Lecture 6: Closure properties

February 5, 2009

This lecture covers the last part of section 1.2 of Sipser (pp. 58–63), beginning of 1.3 (pp. 63–66), and also closure under string reversal. We also include a proof of the string-reversal construction and the union construction for NFAs, which Sipser unfortunately does not.

1 Operations on languages

Regular operations on languages (sets of strings). Suppose L and K are languages.

- ***Union***: $L \cup K = \left\{ x \;\middle|\; x \in L \quad \text{or} \quad x \in K \right\}$.
- ***Concatenation***: $L \circ K = LK = \left\{ xy \;\middle|\; x \in L \quad \text{and} \quad y \in K \right\}$.
- ***Star*** (***Kleene star***):

$$L^* = \left\{ w_1 w_2 \dots w_n \;\middle|\; w_1, \dots, w_n \in L \text{ and } n \geq 0 \right\}.$$

We (hopefully) all understand what union does. The other two have some subtleties. Let

$$L = \{\texttt{under}, \texttt{over}\}, \quad \text{and} \quad K = \{\texttt{ground}, \texttt{water}, \texttt{work}\}.$$

Then

$$LK = \{\texttt{underground}, \texttt{underwater}, \texttt{underwork}, \texttt{overground}, \texttt{overwater}, \texttt{overwork}\}.$$

Similarly,

$$K^* = \left\{ \begin{array}{l} \epsilon, \texttt{ground}, \texttt{water}, \texttt{work}, \texttt{groundground}, \\ \texttt{groundwater}, \texttt{groundwork}, \texttt{workground}, \\ \texttt{waterworkwork}, \dots \end{array} \right\}.$$

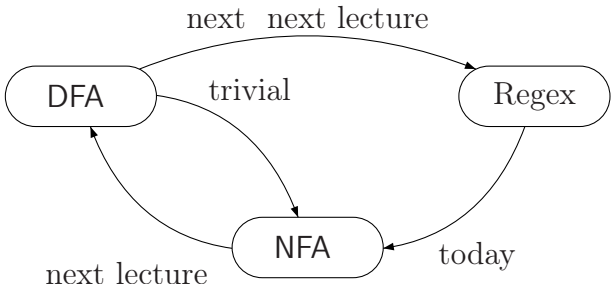
For star operator, note that the resulting set *always* contains the empty string ϵ (because n can be zero).

Also, each of the substrings is chosen independently from the base set and you can repeat. E.g. `waterworkwork` is in K^*.

Regular languages are closed under many operations, including the three "regular operations" listed above, set intersection, set complement, string reversal, "homomorphism" (formal version of shifting alphabets). We have seen (last class) why regular languages are closed under set complement. We will prove the rest of these bit by bit over the next few lectures.

2 Overview of closure properties

We defined a language to be ***regular*** if it is recognized by some DFA. The agenda for the new few lectures is to show that three different ways of defining languages, that is NFAs, DFAs, and regexes, and in fact all equivalent; that is, they all define regular languages. We will show this equivalence, as follows.



One of the main properties of languages we are interested in are closure properties, and the fact that regular languages are closed under union, intersection, complement, concatenation, and star (and also under homomorphism).

However, closure operations are easier to show in one model than the other. For example, for DFAs showing that they are closed under union, intersection, complement are easy. But showing closure of DFA under concatenation and $*$ is hard.

Here is a table that lists the closure property and how hard it is to show it in the various models of regular languages.

Model / Property	$\cap$ intersection	$\cup$ union	$\overline{L}$ complement	$\circ$ concatenation	$*$ star
DFA	Easy (done)	Easy (done)	Easy (done)	Hard	Hard
NFA	Doable (hw?)	Easy: Lemma 4.1	Hard	Easy: Lemma 4.2	Easy: Lemma 4.3
regex	Hard	Easy	Hard	Easy	Easy

Recall what it means for regular languages to be closed under an operation op. If L_1 and L_2 are regular, then L_1 op L_2 is regular. That is, if we have an NFA recognizing L_1 and an NFA recognizing L_2, we can construct an NFA recognizing L_1 op L_2.

The extra power of NFAs makes it easy to prove closure properties for NFAs. When we know all DFAs, NFAs, and regexes are equivalent, these closure results then apply to all three representations. Namely, they would imply that regular languages have these closure properties.

3 Closure under string reversal for NFAs

Consider a word w, we denote by w^R the ***reversed*** word. It is just w in the characters in reverse order. For example, for $w =$ `barbados`, we have $w^R =$ `sodabrab`. For a language L,

the ***reverse*** language is

$$L^R = \left\{ w^R \;\middle|\; w \in L \right\}.$$

We would like to claim that if L is regular, then so is L^R. Formally, we need to be a little bit more careful, since we still did not show that a language being regular implies that it is recognized by an NFA.

Claim 3.1 *If L is recognized by an NFA, then there is an NFA that recognizes L^R.*

Proof: Let M be an NFA recognizing L. We need to construct an NFA N recognizing L^R.

The idea is to reverse the arrows in the NFA $M = (Q, \Sigma, \delta, q_0, F)$, and swap final and initial states. There is a bug in applying this idea in a naive fashion. Indeed, there is only one initial state but multiple final states.

To overcome this, let us modify M to have a single final state q_S, connected to old ones with epsilon transitions. Thus, the modified NFA accepting L, is

$$M' = \left(Q \cup \{q_S\}, \Sigma, \delta', q_0, \{q_S\}\right),$$

where q_S is the only accepting state for M. Note, that δ' is identical to δ, except that

$$\forall q \in F \quad \delta'(q, \epsilon) = q_S. \tag{1}$$

Note, that $L(M) = L(M') = L$.

As such, q_S will become the start state of the "reversed" NFA.

Now, the new "reversed" NFA N, for the language L^R, is

$$N = \left(Q \cup \{q_S\}, \Sigma, \delta'', q_S, \{q_0\}\right).$$

Here, the transition function δ'' is defined as

(i) $\delta''(q_0, t) = \emptyset$ for every $t \in \Sigma$.

(ii) $\delta''(q, t) = \left\{ r \in Q \;\middle|\; q \in \delta'(r, t) \right\}$, for every $q \in Q \cup \{q_S\}$, $t \in \Sigma_\epsilon$.

(iii) $\delta''(q_S, \epsilon) = F$ (the reversal of Eq. (1)). [1]

Now, we need to prove formally that if $w \in L(M)$ then $w^R \in L(N)$. To this end, let us prove the following claim.

Lemma: For every word w, the automaton M' can reach a state q from its initial state q_0 reading w iff the automaton N, when started from state q, can reach the state q_0 reading w^R.

Proof of lemma:

Let $w \in \Sigma^*$ and let M' reach a state q from q_0 on reading w, say using a sequence of states $r_0, r_1, \ldots, r_k$ (where $r_0 = q_0$ and $r_k = q$). Then, since N contains the reversal of the edges in M', the automaton N, when started from state q, can reach q_0 on reading w^R using the sequence $r_k, r_{k-1}, \ldots, r_1, r_0$.

[1] This can be omitted, since it is implied by the (ii) rule.

Similarly, assume that N, when started from state q, can read w and end in state q_0, say using a sequence of states $r_0, r_1, \ldots, r_k$, where $r_0 = q$ and $r_k = q$. Then, since M' contains the reversal of the edges in N, M' when started at q_0 can reach q on reading w^R, using the sequence of states $r_k, r_{k-1}, \ldots, r_1, r_0$.

End of proof of lemma.

Let us now use the claim to prove that $w \in L(M')$ iff $w^R \in L(N)$.

$w \in L(M')$ iff M can reach q_s reading w when started from q_0
iff N can reach q_0 reading w^R when started from q_s
iff $w^R \in L(N)$.

■

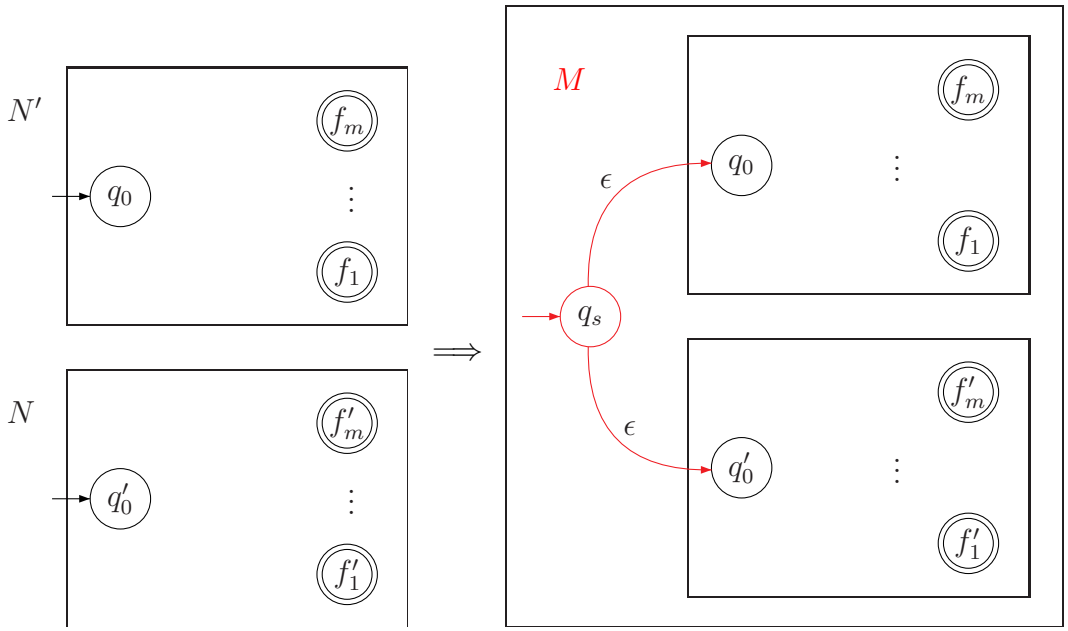
Note, that this will not work for a DFA. First, we can not force a DFA to have a single final state. Second, a state may have two incoming transitions on the same character, resulting in non-determinism when reversed.

4 Closure of NFAs under regular operations

We consider the ***regular operations*** to be union, concatenation, and the star operator.

4.1 NFA closure under union

Given two NFAs, say N and N, we would like to build an NFA for the language $L(N) \cup L(N)$. The idea is to create a new initial state q_s and connect it with an ϵ-transition to the two initial states of N and N. Visually, the resulting NFA M looks as follows.



Formally, we are given two NFAs $N = (Q, \Sigma, \delta, q_0, F)$ and $N' = (Q', \Sigma, \delta', q'_0, F')$, where $Q \cap Q' = \emptyset$ and the new state q_s is not in Q or Q'. The new NFA M is

$$M = (Q \cup Q' \cup \{q_s\}, \Sigma, \delta_M, s_q, F \cup F'),$$

where

$$\delta_M(q,c) = \begin{cases} \delta(q,c) & q \in Q, c \in \Sigma_\epsilon \\ \delta'(q,c) & q \in Q', c \in \Sigma_\epsilon \\ \{q_0, q_0'\} & q = q_s, c = \epsilon \\ \emptyset & q = q_s, c \neq \epsilon. \end{cases}$$

Let us now formally prove that $L(M) = L(N) \cup L(N')$.

First, let us show that $L(N) \cup L(N') \subseteq L(M)$. Let $w \in L(N) \cup L(N')$. So $w \in L(N)$ or $w \in L(N')$. Let us consider the case when $w \in L(N)$ (the other case is similarly argued). Since $w \in L(N)$, by definition, there exists a sequence of states $r_0, r_1, \ldots, r_k$ that is a run of N on w, where $r_0 = q_0$ and $r_k \in F$. Now consider the run $q_s, r_0, \ldots, r_k$ in M. Since $q_0 \in \delta(q_s, \epsilon)$, and the transitions of N are all present in M, this sequence is a run of M on w. Since the final states of M include the final states of N, r_k is a final state of M as well, and hence this run is accepting. Hence $w \in L(M)$. The case when $w \in L(N')$ can be argued similarly. Hence $L(N) \cup L(N') \subseteq L(M)$.

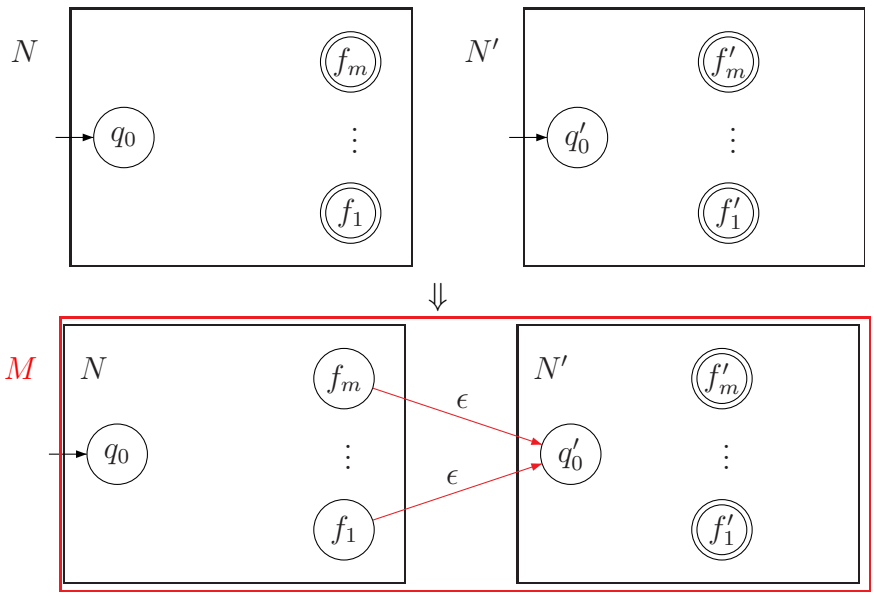
Now let us show that $L(M) \subseteq L(N) \cup L(N')$. Let $w \in L(M)$. Then, by definition, there exists an accepting run $r_0, r_1, \ldots, r_k$ of M on w with $r_0 = q_s$ and $r_k \in F \cup F'$. First, $k \neq 0$, as if $k = 0$, then $r_k = r_0 = q_s \notin F \cup F'$, which is a contradiction. Notice that since there is only one transition from q_s, we must have that $r_1 = q_0$ or $r_1 = q_0'$. Let us assume that $r_1 = q_0$ (the other case is similarly argued). Notice that taking transitions starting from q_0 will keep us within the states of N, and hence $r_k \in F$. Further, since all these transitions belong to N, and since the transition from q_s to q_0 was on ϵ, the run $r_1, \ldots, r_k$ must be a run in N on the word w. Since $r_k \in F$, $w \in L(N)$. In the other case, when $r_1 = q_0'$, we can similarly argue that $w \in L(N')$. Hence $w \in L(N) \cup L(N')$.

We have thus showed the following.

Lemma 4.1 *Given two NFAs N and N', one can construct an NFA M (as given above), such that $L(M) = L(N) \cup L(N')$.*

4.2 NFA closure under concatenation

Given two NFAs N and N', we would like to construct an NFA for the concatenated language $L(N) \circ L(N') = \left\{ xy \;\middle|\; x \in L(N) \text{ and } y \in L(N') \right\}$. The idea is to concatenate the two automata, by connecting the final states of the first automata, by ϵ-transitions, into the start state of the second NFA. We also make the accepting states of N not-accepting. The idea is that in the resulting NFA M, given input w, it "guesses" how to break it into two strings $x \in L(N)$ and $y \in L(N')$, so that $w = xy$. Now, there exists an execution trace for N accepting x, then we can jump into the starting state of N' and then use the execution trace accepting y, to reach an accepting state of the new NFA M. Here is how visually the resulting automata looks like.



Formally, we are given two NFAs $N = (Q, \Sigma, \delta, q_0, F)$ and $N' = (Q', \Sigma, \delta', q'_0, F')$, where $Q \cap Q' = \emptyset$. The new automata is

$$M = (Q \cup Q', \Sigma, \delta_M, q_0, F'),$$

where

$$\delta_M(q, c) = \begin{cases} \delta(q, \epsilon) \cup \{q'_0\} & q \in F, c = \epsilon \\ \delta(q, c) & q \in F, c \neq \epsilon \\ \delta(q, c) & q \in Q \setminus F, c \in \Sigma_\epsilon \\ \delta'(q, c) & q \in Q', c \in \Sigma_\epsilon. \end{cases}$$

Lemma 4.2 *Given two NFAs N and N' one can construct an NFA M, such that $L(M) = L(N) \circ L(N') = L(N)L(N')$.*

Proof: The construction is described above, and the proof of the correctness (of the construction) is easy and sketched above, so we skip it. You might want to verify that you know how to fill in the details for this proof (wink, wink). ■

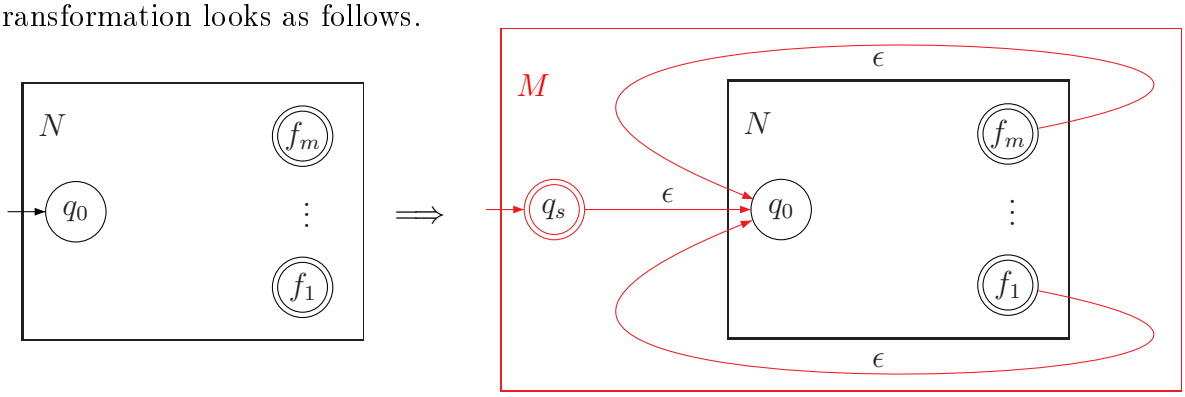
4.3 NFA closure under the (Kleene) star

We are given a NFA N, and we would like to build an NFA for the Kleene star language

$$(L(N))^* = \left\{ w_1 w_2 \dots w_k \;\middle|\; w_1, \dots, w_k \in L(N), k \geq 0 \right\}.$$

The idea is to connect the final states of N back to the initial state using ϵ-transitions, so that it can loop back after recognizing a word of $L(N)$. As such, in the ith loop, during the execution, the new NFA M recognized the word w_i. Naturally, the NFA needs to guess when to jump back to the start state of N. One minor technicality, is that $\epsilon \in (L(N))^*$, but it might not be in $L(N)$. To overcome this, we introduce a new start state q_s (which

is accepting), and its connected by (you guessed it) an ϵ-transition to the initial state of N. This way, $\epsilon \in L(M)$, and as such it recognized the required language. Visually, the transformation looks as follows.



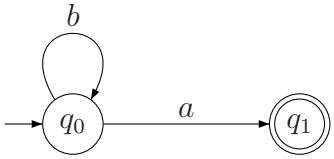
Formally, we are given the NFA $N = (Q, \Sigma, \delta, q_0, F)$, where $q_s \notin Q$. The new NFA is

$$M = \Big(Q \cup \{q_s\},\ \Sigma,\ \delta_M,\ q_s,\ F \cup \{q_s\} \Big),$$

where

$$\delta_M(q, c) = \begin{cases} \delta(q, \epsilon) \cup \{q_0\} & q \in F, c = \epsilon \\ \delta(q, \epsilon) & q \in F, c \neq \epsilon \\ \delta(q, c) & q \in Q \setminus F \\ \{q_0\} & q = q_0, c = \epsilon \\ \emptyset & q = q_0, c \neq \epsilon. \end{cases}$$

Why the extra state? The construction for star needs some explanation. We add arcs from final states back to initial state to do the loop. But then we need to ensure that ϵ is accepted. It's tempting to just make the initial state final, but this doesn't work for examples like the following. So we need to add a new initial state to handle ϵ.



Notice that it also works to send the loopback arcs to the new initial state rather than to the old initial state.

Lemma 4.3 *Given an NFA N, one can construct an NFA M that accepts the language $(L(N))^*$.*

We don't give a proof; but you are encouraged to write a formal proof (as done for the union construction).

5 Regular Expressions

Regular expressions are a convenient notation to specify regular languages. We will prove in a few lectures that regular expressions can represent ***exactly*** the same languages that

DFAs can accept.

Let us fix an alphabet Σ. Here are the basic regular expressions:

regex	conditions	set represented
$\mathtt{a}$	$\mathtt{a} \in \Sigma$	$\{\mathtt{a}\}$
ϵ		$\{\epsilon\}$
$\emptyset$		$\{\}$

Thus, $\emptyset$ represents the empty language. But ϵ represents that language which has the empty word as its only word in the language.

In particular, for a regular expression $\langle \mathtt{exp} \rangle$, we will use the notation $L(\langle \mathtt{exp} \rangle)$ to denote the language associated with this regular expression. Thus,

$$L(\epsilon) = \{\epsilon\} \quad \text{and} \quad L(\emptyset) = \{\}\,,$$

which are two *different* languages.

We will slightly abuse notations, and write a regular expression $\langle \mathtt{exp} \rangle$ when in reality what we refer to is the language $L(\langle \mathtt{exp} \rangle)$. (Abusing notations should be done with care, in cases where it reduces clutter, but it is well defined. Naturally, as Humpty Dumpty does, you need to define your "abused" notations explicitly.[2])

Suppose that $L(R)$ is the language represented by the regular expression R. Here are recursive rules that make complex regular expressions out of simpler ones. (Lecture will add some randomly-chosen small concrete examples.)

regex	conditions	set represented
$R \cup S$ or $R + S$	R, S regexes	$L(R) \cup L(S)$
$R \circ S$ or RS	R, S regexes	$L(R)L(S)$
R^*	R a regex	$L(R)^*$

And some handy shorthand notation:

regex	conditions	set represented
R^+	R a regex	$L(R)L(R)^*$
Σ		Σ

Exponentiation binds most tightly, then multiplication, then addition. Just like you probably thought. Use parentheses when you want to force a different interpretation.

Some specific boundary case examples:

1. $R\epsilon = R = \epsilon R$.
2. $R\emptyset = \emptyset = \emptyset R$.

[2] From *Through the Looking Glass*, by Lewis Carroll:

> 'And only one for birthday presents, you know. There's glory for you!'
>
> 'I don't know what you mean by "glory",' Alice said.
>
> Humpty Dumpty smiled contemptuously. 'Of course you don't – till I tell you. I meant "there's a nice knock-down argument for you!"'
>
> 'But "glory" doesn't mean "a nice knock-down argument",' Alice objected.
>
> 'When I use a word,' Humpty Dumpty said, in rather a scornful tone, 'it means just what I choose it to mean – neither more nor less.'
>
> 'The question is,' said Alice, 'whether you can make words mean so many different things.'
>
> 'The question is,' said Humpty Dumpty, 'which is to be master – that's all.'

This is a bit confusing, so let us see why this is true, recall that

$$R\emptyset = \left\{ xy \;\middle|\; x \in R \text{ and } y \in \emptyset \right\}.$$

But the empty set ($\emptyset$) does not contain any element, and as such, no concatenated string can be created. Namely, its the empty language.

3. $R \cup \emptyset = R$ (just like with any set).
4. $R \cup \epsilon = \epsilon \cup R$.

   This expression can not always be simplified, since ϵ might not be in the language $L(R)$.
5. $\emptyset^* = \{\epsilon\}$, since the empty word is always contain in the language generated by the star operator.
6. $\epsilon^* = \{\epsilon\}$.

5.1 More interesting examples

Suppose $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.

1. $(\Sigma\Sigma)^*$ is the language of all even-length strings.

   (That is, the language associated with the regular expression $(\Sigma\Sigma)^*$ is made out of all the even-length strings over Σ.)
2. $\Sigma(\Sigma\Sigma)^*$ is all odd-length strings.
3. $\mathtt{a}\Sigma^*\mathtt{a} + \mathtt{b}\Sigma^*\mathtt{b} + \mathtt{c}\Sigma^*\mathtt{c}$ is all strings that start and end with the same character.

5.1.1 Regular expression for decimal numbers

Let $D = \{\mathtt{0}, \mathtt{1}, ..., \mathtt{9}\}$, and consider the alphabet $E = D \cup \{-, .\}$. Then decimal numbers have the form

$$(- \cup \epsilon)\, D^*\, (\epsilon \cup .) D^*.$$

But this does not force the number to contain any digits, which is probably wrong. As such, the correct expression is

$$(- \cup \epsilon)(D^+(\epsilon \cup .)D^* \cup D^*(\epsilon \cup .)D^+).$$

Notice that $\mathtt{a}^n$ is **not** a regular expression. Some things written with non-star exponents are regular and some are not. It depends on what conditions you put on n. E.g. $\left\{ \mathtt{a}^{2n} \;\middle|\; n \geq 0 \right\}$ is regular (even length strings of $\mathtt{a}$'s). But $\left\{ \mathtt{a}^n\mathtt{b}^n \;\middle|\; n \geq 0 \right\}$ is not regular.

However, $\mathtt{a}^3$ (or any other fixed power) is regular, as it just a shorthand for $\mathtt{aaa}$. Similarly, if R is a regular expression, then R^3 is regular since its a shorthand for RRR.