Restricted Higher-Order Anti-Unification for Analogy Making

Ulf Krumnack, Angela Schwering, Helmar Gust, and Kai-Uwe Kühnberger

Institute of Cognitive Science, Osnabrück
{krumnack,aschweri,hgust,kkuehnbe}@uos.de

Abstract. Anti-unification has often be used as a tool for analogy making. But while first-order anti-unification is too simple for many applications, general higher-order anti-unification is too complex and leads into theoretical difficulties. In this paper we present a restricted framework for higher-order substitutions and show that anti-unification is well-defined in this setting. A complexity measure for generalizations can be introduced in a quite natural way, which allows for selecting preferred generalizations. An algorithm for computing such generalizations is presented and the utility of complexity for anti-unifying sets of terms is discussed by an extended example.

1 Introduction

Analogies are a central construct of human thinking [1,2,3] and are considered to be an indispensable tool for scientific progress [4]. Analogical reasoning is a high-level cognitive process in which two conceptualizations from a source and a target domain are compared and analyzed for common patterns. The purpose of analogies is to use old information (typically from the source domain) to explain new situations in the target domain. Experimental research supply evidence that structural commonalities between domains are the main driver for the construction of analogies. There exist many approaches for analogy models which apply different mechanisms to analyze and extract commonalities between two domains [5,6]. When the domains are specified formally, the theory of anti-unification can be used for a structural comparison and for representing the commonalities at a general level. Heuristic-Driven Theory Projection (HDTP) is such a symbolic analogy model using anti-unification to detect analogies between different domains.

This paper discusses anti-unification in the context of analogies and presents a spelled-out approach for computing analogies between domain theories. The remainder of the paper is structured as follows: Section 2 explains the theory of anti-unification in the context of analogies. After examining the related work (section 3), we present our approach for restricted higher-order anti-unification to compute structural commonalities (section 4). Section 5 shows how HDTP uses our approach and illustrates the results using the heat-flow example: the analogy between fluid dynamics (water flow) and thermodynamics (heat flow). Section 6 summarizes the paper and gives directions for future work.

2 Anti-Unification for Analogies

Anti-unification, originally introduced in the context of induction [7], is a formal counterpart of unification. While a unifier for a given set of terms T is a term u that is an instance[1] of every term of T, an anti-unifier (or generalization) is a term g that is an anti-instance of every term of T. While unification aims to find the most general unifier (mgu), anti-unification searches for the most specific anti-unifier, normally referred to as least general generalization (lgg).

$$\begin{tikzcd}[column sep=tiny] & X \arrow[dl] \arrow[dr] & \\ a & & b \end{tikzcd} \quad \begin{tikzcd}[column sep=tiny] & f(X,c) \arrow[dl] \arrow[dr] & \\ f(a,c) & & f(b,c) \end{tikzcd} \quad \begin{tikzcd}[column sep=tiny] & Y \arrow[dl] \arrow[dr] & \\ f(a,b) & & g(a,b) \end{tikzcd} \quad \begin{tikzcd}[column sep=tiny] & F(a,b) \arrow[dl] \arrow[dr] & \\ f(a,b) & & g(a,b) \end{tikzcd} \quad \begin{tikzcd}[column sep=tiny] & F(h(X,Y)) \arrow[dl] \arrow[dr] & \\ f(h(x,h(a,b))) & & g(h(a,b)) \end{tikzcd}$$

(a) (b) (c) (d) (e)

Fig. 1. Anti-Unification of terms

Fig. 1 gives several examples for anti-unification. Terms are generalized to an anti-instance where differing constants are replaced by a variable. The terms in (c) and (d) differ w.r.t. the function symbols. While first-order anti-unification fails to detect commonalities when function symbols differ, higher-order anti-unification generalizes the function symbols to a variable F and retains the structural commonality. Example (d) is formally of higher-order, but can be reified into first-order introducing a "meta" function $eval$ with $eval(f,(a,b)) = f(a,b)$. Example (e), however, affects the argument number and therefore cannot be reified: the left/right term F is substituted by f/g, $X \rightarrow x/a$ and $Y \rightarrow h(a,b)/b$. Higher-order anti-unification is necessary to anti-unify complex structures, but we loose the uniqueness of lgg and they are in general not well-defined. However, in the context of analogies, anti-instances have to meet several restrictions: generalized terms shall not be more complex than the domain terms and they shall contain elements that are present in both domains. This paper presents a solution for computing anti-instances with these restrictions for analogies. In section 4 we explain the role of anti-unification in analogy-making with HDTP.

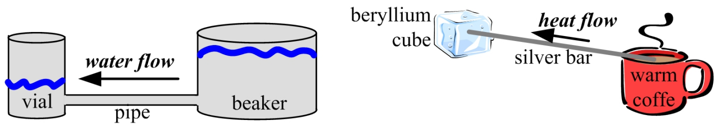

Fig. 2. Analogy between fluid dynamics (water flow) and thermodynamics (heat flow)

Heuristic-Driven Theory Projection (HDTP) is an analogy model using the theory of anti-unification to detect structural commonalities. Due to limited

[1] In this paper a term s is called an *instance* of t, and t an *anti-instance* of s if there is a substitution σ such that σ applied to t produces s. In this case we write $t \xrightarrow{\sigma} s$ or simply $t \rightarrow s$.

space we refer to [8] for the specification of the syntactic, semantic, and algorithmic properties of HDTP. Here we explain its functionality only with an example: The analogy between fluid dynamics and thermodynamics (Fig. 2). There are different possibilities of associating concepts of the target domain with concepts of the source domain. E.g. the bar and the pipe play equivalent roles as bearers of "flowing things" (water resp. heat). The task is to find plausible assignments to roles having some explanatory power. In the source domain one can observe water flowing. Although there cannot be observed anything flowing in the target domain, one can infer by analogy that there must exist something like "heat" that "flows" from the hot coffee to the beryllium cube.

Fluid dynamics (Th_S)	Thermodynamics (Th_T)
sorts $real,\ massterm, object,\ time$ **entities** $vial: object, beaker: object, water: massterm,\ pipe: object$ **functions** $height: object \times time \to real \times \{cm\}$ $footprint: object \times time \to real \times \{cm^2\}$ $in: object \times massterm \to object$ $vol: object \times time \to real \times \{cm^3\}$ **facts** $connected(beaker, vial, pipe)$ $\forall t_1 : time, t_2 : time :$ $footprint(beaker, t_1) > footprint(vial, t_1) \wedge$ $footprint(beaker, t_1) = footprint(beaker, t_2) \wedge$ $footprint(vial, t_1) > footprint(vial, t_2)$ **laws** $\forall t_1 : time, t_2 : time : t_2 > t_1 \wedge$ $height(in(water, beaker), t_1) > height(in(water, vial), t_1)$ $\to height(in(water, beaker), t_1) >$ $height(in(water, beaker), t_2) \wedge$ $vol(in(water, beaker), t_1) - vol(in(water, beaker), t_2)$ $= vol(in(water, vial), t_2) - vol(in(water, vial), t_1)$	**sorts** $real,\ massterm,\ object,\ time$ **entities** $coffee: massterm,\ b\_cube: object,$ $cup: object,\ bar: object$ **functions** $temp: object \times time \to real \times \{C\}$ $in: object \times massterm \to object$ **facts** $connected(in(coffee, cup), b\_cube, bar)$ **laws** $\forall t_1 : time,\ t_2 : time : t_2 > t_1$ $temp(in(coffee, cup), t_1) >$ $temp(b\_cube, t_1)$ $\to$ $temp(in(coffee, cup), t_2) <$ $temp(in(coffee, cup), t_1) \wedge$ $temp(b\_cube, t_2) >$ $temp(b\_cube, t_1)$

The source and target domain are specified by a set of formulas represented in many-sorted first-order logic. Given two theories Th_S and Th_T modeling source and target as input, the HDTP algorithm computes the analogy by selecting successively a formula from Th_S and Th_T (according to a heuristic) and constructing a generalization together with the corresponding substitutions.

source domain Th_S	target domain Th_T	generalization Th_G
$connected(beaker, vial, pipe)$	$connected(in(coffe, cup), b\_cube, bar)$	$connected(A, B, C)$
$height(in(water, beaker), t_1) >$ $height(in(water, vial), t_1)$	$temp(in(coffee, cup), t_1) >$ $temp(b\_cube, t_1)$	$T(A, t_1) >$ $T(B, t_1)$
$height(in(water, beaker), t_1) >$ $height(in(water, beaker), t_2)$	$temp(in(coffee, cup), t_1) >$ $temp(in(coffee, cup), t_2)$	$T(A, t_1) >$ $T(A, t_2)$
$height(in(water, vial), t_2) >$ $height(in(water, vial), t_1)$	$temp(b\_cube, t_2) >$ $temp(b\_cube, t_1)$	$T(B, t_2) >$ $T(B, t_1)$

Th_G contains four variables: A and B play the role of the container for the flowing "thing"; C connects A and B. The function symbol T stands for the function measuring the energy (height of the water in the container/temperature of the heat). The resulting substitutions describe the analogical relation:

$A \to beaker/in(coffee, cup)$	$B \to vial/b\_cube$
$C \to pipe/bar$	$T \to \lambda x, t : height(in(water, x), t)/temp(x, t)$

3 Related Work

Compared to unification, there is few work especially dedicated to anti-unification. However, many results on unification apparently have a counterpart in anti-unification. First of all, it should be noticed that higher-order anti-unification suffers from similar problems as higher-order unification: the *lgg* is no longer unique and without restrictions the notion is not even well-defined as demonstrated by [9]: one can construct infinite chains of ever more and more general generalizations. Therefore different strategies to restrict anti-unification have been proposed.

[10] uses higher-order patterns to restrict the possible generalizations. A λ-term (in β-normal form) is called a higher-order pattern if every free occurrence of a (higher-order) variable F appears as $F(X_1, \ldots, X_n)$ with bound variables X_i all being distinct. It is shown that the substitution ordering on higher-order patterns gives a preorder with unique maximally specific generalization for any pair of terms. Thus the pattern restriction leads to well-defined generalizations. A well-known problem of higher-order patterns is overgeneralization. Given $f(a)$ and $g(a)$, the least generalization would be X instead of $F(a)$ or $F(X)$, since the latter ones are no higher-order patterns.

[9] claims that a major problem of λ-terms is the missing control of how function arguments are used in terms: they can be discarded or doubled. He therefore proposes to use combinator terms instead of λ-terms. Combinator terms are built as composition of basic functions, called combinators, that allow to control how arguments are passed from one function to the next. By restricting the valid combinators to so called relevant combinators, it is shown that a useful notion of generalization can be developed and an algorithm to compute such generalizations is presented. This formalism is also used by [11] in the context of analogical programming.

4 Restricted Higher-Order Anti-Unification

In the context of analogies we aim at computing generalizations that preserve as much of the structure of both domain terms as possible. However, the generalization must not be structurally more complex than the original terms. In this section we define a formal framework that is guided by this requirements.

4.1 Extended Substitutions

We will extend classical first-order terms by introducing variables that can take arguments: for every natural number n we assume an infinite set $\mathcal{V}_n$ of variables with arity n and a finite sets of n-ary function symbols $\mathcal{C}_n$.[2] Here we explicitly allow the case $n = 0$ with $\mathcal{V}_0$ being the set of first-order variables and $\mathcal{C}_0$ being 0-ary function symbols (constants). Variables will be written as uppercase letters

[2] One could, of course, use a more elaborated system for typing, but for the sake of simplicity we will limit to arity here, as this suffices to support the arguments of the paper.

while function symbols are lowercase. A *term* is either a first-order or a higher-order term, i.e. an expression of the form $F(t_1, \dots, t_n)$ with $F \in \mathcal{V}_n$ and terms $t_1, \dots, t_n$. In this setting we will redefine the notion of substitution.

Definition 1 (Basic Substitutions). *We define the following set of basic substitutions:*

1. *A* renaming $\rho^{F,F'}$ *replaces a variable* $F \in \mathcal{V}_n$ *by another variable* $F' \in \mathcal{V}_n$ *of the same arity:*
$$F(t_1, \dots, t_n) \xrightarrow{\rho^{F,F'}} F'(t_1, \dots, t_n).$$
2. *A* fixation ϕ^V_c *replaces a variable* $F \in \mathcal{V}_n$ *by a function symbol* $f \in \mathcal{C}_n$ *of the same arity:*
$$F(t_1, \dots, t_n) \xrightarrow{\phi^F_f} f(t_1, \dots, t_n).$$
3. *An* argument insertion $\iota^{F,F'}_{V,i}$ *with* $0 \leq i \leq n$, $F \in \mathcal{V}_n$, $V \in \mathcal{V}_k$ *with* $k \leq n-i$, *and* $F' \in \mathcal{V}_{n-k+1}$ *is defined by*
$$F(t_1, \dots, t_n) \xrightarrow{\iota^{F,F'}_{V,i}} F'(t_1, \dots, t_{i-1}, V(t_i, \dots, t_{i+k-1}), t_{i+k}, \dots, t_n).$$
4. *A* permutation $\pi^{F,F'}_\alpha$ *with* $F, F' \in \mathcal{V}_n$ *and* $\alpha : \{1, \dots, n\} \to \{1, \dots, n\}$ *bijective, rearanges the arguments of a term:*
$$F(t_1, \dots, t_n) \xrightarrow{\pi^{F,F'}_\alpha} F'(t_{\alpha(1)}, \dots, t_{\alpha(n)}).$$

Argument fixation can be used to replace a variable by a symbol of the same arity, e.g. $f(X) \xrightarrow{\phi^X_a} f(a)$ with $X \in \mathcal{V}_0$ and $F(a) \xrightarrow{\phi^F_f} f(a)$ with $F \in \mathcal{V}_1$. Argument insertion is a bit more complicated: inserting a 0-ary variable V increases the arity of the embedding term by 1, e.g. $F(a) \xrightarrow{\iota^{F,F'}_{V,1}} F'(a, V)$. But inserting a variable $G \in \mathcal{V}_n$ with $n \geq 2$ reduces the arity: $F(a, b, c, d) \xrightarrow{\iota^{F,F'}_{G,1}} F'(a, G(b, c), d)$. Notice that we allow argument insertion to be applied to first-order (i.e. 0-ary) variables, e.g. $X \xrightarrow{\iota^{X,X'}_{c,0}} X'(c)$.

In what follows a *substitution* is always meant to be a composition of basic substitutions. We will write $s \to t$ if there exists a substitution that transforms s into t, and $s \xrightarrow{\sigma} t$ if σ is such a substitution. Notice that every first-order substitution is indeed a substitution, since it can be described as a composition of renaming, argument insertion and fixation, e.g.
$$X \xrightarrow{\iota^{X,X'}_{V,0}} X'(V) \xrightarrow{\phi^V_a} X'(a) \xrightarrow{\phi^{X'}_f} f(a)$$

Hence this framework is a real extension of the first-order case. We will now show that it is still sufficiently restricted to be of practical use. To make this more precise, we define a measure for the structural complexity of terms. This is achieved by counting the number of symbols the term is composed of.

Definition 2 (Information Load). *A term t is assigned the* information load $\mathfrak{il}(t)$ *recursively defined as follows:*

1. $\mathfrak{il}(F(t_1,\ldots,t_n)) = n + \mathfrak{il}(t_1) + \ldots + \mathfrak{il}(t_n)$ *for a n-ary variable $F \in \mathcal{V}_n$ and terms $t_1,\ldots,t_n$.*
2. $\mathfrak{il}(f(t_1,\ldots,t_n)) = 1 + n + \mathfrak{il}(t_1) + \ldots + \mathfrak{il}(t_n)$ *for a function symbol $f \in \mathcal{C}_n$ and terms $t_1,\ldots,t_n$.*

Here we also allow the case $n=0$, i.e. 0-ary variables and constants, e. g. $\mathfrak{il}(X)=0$ for $X \in \mathcal{V}_0$, $\mathfrak{il}(c)=1$ for $c \in \mathcal{C}_0$, and $\mathfrak{il}(f(c,X))=4$ with $f \in \mathcal{C}_2$. Therefore applying substitutions will never reduce the information load of a term:

Lemma 1. *Let s and t be terms. If $s \to t$, then $\mathfrak{il}(s) \leq \mathfrak{il}(t)$.*

Proof. This can be proven by inductive decomposition of the substitution: renaming and permutation do not change the information load. Fixation increases the information load by 1 and so does argument insertion: applying $\iota_{G,i}$ to $F(t_n,\ldots,t_n)$ with $G \in \mathcal{V}_k$ leads to $F'(t_1,\ldots,t_{i-1},G(t_i,\ldots,t_{i+k-1}),t_{i+k},\ldots,t_n)$ with information load $n-k+1+\sum_{j=1}^{i-1}\mathfrak{il}(t_j)+\mathfrak{il}(G(t_i,\ldots,t_{i+k}))+\sum_{j=i+k}^{n}\mathfrak{il}(t_j)$ and $\mathfrak{il}(G(t_i,\ldots,t_{i+k}))=k+\sum_{j=i}^{i+k-1}\mathfrak{il}(t_j)$, summing up to $n+1+\sum_{j=1}^{n}\mathfrak{il}(t_j)$. q.e.d.

As the information load of terms is always finite and there are only finitely many permutations of arguments for a given arity, it follows:

Corollary 1. *For a given term t there are only finitely many (up to renaming) anti-instances (i.e. terms s with $s \to t$).*

4.2 Preferred Generalizations

Based on our extended notions of terms and substitutions, generalization can be defined as usual:

Definition 3 (Generalization). *A generalization for a pair of terms $\langle s,t\rangle$ is a triple $\langle g,\sigma,\tau\rangle$ with a term g and substitutions σ,τ such that $s \xleftarrow{\sigma} g \xrightarrow{\tau} t$.*

Since for every term t there is a substitution $X \to t$, generalizations are guaranteed to exist. As a direct consequence of lemma 1 we get:

Proposition 1. *A pair of terms $\langle s,t\rangle$ has only finitely many (up to renaming) generalizations. For every generalization $\langle g,\sigma,\tau\rangle$ it holds $\mathfrak{il}(g) \leq \mathfrak{il}(s)$ and $\mathfrak{il}(g) \leq \mathfrak{il}(t)$.*

This means that least general generalizations are well-defined in this setting. However, they do not have to be unique:

Having multiple possible generalizations is not necessarily bad, especially in the context of analogies, where normally several mappings with different degree of plausibility may coexist. Nevertheless, it would be useful to have a criterion to rank the alternatives. A plausible approach could be based on the information load of the generalization. However, since we are aiming at anti-unifying sets of terms and want to promote the reuse of substitutions, we will introduce a ordering that is based on their complexity:

$$\begin{tikzcd}
& F(d,G(a)) \arrow[dl] \arrow[dr] & & & f(X,Y) \arrow[dl] \arrow[dr] & \\
f(g(a,b,c),d) & & f(d,\phi(a)) & f(g(a,b,c),d) & & f(d,\phi(a))
\end{tikzcd}$$

Fig. 3. Example with multiple least general generalizations

Definition 4 (Complexity of Substitution). *The* complexity *of a basic substitution σ is defined as*

$$\mathcal{C}(\sigma) = \begin{cases} 0 & \textit{if } \sigma = \rho \\ 1 & \textit{if } \sigma = \phi_c \\ k+1 & \textit{if } \sigma = \iota_{V,i} \textit{ and } V \in \mathcal{V}_k \\ 1 & \textit{if } \sigma = \pi_\alpha \end{cases}$$

For a composition of basic substitutions we define $\mathcal{C}(\sigma_1 \cdots \sigma_m) = \sum \mathcal{C}(\sigma_i)$ and for an arbitrary substitution σ is $\mathcal{C}(\sigma) = \min\{\mathcal{C}(\sigma_1 \cdots \sigma_m) \mid \sigma_1 \cdots \sigma_m = \sigma\}$.[3]

The complexity of a substitution is meant to reflect its processing effort. Therefore permutations have a non-zero complexity even though they do not change the information load of a term. The argument insertion restructures the term, and the higher the arity of the inserted variable, the more arguments are moved and therefore the more complexity is assigned to that operation.[4]

Based on complexity of substitutions, complexity of generalizations can be defined straight forward:

Definition 5 (Complexity of Generalization). *Let $\langle g, \sigma, \tau \rangle$ be a generalization for a pair of terms $\langle s, t \rangle$. Define the complexity of the generalization by $\mathcal{C}(\langle g, \sigma, \tau \rangle) = \mathcal{C}(\sigma) + \mathcal{C}(\tau)$.*

With this complexity measure we can select *preferred generalization* by minimizing their complexity. Obviously, preferred generalizations are always least general, while the contrary is not always the case as demonstrated by Fig. 3.

4.3 Computing Preferred Generalizations

A simple algorithm to compute preferred generalizations is given in Fig. 4. It uses a bottom-up breadth-first strategy. A priority queue of anti-instances is initialized with the *left* and *right* terms that shall be anti-unified. Now the first element of that queue is taken. If there is a compatible anti-instance for that term

[3] The minimum construction is needed, as there exist multiple decompositions of σ with different complexity. With a bit more effort one can define a normal decomposition which can be shown to have minimal complexity. This is left out in this paper due to space limitation.

[4] The complexity values for basic substitutions have proven to be useful in practice. Here, the analysis of different values is subject of future work.

```
function $anti\_unify(left\_term, right\_term)$ returns $generalization$
   variables:
      $Open$: priority queue of $anti\_instance$
      $Closed$: set of $anti\_instance$
      // $anti\_instance$ is a quadruple $\langle$complexity,substitution,generalized_term,index$\rangle$
      // with index $\in \{$"left", "right"$\}$ and $op($"left"$) =$ "right", $op($"right"$) =$ "left"
   initialize:
      $Open = \{\langle 0, id, left\_term, \text{"left"}\rangle, \langle 0, id, right\_term, \text{"right"}\rangle\}$
      $Closed = \{\}$
   while $Open \neq \{\}$ do
      $\langle c, \sigma, g, i\rangle = \texttt{first}(Open)$; $Open = \texttt{rest}(Open)$
      if $\langle \_, \theta, g, op(i)\rangle \in Open \cup Closed$ return $\langle g, \sigma, \theta\rangle$
      $G = \{\langle c', \sigma', g', i\rangle \mid \exists \tau \in basic\_substitutions : g' \xrightarrow{\tau} g, c' = c + \mathcal{C}(\tau), \sigma' = \sigma \circ \tau\}$
      $Open = \texttt{merge}(Open, G)$
      $Closed = Closed \cup \{\langle c, \sigma, t, i\rangle\}$
   end while
end function
```

Fig. 4. Algorithm for computing preferred generalizations

from the other domain, a preferred generalization has been found. Otherwise all anti-instances are computed that can be reached by a single (inverse) application of a basic substitution. These anti-instances are annotated with their complexity and merged back into queue and the whole process starts again.

The anti-instances produced have a decreasing information load and therefore the algorithm is guaranteed to terminate. However, the preferred generalization that is computed may depend on the complexity measure $\mathcal{C}$ that is chosen.

5 Example

This section explains our approach step-by-step with the heat flow analogy. Fig. 5 shows the anti-unification of the terms $connects(beaker, vial, pipe)$ from Th_S and $connected(in(coffee, cup), b\_cube, bar)$ from Th_T with required substitutions. The remaining generalizations are analogous to the anti-unification of $height(in(water, beaker), t_1) > height(in(water, vial), t_1)$ and $temp(in(coffee, cup), t_1) > temp(b\_cube, t_1)$.

Three basic substitutions are needed to transform $connected(A, B, C)$ to the domain-specific term in the source: the fixation ϕ^A_{beaker} substitutes the variable A to the constant $beaker$ (the same for B and C). The substitution the target side is more complex, because A maps on the complex structure $in(coffee, cup)$. $\iota^{A,A'}_{X,0}$ replaces variable A by the complex structure $A'(X)$ and $\iota^{A',A''}_{Y,0}$ inserts the second variable Y. Afterwards the variables are fixated to the required constants. The substitutions have the complexity $\mathcal{C} = 3$ on the source and $\mathcal{C} = 7$ on the target side. To transform $T(A, t_1) > T(B, t_2)$ in domain specific terms, 7 basic substitutions are required on the source and 7 on the target side. However, many

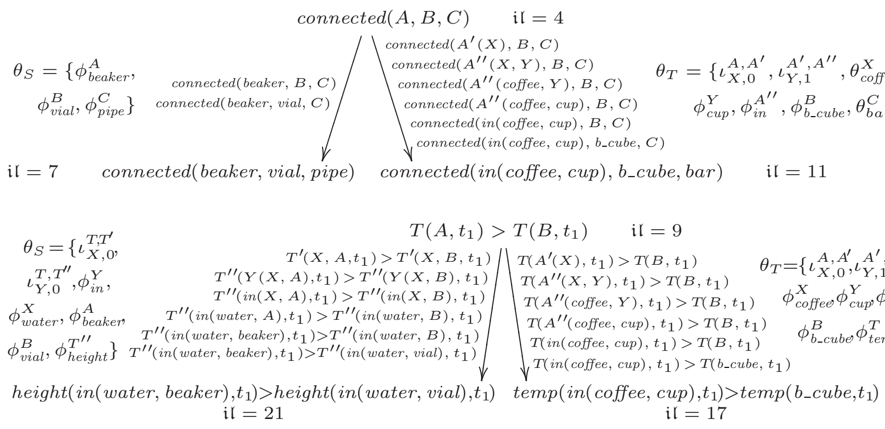
$connected(A, B, C)$ $\mathfrak{il} = 4$
$\theta_S = \{\phi^A_{beaker}, \phi^B_{vial}, \phi^C_{pipe}\}$
$connected(beaker, B, C)$
$connected(beaker, vial, C)$
$connected(A'(X), B, C)$
$connected(A''(X, Y), B, C)$
$connected(A''(coffee, Y), B, C)$
$connected(A''(coffee, cup), B, C)$
$connected(in(coffee, cup), B, C)$
$connected(in(coffee, cup), b\_cube, C)$
$\theta_T = \{\iota^{A,A'}_{X,0}, \iota^{A',A''}_{Y,1}, \theta^X_{coffee}, \phi^Y_{cup}, \phi^{A''}_{in}, \phi^B_{b\_cube}, \theta^C_{bar}\}$
$\mathfrak{il} = 7$ $connected(beaker, vial, pipe)$
$connected(in(coffee, cup), b\_cube, bar)$ $\mathfrak{il} = 11$
$T(A, t_1) > T(B, t_1)$ $\mathfrak{il} = 9$
$\theta_S = \{\iota^{T,T'}_{X,0}, \iota^{T,T''}_{Y,0}, \phi^Y_{in}, \phi^X_{water}, \phi^A_{beaker}, \phi^B_{vial}, \phi^{T''}_{height}\}$
$T'(X, A, t_1) > T'(X, B, t_1)$
$T''(Y(X, A), t_1) > T''(Y(X, B), t_1)$
$T''(in(X, A), t_1) > T''(in(X, B), t_1)$
$T''(in(water, A), t_1) > T''(in(water, B), t_1)$
$T''(in(water, beaker), t_1) > T''(in(water, B), t_1)$
$T''(in(water, beaker), t_1) > T''(in(water, vial), t_1)$
$T(A'(X), t_1) > T(B, t_1)$
$T(A''(X, Y), t_1) > T(B, t_1)$
$T(A''(coffee, Y), t_1) > T(B, t_1)$
$T(A''(coffee, cup), t_1) > T(B, t_1)$
$T(in(coffee, cup), t_1) > T(B, t_1)$
$T(in(coffee, cup), t_1) > T(b\_cube, t_1)$
$\theta_T = \{\iota^{A,A'}_{X,0}, \iota^{A',A''}_{Y,1}, \phi^X_{coffee}, \phi^Y_{cup}, \phi^{A''}_{in}, \phi^B_{b\_cube}, \phi^T_{temp}\}$
$height(in(water, beaker), t_1) > height(in(water, vial), t_1)$ $\mathfrak{il} = 21$
$temp(in(coffee, cup), t_1) > temp(b\_cube, t_1)$ $\mathfrak{il} = 17$

Fig. 5. Computing the generalizations for the heat flow analogy

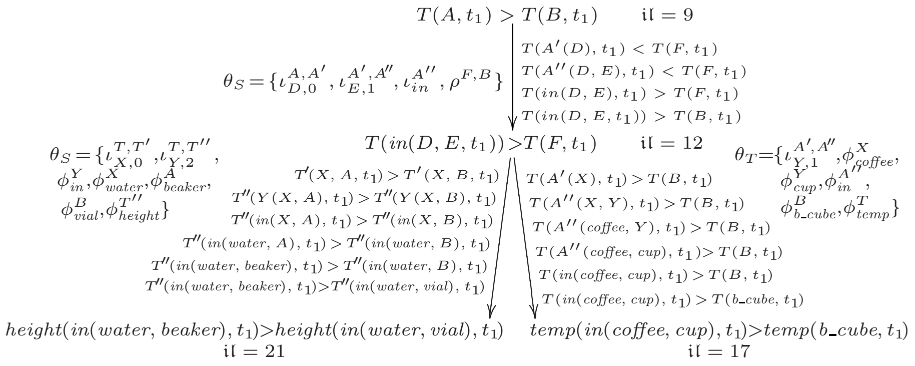
$T(A, t_1) > T(B, t_1)$ $\mathfrak{il} = 9$
$\theta_S = \{\iota^{A,A'}_{D,0}, \iota^{A',A''}_{E,1}, \iota^{A''}_{in}, \rho^{F,B}\}$
$T(A'(D), t_1) < T(F, t_1)$
$T(A''(D, E), t_1) < T(F, t_1)$
$T(in(D, E), t_1) > T(F, t_1)$
$T(in(D, E, t_1)) > T(B, t_1)$
$T(in(D, E, t_1)) > T(F, t_1)$ $\mathfrak{il} = 12$
$\theta_S = \{\iota^{T,T'}_{X,0}, \iota^{T,T''}_{Y,2}, \phi^Y_{in}, \phi^X_{water}, \phi^A_{beaker}, \phi^B_{vial}, \phi^{T''}_{height}\}$
$T'(X, A, t_1) > T'(X, B, t_1)$
$T''(Y(X, A), t_1) > T''(Y(X, B), t_1)$
$T''(in(X, A), t_1) > T''(in(X, B), t_1)$
$T''(in(water, A), t_1) > T''(in(water, B), t_1)$
$T''(in(water, beaker), t_1) > T''(in(water, B), t_1)$
$T''(in(water, beaker), t_1) > T''(in(water, vial), t_1)$
$T(A'(X), t_1) > T(B, t_1)$
$T(A''(X, Y), t_1) > T(B, t_1)$
$T(A''(coffee, Y), t_1) > T(B, t_1)$
$T(A''(coffee, cup), t_1) > T(B, t_1)$
$T(in(coffee, cup), t_1) > T(B, t_1)$
$T(in(coffee, cup), t_1) > T(b\_cube, t_1)$
$\theta_T = \{\iota^{A',A''}_{Y,1}, \phi^X_{coffee}, \phi^Y_{cup}, \phi^{A''}_{in}, \phi^B_{b\_cube}, \phi^T_{temp}\}$
$height(in(water, beaker), t_1) > height(in(water, vial), t_1)$ $\mathfrak{il} = 21$
$temp(in(coffee, cup), t_1) > temp(b\_cube, t_1)$ $\mathfrak{il} = 17$

Fig. 6. Different generalizations with different information load and complexity of substitutions

substitutions are already available from the anti-unification of *connects*: only 5 new substitutions are required for the source and 1 new for the target.

The generalized term $T(A, t_1) > T(B, t_2)$ has the information load $\mathfrak{il} = 9$ and is actually not the *lgg* of the domain specific terms. Fig. 6 shows that the generalization $T(in(D, E), t_1) > T(F, t_1)$ with the $\mathfrak{il} = 12$ is more specific and an anti-instance of the domain-specific terms. However, none of the analogical mappings received by the anti-unification of *connects* is reused and therefore also none of the substitutions. The complexity of the required substitutions on the source side is $\mathcal{C} = 8$ and on the target side $\mathcal{C} = 4$. Since HDTP aims to anti-unify whole theories and reduce the complexity of substitutions across the whole domain-specific substitution, the solution presented in Fig. 5 is preferred.

6 Conclusion and Future Work

We have presented a restricted version of higher-order anti-unification and have proven that in this framework least general generalizations can be defined and are indeed more specific than in the first-order case. We proposed a measure for the complexity of substitutions and presented an algorithm that allows for computing preferred generalizations with respect to this complexity. We further showed that complexity can be useful in the context of analogy making, when not only pairs of terms, but sets of terms shall be anti-unified, as it can be a means to encourage the reuse of substitutions.

The complexity measure for substitutions has been successfully applied in practice. However, alternative definitions are possible and the impact of different modification is the subject of current investigations. The application of complexity to anti-unification of sets of formulas as sketched in section 5 will be further examined and an algorithmic framework will be developed.

References

1. Chalmers, D.J., French, R.M., Hofstadter, D.R.: High-level perception, representation, and analogy: A critique of artificial intelligence methodology. Journal of Experimental and Theoretical Artificial Intelligence 4(3), 185–211 (1992)
2. Forbus, K.D., Gentner, D., Markman, A.B., Ferguson, R.W.: Analogy just looks like high-level perception: why a domain-general approach to analogical mapping is right. Journal of Experimental & Theoretical Artificial Intelligence 10, 231–257 (1998)
3. Holyoak, K., Morrison, R. (eds.): The cambridge handbook on thinking and reasoning. Cambridge University Press, Cambridge (2005)
4. Oppenheimer: Analogy in science. American Psychologist 11, 127–135 (1956)
5. Gentner, D.: The mechanism of analogical learning. In: Vosniadou, S., Ortony, A. (eds.) Similarity and analogical reasoning, pp. 199–241. Cambridge University Press, Cambridge (1989)
6. Hofstadter, D.R., Mitchell, J.: The copycat project: A model of mental fluidity and analogy-making. In: Hofstadter, D.R., group, F.A.R. (eds.) Fluid Concepts and Creative Analogies. Basic Books, pp. 205–267 (1995)
7. Plotkin, G.D.: A note on inductive gneralization. Machine Intelligence 5, 153–163 (1970)
8. Gust, H., Kühnberger, K.U., Schmid, U.: Metaphors and heuristic-driven theory projection (hdtp). Theoretical Computer Science 354(1), 98–117 (2006)
9. Hasker, R.W.: The replay of program derivations. PhD thesis, Champaign, IL, USA (1995)
10. Pfenning, F.: Unification and anti-unification in the Calculus of Constructions. In: Sixth Annual IEEE Symposium on Logic in Computer Science, Amsterdam, The Netherlands, pp. 74–85 (1991)
11. Wagner, U.: Combinatorically restricted higher order anti-unification. An application to programming by analogy. Master's thesis, Technische Universität Berlin (2002)