Finite-state Transducers as Regular Böhm Trees

Gérard Huet and Henri Laulhère

INRIA, Rocquencourt, BP 105, 78153 Le Chesnay Cedex, France

Abstract. We present a uniform translation from finite-state transducers to regular Böhm trees presentations. The corresponding Böhm tree represents directly the trace semantics of all finite and infinite behaviours of the given transducer. We consider variations on this basic idea, and generalisations of finite-state transducers suggested by the general formalism of regular Böhm trees. This work suggests the use of recursive Böhm trees combinators as a machine-language for reactive programming.

Introduction

We give a uniform translation from finite-state transducers to regular Böhm trees. We assume familiarity with the formalism of recursive Böhm trees combinators as introduced by Huet in [3]. Roughly speaking, Böhm trees are the completion of normal forms in the λ-calculus into infinite trees. Regular Böhm trees are those which may be defined by finite families of mutually recursive combinators of a simple kind.

1 Finite-state transducers

We assume a deterministic finite-state transducer described as follows. Its input alphabet is $\mathcal{A} = \{a_1, ..., a_n\}$. Its output alphabet is $\mathcal{B} = \{b_1, ..., b_p\}$. Its set of states is $\mathcal{S} = \{S_1, ..., S_N\}$. Its set of transitions is described by a set of transition rules $\mathcal{T} = \{T_{S,a} | S \in \mathcal{S}, a \in \mathcal{A}\}$ with $T_{S,a} = (S', \mathbf{b})$ where $S' \in \mathcal{S}$ and $\mathbf{b} \in \mathcal{B}^*$. Meaning "In state S, reading input a, the transducer outputs $\mathbf{b}$ and goes in state S'." A deterministic transducer is defined by the quadruple of the input alphabet, the output alphabet, the states and the transition rules : $\langle \mathcal{A}, \mathcal{B}, \mathcal{S}, \mathcal{T} \rangle$. Actually this quadruple defines a family of transducers indexed by $\mathcal{S}$. The index corresponds to the initial state of one transducer in the family.

2 Translation to a Böhm tree presentation

2.1 The encoding

We associate a *Böhm tree presentation* ε to the transducer. This notion, defined in [3], presents forests of Böhm trees as families of recursive combinator definitions of a simple kind. In this particular case, the set of combinators is $\mathcal{S}$ plus a set of auxiliary combinators $\mathcal{T}$ containing m members for each rule in $\mathcal{R}$ whose output word $\mathbf{b}$ is of length m. For every state S, we write a set of combinator definitions as follows:

For each input letter $a \in \mathcal{A}$ and state $S \in \mathcal{S}$, let $T_{S,a} = (S', \mathbf{b})$ with $\mathbf{b} = b_1 \ldots b_m$. We introduce m combinators $T_{S,a,j}$ with $1 \leq j \leq m$. Let $u, v_1, \ldots v_p$ be distinct variables. We define terms $M_{a,j}$ with $1 \leq j \leq m+1$ as follows:

$$M_{a,m+1} \text{ is } S'(v_1, \ldots, v_p)$$

$$M_{a,j} \text{ with } 1 \leq j < m \text{ is } T_{S,a,j}(v_1, \ldots, v_p)$$

and we write the m definitions:

$$T_{S,a,j}\ v_1 \ldots\ v_p := v_{b_j}(M_{a,j+1}) \text{ with } 1 \leq j < m.$$

Let N_a be $M_{a,1}$. We finally write the definition for S as

$$S\ v_1 \ldots\ v_p\ u := u(N_1, \ldots, N_n) \text{ with } N_i = N_{a_i}$$

where n and p are respectively the cardinality of the input alphabet $\mathcal{A}$ and the output alphabet $\mathcal{B}$. Variable u represents the input of the transducer, while v_i represents the ith element of the output alphabet.

Remark. In this encoding we assume that the transducer is *total*, i.e. for every state $S \in \mathcal{S}$ and every letter in the input alphabet $a \in \mathcal{A}$ there is a transition rule in $\mathcal{R}$. If the transducer is partial (not total), we may use dummy combinators, without defining equations in ε, to represent absent transitions. We have two solutions:

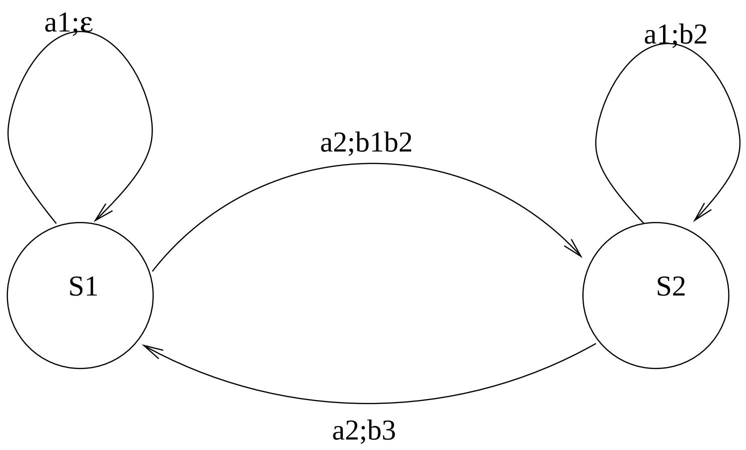
1. using the same combinator $\perp$ for all the absent transitions. In this case we define N_i as $\perp (v_1, \ldots v_p)$.
2. using a different combinator for each absent transition. In this case we define N_i as $U_{S,a}(v_1, \ldots v_p)$. $U_{S,a}$ is an unknown combinator without a definition.

The advantage of the second solution is that it may be completed in an incremental way by adding separate definitions for the various $U_{S,a}$.

2.2 An example

The transducer. We define a particular transducer. Its input alphabet is $\mathcal{A} = \{a_1, a_2\}$. Its output alphabet is $\mathcal{B} = \{b_1, b_2, b_3\}$. Its set of states is $\mathcal{S} = \{S_1, S_2\}$.

Its set of transition rules is defined by the following automaton graph:



The corresponding regular Böhm tree presentation

$$\begin{aligned}
S_1\ v_1\ v_2\ v_3\ u &:= u(S_1(v_1, v_2, v_3), T_1(v_1, v_2, v_3))\\
T_1\ v_1\ v_2\ v_3 &:= v_1(T_2(v_1, v_2, v_3))\\
T_2\ v_1\ v_2\ v_3 &:= v_2(S_2(v_1, v_2, v_3))\\
S_2\ v_1\ v_2\ v_3\ u &:= u(T_3(v_1, v_2, v_3), T_4(v_1, v_2, v_3))\\
T_3\ v_1\ v_2\ v_3 &:= v_2(S_2(v_1, v_2, v_3))\\
T_4\ v_1\ v_2\ v_3 &:= v_3(S_1(v_1, v_2, v_3)).
\end{aligned}$$

2.3 Global variant

There are several variations on such encodings. The Böhm tree corresponding to the above encoding has a bound variable u for each value in the input stream. We can use a different encoding where all these u's are shared. We define the terms $M_{a,j}$ with $0 \leq j \leq m$ as follows.

$$M_{a,m+1} \text{ is } S'(v_1, \ldots, v_p, u)$$

$$M_{a,j} \text{ with } 1 \leq j < m \text{ is } T_{S,a,j}(v_1, \ldots, v_p, u)$$

and we write the m definitions:

$$T_{S,a,j}\ v_1 \ldots\ v_p\ u := v_{b_j}(M_{a,j+1}) \text{ with } 1 \leq j < m.$$

Let N_a be $M_{a,1}$. We finally write the definition for S as

$$S\ v_1 \ldots\ v_p\ u := u(N_1, \ldots, N_n) \text{ with } N_i = N_{a_i}.$$

Example. The global encoding of the preceding example is as follows:

$$
\begin{aligned}
&S_1\ v_1\ v_2\ v_3\ u := u(S_1(v_1, v_2, v_3, u), T_1(v_1, v_2, v_3, u))\\
&T_1\ v_1\ v_2\ v_3\ u := v_1(T_2(v_1, v_2, v_3, u))\\
&T_2\ v_1\ v_2\ v_3\ u := v_2(S_2(v_1, v_2, v_3, u))\\
&S_2\ v_1\ v_2\ v_3\ u := u(T_3(v_1, v_2, v_3, u), T_4(v_1, v_2, v_3, u))\\
&T_3\ v_1\ v_2\ v_3\ u := v_2(S_2(v_1, v_2, v_3, u))\\
&T_4\ v_1\ v_2\ v_3\ u := v_3(S_1(v_1, v_2, v_3, u)).
\end{aligned}
$$

Discussion on the encodings The basic ideas behind our encodings are issued from the classical λ-calculus encodings of arithmetic worked out by Church. In these codings, data structures are explained as unsaturated control structures. Thus the pairing constructor is defined as $p := \lambda x\ y\ u.u(x, y)$ and the pair (x, y) is thus represented as $\lambda u.u(x, y)$. This functional object may then be applied to a selector function in order to project on its components, with $\pi_1 := \lambda x\ y.x$ and $\pi_2 := \lambda x\ y.y$. More generally, the n-tuple constructor is $\lambda x_1 \ldots x_n\ u.u(x_1, \ldots, x_n)$. This idea is exploited directly in the definition of the state combinators S above. They correspond in the Böhm tree to a node with n subtrees. Remember that n is the cardinality of the input alphabet. The n subtrees correspond to all the transitions issued from state S. Dually, the outputs are coded as projection functions, with letter b_j represented as $\lambda v_1 \ldots v_p.v_j$. The transition combinators T are defined as the application of the corresponding letter encoding in the output string to a unique subtree argument, which represents the "continuation", i.e. either a transition combinator if the output string is not exhausted, or the next state combinator otherwise.

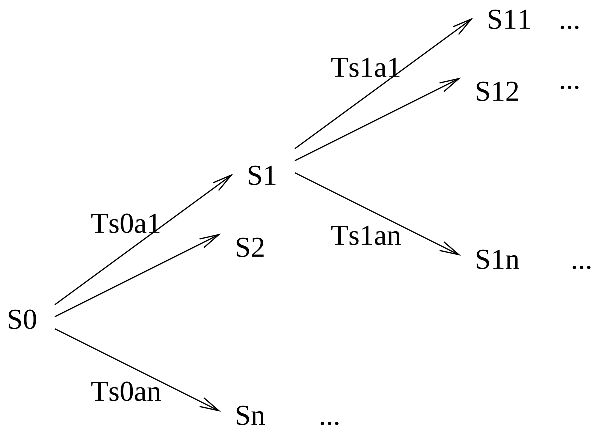
In the first representation, a fresh variable u is generated at each state transition, whereas in the global variant all the n-tuple constructors are identified as a unique variable u bound at the top root of the tree. The first representation has the advantage that the combinator expressions may directly be used to simulate by reduction the behavior of the automaton, by applying the state nodes to actual input characters coded as projections. The global representation is not so different actually, since the unique top u may be instantiated by the n-tuple constructor, which will generate a fresh u at each level. This trick is well known from the Böhm-out construction which is the basis for the proof of Böhm's theorem[4].

3 Semantics of finite state transducers

We shall now actually formalize this intuitive explanation of the link between the Böhm forest represented by our combinator presentation and the forest of all the behaviors of the corresponding transducer family.

3.1 Derivation tree

Definition. The derivation forest of the transducer family is the forest of all its possible traces. It is potentially infinite. A tree in the forest has the following shape:



This defines the tree of behaviors of the finite automaton underlying the transducer. The nodes of the tree are labeled by states, the arcs are labelled by the corresponding transitions. The top node of the tree is the initial state. In other words, each tree in the forest is the unfolding of the automaton graph, starting at the initial state.

3.2 Associated Böhm tree

We now associate a (possibly infinite) Böhm tree to each such derivation tree, where we replace the arc labels by linear Böhm trees encoding the output strings. Assuming that S_0 is the initial state, we associate to the root of the tree the Böhm tree

$$\mathcal{D}(S_0) = \lambda\ v_1\ v_2 ... v_p\ u.u(\mathcal{D}(T_{S_0,a_1}), \ldots, \mathcal{D}(T_{S_0,a_n}))$$

where $\mathcal{D}(T_{S,a_i})$ are the Böhm trees associated to the arcs of the derivation tree.

To each arc in the tree $T_{S,a} = (S', \mathbf{b})$ with $\mathbf{b} = b_1 \ldots b_m$, we associate the Böhm tree:

$$\mathcal{D}(T_{S,a}) = v_{b_1}(v_{b_2}(\ldots (v_{b_m}(\mathcal{D}(S')) \ldots).$$

To each non-root node of the tree we associate the Böhm tree:

$$\mathcal{D}(S) = \lambda u.u(\mathcal{D}(T_{S,a_1}), \ldots, \mathcal{D}(T_{S,a_n})).$$

Fact: The Böhm tree $\mathcal{D}(S_0)$ and the Böhm tree B_{S_0} defined by (ε, S_0) are identical.

Proof. $\mathcal{D}(S_0)$ is a solution of ε (left to the reader). By the Banach fixpoint theorem we know that there is only one solution to this system. This proves that B_{S_0}, the Böhm tree associated to ε and $\mathcal{D}(S_0)$ associated to the derivation tree are identical.

Note. In the global variant we would obtain a very similar Böhm tree except that there is no λu in the internal nodes (and all the u's refer to the one bound at the root).

This result allows us to use the combinators as a mechanism to compute the behavior of the transducer. This formalism can be considered as a simple abstract machine. It can be used as the output language of reactive languages compilers since such languages are usually compiled into transducers. In particular, regular Böhm trees are well suited for functional reactive languages like Lustre [1, 5].

Remark that equality of our Böhm trees corresponds to bisimulation of the corresponding automata. However the notion of extensional equality studied in [3] is not relevant here, since our Böhm trees have fixed arities, and η expansion has no direct interpretation.

4 Discussion

The shape of the Böhm tree presentation ε associated to a transducer is very specific. It seems that all the features of regular Böhm trees are not exploited by our encoding.

In this short version of this note, we just evoke briefly some possible generalisations of regular transducers suggested by the full power of our regular Böhm combinators. These generalisations will be developed in more detail in the full version of this paper.

1. All the input nodes are n-ary. If we relax this condition, we could have a notion of transducer where the possible inputs depend on the current state of the transducer. This would give a kind of typed automaton: states are typed by the alphabet of inputs allowed to occur in the input stream. This could have a natural interpretation in control automata where there is not a global unique input tape, but rather each state allows certain signals to be taken into account. Different signals could correspond to different input variables; that is, the rôle played so far by variable u could be played by several signal variables $u_1, \ldots, u_q$ given with their respective type.
2. Similarly, the output alphabet is fixed so far; i.e., the v_j's are bound initially in the starting state, and then just passed along in every combinator definition; instead, we could 'forget' certain output letters by not transmitting them in a combinator call, and conversely 'generate' new members of the output alphabet by extra bindings; this would give a kind of bounded memory to our transducers: they could output information which would actually represent some kind of partial trace of previous states.
3. All the output nodes are unary. If we relax this condition, we would get the notion of a multitask transducer. This transducer outputs a tree node of arity p and forks into p automata, each of these governing one branch of the generated output. Of course, all these automata share the same control graph. On the other hand, several possible interpretations of how to read the input stream may be considered.

The usefulness of these elaborations is not entirely clear. If we look at Böhm trees in full generality, we are looking at the combinatory richness of a Turing-complete formalism, which may emulate all possible control and data structures: stores, environments, continuations, functional encodings of data structures, etc. On the other hand, if we consider only those trees which may be generated from finite systems of combinators, it is very likely that whatever construct we may interpret could be as well encoded in a plain finite-state transducer, by standard tricks (increasing the states, merging several input tapes into one input tape over a larger alphabet, etc).

5 Conclusion

In this paper, we have explored expressivity of regular Böhm trees. In particular we have shown that deterministic finite-state transducers could be encoded as regular Böhm trees. We propose two different representations:

1. A global encoding with no internal λ binders. It is very similar to regular trees used in the semantics of flowcharts (see [2]).
2. An encoding with internal binders. This encoding allows us to simulate the behavior of the transducer by reduction.

We have also started to investigate how to exploit the full generality of regular Böhm trees to enrich finite-state transducers into more general reactive automata, which could accomodate higher-level notions from languages of the reactive paradigm family (Lustre, Signal, Estérel, Statecharts, etc). So far we only gave a sketch of some possible interpretations of the general formalism.

A possibly more fruitful investigation would be to explore the possibility of representing more powerful automata. For instance, there is a natural generalisation of regular Böhm combinators which yields pushdown automata, but this is getting out of the scope of the present note.

References

1. P. Caspi, M. Pouzet. *Synchronous Kahn Networks.* In ACM SIGPLAN International Conference on Functional Programming, Philadelphia, May 1996.
2. B. Courcelle. *Fundamental properties of infinite trees.* Theoret. Comput. Sci. **25** (1983) 95–109.
3. G. Huet. *Regular Böhm Trees.* To appear, Math. Struct. in Comp. Science. Available at `ftp://ftp.inria.fr/INRIA/Projects/coq/Gerard.Huet/RBT.ps.Z`
4. G. Huet. *An analysis of Böhm's theorem* TCS 121(1993) 145-167.
5. P. Raymond, N. Halbwachs, P. Caspi and D. Pilaud. *The synchronous dataflow programming language LUSTRE.* Proceedings of the IEEE, 79(9) 1305-1320, 1991.

This article was processed using the LaTeX macro package with LLNCS style