Lesson 1 - outline

Automata on Infinite Words
D. Fisman

Introduction

1. What are automata on infinite words? What do they do?
2. Why are automata on infinite words interesting?
3. What are reactive systems?
4. How does one reason about reactive systems?
5. Temporal logic and verification (model checking) in a nutshell

Languages and ω-Languages

An alphabet Σ is a finite set of symbols. The set of finite words over Σ is denoted by Σ*. The set of infinite words over Σ, termed ω-words, is denoted by Σω. The set of finite and infinite words over Σ is denoted Σω. We use ε for the empty word and Σ+ for Σ* \ {ε}. A language is a set of finite words, that is a subset of Σ*, while an ω-language is a set of ω-words, that is a subset of Σω.

Concatenation, Kleene and ω-closures

Given a word u and a word or ω-word v, their concatenation is denoted u · v or simply uv. Given a language U and a language or ω-language V, their concatenation is denoted U · V or simply UV and it is the set {uv | u ∈ U and v ∈ V}. For k > 1 we use Uk to denote U · Uk−1, where U1 and Uk denote U and {ε}, respectively. We use U* for ∪k≥0Uk and Uω for ∪k≥1Uk. For a set U we use Uω for the set {w ∈ Σω | ∃u1, u2, u3, u4, ..., ∈ U \ {ε} s.t. w = u1u2u3u4 ...}.

Regular and ω-regular expressions

Regular expressions over a given alphabet Σ are defined inductively as follows: r ::= ∅ | σ | r1 + r2 | r1 · r2 | r* where σ ∈ Σ. Their semantics (the set of words that they recognize) is defined as follows:

- [∅] = ∅
- [σ] = {σ}
- [r1 + r2] = [r1] ∪ [r2]
- [r1 · r2] = [r1] · [r2]
- [r*] = [r]+

ω-regular expressions over a given alphabet Σ are defined inductively as follows: s ::= ∅ | rω | r · s | s1 + s2 where r is a regular expression. Their semantics (the set of words that they recognize) is defined as follows:

- [∅] = ∅
- [rω] = `{r}ω
- [r · s] = [r] · [s]
- [s1 + s2] = [s1] ∪ [s2]

Automata and Acceptors

An automaton is a tuple A = (Σ, Q, Q0, δ) consisting of an alphabet Σ, a finite set Q of states, an initial set of states Q0 ⊆ Q, and a transition function δ : Q × Σ → 2Q. A run of an automaton on a finite word v = a1a2...an is a sequence of states q0, q1, ..., qn such that q0 ∈ Q0, and for each i ≥ 0, qi+1 ∈ δ(qi, ai+1). A run on an infinite word is defined similarly and results in an infinite sequence of states. The transition function is naturally extended to a function from Q × Σω to 2Q, by defining δ(q, ε) = q and δ(q, av) = δ(δ(q, a), v) for q ∈ Q, a ∈ Σ and v ∈ Σω, and δ(Q′, v) = ∪q′∈Q′δ(q′, v). We say that A is deterministic if |Q0| = 1 and |δ(q, a)| ≤ 1.

By augmenting an automaton with an acceptance condition α, obtaining a tuple (Σ, Q, Q0, δ, α), we get an acceptor, a machine that accepts some words and rejects others. An acceptor accepts a word if at least one of the runs on that word is accepting. For finite words the acceptance condition is a set F ⊆ Q and a run on a word v is accepting if it ends in an accepting state, i.e., if δ(Q0, v) contains an element of F. For infinite words, there are
various acceptance conditions in the literature; all are defined with respect to the set of states visited infinitely often during a run. For a run \( r = q_0q_1q_2 \ldots \) we define \( \text{Inf}(r) = \{ q \in Q \mid \forall i \in \mathbb{N}. \exists j > i. q_j = q \} \). A Büchi acceptance conditions is a set \( F \subseteq Q \). A run of a Büchi automaton is accepting if it visits \( F \) infinitely often. That is, if \( \text{Inf}(r) \cap F \neq \emptyset \). We use \( [A] \) to denote the set of words accepted by a given acceptor \( A \). Two acceptors \( A \) and \( B \) are equivalent if \( [A] = [B] \).

**Acronyms** We use three letter acronyms to describe acceptors, where the first letter is in \{D,N\} and denotes if the automaton is deterministic or nondeterministic. The second letter describes the acceptance condition, for instance, we use F for automata on finite words and B for Büchi automata. The third letter describes the object processed by the automaton, e.g. W for word (or \( \omega \)-words) and T for trees. For instance, DFW and NFW stands for deterministic and non-deterministic automata on finite words, and DBW and NBW stands for deterministic and non-deterministic Büchi automata on \( \omega \)-words. We use blackboard font for the class of languages accepted by a certain type of automaton, for instance \( \text{DBW} \) is the set of languages that can be recognized by a deterministic Büchi automaton.

**Some Examples**

Examples for \( \omega \)-regular languages:

\[
\begin{align*}
L_1 &= \{ w \in \{ a, b \}^\omega \mid \text{there are at least 3 } a \text{'s in } w \} \\
L_2 &= \{ w \in \{ a, b \}^\omega \mid \text{there are at most 2 } a \text{'s in } w \} \\
L_3 &= \{ w \in \{ a, b \}^\omega \mid \text{there is an even number of } a \text{'s in } w \} \\
L_4 &= \{ w \in \{ a, b \}^\omega \mid \text{the number of } a \text{'s in } w \text{ is either even or infinite} \} \\
L_5 &= \{ w \in \{ a, b, c \}^\omega \mid \begin{cases} 
\text{there are infinitely many } a \text{'s} \\
\text{every } a \text{ is followed by a } b \\
\text{there are finitely many } b \text{'s} 
\end{cases} \} \\
L_\infty &= \{ w \in \{ a, b \}^\omega \mid \text{the number of } a \text{'s in } w \text{ is infinite} \}
\end{align*}
\]

We went over these languages, and gave examples of words in the language and words not in the language. We gave an \( \omega \)-regular expression characterization for some of these languages. We described a Büchi automaton (NBW) to some of these languages.