Article Submitted to Journal of Symbolic Computation

Comparing Approaches to the Exploration of the Domain of Residue Classes

Andreas Meier[1], Martin Pollet*[1,2] and Volker Sorge†[2]

[1] *Fachbereich Informatik, Universität des Saarlandes, Germany,*
ameier|pollet@ags.uni-sb.de
http://www.ags.uni-sb.de/~ameier|pollet

[2] *School of Computer Science, University of Birmingham, UK,*
M.Pollet|V.Sorge@cs.bham.ac.uk http://www.cs.bham.ac.uk/~mxp|vxs

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 proof techniques, which are implemented as strategies in a multi-strategy proof planner. The search space of the proof planner can be drastically reduced by employing computations of two computer algebra systems during the planning process. To test the effectiveness of our approach we carried out a large number of experiments and also compared it with some alternative approaches. In particular, we experimented with substituting computer algebra by model generation and by proving theorems with a first order equational theorem prover instead of a proof planner.

1. Introduction

We report on a case study that combines proof planning with computer algebra systems (CAS). 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. That is, we examine structures such as $(\mathbb{Z}_3, \bar{+}), (\mathbb{Z}_5 \backslash \{\bar{0}_5\}, \bar{-}), (\{\bar{1}_6, \bar{3}_6, \bar{5}_6\}, \bar{*}), \ldots$ (here $\bar{1}_6$ denotes the congruence class 1 modulo 6) whether they are groups, monoids, semi-groups, etc. and which of them are isomorphic to each other.

*The author's work was supported in parts by the 'Landesgraduiertenförderung des Saarlandes' and by the Calculemus IHP-RTN EC project, contract code HPRN-CT-2000-00102.

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

The original motivation of our work was to provide the data for interactive algebra courses inside a tutor system using the ΩMEGA theorem proving environment (Benzmüller *et al.*, 1997). 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. Moreover, it is necessary to have proofs in human-oriented reasoning style using different proof techniques. This suggested employing multi-strategy proof planning as a tool for constructing proofs since it allows easy modelling of different human-oriented proof techniques by different strategies. Moreover, proof planning enables us to exploit the power of CASs in a sound way (Kerber *et al.*, 1998; Sorge, 2000) in order to guide and shorten the proof process.

The case study essentially consists of three parts: (1) The implementation of a set of proof planning strategies that realize different proof techniques for the residue class domain. Thereby we were interested in examining basic algebraic properties of given residue class structures in order to automatically classify them into terms of the algebraic structure they form. Furthermore, structures of the same type and cardinality are then classified into sets of isomorphic structures. The implemented proof planning strategies employ the computations of CASs to varying degrees to ease the planning process. (2) For testing the effectiveness of the implemented machinery we conducted a large number of experiments by automatically and systematically classifying residue class structures. Finally, (3) in order to verify the usefulness of the combination of proof planning and computer algebra we also compared our approach with alternative techniques. In particular, we experimented with substituting computer algebra by model generation and by proving theorems with a first order equational theorem prover instead of a proof planner. The former turned out to be quite effective and can fruitfully complement the use of computer algebra. The latter proved to be applicable for constructing most of the required proofs but is less robust in a large case study than our combined proof planning and computer algebra approach.

Part (1) and (2) of the case study were reported in (Meier and Sorge, 2001) and (Meier *et al.*, 2001), where the former was concerned with proofs of simple algebraic properties and the latter with the isomorphism proofs. For an extensive report on both, including a detailed presentation of the constructed proofs, we refer the reader to (Meier *et al.*, 2000). In this paper we present a summary of the overall case study including a detailed report of the results of part (3).

The paper is organized as follows: We first give a brief overview of multi-strategy proof planning in the ΩMEGA system and the integration of computer algebra with proof planning. Section 3 contains a summary of the exploration of the residue class domain with the combined power of proof planning and computer algebra. Section 4 and 5 report on the comparison of our original proof planning approach with alternative techniques. In particular the former is concerned with the substitution of computer algebra by model generation while the latter compares the proof planning approach with traditional automated theorem proving.

2. Proof Planning and Computer Algebra

In this section we give a brief introduction to multi-strategy proof planning and to the integration of computer algebra into proof planning. More detailed introductions can be found in (Melis and Meier, 2000) and (Kerber *et al.*, 1998; Sorge, 2000), respectively.

2.1. Multi-Strategy Proof Planning

Proof planning (Bundy, 1988) 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 plan 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 ΩMEGA system (Benzmüller *et al.*, 1997) the traditional proof planning approach is enriched by incorporating mathematical knowledge into the planning process (see (Melis and Siekmann, 1999) for details). That is, methods can encode general proving steps as well as knowledge particular to a mathematical domain. Moreover, *control rules* provide the possibility to introduce mathematical knowledge on how to proceed in the proof planning process by specifying how to traverse the search space. Depending on the mathematical domain or proof situation, they can influence the planners behavior at choice points (e.g., which goal to tackle next or which method to apply next). ΩMEGA's new proof planner, MULTI (Melis and Meier, 2000), also allows the specification of different planning strategies to control the overall planning behavior. Strategies implement proof techniques by specifying particular sets of methods and control rules. Thus they allow tackling the same problem in different ways. In case more than one strategy is applicable to one problem, MULTI can reason about which strategy to employ and also switch strategies during one proof attempt. In particular, the planner can backtrack from applied strategies and thus perform search on the level of strategies.

2.2. Employing Computer Algebra in Proof Planning

We employ symbolic calculations to guide and simplify the search for proof plans when proof planning in the domain of residue classes. In particular, we use the mainstream CAS MAPLE (Redfern, 1999) and GAP (GAP, 1998), a system specialized on group theory. In this paper we are not concerned with the technical side of the integration since we exploit previous work, in particular (Kerber *et al.*, 1998), that presents the integration of computer algebra into proof planning, and (Sorge, 2000), that exemplifies how the correctness of certain limited computations of a large-scale CAS such as MAPLE can be guaranteed within the proof planning framework. Instead we concentrate on the cooperation between the systems in the context of exploring residue class properties.

We use symbolic calculations in two ways: (1) To guide the proof planner and to prune the search space by computing hints with control rules. (2) To shorten and simplify the proofs by calling MAPLE within the application of a method to solve equations. As side-effect both cases can restrict possible instantiations of meta-variables[‡].

(1) is implemented, for instance, in the control rule `select-instance`. The rule is triggered after the 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 showing the existence of a unit element e in $(\mathbb{Z}_2, \bar{+})$, the control rule supplies a hint as to what e might be (i.e., $\bar{0}_2$). To obtain suitable hints, `select-instance` sends corresponding queries to GAP and MAPLE. If hints can be computed, the meta-variables are instantiated before the proof planning proceeds. However, the instantiations suggested by `select-instance` are treated as a hint by the proof planner; that is, they have to be verified during the subsequent proof planning process. In case the proving attempt fails for a particular instantiation, MULTI backtracks and tries to find an appropriate instantiation by crude search.

(2), the use of calculations, is realized within the `Solve-Equation` 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. Any meta-variables contained in the equation are considered as the variables the equation is to be solved for and they are supplied as additional arguments for `solve`. 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 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 CAS that provides detailed information on its computations in order to construct the expansion. This process is extensively described in (Sorge, 2000).

3. Proof Planning in the Residue Class Domain

In this section we introduce our concrete case study, the exploration of the residue class domain. We are concerned with classifying given structures (1) in terms of the algebraic category (group, monoid, etc.) they form, and (2) in classes of isomorphic structures. During both classification processes, proof obligations have to be discharged, which can be done with several proof planning strategies.

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

To test the effectiveness of our techniques we applied them to a large testbed of examples.

This section starts with a general introduction to the residue class domain and the problems we are dealing with. The sections 3.2 and 3.3 give a short overview of the proof planning strategies for proving simple algebraic properties and isomorphism or non-isomorphism problems. We shall only give a brief overview of each strategy and sometimes illustrate them with a small example. For a detailed account of the strategies we refer the reader to (Meier *et al.*, 2000). However, we shall point out the various uses of computer algebra in the proof planning and classification process. To conclude the section we give a summary of the conducted experiments and discuss their results.

3.1. Introduction to the Residue Class Domain

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 $\mathbb{Z}_n$. Concretely, we are dealing with sets of the form $\mathbb{Z}_3, \mathbb{Z}_5, \mathbb{Z}_3\backslash\{\bar{1}_3\}, \mathbb{Z}_5\backslash\{\bar{0}_5\}$, $\{\bar{1}_6, \bar{3}_6, \bar{5}_6\}, \ldots$ where $\bar{1}_6$ denotes the congruence class 1 modulo 6. 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., $\lambda xy.(x\bar{+}\bar{1}_3)\bar{-}(y\bar{+}\bar{2}_3)$. We often abbreviate the operations $\lambda xy.x\bar{+}y$, $\lambda xy.x\bar{*}y$, $\lambda xy.x\bar{-}y$ by $\bar{+}$, $\bar{*}$, $\bar{-}$, respectively.

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 to refute the following properties:

1. **Closure:** RS_n is closed under $\circ$.
2. **Associativity:** RS_n is associative with respect to $\circ$.
3. **Unit element:** There exists a unit element e with respect to $\circ$ in RS_n.
4. **Inverses:** Every element in RS_n has an inverse element with respect to $\circ$ and the unit element e.
5. **Divisors:** For every two elements $a, b \in RS_n$ there exist two corresponding divisors $x, y \in RS_n$ such that $a \circ x = b$ and $y \circ a = b$ holds.
6. **Abelian:** RS_n is commutative with respect to $\circ$.

For two given structures $(RS_n^1, \circ^1)$ and $(RS_m^2, \circ^2)$ we are also interested in whether they are isomorphic or not. Therefore, we show whether or not there exists a function $h{:}(RS_n^1, \circ^1) \rightarrow (RS_m^2, \circ^2)$ such that h is injective, surjective, and homomorphic.[§]

[§]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 $cl_i(x)$.

Properties, or their refutations, are formalized in ΩMEGA's higher order language. For instance the closure concept is formalized as $\forall x{:}RS_n.\forall y{:}RS_n.(x \circ y) \in RS_n$ and its refutation as $\exists x{:}RS_n.\exists y{:}RS_n.(x \circ y) \notin RS_n$. Here the variables x and y are of sort RS_n; that is, the quantifiers range over the finite domain RS_n.

3.2. Classification wrt. Basic Algebraic Properties

We are interested primarily in classifying residue class sets over the integers and 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) (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)[¶] and whether they are Abelian. To do so, the single properties are successively checked in two steps: First we compute whether or not a property is likely to hold for a given structure. Then an appropriate proof obligation is constructed and discharged by MULTI. If MULTI fails to prove the given problem it tries to prove the respective negation. Only after a valid proof plan is constructed is the next property tested. CASs are used in both testing and proving phase: for the computation of the likely answer and while discharging proof obligations to guide the proof planning process.

3.2.1. Testing Basic Properties

The first property we have to check is whether the given structure is actually closed under the operation. This is done by examining a multiplication table that is constructed in ΩMEGA. If the structure is closed the multiplication table is passed to GAP for testing the associativity, unit element, inverses and Abelian properties. That is, GAP is asked whether a property holds wrt. the given multiplication table. Since there is no appropriate command in GAP to check the divisors property, this test is again done in ΩMEGA using the original multiplication table.

3.2.2. Discharging Proof Obligations with MULTI

For discharging proof obligations we have implemented three different proof techniques with strategies in MULTI, which use symbolic computations to a varying degree.

TryAndError The simplest strategy is `TryAndError`, which performs a naïve *exhaustive case analysis*. This is possible since we are in a finite domain and can always enumerate all possible cases. The case analysis is conducted with respect to the quantification of the problem at hand. For universally quantified formulas where the variable ranges over the residue class set, a case split on all elements of the set is performed. Then the resulting property is proved for every single element separately. For existentially quantified goals all possible

[¶]Naturally property 5 holds for a group as well.

instantiations for the quantified variable are successively checked. At this point we can prune the search with hints from a CAS.

Instead of blindly testing all possible instantiations for the existentially quantified variable and backtracking if necessary, MULTI inserts a meta-variable whose instantiation can be suggested with a hint from the `select-instance` control rule (see section 2.2). The possible instantiations are computed by a hint system in ΩMEGA, which employs some routines of ΩMEGA as well as the CASs MAPLE and GAP. For instance, GAP is employed to compute the unit element and inverses for single elements while an ΩMEGA routine computes possible divisors. The hint system also provides counter examples if a property has to be refuted. It can compute, for example, a pair of elements for which a given structure is not closed. In case a counter example for associativity is needed, MAPLE is used 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. The procedure for commutativity is similar. The hint system is able to provide hints for all existentially quantified variables occurring in proofs of basic properties of residue classes.

As an example, consider the proof that $(\mathbb{Z}_2, \bar{+})$ has inverses with respect to the unit element $\bar{0}_2$: $\forall x{:}\mathbb{Z}_2 . \exists y{:}\mathbb{Z}_2 . (x \bar{+} y = \bar{0}_2) \wedge (y \bar{+} x = \bar{0}_2)$. First, the universally quantified goal is reduced to the two goals $\exists y{:}\mathbb{Z}_2 . (\bar{0}_2 \bar{+} y = \bar{0}_2) \wedge (y \bar{+} \bar{0}_2 = \bar{0}_2)$ and $\exists y{:}\mathbb{Z}_2 . (\bar{1}_2 \bar{+} y = \bar{0}_2) \wedge (y \bar{+} \bar{1}_2 = \bar{0}_2)$ corresponding to a case split over the range of the universally quantified variable $x \in \mathbb{Z}_2$; that is, $x = \bar{0}_2$ or $x = \bar{1}_2$. For both goals the correct instantiation of the variable y ($\bar{0}_2$ and $\bar{1}_2$, respectively) is then computed with GAP.

EquSolve This strategy employs as much as possible *equational reasoning*. Instead of checking the validity of the statements for all possible cases like `TryAndError`, `EquSolve` tries to solve occurring equations in a general way with the `Solve-Equation` method. Universally quantified variables are replaced by constants, while existentially quantified variables are replaced by meta-variables. Via `Solve-Equation` the strategy then employs MAPLE to check the universal validity of the equation. In case the equation contains meta-variables, MAPLE tries to compute an appropriate instantiation, such that the equation is universally valid. The meta-variables are subsequently instantiated in the proof.

Considering again the proof of the inverse property of $(\mathbb{Z}_2, \bar{+})$, we obtain the equations $cl_2(c) \bar{+} cl_2(mv) = \bar{0}_2$ and $cl_2(mv) \bar{+} cl_2(c) = \bar{0}_2$, where c is a constant and mv is a meta-variable. When applied via `Solve-Equation`, MAPLE's algorithm `msolve` returns a general solution for mv, namely $mv = c$. Hence the equations can be closed by `Solve-Equation`.

ReduceToSpecial This last strategy tries to tackle new problems by applying already known theorems from ΩMEGA's knowledge-base. The application of a known theorem can either prove a goal directly or can reduce the goal to

subgoals that can then be tackled by other theorems. The `ReduceToSpecial` strategy does not depend on the help of a CAS.

3.3. Identifying Classes of Isomorphic Structures

The second part of our case study is concerned with checking isomorphisms between two given residue class structures. Unlike the proof techniques for simple algebraic properties presented in the preceding section, which were rather straightforward, to prove whether two residue class structures are isomorphic or not is more complicated. Although we can mainly reuse the strategies designed for the simple properties they have to be partially interleaved. Again the check whether two structures are isomorphic or not is done in two steps. First we perform a test computation that gives the likely answer. Then the corresponding proof obligation is constructed and passed to MULTI. Again in both phases CASs are used.

In the following we are concerned to show isomorphism or non-isomorphism only for two structures with the same cardinality. The case where two structures of different cardinality are involved can be trivially proved by MULTI with the application of a theorem stating that finite sets of different size are trivially not isomorphic.

3.3.1. Testing for Isomorphisms

We test whether two given structures $(RS^1_n, \circ^1)$ and $(RS^2_m, \circ^2)$ are isomorphic by computing a possible isomorphism mapping $h{:}(RS^1_n, \circ^1) \to (RS^2_m, \circ^2)$ with MAPLE. For the function h a system of equations is generated by instantiating the homomorphism equation $h(x \circ^1 y) = h(x) \circ^2 h(y)$ with all elements of the residue class set RS^1_n. This results in a set of equations $h(cl_n(i) \circ^1 cl_n(j)) = h(cl_n(i)) \circ^2 h(cl_n(j))$ for all $cl_n(i), cl_n(j) \in RS^1_n$. 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 $h(cl_n(k)) = h(cl_n(i)) \circ^2 h(cl_n(j))$. In the remainder of the paper we call the resulting system of equations the *instantiated homomorphism equations*. Now, MAPLE is asked to give a solution for the corresponding system of equations $x_k = x_i \circ^2 x_j$ (where $h(cl_n(l))$ becomes the variable x_l) with respect to the modulo factor m using MAPLE's function `msolve`. If MAPLE returns a set of solutions and we can find one solution containing only elements from the integer set corresponding to RS^2_m with $x_i \neq x_j$ for all $i \neq j$, we have a candidate for the isomorphism mapping.

For example, to test whether $(\mathbb{Z}_2, \bar{+})$ and $(\mathbb{Z}_2, \lambda xy.x\bar{+}y\bar{+}\bar{1}_2)$ are isomorphic we generate the corresponding set of instantiated homomorphism equations:

$h(\bar{0}_2) = h(\bar{0}_2\bar{+}\bar{0}_2) = h(\bar{0}_2)\bar{+}h(\bar{0}_2)\bar{+}\bar{1}_2, h(\bar{1}_2) = h(\bar{0}_2\bar{+}\bar{1}_2) = h(\bar{0}_2)\bar{+}h(\bar{1}_2)\bar{+}\bar{1}_2,$
$h(\bar{1}_2) = h(\bar{1}_2\bar{+}\bar{0}_2) = h(\bar{1}_2)\bar{+}h(\bar{0}_2)\bar{+}\bar{1}_2, h(\bar{0}_2) = h(\bar{1}_2\bar{+}\bar{1}_2) = h(\bar{1}_2)\bar{+}h(\bar{1}_2)\bar{+}\bar{1}_2.$

After replacing $h(\bar{0}_2)$ by x_0 and $h(\bar{1}_2)$ by x_1 we ask MAPLE to give a solution for the equations: $x_0 = x_0 + x_0 + 1, x_1 = x_0 + x_1 + 1, x_1 = x_1 + x_0 + 1, x_0 = x_1 + x_1 + 1$ wrt. the modulo factor 2. MAPLE's answer is: $\{x_0 = 1, x_1 = x_1\}$, that is $x_0 = 1$ and x_1 can be both 0 or 1. Thus, one bijective function is $h(\bar{0}_2) = \bar{1}_2$, $h(\bar{1}_2) = \bar{0}_2$.

3.3.2. Isomorphism Proofs with MULTI

In this section we present how MULTI plans isomorphism proofs. It employs the same three strategies introduced in section 3.2.2, namely `TryAndError`, `EquSolve`, and `ReduceToSpecial`. 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.

TryAndError When constructing an isomorphism proof `TryAndError` has to search for a bijective homomorphism h among all existing mappings between the two residue class structures involved. The mapping h is represented as a pointwise defined function, where the image of each element of the domain is explicitly specified as an element of the codomain. The search can be abbreviated by computing a pointwise isomorphism as a hint with MAPLE using the technique described above.

As an example consider the proof that $(\mathbb{Z}_2, \bar{+})$ and $(\mathbb{Z}_2, \lambda xy.x\bar{+}y\bar{+}\bar{1}_2)$ are isomorphic. There exist 4 possible pointwise functions $h : \mathbb{Z}_2 \longrightarrow \mathbb{Z}_2$. The hint computed with MAPLE is the function given in section 3.3.1, namely $h(\bar{0}_2) = \bar{1}_2, h(\bar{1}_2) = \bar{0}_2$. The subsequent subproofs for the properties injectivity, surjectivity, and homomorphy of the pointwise defined function are then performed in the regular fashion of the `TryAndError` strategy as already discussed in section 3.2.2.

Each of the subproofs has the complexity n^2 where n is the cardinality of the structures involved.[‖] 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 Isomorphism proofs can often be simplified by computing a polynomial that interpolates the pointwise defined isomorphic mapping. If an interpolation polynomial can be computed it is introduced into the proof instead of the pointwise mapping. Then the `EquSolve` strategy has a chance to find the subproofs for surjectivity and the homomorphism property by reducing these subproblems to equations which might be solvable with the `Solve-Equation` method. However, 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 from a given pointwise function we again employ MAPLE.

For our example problem that $(\mathbb{Z}_2, \bar{+})$ is isomorphic to $(\mathbb{Z}_2, \lambda xy.x\bar{+}y\bar{+}\bar{1}_2)$ the corresponding pointwise isomorphism mapping is $h(\bar{0}_2) = \bar{1}_2, h(\bar{1}_2) = \bar{0}_2$. MAPLE computes the interpolation polynomial $x \rightarrow (x + 1 \ mod \ 2)$ which is introduced into the proof. The properties of injectivity, homomorphy, and surjectivity are then shown for the polynomial. The subproofs of the latter two properties can indeed be shown with the `EquSolve` strategy. Since the proof for injectivity

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

cannot be constructed with `EquSolve`, MULTI switches either to the strategy `ReduceToSpecial` or `TryAndError` to prove this property.

ReduceToSpecial Like the proofs of simple algebraic properties the strategy `ReduceToSpecial` can be applied to the overall problem directly. Moreover, it can also be applied during the proof of one of the injectivity, surjectivity, or homomorphy subgoals. In particular, it is used to exploit the simple mathematical fact that in finite domains surjectivity implies injectivity and vice versa. Usually MULTI proves first the surjectivity subgoal; then `ReduceToSpecial` shows the injectivity subgoal by applying the following theorem: *A surjective mapping between two finite sets with the same cardinality is injective.*

3.3.3. 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 again use the strategies `ReduceToSpecial` and `TryAndError`. Additionally, we have implemented another strategy, `NotInjNotIso`, which is specialized on proving only non-isomorphism problems. It constructs indirect proofs by showing that no homomorphic mapping between the two given residue class structures can be injective.

TryAndError Proving that two structures are not isomorphic to each other means we have to show that all possible mappings between the structures are not isomorphic. The `TryAndError` strategy performs an exhaustive case split by introducing all possible instantiations for the mapping between the two structures involved, and proves for each that it is either not injective, not surjective, or not a homomorphism. For non-isomorphism problems the strategy does not employ help from a CAS. The application of this naïve approach 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.

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. ΩMEGA's knowledge-base contains some theorems about such properties that can be applied within the strategy `ReduceToSpecial`. For instance, we have one theorem stating that, if for two structures $(S^1, \circ^1)$ and $(S^2, \circ^2)$ there exists an element $a \in S^1$ with order n such that the order of every element of S^2 is different from n, then the two structures are not isomorphic. The `ReduceToSpecial` strategy can apply this theorem to reduce non-isomorphism goals and then the `TryAndError` strategy takes over to complete the proof by showing the resulting subgoals. During these proofs the `TryAndError` strategy obtains hints by computing the orders of elements with GAP. For structures

without a unit element there exists a similar theorem about the order of traces of elements. The traces and their order are then likewise computed with GAP.

In contrast to employing `TryAndError` alone, proofs constructed with the combination of `TryAndError` and `ReduceToSpecial` have only polynomial complexity in the cardinality of the sets involved. Moreover, the search is reduced significantly by providing hints. But this technique is only applicable when the given structures contain elements suitable for our purpose in the sense that either the order of an element or the order of the trace of an element is not reflected in the other structure.

NotInjNotIso The strategy `NotInjNotIso` was implemented particularly for non-isomorphism proofs. The idea is to show that for two structures $(S^1, \circ^1)$ and $(S^2, \circ^2)$ there exist two distinct elements in S^1 that are always mapped to the same element in S^2 under all possible homomorphisms. `NotInjNotIso` constructs an indirect proof by first assuming that there exists a function $h{:}S^1 \to S^2$ which is an isomorphism. Then h is an injective homomorphism and the set of instantiated homomorphism equations $h(x_1 \circ^1 x_2) = h(x_1) \circ^2 h(x_2)$ is introduced into the proof. The strategy then tries to find two elements $c_1, c_2 \in S^1$ with $c_1 \neq c_2$ such that the equation $h(c_1) = h(c_2)$ can be derived. 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.

The search for appropriate c_1 and c_2 can be restricted if the hint system can provide a suitable instantiation. To obtain these instantiations all solutions of the set of instantiated homomorphism equations are computed with MAPLE. Then the set of solutions is examined to see whether there is a pair c_1 and c_2 with $c_1 \neq c_2$, such that $h(c_1) = h(c_2)$ holds in all solutions. If there is such a pair it is provided as a hint. The proof is also shortened by applying the `Solve-Equation` method, and thus MAPLE, to newly derived equations to test their validity.

As example, we prove that the non-Abelian quasi-groups $(\mathbb{Z}_5, \lambda xy.(\bar{2}_5 \bar{*} x) \bar{+} y)$ and $(\mathbb{Z}_5, \bar{-})$ are not isomorphic. Among the set of instantiated homomorphism equations are the two equations (1) $h(\bar{3}_5) = h(\bar{0}_5) \bar{-} h(\bar{3}_5)$, resulting from the homomorphism equation with $x_1 = \bar{0}_5$ and $x_2 = \bar{3}_5$, and (2) $h(\bar{3}_5) = h(\bar{1}_5) \bar{-} h(\bar{1}_5)$, resulting from $x_1 = \bar{1}_5$ and $x_2 = \bar{1}_5$. A suitable choice for c_1 and c_2 is $c_1 = \bar{0}_5$ and $c_2 = \bar{3}_5$ leading to `NotInjNotIso` trying to prove that $h(\bar{0}_5) = h(\bar{3}_5)$. Applying equation (1) to the right hand side yields $h(\bar{0}_5) = h(\bar{0}_5) \bar{-} h(\bar{3}_5)$, which can be further transformed to $h(\bar{0}_5) = h(\bar{0}_5) \bar{-} (h(\bar{1}_5) \bar{-} h(\bar{1}_5))$ applying equation (2). That this equation holds in general can be shown with MAPLE.

`NotInjNotIso` can produce very short proofs even for structures with large sets. However, constructing an appropriate sequence of equality substitutions is the hard part of the proof. 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. To overcome this dilemma we have successfully experimented

with randomization and restart techniques (Meier, 2000) known from AI (Gomes *et al.*, 1998).

3.4. Discharging Proof Obligations Automatically

The described strategies can be chosen independently from each other, depending on what kind of proof is desired. However, when discharging proof obligations automatically the strategies are attempted in a predetermined order. The idea is to try the generally most efficient strategies first and end with the most reliable one. Moreover, not all strategies are equally applicable to all problems. While the `TryAndError` strategy is applicable to all possible occurring problems, `EquSolve` can be applied only to those problems that can be reduced to equations. In particular, it cannot be applied to refute properties or to show closure. Moreover, it sometimes fails even for equational problems when MAPLE returns facts useless in our context (e.g., a term involving a rational number). Although the `ReduceToSpecial` strategy can theoretically be applied to all problems it is limited to those special cases that are covered by given theorems. Finally, the `NotInjNotIso` strategy is, of course, only applicable to non-isomorphism problems.

In detail, the order in which strategies are applied for proofs of simple properties is: `ReduceToSpecial`, `EquSolve`, and finally `TryAndError`. For isomorphism proofs `ReduceToSpecial` is tried first. If it fails MULTI attempts to prove the surjectivity and homomorphy subproblems, if possible, with equational reasoning. If this fails it uses `TryAndError` and, only after these two subproblems are proved, MULTI finishes the proof with the `ReduceToSpecial` strategy by deducing injectivity from surjectivity. When automatically discharging non-isomorphism proof obligations, MULTI tries `ReduceToSpecial` as first strategy. If it fails, the `NotInjNotIso` strategy is preferred to `TryAndError`.

3.5. Experiments

We needed 21 examples to construct the strategies `TryAndError`, `EquSolve`, and `ReduceToSpecial` for proving simple properties of residue class structures as presented in section 3.2. We used 15 examples to develop the extensions of these strategies to handle isomorphism and non-isomorphism proofs and another 4 examples to build the `NotInjNotIso` strategy.

To show the appropriateness of the constructed strategies 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. Altogether, we have classified 18963 structures with respect to their algebraic properties so far, including a large set of structures concerning the sets $\mathbb{Z}_5$, $\mathbb{Z}_6$, and $\mathbb{Z}_{10}$. The results for all explorations as well as for each of $\mathbb{Z}_5$, $\mathbb{Z}_6$, and $\mathbb{Z}_{10}$ are given on the left hand side of table 1. The figures give the number of structures we have found in each algebraic category, omitting those for which we have not found any

representative (i.e., loops, non-Abelian monoids and groups). Note that the total number of explored structures also includes some that were not closed, which are not displayed as a separate category.

	Simple Properties				Iso-Classes		
	All	$\mathbb{Z}_5$	$\mathbb{Z}_6$	$\mathbb{Z}_{10}$	$\mathbb{Z}_5$	$\mathbb{Z}_6$	$\mathbb{Z}_{10}$
Magmas	8567	3049	4152	743	36	7	14
Abelian magmas	244	53	73	24	26	5	6
Semi-groups	2102	161	1114	35	3	8	1
Abelian semi-groups	2100	592	1025	62	1	12	2
Quasi-groups	1891	971	738	70	9	2	10
Abelian Quasi-groups	536	207	257	11	3	2	1
Abelian Monoids	211	97	50	6	1	1	1
Abelian Groups	1001	276	419	49	1	1	1
Total	18963	5406	8128	1000	80	38	36

Table 1: Results of the experiments.

To show the validity of the techniques for isomorphism and non-isomorphism proofs we applied our classification process to the structures involving $\mathbb{Z}_5$, $\mathbb{Z}_6$, and $\mathbb{Z}_{10}$. Thereby we only classified those structures belonging to the same algebraic category; that is, we *a priori* excluded the comparison, for instance, of magmas and semi-groups. The different isomorphism classes we have found so far for the structures of each category are given on the right hand side of table 1.

For the simple properties, MULTI could successfully employ `ReduceToSpecial` to a sample of 20%, `EquSolve` for 23% of the proofs, and the remaining 57% of the examples could only be solved by the `TryAndError` strategy. However, these figures are not as disappointing as they seem at first glance considering that nearly all proofs involving the closure property of non-complete residue class sets (i.e., sets such as $\mathbb{Z}_3\backslash\{\bar{1}_3\}$) and the refutation of properties could only be solved with the `TryAndError` strategy. From the necessary isomorphism proofs 88% were done with the `EquSolve` strategy, the other 12% were done with `TryAndError`. During the automatic classification 1276 non-isomorphism proofs were constructed. Here 18% of the proofs were done with `ReduceToSpecial`; the remaining 82% with the `NotInjNotIso` strategy.

3.6. Discussion

From a performance point of view the strategies `EquSolve` and `ReduceToSpecial` do not depend on the cardinality of the sets involved. For instance, `EquSolve` proves the associativity of $(\mathbb{Z}_5, \bar{+})$ and $(\mathbb{Z}_{10}, \bar{+})$ in the same number of steps. The performance of `NotInjNotIso` on the other hand depends at least indirectly on the cardinality of the structures. While the size of the homomorphism equation system grows quadratically in the cardinality of the sets involved, which complicates search, the number of needed equational reasoning steps does not necessarily grow with the number of homomorphism equations.

Generally, neither the modulo factor nor the cardinality of the structures have a direct influence on the search depth for the problem solution. For instance, the problem $(\mathbb{Z}_n, \lambda xy.x\bar{*}y) \not\sim (\mathbb{Z}_n, \lambda xy.x\bar{-}y)$ can always be solved in two steps independent of the modulo factor n. For a detailed description see (Meier, 2000). The `TryAndError` strategy, however, depends directly on the cardinality of the structures, since the number of cases that have to be considered is in direct correspondence to the number of elements of a structure. Although for universal variables all possible cases have to be painstakingly worked out, for existential variables the search for the right instantiation can be significantly reduced by computing the correct hint immediately.

Although from a theoretical point of view all proofs can be done without the hint system by crude search, in practice the combinatorial explosion makes this infeasible. Thus, a reliable and robust hint system is crucial for the success. Indeed we have not found a single case where the hint system (i.e., mainly the computations of GAP and MAPLE) has failed or was incorrect for the proofs of simple properties. However, the situation is somewhat different for the isomorphism problems. The classification process as well as the hint system for the `TryAndError` and `EquSolve` strategy for isomorphism problems and for the `NotInjNotIso` strategy for non-isomorphism problems depend on the quality of MAPLE's solutions for the system of instantiated homomorphism equations. It turned out that MAPLE sometimes does not return all possible solutions even though it was required to do so. For instance, the two structures $(\mathbb{Z}_6, \lambda xy.\bar{2}_6\bar{*}x\bar{*}y)$ and $(\mathbb{Z}_6, \lambda xy.\bar{4}_6\bar{*}x\bar{*}y)$ are isomorphic (a possible isomorphism is $h(x) = \bar{5}_6\bar{*}x$). However, when called to give the solutions for the corresponding set of instantiated homomorphism equations MAPLE returns the mapping $h(x) = \bar{0}_6$ as sole solution. Although this is a correct solution, it is not the only one. In particular, it is not suitable to construct an isomorphism necessary for both testing in the classification process and providing a hint to the planner. Similarly, for non-isomorphism problems a lack of solutions can lead to a faulty hint for the `NotInjNotIso` strategy.

In our experiments we failed to classify about 200 structures due to this problem. Unfortunately, we could not find a clear characterization of these cases in order to work around the problem.

4. CAS vs. Model Generation

One possible solution to the problem of incomplete solutions for isomorphism hints described in the preceding section is to exchange the CAS by a model generator. Since all queries of the hint system are expressible as model generation problems, we were also interested in the performance and scalability of a generic model generator compared to a specialized CAS like GAP. For our experiments we chose the model generator SEM (Zhang and Zhang, 1996).

4.1. Model Generators

We employ model generation in proof planning to compute witness terms or hints for the planner by constructing appropriate models that contain the required information. In this scenario a model generator is applied similarly to a computer algebra system via control rules to guide the proof planning process. Thus, we essentially replace GAP and MAPLE in the hint system with the model generator SEM (Zhang and Zhang, 1996).

For the exploration of the residue class domain model generation can be used both to test whether a particular property is likely to hold or not to hold, and to compute all necessary hints to guide the proof planning. For instance, SEM can return either a unit element if one exists or a set of element pairs that suffice to show that no unit element exists in a given residue class structure.

The actual call to SEM consists of the multiplication table of the operation of the residue class structure together with the problem at hand. The multiplication table for n elements is encoded as a set of n^2 equations of the form $a \circ b = c$. To obtain, for example, a unit element SEM is asked to compute a model for the equations $x * e = x$ and $e * x = x$, where x is a free variable and e is an unspecified constant function for which a model is computed.

Unlike computer algebra, model generation can be applied to directly compute a hint both for showing and for refuting a property. In particular in the latter case, where the computation of the CAS simply fails and a hint has to be explicitly computed in a post-processing step, SEM is able to directly provide the result.

For example, the associativity property is tested with SEM by generating a model that contains the multiplication table and additionally the equation $(x * y) * z = x * (y * z)$ with variables x, y and z. If there exists a model, then associativity holds for all elements of the residue class structure. If there is no model, then SEM is called again with the negated equation where x, y and z are unspecified constants. A model for the negated equation contains an instantiation for x, y, z. With the standard hint system, associativity is first checked with GAP. If the answer is negative, MAPLE is used to provide the solutions for the associativity equation. The output of MAPLE has to be analysed, whether it is a non-general solution and which elements are a possible counter example.

4.2. Experiments

We compared the performance of the proof planner's hint system using the CASs MAPLE and GAP on one side and the model generator SEM on the other side in a series of additional experiments. We tested both approaches by computing the hints for the classification of 2000 residue class structures wrt. their algebraic property and whether two structures are isomorphic (1000 with $\mathbb{Z}_6$ and 1000 with $\mathbb{Z}_{10}$).

Table 2 gives a runtime comparison of our approaches. All experiments were conducted on a Sun Sparc Ultra with four processors and 2 GB Ram using

MAPLE 7 and SEM 1.7. The figures are average values extracted during the classification of the structures of our testbed and include the runtime of the external systems, communication time, and pre- and post-processing of the output. Since the behavior of the external systems and the processing of their output depends on the result of the query, we distinguish between positive and negative results. We omitted the tests for the closure and divisors properties on the CAS side since there we only use algorithms implemented in ΩMEGA.

Although the difference between the average runtimes of the two approaches is not large there are remarkable variances even within a single category when we take the gross runtimes depending on system load, network traffic, other processes, etc. There it can happen that a query takes several hundred milliseconds even when its average time is below 100 milliseconds and it becomes unpredictable whether SEM or the CASs will be faster. We tried to eliminate these effects from our statistics with a large testbed and three test runs.

We tested the scalability of the model generator approach performing 20 tests for generating hints (without planning the actual proofs) for residue class structures containing up to 100 elements. Even for the most expensive hints SEM was able to return an answer within less than 20 seconds.

Property	$\mathbb{Z}_6$ CAS		$\mathbb{Z}_6$ SEM		$\mathbb{Z}_{10}$ CAS		$\mathbb{Z}_{10}$ SEM	
	true	false	true	false	true	false	true	false
Closure	–	–	63	68	–	–	73	84
Associativity	23	455	67	141	31	430	82	163
Commutativity	37	25	68	129	21	31	79	147
Unit element	20	20	65	135	26	21	93	148
Inverses	63	60	143	200	116	225	168	227
Divisors	–	–	80	129	–	–	108	155
Isomorphism	597	766	73	216	827	1084	103	252

Table 2: Average runtime comparison of the hint system; times are given in milliseconds.

.

4.3. Discussion

In general, both approaches are equally robust and do not outperform each other. In fact, the approaches complement each other in the following sense: (1) There are no appropriate commands in MAPLE or GAP to check for the closure and the divisors properties. For these problems SEM is a new alternative to our algorithms implemented in ΩMEGA. (2) The comparison shows that SEM has better runtimes for isomorphism properties, whereas CASs are faster for most of the other properties. For associativity there is the interesting case that the runtime depends on the result and it cannot be said in advance which system should be preferred. (3) As discussed in section 3.6, MAPLE sometimes does not return all possible solutions for the homomorphism equation system. Thus, the hint system can either not construct an isomorphism mapping or, in the case of

non-isomorphism proofs, it fails to provide a suitable hint for the `NotInjNotIso` strategy. Actually, during our comparison, MAPLE failed to compute all solutions and hence to give suitable hints for 1% of the queries in cases where the structures were isomorphic and 20% of the queries, where the structures were not isomorphic. In contrast, SEM never failed to provide suitable and correct hints during our experiments. However, SEM cannot produce closed polynomial representations of isomorphisms as needed to apply the `EquSolve` strategy. But the pointwise isomorphisms provided by SEM can be passed to MAPLE to create a corresponding polynomial representation (as described in section 3.3.2).

Because of these complementary effects between SEM and the CASs we intend to implement a concurrent use of both. That is, all queries are send to both SEM and the CASs which run concurrently to competitively provide answers.

5. Proof Planning vs. ATP

The successful application of proof planning to the problems of a certain domain depends on the acquisition of mathematical knowledge of the domain and its formalization in methods, control rules, and strategies. If suitable knowledge is available, proof planning can solve problems that are beyond the means of traditional ATPs based on general-purpose machine-oriented logical calculi such as the resolution calculus (Robinson, 1965). Proof planning has been particularly successful in areas where proofs of similar structure have to be constructed, such as for example inductive theorem proving (Bundy, 1988). If the number of problems of a domain is sufficiently large the acquisition of the knowledge and its formalization can prove fruitful but is nevertheless a tedious task.

This generally poses the question of whether there are other means than proof planning to tackle the problems of a certain domain. The problems generated during the exploration of residue class structures are obviously in the range of more conventional automated theorem proving since all occurring quantifiers range over finite sets. To compare the results of our approach with the results of a traditional ATP we replaced the overall proof planning approach in the exploration scenario by the first order equational prover WALDMEISTER (Hillenbrand *et al.*, 1999). The decision to use WALDMEISTER was motivated by experiments with some general first order logic ATPs. They showed that without expert knowledge about suitable control settings for the systems and suitable formalizations of the problems we were hardly able to solve any of our problems. However, for WALDMEISTER we got help from one of its implementors in tuning the system for our problems.

5.1. Proving Residue Class Problems with WALDMEISTER

We employ WALDMEISTER in MULTI by specifying a strategy, `WMonResclass`, whose sole purpose is to give a goal exclusively to WALDMEISTER. The strategy can be applied to all problems occurring during the automatic exploration except

to show that two structures are isomorphic. Problem specifications for WaldMeister consist of three parts: A general axiomatization of the residue class structure and the operations $+, -, *$, a specific formalization of the property to be proved, and a suitable control setting for WaldMeister, for instance, an ordering of the used symbols. Note that the strategy `WMonResclass` calls WaldMeister with two different control settings depending on whether the goal to be proved is a simple property or a non-isomorphism problem.

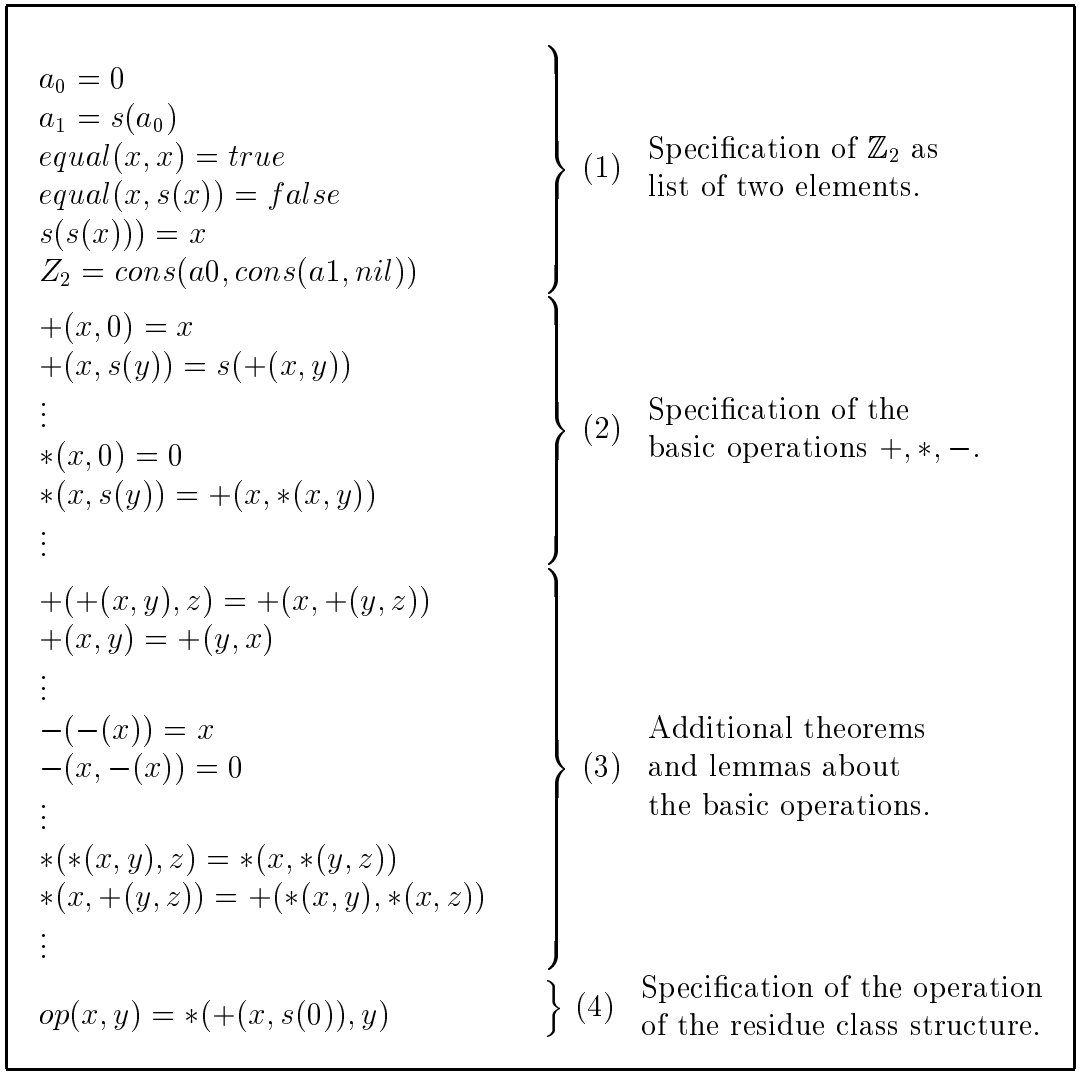

Figure 1: Specification for WaldMeister.

Figure 1 gives the general part of the input specification for the example $(\mathbb{Z}_2, \lambda x y_{\blacksquare}(x \bar{+} \bar{1}_2) \bar{*} y)$. The general part consists of facts that (1) model the given residue class set as a list of distinct elements, (2) model the basic operations $+, -, *$, and (3) add useful known lemmas and theorems about the basic oper-

ations such as the ring properties**. The given operation on residue classes can then be expressed directly by these functions (4); that is, the actual multiplication table of the structure does not need to be formalized. We experimented also with a direct specification of the multiplication table of the structures, similar to the problem specifications for SEM. However, WALDMEISTER performed better when the operation of the residue class structure was defined as composition of basic operations. The reason is that WALDMEISTER needs to be provided with suitable lemmas it can apply during its proof search and therefore the general knowledge on the basic operations given in part (3) is crucial for success. However, if the operation is specified via its multiplication table WALDMEISTER fails due to a lack of suitable lemmas.

To prove simple properties, we have to define the property in question recursively over the list specifying the given set. This can only be done by introducing several auxiliary predicates. The conclusion is then an equation stating that the simple property does or does not hold.

To show that two structures are not isomorphic we use WALDMEISTER to construct an indirect proof. That is, to the specification of the two structures $(RS^1_n, \circ^1)$ and $(RS^2_m, \circ^2)$, the definition of two homomorphisms $h : RS^1_n \to RS^2_m$ and $j : RS^2_m \to RS^1_n$ and the properties $h(j(x)) = x$ and $j(h(x)) = x$ are added. The conclusion consists then of all possible equations between two distinct elements of RS^2_m, such as 0=s(0), etc. If WALDMEISTER succeeds to prove that one of these equations holds then we have a contradiction to the assumption that the two structures are isomorphic. We currently have no suitable formalization to prove that two structures are isomorphic. One possibility would be to formalize a list of all possible functions (defined as pointwise functions) such that the isomorphism problem can be formalized via a recursion about this list. However, this is a very long-winded approach which we did not realize so far.

5.2. Experiments

To compare the combined proof planning/CAS approach with the application of ATP we used WALDMEISTER to explore structures with the sets $\mathbb{Z}_5$ and $\mathbb{Z}_{10}$ which we already classified wrt. their simple algebraic properties in our experiments reported in section 3.5. Moreover, we tackled non-isomorphism problems with the sets $\mathbb{Z}_5$ and $\mathbb{Z}_{10}$. We omitted isomorphism problems since we currently do not have a sensible approach to tackle this kind of problems with WALDMEISTER. The results of our experiments are summarized in Table 3. All experiments were done on a Sun Sparc Ultra with four processors and 2 GB Ram; the maximum time bound for WALDMEISTER was 1500 seconds.

Our experiments show that WALDMEISTER is generally able to solve all considered problems in the residue class domain. However, it turned out that

**In the specifications for WALDMEISTER $-$ is a unary function. Thus our binary minus operation is translated as $+(x, -(y))$.

	$\mathbb{Z}_5$	$\mathbb{Z}_{10}$
Explorations wrt. to simple prop.	1100	316
Failed Explorations	49	247
Single simple property problems	4694	1260
Failed simple properties problems	53	314
Non-isomorphism problems	2400	400
Failed non-isomorphism problems	167	65

Table 3: Results of applying WALDMEISTER to problems of $\mathbb{Z}_5$ and $\mathbb{Z}_{10}$.

on a large testbed WALDMEISTER is less robust than our proof planning approach. Indeed, WALDMEISTER failed on 4% of the $\mathbb{Z}_5$ and 78% of the $\mathbb{Z}_{10}$ explorations. The most brittle categories are the non-associative problems for $\mathbb{Z}_5$, where WALDMEISTER failed on 49 of 888 problems, and divisors and non-divisors problems for $\mathbb{Z}_{10}$, where WALDMEISTER failed on 39 of 39 problems and 197 of 223 problems, respectively. Note that this does not necessarily mean that WALDMEISTER might not be able to prove these problems at all if it were given a more specialized and fine tuned control setting. However, in our experiments we use exactly two different control settings, one suitable for all simple properties and one for non-isomorphism problems. According to our experiments, the overall performance of WALDMEISTER (i.e., whether it succeeds or fails on a problem) depends on the cardinality of the set involved: Higher cardinality implies a higher likelihood of failure.

5.3. Discussion

WALDMEISTER has a clear advantage to the proof planning approach wrt. runtime behavior. When it succeeds, it succeeds very fast independently of the cardinality of the residue class structure (30% of all proofs were produced in less than 1 second, 70% of all proofs were produced in less than 10). The runtime performance of the proof planning approach depends on which strategy can be applied successfully. Problems solved with the `ReduceToSpecial` or the `EquSolve` strategy usually take about 10 to 20 seconds independently of the cardinality of the given set. If `TryAndError` has to be applied it can take considerably longer, depending on the cardinality of the structures.

In our context a disadvantage of WALDMEISTER is, however, its output format. Although, WALDMEISTER has a proof presentation tool that tries to structure the found proof by lemmas, in our experiments this tool failed to successfully present many found proofs (e.g., on almost all associativity problems). And even proofs displayed by the presentation tool are relatively hard to read: On the one hand the proofs are very long, usually between 150 and 300 equational reasoning steps, structured with 10 to 30 lemmas. On the other hand the used lemmas are rather counterintuitive for humans. In contrast, the proof planning approach can produce very short proof plans when `ReduceToSpecial` ($\sim$ 10 steps) or `EquSolve` ($\sim$ 20 steps) are applied. Although proof plans with `TryAndError` can be very long, these proofs are structured in a clear way by the case splits. For

instance, a divisor proof for a structure with cardinality 10 consist of about 3000 nodes comprised of 100 clearly separate cases each consisting of about 30 steps.

It is a common criticism of proof planning to depend on specially prepared and fine-tuned domain knowledge. In contrast, ATPs such as WALDMEISTER seem not to depend on particular knowledge since they are based on general-purpose machine-oriented calculi. However, our experience with WALDMEISTER is that its application to our domain was successful only with a considerable amount of very specific knowledge. The WALDMEISTER strategy `WMonResclass` comprises, for instance, knowledge on how to suitably represent residue class structures for WALDMEISTER, knowledge on which theorems and lemmas for the basic operations should be added, and knowledge on which particular ordering of the used symbols to choose. This knowledge was absolutely crucial for a successful application of WALDMEISTER in our domain. Hence instead of encoding mathematical knowledge for the residue class domain, we had to encode knowledge specific to the theorem prover employed, which we could only do with the help of an expert. Likewise we failed to successfully apply the first-order resolution prover OTTER (McCune, 1994b) in our domain since we lacked the expert knowledge to find a suitable representation for our problems.

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 CASs 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 proof 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 proof techniques in a conceptually clear way into different strategies, but could also combine and interleave these strategies flexibly. We employed the CASs 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. A comparison showed that our approach can compete with alternative techniques. In particular, the comparison with WALDMEISTER showed that although most problems can be successfully tackled with a conventional ATP, our combined proof planning/computer algebra approach was more robust in a large case study. Moreover, the resulting proofs are more understandable to a human user. The experiments with SEM indicated that the used CASs could at least partially be replaced by a model generator. It definitely proved that model generation and computer algebra ideally complement each other in our problem

domain. This suggests a concurrent use of both systems in order to enhance the reliability of the hint system guiding the proof planner.

There are various accounts on experiments of combining computer algebra and theorem proving in the literature (see (Kapur and Wang, 1998) 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 by (Jackson, 1994). Our article 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 ΩMEGA (Kerber *et al.*, 1998; Sorge, 2000).

More concrete work in exploration in finite algebra is reported by Fujita *et al.* (1993); McCune (1994a); Slaney *et al.* (1995) 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 has the opposite motivation 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.

Acknowledgments We would like to thank Thomas Hillenbrandt with his help on the use of WALDMEISTER and Hans deNeville for stimulating discussions.

References

Benzmüller, C., Cheikhrouhou, L., Fehrer, D., Fiedler, A., Huang, X., Kerber, M., Kohlhase, M., Konrad, K., Melis, E., Meier, A., Schaarschmidt, W., Siekmann, J., and Sorge, V. (1997). ΩMega: Towards a Mathematical Assistant. In *Proceedings of CADE–14*, volume 1249 of *LNAI*. Springer Verlag.

Bundy, A. (1988). The Use of Explicit Plans to Guide Inductive Proofs. In *Proceedings of CADE–9*, volume 310 of *LNCS*. Springer Verlag.

Fujita, M., Slaney, J., and Bennett, F. (1993). Automatic generation of some results in finite algebra. In *Proceedings of IJCAI'93*. Morgan Kaufmann.

GAP (1998). *GAP – Groups, Algorithms, and Programming, Version 4*. The GAP Group. `http://www-gap.dcs.st-and.ac.uk/~gap`.

Gomes, C., Selman, B., and Kautz, H. (1998). Boosting combinatorial search through randomization. In *Proceedings of AAAI-98*. AAAI Press.

Hillenbrand, T., Jaeger, A., and Löchner, B. (1999). System description: Waldmeister : Improvements in performance and ease of use. In *Proceedings of CADE–16*, volume 1632 of *LNAI*. Springer Verlag.

Jackson, P. (1994). Exploring Abstract Algebra in Constructive Type Theory. In *Proceedings of CADE–12*, volume 814 of *LNCS*. Springer Verlag.

Kapur, D. and Wang, D., editors (1998). *J. of Automated Reasoning— Special on the Integration of Deduction and Symbolic Computation Systems*, volume 21(3). Kluwer Academic Publisher.

Kerber, M., Kohlhase, M., and Sorge, V. (1998). Integrating Computer Algebra Into Proof Planning. *J. of Automated Reasoning*, **21**(3), 327–355.

McCune, W. (1994a). 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.

McCune, W. W. (1994b). Otter 3.0 reference manual and guide. Technical Report ANL-94-6, Argonne National Laboratory, Argonne, Illinois 60439, USA.

Meier, A. (2000). Randomization and heavy-tailed behavior in proof planning. Seki Report SR-00-03, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany.

Meier, A. and Sorge, V. (2001). Exploring Properties of Residue Classes. In *Proceedings of the Calculemus Symposium 2000*. AK Peters.

Meier, A., Pollet, M., and Sorge, V. (2000). Exploring the domain of residue classes. Seki Report SR-00-04, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany.

Meier, A., Pollet, M., and Sorge, V. (2001). Classifying Isomorphic Residue Classes. In *Proceedings of EuroCAST 2001*, volume 2178 of *LNCS*. Springer Verlag.

Melis, E. and Meier, A. (2000). Proof planning with multiple strategies. In *Proceedings of the First International Conference on Computational Logic*, volume 1861 of *LNAI*. Springer Verlag.

Melis, E. and Siekmann, J. (1999). Knowledge-based proof planning. *Artificial Intelligence*.

Redfern, D. (1999). *The Maple Handbook: Maple V Release 5*. Springer Verlag.

Robinson, J. (1965). A machine-oriented logic based on the resolution principle. *JACM*, **12**.

Slaney, J., Fujita, M., and Stickel, M. (1995). Automated reasoning and exhaustive search: Quasigroup existence problems. *Computers and Mathematics with Applications*, **29**, 115–132.

Sorge, V. (2000). Non-Trivial Computations in Proof Planning. In *Frontiers of combining systems: Third International Workshop, FroCoS 2000*, volume 1794 of *LNCS*. Springer Verlag.

Zhang, J. and Zhang, H. (1996). Generating models by SEM. In *Proceedings of CADE–13*, volume 1104 of *LNAI*. Springer Verlag.