Regular Language Homomorphisms and Transition Complexity: Dealing with Large Alphabets

Stephen M. Watt
Western University
London, Canada
Stephen.Watt@uwo.ca

July 8, 2013

Abstract

Modern computer applications typically use alphabets, such as Unicode, with more than a million symbols, making the usual table-based scanning techniques infeasible. In practice, strings in these alphabets use multi-byte encodings. Viewing these encodings as string monomorphisms can make table-based scanning tools again practical. We have created a software tool, `reflex`, that allows regular languages over standard Universal Character Set to be mapped to `lex` or one of its derivatives, and compiled. A key observation is that the transition complexity of the image language can be significantly lower than the transition complexity of the original language. This leads the notion of the *inherent transition complexity* of a regular language, which we define as the minimum transition complexity under any string monomorphism and which can be exponentially smaller than the usual transition complexity.

Basic notions As usual, a *deterministic finite automaton* (DFA) is a tuple $A = (\Sigma, Q, q_0, F, \delta)$, where Σ is the finite alphabet of input symbols, Q is the finite set of states, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of accepting states, and $\delta : Q \times \Sigma \to Q$ is the transition function. We use $|S|$ to denote the number of elements in the finite set S, $L(A)$ to denote the language accepted by a DFA A and $\min(A)$ to denote the unique (up to relabelling) minimal DFA equivalent to A. If $f : \Sigma \to T^*$ such that f lifted to $\Sigma^* \to T^*$ is injective, we call f a *string monomorphism*. The *state complexity* of a regular language $L(A)$ is the number of states of $\min(A)$. Another measure is the *transition complexity*, the number of transitions of $\min(A)$. For total δ the transition complexity is simply the state complexity multiplied by the alphabet size. For this reason, transition complexity is normally not considered interesting in its own right.

Large alphabets in practice Table-based lexical analyzers typically represent states and symbols as integers and the transition function is represented as a two-dimensional array of size $|Q| \cdot |\Sigma|$ with entries being integers in the range $0..|Q|-1$. In past decades, when applications typically used alphabets of size at most 256, this worked well. Today, it is typical for applications, including modern programming languages, to use the Universal Character Set (UCS), defined by the international standard ISO/IEC 10646, or Unicode, which has the same repertoire of characters and adds rules for character normalization (e.g. combining accents and letters) and bidirectional text rendering (e.g. in Arabic/English mixed language text). Since version 2, this standard has supported 1,112,064 characters, which may be represented in a number of standard multi-byte encodings. For all but the simplest languages, table-based lexical analyzers become impractical, with each state requiring a 1 megabyte row. Consequently, tools to generate scanning tables (e.g. `lex`, `flex`) do not handle UCS. One approach that has been used to deal with this has been to work over a reduced alphabet, with each new symbol representing an equivalence class of symbols in the original language. This only works in some cases, however.

The various standard multi-byte representations for UCS may viewed as defining monomorphisms to strings of 8-bit bytes. For example, the UTF-8 representation of UCS gives each UCS symbol as 1, 2, 3 or

4 8-bit bytes and each UCS string image is uniquely invertible. We have used this fact to create `reflex` [1], a scanner generator for languages over UCS. This tool takes a scanner for a language L defined by regular expressions over UCS and generates a scanner for a language $\phi(L)$, defined by regular expressions over 8-bit bytes using any of the standard UCS encodings. We find in practice that the transition complexity for $\phi(L)$ may be less than the transition complexity of L.

A more abstract problem Let A be the DFA $(\Sigma, Q, q_0, F, \delta)$ for $\Sigma = T^N$, $N > 0$, and let $\phi : \Sigma \to T^N$ be a string monomorphism. We may construct a DFA $A' = (T, Q', q'_0, F', \delta')$ such that $L(A') = \phi(L(A))$, roughly speaking by replacing each state of A with tree of states in A'. We then ask what is the relationship between the transition complexity of $L(A)$ and that of $L(A')$? More specifically, for every state $q \in Q$, let there be $\sum_{i=0}^{N-1} |T|^i$ states in Q', linked by transitions in δ' to give a complete tree of arity $|T|$ and height $N-1$. For state $q \in Q$ denote the root of this tree q_ϕ. We have $q'_0 = q_{0\phi}$ and $F' = \{q_\phi | q \in F\}$. Because δ is total, the transition function δ' is then completely specified by $\delta' : (q_\phi, \phi(a)) \mapsto \delta(q, a)_\phi$ for all $q \in Q$ and $a \in \Sigma$. The number of transitions in A' is $|T| \cdot |Q'| = |Q| \cdot \sum_{i=1}^{N} |T|^i$. For $N > 1$, this is strictly greater than the number of transitions in A, which is $|Q| \cdot |\Sigma| = |Q| \cdot |T|^N$. When A is minimal, A' is not necessarily so.

Inherent transition complexity It seems unsatisfying for a complexity measure of a language to depend critically on how the alphabet is represented. We therefore propose an invariant measure.

Definition 1. *The* inherent transition complexity *of a regular language L is the least transition complexity of any minimized DFA accepting a string monomorphism image of L.*

The inherent transition complexity may be much less than the usual notion of transition complexity.

Theorem 1. *The inherent transition complexity of a language L recognized by DFA A may be exponentially less than the transition complexity of* $\min(A)$.

Proof. Consider $\Sigma = \{0,1\}^k$ and $L_k = \{a_1 a_2 \cdots a_k | a_i[i] = 0\}$. That is, a given symbol can appear in position i in a word only if its i-th bit is zero. All symbols are essential, *i.e.* no two distinct symbols can appear in the same set of positions. Because $|\Sigma| = 2^k$, the transition complexity of L_k is $\Omega(2^k)$. Now consider $T = \{0,1\}$ and $L'_k = \{\prod_{i=1}^{k} \prod_{j=1}^{k} b_{ij} | i = j \Rightarrow b_{ij} = 0\}$. This language may be recognized by a DFA with $k^2 + 2$ states and so has transition complexity $O(k^2)$. □

As string monomorphisms can vary the number of states and alphabet symbols, the number of bits required to represent a transition can vary. We therefore define the following bit complexity measures.

Definition 2. *The* transition bit complexity *of an automaton $A = (\Sigma, Q, q_0, F, \delta)$ is $|\Sigma| \cdot |Q| \cdot \log |Q|$.*

Definition 3. *The* inherent transition bit complexity *of a regular language L is the least transition bit complexity of any minimized DFA accepting a string monomorphism image of L.*

These notions correspond closely to the required table sizes for scanners when large alphabets are implemented as multi-byte sequences. We suggest these complexity measures are useful in analyzing tradeoffs in practical applications.

References

[1] S.L. Huerter and S.M. Watt, "reflex: A Scanner Transformer for Unicode Grammars," Ontario Research Centre for Computer Algebra (ORCCA), University of Western Ontario, Research Report TR-06-07 (34 pages), 2006.