Chapter 4

The module isomorphism problem

Let R be a finite ring and let M_1, M_2 be two finite left R-modules. We present two distinct deterministic algorithms that decide in polynomial time whether or not M_1 and M_2 are isomorphic, and if they are, exhibit an isomorphism. As by-products, we are able to determine the largest isomorphic common direct summand between two modules and the minimum number of generators of a module. By not requiring R to contain a field, avoiding computation of the Jacobson radical and not distinguishing between large and small characteristic, both algorithms constitute improvements to known results. We have not attempted to implement either of the two algorithms, but we have no reason to believe that they would not perform well in practice.

4.1 Introduction

The *module isomorphism problem* (MIP) can be formulated as follows: design a deterministic algorithm that, given a ring R and two left R-modules M_1 and M_2, decides in polynomial time whether they are isomorphic, and if yes, exhibits an isomorphism.

This problem is as fundamental as it is easily stated and has been studied extensively, due both to its intrinsic theoretical value and to its broad range of applications. As a theoretical question it is one in a long series of isomorphism problems. These are some of the most natural questions that occur in algorithmic contexts: given two objects "of the same nature", one wishes to determine if they are equal, or isomorphic. Examples of these problems include the group isomorphism problem, the graph isomorphism problem and the ring isomorphism (see [52, 53]). From a complexity point of view, these problems have a special status, namely, they are thought to be NP-intermediate, i.e. pertaining to the class consisting of problems that are known

This chapter is an extended version of the paper *The module isomorphism problem for finite rings and related results*, arXiv:1512.08365v1 ([17])

to be in NP, but are not known to be in P, or to be NP-complete. The class of NP-intermediate problems is nonempty if and only if P $\neq$ NP (see [73], Theorem 14.1). However, even under the hypothesis that P $\neq$ NP, no "natural" NP-intermediate problems are known.

The practical value of the module isomorphism problem comes from viewing it in the larger context of algorithms for finite rings. Finite rings are fundamental objects and one wishes to have as many algorithms as possible for handling them at one's disposal, that is to say, in one's computer algebra system toolbox.

A brief overview of the results related to the module isomorphism problem is included in Section 4.2. In particular, polynomial-time algorithms for the module isomorphism problem were given in [10, 15] for the case where R is a finite-dimensional algebra over a field and M_1, M_2 are finite-dimensional modules over that field. For our purposes, R will be a finite ring (not necessarily containing a field) and M_1, M_2 will be finite R-modules. We give an algorithm as described by the following theorem:

Theorem 4.1.1. *There exists a deterministic polynomial-time algorithm that, given a finite ring R and two finite R-modules M_1 and M_2, computes a maximum length R-module C that is isomorphic to a direct summand both of M_1 and of M_2. Moreover, the algorithm computes direct complements of C both in M_1 and in M_2, together with the corresponding isomorphisms.*

We establish the result in Theorem 4.1.1 by a direct generalisation of the methods given in [10], where the rings considered were finitely generated algebras over a field and the modules were finite-dimensional over that field. This approach relies on the ability of finding non-nilpotent elements in non-nilpotent ideals of the endomorphism ring $\mathrm{End}_R(M_1)$ (cf. Proposition 4.3.3). As a direct consequence of Theorem 4.1.1 we have that:

Theorem 4.1.2. *There exists a deterministic polynomial-time algorithm that, given a finite ring R and two finite R-modules M_1 and M_2, decides whether M_1 and M_2 are isomorphic, and if they are, exhibits an isomorphism.*

In Section 3, we show how the module isomorphism problem reduces to determining freeness of rank one of a module. We then give an algorithm that computes the minimum number of generators of a given finite module.

Theorem 4.1.3. *There exists a deterministic polynomial-time algorithm that, given a finite ring R and a finite R-module M, computes a set for generators of M of minimum cardinality.*

In particular, we can determine if a module is cyclic or free (by comparing cardinalities), which gives a second deterministic polynomial-time algorithm for the module isomorphism problem.

It is important to note that the algorithms given here work for any finite ring, do not distinguish between large and small characteristic and avoid computation of the

Jacobson radical, thus constituting an improvement to known results. Moreover, the algorithm in Theorem 4.1.3 is an interesting object *per se*, due to its structure and the techniques it employs. A common approach to this type of problems is to reduce to the semisimple case and then "lift" (e.g. [15, 42]). In our algorithm, we work *as if* the ring were semisimple and we have a list, $S_1, \ldots, S_t$, of candidates for the isomorphism classes of simple modules composing it. During the running of the algorithm, we allow ourselves to be contradicted in our assumption about the simplicity of the S_i, in which case we update our list, quotient the ring by an appropriate two-sided nilpotent ideal and start again. If we are not contradicted, we may still draw conclusions. In this way, there is always a side-exit available and what forces an output in polynomial time is that we cannot take the side-exit too many times.

4.2 Context

All of the results referred to in this section deal with the case where the given ring R is a finite-dimensional algebra over a field $\mathbb{F}$ and the two modules are finite-dimensional over that field.

One of the first attempts at tackling the module isomorphism problem made use of the MeatAxe algorithm due to Parker (see [74]). Holt and Rees produce an efficient randomized algorithm in [36], that in the case where $\mathbb{F}$ is finite, tests whether a module is simple, and if it is not, produces a proper submodule. This algorithm is then used to test for isomorphism between two modules of which one is known to be simple. However, their method fails in the cases where the given module has a very special structure.

The first deterministic polynomial-time algorithm for the module isomorphism problem was presented by Chistov, Ivanyos and Karpinski in [15]. For the purpose of their paper, $\mathbb{F}$ is a finite field or an algebraic number field. Their method reduces the problem to the semisimple case by computing the Jacobson radical of R and then further reduces the problem to testing cyclicity of modules. Because this method requires the computation of the Jacobson radical, it does not lend itself to generalisation in the case of general finite rings (cf. Chapter 3, Section 3.4).

The next solution to the module isomorphism problem was given by Brooksbank and Luks in [10]. Their algorithm is based on finding nilpotent endomorphisms of one of the modules and in fact does more than test for module isomorphism – it produces the largest isomorphic common direct summand between two modules, i.e. the maximal length direct summand of one of the modules, which has an isomorphic copy as a direct summand of the other. In principle, this method works for any underlying field, as long as we can perform arithmetic in it.

Another related result was presented by Ivanyos, Karpinski and Saxena in [42]. Their algorithm computes the minimum number of generators of a given module and is in particular useful for testing module cyclicity. The drawback of this method is that it distinguishes between small and large characteristic of the underlying field.

Our goal is to obtain corresponding algorithms for the case that R is a finite ring

(not necessarily containing a field) and the given modules are also finite.

4.3 MIP via non-nilpotent endomorphisms

In this section, the algorithm of Brooksbank and Luks [10] is generalised to solve the module isomorphism problem for finite rings and finite modules over that ring.

4.3.1 Finding non-nilpotent elements in non-nilpotent ideals

An ideal I of a ring R is said to be *nil* if all its elements are nilpotent. The following is a well known fact.

Proposition 4.3.1. *If R is a left-artinian ring, then a left (or right) ideal I of R is nil if and only if it is nilpotent.*

Proof. Clearly if a left (or right) ideal is nilpotent, then it is nil. For the other direction, note that any nil left (or right) ideal of R is contained in the Jacobson radical $\mathrm{J}(R)$ (see [57], Lemma 4.11). So it is enough to show that $J := \mathrm{J}(R)$ is nilpotent.

Suppose J is not nilpotent. Since R is left-artinian, the sequence of consecutive powers of J must stabilize, i.e. $\exists n \in \mathbb{Z}_{>0}$ such that $J^n = J^{2n} \neq 0$. Then there exists a left ideal L s.t. $J^n L \neq 0$. Suppose L is minimal with this property. Since $J^n L \neq 0$, there exists $a \in L$ such that $J^n a \neq 0$. Now $Ra \subseteq L$ and $J^n a \subseteq L$. Moreover, $0 \neq J^n a = J^n J^n a$ and $0 \neq J^n a \subseteq J^n Ra$, so by minimality of L, we have that $L = Ra = J^n a$. Hence $a = xa$, for some $x \in J^n$. It follows that $a = 0$ (otherwise, $1 - x$ would be both a unit and a left zero-divisor, which is not possible). This gives the desired contradiction. □

Note 4.3.2. The "only if" direction in the statement of the previous proposition is not true if we remove the artinianity condition. Consider the commutative ring $R = \mathbb{Z}[x_1, x_2, \ldots]/(x_1^2, x_2^2, \ldots)$. Then the ideal $I = (x_1, x_2, \ldots)$ is nil, but I is not nilpotent.

If R is finite, the proof of Proposition 4.3.1 can be translated into an algorithm for finding a non-nilpotent element in a non-nilpotent left ideal of R.

Proposition 4.3.3. *There exists a deterministic polynomial-time algorithm that, given a finite ring R and a left ideal I, determines whether or not I is nilpotent and if it is not, produces a non-nilpotent element lying inside it.*

Proof. Suppose R is a k-algebra, where k is a commutative ring (R will certainly be an algebra over its centre or its prime subring) and let I be a non-nilpotent left ideal of R. Since R is finite, we can find $n \in \mathbb{Z}_{>0}$ such that $I^n = I^{2n} \neq 0$. Suppose I^n is generated over k by a set A and over R by a set B, i.e.

$$I^n = \sum_{\alpha \in A} k\alpha = \sum_{\beta \in B} R\beta.$$

Then $0 \neq I^{2n} = \sum_{\substack{\alpha \in A \\ \beta \in B}} R\beta\alpha$, so there exists $b \in B$ and $a \in A$ such that $b \cdot a \neq 0$. Consider the ideals $I^n a \subseteq Ra$, where $I^n a \neq 0$ since it contains ba. If this inclusion is in fact an equality, then we can write $a = xa$, for some $x \in I^n$ and since $a \neq 0$, it must be the case that x is non-nilpotent, as required (otherwise $1 - x$ would be a unit and a left zero divisor at the same time, which is impossible). Suppose now that the inclusion is strict. Then there exists $c \in I^n a$ such that $I^n c \neq 0$ (otherwise $I^n I^n a = I^n a = 0$, which is a contradiction, since $0 \neq ba \in I^n a$). Moreover, we will be able to find such a c among k-generators of $I^n a$, otherwise, for all $\alpha \in A$, we would have that $I^n \alpha a = 0$, and so $0 = \sum_{\alpha \in A} k I^n \alpha a = I^{2n} a = I^n a \neq 0$, which is a contradiction.

We have now produced a smaller ideal $Ra \supsetneq I^n a \supset Rc$ such that $I^n c \neq 0$. We replace a by c and keep going. This process must terminate in polynomial time, since $\text{length}(Ra) \leq \text{length}({}_R R) \leq \log_2(|R|)$, and so the length of the descending chain of ideals obtained, and hence the number of steps performed by the algorithm, is bounded above by $\log_2(|R|)$. We also note that I is nilpotent if and only if $I^{\text{length}({}_R R)} = 0$, so it is possible to test if I is nilpotent in polynomial time. □

4.3.2 Splitters

Let R be an algebra over a commutative artinian ring k such that R is finitely generated as a k-module. Let M_1 and M_2 be two R-modules of finite length over k. Note that $\text{Hom}_R(M_1, M_2)$ is a left k-module via the action

$$\begin{aligned} k \times \text{Hom}_R(M_1, M_2) &\to \text{Hom}_R(M_1, M_2) \\ (r, f) &\mapsto (m \mapsto f(mr)), \end{aligned}$$

and a right $\text{End}_R(M_1)$-module via the action

$$\begin{aligned} \text{Hom}_R(M_1, M_2) \times \text{End}_R(M_1) &\to \text{Hom}_R(M_1, M_2) \\ (f, g) &\mapsto f \circ g. \end{aligned}$$

Moreover, $\text{End}_R(M_1)$ is left (and right) artinian. In particular, all its nil ideals are nilpotent and *vice versa*.

Following [10], we make the following definition:

Definition 4.3.4. *Let $f \in \text{Hom}_R(M_1, M_2)$. A decomposition $M_1 = N_1 \oplus K_1$, for $N_1, K_1 \leq M_1$, is called an* f-decomposition *if $N_1 \neq 0$, $\ker(f) \leq K_1$ and the image of N_1 under f, which we denote by fN_1, is a direct summand of M_2. If M_1 has an f-decomposition, we say that f is a* splitter.

Note that the condition $\ker(f) \leq K_1$ implies that $N_1 \cong fN_1 = N_2$, so f induces an isomorphism $N_1 \cong N_2$.

The following proposition and its proof, together with Proposition 4.3.3, allow us to algorithmically decide if a given homomorphism f is a splitter, and if it is, to produce an f-decomposition.

Proposition 4.3.5. *Let $f \in \mathrm{Hom}_R(M_1, M_2)$. Then f is a splitter if and only if there exists $g \in \mathrm{Hom}_R(M_2, M_1)$ such that gf is not nilpotent.*

Proof. The proof of this proposition is the same as the one given in [10], Lemma 3.3, which treats the case when R is a finitely generated algebra over a field and M_1, M_2 are finite-dimensional over that field. For completeness we include the proof of the "if" statement here.

Suppose $g \in \mathrm{Hom}_R(M_2, M_1)$ is such that gf is not nilpotent. Let $s = gf$ and $t = fg$. Since M_1 and M_2 have finite length over k (so are both artinian and noetherian over R), we can apply Fitting's Lemma (see Proposition 1.3.12) to say that $M_1 = \ker(s^d) \oplus \mathrm{im}(s^d)$, $M_2 = \ker(t^d) \oplus \mathrm{im}(t^d)$, for $d = \max\{\mathrm{length}_R(M_1), \mathrm{length}_R(M_2)\}$ and the restriction of t to $\mathrm{im}(t^d)$ is an automorphism. We have that $s^d M_1 \neq 0$ since s is not nilpotent, $\ker(f) \leq \ker(s^d)$ by definition of s and

$$f(s^d M_1) = t^d(fM_1) \subseteq t^d M_2 = fg(t^d M_2) = fs^d(gM_2) \subseteq f(s^d M_1),$$

so $f\,\mathrm{im}(s^d) = f(s^d M_1) = t^d M_2 = \mathrm{im}(t^d)$, which is what is required for f to be a splitter. □

Suppose now that we are given $f \in \mathrm{Hom}_R(M_1, M_2)$ and we wish to know whether it is a splitter. To do this, we first compute a set C of k-generators of $\mathrm{Hom}_R(M_2, M_1)$. Now consider the left ideal I of $\mathrm{End}_R(M_1)$ generated by the set $Cf = \{cf \mid c \in C\}$. If I is nilpotent (which we can determine by Proposition 4.3.3), then since $\mathrm{Hom}_R(M_2, M_1)f = \mathrm{span}_k(Cf) = I$, we will not be able to find a non-nilpotent element of the form gf, so f cannot be a splitter. Otherwise the algorithm of Proposition 4.3.3 will produce some $g \in \mathrm{Hom}_R(M_2, M_1)$ witnessing that f is a splitter and we can produce an f-decomposition by Proposition 4.3.5.

We now have a way of identifying splitters. But we would not like to have to look for them over all homomorphisms. The following proposition tells us that we can restrict our attention to a considerably smaller set.

Proposition 4.3.6. *If a splitter exists, then there exists a splitter in any set of k-module generators of $\mathrm{Hom}_R(M_1, M_2)$ and in any set of $\mathrm{End}_R(M_1)$-generators of $\mathrm{Hom}_R(M_1, M_2)$.*

Proof. To see that a set of $\mathrm{End}_R(M_1)$-module generators is enough, note that

$$f \text{ is not a splitter} \iff \mathrm{Hom}_R(M_2, M_1)f \subseteq \mathrm{J}(\mathrm{End}_R(M_1)).$$

Let B be a set of $\mathrm{End}_R(M_1)$-module generators of $\mathrm{Hom}_R(M_1, M_2)$. Suppose that B does not contain any splitters. Then

$$\begin{aligned}\mathrm{Hom}_R(M_2, M_1)\,\mathrm{Hom}_R(M_1, M_2) &= \sum_{b \in B} \mathrm{Hom}_R(M_2, M_1) b\,\mathrm{End}_R(M_1) \\ &\subseteq \mathrm{J}(\mathrm{End}_R(M_1)),\end{aligned}$$

and therefore $\mathrm{Hom}_R(M_1, M_2)$ cannot contain a splitter.

Finally, note that any set of left-k-module generators is also a set of right-$\mathrm{End}_R(M_1)$-module generators. □

Putting these results together, we can construct an algorithm satisfying the requirements of Theorem 4.1.1 as follows.

Proof of Theorem 4.1.1: We view R as an algebra over its prime subring k. Let C be an auxiliary variable that at the end of the algorithm will become equal to the desired maximal length R-module that is isomorphic to a direct summand both of M_1 and of M_2. At the beginning of the algorithm, we put C equal to zero. Similarly, we define an auxiliary variable $f \in \mathrm{Hom}_R(M_1, M_2)$ and initialise it at zero.

We compute B, a set of k-generators of $\mathrm{Hom}_R(M_1, M_2)$ (or a set of $\mathrm{End}_R(M_1)$-module generators thereof). For each element of B, we test if it is a splitter: by Proposition 4.3.6, if a splitter exists, we will find one inside B. Finding a splitter $b \in B$ also gives us a homomorphism $c \in \mathrm{Hom}_R(M_2, M_1)$ such that cb is not nilpotent, and a decomposition $M_1 = N_1 \oplus K_1$ and $M_2 = N_2 \oplus K_2$, where $N_1 = \mathrm{im}((cb)^d)$, $K_1 = \ker((cb)^d)$, $N_2 = \mathrm{im}((bc)^d)$, $K_2 = \ker((bc)^d)$, and $d = \max\{\mathrm{length}_R(M_1), \mathrm{length}_R(M_2)\}$ (see Proposition 4.3.5). We make the following replacements: $C := C \oplus N_1$ and $f := f \oplus$ (the restriction of b to N_1), $M_1 := K_1$, $M_2 := K_2$, $B :=$ (set of k-module generators of the new $\mathrm{Hom}_R(M_1, M_2)$), and we repeat the process. Note that we are all the time assuming the Krull-Remak-Schmidt Theorem (see Theorem 1.3.13), which ensures existence and uniqueness up to isomorphism of the direct summands of M_1 and M_2.

The algorithm produces the following data: the R-module C and the R-module homomorphism f such that $f : C \xrightarrow{\sim} f(C)$. Put $D := f(C)$. The algorithm also produces injections of C and D into M_1 and M_2 respectively, that define them as submodules. By splitting these injections (see Proposition 2.5.1), we can recover the direct complements of C and D in M_1 and M_2 respectively, together with the corresponding isomorphisms. □

4.4 MIP via an approximation of the Jacobson radical

In this section, another solution to the module isomorphism problem for finite rings is presented. We start with a problem reduction, showing that it is enough to be able to test if a module is free of rank 1. We then construct an algorithm that, given a finite ring and a finite module over that ring, computes a set of generators of minimum cardinality for that module.

4.4.1 Problem reduction

Let R be a finite ring. We observe that determining whether two finite R-modules M_1 and M_2 are isomorphic reduces to determining if a module is free of rank one. Let $E := \mathrm{End}_R(M_1)$ and $K := \mathrm{Hom}_R(M_2, M_1)$. Note that K is a left E-module.

Proposition 4.4.1. *Let R, M_1, M_2, E, K be as above. The following are equivalent:*

(i) *$M_1 \cong M_2$ as R-modules.*

(ii) *$E \cong K$ as E-modules and the image of 1_E in K under any such isomorphism is an isomorphism between M_1 and M_2.*
(iii) *There exists an E-module isomorphism $\phi : E \to K$ such that $\phi(1_E)$ is an isomorphism between M_1 and M_2.*

Proof. The implications (ii)⇒(iii) and (iii)⇒(i) are immediate. For (i)⇒(ii), suppose $f : M_2 \xrightarrow{\sim} M_1$. Then $E \cong K$, so let $\phi : E \xrightarrow{\sim} K$ be any such isomorphism. Let $\lambda := \phi(1)$. Then there exists a unique $\epsilon \in E$ such that

$$f = \phi(\epsilon) = \phi(\epsilon \cdot 1) = \epsilon\phi(1) = \epsilon\lambda,$$

where the third equality follows by E-linearity of ϕ. Since f is injective, so must λ be. Moreover, since M_1 and M_2 have the same length, λ must also be surjective. □

Hence an algorithm that can establish freeness of rank one of a module provides a solution to the module isomorphism problem in the following way. Given M_1 and M_2, we might first test whether they have the same cardinality, otherwise it is pointless to proceed. If the cardinalities do agree, we compute E and K, and test if $E \cong K$ as E-modules (in the case that $E \cong K$, we are assuming this test will also produce an isomorphism). If this is not the case, we conclude that $M_1 \ncong M_2$. Otherwise, suppose we have found an isomorphism $\psi : E \xrightarrow{\sim} K$. Set $\lambda := \psi(1_E)$. Then by Proposition 4.4.1, we have that $M_1 \cong M_2$ if and only if λ is an isomorphism.

For the remainder of this section we will concentrate on the task of computing the minimum number of generators of a given module. In particular, this will allow us to test for cyclicity and, by comparing cardinalities, for freeness of rank 1.

4.4.2 Computing minimum number of generators

Let R be a left-artinian ring. Suppose, along with R, we are given a collection $S_1, \ldots, S_t$ of nonzero left R-modules of finite length. Ideally, we would like this collection to be a set of representatives for the isomorphism classes of simple R-modules. However, for now, we only require that each simple R-module occurs in at least one of the S_i, i.e. it occurs as a quotient in its composition series. We can take, for example, $t = 1$ and $S_1 = {}_RR$.

Let

$$\mathfrak{a} = \bigcap_{i=1}^{t} \operatorname{ann}_R(S_i),$$

where $\operatorname{ann}_R(S_i) = \ker(R \to \operatorname{End}_{\mathbb{Z}}(S_i))$. Again, ideally we would like $\mathfrak{a}$ to be the Jacobson radical, $\mathrm{J}(R)$. In reality, we only have one inclusion, namely $\mathfrak{a} \subseteq \mathrm{J}(R)$, since an element of $\mathfrak{a}$ will kill all the S_i's and so will kill all submodules, quotients and submodules of quotients of the S_i. Since every simple R-module occurs in at least one of the S_i, we have that $\mathfrak{a}$ kills all simple R-modules, which is equivalent to being inside the Jacobson radical. Furthermore, since R is left-artinian, $\mathrm{J}(R)$ is nilpotent and hence $\mathfrak{a}$ is nilpotent.

We will construct an algorithm that, given R and a collection of S_i, either "improves" the sequence of S_i or computes $\mathfrak{a}$ and an isomorphism of R-modules

$$R/\mathfrak{a} \xrightarrow{\sim} \bigoplus_{i=1}^{t} S_i^{a_i}, \tag{4.1}$$

for suitable $a_i \in \mathbb{Z}_{\geq 0}$. Similarly, we construct an algorithm that, when also given a finitely generated R-module M, either "improves" the sequence of S_i or computes an isomorphism of R-modules

$$M/\mathfrak{a}M \xrightarrow{\sim} \bigoplus_{i=1}^{t} S_i^{c_i},$$

for suitable $c_i \in \mathbb{Z}_{\geq 0}$. Consider the quantity $l(S_i)_{i=1}^t = \sum_{i=1}^t (2\,\mathrm{length}(S_i) - 1) \in \mathbb{Z}_{\geq 0}$. An "improvement" in the sequence is measured by a decrease in $l(S_i)_{i=1}^t$ and occurs either when we remove one of the S_i from the list (either because the list already contains an isomorphic copy of it or because it is not needed) or when we discover a nonzero proper submodule T of one of the S_i which witnesses nonsimplicity of S_i and which we use to replace S_i by S_i/T and T, that now stand a better chance of being simple. Note that the factor 2 in the expression of $l(S_i)_{i=1}^t$ ensures it decreases even when we remove an S_i from the list.

Let us first write out the details of the routine that finds an isomorphism as in (4.1). We will call this routine UPDATE and within the main algorithm we will call it whenever we have improved on our sequence of S_i's and we want to update our $\mathfrak{a}$, the sequence of a_i's and the isomorphism $R/\mathfrak{a} \xrightarrow{\sim} \bigoplus_{i=1}^t S_i^{a_i}$.

Let $\mathfrak{S} = \{(S_i)_{i=1}^t \mid t \in \mathbb{Z}_{\geq 0}$, each S_i is a nonzero finite length R-module and each simple R-module occurs as a factor in the composition series of at least one $S_i\}$.

Proposition 4.4.2. *There exists a deterministic polynomial-time algorithm that takes as input a finite ring R and a collection of modules $(S_i)_{i=1}^t \in \mathfrak{S}$ and outputs a sequence of integers $a_1, \ldots, a_t \in \mathbb{Z}_{>0}$, a two-sided nilpotent ideal $\mathfrak{a}'$ of R and an isomorphism $\varphi : R/\mathfrak{a}' \xrightarrow{\sim} \bigoplus_{i=1}^{t'} (S_i')^{a_i}$, where $t' \in \mathbb{Z}_{\geq 0}$, $\mathfrak{a}' = \bigcap_{i=1}^{t'} \mathrm{ann}_R(S_i')$ and $(S_i')_{i=1}^{t'} \in \mathfrak{S}$ is such that $l(S_i')_{i=1}^{t'} \leq l(S_i)_{i=1}^t$.*

Proof. Let $\mathfrak{b}$ be a left ideal of R which we will use as an intermediate variable that at the end of the algorithm will become equal to the desired $\mathfrak{a}'$. We start off by setting $\mathfrak{b} = R$, $t' = t$, $a_i = 0$ and $S_i' = S_i$ for $1 \leq i \leq t$, and $\varphi = 0$. Throughout the algorithm $\varphi : R/\mathfrak{b} \xrightarrow{\sim} \bigoplus_{i=1}^{t'} (S_i')^{a_i}$ and $\bigcap_{i=1}^{t'} \mathrm{ann}_R(S_i') \subseteq \mathfrak{b}$ will be invariant. If for all i we have $\mathfrak{b}S_i' = 0$, then we are done. Otherwise, we choose $1 \leq h \leq t'$ and $s \in S_h'$ such that $\mathfrak{b}s \neq 0$. We define

$$\psi : R \to S_h' \oplus \bigoplus_{i=1}^{t'} (S_i')^{a_i}, \quad \psi(r) = (rs, \varphi(r)).$$

Then $\ker(\psi) = \operatorname{ann}_R(s) \cap \mathfrak{b} \subsetneq \mathfrak{b}$, since $\mathfrak{b}$ did not annihilate s. Let $\overline{\psi}$ be the map induced by ψ on $R/\ker(\psi)$. Then

$$\overline{\psi} \text{ isomorphism} \iff \psi \text{ surjective} \iff S'_h \oplus \{0\} \subseteq \operatorname{im}(\psi) \iff \mathfrak{b}s = S'_h.$$

This comes as a confirmation of our intuition: we are treating the S'_i as if they were simple, so if $0 \neq \mathfrak{b}s \leq S'_h$, then we would want $\mathfrak{b}s$ to be the whole of S'_h.

If ψ is surjective, we make the following replacements: $a_h := a_h + 1$, $\varphi := \psi$ and $\mathfrak{b} := \ker \psi$. Note that now $\mathfrak{b}$ has smaller length than before.

If ψ is not surjective, we replace S'_h by $S'_h/\mathfrak{b}s$, t' by $t'+1$ and put $S'_{t'+1} := \mathfrak{b}s$ and $a_{t'+1} := 0$. In addition, we replace φ by the composition $\pi \circ \varphi$, where π is the canonical map $\pi : \bigoplus_{i=1}^{t'} [\mathrm{old}(S'_i)^{\mathrm{old}\, a_i}] \to \bigoplus_{i=1}^{t'+1} [\mathrm{new}(S'_i)^{\mathrm{new}\, a_i}]$ and we replace $\mathfrak{b}$ by the kernel of our new φ. Note that the new $\mathfrak{b}$ will contain the old $\mathfrak{b}$, but the improvement is now sitting in the new S'_i.

Consider the quantity $l(S'_i)_{i=1}^{t'} = \sum_{i=1}^{t'} (2\,\mathrm{length}(S'_i) - 1)$. Then $0 \leq t' \leq l(S'_i)_{i=1}^{t'}$, since each S'_i has length at least 1. Also, $l(S'_i)_{i=1}^{t'}$ is bounded above by its initial value. At each iteration of the algorithm, we either see a decrease in the value of $l(S'_i)_{i=1}^{t'}$, when we improve our sequence, or we see a decrease in the length of $\mathfrak{b}$, whose length is initially equal to $\mathrm{length}(R)$. Since $\mathrm{length}(R) \leq \log_2(|R|)$ (recall that R was finite), the algorithm runs in polynomial time. □

We now turn to the main algorithm:

Theorem 4.4.3. *There exists a deterministic polynomial-time algorithm that, given a finite ring R and a finite R-module M, computes a set of generators for M of minimum cardinality.*

Proof. We begin by running the UPDATE algorithm presented in Proposition 4.4.2 with input $t = 1$ and $S_1 = {}_R R$. Let $X = \{x_1, \ldots, x_d\}$ be a set of R-generators of M. Then $\overline{X} = \{x_i + \mathfrak{a}M \mid 1 \leq i \leq d\}$ generates $M/\mathfrak{a}M$ over $R/\mathfrak{a}$. This gives a surjective homomorphism $(R/\mathfrak{a})^d \cong \bigoplus_{i=1}^{t} S_i^{a_i d} \twoheadrightarrow M/\mathfrak{a}M$. Relabel the S_i to get a map $\bigoplus_{i \in I} S_i \twoheadrightarrow M/\mathfrak{a}M$. We would like to find a subset $J \subseteq I$ for which this map becomes an isomorphism. In this process, the standard proof of its existence in the case where the S_i are simple (see [60], Chapter XVII, Lemma 2.1) may produce a witness of nonsimplicity of some S_i, in which case we refine the sequence and call the UPDATE subroutine to start over. In the end, we will have produced an isomorphism $\bigoplus_{i=1}^{t} S_i^{c_i} \xrightarrow{\sim} M/\mathfrak{a}M$, for some $c_i \in \mathbb{Z}_{>0}$ (note that the ideal $\mathfrak{a}$ may now be different from the one we started with).

Let $n = \max_i \{\lceil \frac{c_i}{a_i} \rceil\}$. If for all $i \neq h$ we have $\mathrm{Hom}_R(S_i, S_h) = 0$, then there is a surjective map $(R/\mathfrak{a})^n \twoheadrightarrow M/\mathfrak{a}M$ and n is minimal with this property. Since $\mathfrak{a} \subseteq \mathrm{J}(R)$, we can lift this to a map $R^n \twoheadrightarrow M$ and produce n generators of M. If, however, for some $i \neq h$ we have that $\mathrm{Hom}_R(S_i, S_h)$ contains a nonzero element f, then we can once again refine our sequence (either using $\ker(f) \neq 0$ or $\operatorname{im}(f) \neq S_h$ or removing one of S_i or S_h from the list if they are isomorphic) and start over by calling the UPDATE routine on our newly improved sequence.

To establish the running time, consider again the quantity

$$l(S_i)_{i=1}^{t} = \sum_{i=1}^{t} (2\,\mathrm{length}(S_i) - 1).$$

At each iteration of the algorithm, we either produce an output or we improve on our sequence, which results in a decrease in $l(S_i)_{i=1}^{t}$. □

Note 4.4.4. As we have described the algorithm in Theorem 4.4.3, after we have improved our sequence, calling the UPDATE subroutine overwrites our so far acquired knowledge about $\mathfrak{a}$. There is possibly a way of saving some of this information by running a modified version of the UPDATE routine, which would thus make the main algorithm slightly more efficient. However, being more precise about this here would obscure the idea of the algorithm.

4.5 Remark on implementation and performance

Since our goal was to place the module isomorphism problem in the complexity class P, we have not been concerned with calculating running time exponents or performing a detailed complexity analysis. A more careful organisation of the subroutines of the algorithms presented, or indeed considering randomised variations thereof may yield better running times, but such endeavours are beyond our scope at this moment.