Geometrical Languages

Baptiste Blanpain, Jean-Marc Champarnaud, Jean-Philippe Dubernard

Geometrical Languages

B. Blanpain, J.-M. Champarnaud and J.-P. Dubernard
LITIS, University of Rouen, France
{jean-marc.champarnaud,jean-philippe.dubernard}@univ-rouen.fr

Abstract

Our aim is to introduce and generalize a class of languages used in off-line temporal validation of real-time softwares. Computing the feasibility of an application consists in checking whether there exists a sequence of execution which allows each task to run without any time error. This can be done with a model based on automata and languages. Because of certain properties of these languages, a model based on discrete geometry can also be used to make the computing time smaller. In this paper, we develop the link between languages and geometry, by defining a new class of languages, which we call geometrical languages. The main result is an algorithm that checks wether a given regular language is geometrical or not, by means of equations with positive integer variables.

Keywords: Finite automata, languages, discrete geometry, real-time applications, temporal validation.

1 Introduction

Real-time applications are composed of both periodic and sporadic tasks. Periodic tasks can share resources and communicate. Sporadic tasks are independent from others. Time specifications of a real-time task have been defined in [8]. A periodic task τ is composed of four temporal characteristics:

- r, the first activation date, is the date of creation of τ;
- D, the critical delay, is the delay between the activation date of an instance of τ and its deadline (later possible completion date);
- C, the computation time, is the total CPU owning time needed by each instance of τ to end its execution;

- T, the period, is the delay between the activation dates of two instances of τ.

The problem of computing the feasibility of an application (on a given architecture) consists in checking whether there exists a sequence of execution which allows each task to run without any time error. This question is not an easy one, because the number of processors of the architecture should be considered, as well as the fact that the tasks can share resources.

A first well-known model, based on finite automata, can be used to achieve these computations [5]. The temporal behavior of one periodic task is described by the following language over the alphabet $\{a, \bullet\}$:

$$\bullet^r \cdot ((a^C \sqcup\!\sqcup \bullet^{D-C}) \cdot \bullet^{T-D})^*$$

where the symbol a (resp. $\bullet$) is coding a time unit where the task is active (resp. the task does not own a CPU). We can then build the automaton corresponding to this language (see Figure 1).

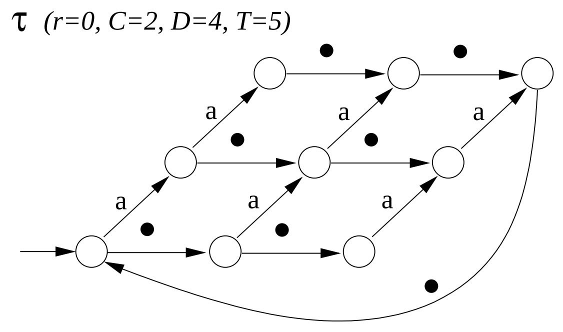

Figure 1: Automaton associated with a periodic task (drawn from [7]).

Then, a homogeneous product [7] of all the automata corresponding to the periodic tasks of the application, gives us an automaton which represents all the possible simultaneous executions of the periodic tasks. Infinite paths in the automaton represent valid sequences of execution of the application.

The main problem in this model is that the computation of the homogeneous product takes a lot of time, because of a combinatorial explosion. Computing an automaton model takes at least two hours for a real application.

Another model based on discrete geometry [7], has been designed to achieve these calculations. Since the automaton associated to a task has a particular pattern, we can transform it into a geometrical figure (see Figure 2). Moreover, the homogeneous product on automata is equivalent to the construction of a n-dimension geometrical volume[1]. The number of trajectories in this volume is computed, and the application is declared valid if and only if it is greater than zero. This new model is very efficient in time. The tests realized by G. Largeteau [7] show that a set of tasks can be evaluated in one second, whereas the model based on automata takes two hours.

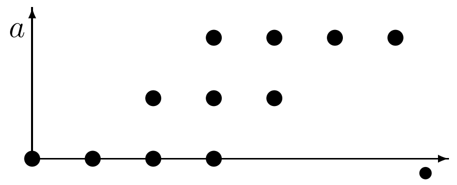

Figure 2: Automaton -> Geometrical figure.

Representing a language with a geometrical figure allows us to withdraw the transitions. This is possible with the languages involved by temporal validation because these languages have good properties, as we can see on the Figure 1. Two points that are close together in discrete geometry correspond to two states linked by some transition in the automaton. Moreover, in a geometrical figure, all the points correspond to final states, which fits with the fact that temporal validation languages are prefix languages. We can then use binary matrixes to represent such automata.

In this article, we study the languages that can be represented by geometrical figures. In the next section, we recall some important notions concerning automata and languages. In the third section we define a class of languages, which we call geometrical languages, and we present some of its properties. In the fourth section we give necessary and sufficient conditions for a language to be geometrical. In the last section, we focus on methods to check whether a given regular language is geometrical or not.

2 Preliminaries

Let us review basic notions and terminology concerning finite automata and regular expressions. For further details, [4, 9] are excellent books.

Let Σ be a non-empty finite set of symbols, called the alphabet. A word over Σ is a finite sequence $(x_1, x_2, ..., x_n)$ of symbols, usually

[1]Notice that, real-time temporal validation adresses finite volumes, whose size depends on the lowest common multiple of the periods of the tasks.

written $x_1x_2...x_n$. The length of a word u, denoted $|u|$, is the number of symbols in u. The empty word, denoted by ε, has a length equal to zero. If $u = x_1x_2...x_n$ and $v = y_1y_2...y_m$ are two words over the alphabet Σ, their concatenation $u \cdot v$, usually written uv, is the word $x_1x_2...x_ny_1y_2...y_m$. A language L is a set of words. We denote by $w^{-1}L$ and we call left residual of L by the word w, the set of words $u \in \Sigma^*$ such that $wu \in L$.

An automaton is a 5-tuple (Q, Σ, δ, I, F) where Q is the set of states, δ is a subset of $Q \times \Sigma \times Q$ whose elements are called transitions of the automaton, and where I and F are subsets of Q, whose elements are respectively called initial states and final states of the automaton. An automaton A is said to be finite if Q is finite. It is said to be deterministic if it has a unique initial state and if for any $q \in Q$ and any $a \in \Sigma$, $\delta(q, a)$ contains at most one element. If the automaton is deterministic, we can use the notation $p.a$ instead of $\delta(p, a)$.

A path in A is a sequence of transitions: $(q_0,a_1,q_1),(q_1,a_2,q_2)...,(q_{n-1},a_n,q_n)$. The word $a_1a_2...a_n$ is called the label of this path. A path is said to be a successful path if $q_0 \in I$ and $q_n \in F$. The language recognized by A is the set of words that are labels of successful paths.

The left language of a state q is the set of words w such that there exists a path in A whose first state is initial, whose last state is q, and whose label is w. Symmetrically, the right language of a state q is the set of words w such that there exists a path in A whose first state is q, whose last state is final, and whose label is w. We denote by $\overleftarrow{L}_q$ the left language of q, and by $\overrightarrow{L}_q$ its right language.

A deterministic automaton A is said to be minimal if any two distinct states of A have distinct right languages. A given language L has a unique minimal automaton (up to isomorphism).

The set of regular languages is the set containing $\emptyset$, $\{\varepsilon\}$, and $\{a\}$ for every a in the alphabet, and which is closed under finite concatenation and union, and star. Any regular language can be denoted by a regular expression, formed by the atoms $\emptyset$, $a \in \Sigma$, and the operations of concatenation $(\cdot)$, union$(\cup)$ and star $(*)$. Kleene's theorem says that a language is recognized by a finite automaton if and only if it is regular.

The shuffle of to words u and v in Σ^*, denoted by $u ш v$, is the set of words defined as follows:

$$u \text{ ш } v = \begin{cases} \{v\} & \text{if } u = \varepsilon \\ \{u\} & \text{if } v = \varepsilon \\ a \cdot (u' \text{ ш } v) \cup b \cdot (u \text{ ш } v') & \text{if } u = a \cdot u',\ v = b \cdot v',\ a, b \in \Sigma \end{cases},$$

The shuffle of two languages L and M is: $L ш M = \{u ш v / u \in L, v \in M\}$.

We say that u in Σ^* is a prefix of w in Σ^*, and we denote $u \in Pref(w)$ if there exists v in Σ^* such that $uv = w$. By extension we denote $Pref(L)$ the set of prefixes of the words of the language L.

3 Geometrical Languages

We call geometrical figure F of dimension d, a set of points of coordinates in $\mathbb{N}^d$, that is $F \subseteq \mathbb{N}^d$. For instance, $F = \{(0,0),(0,1),(1,0),(1,1)\}$ (Figure 3) is a 2-dimension geometrical figure.

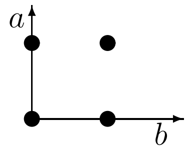

Figure 3: $F = \{(0,0),(0,1),(1,0),(1,1)\}$.

Let $1 \leqslant k \leqslant d$. We denote $\vec{u}_k$ the coordinate vector $(0,0,...,1,...0)$, where 1 is in the k-th position. For instance, in 3-dimension $\vec{u}_2 = (0,1,0)$. We denote $\mathcal{O}$ the point $(0,0,...,0)$, the zero vector. In what follows, we can denote $P = \vec{\mathcal{O}P}$. For instance, $P = (3,2)$.

A trajectory t of length n in the d-dimension figure F is a sequence of points $(P_i)_{0 \leqslant i \leqslant n}$, such that:

- $\forall i, 0 \leqslant i \leqslant n, P_i \in F$,
- $\mathrm{P}_0 = \mathcal{O}$
- for $0 \leqslant i \leqslant n-1$, $\exists k, 1 \leqslant k \leqslant d$ such that $P_{i+1} = P_i + \vec{u}_k$.

We denote by $Traj(F)$ the set of trajectories of F.

In what follows, we only consider d-dimension geometrical figures, with d finite. Moreover, every point of a geometrical figure is reachable from a trajectory. Once defined these notions, we have to connect them to those of words and languages. We use a finite alphabet $\Sigma = \{a_1, a_2, ..., a_d\}$, and we associate the letter a_i to the dimension i. First we introduce a way to associate a word to a trajectory $t = (P_i)_{0 \leqslant i \leqslant n}$ of the figure F. We associate to t the word w of length n, such that $\forall i, 0 \leqslant i \leqslant n-1, w_{i+1} = a_k$, where $P_{i+1} = P_i + \vec{u}_k$. We denote by $word(t) = w$ and we denote $traj(w)$ the set of points in the trajectory associated to w.

We also define the fuction $c : \Sigma^* \longrightarrow \mathbb{N}^d$, associating to a word w the vector $(|w|_{a_1}, |w|_{a_2}, ..., |w|_{a_d})$. This vector is called the coordinate vector of w. Notice that c is a morphism: $c(\varepsilon) = (0,...,0) = \mathcal{O}$, and $c(u.v) = c(u) + c(v)$.

We can now associate a geometrical figure to the language L. We denote this figure $\mathcal{F}(L) \subseteq \mathbb{N}^d$, and we define it by

$$\mathcal{F}(L) = \bigcup_{w \in Pref(L)} traj(w)$$

For example, in Figure 3, $F = \{(0,0),(0,1),(1,0),(1,1)\}$ is the figure associated to the langage $L = \{ab, ba\}$. Notice that for any language L, $\mathcal{F}(L) = \mathcal{F}(Pref(L))$.

It is easily checked that two different languages L and L' ($L \neq L'$) can have the same geometrical figure: $\mathcal{F}(L) = \mathcal{F}(L')$. For instance, if $Pref(L) = Pref(L')$, then $\mathcal{F}(L) = \mathcal{F}(L')$. But two languages can have the same geometrical figure without having the same set of prefixes. For example, $L = \{ab, ba\}$ and $L' = \{ab, b\}$ have the same figure (see figure 3) but have not the same set of prefixes.

Now we define a way to associate a language to a geometrical figure $F \subseteq \mathbb{N}^d$. We denote by $\mathcal{L}(F)$ the language associated to F and defined by

$$\mathcal{L}(F) = \bigcup_{t \in Traj(F)} word(t)$$

We now introduce the main object of this study. Geometrical languages are those of which the set of prefixes is exactly the set words having a trajectory in the figure, as stated by the following definition:

Definition 3.1 (Geometrical language) *Let $L \subseteq \Sigma^*$ be a language. L is said to be geometrical if and only if $Pref(L) = \mathcal{L}(\mathcal{F}(L))$.*

For every language L, $Pref(L) \subseteq \mathcal{L}(\mathcal{F}(L))$. But some languages are such that $\mathcal{L}(\mathcal{F}(L)) \nsubseteq Pref(L)$. It is the case of $\{a, ba\}$ which is not geometrical, whereas $\{ab, ba\}$ is geometrical.

Notice that, according to our definition, a geometrical language can be infinite. Temporal validation languages are examples of regular geometrical languages. But we can also find geometrical languages in other classes of Chomsky's hierarchy.

Example 3.2 *The Dyck language, which is formed by the words with properly balanced parentheses, is context-free [2] and is geometrical. If we look at its geometrical figure (see Figure 4), we can see that every trajectory corresponds to a word with properly balanced parentheses. Reciprocally, every word of the Dyck language has a trajectory in the figure.*

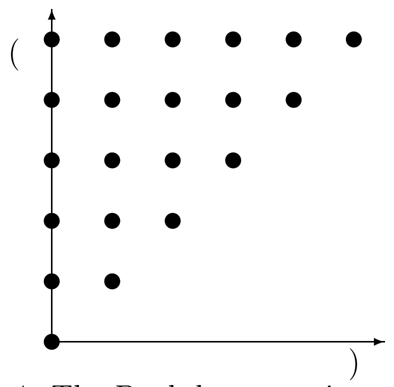

Figure 4: The Dyck language is geometrical.

4 Characterization of Geometrical Languages

Our aim is to state necessary and sufficient conditions for a language to be geometrical. We first adress the general case, and then the regular case.

4.1 General Case

Proposition 4.1 *Let $\Sigma = \{a_1, a_2, ..., a_d\}$ and $L \subseteq \Sigma^*$. The two following conditions are equivalent:*
(1) *L is geometrical.*
(2) $\forall u, v \in Pref(L)$, $\underbrace{\exists k, 1 \leq k \leq d,\ c(u) + \vec{u}_k = c(v)}_{(*)} \Rightarrow u{\cdot}a_k \in Pref(L)$

Example 4.2 *$\Sigma = \{a, b\}$, $L = \{a, ba, bb\}$, $u = a$, $v = ba$. Here, $c(u) + (0, 1) = c(v)$ (condition $(*)$). The proposition tells that, if L was geometrical, then ab would be in L. We conclude that L is not geometrical.*

Proof
(1) $\Rightarrow$ (2) Let L be a language and $u, v \in Pref(L)$, such that $(*)$ is satisfied. We have $u, v \in \mathcal{L}(\mathcal{F}(L))$ and consequently there exist a trajectory $(U_i)_{0 \leqslant i \leqslant |u|}$ and a trajectory $(V_i)_{0 \leqslant i \leqslant |v|}$ in the figure $\mathcal{F}(L)$. From condition $(*)$ there exists k such that $1 \leqslant k \leqslant |\Sigma|$ and $c(u) + \vec{u}_k = c(v)$. Thus there is a trajectory $(W_i)_{0 \leqslant i \leqslant |v|}$ in the figure $\mathcal{F}(L)$, such that $\forall i, 0 \leqslant i \leqslant |v| - 1, W_i = U_i$, and $W_{|v|} = V_{|v|}$. The word associated to this trajectory is ua_k, thus $ua_k \in \mathcal{L}(\mathcal{F}(L))$. Consequently, if L is geometrical, it holds $ua_k \in Pref(L)$.

(2) $\Rightarrow$ (1) Let us assume that L is not geometrical. Then there exists a word w such that $w \in \mathcal{L}(\mathcal{F}(L))$ and $w \notin Pref(L)$. Let u be the longest prefix of w such that $u \in Pref(L)$ and a_k the symbol such that $v = ua_k$ is the prefix of length $|u| + 1$ of w. Notice that u and a_k are well defined since $\varepsilon \in Pref(L)$ and $w \notin Pref(L)$. Since v is a prefix of w we have $v \in \mathcal{L}(\mathcal{F}(L))$ and consequently there exists $v' \in Pref(L)$ such that $c(v') = c(v)$. Since $c(v) = c(u) + \vec{u}_k$ the words u and v' satisfy the condition $(*)$ but ua_k is not in $Pref(L)$. Hence the contradiction.

The following notion is useful to prove the proposition 4.5.

Definition 4.3 (Semi-geometrical language) *A language L is semi-geometrical if and only if $\forall u, v \in Pref(L)$ such that $c(u) = c(v)$, we have $u^{-1}Pref(L) = v^{-1}Pref(L)$.*

Proposition 4.4 *If a language is geometrical then it is semi-geometrical.*

Proof

Let $L \subseteq \Sigma^*$ be an arbitrary language. Let $u, v \in Pref(L)$ with $c(u) = c(v)$. Let w be a word of Σ^* such that $vw \in Pref(L)$. Then there exists a trajectory $(U_i)_{0 \leqslant i \leqslant |u|}$ associated to u and a trajectory $(P_i)_{0 \leqslant i \leqslant |vw|}$ associated to vw in the figure $\mathcal{F}(L)$. Since $c(u) = c(v)$ there exists also a trajectory $(Q_i)_{0 \leqslant i \leqslant |uw|}$, such that $\forall i, 0 \leqslant i \leqslant |u|, Q_i = U_i$, and $\forall i, |u| \leqslant i \leqslant |uw|, Q_i = P_i$. The word associated to this trajectory is uw, thus $uw \in \mathcal{L}(\mathcal{F}(L))$. If L is not semi-geometrical then there exists such a word w satisfying $uw \notin Pref(L)$, which implies $\mathcal{L}(\mathcal{F}(L)) \neq Pref(L)$. Finally, if L is not semi-geometrical then L is not geometrical.

4.2 Regular Case

Notation

Let p be a state of an automaton. We denote by E_p the regular expression of the left language of p.

Proposition 4.5 *Let $L \subseteq \Sigma^*$, and $A = (Q, \Sigma, \delta, q_0, T)$ the minimal automaton of $Pref(L)$. L is geometrical if and only if*

$$\forall p, q \in T, (p, a_k, q) \notin \delta \Rightarrow \exists u \in \overleftarrow{L_p} \text{ and } \exists v \in \overleftarrow{L_q} \text{ such that } c(u) + \vec{u}_k = c(v).$$

Proof

First the implication ($\Rightarrow$), by contraposition:

Let us assume that $\exists u \in \overleftarrow{L_p}$ and $\exists v \in \overleftarrow{L_q}$ such that $c(u)+\vec{u}_k = c(v)$. Thus $c(u \cdot a_k) = c(v)$, but since $(p, a_k, q) \notin \delta$, we have $q_0 \cdot ua_k \neq q_0 \cdot v$.

- If $q_0 \cdot ua_k \in T$, then L is not semi-geometrical, since $c(ua_k) = c(v)$, $ua_k, v \in Pref(L)$ and $q_0 \cdot ua_k \neq q_0 \cdot v$, thus $ua_k{}^{-1}Pref(L) \neq v^{-1}Pref(L)$. Consequently L is not geometrical.

- Else ($q_0 \cdot ua_k = shaft$). Then $u, v \in Pref(L)$, $ua_k \notin Pref(L)$, then by the proposition 4.1, L is not geometrical.

Now let us prove the converse (by contraposition):

Let us assume that L is not geometrical. $\exists u, v \in Pref(L), \exists k, 1 \leq k \leq |\Sigma|$, such that $c(u) + \vec{u}_k = c(v)$, but $ua_k \notin Pref(L)$.

Let $p = q_0 \cdot u \in T$ and $q = q_0 \cdot v \in T$.

If we had $(p, a_k, q) \in \delta$, then we would have $q_0 \cdot ua_k = p \cdot a_k = q \in T$, thus $ua_k \in Pref(L)$, contradicting $ua_k \notin Pref(L)$.

Then $(p, a_k, q) \notin \delta$, that ends the proof.

5 Checking wether a given regular language is geometrical or not

To answer the question "Is a given regular language geometrical ?", we propose a method based on equations systems. Let us first extend the function c that associates to a word w its coordinates vector, and then introduce our algorithm.

5.1 Coordinates of a language

If L is a language, then let $c(L)$ be the set of coordinates of the words of L.

$$c(L) = \bigcup_{w \in L} c(w)$$

To represent a set of coordinates, we use a vector x of arithmetic expressions, added with constraints on some of the integer variables that occur in expressions.

Example 5.1 *The set* $\{(0,4),(1,3),(2,2),(3,1),(4,0)\}$ *is represented by the pair* $x = (n_1, n_2)$*, where* $n_1, n_2 \in \mathbb{N}$*, with the constraint* $n_1 + n_2 = 4$.

This notation is expected to simplify our calculations on regular languages, as illustrated by the following examples.

Example 5.2 *Here are several regular expressions, and the x-representation of the associated set of coordinates:*

$x(a^* \cdot b) = (n, 1)$*,* $n \in \mathbb{N}$*. Any word of* $a * b$ *has* n *a and one b.*

$x((ab)^* \cdot b^* \cdot a) = (n_1 + 1, n_1 + n_2)$, *where* $n_1, n_2 \in \mathbb{N}$.
$x(a + ab) = (n_1 + n_2, n_2)$, *where* $n_1, n_2 \in \mathbb{N}$ *and* $n_1 + n_2 = 1$.
$x((aa + ab)^*) = (2n_1 + n_2, n_2)$, *where* $n_1, n_2 \in \mathbb{N}$.

Here is the inductive definition of x. Let R and S be regular expressions, representing two languages on $\Sigma = \{a_1, a_2, ..., a_d\}$.

$$x(\varepsilon) = (0, 0, ..., 0) = \mathcal{O}$$
$$x(a_k) = \vec{u}_k, \text{ where } a_k \in \Sigma$$
$$x(R \cdot S) = x(R) + x(S)$$
$$x(R + S) = n_1 x(R) + n_2 x(S), \text{ where } n_1, n_2 \in \mathbb{N}, n_1 + n_2 = 1$$
$$x(R^*) = n_1 x(R), \text{ where } n_1 \in \mathbb{N}$$

Example 5.3 *Here is a detailed calculation corresponding to a bottom-up traversal of the syntax tree of the expression* $E = ((b + ab)b)^*$*:*

$$\begin{aligned}
x(b) &= (0, 1) \\
x(a) &= (1, 0) \\
x(b) &= (0, 1) \\
x(ab) &= (1, 1) \\
x(b + ab) &= (n_2, n_1 + n_2) \quad \textit{with} \quad n_1 + n_2 = 1 \\
x(b) &= (0, 1) \\
x((b + ab)b) &= (n_2, n_1 + n_2 + 1) \\
x(((b + ab)b)^*) &= (n_3 n_2, n_3(n_1 + n_2 + 1))
\end{aligned}$$

This result can be simplified by considering the two variables $n'_2 = n_3 n2$ and $n'_1 = n_3 n1$. It comes $x(((b + ab)b)^*) = (n'_2, n'_1 + n'_2 + n3))$ *with* $n'_1 + n'_2 = n_3$.

In order to directly obtain simplified expressions, $x(E)$ can be computed through a topdown parsing of E according to the following rules (with $n \in \mathbb{N}$):

$$nx(\varepsilon) = (0, 0, ..., 0) = \mathcal{O}$$
$$nx(a_k) = n\vec{u}_k, \text{ where } a_k \in \Sigma$$
$$nx(R \cdot S) = nx(R) + nx(S)$$
$$nx(R + S) = n_1 x(R) + n_2 x(S), \text{ where } n_1, n_2 \in \mathbb{N}, n_1 + n_2 = n$$
$$nx(R^*) = nn_1 x(R), \text{ where } n_1 \in \mathbb{N}$$

Example 5.4 *Here is a detailed calculation:*

$$\begin{aligned}
x(((b+ab)b)^*) &= n_1x((b+ab)b)\\
&= n_1x(b+ab)+n_1x(b)\\
&= n_2x(b)+n_3x(ab)+(0,n_1) \quad \textit{with} \quad n_1=n_2+n_3\\
&= (0,n_2)+(n_3,n_3)+(0,n_1)\\
&= (n_3 \ , \quad n_1+n_2+n_3)
\end{aligned}$$

If $w \in ((b+ab)b)^*$, then $\exists n_1, n_2, n_3 \in \mathbb{N}$, with $n_1 = n_2 + n_3$ and such that $|w|_a = n_3$ and $|w|_b = n_1 + n_2 + n_3$.

Notice that we can also deal with regular expressions extended to the shuffle:

$$nx(R ш S) = nx(R) + nx(S)$$

5.2 Our algorithm

We give an algorithm which associates the Proposition 4.5 and the function x, to check wether a given regular language is geometrical:

- Build the minimal automaton recognizing $Pref(L)$.
- For each state p, compute E_p.
- For each pair $(p,q) \in (Q\backslash shaft)^2$, and for each $a_k \in \Sigma$, if $(p, a_k, q) \notin \delta$, then check that $c(E_p) + \vec{u}_k = c(E_q)$ has no solution.

Example 5.5 *$L = (aab + baa)^*$. We will check if L is geometrical. Notice that L is semi-geometrical. First we build an automaton recognizing L (figure 5). Then we compute:*
$E_2 = (aab + baa)^*a$ and $E_5 = (aab + baa)^*ba$.
$c(E_2) = (2n_1+2n_2+1, n_1+n_2)$ and $c(E_5) = (2n_3+2n_4+1, n_3+n_4+1)$.
To simplify, we can write $n_1 := n_1 + n_2$ and $n_3 := n_3 + n_4$, to obtain:
$c(E_2) = (2n_1 + 1, n_1)$ and $c(E_5) = (2n_3 + 1, n_3 + 1)$.
Since the transition $(2, b, 5)$ does not exist, let us verify if $c(E_2) + (0, 1) = c(E_5)$ has solutions. We obtain the following system:

$$\begin{cases} 2n_1 + 1 = 2n_2 + 1 \\ n_1 + 1 = n_2 + 1 \end{cases}$$

This system has an infinite number of solution (take $n_1 = n_2$). That means, by the last proposition, that L is not geometrical. These solutions correspond to the words ab and ba.

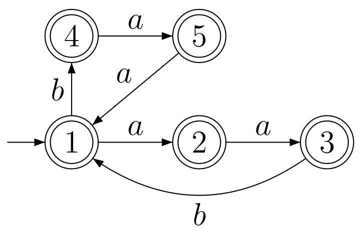
Figure 5: Minimal automaton for $Pref((aab + baa)*)$.

6 Conclusion

Our aim in this paper was to introduce and characterize a new class of languages, that we called "geometrical languages", and a method based on equations systems to check whether a given regular language is geometrical.

These results are just a part of a dissertation done for a research degree [3]. In this report, some others approachs are developed to study the nature of geometrical languages. The cases of finite and context-free languages are also studied.

No doubt this work can be carried on with practical and theoritical purposes.

References

[1] A. Arnold, M. Nivat, *Comportements de Processus*, colloque AFCET, Paris, 1982.

[2] J.-M. Autebert, *Langages Algébriques*, Masson, 1987.

[3] B. Blanpain, *Automates, Langages et Géométrie*, Mémoire de DEA, université de Rouen, 2006.

[4] S. Eilenberg, *Automata, Languages and Machines, Vol.A and B*, Academic Press, 1976.

[5] D. Geniet, *Validation d'Applications Temps-Réel à Contraintes Strictes à l'Aide de Langages Rationnels*, Real Time Systems, Teknea, 2000.

[6] D. Geniet, J.Ph. Dubernard, *Scheduling Hard Sporadic Tasks with Regular Languages and Generating Functions*, Theoritical Computer Science 313, 2004, 119-132.

[7] G. Largeteau, *Quantification du taux d'invalidité d'applications temps-réel à contraintes strictes*, Thèse de l'université de Poitiers, 2004.

[8] C.L. Liu et J.W. Layland, *Scheduling Algorithms for multiprogramming in real-time environment*, Journal of the ACM, 1973, pp. 46-61.

[9] J. Sakarovitch, *Eléments de théorie des automates*, Vuibert Informatique, 2003.