transition. Bottom lines: new concept.

5.3 Generative Study

We want to show that our language can also be used (in a limited manner) to *generate* new concepts. More precisely, our goal is to show that our operators can be used to model something similar to brainstorming.

We use ConceptNet [15], a large knowledge base of commonsense facts. ConceptNet knows, e.g., that cars have wheels, and that they are used for locomotion. Since we are interested in generating new objects, we remove relations that describe words (EtymologicallyDerivedFrom, etc.), as well as relations that describe agents and events. To clean out noise, we also remove all definitions that have 2 or less conjuncts. This leaves us with a terminology $\mathcal{T}$ of 28 relations and 5485 concept definitions, each of which contains 41 conjuncts on average.

To generate new concepts, we use the following formulas. Here, $\mathit{child}_i^{\mathcal{T}}(\cdot)$ retrieves the i^{th} declared child of a concept in $\mathcal{T}$.

1. Addition of a conjunct of a sibling concept:

$$(\mathcal{T}(\mathit{child}_i^{\mathcal{T}}(\mathcal{T}(x) \uparrow_j \top)) \uparrow_k \exists u.\top) - \mathcal{T}(x)$$

For integers i, j, k and a concept x, this formula selects the j^{th} conjunct of the input concept x. In the example of Figure 1 with $j = 1$, this yields $Vehicle$. The formula then asks for the i^{th} child of that conjunct in the terminology. This could be, e.g., $Plane$ (a sibling of Car under $Vehicle$). From this sibling concept, we choose the k^{th} existential conjunct. This could be, e.g., $\exists has.Wing$. We make sure that the chosen conjunct does not already appear in the original concept x. If the result of this operation exists[5], and if the result is not $\top$, we propose to add this conjunct to the original concept.

2. Removal of a conjunct: $\mathcal{T}(x) \uparrow_i \exists u.\top$
   We select the i^{th} existential conjunct of the input concept x. In Figure 1, we could e.g., choose the 3rd conjunct, $\exists hasPart.Wheel$. Then we propose to remove it.
3. Reversal of a conjunct:

$$\exists HasProperty.(\mathcal{T}(\mathcal{T}(x) \rightarrow_i \exists HasProperty.\top) \rightarrow \exists Antonym.\top)$$

   This formula takes the i^{th} HasProperty conjunct of the input concept x (e.g., $\exists HasProperty.Fast$), and picks its target concept ($Fast$). It expands this target concept from the terminology, and finds its antonym ($Slow$). If the result of this operation exists and is not $\top$, we propose to replace the original conjunct by the conjunct with the antonym.

For each of these cases, we automatically generate a human-readable question such as "Can you imagine a car with wings?" (for Addition), "Can you imagine a car without wheels?" (for Removal), and "Can you imagine a car that is slow?" (for Reversal).

Experiment. We randomly chose 100 concepts from our terminology. Then we applied the above formulas to each of them, increasing each variable i, j, k from 1 to 100. For each formula, we took the first two concepts generated this way (if they exist). Then we asked 9 computer science students to judge the generated concepts. We wanted to know whether the proposed modification is nonsense, already exists in the real world, can be imagined, or is considered creative. Some of concepts in our filtered ConceptNet do not describe objects, but people, actions, or events. We could not filter these out automatically, and hence added an option "This sentence does not describe a physical object". A final option is "I don't understand the sentence".

	Addition		Reversal		Removal	
Already exists	15	(19%)	7	(41%)	50	(31%)
Can be imagined	29	(37%)	5	(29%)	66	(40%)
Funny or creative	10	(12%)	1	(5%)	18	(11%)
Nonsense	23	(29%)	4	(23%)	27	(16%)
Total	77	(100%)	17	(100%)	161	(100%)

Table 1. Human judgement of generated concepts per technique

[5] The operation $child_i^{\mathcal{T}}$ may fail if the concept is undefined in the terminology.

In total, we generated 313 new concepts. Of these, 39 did not describe a physical item. 19 used concepts from ConceptNet that the judge did not understand (such as *ny*). For the remaining concepts, Table 1 shows the distribution of human judgements. In up to 29% of the cases, our techniques return nonsensical concepts, e.g., *A solar wind that is used for learning*. The addition of a sibling conjunct is the most risky technique here. In a large number of cases, our techniques return an existing concept. This is not surprising: it indicates that ConceptNet is incomplete. Interestingly, it also indicates that our techniques actually generated a reasonable concept. 29%–40% of the concepts we generate can be imagined, e.g., *A patio that is used for an orchestra to sit*. Finally, around 10% of our concepts are considered funny or creative. Examples are *Broken glass that does not cut feet*, *A front door without a doorbell*, or *Jelly beans that contain chocolate*.

Many improvements to our generative formulas can be envisaged, but a full investigation is outside the scope of the present paper. Here, we only show that one of the applications of our language is to express formulas that can generate concepts. We leave the study of better concept generation and more extensive experimental evaluation (using e.g., criteria proposed in [19]) for future work.

6 Conclusion

In this paper, we have introduced a formal language for combinatorial creativity. We have justified the choice of our operators and discussed their formal properties. In our experiments, we have shown that our language can be used to describe real-world inventions. In another experiment, we have demonstrated that our language can also be used, to a limited degree, to *generate* new concepts. For future work, we plan to investigate how our language can be used to generate reasonable concepts more systematically, thus working towards the goal of making machines truly creative one day.

Acknowledgments. This research was partially supported by Labex DigiCosme (project ANR-11-LABEX-0045-DIGICOSME) operated by ANR as part of the program "Investissement d'Avenir" Idex Paris-Saclay (ANR-11-IDEX-0003-02).

References

1. G. J. A. and D. F. Harrell. *The Structure of Style*, chapter Style: A Computational and Conceptual Blending-Based Approach. Springer, 2010.
2. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
3. F. Baader, R. Küsters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. In *IJCAI*, 1999.
4. T. R. Besold and E. Plaza. Generalize and blend: Concept blending based on generalization, analogy, and amalgams. In *ICCC*, 2015.

5. M. A. Boden. *The Creative Mind: Myths and Mechanisms*. Routledge, 2004.
6. R. Confalonieri, M. Schorlemmer, E. Plaza, M. Eppe, O. Kutz, and R. Peñaloza. Upward refinement for conceptual blending in description logic: An asp-based approach and case study in EL++. In *Joint Ontology Workshops*, 2015.
7. B. Falkenhainer, K. D. Forbus, and D. Gentner. The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41, 1989.
8. G. Fauconnier and M. Turner. Conceptual integration networks. *Cognitive Science*, 22(2), 1998.
9. S. Ferré and S. Rudolph. Advocatus diaboli - exploratory enrichment of ontologies with negative constraints. In *EKAW*, 2012.
10. G. D. Giacomo, M. Lenzerini, A. Poggi, and R. Rosati. On the update of description logic ontologies at the instance level. In *AAAI*, 2006.
11. J. Goguen. *International Conference on Conceptual Structures*, chapter What Is a Concept? Springer, 2005.
12. A. Hatchuel and B. Weil. C-k design theory. *Res. in Engin. Design*, 19(4), 2008.
13. B. Konev, M. Ludwig, D. Walther, and F. Wolter. The logical difference for the lightweight description logic EL. *CoRR*, abs/1401.5850, 2014.
14. O. Kutz, J. Bateman, F. Neuhaus, T. Mossakowski, and M. Bhatt. *Computational Creativity Research: Towards Creative Machines*, chapter E Pluribus Unum. Springer, 2015.
15. H. Liu and P. Singh. Conceptnet. *BT Technology Journal*, 22(4), Oct. 2004.
16. M. Llano, R. Hepworth, S. Colton, J. Charnley, and J. Gow. Automating fictional ideation using conceptnet. In *AISB14 Symp. on Computational Creativity*, 2014.
17. T. D. Noia, E. D. Sciascio, and F. M. Donini. Semantic matchmaking as non-monotonic reasoning: A description logic approach. *Artif. Int. Research*, 29, 2007.
18. G. Qi and F. Yang. A survey of revision approaches in description logics. In *Web Reasoning and Rule Systems*, 2008.
19. G. Ritchie. Some empirical criteria for attributing creativity to a computer program. *Minds and Machines*, 17(1):67–99, 2007.
20. M. Schmidt, U. Krumnack, H. Gust, and K.-U. Kühnberger. *Computational Approaches to Analogical Reasoning: Current Trends*, chapter Heuristic-Driven Theory Projection: An Overview. Springer, 2014.
21. M. Schorlemmer, A. Smaill, K.-U. Kühnberger, O. Kutz, S. Colton, E. Cambouropoulos, and A. Pease. Coinvent: Towards a computational concept invention theory. In *ICCC*, 2014.
22. F. M. Suchanek, C. Menard, M. Bienvenu, and C. Chapellier. A language for combinatorial creativity. Technical report, Telecom ParisTech, 04 2016. Available at https://suchanek.name.
23. G. Teege. Making the difference: A subtraction operation for DL. In *KR*, 1994.
24. P. Thagard and T. C. Stewart. The aha! experience: Creativity through emergent binding in neural networks. *Cognitive Science*, 35(1), 2011.
25. H. Toivonen, S. Colton, M. Cook, and D. Ventura, editors. *International Conference on Computational Creativity*, 2015.
26. T. Veale and D. ODonoghue. Computation and blending. *Cogn. Ling.*, 11, 2000.