Constructing Irreducible Representations of Finite Simple Groups

Brian Bennett and Angela Kraft
Adviser: Klaus Lux

January 15, 2015

Contents

In this paper, we explain how to construct new irreducible representations of finite, simple groups from representations that we already know. We first provide a short introduction to representation and character theory. We then briefly introduce GAP and go through an example construction. Finally, we give a list of partial results as well as some direction on what still needs to be done.

1 Background

1.1 Matrix Representations

Let G be a finite group, F be a field and V be a finite dimensional F-vector space.

Definition 1.1. An **F-representation** of a group G is a homomorphism T from G to $GL(V)$. If a basis is chosen then the composite map

$$G \xrightarrow{T} GL(V) \to GL_n(F)$$

is calledn an **F-matrix representation** of a group G.

If S and T are F-representations of G, then we consider S and T to be equivalent representations if there exists an $X \in GL_n(F)$ such that for all g in G, $X \cdot S(g) \cdot X^{-1} = T(g)$. Note that this just corresponds to a change of basis.

Definition 1.2. A representation T is called **reducible** if there is a non-trivial proper T-invariant subspace. Otherwise it is **irreducible**.

Reducible representations are equivalent to a reprsentation of the form

$$\begin{bmatrix} T_1 & A_{12} & \dots & A_{1k} \\ 0 & T_2 & \dots & A_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & T_r \end{bmatrix}$$

where T_k are irreducible F-representations and A_{ij} are blocks of appropriate size. If a representation T is equivalent to one where all $A_{ij} = 0$, then T is called **completely reducible** and is a direct sum of irreducible F-representations.

Example 1.3. (Permutation Representation) Consider the action of S_3 on the set $\{1, 2, 3\}$. This defines a representation $T : S_3 \to GL_3(\mathbb{C})$ by

$$T((1\ 2)) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } T((1\ 2\ 3)) = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Note that T is reducible since it has an invariant subspace:

$$\left\langle \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\rangle$$

Let

$$X = \begin{bmatrix} 1 & 1 & 0 \\ 1 & -1 & 1 \\ 1 & 0 & -1 \end{bmatrix}$$

Then applying this change of basis we get:

$$X^{-1}T((1\ 2))X = S((1\ 2)) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

and

$$X^{-1}T((1\ 2\ 3))X = S((1\ 2\ 3)) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}.$$

Thus S and T are equivalent representations. Since S is the direct sum of a 1 dimensional irreducible representation (namely the trivial representation) and a 2 dimensional irreducible representation, both S and T are completely reducible.

Theorem 1.4. (Maschke's Theorem) If the characteristic of F does not divide the order of G, then every F-representation of G is completely reducible.

Regardless of the characteristic of a field, any reducible representation can be written so that the irreducible representations appear in block diagonal form and the lower blocks are all 0.

Example 1.5. Consider $G = Z_2 =< x : x^2 = 1 >$ and $F = \mathbb{F}_2$. Note that the characteristic of F divides the order of the group. Define a representation $T : Z_2 \to GL_2(\mathbb{F}_2)$ by

$$T(x) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Then T has an invariant subspace:

$$\left\langle \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\rangle,$$

and its 1 dimensional diagonal blocks are irreducible representations (namely the trivial representation), but $T(x)$ is not similar to a diagonal matrix.

Since the irreducibility of a representation depends on the field F that the representation is over, it is often beneficial to only consider representations that are irreducible over any finite extension of F. A representation is called **absolutely irreducible** if for all finite extensions $F(\alpha)$, T is an irreducible $F(\alpha)$-representation. A field where all irreducible representations of G are absolutely irreducible is called a **splitting field** field for G. Note that G is guaranteed to have a splitting field since any algebraically closed field will be a splitting field for G.

1.2 Character Theory

As the degree of a representation increases it can become cumbersome to work with matrix representations. To overcome this fact, we introduce the concept of character theory. Characters of representations can often give a lot of information about a group and its representations without heavy computation. For this section, assume that F is a splitting field for G.

Definition 1.6. Let $g \in G$ and T an F-representation of G. The **character** of a representation T of G is a function $\chi : G \to F$ defined by

$$\chi(g) = Tr(T(g))$$

Note that the character of a representation is constant on conjugacy classes, so when looking at characters of a group G, one generally justs looks at conjugacy class representatives.

Theorem 1.7. Suppose the characteristic of F is 0. Let S and T be F-representations of G and χ_S, χ_T be their characters, respectively. Then S is equivalent to T if and only if $\chi_S = \chi_T$.

In this case, equivalence classes of representations are uniquely determined by their characters. We say that a character is irreducible if it is the character of an irreducible representation. Any F-character can be written as the sum of irreducible F-characters. This comes from the fact that reducible representations can be written so that irreducible representations occur along the diagonal. Thus the trace of the representation is equal to the sum of the traces of the irreducible representations occurring along the diagonal.

Definition 1.8. A **class function** is a function $f : G \to F$ such that f is constant on the conjugacy classes of G.

In characteristic 0, the number of absolutely irreducible is equal to the number of conjugacy classes.

Let $F \subset \mathbb{C}$ be a splitting field for G. Then we can define and inner product on the vector space of functions $G \to F$ by

$$(\phi, \theta) = |G|^{-1} \sum \{\phi(x)\theta(x^{-1}) : x \in G\}$$

It turns out that the irreducible F-characters form an orthonormal basis for class functions of G. Since if $\chi_1, ..., \chi_r$ are the irreducible F-characters of G then for $1 \leq i, j \leq r$, $(\chi_i, \chi_j) = \delta_{ij}$ and for θ any class function, $\theta = \sum_{k=1}^{r} (\theta, \chi_k)\chi_k$.

Example 1.9. Let $G = S_3$ and $F = \mathbb{C}$. Define $T : S_3 \to GL_3(\mathbb{C})$ by

$$T((1\ 2)) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } T((1\ 2\ 3)) = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

The conjugacy classes of S_3 are determined by cycle type, so S_3 has three conjugacy classes: $\{1\}$, $\{(1\ 2), (1\ 3), (2\ 3)\}$ and $\{(1\ 2\ 3), (1\ 3\ 2)\}$. Thus S_3 must have three irreducible representations: χ_1, χ_2, χ_3. The character table for S_3 is:

	1	(1 2)	(1 2 3)
χ_1	1	1	1
χ_2	1	-1	1
χ_3	2	0	-1

Let χ_T denote the character of T, then we compute:

	1	(1 2)	(1 2 3)
χ_T	3	1	0

Using the inner product we find $(\chi_T, \chi_1) = 1$, $(\chi_T, \chi_2) = 0$, and $(\chi_T, \chi_3) = 1$. Thus $\chi_T = \chi_1 + \chi_3$. In fact, in Example 1.3 we already showed that χ_T decomposed into the direct sum of a 1 dimensional irreducible representation and a 2-dimensional representation. These are the unique representations (up to equivalence) that correspond to χ_1 and χ_2 respectively. Thus if we know all of the irreducible $\mathbb{C}$-representations, we can use the characters to decompose a reducible $\mathbb{C}$-representation into the direct sum of irreducible $\mathbb{C}$-representations. Unfortunately things do not work as nicely when the characteristic of F divides the order of G.

1.3 Brauer Character Theory

If the characteristic of F divides the order of G, then Maschke's theorem does not hold, so representations are not uniquely determined by the irreducible representations occurring along the diagonal, called the composition factors, as in the characteristic 0 case. Thus unlike in characteristic 0, one cannot hope that characters uniquely determine representations (up to equivalence). One would hope that the the characters would uniquely determine the composition factors. Unfortunately, this is not true. In fact, in any characteristic p, characters do not uniquely determine the composition factors. That is, two non-equivalent representations with different composition factors can have the same character.

Example 1.10. Let $G = Z_6 =< x : x^6 = 1 >$ and $F = \mathbb{F}_2$. Define $S : G \to GL_1(\mathbb{F}_2)$ and $T : G \to GL_3(\mathbb{F}_2)$ by

$$S(x) = 1 \text{ and } T(x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Note that S is the trivial representation and T is the direct sum of 3 trivial representations. Let χ_S and χ_T denote the characters of S and T respectively. Then we compute

	1	x	x^2	x^3	x^4	x^5
χ_S	1	1	1	1	1	1
χ_T	1	1	1	1	1	1

Thus S and T have the same character but S and T are not equivalent representations. In essence, the character of a representation does not account for the multiplicity of its composition factors in characteristic p.

To fix this problem, we turn instead to Brauer characters. Suppose that F has characteristic p. An element is said to be **p-regular** if it has order coprime to p.

Definition 1.11. Let α be a bijection between F^* and complex $|F|-1$ roots of unity. Let $\{x_j\}$ be the eigenvalues of $T(g)$ with multiplicity where $g \in G$ is p-regular. The **Brauer character** of an F-representation of G is a function $\beta : G_p \to \mathbb{C}$ defined by

$$\beta(g) = \sum \alpha(x_j)$$

where G_p denotes the set of p-regular elements.

Similar to the characteristic 0 case, the number of irreducible F-representations of G is equal to the number of conjugacy classes of p-regular elements.

Example 1.12. Let $G = Z_6 =< x : x^6 = 1 >$ and $F = \mathbb{F}_2$. Define $S : G \to GL_1(\mathbb{F}_2)$ and $T : G \to GL_3(\mathbb{F}_2)$ by

$$S(x) = 1 \text{ and } T(x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Let β_S and β_T denote the Brauer characters of S and T respectively. Note that Z_6 has 3 2-regular conjugacy classes: $\{1\}, \{x^2\}, \{x^4\}$. Then we compute

	1	x^2	x^4
β_S	1	1	1
β_T	3	3	3

From the Brauer character we can clearly see that S and T are not equivalent representations. In fact since $\beta_T = 3\beta_S$, we know that $T = (S \oplus S \oplus S) + U$ where U is a strictly upper diagonal matrix. That is, T can be written as a direct sum of copies of S plus some upper diagonal matrix (to take into account that T may not be completely reducible).

The Brauer character uniquely determines the composition factors. Since irreducible representations have a single composition factor, the irreducible Brauer characters are in bijection with the irreducible representations. Every Brauer character can be uniquely written as a direct sum of the irreducible Brauer characters. Again, since Maschke's theorem does not hold, we need to be a little careful. The Brauer character will give the composition factors, but it will not give an equivalence class of representations.

Example 1.13. Let $G = Z_6 =< x : x^6 = 1 >$ and $F = \mathbb{F}_4$ is a splitting field for Z_6. Define $R : G \to GL_2(\mathbb{F}_2), S : G \to GL_2(\mathbb{F}_2)$, and $T : G \to GL_2(\mathbb{F}_2)$ by

$$R(x) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, S(x) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \text{ and } T(x) = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

We have the Brauer character table for Z_6:

	1	x^2	x^4
β_1	1	1	1
β_2	1	ζ	ζ^2
β_3	1	ζ^2	ζ

where ζ is a primitive 3rd root of unity. We compute the Brauer characters for R, S and T:

	1	x^2	x^4
β_R	2	2	2
β_S	2	2	2
β_T	2	1	1

Although R and S have the same Brauer character, they are not equivalent representations. They do, however have the same composition factors since $\beta_R = \beta_S = 2\beta_1$. The decomposition of Brauer characters tells us that we can decompose R and S into $T_1 \oplus T_1 + U$ where $U = \begin{bmatrix} 0 & a \\ 0 & 0 \end{bmatrix}$ for some $a \in F_4$. In this case T_1 is the trivial representation corresponding to β_1, so $T_1(x) = 1$. We see that $a = 0$ for R and $a = 1$ for S. That is

$$R = \begin{bmatrix} T_1 & 0 \\ 0 & T_1 \end{bmatrix} \text{ and } S = \begin{bmatrix} T_1 & 1 \\ 0 & T_1 \end{bmatrix}$$

On the other hand, T is neither equivalent to R or S, nor does it have the same composition factors. We can decompose $\beta_T = \beta_2 + \beta_3$. Let T_2 and T_3 be the irreducible representations corresponding to β_2 and β_3 respectively. Then we can decompose $T = T_2 \oplus T_3 + U$ where $U = \begin{bmatrix} 0 & a \\ 0 & 0 \end{bmatrix}$ for some $a \in F_4$. That is T is equivalent to an upper block diagonal representation of the form

$$\begin{bmatrix} T_2 & a \\ 0 & T_3 \end{bmatrix}.$$

1.4 Tensoring Representations

There are numerous ways of building new representations from ones that are already known. One method is tensoring. Let $S : G \to GL(V)$ and $T : G \to GL(W)$ be representations with V and W being F-vector spaces, and with respective characters χ and ψ. Then for all $g \in G$, we have that $S(g) \otimes T(g)$ is a linear transformation of $V \otimes_F W$, with $S(g) \otimes T(g)$ defined by

$$(S(g) \otimes T(g))(\textstyle\sum_k v_k \otimes_F w_k) = \sum_k S(g)v_k \otimes_F T(x)w_k.$$

It's then easy to check that the map $S \otimes T : G \to GL(V \otimes_F W)$ defined by $g \longmapsto S(g) \otimes T(g)$ is a representation.

Furthermore, if $\{v_1, \ldots, v_m\}$ and $\{w_1, \ldots w_n\}$ are bases for V and W respectively, then $\{v_i \otimes w_j : i = 1, \ldots, m, j = 1, \ldots, n\}$ is a basis for $V \otimes_F W$ choosing the natural ordering of the basis (the ordering will matter when trying to write these as matrix representations). So with regards to this (ordered) basis, we have the matrix representation

$$(S \otimes T)(g) = [s_{ik}T(g)] = \begin{bmatrix} s_{11}T(g) & \dots & s_{1n}T(g) \\ \vdots & & \vdots \\ s_{n1}T(g) & \dots & s_{nn}T(g) \end{bmatrix}$$

where $S(g) = [s_{ik}(g)]$. Looking at this definition in terms of matrices, it's then clear that the character of $S \otimes T$ is $\chi \cdot \psi$. We summarize this as a proposition:

Theorem 1.14. If χ and ψ are F-characters of G then the product $\chi \cdot \psi$ is also an F-character.

One thing to note is that if χ and ϕ are irreducible F-characters of G with ϕ linear, then it is easily shown that $\phi \cdot \chi$ is an irreducible F-character. In general though, it is not true that product of two irreducible F-characters is irreducible, and it is this fact that we will be heavily relying on in constructing new matrix representations.

Finally, we state a theorem of Brauer and Burnside:

Theorem 1.15. Let G be a finite group, ψ be a faithful, irreducible character of G, and let ψ take on m distinct values.. Then every irreducible representation of G appears as a summand in at least on of $\psi^0, \psi^1, \dots, \psi^m$.

If we define the kernel of a character χ to be

$$\mathrm{Ker}(\chi) = \{g \in G : \chi(g) = \chi(1)\}$$

then kernels are normal subgroups, and since we're working with finite simple groups, all irreducible characters must have trivial kernel and are thus faithful. This means that given any nontrivial character which we have a representation for, we can theoretically construct all other representations by tensoring. Practically though, this can be very inefficient and computationally expensive. If we only have a representation for a degree 100 character ψ, then looking at ψ^3 we're already working with 1,000,000 $\times$ 1,000,000 matrices! In practice, we choose the most efficient representations to tensor in order to build a new one, rather than tensoring a single representation with itself repeatedly.

2 An Example Computation

Before we go through an example, we'll say a few words about GAP (groups, algorithms, programming). GAP is a system for computational discrete algebra. It can be used for lots of different algebraic structures, but there is an emphasis on computational group theory. Of particular interest to us is a package called AtlasRep. This package contains information

on representations of finite, simple groups. Unfortunately, it's an incomplete library. Many groups are missing actual irreducible representations and our goal is to construct these representations to sort of "complete" the package. Also of note is a package known as the MeatAxe. It contains most of the computationally intensive commands that we utilize. In particular, it contains the commands that actually build the irreducible representations we're interested in.

We'll now go through an example detailing how we construct new matrix representations. We'll consider the group A_{10} and characteristic 2.

The first thing we must do is determine which characters we already have representations for and which characters we don't. To do this, we use the command AllAtlasGeneratingSetInfos("A10",Characteristic,2) to have GAP tell us what representations over fields of characteristic 2 are stored in the AtlasRep package. We then proceed to build modules for each of these representations and use the MeatAxe command MTX.IsAbsolutelyIrreducible to tell us which representations are absolutely irreducible.

With this list in hand, we then must find what characters corresponding to these absolutely irreducible representations. While this may seem to be a simple step as GAP has no problem finding the trace of a matrix, there is a choice to be made for which conjugacy class representative we find the trace. Kyle Pounder and Dan Rossi fixed these choices, which we use when calling the command LocalClasses. So with their results in hand, we use our command gapchars("A10",2) (for a more detailed look at this command, see section 4) to tell us which rows of the character table have absolutely irreducible representations already stored in AtlasRep:

```
gap> gapchars("A10",2);
[ 2, 3, 4, 5, 7, 6, 8, 9, 10, 11 ]
```

So this means that in the character table of A_{10} we already have representations stored for the characters $\chi_1, \ldots, \chi_{11}$ and are only missing a representation corresponding to χ_{12}, the last row of the character table.

```
     2   7   3  3  .  2  .  .  .  .   .   .   .
     3   4   3  3  4  1  .  1  2  2   1   1   1
     5   2   1  .  .  2  2  .  .  .   1   .   .
     7   1   1  .  .  .  .  1  .  .   .   1   1

        1a  3a 3b 3c 5a 5b 7a 9a 9b 15a 21a 21b
     2P 1a  3a 3b 3c 5a 5b 7a 9a 9b 15a 21b 21a
     3P 1a  1a 1a 1a 5a 5b 7a 3c 3c  5a  7a  7a
     5P 1a  3a 3b 3c 1a 1a 7a 9a 9b  3a 21a 21b
     7P 1a  3a 3b 3c 5a 5b 1a 9a 9b 15a  3a  3a

X.1      1   1  1  1  1  1  1  1  1   1   1   1
X.2      8   5  2 -1  3 -2  1 -1 -1   .  -2  -2
X.3     16  -8  4 -2 -4  1  2  1  1   2  -1  -1
X.4     26   8 -1 -1  1  1 -2 -1 -1  -2   1   1
X.5     48   6  .  3 -2 -2 -1  .  .   1  -1  -1
X.6     64 -20  4  1 -6 -1  1  1 -2   .   1   1
X.7     64 -20  4  1 -6 -1  1 -2  1   .   1   1
X.8    160  34 -2 -2  5  . -1  1  1  -1  -1  -1
X.9    198   6 -6  . -2 -2  2  .  .   1  -1  -1
X.10   200 -28 -4  2  .  . -3 -1 -1  -3   .   .
X.11   384 -24  . -3  4 -1 -1  .  .   1   A  *A
X.12   384 -24  . -3  4 -1 -1  .  .   1  *A   A

A = E(21)^2+E(21)^8+E(21)^10+E(21)^11+E(21)^13+E(21)^19
  = (1-Sqrt(21))/2 = -b21
```

Now that we know which characters we have representations for, we go about multiplying these characters and decomposing the products to see which have χ_{12} as a summand. To do this we use our command bplan("A10", 2, gapchars("A10",2)) to get:

```
gap> bplan("A10",2,gapchars("A10",2));
[ [ 1 ], [ 2 ], [ 3 ], [ 4 ], [ 5 ], [ 6 ], [ 7 ], [ 8 ], [ 9 ], [ 10 ],
  [ 11 ], [ 12, [ "16a", "48a" ] ] ]
```

This means that χ_{12} shows up as a summand when we multiply characters 16a and 48a, or characters χ_3 and χ_5 when looking at the character table. An important thing to note here is our command bplan returns the smallest degree character that has χ_{12} as a summand. It could be that χ_5 shows up as a summand of $\chi_{11} \cdot \chi_{11}$, but when it comes time to tensor the representations, it's much more computationally efficient to work with 768×768 matrices instead of $147,456 \times 147,456$ matrices.

Finally, we tensor the two representations corresponding to χ_3 and χ_5 and decompose to get a representation for χ_{12}. To do this we use the MeatAxe command GModuleByMats to create a module over a specified field from a generating set of invertible matrices for each of

the representations corresponding the χ_3 and χ_5. Then we use the command TensorProductGModule to tensor the representations, and finally use MTX.ColectedFactors to decompose the tensor product.

```
gap> m1:=GModuleByMats(AtlasGenerators("A10",9).generators,GF(2^2));
rec( dimension := 16, field := GF(2^2),
  generators := [ < immutable compressed matrix 16x16 over GF(4) >,
      < immutable compressed matrix 16x16 over GF(4) > ], isMTXModule := true )
gap> m2:=GModuleByMats(AtlasGenerators("A10",11).generators,GF(2^2));
rec( dimension := 48, field := GF(2^2),
  generators := [ < immutable compressed matrix 48x48 over GF(4) >,
      < immutable compressed matrix 48x48 over GF(4) > ], isMTXModule := true )
gap> tensor:=TensorProductGModule(m1,m2);
rec( dimension := 768, field := GF(2^2),
  generators := [ < immutable compressed matrix 768x768 over GF(4) >,
      < immutable compressed matrix 768x768 over GF(4) > ], isMTXModule := true )
gap> MTX.CollectedFactors(tensor);
[ rec( IsIrreducible := true, dimension := 384, field := GF(2^2),
          generators := [ < immutable compressed matrix 384x384 over GF(4) >,
              < immutable compressed matrix 384x384 over GF(4) > ]
```

There are a couple notes to make about the above code. Firstly, the "9" and "11" in the GModuleByMats commands correspond to the number that the representations we're interested in are given in the AtlasRep package. Secondly, in the GModuleByMats command, we specified GF(2^2) as the field we were building these representations over. We did this because GF(2^2) is the splitting field for A_{10} in characteristic 2. The representations corresponding to χ_3 and χ_5 actually are both over GF(2), and if we build the modules as such, the resulting tensor will not actually decompose properly since χ_{12} is over GF(2^2). We'll say more about this shortly. Lastly, most of the output for the MTX.CollectedFactors has been suppressed here to only show the generators for the representation we were interested in building. The output actually gives a characteristic polynomial for each factor, and when those polynomials are of degree 384, they're rather difficult to look at.

Finally, we store this representation in the AtlasRep to be used for other calculations. It's a rather technical process which has to do with understanding how things are stored in GAP. We won't go through the process here, but to see the code for how this is done, see section 4.

3 Results and Future Work

After designing a program to do the work of adding character names and constructing representations for us (see Section 4), we ran the program for some simple groups. The following

table lists the groups that we have created private directories for. A "✓" in the names column indicates that we added character names for some of the preexisting representations. A "✓" in any numbered column indicates that all representations have been added in that characteristic, a "PD" in the column indicates that we have added some new representations in that characteristic but not all, an "X" indicates that we have yet to run our program for that group and characteristic and finally a "-" indicates that the group has no representations for that characteristic. Empty boxes indicate that the AtlasRep was not missing any information for that group and characteristic (so either the group already had all representations for that characteristic, or that prime does not divide the order of the group).

Group	names	2	3	5	7	11	13	17	19	23	31
A_5	✓										
A_6	✓										
A_7	✓										
A_8	✓		X	X	X						
A_{11}	✓	X	X	X	X	X					
A_{12}	✓	X	X	X	X	X					
$L2(8)$					✓						
$L2(13)$			✓		✓						
$L2(23)$			✓			✓					
$L2(27)$		✓	✓		✓		✓				
$L2(29)$		✓	PD	✓	✓						
$L2(31)$			✓	✓							✓
$L2(32)$	✓		X			X					X
$L3(2)$	✓										
$L3(3)$	✓	✓									
$L3(4)$	✓										
$L3(5)$	✓	✓	✓								✓
$L3(7)$	✓	X	X		X				-		
$M12$			✓	✓		✓					
$M22$			✓	✓							
$M23$			✓	PD	✓	✓				✓	
$M24$			PD	X	X	X				X	
$S4(4)$	✓	X	X	X				X			
$S4(5)$	✓	X	X	X			-				
$Sz(8)$		✓			✓						
$U3(3)$	✓		✓		✓						
$U3(4)$	✓		✓	✓			✓				
$U4(2)$		✓		✓							

The next step is to finish running the programs for these groups.

Notice that some groups do not have any representations for a given characteristic. Due to this, our methods for building new representations would not work. The next step would

be to use a different method to create at least one representation of each characteristic, and then run our programs to find the rest of the representations for that characteristic. The following table gives more groups which do not have any representations for the listed characteristics:

Group	Characteristics
A_8	3, 5, 7
$L2(25)$	2, 3, 5, 13
$L3(9)$	2, 3, 5, 7, 13
$L4(3)$	2, 3, 5, 13
$O8+(2)$	2, 3, 5, 7
$S6(2)$	5, 7
$U3(5)$	2, 3, 5, 7
$U3(9)$	2, 3, 5, 73
$U4(3)$	2, 3, 5, 7

Throughout our work, we have been working with a splitting field for the group. In general, representations can be realized over much smaller fields. GAP generally prefers to store representations over this smaller field. Future work would be to take our current representations, apply a change of basis, and store them in GAP over a smaller field.

4 Source Code

Note that in order for the following programs to work in GAP, a package called "basic" is needed.

AbsIrredReps The program "AbsIrredReps" takes in two parameters. The first parameter is a string corresponding to a group in the AtlasRep package. The second parameter is the (positive) characteristic. "AbsIrredReps" takes the list of representations in the AtlasRep, removes all the representations that are not absolutely reducible, and returns the remaining list (a list of all the absolutely irreducible representations stored in the Atlas).

```
AbsIrredReps:= function(gapname,p)
local gp, i, k, m, rep, reps, list;
gp := AllAtlasGeneratingSetInfos(gapname);
list := [];
reps := AllAtlasGeneratingSetInfos(gapname,Characteristic,p);
k := Length(reps);
for i in [1..k] do
    rep := AtlasGenerators(gp[reps[i].repnr].identifier);
    m := GModuleByMats(rep.generators,rep.ring);
    if MTX.IsAbsolutelyIrreducible(m) then
```

```
        Add(list, reps[i]);
    fi;
od;
return list;
end;
```

IrrChar The program "IrrChar" takes in three parameters. The first parameter is a string corresponding to a group in the AtlasRep package. The second parameter is the (positive) characteristic. The third parameter is a number corresponding to a specific representation in the AtlasRep. The output is the Atlas character name of the representation (as a string).

```
IrrChar := function(g,p,n)
local gp, a, b, c, d, traces, ct, pos;
gp := AllAtlasGeneratingSetInfos(g);
a := AtlasStraightLineProgram(g,"classes");
    if a = fail then
        a := LocalClasses(g);
            if a = fail then
                return fail;
                fi;
    fi;
b := AtlasGenerators(gp[n].identifier);
c := ResultOfStraightLineProgram(a.program, b.generators);
d := Filtered(c, x -> Order(x) mod p > 0);
traces := List(d, x -> BrauerCharacterValue(x));
ct := CharacterTable(g) mod p;
pos := Position(Irr(ct), traces);
    if pos = fail then
        return fail;
        else
        return AtlasCharacterNames(ct)[pos];
    fi;
end;
```

charreps The program "charreps" takes in two parameters. The first parameter is a string corresponding to a group in the AtlasRep package. The second parameter is the (positive) characteristic. The output is a list of numbers of which characters in the character table have representations in the AtlasRep.

```
charreps := function(G, p)
local charnames, chars, i;
charnames := [];
chars := AbsIrredReps(G,p);
for i in [1..Size(chars)] do
```

```
    if IsBound(chars[i].charactername) = true then
        Add(charnames, chars[i].charactername);
    else
        Add(charnames, IrrChar(G, p, chars[i].repnr));
    fi;
od;
return charnames;
end;
```

gapchars The program "gapchars" takes in two parameters. The first parameter is a string corresponding to a group in the AtlasRep package. The second parameter is the (positive) characteristic. The output is a list of numbers which indicate which characters have corresponding representations stored in the AtlasRep package.

```
gapchars := function(G,p)
local nums, names, ct,i;
nums := [];
names := charreps(G,p);
ct := CharacterTable(G) mod p;
for i in [1..Size(names)] do
    Add(nums, Position(AtlasCharacterNames(ct),names[i]));
od;
return nums;
end;
```

incompletechars The program "incompletechars" takes in two parameters. The first parameter is a string corresponding to a group in the AtlasRep package. The second parameter is the (positive) characteristic. The output is a list of numbers which correspond to the characters in the character table which do not have corresponding representations in the AtlasRep.

```
incompletechars := function(G,p)
local completechars,ct,ichars,i;
completechars := [1];
Append(completechars, gapchars(G,p));
ct:= CharacterTable(G) mod p;
ichars :=[];
for i in [1..Size(Irr(ct))] do
    if Position(completechars, i) = fail then
        Add(ichars, i);
    fi;
od;
return ichars;
end;
```

finishgroup The program "finishgroup" takes in one parameter, a string corresponding to a group in the AtlasRep package. The program prints out whether the group has representations for all prime characteristics dividing the order of the group. If the group is missing certain characteristics, it prints a list of these characteristics. The program also prints out which characteristics do not have any matrix representations stored in the AtlasRep.

```
finishgroup := function(g)
local l, i, p, primes, m,j,none;
l := Collected(Factors(Size(CharacterTable(g))));
primes := [];
m := [];
    for i in [1..Length(l)] do
        p := l[i][1];
        if Length(incompletechars(g,p)) >0 then
            Add(primes,p);
        fi;
    od;
if primes = [] then
    Print(g);
    Print(" has all representations for all primes dividing the order.");
    else
        Print(g);
        Print(" needs representations for ");
        Print(primes);
        none := [];
        for j in primes do
            if AbsIrredReps(g,j) = [] then
                Add(none,j);
            fi;
        od;
        if Length(none) >0 then
            Print(" but the group has no representations
            in the Atlas for ");
            Print(none);
        fi;
fi;
end;
```

smallest The program "smallest" takes in three parameters. The first parameter is a string corresponding to a group in the AtlasRep package. The second parameter is the (positive) characteristic. The third parameter is a list in the form $[n, [k_1, k_2], [k_3, k_4], \ldots, [k_r, k_{r+1}]$ where n is a number corresponding to a specific representation in the AtlasRep and each $[k_i, k_{i+1}]$ is a pair of numbers corresponding to

specific representations in the AtlasRep and the representation corresponding to n is a constituent of the tensor of the representations corresponding to k_i and k_{i+1}. The output is a list in the form $[n, [k_i, k_{i+1}]]$ where $[k_i, k_{i+1}]$ is the smallest dimensional tensor in the list.

```
smallest:= function(G,p,l)
local ct, irr, list, small, sc, nc, i;
ct := CharacterTable(G) mod p;
irr := Irr(ct);
list := l;
if Length(l)>1 then
    small := l[2];
    for i in [2..Length(l)] do
        sc := Tensored([irr[small[1]]],[irr[small[2]]]);
        nc := Tensored([irr[l[i][1]]],[irr[l[i][2]]]);
        if nc[1][1] < sc[1][1] then
            small := l[i];
        fi;
        od;
    list := [l[1],[AtlasCharacterNames(ct)[small[1]],
    AtlasCharacterNames(ct)[small[2]]]];
fi;
return list;
end;
```

bplan The program "bplan" takes in three parameters. The first parameter is a string corresponding to a group in the AtlasRep package. The second parameter is the (positive) characteristic. The third parameter is a list of numbers corresponding to the characters in the character table which have associated representations in the AtlasRep. The output is a list in the form $[[n_1, [k_1, k_2]], [n_2, [k_3, k_4]], \ldots$ where n_i is a number corresponding to the representations characters in the character table, all k_i are strings of Atlas character names, and $[k_i, k_{i+1}]$ corresponds to the smallest tensor (using representations in the list) of which the associated n_i is a constituent.

```
bplan := function(G, p, l)
local ct, irr, list, listb, tensor, decomp, i, j, k, m, n;
ct := CharacterTable(G) mod p;
irr := Irr(ct);
list := [];
listb := [];
for k in [1..Size(Irr(ct))] do
    Add(listb,[k]);
od;
for i in l do
    for j in l do
```

```
    if j>i-1 then
        tensor := Tensored([irr[i]],[irr[j]]);
        decomp := Decomposition(irr, tensor, 5);
        for n in [2..Size(Irr(ct))] do
            if n in l then
                listb:= listb;
                else if decomp[1][n] > 0 then
                    Add(listb[n],[i,j]);
                fi;
            fi;
        od;
    fi;
    od;
od;
for m in [1..Length(listb)] do
    Add(list,smallest(G,p,listb[m]));
od;
return list;
end;
```

newlist The program "newlist" takes in three parameters. The first parameter is a string corresponding to a group in the AtlasRep package. The second parameter is the (positive) characteristic. The third parameter is a list of numbers corresponding to characters with associated representations in the AtlasRep. The output is a list of numbers corresponding to characters in the character table which can be built by tensoring characters from the original list. Note that this program does not record how to build these new representations, just which ones can be built.

```
newlist:= function(G,p,l)
local list,blist,i;
list := l;
blist := bplan(G,p,l);
for i in [1..Length(blist)] do
    if Length(blist[i])> 1 then
        Add(list, blist[i][1]);
    fi;
od;
return list;
end;
```

nplan The program "nplan" takes in two parameters. The first parameter is a string corresponding to a group in the AtlasRep package. The second parameter is the (positive) characteristic. The output is a list in the form $[[n_1, [k_1, k_2]], [n_2, [k_3, k_4]], \ldots$ where n_i is a number corresponding to the representations characters in the character table, all k_i are

strings of Atlas character names, and $[k_i, k_{i+1}]$ corresponds to the smallest tensor (using representations in the list) of which the associated n_i is a constituent. Input: group, prime. Recursively does bplan until you construct all the characters. Returns list of lists. Output i the form [...,n,[a,b],..] where n is the number (in the character table) of the character you don't have a rep for and a and b are the Atlas names of the reps you should tensor to get n. This program recursively does "bplan" until all characters have been constructed.

```
nplan:= function(G,p)
local l,ct,irr,blist,list,i;
l:= gapchars(G,p);
ct := CharacterTable(G) mod p;
irr := Irr(ct);
list := [];
while Length(l)< Size(irr)-1 do
    blist := bplan(G,p,l);
    for i in [1..Length(blist)] do
        if Length(blist[i])>1  then
            Append(list,blist[i]);
        fi;
    od;
    l := newlist(G,p,l);
od;
return list;
end;
```

buildrep The program "buildrep" takes in three parameters. The first parameter is a string corresponding to a group in the AtlasRep package. The second parameter is a number corresponding to a representation in the AtlasRep. The third parameter is the field you wish to build the representation over. "buildrep" creates the GModule representation from the generating information stored in the AtlasRep.

```
buildrep := function(g,n,field)
local gp,rep,m;
gp := AllAtlasGeneratingSetInfos(g);
rep := AtlasGenerators(gp[n].identifier);
m := GModuleByMats(rep.generators, field);
return m;
end;
```

charval The program "charval" takes in three parameters. The first parameter is a string corresponding to a group in the AtlasRep package. The second parameter is a number corresponding to a representation in the AtlasRep. The third parameter is the generators for the GModule representation. The output is the character of the representation (as a list of values).

```
charval := function(g,p,gens)
local a,c,d,traces;
a:= AtlasStraightLineProgram(g,"classes");
    if a = fail then
        a := LocalClasses(g);
            if a = fail then
                return fail;
            fi;
    fi;
c:= ResultOfStraightLineProgram(a.program,gens);
d:= Filtered(c, x -> Order(x) mod p > 0);
traces:= List(d, x -> BrauerCharacterValue(x));
return traces;
end;
```

minfield The program "minfield" takes in two parameters. The first parameter is a string corresponding to a group in the AtlasRep package. The second parameter is the (positive) characteristic. The output is the smallest field that all of the representations (for the given group and characteristic) can be realized over.

```
minfield := function(G,p)
local list, fields, ct, irr, l, i, j;
list := [];
fields := [];
ct := CharacterTable(G) mod p;
irr := Irr(ct);
l := [1..Size(irr)];
for i in l do
    Add(list, List(irr[i],x->FrobeniusCharacterValue(x,p)));
od;
for j in [1..Length(l)] do
    Add(fields, Field(list[j]));
od;
return Maximum(fields);
end;
```

namechars The program "namechars" takes in two parameters. The first parameter is a string corresponding to a group in the AtlasRep package. The second parameter is the (positive) characteristic. The output is a list of the Atlas character names (as a string) of the irreducible representations for the given group and characteristic.

```
namechars := function(G,p)
local abs, names, i;
abs := AbsIrredReps(G,p);
```

```
names := [];
for i in [1..Length(abs)] do
    if IsBound(abs[i].charactername) = true then
        Add(names, abs[i].charactername);
    else
        Add(names, IrrChar(G,p,abs[i].repnr));
    fi;
od;
return names;
end;
```

elrep The program "elrep" takes in three parameters. The first parameter is a string corresponding to a group in the AtlasRep package. The second parameter is the (positive) characteristic. The third parameter is a list of two strings. The strings are the Atlas character names of the characters corresponding to two representations you want to tensor. Note that the AtlasRep must have representations corresponding to these two characters in order for the program to function properly. The output is a list where all the odd entries are numbers corresponding to the characters of the representations that the program built and all the even entries are the generating matrices for the representation.

```
elrep := function(G,p,rl)
local field, ic, ct, irr, abs, j, names, i, r, s, rep1, rep2, tensor,
factors, k, num, brep;
field := minfield(G,p);
ic := incompletechars(G,p);
ct := CharacterTable(G) mod p;
irr := Irr(ct);
abs := AbsIrredReps(G,p);
names := [];
brep := [];
for i in [1..Length(abs)] do
    if IsBound(abs[i].charactername) = true then
        Add(names, abs[i].charactername);
    else
        Add(names, IrrChar(G,p,abs[i].repnr));
    fi;
od;
    r := Position(names, rl[1]);
    s := Position(names, rl[2]);
    rep1 := buildrep(G,abs[r].repnr,field);
    rep2 := buildrep(G,abs[s].repnr,field);
    tensor := TensorProductGModule(rep1,rep2);
    factors := MTX.CollectedFactors(tensor);
    for k in [1..Length(factors)] do
```

```
        num := Position(irr, charval(G,p,factors[k][1].generators));
        for j in ic do
            if num = j then
                Append(brep,[j,factors[k][1]]);
            fi;
        od;
    od;
return brep;
end;
```