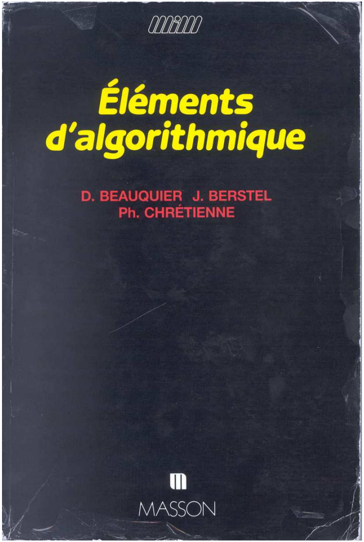
Résultats récents sur deux problèmes anciens

Jean Berstel

Institut Gaspard-Monge, Université Paris-Est

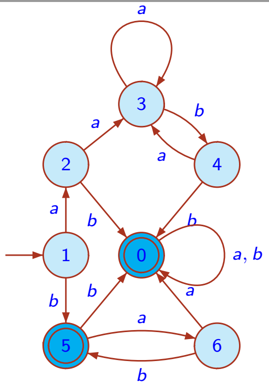
Créteil, 15 juin 2009

Outline



- C'est dans ce livre qu'est paru la première rédaction (et la seule à ce jour, je crois), à l'usage des étudiants d'université, de l'algorithme de Hopcroft.
- Cette rédaction a été faite par Danièle Beauquier.

Automata



Each state q defines a language $L_q = \{w \mid q \cdot w \text{ is final}\}$.

The automaton is minimal if all languages L_q are distinct.

Here $L_2 = L_4$. States 2 and 4 are (Nerode) equivalent.

The Nerode equivalence is the coarsest partition that is compatible with the next-state function.

Refinement algorithm

Starts with the partition into two classes 05 and 12346.
Tries to refine by splitting classes which are not compatible with the next-state function.
A first refinement: $12346 \to 1234|6$ because $6 \cdot a$ is final.
A second refinement: $05 \to 0|5$ because of $0 \cdot a$ is final.

History of Hopcroft's algorithm

History

- Hopcroft has developed in 1970 a minimization algorithm that runs in time $O(n \log n)$ on an n state automaton (discarding the alphabet).
- No faster algorithm is known for general automata.

Question

- Question: is the time estimation sharp ?
- A first answer, by Berstel and Carton: there exist automata where you need $\Omega(n \log n)$ steps if you are "unlucky". These are related to De Bruijn words.
- A better answer, by Castiglione, Restivo and Sciortino: there exist automata where you need always $\Omega(n \log n)$ steps. These are related to Fibonacci words.
- The same holds for all Sturmian words whose directive sequence have bounded geometric means.

Splitter

$\mathcal{A} = (Q, i, F)$ automaton on the alphabet A. Let $\mathcal{P}$ be a partition of Q.

Definition

A splitter is a pair (P, a), with $P \in \mathcal{P}$ and $a \in A$.

The aim of a splitter is to split another class of $\mathcal{P}$.

Definition

The splitter (P, a) splits the class $R \in \mathcal{P}$ if

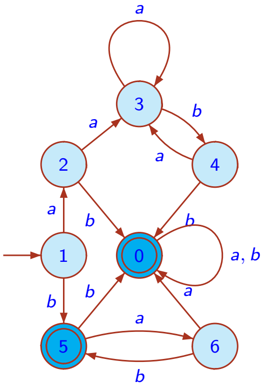
$$\emptyset \subsetneq P \cap R \cdot a \subsetneq R \cdot a \text{ or equivalently if } \emptyset \subsetneq a^{-1}P \cap R \subsetneq R\,.$$

Here $a^{-1}P = \{q \mid q \cdot a \in P\}$.

Notation

In any case, we denote by $(P, a)|R$ the partition of R composed of the nonempty sets among $a^{-1}P \cap R$ and $R \setminus a^{-1}P$. The splitter (P, a) splits R if $(P, a)|R \neq \{R\}$.

Example



- Partition $\mathcal{P} = 05 \mid 12346$.
- Splitter $(05, a)$. One has $a^{-1}05 = 06$.
- The splitter splits both 05 and 12346.
- One gets

$$(05, a)|05 = 0 \mid 5 \quad \text{and} \quad (05, a)|12346 = 1234 \mid 6$$

Notation

$\mathcal{P}$ is the current partition. $\mathcal{W}$ is the waiting set.

Hopcroft's algorithm

$\mathcal{P} \leftarrow \{F, F^c\}$ ▷ The initial partition
for all $a \in A$ **do**
    $\textsc{Add}((\min(F, F^c), a), \mathcal{W})$ ▷ The initial waiting set
while $\mathcal{W} \neq \emptyset$ **do**
    $(W, a) \leftarrow \textsc{TakeSome}(\mathcal{W})$ ▷ takes some splitter in $\mathcal{W}$ and remove it
    for each $P \in \mathcal{P}$ which is split by (W, a) **do**
        $P', P'' \leftarrow (W, a)|P$ ▷ Compute the split
        $\textsc{Replace}$ P by P' and P'' in $\mathcal{P}$ ▷ Refine the partition
        for all $b \in A$ **do** ▷ Update the waiting set
            if $(P, b) \in \mathcal{W}$ **then**
                $\textsc{Replace}$ (P, b) by (P', b) and (P'', b) in $\mathcal{W}$
            else
                $\textsc{Add}((\min(P', P''), b), \mathcal{W})$

Basic fact

Splitting all sets of the current partition by one splitter (C, a) has a total cost of $\mathrm{Card}(a^{-1}C)$.

Polyominoes

History

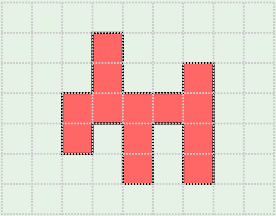
- Danièle Beauquier and Maurice Nivat have characterized those polyominoes that tile the plane by tranlation *On translating one polyomino to tile the plane* Discrete Math. 1991.
- The condition is a combinatorial property of circular words.
- The complexity of checking whether this condition holds is still open.
- In the particular case of socalled pseudo-squares, there exists a linear time algorithm, developed by Srečko Brlek, Xavier Provençal, Jean-Marc Fédou *On the tiling by translation problem*, Discrete Applied Math. 2009.

Exact polyominoes

Definition

A polyomino is a finite set of squares in the discrete plane which are simply 4-connected (without wholes).

Example

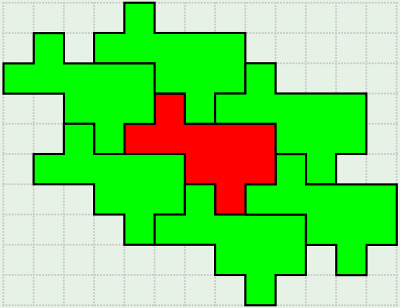


Exact polyominoes

Definition

A polyomino is exact if it tiles the plane by translation.

Example

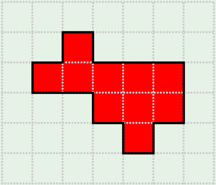


Boundary of a polyomino

Definition

The boundary of a polyomino is the circular word obtained by reading the the polygonal boundary in counterclockwise manner and encoding it over the alphabet $\{a, \bar{a}, b, \bar{b}\}$.

Example



The boundary is

$$aa\bar{b}a\bar{b}ababb\bar{a}\bar{a}\bar{a}b\bar{a}\bar{b}\bar{a}\bar{b}$$

Theorem

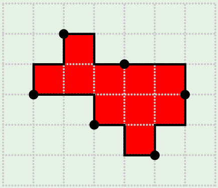
Notation

We denote by $\bar{}$ the mapping defined by $\overline{uv} = \bar{v}\,\bar{u}$ for words u, v.

Theorem (Beauquier, Nivat)

A polyomino tiles the plane by translation if and only if its boundary admits a factorization of the form $u\,v\,w\,\bar{u}\,\bar{v}\,\bar{w}$ for some words u, v, w.

Example



The boundary admits the factorization

$$aa\bar{b} \cdot a\bar{b}a \cdot bab \cdot b\bar{a}\bar{a} \cdot \bar{a}b\bar{a} \cdot \bar{b}\bar{a}\bar{b}$$

Searching for aBN-factorization

A naive algorithm

Given a word w of length n, do for each of the n conjugates of w

- consider all n^2 factorizations $xyzstu$ with $|x| = |s|, |y| = |t|, |z| = |u|$.
- check whether $x = \bar{s}, y = \bar{t}, z = \bar{u}$.

Each positive answer gives a BN-factorization. The complexity is $O(n^4)$.

An algorithm in $O(n^2)$ has been given by Gambini and Vuillon *An algorithm for deciding if a polyomino tiles the plane by translation*2007.

Pseudo-square

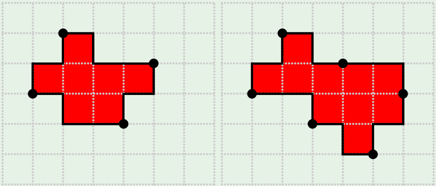
Definition

A pseudo-square is a boundary that has a factorization of the form $xy\bar{x}\bar{y}$ for nonempty words x, y.

Note

A pseudo-polygon is a boundary with a factorization $xyz\bar{x}\bar{y}\bar{z}$ for nonempty words x, y, z.

Example (Pseudo-square and pseudo-polygon)



The first is a pseudo-square, and the second is a pseudo-polygon. BN-factorizations are

$$a\bar{b}aa \cdot bab \cdot \bar{a}\bar{a}b\bar{a} \cdot \bar{b}\bar{a}\bar{b} \quad \text{and} \quad aa\bar{b} \cdot a\bar{b}a \cdot bab \cdot b\bar{a}\bar{a} \cdot \bar{a}b\bar{a} \cdot \bar{b}\bar{a}\bar{b}$$

An algorithm for pseudo-square detection

A linear algorithm

An algorithm for pseudo-square detection that is linear in the length of the boundary has been given by Brlek, Provençal and Fédou.
It uses in a clever way a preprocessing phase that allows to compute in contant time the longest common extension of two words.

Notation

$\rho^i(x)$ is the conjugate of x starting at position i ($\rho^0(x) = x$).

Example

For $x = aabbbaab$, one has $\rho^4(x) = baabaabb$.

Definition (Longest common right and left extension)

The longest common right (left) extension of x at position i and y at position j is the word $lcre(x,i,y,j) = \rho^i(x) \wedge \rho^j(y)$ (resp. $lcle(x,i,y,j) = \rho^i(x) \vee \rho^j(y)$). Here $u \wedge v$ (resp. $u \vee v$) is the longest common prefix (suffix) of u and v.

Example

For $x = aabb \cdot baab$ and $y = babaabb \cdot baabb$, one has

$$lcre(x,4,y,7) = baabaabb \wedge baabbbabaabb = baab$$

and

$$lcle(x,4,y,7) = baabaabb \vee baabbbabaabb = abaabb$$

Definition (Longest common extension)

The longest common extension of x at position i and y at position j is the word

$$lcle(x,i,y,j)lcre(x,i,y,j).$$

Example

For $x = aabb \cdot baab$ and $y = babaabb \cdot baabb$, one has

$$lce(x,4,y,7) = abaabbbaab$$

BN-factorzation

Algorithm

Let w be a boundary of length n. For each $j = 0, \ldots, n-1$

- Compute $x = lce(w, 0, \bar{w}, j)$.
- Locate $\bar{x}$ in w and, if x and $\bar{x}$ do not overlap, factorize w into $w = xy\bar{x}z$.
- check whether $y = \bar{z}$ by checking whether $\mathrm{lcre}(w, k, \bar{w}, 0) = y$, with $k = |x|$.

If the answer is positive, a pseudo-square factorization has been found.

Example

$w = aa\bar{b}aabaab\bar{a}\bar{a}b\bar{a}\bar{a}\bar{b}\bar{a}\bar{a}\bar{b} = aa\bar{b}aabaab\bar{a}\bar{a}b\bar{a}\bar{a}\bar{b}\bar{a}\bar{a}\bar{b} = aa\bar{b}aabaab\bar{a}\bar{a}b\bar{a}\bar{a}\bar{b}\bar{a}\bar{a}\bar{b}$
$\bar{w} = baabaa\bar{b}aa\bar{b}\bar{a}\bar{a}\bar{b}\bar{a}\bar{a}b\bar{a}\bar{a} = baabaa\bar{b}aa\bar{b}\bar{a}\bar{a}\bar{b}\bar{a}\bar{a}b\bar{a}\bar{a} = baabaa\bar{b}aa\bar{b}\bar{a}\bar{a}\bar{b}\bar{a}\bar{a}b\bar{a}\bar{a}$

$lce(w, 0, \bar{w}, 1) = aa$ and $w = aa\bar{b}aabaab\bar{a}\bar{a}b\bar{a}\bar{a}\bar{b}\bar{a}\bar{a}\bar{b}$ bad.
$lce(w, 0, \bar{w}, 4) = aa\bar{b}aa$ and $w = aa\bar{b}aabaab\bar{a}\bar{a}b\bar{a}\bar{a}\bar{b}\bar{a}\bar{a}\bar{b}$ good!.
$lce(w, 0, \bar{w}, 7) = \bar{b}aa\bar{b}$ and $w = aa\bar{b}aabaab\bar{a}\bar{a}b\bar{a}\bar{a}\bar{b}\bar{a}\bar{a}\bar{b}$ good!.

Remark

Since the computation of the lce is in constant time, the algorithm is linear.