Anti-Unification and Natural Language Processing[*]

Nino Amiridze and Temur Kutsia

RISC, Johannes Kepler University Linz, Austria

Abstract

Anti-unification is a well-known method to compute generalizations in logic. Given two objects, the goal of anti-unification is to reflect commonalities between these objects in the computed generalizations, and highlight differences between them.

Anti-unification appears to be useful for various tasks in natural language processing. Semantic classification of sentences based on their syntactic parse trees, grounded language learning, semantic text similarity, insight grammar learning, metaphor modeling: This is an incomplete list of topics where generalization computation has been used in one form or another. The major anti-unification technique in these applications is the original method for first-order terms over fixed arity alphabets, introduced by Plotkin and Reynolds in 1970s, and some of its adaptations.

The goal of this paper is to give a brief overview about existing linguistic applications of anti-unification, discuss a couple of powerful and flexible generalization computation algorithms developed recently, and argue about their potential use in natural language processing tasks.

1 Introduction

In this paper we discuss a formal tool coming from logic (anti-unification), which has been used in various natural language processing tasks, see, e.g., [13–15, 19–21, 31, 36–40]. Our motivation is to introduce novel anti-unification techniques to natural language researchers, presenting the corresponding algorithms and some illustrative examples. We hope that these techniques can have helpful applications in linguistics-related areas, and experts can find more elaborated use to them than we envisaged.

Anti-unification aims at computing generalizations of the given objects. Generalization computation is a pretty common task in learning, where one would like to extract common features from the given concrete examples. In logic, the problem often is formulated for two terms: Given s and t, the problem is to compute their generalization, a term r such that s and t can be obtained from r by some variable substitutions. The interesting generalization are those, which retain maximal similarities between s and t, and abstract over their differences by fresh variables in a uniform way. Such generalizations are called the least general generalizations (lggs). For instance, the terms $f(a, g(a))$ and $f(b, g(b))$, where a and b are constants, have several generalizations: x, $f(x, y)$, $f(x, g(y))$, and $f(x, g(x))$, but the lgg is one: $f(x, g(x))$. It indicates that the original terms have in common the main binary function symbol f and the unary function symbol g in the second argument, and that the first argument of f and the argument of g are the same.

Anti-unification has been introduced in 1970s in two papers, published in the same volume of Machine Intelligence, by Plotkin [32] and Reynolds [33]. The algorithms have been formulated for first-order terms over a fixed arity (ranked) alphabet, and they compute the least general generalization for input terms. The original motivation was the application in inductive reasoning, and later the technique (extended to richer theories) has been used in the

[*] Supported by Austrian Science Fund (FWF) under the project P 28789-N32.

areas such as inductive logic programming, machine learning, analogical reasoning, software code close detection, program analysis, proof generalization, learning custom gestures, etc. The name anti-unification indicates that the process is dual to unification [34], which computes most general common instance of terms.

Semantic parsing is a natural language processing task where anti-unification has been used in the past. The system WOLFIE [39] applies Plotkin's clause anti-unification in the step of computing an initial set of candidate meanings for each possible phrase. In the COCKTAIL system [38], two inductive logic programming approaches have been combined for the task of learning semantic parsers. Also here, Plotkin's anti-unification is used. More modern methods of semantic parsing are based on statistical approaches.

There are other works related to natural language processing, which use some form of generalization computation. One such instance is the construction of information extraction systems, which are supposed to obtain information about certain items from natural language documents. It is a shallow text processing problem. Building a system that would solve it is a laborious task. Machine learning has been seen as a possible means to address this difficulty. The RAPIER system [15] uses machine learning (inductive logic programming) methods to learn rules that can be used for information extraction. In the learning process, it needs to compute generalizations of rules and also of word senses (taken from WordNet). For this, it uses an algorithm of computing least general generalizations, which is a bit more elaborated than the one used in induction logic programming. For instance, for word sense generalization one needs to take into account the semantic hierarchy. In general, inductive logic programming methods, where anti-unification for least general generalization computation is a core mechanism, have been intensively used in natural language processing for tasks which involve learning, for instance, learning grammars, as in, e.g., [31,35,42]. A different kind of grammar learning method, insight grammar learning, also makes use of anti-unification [36,37]. The relation of these works with the focus of this paper is a bit remote and we do not elaborate on their details here. However, we would like to discuss a special form of grounded language learning, which has been introduced recently.

Grounded language learning is an established research topic in computational linguistics. It is related to semantic parsing, mentioned above. According to [12, 13], the problem can be formulated as follows: Input consists of example sentences and a context, and the goal is to learn a mapping between n-grams (sequences of n words) and meanings, where meaning stands for whatever is in common among all the contexts in which that n-gram can be used. Both context and meaning are specified in first-order logic. In the process of learning the meaning of n-grams, their generalizations are computed with the help of Plotkin's anti-unification algorithm.

The above mentioned applications can be largely classified as grammatical inference problems for natural language.

Feature structures [16] (or their equivalent expressions, ψ-terms [1]) are a formalism on which unification grammars [18] are based. In [18], the authors say that they are not aware of proposals to use generalization of feature structures for linguistic applications except of two works on coordination phenomena and grammar learning.

In [19–21], a method to automatically infer semantic features from syntax is proposed. Anti-unification over syntactic parse trees is used to compute similarities between them, which then can help to classify sentences as being informative and can serve as recommendations. We will speak more about this work later.

Motivated by using syntactic generalization for finding similarities in texts, anti-unification has been included in a multi-layer system for semantic textual similarity [40], as a tool to extract the syntactic structure information from texts.

In [23, 24], the authors propose an approach to model and analyze natural language metaphoric expressions, which is a quite hard problem. They use the technique called heuristic-driven theory projection [22], which generalizes of classical anti-unification by permitting formulas as inputs with the corresponding equational/equivalence theories. It is guided by heuristics.

In this paper, we first recall the original anti-unification algorithm, then consider some of its adaptations/generalizations and their applications. After that, we discuss more recent algorithms, obtained from their classical first-order counterparts by relaxing the fixed arity restriction and permitting second order variables, and show advantages of their use for natural language processing tasks.

2 First-Order Ranked Anti-Unification

First-order ranked alphabet consists of the disjoint sets of variables and function symbols, where each function symbol has a fixed arity (rank). *Terms* are defined in the usual way, by grammar $t ::= x \mid f(t_1, \dots, t_n)$, where x is a variable and f is an n-ary function symbol. The letters x, y, z are used for variables and s, t, r for terms.

Substitutions are mappings from variables to terms, where all but finitely many variables are mapped to themselves. They are denoted by lower case Greek letters and are usually written in the form of finite sets, e.g., the substitution σ can be represented as $\{x \mapsto \sigma(x) \mid x \neq \sigma(x)\}$. For a substitution σ and a term t, application of σ on t, written $t\sigma$, is defined as follows: If t is a variable x, then $t\sigma = \sigma(t)$; If t is a compound term $f(t_1, \dots, t_n)$, then $t\sigma = f(t_1\sigma, \dots, t_n\sigma)$.

A term s is *more general* than t if there exists a substitution σ such that $s\sigma = t$. We write $s \preceq t$ to denote this fact. The term t is called an *instance* of s. For instance, $f(x, g(x, y)) \preceq f(b, g(b, h(a)))$, because $f(x, g(x, y))\sigma = f(b, g(b, h(a)))$ for $\sigma = \{x \mapsto a, y \mapsto h(a)\}$. On the other hand, $f(x, x) \npreceq f(a, b)$, because no substitution can map x at the same time to a and b.

The relation $\preceq$ is quasi-ordering. It generates an equivalence relation, denoted by $\simeq$. The strict part of $\preceq$ is denoted by $\prec$. A term s is a generalization of t_1 and t_2 if $s \preceq t_1$ and $s \preceq t_2$. It is a *least general generalization (lgg)* if $s \npreceq r$ for any generalization r of t_1 and t_2. For first-order ranked terms, lggs are unique modulo variable renaming. The terms $f(a, h(a))$ and $f(b, h(b))$ have generalizations (up to variable renaming) x, $f(x, y)$, $f(x, h(y))$, and $f(x, h(x))$, but only the last one is the lgg.

To compute least general generalizations, one can use, e.g., Plotkin's algorithm from [32]. Reynolds' algorithm [33] is essentially the same. Later, Huet [25] formulated a simpler algorithm for the same problem in terms of recursive equations: Assuming that ϕ is a given bijection from a pair of terms to variables, he defined a function AU, which maps pairs of terms to terms:

- $\mathsf{AU}(f(t_1, \dots, t_n), f(s_1, \dots, s_n)) = f(\mathsf{AU}(t_1, s_1), \dots, \mathsf{AU}(t_n, s_n))$, for any f.
- $\mathsf{AU}(t, s) = \phi(t, s)$ otherwise.

Huet's algorithm was analyzed in detail in [30]. A modification of syntactic anti-unification for a finite set of terms was considered in [4], proposing to generalize the given set of terms not necessarily with a single term, but with several of them, to reduce overgeneralization. Each computed term generalizes some subset of the input. The bound on the number of terms in the generalization is a part of the problem.

Plotkin in [32] also proposed an anti-unification algorithm for clauses, which are disjunctions (or sets) of atomic formulas or their negations. The notion of generalization there differs from the same notion for terms: A clause C is more general than a clause D (written $C \preceq D$) iff there exists a substitution ϑ such that $C\vartheta \subseteq D$ (in words, C ϑ-subsumes D). For instance,

$\{\neg q(x,y), \neg q(y,x), p(x)\} \leq \{\neg q(x,x), p(x)\}$, which can be seen easily, taking $\vartheta = \{y \mapsto x\}$. Lgg for clauses is unique modulo $\leq$ and the equivalence generated by the subsumption relation.

In the presence of equational axioms, there is no single lgg, in general. For instance, if f is associative with unit element, then the terms $f(f(a,b), f(b,c))$ and $f(a, f(b,c))$ have two minimal generalizations: $f(f(a,b), f(x,c))$ and $f(f(a,x), f(b,c))$. Even more, for some theories and terms there can be infinitely many incomparable generalizations.

Another kind of knowledge can be a certain type hierarchy or taxonomy among notions function symbols stand for. For instance, such a taxonomy might tell us that a square is a quadrilateral, which is a polygon. A triangle also is a polygon. Then one would generalize $\mathit{position}(\mathit{square}, p_1)$ and $\mathit{position}(\mathit{triangle}, p_2)$ into $\mathit{position}(\mathit{polygon}, x)$. Without the extra knowledge about polygons, the lgg would be just $\mathit{position}(y, x)$.

In applications related to natural language processing, first-order ranked anti-unification and anti-unification for clauses are probably the most frequently used techniques of this kind. A concept taxonomy might be also available, when generalization affects semantic structures.

Anti-unification in inferring semantic properties from syntactic parse trees. A recent work on an interesting use of anti-unification in a linguistics-related application is reported in [21]. The authors try to obtain semantic properties, unobservable on the level of keywords, from syntactic parse trees with the help of generalization. The semantic properties they are interested in require a deep natural language understanding. Hence, instead of shallow parsing, they are interested in obtaining a rich linguistic data structure, such as syntactic parse trees. They also illustrate that for their problems, these structures, as a subject of learning, perform better than keyword-based approaches. The primary motivation of their work is the automation of content management and a delivery platform, to support recommendation forums for a wide variety of products and services. Users as well as automated agents answer questions and provide recommendations based on previous postings by human users. Therefore, finding similarities between various types of texts is required. The use of more rich structures to represent texts' meanings is justified by the fact that it indeed helps to assess similarities and relevance more accurately in practice, as the reported results illustrate. The practical problems addressed in the paper include detecting expressions suitable for automatic ad generation, classifying user postings with respect to how well she understands what product she needs, and classifying search results with respect to their relevance to queries.

The process of generalization computation described in [21] consists of several steps, including pre-processing and post-processing. The actual anti-unification is done only on the processed trees. The authors describe these steps for two sentences as follows:

1. Obtain the parsing tree for each sentence. For each word (tree node), we have a lemma, a part of speech and the form of the word's information. This information is contained in the node label. We also have an arc to the other node.
2. Split sentences into sub-trees that are phrases for each type: verb, noun, prepositional and other types. These sub-trees are overlapping. The sub-trees are coded so that the information about their occurrence in the full tree is retained.
3. All the sub-trees are grouped by phrase types.
4. Extend the list of phrases by adding equivalence transformations.
5. Generalize each pair of sub-trees for both sentences for each phrase type.
6. For each pair of sub-trees, yield an alignment, and generalize each node for this alignment. Calculate the score for the obtained set of trees (generalization results).

7. For each pair of sub-trees of phrases, select the set of generalizations with the highest score (the least general).
8. Form the sets of generalizations for each phrase type whose elements are the sets of generalizations for that type.
9. Filter the list of generalization results: for the list of generalizations for each phrase type, exclude more general elements from the lists of generalization for a given pair of phrases.

Equivalence transformations are performed with the help of predefined rules. For instance, such a rule might convert "camera with digital zoom" into "digital zoom camera". Depending how the word sub-trees are paired, multiple generalizations may be obtained. From those, one may select the interesting ones (or the best one) based on some heuristics, for instance, the parts of speech may be assigned different weights and the generalization with the highest weight is selected. In [21], an example of such a weight assignment is presented, giving to nouns highest weight and to prepositional phrases the lowest, treating common and frequent verbs inferior than less common ones, etc. Generalization scores are computed based on the weights. The same process applies not only to sentences, but to phrases and paragraphs as well.

We take from [21] a simplified example, which can illustrate how the method described in that paper works for sentences. Assume the following three sentences are given:

S1. I am curious how to use the digital zoom of this camera for filming insects.

S2. How can I get short focus zoom lens for digital camera?

S3. Can I get auto focus lens for digital camera?

The parse trees of S2 and S3 are pretty similar and their generalization, written as a list, is {MD-can, PRP-I, VB-get, NN-focus, NN-lens, IN-for JJ-digital NN-camera}. For S1 and S2 they are quite different. Therefore, they are split into chunks of phrases. Examples of chunks for S1 are (SBAR-how to use the digital zoom of this camera for filming insects), (NP-the digital zoom of this camera), (NP-the digital zoom), (DT-the), etc. There are in total 29 such chunks for S1 and 19 for S2. Afterwards, for each sentence, these phrases are grouped so that phrases of one type (NP, PP, VP, etc) are placed in one group. Then one takes a phrase from one group, say, from NP, in S1, another phrase from the same group from S2, and tries to generalize them, provided that certain integrity constraints are satisfied. This process is repeated for all phrase-pairs, aiming to establish correspondences between as many words as possible. The result can be several generalizations, returning meanings which are common to the sentences. For S1 and S2, there are six such generalizations. Two of them are quite interesting: [JJ-digital NN-camera] (the sentences have a common concept "digital camera") and [VBP-* ADJP-* NN-zoom NN-camera] (a phrase talking about "some-kind-of zoom camera").

In [21], further interesting themes are discussed, e.g., how to get logical forms from generalizations, generalization-based search, etc., but they go beyond the scope of our discussion.

Anti-unification in grounded language learning. In [12], the authors propose a method of learning the meaning of phrases from phrase/context pairs in which the phrase's meaning is not explicitly represented. They aim at modeling the way how children learn language. Often, learning from a physical context means to find a correspondence between the phrase elements and observed things. The phrases are assumed to be linked to the context, but it is not required that all context elements are mentioned in phrases.

Contexts and meanings are represented as first-order logic expressions, and an incremental learning algorithm is presented. A phrase is represented as a sequence of words, and a context as a set of ground facts (ground atomic formulas). For instance, a context in which a big red square is to the left of a small green triangle is represented as {object(o1), shape(o1, sq), color(o1, rd), size(o1, bg), object(o2), shape(o2, tr), color(o2, gr), size(o2, sm), relative-position(o1, lo, o2)}. To define the meaning of a sentence, phrase or word, the authors propose a pragmatic solution: the meaning of an n-gram is "whatever is common among all contexts where the n-gram can be used". This "common" knowledge is formalized with the help of lggs with respect to ϑ-subsumption. For instance, for two contexts {obj(o1), clr(o1,re), shp(o1,sq), obj(o2), clr(o2,gr), shp(o2,tr), relpos(o1,lo,o2)} and {obj(o3), clr(o3,gr), shp(o3,tr), obj(o4), clr(o4,re), shp(o4,tr), relpos(o3,lo,o4)}, their lgg, the most specific common pattern of both contexts, is {obj(B), clr(B,re), shp(B,D), obj(E), clr(E,gr), shp(E,tr), obj(A), clr(A,C), shp(A,D), relpos(A,lo,F), obj(F), clr(F,G), shp(F,tr)}, where the capital letters are variables. It states that there are red and green objects (the objects B and E, respectively), E is a triangle, and there is an object A to the left of triangle F. It does not imply that these objects are necessarily distinct.

The meaning of an n-gram is the most specific common pattern of all the contexts where it can be used. There is a simple algorithm that incrementally learns the meaning of specific n-grams: whenever a new example (context/phrase pair) (C, P) appears, update the meaning of each n-gram G in P with respect to C, i.e., use the procedure $\mathsf{Update}(G, C)$. The latter can be defined as

```
$\mathsf{Update}(G, C)$ :
    if $\mathsf{Meaning}(G)$ is undefined then $\mathsf{Meaning}(G) := C$ else $\mathsf{Meaning}(G) := lgg(C, \mathsf{Meaning}(G))$
```

In fact, the learning algorithm from the paper is more involved and takes into account how the meaning of an n-gram depends of the meanings of k-grams for $k < n$, but the main idea and the use of generalization is sufficiently demonstrated by the algorithm above, which we borrowed from [14]. It has been shown that the system, which is based on the implementation of the algorithm from [21], learns to understand and generate simple natural language utterances using only the context/utterance pairs, without taking candidate meanings as input. The latter are constructed by the system itself. Learning from more complex contexts (which may include actions) and learning more complex languages are mentioned among topics for further research.

3 First- and Second-Order Unranked Anti-Unification

In many applications, arity of function symbols is not fixed. XML documents, hedge automata, and the programming language of Mathematica [41] are prominent examples. Alphabets, where symbols do not have fixed arity, are called unranked, variadic, flexary, or polyadic. We use here the term "unranked". It is interesting to notice that parse trees can be also seen as expressions over an unranked alphabet. For instance, in Fig. 1 one can see that the nodes labeled with NP have one, two, three, and four children at different places. It indicates that if we consider NP as a function symbol used in the construction of parse trees, it should be unranked.

In unranked first-order languages function symbols do not have fixed arity.[1] there are also hedge variables, which stand for finite, possibly empty sequences of terms. Such sequences are called hedges. In anti-unification for such languages (see, e.g., [29]), they help to deal with position mismatches for similar argument pairs in terms to be generalized.

Sometimes the expressive power of first-order languages is not enough and one would like to bring in higher-order variables. In the context of generalization computation (see [7]), it gives a capability to detect similarities at different levels in terms. We use a special kind of second-order variables, called context variables. Contexts that we consider here are not the same as

[1] Often there is also a set of ranked function symbols, see, e.g., [27, 28], but for simplicity here we assume only the unranked ones.

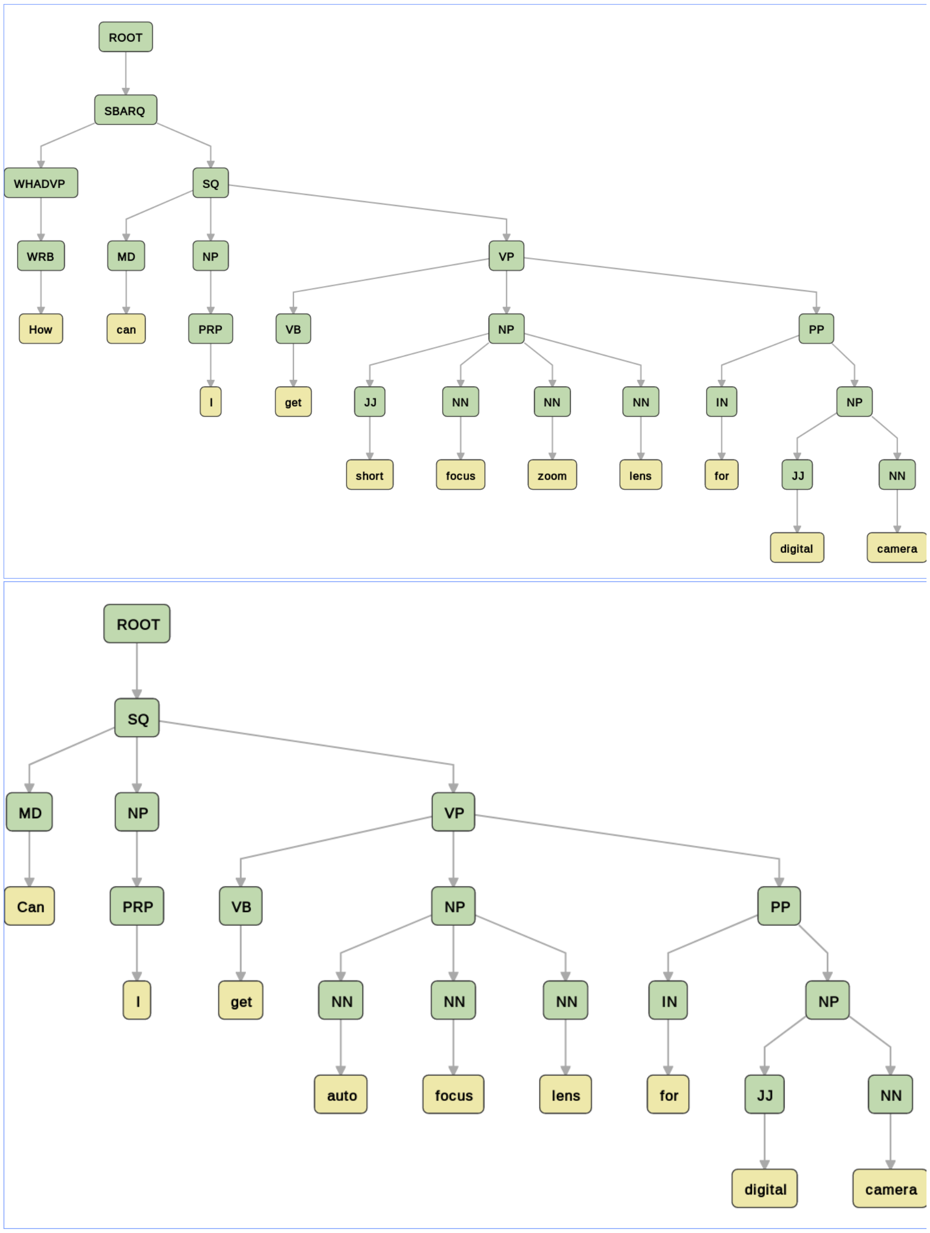

Figure 1: Parse trees of the sentences “How can I get short focus zoom lens for digital camera” and “Can I get auto focus lens for digital camera” (obtained from `http://corenlp.run/`)

contexts in grounded language learning. Our contexts are hedges with a single occurrence of the distinguished symbol “hole”. They are functions which can apply to another context or to a hedge, which are then “plugged” in the place of the hole. Unless otherwise stated, in the rest of the paper the word “context” is used in this sense. Hence, permitting the use of context variables helps to abstract vertical differences between trees, while hedge variables will be used to abstract horizontal differences.

More formally, following [7], we consider pairwise disjoint countable sets of unranked function symbols $\mathcal{F}$, hedge variables $\mathcal{V}_{\mathsf{H}}$, unranked context variables $\mathcal{V}_{\mathsf{C}}$, and a special symbol $\circ$ (the hole), and define *terms*, *hedges*, and *contexts* by the following grammar:

$$\begin{array}{rcll} t & := & X \mid f(\tilde{s}) \mid \overline{X}(\tilde{s}) & \text{(terms)} \\ \tilde{s} & := & t_1, \dots, t_n & \text{(hedges)} \\ \tilde{c} & := & \tilde{s}_1, \circ, \tilde{s}_2 \mid \tilde{s}_1, f(\tilde{c}), \tilde{s}_2 \mid \tilde{s}_1, \overline{X}(\tilde{c}), \tilde{s}_2 & \text{(contexts)} \end{array}$$

where $x \in \mathcal{V}_{\mathsf{H}}$, $f \in \mathcal{F}$, $X \in \mathcal{V}_{\mathsf{C}}$, and $n \geq 0$. (For simplicity, we do not consider here the standard variables, for single terms. If necessary, they can be brought in without any difficulties.)

Hedges are finite sequences of terms, constructed over $\mathcal{F}$ and $\mathcal{V}_{\mathsf{H}} \cup \mathcal{V}_{\mathsf{C}}$. A term can be seen as a singleton hedge. A context can be seen as a hedge over $\mathcal{F} \cup \{\circ\}$ and $\mathcal{V}_{\mathsf{H}} \cup \mathcal{V}_{\mathsf{C}}$, where the hole occurs exactly once. To improve readability, we put non-singleton hedges and contexts between parenthesis. The letters X, Y, Z denote hedge variables and $\overline{X}, \overline{Y}, \overline{Z}$ context variables. The empty hedge is denoted by ϵ. Terms of the form $a(\epsilon)$ are written as just a.

A context $\tilde{c}$ can apply to a hedge $\tilde{s}$, denoted by $\tilde{c}[\tilde{s}]$, obtaining a hedge by replacing the hole in $\tilde{c}$ with $\tilde{s}$. Application of a context to a context is defined similarly.

Examples of a term, hedge, and a context are, resp., $f(f(a), b)$, $(X, \overline{X}(a, X), f(f(a), b))$, and $(X, \overline{X}(a, X), f(f(\circ), b))$. The latter can be applied to a hedge $(a, \overline{X}(a))$, resulting in $(X, \overline{X}(a, X), f(f(\circ), b))[a, \overline{X}(a)] = (X, \overline{X}(a, X), f(f(a, \overline{X}(a)), b))$.

A *substitution* is a mapping from hedge variables to hedges and from context variables to contexts, which is identity almost everywhere. When substituting a context variable X by a context, the context will be applied to the argument hedge of X.

The notions of more general term / substitution, instance, generalization, least general generalization are extended from terms to hedges and contexts straightforwardly.

The second-order unranked anti-unification algorithm described in [7] first constructs a "skeleton" of a generalization of the input hedges, which corresponds to a hedge embedded into each of the input hedges. Next, it inserts context and/or hedge variables into the skeleton, which are supposed to uniformly generalize (vertical and horizontal) differences between input hedges, to obtain an lgg (with respect to the given skeleton). The skeleton computation function is the parameter of the algorithm and the soundness and completeness properties of the anti-unification algorithm do not depend on the particular instantiations of this parameter.

The skeleton computation function offers quite some flexibility in unranked anti-unification. With its help, one can effectively control the computed set of generalizations, which is very handy in linguistic applications like considered above. For instance, choosing common nodes based on predefined priorities (as required in [21]) can be a part of skeleton computation. To give priority to words over parts of speech (POS) can be implemented by requiring to maximize the number of skeleton nodes corresponding to the leaves of the input hedges. Also, in case of the existence of a taxonomy for the involved words, one can put the least upper bound of two distinct words in the skeleton of the corresponding parse trees, e.g, female for woman and lady, instrument for violin and guitar, etc.

Fig. 2 shows an lgg of two hedges. One can see that the variables $\overline{X}$ and Y are used twice, to abstract the same differences in two different places. The context variable $\overline{X}$ generalizes $h(\circ)$ in the first input hedge, and $\circ$ in the second. It is reflected in the triple $\overline{X} : h(\circ) \triangleq \circ$. The hedge variable Y generalizes the two-element hedge (b, b) in the first input, and the term (singleton hedge) d in the second. Besides the generalization, the algorithm returns such difference triples for each variable, which can be used to define anti-unification distance between the original hedges. In general, there is no single lgg for unranked terms, unless the skeleton computation function returns a singleton set. However, the minimal complete set generalizations exists.

We can apply the algorithm to the parse trees of S1, S2, and S3 above. The skeleton function should maximize the score of leaf words in the generalization, where verbs, nouns and

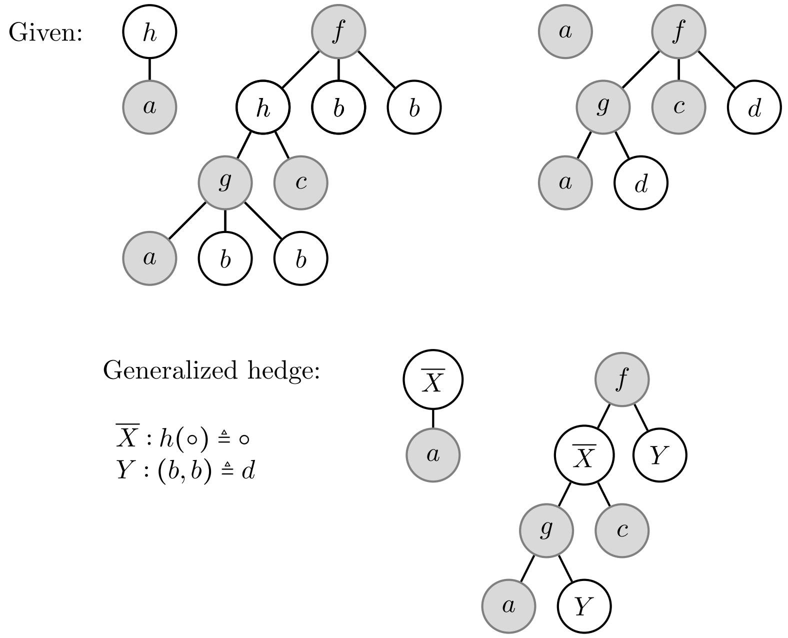

Figure 2: Generalization of hedges $(h(a), f(h(g(a,b,b),c),b,b))$ and $(a, f(g(a,d),c,d))$. The highlighted skeleton is the longest common subforest [3] of the input hedges.

adjectives weight more than other POSs. This would give 6 lggs, whose leaves are $l_1 \in \{\text{I, how}\}$, $l_2 \in \{\text{digital, zoom}\}$ and $l_3 = \text{camera}$. In comparison to [21], computation of these generalizations saves the effort of generating and grouping all those chunks and applying the syntactic anti-unification on the pairs. Unranked anti-unification can do the job directly on the parse trees.

Having said this, we should also mention that the current implementation of the unranked second-order anti-unification has only the longest common subforest as the skeleton computation function. More alternatives is planned to be added in future, including the one that gives priorities to leaves and returns the set of their longest common subsequence.

The generalization tree of the parse trees of S2 and S3 (see Fig. 1) is shown in Fig. 3. Not only the skeleton, but also the corresponding parts of speech are displayed, showing similarities in the input trees. The differences are generalized by one context and two sequence variables.

An useful extension of the unranked anti-unification algorithm is its commutative version, where the argument order of some function symbols does not matter. When combining it to the axiom $f(X,x,Y,x,Z) \approx f(X,x,Y,Z)$ (unranked idempotence), properties of sets can be easily modeled. (A special case of such a combination is used in anti-unification for description logics [26].) It can be used to compute generalizations of grounded language learning contexts in the examples above. We can wrap whole learning context under a commutative-idempotent unranked symbol. Besides, we can make contexts express information in a structured way, by putting all the information about each object under a commutative symbol, e.g., representing the contexts {obj(o1), clr(o1,re), shp(o1,sq), obj(o2), clr(o2,gr), shp(o2,tr), relpos(o1,lo,o2)} and {obj(o3), clr(o3,gr), shp(o3,tr), obj(o4), clr(o4,re), shp(o4,tr), relpos(o3,lo,o4)} as

$$\mathit{context}(\mathit{ob}(\mathsf{obj(o1)}, \mathsf{clr(o1,re)}, \mathsf{shp(o1,sq)}), \mathit{ob}(\mathsf{obj(o2)}, \mathsf{clr(o2,gr)}, \mathsf{shp(o2,tr)}), \mathsf{relpos(o1,lo,o2)})$$
$$\mathit{context}(\mathit{ob}(\mathsf{obj(o3)}, \mathsf{clr(o3,gr)}, \mathsf{shp(o3,tr)}), \mathit{ob}(\mathsf{obj(o4)}, \mathsf{clr(o4,re)}, \mathsf{shp(o4,tr)}), \mathsf{relpos(o3,lo,o4)}),$$

where the unranked symbol *context* is commutative-idempotent and *ob* is commutative. Then anti-unification gives their generalization, computing four (not necessarily distinct) objects:

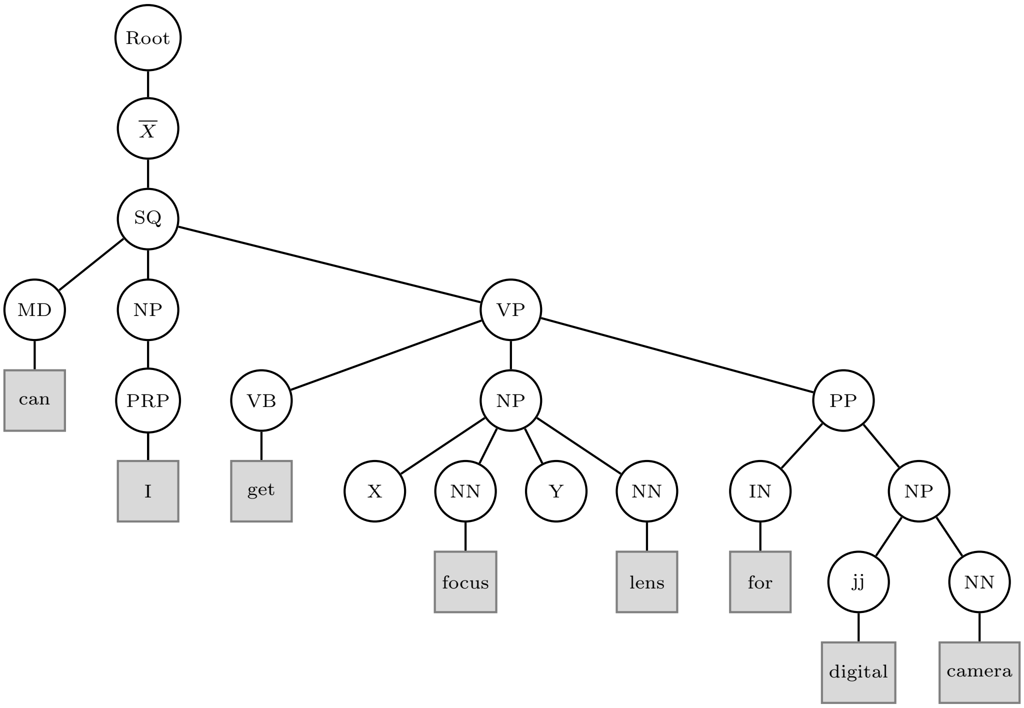

Figure 3: Least general generalization of the parse trees of S2 and S3 from Fig. 1. The skeleton is the longest common subsequence of the leaf nodes sequences. It is highlighted.

$$\mathit{context}(\mathit{ob}(\mathsf{obj}(\mathsf{B}),\mathsf{clr}(\mathsf{B},\mathsf{re}),\mathsf{shp}(\mathsf{B},\mathsf{D})),\mathit{ob}(\mathsf{obj}(\mathsf{E}),\mathsf{clr}(\mathsf{E},\mathsf{gr}),\mathsf{shp}(\mathsf{E},\mathsf{tr})),\\ \mathit{ob}(\mathsf{obj}(\mathsf{A}),\mathsf{clr}(\mathsf{A},\mathsf{C}),\mathsf{shp}(\mathsf{A},\mathsf{D})),\mathit{ob}(\mathsf{obj}(\mathsf{F}),\mathsf{clr}(\mathsf{F},\mathsf{G}),\mathsf{shp}(\mathsf{F},\mathsf{tr})),\mathsf{relpos}(\mathsf{A},\mathsf{lo},\mathsf{F})).$$

Moreover, anti-unification gives information how the original contexts can be obtained from the generalization, saying, e.g., that the objects A and B have the same shape D, which generalizes sq from the first context and tr from the second, etc. This information is in the result computed by anti-unification.

Our anti-unification algorithms are a part of RISC anti-unification algorithm library [6] and can be accessed from `http://www.risc.jku.at/projects/stout/library.html`. Particular algorithms from the library are described in [5, 7–11, 29]. Yet another open-source anti-unification library, for order-sorted equational theories, has been described in [2] and is available online at `http://safe-tools.dsic.upv.es/acuos/`.

4 Summary

We discussed some applications of anti-unification in linguistics, presented recently developed algorithms for this problem, and showed that they can be conveniently used in some natural language processing tasks.

References

[1] H. Ait-Kaci. *A lattice theoretic approach to computation based on a calculus of partially ordered type structures.* PhD thesis, University of Pennsylvania, 1984.

[2] M. Alpuente, S. Escobar, J. Espert, and J. Meseguer. ACUOS: A system for modular ACU generalization with subtyping and inheritance. In Fermé and Leite [17], pages 573–581.

[3] A. Amir, T. Hartman, O. Kapah, B. R. Shalom, and D. Tsur. Generalized LCS. *Theor. Comput. Sci.*, 409(3):438–449, 2008.

[4] H. Arimura, T. Shinohara, S. Otsuki, and H. Ishizaka. A generalization of the least general generalization. In *Machine Intelligence 13*, pages 59–85, 1994.

[5] A. Baumgartner. *Anti-Unification Algorithms: Design, Analysis, and Implementation.* PhD thesis, Johannes Kepler University Linz, 2015. Available from `http://www.risc.jku.at/publications/download/risc_5180/phd-thesis.pdf`.

[6] A. Baumgartner and T. Kutsia. A library of anti-unification algorithms. In Fermé and Leite [17], pages 543–557.

[7] A. Baumgartner and T. Kutsia. Unranked second-order anti-unification. *Inf. Comput.*, 255:262–286, 2017.

[8] A. Baumgartner, T. Kutsia, J. Levy, and M. Villaret. A variant of higher-order anti-unification. In F. van Raamsdonk, editor, *24th International Conference on Rewriting Techniques and Applications, RTA 2013*, volume 21 of *LIPIcs*, pages 113–127. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[9] A. Baumgartner, T. Kutsia, J. Levy, and M. Villaret. Nominal anti-unification. In M. Fernández, editor, *26th International Conference on Rewriting Techniques and Applications, RTA 2015*, volume 36 of *LIPIcs*, pages 57–73. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[10] A. Baumgartner, T. Kutsia, J. Levy, and M. Villaret. Higher-order pattern anti-unification in linear time. *J. Autom. Reasoning*, 58(2):293–310, 2017.

[11] A. Baumgartner, T. Kutsia, J. Levy, and M. Villaret. Term-graph anti-unification. In H. Kirchner, editor, *Third Conference on Formal Structures for Computation and Computation, FSCD 2018*, volume 108 of *LIPIcs*, pages 9:1–9:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[12] L. Becerra-Bonache, H. Blockeel, M. Galván, and F. Jacquenet. A first-order-logic based model for grounded language learning. In É. Fromont, T. D. Bie, and M. van Leeuwen, editors, *Advances in Intelligent Data Analysis XIV, IDA 2015*, volume 9385 of *LNCS*, pages 49–60. Springer, 2015.

[13] L. Becerra-Bonache, H. Blockeel, M. Galván, and F. Jacquenet. Relational grounded language learning. In G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, and F. van Harmelen, editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 1764–1765. IOS Press, 2016.

[14] H. Blockeel. Relational grounded language learning. Presentation slides for the Thirteenth International Conference on Grammatical Inference, ICGI 2016, Delft, The Netherlands. https://lirias.kuleuven.be/bitstream/123456789/552428/1/ICGI+2016.pdf, October 2016.

[15] M. E. Califf and R. J. Mooney. Bottom-up relational learning of pattern matching rules for information extraction. *Journal of Machine Learning Research*, 4:177–210, 2003.

[16] B. Carpenter. *The logic of typed feature structures: with applications to unification grammars, logic programs and constraint resolution*, volume 32. Cambridge University Press, 2005.

[17] E. Fermé and J. Leite, editors. *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014. Proceedings*, volume 8761 of *LNCS*. Springer, 2014.

[18] N. Francez and S. Wintner. *Unification grammars.* Cambridge University Press, 2011.

[19] B. Galitsky. Machine learning of syntactic parse trees for search and classification of text. *Eng. Appl. of AI*, 26(3):1072–1091, 2013.

[20] B. A. Galitsky. Generalization of parse trees for iterative taxonomy learning. *Inf. Sci.*, 329:125–143, 2016.

[21] B. A. Galitsky, J. L. de la Rosa, and G. Dobrocsi. Inferring the semantic properties of sentences by mining syntactic parse trees. *Data Knowl. Eng.*, 81-82:21–45, 2012.

[22] H. Gust, K. Kühnberger, and U. Schmid. Ontological aspects of computing analogies. In *Proceedings of the International Conference on Cognitive Modelling, ICCM 2004*, pages 350–351, 2004.

[23] H. Gust, K. Kühnberger, and U. Schmid. Metaphors and heuristic-driven theory projection (HDTP). *Theor. Comput. Sci.*, 354(1):98–117, 2006.

[24] H. Gust, K.-U. Kühnberger, and U. Schmid. Metaphors and anti-unification. In *Proc. Twenty-First Workshop on Language Technology: Algebraic Methods in Language Processing, Verona, Italy*, pages 111–123, 2003.

[25] G. Huet. Résolution d'Équations dans des langages d'ordre $1, 2, \ldots, \omega$. These d'État, Université de Paris VII, 1976.

[26] B. Konev and T. Kutsia. Anti-unification of concepts in description logic EL. In C. Baral, J. P. Delgrande, and F. Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016*, pages 227–236. AAAI Press, 2016.

[27] T. Kutsia. Solving equations involving sequence variables and sequence functions. In B. Buchberger and J. A. Campbell, editors, *AISC*, volume 3249 of *LNCS*, pages 157–170. Springer, 2004.

[28] T. Kutsia. Solving equations with sequence variables and sequence functions. *J. Symb. Comput.*, 42(3):352–388, 2007.

[29] T. Kutsia, J. Levy, and M. Villaret. Anti-unification for unranked terms and hedges. *J. Autom. Reasoning*, 52(2):155–190, 2014.

[30] J. Lassez, M. J. Maher, and K. Marriott. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann, 1988.

[31] M. Nakamura, Y. Hayashi, and R. Matoba. Simulation of language evolution based on actual diachronic change extracted from legal terminology. In H. J. van den Herik, A. P. Rocha, and J. Filipe, editors, *Proceedings of the 9th International Conference on Agents and Artificial Intelligence, ICAART 2017, Volume 1*, pages 343–350. SciTePress, 2017.

[32] G. D. Plotkin. A note on inductive generalization. *Machine Intel.*, 5(1):153–163, 1970.

[33] J. C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. *Machine Intel.*, 5(1):135–151, 1970.

[34] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[35] U. Schmid and E. Kitzelmann. Inductive rule learning on the knowledge level. *Cognitive Systems Research*, 12(3-4):237–248, 2011.

[36] L. Steels. Basics of fluid construction grammar. *Constructions and frames*, 9(2):178–225, 2017.

[37] L. Steels and P. V. Eecke. Insight grammar learning using pro- and anti-unification. https://www.fcg-net.org/wp-content/uploads/papers/steels-vaneecke.pdf.

[38] L. P. R. Tang. *Integrating Top-down and Bottom-up Approaches in Inductive Logic Programming: Applications in Natural Language Processing and Relational Data Mining.* PhD thesis, The University of Texas at Austin, 2003.

[39] C. A. Thompson and R. J. Mooney. Automatic construction of semantic lexicons for learning natural language interfaces. In J. Hendler and D. Subramanian, editors, *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence*, pages 487–493. AAAI Press / The MIT Press, 1999.

[40] N. P. A. Vo and O. Popescu. A multi-layer system for semantic textual similarity. In A. L. N. Fred, J. L. G. Dietz, D. Aveiro, K. Liu, J. Bernardino, and J. Filipe, editors, *Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2016) - Volume 1: KDIR*, pages 56–67. SciTePress, 2016.

[41] S. Wolfram. *The Mathematica Book.* Wolfram Media, 5th edition, 2003.

[42] J. M. Zelle and R. J. Mooney. Learning semantic grammars with constructive inductive logic programming. In R. Fikes and W. G. Lehnert, editors, *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 817–822. AAAI Press / The MIT Press, 1993.