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 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 [21]. 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, machine learning, theorem proving, and inductive logic programming, among others [5,19,20,23].

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 us introduce the constants `john`, `sam`, `peter`, `tom`, `mary`, `chris`, and `joan`, and

* M. Alpuente, S. Escobar, and J. Espert have been partially supported by the EU (FEDER) and the Spanish MEC/MICINN under grant TIN 2010-21062-C02-02, and by Generalitat Valenciana PROMETEO2011/052. J. Espert has also been supported by the Spanish FPU grant FPU12/06223.

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 that is built by 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., the property `(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 associativity) 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 a substitution 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 [10]. 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/∅}. Actually, for every least general generalizer t, the congruence class 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/∅} 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 ∅, 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)}.

1.1 Our contribution

This work presents ACUOS, a mature and highly developed implementation of the order-sorted ACU least general generalization algorithm that we formalized in [2]. ACUOS has been written in the high-performance programming language Maude [18] that supports reasoning modulo algebraic properties. The system is implemented by taking advantage of the generic programming and reflective capabilities of Maude to express and apply inference rules. 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. To the best of our knowledge, this is the first generalization system

[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 $t \leq_{ACU} s$.

that is able to compute least general generalizers in order-sorted theories modulo equational axioms. We describe the system and discuss how it can be used to address artificial intelligence problems that need a form of ACU generalization. Experimental results in the ACUOS system show that the supported modular ACU generalization performs efficiently in practice.

1.2 Related work

A number of research directions for extending standard generalization have been undertaken within different formal frameworks and application domains [1,2,3,4,17]. Least general generalization in a first-order, order-sorted typed setting was first investigated in [1], where a generalization algorithm was proposed for feature terms, which are sorted, possibly nested, attribute-based structures which extend algebraic terms by relaxing the fixed arity and fixed indexing constraints. This is done by adding *features* (or attributing labels) to a sort as argument indicators. Feature terms were originally proposed as flexible record structures for logic programming and then used to describe different data models, including attributed typed objects in rule-based languages which are oriented towards applications to knowledge representation and natural language processing. Baader [6] discusses generalization for commutative theories. Recently, generalization has been investigated in [17,7] for terms that may contain variadic function symbols (unranked terms) as well as hedge variables, with edges being finite sequences of such terms. The standard least general generalization algorithm is also extended in [2,3,4] to the following: (i) an order-sorted[4] typed setting with sorts and subsorts in [4]; (ii) an equational setting, where function symbols can (independently) obey any combination of associativity, commutativity, and unity axioms (including the empty set of such axioms) in [3]; and (iii) to the combination of both in [2], which results in a modular, order-sorted equational generalization algorithm, thus providing a general yet practical solution to overcome the classical limitations. This opens up new applications for typed equational reasoning systems and typed rule-based languages such as ASF+SDF [8], Elan [9], OBJ [15], CafeOBJ [12], and Maude [18,10], where some function symbols may be declared to obey given algebraic laws (the so-called *equational attributes*) of associativity and/or commutativity and/or unit element. We point interested readers to [2,17] for further discussion of the related literature.

1.3 Plan of the paper

In Section 2, we illustrate the usefulness of the ACUOS system by focusing on a simple and classical artificial intelligence problem that is known as the Rutherford analogy [14,16], and demonstrate how our system fulfills the objective to recognize that atoms resemble tiny solar systems. In Section 3, we give the system

[4] Order-sorted theories not only support types but also support a powerful form of subtype polymorphism that can be used to model *is_a* relationships and multiple inheritance [10].

overview and present a brief generalization session together with some experimental results that qualify ACUOS as a very effective generalization system. We conclude in Section 4.

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* [14], 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 [11]. Note that both input models in the Rutherford example are complete and no property needs to be transferred from one domain to the other; this is generally done in analogy making by first extrapolating unmapped elements and then inserting them into the target domain if proved to be valid [13].

2.1 Problem representation

Let us introduce a meta-representation for models by introducing the HModel sort (i.e., type, in the Maude terminology) 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[5] (`;`), (`.`), 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 [10].

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

[5] Notice the *mixfix* notation [10] in the definition of the operators (e.g., `op _;_ : HModel HModel -> HModel`), which uses underscores `_` to indicate that, in well-formed terms, 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: 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

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.

Solar System	**Rutherford Atom Model**
mass[sun] ;	charge[y] ∧ charge[x] ⇒ coulomb[x,y] ;
mass[planet] ;	charge[electron] ;
distant⟨sun,planet⟩ ;	charge[nucleus] ;
mass[x] ∧ mass[y] ⇒ gravity[x,y] ;	distant⟨electron,nucleus⟩ ;
gravity[x,y] ⇒ attraction[x,y]	coulomb[x,y] ⇒ 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.

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

Generalization of Solar System and Rutherford Atom
P[X] ;
P[Y] ;
distant⟨X,Y⟩ ;
P[x] ∧ P[y] ⇒ Q[x,y] ;
Q[x,y] ⇒ 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

`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 impossible to establish without considering the commutativity and associativity of the operators (_∧_) 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 whereas Figure 6 depicts 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
mass[sun] ; mass[planet] ;
distant⟨sun,planet⟩ ;
mass[x] ∧ mass[y] ⇒ gravity[x,y] ;
gravity[x,y] ⇒ attraction[x,y] ;
heavier-than[sun,planet]

Extended Rutherford Atom Model
charge[y] ∧ charge[x] ⇒ coulomb[x,y] ;
charge[electron] ; charge[nucleus] ;
distant⟨electron,nucleus⟩ ;
coulomb[x,y] ⇒ attraction[x,y] ;
positive[nucleus] ; negative[electron]

Fig. 5: Extended analogy problem representation

3 The ACU Generalization System ACUOS

The ACUOS backend consists of about 1000 lines of Maude code that essentially implement the algorithm of [2], making heavy use of the Maude meta-

Generalization of Extended Solar System and Extended Rutherford Atom
Z ; P[X] ; P[Y] ; distant⟨X,Y⟩ ; P[x] ∧ P[y] ⇒ Q[x,y] ; Q[x,y] ⇒ attraction[x,y] ;

where variable Z belongs to sort HModel and the remaining variables have the same sorts as in Figure 4.

Fig. 6: ACU generalization of the extended problem

programming capabilities based on reflection. The algorithm is formalized as an inference system in the style of [21], 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 [2], and our algorithm terminates for every generalization problem, while computing a complete and minimal generalization set (that is, a set covering all generalizations, and containing no redundant members).

The implementation of [2] 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 [10]. Thanks to these improvements, we can handle terms that are up to 50% larger than the preliminary, naïve implementation reported in [2].

3.1 ACUOS Architecture

The architecture of ACUOS, which is depicted in Figure 7, comprises the following modules:

(i) Search for candidates. The Maude `metaSearch` function is used to exhaustively explore the search space of the inference system.
(ii) Introduction of variables. Unsolved generalization subproblems are substituted by fresh variables. If there is more than one possible sort for a given variable, each of them gives rise to a different candidate.
(iii) Normalization. A deterministic variable renaming algorithm that reduces term equality modulo renaming to syntactic equality is applied. As a result, a complete and finite (but not yet minimal) set of valid generalizers is delivered.
(iv) Minimization. The candidate set is minimized according to the instantiation ordering, producing the set of least general, order-sorted generalizers of the input terms modulo any combination of A, C, and U axioms.

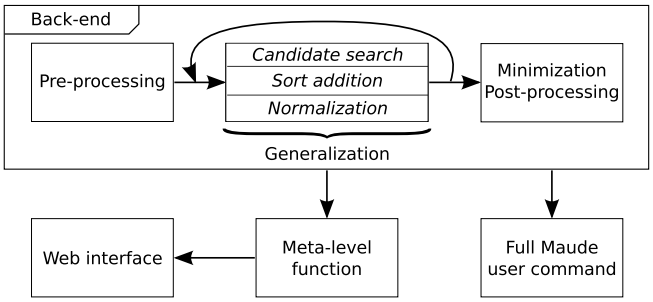

Fig. 7: The ACUOS Architecture

3.2 A Generalization Session with ACUOS

This section describes a generalization session with ACUOS using the `spouses` example of Section 1.

Using the ACUOS Web interface, a generalization session with ACUOS is easily done. The input to the system is entered by filling three fields: the two terms that we wish to generalize and a file containing the Maude modules that describe the signature (including the equational attributes) of all domain operators. The specification can be either directly pasted into the input form, loaded from a file in the user's machine, or obtained from the suite of examples provided by the ACUOS system through its Web interface. In this example, we select the first option "**Spouses(A)**" in the "Select model" menu shown in Figure 8 below, which corresponds to the `spouses` example discussed in Section 1. Then, we press "Load" to enter the Maude module into the ACUOS tool. Finally, we click on the "Generalize!" button and ACUOS delivers the least general generalizers of the two terms, which are shown in Figure 9.

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. Figure 10 shows the (internal) Maude generalization call that corresponds to the Rutherford example. For convenience, the system is endowed with a *Full Maude* [10] user-level command, allowing the user to harness the full power of the tool while being liberated from ancillary meta-level technicalities.

3.3 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 [22] for some theoretical results on the complexity of generalization). We use three symbols for denoting the different sizes: 0 when there is

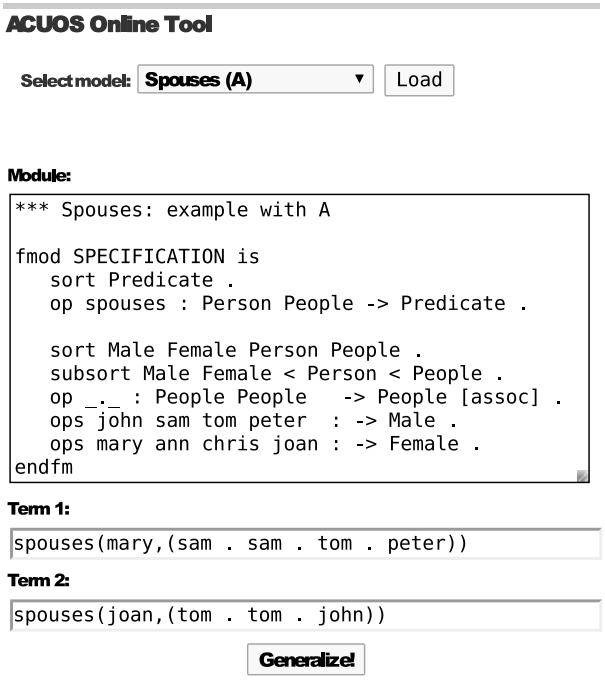

Fig. 8: The ACUOS Web interface

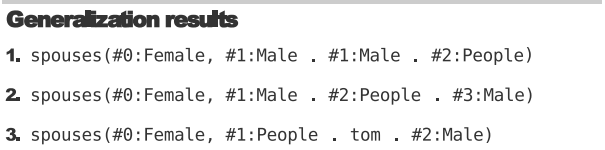

Fig. 9: Snapshot of ACUOS output

no generalizer for two terms (unlike the case of syntactical generalization, in the order-sorted setting the sorts of different *kinds*[6] 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 prob-

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

```
rew lggs(
  upModule('SPECIFICATION, true),
  upTerm(spouses(mary,
    (sam . sam . tom . peter))),
  upTerm(spouses(joan,
    (tom . tom . john))) .
```

```
( lggs in DOMAIN-OPERATORS :
    spouses(mary,
      (sam . sam . tom . peter))
    =?
    spouses(joan,
      (tom . tom . john)) . )
```

Fig. 10: ACUOS Maude (resp. Full Maude) internal calls

lem without any generalizers; (ii) `twins`, `ancestors`, `spouses`, `siblings`, and `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 [10] 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].

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 [2], where only the incompatible types and the twins benchmarks can be run with comparable performace; 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). The elapsed times for the Rutherford example are largely due to the encoding of higher-order generalization as first-order generalization, but remain reasonable, under 500 ms.

4 Conclusions and future work

Generalization is a formal reasoning component of many symbolic frameworks. Order-sorted modular ACU generalization extends ordinary generalization with subtypes and the capability to endow each function symbol with any combination of associativity, commutativity, and unity axioms, making it possible

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

to reason about records, trees, lists, and (multi-)sets of data elements, atoms or rules, as well as classical hierarchical/structural relations such as *is_a* and *part_of* which are naturally supported by the Maude order-sortedness. We have presented ACUOS, which is an order-sorted modular ACU generalization system that enables the use of this technique with good performance, considering the complexity of ACU generalization. ACUOS is endowed with three interfaces that are very easy to use: a Web interface, a Maude routine, and a Full Maude user-level command. It is open source software and integrates easily with third-party tools.

Other directions for extending the generalization problem are to promote the order [16] and/or introduce variadic function symbols in the underlying language (as is done in feature terms and unranked hedges [5,17]). A challenge with these extensions is that, without important restrictions that may limit their application, generalization can be quite expensive [17] and/or not well-defined: chains of increasingly less general generalizations can be constructed [16].

References

1. Aït-Kaci, H.: Outline of a Calculus of Type Subsumption. Tech. rep., U. of Pennsylvania (1983)
2. Alpuente, M., Escobar, S., Espert, J., Meseguer, J.: A Modular Order-sorted Equational Generalization Algorithm. Information and Computation (2014), in press, http://dx.doi.org/10.1016/j.ic.2014.01.006
3. 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)
4. Alpuente, M., Escobar, S., Meseguer, J., Ojeda, P.: Order–Sorted Generalization. ENTCS 246, 27–38 (2009)
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. Baader, F.: Unification, Weak Unification, Upper Bound, Lower Bound, and Generalization Problems. In: Book, R.V. (ed.) Proc. of 4th International Conference on Rewriting Techniques and Applications, RTA-91. LNCS, vol. 488, pp. 86–97. Springer (1991)
7. Baumgartner, A., Kutsia, T., Levy, J., Villaret, M.: A variant of higher-order anti-unification. In: van Raamsdonk, F. (ed.) Proc. of 24th International Conference on Rewriting Techniques and Applications, RTA 2013, June 24-26, 2013, Eindhoven, The Netherlands. LIPIcs, vol. 21, pp. 113–127. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2013)
8. Bergstra, J., Heering, J., Klint, P.: Algebraic Specification. ACM Press (1989)
9. Borovanský, P., Kirchner, C., Kirchner, H., Moreau, P.E.: ELAN from a rewriting logic point of view. Theoretical Computer Science 285, 155–185 (2002)
10. 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)
11. 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 [10], pp. 419–458
12. Diaconescu, R., Futatsugi, K.: CafeOBJ Report, AMAST Series in Computing, vol. 6. World Scientific, AMAST Series (1998)
13. French, R.M.: The computational modeling of analogy-making. In: Trends in Cognitive Sciences. pp. 200–205 (2002)
14. Gentner, D.: Structure-Mapping: A Theoretical Framework for Analogy*. Cognitive Science 7(2), 155–170 (1983)
15. Goguen, J., Winkler, T., Meseguer, J., Futatsugi, K., Jouannaud, J.P.: Introducing OBJ. In: Software Engineering with OBJ: Algebraic Specification in Action, pp. 3–167. Kluwer (2000)
16. 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)
17. Kutsia, T., Levy, J., Villaret, M.: Anti-unification for unranked terms and hedges. In: Schmidt-Schauß, M. (ed.) Proc. of 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 - June 1, 2011, Novi Sad, Serbia. LIPIcs, vol. 10, pp. 219–234. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2011)
18. Meseguer, J.: Conditioned rewriting logic as a united model of concurrency. Theor. Comput. Sci. 96(1), 73–155 (1992)
19. Muggleton, S.: Inductive Logic Programming: Issues, Results and the Challenge of Learning Language in Logic. Artif. Intell. 114(1-2), 283–296 (1999)
20. Ontañón, S., Plaza, E.: Similarity measures over refinement graphs. Machine Learning 87(1), 57–92 (Apr 2012)
21. Plotkin, G.: A note on inductive generalization. In: Machine Intelligence, vol. 5, pp. 153–163. Edinburgh University Press (1970)
22. Pottier, L.: Generalisation de termes en theorie equationelle: Cas associatif-commutatif. Tech. Rep. INRIA 1056, Norwegian Computing Center (1989)
23. 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)