Solving Proportional Analogies by *E*–Generalization

Stephan Weller and Ute Schmid

Department of Information Systems and Applied Computer Science,
Otto-Friedrich-University, Bamberg
{Stephan.Weller,Ute.Schmid}@wiai.uni-bamberg.de

Abstract. We present an approach for solving proportional analogies of the form $A : B :: C : D$ where a plausible outcome for D is computed. The core of the approach is E–Generalization. The generalization method is based on the extraction of the greatest common structure of the terms A, B and C and yields a mapping to compute every possible value for D with respect to some equational theory. This approach to analogical reasoning is formally sound and powerful and at the same time models crucial aspects of human reasoning, that is the guidance of mapping by shared roles and the use of re-representations based on a background theory. The focus of the paper is on the presentation of the approach. It is illustrated by an application for the letter string domain.

1 Introduction

An often quoted observation by the psychologist William James more than a 100 years ago is that "a native talent for perceiving analogy is ... the leading fact in genius of every order" (see [1]). Accordingly, the process of analogy making is studied extensively in cognitive psychology as well as in artificial intelligence [2]. The most fundamental kind of analogies are so called proportional analogies of the form A : B :: C : D. They are studied in verbal settings (*Lungs are to humans as gills are to* [fish]), with geometric figures [3,4], and in the letter string domain [5] (*abc : abd :: kji :* [kjj]).

The core processes of all computational approaches to proportional analogy are (1) construction of a structured representation of the given patterns, usually in the form of terms, (2) identification/calculation of the relation between terms A and B, (3) mapping of terms A and C, and (4) application of the relation found between A and B to term C using substitutions based on the mapping of A and C. Some approaches, namely Pan [4] and Copycat [5], additionally present a mechanism for re-representation of terms, addressing the fact, that the outcome of mapping is dependent on the perceived structure of the given terms. For example, the string *abc* can be perceived as an arbitrary sequence of letters, or as an ascending sequence of three letters.

Our approach differs from the approaches named above in two respects: First, mapping is determined by the common *gestalt*, that is the structural communalities of the base (A) and target (C) terms. Second, arbitrary background

C. Freksa, M. Kohlhase, and K. Schill (Eds.): KI 2006, LNAI 4314, pp. 64–75, 2007.

knowledge can be considered when comparing the structures of terms. The first characteristic is covered by the method of syntactic anti-unification. The second characteristic is covered by an anti-unification modulo equational theory or E–Generalization.

In the following section we will introduce syntactic anti-unification and E–Generalization and discuss their computational advantages as well as relations to human analogy making. Afterwards we will introduce the letter string domain as example domain. We present our algorithmic approach and illustrate it for letter string analogies. We will conclude with an evaluation and further work to be done.

2 Syntactic Anti-unification and E–Generalization

In this section we will introduce the notion of E–Generalization, as used in [6] and [7]. To facilitate the understanding of E–Generalization, we will first introduce syntactic anti-unification, which is a proper subset of E–Generalization.

2.1 Syntactic Anti-unification

Unification is a well known and widely used technique, the probably most prominent application being the programming language *Prolog*. It computes the *most general unifier (MGU)* of two or more terms, i.e. the most general term, such that both terms can be reduced to the *MGU* by applying a substitution. Anti-unification is the dual concept to unification. Instead of computing the most general unifier, it computes the most specific generalization. It can be defined as follows:[1]

Definition 1 (Anti-instance). *Let u and $t_i, i = 1, \ldots, n$ be terms and σ_i substitutions for each term, such that $t_i = u\sigma_i \forall i = 1, \ldots, n$. Then u is called an* anti-instance *of the terms $t_1, \ldots, t_n$.*

u is called the most specific anti-instance *of $t_1, \ldots, t_n$, if for each term u' which is an anti-instance of $t_1, \ldots, t_n$ there exists a substitution θ, such that $u = u'\theta$.*

In contrast to unification, anti-unification is always possible and there is always a single most specific solution (up to variable renaming).

Algorithms for computing the anti-unification of n terms effectively were introduced in [8] and [9] independently.

Anti-unification can be used to establish a relation between two terms in the following way: If we anti-unify two terms, and those terms share some common structure, the result will be a non-trivial term containing some variables. Let us for example consider figure 1 [2], an example of anti-unifying two terms, $5 \cdot 3 + 7$ and $8 \cdot 3 + 9$. The two terms are anti-unified resulting in $x \cdot 3 + y$. The result of the

[1] As commonly used $t\sigma$ denoted the application of the substitution σ to the term t.
[2] Standard arithmetic rules are assumed, i.e. $5 \cdot 3 + 7$ is to be read as $+(\cdot(5, 3), 7)$.

anti-unification conserves as much of the structure of the terms as possible. The generalized structure reflects the roles the objects are playing in the respective expressions. For example, the variable x describes the role that 5 plays in the first term and 8 in the second, namely, that of the first factor in the term. The second term can be obtained from the first one by applying the substitution τ_1 inversely and then applying τ_2.

This is a contrast to a direct mapping approach used in many models for analogies. A direct mapping approach aims at computing the one term directly out of the other, without using intermediate results. This sometimes requires stochastic elements, as used in Copycat ([5]) or the use of heuristics, such as the systematicity principle in SME ([10]). Those may be powerful in solving the analogy, however a stochastical approach is psychologically hardly plausible.

Anti-unification allows for a mapping by using the abstract description of the "roles" some subterms fulfil. In one word, it allows for analogy via abstraction, which has some psychological motivation ([11]) and also yields a formally sound approach.

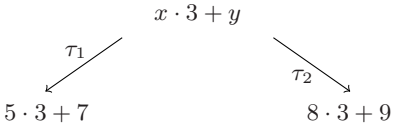

Fig. 1. Simple example for syntactic anti-unification

2.2 *E*–Generalization

Sometimes, for constructing a suitable abstraction, it might be necessary to include knowledge about the domain. E. g., for $t_2' = 9 + (3 \cdot 8)$ and t_1 as above, only the overly general anti-instance $u + v$ can be obtained. But we know, that addition is commutative and therefore, we can rewrite t_2' into its original form t_2. Knowledge about the equality of terms can be represented by an equational theory. The laws for addition constitute such a theory:

$$x + y =_E y + x$$
$$x + (y + z) =_E (x + y) + z.$$

If we use knowledge in form of equational background theories for rewriting terms before syntactical AU is performed, we speak of E–generalization. Models for solving proportional analogies, such as Copycat [5] or PAN [4], allow for an arbitrary sequence of rewritings of the initial representation (re-representation) to find a suitable solution. In contrast, E–generalization allows us to perform abstraction while modeling equivalent representations by appropriate equations between terms. All equivalent representations are considered *simultaneously* in the abstraction process. Therefore, abstraction becomes insensitive to representation changes.

The basic idea is to anti-unify *regular tree grammars* instead of terms. Regular tree grammars are a language class developed in 1968 (cf. [12] and [13]). This language class is located in the Chomsky-Hierarchy between regular and context-free languages (for a very comprehensive introduction to regular tree grammars see [14]).

Regular tree grammars allow for the representation of equivalence classes of terms, it would for example be possible to represent the terms $3 \cdot 8 + 9, 8 \cdot 3 + 9, 9 + 3 \cdot 8$, and $9 + 8 \cdot 3$ by one regular tree grammar, assuming the given background knowledge.

The construction of regular tree grammars from a background theory (for example a canonical equational theory) can in some cases be done automatically (cf. [15] and [16] for criteria when this is possible).

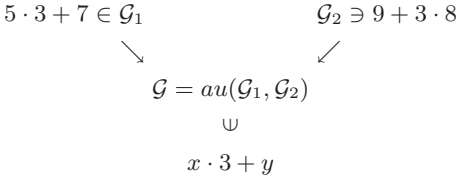

Fig. 2. Simple example for E–Generalization

Assuming regular tree grammars for our terms, we can now anti-unify these regular tree grammars by an algorithm originally developed in [6] and refined in [7]. This process is depicted in figure 2. Unfortunately, this algorithm needs exponential time [3] in general, but [7] shows that in some cases an efficient computation is nevertheless possible.

It should be noted that the result of this E–Generalization process is not a term, but a regular tree grammar of terms. But this is only a natural consequence of representing equivalence classes of terms as regular tree grammars. The result has to be an equivalence class of terms itself. In the next section we will see, that this will make it possible to compute *all* solutions of a proportional analogy in one step.

For an in-depth description of the algorithm mentioned (and also its implementation and application so proportional analogies) see [17].

E–Generalization may be used as a model for analogies even more than syntactic anti-unification. The features described in the last subsection are fulfilled by E–Generalization as well and additionally, we are not limited to one representation. E–Generalization may account for a change rerepresentation of the terms, which is necessary in many cases (see for example [18] for a justification of this claim). We can therefore hope to find a method of solving proportional analogies by E–Generalization.

[3] Exponential in the size of the grammars used.

3 Letter String Analogies

We will illustrate our approach for the letter string domain which has been widely investigated in cognitive science as well as in artificial intelligence [19,20,21,5,1]. This domain has several characteristics which makes it interesting: First, it is very simple and any number of analogies can be constructed. Second, for many examples, there are different plausible solutions. Which solution is generated (by a program or a human subject) is dependent on how the perceived structure of the other strings. Third, in principle any type of proportional analogy problem which can be represented in form of terms can be mapped to the letter string domain.

For example, geometrical patterns can be described by a system of terms, thus matching it to the letter-string-domain. This possibility was described already in 1971 in [22] and is known as *Structural Information Theory (SIT).*

It was first introduced as a coding system for linear one-dimensional patterns. Leeuwenberg represents perceptual structures by three operators named *iteration, alteration,* and *symmetry.* Iteration is supposed to reflect some kind of repetitive process (e.g. $Iter(xy, 3) := xyxyxy$). Symmetry should represent the reversed repetition of a term t after a second term s ($Sym(xyz, ()) := xyzzyx$). Finally, alteration describes the interleaving of a term into a list of terms, such as $Alt(a, (x, y, z)) := axayaz$.

On those operators, Leeuwenberg introduces the notion of *information load*, which is supposed to describe the complexity of an operator. Leeuwenberg claims that the descriptions using the minimal information load correspond to perceptual gestalts (for an introduction to gestalt theory, see [23]). His claim is therefore, that the gestalt principle can be explained by even simpler principles, such as his information load.

A more algebraic version of Structural Information Theory can be found in [21]. Here, even some computational modelling of proportional analogies is done.

4 Solving Proportional Analogies by *E*–Generalization

4.1 Illustration of the Approach

In the following we will show how to apply the method of E–Generalization to solve a proportional analogy of the form $A : B :: C : D$ (read: A is to B as C to D), where D is to be computed.

As an example, let us assume we want to solve the proportional string analogy $abc : abd :: ihg : D$. where D is unknown. Using a representation language similar to *SIT* ([21]) we could represent the term abc as $Iter(a, succ, 3)$, meaning that abc is established by iterating the successor operation three times on the constant a. Another possible representation would be of the form $a \cdot succ(a) \cdot succ(succ(a))$, where $\cdot$ means concatenation and $succ$ is the successor relation.

Our first aim is now to compute the common structure of the terms A and C, or, in our example the terms abc and ghi. At this point it should be noted,

that this common structure could be extracted straightforward by syntactic anti-unification, *if we had knowledge about the structure of the terms* abc *and* ghi. Let us for a moment assume, that we know that the structure of our both terms is $Iter(a, succ, 3)$ and $Iter(i, pred, 3)$ respectively [4]. In this case, an application of syntactic anti-unification would yield the common structure $Iter(x, y, 3)$ and the two substitutions $\tau_1 = \{x \leftarrow a, y \leftarrow succ\}, \tau_2 = \{x \leftarrow i, y \leftarrow pred\}$. Let us further assume the structure $Iter(a, succ, 2) \cdot succ(succ(succ(a))$ for the B-term *abd*. Given this, we could apply τ_1 inversely to the B-term, yielding a new term Q of the form $Iter(x, y, 2) \cdot y(y(y(x)))$. Applying τ_2 to this term would yield the result $Iter(i, pred, 2) \cdot pred(pred(pred(i)))$ which describes the term ihf, which is one possible solution of the analogy.

Seeing this, one possibility to solve a proportional string analogy would be to compute some representation of the participating terms and using syntactic anti-unification (and inverse and normal substitution application) to compute a result. The decision on some representation will thus determine, which result we will obtain.

But instead of choosing one particular representation at the start, we can take the process one step further and use E–Generalization instead of syntactic anti-unification. The complete process is shown in figure 3.

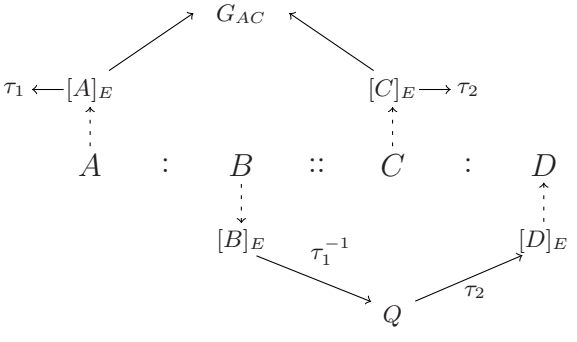

Fig. 3. Solving a proportional string analogy

To every ground term A, B, and C a regular tree grammar representing the equivalence class of all representations of the term is built up. Those regular tree grammars are denoted by $[A]_E, [B]_E$, and $[C]_E$. The process of building the regular tree grammars is denoted by the dotted arrows. Next, the E–Generalization algorithm from [7] is used on the two regular tree grammars $[A]_E$ and $[C]_E$, thus extracing their common structure as a regular tree grammar G_{AC}. This grammar is not needed in the further process, it is a byproduct of the algorithm representing a form of abstraction from the ground terms. What *is* needed in the next step, are the substitutions τ_1 and τ_2 also produced by the E–Generalization step.

[4] Actually, the example is not completely formally correct, as it intermixes first and second order terms for *succ* and *pred*, which is of course not valid, but simplifies the example a lot.

The substitution τ_1 is inversely applied to $[B]_E$ [5] , resulting in a regular tree grammar describing the common structure between B and D. It is denoted by Q in the figure. Again, this grammar forms an abstraction from the actual terms that can be seen as a byproduct of the analogical process.

Finally, the substitution τ_2 is applied to Q, leading to a final grammar $[D]_E$ with the following properties:

- It shares the structure of B, as it is derived by application of (inverse) substitutions to $[B]_E$.
- The constants occuring in the term A are replaced by those in C, as the two (inverse) substitutions τ_1^{-1} and τ_2 are applied.
- It can thus be described as the result of "doing the same thing" to B as it was done to A to get C.
- And therefore, it can be seen as a valid solution of the proportional analogy $A : B :: C : D$.

As mentioned before, the result of this process is not a single term, but a regular tree grammar and as such a whole set of terms. However, it describes *every* result that can be obtained by replacing constants in any representation of B by their counterparts with respect to every possible representation of A and C.

Naturally the question arises, how to extract a term from this tree grammar (when a single term is desired rather than a set of terms). This process is an enumeration of the regular tree grammar, which is in general not possible completely, as infinitely many terms are described by the grammar (imagine for example a grammar for arithmetical operations allowing for the addition of 0 - this alone leads to result grammars describing infinitely many terms in nearly every case). Nevertheless, it may be desirable to enumerate the first n terms according to some ordering relation. The enumeration is not a problem, if the ordering relation is defined. But finding a suitable ordering relation is not a simple task. However, using simple relations, like ordering by number of occuring constants, "depth" of the term etc. are possible. Those might even represent some kind of simplicity used by humans to decide which answer to choose in solving a proportional analogy.

Which ordering relation would correspond to the preference humans use is an empirical question and probably requires deep insight in the cognitive processes used in analogical thinking.

4.2 Algorithmic Realization of E-Generalization

The algorithm used to implement E–Generalization was originally developed in [7]. Computing the E–Generalization of two terms is split up in three parts: Computing *universal substitutions*, *lifting* a grammar and finally intersection of regular tree grammars.

[5] Note that (inverse) application of substitutions is well-defined on regular tree grammars. It is a special case of an inverse tree homomorphism. See [14] for details.

Regular tree grammars are defined as a quadruple $\mathcal{G} = (\Sigma, \mathcal{N}, S, \mathcal{R})$. Σ is a signature, i.e. a set of function symbols f, where each f has a fixed arity. If the arity of one f is 0, it is called a constant. $\mathcal{N}$ is a finite set of nonterminals, $S \in \mathcal{N}$ is a starting symbol. Finally, $\mathcal{R}$ is a finite set of rules of the following form:

$$\mathcal{N} ::= f_1(N_{11}, \ldots, N_{1n_1}) | \ldots | f_m(N_{m1}, \ldots, N_{mn_m})$$

Let us first assume, that the substitutions are already known. This subproblem is also known as *constrained* E–Generalization. In this case, only two steps are needed: *Lifting* the grammars and intersecting them. Lifting the grammar is a process to incorporate the knowledge about variables that could be used by the substitution. In other words, from a grammar G we want to derive a grammar G^σ, such that $G^\sigma\sigma = G$, where G^σ containts all "suitable" variables from the substitution σ. To this end, we define the following algorithm:

Algorithm 1 (Lifting). *For a regular tree grammar $\mathcal{G} = (\Sigma, \mathcal{N}, S, \mathcal{R}$ and a substitution σ define a new grammar $G^\sigma = (\Sigma \cup \text{ dom } \sigma, \{N^\sigma | N \in \mathcal{N}\}, S^\sigma, \mathcal{R}^\sigma)$,*[6] *where N^σ is a new nonterminal, one distinct nonterminal is introduced for each old nonterminal. The same is done to S and the rules $\mathcal{R}^\sigma$ are derived from $\mathcal{R}$ as follows:*

For every rule

$$N ::= \Big|_{i=1}^{m} f_i(N_{i1}, \ldots, N_{in_i})$$

from $\mathcal{R}$ we introduce a new rule

$$N^\sigma ::= \Big|_{i=1}^{m} f_i(N_{i1}, \ldots, N_{in_i}) \Big|_{x \in \text{ dom } \sigma, x\sigma \in \mathcal{L}_\mathcal{G}(N)} x$$

Where $\mathcal{L}_\mathcal{G}(N)$ describes all terms in the grammar $\mathcal{G}$, that can be reached when using N as a starting symbol.

To compute the intersection of the regular tree grammars, a standard algorithm from [14] is used. It is similar to intersection algorithms for regular languages.

Lifting two grammars as defined above and intersecting them afterwards will now result in the E–Generalization of those two grammars. However, to allow for *unconstrained* E–Generalization, we still need some means to extract the substitions required to apply constrained E–Generalization.

It is possible, to find two *universal substitutions* τ_1 and τ_2, which are universal in the following sense: For any two substitutions σ_1, σ_2 we can find a substitution σ, such that $t\sigma_i \in \mathcal{L}(N) \Longrightarrow t\sigma\tau_i \in \mathcal{L}(N)$ for $i = 1, 2, N \in \mathcal{N}$ and any t from the union of both domains.

The existence of such substitutions is proven in [7]. They are constructed in the following two steps:

[6] dom σ denotes the domain of σ, i.e. the set of all terms occuring on the left-hand side of a substitution.

Algorithm 2 (Universal substitutions)

1. *Construct* $\mathcal{N}_{\max}$*:*
   (a) *Set* $N = \emptyset$ *and* $\mathcal{N}_{\max} = \emptyset$
   (b) *For each Nonterminal* $n \in \mathcal{N}$*, compute* $(\bigcap_{x \in N} \mathcal{L}(x)) \cap \mathcal{L}(n)$ *and if the result is not empty, add* n *to* N*.*
   (c) *Add* N *to* $\mathcal{N}_{\max}$*, remove all elements in* N *from* $\mathcal{N}$*, and if* $\mathcal{N} \neq \emptyset$*, set* $N = \emptyset$ *and continue with step (b).*
2. *For a nonterminal* N*, define* $t(N)$ *as an arbitrary term from* $\mathcal{L}(n)$*. For each pair* $(N_1, N_2) \in \mathcal{N}_{\max} \times \mathcal{N}_{\max}$ *introduce a new variable* $v(N_1, N_2)$ *and define* $\tau_i = \{v(N_1, N_2) \leftarrow t(N_i)\}$ *for* $i = 1, 2$*.*

Using those substitutions and applying the constrained E–Generalization will yield the unconstrained E–Generalization. This completes the algorithm:

Algorithm 3 (Unconstrained E-Generalization). *Let* $\mathcal{N}_1, \mathcal{N}_2$ *be two regular tree grammar. Their unconstrained E-Generalization is computed with the following steps:*

1. *Compute two universal substitutions* τ_1, τ_2 *for the regular tree grammars* $\mathcal{N}_1$ *and* $\mathcal{N}_2$ *respectively by algorithm 2.*
2. *Compute the "lifted" grammars* $\mathcal{N}_1^{\tau_1}$ *and* $\mathcal{N}_2^{\tau_2}$ *by algorithm 1.*
3. *Compute the intersection* $\mathcal{N} := \mathcal{N}_1^{\tau} \cap \mathcal{N}_2^{\tau}$ *by a standard algorithm, for example from [14].*

4.3 Using E-Generalization to Solve Proportional Analogies

We have now all ingredients to describe our overall approach algorithmically:

Algorithm 4 (Solving proportional analogies). *Let a proportional analogy of the form* $A : B :: C :?$ *be given by regular tree grammars* $[A]_E$ *for* A*,* $[B]_E$ *for* B *and* $[C]_E$ *for* C*. Compute a solution* D *by the following steps:*

1. *Compute universal substitutions* τ_1, τ_2 *for* $[A]_E$ *and* $[C]_E$ *respectively, using algorithm 2.*
2. *Lift the grammar* $[B]_E$ *with respect to the substitution* τ_1*, using algorithm 1 to get* $Q := [B]_E^{\tau_1}$*.*
3. *Apply the substitution* τ_2 *to* Q *to get the final result* $[D]_E := Q\tau_1$*.*

4.4 Implementation

As mentioned above, the E–Generalization algorithm is exponential in the size of the grammars used in the general case. A proof-of-concept implementation of the algorithm was done, which can be used for small examples. Grammars describing terms like *abd* have typically a size of 30 to 40 rules, one for each letter of the alphabet and several more for the operators (see [17] for example grammars).

The implementation was done in Moscow-ML, an implementation of Standard ML, which is a stricly functional language. This language was selected due to it's support for pattern matching, which enables it to interact with trees and terms very straightforwardly.

Sample grammars were generated for very small mathematical problems and proportional string analogies.

The program was run with prototypical examples, such as $abc : abd :: ghi :?$. Performance time is quite reasonable, typically about 30 sec. The size of the returned grammar was in between 40 and 50 rules. Nearly all of the computing time was spent in the process of generating the universal substitutions. This coincides with the theoretical results ([7]), as this is the step that requires exponential computation effort. More statistics on the program were not done, due to its prototypical nature.

An algorithm to enumerate the resulting grammar is not yet available. As the grammar represents *all* solutions with respect to a given background knowledge, such an enumeration would reflect the *preference* human subjects would have when choosing a solution. The question of enumerating the grammar is therefore more a psychological than an algorithmical one. It is possible to sort the terms in the grammar by complexity (i.e. the depth of the terms). However, wether such an enumeration would correspond to human preference is still an open question.

A more extensive description of the implementation can be found in [17].

For an application to a task which requires more than very small grammars, some restriction to the algorithm is inevitable. Such restrictions would of course depend on the application.

5 Conclusion and Further Work

We have introduced the idea of anti-unification and its extension with background knowledge to E–Generalization. Then we have shown how this method can be applied to solve proportional string analogies in a generic way, that is, without incorportating domain knowledge into the algorithm. We demonstrated the approach for the letter string domain, which can be seen as a representative domain for all other domains accessible to term representation. That is, our approach is applicable to all kinds of proportional analogy problems. The only restriction is given by the fact that the background knowledge has to be represented as a canonical equational theory, which is not always possible (for criteria cf. [6]).

Our approach can be applied to more complex analogical reasoning tasks as well. For example, it can be applied to solve predictive analogies in the domain of naive physics which are addressed in the cognitive model SME [10] as we demonstrated in [11]. Furthermore, an extension to second-order generalization can be applied to the domain of program construction by analogy [24,25].

Solving proportional analogies is only one domain where E–Generalization can be applied. There are for example applications in the field of lemma generation (cf. [6]) or in the completion of number series, as they are used often in

intelligence tests. The latter were also worked on by [5] (cf. chapter 2.2), also making use of analogies. The application of E–Generalization to this problem has been done in [15].

An important property of this method is the calculation of the common structure of the terms as a byproduct of the analogy solving. In contrast to most other models of analogies, this method does account for the emergence of abstract knowledge without any extra computation. The creation of the abstract knowledge about the common structure is not gained by an extra step, but rather as an intrinsic property of the process.

At least in this aspect this is similar to the way humans solve analogies. One cannot "suppress" the abstraction from the concrete terms. To learn something about the common structure of the terms is inherent in the process of solving the analogy.

To investigate further in human solving of proportional analogies, empirical research is necessary. Our next step will therefore be to conduct an empirical study. The aim of this study will be to check wether the results chosen by humans correspond to a certain ordering relation of the terms in the computed grammar or wether terms occur not covered by the grammar at all (this can of course not be ruled out, as human decisions might not be explicable by background knowledge but rather based on intuition or other non-rational processes).

Acknowledgements

We would like to thank Jochen Burghardt for his support of our work.

References

1. Mitchell, M.: Analogy-Making as Perception: A Computer Model. MIT Press, Cambridge, MA (1993)
2. French, R.: The computational modeling of analogy-making. Trends in Cognitive Sciences **6** (2002) 200–205
3. Evans, T.G.: A Program for the Solution of a Class of Gemetric-Analogy Intelligence-Test Questions. In Minsky, M., ed.: Semantic Information Processing. MIT Press (1968) 271–353
4. O'Hara, S.: A model of the redescription process in the context of geometric proportional analogy problems. In: Int. Workshop on Analogical and Inductive Inference (AII '92). Volume 642., Springer (1992) 268–293
5. Hofstadter, D., the Fluid Analogies Research Gr.: Fluid Concepts and Creative Analogies. BasicBooks (1995)
6. Heinz, B.: Anti-Unifikation modulo Gleichungstheorie und deren Anwendung zur Lemmagenerierung. Technical report, GMD - Forschungszentrum Informationstechnick GmbH (1996)
7. Burghardt, J.: E-generalization using grammars. Artificial Intelligence Journal **165** (2005) 1–35
8. Plotkin, G.: A note on inductive generalization. In: Machine Intelligence. Volume 5. Edinburgh University Press (1970) 153–163

9. Reynolds, J.: Transformational Systems and the Algebraic Structure of Atomic Formulas. In: Machine Intelligence. Volume 5. Edinburgh University Press (1970)
10. Falkenheimer, B., Forbus, K.D., Gentner, D.: The structure-mapping engine: Algorithm and examples. Artificial Intelligence **41** (1989) 1–63
11. Schmid, U., Gust, H., Kühnberger, K.U., Burghardt, J.: An algebraic framework for solving proportional and predictive analogies. In Schmalhofer, F., Young, R., Katz, G., eds.: Proceedings of the First European Conference on Cognitive Science (EuroCogSci03), Mahwah, NJ, Lawrence Erlbaum (2003) 295–300
12. Brainerd, W.: The minimalization of tree automata. Information and Control **13** (1968) 484–491
13. Thatcher, J., Wright, J.: Generalized finite automata theory with an application to a decision problem of second–order logic. Mathematical Systems Theory **2** (1968)
14. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata` (1997) release October, 1rst 2002.
15. v. Thaden, M., Weller, S.: Lösen von Intelligenztestaufgaben mit E-Generalisierung (Solving intelligence tasks by E-Generalization). In: Tagungsband der Informatiktage 2003, Gesellschaft für Informatik e.V. (2003) 84–87
16. Emmelmann, H.: Code Selection by Regularly Controlled Term Rewriting. In: Proc. of Int. Workshop on Code Generation. (1991)
17. Weller, S.: Solving Proportional Analogies by Application of Anti-Unification modulo Equational Theory. Available on `http://www-lehre.inf.uos.de/~stweller/ba/` (2005) Bachelor's Thesis, unpublished.
18. Yan, J., Gentner, D.: A theory of rerepresentation in analogical matching. In: Proc. of the 25th Annual Conference of the Cognitive Science Society, Mahwah, NJ, Erlbaum (2003)
19. Burns, B.: Meta-analogical transfer: Transfer between episodes of analogical reasoning. Journal of Experimental Psychology: Learning, Memory, and Cognition **22** (1996) 1032–1048
20. Cornuejlos, A.: Analogy as minimization of description length. In Nakhaeizadeh, N., Taylor, C., eds.: Machine Learning and Statistics. The Interface. Wiley, New York (1997) 321–335
21. Dastani, M., Indurkhya, B., Scha, R.: An Algebraic Approach to Modeling Analogical Projection in Pattern Perception. In: Proceedings of Mind II. (1997)
22. Leeuwenberg, E.: A perceptual coding language for visual and auditory patterns. American Journal of Psychology **84** (1971) 307–349
23. Goldstein, E.B.: Sensation and Perception. Wadsworth Publishing Co., Belmont, California (1980)
24. Hasker, R.W.: The Replay of Program Derivations. PhD thesis, Univ. of Illinois at Urbana-Champaign (1995)
25. Schmid, U., Sinha, U., Wysotzki, F.: Program reuse and abstraction by anti-unification. In: Professionelles Wissensmanagement – Erfahrungen und Visionen, Shaker (2001) 183–185 Long Version: `http://ki.cs.tu-berlin.de/~schmid/pub-ps/pba-wm01-3a.ps`