Thompson Languages

Dora Giammarresi, Jean-Luc Ponty, and Derick Wood

Summary. A finite-state machine is called a Thompson machine if it can be constructed from a regular expression using Thompson's construction. We assign distinct bracket pairs to the source and target states of all machine units that make up a Thompson machine. Each path from the start state to the final state of such a Thompson machine spells out a Dyck string. The collection of all such Dyck strings is the Thompson language of the given machine and the collection of such strings that are spelled out by simple paths is the simple Thompson language of the given machine. We characterize simple Thompson languages and Thompson languages, and we investigate their relationship.

1 Introduction

In 1968, Thompson [6] introduced an elegant and simple inductive construction of a finite-state machine from a regular expression. The resulting finite-state machine has size linear in the size of the original expression and each state has no more than two ingoing transitions and two outgoing transitions. A resurge of interest in the implementation of machines has resulted in some new discoveries about the Thompson construction [2,4].

Recently [5] we characterized the underlying digraphs of the machines resulting from the Thompson construction, **Thompson digraphs** and **Thompson machines,** respectively. The characterization is different from the one of Caron and Ziadi [3] for Glushkov digraphs although it is necessarily related to theirs indirectly.

The development of the characterization of Thompson digraphs [5] led us to the idea of associating labeled left and right brackets with the source and target states, respectively, of the unit machines that are used in the inductive Thompson construction. (We use the designation source state and target state when the corresponding states are not a start state and not a final state which occurs when we discuss a unit that is part of a larger machine.) We can view the bracketing as a means of linearizing the induction tree given by the Thompson construction. Once we have labeled the states in this way each path from the start state to the final state spells out a Dyck string. The set of all Dyck strings spelled out by such paths is a subset of the Dyck language for the appropriate alphabet and we call it a **Thompson language.**

Although Thompson languages appear to be nonregular, they are regular languages as they are specified by the duals of finite-state machines. We establish a Kleene-like theorem for them that is analogous to the standard Kleene theorem for regular languages; see Section 4.

In Section 5, we characterize **simple Thompson languages.** The set of all strings spelled out by simple paths from the start state to the final state of a Thompson machine is a simple Thompson language. We then characterize Thompson languages in terms of simple Thompson languages—our main result.

2 Notation and terminology

We recall the basics of finite-state machines and regular expressions and introduce the notation that we use.

A **finite-state machine**[1] consists of a finite set Q of states, an input alphabet Σ, a start state $s \in Q$, a final state $f \in Q$ and a transition relation $\delta \subseteq Q \times \Sigma_\lambda \times Q$, where λ denotes the null string and $\Sigma_\lambda = \Sigma \cup \{\lambda\}$. Clearly, we can depict the transition relation of such a machine as an edge-labeled digraph (the labels are symbols from Σ_λ); it is usually called the state or transition digraph of the machine.

Let Σ be an alphabet. Then, we define a regular expression E over Σ inductively as follows:

$E = \emptyset$, where $\emptyset$ is the empty-set symbol;
$E = \lambda$, where λ is the null-string symbol;
$E = a$, where a is in Σ;
$E = (F + G)$, where F and G are regular expressions;
$E = (F \cdot G)$, where F and G are regular expressions;
$E = (F^*)$, where F is a regular expression.

We define the language of E inductively, in the usual way [1,7].

We say that a regular expression E is **empty free** if E does not contain any appearance of the empty-set symbol.

3 Thompson machines and languages

Thompson designed a novel method [6] to compile regular expressions into a form that is suitable for pattern matching in text files. The construction is defined inductively (see Fig. 1). We treat only empty-free regular expressions; therefore, we never have to consider the finite-state machine given by Fig. 1(a). The Thompson construction gives a finite-state machine that has a number of pleasing properties:

[1] Normally, a finite-state machine is allowed to have more than one final state and, sometimes, more than one start state. The formulation we have chosen is appropriate for the study of Thompson machines.

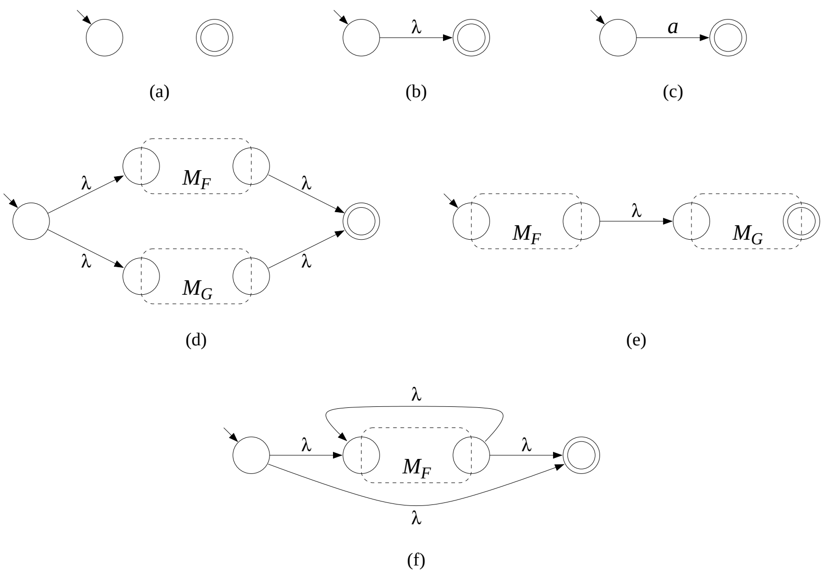

Fig. 1. The Thompson construction. The order of the figures corresponds to the order of the cases in the definition of regular expressions. The finite-state machines correspond to the regular expressions: a. $E = \emptyset$; b. $E = \lambda$; c. $E = a$, $a \in \Sigma$; d. $E = (F + G)$; e. $E = (F \cdot G)$; and f. $E = (F^*)$.

1. It has an even number of states.
2. It has one start state that is **only exiting;** that is, there are no transitions that enter it.
3. It has one final state that is **only entering;** that is, there are no transitions that leave it.
4. Every state is on a simple path from the start state to the final state.
5. Every state has at most two ingoing transitions and at most two outgoing transitions.
6. Its size is at most three times the size of the given regular expression.

In Fig. 2, we give the result of the Thompson construction on the regular expression $(((a + b)^*) \cdot ((b + \lambda) \cdot a))$.

We define a **Thompson machine** to be a finite-state machine that is obtained by the Thompson construction from an empty-free expression. We name the units that the Thompson construction uses to assemble a Thompson machine as follows: the **base units** (Figs. 1(b) and (c)); the **plus unit** (Fig. 1(d)); the **dot unit** (Fig. 1(e)); and the **star unit** (Fig. 1(f)). In Fig. 1, the source and target states for each unit correspond to their start and final states, respectively. Observe that the uniqueness of the start and final states implies that each Thompson machine has an even number of states.

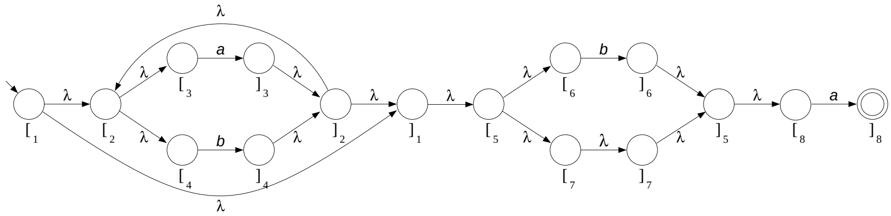

Fig. 2. The result of the Thompson construction on the running example expression $(((a+b)^*) \cdot ((b+\lambda) \cdot a))$.

We now define Thompson strings and languages. Define a countably infinite alphabet Γ of bracket pairs that we enumerate as $[_1,]_1, \ldots, [_i,]_i, \ldots$. For each $n > 0$, we define $\Gamma_n = \{[_i,]_i : 1 \leq i \leq n\}$. Given a language L over Γ_n, the image of L under an injection (or renaming) $h : \Gamma_n \longrightarrow \Gamma$, namely $h(L)$, clearly has the same structure as L. (Indeed, this observation holds for any injection or bijection of L into an appropriately sized alphabet.) Thus, when we write L, we really mean the family $\mathcal{L}$ of languages $\{h(L) : h \text{ is an injection } \Gamma_n \longrightarrow \Gamma\}$; all languages in this family are structurally equivalent.

Given a Thompson machine M that has $2n$ states, we label the states arbitrarily but uniquely with brackets from Γ_n in the following way: For each unit submachine of M, we label its source state with left bracket $[_i$ and its target state with right bracket $]_i$, for some i, $1 \leq i \leq n$.

Having labeled the states of M with the $2n$ brackets in Γ_n, we define a **Thompson string** as any string that is spelled out by some path in M from its start state to its final state. For example, if we label the source and target states of the machine of Fig. 2 with $[_1, \ldots,]_8$ as indicated, the strings

$$[_1\, [_2\, [_3\,]_3\,]_2\, [_2\, [_4\,]_4\,]_2\,]_1\, [_5\, [_6\,]_6\,]_5\, [_8\,]_8$$

and

$$[_1\, [_2\, [_4\,]_4\,]_2\,]_1\, [_5\, [_6\,]_6\,]_5\, [_8\,]_8$$

are Thompson strings of the given machine. Clearly, Thompson strings are Dyck strings over Γ_n; however, not all Dyck strings over Γ_n are Thompson strings. Two simple examples are the strings $x = [_1\,]_1\, [_1\,]_1$ and $y = [_1\, [_1\,]_1\,]_1$; we leave the proofs of their non-Thompson-ness to you. Similarly, we define a **simple Thompson string** as any string that is spelled out by some simple path in M from its start state to its final state (recall that a simple path visits each state at most once). Thus, each simple Thompson string contains at most one pair of each of the n bracket pairs. Of the two preceding examples of Thompson strings, the first string is not a simple Thompson string, but the second string is.

Given a Thompson machine M, its **Thompson language** is the set of all Thompson strings given by M. Similarly, M's **simple Thompson language** is the set of all simple Thompson strings given by M. By definition, every simple Thompson language is finite and some Thompson languages are infinite. Clearly, each Thompson language $L \subseteq {\Gamma_n}^*$ is a subset of the corresponding Dyck language $D(\Gamma_n)$. That each Thompson language does not contain all Dyck strings follows from the previous examples of Dyck strings that are not Thompson. But, Thompson languages are even more restrictive.

We investigate the structure of Thompson and simple Thompson languages and characterize them in terms of their structural properties.

4 A Kleene Theorem

Let $\mathcal{T}$ and $\mathcal{ST}$ denote the families of Thompson and simple Thompson languages, respectively. We introduce variants of the usual regular operations that lead to a Kleene theorem for the two families. Given two Thompson languages $K \subseteq D(\Gamma_m)$ and $L \subseteq D(\Gamma_n)$, we define bracketed versions of the regular operations as follows:

Bracketed union $\oplus$:

$$K \oplus L = \{[_{m+n+1}\}(K \cup h(L))\{]_{m+n+1}\},$$

where $h : \Gamma_n \longrightarrow \{[_i,]_i : m+1 \leq i \leq m+n\}$ is a renaming.

Bracketed product $\odot$:

$$K \odot L = Kh(L),$$

where $h : \Gamma_n \longrightarrow \{[_i,]_i : m+1 \leq i \leq m+n\}$ is a renaming.

Bracketed star $\otimes$:

$$K^{\otimes} = \{[_{m+1}\}K^*\{]_{m+1}\}.$$

Similarly, given two simple Thompson languages $K \subseteq D(\Gamma_m)$ and $L \subseteq D(\Gamma_n)$, we use the bracketed versions of union and product but we replace bracketed star with bracketed prestar defined as follows:

Bracketed prestar $\oslash$:

$$K^{\oslash} = \{[_{m+1}\}(K \cup \{\lambda\})\{]_{m+1}\}.$$

The proof of the following theorem is straightforward.

Theorem 1. *The family $\mathcal{T}$ of Thompson languages is closed under bracketed union, bracketed product and bracketed star.*

The family $\mathcal{ST}$ of simple Thompson languages is closed under bracketed union, bracketed product and bracketed prestar.

As a result of the preceding theorem we can provide an inductive definition of Thompson languages that parallels the inductive definition of Thompson machines. Given an alphabet Γ_n, we can inductively define a Thompson language as follows:

1. The language $\{[_1\,]_1\} \subseteq D(\Gamma_i)$ is a Thompson language.
2. If $K \subseteq D(\Gamma_m)$ and $L \subseteq D(\Gamma_n)$ are Thompson languages, then $K \oplus L$ is a Thompson language.
3. If $K \subseteq D(\Gamma_m)$ and $L \subseteq D(\Gamma_n)$ are Thompson languages, then $K \odot L$ is a Thompson language.
4. If $K \subseteq D(\Gamma_m)$ is a Thompson language, then $K^{\otimes}$ is a Thompson language.

If we replace $\otimes$ with $\oslash$, then we obtain an inductive definition of simple Thompson languages.

The basis for these definitions is the following Kleene theorem.

Theorem 2. *The family $\mathcal{T}$ of Thompson languages is the smallest family of languages that contains $\{[_1\,]_1\}$ and is closed under bracketed union, bracketed product and bracketed star.*

The family $\mathcal{ST}$ of simple Thompson languages is the smallest family of languages that contains $\{[_1\,]_1\}$ and is closed under bracketed union, bracketed product and bracketed prestar.

5 The characterization theorem

We develop a characterization of Thompson languages by first considering simple Thompson languages. We present some necessary conditions for a language $L \subseteq D(\Gamma_n)$ to be a simple Thompson language or a Thompson language.

Property 1: Bracket uniqueness. Each pair $[_i,]_i$ appears at most once in each string in a simple Thompson language L.

The following four properties apply to both simple Thompson and Thompson languages.

Property 2: Starting and ending uniqueness. All strings in L begin with the same left bracket $[_i$, say, and end with the same right bracket $]_j$, say.

Given a Thompson machine M of $2n$ states that are labeled with the brackets in Γ_n, let $L \subseteq D(\Gamma_n)$ be its simple Thompson or its Thompson language. Given a Thompson language $L \subseteq D(\Gamma_n)$, we define $L(i)$ to be the set of all strings that appear between the brackets $[_i,]_i$ in strings of L. (Note that the brackets may appear more than once in Thompson strings, but each appearance can contain only the same substrings.) More formally,

$$\begin{aligned} L(i) = \{x : {} & \text{there are strings } w \text{ and } y \text{ such that} \\ & w[_i x]_i y \in L \text{ and } x \text{ does not contain } [_i \text{ or }]_i\}. \end{aligned}$$

Similarly, we define $L[i]$ to be the set of all substrings of strings in L that begin with $[_i$ and end with $]_i$; that is,

$$\begin{aligned} L[i] = \{[_i x]_i : {} & \text{there are strings } w \text{ and } y \text{ such that} \\ & w[_i x]_i y \in L \text{ and } x \text{ does not contain } [_i \text{ or }]_i\}. \end{aligned}$$

Property 3: Unit conditions.

1. If a pair $[_i,]_i$ labels a base unit, then $L(i) = \{\lambda\}$.
2. If a pair $[_i,]_i$ labels a star unit, then $L(i) \supset \{\lambda\}$.
3. If a pair $[_i,]_i$ labels a plus unit, then $L(i) \neq \emptyset$ and $\{\lambda\} \notin L(i)$.

We consider the restrictions on substrings that occur with plus and star units:

Property 4: Multiplicity conditions.

1. If a pair $[_i,]_i$ labels a star unit, then all strings in $L(i) - \{\lambda\}$ begin with some bracket $[_j$, $j \neq i$, and end with some bracket $]_k$, $k \neq i$.
2. If a pair $[_i,]_i$ labels a plus unit, then all strings in $L(i)$ begin either with $[_j$ or with $[_m$, where $i \neq j$, $i \neq m$ and $j \neq m$, and end either with $]_k$ or with $]_n$, respectively, where $i \neq k$, $i \neq n$ and $k \neq n$.

The next property captures the sequential nature of the product of two machines:

Property 5: Product sequentiality. Let the pairs $[_i,]_i$ and $[_j,]_j$ label two units in M that have been combined by product, then, for all strings u, v, w and x, neither $u[_i v]_i[_n w]_n x$ nor $u[_m v]_m[_j w]_j x$ are in L, where $i \neq m$ and $j \neq n$.

The following result follows immediately from the definitions of the properties.

Lemma 1. *Let L be a simple Thompson language. Then, L satisfies Properties 1, 2, 3, 4 and 5.*

Let L be a Thompson language. Then, L satisfies Properties 2, 3, 4 and 5.

We are now in a position to characterize simple Thompson languages.

Theorem 3. *A language $L \subseteq D(\Gamma_n)$ is a simple Thompson language if and only if L satisfies Properties 1, 2, 3, 4 and 5.*

Proof. We already observed, by Lemma 1 that, if L is a simple Thompson language, then it satisfies Properties 1, 2, 3, 4 and 5. We now prove the converse. Let $L \subseteq D(\Gamma_n)$ be a language that satisfies Properties 1, 2, 3, 4 and 5. To demonstrate that L is a Thompson language, we sketch the construction of a Thompson machine M such that the Thompson language of M is isomorphic to L. The proof is by induction on the size n of the alphabet Γ_n of L.
Basis: In this case $n = 1$, and $L = \{[_1\,]_1\}$ by Properties 1 and 2.
Induction hypothesis: We assume the result holds for all languages $L \subseteq D(\Gamma_m)$ that satisfy Properties 1, 2, 3, 4 and 5, where $1 \leq m \leq n$, for some $n \geq 1$.

Induction step: Consider a language $L \subseteq D(\Gamma_{n+1})$ that satisfies Properties 1, 2, 3, 4 and 5. Since $n \geq 1$, the strings in L must contain at least two different brackets. Consider a string x in L. Then, either x begins with a left bracket $[_i$ and ends with the corresponding right bracket $]_i$ or it does not. In the first case, all strings in L must begin and end the same way by Property 2; therefore, by Property 3, the strings correspond either to a bracketed union or to a bracketed prestar (they cannot correspond to a base unit because that would imply that $n + 1 = 1$, which contradicts the assumption that $n + 1 \geq 2$). By Property 3, we can uniquely identify which alternative the strings correspond to; therefore, we can replace L either with two languages each with fewer brackets than L or with one language that has fewer brackets than L. A corresponding Thompson machine (with state labels but without edge labels) can be constructed inductively in this case.

In the second case, x starts with a left bracket $[_i$ but $]_i$ does not end x; therefore, we can partition x into uv, where u begins with $[_i$ and ends with $]_i$, by the structure of these strings. Since $v \neq \lambda$, we have discovered a product decomposition; namely, by Property 5, there is a left bracket $[_i$ such that, for all strings x in L, x can be decomposed into strings u and v such that $x = uv$, u begins with left bracket $[_i$ and ends with $]_i$, and $v \neq \lambda$. This decomposition of strings induces a decomposition of L into two languages J and K such that $J = L[i]$ and $K = \{v : uv \in L \text{ and } u \in L[i]\}$. Both J and K have fewer brackets than L and both satisfy Properties 1, 2, 3, 4 and 5; therefore, we can inductively construct a Thompson machine that corresponds to L.

The proof of the preceding theorem provides an inductive construction of a Thompson machine M (with state labels but without edge labels) from a simple Thompson language L such that the simple Thompson language of M is L. Based on this construction, we now characterize Thompson languages.

First, observe that, for each Thompson language L, the simple Thompson subset S of L consists of all the strings in L that satisfy Property 1. Second, observe that if the bracket pair $[_i,]_i$ corresponds to a bracketed prestar, then, conceptually, we can replace every string from $S[i]$ in strings of S with $[_i S(i)^*]_i$. We can then repeat this expansion for each bracketed prestar in turn. Unfortunately, after the first expansion, we obtain an infinite language. Therefore, we construct the Thompson machine M_S for the language S, where each bracketed prestar gives a star unit. The Thompson language of M_S is the language L if and only if L is a Thompson language. This proof sketch provides our last result, the sought characterization of Thompson languages.

Theorem 4. *Given a language $L \subseteq D(\Gamma_n)$ and its simple Thompson subset S, L is a Thompson language if and only if L satisfies Properties 2, 3, 4 and 5, and the Thompson language of M_S is L.*

Observe that the characterization is effective if we have an effective specification of L.

References

1. A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation, and Compiling, Vol. I: Parsing*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1972.
2. A. Brüggemann-Klein and D. Wood. The validation of SGML content models. *Mathematical and Computer Modelling*, 25:73–84, 1997.
3. P. Caron and D. Ziadi. Characterization of Glushkov automata. *Theoretical Computer Science*, 1998. To appear.
4. D. Giammarresi, J.-L Ponty, and D. Wood. The Glushkov and Thompson constructions: A synthesis. Submitted for publication, October 1998.
5. D. Giammarresi, J.-L Ponty, and D. Wood. A characterization of Thompson digraphs. Submitted for publication, January 1999.
6. K. Thompson. Regular expression search algorithm. *Communications of the ACM*, 11:419–422, 1968.
7. D. Wood. *Theory of Computation*. John Wiley & Sons, Inc., New York, NY, second edition, 1998. In preparation.