Technical Report 2017-001

Adapting subset construction to automata over list structures

Satoshi Okui[1], Taro Suzuki[2], Yoshiki Iwata[1]

January 10, 2018

[1]Depertment of Computer Science
Chubu University
Matsumoto 1200, Kasugai City
Aichi, 487-8501 Japan

[2]Department of Computer Science
and Engineering
The University of Aizu
Tsuruga, Ikki-Machi,
Aizu-Wakamatsu City
Fukushima, 965-8580 Japan

Technical Report 2017-001

Title:

Adapting subset construction to automata over list structures

Authors:

Satoshi Okui, Taro Suzuki and Yoshiki Iwata

Key Words and Phrases:

string matching, finite automata, subset construction, Aho-Corasick automata, KMP algorithm, Mohri's string matching algorithm

Abstract:

We investigate the subset construction (or powerset construction) introduced by Rabin and Scott seriously. Consider an NFA obtained from a DFA by allowing additional moves from the initial state to itself for any input symbols in the alphabet. Given such NFA's, we construct DFA's whose states are lists of the NFA-states such that no element occurs more than once in each list, then adapting the subset construction so as to deal with lists rather than sets. We show that such a variant of subset construction, with suitable optimizations applied, performs better than the original one in the sense that each construction step yielding a DFA-state is, in practice, irrelevant to the number of the NFA-states. We also adapt our construction so as to produce DFA's with default (or failure) transition, resulting in a new algorithm such as the classical construction of KMP-automata.

Report Date:

1/10/2018

Written Language:

English

Any Other Identifying Information of this Report:

Distribution Statement:

First Issue: 5 copies

Supplementary Notes:

Cognitive Science Laboratory
The University of Aizu
Aizu-Wakamatsu
Fukushima 965-8580
Japan

Adapting Subset Construction to Automata over List Structures

Satoshi Okui, Taro Suzuki and Yoshiki Iwata

Abstract

We investigate the subset construction (or powerset construction) introduced by Rabin and Scott seriously. Consider an NFA obtained from a DFA by allowing additional moves from the initial state to itself for any input symbols in the alphabet. Given such NFA's, we construct DFA's whose states are lists of the NFA-states such that no element occurs more than once in each list, then adapting the subset construction so as to deal with lists rather than sets. We show that such a variant of subset construction, with suitable optimizations applied, performs better than the original one in the sense that each construction step yielding a DFA-state is, in practice, irrelevant to the number of the NFA-states. We also adapt our construction so as to produce DFA's with default (or failure) transition, resulting in a new algorithm such as an extention of the classical construction of KMP-automata.

1 Introduction

The *subset construction* (or *powerset construction*) introduced by M. Rabin and D. Scott [7] is one of the most fundamental techniques in automata theory, having gained practical impact especially in compiler construction and regular expression pattern matching. For example, lexical analyzer is based on the subset construction applied for Thompson's ϵ-NFA [9]. LR-Parser [4] also benifits from it; the construction of LR automata is nothing but the subset construction for some kind of ϵ-NFA where ϵ-transition represents the reduction via production rules in context-free grammar. Recent developments of efficient greedy or leftmost-longest regular expression matching algorithms basically perform the subset construction in call-by-need manner, producing a DFA-state as long as it is really needed. There are also theoretical influences. For example, Brozozowski's DFA-minimalization [2] applies the subset construction twice. Safra's construction [8] is a generalization of the subset construction for non-deterministic Büchi automata.

The subset construction is also relevant to string matching, a very extensive area of computer science with numerous contributions. The shift-OR (or , shift-AND in dual notion) algorithm [1] can be seen as an efficient implementation of the subset construction by bit vectors provided that NFA's are given in very restricted form: conjunction of character classes. It is known that the subset construction is also relevant to Knuth-Morris-Platt string matching algorithm [5]; indeed, it produces, given an NFA came from a string pattern, a DFA isomorphic to the KMP-automaton for the given string (see [3] for example). However, to the authors' best knowledge, no researches taking that relationship seriously into account are found in the literature.

This paper takes the relevance seriously, offering a new variant of subset construction which is closely related to the construction of KMP-automata or Aho-Corasick automata. The key idea is to adapt the subset construction for lists of NFA-states rather than sets; then, the "cdr" operation dropping the first item of a list gives the failure transition. Constructing DFA's over list structures, rather than over sets, also delivers benefits in terms of computational cost. We will show that the cdr-part of a list is always found in the previous subset construction steps provided that the input NFA is made from a DFA by allowing extra transition steps from the initial state to itself. This fact allows much improvements in efficiently of the algorithm.

2 Preliminaries

For a set A, we write the direct sum $A + \{\perp\}$ as $A_\perp$ where $\perp$ represents the undefined value. For the sake of simplicity, we assume $\perp \notin A$, regarding $A_\perp$ just as $A \cup \{\perp\}$. A function $f : A \to B$ is extended to $f_\perp : A_\perp \to B_\perp$ as

$$f_\perp(a) = \begin{cases} f(a) & (a \in A) \\ \perp & (\text{Otherwise}). \end{cases}$$

For two sets A and B, $A \backslash B$ means the *set difference* $\{a \in A \mid a \notin B\}$. We need later the following laws:

$$A \backslash B = (A \cup B) \backslash B, \tag{2.1}$$

$$A \cup B = (A \backslash B) \cup B. \tag{2.2}$$

For a function $f : A \to B$ (or $f : A \to B_\perp$), we define $f : 2^A \to 2^B$ (*direct image* function) as $f(A') = \{b \in B \mid b = f(a) \text{ for some } a \in A'\}$ and $f^{-1} : 2^B \to 2^A$ (*inverse image* function) as $f^{-1}(B') = \{a \in A \mid f(a) \in B'\}$. We will frequently uses the following law:

$$f(\bigcup_{i \in I} A_i) = \bigcup_{i \in I} f(A_i) \tag{2.3}$$

where I is a possibly infinite index and $A_i \subseteq A$ $(i \in I)$. Note that (2.3) implies that f is *monotone* (or *order-preserving*): $A \subseteq B \Rightarrow f(A) \subseteq f(B)$.

3 Deterministic Automata, Homomorphisms and Subautomata

A (possibly infinite and incomplete) deterministic automaton on an *alphabet* Σ is a triple $(\mathcal{Q}, \mathcal{F}, \delta)$ where $\mathcal{Q}$ is a (possibly infinite) set of *states*, $\mathcal{F}(\subseteq \mathcal{Q})$ a set of *final* states, and δ a function from $\Sigma \times \mathcal{Q}$ to $\mathcal{Q}_\perp$ called a *transition*. We write $\delta_a(q)$ rather than $\delta(a, q)$. We often distinguish a particular state in $\mathcal{Q}$ as the *initial* state. The components of M are denoted by $\mathcal{Q}^M$, $\mathcal{F}^M$, δ^M respectively, and the initial state, if specified, is denoted by i^M. A deterministic automaton M is *complete* if $\delta_a^M(q) \neq \perp$ for any $q \in \mathcal{Q}^M$ and $a \in \Sigma$. An automaton M is *finite* if $\mathcal{Q}^M$ is finite. A (resp. finite) deterministic automaton is abbreviated as DA (resp. DFA).

Let $M_i (i = 1, 2)$ be two DA's on Σ. A function $h : \mathcal{Q}^{M_1} \to \mathcal{Q}^{M_2}$ is a *(DA-)homomorphism* from M_1 to M_2 (written as $h : M_1 \to M_2$) if $q \in \mathcal{F}^{M_1} \Leftrightarrow h(q) \in \mathcal{F}^{M_2}$ (or equivalently, $\mathcal{F}^{M_1} = h^{-1}(\mathcal{F}^{M_2})$) and

$$h_\perp(\delta_a^{M_1}(q)) = \delta_a^{M_2}(h(q)) \tag{3.1}$$

for any $q \in \mathcal{Q}_{M_1}$ and $a \in \Sigma$. Note that (3.1) implies $\delta_a^{M_1}(q) = \perp \Leftrightarrow \delta_a^{M_2}(h(q)) = \perp$ for any $q \in \mathcal{Q}_{M_1}$ and $a \in \Sigma$. It also implies

$$\bigcup_{a \in \Sigma} h(\delta_a^{M_1}(A)) = \bigcup_{a \in \Sigma} \delta_a^{M_2}(h(A)) \tag{3.2}$$

for any $A \subseteq \mathcal{Q}^{M_1}$.

We say two automata M and M' on Σ is *isomorphic* if there exist homomorphisms $f : M \to M'$ and $g : M' \to M$ such that $g \circ f = id_M$ and $f \circ g = id'_M$ where id_M (resp. id'_M) is the identity homomorphism on M (resp. M'), or equivalently, there exists a bijective homomorphism $f : M \to M'$.

Let M and M' be automata on Σ such that $\mathcal{Q}^{M'} \subseteq \mathcal{Q}^M$. We say M' is a *subautomaton* of M if the inclusion map from $\mathcal{Q}^{M'}$ to $\mathcal{Q}^M$ gives a DA-homomorphism from M' to M. Note that $\mathcal{Q}^{M'}$ is *closed* under δ_a^M; i.e., $\delta_a^M(\mathcal{Q}^{M'}) \subseteq \mathcal{Q}^{M'}$ for any $a \in \Sigma$. We often identify a subautomaton with its states, leaving the other components unspecified since they are uniquely determined by M; we abuse $\mathcal{F}^M$ and δ_a^M in the place of $\mathcal{F}^{M'}$ and $\delta_a^{M'}$.

Let M be an automaton and $S \subseteq \mathcal{Q}^M$. Consider a sequence $M\langle S\rangle_k$ $(k = 0, 1, \dots)$:

$$M\langle S\rangle_k = \begin{cases} S & (k = 0) \\ M\langle S\rangle_{k-1} \cup \bigcup_{a\in\Sigma} \delta_a(M\langle S\rangle_{k-1}) & (k > 0). \end{cases} \tag{3.3}$$

We define a subautomaton $M\langle S\rangle$ of M by $\mathcal{Q}^{M\langle S\rangle} = \bigcup_{k\geq 0} M\langle S\rangle_k$. Note that when M is finite, we have $\mathcal{Q}^{M\langle S\rangle} = M\langle S\rangle_k$ for some $k \geq 0$ since $M\langle S\rangle_k \subseteq M\langle S\rangle_{k+1}$ holds for any $k \geq 0$. It follows by induction that any subautomaton of M containing S is an upper bound of $M\langle S\rangle_k$ for any $k \geq 0$. Therefore, $M\langle S\rangle$ is the least subautomaton of M containing S; it is called the subautomaton *generated* by S. $M\langle\{q\}\rangle$ (resp. $M\langle\{q\}\rangle_k$) is simply written as $M\langle q\rangle$ (resp. $M\langle q\rangle_k$). $M\langle \mathsf{i}^M\rangle$ (resp. $M\langle \mathsf{i}^M\rangle_k$) is further abbreviated as $M\langle\rangle$ (resp. $M\langle\rangle_k$) with understanding that the initial state of $M\langle\rangle$ is i^M.

A homomorphism maps a subautomaton to a subautomaton; If $h : M_1 \to M_2$ is a homomorphism and M_1' is a subautomaton of M_1 then we can consider a subautomaton $(h(\mathcal{Q}^{M_1'}), \mathcal{F}^{M_2}, \delta_a^{M_2})$ of M_2 since $\delta_a^{M_2}(h(\mathcal{Q}^{M_1'})) = h(\delta_a^{M_1}(\mathcal{Q}^{M_1'})) \subseteq h(\mathcal{Q}^{M_1'})$, which we denote as $h(M_1')$. Moreover, the least subautomaton maps to the least one (Corollary 1 below).

Lemma 1 *$h(M_1\langle S\rangle_k) = M_2\langle h(S)\rangle_k$ for any homomorphism $h : M_1\langle S\rangle \to M_2$ and $k \geq 0$.*

PROOF. The proof is by induction on k. The case $k = 0$ is trivial. Otherwise, we have

$$\begin{aligned} h(M_1\langle S\rangle_k) &= h(M_1\langle S\rangle_{k-1}) \cup \bigcup_{a\in\Sigma} h(\delta_a^{M_1}(M_1\langle S\rangle_{k-1})) && \text{(by (3.3), (2.3))} \\ &= h(M_1\langle S\rangle_{k-1}) \cup \bigcup_{a\in\Sigma} \delta_a^{M_2}(h(M_1\langle S\rangle_{k-1})) && \text{(by (3.2))} \\ &= M_2\langle h(S)\rangle_{k-1} \cup \bigcup_{a\in\Sigma} \delta_a^{M_2}(M_2\langle h(S)\rangle_{k-1}) && \text{(by I.H.)} \\ &= M_2\langle h(S)\rangle_k && \text{(by (3.3)).} \end{aligned}$$

□

Corollary 1 *$h(M_1\langle S\rangle) = M_2\langle h(S)\rangle$ for any homomorphism $h : M_1\langle S\rangle \to M_2$.*

Consider a sequence $M\langle\langle S\rangle\rangle_k$ $(k = 0, 1, \dots)$:

$$M\langle\langle S\rangle\rangle_k = \begin{cases} S & (k = 0) \\ (\bigcup_{a\in\Sigma} \delta_a(M\langle\langle S\rangle\rangle_{k-1}))\backslash(M\langle\langle S\rangle\rangle_0 \cup \dots \cup M\langle\langle S\rangle\rangle_{k-1}) & (k > 0) \end{cases} \tag{3.4}$$

where $S \subseteq \mathcal{Q}^M$.

Lemma 2 *For any $k \geq 0$ we have (i) $M\langle S\rangle_k = M\langle\langle S\rangle\rangle_0 \cup \dots \cup M\langle\langle S\rangle\rangle_k$ and (ii) $M\langle\langle S\rangle\rangle_{k+1} = M\langle S\rangle_{k+1}\backslash M\langle S\rangle_k$.*

PROOF. Let $T_k = M\langle\langle S\rangle\rangle_0 \cup \dots \cup M\langle\langle S\rangle\rangle_k$ $(k \geq 0)$. First note that

$$T_{k+1} = T_k \cup \bigcup_{a\in\Sigma} \delta_a^M(M\langle\langle S\rangle\rangle_k) \tag{3.5}$$

holds since

$$\begin{aligned} T_k \cup \bigcup_{a\in\Sigma} \delta_a^M(M\langle\langle S\rangle\rangle_k) &= T_k \cup (\bigcup_{a\in\Sigma} \delta_a^M(M\langle\langle S\rangle\rangle_k)\backslash T_k) && \text{(by (2.2))} \\ &= T_k \cup M\langle\langle S\rangle\rangle_{k+1} && \text{(by (3.4))} \\ &= T_{k+1}. \end{aligned}$$

We next show (i) by course of values induction. The case $k = 0$ is trivial. The case $k = 1$ immediately follows by (2.2). Suppose $k \geq 2$. We have

$$\begin{aligned}
M\langle S\rangle_k &= \bigcup_{a\in\Sigma} \delta_a^M(M\langle S\rangle_{k-1}) \cup M\langle S\rangle_{k-1} \\
&= \bigcup_{a\in\Sigma} \delta_a^M(T_{k-1}) \cup T_{k-1} \\
&= \bigcup_{a\in\Sigma} \delta_a^M(T_{k-2}) \cup \bigcup_{a\in\Sigma} \delta_a^M(M\langle\!\langle S\rangle\!\rangle_{k-1}) \cup T_{k-1} \\
&= \bigcup_{a\in\Sigma} \delta_a^M(T_{k-2}) \cup T_k
\end{aligned}$$

Applying induction hypothesis twice yields

$$\bigcup_{a\in\Sigma} \delta_a^M(T_{k-2}) = \bigcup_{a\in\Sigma} \delta_a^M(M\langle S\rangle_{k-2}) \subseteq M\langle S\rangle_{k-1} = T_{k-1} \subseteq T_k,$$

which completes the proof of (i). Finally, the equation (ii) is proved from (i) as follows:

$$\begin{aligned}
M\langle S\rangle_{k+1}\backslash M\langle S\rangle_k &= T_{k+1}\backslash T_k \\
&= (T_k \cup \bigcup_{a\in\Sigma} \delta_a^M(M\langle\!\langle S\rangle\!\rangle_k))\backslash T_k \\
&= (\bigcup_{a\in\Sigma} \delta_a^M(M\langle\!\langle S\rangle\!\rangle_k))\backslash T_k \\
&= M\langle\!\langle S\rangle\!\rangle_{k+1}.
\end{aligned}$$

□

4 Non-deterministic Automata and Subset Construction

Unlike the deterministic case, the transition δ of a *non-deterministic* automaton $(\mathcal{Q}, \mathcal{F}, \delta)$ on Σ is a function from $\Sigma \times \mathcal{Q}$ to $2^{\mathcal{Q}}$. A finite non-deterministic automaton is abbreviated as NFA.

For a given NFA N on Σ, we can consider a DFA, written as $\det(N)$, such that $\mathcal{Q}^{\det(N)} = 2^{\mathcal{Q}^N}$, $\mathsf{i}^{\det(N)} = \{\mathsf{i}^N\}$ (if initial state is specified), $\mathcal{F}^{\det(N)} = \{A \subseteq \mathcal{Q}^N \mid A \cap \mathcal{F}^N \neq \emptyset\}$, and $\delta_{\det(N)} : \Sigma \times 2^{\mathcal{Q}^N} \to 2^{\mathcal{Q}^N}$ is the function:

$$\delta_a^{\det(N)}(A) = \bigcup_{q\in A} \delta_a^N(q). \tag{4.1}$$

In this paper, we assume that NFA's are given from an (incomplete) DFA M with initial state i^M by allowing an extra transition from i^M to itself for any input symbol. Precisely, this NFA, say N, consists of exactly the same components as M except for the slightly different transition

$$\delta_a^N(q) = \{\delta_a^M(q)\}\backslash\{\bot\} \cup \{\mathsf{i}^M \mid q = \mathsf{i}^M\}. \tag{4.2}$$

Thanks to the very limited source of non-determinism in N, the transition of $\det(N)\langle\rangle$ is simply given as follow:

$$\delta_a^{\det(N)}(A) = \delta_a^M(A) \cup \{\mathsf{i}^M\} \quad (A \in \mathcal{Q}^{\det(N)\langle\rangle}). \tag{4.3}$$

As a consequence, i^M is contained in any state of $\det(N)\langle\rangle$. We intensionally confuse the DFA M with the NFA N whenever they are clearly distinguished by the context; e.g., we write $\det(M)$ for a DFA M.

The *subset construction* (or *powerset construction*) [7] generally refers to the process obtaining from a given NFA N the DFA $\det(N)\langle\rangle$ by increasingly computing $\det(N)\langle\rangle_k$ $(k = 0, 1, 2, \dots)$ until no longer produce a new state (The termination is ensured as $\det(N)$ is finite). Figure 1 shows a typical subset construction

```
function CONSTRUCT
    $\Delta \leftarrow \emptyset$; $D \leftarrow \{\{\mathsf{i}^M\}\}$; $Q \leftarrow \{\{\mathsf{i}^M\}\}$;
    $F \leftarrow \{\{\mathsf{i}^M\} \mid \mathsf{i}^M \in \mathcal{F}^M\}$;
    function MOVE$(a, S)$
        $T \leftarrow \emptyset$;
        for $q \in S$ do
            $T \leftarrow T \cup \{\delta_a^M(q)\}$;
        return $T$;
    while $D \neq \emptyset$ do
        $D' \leftarrow \emptyset$;
        for $(a, S) \in \Sigma \times D$ do
            $T \leftarrow \mathrm{move}(a, S)$;
            $\Delta \leftarrow \Delta \cup \{(S, a) \mapsto T\}$;
            if $T \notin Q$ then
                $D' \leftarrow D' \cup \{T\}$;
                $Q \leftarrow Q \cup \{T\}$;
                $F \leftarrow F \cup \{T\}$ if $T \cap \mathcal{F}^M \neq \emptyset$;
        $D \leftarrow D'$;
    return $(Q, F, \Delta)$;
```

Figure 1: A subset construction algorithm

algorithm. We do not discuss the correctness of this algorithm in depth here; we only remark that owing to Lemma 2 we have for $k > 0$ $Q_k = \det(M)\langle\rangle_k$, $D_k = \det(M)\langle\rangle_k \backslash \det(M)\langle\rangle_{k-1}$, and Δ_k is $\delta^{\det(M)}$ whose domain is restricted to $\Sigma \times Q_k$ where Q_k (resp. D_k, Δ_k) is the value of Q (resp. D, Δ) just after k-times iteration of the while-loop (line 9–18).

Although the overall computational cost to construct the entire DFA at worst case is exponential, it is often sufficient to compute the values of $\delta_a^{\det(M)\langle\rangle}$ only if they are really needed at the first time (and only once). So, we are rather concerned with the cost for one step of construction generating each DFA-state; namely the cost for each step of inner loop (line 11-17); it is $\mathcal{O}(|\mathcal{Q}^M|)$ since each state in $\det(M)\langle\rangle$ consist of at most $|\mathcal{Q}^M|$ elements.

5 Automata over List Structures

For a set of state $\mathcal{Q}$, consider a list structure $\mathrm{list}(\mathcal{Q})$ on it: $\mathrm{list}(\mathcal{Q}) \cong 1 + \mathcal{Q} \times \mathrm{list}(\mathcal{Q})$. We adopt Haskell-like notation for data structures and write it as follows:

$$\mathrm{list}(\mathcal{Q}) = \mathsf{Nil} \mid \mathsf{Cons}\ \{\mathsf{fst} :: \mathcal{Q}, \mathsf{rest} :: \mathrm{list}(\mathcal{Q})\}$$

where Nil and Cons are constructors while fst and rest are deconstructors.

For a given DFA M on Σ, we introduce a complete DFA $\mathrm{list}(M)$ such that $\mathcal{Q}^{\mathrm{list}(M)} = \mathrm{list}(\mathcal{Q}^M)$, $\mathsf{i}^{\mathrm{list}(M)} = \mathsf{Nil}$, $\mathcal{F}^{\mathrm{list}(M)} = \{s \in \mathrm{list}(\mathcal{Q}^M) \mid v^M(s) \cap \mathcal{F}^M \neq \emptyset\}$, and

$$\delta_a^{\mathrm{list}(M)}(s) = \begin{cases} \delta_a^M(\mathsf{i}^M) : \mathsf{Nil} & (s = \mathsf{Nil}) \\ \delta_a^M(\mathsf{fst}(s)) : \delta_a^{\mathrm{list}(M)}(\mathsf{rest}(s)) & (\text{Otherwise}) \end{cases} \tag{5.1}$$

where $v^M : \mathrm{list}(\mathcal{Q}^M) \to 2^{\mathcal{Q}^M}$ interprets a list as the set of its elements, being inductively defined on list structure:

$$v^M(s) = \begin{cases} \{\mathsf{i}^M\} & (s = \mathsf{Nil}) \\ \{\mathsf{fst}(s)\} \cup v^M(\mathsf{rest}(s)) & (\text{Otherwise}), \end{cases} \tag{5.2}$$

and $q{:}s$ (*optional cons*) is given as follows:

$$q{:}s = \begin{cases} s & (q \in v^M(s) \cup \{\bot\}) \\ \mathsf{Cons}\ q\ s & (\text{Otherwise}). \end{cases} \tag{5.3}$$

Note that the initial state in $\mathrm{list}(M)$ is not $\mathsf{Cons}\ \mathsf{i}^M \mathsf{Nil}$ but just Nil while v^M maps the empty list Nil not to $\emptyset$ but to $\{\mathsf{i}^M\}$. This is because we prefer to choose a simpler formulation. Recall that the states in $\det(M)\langle\rangle$ always contains i^M by (4.3), so that it makes our formulation a bit simpler omitting i^M from each list in $\mathrm{list}(M)$ and instead interpreting each list just like it were always containing i^M at its tail.

We now make a slight modification on $\mathrm{list}(M)$. The naive computation of $v^M(s)$ requires $\mathcal{O}(|s|)$ time. In order to reduce it to $\mathcal{O}(1)$ we use a general technique of *memoization*; we will embed into s its own image $v^M(s)$, so that the list structure is extended as follows:

$$\mathrm{list}_m(\mathcal{Q}^M) = \mathsf{Nil_m} \mid \mathsf{Cons_m}\ \{\mathsf{fst_m} :: \mathcal{Q}^M, \mathsf{rest_m} :: \mathrm{list}_m(\mathcal{Q}^M), \mathsf{as\_set} :: 2^{\mathcal{Q}^M}\}.$$

Instead of $v^M : \mathrm{list}(\mathcal{Q}^M) \to 2^{\mathcal{Q}^M}$, we now consider the function $v^M_m : \mathrm{list}_m(\mathcal{Q}^M) \to 2^{\mathcal{Q}^M}$ just retrieving the memoized values embedded in the lists:

$$v^M_m(s) = \begin{cases} \{\mathsf{i}^M\} & (s = \mathsf{Nil_m}) \\ \mathsf{as\_set}(s) & (\text{otherwise}). \end{cases} \tag{5.4}$$

Let $e^M_m : \mathrm{list}_m(\mathcal{Q}^M) \to \mathrm{list}(\mathcal{Q}^M)$ be a function just dropping all the memoized values:

$$e^M_m(s) = \begin{cases} \mathsf{Nil} & (s = \mathsf{Nil_m}) \\ \mathsf{Cons}\ \mathsf{fst_m}(s)\ e^M_m(\mathsf{rest_m}(s)) & (\text{otherwise}). \end{cases} \tag{5.5}$$

Now the definition of the automaton $\mathrm{list}(M)$ is modified accordingly; we introduce a DFA $\mathrm{list}_m(M)$ such that $\mathcal{Q}^{\mathrm{list}_m(M)} = \mathrm{list}_m(\mathcal{Q}^M)$, $\mathsf{i}^{\mathrm{list}_m(M)} = \mathsf{Nil_m}$, $\mathcal{F}^{\mathrm{list}_m(M)} = \{s \in \mathrm{list}_m(\mathcal{Q}^M) \mid v^M(e^M_m(s)) \cap \mathcal{F}^M \neq \emptyset\}$. The transition is defined similarly:

$$\delta^{\mathrm{list}_m(M)}_a(s) = \begin{cases} \delta^M_a(\mathsf{i}^M){:}\mathsf{Nil_m} & (s = \mathsf{Nil_m}) \\ \delta^M_a(\mathsf{fst_m}(s)){:}\delta^{\mathrm{list}_m(M)}_a(\mathsf{rest_m}(s)) & (\text{Otherwise}) \end{cases} \tag{5.6}$$

but the optional cons has changed in order to maintain the memoized values so that they properly keep the image of v^M over the transitions:

$$q{:}s = \begin{cases} s & (q \in v^M(s) \cup \{\bot\}) \\ \mathsf{Cons_m}\ q\ s\ \{q\} \cup v^M_m(s) & (\text{Otherwise}). \end{cases} \tag{5.7}$$

A list $s \in \mathrm{list}_m(\mathcal{Q}^M)$ is *properly memoized* if $v^M_m(s) = v^M(e^M(s))$. A set $A \subseteq \mathrm{list}_m(\mathcal{Q}^M)$ is properly memoized if any element of A is. $\mathcal{Q}^{\mathrm{list}_m(M)}$ is in general not properly memoized. We will later show that $\mathcal{Q}^{\mathrm{list}_m(M)\langle\rangle}$ is however properly memoized so that v^M_m can be used as a computationally better replacement of v^M.

In what follows, we write $s \xrightarrow{a} t$ for $\delta^{\mathrm{list}_m(M)}_a(s) = t$. $\mathsf{Nil_m} \xrightarrow{a} t$ (resp. $s \xrightarrow{a} t$ $(s \neq \mathsf{Nil_m})$) is called a *head transition step* if $\delta^M_a(\mathsf{i}^M) \notin \{\mathsf{i}^M, \bot\}$ (resp. $\delta^M_a(\mathsf{fst}(s)) \notin \delta^{\mathrm{list}_m(M)}_a(\mathsf{rest_m}(s)) \cup \bot$). We write $s \overset{a}{\hookrightarrow} t$ if $s \xrightarrow{a} t$ is a head transition step.

Lemma 3 *Suppose* $s \xrightarrow{a} t$ *and* $t \neq \mathsf{Nil_m}$*. We have* ${\mathsf{rest_m}}^k(s) \overset{a}{\hookrightarrow} t$ *for some* $k \geq 0$*.*

Proof. By induction on the length of s. If $s \overset{a}{\hookrightarrow} t$, we take $k = 0$. Otherwise, the assumption $t \neq \mathsf{Nil_m}$ enforces $s \neq \mathsf{Nil_m}$. Applying the induction hypothesis to $\mathsf{rest_m}(s) \xrightarrow{a} t$ yields ${\mathsf{rest_m}}^k(s) \overset{a}{\hookrightarrow} t$ for some $k \geq 0$. □

For any $t \in \mathcal{Q}^{\mathrm{list}_m(M)} \backslash \{\mathsf{Nil_m}\}$, $\mathsf{REST}(t)$ denotes the set: $\{\mathsf{rest_m}(t), \mathsf{rest_m}^2(t), \ldots, \mathsf{rest_m}^n(t)\}$ where $\mathsf{rest_m}^n(t) = \mathsf{Nil_m}$.

The next theorem states the important property of $\mathrm{list}_m(M)$ that for each state in $\mathrm{list}_m(M)\langle\rangle$, its proper sublists are found in some previous subset construction step (so that we need not to reproduce them in each subset construction step).

Theorem 1 *If $t \in \mathrm{list}_m(M)\langle\rangle_k \backslash \{\mathsf{Nil_m}\}$ then $\mathsf{REST}(t) \subseteq \mathrm{list}_m(M)\langle\rangle_{k-1}$ for any $k > 0$.*

PROOF. The proof is by induction on k. In the proof, we abbreviate $\mathrm{list}_m(M)\langle\rangle_k$ as S_k. The case $k = 1$ is trivial, so we assume $k > 1$. Consider $t \in S_k \backslash \{\mathsf{Nil_m}\}$. We will first show that $\mathsf{rest_m}(t) \in S_{k-1}$. Take some $s \in S_{k-1}$ and $a \in \Sigma$ such that $s \overset{a}{\to} t$. Then Lemma 3 gives $\mathsf{rest_m}^j(s) \overset{a}{\hookrightarrow} t$ for some $j \geq 0$. Let $u = \mathsf{rest_m}^j(s)$. If $u = \mathsf{Nil_m}$, we immediately obtain $\mathsf{rest_m}(t) = \mathsf{Nil_m} \in S_{k-1}$. Hereafter, we assume $u \neq \mathsf{Nil_m}$. Consider $\mathsf{rest_m}(u) = \mathsf{rest_m}^{j+1}(s)$. Since $u \neq \mathsf{Nil_m}$ implies $s \neq \mathsf{Nil_m}$, applying the induction hypothesis to $s \in S_{k-1} \backslash \{\mathsf{Nil_m}\}$ yields $\mathsf{rest_m}(u) = \mathsf{rest_m}^{j+1}(s) \in S_{k-2}$ from which, since $u \overset{a}{\hookrightarrow} t$ implies $\mathsf{rest_m}(u) \overset{a}{\to} \mathsf{rest_m}(t)$, we obtain $\mathsf{rest_m}(t) \in S_{k-1}$. If $\mathsf{rest_m}(t) = \mathsf{Nil_m}$, we are done: $\mathsf{REST}(t) = \{\mathsf{Nil_m}\} \subseteq S_{k-1}$. Otherwise, we obtain $\mathsf{REST}(\mathsf{rest_m}(t)) \subseteq S_{k-2} \subseteq S_{k-1}$ by induction hypothesis, which concludes $\mathsf{REST}(t) = \mathsf{REST}(\mathsf{rest_m}(t)) \cup \{\mathsf{rest_m}(t)\} \subseteq S_{k-1}$. □

Now, we will confirm that $v_m^M : \mathrm{list}_m(M)\langle\rangle \to \det(M)$ is indeed a homomorphism. Before that, we needs a few lemmata.

Lemma 4 *$\{\mathsf{fst_m}(t)\} \cup v_m^M(\mathsf{rest_m}(t)) = v_m^M(t)$ for any $t \in \mathrm{list}_m(M)\langle\rangle \backslash \{\mathsf{Nil_m}\}$.*

PROOF. t is reachable from another state. Hence, Lemma 3 ensures that we can find some s and $a \in \Sigma$ such that $s \overset{a}{\hookrightarrow} t$. Hence, the result follows by the definition (5.7). □

The following lemma states that the set value embedded in each state $s \in \mathcal{Q}^{\mathrm{list}_m(M)\langle\rangle}$ is indeed $v^M(s)$ in $2^{\mathcal{Q}^M}$.

Lemma 5 *$\mathcal{Q}^{\mathrm{list}_m(M)\langle\rangle}$ is properly memoized.*

PROOF. We will show by induction on k that $\forall k \geq 0, \forall t \in S_k, v^M(e^M(t)) = v_m^M(t)$ where S_k is the abbreviation of $\mathrm{list}_m(M)\langle\rangle_k$; the result then follows since $t \in S_k$ for some $k \geq 0$. The case $t = \mathsf{Nil_m}$, which covers the case $k = 0$, is trivial: $v^M(e^M(t)) = \{\mathsf{i}^M\} = v_m^M(t)$. Now, we treat the remaining case: $t \neq \mathsf{Nil_m}$ and $k > 0$. Since $\mathsf{rest_m}(t) \in S_{k-1}$ by Theorem 1, we have $v^M(e^M(\mathsf{rest_m}(t)) = v_m^M(\mathsf{rest_m}(t))$ by the induction hypothesis. Therefore, we have

$$\begin{aligned} v^M(e^M(t)) &= \{\mathsf{fst_m}(t)\} \cup v^M(e^M(\mathsf{rest_m}(t))) && \text{(by (5.2),(5.5))} \\ &= \{\mathsf{fst_m}(t)\} \cup v_m^M(\mathsf{rest_m}(t)) && \text{(by I.H.)} \\ &= v_m^M(t) && \text{(by Lemma 4).} \end{aligned}$$

□

Proposition 1 *$v_m^M : \mathrm{list}_m(M)\langle\rangle \to \det(M)$ is a homomorphism for any DFA M.*

PROOF. First we have $s \in \mathcal{F}^{\mathrm{list}_m(M)} \Leftrightarrow v^M(e_m^M(s)) \cap \mathcal{F}^M \neq \emptyset \Leftrightarrow v^M(e_m^M(s)) \in \mathcal{F}^{\det(M)}$ for any $s \in \mathcal{Q}^{\mathrm{list}_m(M)\langle\rangle}$. We will show $(v^M \circ e_m^M)_\perp(\delta_a^{\mathrm{list}_m(M)}(s)) = \delta_a^{\det(M)}(v^M(e_m^M(s)))$ for any $s \in \mathcal{Q}^{\mathrm{list}_m(M)\langle\rangle}$ by induction on the list structure; then the result follows by Lemma 5. The base case holds since

$$\begin{aligned} (v^M \circ e_m^M)_\perp(\delta_a^{\mathrm{list}_m(M)}(\mathsf{Nil_m})) &= (\{\delta_a^M(\mathsf{i}^M)\} \backslash \{\perp\}) \cup \{\mathsf{i}^M\} \\ &= \delta_a^{\det(M)}(\mathsf{i}^M) && \text{(by (4.1), (4.2))} \\ &= \delta_a^{\det(M)}(v^M(e_m^M(\mathsf{Nil_m}))) && \text{(by (5.5), (5.2))} \end{aligned}$$

```
function CONSTRUCT
    $\Delta \leftarrow \emptyset$; $D \leftarrow \{\mathsf{Nil_m}\}$; $Q \leftarrow \{\mathsf{Nil_m}\}$; $F \leftarrow \{\mathsf{Nil_m} \mid \mathsf{i}^M \in \mathcal{F}^M\}$;
    function MOVE($a, s, \Delta$)
        return $\delta_a(\mathsf{i}^M)$:$\mathsf{Nil_m}$ if $s = \mathsf{Nil_m}$ else $\delta_a(\mathsf{fst_m}(s))$:$\Delta(a, \mathsf{rest_m}(s))$;
    while $D \neq \emptyset$ do
        $D' \leftarrow \emptyset$;
        for $(a, s) \in \Sigma \times D$ do
            $t \leftarrow \mathrm{move}(a, s, \Delta)$;
            $\Delta \leftarrow \Delta \cup \{(a, s) \mapsto t\}$;
            $D' \leftarrow D' \cup \{t\}$ if $t \notin Q$;
        $D \leftarrow D'$;
        $Q \leftarrow Q \cup D$;
        $F \leftarrow F \cup \{s \in D \mid \mathsf{fst_m}(s) \in \mathcal{F}^M \vee \mathsf{rest_m}(s) \in F\}$;
    return $(Q, F, \Delta)$;
```

Figure 2: Construction of $\mathrm{list}_m(M)\langle\rangle$

for any $a \in \Sigma$ where the first equation follows by (5.6), (5.7), (5.5) and (5.2) according to $\delta_a^M(\mathsf{i}^M) = \bot$ or not. For the induction step case, we have

$$\begin{aligned}
(v^M \circ e_m^M)_\bot(\delta_a^{\mathrm{list}(M)}(\mathsf{Cons_m}\ q\ s)) &= (\{\delta_a^M(q)\}\backslash\{\bot\}) \cup v^M(e_m^M(\delta_a^{\mathrm{list}(M)}(s))) \\
&= (\{\delta_a^M(q)\}\backslash\{\bot\}) \cup \{\mathsf{i}^M \mid q = \mathsf{i}^M\} \cup v^M(e_m^M(\delta_a^{\mathrm{list}(M)}(s))) \\
&= (\{\delta_a^M(q)\}\backslash\{\bot\}) \cup \{\mathsf{i}^M \mid q = \mathsf{i}^M\} \cup \delta_a^{\det(M)}(v^M(e_m^M(s))) \quad \text{(by I.H.)} \\
&= \delta_a^{\det(M)}(\{q\} \cup v^M(e_m^M(s))) \\
&= \delta_a^{\det(M)}(v^M(e_m^M((\mathsf{Cons_m}\ q\ s)))
\end{aligned}$$

for any $a \in \Sigma$ where the first equation follows by (5.6), (5.7), (5.5) and (5.2) according to $\delta_a^M(\mathsf{i}^M) = \bot$ or not. □

Figure 2 presents our algorithm constructing the DFA $\mathrm{list}_m(M)\langle\rangle$ for a given M. In that, the function `move` is a straightforward implemnetion of the transition (5.6). The only but crucial difference is that we refer the transition steps constructed in the previous itaration rather than recomputing them. The justification will be given later.

Let Q_k be the value of Q just after k-times iteration of the while loop (line 6–13). D_k and Δ_k are defined similarly. It is easy to observe that

$$Q_k = \begin{cases} D_0 & (k = 0) \\ Q_{k-1} \cup D_k & (k > 0), \end{cases} \tag{5.8}$$

$$D_k = \begin{cases} \{\mathsf{Nil_m}\} & (k = 0) \\ \{\mathrm{move}(a, s, \Delta_{k-1}) \mid a \in \Sigma, s \in D_{k-1}\}\backslash Q_{k-1} & (k > 0), \end{cases} \tag{5.9}$$

$$\Delta_k = \begin{cases} \emptyset & (k = 0) \\ \Delta_{k-1} \cup \{(a, s) \mapsto \mathrm{move}(a, s, \Delta_{k-1}) \mid a \in \Sigma, s \in D_{k-1}\} & (k > 0), \end{cases} \tag{5.10}$$

$$F_k = \begin{cases} \{\mathsf{Nil_m} \mid \mathsf{i}^M \in \mathcal{F}^M\} & (k = 0) \\ F_{k-1} \cup \{s \in D_k \mid \mathsf{fst_m}(s) \in \mathcal{F}^M \vee \mathsf{rest_m}(s) \in F_{k-1}\} & (k > 0). \end{cases} \tag{5.11}$$

To show the correctness of the algorithm, we first confirm that the set Δ_k $(k \geq 0)$ is a well-defined function.

Lemma 6 *For any $k \geq 0$ we have the following:*

(i) $\mathrm{Dom}(\Delta_{k+1}) \subseteq \Sigma \times Q_k$*;*

(ii) $Q_k \cap D_{k+1} = \emptyset$*;*

(iii) $\mathsf{Nil_m} \notin D_{k+1}$*.*

PROOF. We first prove (i) by induction. The case $k = 0$ is trivial: $\mathrm{Dom}(\Delta_1) = \Sigma \times D_0 = \Sigma \times Q_0$. For $k > 0$, we have

$$\begin{aligned}\mathrm{Dom}(\Delta_{k+1}) &= \mathrm{Dom}(\Delta_k) \cup \Sigma \times D_k && \text{(by (5.10))}\\ &\subseteq (\Sigma \times Q_{k-1}) \cup (\Sigma \times D_k) && \text{(by I.H.)}\\ &= \Sigma \times Q_k && \text{(by (5.8)).}\end{aligned}$$

On the other hand, (5.9) implies that $D_{k+1} \subseteq \mathcal{Q}^M \backslash Q_k$ $(k \geq 0)$, from which (ii) and (iii) immediately follows. □

From (i) and (ii) we see that Δ_k $(k \geq 0)$ are in fact functions, while (iii) ensures $s \neq \mathsf{Nil_m}$ in line 13 so that it is reasonable to refer to $\mathsf{fst_m}(s)$ and $\mathsf{rest_m}(s)$.

Next, we show that $\Delta_k(a, \_)$ indeed coincides with $\delta_a^{\mathrm{list}_m(M)}$ whose domain is restricted to Q_{k-1}. The proof requires to show $D_k = \mathrm{list}_m(M)\langle\!\langle\rangle\!\rangle_k$ and $Q_k = \mathrm{list}_m(M)\langle\rangle_k$ all together.

Lemma 7 *For any* $k \geq 0$*,*

(i) $D_k = \mathrm{list}_m(M)\langle\!\langle\rangle\!\rangle_k$*,*

(ii) $Q_k = \mathrm{list}_m(M)\langle\rangle_k$*, and*

(iii) $\Delta_{k+1}(a, s) = \delta_a^{\mathrm{list}_m(M)}(s)$ *for any* $a \in \Sigma$ *and* $s \in Q_k$*.*

(iv) $F_k = Q_k \cap \mathcal{F}^{\mathrm{list}_m(M)}$

PROOF. We proceed by the induction on k. First consider the case $k = 0$. (i), (ii) and (iv) are obvious while (iii) immediately follows by (5.10) as follows:

$$\Delta_1(a, \mathsf{Nil_m}) = \mathrm{move}(a, \mathsf{Nil_m}, \Delta_0) = \delta_a(\mathsf{i}^M){:}\mathsf{Nil_m} = \delta_a^{\mathrm{list}_m(M)}(\mathsf{Nil_m})$$

Next, suppose $k > 0$. We obtain (i) as follows:

$$\begin{aligned}D_k &= \{\mathrm{move}(a, s, \Delta_{k-1}) \mid a \in \Sigma, s \in D_{k-1}\} \backslash Q_{k-1}\\ &= \{\Delta_k(a, s) \mid a \in \Sigma, s \in D_{k-1}\} \backslash Q_{k-1}\\ &= \bigcup_{a \in \Sigma} \delta_a^{\mathrm{list}_m(M)}(D_{k-1}) \backslash Q_{k-1}\\ &= \bigcup_{a \in \Sigma} \delta_a^{\mathrm{list}_m(M)}(D_{k-1}) \backslash (D_0 \cup \cdots \cup D_{k-1})\\ &= \bigcup_{a \in \Sigma} \delta_a^{\mathrm{list}_m(M)}(M\langle\!\langle\rangle\!\rangle_{k-1}) \backslash (\mathrm{list}_m(M)\langle\!\langle\rangle\!\rangle_0 \cup \cdots \cup \mathrm{list}_m(M)\langle\!\langle\rangle\!\rangle_{k-1})\\ &= \mathrm{list}_m(M)\langle\!\langle\rangle\!\rangle_k\end{aligned}$$

For (ii), we have

$$\begin{aligned}Q_k &= D_0 \cup \cdots \cup D_k\\ &= \mathrm{list}_m(M)\langle\!\langle\rangle\!\rangle_0 \cup \cdots \cup \mathrm{list}_m(M)\langle\!\langle\rangle\!\rangle_k\\ &= \mathrm{list}_m(M)\langle\rangle_k\end{aligned}$$

For (iii), let $a \in \Sigma$ and $s \in Q_k = Q_{k-1} \cup D_k$. We distinguish two cases: either $s \in D_k$ or not. First, we treat the former case. Lemma 6 gives $(a, s) \notin \mathrm{Dom}(\Delta_k)$ and $s \neq \mathsf{Nil_m}$. Since $s \in Q_k \backslash \{\mathsf{Nil_m}\}$, Theorem 1 and (ii) gives $\mathsf{rest_m}(s) \in Q_{k-1}$. Using these facts, we obtain

$$\begin{aligned}
\Delta_{k+1}(a, s) &= \mathrm{move}(a, s, \Delta_k) && \text{(by } (a, s) \notin \mathrm{Dom}(\Delta_k)\text{, (5.10))}\\
&= \delta_a(\mathsf{fst_m}(s)){:}\Delta_k(a, \mathsf{rest_m}(s)) && \text{(by } s \neq \mathsf{Nil_m}\text{)}\\
&= \delta_a(\mathsf{fst_m}(s)){:}\delta_a^{\mathrm{list}_m(M)}(\mathsf{rest_m}(s)) && \text{(by I.H.)}\\
&= \delta_a^{\mathrm{list}_m(M)}(s) && \text{(by (5.6))}
\end{aligned}$$

In the latter case, we immediately obtain the result using (5.10) followed by the induction hypothesis:

$$\Delta_{k+1}(a, s) = \Delta_k(a, s) = \delta_a^{\mathrm{list}_m(M)}(s).$$

For (vi), consider $s \in D_k$. We have $s \in Q_k \backslash \{\mathsf{Nil_m}\}$ by (5.8) and Lemma 6 (iii), hence $\mathsf{rest_m}(s) \in Q_{k-1}$ by Theorem 1 and (ii). It also follows that

$$s \in \mathcal{F}^{\mathrm{list}_m(M)} \Leftrightarrow \mathsf{fst_m}(s) \in \mathcal{F}^M \vee \mathsf{rest_m}(s) \in \mathcal{F}^{\mathrm{list}_m(M)}. \tag{5.12}$$

Using these facts, we obtain

$$\begin{aligned}
F_k &= F_{k-1} \cup \{s \in D_k \mid \mathsf{fst_m}(s) \in \mathcal{F}^M \vee \mathsf{rest_m}(s) \in F_{k-1}\} && \text{(by (5.11))}\\
&= (Q_{k-1} \cap \mathcal{F}^{\mathrm{list}_m(M)}) \cup \{s \in D_k \mid \mathsf{fst_m}(s) \in \mathcal{F}^M \vee \mathsf{rest_m}(s) \in Q_{k-1} \cap \mathcal{F}^{\mathrm{list}_m(M)}\} && \text{(by I.H.)}\\
&= (Q_{k-1} \cap \mathcal{F}^{\mathrm{list}_m(M)}) \cup (D_k \cap \mathcal{F}^{\mathrm{list}_m(M)}) && \text{(by } \mathsf{rest_m}(s) \in Q_{k-1}\text{, (5.12))}\\
&= Q_k \cap \mathcal{F}^{\mathrm{list}_m(M)} && \text{(by (5.8)).}
\end{aligned}$$

□

Theorem 2 *The algorithm* `construct_DFA` *always terminates, producing the DFA* $\mathrm{list}_m(M)\langle\rangle$.

Proof. Since $\mathrm{list}_m(M)\langle\rangle$ is finite, $\mathrm{list}_m(M)\langle\rangle_k \backslash \mathrm{list}_m(M)\langle\rangle_{k-1}$ eventually becomes empty for some k, which implies by Lemma 2 (ii) and Lemma 7 (i) that D_k is empty; we finally escape from the loop, obtaining $Q_k(= Q_{k-1})$ and Δ_k. We have $Q_k = \mathrm{list}_m(M)\langle\rangle_k = \mathcal{Q}^{\mathrm{list}_m(M)\langle\rangle}$ by Lemma 7 (ii) and hence F gives $\mathcal{F}^{\mathrm{list}_m(M)}$ by Lemma 7 (iv). We also have $\Delta_k(a, s) = \delta_a^{\mathrm{list}_m(M)}(s)$ for $a \in \Sigma$, and $s \in \mathcal{Q}^{\mathrm{list}_m(M)}$ by Lemma 7 (iii). Therefore, the algorithm returns the DFA $\mathrm{list}_m(M)\langle\rangle$ as desired. □

Figure 3 shows a slightly modified version of the algorithm which eagerly updates Q and F in the innermost loop. This modification does not affect the resulting values. To be precise, consider the first iteration of the while loop in Figure 2. Let Q_0 be the value of Q and D'_k (resp. t_k) be the values of D' (resp. t) just after k-times iterations of the innermost loop. For $k > 0$ we have $D'_k = D'_{k-1} \cup \{t_k\}$ if $t_k \in Q_0$ or else $D'_k = D'_{k-1}$. Similaly, let $\hat{D}_k$ (resp. $\hat{Q}_k$) be the values of D and Q just after k-times iterations of the innermost loop in Figure 3. For $k > 0$ we have $\hat{Q}'_k = \hat{Q}'_{k-1} \cup \{t_k\}$ if $t_k \in \hat{Q}_{k-1}$ or else $\hat{Q}'_k = \hat{Q}'_{k-1}$. and similarly $\hat{D}'_k = \hat{D}'_{k-1} \cup \{t_k\}$ if $t_k \in \hat{Q}_{k-1}$ or else $\hat{D}'_k = \hat{D}'_{k-1}$. An easy induction yields $\hat{D}'_k = D'_k$ and $\hat{Q}_k = Q_0 \cup D'_k$ for $k \geq 0$ (Note that we have $D'_{k-1} \cup \{t_k\} = D'_{k-1}$ in case $t_k \in D'_{k-1}(= \hat{D}'_{k-1} \subseteq \hat{Q}_{k-1})$). Therefore, we can conclude by iteration that the values of D (resp. Q) at each end of while loop coincide both in Figures 2 and 3.

Now let us discuss the computational cost of the algorithm in Figure 3, especially, the cost concerning with the steps executed in the inner loop (lines 8–13). We aim to state that this cost is $\mathcal{O}(1)$. To this end, consider the usual representation of linked lists using cons-cells and pointers. Since we need to embedded to the memoized set values, the structure of cons-cells is given as follow in C/C++ notation:

```
    struct cell { State fst; *cell rest; SetOfState as_set; }
```

where `State` (state of M) is an integer type of enough size whereas `SetOfState` a set of states implemented by HAMT-based immutable (i.e., persistent) hash map as mentioned above. We assume that **new** cell(q, x, Q)

```
1: function CONSTRUCT
2:     $\Delta \leftarrow \emptyset$; $D \leftarrow \{\mathsf{Nil_m}\}$; $Q \leftarrow \{\mathsf{Nil_m}\}$; $F \leftarrow \{\mathsf{Nil_m} \mid \mathsf{i}^M \in \mathcal{F}^M\}$;
3:     function MOVE$(a, s, \Delta)$
4:         return $\delta_a(\mathsf{i}^M)$:$\mathsf{Nil_m}$ if $s = \mathsf{Nil_m}$ else $\delta_a(\mathsf{fst_m}(s))$:$\Delta(a, \mathsf{rest_m}(s))$;
5:     while $D \neq \emptyset$ do
6:         $D' \leftarrow \emptyset$;
7:         for $(a, s) \in \Sigma \times D$ do
8:             $t \leftarrow \text{move}(a, s, \Delta)$;
9:             $\Delta \leftarrow \Delta \cup \{(a, s) \mapsto t\}$;
10:            if $t \notin Q$ then
11:                $D' \leftarrow D' \cup \{t\}$;
12:                $Q \leftarrow Q \cup \{t\}$;
13:                $F \leftarrow F \cup \{t\}$ if $\mathsf{fst_m}(t) \in \mathcal{F}^M \vee \mathsf{rest_m}(t) \in F$;
14:        $D \leftarrow D'$;
15:    return $(Q, F, \Delta)$;
```

Figure 3: An alternative updating Q and F in the innermost loop

```
1: function OPTCONS$(q, x)$
2:     return $x$ if $q \in v^M(x) \cup \{\bot\}$;
3:     if $H(q, x) = \bot$ then
4:         $H \leftarrow H \cup \{(q, x) \mapsto$ new $\text{cell}(q, x, \{q\} \cup v^M(x))\}$;
5:     return $H(q, x)$;
```

Figure 4: The optional cons operation

allocates a new cons-cell, returning a pointer to it; the fst, rest, as_set fields are set to q, x, Q respectively. Then, the destructores are implemented as usual:

$$\begin{aligned} \mathsf{fst_m}(x) &= x\text{->}\,\text{fst} \\ \mathsf{rest_m}(x) &= x\text{->}\,\text{rest} \\ \mathsf{as\_set}(x) &= x\text{->}\,\text{as\_set} \end{aligned}$$

where x is a pointer to a cons-cell. We borrow a C/C++ notation x-> f meaning $(*x)$.f (dereferencing followed by a field selection).

We consider lists as immutable (persistent) data structure, which means that we never modify the fields of cons-cells once they are created. As a result, no cycle such as $x, x\text{->}\,\text{rest}, \dots, x \cdots \text{->}\,\text{rest}\text{->}\,\text{rest} = x$ ocuurs, hence we can consider the *length* of x; the length of *nil* is zero; the length of **new** cell$(q, x, \{q\} \cup x\text{->}\,\text{as_set})$ is $n + 1$. This allows us to consider the list $\ell(x)$ denoted by x:

$$\ell(x) = \begin{cases} \mathsf{Nil_m} & (x = nil) \\ \mathsf{Cons_m}\ \mathsf{fst_m}(x)\ \ell(\mathsf{rest_m}(x))\ \mathsf{as\_set}(x) & (\text{otherwise}). \end{cases} \tag{5.13}$$

The implementation of the optional cons (:) is slightly unusual. We create a new cons-cell only if it has never been created before. In the following implementation, H is a (initially empty) map taking a pair of state and pointer to a pointer (Figure 4).

6 Automata with Default Transition

A deterministic automata with *default transition* on an alphabet Σ is a DA M with initial state equipped with a function $\mathsf{fail} : \mathcal{Q}^M \backslash \{\mathsf{i}^M\} \rightarrow \mathcal{Q}^M$ called a *failure* transition. We often write δ^M as goto^M and call it a *success* transition. We abbreviate a DA (DFA) with default transition as DAf (DFAf). A DAf is regarded as a DA if we forget the failure transition so that the notions defined for DA in the previous chapters also make sense for DAf.

Let M and M' be DAf's. A DA-homomorphism $f : \mathcal{Q}^M \to \mathcal{Q}^{M'}$ is called a DAf-homomorphism if it additionally satisfies $f(\mathsf{fail}^M(q)) = \mathsf{fail}^{M'}(f(q))$ for any $q \in \mathcal{Q}^M \backslash \{\mathsf{i}^M\}$. DAf-isomorphism is definied similary to the DA case.

The failure transition is used *only if* the value of success transition is undefined; it serves as "default" transition or "fall back." This idea introduces the notion of derivablity. Let M be a DAf on Σ. We say M *derives* a DA M' with initial state if $\mathcal{Q}^{M'} = \mathcal{Q}^M$, $\mathcal{F}^{M'} = \mathcal{F}^M$, $\mathsf{i}^{M'} = \mathsf{i}^M$ and finally $\delta^{M'}$ satisfies the following equation:

$$\delta_a^{M'}(q) = \begin{cases} \mathsf{goto}_a^M(q) & (\mathsf{goto}_a^M(q) \neq \bot) \\ \mathsf{i}^M & (\mathsf{goto}_a^M(q) = \bot \wedge q = \mathsf{i}^M) \\ \delta_a^{M'}(\mathsf{fail}^M(q)) & (\mathsf{goto}_a^M(q) = \bot \wedge q \neq \mathsf{i}^M). \end{cases} \tag{6.1}$$

To make this a proper inductive definition, we require that the sequence $q, \mathsf{fail}^M(q), \mathsf{fail}^M(\mathsf{fail}^M(q)), \ldots$ eventually reaches to i^M. A DFA$_f$ is a nice replacement of the DFA it derives in such a case that Σ is huge and the domain of goto can be kept considerably smaller than that of the transition of the DFA. (The cost is to compute δ^M from goto^M and fail^M for each input word, requiring amortized constant time as we will see later.)

For a given DFA M with initial state, we consider a DAf $\mathrm{list}_f(M)$ which is same as $\mathrm{list}_m(M)$ equipped with the initial state $\mathsf{Nil_m}$ and the failure transition $\mathsf{rest_m}$ except that the (success) transition is restricted to the head transition steps; the optional cons operation (5.7) is slightly changed as follows:

$$q{:}s = \begin{cases} \bot & (q \in v^M(s) \cup \{\bot\}) \\ \mathsf{Cons_m}\ q\ s\ \{q\} \cup v_m^M(s) & (\text{Otherwise}). \end{cases} \tag{6.2}$$

In other words, $\mathsf{goto}^{\mathrm{list}_f(M)}$ is given as

$$\mathsf{goto}_a^{\mathrm{list}_f(M)}(s) = \begin{cases} t & (s \overset{a}{\hookrightarrow} t) \\ \bot & (\text{Otherwise}). \end{cases} \tag{6.3}$$

Lemma 8 *$\mathcal{Q}^{\mathrm{list}_f(M)\langle k\rangle} = \mathcal{Q}^{\mathrm{list}_m(M)\langle k\rangle}$ for any $k \geq 0$.*

Proof. The proof is by induction on k. The case $k = 0$ is trivial: $\mathcal{Q}^{\mathrm{list}_f(M)\langle 0\rangle} = \{\mathsf{Nil_m}\} = \mathcal{Q}^{\mathrm{list}_m(M)\langle 0\rangle}$. Suppose $k > 0$. To show $\mathcal{Q}^{\mathrm{list}_f(M)\langle k\rangle} \supseteq \mathcal{Q}^{\mathrm{list}_m(M)\langle k\rangle}$, consider $q \in \mathcal{Q}^{\mathrm{list}_m(M)\langle k\rangle}$. There exists some $p \in \mathcal{Q}^{\mathrm{list}_m(M)\langle k-1\rangle}$ and $a \in \Sigma$ such that $p \xrightarrow{a} q$. Theorem 1 and Lemma 3 give some $p' \in \mathcal{Q}^{\mathrm{list}_m(M)\langle k-1\rangle}$ such that $p' \overset{a}{\hookrightarrow} q$. Since $p' \in \mathcal{Q}^{\mathrm{list}_f(M)\langle k-1\rangle}$ by induction hypothesis, we conclude $q \in \mathcal{Q}^{\mathrm{list}_f(M)\langle k\rangle}$. The converse case $\mathcal{Q}^{\mathrm{list}_f(M)\langle k\rangle} \subseteq \mathcal{Q}^{\mathrm{list}_m(M)\langle k\rangle}$ immediately follows by induction hypothesis. □

As an immediate consequence of the above lamma, we have a counterpart of Theorem 1 for $\mathrm{list}_f(M)$:

Theorem 3 *If $t \in \mathrm{list}_f(M)\langle\rangle_k \backslash \{\mathsf{Nil_m}\}$ then $\mathsf{REST}(t) \subseteq \mathrm{list}_f(M)\langle\rangle_{k-1}$ for any $k > 0$.*

Lemma 9 *$\mathrm{list}_f(M)\langle\rangle$ derives $\mathrm{list}_m(M)\langle\rangle$.*

Proof. We consider three cases according to (6.1). Let $a \in \Sigma$ and $s \in \mathrm{list}_m(()\mathcal{Q}^M)$. (1) Suppose $\mathsf{goto}_a^{\mathrm{list}_f(M)\langle\rangle}(s) \neq \bot$. In this case, (5.7) and (6.2) coincide. Hence, we immediately have $\delta_a^{\mathrm{list}_m(M)\langle\rangle}(s) = \mathsf{goto}_a^{\mathrm{list}_f(M)\langle\rangle}(s)$. Otherwise, (5.6) gives $\delta_a^{\mathrm{list}_m(M)\langle\rangle}(\mathsf{Nil_m}) = \mathsf{Nil_m}$ when $s = \mathsf{Nil_m}$ and $\delta_a^{\mathrm{list}_m(M)\langle\rangle}(s) = \delta_a^{\mathrm{list}_m(M)}(\mathsf{rest_m}(s))$ when $s \neq \mathsf{Nil_m}$. □

Lemma 10 *For any $k \geq 0$,*

(i) $D_k = \mathrm{list}_m(M)\langle\langle\rangle\rangle_k$,

```
function CONSTRUCT
    $\Delta \leftarrow \emptyset$; $D \leftarrow \{\mathsf{Nil_m}\}$; $Q \leftarrow \{\mathsf{Nil_m}\}$; $F \leftarrow \{\mathsf{Nil_m} \mid \mathsf{i}^M \in \mathcal{F}^M\}$;
    function DERIVE($a, s, \Delta$)
        return $\Delta(a, s)$ if $\Delta(a, s) \neq \bot$;
        return $\mathsf{Nil_m}$ if $s = \mathsf{Nil_m}$;
        return derive($a, \mathsf{rest_m}(s), \Delta$);
    function MOVE($a, s, \Delta$)
        return $\delta_a(\mathsf{i}^M)$:$\mathsf{Nil_m}$ if $s = \mathsf{Nil_m}$ else $\delta_a(\mathsf{fst_m}(s))$:derive($a, \mathsf{rest_m}(s), \Delta$);
    while $D \neq \emptyset$ do
        $D' \leftarrow \emptyset$;
        for $(a, s) \in \Sigma \times D$ do
            $t \leftarrow$ move($a, s, \Delta$);
            continue if $t \in \bot$;
            $\Delta \leftarrow \Delta \cup \{(a, s) \mapsto t\}$;
            $D' \leftarrow D' \cup \{t\}$ if $t \notin Q$;
        $D \leftarrow D'$;
        $Q \leftarrow Q \cup D$;
        $F \leftarrow F \cup \{s \in D \mid \mathsf{fst_m}(s) \in \mathcal{F}^M \vee \mathsf{rest_m}(s) \in F\}$;
    return $(Q, F, \Delta)$;
```

Figure 5: Construction of DFA with default transitions

(ii) $Q_k = \mathrm{list}_m(M)\langle\rangle_k$, *and*

(iii) $\Delta_{k+1}(a, s) = \mathsf{goto}_a^{\mathrm{list}_f(M)}(s)$ *for any* $a \in \Sigma$ *and* $s \in Q_k$.

(iv) $F_k = Q_k \cap \mathcal{F}^{\mathrm{list}_m(M)}$

Proof. We proceed by the induction on k. First consider the case $k = 0$. (i), (ii) and (iv) are obvious while (iii) immediately follows by (5.10) as follows:

$$\Delta_1(a, \mathsf{Nil_m}) = \mathrm{move}(a, \mathsf{Nil_m}, \Delta_0) = \delta_a(\mathsf{i}^M){:}\mathsf{Nil_m} = \mathsf{goto}_a^{\mathrm{list}_m(M)}(\mathsf{Nil_m})$$

Next, suppose $k > 0$. The proofs of (i), (ii) and (iv) are the same as Lemma 7. (Note that Lemma 6 also holds for this time.) For (iii), let $a \in \Sigma$ and $s \in Q_k = Q_{k-1} \cup D_k$. We distinguish two cases: either $s \in D_k$ or not. First, we treat the former case. Lemma 6 gives $(a, s) \notin \mathrm{Dom}(\Delta_k)$ and $s \neq \mathsf{Nil_m}$. Since $s \in Q_k \backslash \{\mathsf{Nil_m}\}$, Theorem 3 and (ii) gives $\mathsf{rest_m}(s) \in Q_{k-1}$. Using these facts, we obtain

$$\begin{aligned}
\Delta_{k+1}(a, s) &= \mathrm{move}(a, s, \Delta_k) && \text{(by } (a, s) \notin \mathrm{Dom}(\Delta_k)\text{, (5.10))}\\
&= \delta_a(\mathsf{fst_m}(s)){:}\mathrm{derive}(a, \mathsf{rest_m}(s), \Delta_k) && \text{(by } s \neq \mathsf{Nil_m}\text{)}\\
&= \delta_a(\mathsf{fst_m}(s)){:}\mathrm{derive}(a, \mathsf{rest_m}(s), \mathsf{goto}^{\mathrm{list}_f(M)\langle\rangle}) && \text{(by I.H.)}\\
&= \delta_a(\mathsf{fst_m}(s)){:}\delta_a^{\mathrm{list}_m(M)\langle\rangle}(\mathsf{rest_m}(s)) && \text{(by Lemma 9)}\\
&= \mathsf{goto}_a^{\mathrm{list}_m(M)}(s) && \text{(by (6.2))}
\end{aligned}$$

In the latter case, we immediately obtain the result using (5.10) followed by the induction hypothesis:

$$\Delta_{k+1}(a, s) = \Delta_k(a, s) = \mathsf{goto}_a^{\mathrm{list}_m(M)}(s).$$

□

The tail recursive call in derive function can be replaced with a while loop. Moreover, Q and F can be pushed into the innermost loop as before. We result in the following simple algorithm (Figure 6).

```
function CONSTRUCT
    $\Delta \leftarrow \emptyset$; $D \leftarrow \{\mathsf{Nil_m}\}$; $Q \leftarrow \{\mathsf{Nil_m}\}$; $F \leftarrow \{\mathsf{Nil_m} \mid \mathsf{i}^M \in \mathcal{F}^M\}$;
    function MOVE$(a, s, \Delta)$
        return $\delta_a(\mathsf{i}^M)$:$\mathsf{Nil_m}$ if $s = \mathsf{Nil_m}$;
        $s' \leftarrow s$;
        $s' \leftarrow \mathsf{rest_m}(s')$ while $\Delta(a, s') = \bot \wedge s' \neq \mathsf{Nil_m}$;
        return $\delta_a(\mathsf{fst_m}(s))$:$(\Delta(a, s')$ if $\Delta(a, s') \neq \bot$ else $\mathsf{Nil_m})$;
    while $D \neq \emptyset$ do
        $D' \leftarrow \emptyset$;
        for $(a, s) \in \Sigma \times D$ do
            $t \leftarrow \mathrm{move}(a, s, \Delta)$;
            continue if $t \in \bot$;
            $\Delta \leftarrow \Delta \cup \{(a, s) \mapsto t\}$;
            if $t \notin Q$ then
                $D' \leftarrow D' \cup \{t\}$;
                $Q \leftarrow Q \cup \{t\}$;
                $F \leftarrow F \cup \{t\}$ if $\mathsf{fst_m}(t) \in \mathcal{F}^M \vee \mathsf{rest_m}(t) \in F$;
        $D \leftarrow D'$;
    return $(Q, F, \Delta)$;
```

Figure 6: An alternative updating Q and F in the innermost loop

7 Concluding Remarks

We have investigated a variant of subset construction in which the DFA-states are represented as lists of NFA-states rather than sets. Apply this to an NFA obtained from a DFA by allowing extra moves from the initial state into itself results, thanks to Theorem 1, in a practically more efficient algorithm than the general subset construction. We also have adapted this to DFA with default (or failure) transitions, obtaining an algorithm similar to KMP but allowing patterns given by arbitrary DFA's. Our algorithm is, as a result, very similar to Mohri's algorithm [6] generalizing KMP.

References

[1] Ricardo Baeza-Yates and Gaston H. Gonnet. A new approach to text searching. *Commun. ACM*, 35(10):74–82, October 1992.

[2] Janusz A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. In *Mathematical theory of Automata*, Volume 12 of MRI Symposia Series, pages 529–561. Polytechnic Press, Polytechnic Institute of Brooklyn, N.Y., 1962.

[3] M. Crochemore and W. Rytter. *Jewels of Stringology.* World Scientific, 2002.

[4] Donald E. Knuth. On the translation of languages from left to rigth. *Information and Control*, 8(6):607–639, 1965.

[5] Donald E. Knuth, James H. Morris, Jr., and Vaughan R. Pratt. Fast pattern matching in strings. 6(2):323–350, June 1977.

[6] Mehryar Mohri. String-matching with automata. *Nordic Journal of Computing*, 4.

[7] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3(2):114–125, April 1959.

[8] Shmuel Safra. On the complexity of omega-automata. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 319–327, 1988.

[9] Ken Thompson. Programming techniques: Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, June 1968.