

Exploring Properties of Residue Classes

Andreas Meier and Volker Sorge*

Fachbereich Informatik, Universität des Saarlandes, Germany

{ameier|sorge}@ags.uni-sb.de

Abstract. *We report on an experiment in exploring properties of residue classes over the integers with the combined effort of a multi-strategy proof planner and two computer algebra systems. An exploration module classifies a given set and a given operation in terms of the algebraic structure they form. It then calls the proof planner to prove or refute simple properties of the operation. Moreover, we use different proof planning strategies to implement various proving techniques: from naive testing of all possible cases to elaborate techniques of equational reasoning and reduction to known cases.*

1 Introduction

We report on an experiment in exploring properties of operations on subsets of residue classes over the integers. The work was done within the scope of a project on an interactive mathematical course in algebra. For tutoring purposes it is necessary to have a large class of examples and counter-examples available to illustrate the difference of notions like group, monoid, etc. The examples also need to be suitable to teach different proving techniques. However, our work only provides the raw data that needs to be further processed to be usable within a tutor system. This process as well as how the examples are suitably presented to students are not topic of this paper.

We have chosen residue classes over the integers as a problem domain for the following reasons: Since operations on residue classes are relatively easy to handle automatically and intuitive to explain to a user they are ideal to exemplify some important concepts in algebra, such as associativity, unit elements, etc. Moreover, they provide a large number of possible examples and counter examples for the different concepts.

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

A user of the interactive course can experiment with constructs such as $(\mathbb{Z}_3, x\bar{*}y)$, $(\mathbb{Z}_3 \setminus \{\bar{1}_3\}, (x\bar{+}x)\bar{*}y)$, \dots . These are automatically classified in terms of algebraic structures, i.e., whether they form a magma, semi-group, quasi-group, monoid, loop, or group, by constructing the appropriate proofs which can in turn be used to guide the user in his experiments.

Technically, we have realized our goals by exploiting the infrastructure of the Ω MEGA system [6]. We use an exploration module that constructs proof obligations with respect to a given set and operation. These proof obligations are theorems of the form: the set is closed under the operation, or the operation is not associative, etc. Proof obligations are passed to Ω MEGA's multi-strategy proof planner MULTI [12] that constructs a proof with the help of the two computer algebra systems MAPLE [14] and GAP [4]. Although the proofs are not very pretentious in their own right they need to be constructed in a way that different proving techniques can be taught to a user of the interactive course. This is achieved by supplying distinct strategies to the proof planner so resulting plans reflect different proof approaches. In particular, we have implemented three different planning strategies: a very naive one testing all possible cases, a more elaborate one using as much as possible equational reasoning, and a third one whose goal is to apply already known theorems. We conducted a large number of experiments to test both the effectiveness and robustness of our proof planning approach.

The paper is organized as follows: Section 2 introduces the problem domain in more detail and Sec. 3 gives an account of multi-strategy proof planning in Ω MEGA. Section 4 explains how computer algebra aids during the proof planning process, Sec. 5 elaborates on the different proof strategies that realize different proving techniques, Sec. 6 reports on the exploration module employed to generate proof obligations, and Sec. 7 gives an account of the experiments we have conducted with the implemented machinery. We conclude by discussing some related work and hinting at some future work.

2 The Problem Domain

In this section we describe the kind of problems we are interested in together with the notation we will use throughout the remainder of the paper.

We are interested in classifying residue class sets over the integers together with given binary operations in terms of their basic algebraic properties. A residue class set over the integers 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 . Con-

cretely, we will be 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 given in λ -function notation. \circ can be of the form $\lambda xy. x, \lambda xy. y, \lambda xy. c$ where c is a constant congruence class (e.g., $\bar{1}_3$), $\lambda xy. x \bar{+} y, \lambda xy. x \bar{*} y, \lambda xy. x \bar{-} y$, 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., $\lambda xy. (x \bar{+} \bar{1}_3) \bar{-} (y \bar{+} \bar{2}_3)$.

Given a residue class set RS_n modulo n and a binary operation \circ with respect to the same modulo factor n we try to establish or refute the following properties:

1. **Closure:** RS_n is closed under \circ . This is formalized by the defined concept $closed(RS_n, \circ)$ that abbreviates $\forall x:RS_n. \forall y:RS_n. (x \circ y) \in RS_n$. Here the variables x and y are of sort RS_n , i.e., the universal quantifiers range over the finite domain RS_n .
2. **Associativity:** RS_n is associative with respect to \circ . ($assoc(RS_n, \circ) \equiv \forall x:RS_n. \forall y:RS_n. \forall z:RS_n. x \circ (y \circ z) = (x \circ y) \circ z$.)
3. **Unit element:** There exists a unit element in RS_n with respect to \circ . ($\exists e:RS_n. unit(RS_n, \circ, e) \equiv \exists e:RS_n. \forall y:RS_n. (y \circ e = y) \wedge (e \circ y = y)$.)
4. **Inverses:** Every element in RS_n has an inverse element with respect to \circ and the unit element e . ($inverse(RS_n, \circ, e) \equiv \forall x:RS_n. \exists y:RS_n. (x \circ y = e) \wedge (y \circ x = e)$.)
5. **Divisors:** For every two elements of RS_n there exist two corresponding divisors in RS_n with respect to \circ . ($divisors(RS_n, \circ) \equiv \forall a:RS_n. \forall b:RS_n. (\exists x:RS_n. a \circ x = b) \wedge (\exists y:RS_n. y \circ a = b)$.)

Hence, we classify (RS_n, \circ) with regard to the algebraic structure it forms, i.e., whether it is a magma (property 1 holds), semi-group (1+2), monoid (1+2+3), quasi-group (1+5), loop (1+5+3), or group (1+2+3+4)¹.

3 Multi-Strategy Proof Planning

Proof planning [2] 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.

¹Naturally property 5 holds for a group as well.

In the Ω MEGA system [6] the traditional proof planning approach is enriched by incorporating mathematical knowledge into the planning process (see [13] 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 preferring, rejecting, or enforcing the application of methods in certain domains or proof situations. Ω MEGA's new proof planner, MULTI [12], allows also for the specification of different planning strategies to control the overall planning behavior. Proof plans in Ω MEGA result in proof objects in a variant of the natural deduction calculus [5]. We will present them in this paper in a linearized style as introduced in [1]. A proof line is of the form $L. \Delta \vdash F (\mathcal{R})$, where L is a unique label, $\Delta \vdash F$ a sequent denoting that the formula F can be derived from the set of hypotheses Δ , and (\mathcal{R}) is a justification expressing how the line was derived.

Methods are generally represented in a declarative way as for instance \exists IResclass given below, a method specific for our problem domain. Its purpose is to instantiate an existentially quantified variable over a residue class set with a witness term for which a certain property P holds and to reduce the initial statement on residue classes to a statement on integers. The witness term has to be a concrete element of the residue class set. However, if the method is applied in an early stage of the proof, the planner generally has no knowledge on the true nature of the witness term. Therefore, the method invokes a middle-out-reasoning [10] process to postpone the actual instantiation, that is, a meta-variable is used as placeholder for the actual witness term which will be determined at a later point in the planning process and subsequently substituted.

Method: \exists IResclass			
Premises	$\oplus L_3, \oplus L_1$		
Appl. Cond.	$ResclassSet(RS_n, n, N_{set})$		
Conclusions	$\ominus L_5$		
Declarative Content	(L_1)	Δ	$\vdash mv \in N_{set}$ (open)
	(L_2)	Δ	$\vdash c \in RS_n$ (ConResclSet L_1)
	(L_3)	Δ	$\vdash P[cl_n(mv)]$ (open)
	(L_4)	Δ	$\vdash P[c]$ (ConRescl L_3)
	(L_5)	Δ	$\vdash \exists x:RS_n. P[x]$ (\exists sortI $L_2 L_4$)

\exists IResclass is given in terms of the original goal, the *conclusion* L_5 , the two new open goals it produces, the *premises* L_1 and L_3 , and the inference steps deriving L_5 from L_1 and L_3 , given in the *declarative content*.

The method is applicable during the planning process if the current planning goal can be matched against the formula of L_5 and if additionally the *application conditions* (*Appl. Cond.*) are satisfied. The condition $ResclassSet(RS_n, n, N_{set})$ is fulfilled if RS_n , the sort of the quantified variable x , qualifies as a residue class set of the form given in Sec. 2. Its successful evaluation binds the method variables n and N_{set} to the modulo factor of RS_n and the set of integers corresponding to the congruence classes of RS_n , respectively. For instance, the evaluation of $ResclassSet(\mathbb{Z}_2, n, N_{set})$ yields $n \mapsto 2$ and $N_{set} \mapsto \{0, 1\}$. The necessary inference steps are indicated by the justifications *ConResclSet* and *ConRescl* in lines L_2 and L_4 . A congruence class with respect to the modulo factor n is denoted as $cl_n(mv)$ in line L_3 where mv is the introduced meta-variable.

To influence the planners behavior control-rules can be specified and are evaluated at certain choice points, e.g., the selection of a planning goal, or of an applicable method, or an instantiation of a meta-variable. An example of a control-rule used in our context is *TryAndErrorStandardSelect*:

```
(control-rule TryAndErrorStandardSelect
  (kind methods)
  (IF (disjunction-supports S))
  (THEN (select (VResclass
                ConCongCl
                (VE** () (S))
                EResclass))))
```

The control-rule is evaluated before a method is chosen by the planner. It states that if the current goal is supported by a disjunctive support line S the application of the methods *VResclass*, *ConCongCl*, *VE***, and *EResclass* is attempted in this order. *VResclass* and *ConCongCl* are domain-specific methods where the latter converts statements on residue classes into corresponding statements on integers. The former reduces goals containing a universal quantification over a residue class set similar to *EResclass*. On the contrary, *VE*** is not a domain-specific method. When applied it performs a case split with respect to a set of disjunctive supports of a goal.

Different problem solving behaviors are realized using the multi-strategy extension *MULTI* [12] of the conventional proof planner. Essentially *strategies* are used to parameterize certain algorithms of the planner. A planning algorithm like forward- or backward-planning can be supplied with termination criteria and a certain set of methods and control rules to determine its behavior. One strategy we developed in the context of residue classes is a parameterization of the standard planning algorithm *PPlanner*:

Strategy: TryAndError		
Appl-Cond	ResidueClassGroupProperty	
Algorithm	<i>PPlanner</i>	
Parameters	Methods	\forall Resclass, ConCongCl, $\forall E^{**}$, \exists Resclass, ...
	C-Rules	TryAndErrorStandardSelect, ...
	Termin.	No-Subgoal

According to its application conditions (Appl-Cond) **TryAndError** can be applied to goals stating one of the properties given in Sec. 2. It employs the standard planning algorithm *PPlanner* with a certain set of methods and a certain set of control-rules. The strategy either fails or terminates when its initial goal is fully justified. The application and the selection of the different strategies can also be influenced by *strategic control-rules* which — similar to control-rules about methods — prefer, reject, or enforce the application of strategies in certain situations.

4 Using Computer Algebra

In this section we describe how we employ symbolic calculations to guide and simplify the search for proof plans. In particular, we use the mainstream computer algebra system MAPLE [14] and GAP [4], a system specialized on algebra. We will not be concerned with the technical side and the soundness issues of integrating computer algebra with proof planning in general and with Ω MEGA in particular since this has already been discussed in [16, 9]. 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 method-applications equations are solved with MAPLE to simplify the proof. As side-effect both cases can restrict possible instantiations of meta-variables.

(1) is implemented in the control rule `select-instance` which is used in the **TryAndError** strategy. 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 , i.e., that 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.

The hint system is implemented as follows: We have implemented a small routine in Ω MEGA that constructs a multiplication table with respect to the given set and operation. It is employed to check the closure property and to check the existence of divisors, i.e., the axiom for quasi-groups. Furthermore, if the computed multiplication table is closed under the respective operation it is used to construct the appropriate magma in GAP. GAP can then be employed to query for associativity and to compute the unit element and inverses for the single elements. Most query functions return useful results in both the positive and the negative case: That is, for instance, if GAP can compute a unit element for a given magma this element is returned. In case GAP fails to find a unit element the multiplication table is used to determine a set of elements that suffice to refute the existence of a unit element for the given magma. A special case is the failure of the query for associativity, since then we try to use MAPLE to compute a particular solution for the associativity equation. If such a non-general solution exists it is exploited to determine a triple of elements for which associativity does not hold.

Use (2) of calculations is realized within the `SolveEquation` method. Its purpose is to justify an equational goal using MAPLE and to instantiate meta-variables if necessary. 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. If the equation contains meta-variables 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.

5 Using Different Strategies

To be able to demonstrate different proving techniques we employ three different strategies for the MULTI proof planner. In this section we elaborate those strategies using examples to point out the major differences while trying to avoid the tedious details.

5.1 Exhaustive Case Analysis

The motivation for the first strategy, called `TryAndError`, is to implement a rather naive approach of proving a property of a residue class set. It proceeds by rewriting statements on residue classes into corresponding statements on integers, especially by transforming the residue class set into a set of corresponding integers. It then exhaustively checks all possible

$L_1.$	L_1	$\vdash c'_1 \in \mathbb{Z}_2$	(Hyp)
$L_2.$	L_1	$\vdash c \in \{0, 1\}$	$(ConResclSet L_1)$
$L_3.$	L_3	$\vdash c = 0$	(Hyp)
		\vdots	
$L_{12}.$	L_1, L_3	$\vdash \exists y: \mathbb{Z}_{2^*} (cl_2(c) \bar{+} y = \bar{0}_2) \wedge (y \bar{+} cl_2(c) = \bar{0}_2)$	$(\exists IResclass L_{11} L_{10})$
$L_{13}.$	L_{13}	$\vdash c = 1$	(Hyp)
$L_{14}.$	L_1, L_{13}	$\vdash 0 = 0$	$(Solve-Equation)$
$L_{15}.$	L_1, L_{13}	$\vdash mv \in \{0, 1\}$	$(\forall IR L_{14})$
$L_{16}.$	L_1, L_{13}	$\vdash 0 = 0$	$(Solve-Equation)$
$L_{17}.$	L_1, L_{13}	$\vdash 0 = 0$	$(Solve-Equation)$
$L_{18}.$	L_1, L_{13}	$\vdash (1 + c) \bmod 2 = 0 \bmod 2$	$(SimplNum L_{13} L_{16})$
$L_{19}.$	L_1, L_{13}	$\vdash (c + 1) \bmod 2 = 0 \bmod 2$	$(SimplNum L_{13} L_{17})$
$L_{20}.$	L_1, L_{13}	$\vdash (c + 1) \bmod 2 = 0 \bmod 2 \wedge$ $(1 + c) \bmod 2 = 0 \bmod 2$	$(\wedge I L_{18} L_{19})$
$L_{21}.$	L_1, L_{13}	$\vdash (cl_2(c) \bar{+} cl_2(mv) = \bar{0}_2) \wedge$ $(cl_2(mv) \bar{+} cl_2(c) = \bar{0}_2)$	$(ConCongCl L_{20})$
$L_{22}.$	L_1, L_{13}	$\vdash \exists y: \mathbb{Z}_{2^*} (cl_2(c) \bar{+} y = \bar{0}_2) \wedge (y \bar{+} cl_2(c) = \bar{0}_2)$	$(\exists IResclass L_{21} L_{15})$
$L_{23}.$	L_1	$\vdash \exists y: \mathbb{Z}_{2^*} (cl_2(c) \bar{+} y = \bar{0}_2) \wedge (y \bar{+} cl_2(c) = \bar{0}_2)$	$(\forall E** L_2 L_{12} L_{22})$
$L_{24}.$		$\vdash \forall x: \mathbb{Z}_{2^*} \exists y: \mathbb{Z}_{2^*} (x \bar{+} y = \bar{0}_2) \wedge (y \bar{+} x = \bar{0}_2)$	$(\forall IResclass L_{23})$
$L_{25}.$		$\vdash inverse(\mathbb{Z}_2, \lambda xy. x \bar{+} y, \bar{0}_2)$	$(DefnCon L_{24})$

combinations of these integers with respect to the property that is to be proved or refuted. `TryAndError` proceeds in two different ways, depending on whether (1) a universal or (2) an existential property has to be proved. Both cases can be observed in the example proof of the statement that \mathbb{Z}_2 has inverses with respect to the operation $\lambda xy. x \bar{+} y$ and the unit element $\bar{0}_2$, given above.

In case (1) a split over all the elements in the set involved is performed and the property is proved for every single element separately. We observe this in the proof of the universally quantified formula in line L_{24} . An application of the method `$\forall IResclass$` to L_{24} yields the lines L_{23} , L_1 , and L_2 . The disjunction contained in L_2 ($c \in \{0, 1\}$) can be viewed as $c = 0 \vee c = 1$ triggers the first case split with the application of `$\forall E**$` . Subsequently `MULTI` tries to prove the goal in line L_{23} twice (in the lines L_{12} and L_{22}), once assuming $c = 0$ (in line L_3) and once assuming $c = 1$ (in line L_{13}).

In case (2) the single elements of the set involved are examined until one is found for which the property in question holds. In our example proof this is, for instance, done after the application of the method `$\exists IResclass$` to L_{22} yielding the lines L_{15} and L_{21} . The case analysis is then performed by successively choosing different possible values for mv with the `$\forall IR$` , `$\forall IL$` methods — in our example mv is either 0 or 1 as given in line L_{15} . For a se-

lected instantiation `MULTI` can then either finish the proof or it backtracks to test the next instantiation. To minimize this search the `TryAndError` strategy enables `MULTI` to invoke the `select-instance` control-rule after the application of `∃IResclass`. As described in Sec. 4 `select-instance` can compute a hint on the likely instantiation for the meta-variable mv it is used as instantiation for the very first case. If no hint can be computed or the proof fails despite the hint `MULTI` proceeds with its conventional search.

After eliminating all quantifiers and performing all possible case splits the `TryAndError` strategy reduces all remaining statements on residue and congruence classes to statements on integers. These are then solved by numerical simplification and equational reasoning.

5.2 Equational Reasoning

The aim of the second strategy, called `EquSolve`, is to use as much as possible equational reasoning to prove properties of residue classes. Similarly to the `TryAndError` strategy it converts statements on residue classes into corresponding statements on integers. But instead of then checking the validity of the statements for all possible cases, it tries to solve occurring equations in a general way. The strategy can be successfully applied to prove associativity, the existence of a unit element, inverses and divisors, only. We observe its approach with a proof of the same example theorem $inverse(\mathbb{Z}_2, \lambda xy. x \bar{+} y, \bar{0}_2)$ as in Sec. 5.1:

$L_1. L_1 \vdash c'_1 \in \mathbb{Z}_2$	(<i>Hyp</i>)
$L_2. L_1 \vdash c \in \{0, 1\}$	(<i>ConResclSet</i> L_1)
$L_{15}. L_1 \vdash mv \in \{0, 1\}$	(<i>Open</i>)
$L_{18}. L_1 \vdash (mv + c) \bmod 2 = 0 \bmod 2$	(<i>Solve-Equation</i> { $mv \leftarrow c$ })
$L_{19}. L_1 \vdash (c + mv) \bmod 2 = 0 \bmod 2$	(<i>Solve-Equation</i>)
$L_{20}. L_1 \vdash (c + mv) \bmod 2 = 0 \bmod 2 \wedge$ $(mv + c) \bmod 2 = 0 \bmod 2$	($\wedge I$ L_{19} L_{18})
$L_{21}. L_1 \vdash (cl_2(c) \bar{+} cl_2(mv) = \bar{0}_2) \wedge$ $(cl_2(mv) \bar{+} cl_2(c) = \bar{0}_2)$	(<i>ConCongCl</i> L_{20})
$L_{22}. L_1 \vdash \exists y: \mathbb{Z}_2. ((cl_2(c) \bar{+} y = \bar{0}_2) \wedge (y \bar{+} cl_2(c) = \bar{0}_2))$	($\exists IResclass$ $L_{21} L_{15}$)
$L_{24}. \vdash \forall x: \mathbb{Z}_2. \exists y: \mathbb{Z}_2. ((x \bar{+} y = \bar{0}_2) \wedge (y \bar{+} x = \bar{0}_2))$	($\forall IResclass$ L_{23})
$L_{25}. \vdash inverse(\mathbb{Z}_2, \lambda xy. x \bar{+} y, \bar{0}_2)$	(<i>DefnCon</i> L_{24})

The construction of the proof is in the beginning (lines L_{25} through L_{20}) nearly analogous to the one in the preceding section. The only exception is that no case splits are carried out after the applications of `∧IResclass` and `∃IResclass`. Instead we get two equations in the lines L_{18} and L_{19} which

can be generally solved using the `SolveEquation` method. As described in Sec. 4 this method uses MAPLE to compute a general solution and possibly substitutes meta-variables. In the case of our example the meta-variable mv is substituted by c during the first application of `SolveEquation`. This is indicated by $\{mv \leftarrow c\}$ in the justification of line L_{18} . However, this substitution is not carried out directly during the proof planning process but merely added as a constraint on the meta-variable. Nevertheless, from there on the proof planner treats any occurrence of mv as if it were substituted. Once a complete proof plan is constructed all computed substitutions for meta-variables are applied. This treatment of meta-variable substitutions simplifies the backtracking procedure of the proof planner.

The constraint on mv changes the formula implicitly in the remaining open goal L_{15} to $c \in \{0, 1\}$. Since the naive testing of the possible cases fails (neither $c = 0$ nor $c = 1$ hold immediately), the goal is reduced to $closed(\mathbb{Z}_2, \lambda xy. x)$, an assertion about a residue class property. This problem is then returned to MULTI which in turn selects and applies again strategies to this goal.

5.3 Applying Known Theorems

The motivation for our third strategy `ReduceToSpecial` is to incorporate the application of already proved theorems. Contrary to `TryAndError` and `EquSolve` whose idea is to reduce expressions on residue classes to expressions on numbers, the `ReduceToSpecial` strategy tries to tackle new problems by applying already known theorems. Theorems are stored in OMEGA's theory database and, in order to keep the number of possible theorems small, only those in the theory of residue classes are eligible for application.

`ReduceToSpecial` uses two methods to apply theorems: the primary method is `ApplyAssertion` that applies theorems that match an open goal. To ensure termination `ApplyAssertion` uses first-order matching with α -equality on λ -abstractions, only. `ApplyAssertion` is a general method to apply arbitrary theorems and is not tailored for particular problem domains and it is, in particular, not always sufficient for our examples. Therefore, it is necessary to have other, domain-specific methods that are able to apply theorems in a more complex way. So far, we have implemented one such method, `ReduceClosed`, that is specialized to apply theorems containing statements on closure properties of residue classes.

We observe the behavior of the `ReduceToSpecial` strategy with the following proof for the theorem $closed(\mathbb{Z}_5, \lambda x, y. (x * y) \bar{+} \bar{3}_5)$:

$L_3.$	$\vdash \bar{3}_5 \in \mathbb{Z}_5$	$(InResclSet)$
$L_4.$	$\vdash 5 \in \mathbb{Z}$	$(InInt)$
$L_5.$	$\vdash closed(\mathbb{Z}_5, \lambda xy_{\bullet}x)$	$(ApplyAss\ ClosedFV)$
$L_6.$	$\vdash closed(\mathbb{Z}_5, \lambda xy_{\bullet}y)$	$(ApplyAss\ ClosedSV)$
$L_7.$	$\vdash 5 \in \mathbb{Z}$	$(InInt)$
$L_8.$	$\vdash closed(\mathbb{Z}_5, \lambda xy_{\bullet}\bar{3}_5)$	$(ApplyAss\ ClosedConst\ L_3)$
$L_9.$	$\vdash closed(\mathbb{Z}_5, \lambda xy_{\bullet}x\bar{*}y)$	$(ReduceClosed\ ClComp\bar{*}\ L_4\ L_5\ L_6)$
$L_{10}.$	$\vdash closed(\mathbb{Z}_5, \lambda xy_{\bullet}(x\bar{*}y)\bar{+}\bar{3}_5)$	$(ReduceClosed\ ClComp\bar{+}\ L_7\ L_8\ L_9)$

The following are the theorems involved (stated informally)²:

1. Each residue class set RS_n is closed with respect to the operations: $\lambda xy_{\bullet}c$ if $c \in RS_n$ (corresponding to the theorem $ClosedConst$), $\lambda xy_{\bullet}x$ ($ClosedFV$), and $\lambda xy_{\bullet}y$ ($ClosedSV$).
2. Each complete residue class set \mathbb{Z}_n which is closed under the binary operations op_1 and op_2 , is also closed under the composed binary operation $\lambda xy_{\bullet}(x\ op_1\ y) \circ (x\ op_2\ y)$ where $\circ \in \{\bar{+}, \bar{-}, \bar{*}\}$ ($ClComp\bar{+}$, $ClComp\bar{-}$, $ClComp\bar{*}$).

While the theorems under (1) are applied by **Apply-Assertion**, the theorems under (2) need to be applied with **ReduceClosed**. This is due to the fact that when, for instance, applying the theorem

$$\forall n : \mathbb{Z}_{\bullet} \forall op_1. \forall op_2. (closed(\mathbb{Z}_n, op_1) \wedge closed(\mathbb{Z}_n, op_2)) \Rightarrow closed(\mathbb{Z}_n, \lambda x, y_{\bullet} (x\ op_1\ y)\bar{+}(x\ op_2\ y))$$

to line L_{10} the necessary instantiations for the operations have to be $op_1 = \lambda xy_{\bullet}x\bar{*}y$ and $op_2 = \lambda xy_{\bullet}\bar{3}_5$. If the theorems under (2) were treated in a general way we would need higher order matching to compute the instantiations and thus **Apply-Assertion** is not sufficient. Instead, we implemented a special decidable matching algorithm employed by the **ReduceClosed** method. The algorithm is suitable to handle the cases under (2) and when the method is applied the premises of the respective theorem involved will result in new open sub-goals.

Some other methods associated with the strategy **ReduceToSpecial** are **InInt** which closes goals of the form $n \in \mathbb{Z}$ if n is an integer and **InResclSet** closing goals of the form $c \in RS_n$ if c is an element of the residue class set RS_n .

We have also experimented with bookkeeping already solved problems and trying to reduce new problems to these. However, this approach is not feasible since for large sets of problems the comparison of a new problem with those already solved is rather expensive.

²Similarly, our database contains theorems suitable for associativity, unit element, inverses, and divisors problems.

5.4 Comparison of the three Strategies

Only the `TryAndError` strategy is applicable to all possible occurring problems. And, judging from the experiments, it also always successfully derives a proof, even when the hints provided by the computer algebra systems fail. The strategy `EquSolve` is just suitable for proving the properties associativity, the existence of a unit element, of inverse elements, and of divisors. It can neither be used to refute any of these properties nor to prove closure. Moreover, it sometimes even fails for problems it can be applied to when MAPLE returns facts useless in our context (e.g., a term involving a rational number). The `ReduceToSpecial` strategy can currently be applied to closure, associativity, unit element, inverses, and divisors problems. Up to now we have no theorems for negated problems in our database. The strategy is, however, quite limited since it can only succeed in those special cases that are covered by the given theorems. Therefore, MULTI has a strategic control rule that tries `ReduceToSpecial`, `EquSolve`, and `TryAndError` in this order.

From a performance point of view, `TryAndError` is generally the most time consuming of the three strategies and delivers the longest proof plans since it painstakingly wades through all possible cases. Both disadvantages increase the larger the involved residue class sets are. Contrary, both `EquSolve` and `ReduceToSpecial` are relatively fast and yield rather lean plans. Moreover, they are independent of the size of the involved sets. However, with a growing number of possible theorems to apply, the efficiency of `ReduceToSpecial` strategy might also decrease.

As far as generality of the involved strategies is concerned, one should note that both `TryAndError` and `EquSolve` possess only three domain specific methods: `forallResclass`, `congruence`, and `existsResclass`. All their other methods are domain independent and used also for proof planning in other domains. Except for `Apply-Assertion` all the methods of the strategy `ReduceToSpecial` are constructed for the strategy. Hence, only a few of these will be useful in other domains.

6 Exploring Properties

We are currently working with a module that enables us to automatically explore the single properties of residue classes. The motivation for this is that a user only has to supply a residue class set and an operation and the module will stepwise construct theorems to either prove or refute single properties. The final result stating the nature of the given algebra is then derived automatically. The module also aids the automatic exploration of

large testbeds as described in Sec. 7.

The input to the exploration module is a set and a corresponding operation. To classify the given algebra it employs the same functionality as described in Sec. 4, i.e., it uses the query function for the single properties. It step-by-step queries about properties, generates theorems according to the obtained results and passes them to MULTI to prove. Unfortunately, the results of the query functions cannot be used in the proof planning process since MULTI can only use information given in the form of methods, strategies, and control rules. However, this is not of great weight since the repeated computations are relatively inexpensive.

In detail the exploration works as follows: It first checks with the help of the multiplication table whether the operation is closed on the given set. If not, the appropriate theorem is generated, passed to MULTI and if proved successfully the exploration stops. If the set is indeed closed under the operation and the according theorem is proved by MULTI we have at least a magma and carry on with the exploration. The next exploration steps work similar, that is, the exploration module generates theorems to prove that a property either holds or that it does not hold.

After establishing that we have at least a magma, the next property checked is associativity to see if we have a semi-group. If this holds, the module checks if there exists a unit element, i.e., whether we have a monoid. Finally, if inverses exist we have a group. In case the check for associativity fails, the exploration module proceeds by checking the existence of divisors to see if we have a quasi-group. And if the given set together with the operation really forms a quasi-group the next check is again for the existence of a unit element, meaning we would have a loop. The exploration will then stop in any case since we already know that the given algebra does not form a group (since it is not associative).

7 Experiments

To design and implement the approach we describe in this paper and especially to extract the necessary methods and control rules we used a total of 21 examples. To show the appropriateness of the methods and control rules and to check the effectivity of the implemented strategies they need to be tested against a large number of examples that differ from those used during the design process. Moreover, considering the purpose of the implementation, i.e., its use in a tutor system where a user can pose virtually any residue class (at least up to a certain order) together with an arbitrary combination of operations as problem, it is essential to test the robustness

of the implementation with a wide range of examples.

We are currently working with a testbed of roughly 13 million automatically generated examples. These examples are constructed from the possible subsets of the residue classes modulo n where n is in between 2 and 10 together with operations that are systematically constructed from the basic operations. We tried to exclude repeating and trivial cases, e.g., sets with only one element or operations like $x - x$, as far as possible. From the testbed we have explored 4152 examples, mainly involving \mathbb{Z}_2 to \mathbb{Z}_6 and some of their subsets. As interesting cases we have encountered 661 magmas, 990 semi-groups, 186 quasi-groups, 22 monoids, and 261 groups so far. However, these figures do not indicate whether the classified algebraic entities are actually distinct since we do not check for isomorphism between the elements in the testbed yet. This is subject of future work. When proving the constructed proof obligations `MULTI` could successfully employ `ReduceToSpecial` to a sample of 15% and `EquSolve` to a different set accounting for another 15% of the examples. The remaining 70% of the examples could only be solved by the `TryAndError` strategy. However, these figures are not as disappointing as they seem at a first glance considering that nearly all proofs involving the closure property of non-complete residue class sets (i.e., sets such as $\mathbb{Z}_3 \setminus \{\bar{1}_3\}$) and the refutation of properties could only be solved with the `TryAndError` strategy.

Additionally, we are experimenting with a set of 100 non-theorems. Although most of these can be successfully detected as false, this generally takes a long time. We are currently testing ways to minimize this time. Furthermore, the methods we developed for proving the group properties of the residue classes proved general enough to tackle also commutativity of single operations and distributivity properties of two operations (only the hint system for the strategy `TryAndError` had to be slightly expanded). We tested this with 10 examples so far.

8 Related Work and Conclusion

We have reported on an experiment in exploring properties of residue classes over integers. Single properties are step-wise checked by an exploration module that generates the appropriate proof obligations. These proof obligations are passed to a multi-strategy proof planner to be proved. The proof planner can draw on different planning strategies in order to solve the problems. We employ the computer algebra systems `GAP` and `MAPLE` to guide and simplify both the exploration and the proof planning process. We gave some empirical evidence that the currently implemented methods

form a robust set of planning operators that suffices to explore the domain of residue classes.

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 application of the combined system 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]. More concrete work in exploration in finite algebra is reported in [3, 11, 15] where model generation techniques are used to tackle quasi-group existence problems. In particular, some open problems in quasi-group theory were solved.

Our work has neither the claim to advance the theoretical and architectural aspects of the integration of deduction and computer algebra systems nor do we expect to make any new discoveries. It rather concentrates on having a robust and flexible machinery for working with examples in the context of tutor systems. Moreover, it shows how a combination of various systems can be fruitfully employed to large classes of examples. Although one major ingredient in our setup is Ω MEGA's multi-strategy proof planner, the presented work does not reflect the possible power of proof planning in general, since the problem solutions are still quite algorithmic and the necessary search is rather limited.

Future work will include to extend the exploration module so we can classify residue classes with two associated operations in terms of rings, integral domains and fields. Moreover, we are currently also implementing appropriate methods and strategies to reason about isomorphism between the residue classes both to detect isomorphic structures among those already classified and to have examples for the interactive course in algebra for the concept of isomorphism.

References

- [1] P. Andrews. Transforming matings into natural deduction proofs. In *Proc. of CADE-5*, p. 281–292, 1980.
- [2] A. Bundy. The use of explicit plans to guide inductive proofs. In *Proc. of CADE-9*, volume 310 of *LNCS*, p. 111–120. Springer, 1988.
- [3] M. Fujita, J. Slaney, and F. Bennett. Automatic generation of some results in finite algebra. In *Proc. of IJCAI'93*, p. 52–57. Morgan Kaufmann, 1993.
- [4] The GAP Group, Aachen, St Andrews. *GAP - Groups, Algorithms, and Programming, Version 4*, 1998. <http://www-gap.dcs.st-and.ac.uk/~gap>.

- [5] G. Gentzen. Untersuchungen über das Logische Schließen I und II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.
- [6] The Ω MEGA Group. Ω Mega: Towards a Mathematical Assistant. In *Proc. of CADE-14*, volume 1249 of *LNAI*, p. 252–255. Springer, 1997.
- [7] P. Jackson. Exploring Abstract Algebra in Constructive Type Theory. In *Proc. of CADE-12*, volume 814 of *LNCS*, p. 590–604. 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. In [8], p. 327–355.
- [10] I. Kraan, D. Basin, and A. Bundy. Middle-out reasoning for logic program synthesis. Tech. Report MPI-I-93-214, MPI, Saarbrücken, Germany, 1993.
- [11] W. McCune. A Davis-Putnam program and its application to finite first-order model search: Quasigroup existence problems. Technical Memorandum ANL/MCS-TM-194, Argonne National Laboratory, USA, 1994.
- [12] E. Melis and A. Meier. Proof planning with multiple strategies. In *Proc. of the First International Conference on Computational Logic*, 2000. Springer.
- [13] E. Melis and J. Siekmann. Knowledge-based proof planning. *Artificial Intelligence*, 1999.
- [14] Darren Redfern. *The Maple Handbook: Maple V Release 5*. Springer, 1999.
- [15] J. Slaney, M. Fujita, and M. Stickel. Automated reasoning and exhaustive search: Quasigroup existence problems. *Computers and Mathematics with Applications*, 29:115–132, 1995.
- [16] Volker Sorge. Non-Trivial Computations in Proof Planning. In *Proc. of FroCoS 2000*, volume 1794 of *LNCS*. Springer, 2000.