Using Tree Automata for XML Mining and Web Mining with Constraints

Sandra de Amo, Cricia Zilda Felicio

Abstract

Most work on pattern mining focus on simple data structures like itemsets or sequences of itemsets. However, a lot of recent applications dealing with complex data like chemical compounds, protein structure, XML and Web Log databases and social network, require much more sophisticated data structures (trees or graphs) for their specification. Here, interesting patterns involve not only frequent object values (labels) appearing in the graphs (or trees) but also frequent specific topologies found in these structures. In a recent work, we have introduced the method CobMiner for tree pattern mining with constraints. CobMiner uses tree automata as a mechanism to specify user constraints over tree patterns. In this paper, we focus on applications of this method in two different contexts: XML Mining and Web Usage Mining. An extensive set of experiments executed over real data in these two fields allow us to conclude that CoBMiner is an efficient method for mining datasets whose elements are specified as trees.

1 Introduction and Motivation

Recently there has been an increasing interest in developing techniques for the discovery of tree patterns in a dataset of trees (Zaki 2002; Miyahara et al. 2001). Different kind of real data can be modeled using tree structures, such as XML documents, Web usage logs, RNA structures in Bioinformatics, etc. The growing interest in developing methods for tree pattern mining can be justified by its potential applicability in those different contexts. However, the amount of discovered patterns involved in the tree pattern task is enormous, due to the highly combinatorial nature of the tree pattern search space. Thus, very often, tree pattern mining is a very expensive process. In recent years, a lot of work has been dedicated to the development of methods allowing users to control the patterns produced by the mining systems. This control can take place in a post-processing phase or during the mining process, by incorporating user-specified constraints. This later approach has been largely exploited in the context of association rule mining (Padmanabhan and Tuzhilin 2000) and sequential pattern mining (Garofalakis et al. 1999; de Amo and Furtado 2005). In (de Amo et al. 2007), we introduced the algorithm CobMiner (Constraint-Based Miner) for constraint-based tree pattern mining allowing users to specify the format of the tree patterns they are most interested in. These user-specified constraints are pushed inside the mining process in such a way that only patterns satisfying them are produced. The mechanism we use for specifying user constraints is Tree Automata. A short survey on the use of tree automata in XML research can be found in (Neven 2002). Briefly, a non deterministic tree automata (NTA) is a structure $B = (Q, q_0, L, \delta)$, where $Q$ is a finite set of states, $q_0$ is the initial state, $L$ is an alphabet of labels and $\delta$ is a mapping $\delta : Q \times L \rightarrow 2^Q$ associating to each pair $(q, a) \in Q \times L$ a regular language over the set $Q$ (that is, a set of strings of states satisfying a regular expression). While (simple) finite automata are designed to accept words (sequences), tree automata are designed to accept trees labeled over the alphabet $L$. Then, tree automata can naturally be thought as a mechanism to characterize a specific set of trees, that is, to specify a constraint over trees. The following example illustrates our approach:
Example 1 Let us suppose an e-commerce site where users navigate through items and eventually make a purchase. Let us suppose that items are classified into two categories: Computers (C) and Printers (Pr). Computers are classified into Notebooks (Nb) and Desktops (Dt). Let us suppose we are interested in discovering specific navigation patterns conforming the following format: the client accesses the site, browses different notebooks brands, buys at least one notebook, besides browsing the printer page, without necessarily making a purchase. The tree patterns fitting to this format are exactly the ones accepted by following tree automaton:

\[
\begin{align*}
\delta(q_0, \text{home}) &= q_1q_2 \\
\delta(q_2, \text{Pr}) &= q_3 \\
\delta(q_4, \text{N}_i) &= q_6 \text{ for } i \in \{1, 2, 3, 4\} \\
\delta(q_6, \text{B}) &= \epsilon \\
\delta(q_5, \text{P}_1) &= \epsilon \text{ for } i \in \{1, 2, 3\}
\end{align*}
\]

Figure 1(a) illustrates the (very simplified) tree structure of the site. In Figure 1(b), we present a pattern verifying the given constraint and in 1(c) how the tree automaton navigates through it, starting at its root and reaching its leaves. The navigation proceeds as follows: \((q_0, \text{root}) \rightarrow q_1q_2\); \((q_1, \text{Nb}) \rightarrow q_3q_4\); \((q_3, \text{N}_1) \rightarrow \epsilon\); \((q_4, \text{N}_2) \rightarrow q_6\); \((q_6, \text{B}) \rightarrow \epsilon\); \((q_2, \text{Pr}) \rightarrow q_5q_5\); \((q_5, \text{P}_1) \rightarrow \epsilon\); \((q_5, \text{P}_2) \rightarrow \epsilon\). Notice that for a pattern to be accepted, the automaton rules must associate the regular expression \(\epsilon\) to each reached leaf, meaning that this node has no children.

In this paper, we focus on applications of the CobMiner method in two different contexts where data is naturally structured as trees: XML Mining and Web Usage Mining. We show how real data is pre-processed in order to be mined by CobMiner and how the constraints (tree automata) are specified by the users in these two different contexts. An extensive set of experiments over real datasets in these two contexts is presented and analyzed.

2 Tree Pattern Mining with Constraints

Trees and String Encodings. Let \(L = \{l_1, \ldots, l_m\}\) be a set of labels. We can assume that labels are mapped into natural numbers, so \(L \subseteq \mathbb{N}\). A labeled tree over \(L\) is a directed, acyclic and connected graph \(T = (N, E)\), where \(N = \{n_0, n_1, \ldots, n_p\}\) and \(E\) denote the set of nodes and edges respectively and where the following conditions are verified: (1) node \(n_0\) is a special and unique node in \(N\) such that there is no \(u \in N\) with \((u, n_0) \in E\). This node is called the root of \(T\). (2) there is a mapping \(l : N \rightarrow L\) which associates labels to each node of \(T\). A tree is said to be ordered if for each node \(n \in N\), the set \(\text{Child}(n) = \{u \in N \mid (n, u) \in E\}\), whose elements are called children of \(n\), is total ordered. We use a string encoding to represent a tree \(T\), built as follows. We assume that the nodes of \(T\) are numbered according to their positions in the depth-first traversal of the tree \(T\). So, the root is the node \(n_0\), its first child is node \(n_1\), the first child of \(n_1\) is node \(n_2\) and so on. The string encoding of \(T\), denoted by \(s_T\) is obtained by performing a depth-first traversal of \(T\), starting at the root and adding the labels of the traversed nodes into \(s_T\). Whenever we backtrack from a child to its parent we add a unique symbol -1 to the string encoding \(s_T\).

Data trees and tree patterns. Let \(T = (N_t, E_t)\) and \(S = (N_s, E_s)\) be trees. We say that \(S\) is an embedded subtree of \(T\) (denoted as \(S \preceq T\)) if: (1) \(N_s \subseteq N_t\) and (2) \((x, y) \in E_s\) if and only if \(x\) is an ancestor of \(y\) in
Tree Automata and Local Tree Automata. Let $L$ be a finite alphabet. A tree automaton over $L$ is a structure $A = (Q, q_0, L, \delta)$, where $Q$ is a finite set of states, $q_0$ is the initial state and $\delta$ is a mapping $\delta : Q \times L \rightarrow 2^Q$ associating to each pair $(q, a) \in Q \times L$ a regular expression over $Q$, i.e., a regular language over the set $Q$. We say that two states $q_1$ and $q_2 \in Q$ are competitors if there exists a symbol $a \in L$ and different rules $\delta(q_1, a) = e_1$ and $\delta(q_2, a) = e_2$; that is, label $a$ can be processed by both states $q_1$ and $q_2$. A local tree automaton is a tree automaton without competing states. Local tree automata have been introduced in (Takahashi 1975) and roughly corresponds to DTDs. For a comprehensive survey on different kinds of tree automata (including local ones), see (Murata et al. 2005). Local Automata are used in this paper, since they can specify a large class of constraints over trees and more important, it reduces the complexity of the mining algorithm. For instance, the automaton $A$ described in Example 1 is not local, since the states $q_3$ and $q_4$ are competitors.

Valid Tree Patterns. Let $A = (Q, q_0, L, \delta)$ be a tree automaton and $T = (N_T, E_T)$ be a tree. A run of $A$ on $T$ is a mapping $\lambda : N_T \rightarrow Q$ such that for every node $v \in N_T$ with $k$ children $v_1, v_2, ..., v_k$, the word $\lambda(v_1) \ldots \lambda(v_k)$ verifies the regular expression $\delta(\lambda(v), l(v))$, where $l(v) \in L$ is the label of $v$ in $T$. We say that $A$ accepts $T$ (or that $T$ is valid with respect to $A$) if there exists a run $\lambda$ such that for each leaf $v$ of $T$ we have $\delta(\lambda(v), l(v)) = \varepsilon$. For instance, the tree pattern depicted in Figure 1(b) is valid with respect to the tree automaton $A$ of example 1. The accepting run is depicted in Figure 1(c).

Our mining task is the following: Given a tree dataset $T$, a tree automaton $A$ and a minimum support threshold $\alpha$, find all frequent tree patterns which are valid w.r.t. the automaton $A$.

The Algorithms CobMiner and TreePatternPP. There are two ways to solve our mining task. The simpler way is to use any well-known tree pattern mining algorithm and follow it by a post-processing phase where the frequent patterns obtained are tested for validity w.r.t. the automaton specifying the constraints. Another way to solve this task is to push the automaton inside the mining process and not only at the post-processing phase. In this way, we eliminate the test for validity. The algorithm CobMiner we introduced in (de Amo et al. 2007) solves the mining task following the second approach, that is, it uses the automaton inside the mining process. In order to evaluate the performance of CobMiner, we consider the algorithm TreeMinerPP which adopts the first approach: it uses the well-known algorithm TreeMiner (Zaki 2002) for obtaining the frequent tree patterns and afterwards, a test for validity w.r.t. the automaton. At a high level, CobMiner’s framework is similar in structure to the Tree Miner algorithm for mining tree patterns without constraints, introduced in (Zaki 2002). Concerning the handling of constraints in the mining process, it follows the ideas introduced in (Garofalakis et al. 1999) in the context of constraint-based sequential pattern mining. However, adapting these ideas in the tree pattern mining context is not a minor task. For more details on the CobMiner algorithm, see (de Amo et al. 2007).

3 XML Mining with Constraints

In this section we present a constraint-based approach for XML Mining based on the CoBMiner method for mining tree patterns satisfying a given tree automata. We first show how XML data is pre-processed in order to be acceptable as input for CoBMiner algorithm. Then, we describe an interface prototype which guides the user to express the properties of the patterns he is interested in. These properties are automatically
transformed into a tree automaton by the system. The entire transformation process is transparent to the user.

**Pre-processing the input data.** The input data before the pre-processing phase is a XML document represented as a unique tree. The pre-processing phase is achieved as follows: (1) The user browses the tree and select a tag \( t \) of his interest. The whole document then gives rise to a set of subtrees having the same tag \( t \) as their roots labels. Figure 2(a) shows a piece of the XML document SigmodRecord. Let us suppose that the user is interested in mining patterns appearing in the input data concerning articles. In this case, his tag of interest is article and the piece of the XML document would produce two input data trees \( T_1 \) and \( T_2 \) shown in the figure. (2) The labels associated to the nodes of the subtrees are codified by natural numbers and each tree \( T \) is transformed into its string encoding \( s_T \). Figure 2(b) shows the string encodings for the subtrees \( T_1 \) and \( T_2 \). Here, we assume the following map for the node labels: article \( \rightarrow 7 \), title \( \rightarrow 8 \), Architectural ... \( \rightarrow 18 \), InitPage \( \rightarrow 10 \), and so on.

**How users specify the constraints.** Constraint specification depends on the structure which must be verified by the mined patterns as well as the contents of their leaves. In order to guide the user to express these information characterizing the interesting patterns according to his point of view, we developed an interface which works as follows: (1) Given a tag of interest \( t \), a maximal tree \( T_{\text{max}} \) with root label \( t \) is found which contains all the subtrees belonging to the pre-processed input data; (2) for each leaf node in \( T_{\text{max}} \) a list of its contents is proposed to the user; (3) the interface allows the user to navigate through the maximal tree \( T_{\text{max}} \), to choose the tags he is interested in and for each leaf node, to select its contents in a menu; (3) the information provided by the user in this way is then internally converted by the system into a tree automaton. The tree depicted in Figure 2(c) shows the tags selected by the user in the maximal tree corresponding to the trees \( T_1 \) and \( T_2 \) described in (a). For each leaf node, it is shown the contents picked up by the user in the corresponding menu. The maximal tree contains the tags article, title, initPage, endPage, authors and author. The user restricted the contents of the tag author indicating by that he is interested only in patterns corresponding to articles whose authors are “Jeffrey D. Ullman” or “Michael Stonebreaker”. The local tree automaton associated to this user restriction is given by:

\[
\delta(q_0, \text{article}) = q_1; \delta(q_1, \text{article}) = \epsilon; \delta(q_2, \text{initPage}) = \epsilon; \delta(q_3, \text{endPage}) = \epsilon; \delta(q_4, \text{authors}) = q_5; \\
\delta(q_5, \text{author}) = (q_0 + q_7); \delta(q_6, \text{Jeffrey D. Ullman}) = \epsilon; \delta(q_7, \text{Michael Stonebreaker}) = \epsilon.
\]

**Experimental results over XML datasets.** All experiments over XML datasets were performed on 2.3GHz Intel Core 2 PC with 4GB memory running Debian Linux 4.0. We have considered three XML dataset consisting of subtrees of the XML documents:

*DB-SigMod: Subtrees of the XML document SigmodRecord.xml obtained from the XML Data Repository*

---

2. The roots of the subtrees correspond to the tag `<article>`. The whole database contains 1504 trees. We used the automaton A-Sig for mining this dataset (given in the Appendix). It restricts the patterns structure in the following way: only articles having title, the initial and final pages and having one or more authors are accepted. The restrictions on the contents are made over the tag `author`.

DB-People: Subtrees of the XML document people55.xml obtained from the Movie Database 3. The XML file contains a list of famous peoples in the movie database. The roots of the subtrees correspond to the tag `<person>`. The whole database contains 3479 trees. We used the automaton A-People for mining this dataset (given in the Appendix). It restricts the pattern’s structure in the following way: only those people for whom the following information is given are considered: name, function code, start year, last work’s year, family name, nickname, date of birth, date of death, birth country and colleagues (people with whom they worked in the same film). Restrictions on the contents are made over the colleague’s names.

DB-Mains: Subtrees of the XML document mains243.xml obtained from the Movie Database 4. The XML file contains a list of movies. The roots of the subtrees correspond to the tag `<film>`. The whole database contains 7000 trees. The original one contains 12000 and was reduced for our purpose. We used the automaton A-Mains for mining this dataset (given in the Appendix). It restricts the patterns structure in the following way: only films where the following information is given are considered: title, director, producer, process used to make the movie and year. Restrictions on the contents are made over `year`.

Performance Analysis. In Figure 3(a), we present the results of the experiments comparing the performances of CobMiner and TreeMinerPP over the DB-SigMod Dataset. When considering a minimum support of 2.5%, CobMiner is about 4 times more efficient than TreeMinerPP. This difference between the performance of the two algorithms is due to the restriction on the search space of CoBMiner, during the generation phase. The mining process is directed by the tree automaton. Figures 3(d) and (e) show the behaviour of both algorithms when the automaton specifying the restrictions change. Figure 3(d) and 3(e) compare the performance of both algorithms where the dataset used was DB-People and the parameters $k$ and $K$ of the automata vary (not simultaneously) 5. In Figure (d), parameter $K$ is fixed as 2 and $k$ varies in $\{1,2,3,4\}$. In Figure (e), parameter $k$ is fixed as 1 and $k$ varies in $\{1,2,3,4\}$. These parameters affect the size of the “hidden” patterns in the dataset. As $k$ and $K$ increase, the average fanout and average depth of the hidden patterns increase as well. As expected, the execution time of both algorithms increase, since the number of levels in the recursive execution is related to the size of the “hidden” patterns.

Scalability Analysis Figure 3(c) shows how both algorithms scale when the number of trees in the database is increased from 1504 to 3479, and from 3479 to 7000. As the three databases have different structures and for the scalability test it would be important to fix the automaton and only vary the dataset size, we try to find similar automata for each of the dataset. These automata are A-SIG, A-People and A-Mains and are given in the Appendix. With a minimum support of 5% and constraints given by these automata, we notice that CobMiner is more efficient than TreeMinerPP. Notice also, that TreeMinerPP scales up almost linearly. However, concerning the results obtained for CoBMiner, the behaviour is not uniform. Unexpectedly, the small database presented the worst execution time. This is explained by the fact that the 3 automata were not exactly the same for the 3 datasets. As CobMiner uses the automaton inside its mining phase, a small change in its complexity can produce a non-negligible increase in the execution time.

4 Web Mining with Constraints

In this section we present a constraint-based approach for Web Mining based on the CoBMiner method for mining tree patterns satisfying a given tree automata. We first describe the pre-processing phase for a dataset


5. $k$ and $K$ are related to the fanout and depth of the valid patterns accepted by the automaton. For more details, see (de Amo et al. 2007).
of web logs and how our interface captures the restrictions provided by the user, in the Web Mining case.

**Pre-processing Web Logs.** In a Web Mining context, the pre-processing phase is a rather more complex task. It takes as input a set of web logs corresponding to users navigations through a given Web portal. This phase consists in four basic steps: (1) data cleansing, (2) user identification, (3) categorization and (4) transformation. The first step, the data cleansing, eliminates irrelevant entries in the web logs like for instance those with suffixes .gif,.css,.js,etc. The remaining pages or documents are only those which have been accessed by the users. The second step consists in identifying the user’s sessions. A user is identified by an IP or hostname, since we assume that the logs do not store information concerning user’s login validation. A user’s session is defined as the navigation corresponding to the same IP and which occurred during a given time interval \( d \). In our experiments, we assumed that \( d = 30 \) minutes. The third step consists in mapping sequences of visited pages into categories. This categorization procedure is executed in order to simplify data tree and pattern specification. A table containing the categorization information is provided as input to the pre-processing algorithm, in the form of a XML document. The fourth step consists in transforming the user’s sessions into trees. The algorithm we implemented to accomplish this task is based on the one presented in (Ivancsy and Vajk 2006). The tree corresponding to a session is built as follows. The first log entry corresponds to its root. The log entries following the first one are inserted into the tree following a depth-first traversal. This process continues until a url which has already been considered in this session is found or until the entire session has been scanned. In case a repeated url is found, the next session entry is inserted into the tree as a sibling of the last inserted entry. Figure 4(a) shows a piece of a web log file containing five log registers. These web logs correspond to the pages visited by a unique user when navigating through the UFU’s web site. We can assume that this is a user session. Figure 4(b) illustrates the categorization table used in the pre-processing phase. In Figure 5(a) we show the categorization table in XML format. In Figure 4(c) we show the access tree corresponding to this particular user session. In this example, the access tree is built as follows: the first log entry, **Serviços** is inserted in the tree as its root. The second and third log entries (after the categorization step) are **Pos Graduação Strictu Sensu** and **Pos Graduação Lato Sensu**. They are inserted in the same path starting at the root **Serviços**. The fourth and second entries are identical. So, the fifth entry is inserted as the second child of the **Pos Graduação Strictu Sensu** entry.

**How users specify the constraints.** The method allowing user’s to specify restrictions on the navigation patterns to be mined is similar to the one developed for mining XML documents, described in the previous

---


7[^7]Serviços

8[^8]Graduate Course and Specialization Course respectively.
section. However, some minor modifications have to be made in order to adapt the method to the Web Mining context. These modifications are due to the fact that, in XML Mining context, the trees in the dataset correspond to subtrees in the whole input document, obtained by selecting a tag of interest. In the Web Mining context, the trees in the dataset correspond to user’s sessions and can be quite unlike. In order to bring some uniformity to these input trees, after the pre-processing phase the set of access trees is stored as a unique XML document. In the XML document corresponding to the access trees, each user session is identified by a tag <user-session>. This tag corresponds to the tag of interest used in the pre-processing algorithm for XML Mining. From this point on, the method for specifying the constraint follows the same lines as in the XML Mining context. A maximal tree is built and by navigating through this tree, the user should identify some nodes. Besides, he should inform the type of each node he has selected. A node can be a AND-type or a OR-type node. In Figures 5(b) and (c), we present two examples of restrictions over the XML document of Figure 5(a) (we remind that this XML document represents the entire input web log file). The first restriction expresses the fact that the user is interested in mining navigation patterns conforming to the following format: the navigation starts at the category Serviços, then follows to the category Pos Graduação Strictu Sensu and after that, follows to the category Unidades Acadêmicas or Pos Graduação Lato Sensu. Notice that the two user sessions represented by the trees \( T_1 \) and \( T_2 \) in Figure 5(a) conform to this format. The second restriction expresses the fact that the user is interested in mining navigation patterns conforming to the following format: the navigation starts at the category Serviços, then follows to the category Pos Graduação Strictu Sensu and after that, follows to the category Unidades Acadêmicas and Pos Graduação Lato Sensu. Notice that both pages Unidades Acadêmicas and Pos Graduação Lato Sensu have to be accessed. In this case, only the first user session represented by the tree \( T_1 \) in Figure 5(a) conforms to this format. For lack of space, we present in the Appendix the local tree automata corresponding to these two constraints.

**Experimental Results over Web Log datasets.** All experiments over Web log datasets were performed on a 2.3GHz Intel Core 2 PC with 4GB memory running Debian Linux 4.0. We used the dataset DB-UfuLog.
consisting of subtrees of the Web log file from the UFU’s web site\(^9\). These logs are the result of user’s access during a one week period. The whole database contains 4399 trees. We used the automaton A-Log for mining this dataset (given in the Appendix). It restricts the pattern’s structure in the following way: only user’s sessions starting in page “Pesquisa e Pos-graduação”\(^10\), then going forward to page “Pos-graduação Strictu Sensu”, then backtracking to “Pesquisa e Pos-graduação” and then going forward to “Pos-graduação Lato Sensu” are considered interesting to be mined.

**Performance Analysis.** In Figure 3(b), we present the results of executing CoBMiner and TreeMinerPP over the DB-UFULog dataset, by changing the minimum support from 0.5% to 10%. The behaviour of both algorithms are as expected: CoBMiner is more efficient than TreePatternPP and the difference between their respective performance decreases as the support increases.

**References**


---

\(^9\)http://www.ufu.br.

\(^{10}\)Research and Graduation Courses
Appendix

Tree Automaton A-L1 = (Q, q₀, Σ, δ)

Q = \{q₀, q₁, q₂, q₃\}

Σ = \{servicos, pos graduacao strictu sensu, pos graduacao lato sensu, unidades academicas\}

δ(q₀, servicos) = q₁;
δ(q₁, pos graduacao strictu sensu) = q₂;q₃;
δ(q₂, pos graduacao lato sensu) = ε;
δ(q₃, unidades academicas) = ε;

Tree Automaton A-L2 = (Q, q₀, Σ, δ)

Q = \{q₀, q₁, q₂, q₃\}

Σ = \{servicos, pos graduacao strictu sensu, pos graduacao lato sensu, unidades academicas\}

δ(q₀, servicos) = q₁;
δ(q₁, pos graduacao strictu sensu) = q₂;q₃;
δ(q₂, pos graduacao lato sensu) = ε;
δ(q₃, unidades academicas) = ε;

Tree Automaton A-Sig = (Q, q₀, Σ, δ)

Q = \{q₀, q₁, q₂, q₃, q₄, q₅, q₆, q₇, q₈, q₉, q₁₀, q₁₁, q₁₂, q₁₃, q₁₄\}

Σ = \{article, title, initPage, endPage, authors, author, James H. Burrows, Jeffrey D. Ullman, Edwin McKenzie, Ahmed K. Elmagarmid\}

δ(q₀, article) = q₁,q₂,q₃,q₄;
δ(q₁, title) = ε;
δ(q₂, initPage) = ε;
δ(q₃, endPage) = ε;
δ(q₄, authors) = q₅,q₆⁺;
δ(q₅, author) = q₆;
δ(q₆, JamesH.Burrows) = ε;
δ(q₆, JeffreyD.Ullman) = ε;
δ(q₆, EdwinMcKenzie) = ε;
δ(q₆, AhmedK.Elmagarmid) = ε;

Tree Automaton A-People = (Q, q₀, Σ, δ)

Q = \{q₀, q₁, q₂, q₃, q₄, q₅, q₆, q₇, q₈, q₉, q₁₀, q₁₁, q₁₂, q₁₃, q₁₄\}

Σ = \{person, pname, pcode, yearstart, yearend, familynm, givennm, dob, dod, background, rels, workedwith, colleague, name, Hitchcock, Mirta Ibarra, Elizabeth Montegomery, Bertolucci\}

δ(q₀, person) = q₁,q₂,q₃,q₄,q₅,q₆,q₇,q₈,q₉,q₁₀,q₁₁,q₁₂,q₁₃,q₁₄;
δ(q₁, pname) = ε;
δ(q₂, pcode) = ε;
δ(q₃, yearstart) = ε;
δ(q₄, yearend) = ε;
δ(q₅, familynm) = ε;
δ(q₆, givennm) = ε;
δ(q₇, dob) = ε;
δ(q₈, dod) = ε;
\[\delta(q_0, \text{background}) = \varepsilon;\]
\[\delta(q_{10}, \text{rels}) = \varepsilon;\]
\[\delta(q_{11}, \text{workedwith}) = \varepsilon;\]
\[\delta(q_{12}, \text{colleague}) = \varepsilon;\]
\[\delta(q_{13}, \text{name}) = \varepsilon;\]
\[\delta(q_{14}, \text{Hitchcock}) = \varepsilon;\]
\[\delta(q_{14}, \text{MirtaIbarra}) = \varepsilon;\]
\[\delta(q_{14}, \text{ElizabethMontgomery}) = \varepsilon;\]
\[\delta(q_{14}, \text{Bertolucci}) = \varepsilon;\]

Tree Automaton A-Mains = (\(Q, q_0, \Sigma, \delta\))

\[Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}\]

\[\Sigma = \{\text{film, t, year, dirs, prods, prcs, 1975, 1976, 1998, 1999}\}\]

\[\delta(q_0, \text{film}) = q_1.q_2.q_3.q_4.q_5;\]
\[\delta(q_1, t) = \varepsilon;\]
\[\delta(q_2, \text{year}) = q_6\]
\[\delta(q_3, \text{dirs}) = \varepsilon;\]
\[\delta(q_4, \text{prods}) = \varepsilon;\]
\[\delta(q_5, \text{prcs}) = \varepsilon;\]
\[\delta(q_6, 1975) = \varepsilon;\]
\[\delta(q_6, 1976) = \varepsilon;\]
\[\delta(q_6, 1998) = \varepsilon;\]
\[\delta(q_6, 1999) = \varepsilon;\]

Tree Automaton A-Log = (\(Q, q_0, \Sigma, \delta\))

\[Q = \{q_0, q_1, q_2\}\]

\[\Sigma = \{\text{pesquisa e pos-graduacao, pos graduacao strictu sensu, pos graduacao lato sensu}\}\]

\[\delta(q_0, \text{pesquisa e pos-graduacao}) = q_1.q_2;\]
\[\delta(q_1, \text{pos graduacao strictu sensu}) = \varepsilon;\]
\[\delta(q_2, \text{pos graduacao lato sensu}) = \varepsilon;\]