Algorithmic aspects of finite semigroup theory, a tutorial

Jean-Éric Pin

LIAFA, CNRS and University Paris 7

6, 8, 9 September 2006, St Andrews

Outline

(1) What this tutorial is about?
(2) Complexity
(3) Presentation and Cayley graphs
(4) Green's relations
(5) Idempotents, weak inverses and inverses
(6) Blocks
(7) Syntactic preorder
(8) Other computations

Warning. In this tutorial, all semigroups are finite.

Part I

What this tutorial is about?

The aim of this tutorial is to present some algorithms to compute finite semigroups.

Programming issues, like data structures, implementation or interface will not be addressed, but most algorithms are implemented in the C-programme `semigroupe`.

This tutorial is addressed to mathematicians, not to computer scientists. For this reason, I will remind a few basic algorithms, when needed.

Computing finite semigroups

Several questions should be answered:

• How is the semigroup given? (transformation semigroup, semigroup of matrices over some (semi)ring, finite presentation, ...)

• What does one wish to compute?

• What is the complexity of the algorithms?

How is the semigroup S given?

I assume that S is a subsemigroup of a larger semigroup U (the universe), like:

- the semigroup of all transformations on a set E,
- the semigroup of $n \times n$-Boolean matrices,
- the semigroup of $n \times n$-matrices with entries in $\mathbb{Z}$,
- a set of words, with a multiplication defined on it,
- etc.

Then S is given as the subsemigroup of U generated by some set A of generators.

Part II

Complexity

Complexity means worth case complexity. The average complexity would be too difficult to define: what would be a random semigroup?

Complexity is usually measured in terms of the following parameters:

- $|S|$, the number of elements of the semigroup,
- $|A|$, the number of generators.

Occasionally, other parameters might be used: number of idempotents, number of $\mathcal{D}$-classes, etc.

Two types of complexity

Space complexity measures the amount of computer memory required to run the algorithm.

Time complexity measures the time spent by the computer to run the algorithm.

Both space and time complexity are measured as a function of the size n of the input data, but are expressed in $O(f(n))$ notation. This makes the notion robust and machine independent.

Meaning of complexity

If an $O(n)$-time algorithm takes 0.1 second on an input of size 10^5, it will spend roughly 1 second on an input of size 10^6 and 10 seconds on an input of size 10^7.

The $O(f(n))$ notation also explains why the cost of elementary operations is irrelevant. Even if your computer is twice faster as mine, computing 1000 additions will take 1000 times as much as a single addition on both computers...

Complexity allows to predict effectiveness and is surprisingly precise in practice.

Usual complexities

$\log_2 n$	3	7	10	13	16	20
n	10	10^2	10^3	10^4	10^5	10^6
$n \log_2 n$	33	664	10^4	$1.3\ 10^5$	$1.6\ 10^6$	$2\ 10^7$
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{12}	10^{12}
2^n	10^3	10^{30}	$\ldots$	$\ldots$	$\ldots$	$\ldots$

linear (time) algorithm $= O(n)$-time algorithm
quadratic (time) algorithm $= O(n^2)$-time algorithm

Practical issues about complexity

In practice, one can run within one minute:

- linear algorithms for data of size $\leqslant 2 \cdot 10^7$
- $O(n \log n)$-time algorithms for data of size $\leqslant 10^6$
- quadratic algorithms for data of size $\leqslant 10^4$

For linear time algorithms, space complexity is often the main issue and most of the time is spent on memory allocation.

Practical issues about semigroup algorithms

Two functions are given:

• one for computing the product of an element of the universe by a generator.
• one for testing the equality of two elements of the universe.

Time complexity is usually measured by the number of accesses to these two functions.

The multiplication table can be computed in quadratic time and space. Therefore all algorithms in $O(|S|^k)$ with $k > 2$ may assume that the multiplication table is known.

Part III

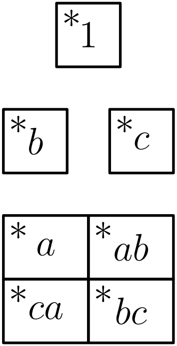
Presentation and Cayley graphs

Data: A universe and an ordered set of generators.

Output: A presentation of the semigroup by generators and relations, a confluent rewriting system for this presentation and the right and left Cayley graphs of the semigroup.

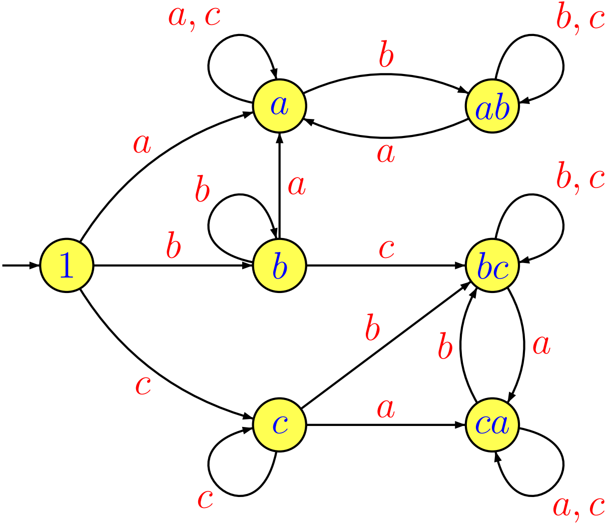
An example (input data in red)

		1	2	3
$*$	1	1	2	3
$*$	a	2	2	2
$*$	b	1	3	3
$*$	c	0	2	3
$*$	ab	3	3	3
$*$	bc	0	3	3
$*$	ca	0	2	2

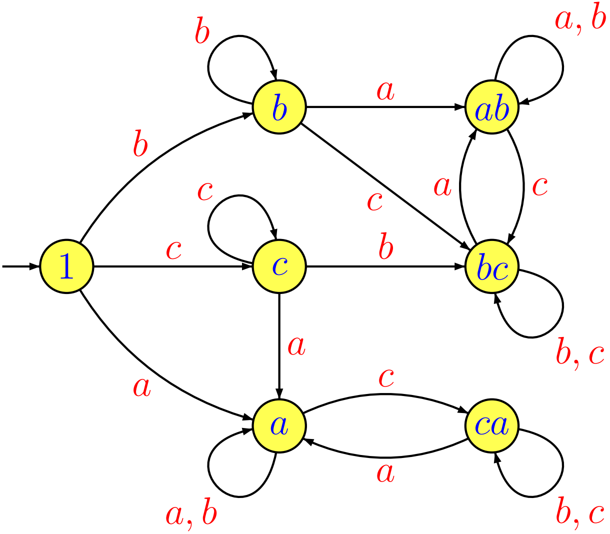


$$aa \longrightarrow a$$
$$ac \longrightarrow a$$
$$ba \longrightarrow a$$
$$bb \longrightarrow b$$
$$cb \longrightarrow bc$$
$$cc \longrightarrow c$$
$$abc \longrightarrow ab$$
$$bca \longrightarrow ca$$
$$cab \longrightarrow bc$$

Right Cayley graph: edges of the form $u \xrightarrow{a} ua$



Left Cayley graph: edges of the form $u \xrightarrow{a} au$



Shortlex order

Lexicographic order ($\leqslant_{lex}$): total order used in a dictionary.

Shortlex order ($\leqslant$): words are ordered by length and words of equal length are ordered by $\leqslant_{lex}$.

If $a < b$, then $ababb \leqslant_{lex} abba$ but $abba < ababb$.

For each rule $u \to v$, one has $v < u$.

$$
\begin{array}{ccc}
aa \longrightarrow a & ac \longrightarrow a & ba \longrightarrow a \\
bb \longrightarrow b & cb \longrightarrow bc & cc \longrightarrow c \\
abc \longrightarrow ab & bca \longrightarrow ca & cab \longrightarrow bc
\end{array}
$$

Properties of the shortlex order

Proposition

Let $u, v \in A^*$ and let $a, b \in A$.

(1) *If $u < v$, then $au < av$ and $ua < va$.*

(2) *If $ua \leqslant vb$, then $u \leqslant v$.*

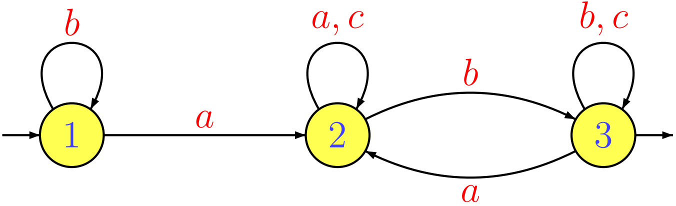
Therefore $\leqslant$ is a stable order on A^*: if $u \leqslant v$, then $xuy \leqslant xvy$ for all $x, y \in A^*$. Further, it is a well order.

Properties of the rewriting system

Let L be the set of left hand sides of the rules.

- All the rules are of the form $u \to v$ with $v < u$.
- The set L is unavoidable (any sufficently long word contains a factor in L or, equivalently, the set $A^* \setminus A^*LA^*$ is finite),
- No word of L has a proper factor in L.
- The set of proper factors of words in L is the set of reduced words.
- The rewriting system is confluent.
- It depends on the order on A.

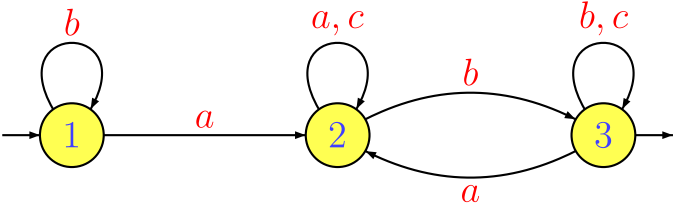
Computation of the presentation



1	1	2	3
a	2	2	2
b	1	3	3
c	-	2	3

Rules:

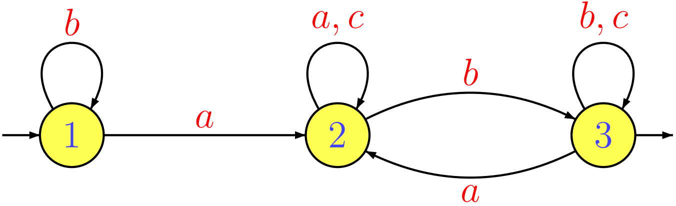
Computation of the presentation



1	1	2	3
a	2	2	2
b	1	3	3
c	-	2	3

Rules:
$aa \to a$

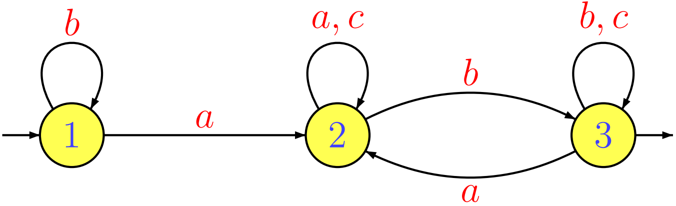
Computation of the presentation



1	1	2	3
a	2	2	2
b	1	3	3
c	-	2	3
ab	3	3	3

Rules:
$aa \to a$

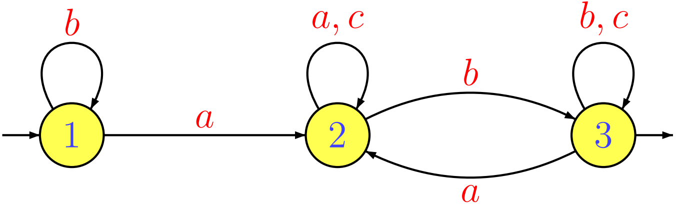
Computation of the presentation



1	1	2	3
a	2	2	2
b	1	3	3
c	-	2	3
ab	3	3	3

Rules:
$aa \longrightarrow a$
$ac \longrightarrow a$

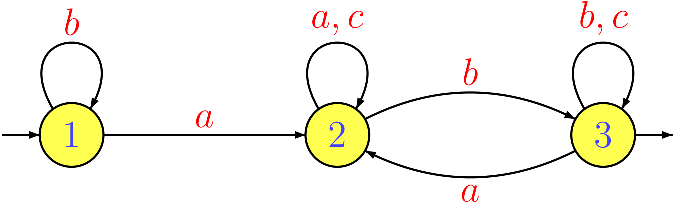
Computation of the presentation



1	1	2	3
a	2	2	2
b	1	3	3
c	-	2	3
ab	3	3	3

Rules:
$aa \longrightarrow a$
$ac \longrightarrow a$
$ba \longrightarrow a$

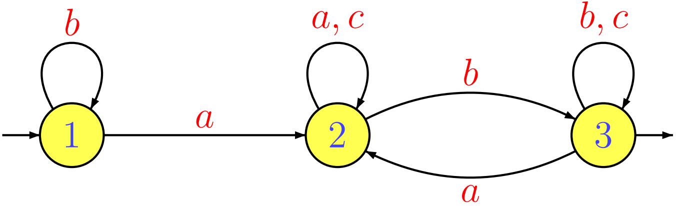
Computation of the presentation



1	1	2	3
a	2	2	2
b	1	3	3
c	-	2	3
ab	3	3	3

Rules:
$aa \longrightarrow a$
$ac \longrightarrow a$
$ba \longrightarrow a$
$bb \longrightarrow b$

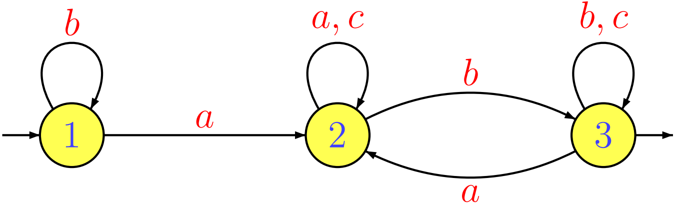
Computation of the presentation



1	1	2	3
a	2	2	2
b	1	3	3
c	-	2	3
ab	3	3	3
bc	-	3	3

Rules:
$aa \longrightarrow a$
$ac \longrightarrow a$
$ba \longrightarrow a$
$bb \longrightarrow b$

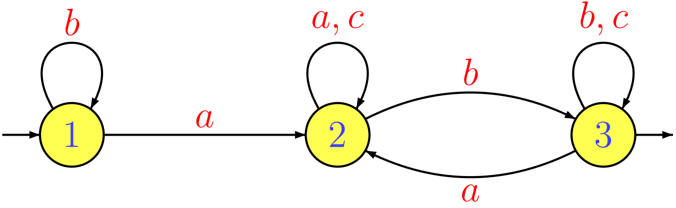
Computation of the presentation



1	1	2	3
a	2	2	2
b	1	3	3
c	-	2	3
ab	3	3	3
bc	-	3	3
ca	-	2	2

Rules:
$aa \longrightarrow a$
$ac \longrightarrow a$
$ba \longrightarrow a$
$bb \longrightarrow b$

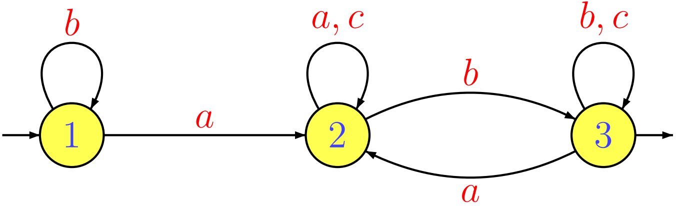
Computation of the presentation



1	1	2	3
a	2	2	2
b	1	3	3
c	-	2	3
ab	3	3	3
bc	-	3	3
ca	-	2	2

Rules:
$aa \longrightarrow a$
$ac \longrightarrow a$
$ba \longrightarrow a$
$bb \longrightarrow b$
$cb \longrightarrow bc$

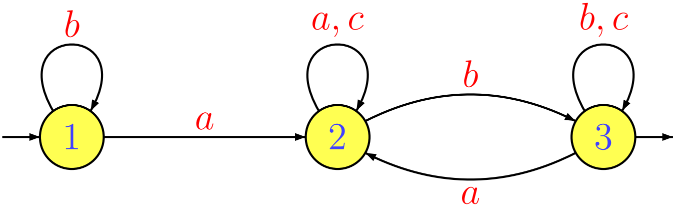
Computation of the presentation



1	1	2	3
a	2	2	2
b	1	3	3
c	-	2	3
ab	3	3	3
bc	-	3	3
ca	-	2	2

Rules:
$aa \longrightarrow a$
$ac \longrightarrow a$
$ba \longrightarrow a$
$bb \longrightarrow b$
$cb \longrightarrow bc$
$cc \longrightarrow c$

Computation of the presentation

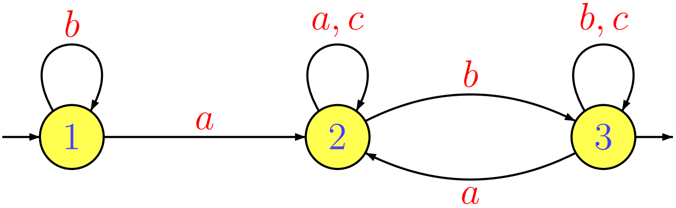


1	1	2	3
a	2	2	2
b	1	3	3
c	-	2	3
ab	3	3	3
bc	-	3	3
ca	-	2	2

Rules:
$aa \to a$
$ac \to a$
$ba \to a$
$bb \to b$
$cb \to bc$
$cc \to c$

$abc \to ab$

Computation of the presentation

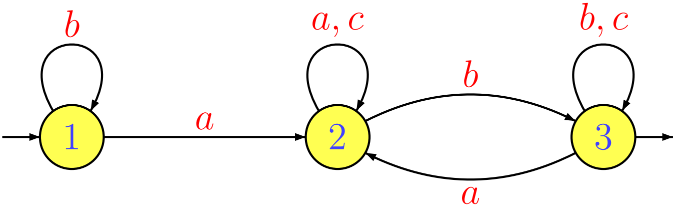


1	1	2	3
a	2	2	2
b	1	3	3
c	-	2	3
ab	3	3	3
bc	-	3	3
ca	-	2	2

Rules:
$aa \longrightarrow a$
$ac \longrightarrow a$
$ba \longrightarrow a$
$bb \longrightarrow b$
$cb \longrightarrow bc$
$cc \longrightarrow c$

$abc \longrightarrow ab$
$bca \longrightarrow ca$

Computation of the presentation

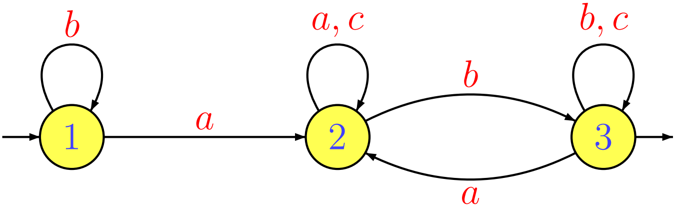


1	1	2	3
a	2	2	2
b	1	3	3
c	-	2	3
ab	3	3	3
bc	-	3	3
ca	-	2	2

Rules:
$aa \rightarrow a$
$ac \rightarrow a$
$ba \rightarrow a$
$bb \rightarrow b$
$cb \rightarrow bc$
$cc \rightarrow c$

$abc \rightarrow ab$
$bca \rightarrow ca$
$cab \rightarrow bc$

Computation of the presentation



1	1	2	3
a	2	2	2
b	1	3	3
c	-	2	3
ab	3	3	3
bc	-	3	3
ca	-	2	2

Rules:
$aa \to a$
$ac \to a$
$ba \to a$
$bb \to b$
$cb \to bc$
$cc \to c$

$abc \to ab$
$bca \to ca$
$cab \to bc$
The end!

Main algorithm

Convention: a, b, c, ... will be generic letters and $p, q, r, \ldots$ will be generic words.

We maintain two tables. One contains the list of reduced words with the corresponding elements of U. The other one contains the list of relations.

1	1	2	3
a	2	2	2
b	1	3	3
c	-	2	3
ab	3	3	3
bc	-	3	3
ca	-	2	2

$$aa \to a$$
$$ac \to a$$
$$ba \to a$$
$$bb \to b$$
$$cb \to bc$$
$$cc \to c$$

$$abc \to ab$$
$$bca \to ca$$
$$cab \to bc$$

Main loops

For each length n,

Computation of the right Cayley graph

> **For** each word u of length n in the table
>   **For** a ranging from the first to the last letter
>   handle ua [next slide]

Computation of the left Cayley graph

> **For** each word u of length n in the table
>   **For** a ranging from the first to the last letter
>   reduce au

Handling ua

For each length n,

For each word u of length n in the table

For a ranging from the first to the last letter

try to reduce the word ua; [next slide]
if it can be reduced with the current rules
switch to the next letter
else
compute the associated element of U;
if it corresponds to some word v
add the relation $ua \to v$
else
add this new element to the table;

Reduction of ua

Put $u = bs$. If $sa \to r$, then $ua = bsa \to br$.

- If $r = 1$, then $ua \to b$.

- If $r \neq 1$, put $r = tc$. Then $r = tc < sa$ implies $t \leqslant s$.

- If $t = s$, then $c < a$, $ua \to br = btc = bsc = uc$ and the reduction of uc has been done, since $c < a$.

- If $t < s$, then $|t| \leqslant |s| < |u|$ and thus the reduction of bt has been done. Assume that $bt \to v$. Then $v \leqslant bt < bs = u$. Then $ua \to br = btc \to vc$, and the reduction of vc has been done, since $v < u$.

Left Cayley graph

If $u = pb$ and $a \in A$, then $au = tb$, where $t = ap$.

Since $|t| \leqslant |u|$, tb has been handled at this stage.

Complexity

Theorem

The number of accesses to the function computing the product in the universe is equal to $|S| + |R| - |A|$ *(where* R *is the set of rules).*

Result for $\mathcal{T}_n$ ($|A| = 3$).

n	$\|S\|$	Nb of Rules	Nb of Calls
3	27	13	37
4	256	83	336
5	3,125	751	3,873
6	46,656	7935	54,588

Benchmarks (in seconds)

Name	$\lvert A\rvert$	$\lvert S\rvert$	S	$\mathcal{D}$	$\mathcal{H}$
S10	2	3,628,800	11.02	15.00	0.01
T7	3	823,543	2.95	2.38	0.74
F7	4	2,097,152	8.87	7.36	0.64
I8	3	1,441,729	5.44	5.63	0.42
RB4	4	63,904	0.37	0.10	0.02
FC13	13	5,200,300	35.63	22.61	1.29
FIC12	12	2,704,156	20.57	11.53	0.71
POPI12	2	16,224,937	56.00	66.73	5.90
Tr6	21	2,097,152	33.09	10.43	1.12
U7	21	2,097,152	40.93	9.28	1.04

Part IV

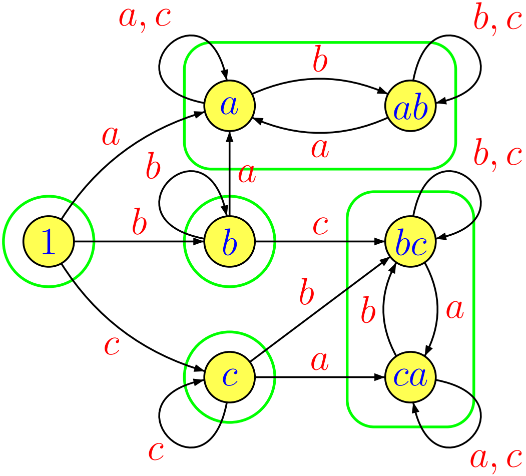
Green's relations and blocks

^{*1}

$^*b$		$^*c$

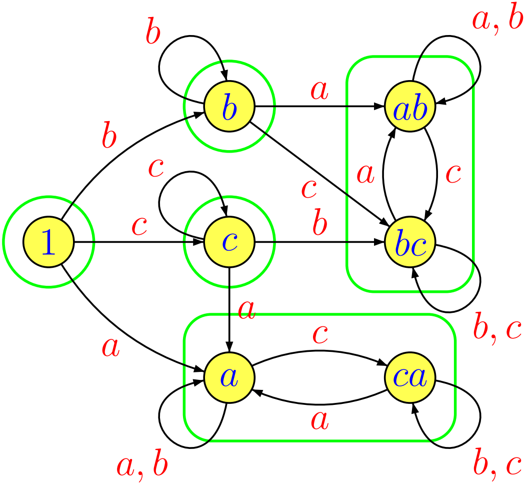
$^*a$	$^*ab$
$^*ca$	$^*bc$

$\mathcal{R}$-classes



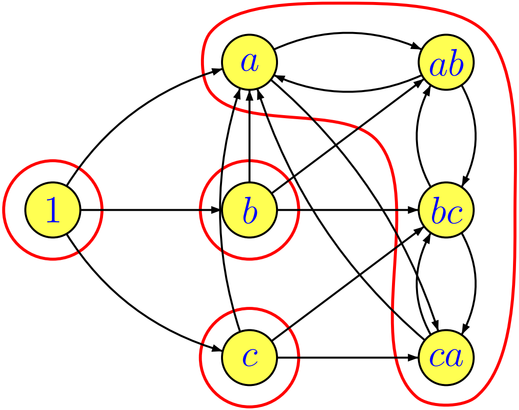
The $\mathcal{R}$-classes are the strongly connected components of the right Cayley graph.

$\mathcal{L}$-classes



The $\mathcal{L}$-classes are the strongly connected components of the left Cayley graph.

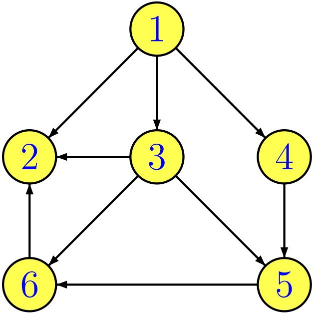
$\mathcal{J}$-classes (loops and labels are omitted)



The $\mathcal{J}$-classes are the strongly connected components of the union of the right and left Cayley graphs.

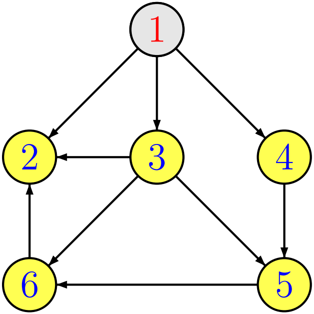
Depth first search (DFS)

A directed graph is given by its set of vertices and for each vertex, the ordered set of its successors.



Depth first search (DFS)

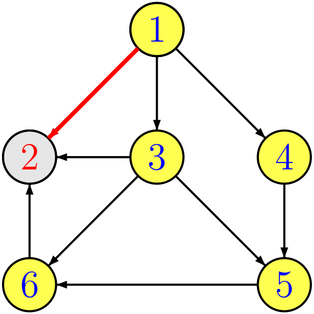
A directed graph is given by its set of vertices and for each vertex, the ordered set of its successors.



Starting from vertex 1

Depth first search (DFS)

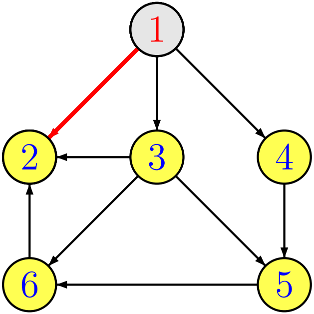
A directed graph is given by its set of vertices and for each vertex, the ordered set of its successors.



Reaching the rightmost neighbour, 2

Depth first search (DFS)

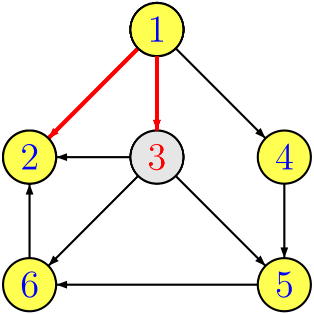
A directed graph is given by its set of vertices and for each vertex, the ordered set of its successors.



Vertex 2 has no neighbour, back to 1

Depth first search (DFS)

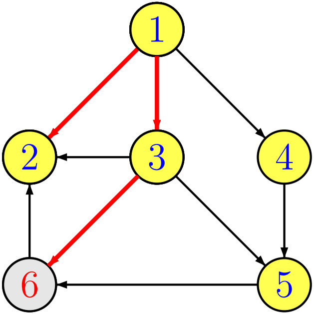
A directed graph is given by its set of vertices and for each vertex, the ordered set of its successors.



Reaching the rightmost neighbour, 3

Depth first search (DFS)

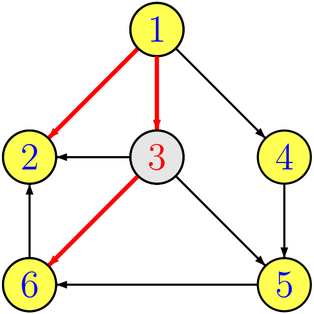
A directed graph is given by its set of vertices and for each vertex, the ordered set of its successors.



Reaching the rightmost neighbour, 6

Depth first search (DFS)

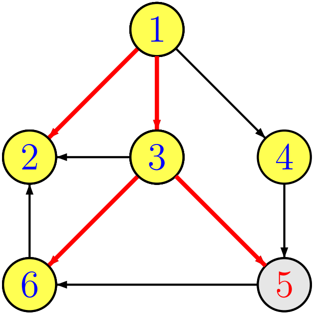
A directed graph is given by its set of vertices and for each vertex, the ordered set of its successors.



Vertex 6 has no neighbour, back to 3

Depth first search (DFS)

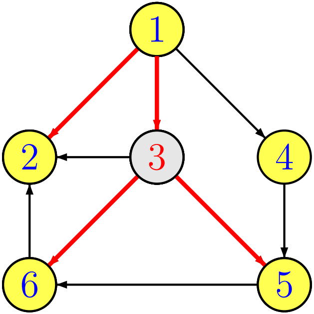
A directed graph is given by its set of vertices and for each vertex, the ordered set of its successors.



Reaching the rightmost neighbour, 5

Depth first search (DFS)

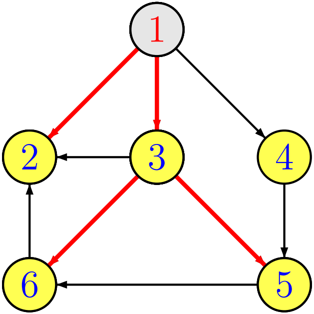
A directed graph is given by its set of vertices and for each vertex, the ordered set of its successors.



Vertex 5 has no free neighbour, back to 3

Depth first search (DFS)

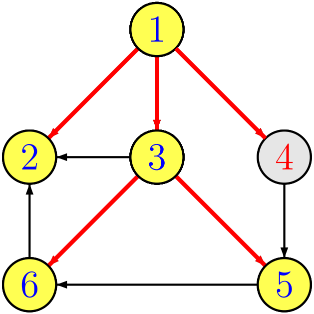
A directed graph is given by its set of vertices and for each vertex, the ordered set of its successors.



Vertex 3 has no free neighbour, back to 1

Depth first search (DFS)

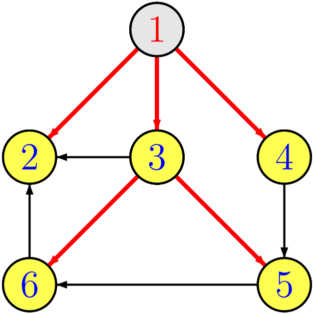
A directed graph is given by its set of vertices and for each vertex, the ordered set of its successors.



Reaching the rightmost neighbour, 4

Depth first search (DFS)

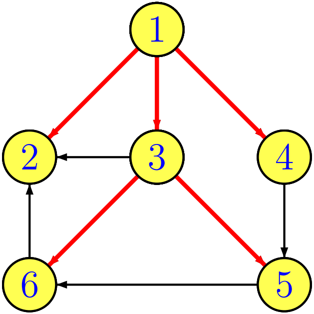
A directed graph is given by its set of vertices and for each vertex, the ordered set of its successors.



Vertex 4 has no free neighbour, back to 1

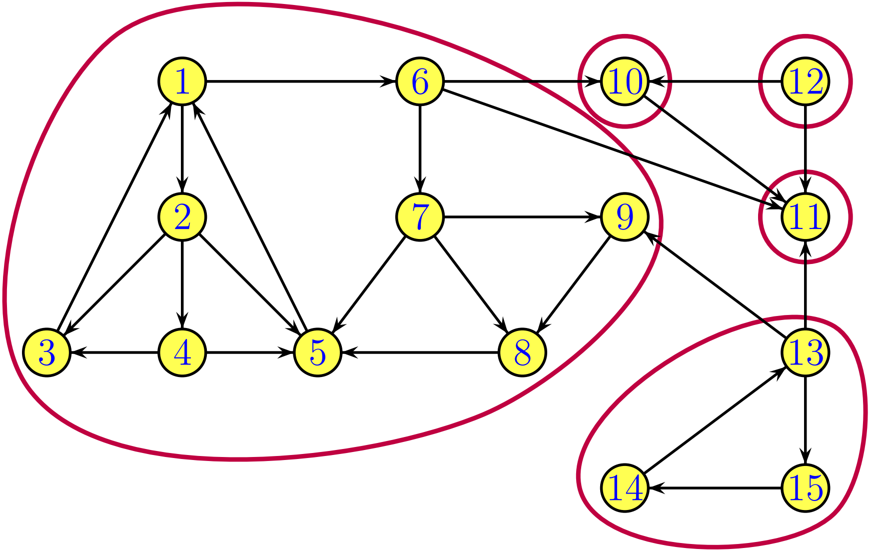
Depth first search (DFS)

A directed graph is given by its set of vertices and for each vertex, the ordered set of its successors.

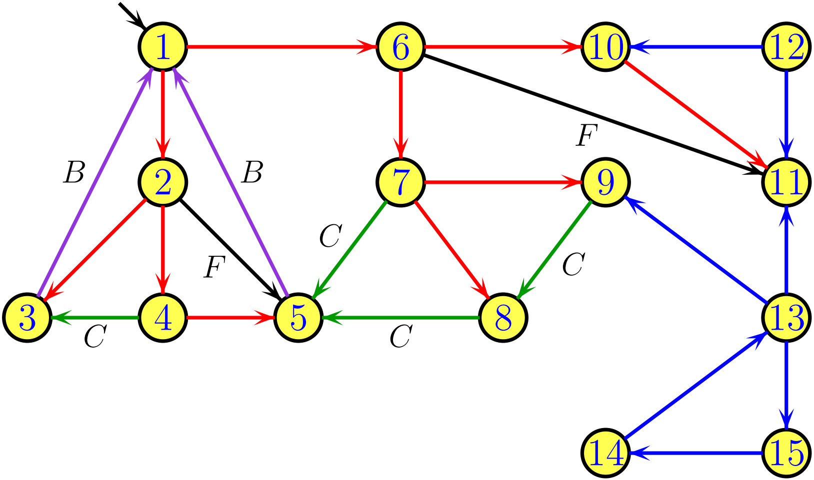


The end.

Tarjan's algorithm (1)



Tree, Backward, Cross and Forward edges



Ties (in the tree)

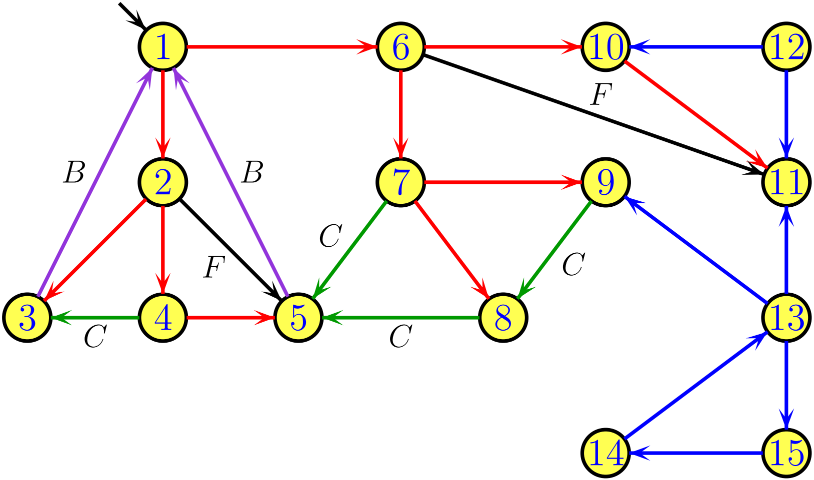
The tie $t(x)$ of a vertex x is the least y such that there is a path (possibly empty) from x to y containing at most one backward or cross edge.

Theorem

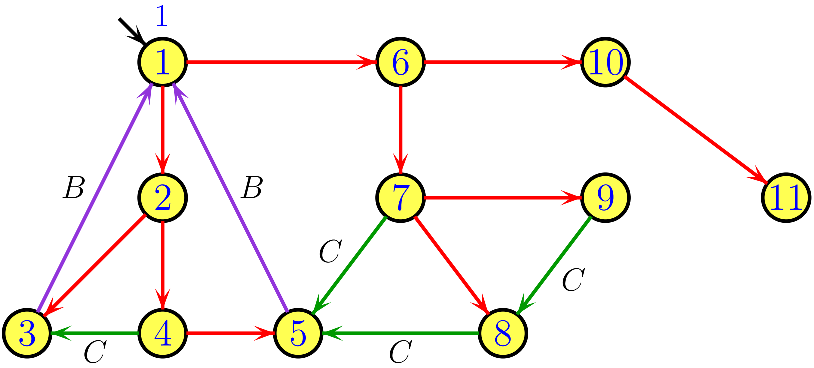
If $t(x) = x$ and if $t(y) < y$ for every descendant y of x, then the set of descendants of x is a strongly connected component (SCC).

Algorithm : Find the deepest SCC, remove it and look for the next one.

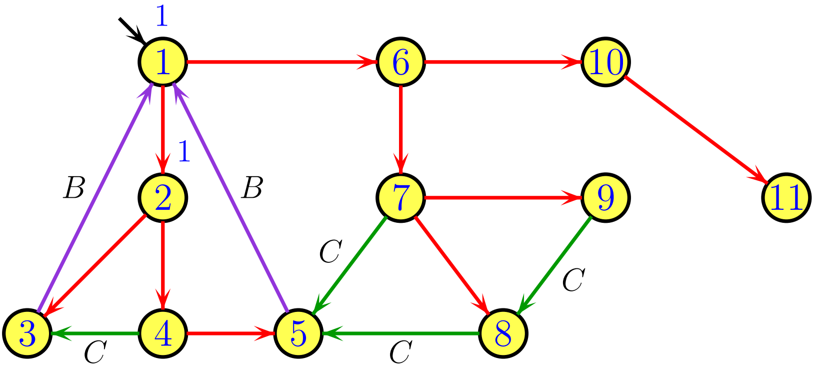
Computing the ties



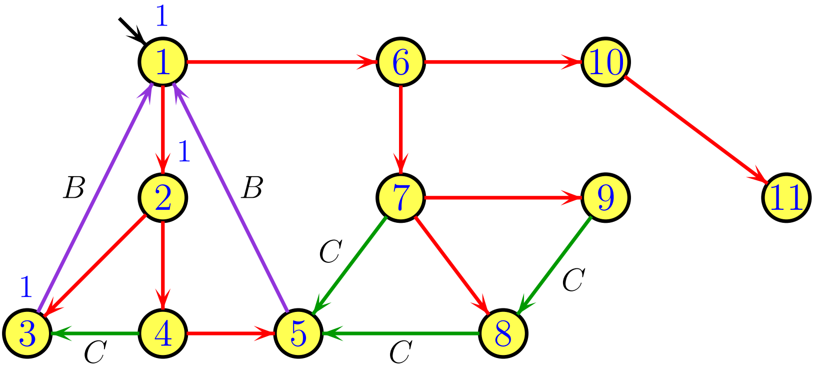
Computing the ties



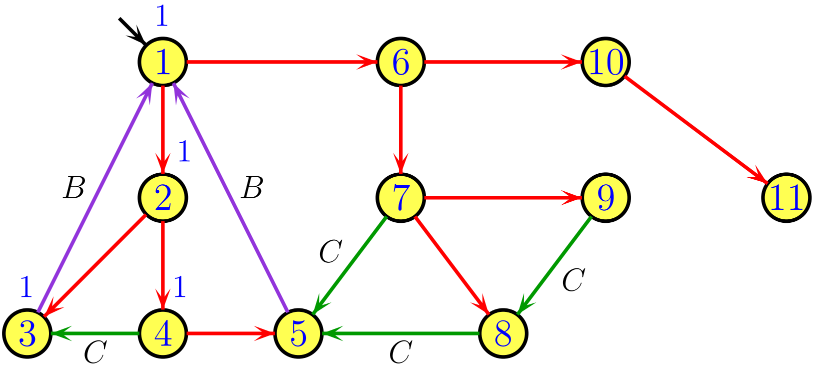
Computing the ties



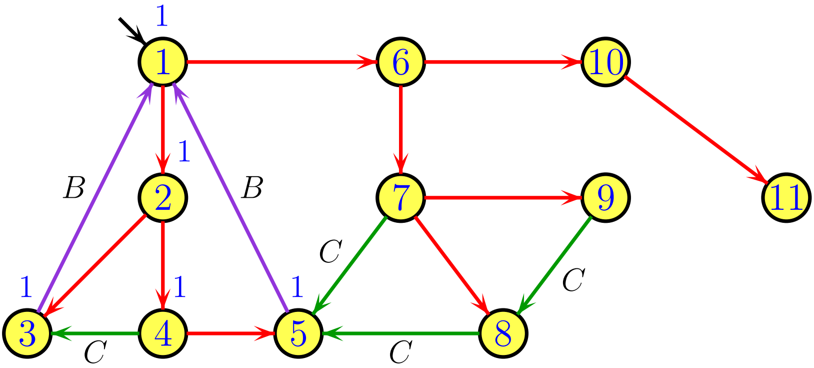
Computing the ties



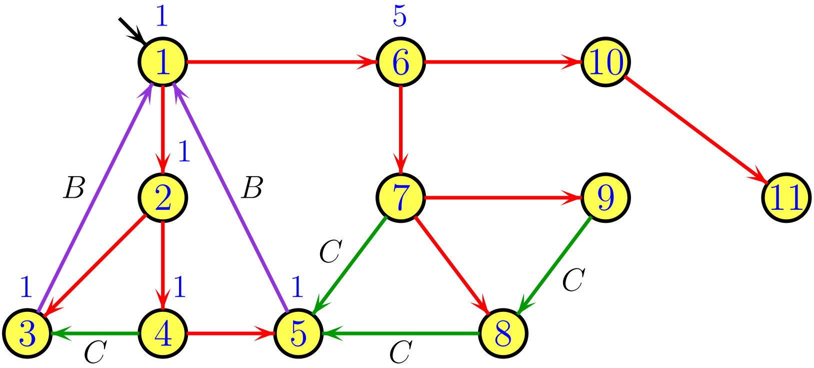
Computing the ties



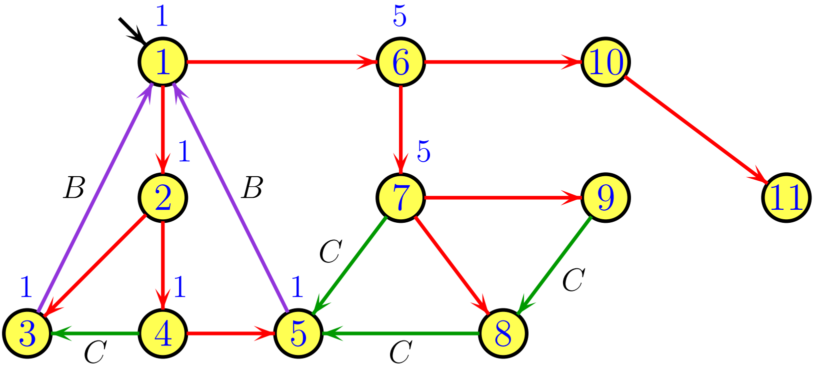
Computing the ties



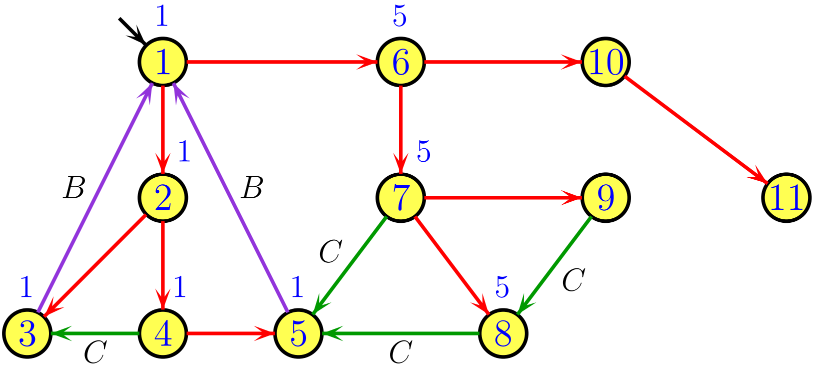
Computing the ties



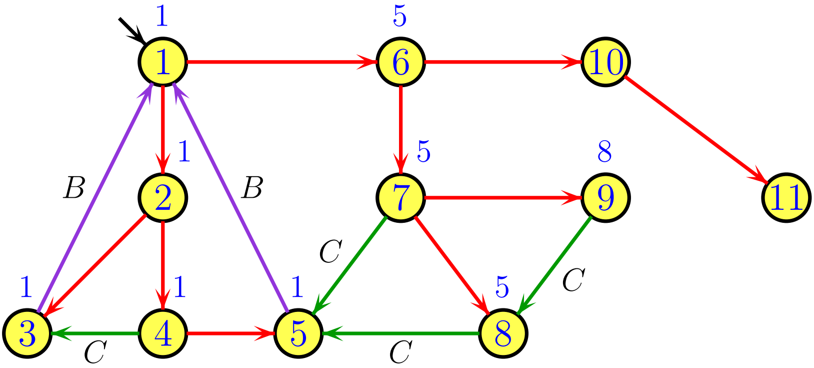
Computing the ties



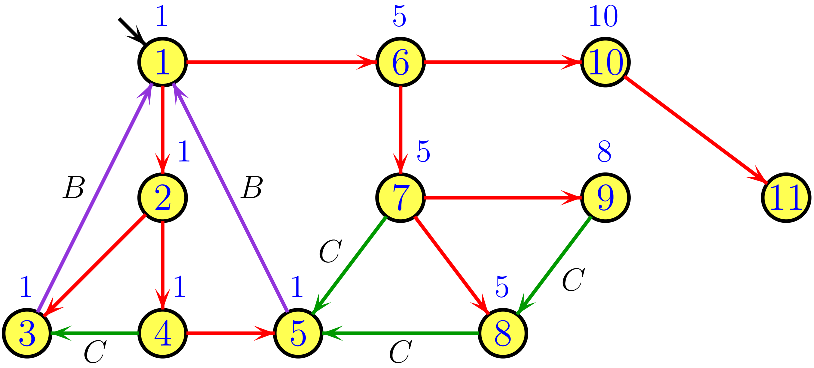
Computing the ties



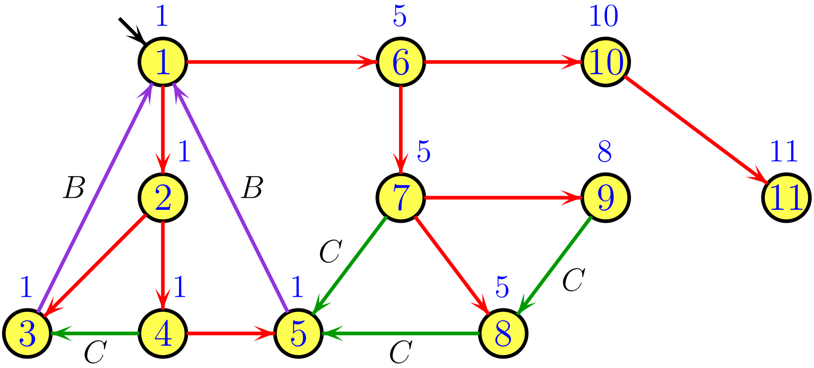
Computing the ties



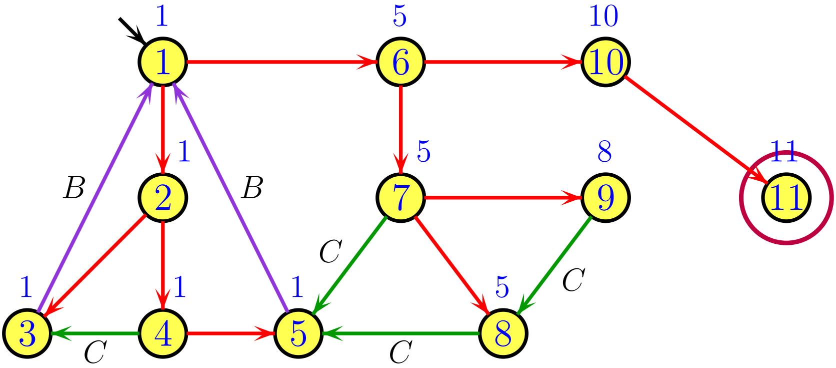
Computing the ties



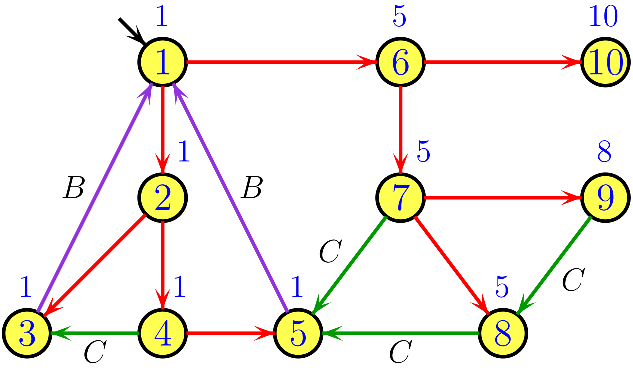
Computing the ties



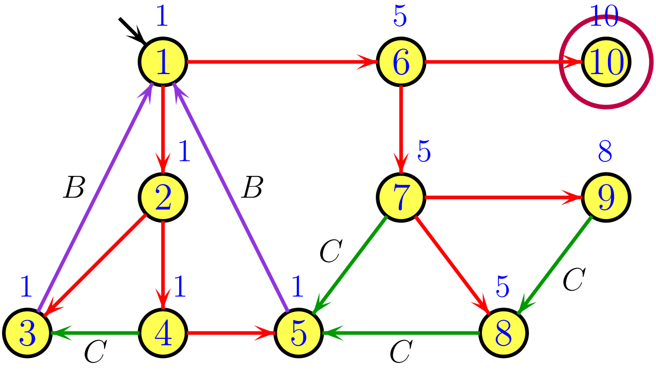
Computing the ties



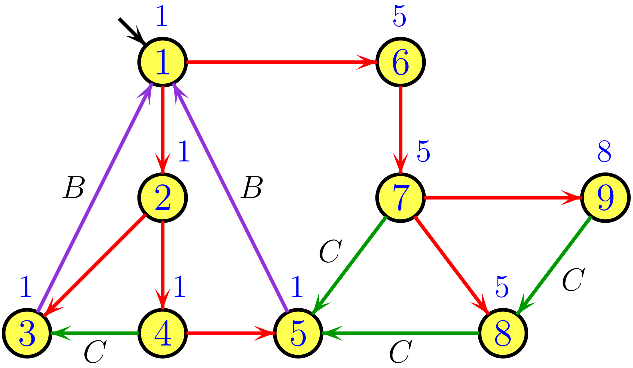
Computing the ties



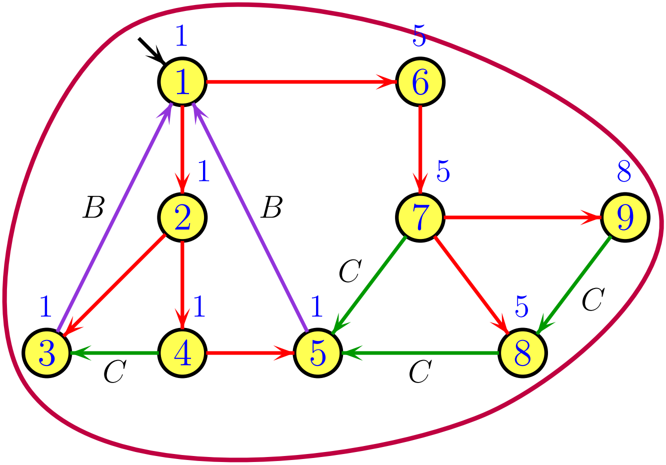
Computing the ties



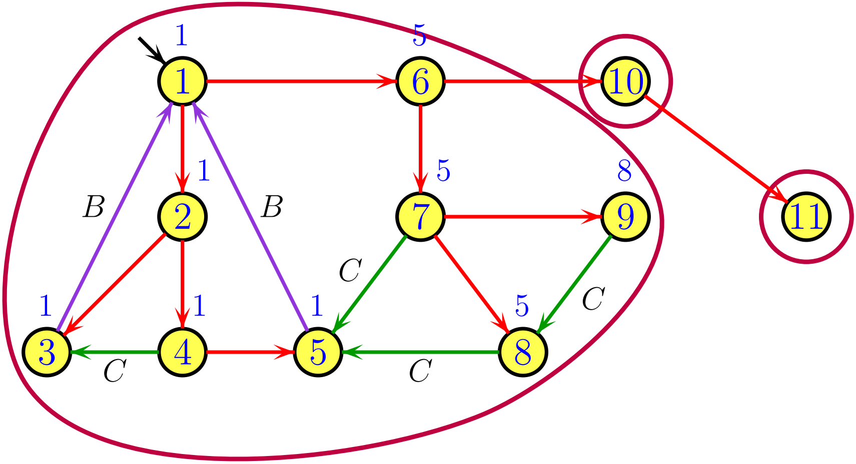
Computing the ties



Computing the ties



Computing the ties



Tarjan computes the SCC on $O(|E| + |V|)$ time.

Computing Green's relations

To obtain the $\mathcal{R}$-classes, the $\mathcal{L}$-classes and the $\mathcal{D}$-classes, it suffices to compute the strongly connected of the right Cayley graph, the left Cayley graph and the union G of the two Cayley graphs.

It can be done in time linear in the size (number of vertices + number of edges) of the Cayley graphs, that is, in $O(|A||S|)$. Space complexity is also linear.

The depth-first search of G also gives the "deepest" strongly connected component, that is, the minimal ideal.

$\mathcal{H}$-classes

At this stage, one gets the table of all $\mathcal{D}$-classes, $\mathcal{R}$-classes and $\mathcal{L}$-classes:

Element	1	2	3	4	5	6	7	8	9	10	11	12	13
$\mathcal{D}$-class	3	3	2	2	2	1	2	2	1	2	2	1	2
$\mathcal{R}$-class	4	4	3	2	3	1	2	3	1	2	3	1	2
$\mathcal{L}$-class	6	6	5	5	3	4	3	5	2	5	3	1	3

How to find the $\mathcal{H}$-classes?

Step one: sorting the elements by $\mathcal{R}$-class number

Element	1	2	3	4	5	6	7	8	9	10	11	12	13
$\mathcal{R}$-class	4	4	3	2	3	1	2	3	1	2	3	1	2

Element	1	2	3	4	5	6	7	8	9	10	11	12	13
Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

This can be done in linear time by first counting the number of elements in each $\mathcal{R}$-class.

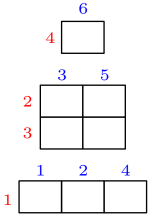
$\mathcal{R}$-class	1	2	3	4
Number	3	4	4	2

Step two: browsing the $\mathcal{L}$-class table

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{L}$-class	4	2	1	5	3	5	3	5	3	5	3	6	6
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

$\mathcal{L}$-class	1	2	3	4	5	6
$\mathcal{H}$-class	0	0	0	0	0	0

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{H}$-class													



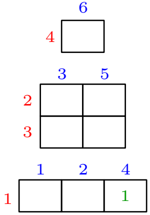
Step two: browsing the $\mathcal{L}$-class table

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{L}$-class	4	2	1	5	3	5	3	5	3	5	3	6	6
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

$\mathcal{L}$-class	1	2	3	4	5	6
$\mathcal{H}$-class	0	0	0	1	0	0

New $\mathcal{R}$-class

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{H}$-class	1												

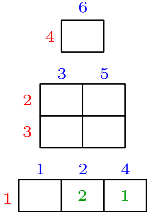


Step two: browsing the $\mathcal{L}$-class table

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{L}$-class	4	2	1	5	3	5	3	5	3	5	3	6	6
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

$\mathcal{L}$-class	1	2	3	4	5	6
$\mathcal{H}$-class	0	2	0	1	0	0

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{H}$-class	1	2											

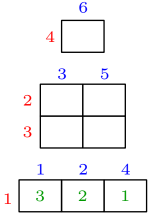


Step two: browsing the $\mathcal{L}$-class table

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{L}$-class	4	2	1	5	3	5	3	5	3	5	3	6	6
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

$\mathcal{L}$-class	1	2	3	4	5	6
$\mathcal{H}$-class	3	2	0	1	0	0

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{H}$-class	1	2	3										



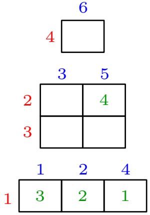
Step two: browsing the $\mathcal{L}$-class table

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{L}$-class	4	2	1	5	3	5	3	5	3	5	3	6	6
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

$\mathcal{L}$-class	1	2	3	4	5	6
$\mathcal{H}$-class	3	2	0	1	4	0

New $\mathcal{R}$-class

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{H}$-class	1	2	3	4									

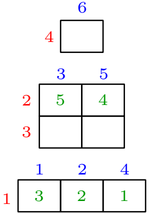


Step two: browsing the $\mathcal{L}$-class table

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{L}$-class	4	2	1	5	3	5	3	5	3	5	3	6	6
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

$\mathcal{L}$-class	1	2	3	4	5	6
$\mathcal{H}$-class	3	2	5	1	4	0

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{H}$-class	1	2	3	4	5								

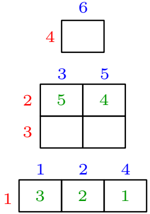


Step two: browsing the $\mathcal{L}$-class table

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{L}$-class	4	2	1	5	3	5	3	5	3	5	3	6	6
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

$\mathcal{L}$-class	1	2	3	4	5	6
$\mathcal{H}$-class	3	2	5	1	4	0

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{H}$-class	1	2	3	4	5	4							

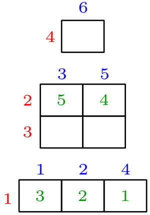


Step two: browsing the $\mathcal{L}$-class table

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{L}$-class	4	2	1	5	3	5	3	5	3	5	3	6	6
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

$\mathcal{L}$-class	1	2	3	4	5	6
$\mathcal{H}$-class	3	2	5	1	4	0

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{H}$-class	1	2	3	4	5	4	5						



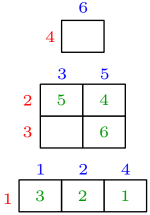
Step two: browsing the $\mathcal{L}$-class table

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{L}$-class	4	2	1	5	3	5	3	5	3	5	3	6	6
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

$\mathcal{L}$-class	1	2	3	4	5	6
$\mathcal{H}$-class	3	2	5	1	6	0

New $\mathcal{R}$-class

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{H}$-class	1	2	3	4	5	4	5	6					

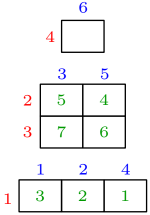


Step two: browsing the $\mathcal{L}$-class table

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{L}$-class	4	2	1	5	3	5	3	5	3	5	3	6	6
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

$\mathcal{L}$-class	1	2	3	4	5	6
$\mathcal{H}$-class	3	2	7	1	6	0

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{H}$-class	1	2	3	4	5	4	5	6	7				

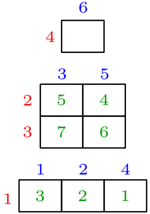


Step two: browsing the $\mathcal{L}$-class table

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{L}$-class	4	2	1	5	3	5	3	5	3	5	3	6	6
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

$\mathcal{L}$-class	1	2	3	4	5	6
$\mathcal{H}$-class	3	2	7	1	6	0

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{H}$-class	1	2	3	4	5	4	5	6	7	6			

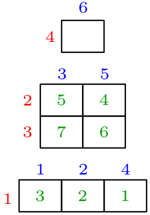


Step two: browsing the $\mathcal{L}$-class table

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{L}$-class	4	2	1	5	3	5	3	5	3	5	3	6	6
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

$\mathcal{L}$-class	1	2	3	4	5	6
$\mathcal{H}$-class	3	2	7	1	6	0

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{H}$-class	1	2	3	4	5	4	5	6	7	6	7		



Step two: browsing the $\mathcal{L}$-class table

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{L}$-class	4	2	1	5	3	5	3	5	3	5	3	6	6
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

$\mathcal{L}$-class	1	2	3	4	5	6
$\mathcal{H}$-class	3	2	7	1	6	8

New $\mathcal{R}$-class

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{H}$-class	1	2	3	4	5	4	5	6	7	6	7	8	

	6
4	8

	3	5
2	5	4
3	7	6

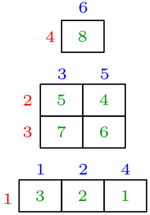
	1	2	4
1	3	2	1

Step two: browsing the $\mathcal{L}$-class table

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{L}$-class	4	2	1	5	3	5	3	5	3	5	3	6	6
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

$\mathcal{L}$-class	1	2	3	4	5	6
$\mathcal{H}$-class	3	2	7	1	6	8

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{H}$-class	1	2	3	4	5	4	5	6	7	6	7	8	8



Step two: browsing the $\mathcal{L}$-class table

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{L}$-class	4	2	1	5	3	5	3	5	3	5	3	6	6
$\mathcal{R}$-class	1	1	1	2	2	2	2	3	3	3	3	4	4

$\mathcal{L}$-class	1	2	3	4	5	6
$\mathcal{H}$-class	3	2	7	1	6	8

Sorted	6	9	12	4	7	10	13	3	5	8	11	1	2
$\mathcal{H}$-class	1	2	3	4	5	4	5	6	7	6	7	8	8

Element	1	2	3	4	5	6	7	8	9	10	11	12	13
$\mathcal{H}$-class	8	8	6	4	7	1	5	6	2	4	7	3	5

Part V

Idempotents and (weak) inverses

Computing the idempotents can be done by testing whether $x = x^2$ in the universe, or by using the rewriting system.

Possible improvements

• If the $\mathcal{H}$-classes of the minimal ideal are trivial (which is easy to test), then all elements of the minimal ideal are idempotent.

• If there is only one $\mathcal{H}$-class, the semigroup is a group, and 1 is the unique idempotent.

• If one makes use of the rewriting system, one reads the word x from the node x on the right Cayley graph, but one can stop if one leaves the $\mathcal{R}$-class of x.

Inverses and weak inverses

Recall that t is a weak inverse of s if $tst = t$. If, further, $sts = s$, then t is an inverse of s.

Algorithm: for each $t \in S$, start a depth first search of G from t. Note that each visited s is $\mathcal{J}$-below t. One checks whether:

(1) st is idempotent,

(2) $st \mathrel{\mathcal{R}} s$

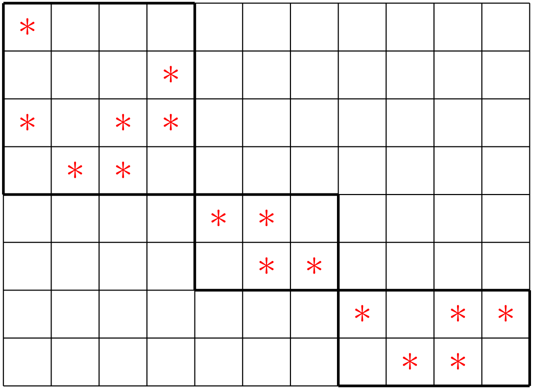
(3) $s \mathrel{\mathcal{J}} t$

Then s is a weak inverse [inverse] of t iff (1–2) [(1–3)] are satisfied.

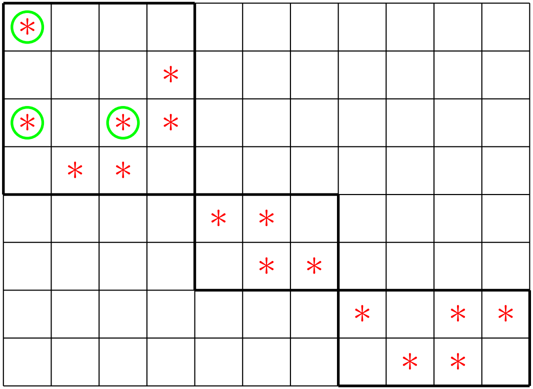
Part VI

Blocks

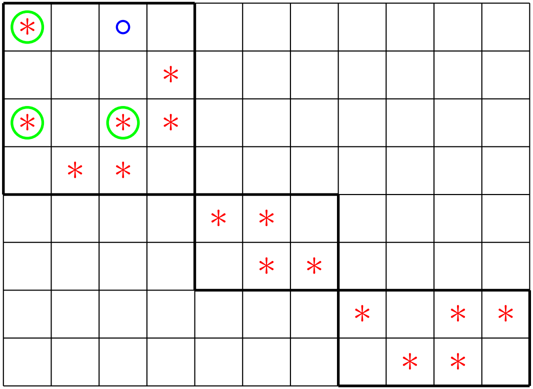
Blocks



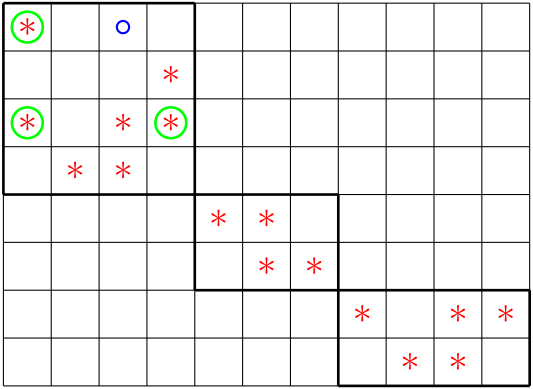
Blocks



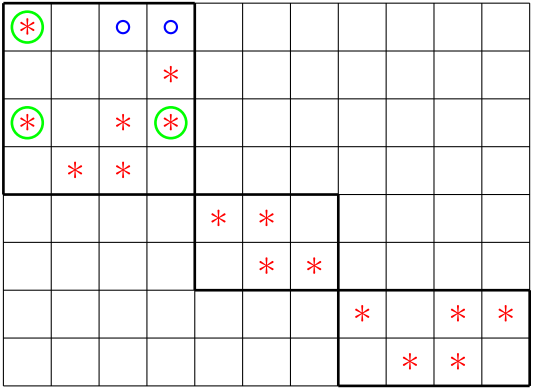
Blocks



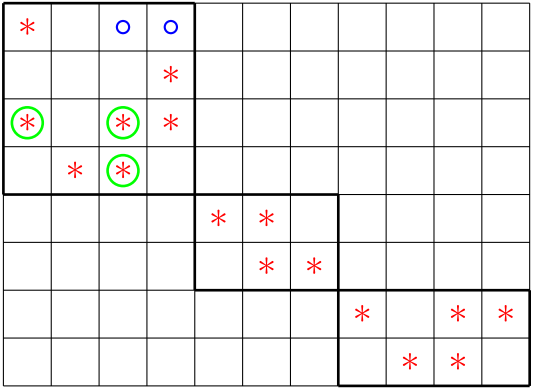
Blocks



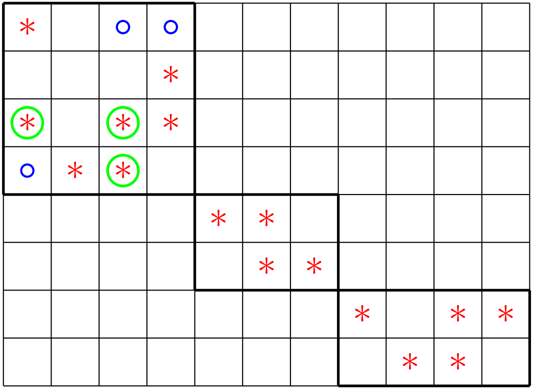
Blocks



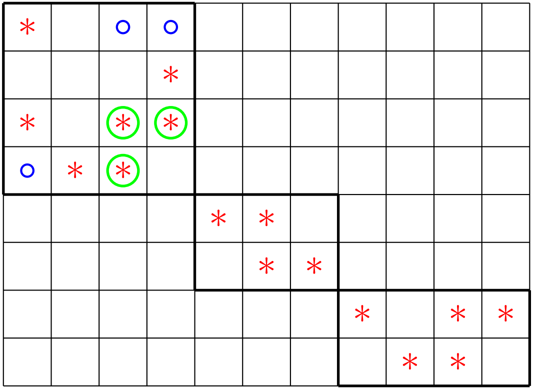
Blocks



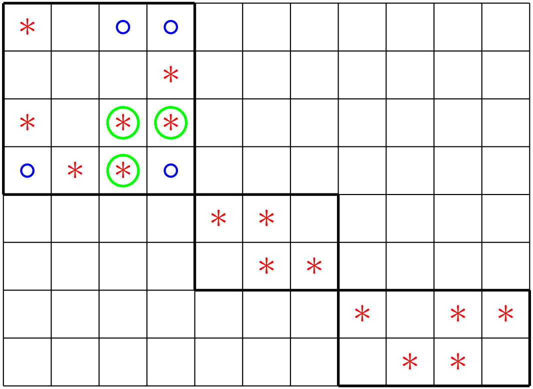
Blocks



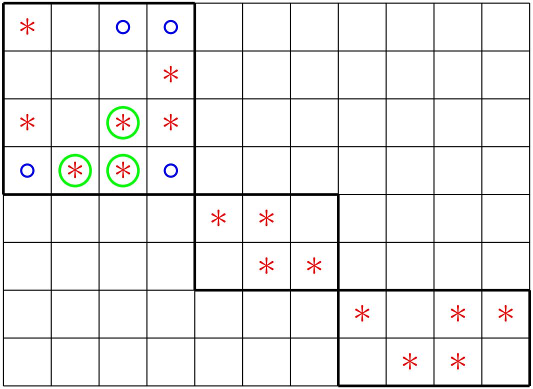
Blocks



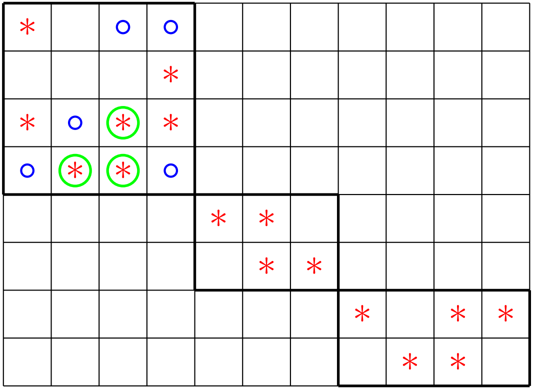
Blocks



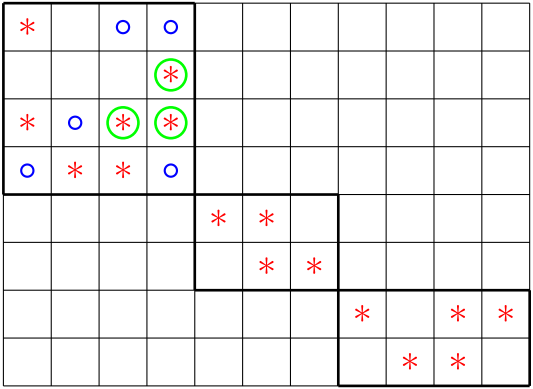
Blocks



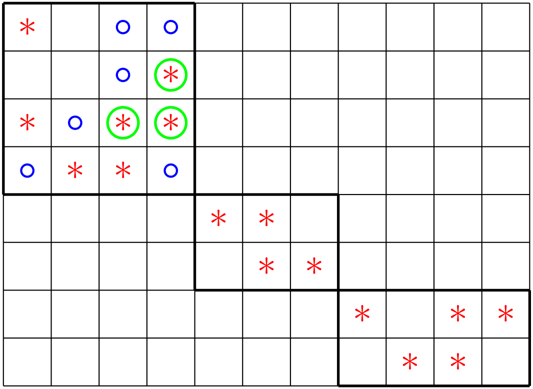
Blocks



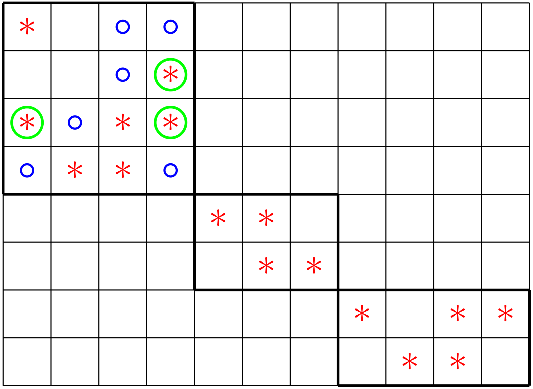
Blocks



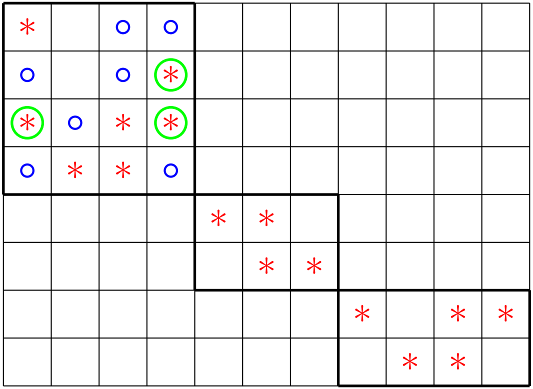
Blocks



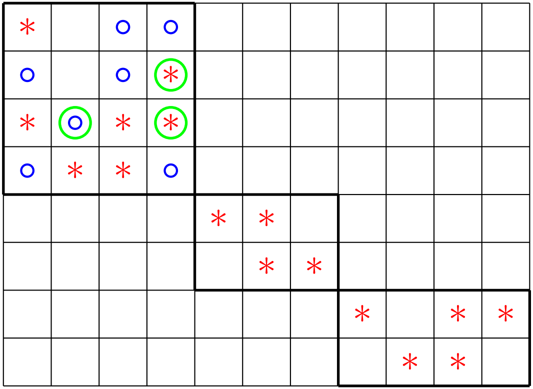
Blocks



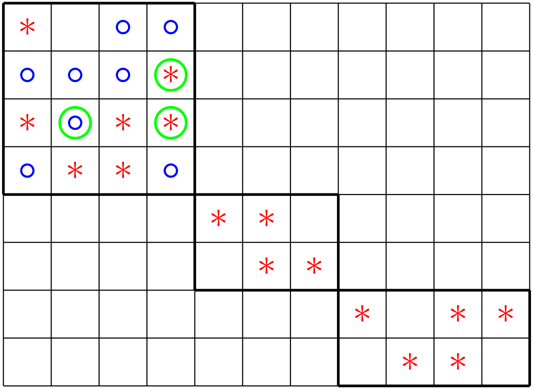
Blocks



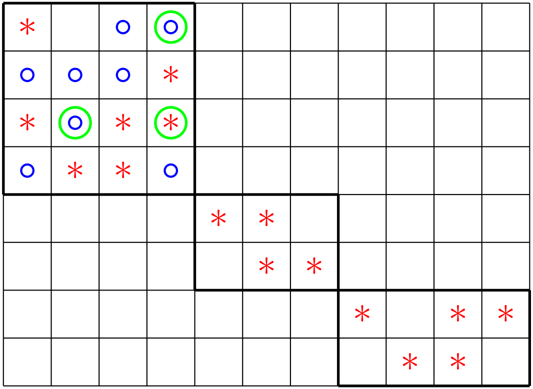
Blocks



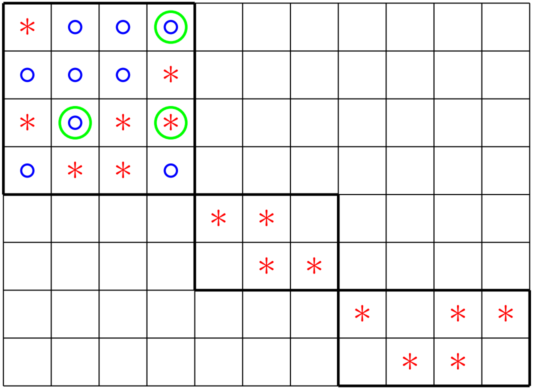
Blocks



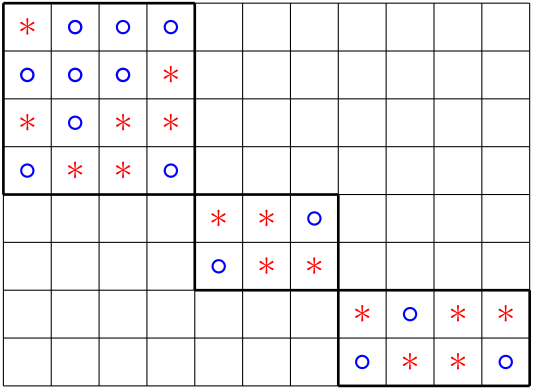
Blocks



Blocks



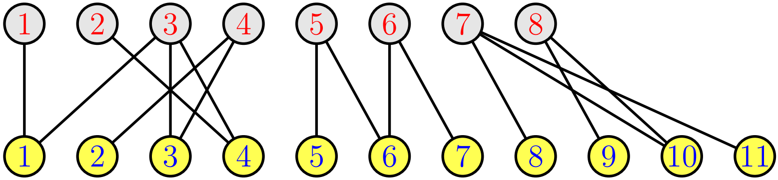
Blocks



This computation amounts to finding the connected components of a certain graph.

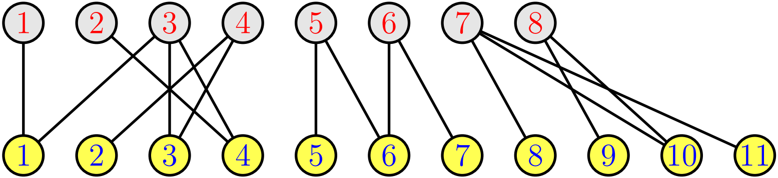
Computation of the blocks

	1	2	3	4	5	6	7	8	9	10	11
1	*										
2				*							
3	*		*	*							
4		*	*								
5					*	*					
6						*	*				
7								*		*	*
8									*	*	



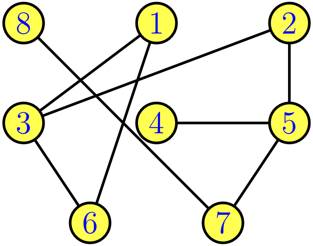
Computation of the blocks (2)

One could use again Tarjan's algorithm, but using the "Union-Find" algorithm is a bit simpler.



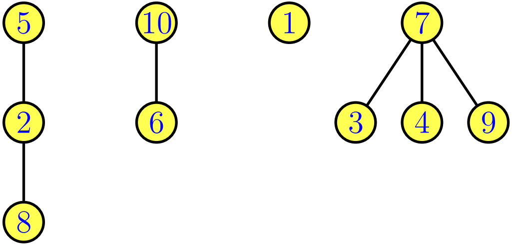
Union-Find

The vertices (1, 3), (6, 3), (1, 6), (2, 3), (5, 4), (7, 8), (7,5) and (2, 5) are connected. Are 4 and 6 connected?



Representing a forest by an array

In computer science, trees are represented top-down... The root of each tree is its own parent.



Vertex	1	2	3	4	5	6	7	8	9	10
Parent	1	5	7	7	5	10	7	2	7	10

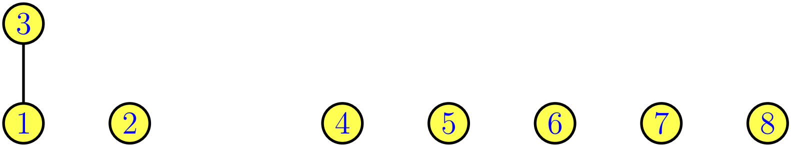
Union-find (1)

Rule: to add (x, y), find the root x' $[y']$ of the tree containing x $[y]$. If $x' \neq y'$, add the edge (x', y').

Adding (1, 3)

Union-find (1)

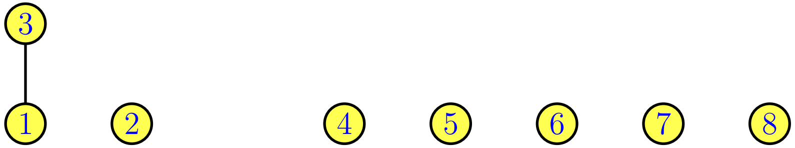
Rule: to add (x, y), find the root x' $[y']$ of the tree containing x $[y]$. If $x' \neq y'$, add the edge (x', y').



Adding (1, 3)

Union-find (1)

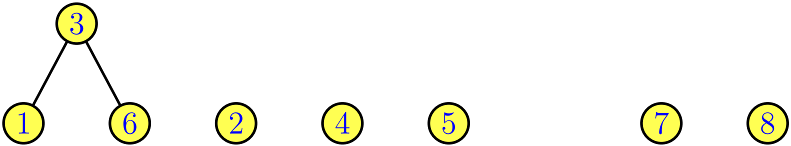
Rule: to add (x, y), find the root x' $[y']$ of the tree containing x $[y]$. If $x' \neq y'$, add the edge (x', y').



Adding (6, 3)

Union-find (1)

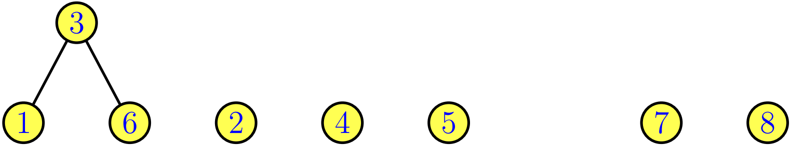
Rule: to add (x, y), find the root x' $[y']$ of the tree containing x $[y]$. If $x' \neq y'$, add the edge (x', y').



Adding (6, 3)

Union-find (1)

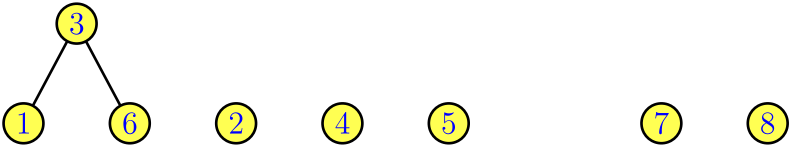
Rule: to add (x, y), find the root x' $[y']$ of the tree containing x $[y]$. If $x' \neq y'$, add the edge (x', y').



Adding (1, 6)

Union-find (1)

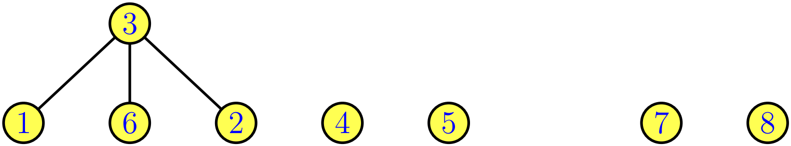
Rule: to add (x, y), find the root x' $[y']$ of the tree containing x $[y]$. If $x' \neq y'$, add the edge (x', y').



Adding (2, 3)

Union-find (1)

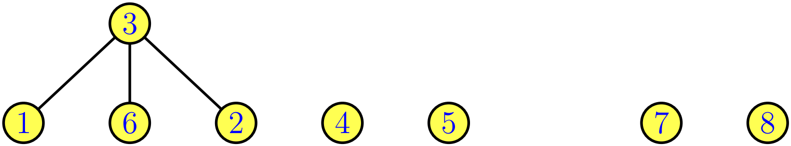
Rule: to add (x, y), find the root x' $[y']$ of the tree containing x $[y]$. If $x' \neq y'$, add the edge (x', y').



Adding (2, 3)

Union-find (1)

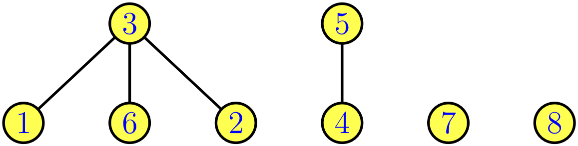
Rule: to add (x, y), find the root x' $[y']$ of the tree containing x $[y]$. If $x' \neq y'$, add the edge (x', y').



Adding (5, 4)

Union-find (1)

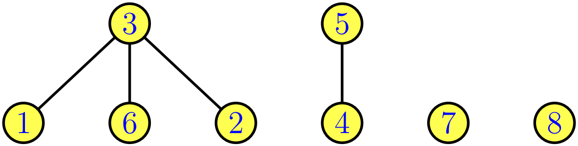
Rule: to add (x, y), find the root x' $[y']$ of the tree containing x $[y]$. If $x' \neq y'$, add the edge (x', y').



Adding (5, 4)

Union-find (1)

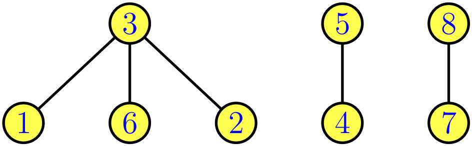
Rule: to add (x, y), find the root x' $[y']$ of the tree containing x $[y]$. If $x' \neq y'$, add the edge (x', y').



Adding (7, 8)

Union-find (1)

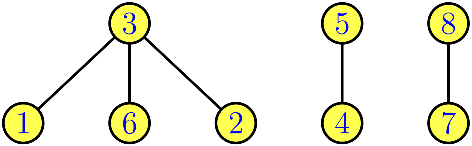
Rule: to add (x, y), find the root x' $[y']$ of the tree containing x $[y]$. If $x' \neq y'$, add the edge (x', y').



Adding (7, 8)

Union-find (1)

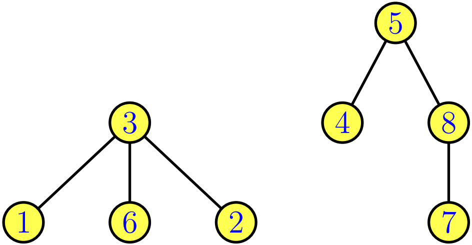
Rule: to add (x, y), find the root x' $[y']$ of the tree containing x $[y]$. If $x' \neq y'$, add the edge (x', y').



Adding (7, 5)

Union-find (1)

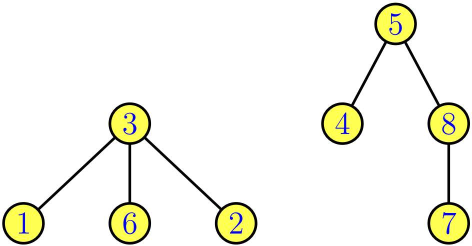
Rule: to add (x, y), find the root x' $[y']$ of the tree containing x $[y]$. If $x' \neq y'$, add the edge (x', y').



Adding (7, 5)

Union-find (1)

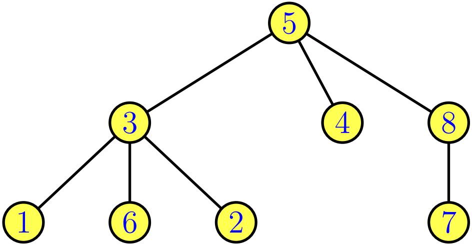
Rule: to add (x, y), find the root x' $[y']$ of the tree containing x $[y]$. If $x' \neq y'$, add the edge (x', y').



Adding (2, 5)

Union-find (1)

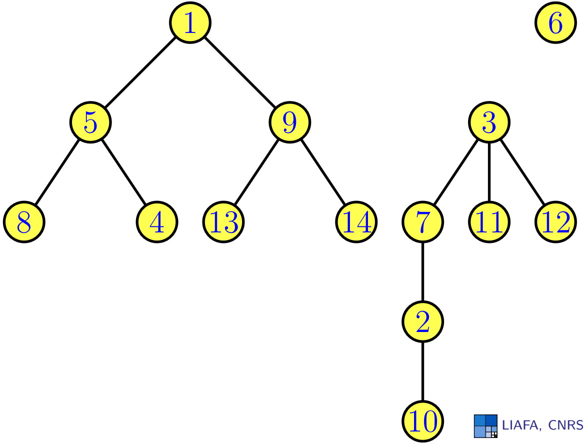
Rule: to add (x, y), find the root x' $[y']$ of the tree containing x $[y]$. If $x' \neq y'$, add the edge (x', y').



Adding (2, 5)

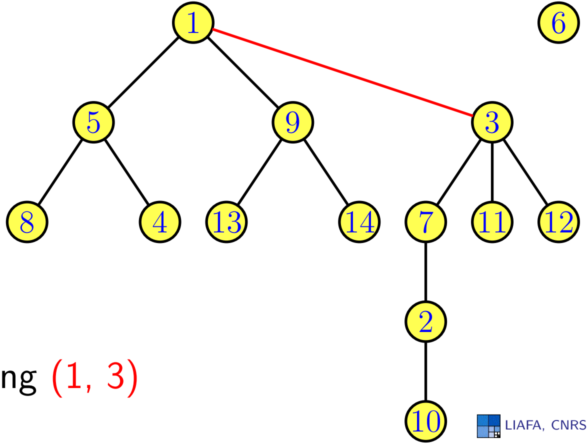
Union-find, union by size

When merging two trees, attach the root of the tree with fewer nodes to the root of the tree with more nodes.



Union-find, union by size

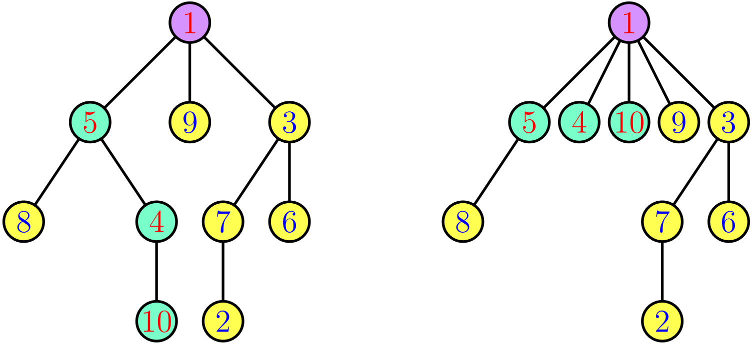
When merging two trees, attach the root of the tree with fewer nodes to the root of the tree with more nodes.



Adding (1, 3)

Union-find, path compression

Do twice the search for the root. The second time, attach all nodes on the path to the root.



Complexity of Union-Find

Tarjan and van Leeuwen have shown that performing m Finds and $n-1$ Unions, with $m \geqslant n$, can be done in $O(n + m\alpha(m,n))$ where α is a kind of inverse of the Ackermann's function.

This function is so slow that for $n < 2^{65536}$ and $m \geqslant n$, $\alpha(m,n) \leqslant 2$. Thus, the algorithm is linear in practice.

In particular, the blocks can be computed in (quasi)-linear time in the number of idempotents.

Summary on complexity

The computation of the elements, the right and left Cayley graphs, the Green's relations, the blocks, the idempotents, the minimal ideal, can be done in time $O(|A||S|)$.

Benchmarks (in seconds)

Name	$|A|$	$|S|$	S	$\mathcal{D}$	$\mathcal{H}$
S10	2	3,628,800	11.02	15.00	0.01
T7	3	823,543	2.95	2.38	0.74
F7	4	2,097,152	8.87	7.36	0.64
I8	3	1,441,729	5.44	5.63	0.42
RB4	4	63,904	0.37	0.10	0.02
FC13	13	5,200,300	35.63	22.61	1.29
FIC12	12	2,704,156	20.57	11.53	0.71
POPI12	2	16,224,937	56.00	66.73	5.90
Tr6	21	2,097,152	33.09	10.43	1.12
U7	21	2,097,152	40.93	9.28	1.04

Part VII

Group radical

Group radical

The group radical of a finite monoid M is the smallest submonoid $D(M)$ of M containing the idempotents and closed under weak conjugation: if $sts = s$ and $d \in D(M)$, then $sdt, tds \in D(M)$.

Computation of the radical

Initialisation : $D(M) = E(M)$

For each d in $D(S)$

> **For** each weakly conjugate pair (s, t)
>   add sdt and tds to $D(S)$
> add $D(S)d$ to $D(S)$.

Time complexity in $0(|S|^3)$.

Part VIII

Syntactic ordered monoid

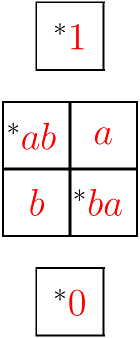
If P is a subset of a monoid M, the syntactic preorder $\leqslant_P$ is defined on M by $u \leqslant_P v$ iff, for all $x, y \in M$,

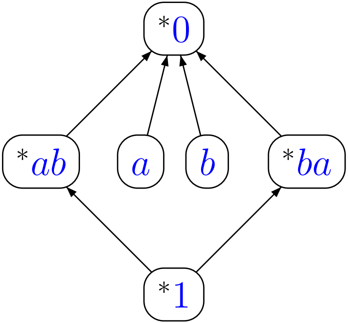
$$xvy \in P \Rightarrow xuy \in P$$

Denote by $\bar{P}$ the complement of P. Then $u \not\leqslant_P v$ iff there exist $x, y \in M$ such that

$$xuy \in \bar{P} \text{ and } xvy \in P$$

The syntactic ordered monoid of ab in B_2^1





An algorithm for the syntactic preorder

Let G be the graph with $M \times M$ as set of vertices and edges of the form $(ua, va) \to (u, v)$ or $(au, av) \to (u, v)$.

We have seen that $u \not\leqslant_P v$ iff there exist $x, y \in M$ such that

$$xuy \in \bar{P} \text{ and } xvy \in P$$

Therefore, $u \not\leqslant_P v$ iff the vertex (u, v) is reachable in G from some vertex of $\bar{P} \times P$.

The algorithm (2)

(1) Label each vertex (u, v) as follows:

$$\begin{cases} (0,1) & \text{if } u \notin P \text{ and } v \in P \quad [u \not\leqslant_P v] \\ (1,0) & \text{if } u \in P \text{ and } v \notin P \quad [v \not\leqslant_P u] \\ (1,1) & \text{otherwise} \end{cases}$$

(2) Do a depth first search (starting from each vertex labeled by $(0, 1)$) and set to 0 the first component of the label of all visited vertices.

Constraint propagation

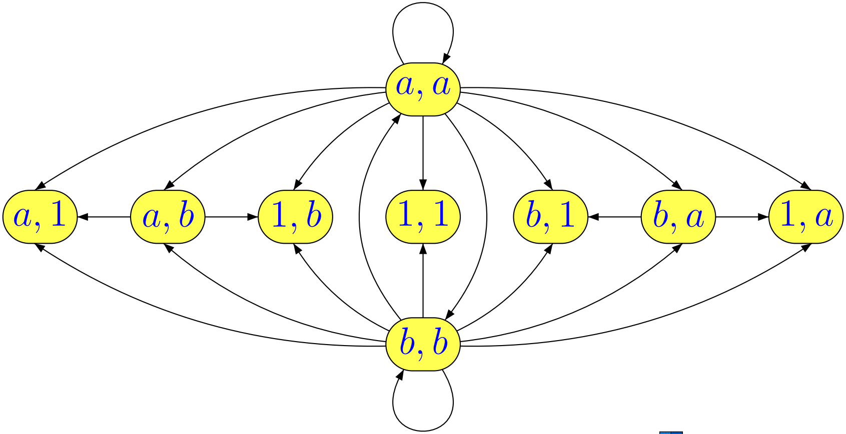
(3) Do a depth first search (starting from each vertex labeled by $(0,0)$ or $(1,0)$) and set to 0 the second component of the label of all visited vertices.

(4) The label of each vertex now encodes the syntactic preorder of P as follows:

$$\begin{cases} (1,1) & \text{if } u \sim_P v \\ (1,0) & \text{if } u \leqslant_P v \\ (0,1) & \text{if } v \leqslant_P u \\ (0,0) & \text{if } u \text{ and } v \text{ are incomparable} \end{cases}$$

Computation of the syntactic preorder

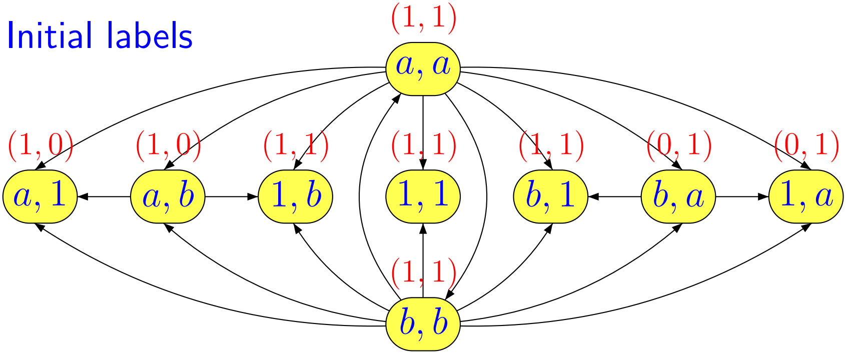
Let $M = \{1, a, b\}$ with $aa = ba = a$ and $ab = bb = b$. Let $P = \{a\}$.



Computation of the syntactic preorder

Let $M = \{1, a, b\}$ with $aa = ba = a$ and $ab = bb = b$. Let $P = \{a\}$.

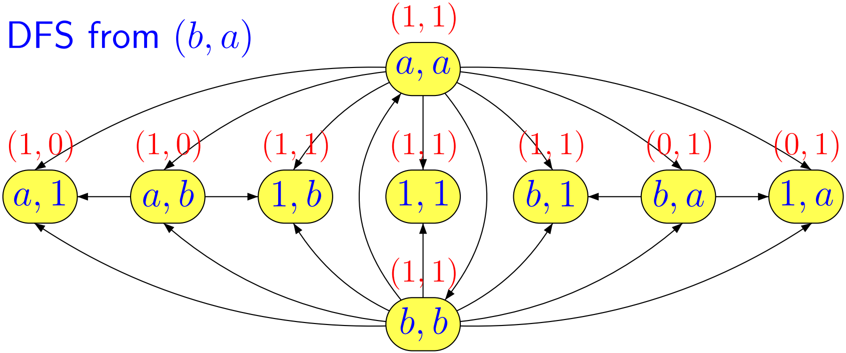
Initial labels



Computation of the syntactic preorder

Let $M = \{1, a, b\}$ with $aa = ba = a$ and $ab = bb = b$. Let $P = \{a\}$.

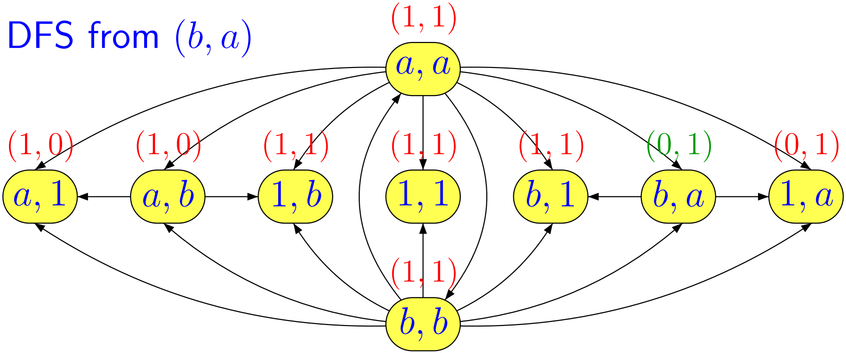
DFS from (b, a)



Computation of the syntactic preorder

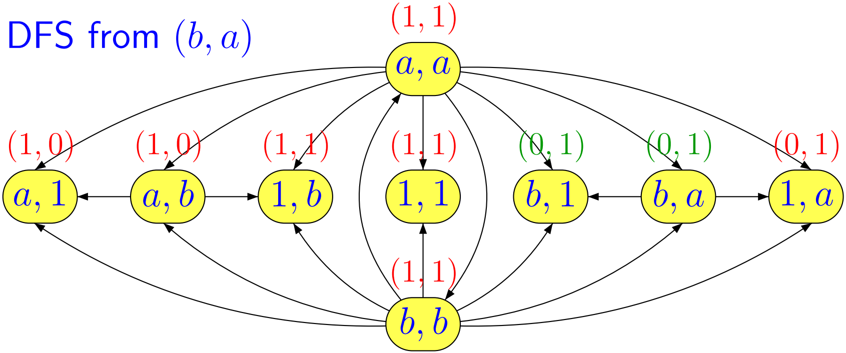
Let $M = \{1, a, b\}$ with $aa = ba = a$ and $ab = bb = b$. Let $P = \{a\}$.

DFS from (b, a)



Computation of the syntactic preorder

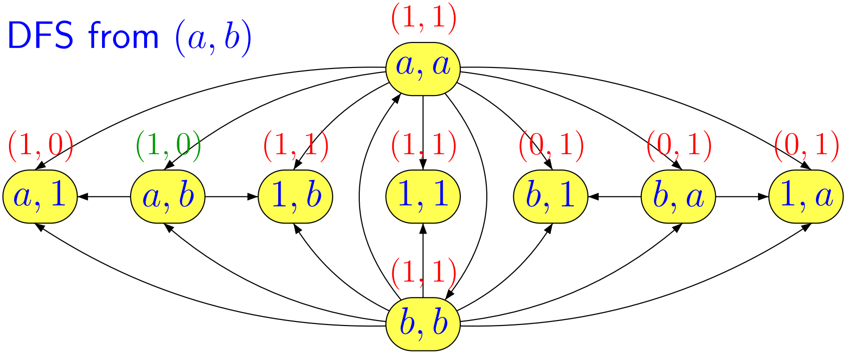
Let $M = \{1, a, b\}$ with $aa = ba = a$ and $ab = bb = b$. Let $P = \{a\}$.



Computation of the syntactic preorder

Let $M = \{1, a, b\}$ with $aa = ba = a$ and $ab = bb = b$. Let $P = \{a\}$.

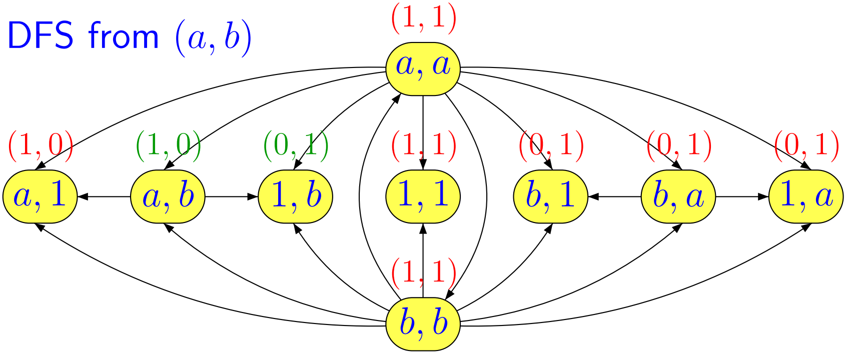
DFS from (a, b)



Computation of the syntactic preorder

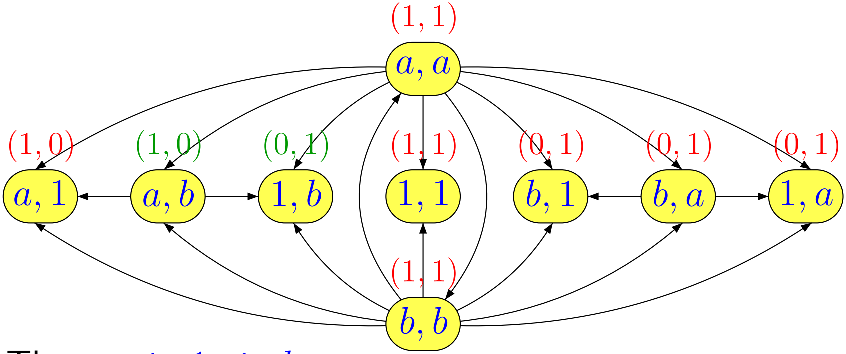
Let $M = \{1, a, b\}$ with $aa = ba = a$ and $ab = bb = b$. Let $P = \{a\}$.

DFS from (a, b)



Computation of the syntactic preorder

Let $M = \{1, a, b\}$ with $aa = ba = a$ and $ab = bb = b$. Let $P = \{a\}$.



Thus $a \leqslant_P 1 \leqslant_P b$

Complexity of the algorithm

The syntactic preorder can be computed in $O(|A||M|^2)$ time and space.

Aperiodicity

Theorem (Cho-Huynh 1991)

Testing aperiodicity of a deterministic n-state automaton is P-space complete.

Proposition

One can test in $O(|A||S|)$-time whether an A-generated finite semigroup S is aperiodic.

It suffices to test whether the $\mathcal{H}$-classes are trivial.

Other varieties

Proposition

One can test in $O(|A||S|)$-time whether an A-generated finite semigroup S is $\mathcal{R}$-trivial [$\mathcal{L}$-trivial, $\mathcal{J}$-trivial, commutative, idempotent, nilpotent, a group, a block-group].

Testing a set of identities

This is a difficult problem for several reasons:

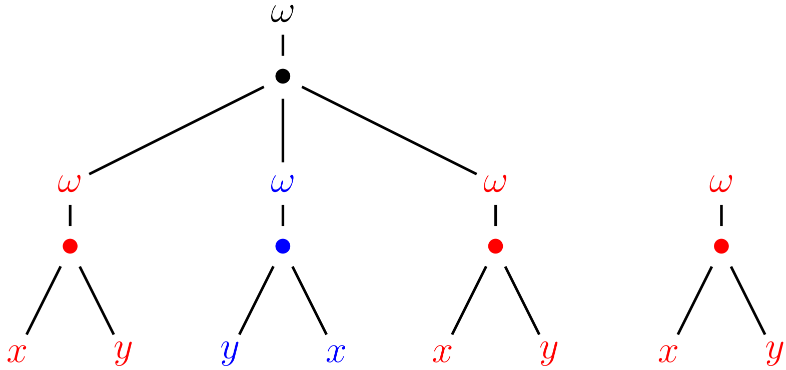
• It may happen that testing whether a set of identities is satisfied is much easier than testing whether any of the individual identities is satisfied.

• Identities for finite semigroups are profinite identities. The operations x^ω and $x^{\omega-1}$ are frequently needed, but other operators might be needed.

• There might be some tricky tree pattern-matching problems to solve.

Tree pattern-matching problems

A simple example: the variety $\mathbf{DS}$ is defined by the identity $((xy)^\omega(yx)^\omega(xy)^\omega)^\omega = (xy)^\omega$



Semigroup theory might help...

Proposition

One can test in $O(|A||S|)$-time whether an A-generated finite semigroup S belongs to $\mathbf{DS}$.

Indeed, a semigroup belongs to $\mathbf{DS}$ iff every regular $\mathcal{D}$-class is union of groups. Therefore, it suffices to test whether the number of regular $\mathcal{H}$-classes is equal to the number of idempotents.

Part IX

New directions

A stamp is a morphism from a finitely generated free monoid onto a finite monoid. An ordered stamp is a stamp onto an ordered monoid.

$$\varphi : A^* \to M$$

Stable subsemigroup

Let $\varphi : A^* \to M$ be a stamp and let $Z = \varphi(A)$. Then Z belongs to the monoid $\mathcal{P}(M)$ of subsets of M.

Since $\mathcal{P}(M)$ is finite, Z has an idempotent power. The stability index of φ is the least positive integer such that $\varphi(A^s) = \varphi(A^{2s})$.

The set $\varphi(A^s)$ is a subsemigroup of M called the stable semigroup of φ and the monoid $\varphi(A^s) \cup \{1\}$ is called the stable monoid of φ.

Applications to logic

Theorem (McNaughton-Paper 1971, Schützenberger 1965)

A language is $\mathbf{FO}[<]$-definable iff its syntactic semigroup is aperiodic.

Theorem (Barrington, Compton, Straubing, Thérien 1992)

A language is $\mathbf{FO}[< + \mathrm{MOD}]$-definable iff the stable semigroup of its syntactic stamp is aperiodic.

A bit of logic

To each nonempty word u is associated a structure

$$\mathcal{M}_u = (\{0, 1, \ldots, |u| - 1\}, <, (\mathbf{a})_{a \in A})$$

where $\mathbf{a}$ is interpreted as the set of integers i such that the i-th letter of u is an a, and $<$ as the usual order on integers.

If $u = abbaab$, then $\mathrm{Dom}(u) = \{0, 1, 2, 3, 4, 5\}$, $\mathbf{a} = \{0, 3, 4\}$ and $\mathbf{b} = \{1, 2, 5\}$.

Modular predicates

Let $d > 0$ and $r \in \mathbb{Z}/d\mathbb{Z}$. We define two new symbols (the modular symbols):

- The unary symbol MOD_r^d:

$$\mathrm{MOD}_r^d(n) = \{i < n \mid i \bmod d = r\}$$

- A constant symbol m for the last position in a word

Fragments of first order logic

$\mathbf{FO}[<]$ denotes the set of first order formulas in the signature $\{<, (\mathbf{a})_{a \in A}\}$.

$\mathbf{FO}[< + \text{MOD}]$ denotes the logic obtained by adjoining all modular symbols.

Fragments of first order logic

$\mathbf{FO}[<]$ denotes the set of first order formulas in the signature $\{<, (\mathbf{a})_{a \in A}\}$.

$\mathbf{FO}[< + \text{MOD}]$ denotes the logic obtained by adjoining all modular symbols.

$\mathbf{\Sigma_1}$ denotes the set of existential formulas:

$$\exists x_1 \cdots \exists x_n \ \varphi(x_1, \ldots, x_n)$$

where φ is quantifier-free.

$\mathcal{B}\mathbf{\Sigma_1}$ denotes the set of Boolean combinations of $\mathbf{\Sigma_1}$-formulas.

Some examples

The formula $\exists x \ \mathbf{a}x$ is interpreted as:

> *There exists an integer x such that, in u, the letter in position x is an a.*

This defines the language A^*aA^*.

Some examples

The formula $\exists x \ \mathbf{a}x$ is interpreted as:

> *There exists an integer x such that, in u, the letter in position x is an a.*

This defines the language A^*aA^*.

The formula $\exists x \ \exists y \ (x < y) \wedge \mathbf{a}x \wedge \mathbf{b}y$ defines the language $A^*aA^*bA^*$.

Some examples

The formula $\exists x \ \mathbf{a}x$ is interpreted as:

> *There exists an integer x such that, in u, the letter in position x is an a.*

This defines the language A^*aA^*.

The formula $\exists x \ \exists y \ (x < y) \wedge \mathbf{a}x \wedge \mathbf{b}y$ defines the language $A^*aA^*bA^*$.

The formula $\exists x \ \forall y \ (x < y) \vee (x = y) \wedge \mathbf{a}x$ defines the language aA^*.

Simple languages

A simple language is a language of the form

$$A^*a_1A^*a_2A^* \cdots \ a_kA^*$$

where $k \geqslant 0$ and $a_1, a_2, \ldots, a_k \in A$.

A modular simple language is a language of the form

$$(A^d)^*a_1(A^d)^*a_2(A^d)^* \cdots \ a_k(A^d)^*$$

where $d > 0$, $k \geqslant 0$ and $a_1, a_2, \ldots, a_k \in A$.

Logical description of simple languages

The language $A^*a_1A^*a_2A^* \cdots\ a_kA^*$ can be defined by the Σ_1-formula

$$\exists x_1\ \ldots\ \exists x_k\ (x_1 < \ldots < x_k) \wedge (\mathbf{a}_1 x_1 \wedge \cdots \wedge \mathbf{a}_k x_k)$$

Logical description of simple languages

The language $A^*a_1A^*a_2A^* \cdots\ a_kA^*$ can be defined by the Σ_1-formula

$$\exists x_1\ \ldots\ \exists x_k\ (x_1 < \ldots < x_k) \wedge (\mathbf{a}_1 x_1 \wedge \cdots \wedge \mathbf{a}_k x_k)$$

The language $(A^d)^*a_1(A^d)^*a_2(A^d)^* \cdots\ a_k(A^d)^*$ can be defined by the Σ_1-formula

$$\exists x_1\ \ldots\ \exists x_k\ (x_1 < \ldots < x_k) \wedge (\mathbf{a}_1 x_1 \wedge \cdots \wedge \mathbf{a}_k x_k) \wedge \\ (\mathrm{MOD}_0^d\, x_1 \wedge \mathrm{MOD}_1^d\, x_2 \wedge \cdots \wedge \mathrm{MOD}_{k-1}^d\, x_k \wedge \mathrm{MOD}_{k-1}^d\, m)$$

First order

Theorem (McNaughton-Paper 1971, Schützenberger 1965)

A language is $\mathbf{FO}[<]$*-definable iff its syntactic semigroup is aperiodic.*

Theorem (Barrington, Compton, Straubing, Thérien 1992)

A language is $\mathbf{FO}[< + \text{MOD}]$*-definable iff the stable semigroup of its syntactic stamp is aperiodic.*

Existential formulas (Σ_1)

Proposition

A language is definable in $\Sigma_1[<]$ *iff it is a finite union of simple languages.*

Proposition

A language is definable in $\Sigma_1[< + \mathrm{MOD}]$ *iff it is a finite union of modular simple languages.*

Algebraic characterization

Theorem (Thomas 1982, Perrin-Pin 1986)

A language is definable in $\mathbf{\Sigma_1}[<]$ *iff its ordered syntactic monoid satisfies the identity* $x \leqslant 1$.

Theorem (Chaubard, Pin, Straubing 2006)

A language is definable in $\mathbf{\Sigma_1}[< + \mathrm{MOD}]$ *iff the stable ordered monoid of its ordered syntactic stamp satisfies the identity* $x \leqslant 1$.

lm-morphisms

A morphism $f : A^* \to B^*$ is length-multiplying (lm for short) if there exists an integer k such that the image of each letter of A is a word of B^k.

For instance, if $A = \{a, b\}$ and $B = \{a, b, c\}$, the morphism defined by $\varphi(a) = abca$ and $\varphi(b) = cbba$ is length-multiplying.

lm-identities

Let u, v be two words on the alphabet B. A morphism $\varphi : A^* \to M$ satisfies the lm-identity $u = v$ if, for every lm-morphism $f \colon B^* \to A^*$, $\varphi \circ f(u) = \varphi \circ f(v)$.

For instance, $\varphi : A^* \to M$ satisfies the lm-identity $xyx = xy$ if for any pair of words of the same length x, y of A^*, $\varphi(xyx) = \varphi(xy)$.

lm-identities

Let u, v be two words on the alphabet B. A morphism $\varphi : A^* \to M$ satisfies the lm-identity $u = v$ if, for every lm-morphism $f\colon B^* \to A^*$, $\varphi \circ f(u) = \varphi \circ f(v)$.

For instance, $\varphi : A^* \to M$ satisfies the lm-identity $xyx = xy$ if for any pair of words of the same length x, y of A^*, $\varphi(xyx) = \varphi(xy)$.

If M is ordered, we say that φ satisfies the lm-identity $u \leqslant v$ if, for every lm-morphism $f\colon B^* \to A^*$, $\varphi \circ f(u) \leqslant \varphi \circ f(v)$.

Characterization by lm-identities

Theorem (Thomas 1982, Perrin-Pin 1986)

A language is definable in $\mathbf{\Sigma_1}[<]$ *iff its ordered syntactic monoid satisfies the identity* $x \leqslant 1$.

Theorem (Chaubard, Pin, Straubing 2006)

A language is definable in $\mathbf{\Sigma_1}[< + \mathrm{MOD}]$ *iff its ordered syntactic stamp satisfies the* lm*-identities* $x^{\omega-1}y \leqslant 1$ *and* $yx^{\omega-1} \leqslant 1$.

Boolean combination of existential formulas

Theorem (Thomas 1982)

A language is definable in $\mathcal{B}\Sigma_1[<]$ iff it is a Boolean combination of simple languages.

Theorem (Chaubard, Pin, Straubing 2006)

A language is definable in $\mathcal{B}\Sigma_1[< + \mathrm{MOD}]$ iff it is a Boolean combination of modular simple languages.

Algebraic characterization

Theorem (Simon 1972, Thomas 1982)

A language is definable in $\mathcal{B}\mathbf{\Sigma_1}[<]$ iff its syntactic monoid is $\mathcal{J}$-trivial.

Theorem (Chaubard, Pin, Straubing 2006)

A language is definable in $\mathcal{B}\mathbf{\Sigma_1}[< + \mathrm{MOD}]$ iff its syntactic stamp belongs to the lm-variety $\mathbf{J} * \mathbf{MOD}$.

Derived category of a stamp $\varphi : A^* \to M$

Let $\pi_n(u) = |u| \bmod n$.

$$\begin{tikzcd} & A^* \arrow[dl, "\varphi"'] \arrow[dr, "\pi_n"] & \\ M & & \mathbb{Z}/n\mathbb{Z} \end{tikzcd}$$

Let $C_n(\varphi)$ be the category whose objects are elements of $\mathbb{Z}/n\mathbb{Z}$ and whose arrows from i to j are the triples (i, m, j) where $j - i \in \pi_n(\varphi^{-1}(m))$.

Composition is given by $(i, m_1, j)(j, m_2, k) = (i, m_1m_2, k)$.

A decidable characterization

Theorem (Chaubard, Pin, Straubing 2006)

Let φ be a stamp of stability index s. Then φ belongs to $\mathbf{J} * \mathbf{MOD}$ iff $C_s(\varphi)$ is in $g\mathbf{J}$.

Corollary

Let φ be the syntactic stamp of a language L and let s be its stability index. Then L is definable in $\mathcal{B}\mathbf{\Sigma_1}[< + \mathrm{MOD}]$ iff $C_s(\varphi)$ is in $g\mathbf{J}$.

No characterization by lm-identities is known at the moment.

What would be useful in GAP 4...

- Define stamps as a basic object.

- Compute stable semigroups and monoids of stamps.

- Test for length-preserving and length-multiplying identities.

- Compute derived categories

References I

- V. Froidure and J.-E. Pin, Algorithms for computing finite semigroups, in *Foundations of Computational Mathematics*, F. Cucker et M. Shub (éd.), Berlin, 1997, pp. 112–126, Springer.
- Semigroupe, C programme, available at `http://www.liafa.jussieu.fr/~jep/Logiciels/Semigroupe/semigroupe.html`
- L. Chaubard, J.-E. Pin and H. Straubing, First order formulas with modular predicates, in *Proceedings of LICS'06*, 2006.