Machines and their languages (G51MAL)
Lecture notes
Spring 2003

Thorsten Altenkirch

January 29, 2004

1 Introduction

Most references refer to the course text [HMU01]. Please, note that the 2nd edition is quite different to the first one which appeared in 1979 and is a classical reference on the subject.
I have also been using [Sch99] for those notes, but note that this book is written in german.
The online version of this text contains some hyperlinks to webpages which contain additional information.

1.1 Examples on syntax

In PR1 and PR2 you are learning the language JAVA. Which of the following programs are *syntactically correct*, i.e. will be accepted by the JAVA compiler without error messages?

Hello-World.java

```
public class Hello-World {

  public static void main(String argc[3]) {
      System:out.println('Hello World');
  }
}
```

A.java

```
class A {
  class B {
      void C () {
          {} ; {{}}
      }
  }
}
```


I hope that you are able to spot all the errors in the first program. It may be actually suprising but the 2nd (strange looking) program is actually correct.
How do we know whether a program is syntactically correct? We would hope that this doesn't depend on the compiler we are using.

1.2 What is this course about?

1. Mathematical models of computation, such as:
   - Finite automata,
   - Pushdown automata,
   - Turing machines
2. How to specify formal languages?
   - Regular expressions
   - Context free grammars
   - Context sensitive grammars
3. The relation between 1. and 2.

1.3 Applications

- Regular expressions

  Regular expressions are a convenient ways to express patterns (i.e. for search). There are a number of tools which use regular expressions:

  - `grep` pattern matching program (UNIX)
  - `sed` stream editor (UNIX)
  - `lex` A generator for lexical analyzers (UNIX)

- Grammars for programming languages.

  The appendix of [GJSB00] contains a *context free grammar* specifying the syntax of JAVA.

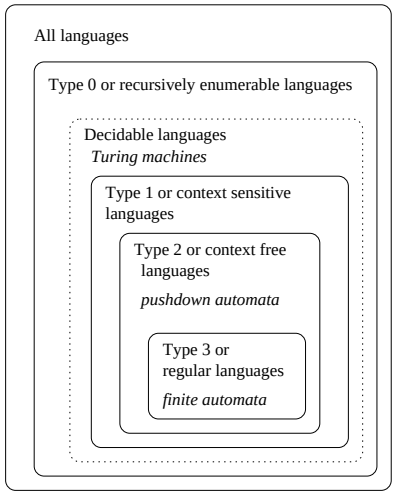
  YACC is a tool which can be used to generate a C program (a parser) from a grammar. Parser generators now also exist for other languages, like Java CUP for Java.

- Specifying protocols

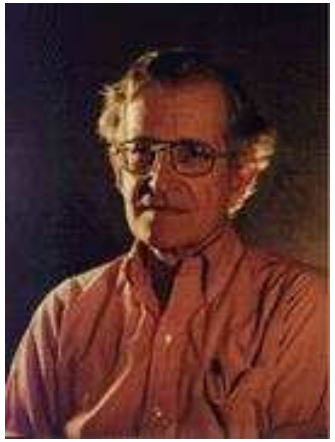
  See section 2.1 (pp 38 - 45) contains a simple example of how a protocol for electronic cash can be specified and analyzed using finite automata.

1.4 History

1.4.1 The Chomsky Hierarchy



Noam Chomsky introduced the *Chomsky hierarchy* which classifies grammars and languages. This hierarchy can be amended by different types of machines (or automata) which can recognize the appropriate class of languages.
Chomsky is also well known for his unusual views on society and politics.



1.4.2 Turing machines

Alan Turing (1912-1954) introduced an abstract model of computation, which we call Turing machines, to give a precise definition which problems can be solved by a computer. All the machines we are introducing can be viewed as restricted versions of Turing machines.
I recommend Andrew Hodges biography Alan Turing: the Enigma.

1.5 Languages

In this course we will use the terms *language* and *word* different than in everyday language:

- A **language** is a set of words.
- A **word** is a sequence of symbols.

This leaves us with the question: what is a symbol? The answer is: anything, but it has to come from an alphabet Σ which is a finite set. A common (and important) instance is $\Sigma = \{0, 1\}$.
More mathematically we say: Given an alphabet Σ we define the set Σ^* as set of words (or sequences) over Σ: the empty word $\epsilon \in \Sigma^*$ and given a symbol $x \in \Sigma^*$ and a word $w \in \Sigma^*$ we can form a new word $xw \in \Sigma^*$. These are all the ways elements on Σ^* can be constructed (this is called an *inductive definition*). E.g. in the example $\Sigma = \{0, 1\}$, typical elements of Σ^* are 0010, 00000000,ϵ. Note, that we only write ϵ if it apperas on its own, instead of 00ϵ we just write 0ϵ. It is also important to realzie that although there are infinite many words, each word has a finite length.
An important operation on Σ^* is concatenation. Confusingly, this is denoted by an *invisible* operator: given $w, v \in \Sigma^*$ we can construct a new word $wv \in \Sigma^*$ simply by concatenating the two words. We can define this operation by

primitive recursion:

$$\begin{aligned} \epsilon v &= v \\ (xw)v &= x(wv) \end{aligned}$$

A language L is a set of words, hence $L \subseteq \Sigma^*$ or equivalently $L \in \mathcal{P}(\Sigma)$. Here are some informal examples of languages:

- The set $\{0010, 00000000, \epsilon\}$ is a language over $\Sigma = \{0, 1\}$. This is an example of a finite language.
- The set of words with odd length over $\Sigma = \{1\}$.
- The set of words which contain the same number of 0s and 1s is a language over $\Sigma = \{0, 1\}$.
- The set of words which contain the same number of 0s and 1s modulo 2 (i.e. both are even or odd) is a language over $\Sigma = \{0, 1\}$.
- The set of palindroms using the english alphabet, e.g. words which read the same forwards and backwards like `abba`. This is a language over $\{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}$.
- The set of correct Java programs. This is a language over the set of UNICODE characters (which correspond to numbers between 0 and $2^{16} - 1$).
- The set of programs, which if executed on a Windows machine, will print the text "Hello World!" in a window. This is a language over $\Sigma = \{0, 1\}$.

2 Finite Automata

Finite automata correspond to a computer with a fixed finite amount of memory. We will introduce *deterministic finite automata* (DFA) first and then move to *nondeterministic finite automata* (NFA). An automaton will accept certain words (sequences of symbols of a given alphabet Σ) and reject others. The set of accepted words is called the language of the automaton. We will show that the class of languages which are accepted by DFAs and NFAs is the same.

2.1 Deterministic finite automata

2.1.1 What is a DFA?

A *deterministic finite automaton* (DFA) $A = (Q, \Sigma, \delta, q_0, F)$ is given by:

1. A finite set of states Q,
2. A finite set of input symbols Σ,
3. A transition function $\delta \in Q \times \Sigma \to Q$,
4. An initial state $q_0 \in Q$,
5. A set of accepting states $F \subseteq Q$.

As an example consider the following automaton

$$\mathrm{D} = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta_\mathrm{D}, q_0, \{q_2\})$$

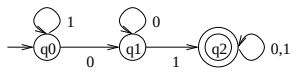
where

$$\delta_\mathrm{D} = \{((q_0, 0), q_1), ((q_0, 1), q_0), ((q_1, 0), q_1), ((q_1, 1), q_2), ((q_2, 0), q_2), ((q_2, 1), q_2)\}$$

The DFA may be more conveniently represented by a transition table:

	0	1
$\to q_0$	q_1	q_0
q_1	q_1	q_2
$*q_2$	q_2	q_2

The table represents the function δ, i.e. to find the value of $\delta(q, x)$ we have to look at the row labelled q and the column labelled x. The initial state is marked by an $\to$ and all final states are marked by $*$.
Yet another, optically more inspiring, alternative are transition diagrams:



There is an arrow into the initial state and all final states are marked by double rings. If $\delta(q, x) = q'$ then there is an arrow from state q to q' which is labelled x.
We write Σ^* for the set of words (i.e. sequences) over the alphabet Σ. This includes the empty word which is written ϵ. I.e.

$$\{0, 1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$$

2.1.2 The language of a DFA

To each DFA A we associate a language $L(A) \subseteq \Sigma^*$. To see whether a word $w \in L(A)$ we put a marker in the initial state and when reading a symbol forward the marker along the edge marked with this symbol. When we are in an accepting state at the end of the word then $w \in L(A)$, otherwise $w \notin L(A)$. In the example above we have that $0 \notin L(\mathrm{D})$,$101 \in L(\mathrm{D})$ and $110 \notin L(\mathrm{D})$. Indeed, we have

$$L(\mathrm{D}) = \{w \mid w \text{ contains the substring } 01\}$$

To be more precise we give a formal definition of $L(A)$. First we define the extended transition function $\hat{\delta} \in Q \times \Sigma^* \to Q$. Intuitively, $\hat{\delta}(q, w) = q'$ if starting from state q we end up in state q' when reading the word w. Formally, $\hat{\delta}$ is defined by primitive recursion:

$$\begin{aligned} \hat{\delta}(q, \epsilon) &= q & (1)\\ \hat{\delta}(q, xw) &= \hat{\delta}(\delta(q, x), w) & (2) \end{aligned}$$

Here xw stands for a non empty word whose first symbol is x and the rest is w. E.g. if we are told that $xw = 010$ then this entails that $x = 0$ and $w = 10$. w may be empty, i.e. $xw = 0$ entails $x = 0$ and $w = \epsilon$.
As an example we calculate $\hat{\delta}_{\mathrm{D}}(q_0, 101) = q_1$:

$$\begin{aligned} \hat{\delta}_{\mathrm{D}}(q_0, 101) &= \hat{\delta}_{\mathrm{D}}(\delta_{\mathrm{D}}(q_0, 1), 01) && \text{by (2)}\\ &= \hat{\delta}_{\mathrm{D}}(q_0, 01) && \text{because } \delta_{\mathrm{D}}(q_0, 1) = q_0\\ &= \hat{\delta}_{\mathrm{D}}(\delta_{\mathrm{D}}(q_0, 0), 1) && \text{by (2)}\\ &= \hat{\delta}_{\mathrm{D}}(q_1, 1) && \text{because } \delta_{\mathrm{D}}(q_0, 0) = q_1\\ &= \hat{\delta}_{\mathrm{D}}(\delta_{\mathrm{D}}(q_1, 1), \epsilon) && \text{by (2)}\\ &= \hat{\delta}_{\mathrm{D}}(q_2, \epsilon) && \text{because } \delta_{\mathrm{D}}(q_1, 1) = q_2\\ &= q_2 && \text{by (1)} \end{aligned}$$

Using $\hat{\delta}$ we may now define formally:

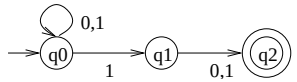
$$L(A) = \{w \mid \hat{\delta}(q_0, w) \in F\}$$

Hence we have that $101 \in L(\mathrm{D})$ because $\hat{\delta}_{\mathrm{D}}(q_0, 101) = q_2$ and $q_2 \in F_{\mathrm{D}}$.

2.2 Nondeterministic Finite Automata

2.2.1 What is an NFA?

Nondeterministic finite automata (NFA) have transition functions which may assign several or no states to a given state and an input symbol. They accept a word if there is any possible transition from the one of initial states to one of the final states. It is important to note that although NFAs have a nondeterministic transition function, it can always be determined whether or not a word belongs to its language ($w \in L(A)$). Indeed, we shall see that every NFA can be translated into an DFA which accepts the same language.
Here is an example of an NFA C which accepts all words over $\Sigma = \{0, 1\}$ s.t. the symbol before the last is 1.



A *nondeterministic finite automaton* (NFA) $A = (Q, \Sigma, \delta, q_0, F)$ is given by:

1. A finite set of states Q,
2. A finite set of input symbols Σ,
3. A transition function $\delta \in Q \times \Sigma \to \mathcal{P}(Q)$,
4. A set of initial state $S \subseteq Q$,
5. A set of accepting states $F \subseteq Q$.

The differences to DFAs are to have start states instead of a single one and the type of the transition function. As an example we have that

$$C = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta_C, \{q_0\}, \{q2\})$$

where δ_C so given by

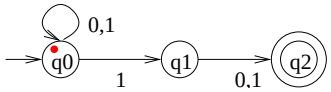
δ_C	0	1
$\to q_0$	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_2\}$	$\{q_2\}$
$*q_2$	$\{\}$	$\{\}$

Note that we diverge he slightly from the definition in the book, which uses a single initial state instead of a set of initial states. Doing so means that we can avoid introducing ϵ-NFAs (see [HMU01], section 2.5).

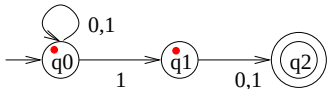
2.2.2 The language accepted by an NFA

To see whether a word $w \in \Sigma^*$ is accepted by an NFA ($w \in L(A)$) we may have to use several markers. Initially we put one marker on the initial state. Then each time when we read a symbol we look at all the markers: we remove the old marker and put markers at all the states which are reachable via an arrow marked with the current input symbol (this may include the state which was

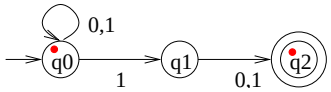
marked in the previously). Thus we may have to use several marker but it may also happen that all markers disappear (if no appropriate arrows exist). In this case the word is not accepted. If at the end of the word any of the final states has a marker on it then the word is accepted.
E.g. consider the word 100 (which is not accepted by C). Initially we have



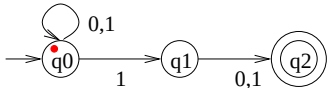
After reading 1 we have to use two markers because there are two arrows from q_0 which are labelled 1:



Now after reading 0 the automaton has still got two markers, one of them in an accepting state:



However, after reading the 2nd 0 the second marker disappears because there is no edge leaving q_2 and we have:



which is not accepting because no marker is in the accepting state.
To specify the extended transition function for NFAs we use an generalisation of the union operation $\cup$ on sets. We define $\bigcup$ to be the union of a (finite) set of sets:

$$\bigcup\{A_1, A_2, \ldots A_n\} = A_1 \cup A_2 \cup \cdots \cup A_n$$

In the special cases of the empty sets of sets and a one element set of sets we define:

$$\bigcup\{\} = \{\} \qquad \bigcup\{A\} = A$$

As an example

$$\bigcup\{\{1\},\{2,3\},\{1,3\}\} = \{1\} \cup \{2,3\} \cup \{1,3\} = \{1,2,3\}$$

Actually, we may define $\bigcup$ by comprehension, which also extends the operation to infinite sets of sets (although we don't need this here)

$$\bigcup B = \{x \mid \exists A \in B.x \in A\}$$

We define $\hat{\delta} \in \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$ with the intention that $\hat{\delta}(S, w)$ is the set of states which is marked after having read w starting with the initial markers given by S.

$$\hat{\delta}(S, \epsilon) = S \tag{3}$$
$$\hat{\delta}(S, xw) = \hat{\delta}(\bigcup\{\delta(q, x) \mid q \in S\}, w) \tag{4}$$

As an example we calculate $\hat{\delta}_C(q_0, 100)$ which is $\{q_0\}$ as we already know from playing with markers.

$$\begin{array}{rcll}
\hat{\delta}_C(\{q_0\}, 100) & = & \hat{\delta}_C(\bigcup\{\delta_C(q,1) \mid q \in \{q_0\}\}, 00) & \text{by (4)} \\
& = & \hat{\delta}_C(\delta_C(q_0, 1), 00) & \\
& = & \hat{\delta}_C(\{q_0, q_1\}, 00) & \\
& = & \hat{\delta}_C(\bigcup\{\delta_C(q,0) \mid q \in \{q_0, q_1\}\}, 0) & \text{by (4)} \\
& = & \hat{\delta}_C(\delta_C(q_0, 0) \cup \delta_C(q_1, 0), 0) & \\
& = & \hat{\delta}_C(\{q_0\} \cup \{q_2\}, 0) & \\
& = & \hat{\delta}_C(\{q_0, q_2\}, 0) & \\
& = & \hat{\delta}_C(\bigcup\{\delta_C(q,0) \mid q \in \{q_0, q_2\}\}, \epsilon) & \text{by (4)} \\
& = & \hat{\delta}_C(\delta_C(q_0, 0) \cup \delta_C(q_2, 0), 0) & \\
& = & \hat{\delta}_C(\{q_0\} \cup \{\}, \epsilon) & \\
& = & \{q_0\} & \text{by (3)}
\end{array}$$

Using the extended transition function we define the language of an NFA as

$$L(A) = \{w \mid \hat{\delta}(S, w) \cap F \neq \{\}\}$$

This shows that $100 \notin L(C)$ because

$$\hat{\delta}(\{q_0\}, 100) = \{q_0\} \cap \{q_2\} = \{\}$$

2.2.3 The subset construction

DFAs can be viewed as a special case of NFAs, i.e. those for which the the there is precisely one start state $S = \{q_0\}$ and the transition function returns always one-element sets (i.e. $\delta(q, x) = \{q'\}$ for all $q \in Q$ and $x \in \Sigma$).
Below we show that for every NFA we can construct a DFA which accepts the same language. This shows that NFAs aren't more powerful as DFAs. However, in some cases NFAs need a lot fewer states than the corresponding DFA and they are easier to construct.
Given an NFA $A = (Q, \Sigma, \delta, S, F)$ we construct the DFA

$$D(A) = (\mathcal{P}(Q), \Sigma, \delta_{D(A)}, S, F_{D(A)})$$

where

$$\delta_{D(A)}(S, x) = \bigcup\{\delta(q, x) \mid q \in S\}$$
$$F_{D(A)} = \{S \subseteq Q_N \mid S \cap F \neq \{\}\}$$

The basic idea of this construction (*the subset construction*) is to define a DFA whose states are sets of states of the NFA. A final state of the DFA is a set

which contains at least a final state of the NFA. The transitions just follow the active set of markers, i.e. a state $S \in \mathcal{P}(Q_N)$ corresponds to having markers on all $q \in S$ and when we follow the arrow labelled x we get the set of states which are marked after reading x.
As an example let us consider the NFA C above. We construct a DFA $D(C)$

$$D(C) = (\mathcal{P}(\{q_0, q_1, q_2\}, \{0, 1\}, \delta_{D(C)}, \{q_0\}, F_{D(C)})$$

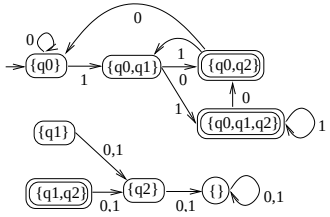
with $\delta_{D(C)}$ given by

$\delta_{D(C)}$	0	1
$\{\}$	$\{\}$	$\{\}$
$\rightarrow \{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\{q_2\}$	$\{q_2\}$
$*\{q_2\}$	$\{\}$	$\{\}$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$*\{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1\}$
$*\{q_1, q_2\}$	$\{q_2\}$	$\{q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

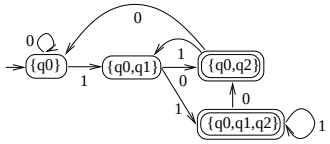
and $F_{D(C)}$ is the set of all the states marked with $*$ above,i.e.

$$F_{D(C)} = \{\{q_2\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$$

Looking at the transition diagram:



we note that some of the states ($\{\}, \{q_1\}, \{q_2\}, \{q_1, q_2\}$) cannot be reached from the initial state, which means that they can be omitted without changing the language. Hence we obtain the following automaton:



We still have to convince ourselves that the DFA $D(A)$ accepts the same language as the NFA A, i.e. we have to show that $L(A) = L(D(A))$.
As a lemma we show that the extended transition functions coincide:

Lemma 2.1

$$\hat{\delta}_{D(A)}(S, w) = \hat{\delta}_A(S, w)$$

The result of both functions are sets of states of the NFA A: for the left hand side because the extended transition function on NFAs returns sets of states and for the right hand side because the states of $D(A)$ are sets of states of A.
Proof: We show this by *induction over the length of the word* w, let's write $|w|$ for the length of a word.

$|w| = 0$ Then $w = \epsilon$ and we have

$$\begin{array}{rcll} \hat{\delta}_{D(A)}(S, \epsilon) & = & S & \text{by (1)} \\ & = & \hat{\delta}_A(S, \epsilon) & \text{by (3)} \end{array}$$

$|w| = n + 1$ Then $w = xv$ with $|v| = n$.

$$\begin{array}{rcll} \hat{\delta}_{D(A)}(S, xv) & = & \hat{\delta}_{D(A)}(\delta_{D(A)}(S, x), v) & \text{by (2)} \\ & = & \hat{\delta}_A(\delta_{D(A)}(S, x), v) & \text{ind.hyp.} \\ & = & \hat{\delta}_A(\bigcup\{\delta_A(q, x) \mid q \in S\}, v) & \\ & = & \hat{\delta}_A(S, xv) & \text{by (4)} \end{array}$$

□

We can now use the lemma to show

Theorem 2.2

$$L(A) = L(D(A))$$

Proof:

$$\begin{array}{c} w \in L(A) \\ \iff \text{Definition of } L(A) \text{ for NFAs} \\ \delta_A(\hat{S}, w) \cap F_A \neq \{\} \\ \iff \text{Lemma 2.1} \\ \hat{\delta}_D(A)(S, w) \cap F_A \neq \{\} \\ \iff \text{Definition of } F_D(A) \\ \hat{\delta}_D(A)(S, w) \in F_{D(A)} \\ \iff \text{Definition of } L(A) \text{ for DFAs} \\ w \in L_{D(A)} \end{array}$$

□

Corollary 2.3 *NFAs and DFAs recognize the same class of languages.*

Proof: We have noticed that DFAs are just a special case of NFAs. On the other hand the subset construction introduced above shows that for every NFA we can find a DFA which recognizes the same language. □