Lecture 8

Hidden Alphabets *or* The Horrors of Abstractness

1 Adding a Hidden Alphabet

1.1 Refined Strictly Local Grammars

Definition 8.1 (Run). Let Σ and Q be overt and *hidden* alphabets, respectively. Each $q \in Q$ is called a *state*. Given a string w over Σ, a (Σ, Q)-*run* over w is a total function mapping each node n of w to some $q \in Q$. We also say that q *is assigned to* n, and we represent the set of all (Σ, Q)-runs over w by Q^w. We may omit mention of Σ and Q if they are clear from the context. Overloading our terminology, we also use run to refer to the image of w under a given run, or the string of pairs $p_1 \cdots p_n$, where each p_i consists of the i-th node of w and its image under some fixed Q-run.

Example 8.1 Runs Over a Short String

Suppose that $Q := p, q, r$ and that $w := abbca$. Then the set of Q-runs over w includes, among others, $qpprq$, $pqrpq$, or even $ppppp$, but not $qppqs$ (because $s \notin Q$), qpp (too few symbols), or $pqrpqr$ (too many symbols). If we view runs as strings of pairs, then the three runs and non-runs look as below:

q	p	p	r	q		p	q	r	p	q		p	p	p	p	p
a	b	b	c	a		a	b	b	c	a		a	b	b	c	a
q	p	p	q	s		q	p	p				p	p	p	p	p	r
a	b	b	c	a		a	b	b	c	a		a	b	b	c	a	

Definition 8.2 (Refined Grammar). A refined strictly k-local grammar G is a finite set of k-grams over alphabet $\Sigma \times Q$. Such a grammar generates the language $L(G) := \{w \mid \exists r \in Q^w, k\text{-grams}(r) \subseteq G\}$. A language is refined strictly k-local iff it is generated by a refined strictly k-local grammar. The class of all refined strictly k-local languages is denoted SL_k^R.

Example 8.2 A Refined Grammar for Even Counting

Recall that the language $(aa)^+$, which contains all non-empty strings over a whose length is even, is neither strictly local nor strictly piecewise. However, it can be generated by a refined strictly 2-local grammar with only a handful of bigrams.

$$\frac{eo}{\rtimes a} \quad \frac{oe}{aa} \quad \frac{eo}{aa} \quad \frac{ee}{a \ltimes}$$

This grammar generates *aaaa*, for instance, because there is a run such that each bigram of that run is licensed by the grammar. In contrast, the slightly longer *aaaaa* is not generated because every run contains at least one bigram that is not licensed by the grammar. We indicate this by not assigning a state to the node for which an irreconcilable conflict arises.

e	o	e	o	e	e		e	o	e	o	e	o	
⋊	a	a	a	a	⋉		⋊	a	a	a	a	a	⋉

1.2 Generative Capacity

Due to how refined strictly local grammars are defined, they trivially subsume the strictly local grammars as a special case where the hidden alphabet Q contains only one state. The fact that a refined strictly 2-local grammar can generate (aa^+) shows that the inclusion is proper — refined strictly local grammars are strictly more powerful than strictly local grammars without a hidden alphabet of states. But this result can be even strengthened: ever strictly local language is refined strictly 2-local.

Lemma 8.3. $\mathrm{SL} \subsetneq \mathrm{SL}_2^R$ ⌟

To see this, just keep in mind that in order to determine the well-formedness of a string with respect to a strictly k-local grammar, it suffices to memorize all the k-grams in the string and see if all of them are licensed by the grammar. This was exactly the way our first scanner implementation operated. In a refined strictly local grammar, we can use the states to keep track of all the k-grams, and the only states that can be assigned to the right edge marker ⋉ are those that represent a subset of the strictly k-local grammar.

Proof. We already have $\mathrm{SL}_2^R \not\subseteq \mathrm{SL}$ thanks to the example of $(aa)^+$, so it suffices to show $\mathrm{SL} \subseteq \mathrm{SL}_2^R$. Let $L \in SL_k$ be a language over alphabet Σ, and G a positive strictly k-local grammar such that $L(G) = L$. The grammar G_2^R is the largest subset of $(\Sigma \times Q)^2$ such that

- Q consists of pairs $\langle g, S \rangle$, where g is a k-gram and S a set of k-grams over Σ,
- $\frac{pq}{ab} \in G_2^R$ only if, for $p := \langle g_p, S_p \rangle$ and $q := \langle g_q, S_q \rangle$
  - if $a = \rtimes$, then $p := \langle \rtimes^k, \{\rtimes^{k-1} \cdot b\} \rangle$,
  - $g_q := u \cdot b$, where $u \in \Sigma^{k-1}$ and $g_p := x \cdot u$,

- $S_q := S_p \cup \{g_q\}$,
- $S_q \subseteq G$.

Since S_q must be a subset of G, it follows immediately that $L(G_2^R) \subseteq L(G)$. In the other direction, suppose that some $w \in L(G)$ is not contained in $L(G_2^R)$. Then there is no $r \in Q^w$ for which k-grams$(r) \subseteq G_2^R$. However, G_2^R is a maximal subset and thus contains every $\frac{pq}{ab}$ unless one of the conditions above is violated. But since $w \in L(G)$, k-grams$(w) \subseteq G$, so g_p, g_q and S_q are well-defined, and $S_q \subseteq G$. □

The strategy above can be modified to keep track of subsequences instead of substrings, which entails that every strictly piecewise language is refined strictly 2-local. And because (aa^+ is not strictly piecewise but refined strictly 2-local, we get yet another proper subsumption relation.

Lemma 8.4. $\text{SP} \subsetneq \text{SL}_2^R$ ⌟

At this point it shouldn't come as a surprise that even the strict threshold testable languages are refined strictly 2-local.

Lemma 8.5. $\text{STT} \subsetneq \text{SL}_2^R$ ⌟

Proof. Left as an exercise to the reader. □

As you can see, the addition of a hidden alphabet has made our formalism much more powerful, to the extent where we can't even make any distinctions between local and non-local dependencies: they all belong to SL_2^R. And as we will see next, things are even worse insofar as locality plays no role at all in refined strictly local grammars.

2 The Loss of Locality

2.1 Reduction to 2-Locality

Theorem 8.6. If a language is generated by a refined strictly k-local grammar with hidden alphabet Q ($k \geq 2$) then it is generated by a refined strictly 2-local grammar with hidden alphabet Q'.

The proof for this theorem is fairly cumbersome to read, but the intuition of the construction is very simple: since a refined strictly k-local grammar has only a finite number of k-grams, we can emulate the computations of the whole grammar in our hidden alphabet.

Example 8.3 Emulating a Grammar via a Hidden Alphabet

Let L be the language of all strings over $\{a, b\}$ such that consecutive substrings of bs must have even length and may only occur after at least three as. This language includes strings like *aaa*, *aaabb*, and *aaabbbba*, but not *bbaaa* or *aaab*. This can be easily handled by a refined strictly 4-local grammar where the hidden alphabet keeps track of the length requirement while the presence of three as is enforced directly via

the 4-grams. The initial 4-grams are very simple, since no b can occur at the start of a string:

$$\frac{eeee}{\rtimes\rtimes\rtimes a} \quad \frac{eeee}{\rtimes\rtimes aa} \quad \frac{eeee}{\rtimes aaa}$$

The final 4-grams already have to take quite a bit of variation into account:

$$\frac{eeee}{a\ltimes\ltimes\ltimes} \quad \frac{eeee}{aa\ltimes\ltimes} \quad \frac{eeee}{aaa\ltimes} \quad \frac{eoee}{abb\ltimes}$$
$$\frac{eeee}{b\ltimes\ltimes\ltimes} \quad \frac{eeee}{ba\ltimes\ltimes} \quad \frac{eeee}{baa\ltimes}$$
$$\frac{oeee}{bb\ltimes\ltimes} \quad \frac{oeee}{bba\ltimes}$$
$$\frac{eoee}{bbb\ltimes}$$

The non-initial, non-final 4-grams are also much fewer than one might expect thanks to the restricted distribution of bs:

$$\frac{eeee}{aaaa} \quad \frac{eeeo}{aaab} \quad \frac{eeoe}{aabb} \quad \frac{eoee}{abba} \quad \frac{eoeo}{abbb}$$
$$\frac{eoeo}{bbbb} \quad \frac{oeoe}{bbbb} \quad \frac{eoee}{bbba} \quad \frac{oeee}{bbaa} \quad \frac{eeee}{baaa}$$

Let's quickly verify that this grammar generates the string *aaabb* but not *aaabbabb* or *aaabbaaab*.

e	e	e	e	e	e	o	e	e	e	e
$\rtimes$	$\rtimes$	$\rtimes$	a	a	a	b	b	$\ltimes$	$\ltimes$	$\ltimes$

e	e	e	e	e	e	o	e	e	e				
$\rtimes$	$\rtimes$	$\rtimes$	a	a	a	b	b	a	a	b	b	$\ltimes$	$\ltimes$	$\ltimes$

e	e	e	e	e	e	o	e	e	e	e	o			
$\rtimes$	$\rtimes$	$\rtimes$	a	a	a	b	b	a	a	a	b	$\ltimes$	$\ltimes$	$\ltimes$

We are now going to convert this grammar into a refined strictly 2-local one. The operation is very simple to carry out: every 4-gram is first split into two 3-grams, one of which includes the first three positions, the other one the last three positions. We then use these two 3-grams as the hidden symbols for the 2-gram that consists of the last two positions of the 4-gram. For example, $\frac{eoee}{abba}$ is turned into the bigram below.

$$\frac{\frac{eoe}{abb}\ \frac{oee}{bba}}{b\quad a}$$

The whole grammar consists of the following bigrams (the translation also produces a few refined bigrams for $\ltimes\ltimes$, but those are never useful in a strictly 2-local grammar

and can safely be discarded):

$$\frac{\frac{eee}{\rtimes\rtimes\rtimes}\ \frac{eee}{\rtimes\rtimes a}}{\rtimes\ \ a} \qquad \frac{\frac{eee}{\rtimes\rtimes a}\ \frac{eee}{\rtimes aa}}{a\ \ a} \qquad \frac{\frac{eee}{\rtimes aa}\ \frac{eee}{aaa}}{a\ \ a}$$

$$\frac{\frac{eee}{aaa}\ \frac{eee}{aa\ltimes}}{a\ \ltimes} \qquad \frac{\frac{eoe}{abb}\ \frac{oee}{bb\ltimes}}{b\ \ltimes} \qquad \frac{\frac{eee}{baa}\ \frac{eee}{aa\ltimes}}{a\ \ltimes} \qquad \frac{\frac{oee}{bba}\ \frac{eee}{ba\ltimes}}{a\ \ltimes} \qquad \frac{\frac{eoe}{bbb}\ \frac{oee}{bb\ltimes}}{b\ \ltimes}$$

$$\frac{\frac{eee}{aaa}\ \frac{eee}{aaa}}{a\ \ a} \qquad \frac{\frac{eee}{aaa}\ \frac{eeo}{aab}}{a\ \ b} \qquad \frac{\frac{eeo}{aab}\ \frac{eoe}{abb}}{b\ \ b} \qquad \frac{\frac{eoe}{abb}\ \frac{oee}{bba}}{b\ \ a} \qquad \frac{\frac{eoe}{abb}\ \frac{oeo}{bbb}}{b\ \ b}$$

$$\frac{\frac{eoe}{bbb}\ \frac{oeo}{bbb}}{b\ \ b} \qquad \frac{\frac{oeo}{bbb}\ \frac{eoe}{bbb}}{b\ \ b} \qquad \frac{\frac{eoe}{bbb}\ \frac{oee}{bba}}{b\ \ a} \qquad \frac{\frac{oee}{bba}\ \frac{eee}{baa}}{a\ \ a} \qquad \frac{\frac{eee}{baa}\ \frac{eee}{aaa}}{a\ \ a}$$

When we use this grammar to determine the well-formedness of the previous three example strings, we once again see that only the first one has a run and is thus generated by the grammar.

$$\frac{eee}{\rtimes\rtimes\rtimes}\ \frac{eee}{\rtimes\rtimes a}\ \frac{eee}{\rtimes aa}\ \frac{eee}{aaa}\ \frac{eeo}{aab}\ \frac{eoe}{abb}\ \frac{oee}{bb\ltimes}$$
$$\rtimes\ \ a\ \ a\ \ a\ \ b\ \ b\ \ \ltimes$$

$$\frac{eee}{\rtimes\rtimes\rtimes}\ \frac{eee}{\rtimes\rtimes a}\ \frac{eee}{\rtimes aa}\ \frac{eee}{aaa}\ \frac{eeo}{aab}\ \frac{eoe}{abb}\ \frac{oee}{bba}\ \frac{eee}{baa}$$
$$\rtimes\ \ a\ \ a\ \ a\ \ b\ \ b\ \ a\ \ a\ \ b\ \ b\ \ \ltimes$$

$$\frac{eee}{\rtimes\rtimes\rtimes}\ \frac{eee}{\rtimes\rtimes a}\ \frac{eee}{\rtimes aa}\ \frac{eee}{aaa}\ \frac{eeo}{aab}\ \frac{eoe}{abb}\ \frac{oee}{bba}\ \frac{eee}{baa}\ \frac{eee}{aaa}\ \frac{eeo}{aab}$$
$$\rtimes\ \ a\ \ a\ \ a\ \ b\ \ b\ \ a\ \ a\ \ a\ \ b\ \ \ltimes$$

As this example shows, we never need a search window bigger than 2 because the hidden alphabet can keep track of all the extra information a bigger window would provide. Turning this intuition into a proof is fairly straight-forward, but requires a loot of bookkeeping via indices.

Proof. We show that every strictly k-local grammar G_k with refined alphabet $\langle \Sigma_k, Q_k \rangle$ can be converted into a strictly 2-local grammar G_2 with refined alphabet $\left\langle \Sigma_k, (\Sigma_k \times Q_k)^{k-1} \right\rangle$ such that $L(G_k) = L(G_2)$. Let G_2 be the smallest set of bigrams such that if $g := \frac{q_1 \cdots q_k}{\sigma_1 \cdots \sigma_k}$ is a k-gram of G_k, then G_2 contains $g' := \frac{\phi\rho}{\sigma_{k-1}\sigma_k}$, where $\phi := \frac{q_1 \cdots q_{k-1}}{\sigma_1 \cdots \sigma_{k-1}}$ and $\rho := \frac{q_2 \cdots q_k}{\sigma_2 \cdots \sigma_k} \in Q_k^{k-1}$. We give an inductive proof that $L(G_2) = L(G_k)$.

Suppose $w \in L(G_k)$, and that $g_k := \frac{q_1 \cdots q_k}{\sigma_1 \cdots \sigma_k}$ is the refined k-gram spanning from the m-th to the $(m+k-1)$-th symbol of the k-augmented counterpart $\hat{w}_k$ of w.

For the base case assume $m = 1$. Then the first $k-1$ symbols of g_k are left edge markers: $\sigma_i = \rtimes$, $1 \leq i < k$, while the 2-augmented counterpart $\hat{w}_2$ lacks the first

$k-2$ symbols of $\hat{w}_k$. Given the construction above, G_2 contains the refined bigram

$$g_2 := \frac{\dfrac{q_1 \cdots q_{k-1}}{\sigma_1 \cdots \sigma_{k-1}} \cdot \dfrac{q_2 \cdots q_k}{\sigma_2 \cdots \sigma_k}}{\rtimes \qquad \sigma_k},$$

which can be assigned to the first two positions of $\hat{w}_2$. In the other direction, if g_2 is assigned to the first 2 positions of $\hat{w}_2$, then g_k can be assigned to the first k positions of $\hat{w}_k$. Since $g_2 \in G_2$ iff $g_k \in G_k$, the first 2 positions of $\hat{w}_2$ are well-formed wrt G_2 iff the first k positions of $\hat{w}_k$ are well-formed with respect to G_k.

For arbitrary m, suppose that g_k spans from the m-th to the n-th position of $\hat{w}_k$, where $n = m+k-1$. By our induction hypothesis, some bigram spans the positions in $\hat{w}_2$ that correspond to $n-2$ and $n-1$ of $\hat{w}_k$— namely $n-(k-2)-2 = m-1$ and $n-(k-2)-1 = m$, respectively. Moreover, the second component of this bigram is

$$\frac{\dfrac{q_1 \cdots q_{k-1}}{\sigma_1 \cdots \sigma_{k-1}}}{\sigma_{k-1}}.$$

By our construction, then, there is a $g_2 \in G_2$ that spans from m to $m+1$ and has the shape

$$\frac{\dfrac{q_1 \cdots q_{k-1}}{\sigma_1 \cdots \sigma_{k-1}} \cdot \dfrac{q_2 \cdots q_k}{\sigma_2 \cdots \sigma_k}}{\sigma_{k-1} \qquad \sigma_k}$$

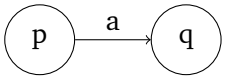
iff $g_k \in G_k$. □

Theorem 8.6 shows that every refined strictly local language is refined strictly 2-local. Rather than the infinite hierarchies of increasing complexity that we saw with the strictly local and strictly piecewise languages, this class is flat, at least with respect to whether a language can be generated by a refined strictly 2-local language. To emphasize this flatness, we don't just drop the locality parameter — which might be mistaken as refering to a more powerful class that is the union of all refined strictly k-local languages — but coin a completely new term.

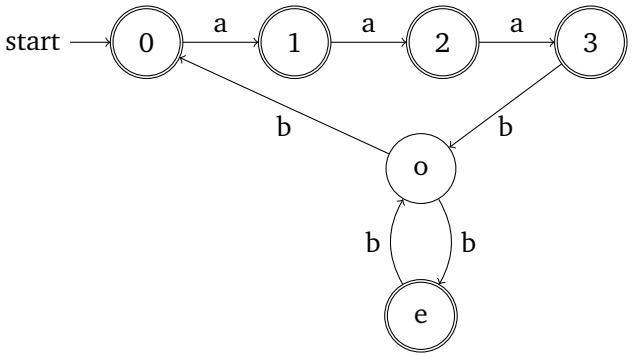
Definition 8.7 (Regular Languages). A language is *regular* iff it is generated by a refined strictly 2-local grammar.

Regular languages are one of the most important classes of string languages in computer science and have been defined in numerous equivalent ways. We do not have time to look at all of them, but most of them are covered in every decent textbook on formal language theory such as Sipser (2005), Kozen (1997), or Hopcroft and Ullman (1979) (listed in increasing order of difficulty).

At least one of these equivalent definitions of regular languages can be inferred rather easily from example 8.3, though. In this example, we saw that parts of a k-gram can be crammed into the hidden alphabet of a bigram. But in principle we could do the same thing to the bigram and copy the overt symbols into the hidden alphabet, too. In this case, we can infer the state of a node purely from the label of that node and the state of the preceding node; the label of the preceding state is no longer needed. Consequently, every bigram $\frac{pq}{ba}$ can be represented by a simple graph.



We can think of this graph as a machine or *automaton* that switches from state p to q if it reads in an a. The whole grammar is an automaton that combines the mini-automata of all the bigrams. For the language in example 8.3, the automaton is actually much easier to understand than the refined grammar we constructed.



The graph uses start to indicate which state(s) may be assigned to the first node, whereas only circled states may be assigned to the final node. The runs for the three test strings from the previous example now look slightly different, but they still lead to the same conclusions.

0	1	2	*o*	*e*
a	*a*	*a*	*b*	*b*

0	1	2	*o*	*e*	0	1		
a	*a*	*a*	*b*	*b*	*a*	*a*	*b*	*b*

0	1	2	*o*	*e*	0	1	2	*o*
a	*a*	*a*	*b*	*b*	*a*	*a*	*a*	*b*

Definition 8.8 (Finite-State Automaton). A *finite state automaton* (FSA) is a quintuple $A := \langle \Sigma, Q, I, F, \Delta \rangle$, where

- Σ is an alphabet,
- Q is a finite set of states,
- $I \subseteq Q$ is the set of *initial* states,
- $F \subseteq Q$ is the set of *final* states,
- $\Delta \subseteq Q \times \Sigma \times Q$ is a finite set of transition rules.

The FSA is *deterministic* iff I is a singleton set and Δ contains no two $\langle p, a, q \rangle$ and $\langle p, a, r \rangle$ with $q \neq r$. Otherwise it is non-deterministic.

---

determinization via powerset construction

3 Properties of Regular Languages

cylindrification
- boolean closure
- relabeling closure
- closure under reversal
- Myhill-Nerode characterization
- closure under finite-state transductions