Algorithmic aspects

of

finite semigroup and automata theory

using the

computer algebra system GAP

Manuel Delgado

CMUP Centro de Matemática Universidade do Porto

FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO Departamento de Matemática Pura

Soria Summer School on Computational Mathematics

Applied Computational Algebraic Geometric Modelling

Soria, 20-24/07/2009

Outline I

Outline II

- Example

4. More on automata
   - Non-deterministic automata
   - varia
   - Kleene's Theorem
   - Applications
5. Computing kernels of monoids
   - Definitions and easy consequences
   - Motivation
   - On the algorithms
   - Relative kernels and solvability
6. Numerical Semigroups
   - Motivation
   - PM semigroups
   - A database

Outline III

7. References
   - Software references
   - Basic references
   - Not so basic references
   - Specific references

web pages

- GAP: http://www.gap-system.org
- automata: http://www.gap-system.org/Packages/automata.html
- SgpViz: http://www.gap-system.org/Packages/sgpviz.html
- numericalsgps: http://www.gap-system.org/Packages/numericalsgps.html
- "GAP básico para la docencia": http://www.ugr.es/ pedro/gap/

Generalities on GAP

GAP stands for "Groups, Algorithms and Programming".

GAP is a free and open computer algebra system (you can access the system's algorithms). It is extensible in the sense that the users can write their own programs and use them the same way as those of the system.

GAP is developed internationally, through the cooperation of many people. It appeared in 1986 in Aachen, Germany. In 1997, the coordination center was transferred to St. Andrews in Scotland. Currently, centers in Aachen, Braunschweig, Fort Collins and St. Andrews coordinate together the development of GAP.

There are easy ways to make the installation both in Linux and in Windows or Mac.

GAP contains also a high level programming language (i.e., it is a language which is close to the language we speak).

The aspect of a shell where GAP is running:

```
File  Edit  View  Terminal  Tabs  Help
mdelgado@asa:~$ gap4

            #########           ######         ###########           ###
         #############          ######         ############         ####
        ##############         ########        #############       #####
       ###############         ########        #####   ######      #####
      ######         #         #########       #####    #####     ######
     ######                   ##########       #####    #####    #######
     #####                    ##### ####       #####   ######   ########
     ####                    #####  #####      #############   ###  ####
     #####     #######       ####    ####      ###########    ####  ####
     #####     #######      #####    #####     ######        ####   ####
     #####     #######      #####    #####     #####         #############
      #####     #####     ################     #####         #############
      ######    #####     ################     #####         #############
      ################    ##################   #####                ####
       ###############    #####        #####   #####                ####
         #############    #####        #####   #####                ####
          #########      #####          #####  #####                ####

     Information at:  http://www.gap-system.org
     Try '?help' for help. See also  '?copyright' and  '?authors'

   Loading the library. Please be patient, this may take a while.
```


GAP Semigroups Automata Automata (II) Semigroups (II) NS References

Generalities
Basic instructions
GAP Functions

The last line of working area

gap>

is ready to receive the instructions.

```
File  Edit  View  Terminal  Tabs  Help
Components:  small 2.1, small2 2.0, small3 2.0, small4 1.0, small5 1.0,
             small6 1.0, small7 1.0, small8 1.0, small9 1.0, small10 0.2,
             id2 3.0, id3 2.1, id4 1.0, id5 1.0, id6 1.0, id9 1.0, id10 0.1,
             trans 1.0, prim 2.1  loaded.
Packages:    Automata 1.12, GAPDoc 1.2, IO 2.3, SgpViz 0.998,
             NumericalSgps 0.96, FinSemi 0.001, AClib 1.1, Polycyclic 2.4,
             Alnuth 2.2.5, nq 2.2, CrystCat 1.1.3, Cryst 4.1.6, AutPGrp 1.2,
             CRISP 1.3.2, CTblLib 1.1.3, TomLib 1.1.4, FactInt 1.5.2,
             FGA 1.1.0.1, IRREDSOL 1.1.2, LAGUNA 3.4, Sophus 1.23,
             Polenta 1.2.7, ResClasses 2.5.3  loaded.
gap> 3*5;
15
gap> PrintFactorsInt(Factorial(30));Print("\n");
2^26*3^14*5^7*7^4*11^2*13^2*17*19*23*29
gap> Gcd(287,123);
41
gap> IsAbelian(Group((2,3,1),(1,2)));
false
gap> Phi(287);
240
gap> Filtered([1..100],IsPrime);
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,
  73, 79, 83, 89, 97 ]
gap> 
```

Manual

The manual of GAP is composed by 5 books:

- Tutorial,
- Reference Manual,
- Programming Tutorial,
- Programming Reference Manual *and*
- New Features for Developers.

In the *doc* folder, there is a sub-folder named *htm* providing access to the content of the manuals in *html* format. The complete path is `file:///usr/local/lib/gap4r4/doc/htm/index.htm` in a Linux standard installation. (In windows it is `C:\gap4r4\doc\htm`.) The manuals are also available in other formats, for example in *pdf*. The complete path in a Linux standard installation for the reference manual is `/usr/local/lib/gap4r4/doc/ref/manual.pdf`.

Basic instructions

To end a GAP session, one has simply to write `quit;` followed by *return*, or to press simultaneously the keys *ctrl-d*.
When an error occurs, GAP enters a *break loop*. This is indicated by

```
brk>
```

The get out of the loop one can proceed as to end a GAP session: to write `quit;` followed by *return*, or pushing up the keys *ctrl-d* at the same time.

To make a comment one should use the symbol #. All that is written in same line to the right of this symbol is ignored by GAP. In what follow, we also use this symbol to clarify some details in the examples given.

Example

```
gap> 5 +34 * 2;(3 +9) * 4;
73 # The priority of the operations...
48
gap> 5(8 +9)
Syntax error:; expected
5(8 +9);
  ^
gap> 5 *(8 +9);
85
gap>(9-7) *(46 +4;
Syntax error:) expected
(9-7) *(46 +4;
             ^
gap> # Using the `` up'' arrow we can fix it:

gap>(9-7) *(46 +4);
100
```


In GAP the level of priority of arithmetic operators is as usual.

Example

```
gap> 24 * 2-5 ^ 2;
23
gap> 45 = 34 +11 and 45 = 45;
true
gap> 34 <> 67;
true
gap> 56 = 56 +1;
false
```

Each instruction given should always end with `;` (semicolon) then GAP executes the instruction and gives the answer.

Two consecutive semicolons `;;` following an instruction make GAP to execute the instruction but not to show the answer to the user.

To save the work developed in a GAP session, one may write the instruction `LogTo("`*path / filename*`");`. To stop saving to the file `filename` should be written in `LogTo();`. The file can then be read and edited with a text editor.

Example

```
gap> LogTo"example1";
gap> a: = 12;
12
gap> a: = a +3;
15
gap> LogTo;
```

The command `InputLogTo` works like `LogTo`, but rather than writing the *inputs* and *output*, only writes the *inputs*.

Usually it is more practical to develop programs in a text editor (where you can save, read later and easily change) and copy them to GAP or read them with the command `Read` in GAP. To read a file one must specify the path.

Example

```
gap> Read "Desktop/Soria_2009/gap/example2.g";
```

GAP Functions

A program written in the GAP language is called a **function**. In GAP there are many pre-defined functions.

Example

```
gap> Factorial{23};
25852016738884976640000
gap> gcd{18,32,8};
2
gap> s5 := SymmetricGroup(5);;
gap> GeneratorsOfGroup(s5);
[ (1,2,3,4,5), (1,2) ]
gap> IsAbelian(s5);
false
...
brk>
```


Functions may be defined in two ways. One is by using the symbol ->, as the example below illustrates.

Example

```
gap> triple: = x-> x * 3;
function(x) ... end
```

After defining the function, it can be applied to concrete cases many times. For instance, we calculate the triple to 15.

Example

```
gap> triple{15};
45
```

It is also possible to define functions using the following syntax:

functionname`:=function(`*arguments*`)`
instructions
`end;`

Example

```
gap> name:=function(n)
> Print("My name is ",n,"\n");
> end;
function( n ) ... end
gap> name("Manuel");
My name is Manuel
```


The names of the functions which are part of GAP always begin with capitalized letters. Thus, a function whose name begins with a lowercase letter will not have compatibility problems.

To complete the editing of functions, (from the GAP or defined by the user) can use to key *tab* after typing the first letters of the name of the desired function.

In GAP, as in other programming languages, there are the following pre-defined: *if* (if ... then); *while* (while ... do); *repeat* (repeat ... until); *for* (for each object of the list do ...)

The syntax of the phrase *if* is:

- `if` *condition* `then` *instructions* `fi;`

The *else* part may be omitted.
One can write the condition *if* inside another several times, both in full and in abbreviated form.

- `if` *condition* `then` *instructions* else if *condition* `then` *instructions* `else` *instructions* `fi, fi;`
- `if` *condition* `then` *instructions* `elif` *condition* `then` *instructions* `else` *instructions* `fi;`

The *while*, *repeat* and *for* are known as *loops*, since they allow to loop over the same set of instructions. The syntax:

- `while` *bool-expr* `do` *statements* `od;`
- `repeat` *statements* `until` *bool-expr* `;`
- `for` *variable* `in` *list* `do` *statements* `od;`

Semigroups and monoids

A **semigroup** $(S, \odot)$ is a non empty set S, the **underlying set** of the semigroup, in which an operation

$$\odot : S \times S \to S$$

is defined.
The notation $x \odot y$ instead of $\odot(x, y)$ is commonly used. It is also common to represent the operation simply by $\cdot$ (or even to omit it) and call it **product**.

When there is no danger of confusion, we write S for $(S, \odot)$.

Saying that the operation is **associative** means that,

$$\forall x, y, z \in S, (x \odot y) \odot z = x \odot (y \odot z).$$

This just means that the **identity** $(x \odot y) \odot z = x \odot (y \odot z)$ **is satisfied**.

A **monoid** is a semigroup with a **neutral element**, i.e., an element 1 such that $1x = x = x1$, to any element x of the underlying set.

Note that a semigroup can not have more than a neutral element. Therefore, we can use a special notation for it, which can be the same for all monoids. Sometimes, to avoid confusion, we write 1_M to denote the neutral element of the monoid M.

If a semigroup S has no identity element, we can add to S an element $1 \notin S$ satisfying $1 \cdot 1 = 1$ and $1 \cdot a = a \cdot 1 = a$, for any $a \in S$. We define

$$S^1 = \begin{cases} S & \text{if } S \text{ has neutral element} \\ S \cup \{1\} & \text{otherwise} \end{cases}$$

Note that S^1 is the smallest monoid (under inclusion) containing S.

A semigroup is said to have a **zero** if it has an element 0 such that the identities $x0 = 0x = 0$ are satisfied. It is immediate to see that a semigroup can not have more than one zero.

An element e of a semigroup is said to be **idempotent** if $e^2 = e \cdot e = e$.

A **group** is a monoid such that any element x has an inverse, i.e., there exists an element x^{-1} of the underlying set such that $xx^{-1} = x^{-1}x = 1$. Note that an element of a group can not have more than one inverse.

Example

- Let $\mathbb{N}$ and $\mathbb{N}_0$ denote the set of positive integers and non negative integers, respectively. The (usual) addition in any of these sets is denoted by $+$.
  - $(\mathbb{N}, +)$ is a semigroup (but not a monoid);
  - $(\mathbb{N}_0, +)$ is a monoid whose neutral element is 0.
- Let A be a set and let A^A be the set of mappings from A to A. Denote by $\circ$ the composition of functions..
  - $(A^A, \circ)$ is a monoid whose neutral element is the identity function.

Let M be a monoid with neutral element 1. A subset N of M containing 1 which is closed under the product in M is said to be a **submonoid** of M. (If the condition $1 \in N$ is not required, then N is a subsemigroup of M.)

It is clear that N with the induced operation is a monoid itself. The notation $N \leq M$ means that N is a submonoid of M.

Example

- For any monoid M, $\{1\}$ and M are submonoids of M.
- The submonoids of $(\mathbb{N}_0, +)$ whose complement in $\mathbb{N}_0$ is finite are called **numerical semigroups**.

It is easy to show that the intersection of a family of submonoids of a given monoid M is a submonoid of M.

Let X be a subset of a monoid M. The submonoid

$$X^* = \bigcap_{\substack{Y \leq M \\ X \subseteq Y}} Y$$

is said to be the **submonoid of M generated by X**.
Note that X^* is the smallest (under inclusion) submonoid of M containing X.
When dealing with semigroups it is used the notation X^+.

Let X and Y be subsets of a monoid M.
We define the **product** of X by Y to be the following subset of M:

$$XY = \{xy \mid x \in X, y \in Y\}.$$

It is immediate that the power-set $\mathcal{P}(M)$ of M endowed with this operation is a monoid (with neutral element $\{1\}$).
Singular sets are usually represented by the single letter they contain. Thus, it is common to write, for instance, xY instead of $\{x\}Y$.

Let X be a subset of the monoid M. We define:

- $X^0 = \{1\}$;
- $X^n = \{x_1 \cdots x_n \mid x_1, \ldots, x_n \in X\}$, for $n \in \mathbb{N}$.

It is easy to show that if X is a subset of a monoid M, then $X^* = \bigcup_{n \in \mathbb{N}_0} X^n$.

The notation $\langle X \rangle$ is also used for the submonoid of M (or the subsemigroup of S) generated by X, that is, the least submonoid (subsemigroup) of M (S) containing X.

A monoid (or a semigroup) is said to be **finitely generated** if it is generated by a finite set.

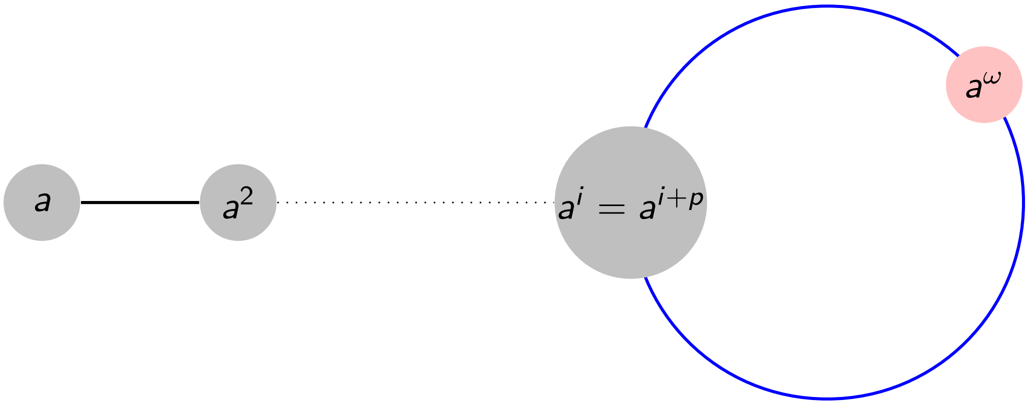
If $S = \langle A \rangle$, A is said to be a **generating set** of the semigroup S.
If $A = \{a\} \subseteq S$ is a singular set, then we represent by $\langle a \rangle$ the subsemigroup of S generated by a, $\langle a \rangle = \left\{a, a^2, a^3, \ldots\right\}$.
If there are no positive integers n and m such that $a^n = a^m$, then it is easy to see that $\langle a \rangle$ is isomorphic to the additive semigroup $\mathbb{N}$ and is infinite. If there are such positive integers, $\langle a \rangle$ is finite and it is not too difficult to prove the following:

Proposition 2.1

If $S = \langle a \rangle$ is finite of order n, then there are unique positive integers i and p such that:

i) $S = \left\{a, a^2, a^3, \ldots, a^{i+p-1}\right\}$.
ii) $a^i = a^{i+p}$.
iii) $n = i + p - 1$.
iv) $G = \left\{a^i, \ldots, a^{i+p-1}\right\}$ *is a cyclic group, whose neutral element is the only idempotent of S.*

The following picture illustrates the proposition.



The neutral element of the group G appearing in preceding proposition is denoted by a^ω.

Note that as a consequence we have that if S is a finite semigroup and $a \in S$, then there exists a natural number k such that a^k is an idempotent.

Relations

Let A and B be sets. A **relation** from A to B is s subset R of $A \times B$. When $B = A$ we say that R is a **binary relation** on A.
A binary relation that is simultaneously reflexive, symmetric and transitive is said to be an **equivalence relation**.
An equivalence relation on a set splits the set into equivalence classes.
A **(partial) function** φ from A to B, denoted $\varphi : A \to B$, is a relation from A to B such that for each element $x \in A$ there exists (at most) one element $y \in B$ such that $(x, y) \in \varphi$.
One may compose relations in a natural way: if $\varphi : A \to B$ $\psi : B \to C$ are relations, then $(a, c) \in \psi \circ \varphi : A \to C$ if there exists $b \in B$ such that $(a, b) \in \varphi$ and $(b, c) \in \psi$.
A function, as defined usually, may be seen as a relation: if $\rho : A \to B$ is a function, then the graph $\{(a, \rho(a)) \mid a \in A\}$ is a relation from A to B.
The "inverse" of a function may also be seen as a relation.

Homomorphisms

A **semigroup homomorphism** is a function $f : S \to T$ from a semigroup S into a semigroup T such that

$$f(xy) = f(x)f(y), \forall x, y \in S.$$

If, in addition, S and T are monoids and the image by f of the identity of S is the identity of T, we say that f is a **monoid homomorphism**.

Usually, when we are dealing with monoids, we only consider monoid homomorphisms and, when there is no risk of confusion, we say just **homomorphism**.

If f is an onto homomorphism, we say that T is an **homomorphic image** of S. A homomorphism that is one-one and onto is said to be an **isomorphism**. If there is an isomorphism from a monoid M to a monoid N, we say that the monoids M and N are **isomorphic**.

Relational morphisms

Let S and T be monoids. A **relational morphism** of monoids $\tau : S \multimap T$ is a function from S into $\mathcal{P}(T) = 2^Q$, the power set of T, such that:

- for all $s \in S$, $\tau(s) \neq \emptyset$;
- for all $s_1, s_2 \in S$, $\tau(s_1)\tau(s_2) \subseteq \tau(s_1 s_2)$;
- $1 \in \tau(1)$.

A relational morphism $\tau : S \multimap T$ is, in particular, a relation in $S \times T$. Thus, composition of relational morphisms is naturally defined.

Homomorphisms, seen as relations, and inverses of onto homomorphisms are examples of relational morphisms.

A monoid M is said to **divide** a monoid N, and we write $M \mid N$ in this case, if there exists a submonoid S of N and an onto homomorphism $\varphi : S \to M$, that is, M is a homomorphic image of a submonoid of N.

$$\begin{tikzcd} S \arrow[r, hook, "\iota"] \arrow[d, two heads, "\varphi"'] & N \\ M & \end{tikzcd}$$

Note that $\iota \circ \varphi^{-1}$ is a relational morphism.

Congruences

An equivalence relation ρ in a monoid M is said to be a **congruence** in M if

$$\forall a, b, c \in M, (a, b) \in \rho \implies (ac, bc), (ca, cb) \in \rho.$$

Example

Let $\mathbb{Z} = (\mathbb{Z}, +)$. For $n \in \mathbb{N}$, the relation $\equiv_n$ defined by

$$a \equiv_n b \iff n \text{ divide } b - a$$

is a congruence in $\mathbb{Z}$.

Let ρ be a congruence in a monoid M. We can define a binary operation in the set M/ρ of the equivalence classes of ρ through the following rule

$$(a\rho) \cdot (b\rho) = (ab)\rho.$$

Showing that the preceding rule does not depend on the representatives chosen, one can obtain the following:

Proposition 2.2

Let ρ be a congruence on the monoid M. The set M/ρ of the equivalence classes of ρ endowed with the product

$$(a\rho) \cdot (b\rho) = (ab\rho)$$

is a monoid. □

The monoid M/ρ with this operation is said to be the **quotient** of M by ρ.

Example

Given $n \in \mathbb{N}$, $(\mathbb{Z}/\equiv_n, +)$ is a monoid. It is even a group, called the group of the integers modulo n.

GAP Semigroups Automata Automata (II) Semigroups (II) NS References
Definitions
Morphisms and congruences
Syntactic congruence
Free monoids
Green's relations

Proposition 2.3

Let ρ be a congruence on M and let

$$\begin{array}{rcl} \varphi : M & \longrightarrow & M/\rho \\ a & \mapsto & a\rho \end{array}.$$

Then φ is an onto homomorphism.

Proof. Let $a, b \in M$. We have $\varphi(ab) = (ab)\rho = (a\rho)(b\rho) = \varphi(a)\varphi(b)$. Furthermore $\varphi(1) = 1\rho$ is the neutral element of M/ρ. The surjectivity is immediate. □

The homomorphism defined in the preceding proposition is referred as the **canonical homomorphism** from M into the quotient monoid M/ρ.

Let $\varphi : M \to N$ be a homomorphism. The **kernel** of φ is the relation $\ker\varphi$ on M defined by

$$(a, b) \in \ker\varphi \iff \varphi(a) = \varphi(b).$$

The following proposition, stated for monoids, holds in many other contexts of abstract algebra. Part **3** is known as the **homomorphism theorem**.

Proposition 2.4

Let M and N be monoids and let $\varphi : M \to N$ be a homomorphism. Then

1. $\ker\varphi$ *is a congruence on* M*;*
2. $\varphi(M)$ *is a submonoid of* N*;*
3. $M/\ker\varphi$ *is isomorphic to* $\varphi(M)$*.*

Syntactic congruence

A subset of a monoid M is said to be a M-**language**. When the monoid M is understood, one usually says simply "language".

Let L be a M-language. We define the relation $\sim_L$ on M as follows:

$$(a \sim_L b) \text{ if and only if } (xay \in L \iff xby \in L, \forall x, y \in M).$$

From the definition, it follows immediately that $\sim_L = \sim_{M \setminus L}$.

Proposition 2.5

The relation $\sim_L$ is a congruence on M.

Proof. To check that $\sim_L$ is an equivalence relation is straightforward. Suppose now that $a \sim_L b$ and that $c \in M$. For any $x, y \in M$ we have $x(ac)y \in L \iff xa(cy) \in L \iff xb(cy) \in L \iff x(bc)y \in L$, thus $(ac) \sim_L (bc)$. One can show analogously that $(ca) \sim_L (cb)$, thus $\sim_L$ is a congruence on M. $\square$

The congruence $\sim_L$ is said the **syntactic congruence** of L and the quotient $M/{\sim_L}$ is said the **syntactic monoid** of L.

GAP Semigroups Automata Automata (II) Semigroups (II) NS References

Definitions
Morphisms and congruences
Syntactic congruence
Free monoids
Green's relations

Let L be a M-language. We say that a congruence ρ on M **saturates** L if L is a union of ρ equivalence classes.

Proposition 2.6

Let M be a monoid and let L be a M-language. The syntactic congruence $\sim_L$ is the biggest congruence (under inclusion) that saturates L

Proof. Suppose that $u \sim_L v$. If $u \in L$, then $u = 1u1 \in L$ thus $v = 1v1 \in L$. It follows that $\sim_L$ saturates L.
Suppose that ρ is a congruence on M that saturates L and suppose that $u \,\rho\, v$. Then $xuy \;\rho\; xvy$ for any $x, y \in M$, thus, for any choice of $x, y \in M$, either $xuy, xvy \in L$ or $xuy, xvy \notin L$. This means that, for any $x, y \in M$,

$$xuy \in L \text{ if and only if } xvy \in L.$$

Thus $u \sim_L v$. $\square$

Free monoids

Let Σ be a finite non empty set.

It will be convenient to refer Σ as an **alphabet** and its elements as **letters**.

A **word** in Σ is a finite sequence of letters. This sequence may be empty. The empty sequence is called the **empty word** and is represented by 1. The notation ε is also used.

Given a word u in Σ, we use the convention $u^0 = 1$ and, for an integer $n \geq 1$, $u^n = u^{n-1} \cdot u$, that is,

$$u^n = \underbrace{u \cdots u}_{n \text{ times}}.$$

GAP Semigroups Automata Automata (II) Semigroups (II) NS References

Definitions
Morphisms and congruences
Syntactic congruence
Free monoids
Green's relations

Let $w = \sigma_1 \cdots \sigma_n$ ($\sigma_i \in \Sigma$) be a non empty word in Σ.

The integer n is said to be the **length** of w and is denoted by $|w|$.

The notation $|w|_\sigma$ is used for the number of occurrences of the letter σ in w. The **content** of w is the set $\{\sigma_1, \ldots, \sigma_n\}$ of letters that occur in w and is denoted by $c(w)$. We define $|1| = 0$ and $c(1) = \emptyset$.

Example

Let $\Sigma = \{\sigma, \tau\}$. Then

1. $\{1, \sigma, \tau, \sigma^2, \sigma\tau, \tau\sigma, \tau^2\} = \{w \in \Sigma^* : |w| \leq 2\}$;
2. $c(\sigma^2\tau\sigma) = \{\sigma, \tau\}$.

Let u and v be words in Σ. We say that

- u is **prefix** of v if $v = uw$ for some word w in Σ;
- u is **suffix** of v if $v = wu$ for some word w in Σ;
- u is **factor** of v if $v = zuw$ for some words w and z in Σ.
- $u = \sigma_1 \cdots \sigma_n$ is a **sub-word** of v if $v = u_0\sigma_1 u_1\sigma_2 \cdots \sigma_n u_n$ for some words $u_0, \dots, u_n$ in Σ.

Example

Let $w = abacbacb$. Then aba is a prefix of w, acb is a suffix, $bacb$ is a factor and $bcbc$ is a sub-word of w.

One defines a multiplication among words in Σ which is called **concatenation**, which consists of juxtaposition of words, when these are non-empty.

Let $\sigma_1 \cdots \sigma_n, \sigma'_1 \cdots \sigma'_m$ be two non empty words in Σ. We define

$$(\sigma_1 \cdots \sigma_n) \cdot (\sigma'_1 \cdots \sigma'_m) = \sigma_1 \cdots \sigma_n \sigma'_1 \cdots \sigma'_m;$$
$$1 \cdot (\sigma_1 \cdots \sigma_n) = (\sigma_1 \cdots \sigma_n) \cdot 1 = \sigma_1 \cdots \sigma_n;$$
$$1 \cdot 1 = 1.$$

It is clear that this operation is associative and that the empty word is the neutral element.
We then have that the set of words in Σ endowed with the operation just defined is a monoid, called the free monoid over Σ. It is denoted by Σ^*.
Each letter is naturally identified with the word of length 1 of Σ^* constituted by that letter.

The algebraic importance of the free monoid is mainly due to the following proposition (which says that the free monoid satisfies the so called **universal property**).

Proposition 2.7

Let Σ be a non empty finite set, M a monoid and $\varphi : \Sigma \to M$ a function. Then there exists one and only one homomorphism $\Phi : \Sigma^* \to M$ such that $\Phi|_\Sigma = \varphi$, that is, such that the following diagram commutes.
(The function ι is the inclusion.)

$$\begin{tikzcd} \Sigma \arrow[r, "\varphi"] \arrow[dr, hook, "\iota"'] & M \\ & \Sigma^* \arrow[u, dashed, "\Phi"'] \end{tikzcd}$$

(To say that the **diagram commutes** means that $\Phi(i(\sigma)) = \varphi(\sigma)$,

Proposition 2.8

Every monoid is a homomorphic image of a free monoid.

Proof. Let M be a monoid and let Σ be a generating set of M. (Note that we can take $\Sigma = M$, but usually Σ can be taken much smaller.)

Let $\varphi : \Sigma \hookrightarrow M$ be the. By the universal property, there exists a homomorphism $\Phi : \Sigma^* \to M$ such that $\Phi|_\Sigma = \varphi$. Φ is an onto homomorphism, since any element of M is the product $\sigma_1 \cdots \sigma_n$ of elements of Σ, thus image by Φ of $\sigma_1 \cdots \sigma_n \in \Sigma^*$. $\square$

The following is an immediate consequence of previous results.

Corollary 2.9

Every monoid is isomorphic to a quotient of a free monoid. $\square$

Green's relations

Next we introduce the equivalence relations $\mathscr{L}$, $\mathscr{R}$, $\mathscr{J}$, $\mathscr{H}$ e $\mathscr{D}$. They are known as **Green's relations** and are an essential ingredient in semigroup theory. They have been introduced by J. Green in 1951

Definition

Let S be a semigroup and let $a, b \in S$

$$a \mathscr{R} b \iff aS^1 = bS^1;$$

$$a \mathscr{L} b \iff S^1 a = S^1 b;$$

$$a \mathscr{J} b \iff S^1 a S^1 = S^1 b S^1;$$

GAP Semigroups Automata Automata (II) Semigroups (II) NS References

Definitions
Morphisms and congruences
Syntactic congruence
Free monoids
Green's relations

The relation $\mathscr{H}$ is defined as $\mathscr{H} = \mathscr{L} \cap \mathscr{R}$ and the relation $\mathscr{D}$ is the least equivalence relation containing both $\mathscr{L}$ and $\mathscr{R}$.
It is not difficult to observe that $\mathscr{D} = \mathscr{R} \circ \mathscr{L} = \mathscr{L} \circ \mathscr{R} = \mathscr{L} \vee \mathscr{R}$.
An important result, when dealing with finite semigroups, is the following:

Theorem 2.10

If S is a finite semigroup, then $\mathscr{D} = \mathscr{J}$.

Let S be a semigroup and let $\mathscr{K}$ be one of the Green's relations on S. We denote by $\mathscr{K}_a$ the equivalence class of the element $a \in S$ for the corresponding relation.

The following results are due to Green.

Proposition 2.11

All $\mathscr{H}$-classes of a $\mathscr{J}$-class J of a finite semigroup have the same cardinality. Similarly, all $\mathscr{L}$-classes of J have the same size and all $\mathscr{R}$-classes of J have the same size. All $\mathscr{L}$-classes of J contain the same number of $\mathscr{H}$-classes and all $\mathscr{R}$-classes of J contain the same number of $\mathscr{H}$-classes.

Proposition 2.12

Every $\mathscr{H}$-class of a semigroup S containing an idempotent is a group.

In particular, no $\mathscr{H}$-class contains more than one idempotent.
Let S be a semigroup. A subsemigroup of S that happens to be a group is usually called a **subgroup** of the semigroup.

GAP Semigroups Automata Automata (II) Semigroups (II) NS References

Definitions
Morphisms and congruences
Syntactic congruence
Free monoids
Green's relations

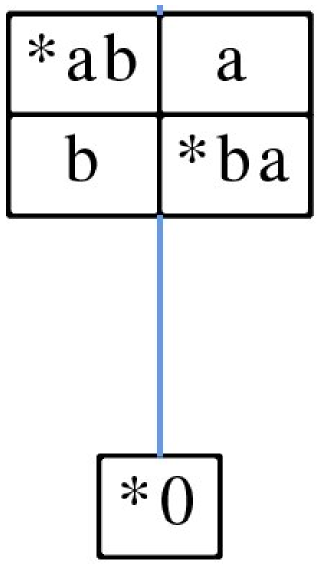
Its worth to mention some other facts:

Proposition 2.13

Let S be a semigroup.

- *The maximal subgroups of S are precisely the $\mathscr{H}$-classes containing idempotents.*
- *The maximal subgroups of S contained in a $\mathscr{D}$ class are isomorphic.*

In virtue of the above results, the $\mathscr{D}$-classes of a finite semigroup are usually depicted by means of the so-called "egg-box" pictures.



The presence of an idempotent in a $\mathscr{H}$ class is signaled by a "star".

Several important classes of finite semigroups can be defined using Green's relations.

The classes of $\mathscr{H}$-trivial or $\mathscr{J}$-trivial semigroups are such examples.

A finite semigroup S is said to be **aperiodic** if and only if all its subgroups are trivial.
One can prove that this is equivalent to being $\mathscr{H}$-trivial.

Recognizable languages
Rational languages
Graphs
Σ^*-automata
Example

Automata and recognizable languages

Recall that a subset of a monoid M is said to be a M-**language**. We say that a M-language L is **recognized** by a monoid N if there exists a homomorphism $\varphi : M \to N$ and a subset P of N such that $L = \varphi^{-1}(P)$.

This is equivalent to saying that $\varphi^{-1}(\varphi(L)) = L$, or that $\varphi(L) \cap \varphi(M \setminus L) = \emptyset$ (exercise).
We also say in that the **homomorphism** φ **recognizes** L.

We say that L is **recognizable by a monoid** if it is recognized by a finite monoid.

Recognizable languages
Rational languages
Graphs
Σ^*-automata
Example

Example

1. Let $\mathbb{Z} = (\mathbb{Z}, +)$ and $n \in \mathbb{N}$. The consideration of the homomorphism
$$\begin{array}{rcl} \varphi : \mathbb{Z} & \to & \mathbb{Z}/\equiv_n \\ a & \mapsto & a \equiv_n \end{array}$$
allows us to conclude that the set L of the multiples of n is recognizable by a monoid. Note that $L = \varphi^{-1}(0 \equiv_n)$.
2. For any monoid M, the homomorphism $\varphi : M \to \{1\}$ recognizes the languages M and $\emptyset$.
3. Let $\Sigma = \{\sigma, \tau\}$. The language $L = \{w \in \Sigma^* \mid |w| \text{ is a multiple of } n\}$ is recognizable by a monoid. In fact, if we consider the homomorphism $\varphi : \Sigma^* \to \mathbb{Z}/\equiv_n$ defined by $\varphi(\sigma) = \varphi(\tau) = 1$, we have that $L = \varphi^{-1}(0 \equiv_n)$.

A **deterministic** M**-automaton** is a quadruple $\mathcal{A} = (Q, i, F, \delta)$ where Q is a non-empty set (known as the **set of states**) i is an element of Q (known as the **initial state**) F is a non-empty subset of Q (known as the **set of final states**) and

$$\begin{array}{rcl} \delta : Q \times M & \rightarrow & Q \\ (q, m) & \mapsto & qm \end{array}$$

is a partial function such that

- $q1 = q$,
- $(qm)n = q(mn)$,

for any $q \in Q$, $m, n \in M$ (that is, δ is an **action** of M over Q).

The partial function δ is said to be the **transition function** of the automaton.

Note that, instead of the notation $\delta((q, m))$, we are using qm to denote the image of $(q, m) \in Q \times M$ by δ.

The M**-language recognized by** $\mathcal{A}$ is

$$L(\mathcal{A}) = \{m \in M \mid im \in F\}.$$

An automaton is said to be **finite** if it has a finite number of states.

A M-language is said to be **recognizable by a (deterministic) automaton** if it is recognized by some finite M-deterministic automaton.

Proposition 3.1

Let L be a M-language. The following conditions are equivalent:

1. *L is recognizable by a monoid;*
2. *L is recognizable by a (deterministic) automaton;*
3. *the syntactic monoid $M/{\sim_L}$ is finite.*

Proof. $(1 \implies 2)$ Suppose that there exist a finite monoid N, a homomorphism $\varphi : M \to N$ and a subset $P \subseteq N$ such that $\varphi^{-1}(P) = L$.

Let $\mathcal{A} = (Q, i, F, \delta)$ with $Q = N$, $i = 1$, $F = P$ and $nm = \delta(n, m) = n\varphi(m)$, with $n \in N$ and $m \in M$. One has

$$n1 = n$$

and

$$(nm)m' = [n\varphi(m)]m' = [n\varphi(m)]\varphi(m') = n[\varphi(m)\varphi(m')] = n[\varphi(mm')] = n(mm'),$$

for any $n \in N$, $m, m' \in M$, thus δ is an action of M over N and, therefore, $\mathcal{A}$ is an M-automaton.

One has $L(\mathcal{A}) = \{m \in M \mid im \in P\} = \{m \in M \mid \varphi(m) \in P\} = L$. Thus L is recognizable by a (deterministic) automaton.

$(2 \implies 3)$ Let $\mathcal{A} = (Q, i, F, \delta)$ be a finite M-automaton such that $L = L(\mathcal{A})$. Each $m \in M$ induces a partial function $\delta_m : Q \to Q$ defined through $\delta_m(q) = qm$. As Q is finite, there exists a finite number of such functions (no more than $|Q|^{|Q|}$).
We define in M he following relation τ:

$$m \,\tau\, m' \text{ if and only if, for all } q \in Q, qm = qm'.$$

It is straightforward that τ is an equivalence relation in M.
As two elements $m, m' \in M$ are τ-equivalent if and only if $\delta_m = \delta_{m'}$, we have that the set M/τ of equivalence classes is finite.

To conclude that the syntactic monoid $M/\!\sim_L$ is finite it suffices to show that $\tau \subseteq\sim_L$.

Suppose that $m, m' \in M$ are τ-equivalent. Let $x, y \in M$ and suppose that $xmy \in L$, that is, $i(xmy) \in F$. We then have $i(xmy) = ((ix)m)y = ((ix)m')y = i(xm'y)$, thus $i(xm'y) \in F$ and, therefore, $xm'y \in L$.
Analogously, $xm'y \in L$ implies $xmy \in L$. It follows that $m \sim_L m'$.

$(3 \implies 1)$ Suppose that $M/{\sim_L}$ is finite and let

$$\begin{array}{rcl} \varphi : M & \to & M/{\sim_L} \\ a & \mapsto & a{\sim_L} \end{array}.$$

We will see that $\varphi^{-1}(\varphi(L)) = L$.
First note that $L \subseteq \varphi^{-1}(\varphi(L))$ holds for any function φ.

In order to prove the reverse inclusion, let $m \in \varphi^{-1}(\varphi(L))$. Then $\varphi(m) = \varphi(\ell)$ for some $\ell \in L$. This implies that $m \sim_L \ell$. As $\sim_L$ saturates L, we have that $m \in L$ and, therefore, $\varphi^{-1}(\varphi(L)) \subseteq L$. this proves that L is recognizable by a monoid. $\square$

A M-language is said to be **recognizable** if it satisfies any (and thus all) of the equivalent conditions of previous proposition.
The set of all recognizable M-languages is denoted by $\mathrm{Rec}\, M$.

GAP Semigroups Automata Automata (II) Semigroups (II) NS References

Recognizable languages
Rational languages
Graphs
Σ^*-automata
Example

Proposition 3.2

Let $L \in \mathrm{Rec}\, M$. Then $M \setminus L \in \mathrm{Rec}\, M$.

Proof. Let $\varphi : M \to N$ be a homomorphism from M into a finite monoid N and $P \subseteq N$ be such that $L = \varphi^{-1}(P)$. Then $M \setminus L = \varphi^{-1}(N \setminus P)$. $\square$

Proposition 3.3

Let $L_1, L_2 \in \mathrm{Rec}\, M$. Then $L_1 \cap L_2, L_1 \cup L_2 \in \mathrm{Rec}\, M$.

Proof. Let $\varphi_1 : M \to N_1$ and $\varphi_2 : M \to N_2$ be homomorphisms from M into the finite monoids N_1 and N_2 respectively, and let $P_1 \subseteq N_1$ and $P_2 \subseteq N_2$ be such that $L_1 = \varphi_1^{-1}(P_1)$ and $L_2 = \varphi_2^{-1}(P_2)$.
Then the homomorphism $\varphi : M \to N_1 \times N_2$ defined by $\varphi(m) = (\varphi_1(m), \varphi_2(m))$ is such that $\varphi^{-1}(P_1 \times P_2) = L_1 \cap L_2$ and $\varphi^{-1}(P_1 \times N_2 \cup N_1 \times P_2) = L_1 \cup L_2$. $\square$

Recognizable languages
Rational languages
Graphs
Σ^*-automata
Example

As a consequence of the fact that $\mathrm{Rec}\, M$ is closed under union, we have that if we generalize the definition of automaton to allow more than one initial state and we define, as would be natural to do, the language recognized by one of these automata as the set of all elements leading from some initial state to some final state, we do not get the recognition of new languages.

Proposition 3.4

Let $\psi : M \to M'$ be a monoid homomorphism and let $L' \in \mathrm{Rec}\, M'$. Then $\psi^{-1}(L') \in \mathrm{Rec}\, M$.

Proof. Let $\varphi : M' \to N$ be a homomorphism from M' into a finite monoid N and $P \subseteq N$ be such that $L' = \varphi^{-1}(P)$.
Then $\psi^{-1}(L') = \psi^{-1}(\varphi^{-1}(P)) = (\varphi \circ \psi)^{-1}(P)$, which shows that $\psi^{-1}(L')$ is recognized by $\varphi \circ \psi$. □

Recognizable languages
Rational languages
Graphs
Σ^*-automata
Example

Rational languages

The **rational subsets** of a monoid M form the least class $\mathrm{Rat}\, M$ of M-languages such that:

(a) the empty set $\emptyset$ and all the singular subsets $\{m\}$ of M belong to $\mathrm{Rat}\, M$;

(b) if S and T belong to $\mathrm{Rat}\, M$, then ST and $S \cup T$, belong to $\mathrm{Rat}\, M$;

(c) if S belongs to $\mathrm{Rat}\, M$, the same happens with S^*.

A subset A of a monoid M obtained from the singular subsets through a finite number of "unions", "products" and "stars" belongs to $\mathrm{Rat}\, M$. Furthermore, all subsets of M obtained in this way, together with the empty set, satisfy (a)-(c).
Thus, we can express any non empty rational subset starting from singular sets and using a finite number of times the union, the product and the star operation. Such an expression is said to be a **rational expression** of the subset.

Note that different rational expressions may represent the same set.

Proposition 3.5

Let $\varphi : M \to N$ be a homomorphism and let $L \in \mathrm{Rat}\, M$. Then $\varphi(L) \in \mathrm{Rat}\, N$.

Proof. Let $F = \{A \subseteq M \mid \varphi(A) \in \mathrm{Rat}\, N\}$. We want to show that $\mathrm{Rat}\, M \subseteq F$. It is clear that $\emptyset \in F$ and $\{m\} \in F$, for all $m \in M$. It remains to prove that F is closed for the operators "union", "product" and "star".

Let $A, B \in F$. We have:

- $\varphi(A \cup B) = \varphi(A) \cup \varphi(B) \in \mathrm{Rat}\, N$, thus $A \cup B \in F$;
- $\varphi(AB) = \varphi(A)\varphi(B) \in \mathrm{Rat}\, N$, thus $AB \in F$;
- $\varphi(A^*) = \varphi(\cup_{n\geq 0} A^n) = \cup_{n\geq 0}\varphi(A^n) = \cup_{n\geq 0}(\varphi(A))^n = (\varphi(A))^* \in \mathrm{Rat}\, N$, thus $A^* \in F$.

We have that F is closed under the three operators under consideration, thus $\mathrm{Rat}\, M \subseteq F$. In particular, $\varphi(L) \in \mathrm{Rat}\, N$. $\square$

Graphs

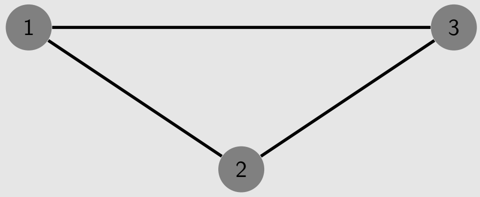
A **graph** G is an ordered pair of disjoint sets (V, E) such that E is a set of subsets of V containing two distinct elements.

The set V is said to be the set of **vertices** and E is said to be the set of **edges** We say that an edge $\{x, y\}$ **connects** the vertices x and y, being these vertices said to be the **ends** of the edge. Two vertices connected by an edge are said to be **adjacent**. Two edges sharing a common vertex are said to be **adjacent**.

As the terminology suggests, one usually does not think in a graph as an ordered pair, but as a collection of vertices some of which are connected by edges.

Example

The graph $(\{1, 2, 3\}, \{\{1, 2\}, \{1, 3\}, \{2, 3\}\})$ is described through the following picture.



We say that a graph $G' = (V', E')$ is a **subgraph** of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.
Two graphs are said to be **isomorphic** if there exists a correspondence between its vertex vets that preserves adjacency.
(Formally: $G = (V, E)$ is isomorphic to $G' = (V', E')$ if the exists a one-one and onto function $\varphi : V \to V'$ such that $\{x, y\} \in E$ if and only if $\{\varphi(x), \varphi(y)\} \in E'$.)

GAP | Semigroups | Automata | Automata (II) | Semigroups (II) | NS | References

Recognizable languages
Rational languages
Graphs
Σ^*-automata
Example

A **path** C in a graph G is an alternate sequence of vertices and edges

$$x_0, \alpha_1, x_1, \ldots, \alpha_n, x_n$$

such that $\alpha_i = \{x_{i-1}, x_i\}, 0 < i \leq n$.
The **length** of the path $x_0, \alpha_1, x_1, \ldots, \alpha_n, x_n$ is the integer n.
The path C can also be represented as $x_0, x_1, \ldots, x_n$, since the edges are completely determined by its ends.
If $x_0 = x_n$, we say that C is a **circuit**.
(Some authors use the terminology "walk" for the notion of path just defined. In this case, the term "path" is reserved to walks with no vertex repetition.)

A graph is said to be **connected** if any two vertices can be connected by a path.

Let Σ be a set. If to every edge e of a graph G is associated an element $\sigma \in \Sigma$ (which is said to be the **label** of e), we say that G is **labeled** by Σ.

There are many notions closely related to the notion of graph. Among them are the notions of multi-graph, directed graph and directed multi-graph.

The notion of **multi-graph** is obtained with the following modification: there are permitted multiple edges between pairs of vertices and **loops** (i.e., edges with a single end).

The notion of **directed graph** is obtained by requiring the edges to be ordered pairs, instead of two element sets.

An ordered pair (x, y) is then said to be a **directed edge** from x to y or an edge whose **beginning** is x and whose **end** is y.

The notion of **directed multi-graph** is obtained in a similar way.

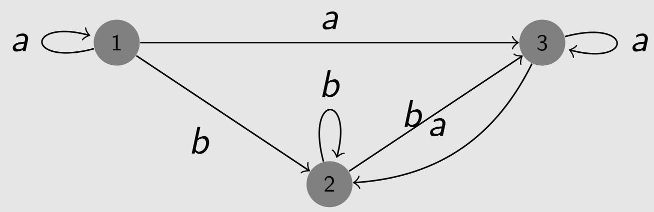
The notions of **path**, **circuit** and **labeled graph** previously defined for graphs may also be defined, with the obvious changes, for multi-graphs, directed graphs and directed multi-graphs.

The notion of path in a directed multi-graph may also be defined as a sequence of consecutive edges $\alpha_1, \dots, \alpha_n$ (the end of α_{i-1} is the beginning of α_i, $i \in \{2, \dots, n\}$).

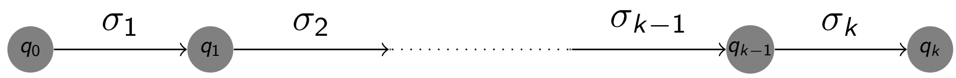
Usually we use the notation (x, a, y) to indicate that the edge (x, y) of a directed graph has label a. This notation is often convenient to represent the edges of a directed multi-graph. The labels can then help to distinguish between the various edges connecting two vertices.

Example

A directed multi-graph labeled by $\Sigma = \{a, b\}$.



A path in a directed labeled multi-graph is usually described through a picture like the one that follows.



If the labels $\sigma_1, \ldots, \sigma_m$ of the edges forming a path belong to a monoid, then the product of the labels $\sigma_1 \cdots \sigma_m$ is said to be the **label** of the path.

Σ^*-automata

From now on we will only consider the particular case of the Σ^*-automata.
It is probably the most important case, since it has many applications.
The prefix Σ^* is usually omitted, and we say just **automaton**. In this case, it is common to include the alphabet in the list of elements used to describe the automaton.
So we can say "the deterministic Σ^*-automaton (Q, i, F, δ)" or the "deterministic automaton $(Q, \Sigma, i, F, \delta)$", with the same meaning.
For any $q \in Q$ and $\sigma_1, \sigma_2, \cdots, \sigma_n \in \Sigma$, we have

$$q(\sigma_1\sigma_2 \cdots \sigma_n) = (q\sigma_1)(\sigma_2 \cdots \sigma_n) = \ldots = ((\cdots(q\sigma_1)\sigma_2)\cdots)\sigma_n,$$

thus the partial function $\delta : Q \times \Sigma^* \to Q$ is completely determined by its restriction to $Q \times \Sigma$.

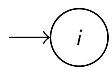
The observation just made enables us to give an elegant description of Σ^*-automata through directed multi-graphs labeled by Σ.

In this context, it is common to use the terminology "graph" instead of "directed labeled multi-graph".

A deterministic automaton $\mathcal{A} = (Q, \Sigma, i, F, \delta)$ may then be seen as a graph whose vertex and edge sets are respectively Q and $E = \{(p, \sigma, q) \in Q \times \Sigma \times Q \mid p\sigma = q\}$.

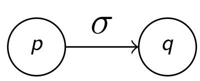
So, the deterministic automaton $\mathcal{A}$ may be given through a set E of edges (with the restriction: "to each pair $(p, \sigma) \in Q \times \Sigma$, there exists **at most** a state $q \in Q$ such that $(p, \sigma, q) \in E$") instead of the partial function δ.
The automaton $\mathcal{A}$ is then described through a vector (Q, Σ, i, F, E).
The terminology "state" or "vertex" for an element of Q is used indistinctly.
The initial and terminal vertices are naturally distinguished in this representation.

In a graphical representation the initial state i is represented by



while a terminal state may be represented by one of the following ways:

The edge (p, σ, q) ($p, q \in Q$ and $\sigma \in \Sigma$ such that $p\sigma = q$) is represented by the subgraph

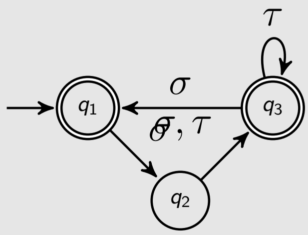


A path that goes from the initial state to a final state is said to be **successful**.

Example

Let $\Sigma = \{\sigma, \tau\}$ be an alphabet. The automaton $\mathcal{A} = (Q, \Sigma, i, F, \delta)$ with $Q = \{q_1, q_2, q_3\}$, $i = q_1$, $F = \{q_1, q_3\}$ and the partial function δ given by the table:

δ	q_1	q_2	q_3
σ	q_2	q_3	q_1
τ	$-$	q_3	q_3



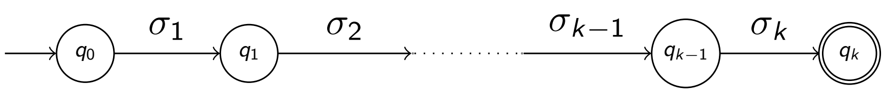
has the graphical description:

Before presenting some generalizations of the concept of automaton that, although not recognizing more languages, give us greater flexibility, we prove the following result, known as the Pumping Lemma.

Theorem 3.6

Let L be a recognizable Σ^*-language. Then there exists a positive integer N such that, for all word $u \in L$ with $|u| \geq N+1$, there exist $x, v, y \in \Sigma^*$ such that $|xv| \leq N$, $v \neq 1$, $u = xvy$ and $xv^*y \subseteq L$.

Proof. One has $L = L(\mathcal{A})$, for some finite Σ^*-automaton $\mathcal{A}$ with N states.
Let $u = \sigma_1\sigma_2 \cdots \sigma_k \in L(\mathcal{A})$, $\sigma_i \in \Sigma$, with $|u| = k > N$.



In the successful path labeled $u = \sigma_1\sigma_2 \cdots \sigma_k$ there exists at least a repetition among the first $N+1$ states $q_0, q_1, \ldots, q_N$. Let q_r, with $r \geq 0$, be the first state that repeats and let q_{r+s} be the first repetition. Observe that $r \geq 0, s > 0$ and $r + s \leq N$.
Then we have $u = xvy$ with $x \in \Sigma^*$ the label of the path from q_0 to q_r, $v \in \Sigma^+$ the label of the path from q_r to q_{r+s} and $y \in \Sigma^*$ the label of the path from q_{r+s} to q_k.
The path from q_0 to q_k includes the circuit

$$q_r \to q_{r+1} \to \cdots \to q_{r+s-1} \to q_{r+s} = q_r$$

labeled by v. We obtain successful paths if we loop over this circuit any number of times (including none). Thus $xv^my \in L$, for any $m \geq 0$. It remains to observe that $|xv| = r + s \leq N$. □

Informally, the Pumping Lemma says that if L is recognizable, then any "sufficiently long" word of L contains a factor which may be repeated any number of times, keeping the resulting word in L. In particular, if L has a sufficiently long word, then L is infinite.

Since the Pumping Lemma gives a necessary condition for a language to be recognizable, its use is often by the negative: it is useful to prove that certain languages are not recognizable.

Example

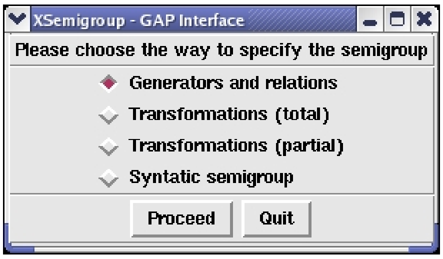
Let $\Sigma = \{\sigma, \tau\}$ be an alphabet. The language $L = \{\sigma^n\tau^n : n \in \mathbb{N}\}$ is non recognizable.
If it were recognizable, then it would be recognized by some finite Σ^*-automaton $\mathcal{A}$ with, say, N states. Let us look to the word $u = \sigma^n\tau^n$ with $n > N$. By the Pumping Lemma, u may be written as xvy with $|xv| \leq N$ and $v \neq 1$, having also $xv^2y \in L$. Since v is of the form σ^k, xv^2y contains more occurrences of σ than those of τ, which is absurd.

Recognizable languages
Rational languages
Graphs
Σ^*-automata
Example

If the package “SgpViz” has been loaded into a GAP session, by typing “XSemigroup();” in the command line:

```
gap> XSemigroup();
```

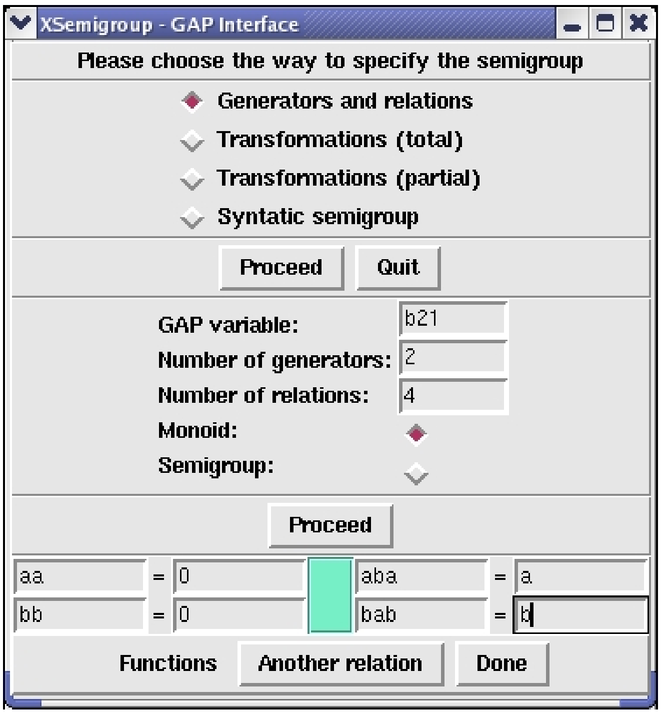
a window like the following pops up



One may then choose to give a semigroup (or a monoid) by specifying a presentation, by giving (partial) transformations generating it or by giving it as the syntactic semigroup of a rational language or a finite state automaton.

Recognizable languages
Rational languages
Graphs
Σ^*-automata
Example

Next, the Brandt 6 element monoid is specified through a presentation. (Note that “0” is just an abbreviation.)



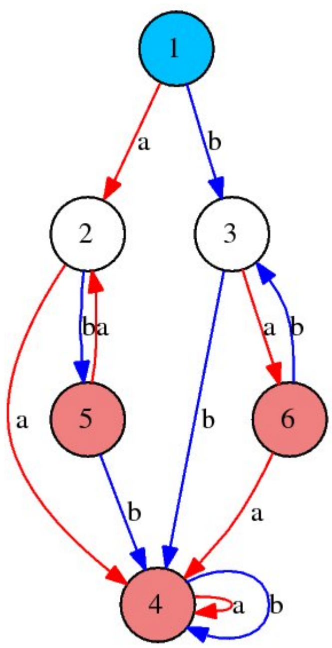
The following appears in the GAP shell:

```
gap> fxsgp:=FreeMonoid("a","b");;
<free monoid on the generators [ a, b ]>
gap> a:=GeneratorsOfMonoid( fxsgp )[ 1 ];;
a
gap> b:=GeneratorsOfMonoid( fxsgp )[ 2 ];;
b
gap> rxsgp:=[[a*a*a,a*a],[a*a*a,a*a],[a*a*b,a*a],[b*a*a,a*a],[a*b*a,a],
[b*b*a,b*b],[a*b*b,b*b],[b*b*b,b*b],[b*b*b,b*b],[b*a*b,b]];;
[ [ a^3, a^2 ], [ a^3, a^2 ], [ a^2*b, a^2 ], [ b*a^2, a^2 ], [ a*b*a, a ],
[ b^2*a, b^2 ], [ a*b^2, b^2 ], [ b^3, b^2 ], [ b^3, b^2 ], [ b*a*b, b ] ]
gap> b21:=fxsgp/rxsgp;
<fp monoid on the generators [ a, b ]>
gap>
```

(Of course, this gives an alternative way to give the same semigroup to GAP.)
Now one may use the GAP shell to perform computations, but (for some) one may use the Tcl/Tk graphical interface as well.

```
gap> Elements(b21);
[ <identity ...>, a, b, a^2, a*b, b*a ]
```

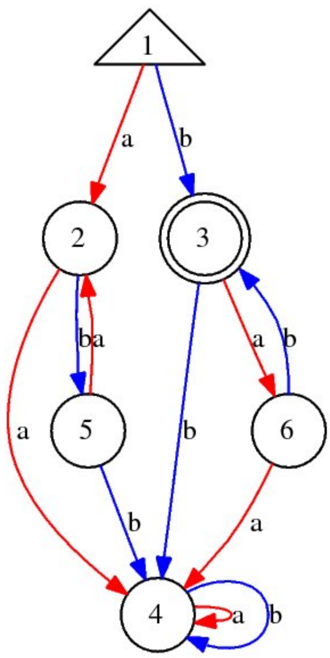

By typing “DrawRightCayleyGraph(b21);” in the GAP command line, a picture of the right Cayley graph of the Brandt monoid pops up. The same may be obtained by pressing the “Draw Cayley Graph” bottom in the “Functions” menu of the Tcl/Tk interface.



If we want to compute $\varphi^{-1}(b)$ we should start by turning the states 1 and 3, respectively, into the initial and the final state.

```
gap> rcg := RightCayleyGraph(b21);;
gap> SetInitialStatesOfAutomaton(rcg,1);;
gap> SetFinalStatesOfAutomaton(rcg,3);;
gap> DrawAutomaton(rcg);
```

A document viewer containing the following image pops up.



A rational expression for $\varphi^{-1}(b)$:

```
gap> AutomatonToRatExp(rcg);
b(ab)*
```

Its commutative image is the semilinear set $(0,1) \cup ((1,2) + (1,1)\mathbb{N})$.

Proposition 3.7

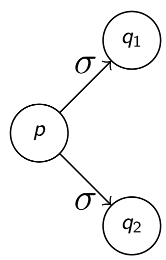
The Ab-closure of a semilinear set may be obtained by replacing the $\mathbb{N}$'s by $\mathbb{Z}$'s.

Thus, the Ab-closure of a semilinear set is a finite union of cosets of subgroups of the free abelian group.
For this case we obtain $(1,2) + (1,1)\mathbb{Z}$.

```
gap> FAtoZSmlExp(rcg);
[ < Z-linear subset of $ZZ^2$ > ]
gap> Display(last[1]);
[ 0, 1 ] + [ 1, 1 ] Z
```


Non-deterministic automata

If we do not require a single initial state and permit the graph describing the automaton to have a configuration like the following



ceases the determinism, in the sense that after reaching a state p, the reading of a letter σ may lead to any one of several states.

In a deterministic automaton this can not occur, since δ is required to be a partial function ($\delta(p,\sigma)$ has, at most,one image).

In this more general setting, we may think of δ as a function $\delta : Q \times \Sigma^* \to 2^Q$ (or as a relation $\delta \subseteq (Q \times \Sigma^*, Q)$).

We define a **non-deterministic automaton** over a finite alphabet Σ as a vector (Q, Σ, I, F, E), with Q a set (the **set of states**), Σ the **automaton alphabet**, I and F subsets of Q (said respectively **set of initial states** and **set of final states**) and $E \subseteq Q \times \Sigma \times Q$ a set of **edges**.

Often we will just say **automaton** without specifying whether the automaton is deterministic or not (we'll see later that both recognize the same languages).

An automaton is said to be **finite** if it has a finite number of states.

A **path** in an automaton $\mathcal{A} = (Q, \Sigma, I, F, E)$ is a sequence $c = (e_i)_{1 \leq i \leq n}$ of consecutive edges $e_i = (q_i, \sigma_i, q_{i+1})$.

The word $w = \sigma_1 \cdots \sigma_n$ is said the **label** of the path. The vertex q_1 is said the **beginning** of the path, and q_{n+1} it's **end**. The integer n is said to be the **length** length of the path.

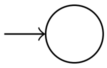
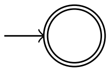
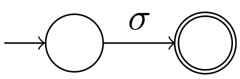
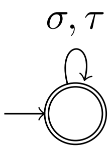
A path that goes from an initial state to a final state is said to be **successful**. A word w is said to be **recognized** by an automaton $\mathcal{A}$ if it is the label of a successful path. The set of all words recognized by an automaton $\mathcal{A}$ is precisely the **language recognized** by $\mathcal{A}$. We denote it by $L(\mathcal{A})$.

A Σ^*-automaton $\mathcal{A} = (Q, I, F, E)$ is said to be **complete** if, for any pair $(p, \sigma) \in Q \times \Sigma$, there exists **at least** a state $q \in Q$ such that $(p, \sigma, q) \in E$.

Note that $L(\mathcal{A})$ is independent of the name of the states. Thus, in a graphical description of an automaton, we may not indicate the names of the vertices.

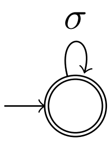
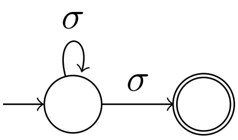
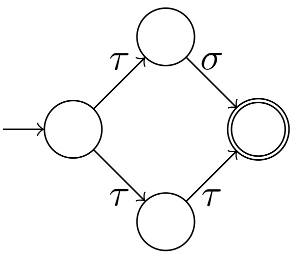
Examples

Let $\Sigma = \{\sigma, \tau\}$.

1. $\mathcal{A}$:  : $L(\mathcal{A}) = \emptyset$;
2. $\mathcal{A}$:  : $L(\mathcal{A}) = 1$;
3. $\mathcal{A}$:  : $L(\mathcal{A}) = \sigma$;
4. $\mathcal{A}$:  : $L(\mathcal{A}) = \Sigma^*$;

GAP Semigroups Automata Automata (II) Semigroups (II) NS References

Non-deterministic automata
varia
Kleene's Theorem
Applications

5. $\mathcal{A}$:  : $L(\mathcal{A}) = \sigma^*$;
6. $\mathcal{B}$:  : $L(\mathcal{B}) = \sigma^+$;
7. $\mathcal{B}$:  : $L(\mathcal{B}) = \tau\sigma \cup \tau^2$.

Since any deterministic Σ^*-automaton is also non-deterministic, we have that the class of the Σ^*-languages recognized by non-deterministic Σ^*-automata contains $\mathrm{Rec}\,\Sigma^*$. Next we will prove that it is exactly $\mathrm{Rec}\,\Sigma^*$. To this effect we will make use of the **subset construction** indicated in what follows.

Let $\mathcal{A} = (Q, \Sigma, I, F, E)$ be a non-deterministic Σ^*-automaton. We define the Σ^*-automaton $\hat{\mathcal{A}} = (\hat{Q}, \Sigma, \hat{i}, \hat{F}, \hat{E})$ by:

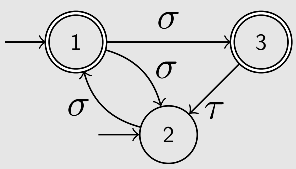
- $\hat{Q} = 2^Q = \{P \mid P \subseteq Q\}$;
- $\hat{i} = I$;
- $\hat{F} = \{P \subseteq Q \mid P \cap F \neq \emptyset\}$;
- $\hat{E} = \{(P, \sigma, R) \in 2^Q \times \Sigma \times 2^Q \mid R = \{q \mid (p, \sigma, q) \in E, \text{ for any } p \in P\}\}$.
  (In particular, $P\sigma = \{q \mid (p, \sigma, q) \in E, \text{ for any } p \in P\}$.)

It is straightforward that $\hat{\mathcal{A}}$ is deterministic and complete.

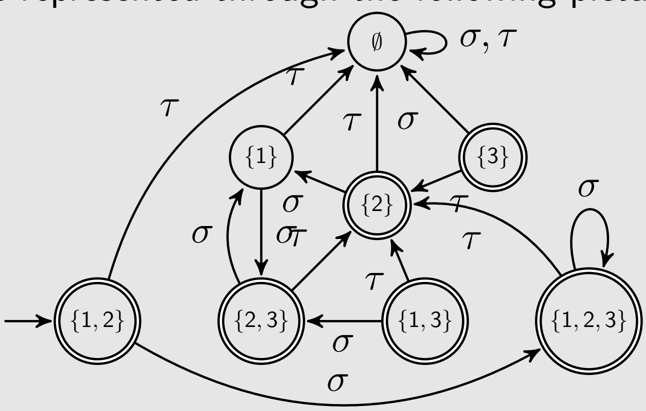
Non-deterministic automata
varia
Kleene's Theorem
Applications

Example

Let Σ be the alphabet $\{\sigma, \tau\}$ and let $\mathcal{A}$ be the following automaton.



Then $\hat{\mathcal{A}}$ may be represented through the following picture:



Proposition 4.1

Let c be a path in $\hat{\mathcal{A}}$ beginning in Q, ending in Q' and having label u. Then $Q' = \{q' \mid$ exists $q \in Q$ such that there is a path in $\mathcal{A}$ from q to q' and labeled by $u\}$.

The proof can be done easily by induction on the length of u. From the definitions, it comes out easily the following:

Corollary 4.2

One has: $L(\mathcal{A}) = L(\hat{\mathcal{A}})$.

We have thus proved:

Proposition 4.3

To any finite non-deterministic Σ^*-automaton there is a finite and complete deterministic Σ^*-automaton recognizing the same language.

Automata with ε-transitions

Next we introduce a new generalization of the concept of automaton. Again, we do not get the recognition of new languages.

Automata with ε-transitions are Σ^*-automata where edges of the form (p, ε, q) are allowed, that is, automata with ε-transitions may have edges labeled by the empty word.
For a word, being **recognized** by an automaton with ε-transitions has the obvious meaning.

As any Σ^*-automaton is an automaton with ε-transitions, we have that the languages recognized by automata are also recognized by automata with ε-transitions. In fact, they recognize the same languages, as follows from the next statement:

Proposition 4.4

Let $\mathcal{A} = (Q, I, F, E)$ be a Σ^*-automaton with ε-transitions. Then $L(\mathcal{A}) \in \mathrm{Rec}\,\Sigma^*$.

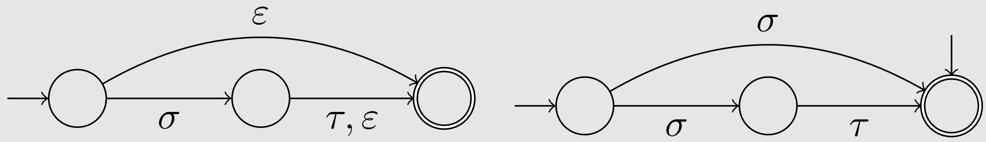
The the proof of the above proposition (which may be found in any basic textbook on automata) involves the the construction of an automaton from an automaton with ε-transitions indicated in what follows.

Let $\mathcal{A} = (Q, I, F, E)$ be a Σ^*-automaton with ε-transitions. We define the Σ^*-automaton $\mathcal{A}' = (Q, I', F, E')$ by making:

- $I' = I\varepsilon \cup I$;
- $E' = \{(p, \sigma, q') \mid (p, \sigma, q) \in E, \sigma \in \Sigma, q' \in q\varepsilon\}$.

Example

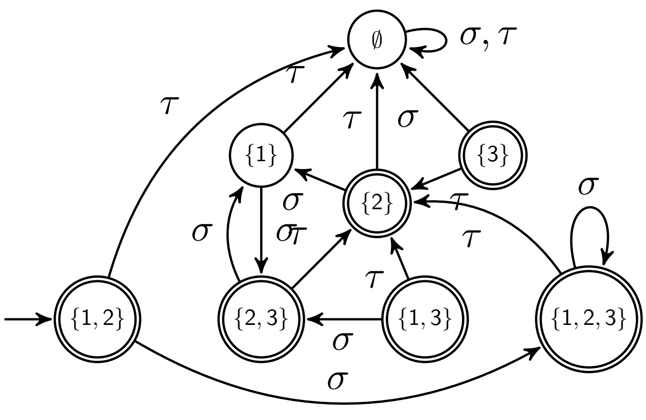
Let $\Sigma = \{\sigma, \tau\}$ and let $\mathcal{A}$ and $\mathcal{A}'$ be the following automata:



We have $L(\mathcal{A}) = L(\mathcal{A}')$.

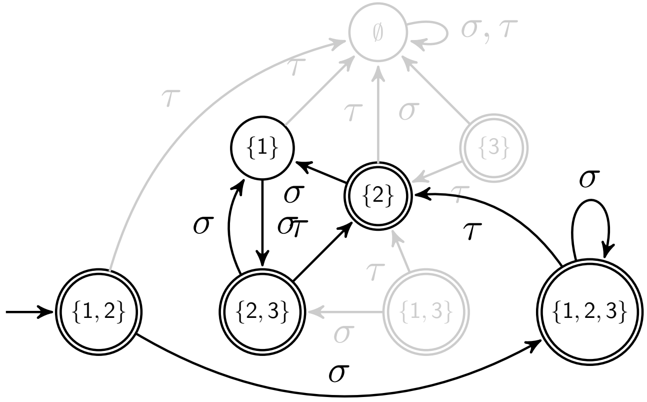
Minimal automata (motivation)

When performing the subset construction above, we have obtained the following automaton:



As we are interested in the language recognized by this automaton, the following question is natural: are there superfluous states?

In this case, the answer is "yes". It is clear that the states $\emptyset$, $\{3\}$, $\{1,3\}$ are not vertices of any successful path and thus the recognized language is not affected if we remove them.



(This suggests that one can easily improve the subset construction to obtain deterministic automata that recognize the same languages than non-deterministic ones.)

More generally, one could ask the question: given a recognizable language, is there a deterministic automaton recognizing it that is minimal in some sense?

The answer is "yes". There exists a unique (up to isomorphism) deterministic automaton that is accessible and co-accessible, with a minimum number of states that recognizes the language. One may construct such an automaton (there are various algorithms in the literature; one of them is implemented in the package "automata"). (The concept of "isomorphism" has to be defined, of course.)

Non-deterministic automata
varia
Kleene's Theorem
Applications

Transition monoid

Let $\mathcal{PT}(X)$ be the set of partial transformations on a set X.
It is easy to see that $\mathcal{PT}(X)$ with the operation $fg = g \circ f$ is a monoid, since the composition of partial functions is associative and the identity is the neutral element
The monoid $\mathcal{PT}(X)$ is known as the **monoid of partial transformations on X**.

Let $\mathcal{A} = (Q, \Sigma, i, F, E)$ be a deterministic automaton. It is immediate that the function

$$\Phi : \Sigma^* \to \mathcal{PT}(Q)$$

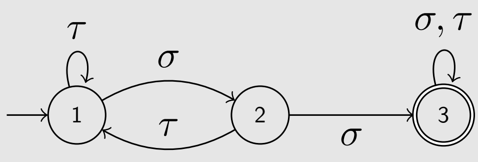
defined by $\Phi(u) = \Phi_u$, where $\Phi_u : Q \to Q$ is the partial function $\Phi_u(q) = qu$, is a monoid homomorphism.

Non-deterministic automata
varia
Kleene's Theorem
Applications

The image of Φ, $\Phi(\Sigma^*)$, is a submonoid of $\mathcal{PT}(Q)$, is called the **transition monoid of the automaton** and is denoted by $M_{\mathcal{A}}$.
Clearly, $M_{\mathcal{A}}$ is generated by the partial transformations defined by the letters.

Example

Consider the automaton
We have



δ	1	2	3
σ	2	3	3
τ	1	1	3
σ^2	3	3	3
$\sigma\tau$	1	3	3
$\tau\sigma$	2	2	3

Continuing the computation, we can observe that τ, τ^2 and $\tau\sigma\tau$ define the same mapping from Q into Q. The same happens with $\sigma^2, \sigma^3, \sigma^2\tau$ and $\tau\sigma^2$ and also with σ and $\sigma\tau\sigma$. The transition monoid of the automaton contains 6 elements, the transformations corresponding to the words $1, \sigma, \tau, \sigma^2, \sigma\tau, \tau\sigma$.
Note that the transformation corresponding to σ^2 is a zero of the monoid.

Proposition 4.5

Let $L \in \operatorname{Rec}\Sigma^*$, $L \neq \emptyset$. Then the syntactic monoid of L is isomorphic to the transition monoid of the minimal automaton of L, that is, $\Sigma^*/{\sim_L} \simeq M_{min_L}$.

Kleene's Theorem

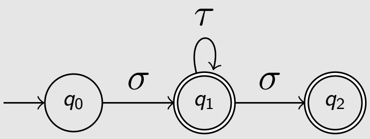
We already know that $\operatorname{Rec}\Sigma^*$ is closed under union. We have shown it using the definition of recognition by a monoid. Another way to show it is to note that if $\mathcal{A} = (Q, I, F, E)$ and $\mathcal{A}' = (Q', I', F', E')$ are automata, then the **disjoint union** $\mathcal{A}'' = (Q \dot\cup Q', I \dot\cup I', F \dot\cup F', E \dot\cup E')$ recognizes $L(\mathcal{A}) \cup L(\mathcal{A}')$.

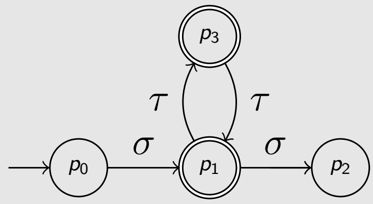
To produce similar constructions for the operators "product" and "star" we will use automata with ε-transitions.
They are left as exercises to be done during the break.
(Illustrates of the constructions are given...)

Lemma 4.6

Let $L, L' \in \mathrm{Rec}\,\Sigma^*$. *Then* $L \cdot L' \in \mathrm{Rec}\,\Sigma^*$.

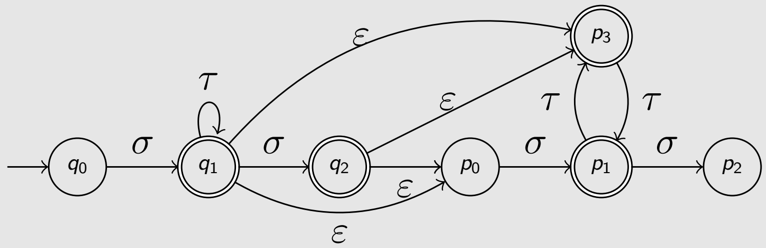
Illustration of the construction...

$\mathcal{A}$: 

$\mathcal{A}'$: 

,

Then

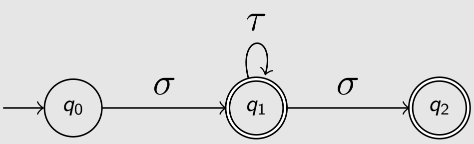


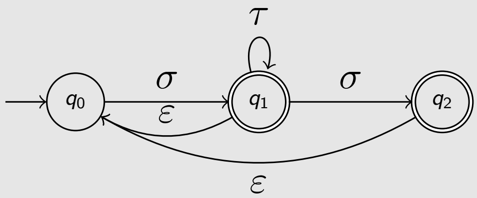
recognizes $L(\mathcal{A}) \cdot L(\mathcal{A}')$.

Lemma 4.7

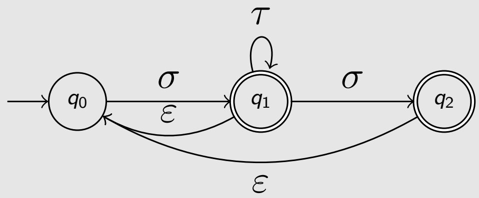
Let $L \in \mathrm{Rec}\,\Sigma^*$. *Then* $L^+, L^* \in \mathrm{Rec}\,\Sigma^*$.

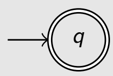
Illustration of the construction...

Let $\mathcal{A}$: 

Then 

recognizes $L(\mathcal{A})^+$ and





recognizes $L(\mathcal{A})^*$.

Generalized transition graphs

Next we introduce a new variation of the notion of automaton. The definition of **generalized transition graph** (abbreviated: GTG) $\mathcal{G}$ over a given alphabet may be obtained from the definition of automaton by requesting:

- $\mathcal{G}$ has a single initial state q_I and a single final state q_F, with $q_F \neq q_I$;
- given two states of $\mathcal{G}$ there is exactly one edge beginning in one of them and ending in the other;
- the labels of the edges of $\mathcal{G}$ are rational sets (instead of letters, as happens in the automata case) or, more precisely, rational expressions representing them.

For states p, q of $\mathcal{G}$ we denote by $\lambda(p, q)$ the label of the single edge beginning in p and ending in q.

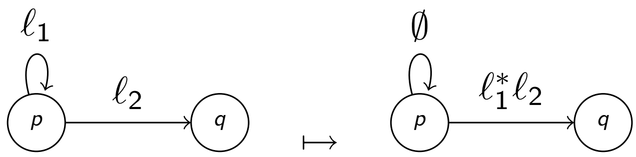
A word w is **recognized** by $\mathcal{G}$ if and only if there is a finite sequence $q_I = p_0, \ldots, p_n = q_F$ of states of $\mathcal{G}$ and a factorization $w = u_1 \cdots u_n$ of w such that, for $1 \leq i \leq n$, u_i belongs to $\lambda(p_{i-1}, p_i)$. The **language recognized** by $\mathcal{G}$ is the set of words recognized by $\mathcal{G}$.

Let $\mathcal{A}$ be a finite automaton and let Q be its set of states. The **output** of the following algorithm, whose **input** is $\mathcal{A}$, is the label of the single edge from the initial state to the final state of the GTG obtained at the end. It will be a rational expression for the language recognized by $\mathcal{A}$.

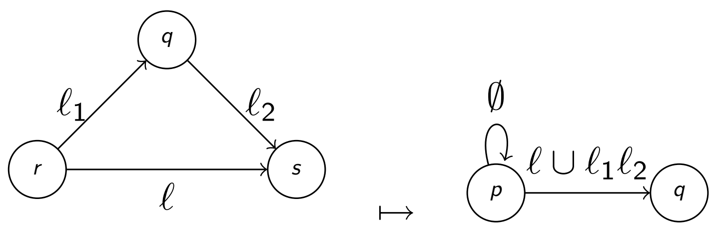
Algorithm 4.8

- *Construct the following generalized transition graph $\mathcal{G}$: the set of states is $Q' = Q \cup \{q_I, q_F\}$ where q_I and q_F do not belong to Q; the edges are labeled in the following way:*
  the label of an edge from q_I to any initial state of $\mathcal{A}$ is the empty word and the same happens with the label of the edge from any final state of $\mathcal{A}$ to q_F. The remaining edges adjacent to q_I or to q_F are labeled by $\emptyset$, the empty set. The label $\lambda(p,q)$, $p,q \in Q$, is the set (eventually $\emptyset$) of letters labeling the edges from p to q in $\mathcal{A}$.
- *Execute the following cycle:*
  While $Q \neq \emptyset$,
  – choose $q \in Q$;
  – **destroy** *the loop in q, then* **eliminate** *q.*

The way the **destruction** of a loop and the **elimination** of a state are performed is indicated in what follows.
Destroy a loop at q: if the label of the loop at q is non-empty and $p \in Q \setminus \{q\}$, replace the label $\lambda(q,p)$ of the edge from q to p by $(\lambda(q,q))^*\lambda(q,p)$ and the label $\lambda(q,q)$ of the loop in q by $\emptyset$.



Eliminate a state q (with $\lambda(q,q) = \emptyset$): for all states r, s of $\mathcal{G}$ such that $r, s \neq q$, the label $\lambda(r,s)$ of the edge from r to s is replaced by $\lambda(r,q)\lambda(q,s) \cup \lambda(r,s)$. The state q and all edges adjacent to q are then removed.



We observe that the language recognized by the generalized transition graph $\mathcal{G}$ constructed in the first step of the algorithm is precisely the language recognized by $\mathcal{A}$. We observe also that the language recognized by a GTG obtained from a GTG by destruction of a loop at some state followed by the elimination of this state is the language recognized by the original GTG. So, the output of Algorithm 4.8 is a rational expression for the language recognized by the automaton $\mathcal{A}$ given.

GAP Semigroups Automata Automata (II) Semigroups (II) NS References

Non-deterministic automata
varia
Kleene's Theorem
Applications

Theorem 4.9 (Kleene's Theorem)

Let Σ be a finite alphabet. Then

$$\mathrm{Rat}\,\Sigma^* = \mathrm{Rec}\,\Sigma^*.$$

Proof. To prove that $\mathrm{Rat}\,\Sigma^* \subseteq \mathrm{Rec}\,\Sigma^*$, it suffices to observe that $\emptyset, \{\sigma\} \in \mathrm{Rec}\,\Sigma^*$, for any $\sigma \in \Sigma$, and that $\mathrm{Rec}\,\Sigma^*$ is closed under the operators "union", "product" e "star".

The preceding algorithm shows the other inclusion, which concludes the proof. $\square$

Piecewise testable languages: Simon's theorem

A Σ^*-language is said to be **piecewise testable** if it can be obtained from languages of the form

$$\Sigma^*\sigma_1\Sigma^* \cdots \Sigma^*\sigma_n\Sigma^*, \text{ com } \sigma_1, \ldots, \sigma_n \in \Sigma$$

using a finite number of times the operators of union and complementation.

Theorem 4.10 (Simon)

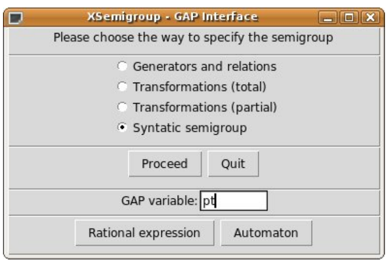
A rational language is piecewise testable if and only if its syntactic monoid is $\mathcal{J}$-trivial.

The following question is a simple exercise (assuming we have Simon's Theorem at hand).

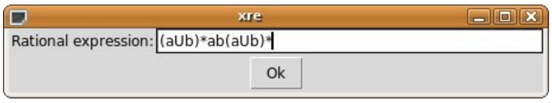
Question 4.11

Is the rational language $\Sigma^*ab\Sigma^*$ piecewise testable?

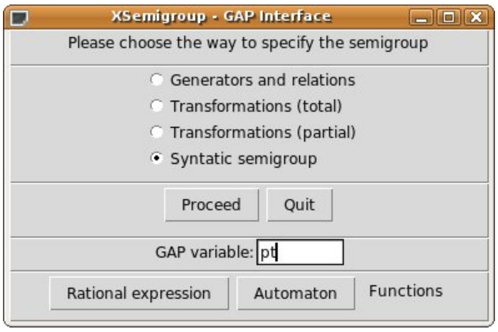
The answer is "depends"...
Answer: Volkov said it! (Proof by eminent authority...)
Alternatively, we can get convinced by using the following sequence of images obtained with the GAP packages already mentioned:



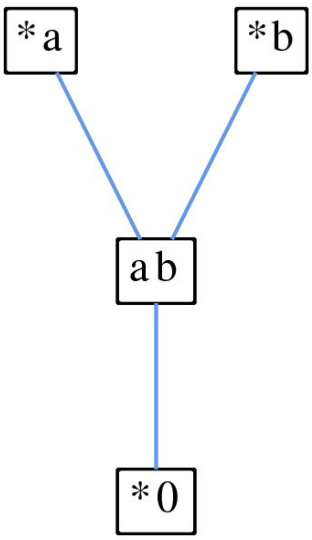
The following is popped up.



After pressing the "Ok", the "functions" bottom appears and is ready to be used.



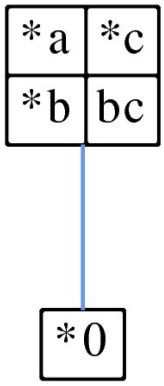
By pressing the "Draw D-Classes" bottom, we obtain the following picture



which shows that each $\mathcal{D}$-class has just an element and thus the monoid is $\mathcal{J}$-trivial.
By Simon's theorem, the language is piecewise testable.

As an alternative to the use of "XSemigroup", by writing the following sequence of commands in a GAP session, one get an image that shows that the syntactic semigroup of the language $\Sigma^* ab \Sigma^*$ is not $\mathcal{J}$-trivial and thus the language is not piecewise testable, if the alphabet has 3 letters.

```
reg := RationalExpression("(aUbUc)*ab(aUbUc)*");
autxsgp:=RatExpToAut(reg);
gap> ts := TransitionSemigroup(autxsgp);
gap> DrawDClasses(ts);
```



Definitions

A **pseudovariety** H of groups (monoids) is a class of finite groups (monoids) closed under formation of finite direct products, subgroups (submonoids) and quotients.

Given a pseudovariety H of groups, the H-**kernel** of a finite monoid S is the submonoid

$$K_{\mathsf{H}}(S) = \bigcap \tau^{-1}(1),$$

with the intersection being taken over all groups $G \in \mathsf{H}$ and all relational morphisms of monoids $\tau : S \,{-\!\!\!\circ\!\!\!\rightarrow}\, G$.

Easy consequences

Since a relational morphism into a group belonging to a certain pseudovariety H_1 of groups is also a relational morphism into a group belonging to a pseudovariety H_2 containing it, the following fact follows.

Fact 5.1

Let M be a finite monoid and let H_1 and H_2 be pseudovarieties of groups such that $\mathsf{H}_1 \subseteq \mathsf{H}_2$. Then $\mathsf{K}_{\mathsf{H}_2}(M) \subseteq \mathsf{K}_{\mathsf{H}_1}(M)$.

Proposition 5.2 (˜ , 98)

Let G be a group and H a pseudovariety of groups. Then $\mathsf{K}_{\mathsf{H}}(G)$ is the smallest normal subgroup of G such that $G/\mathsf{K}_{\mathsf{H}}(G) \in \mathsf{H}$.

Corollary 5.3

Any relative abelian kernel of a finite group contains its derived subgroup.

As the restriction $\tau_|$ of a relational morphism $\tau : S \rightarrow G$ to a subsemigroup T of S is a relational morphism $\tau_| : T \rightarrow G$, we have the following:

Fact 5.4

If T is a subsemigroup of a finite semigroup S, then $\mathsf{K}_{\mathsf{H}}(T) \subseteq \mathsf{K}_{\mathsf{H}}(S)$.

Let e be an idempotent of a finite semigroup S. As for every relational morphism $\tau : S \rightarrow G$ into a group G we have $\tau(e)\tau(e) \subseteq \tau(e)$, we get that $\tau(e)$ is a subgroup of G.

It follows that $e \in \tau^{-1}(1)$. If $x, y \in \tau^{-1}(1)$, then $1 \in \tau(x)\tau(y) \subseteq \tau(xy)$, therefore $xy \in \tau^{-1}(1)$, thus $\tau^{-1}(1)$ is a subsemigroup of S containing the idempotents.

As the non-empty intersection of subsemigroups is a subsemigroup, we have the following fact.

Fact 5.5

Let H be a pseudovariety of groups and let M be a finite monoid. The relative kernel $\mathsf{K}_{\mathsf{H}}(M)$ is a submonoid of M containing the idempotents. □

Fact 5.4 may be used to determine elements in the H-kernel of a monoid without its complete determination.
Note that, for example, if we can determine a set X of generators of a monoid M such that $X \subseteq \mathsf{K}_{\mathsf{H}}(M)$, then we can conclude by Fact 5.5 that the $M = \langle X \rangle \subseteq \mathsf{K}_{\mathsf{H}}(M)$.

Motivation

Rhodes Type II conjecture proposed an algorithm to compute $\mathsf{K}_{\mathsf{G}}(S)$, where G is the class of all finite groups and S is a given finite monoid.

Solutions were given by Ash and by Ribes and Zalesskiĭ in the early nineties.

Pin showed that the problem of computing $\mathsf{K}_{\mathsf{G}}(S)$ can be reduced to that of computing the closure (relative to the profinite topology) of a rational subset of the free group. This approach led to the solution given by Ribes Ribes and Zalesskiĭ.

Algorithms to compute other relative kernels (e.g., kernels relative to pseudovarieties of p-groups and pseudovarieties of abelian groups) followed the idea of Pin.

A different algorithm has been given by Steinberg.

The Mal'cev product, when the rightmost factor is a pseudovariety of groups, may be defined as follows: for a pseudovariety V of monoids and a pseudovariety H of groups, the **Mal'cev product** of V and H is the pseudovariety

$$\mathsf{V} \circledcirc{m} \mathsf{H} = \{S \mid \mathsf{K}_{\mathsf{H}}(S) \in \mathsf{V}\}.$$

Algorithms to compute relative kernels may lead to decidability results.

On the algorithms

Let M be a finite n-generated monoid. There exists a finite ordered set A of cardinality n and a surjective homomorphism $\varphi : A^* \to M$ from the free monoid on A onto M.

Proposition 5.6 (Pin, 88)

Let $x \in M$. Then $x \in \mathsf{K}_{\mathsf{G}}(M)$ if and only if $1 \in \mathsf{Cl}_{\mathsf{G}}(\varphi^{-1}(x))$ (the closure is taken for the profinite group topology of A^*).

Commutative images of languages in A^* are used for the abelian kernel case, that is, the canonical homomorphism $\gamma : A^* \to \mathbb{Z}^n$ defined by $\gamma(a_i) = (0, \ldots, 0, 1, 0, \ldots, 0)$ (1 in position i), where a_i is the i^{th} element of A, is considered.

Proposition 5.7 (˜ , 98)

Let $x \in M$. Then $x \in \mathsf{K}_{\mathsf{Ab}}(M)$ if and only if $0 \in \mathsf{Cl}_{\mathsf{Ab}}(\gamma(\varphi^{-1}(x)))$.

This proposition, similar to the former one of Pin, leads to an algorithm to compute the abelian kernel of a finite monoid.
A generalization, to all pseudovarieties of abelian groups, was obtained by Steinberg.

A **supernatural number** is a formal product of the form

$$\prod p^{n_p}$$

where p runs over all positive prime numbers and $0 \leq n_p \leq +\infty$.

To a supernatural number π one associates the pseudovariety H_π generated by the cyclic groups $\{\mathbb{Z}/n\mathbb{Z} \mid n \text{ divides } \pi\}$.

- $\mathsf{H}_{2^{+\infty}}$ is the pseudovariety of all 2-groups which are abelian;
- to the supernatural number $\prod p^{+\infty}$, where p runs over all positive prime numbers, is associated the pseudovariety Ab of all finite abelian groups.

Proposition 5.8 (Steinberg, 99)

Let π be an infinite supernatural number and let $x \in M$. Then $x \in \mathsf{K}_{\mathsf{H}_\pi}(M)$ if and only if $0 \in \mathsf{Cl}_{\mathsf{H}_\pi}(\gamma(\varphi^{-1}(x)))$.

As a way to compute (a rational expression for) $\varphi^{-1}(x)$ one can consider the automaton $\Gamma(M, x)$ obtained from the right Cayley graph of M by taking the neutral element as the initial state and x as final state. Note that the language of $\Gamma(M, x)$ is precisely $\varphi^{-1}(x)$.

This motivated the appearance of the GAP package "automata", a GAP package to deal with finite state automata.

There exist implementations in GAP of the mentioned algorithms to compute kernels of finite monoids relative to G, Ab, H_π and G_p.

The first ones follow the above strategy, while the implemented algorithm to compute kernels relative to G_p is due to Steinberg. It has been achieved with the collaboration of J. Morais and benefits also of the existence of the package "automata".

Definitions and easy consequences
Motivation
On the algorithms
Relative kernels and solvability

The usefulness of visualizing the results motivated the GAP package "sgpviz".

Definitions and easy consequences
Motivation
On the algorithms
Relative kernels and solvability

Given a finite group G and a positive integer k, denote by $G^{[k]}$ the subgroup of G generated by the commutators of G and by the the elements of the form x^k, $x \in G$. In other words, let $G^{[k]}$ be the smallest subgroup of G containing the derived subgroup G' and the k-powers.

Jointly with Cordeiro and Fernandes for finite superatural numbers and with Cordeiro for the general case, we obtained:

Proposition 5.9

Let π be a supernatural number, G a finite group and le $k = gcd(|G|, \pi)$. Then we have: $\mathsf{K}_{\mathsf{H}_\pi}(G) = G^{[k]}$.

We define recursively $\mathsf{K}^n_{\mathsf{H}}(S)$ as follows:

- $\mathsf{K}^0_{\mathsf{H}}(S) = S$;
- $\mathsf{K}^n_{\mathsf{H}}(S) = \mathsf{K}_{\mathsf{H}}(\mathsf{K}^{n-1}_{\mathsf{H}}(S))$, for $n \geq 1$.

Since S is finite and the operator K_{H} is non-increasing, it follows that the sequence $\mathsf{K}^n_{\mathsf{H}}(S)$ is eventually constant; we denote this constant value by

$$\mathsf{K}^\omega_{\mathsf{H}}(S).$$

Observe that $\mathsf{K}^\omega_{\mathsf{H}}(S)$ is the largest subsemigroup of S fixed by K_{H}.

For a pseudovariety V and $n \geq 0$, we define the operator $(-)^n ⓜ \mathsf{H}$ recursively as follows:

- $\mathsf{V}\ {}^0ⓜ\ \mathsf{H} = \mathsf{V}$;
- $\mathsf{V}\ {}^{n+1}ⓜ\ \mathsf{H} = (\mathsf{V}\ {}^nⓜ\ \mathsf{H})\ ⓜ\ \mathsf{H}$;
- $\mathsf{V}\ {}^\omega ⓜ \mathsf{H} = \cup_{n\geq 0} \mathsf{V}\ {}^nⓜ\ \mathsf{H}$.

Definitions and easy consequences
Motivation
On the algorithms
Relative kernels and solvability

It is easy to see

$\mathsf{V}\ {}^nⓜ\ \mathsf{H} = \{S \mid \mathsf{K}^n_{\mathsf{H}}(S) \in \mathsf{V}\}$ and $\mathsf{V}\ {}^\omega ⓜ \mathsf{H} = \{S \mid \mathsf{K}^\omega_{\mathsf{H}}(S) \in \mathsf{V}\}$.

In a joint work with Fernandes (2005), a semigroup was defined to be H-**solvable** if iterating the H-kernel operator eventually arrives at the subsemigroup generated by the idempotents.

A semigroup with commuting idempotents has been proved to be Ab-solvable if and only if its subgroups are solvable groups.

A much more general result has then been obtained in a joint work with Fernandes, Margolis and Steinberg (2004). It states that:

for a non-trivial pseudovariety H of groups, a semigroup with an aperiodic idempotent-generated subsemigroup is H-solvable if and only if it subgroups are H-solvable.

We proved, in particular, that

$$\mathsf{EA} = \mathsf{A}\ {}^{\omega}\!\textcircled{m}\mathsf{G}$$

where we denote by EA the pseudovariety consisting of all monoids whose idempotents generate an aperiodic submonoid

By using a modification of the technique, it has been shown in a joint work with Steinberg that:

a semigroup S is H-solvable if and only if, for each idempotent $e \in S$, there is a subnormal series with smallest element the maximal subgroup at e of the idempotent-generated subsemigroup of S and largest element the maximal subgroup of S at e such that the successive quotients belong to H.

Motivation

Let us consider an automatic selling machine

Prices:

a: 1 Cent

b: 2 Cent

c: 3 Cent

...

x: 1.000.000.000 EUR

WARNING:
Exact amount, please

Suppose that we have infinitely many coins

Question

Which is the price of the most expensive product that we cannot buy?

Let us consider

$$\begin{aligned} S \quad = \quad <3,5> &= \{x\cdot 3 + y \cdot 5 \mid x, y \in \mathbb{N}_0\} \\ &= \{0,3,5,6,8,9,10,\rightarrow\} \end{aligned}$$

S is a **numerical semigroup**, i.e. a co-finite submonoid of $(\mathbb{N}_0,+)$.

- $m(S) = 3$ is the **multiplicity** of S.
- $H(S) = \mathbb{N} \setminus S = \{1,2,4,7\}$ is the set of **gaps** of S.
- $F(S) = maxH(S) = 7$ is the **Frobenius number** of S.

So, the answer to our question is: 7.

If S is a numerical semigroup, then the greatest integer that is not in S is called the **Frobenius number** of S and is denoted by $F(S)$.

There exists a formula for the Frobenius number of a numerical semigroup generated by two elements...

If we had the following coins, how could we answer the question posed above?

We could install the GAP package **numericalsgps** and let the computer do the work.

Remark

Unlike what happens for a semigroup generated by two elements, no general formula is known for a $F(< a, b, c >)$.

The Frobenius problem

Remark

If S is a numerical semigroup, then the set

$$gen(S) = (S \setminus \{0\}) \setminus ((S \setminus \{0\}) + (S \setminus \{0\}))$$

is a set of generators of S, (i.e. all the elements of S may be written as non-negative integer linear combinations of elements of $gen(S)$). Moreover, $gen(S)$ is finite. It is a minimal set of generators of S (which is unique).

Frobenius problem

Given positive integers $a_1, \ldots, a_n$, with $gcd(a_1, \ldots, a_n) = 1$, which is the greatest integer that cannot be written as a positive linear combination of $a_1, \ldots, a_n$?

The Frobenius problem, although apparently too specialised, appears naturally in several areas of mathematics.

So, to find "formulas" or "eficient algorithms" even for some particular cases may have some interest.

Proposition 6.1 (Sylvester)

Let p, q be relatively prime positive integers. Then

$$F(< p, q >) = pq - p - q$$

For much more on the Frobenius problem, one may consult a book by Ramírez Alfonsín.

Proportionally modular numerical semigroups

Let a, b, c be positive integers. The set

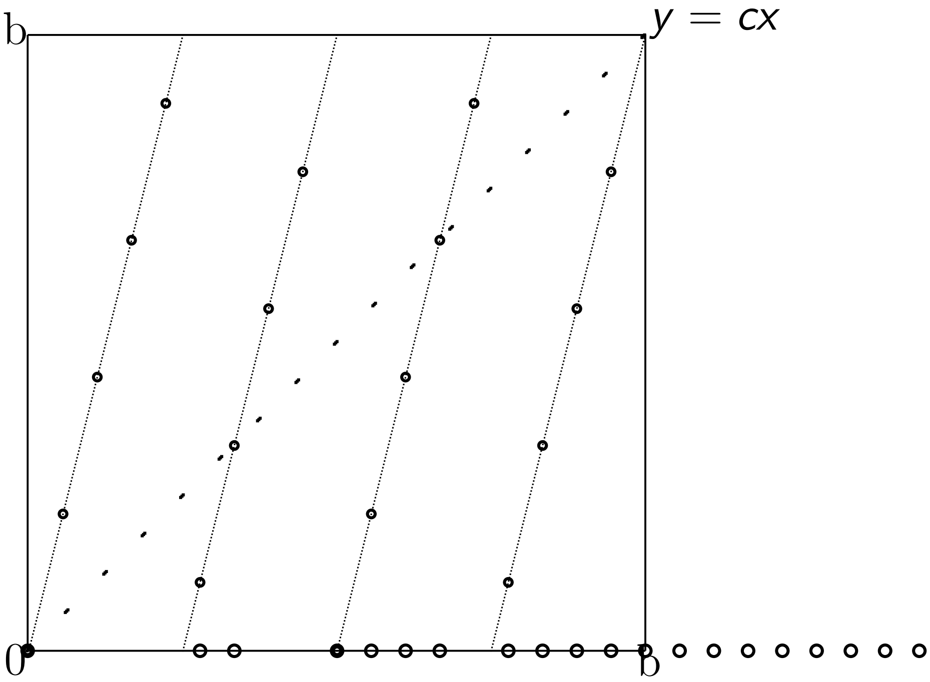
$$S(a, b, c) = \{x \in \mathbb{Z} \mid ax \mod b \leq cx\}$$

is a numerical semigroup. A semigroup of this form is said to be **proportionally modular**.

As the inequality $ax \mod b \leq cx$ has precisely the same integer solutions than the inequality $(a \mod b)x \mod b \leq cx$, we do not loose generality by supposing that $a < b$. If $c \geq a$, then $\mathrm{S}(a, b, c) = \mathbb{N}$, thus we may also suppose that $c < a$.
It is not difficult to show that $\mathrm{S}(a, b, c) = \mathrm{S}(b + c - a, b, c)$, which has as a consequence that we may also suppose that $a \leq \frac{b+c}{2}$.

Example 6.2

$S(4, 18, 1) = \{0, 5, 6, 9, 10, 11, 12, 14, \rightarrow\}$.



For a rational number r, $\lceil r \rceil$ denotes the least integer not smaller than r and $\lfloor r \rfloor$ denotes the greatest integer not bigger than r.
Ona can show:

Theorem 6.3

$$F(\mathrm{S}(a, b, c)) \in \left\{ b - \left\lfloor \frac{kb}{a} \right\rfloor - 1 \mid k \in \{1, \ldots, a-1\} \right\}.$$

Let

$$\xi = \min \left\{ k \in \{1, \ldots, a-1\} \mid kb \mod a + \left\lfloor \frac{kb}{a} \right\rfloor c > (c-1)b + a - c \right\}.$$

Corollary 6.4

$$F(\mathrm{S}(a, b, c)) = b - \left\lfloor \frac{\xi b}{a} \right\rfloor - 1.$$

The preceding corollary gives an algorithm to compute the Frobenius number of a proportionally modular Diophantine inequality. Note that one has to do at most $a - 1$ tests and recall that we may suppose that $a \leq \frac{b+c}{2}$.

Motivation
PM semigroups
A database

This algorithm can be improved:

Algorithm 6.5

INPUT: *positive integers* a, b *and* c.
OUTPUT: $F(\mathrm{S}(a,b,c))$.

1. *If* $a \leq c$, *then* return -1.
2. $a := a \mod b$.
3. *If* $a = 0$, *then* return -1.
4. *If* $a > \frac{b+c}{2}$, *then* $a := b + c - a$.
5. *If* $a = c + 1$, *then* return $g = b - \left\lfloor \frac{(a-1)b}{a} \right\rfloor - 1$.
6. *Compute* $\alpha = \left\lceil a - \frac{a}{c} - \frac{a}{b} + \frac{2a}{cb} \right\rceil$.
7. while $\alpha b \mod a + \left\lfloor \frac{\alpha b}{a} \right\rfloor c \leq (c-1)b + a - c$ do $\alpha := \alpha + 1$.
8. return $b - \left\lfloor \frac{\alpha b}{a} \right\rfloor - 1$.

GAP Semigroups Automata Automata (II) Semigroups (II) NS References

Motivation
PM semigroups
A database

After obtaining an upper bound for ξ, we have been able to give a formula to compute the Frobenius number of a large family of proportionally modular numerical semigroups.

Theorem 6.6

If $a(a-c) < bc$ *and* $\alpha = \left\lceil a - \frac{a}{c} - \frac{a}{b} + \frac{2a}{cb} \right\rceil$, *then* $F(\mathrm{S}(a,b,c)) =$

$$\begin{cases} b - \left\lfloor \frac{\alpha b}{a} \right\rfloor - 1 & \text{if} \quad \alpha b \mod a + \left\lfloor \frac{\alpha b}{a} \right\rfloor c > (c-1)b + a - c; \\ b - \left\lfloor \frac{(\alpha+1) b}{a} \right\rfloor - 1 & \text{if} \quad \alpha b \mod a + \left\lfloor \frac{\alpha b}{a} \right\rfloor c \leq (c-1)b + a - c \\ & \quad < (\alpha+1) b \mod a + \left\lfloor \frac{(\alpha+1) b}{a} \right\rfloor c; \\ b - \left\lfloor \frac{(\alpha+2) b}{a} \right\rfloor - 1 & \text{otherwise}. \end{cases}$$

Proposition 6.7

If b is a multiple of a, then $F(\mathrm{S}(a,b,c)) = b - \left\lfloor \frac{\gamma b}{a} \right\rfloor - 1$, *where* $\gamma = \left\lfloor a - \frac{a}{c} + \frac{a^2}{bc} - \frac{a}{b} + 1 \right\rfloor$.

These results appeared have been obtained in a joint work with Rosales.

A database on numerical semigroups

Given a numerical semigroup S, $g(S) = \#H(S)$ is said to be the **genus** of S.

Remark

[Rosales & García-Sánchez] Numerical semigroups of genus $n+1$ are obtained from numerical semigroups of genus n by removing a minimal generator greater than the Frobenius number.

Example

$S =< 2, 3 >$ is the only numerical semigroup of genus 1. As $F(S) = 1$, the semigroups of genus 2 are $S \setminus \{2\}$ and $S \setminus \{3\}$

We want to create a database of numerical semigroups of genus smaller than a certain number.

Remark

- If $g > m(S)$, then $m(S \setminus \{g\}) = m(S)$.
- If $g = m(S)$, then $m(S \setminus \{g\}) = m(S) + 1$.
- $m(S) \leq g(S) + 1$.

As a consequence, the computation can be "naively" parallelized: Suppose we have computed all the numerical semigroups of genus $n - 1$. Then we can put n processors together to calculate the numerical semigroups of genus n:

- the first processor calculates the numerical semigroup(s) of multiplicity 2 and genus n;
- the second processor calculates the numerical semigroups of multiplicity 3 and genus n;
- ...

- ...
- the i^{th} processor calculates the numerical semigroup(s) of multiplicity $i + 1$ and genus n;
- ...
- the n^{th} processor calculates the numerical semigroup(s) of multiplicity $n + 1$ and genus n;

We can keep on going, adding a processor from time to time...

Remark

The heavy work occurs around multiplicity $n/2$.

Let n_g and npm_g denote the number of numerical semigroups of genus g and the number of proportionally modular numerical semigroups, respectively.

A table...

g	n_g	$n_{g-1}+n_{g-2}$	$\frac{n_{g-1}+n_{g-2}}{n_g}$	$\frac{n_g}{n_{g-1}}$	npm_g	$\frac{npm_g}{n_g}$	$\frac{npm_g}{npm_{g-1}}$
2	2	2	1	2	2	1	2
3	4	3	0.75	2	4	1	2
4	7	6	0.8571428	1.75	6	0.8571428	1.5
5	12	11	0.9166666	1.714285	9	0.75	1.5
6	23	19	0.8260869	1.916666	15	0.6521739	1.666666
7	39	35	0.8974358	1.695652	18	0.4615384	1.2
8	67	62	0.9253731	1.717948	22	0.3283582	1.222222
9	118	106	0.8983050	1.761194	32	0.2711864	1.454545
10	204	185	0.9068627	1.728813	36	0.1764705	1.125
11	343	322	0.9387755	1.681372	42	0.1224489	1.166666
12	592	547	0.9239864	1.725947	57	0.0962837	1.357142
13	1001	935	0.9340659	1.690878	58	0.0579420	1.017543
14	1693	1593	0.9409332	1.691308	69	0.0407560	1.189655
15	2857	2694	0.9429471	1.687536	87	0.0304515	1.260869
16	4806	4550	0.9467332	1.682184	93	0.0193508	1.068965
17	8045	7663	0.9525170	1.673949	105	0.0130515	1.129032
18	13467	12851	0.9542585	1.673958	125	0.0092819	1.190476
19	22464	21512	0.9576210	1.668077	130	0.0057870	1.04

g	n_g	$n_{g-1}+n_{g-2}$	$\frac{n_{g-1}+n_{g-2}}{n_g}$	$\frac{n_g}{n_{g-1}}$	npm_g	$\frac{npm_g}{n_g}$	$\frac{npm_g}{npm_{g-1}}$
20	37396	35931	0.9608246	1.664707	145	0.0038774	1.115384
21	62194	59860	0.9624722	1.663119	169	0.0027173	1.165517
22	103246	99590	0.9645894	1.660063	173	0.0016756	1.023668
23	170963	165440	0.9676947	1.655880	188	0.0010996	1.086705
24	282828	274209	0.9695256	1.654322	224	0.0007920	1.191489
25	467224	453791	0.9712493	1.651972	218	0.0004665	0.973214
26	770832	750052	0.9730421	1.649812	238	0.0003087	1.091743
27	1270267	1238056	0.9746423	1.647916	275	0.0002164	1.155462
28	2091030	2041099	0.9761213	1.646134	273	0.0001305	0.992727
29	3437839	3361297	0.9777354	1.644088	303	0.0000881	1.109890
30	5646773	5528869	0.9791201	1.642535	359	0.0000635	1.184818
31	9266788	9084612	0.9803409	1.641076	353	0.0000380	0.983286
32	15195070	14913561	0.9814736	1.639734	375	0.0000246	1.062322
33	24896206	24461858	0.9825536	1.638439	401	0.0000161	1.069333
34	40761087	40091276	0.9835673	1.637240	405	0.0000099	1.009975

Remark

- M. Bras-Amorós conjectures that the number of numerical semigroups of a given genus has a Fibonacci-like behavior:

$$\frac{n_{g-1}+n_{g-2}}{n_g} \to_g 1$$

- it is not even known whether $n_{g+1} > n_g$;
- it seems that $n_{g+1} \geq (1.6)n_g$. This implies that $n_{g+k} \geq (1.6)^k n_g$ and, thus the growth of the number of numerical semigroups of a given genus is exponential.

Software references

M. Delgado, S. Linton and J. Morais, *Automata: a* GAP *package on finite automata*. (http://www.gap-system.org/Packages/automata.html).

M. Delgado, P. A. García-Sánchez and J. Morais, "numericalsgps": a GAP package on numerical semigroups. (http://www.gap-system.org/Packages/numericalsgps.html).

M. Delgado and J. Morais, *SgpViz, a* GAP *[3] package to visualize finite semigroups*, (http://www.gap-system.org/Packages/sgpviz.html).

The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.4*, 2004. (http://www.gap-system.org).

Basic references

- J.M. Howie, "Fundamentals of Semigroup Theory", Oxford University Press, 1995.
- J.M. Howie, "Automata and Languages", Clarendon Press, 1991.
- J. E. Hopcroft J. D. Ullman, "Introduction to Automata Theory, Languages and Computation", Addison Wesley, 1979.
- D. C. Kozen, "Automata and Computability", Springer, 1997.
- G. Lallement, "Semigroups and Combinatorial Applications", John Wiley & Sons, New York, 1979.
- J.-E. Pin, "Varieties of Formal Languages", Plenum, London, 1986.

GAP Semigroups Automata Automata (II) Semigroups (II) NS References

Software references
Basic references
Not so basic references
Specific references

Not so basic references

- J. Almeida, "Finite Semigroups and Universal Algebra", World Scientific, Singapore, 1995.
- J. Rhodes and B. Steinberg, "The q-theory of finite semigroups", Springer Monographs in Mathematics, 2009
- J.C. Rosales and P. A. García-Sánchez, "Numerical Semigroups", Springer. To appear.
- J. L. Ramírez Alfonsín, "The Diophantine Frobenius Problem", *Oxford Lectures Series in Mathematics and its Applications* **30**, Oxford University Press, (2005).

Specific references

- M. Bras-Amorós, *Fibonacci-like behavior of the number of numerical semigroups of a given genus*, Semigroup Forum, 76 (2008) 379–384.

- E. Cordeiro, M. Delgado and V.H. Fernandes, *Relative abelian kernels of some classes of transformation monoids*, Bull. Austral. Math. Soc. **73** (2006) 375–404.

- M. Delgado, *Abelian pointlikes of a monoid*, Semigroup Forum **56** (1998) 339–361.

- M. Delgado, V.H. Fernandes, S. Margolis and B. Steinberg, *On semigroups whose idempotent-generated subsemigroup is aperiodic*, Int. J. Algebra Comput. **14** (2004) 655-665.

- M. Delgado and J. C. Rosales, On the Frobenius number of a proportionally modular Diophantine inequality. Portugaliae Mathematica, 63 (2006) 415-425.