SCIENCE CHINA
Information Sciences

• HIGHLIGHT •

February 2018, Vol. 61 028101:1–028101:3
doi: 10.1007/s11432-017-9155-x

The verification of conversion algorithms between finite automata

Dongchen JIANG[1*] & Wei LI[2]

[1]*School of Information Science & Technology, Beijing Forestry University, Beijing* 100083*, China;*
[2]*State Key Laboratory of Software Development Environment, Beihang University, Beijing* 100191*, China*

Received 29 May 2017/Accepted 29 June 2017/Published online 17 November 2017

Citation Jiang D C, Li W. The verification of conversion algorithms between finite automata. Sci China Inf Sci, 2018, 61(2): 028101, doi: 10.1007/s11432-017-9155-x

The conversion algorithms between finite automata are frequently used in language recognition. In recent years, there is a tendency to formalize and verify the correctness of conversion algorithms in interactive theorem provers. In 1997, Filliâtre [1] first formalized finite automata in Coq, and a conversion algorithm from regular expression to ε-NFA was verified and all sets were formalized with lists for program extraction. In 2010, Braibaut and Pous [2] contributed the complete verified procedures of converting regular expressions into automata with the help of matrices. Their implementation is efficient, but the matrix-based proof calls for efforts. In 2012, Lammich and Tuerk [3] adopted another approach to verify the Hopcroft's algorithm by using a refinement framework which replaces operations on abstract datatypes by corresponding operations on concrete datatypes in Isabelle/HOL. In 2015, Paulson [4] introduced the new datatype of the hereditarily finite set, and used it to simplify the constructive proofs of the correctness of conversion algorithms.

The purpose of this article is to propose three completely verified conversion algorithms between automata for code generation, i.e., the algorithm (Ne2N) to remove all the ε-transitions of ε-NFA; the powerset construction algorithm (N2D) to convert NFA into DFA; the Hopcroft's algorithm (Mini) to minimize DFA. The specifications should be concise and easy to implement, and the correctness proofs should be direct and avoid complex constructions. Different from the existing approaches, the formalization and verification of these algorithms are directly conducted on lists. Its main advantage is to make full use of the properties of specifications to shorten the length of verification. Moreover, the list-based specifications are more convenient for code generation by Isabelle Collection Framework [5].

Preliminaries. In this article, a record type Record (α, s)fa $= (Q :: s$ list, $\Sigma :: \alpha$ list, $\delta :: (s, \alpha, s)$ list, $q_0 :: s$, $F :: s$ list) is constructed as datatype of finite automata. Different from the definition in the textbooks, in this article every set of a finite automaton carries a datatype of list: Q is a state list, Σ is an alphabet (list), q_0 is the initial state, and F is the list of accept states. The transition function is also formalized by a list δ of transition triples: if (q, a, q') is an element of δ, then there is a transition from q to q' labeled by a. In Isabelle/HOL, list has the following functional applications: size (to calculate the length of a list), set (to convert a list into a set of its elements), [] (the empty list), # (to add an element in front of an existing list), @ (to append a list to another), hd (to acquire the first element) and $xs!i$ (the ith element of list xs).

Then the concept of NFA can be formalized by the property distinct Q $\wedge$ distinct $\Sigma \wedge$ distinct $\delta \wedge$ distinct $F \wedge$ $q_0 \in$ set Q $\wedge$ set $F \subseteq$ set $Q \wedge$

*Corresponding author (email: jiangdongchen@bjfu.edu.cn)
The authors declare that they have no conflict of interest.

Algorithm 1: *N2D*

```
   Input: NFA nfa = $(Q, \Sigma, \delta, q_0, F)$
   Output: DFA dfa = $(\mathcal{Q}, \Sigma, \Delta, S_0, \mathcal{F})$
1  $\Delta, \mathcal{F} \leftarrow []$;
2  $S_0 \leftarrow \{q_0\}$;
3  $\mathcal{Q}, W \leftarrow [S_0]$;
4  while $W \neq []$ do
5      pick $S$ from $W$;
6      if $S \cap$ set $F \neq$ set then
7          add $S$ to $\mathcal{F}$;
8      forall the $a \in$ set $\Sigma$ do
9          $S' \leftarrow \{q' \mid (q, a, q') \in$ set $\delta, q \in S\}$;
10         if $S' \notin$ set $\mathcal{Q}$ then
11             add $S'$ to $W$;
12             add $S'$ to $\mathcal{Q}$;
13         add $(S, a, S')$ to $\Delta$;
```

Algorithm 2: *Ne2N*

```
   Input: NFA-$\varepsilon$ nfae = $(Q, \Sigma_\varepsilon, \delta, q_0, F)$
   Output: NFA nfa = $(Q', \Sigma, \delta', q_0, F')$
1  $\delta', F' \leftarrow []$;
2  $Q', W \leftarrow [q_0]$;
3  while $W \neq []$ do
4      pick $q$ from $W$;
5      if set (FiniteEpsReach $\delta$ $q$) $\cap$ set $F \neq$ set then
6          add $q$ to $F'$;
7      forall the $a \in$ set $\Sigma$ do
8          forall the $q' \in$ set(EpsCharReach nfae $q$ $a$) do
9              if $q' \notin$ set $Q'$ then
10                 add $q'$ to $W$;
11                 add $q'$ to $Q'$;
12             add $(q, a, q')$ to $\delta'$;
```

Algorithm 3: Mini_State

```
   Input: DFA dfa = $(Q, \Sigma, \delta, q_0, F)$
   Output: A partition $P$ of $Q$
1  if set $F$ = set or set $Q$ − set $F$ = set then
2      $P \leftarrow$ [set $Q$];
3  else
4      $P \leftarrow$ [set $F$, set $Q$ − set $F$];
5      while $P$ is unstable do
6          find $B, B' \in$ set $P$, $a \in$ set $\Sigma$ s.t. $(a, B')$ splits $B$;
7          replace $B$ with $\{q \in B \mid$ (delta_fun $\delta$) $q$ $a \in B'\}$
8                  and $\{q \in B \mid$ (delta_fun $\delta$) $q$ $a \notin B'\}$;
```

Figure 1 Conversion algorithms between finite automata.

$(\forall (q, a, q') \in \text{set } \delta.\ q \in \text{set } Q \land a \in \text{set } \Sigma \land q' \in \text{set } Q)$. Besides this property, to formalize DFA, it also needs to show every state has exactly one transition triple for each character, i.e., $\forall q \in \text{set } Q.\ \forall a \in \text{set } \Sigma.\ \exists! q' \in \text{set } Q.\ (q, a, q') \in \text{set } \delta$.

Formalization of conversion algorithms. All three conversion algorithms come from the text book Automata Theory: An Algorithmic Approach [6]. We select these algorithms because they are important in practice, uncomplicated to construct, and easy to implement. The algorithms N2D and Ne2N, and the core part (state collection) of Mini are presented in Figure 1.

Both N2D and Ne2N are worklist-based algorithms. For each algorithm, a worklist W is used for storing all the unprocessed elements and another list ($\mathcal{Q}$ or Q') for storing intermediate results. The algorithm begins with an initial state (S_0 or q_0), picks an element from W and calculates all the one-step reachable states from the element in each iteration step. If a one-step reachable state is not in the intermediate result, it will be added to the intermediate result and W. The iteration continues until W becomes empty. To formalize this kind of worklist-based algorithms, a general procedure collect_option is formalized by while_option and applied to a state list slist and a state-expansion function expands to collect all the reachable states from slist according to expansion. With the help of this procedure, what is left is to construct the specific expansion functions for N2D and Ne2N.

As to the formalization of N2D, the key lies in the formalization of the operation (in line 9 of Algorithm 1) that calculates the a-reachable state set S' from the given S. For this, CharReach$_D$, the actual transition function of the output automaton, is recursively constructed to collect all the reachable states of the NFA from S with a. After this formalization, it can be proved that if the input nfa is an NFA then $N2D$ nfa is a DFA.

For Ne2N, the key of formalization lies in lines 5 and 8 of Algorithm 2. In the formalization, FiniteEpsReach δ q is used to collect all the ε-reachable states from q according to δ. The function can also be formalized by collect_option where the expansion function is constructed by all the ε-transitions of the input δ. Based on FiniteEpsReach, EpsCharReach can be conducted to collect all the ε-a-reachable states of an ε-NFA nfae from q with a. Thus, lines 5 and 8 of Ne2N can be formalized, and it can be proved that if the input of Ne2N is a ε-NFA the output will be a normal NFA.

For each DFA, its unique minimized DFA can be constructed by the Hopcroft's algorithm Mini, whose core part is Mini_State, which calculates all the states of the minimized DFA, as presented in Algorithm 3 of Figure 1. Unlike $D2N$ and Ne2N which both use collect_option for states collection, Mini_State obtains the state list P by partitioning Q into different blocks: if neither set Q nor set Q-set F is $\emptyset$, the algorithm initials P with [set F, set Q-set F] and conducts partitions repeatedly according to the stability of P. Here, P is unstable if there are blocks B, B' in P, a in Σ and q_1, q_2 in B such that q_1 is transited to a state in B' with a and q_2 is transited to a state out of B' with the same a. If P is unstable, B will be split into two sub-blocks of B by (B', a). In Algorithm 3, delta_fun is the function to convert a transition list δ into the corresponding transition function.

According to unstable, the states in the same block of P will have the same transition behaviors when Mini_State terminates, i.e., for any $a \in \Sigma$ and $B \in$ set P, the states of B must be transited to the states of some block of P with a-transitions. Thus, the according transition triple list of the minimized DFA can be constructed. Then the

Hopcroft's algorithm can be obtained with ease.

Correctness verification. To prove a conversion algorithm is correct, we need to show that (1) the output of the algorithm is a target automaton and (2) the input and the output automata recognize the same language. As (1) can be proved directly by following the definition of DFA or NFA, we then only need to illustrate the verification of (2) for each algorithm. In this article, a language refers to a set of words, and a word is a list of characters. An automaton recognizes a language L if and only if it accepts every word of L. Here, a finite automaton fa accepting a word w can be formalized as: accept fa $w \equiv (\exists \text{slist. size slist} = \text{size } w + 1 \wedge \text{slist!}0 = q_0 \wedge (\forall i < \text{size } w.\ (\text{slist!}i, w!i, \text{slist!Suc } i) \in \text{set } \delta) \wedge \text{slist!(size } w) \in \text{set } F)$. The most direct way to prove that an automaton accepts a word is to construct a state list and prove that the list satisfies the definition of accept.

If the automaton is a DFA, the state list can be obtained by running the DFA from the initial state with the given word and collecting all the passed states. In this article, a recursive function run is constructed to simulate the running of a DFA starting from a state q with a given word:

run [] dfa $q = [q]$,
run $(x\#xs)$ dfa $q = q\#$(run xs dfa (delta_fun δ q x)).

By runing the corresponding DFA from the initial state with the given word, we can prove that DFA dfa $\Longrightarrow$ accept dfa w $\longleftrightarrow$ (Mini dfa) w and NFA nfa $\Longrightarrow$ accept nfa w $\longrightarrow$ accept (N2D nfa) w.

To prove an NFA accepts a word needs to find one state list satisfying the definition of accept. To prove accept (N2D nfa) w $\longrightarrow$ accept nfa w, first we need to obtain a state list slist$_d$ of (N2D nfa) that satisfies the properties of accept. Then a state list of nfa can be constructed by picking one state of each element of slist$_d$ and combining them into a state list. Then it can be proved that the constructed list satisfies the properties of accept.

To show the input and the output of Ne2N recognize the same language, it is inaccurate to formalize the word acceptance of ε-NFA by accept because the actual alphabet of ε-NFA also include ε. Thus, it needs to redefine the word acceptance for ε-NFA. Here, we propose the concept of eps_word: an eps_word begins with finite εs and ends with a normal character, i.e., eps_word $w \equiv (\forall i < \text{size } w - 1.\ w!i = \text{eps}) \wedge w!(\text{size } w - 1) \neq \varepsilon \wedge w \neq [])$. Each eps_word corresponds to one unique character (its last element), thus the word acceptance accept$_\varepsilon$ of ε-NFA can be redefined by replacing the character-transitions in accept with the corresponding eps_word-transitions. Then it can be proved that NFA nfae $\Longrightarrow$ accept$_\varepsilon$ nfae w $\longleftrightarrow$ accept (Ne2N nfae) w.

Conclusion. In this article, we have formalized three conversion algorithms between finite automata with lists and verified their functional correctness in Isabelle/HOL. This work is part of the Computer Aided Verification of Automata project whose aim is to formalize and verify important algorithmic parts of automata theory and model checking. Different from the existing researches using matrices or sets to formalize automata, we adopt the datatype of list for algorithm formalization. The direct list-based specifications are more alike to those we implement in programs, and can be directly used for code generation. The list-based constructive proofs, which are similar to the textbook proofs, can save additional conversion constructions and shorten the length of proofs. The whole work only includes 226 lines of specification formalization and around 2250 lines of constructive proofs. All the specifications and proofs can be downloaded at http://pan.baidu.com/s/1nu8FGbn.

Acknowledgements This work was supported by Beijing Natural Science Foundation (Grant No. 4164092) and Fundamental Research Funds for Central Universities (Grant No. BLX2015-17).

Supporting information Appendixes A and B. The supporting information is available online at info.scichina.com and link.springer.com. The supporting materials are published as submitted, without typesetting or editing. The responsibility for scientific accuracy and content remains entirely with the authors.

References

1 Filliâtre J C. Finite Auotmata Theory in Coq: a Constructive Proof of Kleene's Theorem. Research Report 97 C 04, LIP-ENS Lyon. 1997
2 Braibant T, Pous D. An efficient Coq tactic for deciding Kleene algebras. In: Proceedings of International Conference on Interactive Theorem Proving, Edinburgh, 2010. 163–178
3 Lammich P, Tuerk T. Applying data refinement for monadic programs to Hopcroft's algorithm. In: Proceedings of International Conference on Interactive Theorem Proving, New Jersey, 2012. 166–182
4 Paulson L C. A Formalisation of finite automata using hereditarily finite sets. In: Proceedings of CADE-25-International Conference on Automated Deduction, Berlin, 2015. 231–245
5 Lammich P, Lochbihler A. The isabelle collections framework. In: Proceedings of International Conference on Interactive Theorem Proving, New Jersey, 2010. 339–354
6 Esparza J. Automata Theory: an Algorithmic Approach. 2016. https://www7.in.tum.de/~esparza/autoskript.pdf