ACUOS: A System for Modular ACU Generalization with Subtyping and Inheritance*

M. Alpuente[1], S. Escobar[1], J. Espert[1], and J. Meseguer[2]

[1] DSIC-ELP, Universitat Politècnica de València, Spain
{alpuente,sescobar,jespert}@dsic.upv.es
[2] University of Illinois at Urbana-Champaign, USA
meseguer@illinois.edu

Abstract. Computing generalizers is relevant in a wide spectrum of automated reasoning areas where analogical reasoning and inductive inference are needed. The ACUOS system computes a complete and minimal set of semantic generalizers (also called "anti-unifiers") of two structures in a typed language *modulo* a set of equational axioms. By supporting types and any (modular) combination of associativity (A), commutativity (C), and unity (U) algebraic axioms for function symbols, ACUOS allows reasoning about typed data structures, e.g. lists, trees, and (multi-)sets, and typical hierarchical/structural relations such as *is_a* and *part_of*. This paper discusses the modular ACU generalization tool ACUOS and illustrates its use in a classical artificial intelligence problem.

1 Introduction

Generalization is the dual of unification [14]. Roughly speaking, in this work the generalization problem for two expressions t_1 and t_2 means finding their *least general generalization* (lgg), i.e., the least general expression t such that both t_1 and t_2 are instances of t under appropriate substitutions. For instance, the expression `father(X,Y)` is a generalizer of both `father(john,sam)` and `father(tom,sam)`, but their least general generalizer, also known as *most specific generalizer* (msg) and *least common anti–instance* (lcai), is `father(X,sam)`. Applications of generalization arise in many artificial intelligence areas, including case-based reasoning, analogy making, web and data mining, ontology learning, machine learning, theorem proving, and inductive logic programming, among others [5,12,13,16].

While ordinary, syntactic generalization is useful for some applications, it has two important limitations. First, it cannot generalize common data structures such as records, lists, trees, or (multi-)sets, which satisfy specific premises such as the order among the elements in a set being irrelevant. For instance, let

* This work has been partially supported by the EU (FEDER) and the Spanish MINECO under grants TIN 2010-21062-C02-02 and TIN 2013-45732-C4-1-P, by Generalitat Valenciana PROMETEO2011/052, and by NSF Grant CNS 13-10109. J. Espert has also been supported by the Spanish FPU grant FPU12/06223.

us introduce the constants `john`, `sam`, `peter`, `tom`, `mary`, `chris`, and `joan`, and consider the predicate symbols `twins`, `ancestors`, `spouses`, and `children` that establish several relations among (a selection of) such constants. Since `twins` is a symmetric relation, we would like the pair "`john` and `sam`" to be in the relation `twins` if the pair "`sam` and `john`" is in the relation `twins`. For the time being, let us introduce a new *tuple constructor* symbol (;) to satisfy commutativity and an overloaded use of `twins` as a unary symbol such that the expressions `twins((john;sam))` and `twins((sam;john))` are equivalent *modulo* the commutativity of the (;) operator. Then, we can generalize `twins((john;sam))` and `twins((sam;tom))` as `twins((X;sam))`, whereas without equational attributes the least general (or most specific) generalizer of `twins(john,sam)` and `twins(sam,tom)` is `twins(X,Y)`.

Similarly, we can express the relation given by the ancestors of a person by means of a list using the *list concatenation* operator (.). We assume that a person's name is automatically coerced into a singleton list. Due to the associativity of list concatenation, i.e.,`(x.y).z = x.(y.z)`, we can use the flattened list `(john.sam.mary.peter)` as a very compact and convenient representation for the congruence class modulo associativity whose members are the different parenthesized list expressions, e.g., `((john.sam).mary).peter`, `john.(sam.mary).peter`, `john.(sam.(mary.peter))`, etc. Then, for the expressions `ancestors(chris,(john.sam.mary.peter))` and `ancestors(joan,(tom.mary.john))`, the least general generalizer is `ancestors(X,(Y.mary.Z))`, which reveals that `mary` is the only common ancestor of `chris` and `joan`. Note that `ancestors(chris,(john.sam.mary.peter))` is an instance (modulo A) of `ancestors(X,(Y.mary.Z))` by the substitution {`X/chris, Y/(john.sam), Z/peter`}.

Due to the equational axioms, in general there can be more than one least general generalizer of two expressions. For instance, let us record the marriage history of a person using a list, e.g. `sam.sam.tom.peter` for the marriage history of `mary`, where she divorced `sam` and married him again. Then, the expressions `spouses(mary,(sam.sam.tom.peter))` and `spouses(joan,(tom.tom.john))` have two incomparable least general generalizers: (a) `spouses(X, (Y.tom.Z))` and (b) `spouses(U, (V.V.W))`, respectively meaning that both `mary` and `joan` have married `tom`, and they both repeated marriage (consecutively) with their first husband. Note that the two generalizers are least general and incomparable, since neither one is an instance (modulo associativity) of the other.

Furthermore, if we consider the set of children of a person, this set should be recognized irrespectively of the order in which the children's names are written in the set. Let us introduce a new symbol (&) that satisfies associativity, commutativity, and unit element $\emptyset$; i.e., `X & ∅ = X` and `∅ & X = X`. Then, we can use the flattened multiset `(john & mary & peter & sam)` (with a total order on elements given, e.g., by the lexicographic order) as a very compact and convenient representation for the congruence class modulo associativity, commutativity, and unit element (written ACU) whose members are the different parenthesized expressions with all permutations of the elements and

as many occurrences of $\emptyset$ as needed, due to unity [6]. Working modulo ACU, the expressions (i) `children(chris,(john & sam & mary & peter))` and (ii) `children(joan,(tom & sam & john))` can be generalized as `children(P,(john & sam & X))` but they can also be generalized as `children(P',(john & sam & X' & Y))` since `children(joan,(tom & sam & john))` is an instance (modulo ACU) of `children(P',(john & sam & X' & Y))` by the substitution {`P'/joan, X'/tom, Y/`$\emptyset$}. Actually, for every least general generalizer t, the set of all ACU generalizers that are equivalent to t *modulo* ACU-renaming[3] is infinite, i.e.,

```
children(P0, (john & sam & X0)),
children(P1, (john & sam & X1 & Y1)),
children(P2, (john & sam & X2 & Y2 & Z2)), ...
```

yet we can choose one of them, typically the smallest one, as the class representative. Note that `children(P,(john & sam & X))` is an instance (modulo ACU) of `children(P',(john & sam & X' & Y))` by the substitution {`X'/X, Y/`$\emptyset$} but also `children(P',(john & sam & X' & Y))` is an instance (modulo ACU) of `children(P,(john & sam & X))` by the substitution {`X/(X' & Y)`}.

The second problem with ordinary generalization is that it does not cope with types and subtypes, which can lead to more specific generalizers. For instance, assume that the constants `john`, `sam`, `peter`, and `tom` belong to type Male and that `mary`, `joan`, and `chris` belong to type Female. Let us introduce another type People for the typed version of the ACU (multi-)set structures on which the relation `children` described above is defined. The Male and Female types can be considered as subtypes of a common type Person, which is itself a subtype of People representing a singleton set. Subtyping implies automatic coercion. Note that the empty set, denoted by $\emptyset$, belongs to People. Then, the above expressions (i) and (ii) have one typed ACU least general generalizer `children(P:Female,(john & sam & X:Male & Y:People))` that we choose as the representative of the infinite ACU congruence class. Note that `children(P':Female,(john & sam & X':People))` is not a least general generalizer since it is strictly more general; it suffices to see that the class representative is an instance of it with substitution {`P':Female/P:Female,X':People/(X:Male & Y:People)`}.

This work presents ACUOS, a mature and highly developed implementation of the order-sorted ACU least general generalization algorithm that we formalized in [1]. ACUOS has been written in the high-performance language Maude [11] that supports reasoning modulo algebraic properties and reflection. To the best of our knowledge, this is the first generalization system that is able to compute least general generalizers in order-sorted theories modulo equational axioms.

In Section 2, we describe the system and discuss how it can be used to address artificial intelligence problems that need a form of ACU generalization. This is done by focusing on a simple and classical artificial intelligence problem that is known as the Rutherford analogy [8,9], proving that our system fulfills

[3] i.e., the equivalence relation $\approx_{ACU}$ induced by the relative generality (subsumption) preorder $\leq_{ACU}$: $s \approx_{ACU} t$ iff $s \leq_{ACU} t$ and $t \leq_{ACU} s$.

the objective to recognize that atoms resemble tiny solar systems. Experimental results given in Section 3 show that ACUOS performs efficiently in practice. For a discussion of the related literature, we refer to [2,3,1,4,10]

2 Use Case: Extracting Analogies

In this section, we analyze and extract structural commonalities between two representative sets of physical assertions, one of which regards the electromagnetic forces in the atom while the other one considers gravitational forces in the solar system. First, we provide a functional representation for the solar system and the Rutherford model for the atom and then we use ACUOS to automatically extract a precise correspondence between them. Note that this is a classical example of *higher-order generalization* [8], in the sense that function symbols themselves are generalized by using function variables. We explain how higher-order reasoning can be achieved within our first-order setting by using reflection through the Maude meta-programming capabilities [7].

2.1 Problem representation

Let us introduce a meta-representation for models by introducing the HModel sort (or type) that is defined in Figure 1, using (sub-)sorts HTerm and HOperator. The generic Maude implementation given in Figure 1 is then used in Figure 2 to specify the operators that describe the two considered systems (i.e., the domain relations). Each relation r such as `mass`, `charge`, or `attraction` is represented by an HTerm that is rooted by a suitable operator that is given appropriate equational axioms, similarly to the operators[4] (`;`), (`.`), and (`&`) discussed in Section 1. In other words, the semantic information concerning each domain is encoded using appropriate equational attributes for the relation r itself (e.g., the action-reaction principle of gravitational forces is captured by the commutativity property of the `attraction` operator). In Maude syntax, this can be done by declaring the equational attributes of any given symbol through the use of special tags. Not only is this concise, it is also efficient because it takes advantage of the powerful optimizations included in the Maude interpreter [6].

Maude syntax is almost self-explanatory, using explicit keywords such as `fmod`, `sort`, and `op` to introduce a module, sort, and operator, respectively. The declaration `subsort A1 ... An < B` denotes that A1 ... An are subsorts of B and implies automatic coercion. The keywords `assoc` and `comm` respectively specify associativity and commutativity axioms for an operator. The keyword `prec` establishes the precedence of an operator. Module inclusion is denoted by `inc`. Using this representation, our knowledge of each domain can simply be encoded as a first-order term of sort HTerm, as shown in Figure 3, which depicts the two terms that respectively encode the gravitational solar system and the Rutherford model for the atom.

[4] Notice the *mixfix* notation [6] in the definition of the operators (e.g., `op _;_ : HModel HModel -> HModel`), which uses underscores `_` to indicate that each argument of the function will replace one of the underscores (e.g., the term `(x;y)`).

```
fmod HIGHER-ORDER-metarepresentation is
  sorts HModel HTerm HOperator HVariable .
  sorts HTermList HTermPair HConj HRule .
  subsort HOperator HVariable < HTerm .
  subsort HTerm < HTermList HConj HModel .
  subsort HRule < HTerm .
  op _[_]  : HOperator HTermList -> HTerm [prec 10] .
  op __    : HOperator HTermPair -> HTerm [prec 10] .
  op _,_   : HTermList HTermList -> HTermList [assoc prec 20] .
  op <_,_> : HTerm HTerm -> HTermPair [comm prec 20] .
  op _/\_  : HConj HConj -> HConj [assoc comm prec 30] .
  op _=>_  : HConj HTerm -> HRule [prec 40] .
  op _;_   : HModel HModel -> HModel [assoc comm prec 50] .
endfm
```

Fig. 1: Generic higher-order meta-representation

```
fmod DOMAIN-OPERATORS is inc HIGHER-ORDER .
  ops mass sun planet gravity : -> HOperator .
  ops charge coulomb electron nucleus : -> HOperator .
  ops attraction distant : -> HOperator [comm] .
  ops x y : -> HVariable .
endfm
```

Fig. 2: Signature of the analogy domain operators

Solar System	**Rutherford Atom Model**
mass[sun] ;	charge[y] $\wedge$ charge[x] $\Rightarrow$ coulomb[x,y] ;
mass[planet] ;	charge[electron] ;
distant$\langle$sun,planet$\rangle$;	charge[nucleus] ;
mass[x] $\wedge$ mass[y] $\Rightarrow$ gravity[x,y] ;	distant$\langle$electron,nucleus$\rangle$;
gravity[x,y] $\Rightarrow$ attraction[x,y]	coulomb[x,y] $\Rightarrow$ attraction[x,y]

Fig. 3: Analogy problem representation

After feeding the ACUOS generalization tool with the Maude specification given in Figures 1 and 2, together with the two input terms of Figure 3, we obtain the least general ACU generalizer shown in Figure 4. For clarity, we omit the sorting information in the results and summarize it as an annotation at the bottom of the figure.

Generalization of Solar System and Rutherford Atom
P[X] ;
P[Y] ;
distant$\langle$X,Y$\rangle$;
P[x] $\wedge$ P[y] $\Rightarrow$ Q[x,y] ;
Q[x,y] $\Rightarrow$ attraction[x,y]

where variables P, Q belong to sort HOperator and variables X, Y to sort HTerm; note that P,Q encode higher-order variables in our first-order setting.

Fig. 4: ACU generalization of the analogy problem

2.2 Further generalization capabilities

The analogy extracted so far relates a planet in the solar system with an electron in the atom, and the Sun with the atom nucleus. The related entities `planet` and `electron` are the only argument of the relations `mass` and `charge`, respectively. However, they both appear as arguments of the relations `gravity` and `coulomb`, though in different order. Also, the order of appearance of the definitions for the relations `coulomb` and `gravity` differs in both models. Therefore, the correspondence between the two models would have been hard to establish without considering the commutativity and associativity of the operators $(\_\wedge\_)$ and $(\_;\_)$.

We must often extract analogies from large deductive databases that, unlike our previous example, contain irrelevant information with respect to the analogies that we intend to extract. Let us further illustrate the advantages of our order-sorted, equational generalization approach by slightly modifying our example with the introduction of irrelevant knowledge. Specifically, suppose that we add the assertions `positive(nucleus)` and `negative(electron)` into the Rutherford Atom description and the assertion `heavier-than(sun,planet)` into the solar system model. Figure 5 below shows the extended domain representation together with the recomputed least general generalization result; the only difference is the addition of a variable Z (of sort HModel), which can be thought of as a container for the unnecessary pieces of information that are automatically disregarded in this case.

Extended Solar System	**Extended Rutherford Atom Model**
mass[sun] ; mass[planet] ;	charge[y] $\wedge$ charge[x] $\Rightarrow$ coulomb[x,y] ;
distant$\langle$sun,planet$\rangle$;	charge[electron] ; charge[nucleus] ;
mass[x] $\wedge$ mass[y] $\Rightarrow$ gravity[x,y] ;	distant$\langle$electron,nucleus$\rangle$;
gravity[x,y] $\Rightarrow$ attraction[x,y] ;	coulomb[x,y] $\Rightarrow$ attraction[x,y] ;
heavier-than[sun,planet]	positive[nucleus] ; negative[electron]

Generalization of Extended Solar System and Extended Rutherford Atom
Z ; P[X] ; P[Y] ; distant$\langle$X,Y$\rangle$; P[x] $\wedge$ P[y] $\Rightarrow$ Q[x,y] ; Q[x,y] $\Rightarrow$ attraction[x,y] ;

Fig. 5: ACU generalization of the extended analogy problem

3 The ACU Generalization System ACUOS

The ACUOS backend consists of about 1000 lines of Maude code that essentially implement the algorithm of [1], making heavy use of the Maude metaprogramming capabilities based on reflection. The algorithm is formalized as an inference system in the style of [14], with specific rules for solving and decomposing constraints (i.e., generalization subproblems) involving symbols that obey equational axioms, such as ACU and their combinations. The number of independent, order-sorted least general generalizers modulo E-renaming, where E consists of any combination of associativity, commutativity, and unity axioms of two expressions, is always finite [1], and our algorithm terminates for every

generalization problem, while computing a complete and minimal generalization set (that is, a set covering all independent generalizations).

The implementation of [1] in ACUOS has been optimized as follows. First, we identify many generalization subproblems that are equal modulo (equational) variable renaming, which enables the use of Maude memoization thus leading to exponential speed-ups for common generalization problems. Second, we delay adding any *sort* information for new variables until needed, which avoids repeated computation of subsorts for the same terms. Finally, those computations that are deterministic are encoded as Maude equations (instead of rules), thereby greatly reducing the search space as well as the memory usage due to the different treatment of rules and equations in Maude [6]. Thanks to these improvements, we can handle terms that are up to 50% larger than the preliminary, naïve implementation reported in [1].

ACUOS is publicly available at `http://safe-tools.dsic.upv.es/acuos` and comes with an intuitive web interface which allows the tool to be used through a Java Web application. Alternatively, ACUOS can also be used without the Web interface, by directly invoking the Maude generalization routine `lggs` that is implemented in the ACUOS backend. This is the preferred approach to integrate ACUOS with third-party software. For convenience, the system is also endowed with a *Full Maude* [6] user-level command allowing the user to harness the full power of the tool while being liberated from ancillary meta-level technicalities.

3.1 Experiments

In this section, we report on some experiments we have conducted with the ACUOS system. When computing modulo equational axioms, the size of the equivalence classes of the least general generalizers gives a measure of the complexity of the problem (see [15] for some theoretical results on the complexity of generalization). We use three symbols for denoting the different sizes: 0 when there is no generalizer for two terms (unlike the case of syntactical generalization, in the order-sorted setting the sorts of different *kinds*[5] are incompatible and then the terms of these sorts have no generalization, not even a variable); ω when there is a finite number of elements in the equivalence classes of the generalizers; and ∞ when the equivalence classes (w.r.t. $\approx_{ACU}$) can have an infinite number of ACU-equivalent generalizers. Any combinations of the A and C axioms are in the ω class. The introduction of the U axiom leads to size ∞ (even if the number of ACU least general generalizers is still finite).

We have tested our tool with several representative generalization problems taken from the literature that can be found online and in the distribution package. The benchmarks used for the analysis are: (i) `incompatible types`, a problem without any generalizers; (ii) `twins`, `ancestors`, `spouses`, `siblings`, and

[5] Each connected component in the poset of sorts has a top sort that is called the *kind*.

`children`, as described in the introduction; (iii) `only-U`, a generalization problem modulo (just) unity axioms, i.e., without A and C; (iv) `synthetic`, an involved example mixing A, C, and U axioms for different symbols; (v) `multiple inheritance`, which uses a classic example of multiple subtyping from [6] to illustrate the interaction of advanced type hierarchies with order-sorted generalization; (vi) `rutherford`, the example of Section 2; (vii) and `chemical`, a variant of the case-based reasoning problem for chemical compounds discussed in [5].

Test	G	#	N	ms.
incompatible types	0	2	0	16
twins (C)	ω	6	1	16
ancestors (A)	ω	22	5	40
spouses (A)	ω	16	3	16
spouses (AU)	∞	16	6	360
siblings (AC)	ω	14	2	80
children (ACU)	∞	12	1	288
only-U (U)	∞	10	1	16
synthetic	ω	20	2	20
multiple inheritance	ω	10	4	28
rutherford	ω	54	1	462
chemical	ω	20	2	240

Table 1: Experimental results

Table 1 shows our experimental results. For each problem, we show its generalization class (G), the size (number of symbols) of the input terms (#), the number of least general generalizers for each problem (N), and the total computation time (ms). As mentioned in Section 3, we achieve a dramatic improvement w.r.t. the preliminary tool reported in [1], where only the incompatible types and the twins benchmarks can be run with comparable performance; the rest of the examples time out for AC or ACU terms with more than six symbols, with the computation times surpassing one minute. Table 1 reflects that the runtimes of our algorithm do not just depend on the equational attributes given to each symbol and the size of the input terms but also on the actual shape of the terms (in particular, whether there are repeated subterms or not). This demonstrates the effectivity of the memoization mechanism that we introduced as an improvement in Section 3. Actually, we achieve up to 90% of reduction in the size of the search space w.r.t. the coarse search space generated without the improvements discussed in Section 3.

Considering the high combinatorial complexity of the ACU generalization problem, our implementation is reasonably time efficient. For example, most of the examples discussed in Section 1 took on the order of 10 ms on standard hardware (3.30 GHz Intel Xeon E3-1240 with 8Gb of RAM memory).

References

1. Alpuente, M., Escobar, S., Espert, J., Meseguer, J.: A Modular Order-sorted Equational Generalization Algorithm. Information and Computation 235, 98–136 (2014)
2. Alpuente, M., Escobar, S., Meseguer, J., Ojeda, P.: A Modular Equational Generalization Algorithm. In: Proc. LOPSTR 2008, Revised Selected Papers. LNCS, vol. 5438, pp. 24–39. Springer (2009)
3. Alpuente, M., Escobar, S., Meseguer, J., Ojeda, P.: Order–Sorted Generalization. ENTCS 246, 27–38 (2009)
4. Alpuente, M., Espert, J., Escobar, S., Meseguer, J.: ACUOS: A System for Modular ACU Generalization with Subtyping and Inheritance (extended version). Tech. rep., DSIC-UPV (2014), `http://riunet.upv.es/handle/10251/38854`
5. Armengol, E.: Usages of Generalization in Case-Based Reasoning. In: Proc. of ICCBR 2007. LNCS, vol. 4626, pp. 31–45. Springer-Verlag, Berlin, Heidelberg (2007)
6. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L. (eds.): All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic, Lecture Notes in Computer Science, vol. 4350. Springer (2007)
7. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: Reflection, metalevel computation, and strategies. In: All About Maude [6], pp. 419–458
8. Gentner, D.: Structure-Mapping: A Theoretical Framework for Analogy*. Cognitive Science 7(2), 155–170 (1983)
9. Krumnack, U., Schwering, A., Gust, H., Kühnberger, K.: Restricted higher-order anti-unification for analogy making. In: Proc. of AI 2007. LNAI, vol. 4830, pp. 273–282. Springer (2007)
10. Kutsia, T., Levy, J., Villaret, M.: Anti-Unification for Unranked Terms and Hedges. Journal of Automated Reasoning 520, 155–190 (2014)
11. Meseguer, J.: Conditioned rewriting logic as a united model of concurrency. Theor. Comput. Sci. 96(1), 73–155 (1992)
12. Muggleton, S.: Inductive Logic Programming: Issues, Results and the Challenge of Learning Language in Logic. Artif. Intell. 114(1-2), 283–296 (1999)
13. Ontañón, S., Plaza, E.: Similarity measures over refinement graphs. Machine Learning 87(1), 57–92 (Apr 2012)
14. Plotkin, G.: A note on inductive generalization. In: Machine Intelligence, vol. 5, pp. 153–163. Edinburgh University Press (1970)
15. Pottier, L.: Generalisation de termes en theorie equationelle: Cas associatif-commutatif. Tech. Rep. INRIA 1056, Norwegian Computing Center (1989)
16. Schmid, U., Hofmann, M., Bader, F., Häberle, T., Schneider, T.: Incident Mining using Structural Prototypes. In: Proc. of IEA-AIE 2010. LNCS, vol. 6097, pp. 327–336. Springer-Verlag, Berlin, Heidelberg (2010)