Towards a routing specification language: formalization of the request routing problem and finite automata with actions*

Haochen Xie
haochenx@acm.org
Nagoya University

Abstract

In this research, the author formalized the *request routing problem* and proposed a new family of finite automata, *finite automata with actions* (α-FA), that could be used to solve a subclass of the routing problem. The author also defined the *equivocality* for α-FA and discussed its decidability. Finally, the author showed that the subclass of routing problem that is solvable with α-FA is exactly the same subclass solvable with the widely deployed *regex-match then route* method.

Keywords request routing problem, finite automata with actions, finite automata, specification language, URL routing

Familiarities with the formal treatments of standard finite automata and basic knowledge of computability theory are required to understand this paper, for which the readers could consult related chapters in [3] and [4].

1 Overview

```
...
/.*.php              --> GeneralRedirectionHandler
...
/forum/topics        --> ForumTopicSummaryHandler
/forum/new-posts.php --> ForumTopicSummaryHandler
...
```

Figure 1. Regex-action list example

Request routing, *e.g.* deciding which handler should handle an HTTP[1] request in a web application server, is a common task in software. In general, we call the challenge to decide what *action* should be performed when a *request* reaches a system the *request routing problem*. Currently this task is usually addressed with a set of routing rules and a mechanism executing them. One of the most widely deployed such methods involves a list of entries, each consisting of a string pattern in regular expression[2] and a corresponding action. When a request arrives, the system extract the *characteristic string* from the request and try to match it against the pattern of each entry in the order they appear in the list. In this paper, we would refer such a list as *regex-action list*, or *RAL* in short. When the system finds a matching entry, it performs the action in the entry and the handling for the request is done. One example of such a list taken from a web application is given in Fig. 1. The situation in this example is that a PHP [7] web application is being migrated. The characteristic string for each HTTP request is its URL.

In reality, the method mentioned above is sufficient for most web applications, as regular expressions are fairly expressive. However, this method is unfortunately error-prone when the size of the rule set gets larger. Consider the example again, clearly the intention of the author of this regex-action list is to have the requests with a URL of `/forum/new-posts.php` handled by the `ForumTopicSummaryHandler`, but since the first shown entry in Fig. 1 also matches such requests and takes a higher priority, the web application would decide that `GeneralRedirectionHandler` should handle them instead. So this is a programming mistake here, and such mistakes are hard to spot if for example there are other dozens of entries in between the first and the second shown entries in Fig. 1, a situation that is not uncommon in real web applications.

Such error-proneness is due to the order-dependent nature of the regex-action list. Such nature prevents efficient implementations of modularity, which furthermore prevents concise and clear specification of routing rules in large scale web applications. There are limited *ad hoc* solutions addressing this composability problem in popular web server implementations that utilize such routing mechanism, *e.g.* the configuration language for NGINX has a `location` directive [5], but there is yet no satisfying solution.

The author believes that a carefully designed specification language for request routing would benefit numerous software projects. But since the theoretical foundation for crafting such a language is still yet to be established, the author attempted to fill this vacuum in this study by giving a formalization for the request routing problem and proposing a new family of finite automata, the *finite automata with actions*, to better model the class of routing mechanisms that are based on regular languages. The details are presented in the next section of this paper.

To the best knowledge of the author, such formal treatments of the request routing problem presented in this paper is the first of its kind. And the results paved the way for the author's attempt to create a routing specification language for web applications.

2 Detailed Approach

In this section, we present formal treatments of core concepts in this research. Extended discussions and justifications are omitted due to space constraints. A version of the request routing problem that supports multiple characteristic strings is also studied, but the results are omitted, again, due to space constraints.

Definition 2.1. Given Σ, a finite set and A, a countable set, we call a partial function $\rho : \Sigma^* \nrightarrow A$ a *request routing specification*, or *routing specification* in short, where Σ^* denotes the Kleene closure of Σ. Here, we call Σ the *alphabet*, Σ^* the *language*, and A the *action space*.

*This extended abstract is written as a submission in the undergraduate category to the Student Research Competition held at ICFP 2017. The title of the research which produced this paper is "Formulation of a routing specification language for web applications".

[1] for the definition of URL and HTTP, we refer to [1] and [2], respectively. In this paper, we only concern the *path* part of the URL of an HTTP request.

[2] for the definition of regular expression, we refer to [3]. We may use the word *regex* in referring to regular expressions. In this paper, we adopt the POSIX Basic Regular Expression notation for formulating regular expressions [6].

Definition 2.2. Given ρ, a routing specification, we define the *request routing problem*, or *routing problem* in short, for ρ to be the challenge to: (a) give a formal formulation for ρ, and (b) give an algorithmic implementation that simulates[3] ρ.

Definition 2.3. Given P, a class of routing specifications, we call a method that assigns a solution to the routing problem for each $\rho \in \mathrm{P}$ a *request routing device*, or *routing device* in short. Here, we say that such a routing device solves P, or any $\rho \in \mathrm{P}$. Furthermore, we use the notation $\mathrm{P}(DEV)$ to denote the class of routing specifications that could be solved by the routing device DEV.

Definition 2.4. Given DEV_1 and DEV_2, two routing devices, we define them to be *equivalent* if $\mathrm{P}(DEV_1) = \mathrm{P}(DEV_2)$ and write it as $DEV_1 \simeq DEV_2$.

Observe that to define a routing device, it suffices to define a language in which routing specifications can be formulated and to show that the semantic denotation of each term in the language is computable.

Definition 2.5. Given Σ, an alphabet, and A, an action space, we call l, a finite list on $\mathrm{RE}(\Sigma) \times A$, a *regex-action list*, or *RAL* in short, where $\mathrm{RE}(\Sigma)$ denotes the regular expressions over Σ. We also use $\mathrm{RAL}(\Sigma, A)$ to denote the set of all regex-action list over Σ and A.

Given $l : \mathrm{RAL}(\Sigma, A)$, we define its semantic denotation $[\![l]\!] : \Sigma^* \rightharpoonup A$ as

$$[\![l]\!](w) = \begin{cases} \alpha, & \text{for the first } \langle \phi, \alpha \rangle \in l \text{ s.t. } w \in L(\phi) \\ \bot, & \text{if there exists no } \langle \phi, \alpha \rangle \in l \text{ s.t. } w \in L(\phi) \end{cases}$$

where $L(\phi)$ denotes the language of the regular expression ϕ.

Since the list l is finite in length and the membership check for regular languages is known to be decidable, the denotation of each RAL is computable for any alphabet and action space. We thus treat the language we defined above as a routing device and denote it as *RMR*, for (the method of) *regex-match then route*.

Definition 2.6. Given Σ, A, and Q, an alphabet, action space, and finite set respectively, we define the 7-tuple

$$D = \langle \Sigma, A, Q, q_0 \in Q, \delta : Q \times \Sigma \to Q, F \subseteq Q, \tau : F \to \mathcal{P}_{\mathrm{fin}}(A) \rangle$$

to be a *deterministic finite automata with actions*, or *α-DFA* in short, and the 7-tuple

$$N = \langle \Sigma, A, Q, q_0 \in Q, \delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q), F \subseteq Q, \tau : F \to \mathcal{P}_{\mathrm{fin}}(A) \rangle$$

to be a *non-deterministic finite automata with actions*, or *α-NFA* in short, where $\mathcal{P}_{\mathrm{fin}}(S)$ denotes all finite subsets and $\mathcal{P}(S)$ denotes the powerset of S for any set S; and ϵ denotes the empty string. Here, we call Q the *set of states*, q_0 the *initial state*, δ the *transition function*, F the *set of accepting states*, and τ the *action assignment function*. Furthermore we call α-DFA and α-NFA collectively as α-FA and use the meta-variable M to range through α-FA.

The only essential difference between α-FA and standard finite automata is the addition of the action assignment function. Intuitively, this function assigns the action to be performed when an automaton lands at some accepting state after processing a string.[4]

[3] in the sense that a program p in some programming language L s.t. $[\![p]\!]^L = \rho$ exists. The expressions "having an *algorithmic implementation*" and "*computable*" are used interchangeably in this paper. We also use the expression "predicate X on set A is *decidable*" in the sense that the total function $\chi_X : A \to \{\top, \bot\}$ s.t. $\chi_X(a) = \top$ iff $X(a)$ is computable.

[4] We choose to use $\mathcal{P}_{\mathrm{fin}}(A)$ as the type of its codomain only to ease technical treatments. It is then natural to give the definitions of the behavior functions for α-FA as below.

Definition 2.7. We firstly define $\hat{\tau} : Q \times \Sigma^* \to \mathcal{P}_{\mathrm{fin}}(A)$, the *extended action assignment function* for α-DFA and α-NFA respectively as

$$\hat{\tau}_D(q, w) = \tau^*(\hat{\delta}(q, w))$$
$$\hat{\tau}_N(q, w) = \bigcup_{p \in \hat{\delta}(q, w)} \tau^*(p)$$

where $\hat{\delta}$ is just the conventional extended transition function for finite automata that has type $Q \times \Sigma^* \to Q$ for deterministic automata and type $Q \times \Sigma^* \to \mathcal{P}(Q)$ for non-deterministic automata; $\tau^* : Q \to \mathcal{P}_{\mathrm{fin}}(A)$ is defined as $\tau^*(q) = \tau(q)$ if $q \in F$ and $\emptyset$ otherwise for both types of α-FA.

We then define $\hat{a} : \Sigma^* \rightharpoonup \mathcal{P}_{\mathrm{fin}}(A)$, the *behavior function* for an α-FA as

$$\hat{a}(w) = \begin{cases} \hat{\tau}(q_0, w), & w \in L(M) \\ \bot, & w \notin L(M) \end{cases}$$

where $L(M)$ denotes the language[5] denoted by the automata M.

Definition 2.8. Given M_1 and M_2, two α-FA, we define them to be *equivalent* if $\hat{a}_{M_1} = \hat{a}_{M_2}$ and write it as $M_1 \simeq M_2$.

Lemma 2.9. *For any α-NFA N, there exists an α-DFA D s.t. $N \simeq D$.*

Proof. This can be shown with a slightly modified version of the conventional proof that shows ϵ-NFA is not more expressive than DFA with a straightforwardly extended powerset construction. □

The meaning of behavior function matches our intuition that for a given α-FA M and $w \in L(M)$, $\hat{a}(w)$ yields the action that should be performed as specified by M. Used as a routing device, $\hat{a}(w)$ should always yield exactly a singleton, or we should attribute the α-FA as ill-formed. We thus define the following property, the equivocality[6], for α-FA to identify such ill-formedness.

Definition 2.10. Given M, an α-FA, we define it to be *unequivocal* if for all $w \in L(M), |\hat{a}(w)| = 1$. Conversely, we define M to be *equivocal* if for some $w \in L(M), |\hat{a}(w)| \neq 1$, *i.e.* M is equivocal iff M is not unequivocal.

Proposition 2.11. *For any α-FA, its equivocality is decidable.*

Proof. Firstly observe that the equivocality of any α-DFA is decidable. This is because $\forall w \in L(D), |\hat{a}(w)| = 1$ is equivalent to $\forall q \in F, \left|\tau(q)\right| = 1$ for any α-DFA D. The latter is clearly decidable since τ is finite in the sense that its domain is a finite set and each element in its codomain is also a finite set.

We then conclude by observing this proposition is just a corollary of Lemma 2.9 due to the fact that if $M_1 \simeq M_2$, M_1 is equivocal iff M_2 is equivocal. □

[5] We adopt the standard definition here, where $L(M) = \{w \in \Sigma^* | \hat{\delta}(q_0, w) \in F\}$ if M is deterministic and $L(M) = \{w \in \Sigma^* | \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$ if M is non-deterministic.

[6] The naming is due to the fact that the term "ambiguity" is already taken in automata theory and has a different meaning.

[7] A few notes for this diagram:

(a) it is known that for all regular expression ϕ we could construct an ϵ-NFA N s.t. $L(N) = L(\phi)$ and N has only one accepting state, which justifies our using regular expressions as the label of transitions in this diagram.

(b) the action assignment function for the α-NFA is defined as $\tau(q_{Ai}) = \{\alpha_i\}$ as illustrated in the diagram, where q_{Ai} being the i^{th} accepting state (more

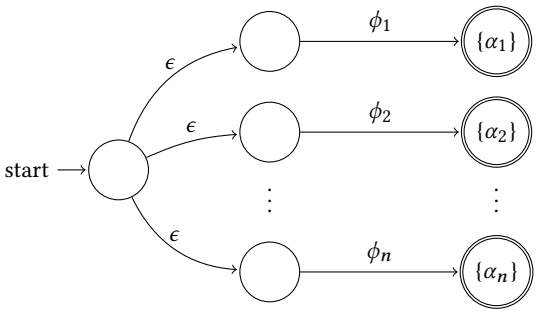

Figure 2. Canonical α-NFA construction[7], for $s = \{\langle \phi_i, \alpha_i \rangle\}_{i=1..n}$

Definition 2.12. Given Σ, an alphabet, and A, an action space, we call s, a finite set on $\mathrm{RE}(\Sigma) \times A$, a *regex-action set candidate*. We use $\mathrm{RAS}'(\Sigma, A)$ to denote the set of all such candidates.

Given $s : \mathrm{RAS}'(\Sigma, A)$, we define $M(s)$, the *canonical α-NFA* of s, to be the α-NFA illustrated in Fig. 2. We call such an s a *regex-action set*, or *RAS* in short, if its canonical α-NFA is unequivocal. We use $\mathrm{RAS}(\Sigma, A)$ to denote the set of all regex-action sets over Σ and A.

Given $s : \mathrm{RAS}(\Sigma, A)$, we define its semantic denotation $[\![s]\!] : \Sigma^* \rightharpoonup A$ as

$$[\![s]\!](w) = \alpha, \text{for } \hat{a}(w) = \{\alpha\}$$

where $\hat{a}$ is the behavior function of the canonical α-NFA of s.

Now observe that for each $s \in \mathrm{RAS}(\Sigma, A)$, its canonical α-NFA itself provides an algorithmic implementation for the routing specification $[\![s]\!]$. We thus treat the language defined above as a routing device and denote it as *FAA*, for (the method of) *finite automata with actions*.

Proposition 2.13. *Routing devices RMR and FAA are equivalent.*

Proof. It is to show that $\mathrm{P}(RMR) = \mathrm{P}(FAA)$, for which we show the two directions separately.

For the direction $\mathrm{P}(RMR) \subseteq \mathrm{P}(FAA)$, observe that for any RAL l we can always construct another RAL l' that is (a) unambiguous in the sense that the regexes of any two entries of l' denote non-intersecting regular languages and (b) that l' is equivalent to l in the sense that $[\![l]\!] = [\![l']\!]$.This could be trivially done due to the fact that the class of regular languages is closed under relative complement[8]. The unambiguous RAL l' could then be treated as a RAS denoting the same routing specification.

For the direction $\mathrm{P}(FAA) \subseteq \mathrm{P}(RMR)$, observe that for any RAS s, if we compile its element as a list l in any order, l would be a RAL that denotes the same routing specification. □

References

[1] T. Berners-Lee, R. Fielding, and L. Masinter. 2005. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Internet Standard). (Jan. 2005), 61 pages. https://doi.org/10.17487/RFC3986 Updated by RFCs 6874, 7320.

[2] R. Fielding and J. Reschke. 2014. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230 (Proposed Standard). (June 2014), 89 pages. https://doi.org/10.17487/RFC7230

[3] John. E Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2007. *Introduction to Automata Theory, Languages, and Computation* (3rd ed.). Pearson Addison Wesley.

[4] Neil D. Jones. 1997. *Computability and Complexity: From a Programming Perspective.* MIT Press, Cambridge, MA, USA.

[5] NGINX developers. 2017. *Documentation for NGINX: Module ngx_http_core_module.* https://nginx.org/en/docs/http/ngx_http_core_module.html

[6] Portable Applications Standards Committee of the IEEE Computer Society. 2016. Regular Expressions. IEEE Std 1003.1-2008, 2016 Edition. *Portable Operating System Interface (POSIX) Base Specifications Issue 7* Base Definitions (2016). http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html

[7] The PHP Group. 2017. *PHP Manual: What is PHP?* http://php.net/manual/en/intro-whatis.php

precisely, the one that has a transition onto it labeled ϕ_i); the action space for the α-NFA is clear from the the type of s.

(c) formal formulation of this construction is omitted due to space constraints but the construction itself should be clear from this illustration.

[8] *i.e.* for any regular languages L_1 and L_2, the language $\{w | w \in L_1 \text{and} w \notin L_2\}$ is also a regular language.