**APPLICATION OF FINITE AUTOMATA**

**0.1.1 A DFA-based Text Filter in Java**

The first thing to deal with is the input alphabet. The DFA above uses the  
alphabet 0, 1, which is the alphabet of interest for this problem. But the  
program will work with a typed input string, so we do not have the luxury  
of restricting the alphabet in this way. The program should accept ”011” (a  
representation for the number 3) and reject ”101” (a representation for the  
number 5), but it must also properly reject strings like ”01i” and ”fred”. The  
alphabet for the Java implementation must be the whole set of characters  
that can occur in a Java stringthat is, the whole set of values making up the  
Java char type. The DFA we actually implement will have four states, like  
this:  
An object of the Mod3 class represents such a DFA. A Mod3 object has  
a current state, which is encoded using the integers 0 through 3. The class  
definition begins like this:  
/\*\*  
A deterministic finite-state automaton that  
recognizes strings that are binary representations  
of natural numbers that are divisible  
by 3. Leading zeros are permitted, and the  
empty string is taken as a representation for 0  
(along with ”0”, ”00”, and so on).  
  
/  
public class Mod3  
/  
Constants q0 through q3 represent states, and  
a private int holds the current state code.  
/  
private static final int q0 = 0;  
private static final int q1 = 1;  
private static final int q2 = 2;  
private static final int q3 = 3;  
private int state;  
The int variables q0, q1, q2, and q3 are private (visible only in this class),  
static (shared by all objects of this class), and final (not permitted to change  
after initialization).

***0.1.2 Acceptors and recognizers***

Acceptors and recognizers (also sequence detectors) produce a binary out-  
put, saying either yes or no to answer whether the input is accepted by the  
machine or not. All states of the FSM are said to be either accepting or not  
accepting. At the time when all input is processed, if the current state is an  
accepting state, the input is accepted; otherwise it is rejected. As a rule the  
input are symbols (characters); actions are not used. The example in figure  
2 shows a finite state machine which accepts the word ”nice”. In this FSM  
the only accepting state is number 7.  
The machine can also be described as defining a language, which would  
contain every word accepted by the machine but none of the rejected ones;  
we say then that the language is accepted by the machine. By definition, the  
languages accepted by FSMs are the regular languages - that is, a language  
is regular if there is some FSM that accepts it.  
Start state  
The start state is usually shown drawn with an arrow ”pointing at it from  
any where”  
Accept state  
An accept state (sometimes referred to as an accepting state) is a state at  
which the machine has successfully performed its procedure. It is usually  
represented by a double circle.  
An example of an accepting state appears on the right in this diagram of  
a deterministic finite automaton (DFA) which determines if the binary input  
4  
contains an even number of 0s.  
S1 (which is also the start state) indicates the state at which an even num-  
ber of 0s has been input and is therefore defined as an accepting state. This  
machine will give a correct end state if the binary number contains an even  
number of zeros including a string with no zeros. Examples of strings ac-  
cepted by this DFA are epsilon (the empty string), 1, 11, 11..., 00, 010, 1010,  
10110 and so on.  
  
***0.1.3 Transducers***

Transducers generate output based on a given input and/or a state using  
actions. They are used for control applications and in the field of computa-  
tional linguistics. Here two types are distinguished:  
Moore machine  
The FSM uses only entry actions, i.e. output depends only on the state.  
The advantage of the Moore model is a simplification of the behaviour. Con-  
sider an elevator door. The state machine recognizes two commands ”com-  
mand open” and ”command close” which trigger state changes. The entry  
action (E:) in state ”Opening” starts a motor opening the door, the entry  
action in state ”Closing” starts a motor in the other direction closing the  
door. States ”Opened” and ”Closed” don’t perform any actions. They sig-  
nal to the outside world (e.g., to other state machines) the situation: ”door  
is open” or ”door is closed”.  
Mealy machine  
The FSM uses only input actions, i.e., output depends on input and state.  
The use of a Mealy FSM leads often to a reduction of the number of states.  
The example in figure 4 shows a Mealy FSM implementing the same be-  
haviour as in the Moore example (the behaviour depends on the implemented  
FSM execution model and will work, e.g., for virtual FSM but not for event  
driven FSM). There are two input actions (I:): ”start motor to close the door  
if command close arrives” and ”start motor in the other direction to open the  
door if command open arrives”. The ”opening” and ”closing” intermediate  
states are not shown.  
Figure 2: Transducer FSM: Mealy model example  
A further distinction is between deterministic (DFA) and non-deterministic  
(NDFA, GNFA) automata. In deterministic automata, for each state there is  
exactly one transition for each possible input. In non-deterministic automata,  
there can be none, one, or more than one transition from a given state for  
a given possible input. This distinction is relevant in practice, but not in  
theory, as there exists an algorithm which can transform any NDFA into an  
equivalent but much more complex DFA.  
The FSM with only one state is called a combinatorial FSM and uses only  
input actions. This concept is useful in cases where a number of FSM are  
required to work together, and where it is convenient to consider a purely  
6  
combinatorial part as a form of FSM to suit the design tools.

0.1.4 UML state machines  
The Unified Modeling Language has a very rich semantics and notation  
for describing state machines. UML state machines overcome the limitations  
of traditional finite state machines while retaining their main benefits. UML  
state machines introduce the new concepts of hierarchically nested states  
and orthogonal regions, while extending the notion of actions. UML state  
machines have the characteristics of both Mealy machines and Moore ma-  
chines. They support actions that depend on both the state of the system  
and the triggering event, as in Mealy machines, as well as entry and exit ac-  
tions, which are associated with states rather than transitions, as in Moore  
machines.  
Figure 3: UML state machine example (a toaster oven)

***0.1.5 Hardware applications***

In a digital circuit, an FSM may be built using a programmable logic  
device, a programmable logic controller, logic gates and flip flops or relays.  
More specifically, a hardware implementation requires a register to store state  
variables, a block of combinational logic which determines the state transi-  
tion, and a second block of combinational logic that determines the output  
of an FSM. One of the classic hardware implementations is the Richards  
controller.  
Mealy and Moore machines produce logic with asynchronous output, be-  
cause there is a propagation delay between the flip-flop and output. This  
causes slower operating frequencies in FSM. A Mealy or Moore machine can  
be convertable to a FSM which output is directly from a flip-flop, which  
makes the FSM run at higher frequencies. This kind of FSM is sometimes  
  
called Medvedev FSM. A counter is the simplest form of this kind of FSM.

SOHand :an ontology-based platform for building services to exploit contex-  
tual handovers information.  
Introduction:  
In future communication scenarios, mobile devices with multiple wireless in-  
terfaces will be able to seamlessly roam around, hoping from a network to an-  
other (and using different technologies) without losing their IP connection. A  
combination of technologies which includes Local Area Networks (WLANs),  
Wireless Personal Area Networks (PANs), Cellular Networks (GSM, GPRS,  
UTMS) and Community Area Networks (WiMax, 802.11) will provide the  
infrastructure needed for this environment. One challenge faced for the de-  
signers of this scenario is to minimize the perception of the change in quality  
when a handover (the process of changing the point of network attachment)  
occurs. Handovers can happen between 2 domains using the same tech-  
nology (horizontal handover) or from different technologies (vertical han-  
dover). Moreover, the vertical handover is further split in downward (when  
the handover allows for a decrease of bandwidth) and upward (when the new  
bandwidth is larger). Such a rich environment could bring up a new world  
of business possibilities and the proposal described here tries to create the  
proper conditions to exploit those innovations. The Service Oriented Han-  
dover SOHand platform adds to the handover process the awareness of the  
ecosystem in which the event is embedded [14 ]. Uses of the Platform  
1. From the network management perspective (the access provider):  
• It may be interested on grabbing usual information which would be inter-  
esting to improve long term relationship with the user;  
• It may collect information about the handovers of a user and relate them  
with positioning information;  
• It may correlate information about routes and timing of accesses.  
2. From the content provider perspective:  
• It may adapt the delivery of media to specifics of devices, location, timing,  
type of user, etc  
• It can understand the criteria by which the user chooses the access providers;  
• It can explore the contextual information to add value to the content (ad-  
vertisement, linking to 3rd party products, etc);  
• As the user has the control over the mobility aspects, the access provider  
can focus on providing better and more varied services[4];  
• It can provide brokerage services based on the common information available.  
3. From the user perspective:  
• It can choose the provider based upon several criteria:  
• Best price  
• Better response  
• Best matching to his/her requirements  
• Economy (use a home user-owned WLAN, during a traffic jam)  
• It can use contextual information to adapt the user device profile of usage:  
• Power management (CPU, memory, display, network interface)  
• Streaming control (proximity to known blind spots such as tunnels)  
• Optimization of content delivery to a new profile, on vertical handovers  
with a lower bandwidth provider.  
The Relationship with PROTON  
A Mobile IPv6 environment, connected to a Vodafones GPRS network, has  
been set up at the Computer Laboratory of the University of Cambridge in an  
effort to demonstrate a 4G mobile scenario[1]. The main aims for the testbed  
were to assess the performance issues involved in the wireless overlays in a  
heterogeneous environment and to improve the seamlessly capabilities of the  
handover process. Figure 2 shows the system implemented at the Computer  
  
Lab.  
The structure of the Ontology  
A service is a facility (a video streamer, a voice channel or a game appli-  
cation) which a content provider offers, during a session, to a user through  
an access provider. One entity can offer both access and content at the same  
time. In the course of enjoying a service, the user seamlessly roams through  
a net of access providers. Context and Handover information, gathered by  
positioning sensors and other sources from the user device or by any other  
related service, can be used both by the session to frame Security, Privacy,  
QoS and other policies. Positioning information can also be used for the def-  
inition route patterns used for one user, which could lead to better pricing  
strategies for the user. Some sort ofClasses and relationship  
SLA is signed between all the entities involved in order to offer to the user  
some parameters by which they can measure his/her Quality of Experience  
while using the system Figure 4.  
This ontology will allow for:  
• The creation of a common vocabulary of terms which would easy the de-  
sign and reuse in new services in the communications industry, with faster  
deployment and exploration in the added value chain;  
• Definition of complex relationships between the terms which would make  
possible to correlate the business processes, exploring new possibilities de-  
rived from the positioning and context awareness technologies as well as from  
security/privacy policies;  
• A structured integration of the Access Networks, Subscriber Profiles, Ap-  
plications and Data both by a provider, or by a group of providers;  
• Other ontologies in the IT management domain can be imported, increas-  
ing the management boundaries.  
Each provider will have its own policies about IT Management. Some of  
these policies can be shared between providers to deliver ubiquitous services,  
and others can be protected or hidden for business reasons.  
Methodology and adherence to standards  
We have particular preoccupation on following established or emerging stan-  
dards in the implementation of this project. The framework is based on  
SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications)[7].  
This is an interesting model which divides the system in core ontologies and  
extension ontologies Figure 6. SOUPA Core defines wide range terms and  
relationships that are of general use for different ubiquitous applications.  
  
SOUPA Extension defines vocabularies for specific types of applications. We  
understand that, for this platform, the core ontology will take care of the  
common vocabulary which will be used by all the entities belonging to the  
environment. Some of the extensions will be designed by the providers in  
order to create their own private vocabularies, most of them will not be  
available to the other (most of the times competing) entities. The overall  
methodology which will be used for ontology development is the Methontol-  
ogy[8].  
The ontology will be described in OWL[9] (Web Ontology Language),  
which is the prime language created for the Semantic Web. OWL is incor-  
porated on the Protg tool[10], which is a good platform to design ontologies.  
The Jena API[11] is used by Protg-OWL for various tasks during the devel-  
opment and prototyping of the ontology and its applications.  
The development of the applications that use the ontology will be based on  
Service Components, a set of programs and data that executes functionalities  
which are relevant in the context of a business. In this model, the activities  
can be set as having technical meaning (the updating of a table in a database)  
or a business meaning (the update of a customer address). A model which is  
emerging as a good standard for developing applications is the Service Ori-  
ented Architecture (SOA)[12]. SOA-based applications make available inter  
faces for other applications via service components. Through the pipelining of  
multiple components via request/reply remote calls more complex composite  
applications, a logical module in a larger business model, can be constructed.  
Another model, the Event Driven Architecture (EDA)[13], defines a model  
of developing application components which exchange events to implement  
business functions. It differentiates itself from SOA as in EDA all the com-  
ponents keep working, processing messages, while awaiting for response of a  
previous calls, while SOA will block until the query is answered. There are  
long arguments about the preferences on using SOA or EDA model, or if any  
of them should provide the right paradigm, however, we intend to follow at  
least the general ideas from them in developing the prototyping applications  
to test SOHand.  
The environment has been installed composed by APs from CISCO, 4  
PDAs from HP (5550), 2 notebooks, 3 GPSs, 2 Switches 3COM and 4 Desk-  
tops. Linux have been used on the notebooks and windows on the PDAs.