5th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence and Informatics
January 25-26, 2007 ▒ Poprad, Slovakia

Tree Automata in the Mathematical Theory

Viliam Slodičák, Valerie Novitzká, Daniel Mihályi

Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics
Technical University of Košice
Letná 9/A, 040 01 Košice, Slovakia
viliam.slodicak@tuke.sk, valerie.novitzka@tuke.sk, daniel.mihalyi@tuke.sk

Abstract: The aim of our research is mathematical description of program development process as logical reasoning using proposition-as-types paradigm. For describing a computation process in programs we use reasoning in intuitionistic linear logic, which is powerful means for describing changes of states in running programs and it makes possible the application of Curry-Howard corespondence formulating proofs-as-expressions paradigm. This mechanism allows us to describe construction of programs as construction of proofs. Another useful structures that enable to describe evaluation of expressions are tree automata. In our contribution we discuss these two approaches and show the relations between them.

Keywords: linear logic, tree automaton, term computation, category theory, linear type theory

1 Introduction

To solve large scientific problems by mathematical machines (i.e. computers) we always start with the formulation of their theoretical foundations. We need to formalize these theoretical foundations as logical reasoning in some several mathematical theories because the program should really prove the correctness of their results.

The aim of our research is to formulate theoretical foundations of program development process. Our idea is that the running program can be described as proofs in sequent calculus of intuistionic linear logic introduced by Girard [1]. Precisely, the reductions of linear terms corresponding to proofs in the intuitionistic linear logic can be regarded as program computations. The most useful feature of linear logic is that linear formulae are considered as actions. While classical and intuitionistic logic treat with the sentences that are always true or false, the truth values in linear logic depend on an internal state of dynamic system.

Second part of our contribution deals with theory of tree automata. Tree automata are devices which handle labelled trees analogously as sequential automata handle sequences (words) of input symbols. Automata are also powerful tools for describing actions. The 'action' of automaton consists of taking any *n*-ary tree with leaves labelled by variables or nullary symbols, computing the tree and giving an output.

In this contribution we want to find correspondence between type system of linear logic and the tree automata as a morphism.

2 Linear Logic

Linear logic was introduced by Jean-Yves Girard in 1987 and it becames a natural mean for research and applications in computer science [1]. It is able to describe systems that are changed during they are used. Applications of linear logic cover the theory of concurrent processes, logic programming and functional programming.

In mathematical logic, linear logic is a type of substructural logic that denies the structural rules of weakening and contraction (it allows only restricted versions of that rules). The interpretation in linear logic is of hypotheses as resources: every hypothesis must be consumed exactly once in a proof. Formulae in linear logic describe *actions*. This differs from usual logics where the governing judgement is of truth, which may be freely used as many times as necessary. While classical and intuitionistic logic treat with sentences that are always true or false, in linear logic truth values depend on the internal state of dynamic system. Classical and intuitionistic logic are included in linear logic.

In our contribution we consider intuitionistic linear logic (in the following denoted by ILL) because our aim is to describe a construction of computable solution (execution of program) as a proof in ILL. [4]

3 Intuitionistic Linear Logic and its Model

A formula of ILL is of the form $\phi(s)$ where ϕ is a predicate applied on a linear term s of type σ. A sequent of ILL has a form $\Theta \mid -\phi(s)$ where Θ is a finite set of linear formulae. ILL has the important *variable balancing* property: every variable occuring in Θ has exactly one occurrence each side of a sequent. This property is essential for the interpretation of sequents.

Intuitionistic linear logic over the linear type theory $LinCl(\Sigma)$ can be constructed as categorical structure by the syntactic fibration l (left side of diagram in Fig. 1). There are many models of intuitionistic linear logic .In our previous works we constructed model as a morphism of fibrations $(i, j, [\![-]\!])$. Next figure shows the categorical model of linear logic as a morphism of fibrations to symmetric monoidal closed categories as it was constructed in [4].

$$\begin{array}{ccc} \mathcal{LL}(\Sigma, \mathcal{E}) & \xrightarrow{[\![-]\!]} & (\mathbf{L}, \otimes, \mathbf{1}, \multimap) \\ \Big\downarrow l & & \Big\downarrow z \\ LinCl(\Sigma) & \xrightarrow{(i,j)} & (\mathbf{C}, \bullet, I, hom(-,-)) \end{array}$$

Figure 1
Constructed categorical model of linear logic

In Figure 1, $LL(\Sigma, \varepsilon)$ is a category containing ILL over signature Σ with axioms in ε, $LinCl(\Sigma)$ is the classifying category over a signature Σ, $(L,\otimes,1,-o)$ is symmetric monoidal closed category containing ILL and $(C,\bullet,I,\hom(-,-))$ is symmetric monoidal closed category of linear type theory.

4 Trees

In computer science, a tree is a widely-used data structure that emulates a tree structure with a set of linked nodes. It is a special case of a graph. Each node has zero or more child nodes, which are below it in the tree (by convention, trees grow down). A node that has a child is called the child's parent node (or ancestor node, or superior). A node has at most one parent.

4.1 Terms and Trees

We denote by N the set of positive integers. We denote set of finite strings over N by N^*. The empty string is denoted ε. A *ranked alphabet* is a couple $\Sigma = (F, Arity)$, where F is a finite set of function symbols and $Arity$ is a mapping from F into N. The arity of a symbol $\sigma \in F$ is $Arity(\sigma)$. The set of symbols of arity p is denoted F_p. Elements of arity $0,1,\ldots p$ are respectively

called constants, unary, …, p - ary symbols. We assume that F contains at least one constant.

Let X be a set of constants called *variables.* We assume that the sets X and F_0 are disjoint. The set $T(F,X)$ of *terms* over ranked alphabet Σ and set of variables X is the smallest set defined by:

$-F_0 \subseteq T(F,X)$ and
$-X \subseteq T(F,X)$ and
$-p \geq 1, \sigma \in F_p$ and $t_1,\ldots,t_p \in T(F,X)$, then $\sigma(t_1,\ldots,t_p) \in T(F,X)$

A term t in $T(F,X)$ is *linear* if each variable occurs at most once in t.

A finite ordered *tree* t over a set of labels E is a mapping from a prefix-closed set $Pos(t) \subseteq N^*$ into E. Thus, a term $t \in T(F,X)$ may be viewed as a finite ordered ranked tree, the leaves of which are labeled with variables or constant symbols and the internal nodes are labeled with symbols and the internal nodes are labeled with symbols of positive arity, with out-degree equal to the arity of the label. So term $t \in T(F,X)$ can also be defined as a partial function $t : N^* \to F \cup X$ with domain $Pos(t)$ satisfying the following properties:

(1) $Pos(t)$ is nonempty and prefix - closed.
(2) $\forall p \in Pos(t)$, if $t(p) \in F_n, n \geq 1$, then $\{j \mid pj \in Pos(t)\} = \{1,\ldots,n\}$.
(3) $\forall p \in Pos(t)$, if $t(p) \in X \cup F_0$, then $\{j \mid pj \in Pos(t)\} = \emptyset$.

Each element of $Pos(t)$ is called a *position.* A *frontier position* is a position p such that $\forall j \in N, pj \notin Pos(t)$. Each position p in t such that $t(p) \in X$ is called a *variable position*. We denote by $Head(t)$ the *root symbol* of t which is defined by $Head(t) = t(\varepsilon)$.

4.2 Subterms. Substitutions

A *subterm* $t|_p$ of a term $t \in T(F,X)$ at position p is defined by the following:

$-Pos(t|_p) = \{j \mid pj \in Pos(t)\}$,
$-\forall q \in Pos(t|_p), t|_p(q) = t(pq)$

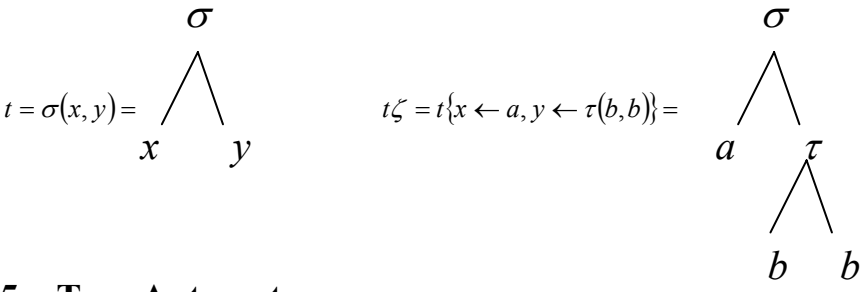
We denote by $t[u]_p$ the term obtained by replacing in t the subterm $t|_p$ by u.

A *substitution* ζ is a mapping from X into $T(F,X)$. The *domain* of a substitution ζ is the subset of variables $x \in X$ such that $\zeta(x) \neq x$. The substitution $\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ is the identity on $X \setminus \{x_1, \ldots, x_n\}$ and maps $x_i \in X$ on $t_i \in T(F,X)$, for every index $1 \leq i \leq n$. Substitutions can be extended to $T(F,X)$ in such a way:

$$\forall \sigma \in F_n, \forall t_1, \ldots, t_m \in T(F,X). \zeta(\sigma(t_1, \ldots, t_n)) = \zeta(\sigma(t_1) \ldots, \sigma(t_n)).$$

Substitution will be used in postfix notation: $t\zeta$ is the result of applying ζ to a term t.

Example
Let $F = \{\sigma(,), \tau(,), a, b\}$ and $X = \{x, y\}$. Let us consider term $t = \sigma(x, y)$. Let us consider the substitution $\zeta = \{x \leftarrow a, y \leftarrow \tau(b,b)\}$.
Then



5 Tree Automata

Tree automata are devices which handle labelled trees analogously as sequential automata handle sequences (words) of input symbols. The internal structure of a sequential automaton is an unary algebra; for a tree automaton it is an algebra of an arbitrary type. [2] As *finitary type* we usually take a ranked alphabet – finitary set of operations and the arity map.

Definition. A Σ *-tree automaton* is a sixtuple $A = (Q, \{\delta_\sigma\}_{\sigma \in \Sigma}, \Gamma, \gamma, I, \lambda)$ where:

Q is a set, called the *set of states*;

$\delta_\sigma : Q^n \to Q$ ($\sigma \in F$, $Arity(\sigma) = n$) are operations on Q;

Γ is a set, called the *output alphabet*;

$\gamma : Q \to \Gamma$ is a map, called the *output map*;

I is a set, called the *set of variables*;

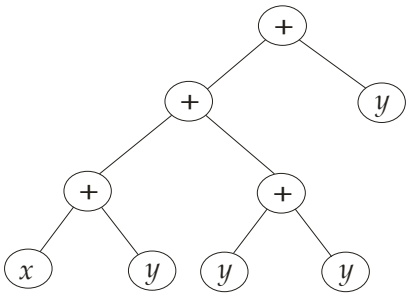
$\lambda : I \to Q$ is a map, called the *initialization*.

Example. Let $\Sigma : F = F_2 = \{+\}$ and let $(Z,+)$ be the additive Σ-algebra of integers. Put $\Gamma = \{0,1\}$, and let γ be the parity map:

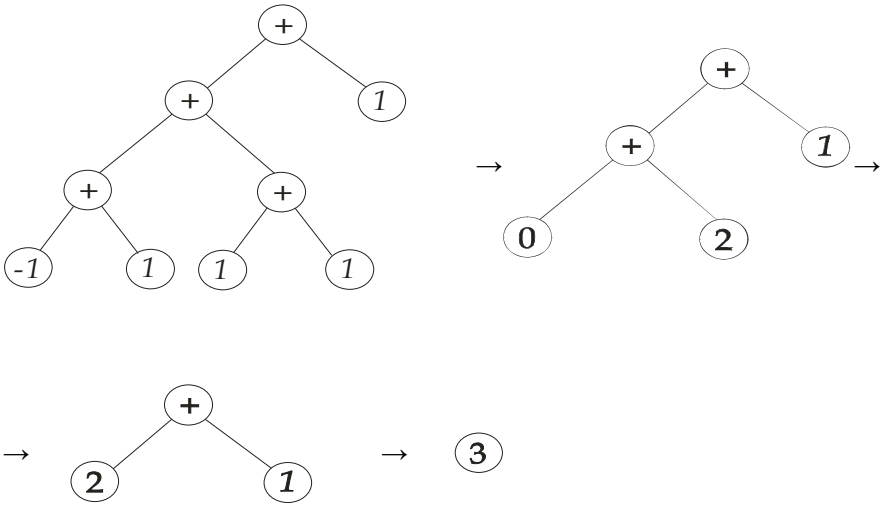
$\gamma(z) = 0$ if z is odd and $\gamma(z) = 1$ if z is even.

Then we have a Σ-tree automaton $A = (Z, \{+\}, \Gamma, \gamma, \{x, y\}, \lambda)$ with initial assignment $\lambda(x) = -1$ and $\lambda(y) = 1$.

The 'action' of this automaton consists of taking any binary tree with leaves by x and y, computing the tree and giving an output [2,3]. For example, the following tree



has the following computational sequence:



The result of computation of the tree is 3, resulting output is $\gamma(3) = 1$.

The external behavior of the automaton A is expressed by the map β assigning to each of these trees t the value $\beta(t)$ from Γ of the output which results after the computation of t.

We denote set of all Σ-trees over I by $I^{\#}$. This set carries a natural structure of a Σ-algebra. For each $\sigma \in \Sigma$ we have the operation $\varphi_\sigma : (I^{\#})^n \to I^{\#}$ of the tree-tupling: given trees $t_0, \ldots, t_{n-1} \in I^{\#}$ we form the following tree $t = \varphi_\sigma(t_0, \ldots, t_{n-1})$:

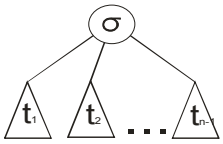


Figure 2
Σ-tree

The algebra of finite trees $(I^{\#}, \{\varphi_\sigma\}_{\sigma\in\Sigma})$ is the free Σ-algebra generated by the set I. For the Σ-tree automaton $A = (Q, \{\delta_\sigma\}_{\sigma\in\Sigma}, \Gamma, \gamma, I, \lambda)$ the unique homomorphism $\rho : (I^{\#}, \{\varphi_\sigma\}_{\sigma\in\Sigma}) \to (Q, \{\delta_\sigma\})$ extending the initialization map λ is called the *run map* of the automaton A. the map $\beta = \rho \circ \gamma : I^{\#} \to \Gamma$ is called the behavior of the automaton A. For each tree t, the result of the computation of t (after interpreting the variables x as the states $\lambda(x)$) is the state $\rho(t)$ and the resulting output is $\beta(t)$.

6 Category of Trees

For studying relations between linear logic and tree automata we need represent automata and trees in a categories. Category is a structure which contains objects (*A, B, C, ...*), morphisms between objects ($f : A \to B$), identity morphism for any object of category ($id_X : X \to X$). Morphisms are composable (if $f : A \to B$ and $g : B \to C$ then $g \circ f : A \to C$).

Definition. A category of trees, denoted ***Tree*** consists of:

- trees as objects: $t = \sigma(t_1, \cdots, t_{n-1})$, $t' = \vartheta(t'_1, \cdots, t'_{m-1})$

- computations with substitutions on required positions as morphisms:

$\rho : t \to t'$ where $t = \sigma(t_1, \ldots, t_{n-1})$ and $t' = \sigma(t'_1 = \rho(t_1), \ldots, t'_{n-1} = \rho(t_{n-1}))$

- for every tree there is an identity morphism: $id_t : t \to t$

- if $\rho : t_1 \to t_2$ and $\theta : t_2 \to t_3$ then $\theta \circ \rho : t_1 \to t_3$: a composition of morphisms.

Example. Let us consider the tree given by the term $t = +(t_1, t_2) = +(+(2,3), +(3,4))$. The subtrees of t are: $t_1 = +(2,3)$ and $t_2 = +(3,4)$.

A morphism between trees we can illustrate by following computation and substitution: $t_1 = +(2,3) \to t'_1 = (5)$ and $t_2 = +(3,4) \to t'_2 = (7)$ so the new tree t' (as an image of t given by morphism $\rho_1 : t \to t'$) has a form $t' = +(5,7)$. Next step is to compute the tree t', this is done by morphism $\rho_2 : t' \to t''$, where $t'' = (12)$. We can see that composition $\rho_2 \circ \rho_1 : t \to t''$ is done by $+(+(2,3),+(3,4)) \to +(2,3) \circ +(3,4) \to (12)$.

7 Tree Automata in a Category

The category of sets, denoted ***Set***, has sets as objects and functions between sets as morphisms. Now we can represent Σ-tree automata. Assume, for simplicity, that Σ has just one binary operation. Then Σ-tree automaton is a diagram as in Fig. 3 [2]:

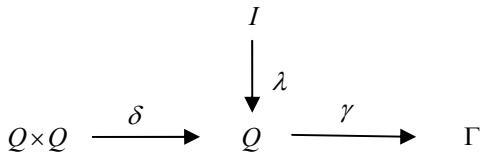


Figure 3
The diagram of Σ -tree automaton in a category of sets

The sets of an automaton - Q, I, Γ and also $Q^n, n-$arity are objects and functions δ, λ, γ are morphisms of category ***Set*** each for some automaton. Category of sets representing the tree automata we denote ***TAESet*** (Tree Automata Encapsulated in ***Set***). Our goal is to assign to any tree the equivalent automaton which is able to make computations with this tree. We will construct this assignation as a functor. The next step is formulating the relation between linear type theory and tree automata.

8 The Relation between Linear Type Theory and Tree Automata

In previous work [4] we showed how to interpret linear type theory in symmetric monoidal closed category (SMCC) and we constructed categorical model of linear logic as a morphism of fibrations to symmetric monoidal closed categories. There are also many approaches that use another representation for type theory, e.g. in

[5, 6] but they are more pragmatical than our theoretical approach. Now we want to construct relation between linear type theory and tree automata.

We assume linear classifying category over a signature *Sig* defined in [4]. A signature *Sig* consists of set *T* of linear types and set *F* of function symbols. To interpret linear types and function symbols we define couple of morphisms (i, j):

$$i : T \to Obj(TAESet)$$
$$j : F \to Morph(TAESet)$$

by

$$i(\tau_1 \times \tau_2) = i(\tau_1) \times i(\tau_2)$$
$$i([\tau_1 \to \tau_2]) = \hom(i(\tau_1), i(\tau_2))$$

for $f : \tau_1, \ldots, \tau_n \to \tau$ then $j(f) : i(\tau_1), \ldots, i(\tau_n) \to i(\tau)$. We consider the objects $i(\tau_1), \ldots i(\tau_n)$ as a sets of category ***TAESet***: to every type is assigned set of states of automaton making computations over assigned type. We can write:

$$i(\tau) = Q$$
$$i(\tau \times \tau) = Q \times Q$$
$$i([\tau \times \tau \to \tau]) = \hom(Q \times Q, Q) = \{\delta\}, \text{where } i(\tau) = Q$$

The relation between linear type theory and tree automata is given by functor (i, j):

$$LinCl(\Sigma) \xrightarrow{(i,j)} TAESet$$

Figure 4
Morphism between linear type theory and category of tree automata

Now we define morphism between categories ***Tree*** and ***TAESet***. We need to assign to every term an automaton which is able to compute this term. An operation of initial substituion in terms $\zeta : X \to T(F, X)$ is equivalent to initialization of automaton, mapping $\lambda : I \to Q$. To every tree is assigned corresponding set of states of the automaton. To any morphism between trees is assigned an operation $\delta : Q^n \to Q$ between states. Identity morphism on any tree has a corresponding identity morphism on equivalent set of states. Composition of computations has a corresponding composition of operations of the tree automaton. So we have a morphism (φ, ψ) between categories ***Tree*** and ***TAESet***.

$$\varphi : Obj(Tree) \to Obj(TAESet)$$
$$\psi : Morph(Tree) \to Morph(TAESet)$$

Finally we construct diagram showing relations between linear type theory, tree automata and trees.

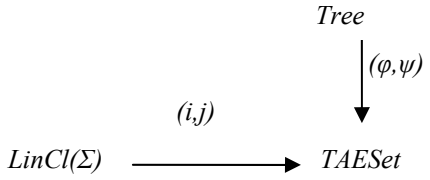


Figure 5
Morphisms between linear type theory, tree automata and trees

Conclusion

In this contribution we showed how to interpret trees in a category of trees and formulated a relation between linear type theory and tree automata computing over trees. Our next goal is to construct a functor expressing the relation between tree automata and our symmetric monoidal closed category of intuitionistic linear logic.

Acknowledgement

This work was supported by VEGA Grant No.1/2181/05: Mathematical Theory of Programming and its Application in the Methods of Stochastic Programming.

References

[1] Girard, J.-Y.: Linear Logic: Its Syntax and Semantics, Theoretical Computer Science 50, 1987, pp. 1-102

[2] Adámek, J., Trnková, V.: Automata and Algebras in Categories, Kluwer Academic Publishers, Prague, 1990, pp. 1-473

[3] Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications, free book, Sept 2005, pp. 1-222

[4] Novitzká, V., Mihályi, D., Slodičák, V.: Categorical Models of Logical Systems in the Mathematical Theory of Programming, 6th Joint Conference on Mathematics and Computer Science, Pécs, Hungary, July 12-15, 2000

[5] Porkoláb, Z., Zólyomi, I.: A Feature Composition Problem and a Solution Based on C++ Template Metaprogramming, Generative and Transformational Techniques in Software Engineering, Lecture Notes in Computer Science, Vol. 4143, Springer-Verlag, 2006, pp. 459-470

[6] Zólyomi, I., Porkoláb, Z., Kozsik, T.: An Extension to the Subtype Relationship in C++ Implemented with Template Metaprogramming, Generative Programming and Component Engineering, Lecture Notes in Computer Science, Vol. 2830, Springer-Verlag, 2003, pp. 209-227