Automata and Pattern Matching in Planar Directed Acyclic Graphs

F. BOSSUT and B. WARIN
Laboratoire d'Informatique Fondamentale de Lille , U.A. C.N.R.S. 369,
University of Lille Flandres Artois,
59655 Villeneuve d'Ascq Cedex, FRANCE.

Abstract. We extend the algorithm of Knuth, Morris and Pratt of pattern matching on strings to a class of planar acyclic graphs and we analyse the relationship between this algorithm and deterministic finite automaton of graphs.

Introduction

The problem of locating the occurrences of a pattern string or tree in a (presumably) larger text is of interest in several areas of computer science of interpreters for non procedural programming languages, code optimization in compilers, symbolic computation, context searching in structure editors. Pattern matching on strings became very well known after the discovery of linear time algorithm, the Knuth Morris and Pratt (KMP for short) algorithm [11], for this problem. Tree Pattern matching is analogous to the problem of pattern matching in strings, and the KMP algorithm can be generalized with some variations to bottom-up tree pattern matching. Hoffmann and O'Donnell work [8] about bottom-up tree pattern matching has stimulated papers offering improvements ([5, 6]). The KMP algorithm can be seen as computation of deterministic automaton. In fact, the preprocessed failure table enables to construct a deterministic finite automaton for a regular expression of the form any*(x_1 +x_2+ ... +x_n)any* in time linearly proportional to the sum of the lengths of the x_i's where "*any*" stands for any symbol. Our goal is to extend the KMP algorithm to pdag (planar directed ordered acyclic graphs) pattern matching where patterns are particular pdags that we call r-dags (rooted pdags) and to analyse the relationship between this algorithm and deterministic finite automaton of pdags.

This paper is organized as follows: in section 1, we define the algebraic frame in which we define our pdags. In section 2, we propose an algorithm of pattern matching. In section 3, we analyse our algorithm in term of deterministic pdags automaton.

1 - Preliminaries

We extend the notion of d-dags introduced by Kamimura and Slutzki [10].

1.1. Definitions of pdags. A *doubly-ranked alphabet* Σ is a finite set of letters on which are defined two mappings into N called *head-rank* and *tail-rank.* For n, m integer, we denote by: Σ_m the set of letters of tail-rank m; ${}_n\Sigma$ the set of letters of head-rank n and ${}_n\Sigma_m = {}_n\Sigma \cap \Sigma_m$. We interpret a letter of ${}_n\Sigma_m$ as a labelled node of a graph that has n ordered inputs and m ordered outputs.

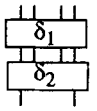
We define $M(\Sigma)$, the set of pdags over Σ, as $\bigcup_{m,n \in N} {}_nM(\Sigma)_m$ where the sets ${}_nM(\Sigma)_m$ of pdags over Σ with

n inputs and m outputs are recursively defined by:

(i). if $a \in {}_n\Sigma_m$ then $a \in {}_nM(\Sigma)_m$.

(ii). if $\delta_i \in {}_{pi}M(\Sigma)_{qi}$ for i=1, 2 then the *parallel composition* of δ_1 and δ_2, denoted by $\delta_1 \theta \delta_2$, belongs to ${}_{p1+p2}M(\Sigma)_{q1+q2}$.

(iii). if $\delta_i \in {}_{pi}M(\Sigma)_{qi}$ for i=1, 2 and q1=p2 then the *serial composition* of δ_1 and δ_2, denoted by $\delta_1.\delta_2$, belongs to ${}_{p1}M(\Sigma)_{q2}$. The drawing of figure 1 represents the result of this composition.



the *i*th output of δ_1 is connected to the *i*th input of δ_2

Figure 1

And these operators satisfy the following axiomatic equalities:

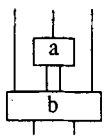
(iv). A particular pdag of ${}_1M(\Sigma)_1$, denoted by ε, is the unit element of the serial composition in the following sense:

For any integer p, let ε_p be $\varepsilon \theta \varepsilon \theta ... \theta \varepsilon$ (p times), then for $\delta \in {}_pM(\Sigma)_q$, $\varepsilon_p.\delta=\delta.\varepsilon_q=\delta$.

ε_0 denotes the absence of ε.

(v). The serial and parallel compositions are assosiative operators, and if a.a' and b.b' are defined then $(a.a')\theta(b.b') = (a \theta b).(a' \theta b')$.

1.2. Example. For $a \in {}_1\Sigma_2$ and $b \in {}_4\Sigma_2$, the pdag obtained by the serial composition of the parallel composition of ε, a and ε with b is denoted by $(\varepsilon \theta a \theta \varepsilon).b$ and can be represented by the graph of figure 2.



The direction of the arc is top-down .
The arcs without source or target specify head-rank or tail-rank of the node

Figure 2

The underlying algebraic structure for M(Σ) is the *free magmoid generated by* Σ. More details about this structure can be found in Arnold and Dauchet [1, 2], Schnorr [14], Hotz [9], Bossut and Warin [3]. Σ* denotes the free monoïd generated by the alphabet Σ and the operation θ. We said that δ' is a *subgraph* of a pdag δ if for some pdags δ_1, δ_2, δ_3, δ_4 we have $\delta = \delta_1.(\delta_2 \theta \delta' \theta \delta_3).\delta_4$. For δ' *subgraph* of a pdag δ, we shall say that: δ' is a *proper subgraph* of δ if δ' is not equal to δ and δ' is *an initial subgraph* of δ if δ_1 belongs to ε*. We say that a arc is *an input of a graph T* if there is no predecessor

2 - Pattern Matching in Pdags

2.1. Definitions. A pdag d is a *rooted pdag* or simpler a *r-dag*, if d is ε, or there is a node r of d such that, there exists a directed path from every node (except r) of d to the node r. The node r is called the *root* of the r-dag d. For a pdag g, and a node n of g, we define the *r-dag of root n in g*, as the r-dag of root *n*, sub-

graph of *g*, that contains all the nodes of g for which there exist a path to n. See for example figure 3. For a r-dag D, a r-dag d is a prefix of D, if d is ε or if for some node n of D, d is the r-dag of root n in D. For a r-dag D, we denote by Prefix(D) = { d / d is prefix of D }. For a r-dag D of root n, a r-dag d is a suffix of D, if d is ε or if d is a subgraph of D and the root of d is n too. For a r-dag D, we denote by Suffix(D) = { d / d is suffix of D }. If d is a suffix of D, we say that $d \leq D$.

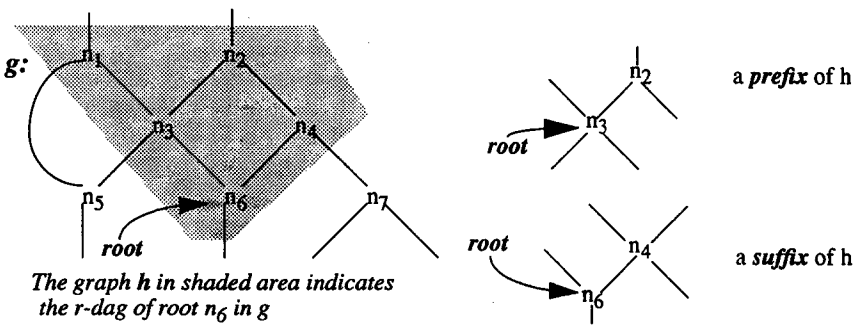


Figure 3.

Proposition. The relation "≤" (*to be a suffix of*) over the r-dag is a partial order.

2.2. Superposition.

For D a finite set of r-dags, we denote by *Sup(D)* the set {δ r-dag / ∀ d ∈ D, d ≤ δ }

Proposition. For a finite set D of r-dag, if Sup(D) is not empty then Sup(D) has a smallest member that we call *Superposition(D)*

Example

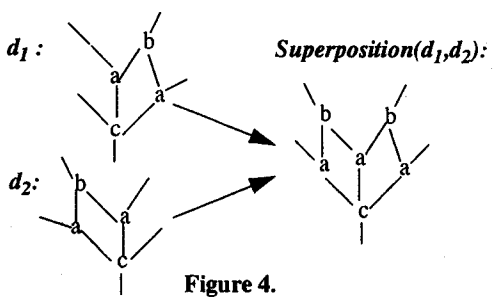


Figure 4.

The superposition of two r-dags d_1 and d_2 is a r-dag d which spanning tree is the first-order unification of the spanning trees of d_1 and d_2. So from the first-order unification algorithm in trees[12, 13], we can deduce an algorithm to construct the superposition of two r-dags [4]. If d_1 and d_2, subgraphs of some pdag, are r-dags with the same root, then Superposition(d_1, d_2) exists always and is simply the union of the edges and nodes of d_1 and d_2.

2.3.The Top-down Pattern Matching Algorithm. For searching an occurrence of a r-dag d in a pdag T, at the beginning , we associate with each input of T the empty r-dag ε, then we go through the pdag T from top to bottom and each node transmits to its succesors, the r-dag Superposition(prefix(d) ∩ suffix(δ))

where δ is defined as follows:

Let be a_i the node which is directly linked to the i^{th} input of the node r and δ'_i the r-dag transmitted to its sons by a_i. The r-dags δ'_i are not generally separated r-dags as they are in the case of trees. Then δ is the ***greatest*** suffix of the r-dag of root r in T such that

if $\delta'_i \neq \varepsilon$, the r-dag of root a_i in δ is a suffix of δ'_i and

otherwise, in δ, the i^{th} input of r has no predecessor.

In some cases, the r-dag linked to i^{th} input of the node r in δ is a proper suffix of δ'_i. This kind of situation arises when too much information has been brought by a node, comparatively to those brought by the others. Figure 5 illustrate how we determine the r-dag δ for a node r

If the two r-dags δ_1 and δ_2 are transmitted to the inputs of the node r of the pdag below

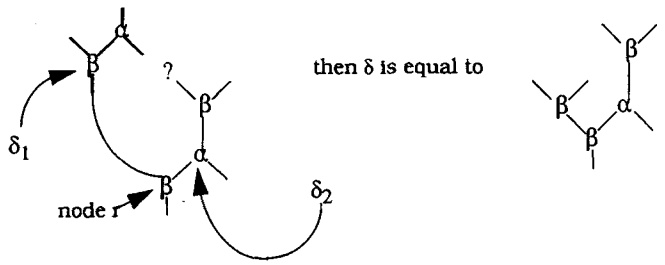
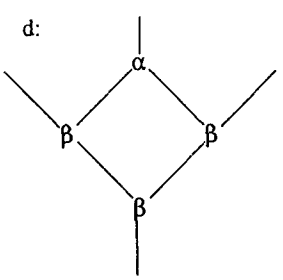


Figure 5.

When d belongs to Suffix(δ), an occurrence of d has been located. If Superposition(Prefix(d)$\cap$Suffix(δ)) is empty, then r transmits ε to its sons.

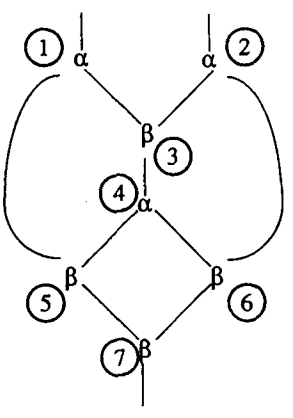
2.3.1. A developped example. Assume the alphabet Σ is $\{\alpha,\beta\}$, where α is a symbol of arity 2 and co-arity 1 and β a symbol of arity 1 and co-arity 2. Consider a matching problem in which the pattern is

d:



d=(ε θ α θ ε).(β θ β).β in our formalism.

and the pdag in which we wish to locate the occurrences of d:



For each node of T, we get in the following table, the pdag δ, the set Suffix(δ) ∩ Prefix(d) and the r-dag δ' that this node transmits to its sons.

node	δ	Prefix(d)∩Suffix(δ)	δ'
1	α	{α}	α
2	α	{α}	α
3	(αθα).(εθβθε)	{(αθε).(εθβ), (εθα).(βθε) }	(αθα).(εθβθε)
4	(αθα).(εθβθε).(εθαθε)	{α}	α
5	(αθα). (εθβθε)	{(εθα).(βθε) }	(εθα).(βθε)
6	(αθα). (εθβθε)	{(αθε).(εθβ)}	(αθε).(εθβ)
7	(εθαθε).(βθβ).β = d		

So, we can write the pattern matching algorithm.

2.3.2. Algorithm. We assume that

For a node n of T, we call *n.input[i]* the i^{th} input of n, and *n.output[j]* the j^{th} output of n. We say that *(n.output[i], n'.input[j]) is an edge*, if there is an edge betwen the i^{th} output of the node n and the j^{th} input of the node n'.

We say that n.input[i] is *an input of the graph T* if there is no node n' such that (n'.output[j], n.input[i]) is an edge of T,
The nodes of the pdag T can store the r-dags transmitted by its fathers. For a node n,we denote by *n.message[i]* the near-tree transmitted to its i^{th} input. These memory areas are initialized with ε .

For the pdag T, the list of its inputs is supplied.

We use a pushdown store (first-in- last-out list) of nodes. When a node n is entered at the top, we denote this action by *push(n)*; similary, when a node n is removed from the top, we denote this by *pop(n)*.

So we define the procedure *Trans(n:node)* as follows:

```
Procedure Trans(n: node);
begin
   From n.message[i], determine δ
   If d is a suffix of δ then
      location of an occurrence of d on node n
   endif
   δ' = Superposition(Prefix(d)∩Suffix(δ))
   For each output of n begin
         n'.message[j] = δ' for (n.output[i], n'.input[j]) edge of T
         if n' has receveid the messages sended by all of its fathers then
          push(n') endif
   end for
end procedure Trans
```

and then the main program:

```
Program Pattern_Matching ;
   /* initialisation */
   For each n.input[i] input of the pdag T
         n.message[i] = ε
         if all of the inputs of n are also inputs of T then push(n) endif
   end for
   while the pushdown store of nodes is not empty do pop(n); Trans(n); end do
end program Pattern_Matching
```

*2.3.3. **Extension.*** We can extend the pattern matching algorithm for one pattern to a finite set of patterns. In this case δ' is computed as follow:

$$\text{Superposition}\left(\bigcup_{(d_i \in D)} \text{Prefix}\,(d_i) \cap \text{Suffix}\,(\delta)\right)$$

*2.3.4. **Optimization.*** We can only consider proper prefixes of the d_i . We can also consider for a pdag δ which root is labelled by a letter x, the prefix of the d_i 's whose roots are labeled by the same letter. We can even preprocess for a finite set of patterns all the configurations $< \delta,\delta' >$ where δ can be decomposed in the r-dags δ_i linked to each input of its root, and the label x of its root, note that if we dislocate δ in this way, we must precise, for the roots of the δ_i's of co-arity greater than one, which output is connected with the root of δ; so for a label x of co-arity n (n>0), we denote by x_i ($1 \leq i \leq n$) the i th output of a node labeled by x. These informations can be collected in a table as it can be done in the KMP algorithm (failure table) and in bottom-up tree pattern matching. The rows of this table can be easily transposed into transitions of a pdag automata because the set of the δ_i's is finite and the set of labels is finite too.

3. Relation between pattern matching and pdag automaton

3.1. Pdag automaton. A *top-down pdag automaton* is a construct M=<Σ,Q,I,F,R> where

- Σ is a doubly-ranked alphabet
- Q is a finite set of *states*
- I is the *initial control*, F the *final control*, I and F are recognizable languages of Q*
- R is a finite set of rules of the form $\alpha \rightarrow \beta$, α and β are respectively the left-hand side and the right hand-side of the rule $\alpha \rightarrow \beta$.

More precisely, the rules are of the form

$$(p_1 \theta\, p_2 \theta \ldots \theta p_n).\sigma \rightarrow \sigma.\,(q_1\, \theta\, q_2\, \theta \ldots \theta\, q_p) \text{ with } \sigma \in \Sigma_{n,p}$$

and $p_1, p_2, \ldots, p_n, q_1, q_2, \ldots, q_p \in Q$

The case bottom-up is analogous. An automaton M=<Σ,Q,I,F,R> is *deterministic* if two different rules have different left-hand sides and if I does not contain two strings of the same length. Let us remark that there exists a decision procedure to know if two strings of the same length belong to a recognizable string language. A *configuration* of M is a pdag of the form d.<q>.d' where d and d' are pdags over Σ and <q>

denotes a parallel composition of states, each state being identified to a doubly-ranked letter of head-rank and tail-rank equal to 1. A configuration is *initial* if it is of the form <q>.d with <q> ∈ I; likewise a configuration is *final* if it is of the form d.<q> where <q> ∈ F (here <q> is identified to a string). We define the relation -M→ between configurations of M as follows: C-M→C' if C' is obtained from C by replacing the left-hand side of some rule of R by its right hand side. -M^*→ is the reflexive-transitive closure of -M→ . A pdag d is *accepted* by M is <q>.d -M^*→ d.<q'> for some <q> ∈ I and <q'> ∈ F. $L(M)$ denotes the language of pdags accepted by M. A language L is *automaton-definable* if there is some automaton M such L = L(M).
Contrary to the case of string automata or bottom-up tree automata, we could not reduce the undeterminism in the case of top-down (or bottom-up) pdags automata [3] and the closure under complementation of the class of automaton-definable pdag languages remains opened. Nevertheless, for a deterministic automaton-definable language L, we can construct the automaton that accepts the complement of L by analogous technics as these ones used in the cases of strings or trees.

We denote by AD_D the class of the deterministic automaton-definable language of M(Σ). For a r-dag δ, we denote by Δ(δ) the set of pdags which admit δ as subgraph. And we denote by Δ the class of *locally testable sets* defined as the boolean closure of the sets Δ(δ) for δ r-dag of M(Σ) i.e

∀ δ r-dag of M(Σ), Δ(δ) ∈ Δ
∀ υ,σ ∈ Δ,υ ∩σ ∈ Δ,υ ∪σ ∈ Δ
∀υ ∈ Δ, the complement of υ belongs to Δ.

We are going to state that Δ is a class of deterministic automaton-definable languages. First of all, let us remind that the top-down pattern matching algorithm suggest to us a method to construct a deterministic automaton that accepts the pdags that admit the given patterns as subgraphs

.

Lemma 3.2. For any finite set D={d_1,d_2,...,d_n} of r-dags, $\bigcup_{\delta \in D} \Delta(\delta) \in AD_D$

Proof.
Let be L = {l_1, l_2, ..., l_n} a set of symbols associated one-to-one with the r-dags of D.
Let be M <Σ, Q, I, F, R> where

- Q = {δ that could appear in the pattern matching algorithm for the set D of patterns } × { ω ∈ L^j / j ≤ n }
- I = (ε,ε)*
- F = {w ∈Q* / π(w) ≠ ε } where π is a morphism from Q* into L* such that π((δ,ω))=ω.
- R = { (δ_1,w_1)θ(δ_2,w_2)θ...θ(δ_n,w_n). x → x. (δ',w)θ...θ(δ',w) such that:
  .δ' is the r-dag resulting from the meeting of the δ_i 's on a node labeled by x in the pattern matching algorithm
  .w=$h_{(w1w2...wn,\delta')}$(l_1l_2, ...l_n) where $h_{(u,\delta')}$ is a morphism defined as follow:
  ∀1≤i≤n $h_{(u,\delta')}(l_i) = l_i$ if u ∈ L*l_iL* or d_i is a suffix of δ'

This automaton is deterministic, completly specified and such that δ∈L(M) if and only if at least one of the r-dag of D is a subgraph of δ.

Theorem 3.3. $\Delta \subset AD_D$

Proof. Let be ζ a member of Δ, then ζ can be expressed as a formula composed of constant terms Δ(d_1), Δ(d_2),..., Δ(d_n) and operators of complement, union and intersection;
let us denote by ℘(Δ(d_1), Δ(d_2), ..., Δ(d_n)) this formula.
Let be D={ d_1, d_2, ..., d_n} and M=<Σ, Q, I, F', R> be the pdag automaton defined as the one of lemma 3.2 except the final control F' defined as follow:

$$F' = \{ w \in Q^* / \pi(w) \in \wp(L^*l_1L^*, L^*l_2L^*, ..., L^*l_nL^*) \}$$

where π is a morphism from Q* into L* such that π((δ,ω))=ω.

F' is a recognizable string language. M is a deterministic, completly specified and as the second component of the states keep traces of the met r-dags of D while the computation, we have obvioulsy $\delta \in L(M)$ if and only if $\delta \in \wp(\Delta(d_1), \Delta(d_2), ..., \Delta(d_n))$ ♦

We obtain as an obvious extension of the previous result that the class of *locally threshold testable sets of pdags*, introduced by Thomas [15], is also included in the class of deterministic automaton-definable languages. The languages of this class are defined as boolean combination of sentences of the form:

$\varphi_{\delta,n}$: "there are $\geq$ n occurrences of δ (r-dag) as a subgraph ".

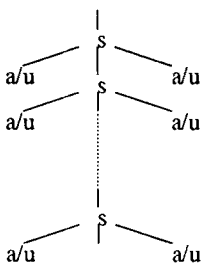
Remark. This result could be used in study of pdags rewriting systems, for example, if the rules are of the form $\delta \rightarrow \sigma$ where δ are r-dags then the set of irreducible pdags belongs to AD_D.
As for any pdag δ, $\Delta(\delta)$ is obviously automaton-definable, it may be asked whether the previous result can be can be extended to a larger class of pdags than the class of r-dags. We will answer no this question and exhibit a counter-example, that is to say a pdag δ for which there is no deterministic top-down pdag automaton M such that $\Delta(\delta) = L(M)$.

Proposition 3.4. For some pdag δ, $\Delta(\delta) \notin AD_D$.
Proof. let Σ be composed of a,u ($\in \Sigma_{1,1}$) and s ($\in \Sigma_{1,3}$). The elements of $M(\Sigma)$ are trees. Let us consider the pattern $d=s.(a\theta\epsilon\theta u)$ and suppose that there is a deterministic pdag automaton M such that $\Delta(\delta) = L(M)$.
Let be $M=<\Sigma, Q, q_0, F, R>$

and $\{ q_i . s \rightarrow s. (g_i \theta k_i \theta d_i) \in R\}$ $\{ g_i . a \rightarrow a. \acute{a}_i \in R\}$, $\{d_i . a \rightarrow a. \grave{a}_i \in R\}$

$\{ g_i. u \rightarrow u. \acute{u}_i \in R\}$, $\{ d_i. u \rightarrow u. \grave{u}_i \in R\}$

Then, for any pdag δ of the form exposed in the following figure



$<q_0> \delta$ -M*$\rightarrow \delta. <m>$ where $m = m_1 k m_2$ with $m_1 \in (\sum \acute{a}_i + \sum \acute{u}_i)^*$, $k \in \sum k_i$, $m_2 \in (\sum \grave{a}_i + \sum \grave{u}_i)^*$ and $length(m_1) = length(m_2)$.
So δ belongs to $\Delta(\delta)$ if and only if there is an j such that the j^{th} letter of m_1 is $\acute{a}_i$ and the letter at position $length(m_2)-j+1$ in m_2 is $\grave{u}_i$. This kind of constraint can not be checked by a string automaton, so F can not be recognizable.

The previous proposition also shows that even if we consider pdags automaton on trees likewise top-down tree automaton with recognizable final control, there is rational forest that can not be accepted by deterministic top-down pdag automaton. Nevertheless, we can state that deterministic pdag automaton considered as tree automaton are more powerfull than deterministic top-down tree automaton.
Let us denote by *Rec* the set of rational forest, Rec_D the set of rational forests accepted by deterministic top-down tree automaton and AD_D the languages of trees accepted by deterministic top-down pdag automaton, so we have :

Proposition 3.5. $Rec_D \subset AD_D \subset Rec$ and $Rec_D \neq AD_D$ and $AD_D \neq Rec$

Proof.

-$Rec_D \subset AD_D$: obvious

-$AD_D \subset Rec$ is a consequence of a theorem about pdag automaton. This theorem says that , for any pdag automaton, there is an equivalent final control-free pdag automaton (see Bossut, Dauchet, Warin [3]) and proposition 3.4 implies that $AD_D \neq Rec$.

- $Rec_D \neq AD_D$:

Let T be the following tree automaton $T = <\Sigma, Q, I, F, R>$ where

- $\Sigma = \Sigma_{1,2} = \{a,b\}$ - $Q = \{q_1 , q_0, q_f\}$ - $I = Q$ and $F = \{q_f\}$

- $R = \{ q_0.a \rightarrow a.(q_0,q_0),\ q_1.b \rightarrow b.(q_0,q_0),\ q_0.a \rightarrow a.(q_1,q_1),\ q_1.b \rightarrow b.(q_1,q_1)$
$q_0.a \rightarrow a.(q_f,q_f),\ q_1.b \rightarrow b.(q_f,q_f)\ \}$

The trees accepted by this automaton are such if a node has successors then it has two direct successors that have the same label. This automaton is not deterministic and it is obvious that this forest F can not be recognized by any deterministic top-down tree automaton, but there is a deterministic top-down pdag automaton M such that L(M)=F. The states of this automaton carry informations about the covered path and the met letters when the state is produced. These states are denoted :

q_l : reached after going down from the root along right hand-side branches;

q_r : reached after going down from the root along left hand-side branches;

$q_($: reached after going down from a node labeled by "a" (except the root) which is the left hand-side son of its father along right hand-side branches;

$q_)$: reached after going down from a node labeled by "a" (except the root) which is the right hand-side son of its father along left hand-side branches;

$q_[$: reachs after going down from a node labeled by "b" (except the root) which is the left hand-side son of its father along right hand-side branches;

$q_]$: reachs after going down from a node labeled by "b" (except the root) which is the right hand-side son of its father along left hand-side branches.

So the automaton M is defined as follow:

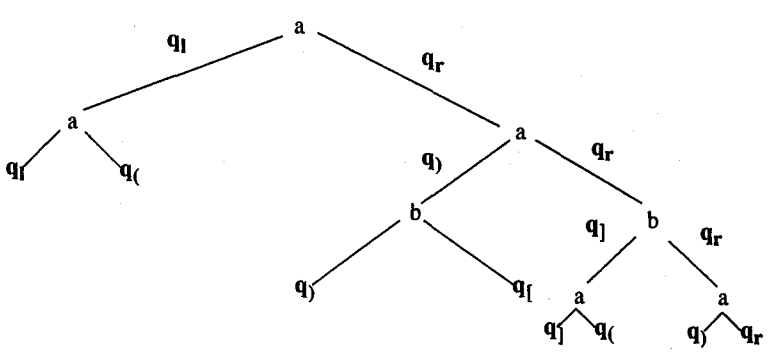
$T = <\Sigma, Q, I, F, R>$ where

- $Q = \{q, q_l, q_r, q_(, q_), q_[, q_]\}$ - $I = \{ q \}$ and $F = q_l.(q_(q_)+ q_[q_])^*.q_r$

- $R = \{ q.a \rightarrow a.(q_l, q_r),\ q.b \rightarrow b.(q_l, q_r),\quad q_l.a \rightarrow a.(q_l, q_(),\ q_l.b \rightarrow b.(q_l, q_[);$
$q_r.a \rightarrow a.(q_), q_r),\ q_r.b \rightarrow b.(q_], q_r)\quad q_(.a \rightarrow a.(q_), q_(),\ q_(.b \rightarrow b.(q_], q_()$
$q_[.a \rightarrow a.(q_), q_[),\ q_[.b \rightarrow b.(q_], q_[)\quad q_).a \rightarrow a.(q_), q_(),\ q_).b \rightarrow b.(q_), q_[)$
$q_].a \rightarrow a.(q_], q_(),\ q_].b \rightarrow b.(q_l, q_[)\ \}$

This automaton is deterministic and completly specified.

Example of computation:



Conclusion. We have shown that pattern matching technics applied in strings and trees can be extented to a wider class of objects, especially to pdags when patterns are r-dags. In this particular case, we can establish a connection between the pattern matching algorithm and deterministic automaton that recognizes the language of pdags that contain one of the considered patterns. This algorithm must be now implemented and its time and space complexity evaluated. In first approximation, we can give an upper bound of its time complexity:

let us denote by

patnode : the number of nodes of the pattern
patedge : the number of edges of the pattern
subnode : the number of nodes of the subject pdag
subedge : the number of edges of the subject pdag

The superposition, the comparison of two r-dags, the check to determine if a r-dag is a suffix of an another is essentially a parallel exploration of the two r-dags and then these processing require $O(n+n'+e+e')$ steps where n, n' denote the numbers of nodes of the two r-dags and e, e' their numbers of edges.

A r-dag of n nodes has exactly n prefixes.

The r-dag δ is a subgraph of the subject pdag which is directly deduced from the δ'_i. The maximum size of δ is approximately the size of d.

Then the computation of $prefix(d) \cap suffix(\delta)$ requires $O(patnode \times (patnode+patedge))$ steps.

The set $prefix(d) \cap suffix(\delta)$ contains at most *patnode* r-dags, then the computation of $Superposition(prefix(d) \cap suffix(\delta))$ requires $O(patnode \times (patnode+patedge))$ steps.

So, as the patttern matching algorithm is in fact an exploration of the subject pdag during which, for each node, we compute $Superposition(prefix(d) \cap suffix(\delta))$, the matching requires at most $O(subedge+subnode \times patnode \times (patnode+patedge))$ steps. Following this work, efficient preprocessing algorithm should be designed.

Another interesting question asks for effective procedures which decide if an automaton-definable set of pdags is or not locally (threshold) testable. This work could be a starting point to study the relation between first-order properties of pdags and other characterizations as Heuter [7] do it for finite trees.

References

1. ARNOLD A. and DAUCHET M. (1978), "Théorie des magmoides", RAIRO IT 12-3, 235-257.

2. ARNOLD A. and DAUCHET M. (1979), "Théorie des magmoides", RAIRO IT 13-2, 135-154.

3. BOSSUT F., DAUCHET M. and WARIN B. (1988), "Automata and rational expressions on planar graphs", Mathematical Foundations of Computer Science, Carlsbad, august 1988, Lecture Notes Comp. Science, 324, 190-200.

4. BOSSUT F. and WARIN B. (1991), "Automata and pattern matching in planar directed acyclic graphs", intern. publication IT 206 of LIFL, University of Lille 1, 59655 Villeneuve d'Ascq, FRANCE.

5. CAI J., PAIGE R. and TARJAN R. (1990), "More efficient bottom-up Tree Pattern Matching", in Proc. CAAP 90, LNCS 431.

6. CHASE D. (1987), "An improvement to bottom-up pattern matching", in Proc. Fourteenth Annual ACM Symposium on principles of programming Languages, pp 168-177.

7. HEUTER U. (1991) "First-order properties of trees, star-free expressions, and aperiodicity", Informatique théorique et Applications, vol.25,n 2, p 125-145.

8. HOFFMANN C.M. and O'DONNELL M. J.(1982),"Pattern Matching in Trees", JACM Vol 29, 68-95.

9. HOTZ G. (1965), "Eine Algegraisierung des Syntheseproblems von Schaltkreisen", EIK 1, 185-205, 209-231.

10. KAMIMURA T. and SLUTZKI G. (1981), "Parallel and Two-Way Automata on Directed Ordered Acyclic Graphs", Information and Control 49, 10-51.

11. KNUTH D. E., MORRIS J. H., and PRATT V. R. (1977), "Fast Pattern Matching in Strings", SIAM J. Computing 6:2 (1977) 323-350.

12. MARTELLI and MONTANARI (1982), "An efficient unification algorithm", Transactions on programming languages and syst., 256-282.

13. ROBINSON J.A. (1965), "A machine-oriented logic based on the resolution principle", JACM 12, 32-41.

14. SCHNORR C.P. (1969), "Transformational classes of grammar", Information and control 14, 252-277.

15. THOMAS W.(1991), "On logics, Tilings, and Automata", Bericht Nr.9108, Christian-Albrechts Universitat.