Specifying A Software Module by Enumeration, and Transformations to and from Other Formal Representations

Lan Lin, Jesse H. Poore, Stacy J. Prowell

Department of Computer Science
University of Tennessee
Knoxville, TN 37996

ut-cs-05-556

June 2005

Contents

List of Figures

1 Transition Diagram

With a stimulus set S we can specify a software module in terms of an enumeration, a state machine, a set of regular expressions, or a set of prefix-recursive functions. Each representation contains enough information to allow for some operation on its own.

Given a software module to be specified, if we do not already have a state machine in mind to verify, we want most to work at the enumeration level, to discover and derive a state machine. Some model checkers work with state machines or regular expressions directly. Some theorem provers like ACL2 manipulate prefix-recursive functions to prove assertions. It thus becomes valuable to provide an algorithm for the transition between any of these forms.

Figure 1.1 shows the transition diagram between these forms as a square, where each can be obtained from any other form algorithmically. Each representation is as rich as, but no more powerful than any of the other three in the specification of the module. One is correct with regard to the underlying black box function if and only if any other is correct.

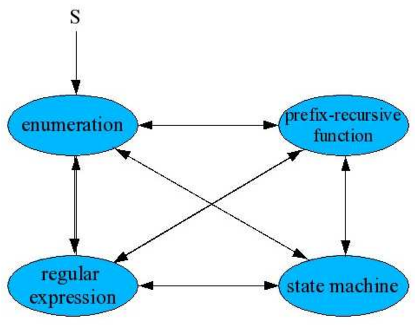

Figure 1.1: Transition Diagram

2 An Enumeration

Notation: Let A, B, and C be sets, and consider partial functions of the form $T : A \to B \times C$. We define the following notation. If $T(u) = \langle r, v \rangle$, then we write $u \overset{T}{\mapsto} r$ and $u \overset{T}{\triangleright} v$. The T will be dropped where it is obvious from context. Let dom $T = \{u \,|\, u \in A,\ T(u)$ is defined$\}$.

Definition 1 (Distinguishability): Let S be a finite, nonempty set, R be a non-empty set containing 0, ω, and at least one other member, and let $T : S^* \rightarrow R \times S^*$ be a partial function. We define relation $\not\sim \subseteq$ dom $T \times$ dom T as follows:

1. *If $a, b \in$ dom T and $\exists x \in S$ such that $ax \mapsto r$ and $bx \not\mapsto r$ for some $r \in R$, then $a \not\sim b$.*
2. *If $a \not\sim b$, and $\exists x \in S$ such that $cx \triangleright a$ and $dx \triangleright b$ for $c, d \in$ dom T, then $c \not\sim d$.*
3. *Nothing else is in $\not\sim$ except by a finite number of applications of the above two rules.*

Definition 2 (An Enumeration/A Complete Enumeration/A Complete and Minimal Enumeration/A Complete and Finite Enumeration): Let S be a finite, nonempty set equipped with a total order $<$ (the alphabetical order), and let the order be extended to a total order on S^* by length, and then alphabetically. Let R be a non-empty set containing 0, ω, and at least one other member. Then a partial function $\mathcal{E} : S^* \rightarrow R \times S^*$ is an enumeration iff 1–7 below hold $\forall x \in S, \forall u, v \in S^*$. The enumeration is complete iff 8 also holds. A complete enumeration is minimal if 9 further holds. A complete enumeration is finite if 10 further holds, where $\mathcal{N}$ denotes the set of natural numbers.

1. *$\lambda \mapsto 0, \lambda \triangleright \lambda$. (Empty sequence)*
2. *$u \triangleright v$ implies $v < u$ or $v = u$. (Partial order)*
3. *$u \triangleright v$ implies $v \triangleright v$. (No reduction to reduced)*
4. *$ux \in$ dom $\mathcal{E}$ implies $u \not\mapsto \omega$. (No extensions of illegals)*
5. *$ux \in$ dom $\mathcal{E}$ implies $u \triangleright u$. (No extensions of reduced)*
6. *$u \triangleright v, u \not\mapsto \omega$ implies $v \not\mapsto \omega$. (No reduction to illegals for legals)*
7. *$u \triangleright v, u \mapsto \omega$ implies $v \mapsto \omega$. (No reduction to legals for illegals)*
8. *$u \not\mapsto \omega, u \triangleright u$ implies $ux \in$ dom $\mathcal{E}$. (Completeness)*
9. *$u, v \in$ dom $\mathcal{E}, u \not\mapsto \omega, u \triangleright u, v \not\mapsto \omega, v \triangleright v, u \neq v$ implies $u \not\sim v$. (Minimality)*
10. *$\exists n \in \mathcal{N}$ such that $|$dom $\mathcal{E}| = n$. (Finiteness)*

For the rest of the discussion, we assume we are always working with a complete and finite enumeration $\mathcal{E}$ and its equivalent form in the transitions, unless stated otherwise. Infinite enumerations are not addressed as a result of realistic concerns. Incomplete enumerations can be made complete before applying any algorithm by mapping all unmapped sequences to some response i (for incompleteness) and reducing all such sequences to the first such sequence in canonical order.

3 Enumeration to State Machine

Given a complete and finite enumeration $\mathcal{E} : S^* \to R \times S^*$, we can construct the state machine M (a Mealy machine) as follows.

Let $C = \{u \,|\, u \not\mapsto \omega, u \rhd u\}, |C| = n$. Let $C = \{c_0, \ldots, c_{n-1}\}$, where $c_0 < \ldots < c_{n-1}$. It must be the case that $c_0 = \lambda$. We define $M = \langle Q, S, \delta, q_0, R, \nu, \phi\rangle$, where $Q = \{q_0, \ldots, q_{n-1}, q_\omega\}$, $\delta : Q \times S \to Q$ and $\nu : Q \times S \to R$ are both total functions defined by: $\delta(q_i, a) = q_j$ iff $c_i a \rhd c_j, \forall c_i, c_j \in C, \forall a \in S$; $\delta(q_i, a) = q_\omega$ iff $c_i a \mapsto \omega, \forall c_i \in C, \forall a \in S$; $\delta(q_\omega, a) = q_\omega, \forall a \in S$. $\nu(q_i, a) = r$ iff $c_i a \mapsto r, \forall c_i \in C, \forall a \in S, \forall r \in R$; $\nu(q_\omega, a) = \omega, \forall a \in S$.

4 State Machine to Enumeration

Given a state machine $M = \langle Q, S, \delta, q_0, R, \nu, \phi\rangle$, where $Q = \{q_0, \ldots, q_{n-1}, q_\omega\}$, $\delta : Q \times S \to Q$ and $\nu : Q \times S \to R$ are both total functions, how can we get the implied enumeration $\mathcal{E} : S^* \to R \times S^*$?

Extend $\delta : Q \times S \to Q$ to $\hat{\delta} : Q \times S^* \to Q$ by $\hat{\delta}(q, \lambda) = q, \forall q \in Q; \hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a), \forall q \in Q, \forall w \in S^*, \forall a \in S$, and simply refer to $\hat{\delta}$ as δ. Let $\delta(q_0, c_i) = q_i$ such that $\forall z \in S^*, \delta(q_0, z) = q_i, z \neq c_i \Rightarrow c_i < z, \forall\, 0 \leq i \leq n-1$. It must be the case that $c_0 = \lambda$. For λ, $\lambda \mapsto 0, \lambda \rhd \lambda$. It is extended by each a in S and the extensions are considered in canonical order. Each string u considered after λ can be written as $u = wa$, for some $w \in S^*, a \in S$. Thus $u \mapsto \nu(\delta(q_0, w), a), u \rhd c_i$ if $\delta(q_0, u) = q_i$ for some $0 \leq i < n$, $u \rhd u$ otherwise. u is further extended by each a in S iff $u \not\mapsto \omega, u \rhd u$. The process continues until $\mathcal{E}$ is complete.

5 Enumeration to Regular Expression

Given a complete and finite enumeration $\mathcal{E} : S^* \to R \times S^*$, we want to derive a set of regular expressions for each legal equivalence class and for each response.

Let $C = \{u \mid u \not\mapsto \omega, u \triangleright u\}, |C| = n$. Let $C = \{c_0, \ldots, c_{n-1}\}$, where $c_0 < \ldots < c_{n-1}$. It must be the case that $c_0 = \lambda$. Let $r_{c_0}, \ldots, r_{c_{n-1}}$ be regular expressions for each legal equivalence class. For each r_{c_i}, let $E_{c_i} = \{u \mid u \triangleright c_i\}$. Assume $E_{c_i} = \{e_{i,1}, \ldots, e_{i,p_i}\}$ for some integer $p_i > 0$. Each $e_{i,j} \in E_{c_i}$ contributes to one item $t_{i,j}$ on the RHS of the regular expression equation for r_{c_i}. The RHS is these p_i items unioned together. $r_{c_i} = t_{i,1} + \ldots + t_{i,p_i}$. $t_{i,j} = \lambda$ if $e_{i,j} = \lambda$, otherwise $e_{i,j} = c_k a$ for some $c_k \in C, a \in S$, and thus $t_{i,j} = r_{c_k} a$. This writing gives us a regular expression equation array for each element in C as follows:

$$\begin{cases} r_{c_0} & = & t_{0,1} + \ldots + t_{0,p_0} \\ \ldots & = & \ldots \\ r_{c_{n-1}} & = & t_{n-1,1} + \ldots + t_{n-1,p_{n-1}} \end{cases}$$

Now that we have n equations and n variables $r_{c_0}, \ldots, r_{c_{n-1}}$ to solve. We can apply substitution and elimination to get the solved expressions for each of them. They form a base set in that once they become known, all the other regular expressions can be obtained easily by substitution.

This is because we can write another array of equations for each response in R. Assume $R = \{r_0, \ldots, r_{m-1}\}$, and $r_{m-1} = \omega$. For each r_{r_i}, let $E_{r_i} = \{u \mid u \mapsto r_i\}$. Assume $E_{r_i} = \{e'_{i,1}, \ldots, e'_{i,q_i}\}$ for some integer $q_i \geq 0$. The derivation of $t'_{i,j}$ from $e'_{i,j}$ is exactly the same as that of $t_{i,j}$ from $e_{i,j}$. The array looks like:

$$\begin{cases} r_{r_0} & = & t'_{0,1} + \ldots + t'_{0,q_0} \\ \ldots & = & \ldots \\ r_{r_{m-2}} & = & t'_{m-2,1} + \ldots + t'_{m-2,q_{m-2}} \\ r_{r_{m-1}} & = & (t'_{m-1,1} + \ldots + t'_{m-1,q_{m-1}})S^* \end{cases}$$

The RHS of the equation for r_ω(i.e. $r_{r_{m-1}}$) is further concatenated with S^*, to reflect that any extension of an illegal sequence remains illegal.

In this array, all the RHSs are expressions in terms of $r_{c_0}, \ldots, r_{c_{n-1}}$, which when substituted for their solved expressions, give us every one we need on the LHS.

6 Regular Expression to Enumeration

Given $r_{c_0}, \ldots, r_{c_{n-1}}$ for each legal equivalence class, where $c_0 < \ldots < c_{n-1}$, and $r_{r_0}, \ldots, r_{r_{m-1}}$ for each possible response, here is a way to recover the implied enumeration $\mathcal{E} : S^* \rightarrow R \times S^*$.

Derive the first strings in canonical order represented by $r_{c_0}, \ldots, r_{c_{n-1}}$. They form the set $C = \{c_0, \ldots, c_{n-1}\}$ of canonical sequences, where $c_0 < \ldots < c_{n-1}$. We start with λ. It must be the case that $\lambda \in r_0, \lambda \in r_\lambda$ (i.e. r_{c_0}). Thus $\lambda \mapsto 0, \lambda \rhd \lambda$, and it is extended by each a in S. The extensions are considered in canonical order. For each u that is considered in turn, $u \mapsto r_i$ iff $u \in r_{r_i}$, $u \rhd c_j$ iff $u \in r_{c_j}$, $u \rhd u$ if $u \in r_\omega$. u is further extended by each a in S iff $u \not\mapsto \omega, u \rhd u$, and the extensions are further considered in canonical order. The process continues until $\mathcal{E}$ is complete.

To decide whether $u \in r$ for string u and regular expression r, the following procedure can be used. Apply Theorem 2.3 in [1] to construct an NFA with ϵ-transitions that accepts r, and apply Theorems 2.2 and 2.1 to obtain a DFA, M_r, accepting r. $u \in r$ iff $u \in L(M_r)$.

7 Enumeration to Prefix-recursive Function

Given a complete and finite enumeration $\mathcal{E} : S^* \to R \times S^*$, we need to define two functions in prefix-recursive form, the specification function and the black box function.

Let $C = \{u \mid u \not\mapsto \omega, u \rhd u\}, |C| = n$. The specification function $f : S^* \to \{0, \ldots, n-1\}$ distinguishes n canonical sequences. The integer values can be replaced with meaningful values when necessary without loss of generality. Let $C = \{c_0, \ldots, c_{n-1}\}$, where $c_0 < \ldots < c_{n-1}, f(c_i) = i, \forall\, 0 \leq i < n$. For each $u \in \operatorname{dom} \mathcal{E}$ with $u \rhd v, v \in C, u \neq \lambda$, u can be written as $u = wa$, for some $w \in C, a \in S$. This gives $f(ux) = f(v)$ if $f(u) = f(w) \wedge x = a, \forall\, u \in S^*, \forall\, x \in S$. The base case for the inductive definition is $f(\lambda) = 0$. Because we do not care about the values for illegal sequences, we have the recursive rule $f(ux) = f(u)$ otherwise $\forall\, u \in S^*, \forall\, x \in S$.

The black box function $BB : S^* \to R$ is defined with the help of f. For the base case we have $BB(\lambda) = 0$. For each $u \in \operatorname{dom} \mathcal{E}$ with $u \mapsto r, r \in R$ (allowing $r = \omega$), $u \neq \lambda$, u can be written as $u = wa$, for some $w \in C, a \in S$. This gives $BB(ux) = r$ if $f(u) = f(w) \wedge x = a \wedge BB(u) \neq \omega, \forall\, u \in S^*, \forall\, x \in S$. Because the values of specification function f for illegal sequences appearing in the enumeration are chosen arbitrarily as the values for their legal canonical prefixes (after removing the last symbol), we have $BB(ux) = \omega$ if $BB(u) = \omega$ to avoid messing up with the illegal sequences.

8 Prefix-recursive Function to Enumeration

Given a specification function f and a black box function BB in prefix-recursive form and defined everywhere on S^*, how can we extract a complete enumeration $\mathcal{E} : S^* \to R \times S^*$?

We can start with the empty sequence λ. We must have $f(\lambda) = 0$ and $BB(\lambda) = 0$. Let $c_i \in \{u \,|\, f(u) = i, u < v, \forall\, v \in S^*$ such that $f(v) = i, v \neq u\}, \forall\, 0 \leq i < n$. We have $c_0 = \lambda, \lambda \mapsto 0, \lambda \rhd \lambda$. Next we extend λ by each $a \in S$, since it is canonical. For each sequence u to be defined after λ, it can be written as $u = wa$ for some $w \in C = \{c_0, \ldots, c_{n-1}\}, a \in S$, because only canonical sequences are extended. We have already known $f(w) = i$ for some i, $BB(w) \neq \omega$, otherwise w would not have been extended by a. By definition we also know $f(u)$ and $BB(u)$. If $BB(u) = \omega$, $u \mapsto \omega, u \rhd u$, otherwise let $f(u) = j$ for some $0 \leq j < n$, $u \mapsto BB(u), u \rhd c_j$. Only when $u \not\mapsto \omega$ and $u \rhd u$ do we further extend u by each $a \in S$, and consider the extensions in canonical order. This process continues until a complete enumeration $\mathcal{E}$ is obtained.

9 State Machine to Regular Expression

Given a state machine $M = \langle Q, S, \delta, q_0, R, \nu, \phi \rangle$, where $Q = \{q_0, \ldots, q_{n-1}, q_\omega\}$, $\delta : Q \times S \to Q$ and $\nu : Q \times S \to R$ are both total functions, how can we derive the set of regular expressions for each legal equivalence class and each response?

State q_ω can be identified as the one from which all outgoing arcs are associated with the illegal response. Every other state represents a legal equivalence class. Assume they are the states denoted by $q_0, \ldots, q_{n-1}$, and the regular expressions for them are denoted by $r_{c_0}, \ldots, r_{c_{n-1}}$.

To get each r_{c_i}, we make q_i the unique final state in M, and denote the modified automaton M_i. $M_i = \langle Q, S, \delta, q_0, R, \nu, \{q_i\} \rangle$, where everything is defined the same as for M except for the final state set. $r_{c_i} = L(M_i)$. Theorem 2.4 in [1] gives us a constructive proof that can serve as an algorithm to get r_{c_i} from M_i.

Assume the regular expression for each response $r_i \in R$ is denoted by r_{r_i}. Let $E_{r_i} = \{\langle q, a \rangle \,|\, \nu(q, a) = r_i, q \in Q, q \neq q_\omega, a \in S\}$. Assume $E_{r_i} = \{\langle q_{i1}, a_{i1} \rangle, \ldots, \langle q_{ip}, a_{ip} \rangle\}$ for some integer $p \geq 0$. $r_{r_i} = r_{c_{i1}} a_{i1} + \ldots + r_{c_{ip}} a_{ip}$. If $\lambda \notin r_0$, union λ to the RHS of the equation for r_0. Further concatenate S^* with the RHS of the equation for r_ω if $E_\omega \neq \phi$ to reflect that any extension of an illegal sequence is illegal.

10 Regular Expression to State Machine

Given $r_{c_0}, \ldots, r_{c_{n-1}}$ for each legal equivalence class, and $r_{r_0}, \ldots, r_{r_{m-1}}$ for each possible response, how can we reconstruct the implied state machine M?

$M = \langle Q, S, \delta, q_0, R, \nu, \phi \rangle$, where $Q = \{q_0, \ldots, q_{n-1}, q_\omega\}$, δ and ν can be defined as follows. For each r_{c_i}, consider $r_{c_i}a, \forall\, a \in S$. Either $r_{c_i}a \subseteq r_{c_j}$ for some j, or $r_{c_i}a \subseteq r_\omega$. In the former case we have $\delta(q_i, a) = q_j$, and in the latter case we have $\delta(q_i, a) = q_\omega$. With regard to response, $r_{c_i}a \subseteq r_{r_k}$ for some k, thus we have $\nu(q_i, a) = r_k$. For $q_\omega, \delta(q_\omega, a) = q_\omega, \nu(q_\omega, a) = \omega, \forall\, a \in S$. At the end of the process δ and ν are fully defined on $Q \times S$.

To decide whether $r_1 \subseteq r_2$ for regular expressions r_1 and r_2, we can apply Theorem 2.3 in [1] to construct two NFAs with ϵ-transitions that accept r_1 and r_2 respectively, then apply Theorem 2.2 and Theorem 2.1 to build two DFAs accepting r_1 and r_2. Let the two DFAs be M_1 and M_2. We construct M_3 such that $L(M_3) = L(M_1) \cap \overline{L(M_2)}$ (i.e. $L(M_3) = r_1 - r_2$) by applying Theorem 3.2 and Theorem 3.3. $r_1 \subseteq r_2$ iff $L(M_3) = \phi$ as a result of applying Theorem 3.7.

11 State Machine to Prefix-recursive Function

Given a state machine $M = \langle Q, S, \delta, q_0, R, \nu, \phi \rangle$, where $Q = \{q_0, \ldots, q_{n-1}, q_\omega\}$, δ and ν are both total functions, we first identify q_ω to be the state with coming arcs associated with the illegal response. The first trace in canonical order is identified for every other state, which gives us the set of all canonical sequences $\{c_0, \ldots, c_{n-1}\}$. Without loss of generality, assume $c_0 < \ldots < c_{n-1}$. Correspondingly the states are assigned values from 0 to $n - 1$ for the specification function f. From then on we can treat these states as $q_{c_0}, \ldots, q_{c_{n-1}}$, and plug them into the definition for δ and ν. The base case for f is defined to be $f(\lambda) = 0$. For each $\delta(q_{c_i}, a) = q_{c_j}$, we have $f(ux) = j$ if $f(u) = i \wedge x = a, \forall\, u \in S^*, \forall\, x \in S$. Because we only consider transitions between states other than q_ω, we arbitrarily assign specification function values to illegal sequences by declaring $f(ux) = f(u)$ otherwise.

The base case for BB is defined to be $BB(\lambda) = 0$. For each $\nu(q_{c_i}, a) = r$(allowing $r = \omega$), we have $BB(ux) = r$ if $f(u) = i \wedge x = a \wedge BB(u) \neq \omega, \forall\, u \in S^*, \forall\, x \in S$. For extensions of an illegal sequence we have $BB(ux) = \omega$ if $BB(u) = \omega$.

12 Prefix-recursive Function to State Machine

Given f and BB defined in prefix-recursive form, how do we reconstruct the implied state machine M?

Suppose we have possible values from 0 to $n-1$ for f, we will have states from q_0 to q_{n-1} in M, corresponding to each value. M has one other state for all illegal sequences denoted by q_ω. In the inductive definition for BB, whenever we have $BB(ux) = r$(allowing $r = \omega$) if $f(u) = i \wedge x = a \wedge BB(u) \neq \omega, \forall\, u \in S^*, \forall\, x \in S$, correspondingly we have an output $\nu(q_i, a) = r$. In case $r \neq \omega$, we check the corresponding entry in the inductive definition for f. If $f(ux) = j$ if $f(u) = i \wedge x = a$, we define a transition $\delta(q_i, a) = q_j$. In accordance with $BB(ux) = \omega$ if $BB(u) = \omega$, we define $\nu(q_\omega, a) = \omega, \forall\, a \in S$ and $\delta(q_\omega, a) = q_\omega, \forall\, a \in S$. M is defined as $M = \langle Q, S, \delta, q_0, R, \nu, \phi \rangle$, where $Q = \{q_0, \ldots, q_{n-1}, q_\omega\}$.

13 Regular Expression to Prefix-recursive Function

Given a set of regular expressions for each legal equivalence class, and a set of regular expressions for each response, here is a way to write the specification function and the black box function in prefix-recursive form.

Suppose $r_{c_0}, \ldots, r_{c_{n-1}}$ correspond to regular expressions for each legal equivalence class, and r_{r_i} is for the regular expression for $r_i \in R$. The set $\{r_{c_0}, \ldots, r_{c_{n-1}}\}$ forms a base set B in that if members in this set are known, any r_{r_i} can be derived. The base set corresponds to values of f from 0 to $n-1$.

Then we consider $ra, \forall\, r \in B, \forall\, a \in S$ and write down each item's contribution to the RHSs of the above expressions. λ contributes to the RHS of r_λ and r_0. In this way each regular expression given can be expressed as the union (0 or more) of items in the form of either λ or a one-symbol extension of a base regular expression on the RHS. We further concatenate the RHS for r_ω by $(s_1 + \ldots + s_k)^*$ where $s_1, \ldots, s_k$ represent all elements in S, because any extension of an illegal sequence is illegal. $r_{c_i}a$ is one unioned item on the RHS of the equation for r_{c_j} iff $r_{c_i}a \subseteq r_{c_j}$, which is further decidable as follows. Apply Theorem 2.3 in [1] to build two NFAs with ϵ-transitions that accept $r_{c_i}a$ and r_{c_j} respectively. Then apply Theorem 2.2 and Theorem 2.1 to construct two DFAs for the same two languages. Let the DFAs be M_1 and M_2 respectively. Construct M_3 such that $L(M_3) = L(M_1) \cap \overline{L(M_2)}$ applying Theorem 3.2 and Theorem 3.3. $r_{c_i}a \subseteq r_{c_j}$ iff $L(M_3) = \phi$, which is decidable by the proof of Theorem 3.7.

Now that we have a regular expression array whose LHSs are the regular expressions in interest only. The RHSs are put into a form easily convertable to prefix-recursive function definition. The base case for f is $f(\lambda) = 0$. The base case for BB is $BB(\lambda) = 0$. If $r_{c_i}a$ appears as one item on the RHS of the equation for r_{c_j}, we have $f(ux) = j$ if $f(u) = i \wedge x = a, \forall\, u \in S^*, \forall\, x \in S$. If $r_{c_i}a$ appears as one item on the RHS of the equation for r_{r_j}, we have $BB(ux) = r_{r_j}$ if $f(u) = i \wedge x = a \wedge BB(u) \neq \omega$. We can ignore the concatenated $(s_1 + \ldots + s_k)^*$ temporarily and treat r_ω the same way as any other expression. For illegal sequences we further add $f(ux) = f(u)$ otherwise $\forall\, u \in S^*, \forall x \in S$, $BB(ux) = \omega$ if $BB(u) = \omega$.

14 Prefix-recursive Function to Regular Expression

Given f and BB defined in prefix-recursive form, it is an opposite process to get regular expressions for each legal equivalence class as well as for each response.

Suppose f takes the values from 0 to $n-1$, find the first sequences in canonical order that are mapped to these values by f. They form the canonical sequence set $C = \{c_0, \ldots, c_{n-1}\}$. We will denote the regular expressions for legal equivalence classes by $r_{c_0}, \ldots, r_{c_{n-1}}$, each of which is referred to as a base regular expression, and the regular expressions for each response by $r_{r_i}, \forall\, r_i \in R$. Without loss of generality, assume $c_0 < \ldots < c_{n-1}$.

To come up with a regular expression array, we put each of the regular expressions on the LHS of an equation only. On the RHSs we consider each possible contribution in the form of either λ or a one-symbol extension of a base regular expression. λ contributes to the RHSs of r_0 and r_{c_0}. If we have $BB(ux) = r_{r_j}$ if $f(u) = i \wedge x = a \wedge BB(u) \neq \omega, \forall\, u \in S^*, \forall\, x \in S$, $r_{c_i}a$ is one item contributed to the RHS of the equation for r_{r_j}. After we have walked through the definition for BB, we further concatenate the RHS for r_ω by $(s_1 + \ldots + s_k)^*$, where $s_1, \ldots, s_k$ represent all elements in S, because of the definition of $BB(ux) = \omega$ if $BB(u) = \omega$. Similarly for each $BB(ux) = r_{r_j}$ if $f(u) = i \wedge x = a \wedge BB(u) \neq \omega, \forall\, u \in S^*, \forall\, x \in S$, if $r_{r_j} \neq \omega$, we check the corresponding entry in the definition for f. If $f(ux) = l$ if $f(u) = i \wedge x = a$, $r_i a$ is one item contributed to the RHS for r_{c_l}. At the end of this process we have a regular expression array whose LHSs are each base regular expression in interest, and the RHSs are expressions of itself and/or other base expressions. Solving the array containing equations for the base set gives us regular expressions for legal equivalence classes, which when further substituted for their notations, give us solutions for all the rest.

References

[1] John E. Hopcroft, Jeffrey D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 1979.