

Classifying Isomorphic Residue Classes

Andreas Meier, Martin Pollet*, and Volker Sorge**

Fachbereich Informatik, Universität des Saarlandes, Germany,
{ameier|pollet|sorge}@ags.uni-sb.de
<http://www.ags.uni-sb.de/~{ameier|pollet|sorge}>

Abstract. We report on a case study on combining proof planning with computer algebra systems. We construct proofs for basic algebraic properties of residue classes as well as for isomorphisms between residue classes using different proving techniques, which are implemented as strategies in a multi-strategy proof planner. We show how these techniques help to successfully derive proofs in our domain and explain how the search space of the proof planner can be drastically reduced by employing computations of two computer algebra systems during the planning process. Moreover, we discuss the results of experiments we conducted which give evidence that with the help of the computer algebra systems the planner is able to solve problems for which it would fail to create a proof otherwise.

1 Introduction

We report on a case study that combines proof planning with computer algebra systems. We classify residue class sets over the integers together with given binary operations in terms of their basic algebraic properties and additionally into sets of isomorphic structures. A residue class set over the integers, RS_n , is either the set of all congruence classes modulo an integer n , i.e., \mathbb{Z}_n , or an arbitrary subset of \mathbb{Z}_n . Concretely, we are dealing with sets of the form $\mathbb{Z}_3, \mathbb{Z}_5, \mathbb{Z}_3 \setminus \{\bar{1}_3\}, \mathbb{Z}_5 \setminus \{\bar{0}_5\}, \{\bar{1}_6, \bar{3}_6, \bar{5}_6\}, \dots$ where $\bar{1}_3$ denotes the congruence class 1 modulo 3. If c is an integer we write also $cl_n(c)$ for the congruence class c modulo n . A binary operation \circ on a residue class set is usually written in λ -function notation. \circ can be of the form $\lambda xy \bullet x, \lambda xy \bullet y, \lambda xy \bullet c$ where c is a constant congruence class (e.g., $\bar{1}_3$), $\lambda xy \bullet x \bar{+} y, \lambda xy \bullet x \bar{*} y, \lambda xy \bullet x \bar{-} y$, where $\bar{+}, \bar{*}, \bar{-}$ denote addition, multiplication, and subtraction on congruence classes over the integers, respectively. Furthermore, \circ can be any combination of the basic operations with respect to a common modulo factor, (e.g., $\lambda xy \bullet (x \bar{+} \bar{1}_3) \bar{-} (y \bar{+} \bar{2}_3)$).

The case study was carried out in the Ω MEGA theorem proving environment [2]. It consisted of two parts: (1) To examine the basic algebraic properties of given residue class structures and classify them into terms of the algebraic structure they form (e.g., group, monoid, or quasi-group). (2) Structures of the same type and cardinality are then further investigated to identify isomorphism

* The author's work was supported by the 'Graduierförderungs des Saarlandes'.

** The author's work was supported by the 'Studienstiftung des Deutschen Volkes'.

classes. The first part of the case study was reported in [13]. In this paper we concentrate on how we determine and prove whether residue class structures are isomorphic to each other or not. For an extensive report on the whole case study, including a detailed presentation of the constructed proofs, we refer the reader also to [12].

The proof constructions are used within a tutor system for an interactive mathematical course in algebra. For tutoring purposes it was necessary to have proofs in human-oriented reasoning style using different proving techniques. Therefore, we chose multi-strategy proof planning as our main tool for constructing proofs. On the one hand, it allows us to easily model different proving techniques by different strategies. On the other hand, it enables us to exploit the power of computer algebra systems in a sound way [9, 18] to speed up the proving process. The aim of the paper is to present how multi-strategy proof planning is used to determine and verify isomorphism of residue class structures and how computer algebra is employed to guide and shorten the search during the proof planning process. We do, however, not explain how the examples are actually used in a tutor system; we instead refer the reader to [3], where a system description of the ACTIVE MATH learning environment is given.

The paper is organized as follows: we first give a brief overview of multi-strategy proof planning in the Ω MEGA system and the integration of computer algebra with proof planning. Section 3 contains a summary of the classification of residue class structures with respect to their simple algebraic properties, which is described in detail in [13]. Section 4 is the major part giving the details of how isomorphic residue class structures are identified, and how the necessary isomorphism and non-isomorphism proofs are constructed. We then give, in Sec. 5, a brief account of some of the experiments we carried out and conclude with a discussion of some of the related work.

2 Proof Planning and Computer Algebra

2.1 Multi-Strategy Proof Planning

Proof planning [1] considers mathematical theorems as planning problems where an *initial partial plan* is composed of the proof *assumptions* and the theorem as *open goal*. A proof is then constructed with the help of abstract planning steps, called *methods*, that are essentially partial specifications of tactics known from tactical theorem proving. In order to ensure correctness, proof plans have to be executed to generate a sound calculus level proof.

In the Ω MEGA system [2] the traditional proof planning approach is enriched by incorporating mathematical knowledge into the planning process (see [15] for details). That is, methods can encode general proving steps as well as knowledge particular to a mathematical domain. Moreover, *control rules* specify how to traverse the search space by influencing the ordering of method application and the choice of the next goal depending on certain domains or proof situations. Ω MEGA's new proof planner, MULTI [14], allows also for the specification of different planning strategies to control the overall planning behavior.

Methods in Ω MEGA are essentially tactics known from tactical theorem proving augmented with pre- and postconditions, so-called *premises* and *conclusions*. Premises and conclusions indicate the inference of the method (the conclusions should follow from the premises) and indicate the role of the method in the planning process. For instance, *Indirect* is a method whose purpose is to prove a goal P by contradiction. If *Indirect* is applied to a goal P then it closes this goal and introduces the new goal \perp . Moreover, it adds the new hypothesis $\neg P$ to the proof plan such that the contradiction \perp can be constructed using also $\neg P$. Thereby, P is the conclusion of the method, whereas \perp is the premise of the method.

Control rules provide the possibility to introduce mathematical knowledge on how to proceed in the proof planning process. They can influence the planners behavior at choice points (e.g., which goal to tackle next or which method to apply next) by preferring, rejecting, or enforcing members of the corresponding alternative lists (e.g., the list of possible goals or the list of possible methods). This promotes certain promising search paths and can thus prune the search space. In particular, we employ control rules to prefer a particular instance from a list of possible variable instantiations. As example we present the `select-instance` control rule in the next section.

In Ω MEGA different proof techniques for a problem class can be realized by different *planner strategies* [14]. The planner strategies can employ different sets of methods and control rules and can thus allow to tackle the same problem in different ways. The reasoning about which strategy to employ on a problem (provided there are several applicable strategies) and about the switching of strategies is an additional choice point in MULTI. Therefore, the planner can also backtrack from applied strategies and perform search on the level of strategies.

2.2 Using Computer Algebra in Proof Planning

When proof planning in the domain of residue classes we employ symbolic calculations to guide and simplify the search for proof plans. In particular, we use the mainstream computer algebra system MAPLE [16] and GAP [5], a system specialized on group theory. We are not concerned with the technical side of the integration since we exploit previous work, in particular [9] that presents the integration of computer algebra into proof planning, and [18] that exemplifies how the correctness of certain limited computations of a large-scale computer algebra system such as MAPLE can be guaranteed within the proof planning framework. In this paper we rather concentrate on the cooperation between the systems in the context of exploring residue class properties.

We use symbolic calculations in two ways: (1) in control rules hints are computed to help guiding the planning process, and (2) within the application of a method, equations are solved with MAPLE to simplify the proof. As side-effect both cases can restrict possible instantiations of meta-variables¹.

¹ Meta-variables are place-holders for terms whose actual form is computed at a later stage in the proof search.

(1) is implemented, for instance, in the control rule `select-instance` (used in the strategy `TryAndError`, see next section). The rule is triggered after decomposition of an existentially quantified goal which results in the introduction of a meta-variable as substitute for the actual witness term. After an existential quantifier is eliminated the control rule computes a hint with respect to the remaining goal that is used as a restriction for the introduced meta-variable. For instance, when proving that the residue class set RS_n is not closed under the operation \circ , that is, there exist $a, b \in RS_n$ such that $a \circ b \notin RS_n$, the control rule would supply a hint as to what a and b might be. If hints can be computed the meta-variables are instantiated before the proof planning proceeds.

To obtain suitable hints `select-instance` sends corresponding queries to GAP and MAPLE (a detailed description of the hint system for simple algebraic properties is given in [13], the hint system when classifying isomorphic structures is described in Sec. 4). However, all such computations by MAPLE and GAP are treated as a hint by the proof planner; that is, in case the proving attempt fails for a particular instantiation computed by the computer algebra systems the planner falls back to its regular search.

In (2), the use of calculations is realized within the `SolveEquation` method. Its purpose is to justify an equational goal using MAPLE and, if necessary, to instantiate meta-variables. In detail, it works as follows: if an open goal is an equation, MAPLE's function `solve` is applied to check whether the equality actually holds. Should the equation contain meta-variables then these are considered as the variables the equation is to be solved for, and they are supplied to `solve` as additional arguments. In case the equation involves modulo functions with the same factor on both sides, MAPLE's function `msolve` is used instead of `solve`. If MAPLE can successfully solve the equation, the method is applied and possible meta-variables are instantiated accordingly. The computation is then considered correct for the rest of the proof planning process. However, once the proof plan is executed MAPLE's computation is expanded into low level logic derivations to check its correctness. This is done with the help of a small self-tailored computer algebra system that provides detailed information on its computations in order to construct the expansion. This process is extensively described in [18].

3 Checking Simple Properties

First, we are interested in classifying residue class sets over the integers together with given binary operations in terms of what algebraic structure they form. We automatically classify structures of the form (RS_n, \circ) in terms of magma (also called groupoid), semi-group, monoid, quasi-group, loop, group, and whether they are Abelian. The classification is done by first checking successively if the properties: closure, associativity, existence of the unit element, existence of inverse elements, and the quasi-group axiom (i.e., that for each two elements $a, b \in RS_n$ there exist elements $x, y \in RS_n$ such that $a \circ x = b$ and $y \circ a = b$) hold and then constructing and discharging an appropriate proof obligation. The properties are checked mainly with GAP and proofs for the constructed obliga-

tions are planned with MULTI. For instance, GAP is used to check whether a given structure contains a unit element; depending on the result, a proof obligation is constructed stating there exists or there does not exist a unit element in the structure. MULTI then tries to produce a proof plan for this statement. If it succeeds the next property is checked; if it fails MULTI tries to prove the negation. For discharging proof obligations we have implemented three different proving techniques with strategies in MULTI, which use symbolic computations to a varying degree.

The simplest strategy is **TryAndError** which performs a naïve *exhaustive case analysis*. This technique is possible since we are in a finite domain and can always enumerate all occurring cases. The planning process usually starts with the expansion of defined concepts such as *unit*, *associative*, etc. For resulting universally quantified goals ranging over a residue class a case split on all elements of the structure is performed. For existentially quantified goals all possible instantiations for the quantified variable are successively checked. The latter is done by introducing a meta-variable that is bound successively to the different elements of the residue class set. Here the search space can be reduced by providing the (probably) correct instantiation immediately as a hint with the control rule **select-instance** (see last section). For instance, when showing that for each element in the structure there indeed exists an inverse, GAP can compute the respective inverses. When the subsequent subgoals cannot be proved, MULTI backtracks to the instantiation of the meta-variable and chooses the next element. After the quantifiers are eliminated, the statements about residue classes are transformed to statements about integers which can be solved by numerical simplifications.

The second proving technique is **EquSolve**. This strategy employs as much as possible *equational reasoning*. Problems are decomposed to the level of equations; universally quantified variables are replaced by constants and existentially quantified variables by meta-variables. The property then holds, when all equations can be solved by the **Solve-Equation** method. The strategy employs MAPLE to check the universal validity of the equation or, in case the equation contains meta-variables, if there is an instantiation of these meta-variables, such that the equation is universally valid. The technique can, however, be applied only to those problems that can be reduced to equations (associativity, unit element, inverse elements, and quasi-group axiom). In particular, it cannot be applied to refute properties or to show closure. **EquSolve**, like **TryAndError**, reduces statements on residue classes to statements on integers. For instance, the equation for the inverse element $cl_n(c) \bar{+} cl_n(mv) = \bar{0}_n$ containing congruence classes (where c is a constant and mv is a meta-variable) is reduced to the corresponding equation on integers $(c + mv) \bmod n = 0 \bmod n$ before MAPLE returns a general solution for mv .

The last technique is **ReduceToSpecial** which applies *already known theorems*. Here MULTI uses theorems from Ω MEGA's knowledge-base to reduce a given problem. This strategy does not depend on the help of a computer algebra system. Moreover, the theorems are applied to statements about residue class

structures directly; a reduction to statements about integers as in `TryAndError` and `EquSolve` is not necessary.

When automatically discharging proof obligations `MULTI` attempts first the application of theorems, then equational reasoning and lastly the exhaustive case analysis. That is, we start with the strategy that is generally the most efficient one and end with the most reliable strategy.

In order to test our approach we constructed a large testbed of automatically generated examples from the possible subsets of the residue classes modulo n , where n ranges from 2 to 10, together with operations that are systematically constructed from the basic operations. We classified 14337 structures with one operation so far. We found 5810 magmas, 109 Abelian magmas, 2064 semi-groups, 1670 Abelian semi-groups, 1018 quasi-groups, 461 Abelian quasi-groups, 93 Abelian monoids, and 780 Abelian groups (the other structures we tested are not closed). Note, that these figures do not mean that we have so many distinct algebraic entities, since our testbed contains many isomorphic structures. For the proofs of the single properties that were tested during the classification, `MULTI` successfully employed `ReduceToSpecial` at the rate of 19% and `EquSolve` to a different set accounting for 21% of the examples. The remaining 60% of the examples could be solved only by the `TryAndError` strategy. For a more detailed report on the exploration of simple properties of residue structures see [13, 12].

4 Identifying Classes of Isomorphic Structures

Checking simple algebraic properties of residue class structures as described in the preceding section allows to classify given structures in terms of the algebraic entity they form. This, however, does not indicate how many of these structures are actually different (i.e., not isomorphic to each other) or are just different representations of the same structure. In this section we present how we classify given sets of residue class structures into *equivalence classes of isomorphic structures*. To ease the task we already exclude structures that are trivially not isomorphic to each other. Hence, we only examine structures which are of the same algebraic category (e.g., monoids are only compared with other monoids and not with groups) and we consider only structures of the same cardinality since for finite sets structures of different size are trivially not isomorphic.

The idea of the isomorphism classification algorithm is to partition a set of residue class structures into disjoint classes of isomorphic structures. Given such a set we initialize the first isomorphism class with the very first structure of the set. Each further structure S is checked whether it belongs to an already existing isomorphism class. If we can prove that there is a corresponding isomorphism class, S is added to this class otherwise a new class is initialized. Whether or not S belongs to a certain isomorphism class is tested first with `MAPLE` by constructing a pointwise defined isomorphism mapping S to a structure S' of an existing class (the actual computation of this pointwise function is described in more detail in Sec. 4.1). If `MAPLE`'s computation suggests that S is isomorphic to S' the corresponding proof obligation is constructed and passed to `MULTI`.

If MAPLE cannot find any existing isomorphism class for S then the classification algorithm assumes that S belongs to a new isomorphism class. Hence, for each existing isomorphism class it constructs a proof obligation stating that S is not isomorphic to a structure S' of this class and sends these proof obligations to MULTI. If MULTI succeeds to prove all of these obligations, then a new isomorphism class is initialized with S .

In the following we describe how MULTI discharges isomorphism and non-isomorphism proof obligations. In particular, we describe how MAPLE and GAP are employed to support the proof planning process.

4.1 Isomorphism Proofs

In this section we present how MULTI plans isomorphism proofs. It employs the same three strategies mentioned in Sec. 3, namely `TryAndError`, `EquSolve`, and `ReduceToSpecial`. We just had to add two methods for the introduction of isomorphism mappings to the `TryAndError` and `EquSolve` strategies and one additional theorem for the `ReduceToSpecial` strategy. Contrary to the proofs of simple properties of structures that could be solved in most cases within one strategy, for isomorphism proofs different subproofs can be solved by different strategies. This means that the strategy `EquSolve` switches to `TryAndError`, while `ReduceToSpecial` uses `EquSolve` and `TryAndError` to prove some of the occurring subgoals.

TryAndError To prove that two given structures (RS_n^1, \circ^1) and (RS_m^2, \circ^2) are isomorphic we have to show that there exists a function $h: (RS_n^1, \circ^1) \rightarrow (RS_m^2, \circ^2)$ such that h is injective, surjective, and homomorphic.² As described in Sec. 3 `TryAndError` checks for existentially quantified goals all possible instantiations for the quantified variable successively. In the context of finite sets each possible mapping h can be represented as a pointwise defined function, where the image of each element of the domain is explicitly specified as an element of the codomain.

MULTI abbreviates the search for the right instantiation of h by computing a hint. For an isomorphism $h: (RS_n^1, \circ^1) \rightarrow (RS_m^2, \circ^2)$, a system of equations is generated by instantiating the homomorphism equation $h(cl_n(i) \circ^1 cl_n(j)) = h(cl_n(i)) \circ^2 h(cl_n(j))$ with all elements of the residue class set RS_n^1 . When we take $cl_n(k)$ to be the result of $cl_n(i) \circ^1 cl_n(j)$, we obtain a system of equations of the form $h(cl_n(k)) = h(cl_n(i)) \circ^2 h(cl_n(j))$. Now, MAPLE is asked to give a solution for the corresponding system of equations $x_k = x_i \circ^2 x_j$ with respect to the modulo factor m using MAPLE's function `msolve`, where $h(cl_n(l))$ becomes the variable x_l . When MAPLE returns a solution for the variables containing only elements from the integer set corresponding to RS_m^2 we have found a homomorphism between the structures. When there is a disjoint solution with $x_i \neq x_j$, for all $i \neq j$, we have a candidate for the isomorphism.

² Observe that we avoid confusion between indices and modulo factors by writing indices as superscripts, except in indexed variables such as x_i, y_j as they are clearly distinct from congruence classes of the form $cl_i(x)$.

As a simple example we consider the proof that $(\mathbb{Z}_2, \bar{+})$ is isomorphic to $(\mathbb{Z}_2, \lambda xy \bullet x \bar{+} y \bar{+} \bar{1}_2)$. The possible pointwise functions $h : \mathbb{Z}_2 \rightarrow \mathbb{Z}_2$ are:

$$h^1(x) = \begin{cases} \bar{0}_2, & \text{if } x = \bar{0}_2 \\ \bar{0}_2, & \text{if } x = \bar{1}_2 \end{cases}, \quad h^3(x) = \begin{cases} \bar{0}_2, & \text{if } x = \bar{0}_2 \\ \bar{1}_2, & \text{if } x = \bar{1}_2 \end{cases}, \\ h^2(x) = \begin{cases} \bar{1}_2, & \text{if } x = \bar{0}_2 \\ \bar{0}_2, & \text{if } x = \bar{1}_2 \end{cases}, \quad h^4(x) = \begin{cases} \bar{1}_2, & \text{if } x = \bar{0}_2 \\ \bar{1}_2, & \text{if } x = \bar{1}_2 \end{cases}.$$

During the proof MAPLE is asked to give a solution for the equations $x_0 = x_0 + x_0 + 1$, $x_1 = x_0 + x_1 + 1$, $x_1 = x_1 + x_0 + 1$, $x_0 = x_1 + x_1 + 1$ with modulo factor 2 and returns $\{x_0 = 1, x_1 = x_1\}$. The solutions are analyzed by the hint system, and $x_0 = 1, x_1 = 0$ is suggested because it is both a disjoint solution and all elements are in the codomain. Therefore, $h^2(x)$ is inserted as the pointwise defined isomorphic function. The subsequent subproofs for the properties injectivity, surjectivity, and homomorphism of the pointwise defined function are then performed in the regular fashion of the `TryAndError` strategy as already discussed in Sec. 3: defined concepts are expanded, quantifiers are eliminated by introducing case splits for all possible values, and statements about residue classes are rewritten into statements about integers.

Proving the properties injectivity, surjectivity, and homomorphism with the `TryAndError` strategy has the complexity n^2 where n is the cardinality of the structures involved.³ However, if no suitable hint can be computed there are n^n pointwise defined functions to check, which becomes infeasible already for relatively small n .

EquSolve During the isomorphism proof we have to show injectivity, surjectivity, and the homomorphism property for the introduced mapping. Doing so by a complete case analysis can become quite lengthy and therefore it is desirable to represent the isomorphism function in a more compact form. Often this can be realized by computing a polynomial that interpolates the pointwise defined function. If we can compute such an interpolation polynomial the `EquSolve` strategy has a chance to find the subproofs for surjectivity and the homomorphism property. Namely these subproblems can then be reduced to equations which could be solvable with the `SolveEquation` method. In the subproof for injectivity we have to show for each two distinct elements that their images differ, which cannot be concluded by equational reasoning.

For the construction of the interpolation polynomial we employ again MAPLE. However, we do not use one of the standard algorithms from the literature for interpolating sparse polynomials (see for example [20, 19]), as they do not necessarily give us the best possible interpolation polynomial. Moreover, some of the implemented algorithms, for instance in MAPLE, do not always suffice for our purposes.⁴ We have thus decided to implement a simple search algorithm to

³ The proof of each of these properties results in formulas with two nested quantifications ranging over sets of cardinality n . This results into n^2 possible cases.

⁴ MAPLE's algorithms `interp` and `Interp` cannot always handle the interpolation of functions where a non-prime modulo factor is involved.

find a suitable interpolation polynomial of minimal degree. This is feasible as we have to handle only relatively small mappings.

In detail, the interpolation proceeds as follows: Given a pointwise defined isomorphism function $h: cl_n(x_i) \in RS_n^1 \rightarrow cl_m(y_i) \in RS_m^2$ we let MAPLE solve the system of equations $(a_d x_i^d + \dots + a_1 x_i + a_0) \bmod m = y_i \bmod m$ for all x_i, y_i . When MAPLE returns a solution for a_d, \dots, a_0 we have found an interpolating polynomial. If there is no solution, a polynomial with degree $d + 1$ will be sent to MAPLE. This procedure terminates at the latest when $d = m - 1$.

We illustrate the approach of the `EquSolve` strategy using once more the example of the proof that $(\mathbb{Z}_2, \bar{+})$ is isomorphic to $(\mathbb{Z}_2, \lambda x y, x \bar{+} y \bar{+} \bar{1}_2)$. The corresponding pointwise isomorphism mapping is $h(\bar{0}_2) = \bar{1}_2, h(\bar{1}_2) = \bar{0}_2$ for which the interpolation polynomial $x \rightarrow (x + 1 \bmod 2)$ can be computed. This polynomial is introduced into the proof instead of the pointwise defined function. The properties of injectivity, homomorphism, and surjectivity, are then shown for the polynomial. During the subproofs of the latter two properties, the problem is reduced to an equation over integers that can be generally solved by the `SolveEquation` method. As already mentioned the proof for injectivity cannot be constructed within the `EquSolve` strategy. Therefore, `MULTI` switches either to the strategy `ReduceToSpecial` or `TryAndError` to prove this property. How the former is applied in this context is described in the next paragraph.

The success of `EquSolve` depends on the capability of MAPLE. Often equations in isomorphism proofs contain terms with different modulo factors nested inside, resulting from the mapping between residue class sets RS_n and RS_m with $n \neq m$, which are not solvable by MAPLE. So `EquSolve` is limited to proofs for residue class sets with the same modulo factor.

ReduceToSpecial The strategic control rules in `MULTI` specify that on residue class problems the strategies `ReduceToSpecial`, `EquSolve`, and `TryAndError` are always tested in this order. This holds for isomorphism or non-isomorphism problems as well as for possible arising subproblems such as to show injectivity, surjectivity, or homomorphism. For instance, if `EquSolve` can introduce a suitable polynomial function but fails to prove the arising injectivity, surjectivity, or homomorphism subgoals, `MULTI` has to deal with those subproblems again on the strategy level. Since we do not have theorems to handle isomorphism problems in general, `ReduceToSpecial` is not applicable to the original theorem, but it comes into play when a subgoal, in particular the injectivity subgoal, has to be proved. Here we can exploit the simple mathematical fact that in finite domains surjectivity implies injectivity and vice versa with the following theorem: *A surjective mapping between two finite sets with the same cardinality is injective.*

Thus, the proof of injectivity consists only of the application of this theorem, if we can prove that our mapping is surjective. Hence, the idea for the most efficient isomorphism proofs is to start with `EquSolve` on the whole isomorphism problem, prove the surjectivity and homomorphism subproblem, if possible, with equational reasoning, and, since `EquSolve` will always fail on the injectivity subgoal, to let `ReduceToSpecial` finish the proof.

4.2 Non-Isomorphism Proofs

During the classification process it is also necessary to prove that two given structures are not isomorphic. To discharge this proof obligation we use again the already introduced strategies `ReduceToSpecial` and `TryAndError`. While the latter can be applied independently by performing the usual exhaustive case analysis, the former is combined with `TryAndError` to deduce non-isomorphism of two structures by showing the existence of substructures of different order. Additionally, we have implemented another strategy, `NotInjNotIso`, which is specialized on proving non-isomorphism problems. It constructs an indirect proof by showing that no homomorphic mapping between the two given residue class structures can be injective.

TryAndError Proving that two structures are not isomorphic results in proving a universally quantified goal: `MULTI` has to prove that each possible mapping between the two structures involved is either non-injective, non-surjective, or non-homomorphic. When dealing with universally quantified statements the `TryAndError` strategy performs an exhaustive case split on all possible instantiations. As described in Sec. 4.1 all possible mappings can be presented by pointwise defined functions. Hence, in the case of non-isomorphism proofs the top-most case split in the `TryAndError` strategy is on every possible pointwise defined function and for each function a subproof is constructed, to show that it is either non-injective, non-surjective, or non-homomorphic. The subproofs are constructed with an exhaustive case analysis similar to the proofs of the simple properties in Sec. 3.

The application of this naïve technique suffers from combinatorial explosion on the possibilities for the pointwise defined function. For two structures whose sets have cardinality n we have to consider n^n different possible functions. Thus, in practice this strategy is not feasible if structures of cardinality larger than four are involved. Despite this fact the strategy is our fall back if the other techniques presented in the sequel should fail.

ReduceToSpecial If two structures are isomorphic, they share the same algebraic properties. Thus, in order to show that two structures are not isomorphic it suffices to show that one particular property holds for one structure but not for the other. In this paragraph we discuss two such properties and explain how `MULTI` combines the strategies `ReduceToSpecial` and `TryAndError` to establish that two structures are not isomorphic. Thereby `ReduceToSpecial` contains theorems that can reduce the original goal to subgoals stating that a property does not hold for one structure whereas it holds for the other structure. These subgoals can then be proved with `TryAndError`.

First we introduce the concepts *order*, *trace*, and *order of the trace* of elements of a structure (S, \circ) , where S is a finite set:

- An element $a \in S$ has the *order* n if $n \in \mathbb{N}$ is the smallest positive integer such that $a^n = \underbrace{a \circ \dots \circ a}_{n\text{-times}} = e$, where $e \in S$ is the unit element with respect to \circ . In the following we write this as *order*(a).
- The *trace* of an element $a \in S$ is the set $\{a^n | n \in \mathbb{N}\}$. The cardinality of this set is referred to as the *order of the trace* of a . This is written as *ordertr*(a) in the following.

The latter concept is a generalization of the former so we can also deal with elements that do not have an order or with structures which do not have a unit element. Note also, that both the order of an element a and the order of its trace always range between 1 and the cardinality of S .

For two structures (S^1, \circ^1) and (S^2, \circ^2) we know that if they are isomorphic then for each element $a_1 \in S^1$ with order n there exists an element $a_2 \in S^2$ with the same order. Moreover, we know an analogous statement for the order of the traces. Thus, to prove that two structures are not isomorphic it is sufficient to prove that one structure contains an element a_1 such that the other structure contains no element a_2 whose order (order of the trace) is equal to the order (order of the trace) of a_1 . This can be formalized in the following theorems, where $[1, \text{card}(S^1)]$ denotes the integer interval from 1 to the cardinality of S^1 :

- $(\exists n: [1, \text{card}(S^1)] \bullet (\exists x_1: S^1 \bullet \text{order}(x_1, S^1, \circ^1) = n) \wedge (\neg \exists x_2: S^2 \bullet \text{order}(x_2, S^2, \circ^2) = n)) \Rightarrow \neg \text{iso}(S^1, \circ^1, S^2, \circ^2)$
- $(\exists n: [1, \text{card}(S^1)] \bullet (\exists x_1: S^1 \bullet \text{ordertr}(x_1, S^1, \circ^1) = n) \wedge (\neg \exists x_2: S^2 \bullet \text{ordertr}(x_2, S^2, \circ^2) = n)) \Rightarrow \neg \text{iso}(S^1, \circ^1, S^2, \circ^2)$

The **ReduceToSpecial** strategy can apply these two theorems to reduce non-isomorphism goals and then **TryAndError** takes over to complete the proof. For instance, the application of the second theorem to the problem to prove that the two Abelian semi-groups $(\mathbb{Z}_4, \lambda xy \bullet x \bar{*} y \bar{*} \bar{2}_4)$ and $(\mathbb{Z}_4, \lambda xy \bullet \bar{2}_4)$ are not isomorphic results in the problem to prove that there exists an integer n such that: (1) there exists an $x_1 \in \mathbb{Z}_4$ such that the cardinality of the trace of x_1 with respect to the first operation $\lambda xy \bullet x \bar{*} y \bar{*} \bar{2}_4$ equals n , (2) for all $x_2 \in \mathbb{Z}_4$ the cardinality of the trace of x_2 with respect to the second operation $\lambda xy \bullet \bar{2}_4$ is not n . Since the order of the trace can be at most the cardinality of the involved set \mathbb{Z}_4 , the possible values for n can be restricted to 1, 2, 3, and 4. Since n ranges also over a finite set we can apply the **TryAndError** strategy to prove this goal. To restrict the search, **MULTI** obtains hints for suitable instantiations for n and x_1 . The hints are computed by constructing the traces with **GAP**. In our example the suitable instantiations are $n=3$ and $x_1=\bar{1}_4$ (the trace of $\bar{1}_4$ in $(\mathbb{Z}_4, \lambda xy \bullet x \bar{*} y \bar{*} \bar{2}_4)$ is $\{\bar{1}_4, \bar{2}_4, \bar{0}_4\}$) since the traces of all elements of $(\mathbb{Z}_4, \lambda xy \bullet \bar{2}_4)$ are either of order 1 or 2.

In contrast to employing **TryAndError** alone, proofs constructed with the combination of **TryAndError** and **ReduceToSpecial** have only polynomial complexity in the cardinality of the involved sets. Moreover, the search is reduced significantly by providing hints. But this technique is only applicable when structures involved contain elements suitable for our purpose in the sense that either

their order or the order of their trace is not reflected in the respective other structure.

NotInjNotIso The strategy `NotInjNotIso` was particularly implemented for non-isomorphism proofs. Its idea is to construct an indirect proof, which shows that two structures (S^1, \circ^1) and (S^2, \circ^2) are not isomorphic. We first assume that there exists a function $h: S^1 \rightarrow S^2$ which is an isomorphism. Then h is an injective homomorphism. The strategy `NotInjNotIso` tries to find two elements $c_1, c_2 \in S^1$ with $c_1 \neq c_2$ such that we can derive the equation $h(c_1) = h(c_2)$. This contradicts the assumption of injectivity of h where $h(c_1) \neq h(c_2)$ has to hold if $c_1 \neq c_2$. Note, that the proof is with respect to all possible homomorphisms h and we do not have to give a particular mapping.

We explain the `NotInjNotIso` strategy for our example that $(\mathbb{Z}_4, \lambda xy \bullet \bar{2}_4)$ is not isomorphic to $(\mathbb{Z}_4, \lambda xy \bullet x \bar{*} y \bar{*} \bar{2}_4)$. The strategy first constructs the situation for the indirect argument. From the hypothesis that the two structures are isomorphic follow the two assumptions that there exists a function h that is (1) injective and (2) a homomorphism. By the first assumption a contradiction can be concluded, when we are able to show that h is not injective.

`MULTI` continues by applying a method to the second assumption, that introduces the homomorphism equation $h(x \circ^1 y) = h(x) \circ^2 h(y)$ instantiated for every element of the domain as new assumptions. In the above example 16 equations like

$$h(\bar{0}_4) = \bar{2}_4 \text{ for } x = \bar{0}_4, y = \bar{0}_4, \quad h(\bar{2}_4) = \bar{2}_4 \text{ for } x = \bar{1}_4, y = \bar{1}_4, \quad \dots$$

are introduced, where the actual operations $\circ^1 = \lambda xy \bullet x \bar{*} y \bar{*} \bar{2}_4$ and $\circ^2 = \lambda xy \bullet \bar{2}_4$ are already applied to the given arguments.

From the introduced system of equations the `NotInjNotIso` strategy tries to derive that h is not injective. To prove this we have to find two witnesses c_1 and c_2 such that $c_1 \neq c_2$ and $h(c_1) = h(c_2)$. In our example $\bar{0}_4$ and $\bar{2}_4$ are chosen for c_1 and c_2 , respectively, which leads to $h(\bar{0}_4) = h(\bar{2}_4)$. This goal is transformed into an equation that can be solved in a general way, by successively applying equations from the equation system. In our example $h(\bar{0}_4) = h(\bar{2}_4)$ is reduced to $\bar{2}_4 = \bar{2}_4$ by two substitutions using the equalities given above. The last equation is justified by the reflexivity of equality and the proof is finished.

In order to restrict the search for appropriate c_1 and c_2 `NotInjNotIso` employs a control rule to obtain a hint. The control rule calls `MAPLE` to compute all possible solutions for the system of instantiated homomorphism equations with respect to the corresponding modulo factor using `MAPLE`'s function `msolve`. Then the solutions are checked for whether there is a pair c_1 and c_2 with $c_1 \neq c_2$, such that in all of the solutions $h(c_1) = h(c_2)$ holds. If there is such a pair it is provided has a hint. Although the control rule cannot always come up with a hint, our experiments have shown that the `NotInjNotIso` strategy is also often successful when no hint can be computed.

In our example the equational reasoning involved is still relatively simple and could be done by a more specialized system such as a term rewriting system. However, this is not possible in the general case. Then the equations contain

more complex terms involving addition, multiplication, and subtraction of constant congruence classes of the form $h(cl_n(i))$ and thus additionally have to be performed with respect to the correct modulo factor. The solution of the equations is therefore beyond the scope of any term rewriting system but requires symbolic computation.

`NotInjNotIso` can produce very short proofs even for structures with large sets. However, to construct an appropriate sequence of equality substitutions is generally the hard part of proofs with `NotInjNotIso`. In fact, for problems with the same complexity (i.e., problems involving structures of the same cardinality) the length of the proofs can vary drastically. Moreover, the equational reasoning process does not have to terminate. Therefore, we experimented with randomization and restart techniques known from Artificial Intelligence [6]. It turned out, that the introduction of a stochastic element when choosing the next instantiated homomorphism equation to apply, coupled with restarts based on statistical measures (i.e., a proving attempt is interrupted after a certain time interval and a new proving attempt is started instead) can significantly increase the efficiency and robustness of proof planning with the `NotInjNotIso` strategy. A complete description of the performed experiments as well as how randomization and restarts are realized in `MULTI` can be found in [11].

`ReduceToSpecial` is the first strategy that is tried when automatically discharging non-isomorphism proof obligations. If it fails first the `NotInjNotIso` strategy is tried before `TryAndError`. The `EquSolve` strategy is not applicable to non-isomorphism problems.

5 Experiments and Results

The proving techniques presented in this paper mainly build on the strategies already constructed for the proofs of simple properties of the residue class structures as presented in [13]. There we used a total of 21 examples to construct the basic versions of the `ReduceToSpecial`, `TryAndError`, and `EquSolve` strategies. To develop the extensions of these strategies to handle isomorphism and non-isomorphism proofs we used 15 examples and another 4 examples to build the `NotInjNotIso` strategy.

To show the validity of the techniques for isomorphism and non-isomorphism proofs we applied our classification process to 8128 structures with the set \mathbb{Z}_6 . Here, we found 4152 magmas, 73 Abelian magmas, 1114 semi-groups, 1025 Abelian semi-groups, 738 quasi-groups, 257 Abelian quasi-groups, 50 Abelian monoids, and 419 Abelian groups. For the quasi-groups and the Abelian quasi-groups we found that they belong to two different classes, respectively. All Abelian monoids and the Abelian groups belong to the same class, respectively. Furthermore, we found 12 non-isomorphic classes of Abelian semi-groups, eight classes of semi-groups, five classes of Abelian magmas, and seven classes of magmas. 90% of the necessary isomorphism proofs were done with the `EquSolve` strategy, the other 10% were done with `TryAndError`. During the automatic classification 121 non-isomorphism proofs were constructed. Here 80% of the proofs

were done with the `NotInjNotIso` strategy and the remaining 20% with the combination of `TryAndError` and `ReduceToSpecial`. In addition to the automatic classification process we did separate experiments with 800 non-isomorphism proofs to obtain suitable cutoff values (i.e., when to restart the `NotInjNotIso` strategy) by analyzing the search spaces [11].

6 Related Work and Conclusions

We have presented an experiment in exploring properties of residue classes over the integers with the combined effort of the multi-strategy proof planner `MULTI` and the two computer algebra systems `MAPLE` and `GAP`. In our experiments we classify residue class sets over the integers together with binary operations in terms of what algebraic structure they form and then we divide structures of the same algebraic category into isomorphism classes. Arising proof obligations are discharged by `MULTI` with several strategies that realize different proving techniques of the problem domain. The proof planning in our problem domain benefits considerably from the possibilities `MULTI` provides. Using `MULTI` we were not only able to encode several different proving techniques in a conceptually clear way into different strategies, but could also combine and interleave these strategies flexibly. We employed the computer algebra systems to guide and simplify both the classification and the proof planning process. We have tested the validity of our techniques with a large number of experiments. It turned out that the implemented machinery is not only robust but that the elaborate strategies are successful on a large number of examples. Overall a considerable part of the problems have been proved with various usages of computer algebra. Although not explicitly mentioned in this paper we can also deal with direct products of residue class sets and classify structures with two operations [12].

There are various accounts on experiments of combining computer algebra and theorem proving in the literature (see [8] for just a few). However, they generally deal with the technical and architectural aspects of those integrations as well as with correctness issues and not with the application of the combined systems to a specific problem domain. A possibly fruitful cooperation between the deduction system `Nuprl` and the computer algebra system `Weyl` in the domain of abstract algebra is sketched in [7]. Our paper in contrast presents the application of an already existing combination of proof planning and computer algebra to a specific problem domain. We thereby exploit work previously done in `OMEGA` [9, 18].

More concrete work in exploration in finite algebra is reported in [4, 10, 17] where model generation techniques are used to tackle quasi-group existence problems. In particular, some open problems in quasi-group theory were solved. The motivation for all this work is roughly to specify certain properties of an algebra and then to try to automatically construct a structure that satisfies the required properties. Thus, the constructed algebra might actually be a new discovery. Our work is diametrical in the sense that we start out with given structures and classify them with respect to their algebraic properties and whether they

are isomorphic. Likewise, our automatic exploration processes depend on sets of pre-constructed residue class sets and operations. In addition both classification and exploration is currently not designed to intentionally discover new algebraic structures.

References

1. A. Bundy. The Use of Explicit Plans to Guide Inductive Proofs. In *Proceedings of CADE-9*, volume 310 of *LNCS*. Springer, 1988.
2. C. Benz Müller et al. Ω Mega: Towards a Mathematical Assistant. In *Proceedings of CADE-14*, volume 1249 of *LNAI*. Springer, 1997.
3. E. Melis et al. ACTIVE MATH system description. In *Artificial Intelligence and Education*, 2001.
4. M. Fujita, J. Slaney, and F. Bennett. Automatic generation of some results in finite algebra. In *Proceedings of IJCAI'93*. Morgan Kaufmann, 1993.
5. The GAP Group. *GAP - Groups, Algorithms, and Programming, Version 4*, 1998. <http://www-gap.dcs.st-and.ac.uk/~gap>.
6. C. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of AAAI-98*. AAAI Press, 1998.
7. P. Jackson. Exploring Abstract Algebra in Constructive Type Theory. In *Proceedings of CADE-12*, volume 814 of *LNCS*. Springer, 1994.
8. D. Kapur and D. Wang, editors. *Journal of Automated Reasoning— Special Issue on the Integration of Deduction and Symbolic Computation Systems*, volume 21(3). Kluwer Academic Publisher, 1998.
9. M. Kerber, M. Kohlhase, and V. Sorge. Integrating Computer Algebra Into Proof Planning. *Journal of Automated Reasoning*, 21(3), 1998.
10. W. McCune. Otter 3.0 Reference Manual and Guide. Technical Report ANL-94-6, Argonne National Laboratory, Argonne, IL, USA, 1994.
11. A. Meier. Randomization and heavy-tailed behavior in proof planning. Seki Report SR-00-03, Fachbereich Informatik, Universität des Saarlandes, 2000.
12. A. Meier, M. Pollet, and V. Sorge. Exploring the domain of residue classes. Seki Report SR-00-04, Fachbereich Informatik, Universität des Saarlandes, 2000.
13. A. Meier and V. Sorge. Exploring Properties of Residue Classes. In *Proceedings of Calculemus 2000*. AK Peters, 2001.
14. E. Melis and A. Meier. Proof planning with multiple strategies. In *Proc. of the First International Conference on Computational Logic*. Springer, 2000.
15. E. Melis and J. Siekmann. Knowledge-based proof planning. *Artificial Intelligence*, 115(1), 1999.
16. D. Redfern. *The Maple Handbook: Maple V Release 5*. Springer, 1999.
17. J. Slaney, M. Fujita, and M. Stickel. Automated reasoning and exhaustive search: Quasigroup existence problems. *Computers and Mathematics with Applications*, 29, 1995.
18. V. Sorge. Non-Trivial Computations in Proof Planning. In *Proceedings of FroCoS 2000*, volume 1794 of *LNCS*. Springer, 2000.
19. Z. Zilic and K. Radecka. On feasible multivariate polynomial interpolations over arbitrary fields. In *Proceedings of ISSAC-99*. ACM Press, 1999.
20. R. Zippel. Interpolating polynomials from their values. *Journal of Symbolic Computation*, 9(3), 1990.