Certified Parsing of Regular Expressions in Agda

Wai Tak, Cheung
Student ID: 1465388
Supervisor: Dr. Martín Escardó

Submitted in conformity with the requirements
for the degree of BSc. Computer Science
School of Computer Science
University of Birmingham

Abstract

Certified Parsing of Regular Expressions in Agda

Wai Tak, Cheung

Blah blah blah. Blah blah blah.

Keywords: language, regular expression, finite automata, agda, thompson's construction, powerset construction, proofs

Acknowledgments

Blah blah blah. Blah blah blah.

All software for this project can be found at
https://codex.cs.bham.ac.uk/svn/projects/2015/wtc488/

Contents

1 List of Abbreviations

ϵ-NFA	Non-deterministic finite automata with ϵ-transition
NFA	Non-deterministic finite automata
DFA	Deterministic finite automata

2 Introduction

This work aims at implementing the parts in Formal Language Theory and Automata Theory that are related to regular languages and finite automata using Agda. The project is separated into three parts: 1) translating regular expressions to NFA and DFA, 2) proving the correctness of the translation and 3) formalising the Myhill-Nerode Theorem.

Here will be a paragraph emphasising the importance of parsing/automata.

The next section will be an introduction on proof assistance and a review on related reseach topic. After that, the thrid section will be a detail description of our approach on the project followed by the evaulation. Finally, we will draw the conclusions.

3 Literature Review

3.1 Curry-Howard Isomorphism

Relationship between programs and proofs.

3.2 Agda

Brief introduction on how agda works as a proof assistant, and how to write proofs in agda.

3.3 Related Work

The matrix representation.

4 Agda Formalisation

Let us recall the three objectives of the project: 1) translating regular expressions to NFA and DFA, 2) proving the correctness of the translation and 3) formalising the Myhill-Nerode Theorem. In part 1), we followed Thompson's construction algorithm to build an ϵ-NFA from a regular expression. Then we removed all its ϵ-transitions by computing the ϵ-closure for each state. Finally, we used powerset construction to determinise the automata. (Minimize?)

In this section, we will walk through the agda formalisation of each of these steps together with their correctness proofs. However, before we can go into these steps, we will have to define a representation of subset it plays a major role in the Formal Language theory.

4.1 Subsets and Decidable Subsets

Agda Please refers to Subset.agda and Subset/DecidableSubset.agda

Definition 1.1 Suppose A is a set, in Agda, we represents its subset as a unary function on A, i.e. $Subset\ A = A \to Set$.

When declaring a subset, we can write $SubA = \lambda\ a \to\ ?$ in Agda, the ? here corresponds to the condition for an element of A to be included in $SubA$. This construction is very similar to set comprehension. For example, the subset $\{a \mid a \in A,\ P(a)\}$ corresponds to $\lambda\ a \to P\ a$ in Agda. As we mentioned before, a function in the form of $A \to Set$ is also a predicate of A. Therefore, Subset is also a unary predicate of A. Thus, the decidibilty of Subset will remain unknown until it is proved.

Definition 1.2 The other representation of subset is $DecSubset\ A = A \to Bool$. Unlike $Subset$, its decidability is ensured by its definition.

The two definitions has different purposes. $Subset$ was used to represent $Language$ as not every language is decidable. For other parts in the project such as the subset of states of an automata, $DecSubset$ was used as the decidability is assumed in the definition. The two definition are defined in Subset.agda and Subset/DecidableSubset.agda respectively as stated before, operations such as membership ($\in$), subset ($\subseteq$), superset ($\supseteq$) and eqaulity ($=$) can also be found in the two files.

Now, by using the subset representation, we can define languages, regular expressions and finite autotmata. Their formalisation in this project followed tightly to the definition from Aho, A. and Ullman, J. (1972).

4.2 Languages

Agda Please refers to Language.agda

Suppose we have a set of alphabet Σ. In Agda, it will be a data type, i.e. $\Sigma : Set$.

Definition 2.1 We first define Σ^* as the set of all strings over Σ. In our approach, it was defined as a list of Σ, i.e. $\Sigma^* = List\ \Sigma$. For example, $(A :: g :: d :: a :: [])$ represents the string 'Agda' and an empty string will be represented as the empty list []. In this way, when running the automata, we can pattern match on the input string.

Definition 2.2 A language is a subset of Σ^*; in Agda, $Language = Subset\ \Sigma^*$. $Subset$ instead of $DecSubset$ was used because not every language is decidable.

4.2.1 Operations on Languages

Definition 2.3 If L_1 and L_2 are languages, then the union of the two languages $L_1 \cup L_2$ is defined as $\{w \mid w \in L_1\ or\ w \in L_2\}$. In Agda, we defined it as $L_1 \cup L_2 = \lambda\ w \rightarrow w \in L_1\ \uplus\ w \in L_2$ where $\uplus$ is the Sum type representing the proposition OR.

Definition 2.4 If L_1 and L_2 are languages, then the concatenation of the two languages $L_1 \bullet L_2$ is defined as $\{w \mid \exists u \in L_1.\ \exists v \in L_2.\ w = uv\}$. In Agda, we defined it as $L_1 \bullet L_2 = \lambda\ w \rightarrow \exists[u \in \Sigma^*]\ \exists[v \in \Sigma^*](u \in L_1 \times v \in v \in L_2 \times w \equiv u\ ++\ v)$ where $\times$ is the Product type representing the proposition AND and $\equiv$ represents the equivalency of two data.

Definition 1.5 If L is a language, then the closure of L $L*$ is defined as $\bigcup_{n \in N} L^n$ where $L^n = L \bullet P^{n-1}$ and $L^0 = \{\epsilon\}$. Agda formalisation?

4.3 Regular Languages and Regular Expressions

Agda Please refers to RegularExpression.agda

Definition 3.1 We define regular languages over Σ inductively as follows:

1. $\emptyset$ is a regular language
2. $\{\epsilon\}$ is a regular language
3. $\forall a \in \Sigma.\ \{a\}$ is a regular language
4. if L_1 and L_2 are regular languages, then
   (a) $L_1 \cup L_2$ is a regular language
   (b) $L_1 \bullet L_2$ is a regular language
   (c) $L_1 *$ is a regular language
5. nothing else is a regular language

The equivalent Agda definition:

```
data Regular : Language → Set₁ where
  nullL : Regular Ø
  empty : Regular ⟦ϵ⟧
  singleton : (a : Σ) → Regular ⟦a⟧
  union : ∀ L₁ L₂ → Regular L₁ → Regular L₂ → Regular (L₁ ∪ L₂)
  conca : ∀ L₁ L₂ → Regular L₁ → Regular L₂ → Regular (L₁ • L₂)
  kleen : ∀ L → Regular L → Regular (L ⋆)
```

Definition 3.2 Here we define regular expressions over Σ as follows:

1. $\emptyset$ is a regular expression denoting the regular language $\emptyset$
2. ϵ is a regular expression denoting the regular language $\{\epsilon\}$
3. $\forall a \in \Sigma$. a is a regular expression denoting the regular language $\{a\}$
4. if e_1 and e_2 are regular expression denoting the regular languages L_1 and L_2 respectively, then
   (a) $e_1 \mid e_2$ is a regular expressions denoting the regular languag $L_1 \cup L_2$
   (b) $e_1 \cdot e_2$ is a regular expression denoting the regular language $L_1 \bullet L_2$
   (c) $e_1\ ^*$ is a regular expression denoting the regular language $L_1 *$
5. nothing else is a regular expression

4.4 Non-deterministic Finite Automata with ϵ-transitions

Agda Please refers to module **ϵ-NFA** in Automata.agda

By now, every string we have considered are in the form of *List* Σ^*. However, this definition gives us no way to pattern match an ϵ-step in the automata. Therefore, we need to introduce another set of alphabet that includes ϵ. (For Definition 4.1 and 4.2, please refers to Language.agda)

Definition 4.1 We define Σ^e as the union of Σ and $\{\epsilon\}$, i.e. $\Sigma^e = \Sigma \cup \{\epsilon\}$. In Agda, this is simply a datatype definition:

```
data Σᵉ : Set where
  α : Σ → Σᵉ
  E : Σᵉ
```

Definition 4.2 Now we define Σ^{e*}, the set of all strings over Σ^e in a way similar to Σ^*, i.e. $\Sigma^{e*} =$ *List* Σ^e. For example, the string 'Agda' can be represented as $(\alpha\ A :: \alpha\ g :: E :: \alpha\ d :: E :: \alpha\ a :: [])$.

Definition 4.3 A ϵ-NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where

4.5 Thompson's Construction

Theorem 1.1 The language accepted by a regular expression is equal to the language accepted by the translated ϵ-NFA.

4.6 Non-deterministic Finite Automata without ϵ-transitions

4.7 Removing ϵ-transitions

4.8 Deterministic Finite Automata

4.9 Powerset Construction

4.10 Myhill-Nerode Theorem

5 Evaluation

(?)

6 Conclusion

(?)

7 References

Aho, A. and Ullman, J. (1972). The Theory of Parsing, Translation and Compiling. Volume I: Parsing. United States of America: Prentice-Hall, Inc.

8 Appendix

Agda Code?