SAN JOSE STATE UNIVERSITY
DEPARTMENT OF ELECTRICAL ENGINEERING

CS 154 Formal Languages and Computability Spring 2012
Section 1 Room MH 225 Class 4: 02-06-12

T. Howell

Homework is due on Wednesday. Submit paper copies in class or soft copies by dropbox.

Summary of last class

We covered $\hat{\delta}$, the product construction, and started nondeterminism. We did a proof by induction that $\hat{\delta}$ for binary numbers divisible by 3 works as advertised.

We looked at a nondeterministic finite automaton that accepts A = {x ∈ {0,1}* | the fifth symbol from the right is 1.}

We converted an NFA N for the language of binary strings ending in 01 to a DFA D using the subset construction. The result looks like this.

		0	1
→	{u}	{u, v}	{u}
	{u, v}	{u, v}	{u, w}
	*{u, w}	{u, v}	{u}

This DFA is equivalent to the original NFA: L(D) = L(N). The proof is by induction on the length of x. Hypothesis: for |x| ≤ n, the state of D is the set of states that N could be in after processing x. Basis: |x| = 0, N is in start state u, and D is in {u}. If hypothesis is true for string x with |x| = n, one step using the $\hat{\delta}$ definition shows the hypothesis is true for |x| = n+1. By induction the hypothesis is true for all n ≥ 0.

Pattern matching

NFA's are useful to describe pattern matching. The Unix `grep` command is one example. Demonstrate in directory cs47prg:

```
grep decimal h2d.c d2h.c

h2d.c:Converts a list of hexadecimal integers to decimal
h2d.c:Outputs are two's complement decimal values.
h2d.c:/* Convert one hexadecimal number into decimal and print both*/
h2d.c:   if(argc < 2) printf("Usage: h2d list of hexadecimal values\n");
d2h.c:Converts a list of decimal integers to hexadecimal
d2h.c:/* Convert one decimal number into hex and print both*/
d2h.c:   if(argc < 2) printf("Usage: d2h list of decimal values\n");


grep /[0-9]*/ h2d.c d2h.c
```

```
h2d.c:T. Howell Section 1.        revised 1/12/11
h2d.c:   if(*valstr == '0' && *(valstr+1) == 'x') valstr += 2; // skip "0x"
h2d.c:   while ((c = *valstr++) != '\0')   //continue until end of string
h2d.c:   for (i = 1; i < argc; i++) // iterate through the list of arguments
d2h.c:T. Howell Section 1         1/12/11
d2h.c:      while ((c = *valstr++) != '\0')   //continue until end of string
```

The patterns are regular expressions which we will meet shortly. They are another way of describing regular languages. See below for some more detailed material from Kozen.

ε-transitions

NFA's can be extended with transitions labeled ε. These transitions can be taken without "using up" any input characters. Note that ε is *not* in the alphabet of the language accepted by the NFA. Again, ε-transitions don't add any languages to the set that can be described, but they sometimes make the job easier. They will be useful when we come to showing the equivalence of regular languages and languages described by regular expressions.

Example (Accepts decimal numbers)

		0 - 9	+, -	.	ε
→	q_0	∅	$\{q_1\}$	∅	$\{q_1\}$
	q_1	$\{q_1, q_4\}$	∅	$\{q_2\}$	∅
	q_2	$\{q_3\}$	∅	∅	∅
	q_3	$\{q_3\}$	∅	∅	$\{q_5\}$
	q_4	∅	∅	$\{q_3\}$	∅
	*q_5	∅	∅	∅	∅

The ε-closure of a state q is the set of states reachable from q by paths whose arcs are labeled ε. The ε-closure of a set of states is the union of the ε-closures of the states in the set.

Given an NFA with ε-transitions, we can apply essentially the same subset construction as before to convert it to a DFA (without ε-transitions). Whenever we create a set of NFA states to be the name of a new DFA state, we have to take its ε-closure first.

For our example accepting decimal numbers, this procedure leads to

		0 - 9	+, -	.
→	$\{q_0, q_1\}$	$\{q_1,q_4\}$	$\{q_1\}$	$\{q_2\}$
	$\{q_1\}$	$\{q_1, q_4\}$	∅	$\{q_2\}$
	$\{q_2\}$	$\{q_3,q_5\}$	∅	∅
	*$\{q_2,q_3,q_5\}$	$\{q_3,q_5\}$	∅	∅
	$\{q_1,q_4\}$	$\{q_1,q_4\}$	∅	$\{q_2,q_3,q_5\}$
	*$\{q_3,q_5\}$	$\{q_3,q_5\}$	∅	∅

The textbook has inductive definitions and proofs. I will skip them here.

Grammars

Regular languages can be described by (right or left regular) grammars. We just mention them briefly because we will use regular expressions instead. But we will come back to grammars for context-free languages.

Example: (Left regular) grammar for $\{x \in \{0, 1\}^* \mid x \text{ represents a multiple of } 3\}$.

$Z \rightarrow 0 \mid Z0 \mid O1$
$O \rightarrow 1 \mid Z1 \mid T0$
$T \rightarrow O0 \mid T1$

Draw automaton and show the connection between it and the grammar: Transition from Q to R on input `a` generates a production $Q \rightarrow R\texttt{a}$. If R is accept state it also generates $Q \rightarrow \texttt{a}$.

Typical derivation: $Z \rightarrow O1 \rightarrow T01 \rightarrow O001 \rightarrow 1001$, representing 9, a multiple of 3.

Gradiance

I have created two gradiance assignments for you. Try them and tell me how you like them. There are several problems from the gradiance system and a few of my own. Warning: some of the gradiance problems have no correct answers, so it is very hard to get an assignment all right.

Let's try one live in class if we have time.

We might have time for homework questions, too.

Pattern matching notes from Kozen, D. Automata and Computability.

A real-world application of regular languages is the Unix pattern matching utility "grep". A pattern is given and certain strings *match* it.

We define a *pattern* α inductively as an atomic pattern or a compound pattern built up from atomic patterns using operators.
A pattern defines a language: $L(\alpha) = \{x \in \Sigma^* \mid x \text{ matches } \alpha\}$.

Atomic patterns are:

Single symbol	$L(a) = \{a\}$ for every $a \in \Sigma$.
Empty string	$L(\boldsymbol{\varepsilon}) = \{\varepsilon\}$.
Empty set	$L(\boldsymbol{\varnothing}) = \varnothing$.
Wildcard symbol	$L(\#) = \Sigma$.
Wildcard string	$L(@) = \Sigma^*$

Operators are:

Union	$L(\alpha + \beta) = L(\alpha) \cup L(\beta)$
Intersection	$L(\alpha \cap \beta) = L(\alpha) \cap L(\beta)$
Concatenation	$L(\alpha\beta) = L(\alpha)\, L(\beta) = \{yz \mid y \in L(\alpha) \text{ and } z \in L(\beta)\}$
Complement	$L(\sim\alpha) = \Sigma^* - L(\alpha)$
Asterate	$L(\alpha^*) = (L(\alpha))^* = L(\alpha)^0 \cup L(\alpha)^1 \cup L(\alpha)^2 \cup \ldots$
Proper asterate	$L(\alpha^+) = (L(\alpha))^+ = L(\alpha)^1 \cup L(\alpha)^2 \cup L(\alpha)^3 \cup \ldots$

A pattern is a string of symbols over $\Sigma \cup \{\boldsymbol{\varepsilon}, \boldsymbol{\varnothing}, \#, @, +, \cap, \sim, {}^*, {}^+, (,)\}$.

Examples:

$L(\#^*) = L(@) = \Sigma^*$
$L(x) = \{x\}$. Note that any string x is a pattern.
Finite sets. $L(x_1 + x_2 + \ldots + x_n) = \{x_1, x_2, \ldots, x_n\}$

@a@a@a@ = any string with at least 3 a's
@a@b@ = any string with a and b (in that order).
$\# \cap \sim a$ = any letter except a
$(\# \cap \sim a)^*$ = any string not containing a

Mention double use of ε and ∅. Also use of pattern to mean L(pattern).