TYPED TREE AUTOMATA

by

Virendra Bhavsar
Lev Goldfarb
Andrew Mironov

TR99-126, October 1999

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
E-mail: fcs@unb.ca
www: http://www.cs.unb.ca

Contents

Introduction

Trees are used in most areas of computer science, both practical and theoretical, such as for example in mathematical logic, compiling theory, program verification theory, pattern recognition (see [4], [5], [18], [19], [24], [27]).

One of the most important problems related to trees is a problem of tree classes representation. For example, in structural pattern recognition theory a class C of trees can be considered as a set of various representatives of a some concept, and the following problem has a big importance: to construct a representation of the class C, such that the problem of checking is a tree t a representative of this concept, or no (i.e. is $t \in C$ or $t \notin C$) is decidable in real time.

Many methods and tools can be fruitfully applied to the definition and study of tree classes: grammars, systems of substitutions, congruences, etc. (see for example [15]) One of them is tree automata approach.

Tree automata have been studied for a long time (see [13] [17], [9], [10], [12], [14], [31], [32], [33], [35], [36]). In logic tree automata theory has used for characterization of decidable second order theories (see [30]). In computer science it was established dependency between complexity of decision procedures for most temporal and dynamic logics, and the complexity of testing of emptiness of corresponding tree automata(see [26]).

The present paper is related to the generalization of a tree automaton to the concept which is appropriate for representation of classes of typed trees.

Typed trees are related to representation of objects associated with some type system. A type system classifies objects under consideration according to their specific structure or properties. The notion of type expresses the fact that one just cannot apply any operator to any value. The concept of a typed structures in mathematics and computer science have been studied for a long time (see, for example, [1], [2], [6], [8], [11], [20], [22], [23], [25], [28]).

In computer science, the earliest type systems, beginning in the 1950s (e.g., FORTRAN), were used to improve efficiency of numerical calculations by distinguishing between natural-number-valued variables and arithmetic expressions and real-valued ones. In the late 50s and early 60s, the classification was extended to structured data (arrays, lists, etc.) and higher-order functions. At the present time many areas of active research in computer science related to type approach: for example, in the theory of object-oriented programming there are investigated the concepts of polimorphism (which

allows a single term to be used with many different types, subtyping and object types (which address the special needs of object-oriented programming styles), etc.

We consider algebraic formalization of a concept of a typed tree, i.e. trees in the present paper are actually terms over finite typed signature. The algebraic structure on the set of trees is necessary for discussing recognizable classes of trees. We define a typed tree automata that accept the recognizable these classes. The concept of a typed tree automaton is a natural generalization of the concept of an ordinary tree automaton. One of the most essential deficiencies of the concept of an ordinary tree automaton is its untyped structure, that not allows represent classes of typed trees in a natural way.

We generalize the language of tree automata by introducing the concept of a typed set of states and typed transition functions. For typed tree automata we prove an analog of the well-known theorem (see [17], [7], [16], [21]) about construction of a minimal tree automaton which represents given class of trees.

We now describe the content of the present paper.

The paper consists of five sections and conclusion. In the first section, we give some basic definitions related to the concept of a typed tree. In the second section we represent two examples of encoding of some classes of patterns (pictures with hierarchical structure and letters) by typed trees. In the third section we recall some definitions related to typed universal algebra. In the fourth section we define the notion of typed tree automata and give example of a typed tree automaton that represents some class of typed trees. In the fifth section we introduce the necessary algebraic definitions and prove the central results of the paper: the theorem about minimal typed tree automaton that represents given class of typed trees. In conclusion we discuss possible future directons of the research.

1 Σ–trees

In this section we represent some basic definitions related to the concept of a typed tree.

1.1 Γ–sets

We assume that there is given some set Γ, elements of which are called **data types**.

The concept of a data type in our model is a generalization of the same concept as it appears in programming languages. For example, the following entities can be considered as types: real numbers, integer numbers, lists, sets, etc.

A Γ–**set** is an arbitrary Γ–tuple M of disjoint sets of the form

$$M = \{M_i \mid i \in \Gamma\},$$

some of which can be empty.

For every $i \in \Gamma$ the set M_i is called a **domain of data** of the type i.

Below we will identify the Γ–tuple $M = \{M_i \mid i \in \Gamma\}$ with the disjoint union $\bigsqcup\{M_i \mid i \in \Gamma\}$ and will denote this disjoint union by the same symbol M.

For every $x \in M$ the symbol $type(x)$ denotes an element $i \in \Gamma$ such that $x \in M_i$.

Let $M = \{M_i \mid i \in \Gamma\}$ and $N = \{N_i \mid i \in \Gamma\}$ be a pair of Γ–sets.

A Γ–**mapping** from M to N is an arbitrary Γ–tuple f of mappings of the form

$$f = \{f_i : M_i \to N_i \mid i \in \Gamma\}.$$

1.2 Γ–signatures

Remind (see [29]) that an **algebraic** Γ–**type** is any $n+1$–tuple T of the form

$$(i_1, \ldots, i_n; j),$$

where $n \geq 0$ and $i_1, \ldots, i_n, j \in \Gamma$. If $n = 0$, then T has the form $(; j)$.

The concept of an algebraic Γ–type is a generalization of the concept of an arity of an algebraic operation.

A Γ–**signature** is a set Σ, every elememt σ of which is associated with some algebraic Γ–type $T(\sigma)$.

Elements of the Γ–signature Σ are called **operational symbols**.

An example of a Γ–signature related to the problem of a class description will be presented below.

For every $\sigma \in \Sigma$ the algebraic Γ–type $T(\sigma)$ is called a **type of the operational symbol** σ.

1.3 Definition of a Σ–tree

We assume that there is given some Γ–signature Σ.

Define a Γ–set $Tr(\Sigma)$, elements of which are called Σ–**trees**:

- $\forall i \in \Gamma$ every operational symbol $\sigma \in \Sigma$ such that $T(\sigma) = (;i)$ is a Σ–tree of the type i (i.e. is an element of the domain $Tr(\Sigma)_i$),
- for
  - every $n \geq 1$,
  - every $\sigma \in \Sigma$, such that $T(\sigma)$ is of the form $(i_1, \ldots, i_n; j)$,
  - and every n–tuple $(E_1, \ldots, E_n)$ of Σ–trees, such that

$$type(E_1) = i_1, \ldots, type(E_n) = i_n,$$

  the string $\sigma(E_1, \ldots, E_n)$ is a Σ–tree of the type j.

The Σ–trees can be represented by pictures as follows:

- every Σ–tree E of the form $\sigma \in \Sigma$, where $T(\sigma) = (;i)$ for some $i \in \Gamma$, is represented by a one-node graph without edges, and the unique node of this graph has a label σ:

Figure 1

- every Σ–tree E of the form $\sigma(E_1, \ldots, E_n)$, where $n \geq 1$, is represented by the graph $gr(E)$ of the form

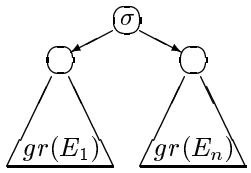


Figure 2

where $gr(E_1), \ldots, gr(E_n)$ are the graphs that represent the Σ–trees $E_1, \ldots, E_n$ correspondingly.

2 Examples of representation of patterns by Σ–trees

The language of Σ–trees is a powerful formalizm for representation of patterns. In this section we consider examples of representation of geometrical patterns by Σ–trees.

2.1 Example 1: representation of hierarchical patterns by Σ–trees

We consider the patterns with hierarchical structure like the following:

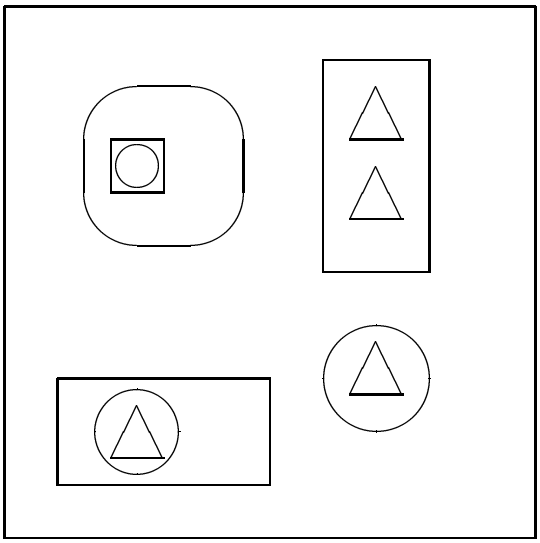


Figure 3

Every pattern is some set of geometrical figures: triangles, circles, squares and rectangles, which can contain each in other. Every geometrical figure in the pattern is associated in a natural manner with some number, which is called **a depth**, and represents its level of inclusion.

We are interesting only the relation of inclusion on the figures, i.e. our rules of encoding of patterns by Σ–trees must respect only the relation of inclusion of figures, and no more.

The patterns of the above form can be obtained, for example, from analysis of geographical maps, satellite photographies, etc.

Let

- the set Γ of data types consists of the following elements:
  - **arrow**,
  - **node**,
- the set Σ of operational symbols consists of the following symbols:

– t (triangle), $T(t) \stackrel{\text{def}}{=} (\mathbf{node};\ \mathbf{arrow})$,

– c (circle), $T(c) \stackrel{\text{def}}{=} (\mathbf{node};\ \mathbf{arrow})$,

– s (square), $T(s) \stackrel{\text{def}}{=} (\mathbf{node};\ \mathbf{arrow})$,

– r (rectangle), $T(r) \stackrel{\text{def}}{=} (\mathbf{node};\ \mathbf{arrow})$,

– $\forall n \geq 0$ the symbol n belongs to the set Σ, and

$$T(n) \stackrel{\text{def}}{=} (\underbrace{\mathbf{arrow}, \ldots, \mathbf{arrow}}_{n};\ \mathbf{node}).$$

The above pattern can be represented by the following Σ–tree, nodes of which are corresponded to the subpatterns of this pattern, and structure of which is corresponded to the hierarchy on the subpatterns:

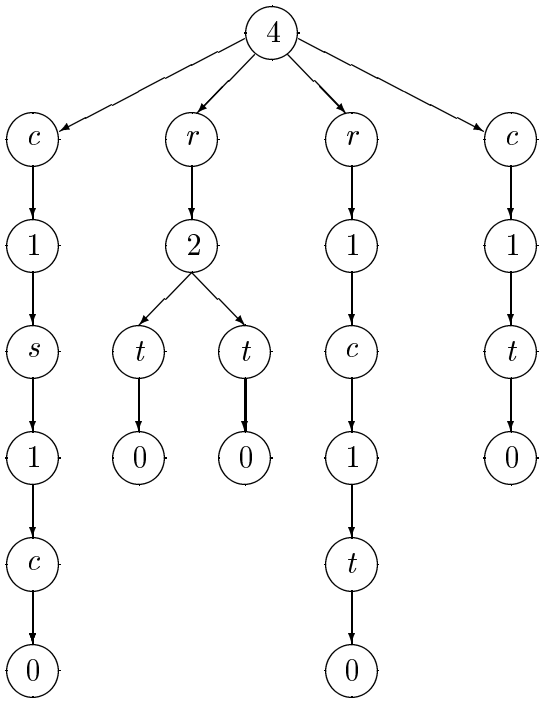


Figure 4

The algorithm of construction of the Σ–tree for the geometrical patterns with hierarchical structure consists of the following steps:

1. draw a node with a label n, where n is a number of figures of the depth 1 in the pattern, this node has a level 0, i.e. is a root of the Σ–tree,

2. draw n edges from the root of level 0, where every edge is associated with some figures of the depth 1 in the pattern,

3. draw nodes at the end of every edge with corresponding label (i.e. if the figure associated with this edge is a circle, then the label of the end of this edge is "c", etc.), these nodes have a level 1,

4. for every node N of the depth 1 draw one edge,

   - beginning of which is this node,
   - and end of which is a node with a label $n(N)$, where $n(N)$ is a number of figures of the depth 2 in the pattern, which contain in the figure associated with the node N,

5. et cetera.

Note that a geometrical pattern can be represented in general case by several Σ–trees, for example the above pattern can be represented also by

the following tree:

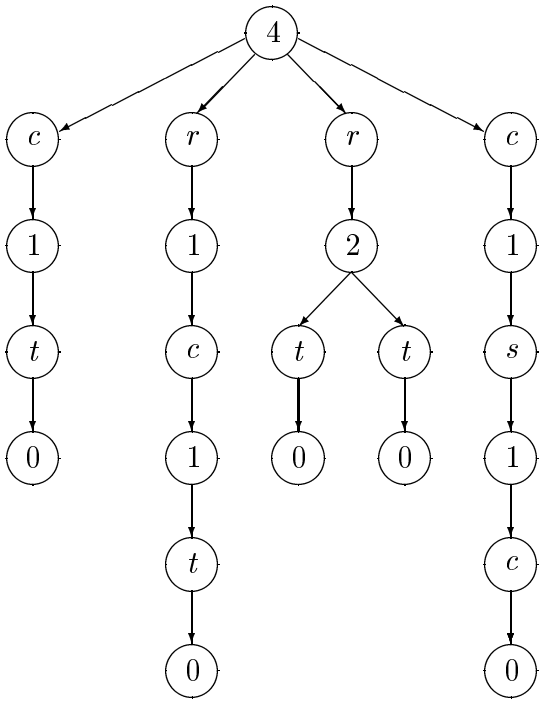


Figure 5

2.2 Example 2: letters

Letters in the arrow representation are the patterns of the following forms:

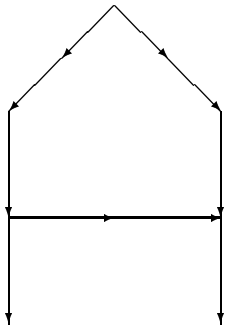


Figure 6: the letter "A"

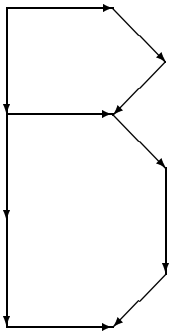


Figure 7: the letter "B"

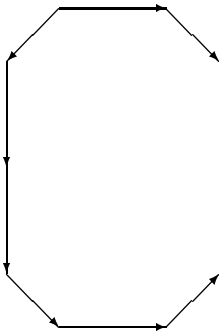


Figure 8: the letter "C"

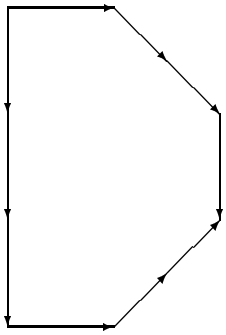


Figure 9: the letter "D"

Let

- the set Γ of data types consists of the following elements:
  - **arrow**,
  - **node**,
- the set Σ of operational symbols consists of the following symbols:
  - N (north arrow),
    $T(N) \stackrel{\text{def}}{=} (\mathbf{node};\ \mathbf{arrow})$,
  - S (south arrow),
    $T(S) \stackrel{\text{def}}{=} (\mathbf{node};\ \mathbf{arrow})$,
  - W (west arrow),
    $T(W) \stackrel{\text{def}}{=} (\mathbf{node};\ \mathbf{arrow})$,
  - E (east arrow),
    $T(E) \stackrel{\text{def}}{=} (\mathbf{node};\ \mathbf{arrow})$,
  - NW (north-west arrow),
    $T(NW) \stackrel{\text{def}}{=} (\mathbf{node};\ \mathbf{arrow})$,
  - NE (north-east arrow),
    $T(NE) \stackrel{\text{def}}{=} (\mathbf{node};\ \mathbf{arrow})$,
  - SW (south-west arrow),
    $T(SW) \stackrel{\text{def}}{=} (\mathbf{node};\ \mathbf{arrow})$,

– SE (south-east arrow),
$T(SE) \stackrel{\text{def}}{=} (\mathbf{node};\ \mathbf{arrow})$,

– $\forall n \geq 0$ the symbol n belongs to the set Σ, and

$$T(n) \stackrel{\text{def}}{=} (\underbrace{\mathbf{arrow}, \ldots, \mathbf{arrow}}_{n};\ \mathbf{node}).$$

– Let C be a set of symbols of the form

$$C \stackrel{\text{def}}{=} \{c_1, c_2, \ldots\}$$

Then $\forall n \geq 0,\ \forall k \geq 1$ the pair (n, c_k) belongs to the set Σ, and

$$T(n, c_k) \stackrel{\text{def}}{=} (\underbrace{\mathbf{arrow}, \ldots, \mathbf{arrow}}_{n};\ \mathbf{node}).$$

An idea of encoding of patterns in the arrow representation by trees is the following.

Let AR be an arrow representation of some pattern.

The tree $Tree(AR)$ corresponded to AR is a development of AR with a labelling of nodes.

More detail, for every node N of AR there is defined a set $TreeNodes(N)$ of corresponding nodes in the tree $Tree(AR)$:

- if N has at most one incoming arrows, then the set $TreeNodes(N)$ consists of one node with the label "n", where n is a quantity of outcoming arrows from the node N,
- if N has m incoming arrows, where $m > 1$, then the set $TreeNodes(N)$ consists of m nodes, every of which is associated with some arrow incoming from N:
  - one node "$(n, c(N))$", where
    - n is a quantity of outcoming arrows from the node N,
    - and $c(N)$ is an element of the above set C,
  - and $m-1$ nodes with the label "$(0, c(N))$", (the second component of the label $(0, c(N))$ is the common for all nodes from the set $TreeNodes(N)$).

We assume that if N_1 and N_2 are the pair of different nodes in AR with several incoming edges, then $c(N_1) \neq c(N_2)$.

Every arrow E in AR is encoded by a node $TreeNode(E)$ in $Tree(AR)$ with the label that displays a direction of this arrow.

For every arrow E in AR there is defined a pair of corresponding edges in the tree $Tree(AR)$: let the beginning of E is N_1, and the end of E is N_2, then the tree $Tree(AR)$ contains the edge

- beginning of which is a node $TN_1 \in TreeNodes(N_1)$ such that its label is "n" or "(n, c)", for some $n > 0$,
- and end of which is the node $TreeNode(E)$,

and the edge

- beginning of which is the node $TreeNode(E)$,
- and end of which is a node in the set $TreeNodes(N_2)$ which is associated with E.

The correspondence between AR and $Tree(AR)$ can be realized from the following example encoding of the above arrow representation of the letter "A":

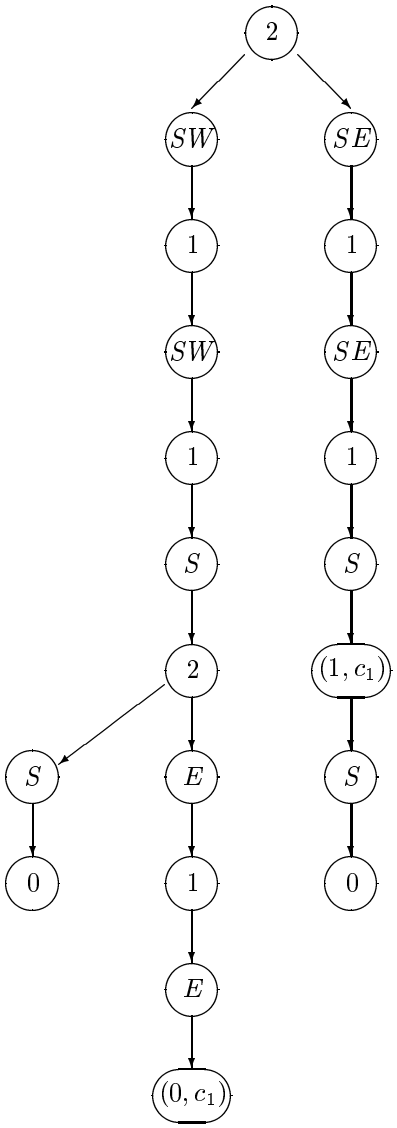


Figure 10: the encoded letter "A"

3 Σ–algebras

In this section we recall some definitions related to typed universal algebra.

3.1 Algebraical operations on Γ–sets

Let

- $Q = \{Q_i \mid i \in \Gamma\}$ be a Γ–set,
- and $T \stackrel{\text{def}}{=} (i_1, \ldots, i_n; j)$ be some algebraic Γ–type.

An **algebraic operation** on Q of the type T is an arbitrary mapping ω of the form

$$\omega : Q_{i_1} \times \ldots \times Q_{i_n} \to Q_j.$$

If $n = 0$, i.e. T is of the form $(; j)$, then the mapping ω has the form $\omega : \mathbf{1} \to Q_j$, where $\mathbf{1} = \{e\}$ is a one-element set. In this case the element $\omega(e) \in Q_j$ is denoted by the same symbol ω.

3.2 Σ–algebras

Let Σ be some Γ–signature.

A Σ–**algebra** is an arbitrary pair of the form (Q, δ), where

- Q is a Γ–set,
- $\delta \stackrel{\text{def}}{=} \{\delta_\sigma \mid \sigma \in \Sigma\}$ is a Σ–tuple of algebraic operations on Q, such that $\forall \sigma \in \Sigma$ a type of the operation δ_σ is equal to $T(\sigma)$.

  A result of applying of the operation δ_σ (where $T(\sigma)$ is of the form $(i_1, \ldots, i_n; j)$) to the n–tuple $(q_1, \ldots, q_n)$ of elements of corresponding types will be denoted by the symbol $\sigma(q_1, \ldots, q_n)$ (or σ, if $n = 0$), without mention of δ.

3.3 Example of a Σ–algebras

One of examples of Σ–algebras is the Σ–algebra

$$(Tr(\Sigma), \delta),$$

where $\delta \stackrel{\text{def}}{=} \{\delta_\sigma \mid \sigma \in \Sigma\}$ is the Σ–tuple of the following operations:

- if $T(\sigma) = (; j)$, then the operation

$$\delta_\sigma : \mathbf{1} \to Tr(\Sigma)_j$$

maps a unique element of the set $\mathbf{1}$ to the Σ–tree $\sigma \in Tr(\Sigma)_j$,

- if $T(\sigma) = (i_1, \ldots, i_n;\ j)$, where $n \geq 1$, then the mapping

$$\delta_\sigma : Tr(\Sigma)_{i_1} \times \ldots \times Tr(\Sigma)_{i_n} \to Tr(\Sigma)_j$$

maps every n–tuple $(E_1, \ldots, E_n)$ of Σ–trees of corresponding types to the Σ–tree $\sigma(E_1, \ldots, E_n)$.

3.4 Morphisms of Σ–algebras

Let (Q, δ) and (Q', δ') be a pair of Σ–algebras.

A **morphism** from (Q, δ) to (Q', δ') is an arbitrary Γ–tuple f of mappings of the form

$$f = \{f_i : Q_i \to Q'_i \mid i \in \Gamma\},$$

such that for every $\sigma \in \Sigma$

- if $T(\sigma)$ is of the form $(; j)$, then

$$f_j(\delta_\sigma) = \delta'_\sigma,$$

- if $T(\sigma)$ is of the form $(i_1, \ldots, i_n; j)$ where $n \geq 1$, then $\forall q_1 \in Q_{i_1}, \ldots, \forall q_n \in Q_{i_n}$

$$f_j(\delta_\sigma(q_1, \ldots, q_n)) = \delta'_\sigma(f_{i_1}(q_1), \ldots, f_{i_n}(q_n)).$$

For every $i \in \Gamma$ and every $q \in Q_i$ the element $f_i(q)$ has the equivalent notation $f(q)$.

3.5 Σ–tree morphism associated with a Σ–algebra

For every Σ–algebra (Q, δ) there exists a unique morphism from $(Tr(\Sigma), \delta)$ to (Q, δ), which is called **a Σ–tree morphism associated with** (Q, δ) and will be denoted by the symbol δ^*.

The morphism δ^* can be described by induction:

- for every $E \in Tr(\Sigma)$ which has the form σ, where $\sigma \in \Sigma$ is such that $T(\sigma)$ is of the form $(;i)$,
$$\delta^*(E) \stackrel{\text{def}}{=} \delta_\sigma,$$
- for every $E \in Tr(\Sigma)$ which has the form $\sigma(E_1, \ldots, E_n)$,
$$\delta^*(E) \stackrel{\text{def}}{=} \delta_\sigma(\delta^*(E_1), \ldots, \delta^*(E_n)).$$

It is not so difficult to prove that the Γ–subset
$$\delta^*(Tr(\Sigma)) \stackrel{\text{def}}{=} \{\delta^*(E) \mid E \in Tr(\Sigma)\}$$
of the Γ–set $Tr(\Sigma)$ is closed with respect to algebraic operations on the Σ–algebra (Q, δ), i.e. the Γ–subset $\delta^*(Tr(\Sigma))$ can be equipped by the structure of a Σ–algebra:
$$(\delta^*(Tr(\Sigma)), \delta)$$
(we denote the Σ–tuple of corresponding operations on $\delta^*(Tr(\Sigma))$ by the same symbol δ).

4 Representation of Σ–tree classes by Σ-tree automata

In this section we define the notion of typed tree automata and give example of a typed tree automaton that represents some class of typed trees.

4.1 Σ–tree classes and Σ–tree automata

A Σ–**tree class** is any subset $C \subseteq Tr(\Sigma)$ of the set $Tr(\Sigma)$.

A Σ–**tree automaton** is a pair $M \stackrel{\text{def}}{=} ((Q, \delta), \beta)$, where:

- (Q, δ) is a Σ–algebra,
- β is a subset of the Γ–set Q.

A Σ–tree class $C(M)$ associated with M is a subset $C(M) \subseteq Tr(\Sigma)$, which is defined as follows:
$$C(M) \stackrel{\text{def}}{=} \{E \in Tr(\Sigma) \mid \delta^*(E) \in \beta\}.$$

4.2 An example of a Σ–tree class which is represented by some Σ–tree automaton

Consider the Σ–tree class that consists of all patterns that represent the letter "A" in the arrow representation that was considered above. These patterns correspond to all possible hand-written letters "A" which can contain some deformations.

Any pattern from this class can be obtained by the following procedure:

- draw a point on the plane,
- draw a pair of disjoint vector traces starting from this point, which are called "a left trace" and "a right trace",
- choose an internal point A on the left trace and an internal point B on the right trace,
- draw a vector trace, the beginning of which is A and the end of which is B, and which has no intersictions with the legt trace and with the right trace.

For example, this class contains the following patterns:

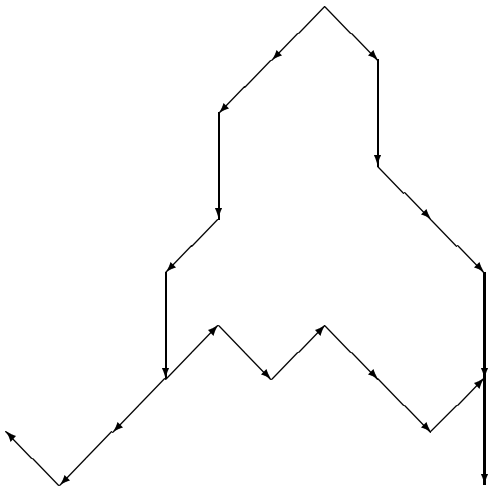


Figure 11: the letter "A" with some deformations

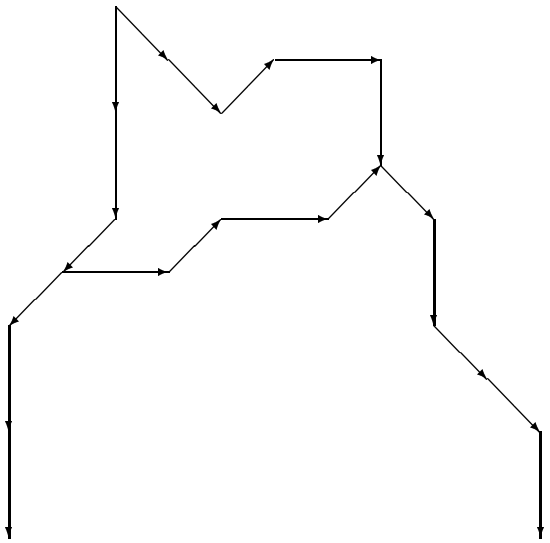


Figure 12: the letter "A" with some deformations

Let Σ be the above Γ–signature, where $\Gamma = \{\mathbf{arrow}, \mathbf{node}\}$:

$$\Sigma = \{N, S, W, E, NW, NE, SW, SE\} \cup$$
$$\cup \{n \mid n \geq 0\} \cup \{(n, c) \mid n \geq 0,\ c \in C\}.$$

Define the Σ–algebra (Q, δ) as follows.

The Γ–set Q consists of the following domains $Q_{\mathbf{node}}$ and $Q_{\mathbf{arrow}}$:

$$Q_{\mathbf{node}} \stackrel{\text{def}}{=} \{Q^i_{\mathbf{node}} \mid i = 1, \ldots, 10\},$$

$$Q_{\mathbf{arrow}} \stackrel{\text{def}}{=} \{Q^i_{\mathbf{arrow}} \mid i = 1, \ldots, 5\},$$

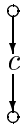
where the meaning of the elements $Q^i_{\mathbf{node}}$ and $Q^i_{\mathbf{arrow}}$ can be realized from the following graphical interpreration (below the symbol " ■ " denotes the garbage states in $Q_{\mathbf{node}}$ and $Q_{\mathbf{arrow}}$)

- $Q^1_{\mathbf{node}} \stackrel{\text{def}}{=}$ ■ $\quad Q^1_{\mathbf{arrow}} \stackrel{\text{def}}{=}$ ■
- $Q^2_{\mathbf{node}} \stackrel{\text{def}}{=}$ ○
- $Q^3_{\mathbf{node}} \stackrel{\text{def}}{=}$ c
- $Q^4_{\mathbf{node}} \stackrel{\text{def}}{=}$  $\quad Q^2_{\mathbf{arrow}} \stackrel{\text{def}}{=}$ 

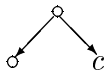
- $Q^5_{\mathbf{node}} \stackrel{\text{def}}{=}$ 

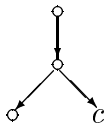
$Q^3_{\mathbf{arrow}} \stackrel{\text{def}}{=}$ 

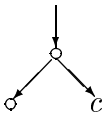
- $Q^6_{\mathbf{node}} \stackrel{\text{def}}{=}$ 

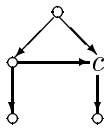
- $Q^7_{\mathbf{node}} \stackrel{\text{def}}{=}$ 

$Q^4_{\mathbf{arrow}} \stackrel{\text{def}}{=}$ 

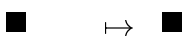
- $Q^8_{\mathbf{node}} \stackrel{\text{def}}{=}$ 

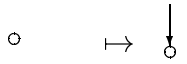
- $Q^9_{\mathbf{node}} \stackrel{\text{def}}{=}$ 

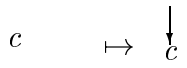
$Q^5_{\mathbf{arrow}} \stackrel{\text{def}}{=}$ 

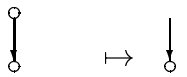
- $Q^{10}_{\mathbf{node}} \stackrel{\text{def}}{=}$ 

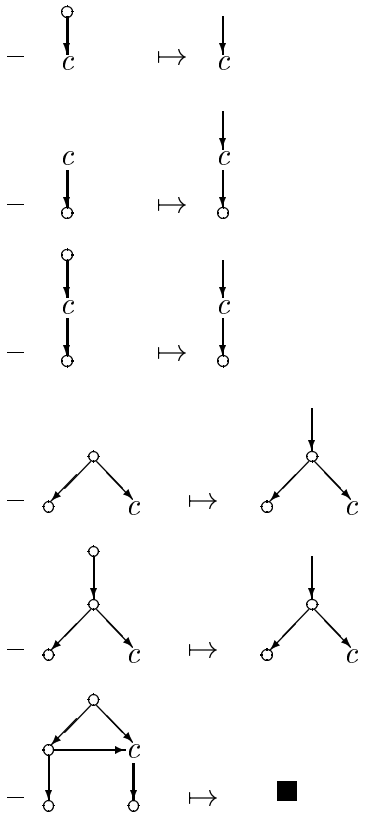
The Σ–tuple δ of mappings is defined as follows.

- The mappings $\delta_N, \delta_S, \delta_W, \delta_E, \delta_{NW}, \delta_{NE}, \delta_{SW}, \delta_{SE}$ are equal mappings of the form $Q_{\mathbf{node}} \to Q_{\mathbf{arrow}}$ and are defined as follows:
  - 

  - 

  - 

  - 

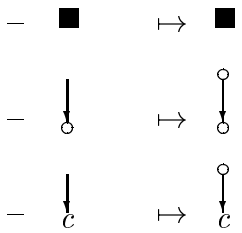


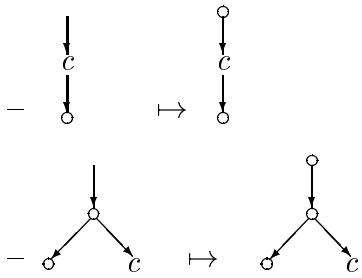
- The symbol δ_0 is interpreted by the state $\circ$

  the symbol $\delta_{(0,c)}$ is interpreted by the state c for every $c' \in C \backslash \{c\}$

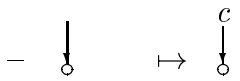
  the symbol $\delta_{(0,c')}$ is interpreted by the state $\blacksquare$

- The mapping δ_1 has the form $\delta_1 : Q_{\mathbf{arrow}} \to Q_{\mathbf{node}}$ and is defined as follows:



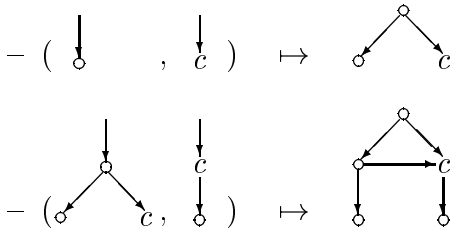


- The mapping $\delta_{(1,c)}$ has the form $\delta_{(1,c)} : Q_{\mathbf{arrow}} \to Q_{\mathbf{node}}$ and is defined as follows:

  - all other states from $Q_{\mathbf{arrow}}$ $\delta_{(1,c)}$ maps to the garbage state ■.

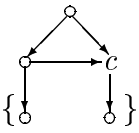
- The mapping δ_2 has the form $\delta_2 : Q_{\mathbf{arrow}} \times Q_{\mathbf{arrow}} \to Q_{\mathbf{node}}$ and is defined as follows:

  - all other pairs from $Q_{\mathbf{arrow}} \times Q_{\mathbf{arrow}}$ the mapping δ_2 maps to the garbage state ■.

- all other mappings from the Σ–tuple δ maps all its arguments to the garbage state ■.

The Σ–tree automaton which is associated with the class of all arrow representations of the letter "A" is the pair $M \stackrel{\text{def}}{=} ((Q, \delta), \beta)$, where

- (Q, δ) is the above Σ–algebra,
- and β is the one-element subset $\{$  $\}$ of the Γ–set Q.

The procedure of working of the Σ–tree automaton M on a Σ–tree t consists of parallel computation of states for every node of the tree t "bottom-up", starting from the leaves:

1. at first every leaf of the tree t is associated with an element of the set Q, which is an interpretation of the label of this leaf in the Σ–algebra (Q, δ),

2. for every node N of the tree t, such that states for all its childs $N_1, \ldots, N_k$ are already computed and are equal to $q_1, \ldots, q_k$ correspondingly, the state for the node N is equal by definition to the state $\delta_{label(N)}(q_1, \ldots, q_k)$,

3. after completing the procedure of computation of states for the nodes of the tree t we check the formula $q \in \beta$, where q is a state associated with the root of the t: if this formula is hold, then t belongs to the class $C(M)$.

The above procedure can be realized from the following example of working of the Σ–tree automaton on the above Σ–tree representation of the letter "A":

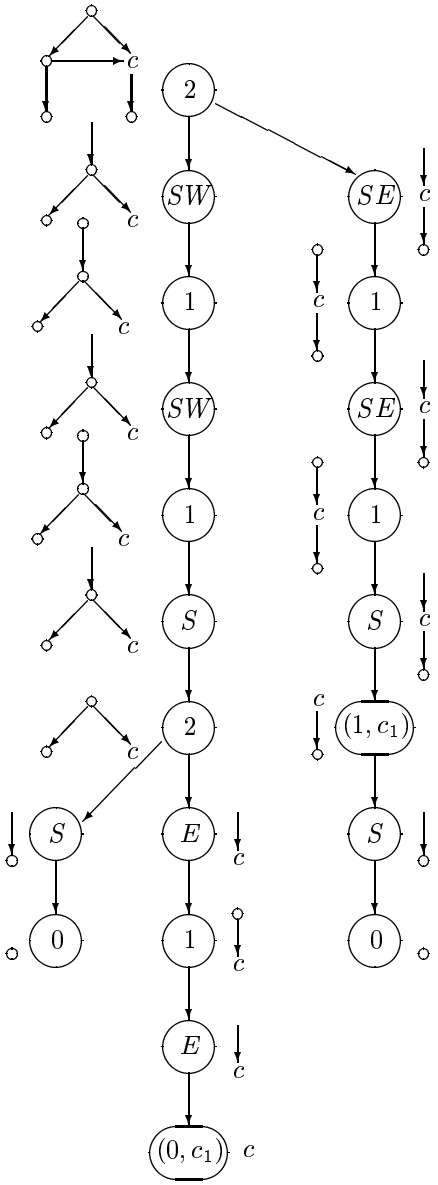


Figure 13: the procedure of working of the Σ–tree automaton on the Σ–tree

In this picture we write the computed states near from passed nodes of the Σ–tree.

The picture provides some explanation of our notations for states: all states of the automaton are represent some essential properties of subpatterns of analysed patterns. Every node of the Σ–tree is associated in a natural way with a subtree which represents some subpattern of the original pattern, and the state which is associated with this node reflects some properties of this subpattern. Computation of states in the process of functioning of the tree automaton reflects an idea of *aggregation* of subpatterns.

5 Minimal Σ–tree automata

The present section deals with the following problem: given a Σ–tree class C, find a minimal (in a certain meaning) Σ–tree automaton M, such that $C = C(M)$.

5.1 Congruences on Σ–algebras

Let (Q, δ) be a Σ–algebra.

A **congruence** on (Q, δ) is a Γ–tuple

$$R = \{R_i \subseteq Q_i \times Q_i \mid i \in \Gamma\}$$

of equivalency relations on domains of the Γ-set Q, such that for

- every $\sigma \in \Sigma$, such that $T(\sigma)$ is of the form $(i_1, \ldots, i_n;\ j)$, where $n \geq 1$,
- and every n–tuple of pairs of the form

$$(q_1, q_1') \in R_{i_1}, \ldots, (q_n, q_n') \in R_{i_n}$$

the pair $(\delta_\sigma(q_1, \ldots, q_n), \delta_\sigma(q_1', \ldots, q_n'))$ belongs to R_j.

Let

- $Q/R \stackrel{\text{def}}{=} \{Q_i/R_i \mid i \in \Gamma\}$
- and $\pi \stackrel{\text{def}}{=} \{\pi_i : Q_i \twoheadrightarrow (Q_i/R_i) \mid i \in \Gamma\}$

be the Γ–tuples of the factor-sets and canonical projections corresponded to the equivalency relations $\{R_i \mid i \in \Gamma\}$.

It is not difficult to prove that there exists a unique Σ–tuple $\delta' \stackrel{\text{def}}{=} \{\delta'_\sigma \mid \sigma \in \Sigma\}$ of algebraic operations of corresponding types on the Γ–set Q/R, such that the Γ–tuple π is a morphism from (Q, δ) to $(Q/R, \delta')$.

The set of all congruences on the Σ–algebra (Q, δ) can be considered as a partially ordered set: for every pair $R' = \{R'_i \mid i \in \Gamma\}$, $R'' = \{R''_i \mid i \in \Gamma\}$ of congruences on (Q, δ)

$$R' \leq R'' \quad \Leftrightarrow \quad \forall i \in \Gamma \quad R'_i \subseteq R''_i.$$

5.2 The congruence R^f

Let

- (Q, δ) be a Σ–algebra,
- Y be a Γ–set,
- and f be any Γ–mapping from Q to Y:

$$f = \{f_i : Q_i \to Y_i \mid i \in \Gamma\}.$$

Define the Γ–tuple of equivalency relations

$$Ker(f) \stackrel{\text{def}}{=} \{Ker(f)_i \subseteq Q_i \times Q_i \mid i \in \Gamma\}$$

as following:

$$\forall i \in \Gamma \; Ker(f)_i \stackrel{\text{def}}{=} \{(q, q') \in Q_i \times Q_i \mid f_i(q) = f_i(q')\}.$$

Below we define a congruence R^f on the Σ–algebra (Q, δ), which will be a greatest (with respect to the above relation of partial order) congruence among all congruences R on (Q, δ) which satisfy the condition $R \leq Ker(f)$.

5.2.1 Derivative unary operations

Let there is given

- a pair $i, j \in \Gamma$
- an element $\sigma \in \Sigma$, such that
  - $T(\sigma)$ is of the form $(i_1, \ldots, i_n;\ j)$, where $n \geq 1$,
  - $i = i_k$ for some $k \in \{1, \ldots, n\}$,
- a $(n-1)$–tuple

$$\{q_s \in Q_{i_s} \mid s = 1, \ldots, k-1, k+1, \ldots, n\}$$

  of elements of the Γ–set Q.

An **elementary derivative unary operation of the type** $(i; j)$ **on the** Γ–**set** Q is a mapping

$$u : Q_i \to Q_j$$

such that for every $x \in Q_i$

$$u(x) = \delta_\sigma(q_1, \ldots, q_{k-1}, x, q_{k+1}, \ldots, q_n).$$

The fact that u is of the type $(i; j)$, is denoted by the symbol $T(u) = (i; j)$.

It is clear that if

- $R = \{R_i \mid i \in \Gamma\}$ is any congruence on the Σ–algebra (Q, δ),
- q, q' are elements of the domain Q_i, such that $(q, q') \in R_i$,
- and $u : Q_i \to Q_j$ is an elementary derivative unary operation,

then $(u(q), u(q')) \in R_j$.

Let $i, j \in \Gamma$.

A **derivative unary operation** of the type $(i; j)$ on the Σ–algebra (Q, δ) is

- either identical mapping $id_{Q_i} : Q_i \to Q_i$ (in this case the equality $i = j$ must be hold),

- or a composition $u_n \circ \ldots \circ u_1$ of a n–tuple of elementary derivative unary operations on (Q, δ), such that
  - $T(u_1) = (i, i_1)$,
  - $T(u_2) = (i_1, i_2)$,
  - $\ldots$
  - $T(u_n) = (i_{n-1}, j)$.

The set of all derivative unary operations of the type $(i; j)$ on (Q, δ) will be denoted by the symbol

$$DUO(Q, \delta)_{(i;j)}.$$

It is not so difficult to prove (by induction) that if

- $R = \{R_i \mid i \in \Gamma\}$ is any congruence on the Σ–algebra (Q, δ),
- q, q' are elements of the set Q_i, such that $(q, q') \in R_i$,
- and $u \in DUO(Q, \delta)_{i,j}$,

then $(u(q), u(q')) \in R_j$.

5.2.2 Definition of the congruence R^f on the Σ–algebra (Q, δ)

The congruence R^f on the Σ–algebra (Q, δ) is the Γ–tuple

$$R^f \stackrel{\text{def}}{=} \{R_i^f \mid i \in \Gamma\} :$$

for every $i \in \Gamma$
$R_i^f \stackrel{\text{def}}{=} \{(q, q') \in Q_i \times Q_i \mid \forall j \in \Gamma,\ \forall u \in DUO(Q, \delta)_{(i;j)}$
$(f_j \circ u)(q) = (f_j \circ u)(q')\}$.

It is clear that $\forall i \in \Gamma$ R_i^f is an equivalency relation, and $R^f \leq Ker(f)$.

Theorem.

R^f is a congruence on the Σ–algebra (Q, δ).

Proof:

It is necessary to prove that for

- every $\sigma \in \Sigma$, such that $T(\sigma)$ is of the form $(i_1, \ldots, i_n;\ j)$, where $n \geq 1$,
- and every n–tuple of pairs of the form

$$(q_1, q_1') \in R^f_{i_1}, \ldots, (q_n, q_n') \in R^f_{i_n}$$

the pair $(\delta_\sigma(q_1, \ldots, q_n), \delta_\sigma(q_1', \ldots, q_n'))$ belongs to R^f_j, i.e. that for

- every $k \in \Gamma$
- and every $u \in DUO(Q, \delta)_{(j;k)}$

$(f_k \circ u)(\delta_\sigma(q_1, \ldots, q_n)) = (f_k \circ u)(\delta_\sigma(q_1', \ldots, q_n'))$.

Define the following n–tuple of elementary derivative unary operations:

- $u_1 \in DUO(Q, \delta)_{(i_1;j)}$, $u_1 : x \mapsto \delta_\sigma(x, q_2, q_3, \ldots, q_n)$,
- $u_2 \in DUO(Q, \delta)_{(i_2;j)}$, $u_2 : x \mapsto \delta_\sigma(q_1', x, q_3, \ldots, q_n)$,
- $\ldots$
- $u_n \in DUO(Q, \delta)_{(i_n;j)}$, $u_n : x \mapsto \delta_\sigma(q_1', \ldots, q_{n-1}', x)$.

We have:

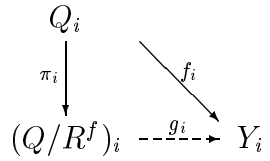
$(f_k \circ u)(\delta_\sigma(q_1, \ldots, q_n)) =$
$= (f_k \circ u)(u_1(q_1)) = (f_k \circ (u \circ u_1))(q_1) = \quad ((q_1, q_1') \in R^f_{i_1})$
$= (f_k \circ (u \circ u_1))(q_1') = (f_k \circ u)(u_1(q_1')) =$
$= (f_k \circ u)(\delta_\sigma(q_1', q_2, q_3, \ldots, q_n)) =$
$= (f_k \circ u)(u_2(q_2)) = (f_k \circ (u \circ u_2))(q_2) = \quad ((q_2, q_2') \in R^f_{i_2})$
$= (f_k \circ (u \circ u_2))(q_2') = (f_k \circ u)(u_2(q_2')) =$
$= (f_k \circ u)(\delta_\sigma(q_1', q_2', q_3, \ldots, q_n)) = \ldots =$
$= (f_k \circ u)(\delta_\sigma(q_1', \ldots, q_n'))$.

■

5.2.3 Properties of the congruence R^f

As it was stated before, there exists a unique Σ–tuple $\delta^f \stackrel{\text{def}}{=} \{\delta^f_\sigma \mid \sigma \in \Sigma\}$ of algebraic operations of corresponding types on the Γ–set Q/R^f, such that the Γ–tuple π of canonical projections is a morphism from (Q, δ) to $(Q/R^f, \delta^f)$.

Also it is clear that $R^f \leq Ker(f)$, and this inequality implies existence of a Γ–tuple $g = \{g_i \mid i \in \Gamma\}$, such that the following diagrams are commutative for every $i \in \Gamma$:



Theorem.

R^f is a greatest congruence (with respect to the above relation of partial order) among all congruences R on (Q, δ) which satisfy the condition $R \leq Ker(f)$.

Proof:

Let R be a congruence on (Q, δ) such that $R \leq Ker(f)$.

Prove that $R \subseteq R^f$, i.e. prove that if an element $i \in \Gamma$ and a pair $q, q' \in Q_i$ are such that $(q, q') \in R_i$, then

- for every $j \in \Gamma$
- and every $u \in DUO(Q, \delta)_{(i;j)}$

$(f_j \circ u)(q) = (f_j \circ u)(q')$.

As stated above, the formula $(q, q') \in R_i$ implies the formula $(u(q), u(q')) \in R_j$ for every $j \in \Gamma$ and every $u \in DUO(Q, \delta)_{(i;j)}$.

The formula

$$(u(q), u(q')) \in R_j$$

and the inequality

$$R \leq Ker(f)$$

imply the desired equality

$$(f_j \circ u)(q) = (f_j \circ u)(q').$$

■

5.3 Construction of a minimal Σ–tree automaton for a given Σ–tree class

Let there is given a class $C \subseteq Tr(\Sigma)$, i.e. $C = \{C_i \mid i \in \Gamma,\ c_i \subseteq Tr(\Sigma)_i\}$.

Let $Y = \{Y_i \mid i \in \Gamma\}$ be a Γ–set, such that $\forall i \in \Gamma$ the set Y_i consists of two elements: $Y_i = \{0_i, 1_i\}$.

Define the following Γ–mapping $f : Tr(\Sigma) \to Y$:

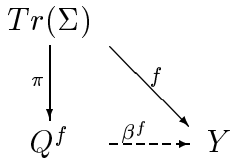
$$\forall i \in \Gamma,\ \forall E \in Tr(\Sigma)_i \quad f_i(E) \stackrel{\text{def}}{=} \begin{cases} 1_i, & \text{if} E \in C_i, \\ 0_i, & \text{if} E \notin C_i, \end{cases}$$

Let $R^f = \{R_i^f \mid i \in \Gamma\}$ be a greatest congruence on the Σ–algebra $(Tr(\Sigma), \delta)$ with the property $R^f \leq Ker(f)$, and (Q^f, δ^f) be a factor-algebra of $(Tr(\Sigma), \delta)$ corresponded to this congruence, i.e. a Σ–algebra such that

- Q^f is the Γ–set of classes of the congruence R^f on $(Tr(\Sigma), \delta)$,
- δ^f is a unique Σ–tuple of algebraic operations of corresponding types on the Γ–set Q^f, such that the Γ–tuple $\pi = \{\pi_i : Tr(\Sigma)_i \twoheadrightarrow Q_i^f \mid i \in \Gamma\}$ of canonical projections is a morphism from $(Tr(\Sigma), \delta)$ to (Q^f, δ^f).

The required minimal Σ–tree automaton M^f is the pair $M^f \stackrel{\text{def}}{=} ((Q^f, \delta^f), \beta^f)$, where

- (Q^f, δ^f) is the above Σ–algebra,
- β^f is (a unique) Γ–mapping such that the diagram



  is commutative.

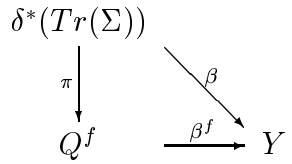
It is not so difficult to prove that

- $C(M^f) = C$,

- for every Σ–automaton $M = ((Q, \delta), \beta)$, which represents the same class C, there exists a surjective morphism π of Σ–algebras of the form

$$\pi : (\delta^*(Tr(\Sigma)), \delta) \twoheadrightarrow (Q^f, \delta^f),$$

and this epimorphism makes the following diagram be commutative:



In particular, if the Σ–automaton $M^f = ((Q^f, \delta^f), \beta^f)$ is such that the set Q^f is finite, then for every Σ–automaton $M = ((Q, \delta), \beta)$, which represents the same class C and is such that the set Q is also finite, the following inequality holds:

$$|Q^f| \leq |Q|,$$

i.e. in this case M^f has minimal possible number of states.

6 Conclusion

In the paper is delivered a generalization of the representation of classes of trees by automata. We have introduced a concept of a typed tree automaton (Σ–tree automaton), and generalized the theorem about minimal tree automata related to tree classes.

Representation of Σ–tree classes by typed automata

- displays essential aspects of the structure of the class which it represents (like topological invariants of objects of the class),
- provides an efficient distributed implementation of a recognition process.

In the following papers we will generalize the proposed constructions on the case of fuzzy classes of Σ–trees. The problem of representation of fuzzy classes of trees by fuzzy tree automata still not was considered in the research

works on automata theory even for untyped case. We will show that the solution of the problem of construction of canonical fuzzy Σ-tree automaton for representation of a class of fuzzy Σ–trees can be obtained using topos theory and general theory of systems in categories of Anderson B.D.O., Arbib M.A. and Manes E.G. (see [3]). We will introduce the concept of a fuzzy distributed agent and will construct ones for distributed implementation of fuzzy Σ-tree automata.

References

[1] **M.Abadi, L. Cardelli, B.Pierce, G.Plotkin:** Dynamic typing in a statically typed language, *Transactions on programming languages and systems, 13(2): 237-268, April 1991.*

[2] **M. Abadi, L. Cardelli, B.Pierce, D.Remy:** Dynamic typing in polymorphic languages, *Journal of functional programming, 5(1):111-13-, January 1995.*

[3] **Anderson B.D.O., Arbib M.A., Manes E.G.**: Foundations of system theory : finitary and infinitary conditions, *Lecture Notes in Economics and Mathematical Systems,* **115,** *Berlin-Heidelberg-New York, Springer-Verlag* (1976).

[4] **M.A. Arbib and E.G. Manes:** Tree transformations and semantics of loop-free programs. *Acta Cybernetica, 4:11-17, 1978.*

[5] **B.S. Baker:** Generalized syntax directed translation, tree transducers, and linear space. *Journal of Comput. and Syst. Sci., 7:876-891, 1978.*

[6] **H.Barendregt:** Introduction to generalized type systems, *Journal of Functional programming, 1992.*

[7] **Walter S. Brainerd:** The minimalization of tree automata. *Information and Control, 13(5):484-491, November 1968.*

[8] **L. Cardelli:** Type systems, *In Allen B. Tucker, editor: Handbook of Computer Science and Engineering, CRC Press, 1996*

[9] **J.L. Coquide, M. Dauchet, R. Gilleron, and S. Vagvolgyi:** Bottom-up tree pushdown automata : Classification and connection with rewrite systems. *Theorical Computer Science, 127:69-98, 1994.*

[10] **B. Courcelle:** On Recognizable Sets and Tree Automata, chapter Resolution of Equations in Algebraic Structures. *Academic Press, m. Nivat and Ait-Kaci edition, 1989.*

[11] **L. Damas, R.Milner:** Principal type schemes for functional programs. *In Proceedings of the 9th ACM Symposium on Principles of Programming Languages, pages 207-212, 1982.*

[12] **M. Dauchet:** Rewriting and tree automata. *In H. Comon and J.-P. Jouannaud, editors, Proc. Spring School on Theoretical Computer Science: Rewriting, Lecture Notes in Computer Science, Odeillo, France, 1994. Springer Verlag.*

[13] **J. E. Doner:** Tree acceptors and some of their applications. *Journal of Comput. and Syst. Sci., 4:406-451, 1970.*

[14] **S. Eilenberg and J. B. Wright:** Automata in general algebras. *Information and Control, 11(4):452-470, 1967.*

[15] **K.S.Fu:** Syntactic Pattern Recognition and Applications. *Prentice Hall, Englewood Cliffs, NJ, 1982.*

[16] **J.H. Gallier and R.V. Book:** Reductions in tree replacement systems. *Theorical Computer Science, 37(2):123-150, 1985.*

[17] **F. Gécseg and M. Steinby:** Tree Automata. *Akademiai Kiado, 1984.*

[18] **F. Gécseg and M. Steinby:** Tree languages. *In G. Rozenberg and A. Salomaa, editors, Handbook of Formal Languages, volume 3, pages 1-68. Springer Verlag, 1996.*

[19] **R. Gilleron and S. Tison:** Regular tree languages and rewrite systems. *Fundamenta Informaticae, 24:157-176, 1995.*

[20] **M. Hofmann, B.Pierce:** A unifying type-theoretic framework for objects *Journal of Functional programming, 5(4): 593-635, October 1995.*

[21] **D. Kozen:** On the Myhill-Nerode theorem for trees. *Bulletin of the European Association of Theoretical Computer Science, 47:170-173, June 1992.*

[22] **X. Leroy:** Manifest types, modules and separate compilation. *In Conference record of POPL'94: 21st ACM SIGPLAN-SIGACT Symposium on principles of programming languages, pages 109-122, Portland, OR, January 1994.*

[23] **X.Leroy:** A syntactic theory of type generativity and sharing, *Journal of functional programming, 6(5): 667-698, September 1996.*

[24] **Denis Lugiez and Jean-Luc Moysset:** Tree automata help one to solve equational formulae in ac-theories. *Journal of Symbolic Computation, 18(4):297-318, 1994.*

[25] **R.Milner:** A theory of type polymorphism in programming, *Journal of Computer and system sciences, 17: 348-375, August 1978.*

[26] **D.Muller, A.Saoudi and P, Schupp:** Weak alternating automata give a simple explanation of why most temporal and dynamic logic are decidable in exponential time. *Proc. of the 3rd Annual Symp. on Logic in Computer Science, July, 1988.*

[27] **M. Nivat and A. Podelski:** Resolution of Equations in Algebraic Structures, volume 1, chapter Tree monoids and recognizable sets of finite trees, *pages 351-367. Academic Press, New York, 1989.*

[28] **B.Pierce, D.Turner:** Simple type-theoretic foundations for object-oriented programming, *Journal of functional programming, 4(2): 207-247, April 1994.*

[29] **B.I.Plotkin:** Universal algebra, algebraic logic and databases, *1992.*

[30] **M.O. Rabin:** Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of the American Mathematical Society, 141:1-35, 1969.*

[31] **J.-C. Raoult:** A survey of tree transductions. *In M. Nivat and A. Podelski, editors, Tree Automata and Languages, pages 311-325. Elsevier Science, 1992.*

[32] **K. Salomaa:** Synchronized tree automata. *Theorical Computer Science, 127:25-51, 1994.*

[33] **G. Slutzki:** Alternating tree automata. *Theorical Computer Science, 41:305-318, 1985.*

[34] **G. Slutzki and S. Vagvolgyi:** Deterministic top-down tree transducers with iterated look-ahead. *Theorical Computer Science, 143:285-308, 1995.*

[35] **J.W. Thatcher:** Tree automata: an informal survey. *In A.V. Aho, editor, Currents in the theory of computing, pages 143-178. Prentice Hall, 1973.*

[36] **W. Thomas:** Handbook of Theoretical Computer Science, volume B, chapter Automata on Infinite Objects, *pages 134-191. Elsevier, 1990.*