MAT 444 Honors Contract
GrCal, Group Calculator

Grant Marshall
grant.a.marshall@asu.edu

May 5, 2014

1 Introduction

In a classroom setting, it is common for mathematics to be taught incrementally, using smaller theorems and lemmas to prove progressively stronger results. An alternative (or supplementary) approach to proving theorems about particular groups (such as the group of all integers under addition) is to look at concrete examples of when the theorem holds or doesn't hold. When doing this, automation can be immensely helpful for generating such examples and checking if the desired properties hold. For this honors contract, I developed a set of programming tools for the construction and analysis of group structures, then applied it the developing conjectures about them.

2 Features

The software written for this project addresses a few select problems. First, there is a set of functions designed for the parsing of arbitrary group tables from a file. Second, there are functions designed to verify that a set and a binary operation on that set forms a valid group. Third, there are functions for quickly constructing common groups (like $\mathbb{Z}_n$ and $\mathbb{Z}_n^*$) to avoid the laborious process of manually creating group tables for every test. Finally, there are a number of utility functions for figuring out certain properties of groups and elements of those groups. We will go into further depth for each of these features.

A note about notation: when writing source code inline (like data structure names or function names), `monospace` font will be used.

2.1 Parsing Groups

Parsing of the group tables is handled entirely in the file Parsing.hs. This code is able to be very terse because it uses the library Parsec, which exposes an extremely simple API for composing monadic parsers. The format of the group tables is as follows: The first line is an ordered list of strings $x_1, x_2, \ldots, x_n$ where each string is separated by a tab character (\t). This permits arbitrary strings for the names of elements of the group. After this first line, there are n lines with n tab-separated elements per line. So if the symbol x_k appears in row i column j, it means that $*(x_i, x_j) = x_k$.

The main topic of this project isn't parsing, though, so that's all I'll really say about this here. There is also code for parsing simple expressions of the form "let $G \leftarrow$ groupZ 2" and other simple assignment/membership expressions, but this code went unused in the end.

2.2 Verifying Groups

Simply parsing a table and verifying it follows the general structure outlined in the parsing section is not sufficient to guarantee that the provided input table is actually a group. When

we parse a group table we only guarantee two things: we have a list of size n and the operation is defined on a domain of size n^2. First we have to check that the members of the list are unique, which is a very simple. Another simple check is to verify that all of the members of the two-dimensional table are in the set. Next, we check for an identity and check the sudoku rules, which enforces the requirement for inverses in the group. Finally, we check the associativity of the group by generating the triples of the group and ensuring that $(a * (b * c)) = ((a * b) * c)$ for all a, b, c in the set of elements.

The functions for performing these checks are exposed in the file GroupInput.hs. To check that an arbitrary `Table` is a group, we pass it to the `checkAll` function, which returns either a list of errors, or `True` to indicate that the `Table` is valid. If this returns `True`, we can pass it to the `convertTable` function to get a `Group` data structure. Once we have acquired a `Group` data structure, we can use the functions that will be detailed in subsection 2.4.

2.3 Constructing Groups

Though in sections 2.1 and 2.2 we describe ways to load arbitrary groups from a file, it can be quite tedious to have to manually describe each group when there is a more succinct way of describing the group to be programmatically generated. It was very straightforward to define a function for generating a `Group` representing the group $\mathbb{Z}_n$. The function `groupZ` located in Group.hs does exactly this. In the process of working on the use cases described in sections 3.1 and 3.2, there are also functions defined to generate $\mathbb{Z}_n^*$ and the semidirect product groups in the file Experiments.hs. Functions for programmatically generating groups or tables (the distinction being that we aren't certain that a table follows the requirements for a group) are very useful for section 3.

2.4 Properties of Elements/Groups

Now that we have described a number of ways of acquiring data structures representing groups, we need ways to find interesting properties from these data structures. In Group.hs, there are a number of functions for doing this. The functions `cyclic`, `abelian`, `elementOrders`, and `order` all have clear purposes. There are also more general functions like `groupPower` that allow you to take an element of a group and an integer n and find the nth power of that element. The cyclic function in particular was useful for section 3.1. Because a `Group` is simply represented a `Set` with a `Map` from `(String, String)` to `String`, defining these functions is very straightforward.

3 Use Cases

In this section are two examples of using the functions outlined in section 2 to develop conjectures. The first question we will ask is for what values of n is $\mathbb{Z}_n^*$ cyclic? We'll tackle this question by first creating functions to generate the group $\mathbb{Z}_n^*$, then using the function `cyclic` to figure out which of the generated $\mathbb{Z}_n^*$ groups are cyclic. The second question of interest is for which sets is $\mathbb{Z}_n \times \mathbb{Z}_m$ under the semidirect product operation a group? We'll flesh out this question further in section 3.2.

3.1 When is $\mathbb{Z}_n^*$ Cyclic?

To address this question, the `zStar` function was created in the Experiments.hs file. It accepts an integer n and produces a `Group` representing $\mathbb{Z}_n^*$. Using ghci, the interpreter for Haskell, we can construct the groups $\mathbb{Z}_2^*, \mathbb{Z}_3^*, \ldots, \mathbb{Z}_{100}^*$ and record which ones return `True` for the `cyclic` function (Table 1 in Appendix). After doing that, I noticed that all primes and prime powers were in the list, but there were some others that didn't fit the pattern. To investigate this further, I

passed the list of integers that produce a cyclic $\mathbb{Z}_n^*$ into the `Factor` function of Mathematica. The resulting factorization made the pattern apparent and resulted in the following conjecture:

Conjecture 1. *$\mathbb{Z}_n^*$ is cyclic iff one of the following conditions is true:*

a) *$n < 8$,*

b) *$n = p^k$ where p is prime and $k \in \mathbb{Z}^+$, or*

c) *$n = 2p^k$ where p is prime and $k \in \mathbb{Z}^+$.*

To test this further, integers up to 1000 were checked as well and the conjecture continued to hold. The proof in the forward direction can be tackled as follows: prove part 1 by exhaustion, prove part 2 by induction (base case of $\mathbb{Z}_p^*$ holds because $\mathbb{Z}_p$ is a field), prove part 3 by showing isomorphism between $\mathbb{Z}_{p^k}$ and $\mathbb{Z}_{2p^k}$. The point of this section isn't necessarily to prove these conjectures, but to show how computation can aid in coming up with conjectures.

3.2 Semidirect Product Groups

Before we tackle the question we have posed, we need to more concretely define the set we are working on and the operation we're using. We will define our (possible) group as follows: the set is $\mathbb{Z}_n \times \mathbb{Z}_m$ for some choice of integers m and n. Choose some $a \in \mathbb{Z}_n$. The operation $*$ is defined by $*((u, v), (x, y)) = (u + a^v x, v + y)$. Now our more specific problem is the following: is $\langle \mathbb{Z}_n \times \mathbb{Z}_m, * \rangle$ a group?

Now with our definitions out of the way, we will begin approaching the problem. It speeds up our computations to remove some obvious problems with the question. It is clear that if $(n, a) > 1$, we will have problems because for any $(u, v) \in \mathbb{Z}_n \times \mathbb{Z}_m$, there is either no (x, y) such that $(u, v) * (x, y) = (0, 0)$ or multiple such (x, y), which means $\langle \mathbb{Z}_n \times \mathbb{Z}_m, * \rangle$ is not a group in this case. This lets us speed up our computations by not bothering to check choices of a with $(n, a) > 1$.

Now we can proceed with creating the machinery to check for groups. First, we describe a function for creating a `Table` representing the set and operation described. We create a `Table` as opposed to a `Group` because it isn't clear for which values we will get a valid group. The function that does this is called `semidirectProductTable` and is located in the Experiments.hs file. From here, we can use the `checkAll` function that was defined for parsing group tables to check if the table is a valid group.

Now let's see what happens when we run it on actual choices. It is possible to easily check all choices of a for $m, n \leq 12$ on my computer using the routines written, so those are the ones the following conjecture is based on:

Conjecture 2. *Let $S = \mathbb{Z}_n \times \mathbb{Z}_m$ where $n, m \in \mathbb{Z}^+$ and $a \in \mathbb{Z}_n$. We define a binary operation $*$ on $(u, v), (x, y) \in S$ by $(u, v) * (x, y) = (u + a^v x, v + y)$. The set S is a group under the operation $*$ iff $(a, n) = 1$ and the integers n, m, a satisfy one of the following conditions:*

1. *a is 1. This results in $\langle S, * \rangle$ being isomorphic to $\mathbb{Z}_n \times \mathbb{Z}_m$,*
2. *m is 1 and $(n, m) = 1$. This results in $\langle S, * \rangle$ being isomorphic to $\mathbb{Z}_n$, or*
3. *a is m-torsion in $\mathbb{Z}_n^*$.*

This conjecture holds for all of the values up to this point. Once we're working in $\mathbb{Z}_{12} \times \mathbb{Z}_{12}$, the costs of checking all choices of a and the associativity of the operation becomes very expensive — $O(n^3)$ where n is the number of elements in the group — for each choice of a. This restricts the choices of n and m that we can compute for. As with the previous conjecture, a rigourous proof is not included here.

4 Conclusion

Creating these programs was a worthwhile experience. It was a refreshing change of pace to try coming up with conjectures from examples first. Developing programs to compute various group properties was interesting as well because when learning about theorems, I had often wondered which of the properties were easily computable. Once the functions were working correctly, it was fun to use them to address the questions posed in section 3. I think working on this honors contract has piqued an interest in computational algebra systems I didn't quite have before.

5 Appendix

Source Code for this software can be located on GitHub (https://github.com/gr-a-m/GrCal).

Table 1: Choices of $n \leq 100$ Such that $\mathbb{Z}_n^*$ is Cyclic

2, 3, 4, 5, 6, 7, 9, 10, 11, 13, 14, 17, 18, 19, 22, 23, 25, 26, 27, 29, 31, 34, 37, 38, 41, 43, 46, 47, 49, 50, 53, 54, 58, 59, 61, 62, 67, 71, 73, 74, 79, 81, 82, 83, 86, 89, 94, 97, 98

Table 2: Results of Semidirect Product Experiments (n, m, a)

Valid Groups	Invalid Groups
(12,12,11), (12,12,7), (12,12,5), (12,10,11), (12,10,7), (12,10,5), (12,8,11), (12,8,7), (12,8,5), (12,6,11), (12,6,7), (12,6,5), (12,4,11), (12,4,7), (12,4,5), (12,2,11), (12,2,7), (12,2,5), (12,1,11), (12,1,7), (12,1,5), (11,12,10), (11,10,10), (11,10,9), (11,10,8), (11,10,7), (11,10,6), (11,10,5), (11,10,4), (11,10,3), (11,10,2), (11,8,10), (11,6,10), (11,5,9), (11,5,5), (11,5,4), (11,5,3), (11,4,10), (11,2,10), (11,1,10), (11,1,9), (11,1,8), (11,1,7), (11,1,6), (11,1,5), (11,1,4), (11,1,3), (11,1,2), (10,12,9), (10,12,7), (10,12,3), (10,10,9), (10,8,9), (10,8,7), (10,8,3), (10,6,9), (10,4,9), (10,4,7), (10,4,3), (10,2,9), (10,1,9), (10,1,7), (10,1,3), (9,12,8), (9,12,7), (9,12,5), (9,12,4), (9,12,2), (9,10,8), (9,9,7), (9,9,4), (9,8,8), (9,6,8), (9,6,7), (9,6,5), (9,6,4), (9,6,2), (9,4,8), (9,3,7), (9,3,4), (9,2,8), (9,1,8), (9,1,7), (9,1,5), (9,1,4), (9,1,2), (8,12,7), (8,12,5), (8,12,3), (8,10,7), (8,10,5), (8,10,3), (8,8,7), (8,8,5), (8,8,3), (8,6,7), (8,6,5), (8,6,3), (8,4,7), (8,4,5), (8,4,3), (8,2,7), (8,2,5), (8,2,3), (8,1,7), (8,1,5), (8,1,3), (7,12,6), (7,12,5), (7,12,4), (7,12,3), (7,12,2), (7,10,6), (7,9,4), (7,9,2), (7,8,6), (7,6,6), (7,6,5), (7,6,4), (7,6,3), (7,6,2), (7,4,6), (7,3,4), (7,3,2), (7,2,6), (7,1,6), (7,1,5), (7,1,4), (7,1,3), (7,1,2), (6,12,5), (6,10,5), (6,8,5), (6,6,5), (6,4,5), (6,2,5), (6,1,5), (5,12,4), (5,12,3), (5,12,2), (5,10,4), (5,8,4), (5,8,3), (5,8,2), (5,6,4), (5,4,4), (5,4,3), (5,4,2), (5,2,4), (5,1,4), (5,1,3), (5,1,2), (4,12,3), (4,10,3), (4,8,3), (4,6,3), (4,4,3), (4,2,3), (4,1,3), (3,12,2), (3,10,2), (3,8,2), (3,6,2), (3,4,2), (3,2,2), (3,1,2)	(12,11,11), (12,11,7), (12,11,5), (12,9,11), (12,9,7), (12,9,5), (12,7,11), (12,7,7), (12,7,5), (12,5,11), (12,5,7), (12,5,5), (12,3,11), (12,3,7), (12,3,5), (11,12,9), (11,12,8), (11,12,7), (11,12,6), (11,12,5), (11,12,4), (11,12,3), (11,12,2), (11,11,10), (11,11,9), (11,11,8), (11,11,7), (11,11,6), (11,11,5), (11,11,4), (11,11,3), (11,11,2), (11,9,10), (11,9,9), (11,9,8), (11,9,7), (11,9,6), (11,9,5), (11,9,4), (11,9,3), (11,9,2), (11,8,9), (11,8,8), (11,8,7), (11,8,6), (11,8,5), (11,8,4), (11,8,3), (11,8,2), (11,7,10), (11,7,9), (11,7,8), (11,7,7), (11,7,6), (11,7,5), (11,7,4), (11,7,3), (11,7,2), (11,6,9), (11,6,8), (11,6,7), (11,6,6), (11,6,5), (11,6,4), (11,6,3), (11,6,2), (11,5,10), (11,5,8), (11,5,7), (11,5,6), (11,5,2), (11,4,9), (11,4,8), (11,4,7), (11,4,6), (11,4,5), (11,4,4), (11,4,3), (11,4,2), (11,3,10), (11,3,9), (11,3,8), (11,3,7), (11,3,6), (11,3,5), (11,3,4), (11,3,3), (11,3,2), (11,2,9), (11,2,8), (11,2,7), (11,2,6), (11,2,5), (11,2,4), (11,2,3), (11,2,2), (10,11,9), (10,11,7), (10,11,3), (10,10,7), (10,10,3), (10,9,9), (10,9,7), (10,9,3), (10,7,9), (10,7,7), (10,7,3), (10,6,7), (10,6,3), (10,5,9), (10,5,7), (10,5,3), (10,3,9), (10,3,7), (10,3,3), (10,2,7), (10,2,3), (9,11,8), (9,11,7), (9,11,5), (9,11,4), (9,11,2), (9,10,7), (9,10,5), (9,10,4), (9,10,2), (9,9,8), (9,9,5), (9,9,2), (9,8,7), (9,8,5), (9,8,4), (9,8,2), (9,7,8), (9,7,7), (9,7,5), (9,7,4), (9,7,2), (9,5,8), (9,5,7), (9,5,5), (9,5,4), (9,5,2), (9,4,7), (9,4,5), (9,4,4), (9,4,2), (9,3,8), (9,3,5), (9,3,2), (9,2,7), (9,2,5), (9,2,4), (9,2,2), (8,11,7), (8,11,5), (8,11,3), (8,9,7), (8,9,5), (8,9,3), (8,7,7), (8,7,5), (8,7,3), (8,5,7), (8,5,5), (8,5,3), (8,3,7), (8,3,5), (8,3,3), (7,11,6), (7,11,5), (7,11,4), (7,11,3), (7,11,2), (7,10,5), (7,10,4), (7,10,3), (7,10,2), (7,9,6), (7,9,5), (7,9,3), (7,8,5), (7,8,4), (7,8,3), (7,8,2), (7,7,6), (7,7,5), (7,7,4), (7,7,3), (7,7,2), (7,5,6), (7,5,5), (7,5,4), (7,5,3), (7,5,2), (7,4,5), (7,4,4), (7,4,3), (7,4,2), (7,3,6), (7,3,5), (7,3,3), (7,2,5), (7,2,4), (7,2,3), (7,2,2), (6,11,5), (6,9,5), (6,7,5), (6,5,5), (6,3,5), (5,11,4), (5,11,3), (5,11,2), (5,10,3), (5,10,2), (5,9,4), (5,9,3), (5,9,2), (5,7,4), (5,7,3), (5,7,2), (5,6,3), (5,6,2), (5,5,4), (5,5,3), (5,5,2), (5,3,4), (5,3,3), (5,3,2), (5,2,3), (5,2,2), (4,11,3), (4,9,3), (4,7,3), (4,5,3), (4,3,3), (3,11,2), (3,9,2), (3,7,2), (3,5,2), (3,3,2)