All centralising monoids on the set $\{0, 1, 2\}$, including their witnesses

Mike Behrisch*†‡ Leon Renkin§

15th February 2023

This data set provides supplementary material for the description of witnesses for all 192 centralising monoids [6, 7] on the three-element set $A = \{0, 1, 2\}$ as carried out in [1], cp. also [8].

Centralising monoids are the closed sets on one side of the Galois connection between finitary functions in $\mathcal{O}_A = \bigcup_{n \in \mathbb{N}_+} A^{A^n}$ and $\mathcal{O}_A^{(1)} = A^A$ induced by the relation of *commutation*: we say that $f\colon A^n \to A$ for some $n \in \mathbb{N}_+$ commutes with some unary function $s\colon A \to A$ if and only if $s\colon \langle A; f \rangle \to \langle A; f \rangle$ is an endomorphism of the algebra $\langle A; f \rangle$, i.e., if $s \in \operatorname{End}(\langle A; f \rangle)$, meaning in more detail that $s(f(x_1, \dots, x_n)) = f(s(x_1), \dots, s(x_n))$ for all $x_1, \dots, x_n \in A$. We denote this by $f \perp s$ or by $f \in \{s\}^*$ or $s \in \{f\}^*$ (more precisely $s \in \{f\}^{*(1)}$). A centralising monoid is a set of the form $F^{*(1)} = \left\{ s \in \mathcal{O}_A^{(1)} \;\middle|\; \forall f \in F\colon f \perp s \right\}$ for some $F \subseteq \mathcal{O}_A$. Note that $F^{*(1)} = \bigcap_{f \in F} \{f\}^{*(1)}$. Any set $F \subseteq \mathcal{O}_A$ describing the monoid $M = F^{*(1)}$ is called a *witness* of the centralising monoid; witnesses are not unique.

Machida and Rosenberg in [6] presented a strategy to determine all centralising monoids on $A = \{0, 1, 2\}$ and included a list of the resulting 192 monoids in [7], which is not easily, for example, by a computer, verifiable since there are no witnesses given. In [8, 1], we rephrased the description of all centralising monoids using formal concept analysis [4, 3]. Namely, the set of centralising monoids can be seen as the set of intents of the formal context $\mathbb{K}_1 = \left(\mathcal{O}_A, \mathcal{O}_A^{(1)}, \perp\right)$. In [8, 1] we demonstrate how to find a finite subcontext $\mathbb{K}'$ of $\mathbb{K}_1$ having only 183 objects, but the same set of intents as $\mathbb{K}_1$. The subcontext $\mathbb{K}'$ can be further reduced to a subcontext $\mathbb{K}''$ with only 51 objects, which is still able to describe all centralising monoids on $\{0, 1, 2\}$. The advantage of this approach is twofold: we may apply well-known algorithms (cf. [3, Chapter 2] and [4, Chapter 2], see also the implementations [2, 5]) to automatically enumerate all concepts (and thus in particular all intents) of $\mathbb{K}'$; furthermore, the corresponding extents of the concepts provide witnesses that were missing in [7]. In this way we are able to confirm the correctness of the 192 monoids derived by Machida and Rosenberg

*Institute of Discrete Mathematics and Geometry, TU Wien, Wien, Austria https://orcid.org/0000-0003-0050-8085

†Institut für Algebra, Johannes Kepler Universität Linz, Austria

‡Partial support by FWF grant P33878 is gratefully acknowledged.

§Department of Mathematics, ETH Zürich, Zürich, Switzerland https://orcid.org/0000-0003-3343-7869

and of their 48 conjugacy types; furthermore, our list of witnesses allows for a quick verification that the 192 monoids we present are indeed centralising monoids. A more detailed discussion of our method can be found in [8] and [1]. Here we only comment on certain aspects that are needed to make sense of the files in this data set (in conjunction with consulting [1] or [8] for the mathematical background); moreover we show how our data shows the correctness of the results of [6, 7].

1 Formal contexts

1.1 Object clarification

The context $\mathbb{K}_1$ contains many duplicate rows, that is, functions $f_1 \neq f_2 \in \mathcal{O}_A$ with $\{f_1\}^{*(1)} = \{f_2\}^{*(1)}$, and it is clearly unnecessary to store duplicate rows in order to give the lattice of intents of $\mathbb{K}_1$. Keeping only one representative for each duplicate row is called *object clarification* in the language of formal concept analysis. As $\mathbb{K}_1$ contains an infinite number of rows, we have independently written two programmes that construct an object clarified subcontext $\mathbb{K}' = \left(W_A, \mathcal{O}_A^{(1)}, \perp\right)$ of $\mathbb{K}_1$, where $|W_A| = 183$ (files `comm-ternary-unary-3.cxt`[1] and `context_of_commutation.cxt`[2]). The contexts $\mathbb{K}'$ produced by the two programmes contain different sets of witnesses, as the codes differ in the order in which they enumerate certain functions as objects and thus the results of the object clarification process are distinct. However, their object intents (and thus the set of all intents, that is, of centralising monoids) is identical in both cases.

1.2 Encoding of operations used as objects or attributes

In order to understand the objects and attributes in the two mentioned context files representing variants of $\mathbb{K}'$, we have to define the coding of operations used in these files. The operations appearing in the files are operations of arity at most three, that is, functions of the form $f\colon A^n \to A$ where $A = \{0,1,2\}$ and $n \in \mathbb{N}$, $1 \leq n \leq 3$. In the context files, these functions are represented by finite strings of digits from $A = \{0,1,2\}$ that stand for the value tables of the operations. In order to identify a function with such a linearised value table, we have to specify the order of the arguments used in the string of digits. We represent functions $f\colon A \to A$ as the string $(f(0), f(1), f(2)) \in A^3$ (in the context files, no commas or parentheses are used since each letter is a single symbol, for example, the identity operation id_3 is represented by the string `012`). For binary operations $f\colon A^2 \to A$ there are two natural representations, *row-wise* and *column-wise* coding. Row-wise coding means that f is stored as the string $(f(0,0), f(0,1), f(0,2), f(1,0), f(1,1), f(1,2), f(2,0), f(2,1), f(2,2)) \in A^9$, in other words, in row-wise coding a string $(a_0, \dots, a_8) \in A^9$ represents the function

[1]We have also added the c++-file `commutation.cpp` with which this data was produced from the command `compute_ternary_unary("comm-ternary-unary-3.cxt");`. Let us note that the same programme is also capable of outputting the contexts $\left(\mathcal{O}_A^{(1)}, \mathcal{O}_A^{(1)}, \perp\right)$ via `write_context_unary_unary("comm-unary-unary-3.cxt");` and of $\left(\mathcal{O}_A^{(2)}, \mathcal{O}_A^{(1)}, \perp\right)$ via `write_context_binary_unary("comm-bin-unary-3.cxt");` if these commands are uncommented.

[2]This file has been produced by the c++-file `context_of_commutation_code.cpp`, written by Leon Renkin, cf. [8, Section 7.1].

with value table

$$\begin{array}{c|ccc} x\backslash y & 0 & 1 & 2 \\ \hline 0 & a_0 & a_1 & a_2 \\ 1 & a_3 & a_4 & a_5 \\ 2 & a_6 & a_7 & a_8 \end{array}, \quad \text{where } f(x,y) \text{ appears in row } x\text{, column } y.$$

On the other hand, column-wise coding means that $f\colon A^2 \to A$ is stored as the string $(f(0,0), f(1,0), f(2,0), f(0,1), f(1,1), f(2,1), f(0,2), f(1,2), f(2,2)) \in A^9$, in other words, in column-wise coding a string $(a_0, \ldots, a_8) \in A^9$ represents the function with the value table

$$\begin{array}{c|ccc} x\backslash y & 0 & 1 & 2 \\ \hline 0 & a_0 & a_3 & a_6 \\ 1 & a_1 & a_4 & a_7 \\ 2 & a_2 & a_5 & a_8 \end{array}, \quad \text{where } f(x,y) \text{ appears in row } x\text{, column } y.$$

If one were to accidentally confuse the two encodings, i.e., to construct the opposite table out of a row-wise encoded function by mistaking it as a column-wise coding, not much harm would arise, as both function have identical properties with respect to commutation, that is, they commute with the same functions and hence define the same centraliser.

For ternary operations $f\colon A^3 \to A$ there are again two major ways of encoding them, coming from listing the arguments lexicographically such that either left-most letters take priority or right-most letters take priority. For brevity, we will call these possibilities again row-wise and column-wise coding. That is, in row-wise coding, a function $f\colon A^3 \to A$ is represented as $(f(0,0,0), f(0,0,1), f(0,0,2), f(0,1,0), f(0,1,1), f(0,1,2), \ldots, f(2,2,2)) \in A^{26}$, i.e., $f(x,y,z)$ appears at the index $3^2x+3y+z$ of the string, while in column-wise coding, $(f(0,0,0), f(1,0,0), f(2,0,0), f(0,1,0), f(1,1,0), f(2,1,0), \ldots, f(2,2,2))$ in A^{26} is used to represent f, i.e., $f(x,y,z)$ appears at the index $x + 3y + 3^2z$ of the string.

In the programme `commutation.cpp` and the files `comm-unary-unary-3.cxt`, `comm-bin-unary-3.cxt`, `comm-ternary-unary-3.cxt` produced by it, always row-wise encoding is used. In the programme `context_of_commutation_code.cpp`, which produces the file `context_of_commutation.cxt`, column-wise encoding is used for the binary and ternary operations appearing as objects.

For a more concise representation of the closures, the objects and attributes of the context `context_of_commutation.cxt` have been consecutively numbered in the form $h_0, \ldots, h_{155}, f_{156}, \ldots, f_{182}$ (for the objects) and $s_0 = \texttt{000}, \ldots, s_{26} = \texttt{222}$ (for the attributes). The precise correspondence between the string representations of the functions and the newly introduced identifiers is given in [8, Section 7.2], the corresponding context with renamed objects and attributes is shown in [8, Section 7.3] and has been added to this data set as file `context_of_commutation_names.cxt`.

1.3 Object reduction

Another simplification method that still keeps the set of intents intact is *object reduction* [3, Section 1.4.3, p. 27 et seqq.], also called *row reduction.* In an object reduced subcontext $\mathbb{K}''$ of $\mathbb{K}'$ only 51 objects from the 183 functions in W_A remain; a representation of $\mathbb{K}''$ is given in [1, Table 1], cf. `context_of_commutation_names_obj_red.cxt`. Applying also the dual operation to the columns, one obtains a fully reduced context $\mathbb{K}'''$ from $\mathbb{K}'$ that has 51 objects (rows) and 25 of the 27 attributes (columns) of $\mathbb{K}_1$ or $\mathbb{K}'$. The context $\mathbb{K}'''$ has an isomorphic lattice of intents, but obviously not the same set of intents as $\mathbb{K}'$ or $\mathbb{K}$; it is the standard context belonging to $\mathbb{K}_1$ and is presented in [8, Section 7.4], cf. `context_of_commutation_names_red.cxt`. The reduction process has been carried out with readily available formal concept analysis software, such as ConExp[3] or Conexp-clj (cf. [2, 5]). Depending on which file representation of $\mathbb{K}'$ we start with, we obtain different object reduced ($\mathbb{K}''$) or fully reduced ($\mathbb{K}'''$) context files:

$\mathbb{K}'$	object reduced $\mathbb{K}''$	standard context $\mathbb{K}'''$
`comm-ternary-unary-3.cxt`	`comm-ternary-unary-obj-red-3.cxt`	`comm-ternary-unary-red-3.cxt`
`context_of_commutation.cxt`	`context_of_commutation_obj_red.cxt`	`context_of_commutation_red.cxt`
`context_of_commutation_names.cxt`	`context_of_commutation_names_obj_red.cxt`	`context_of_commutation_names_red.cxt`

For further computation of lists of concepts, intents (cf. [8, Section 7.5]), and intents up to conjugacy (cf. [1]) we have proceeded from the context $\mathbb{K}'$ as given in `context_of_commutation.cxt` or from $\mathbb{K}''$ as given in `context_of_commutation_names_obj_red.cxt`.

1.4 Context file format

All formal contexts in this data set are given in Burmeister format, which can be read by standard formal concept analysis software like ConExp or Conexp-clj [2] and complies with the following specification:

- A line with the letter `B` (for Burmeister).
- A blank line
- A line with the integer number m of objects.
- A line with the integer number n of attributes.
- A blank line
- m lines with labels for the objects.
- n lines with labels for the attributes.
- m lines representing the context, that is, the incidence relation between objects and attributes. The i-th of these lines corresponds to the i-th object g_i and contains a string of n characters `X` or `.`, `X` standing for g_i being in relation with the corresponding attribute and `.` for the opposite. This is an encoding of the characteristic tuple of the object intent given by g_i where `X` represents 1 and `.` represents 0.

[3] `http://sourceforge.net/projects/conexp`

Example. *The following listing is in Burmeister format:*

```
B

3
2

f
g
h
P
Q
X.
.X
XX
```

It represents the Galois connection between $\{f, g, h\}$ and $\{P, Q\}$, where f has property P but not Q, g has Q but not P, and h has both. As a binary incidence relation $I \subseteq \{f, g, h\} \times \{P, Q\}$ this would be represented as $I = \{(f, P), (g, Q), (h, P), (h, Q)\}$, which commonly is summarised in a tabular form (note the resemblance to the last three lines of the listing) as

$$\begin{array}{c|cc} & P & Q \\ \hline f & \times & \\ g & & \times \\ h & \times & \times \end{array} .$$

2 Centralising monoids

One of the main goals of [8, 1] was to derive a finite context $\mathbb{K}'$, the intents of which are all centralising monoids on $\{0, 1, 2\}$. This has been achieved using the programme `context_of_commutation_code.cpp` in the files `context_of_commutation.cxt` and `context_of_commutation_names_obj_red.cxt`. The intents of these contexts are the centralising monoids, the corresponding extents may serve as witnesses. Concepts and intents were computed using [2], see also [5]. The number of concepts/intents computed agrees with the figure 192 reported in [6, 7]. In Section 5 we shall describe that we were also able to confirm the correctness of the lists of 192 monoids given in [7].

The following files were obtained from $\mathbb{K}'$ as represented in `context_of_commutation.cxt` with the help of [2][4] and some manual change of the formatting (line breaks, separating symbols, etc.): `concepts.txt`, the 192 concepts of $\mathbb{K}'$, i.e., the centralising monoids together with witnesses; `intents.txt`, the 192 intents of $\mathbb{K}'$; `intents_numbers.cppinput`, the 192 intents of $\mathbb{K}'$ to be used as an input for the programme `reduce_monoids_conj.cpp` (here the attribute labels `abc` for unary operations $s \in \mathcal{O}_A^{(1)}$, where $s(0) = a$, $s(1) = b$, $s(2) = c$, have been replaced by an integer in $\{0, \ldots, 26\}$ computed as $3^2 \cdot s(0) + 3 \cdot s(1) + s(2) = 9a + 3b + c$, and line endings have been marked by the value 30; `intents_numbers.cppinput` was obtained from `intents.txt` with the help of the GNU `sed` command (version 4.2.2) via `sed -f int2numb.sed intents.txt > intents_numbers.cppinput`).

[4]Use the commands `(def cxt (read-context "path_to/context_of_commutation.cxt"))` followed by `(concepts cxt)` or `(intents cxt)`.

For the presentation of intents (monoids) and witnesses in [1, 8], a different notation (labelling) of the objects and attributes was needed. As explained in Section 1.2 on encoding of operations in the formal contexts, the file `context_of_commutation.cxt` was changed to `context_of_commutation_names.cxt` for this purpose, cf. also [8, Sections 7.2, 7.3] for more details. With the help of ConExp/Conexp-clj this file was object reduced, resulting in the context file `context_of_commutation_names_obj_red.cxt` representing $\mathbb{K}''$. Using [2] and some manual formatting as above, the concepts (monoids and their witnesses) of `context_of_commutation_names_obj_red.cxt` were computed and stored in the file `concepts_new_format.txt` (cf. [8, Section 7.5]), and the corresponding intents were stored in `intents_new_format.txt`.

3 Centralising monoids up to conjugacy

For [1, Table 2] we also computed representatives of the 192 centralising monoids on $A = \{0, 1, 2\}$ up to conjugacy by inner automorphisms $s \in \mathrm{Sym}(A)$. In this way we were able to confirm the number 48 of conjugacy types mentioned in [6, 7]. We were also able to verify the correctness of the 48 conjugacy representatives shown in [6, Table 3]; for more details see Section 4.

To clarify the terminology, we define the *conjugate* of an n-ary operation $f\colon A^n \to A$ by a permutation $s \in \mathrm{Sym}(A)$ to be the function $f^s\colon A^n \to A$ that is given by $f^s(x_1, \ldots, x_n) := s\left(f\left(s^{-1}(x_1), \ldots, s^{-1}(x_n)\right)\right)$ for all $(x_1, \ldots, x_n) \in A^n$. Functions $f, g \in \mathcal{O}_A^{(n)}$ are said to be conjugate if there is $s \in \mathrm{Sym}(A)$ such that $f = g^s$, and this is clearly an equivalence relation. We extend the notion of conjugacy element-wise to sets $F \subseteq \mathcal{O}_A^{(n)}$ by putting $F^s := \{\, f^s \mid f \in F\}$ for $s \in \mathrm{Sym}(A)$. In fact, we use this definition here only for the case where $n = 1$ and $F \subseteq \mathcal{O}_A^{(1)}$. In this way we can speak of the *conjugate* F^s *by* s of a transformation monoid $F \subseteq \mathcal{O}_A^{(1)}$.

The classification of all centralising monoids on $\{0, 1, 2\}$ up to conjugacy was done as follows: the programme `reduce_monoids_conj.cpp` reads all 192 centralising monoids on $A = \{0, 1, 2\}$ as lists of integers in $\{0, \ldots, 26\}$ from the file `intents_numbers.cppinput` and outputs one representative monoid of each conjugacy class to the file `intents_up_to_conj.txt`. In the programme `reduce_monoids_conj.cpp` (and its in- and output) unary operations $s \in \mathcal{O}_A^{(1)}$ are represented by their hash value $9\cdot s(0)+3\cdot s(1)+s(2) \in \{0, \ldots, 26\}$. The code is rather straightforward: each monoid (line) of `intents_numbers.cppinput` is read in as a set of integers, these 192 sets are stored as a set. One then in `compute_monoids_up_to_conj` loops over each monoid F in that set and every $s \in \mathrm{Sym}(A)$ and conjugates all the elements of the monoid F by s, obtaining F^s. If the resulting isomorphic monoid F^s is distinct from the current monoid F (that is, it is a distinct isomorphic copy in the same conjugacy class), it is removed from the set of monoids, and thus only the representatives up to conjugacy remain. These are then output likewise to `intents_up_to_conj.txt`; this file has 48 lines.

For the presentation of the representative monoids of the 48 conjugacy types in [1, Table 2], it was necessary to find these 48 intents among the 192 concepts of `concepts_new_format.txt`. For this, the 192 concepts from `concepts_new_format.txt` were split at the `;` into their extent and intent part, and some delimiting characters such as `(`, `)`, `\{`, `\}`, `_{` and `}`, as well as the prefixes `s` for the attributes of the intents, were removed. This resulted in the files `192_extents.txt` and `192_intents.txt`.

To identify the 48 conjugacy representatives from `intents_up_to_conj.txt` among the concepts of `concepts_new_format.txt`, that is, among the extents and intents of `192_extents.txt` and `192_intents.txt`, respectively, the `python` script `finding_48_conjugacy_types_among_192_intents_extents.py` was written. The lines of `192_extents.txt` form the entries (sets) of the list `extentstrlist`, the same lines without the `h` and `f` prefixes are the entries of the list `extentlist` (removing the prefixes is sufficient to identify the objects of the extents uniquely). The lines of `192_intents.txt` are the entries of the list `intentlist`. The lines of `intents_up_to_conj.txt` are the entries of the list `listoftypes`. The procedure `print_sorted_intents_extents_of_types` then searches for the occurrence of each monoid in `listoftypes` in the list `intentlist` and outputs the corresponding index in the list, the intent (monoid) and the corresponding extent (witness). The result of running the script `finding_48_conjugacy_types_among_192_intents_extents.py` is stored in the file `conjugacy_types_intents_extents.txt`. From this file the lines containing 'Ext:' have been extracted to the file `48_extents.txt`, and the lines containing 'Int:' have been extracted to the file `48_intents.txt`. Based on those files, [1, Table 2] has been constructed. Moreover, from the output file `conjugacy_types_intents_extents.txt` we also extracted the list `192_extents_sorted.txt` of the 192 extents corresponding line-wise to the ones in `192_extents.txt`, but where the objects are mentioned in ascending order of their indices. Similarly, we extracted the list `192_intents_sorted.txt` of the 192 intents (monoids) corresponding line-wise to the ones in `192_intents.txt`, but again the functions in the monoids have been listed in ascending order of their hash values.

4 Comparison with results from [6]

We went to compare our results (in this data set and those mentioned in [1]) with the data available from [6]. The figures 192 and 48 reported in [6, Proposition 3.5] are already confirmed by the numbers of intents in `intents.txt` and `intents_up_to_conj.txt`, respectively. The other central piece of data is Table 3 from [6, p. 280]. We captured this table in `MachidaRosenberg2012_Table3.txt`, and, in order to better compare it with our results, reordered the columns of this table such that the operations $s \in \mathcal{O}_A^{(1)}$ are listed in ascending order of their hash values $9 \cdot s(0) + 3 \cdot s(1) + s(2)$ in $\{0, \ldots, 26\}$. That is, after reordering, the first column corresponds to the operation s_0 and the last one to s_{26} as in `context_of_commutation.cxt` and `context_of_commutation_names.cxt`. The reordered table can be found in the file `MachidaRosenberg2012_Table3_ordered_0-26.txt`, and it is stored as `MR2012tbl_reordered` in the file `finding_intents_up_to_conj_as_conjugates_of_MR2012_Table3.py`. We used the function `print_converted_list_to_integers(MR2012tbl_reordered)` to print the 48 monoids from [6, Table 3] as 48 lines of integer hash values in $\{0, \ldots, 26\}$, keeping the order of the table rows. The result is stored in the file `MachidaRosenberg2012_Table3_numbers.txt`. We then used the function call `find_conjugacy_types_as_conjugates_of_`

MR2012Table3(listoftypes,MR2012tbl_reordered,"human") in the script finding_intents_up_to_conj_as_conjugates_of_MR2012_Table3.py to find for each of our 48 representative monoids M from intents_up_to_conj.txt a monoid F listed in some row of [6, Table 3] and a permutation $s \in \mathrm{Sym}(3)$ such that $M = F^s$. The result of this computation has been stored in the file intents_up_to_conj_as_conjugates_of_MachidaRosenberg2012_Table3.txt and identifies the 48 representatives from intents_up_to_conj.txt in a one-to-one fashion with the representatives given in [6, Table 3]. This identification is also mentioned in [1, Table 2], and, since all 48 representatives from [6, Table 3] occur there, *the conjugacy classes computed in [6] and those represented by the monoids in* intents_up_to_conj.txt *coincide*; moreover intents_up_to_conj_as_conjugates_of_MachidaRosenberg2012_Table3.txt shows how the two presentations can be translated into each other.

5 Comparison with results from [7]

We also compared our computational results with the data available from [7, Tables 2–6, pp. 61–67]. To this end we first collected these five tables as accurately as possible in the files MachidaRosenberg2013_Table2.txt up to MachidaRosenberg2013_Table6.txt. We then joined these tables into one large table in the file MachidaRosenberg2013_Tables2-6.txt. As the next step we reordered the columns of this table (as in Section 4) in ascending order of the hash values $9 \cdot s(0) + 3 \cdot s(1) + s(2)$ of the unary operations $s \in \mathcal{O}_A^{(1)}$ that form the monoids represented in the table. The reordered table can be found (with additional information that was added later) in the file MachidaRosenberg2013_Tables2-6_ordered_0-26.txt.

We then inserted the contents of this table as MR2013tbl_reordered into the python script compute_hash_values_of_192_monoids.py. With the call print_converted_list_to_integers(MR2013tbl_reordered) we produced a list of the 192 monoids listed in [7, Tables 2–6] as sequences of integers in $\{0, \ldots, 26\}$ (keeping the order of the monoids as listed in MachidaRosenberg2013_Tables2-6.txt). The result of this computation is contained the file MachidaRosenberg2013_Tables2-6_numbers.txt. With the help of the call print_hash_values_for_list_of_characteristic_tuples(MR2013tbl_reordered) we computed hash values for the 192 monoids from [7, Tables 2–6]; the result is shown in the csv-file MachidaRosenberg2013_Tables2-6_ordered_0-26.csv (with ; as a delimiter) and has also been added to the table MachidaRosenberg2013_Tables2-6_ordered_0-26.txt. We further included the list of the 192 intents as shown in the file 192_intents.txt in the list intentlist. We then called print_hash_values_for_list_of_sets_of_integers_at_most_m(intentlist,26) in order to produce hash values for our 192 intents; the result is contained in the file 192_intent_hashes.txt and has been combined with the contents of 192_extents_sorted.txt and 192_intents_sorted.txt in the file 192_concepts_with_hashes.csv. We then pasted the files MachidaRosenberg2013_Tables2-6_ordered_0-26.csv and 192_concepts_with_hashes.csv in order to create the open document spreadsheet file 192_monoids_and_concepts.ods.

We sorted each part of the pasted table (in 192_monoids_and_concepts.ods) individually by ascending order of the hash values of the monoids; after that in each table row the two hash values coincided, thus we identified which monoid from `MachidaRosenberg2013_Tables2-6_ordered_0-26.txt` corresponds to which of the 192 concepts as listed in `192_extents_sorted.txt` and `192_intents_sorted.txt` or `concepts_new_format.txt`. The result is that the monoids we computed in `192_intents.txt` *are in a one-to-one correspondence* with the monoids in [7, Tables 2–6]. We thus prepended the suitable labels used for the monoids in [7, Tables 2–6] to the corresponding monoids in `192_concepts_with_hashes.csv`, resulting in the file `192_concepts_with_hashes_and_MR2013labels.csv`. We then sorted the whole table in `192_monoids_and_concepts.ods` once in the order of the lines of `192_intents_sorted.txt` (or of `concepts_new_format.txt`); this produced the file `192_monoids_and_concepts_reordered_by_concepts.csv`. We also sorted the whole table in the order of the monoids as listed in [7, Tables 2–6]; this produced the file `192_monoids_and_concepts_reordered_by_MRmonoids.csv`. Both files witness the fact that the monoids we computed via [2] and the context produced by `context_of_commutation_code.cpp` are indeed the same as those shown by Machida and Rosenberg in [7].

We have also used the file `192_monoids_and_concepts_reordered_by_MRmonoids.csv` to update `MachidaRosenberg2013_Tables2-6_ordered_0-26.txt` with the appropriate extents that witness the monoids from [7].

6 Overview of files with their uses

`commutation.cpp`	code computing the contexts $(\mathcal{O}_A^{(1)}, \mathcal{O}_A^{(1)}, \perp)$, $(\mathcal{O}_A^{(2)}, \mathcal{O}_A^{(1)}, \perp)$ and $\mathbb{K}'$ in the files `comm-unary-unary-3.cxt`, `comm-bin-unary-3.cxt` and `comm-ternary-unary-3.cxt`, respectively.
`comm-unary-unary-3.cxt`	the context $(\mathcal{O}_A^{(1)}, \mathcal{O}_A^{(1)}, \perp)$ as computed by `commutation.cpp`
`comm-bin-unary-3.cxt`	the context $(\mathcal{O}_A^{(2)}, \mathcal{O}_A^{(1)}, \perp)$ as computed by `commutation.cpp`
`comm-ternary-unary-3.cxt`	a (witness-complete) subcontext $\mathbb{K}'$ of $\mathbb{K}_1$ as computed by `commutation.cpp`
`comm-ternary-unary-obj-red-3.cxt`	the object reduced subcontext $\mathbb{K}''$ of $\mathbb{K}'$ as computed from `comm-ternary-unary-3.cxt` using ConExp
`comm-ternary-unary-red-3.cxt`	the fully reduced subcontext $\mathbb{K}'''$ of $\mathbb{K}'$ (standard context of $\mathbb{K}_1$) as computed from `comm-ternary-unary-3.cxt` using ConExp
`context_of_commutation_code.cpp`	code computing a different variant of the context $\mathbb{K}'$ as shown in the file `context_of_commutation.cxt`
`context_of_commutation.cxt`	a (witness-complete) subcontext $\mathbb{K}'$ of $\mathbb{K}_1$ as computed by `context_of_commutation_code.cpp`

`context_of_commutation_obj_red.cxt`	the object reduced subcontext $\mathbb{K}''$ of $\mathbb{K}'$ as computed from `context_of_commutation.cxt` using ConExp
`context_of_commutation_red.cxt`	the fully reduced subcontext $\mathbb{K}'''$ of $\mathbb{K}'$ (standard context of $\mathbb{K}_1$) as computed from `context_of_commutation.cxt` using ConExp
`context_of_commutation_names.cxt`	a different representation of $\mathbb{K}'$ by changing the object and attribute labels in `context_of_commutation.cxt`
`context_of_commutation_names_obj_red.cxt`	the object reduced subcontext $\mathbb{K}''$ of $\mathbb{K}'$ as computed from `context_of_commutation_names.cxt` using ConExp, cf. [1, Table 1]
`context_of_commutation_names_red.cxt`	the fully reduced subcontext $\mathbb{K}'''$ of $\mathbb{K}'$ (standard context of $\mathbb{K}_1$) as computed from `context_of_commutation_names.cxt` using ConExp
`concepts.txt`	a text file containing one concept of $\mathbb{K}'$ per line as computed from `context_of_commutation.cxt` via [2], with slight manual formatting (e.g., splitting lines, changing of brackets etc.)
`intents.txt`	a text file containing one intent of $\mathbb{K}'$ per line as computed from `context_of_commutation.cxt` via [2], with slight manual formatting
`intents_numbers.cppinput`	the result of running the command `sed -f int2numb.sed intents.txt > intents_numbers.cppinput`, which is used as an input file to `reduce_monoids_conj.cpp`, computing representatives of all monoids up to conjugacy
`int2numb.sed`	a `sed`-script to transform `intents.txt` into `intents_numbers.cppinput`
`concepts_new_format.txt`	a text file containing one concept of $\mathbb{K}''$ per line, computed from `context_of_commutation_names_obj_red.cxt` via [2], with slight manual formatting (splitting lines, separation symbols, etc.) to be included in a `.tex` file, cf. [8, Section 7.5]
`intents_new_format.txt`	a text file containing one intent of $\mathbb{K}''$ per line, computed from `context_of_commutation_names_obj_red.cxt` via [2], with slight manual formatting to be included in a `.tex` file.
`192_extents.txt`	The 192 concepts of `concepts_new_format.txt` were split up into their extent and intent part, removing some separating symbols, but preserving the order of the concepts and the order of the objects in the extents. The resulting 192 extents were stored in `192_extents.txt`.

`192_intents.txt`	The 192 concepts of `concepts_new_format.txt` were split up into their extent and intent part, removing some separating symbols, but preserving the order of the concepts and the order of the attributes (without the `s` prefix) in the intents. The resulting 192 intents were stored in `192_intents.txt`.
`reduce_monoids_conj.cpp`	code computing representatives of all intents of $\mathbb{K}'$ as stored in `intents_numbers.cppinput` (derived via `intents.txt` from `context_of_commutation.cxt`) up to conjugacy; this produces the file `intents_up_to_conj.txt`.
`intents_up_to_conj.txt`	the result of running `reduce_monoids_conj.cpp` on `intents_numbers.cppinput`; representatives of all 192 centralising monoids arising from `context_of_commutation.cxt` up to conjugacy
`finding_48_conjugacy_types_among_192_intents_extents.py`	a `python` script finding the 48 conjugacy types from `intents_up_to_conj.txt` among the 192 concepts of `concepts_new_format.txt` (stored in `192_extents.txt` and `192_intents.txt`). The result of running the script is contained in the file `conjugacy_types_intents_extents.txt`.
`conjugacy_types_intents_extents.txt`	the result of invoking the `python` script `finding_48_conjugacy_types_among_192_intents_extents.py`
`48_extents.txt`	the lines starting with 'Ext:' in `conjugacy_types_intents_extents.txt`, used for [1, Table 2]
`48_intents.txt`	the lines starting with 'Int:' in `conjugacy_types_intents_extents.txt`, used for [1, Table 2]
`192_extents_sorted.txt`	the same sets of witnesses, line by line, as in the file `192_extents.txt`, but within each line the elements have been listed in ascending order of their indices
`192_intents_sorted.txt`	the same sets of monoids, line by line, as in the file `192_intents.txt`, but within each line (monoid) the functions have been listed in ascending order of their indices (hash values)
`MachidaRosenberg2012_Table3.txt`	a faithful representation of [6, Table 3]
`MachidaRosenberg2012_Table3_ordered_0-26.txt`	a representation of [6, Table 3] after re-ordering the columns such that the unary operations are listed in the order $s_0 = \mathtt{000}, \dots, s_{26} = \mathtt{222}$

`MachidaRosenberg2012_Table3_numbers.txt`	a representation of the content of [6, Table 3] by listing the unary operations s_j in each monoid from `MachidaRosenberg2012_Table3_ordered_0-26.txt` in ascending order of their hash values j; each line represents a monoid (row) of [6, Table 3], the order of the rows has been left unchanged. The file has been computed via calling `print_converted_list_to_integers(MR2012tbl_reordered)` in `finding_intents_up_to_conj_as_conjugates_of_MR2012_Table3.py`
`finding_intents_up_to_conj_as_conjugates_of_MR2012_Table3.py`	a `python` script finding the 48 conjugacy types from `intents_up_to_conj.txt` as conjugates of the 48 representative monoids in [6, Table 3] as they are represented in `MachidaRosenberg2012_Table3_ordered_0-26.txt`. Calls in this script have produced the files `MachidaRosenberg2012_Table3_numbers.txt` and `intents_up_to_conj_as_conjugates_of_MachidaRosenberg2012_Table3.txt`.
`intents_up_to_conj_as_conjugates_of_MachidaRosenberg2012_Table3.txt`	output of the script `finding_intents_up_to_conj_as_conjugates_of_MR2012_Table3.py` via the call `find_conjugacy_types_as_conjugates_of_MR2012Table3(listoftypes, MR2012tbl_reordered,"human")`.
`MachidaRosenberg2013_Table2.txt`	a faithful representation of [7, Table 2]
`MachidaRosenberg2013_Table3.txt`	a faithful representation of [7, Table 3]
`MachidaRosenberg2013_Table4.txt`	a faithful representation of [7, Table 4]
`MachidaRosenberg2013_Table5.txt`	a faithful representation of [7, Table 5]
`MachidaRosenberg2013_Table6.txt`	a faithful representation of [7, Table 6]
`MachidaRosenberg2013_Tables2-6.txt`	a faithful representation of [7, Tables 2–6]; obtained by concatenating the tables in the files `MachidaRosenberg2013_Table2.txt`, `MachidaRosenberg2013_Table3.txt`, `MachidaRosenberg2013_Table4.txt`, `MachidaRosenberg2013_Table5.txt`, `MachidaRosenberg2013_Table6.txt` in this order.
`MachidaRosenberg2013_Tables2-6_ordered_0-26.txt`	a representation of the table in `MachidaRosenberg2013_Tables2-6.txt` after re-ordering the columns such that the unary operations are listed in the order $s_0 = \mathtt{000}, \ldots, s_{26} = \mathtt{222}$; moreover hash values from `MachidaRosenberg2013_Tables2-6_ordered_0-26.csv` and witnesses from `192_monoids_and_concepts_reordered_by_MRmonoids.csv` have been added to this file.

compute_hash_values_of_192_monoids.py	a python script computing hash values for the 192 monoids from [7, Tables 2–6] as listed in MachidaRosenberg2013_Tables2-6_ordered_0-26.txt and for the 192 intents (monoids) from 192_intents.txt. The former hash values have been used to produce the file MachidaRosenberg2013_Tables2-6_ordered_0-26.csv, the latter to obtain the files 192_intent_hashes.txt and 192_concepts_with_hashes.csv. The file MachidaRosenberg2013_Tables2-6_numbers.txt is also an output of this script.
MachidaRosenberg2013_Tables2-6_numbers.txt	a representation of the content of [7, Tables 2–6] by listing the unary operations s_j in each monoid from MachidaRosenberg2013_Tables2-6_ordered_0-26.txt in ascending order of their hash values j; each line represents a monoid (row) of [7, Tables 2–6], the order of the rows has been left unchanged. The file has been computed via calling print_converted_list_to_integers(MR2013tbl_reordered) in compute_hash_values_of_192_monoids.py
MachidaRosenberg2013_Tables2-6_ordered_0-26.csv	a combination of the hash values for the 192 monoids from [7, Tables 2–6] as listed in MachidaRosenberg2013_Tables2-6_ordered_0-26.txt, produced by the script compute_hash_values_of_192_monoids.py via the call print_hash_values_for_list_of_characteristic_tuples(MR2013tbl_reordered), with the original table data in MachidaRosenberg2013_Tables2-6_ordered_0-26.txt as a csv-file separated by ;
192_intent_hashes.txt	hash values for the computed 192 intents (monoids) from the file 192_intents.txt as output by compute_hash_values_of_192_monoids.py upon calling print_hash_values_for_list_of_sets_of_integers_at_most_m(intentlist,26); intentlist is the same list as in the script finding_48_conjugacy_types_among_192_intents_extents.py
192_concepts_with_hashes.csv	using the command paste -d ';' we combined a list of integers from 1 to 192 and the files 192_extents_sorted.txt, 192_intents_sorted.txt and 192_intent_hashes.txt into a ;-delimited csv-file 192_concepts_with_hashes.csv; this file contains the 192 concepts together with the hash value of each monoid

`192_monoids_and_concepts.ods`	using the command `paste -d ';'` on `MachidaRosenberg2013_Tables2-6_ordered_0-26.csv` and `192_concepts_with_hashes.csv` we produced a combined table with 192 rows; we used spreadsheet software to sort both parts of the combined table according to the column containing the hash values; in each row we found identical hash values, that is, matching monoids; we then sorted the whole table once according to the order given by the lines of `MachidaRosenberg2013_Tables2-6_ordered_0-26.csv` and once according to the order given by the lines of `192_concepts_with_hashes.csv`, the former resulting in `192_monoids_and_concepts_reordered_by_MRmonoids.csv` and the latter in `192_monoids_and_concepts_reordered_by_concepts.csv`; both tables are also contained in the open document spreadsheet `192_monoids_and_concepts.ods`. We also prepended the monoid labels used in [7, Tables 2–6] to the corresponding monoids in `192_concepts_with_hashes.csv`, resulting in the file `192_concepts_with_hashes_and_MR2013labels.csv`.
`192_concepts_with_hashes_and_MR2013labels.csv`	the 192 concepts listed in `192_concepts_with_hashes.csv` together with the appropriate identifiers for each monoid (intent) used in [7, Tables 2–6]. The file was obtained by a `paste -d ';'` of the column of identifiers extracted from `192_monoids_and_concepts_reordered_by_concepts.csv` and the file `192_concepts_with_hashes.csv`
`192_monoids_and_concepts_reordered_by_concepts.csv`	the 192 centralising monoids in the order of the file `192_concepts_with_hashes.csv` (i.e., the order of the file `192_intents.txt`, ≡ as in `concepts_new_format.txt`) with the corresponding monoid from [7, Tables 2–6] mentioned in the same line; this file was produced by exporting data from a sheet in `192_monoids_and_concepts.ods`.

`192_monoids_and_concepts_reordered_by_MRmonoids.csv`	the 192 centralising monoids occurring in the order of the file `MachidaRosenberg2013_Tables2-6_ordered_0-26.csv` (i.e., the order of `MachidaRosenberg2013_Tables2-6_ordered_0-26.txt`, ≡ as in `MachidaRosenberg2013_Tables2-6.txt`, ≡ as in [7, Tables 2–6]) with the corresponding intent (monoid) from `192_intents_sorted.txt` and extent (witness) from `192_extents_sorted.txt` mentioned in the same line; this file was produced by exporting data from a sheet in `192_monoids_and_concepts.ods`.
`cent_mons3_witn.tex`	LaTeX source file to produce this documentation
`cent_mons3_witn.pdf`	this documentation

Acknowledgement

The research of the first-named author that led to the results and the curation of the material presented in this data set has been partially supported by the Austrian Science Fund (FWF), grant number P33878, and is hereby gratefully acknowledged.

References

[1] Mike Behrisch and Leon Renkin. Computing witnesses for centralising monoids on a three-element set. To appear in *Proc. ICFCA 2023, Kassel, Germany, 17–21 July, 2023*, 16 pages, 2023.

[2] Daniel Borchmann and Tom Hanika. ConExp-clj, 2023. Available from `https://github.com/tomhanika/conexp-clj`, accessed on 5 Feb 2023.

[3] Bernhard Ganter and Sergei Obiedkov. *Conceptual exploration*. Springer-Verlag, Berlin, Heidelberg, 2016. doi: `https://doi.org/10.1007/978-3-662-49291-8`.

[4] Bernhard Ganter and Rudolf Wille. *Formal concept analysis. Mathematical foundations*. Springer-Verlag, Berlin, 1999. Translated from the 1996 German original by Cornelia Franzke. doi: `https://doi.org/10.1007/978-3-642-59830-2`.

[5] Tom Hanika and Johannes Hirth. Conexp-Clj - A research tool for FCA. In Diana Cristea, Florence Le Ber, Rokia Missaoui, Léonard Kwuida, and Barış Sertkaya, editors, *Supplementary Proc. ICFCA 2019, Frankfurt, Germany, 25–28 June, 2019*, volume 2378 of *CEUR Workshop Proceedings*, pages 70–75. CEUR-WS.org, 2019. Available on-line from `http://ceur-ws.org/Vol-2378/shortAT8.pdf`.

[6] Hajime Machida and Ivo G. Rosenberg. Centralizing monoids on a three-element set. In D. Michael Miller and Vincent C. Gaudet, editors, *2012 IEEE 42nd International Symposium on Multiple-Valued Logic—ISMVL 2012, Victoria, BC, Canada, 14–16 May 2012*, pages 274–280. IEEE Computer Soc., Los Alamitos, CA, 2012. doi: `https://doi.org/10.1109/ISMVL.2012.50`.

[7] Hajime Machida and Ivo G. Rosenberg. Report on centralizing monoids on a three-element set. In Hajime Machida, editor, *Clone Theory and Discrete Mathematics, June 13–15, 2005, Algebra and Logic Related to Computer Science, October 22–23, 2009*, volume 1846 of *RIMS Kôkyûroku*, pages 53–65, Kyoto, Japan, August 2013. RIMS (Research Institute for Mathematical Sciences), Kyoto University. Available on-line from `https://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/1846.html`.

[8] Leon Renkin. Centralizing monoids on a three-element set. Bachelor's thesis, Technische Universität Wien, August 2022. Available on-line from doi: `https://doi.org/10.5281/zenodo.7653085`.