MAU22C00 Lecture 29

John Stalker

Trinity College Dublin

From finite state automata to regular expressions

Any language which can be recognised by a finite state automaton can be described by a regular expression.

There is a proof of this in the notes. The idea is to define generalised finite state automata in such a way that:

- Every finite state automaton can be made into a generalised finite state automaton with a single initial state and a single accepting state, which recognises the same language.
- Given a generalised finite state automaton with a single initial state, a single accepting state, and some positive number of intermediate states we can find a generalised finite state automaton with one intermediate state fewer, which recognises the same language.
- A generalised finite state automaton with a single initial state, and no intermediate states is described by a single regular expression.

The proof is constructive in principle, but you wouldn't want to construct regular expressions this way in practice.

From regular expressions to grammars

It's easy, at least in theory, to convert a regular expression to a grammar.

If the regular expression is a single character we can clearly do this. We can make the grammar left or right regular.

Every regular expression is built from such regular expressions by union, concatenation and Kleene star.

We know how to implement all of those operations for left or right regular grammars, so we can handle any regular expression.

As with the previous construction, this is constructive, but usually inefficient.

Equivalent descriptions of a language

We the following constructions for going from one description of a language to another. Blue arrows indicate constructions which reverse the language.

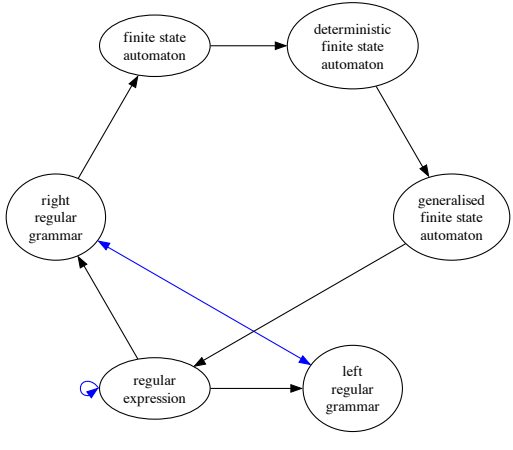

Figure 1: Descriptions of a language

Equivalent descriptions of a language

From any node on the graph to any other node there is a walk with an even number of blue edges, so given any of these descriptions of a language we can find any other description.

For example, to convert a left regular grammar to a right regular grammar for the same language we could convert it to a right regular grammar for the reverse language, convert that to a finite state automaton, then to a deterministic finite state automaton, a generalised finite state automaton then a regular expression, all for the reversed language, then reverse the regular expression and convert that to a regular grammar.

From any node on the graph to any other node there is a walk with an odd number of blue edges, so given any of these descriptions of a language we can find any other description for the reversed language.

For example, given a finite state automaton we can construct one for the reverse language.

The set of languages on a given set of tokens describable in any of these ways is the same. These are called the regular languages.

Closure properties of regular languages

Any closure property we've proved for any one of these descriptions now applies to all of them. That includes

- Union
- Intersection
- Relative complement
- Concatenation
- Kleene star
- Reversal

Are there languages which aren't regular?

An earlier counting argument showed there are languages which can't be generated by a grammar. Every regular language can be generated by a grammar so there are languages which are not regular.

Can we show that some *particular* language, e.g. the language of balanced parentheses, is not regular?

At the moment we have the following options:

- Show that there is no left regular grammar which generates it.
- Show that there is no right regular grammar which generates it.
- Show that there is no finite state automaton which recognises it.
- Show that there is no deterministic finite state automaton which recognises it.
- Show that there is no generalised finite state automaton which recognises it.
- Show that there is no regular expression which describes it.

There are infinitely many grammars, automata, or regular expressions, so we can hardly check all of them.

Showing languages aren't regular

The only way to prove a language isn't regular is to prove that all regular languages have a certain property which the given language doesn't have.

There are two main options:

- The Pumping Lemma:
  - relatively easy to state
  - relatively easy to prove
  - tricky to apply
  - only works for some non-regular languages
  - has a nice generalisation to context free languages
- The Myhill-Nerode Theorem
  - harder to state and prove
  - easier to apply–at least I find it easier
  - fails in a useful way in the regular case
  - works for all non-regular languages
  - does not have a nice generalisation to context free languages

The notes contain proofs of both. In lecture I'll concentrate on applying them.

Pumping Lemma (statement)

An integer p is called a pumping length for a language L if every list (string) of length at least p in L contains a non-empty segment (substring) within its first/last p tokens (characters) which can be replaced by arbitrarily many repetitions of itself, without leaving L.

Example: the language of integers has a pumping length of three. If there are at least three characters we can repeat the second one as often as we like.

Why not two?

Is the string `0` a problem?

A language is said to have the pumping property if it has a pumping length.

The Pumping Lemma is the statement that every regular language has the pumping property.

Warning: Not every language with the pumping property is regular!

Pumping Lemma (strategy)

The Pumping Lemma is only every used in proofs by contradiction. The steps are always

- Assume the language is regular, so there is some pumping length p for it.
- Choose a list (string), depending on p.
- Show that for the chosen string and *every* non-empty segment in the first p tokens (characters) there is some number of repetitions which takes us outside the language.
- Conclude that the language wasn't really regular, since it doesn't satisfy the pumping property.

Note that we we negate a statement all universal quantifiers become existential and vice versa, e.g. the negation of "there is a p such that for all lists of length at least p there is a non-empty segment ..." is "For all p there is a list of length at most p such for all non-empty segments ...".

Example (balanced parentheses)

For the language of balanced parentheses we choose a string with p (’s followed by p)’s.

This is a member of the language of balanced parentheses.

Any substring within the first p characters consists only of (’s.

If we repeat that substring 0 times we get fewer (’s than)’s. The last) has no (to match it, so this string is not in the language.

We’re done. The language of balanced parentheses is not regular.

Here it was fairly easy to guess what string to chose. Often it’s not.

The Myhill-Nerode theorem

For each list there is a (possibly empty) set of valid continuations, i.e. lists you could append to it to get a member of the language.

We'll say that two lists are equivalent of they have the same set of valid continuations. This is an equivalence relation.

Consider the set of equivalence classes. The Myhill-Nerode Theorem, or one of the three versions of it, says that this set of equivalence classes is finite if and only if the language is regular.

The idea of the proof is to show the following two facts:

- There is a strongly deterministic (finite?) state automaton which has one state for each equivalence class and which recognises the language.
- Any other strongly deterministic (finite?) state automaton which recognises the language must have at least as many states as that one.

The second fact is what allows us to show certain languages aren't regular. The first allows us to construct optimal finite state automata for languages which are regular.