Craig Silverstein
Gates 470
Phone: (72)3-1787
Office Hours: Wed 4:30–5:30, Fri 1–3, Gates 470
Section: Monday 2:15–3:15, 380-380W

Computer Science 154
School of Engineering
Stanford University

Section Handout #1

Infinity vs. Arbitrarily Large

There are an infinite number of integers. Despite this fact, the length of any given integer is finite: if you give me an integer (say $10^{10000000}$), I can calculate its length and then come up with a number that is longer. This is not true for the real numbers: I cannot come up with any number that is longer than the length of π, because π never ends.

On the other hand, there is no fixed limit on how long integers can be. If I come up with a length (say 3), you can come up with a number that is longer (say 100000). Hence integers can be made to be *arbitrarily large*. The difference between this case and the one above is what order information is divulged: in the first case, you give me an integer and I get to pick a length knowing your integer, while in the second case you pick your integer *after* I've picked my length. This distinction is key to understanding infinity arguments.

Strings

There are many parts to strings (a.k.a "words"); the trick is not to get them confused.

Alphabet (Σ): Just a set: $\{a, b\}$, $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 0\}$, $\{a, b, *, (,), \cup\}$. For cs154, the alphabet must be finite. σ is used as a variable to refer to elements of the alphabet.

String or **word (S):** A collection of symbols from te alphabet. This must also be finite, though it may be arbitrarily long (strings are like integers in this respect). That is, all strings must end, but for any string I give you, you can come up with one longer.

Language ($\mathcal{L}$): A set of strings. This is the only thing in string theory that can be infinite (though it can be finite too). A language divides all strings over a given alphabet into two categories: the strings in the language set, and the strings outside of it. A language can be described by listing all the strings in the set, but this is time-consuming if the set is infinite. Instead we often use one of two techniques:

- A function takes a string as input and outputs "yes" if the string is in the language and "no" if not. This function is usually represented graphically as a Finite State Automaton.
- A shorthand for representing strings; any string that fits the format of the shorthand is in the language. The shorthand used is called Regular Expression format.

The important thing to understand is that these two techniques cannot list all possible languages: there are ways to divide up strings that cannot be expressed by state automata or by regular expressions (for instance, trying making up a Regular Expression for the language of palindromes over the language $\{a, b\}$). Therefore, these techniques of describing language sets are not considered very "powerful."

Deterministic Finite Automata

I present a way of looking at DFAs which is different from the one presented in lecture. If you understand DFAs but don't understand this section, ignore the section.

One (admittedly naive) view of human reading comprehension is that, as we scan the letters of a word, we form hypotheses of what the word means. For instance, if the word is "carpool," we might, after reading the first letter, think "this is a 'c' word," and after the third letter, think "this is a word having to do with automobiles." The idea behind this theory that after every letter, we find ourselves in a **state** encoding our current understanding of the word. The state we're in after reading the entire word determines what we think the word means.

Though the human mind doesn't work this way, DFAs do. The heart of the DFA is thus the transition function δ, which determines how the state changes as we read more and more of the word. When we scan the next letter of our input word, we ask the transition function what state we should go to. The human mind might take into account the entire word up to that point, plus a passing knowledge of the dictionary. The DFA takes into account only the letter just read, plus a knowledge of what state we're in now. This is why DFAs are so non-powerful: they can't look at the whole string at once ("there's no going back") and their knowledge of the world consists not in a big, fat dictionary, but in which of a small number of states we find ourselves in at the moment.

This view of the world suggests a way for us to express a DFA working on a string: we put, after each letter, the state we end up in. Thus, for the string abbaba, some DFA will give us <0> a <1> b <3> b <3> a <2> b <1> a <1>. <0> is the start state, and <1> is the stae we end up in. I use numbers in angle brackets to represent states, but this isn't really so inspired, because it gives no indication that states are where we store our knowledge of the world. Consider the following mark-up instead, of the string 00101:

```
<even-0s> 0 <odd-0s> 0 <even-0s> 1 <even-0s> 0 <odd-0s> 1 <odd-0s>
```

The names of the two states tell you right away what information the DFA is keeping track of. A different DFA might keep track of something else:

```
<#0=#1> 0 <#0=#1+1> 0 <#0=#1+2> 1 <#0=#1+1> 0 <#0=#1+2> 1 <#0=#1+1>
```

How many states would I need, in general, to keep track of the difference between the number of 0s and the number of 1s in a string?

Regular Expressions

RE's consist of all the symbols in the alphabet plus the special symbols *, (,), and $\cup$. * is the important symbol; it is used to represent "an arbitrary number of" (not, of course, "an infinite number of," since strings must have finite length). So &* represents "the set of all strings consisting of an arbitrary number of ampersands (possibly none, by the way). If we had an alphabet $\{\&\}$, for instance, &* would represent the language consisting of all the strings in this alphabet. If our alphabet were $\{\&, @\}$ instead, &* would represent the language consisting of all the strings that use just the &. &*@* would represent the language consisting of all strings in which all the ampersands precede all the at-signs. The string &&@ is in this language, as it @, but &@&@ is not. Parentheses are used for grouping: (&@)* represents a language with strings such as &@, &@&@&@, and ϵ (the empty string); this is different from &*@*. The union symbol does what it would seem to: &* $\cup$ @* represents a language consisting of either an arbitrary number of &s or an arbitrary number of @s, such as && or @@@. &@, on the other hand, is outside of this language.