Formal Language Theory: Origins and New Directions

Class Notes for CS682: "Formal Language Theory II"

Instructor: Dr. Kenneth W. Regan

Students and Scribes, Fall 1993:

Daniel F. Boyd
Deborah T. Burhans
Kannan Govindarajan
D. Sivakumar
Michael J. Smolowitz

December 1993

Department of Computer Science
State University of New York at Buffalo
226 Bell Hall, Buffalo, NY 14260 USA

Authors' e-mail addresses: {regan,boyd,burhans,govin-k,sivak-d,ms}@cs.buffalo.edu

1. Preface

These notes were compiled by the instructor and students during the teaching of the course "CS682: Formal Language Theory II" in the Computer Science Department of the State University of New York at Buffalo during the Fall Semester, 1993. The system of taking and preparing class notes was modeled on the treatment of 6xx courses at MIT. The students were assigned responsibility for compiling the notes in LaTeX. This began in the second week of term after a special session describing the LaTeX macros used for the notes and giving students some practice.

In this compilation, I have removed the MIT macros for numbering by lecture, and have grouped the material by topics rather than by weeks. After each section heading I indicate which students scribed that section and the time period covered. As of this writing (12/14/93), I have not yet made intended revisions and additions to the notes as they were covered in class.

The main motivation for compiling these notes is to provide a single, *good* source for the special topics that formed the second half of the course, namely the connections among Smullyan's *rudimentary relations*, other systems of first-order logic, and (to use a term of Harvey Rose [Ros84]) "small classes" of languages. Since 1981, the year of both the article by S. Greibach titled "Formal Languages: Origins and Directions" [Gre81] and the first publication of the Furst-Saxe-Sipser theorem [FSS81, FSS84], there have been many notable advances in the part of complexity theory having to do with small classes. (Hence my modification to Greibach's title.) Parts of the foundation for this work were laid in papers from the 1960s and 1970s, and other parts, cast by Barrington, Immerman, and Straubing [Imm87, Bar89, BIS90] from molds cut by earlier researchers in logic and circuit complexity, are newer. However, the sources-of-record for these developments are journal papers, from which I selected the above-cited and [Jon69, Jon75, Wra78] (plus the book [MP71] and some nice expository papers such as [Lin92]). Taken collectively, these papers do not (and being separate in time and space, would not be expected to) provide a cohesive picture of the subject. Also, IMHO, they are on the whole not particularly well written. They provided an all-too-typical challenge for the students of learning how to read real journal papers, warts and all. The purpose of the course was to draw connections among these papers, and to stimulate deeper insights as a basis for further research. One paper which was very valuable in this last service was a well-written note by Allender and Gore [AG91b] connecting Jones' work and [BIS90]. I used this in the final part of the course, but for variety, carried out the cycle of proofs of equivalence in the opposite direction.

A second service of these notes is that several of the above-cited papers refer to earlier work for key points in their proofs, and this "citation trail" goes all the way back to several Ph.D. theses in the 1960s which are not so readily obtainable. One such example is the trick for writing a language in NLIN as an intersection of CFLs, which goes from [Wra78] through [BG70] and [GH69] to a citation of a thesis. Another hard-to-track lemma is the one for converting an alternating TM to an ATM which queries its input at most once along any path, *while increasing the number of alternations by at most 1* (most references omit the last clause). We also found an anticipation of the binary-counting lemma of [BIS90] in a lemma of Jones [Jon69]; this is even earlier than a similar lemma of Lipton [Lip78], who shares credit with Pippenger. Also in these notes (as of 12/14/93, a problem on the take-home final) is an "elementary" proof that every first-order definable language is star-free regular; the proof is different from those in [Mey69, MP71, Lad77, Tho82, PP86, Bar89], and the idea of writing numbers in unary notation appears to be entirely new. It is natural for established researchers who are familiar with these matters to brush them off, but they can be hard obstacles for students if not done *right*. Put another way, we more-advanced people should give our students every opportunity to spot something in these "old" lemmas that we ourselves have always missed!

The plan of this course was to stay with the established text by Hopcroft and Ullman [HU79] in the first half, concentrating on chapters 10-12, before making the leap to less-structured sources. To my own chagrin, I found that the material on LR(k) parsing in [HU79] is not in as good a shape as I expected it to be, and several other well-known texts were not much help in my initial flailing, until I figured out which concepts were primary and which could be deferred. Hopcroft and Ullman do the LR(0) case and sketch the LR(1) case. From what I found, it is *better* to begin with the general LR(k) case—when one has good notation, this is really not any more overhead than for LR(1). Indeed, starting with LR(k) makes the requirements clearer. The case $k = 0$ should in retrospect be regarded as degenerate. The most instructive way to appreciate the difference between LR(1) and LR(0) is to do the construction from a DPDA to an LR(1) grammar G, and then challenge the students to find the one point where G fails to be LR(0). Then the result about $L \cdot \$$ having an LR(0) grammar, which is done first in [HU79] and other texts, can be understood as a patch on the previous proof. Also in my opinion, the concept of a *handle* in a derivation is unnecessary before presenting the principle and gets in the way; it is best introduced later when one discusses ways of *improving* the efficiency of implementations of the principle. Thus these notes also unintendedly provide a streamlined new treatment of the theory behind LR(k) parsing.

These notes are preliminary, and will be refined and augmented later on. They are also incomplete. The unit on LR(k) parsing was intended to serve a one-week treatment of DPDAs with two-way input tape (presenting Cook's linear-time simulation on a RAM by the method of "building a table"), with the speculative purpose of basing a generalization of LR(k) parsing on them. I had to cut this and advance to the Chomsky-Schützenberger Representation Theorem as a preface to Wrathall's paper, but part of this is the topic of D. Boyd's short essay, which will be integrated with these notes at a later date. I have not yet established just what sort of generalization will work; K. Govindarajan's essay may provide some useful guidelines. The three other "2-3 page research essays" and the problem sets in the course will be integrated later as well. There were also two weeks of intended material on classes within NC^1 that were not presented by the end of term; there is only a one-page sketch in the last section as of now. Comments on these notes are welcome and invited. I am grateful for advice and help in certain particulars from Professors David Mix Barrington, John Case, Ronald Book, Oscar Ibarra, Michael Loui, and Eric Allender. Barrington provided me his current skeletal outline of the book on this subject that he has just begun; these notes are mostly independent of it.

2. Review of Formal Language Theory and Preview of Course

[First week of class, lectures 01–03, scribed by K.W. Regan; some notes were added later.]

The first week of lectures is a review of the basic *types* of objects seen in formal language theory, followed by an important example of "generalized" homomorphisms and how they can simulate Turing machines.

The basic types are organized around what is sometimes called the "ladder of abstraction." Going one rung up the latter means talking about *sets of* objects at the previous level.

Level 0. `Char`acters over some alphabet Σ. Usually Σ is finite. Most often $\Sigma = \{0,1\}$ or $\Sigma = \{a,b\}$.

Level 1. `String` = ARRAY $[0 \ldots n-1]$ OF `Char`. Consider the following correspondence:

Strings	Σ^*	=	λ 0, 1, 00, 01, 10, ...
Numbers	$\mathbf{N}$	=	0, 1, 2, 3, 4, 5, ...

In *dyadic notation*, the number $Val(x)$ corresponding to a binary string x of length n equals $\sum_{i=0}^{n-1}(x_i + 1) \cdot 2^i$. This is often superior to standard binary notation—it eliminates the problem that "leading zeroes" give multiple names for the same number.

There is another way to look at strings: rather than ordered arrays of characters, they can be unordered sets of "events." Define `Event` to be the type (`Char`,*position*), where *position* is a natural number. Then for instance, the string 10110 is given by $\{(1,3),(1,2),(1,0),(0,1),(0,4)\}$, with the order immaterial. This notation also allows for "partial strings"; i.e., where the set does not fill all positions up to the highest one in the set. I'm mostly saying this because the type definition `String` = SET OF `Event` has the same form as what follows. (A second reason: the notion of a "non-consecutive substring" such as 010 in positions 1,2,4 in 10110 may become useful later.)

As an "aside," let me mention several extensions of the notion of a string:

- "Compound strings" of the form (x, y) where x and y are strings. These have been studied. For example, one representation of the important Pattern Matching language is $L_{\text{pat}} = \{(p,t) : p,t \in \Sigma^* \ \wedge \ (\exists u, v \in \Sigma^*) t = upv\}$. Then $((\epsilon,0)+(\epsilon,1))^*((0,0)+(1,1))^*((\epsilon,0)+(\epsilon,1))^*$ is a "compound regular expression" which defines L_{pat}. L_{pat} is accepted by a two-input-tape NFA, but not by any two-tape DFA (even stronger, not by any DFA with two heads on a single tape containing $p\#t$). Thus the "NFA=DFA" part of Kleene's Theorem fails in the case of compound strings.

- The second typing for strings above can also be thought of as the type *position* $\rightarrow$ *Char*. That is, a string is a function $c(t)$ which gives the character at each position t. If we think of t as "time," we can generalize this to a notion of "continuously-varying strings." This is not as far-fetched as it sounds, and without having any specific knowledge, I suspect that this notion is implicit if not explicit in much work on real-time process control.

- One can think of *two-dimensional strings* as 2D arrays of characters. Two-dimensional pattern-matching is a field with huge practical importance. Dr. Alan Huang of Bell Labs, who is one of the foremost researchers in optical computing today, has developed a theory of computing via optical 2D pattern matching which he calls *Symbolic Substitution*. I know of no general work on extending the theory of 1D strings and formal languages to 2D strings. It might be interesting in particular to investigate which properties are "really 2D," in the sense that they are not simply lifted from properties of 1D *rasterizations* of 2D strings.

OK, I digressed, but let me ask: In how many fields can one be at the frontier of current research 15 minutes into the first lecture of term?

Level 2. `Language` = SET OF `Strings`. Recall that there are $2^{\aleph_0}$-many languages over a given finite alphabet, only $\aleph_0$ of which are r.e.

Level 3. `Class` = SET OF `Languages`. Formal language theorists like to talk in terms of classes, rather than any of the more-concrete objects lower down on the ladder of abstraction. (:-))

Level 4. `Hierarchy` = SET OF `Classes`. Two examples are:

1. *The Chomsky Hierarchy* = REG, CFL, CSL, RE.
2. *The Arithmetical Hierarchy* = REC, RE, $\mathrm{RE}^{\mathrm{RE}}$, $\mathrm{RE}^{\mathrm{RE}^{\mathrm{RE}}}, \ldots$, equivalently written as $\Sigma_0^0, \Sigma_1^0, \Sigma_2^0, \Sigma_3^0, \ldots$.

From CS596, recall that the "stacks of RE" notation referred to definitions of classes via *oracle Turing machines*, while the Σ_k^0 notation referred to classes of languages defined by logical formulas of arithmetic—for Σ_k^0, the formulas are required to be in *prenex form* with at most k blocks of like quantifiers, the first (outermost) block being existential. The statement that the two ways of defining the hierarchy are equivalent is sometimes called the "Weak Hierarchy Theorem" of Kleene, while Kleene's "Strong Hierarchy Theorem" is that all these classes are distinct.

In complexity theory we've talked about the *Polynomial Hierarchy*, with notation $\mathrm{P} = \Sigma_0^p$, $\mathrm{NP} = \Sigma_1^p$, $\mathrm{NP}^{\mathrm{NP}} = \Sigma_2^p$, etc. Here again, the Σ_k^p notation requires to definitions by quantifiers, this time by *polynomial-length bounded*) quantifiers, and there is a "weak hierarchy theorem" stating equivalence to "oracle" definitions, this time involving polynomial-time bounded (non-)deterministic oracle TMs. But there is no known analogue of the strong-hierarchy theorem; whether the polynomial hierarchy "collapses" or is infinite includes the P vs. NP question. In this course we will study two other hierarchies. The *Linear-Time Hierarchy*, for which I'll use the notation Σ_k^{lin}, was apparently first defined and studied as such by Celia Wrathall, in her Ph.D. thesis [Wra75] and journal paper [Wra78]. Let LINH = $\cup_{k=1}^{\infty}$. In addition to proving the equivalence of several ways of defining Σ_k^{lin}, including one involving *alternating* linear-time Turing machines, Wrathall proved that LINH equals the class RUD of languages definable in Raymond Smullyan's system of *rudimentary* relations. Neil Jones [Jon69] had earlier observed that RUD contains all CFLs and many non-CFLs besides, but it was Wrathall who really pinpointed the complexity-theoretic power of these relations. Again no "strong hierarchy theorem" is known here, except that the class Σ_1^{lin} = NLIN (viz., nondeterministic linear time on Turing machines) is known to properly contain DLIN (deterministic linear time), by the theorem of Paul, Pippenger, Szemerédi, and Trotter [PPST83]. However, the NLIN $\neq$ DLIN proof turns on some very gritty limitations of the 1-dimensional tapes of TMs, and is not known to carry over when TMs can have 2-D tapes. That is, whereas NLIN remains the same under this and several other variations of the TM model (so to speak, it is a quite *robust* complexity concept), DLIN seems sensitive to picky details, and I'm not sure the TM version deserves the name "Σ_0^{lin}." The other hierarchy is at a *much lower* level of complexity, namely that of *alternating* Turing machines which do (and thanks to a special "random-access" input mechanism, can) run in *logarithmic* time. Here the notation will be $\Sigma_k^{\log}$ and LOGH, and the final phase of the course will concern many equivalent characterizations of LOGH: by systems of first-order logic, by "log-bounded rudimentary relations" [Jon75, AG91b], and by constant-depth unbounded fan-in circuits. (At this point I am not aware of sharp characterizations of the individual $\Sigma_k^{\log}$ levels; in fact, there seem to be nagging differences in the alternating log-time TM model itself, especially in the definition of DLOGTIME.)

Level 5. Well, this would be: sets of hierarchies. I've never seen anyone really draw attention to this as a *concept*. Despite how things may seem from the above, I don't want you to do so either.

Instead, I will emphasize other kinds of formal objects "closer to Earth" and on a par with strings and languages. First consider

Predicates: type `String` $\rightarrow$ `Bool`. Given a predicate R with one free variable $\hat{x}$, define $L_R := \{\, x \in \Sigma^* : R(x) \text{ holds} \,\}$. Sometimes we'll just call this language R.

[Note: I won't always annotate the distinction between the formal name of a variable $\hat{x}$ and substituting a particular value x for $\hat{x}$; you'll be expected to get used to my writing just x in either sense. However, I *will* try to preserve the sense of x standing for an object of type `String`, and in particular, for "the main input string." In many references, "x" is used for variables in other contexts, most annoyingly, for *positions* in the input string.]

Characteristic functions. Given a language L, define $\chi_L(x) = 1$ if $x \in L$, $= 0$ otherwise. If we identify $\{\,0,1\,\}$ with the type `Bool`, then this has the same type as a predicate with one free variable. However, usually we'll think of predicates as finite *syntactic* objects that we can encode as strings. I'll try to use the letters $P, Q, R, S \ldots$ for predicates and $L, A, B, C, D \ldots$ for languages. We'll be free to write e.g. $A(x) = B(x)$ interchangeably with $\chi_A(x) = \chi_B(x)$ or $x \in A \iff x \in B$, and $P(x)$ can be identified with $L_P(x)$ or $\chi L_P(x)$. Here's a subtler convention which seems to be taking root in the field. If I write $P(x) \longleftrightarrow Q(x)$, this carries the connotation that I am most interested in the *formal object* $P \longleftrightarrow Q$, while double arrows $\Longrightarrow$, $\iff$ are used when only the *truth* of a particular statement is concerned. (A more-dubious habit is writing "$P(x)$" to mean "P is a predicate with one free variable x"; this is like writing "$f(x)$" to refer to a function.)

String relations of arity $k \geq 2$: Suppose P is a predicate with two free variables; i.e. $P : \Sigma^* \times \Sigma^* \rightarrow$ `Bool`. Then we let ρ_P stand for the *relation*

$$\rho_P = \{\, (x, y) : P(x, y) \,\}.$$

Using some sort of injective *tupling function* $\langle\cdot,\cdot\rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$, we could represent P by the language $\theta_P = \{ \langle x, y\rangle : P(x, y) \}$ (the "θ" is Wrathall's notation). However, one of the distinctive features of my approach will be the desire to treat relations as entities in their own right. When we get to the topic of the so-called *rational relations*, we'll see several important differences from regular languages, whose appreciation would be diminished if we always "flattened" relations to languages. Here's a principal motivation: A predicate $P(x)$, with the one free variable x, often has other variables $y, z, \ldots$ which were quantified "at earlier stages in the building of P." Logic has a natural inductive structure on which to prove results, or to locate the precise point at which some property is lost, but the most natural paths through that structure to a desired predicate $P(x)$ generally all go through predicates of higher arity. What I'm hoping for is that working directly with the *relations* defined by these intermediate predicates will reveal secrets and structures that flattening everything to languages might hide. In combinatorics, the simple trick of interchanging the order of variables in a summation is often what turns the intuitive corner toward a new result. My analogy here is the idea of transposing a k-ary relation of strings of length (say) n into an n-ary relation of strings of length k, or alternatively into a language of single strings over a huge alphabet of k-tuples.

Two simple tupling functions deserve mention here. Let $\Sigma = \{ 0, 1 \}$, $\Gamma = \{ 0, 1, \# \}$. Let $h : \Gamma \to \Sigma^2$ be given by $h(0) = 00$, $h(1) = 11$, $h(\#) = 01$. Then h determines a unique *homomorphism* $h_* : \Gamma^* \to \Sigma^*$ via the inductive definition:

- *Basis.* $h_*(\epsilon) = \epsilon$.
- *Induction.* For all $c \in \Gamma$ and $w \in \Gamma^*$, $h_*(cw) = h(c) \cdot h_*(w)$.

It is a little bit more precise to justify the induction step here by noting that: given $x \in Gamma^*$ such that $x \neq \epsilon$, there is a unique way to write $x = cw$ such that $c \in \Gamma$ and $w \in \Gamma^*$; then we may define $h_*(x) = h(c) \cdot h_*(w)$. As in CS596, I will eventually drop the notational distinction between the finite function h and its extension h_*. Now define for all sequences $(x_1, x_2, \ldots x_k)$ of strings over Σ,

$$\langle (x_1, x_2, \ldots x_k\rangle = h(x_1 \# x_2 \# \ldots \# x_k).$$

This function is 1-1, and is "easily" computed. We associate to it the *projection functions* π_j, for $1 \leq j \leq k$, defined for all $z \in \Sigma^*$ by $\pi_j(z) = x_j$ if there exist $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots x_k \in \Sigma^*$ such that $\langle x_1, x_2, \ldots, x_k\rangle = z$, *undefined* otherwise. Note that since $\langle\ldots\rangle$ is not onto, $\pi_j(z)$ may be undefined. If one cares about this, say for $k = 2$, one needs the notion of a *pairing function*, namely a *bijection* from $\Sigma^* \times \Sigma^*$ onto Σ^* (or from $\mathbf{N} \times \mathbf{N}$ onto $\mathbf{N}$). In this course, I have not found a result where this matters; though for the record, I seem to have been the first person to construct a pairing function which is computable by a finite automaton (which implies that its graph is a 3-ary rational relation). Heading in the other direction, a useful feature of this tupling function is that it works unambiguously for any k; in other words, it is a full-fledged *sequence encoding function*. Logicians working in terms of the arithmetical operations $+$ and $\times$ found themselves needing to resort to exponentiation to define sequence-encoding functions, the most famous of which is called *Gödel's β-function*, namely $\beta(x_1, \ldots, x_k) = 2^{x_1}3^{x_2}5^{x_3}\ldots$, using the first k primes. [1] Smullyan's thesis [Smu61] was hailed for the simplifications that result from considering *concatenation* of strings as a basic operation, and in retrospect, for giving logic a more "computer-sciencey" look.

[1] I still need to check my memory of this.

The other tupling function, attributed to UB's John Myhill by Wrathall, pads all of $x_1, \ldots, x_k$ out to the same length by trailing "dummy characters" @, and then *shuffles* their bits. I'll use the notation $[x_1, \ldots x_k]$ when the distinction between this "parallel" tupling function and the above "serial" function matters.

2.1. Generalized Homomorphisms

Fix numbers $d, e > 0$. Given a finite function $h : \Sigma^d \to \Gamma^e$, consider the function $h_* : \Sigma^* \to \Gamma^*$ defined inductively by:

- *Basis.* For all $x \in \Sigma^*$ such that $|x| < d$, $h_*(x) := \epsilon$.
- *Induction.* For all $x \in \Sigma^*$ such that $|x| \geq d$, there is a unique way to write $x = vw$ such that $|v| = d$, and then $h_*(x) := h(v)h_*(w)$.

Then call h_* a *generalized homomorphism with fixed pitch d:e*, or for short, a *d:e gh*. Note that e/d need not be a fraction in lowest terms. A d:e gh can be identified with an ordinary homomorphism from the "compound alphabet" (Σ^d) to (Σ^e), except for the way the basis truncates strings whose lengths are not an integral multiple of d.

For example, the function $a(00) = 0$, $a(01) = 0$, $a(10) = 0$, $a(11) = 1$ defines a 2:1 gh which takes the logical AND of successive pairs of bits, and "vectorized OR" is likewise a 2:1 gh. Negation, i.e. flipping bits, is a 1:1 gh. The concept of a ghincludes any ordinary homomorphism $h : \Sigma^* \to \Gamma^*$ such that the values $h(c)$ over $c \in \Sigma$ all have the same length (I've seen this called a "balanced homomorphism"). However, it does not include homomorphisms with varying lengths, in particular *erasing homomorphisms* such as $e(1) = 1$, $e(0) = \epsilon$. Moreover, because of the way truncation is handled, a d:e gh h may lack the characteristic property that for all $x, y \in \Sigma^*$, $h(xy) = h(y)h(y)$. The *intent* of the definition is that each output symbol depends only on finitely-many input symbols in pre-set locations. More important, if one draws lines from each output symbol to the input symbols it depends on, the resulting diagram is a string of *disjoint* "blocks." [At this point I drew diagrams of the above examples and some others.] The other principal motivation of the definition is the following simulation of a Turing machine.

Theorem 2.1. *For every single-tape Turing machine M, there is a composition $g_2 \circ g_1 \circ g_0$ of three gh's such that for every ID I of M, $g_2 \circ g_1 \circ g_0(I) = J$ such that $I \vdash_M J$. That is, the gh's compute the next-ID function of M.*

Proof Sketch. Recall the coding of IDs using the *ID alphabet* $\Gamma_I = \Gamma \cup (Q \times \Gamma)$ of M. The *idea* is that g_0 updates cells $0, 3, 6, 9, \ldots$ of the tape, g_1 updates cells $1, 4, 7, 10, \ldots$, and g_2 updates cells $2, 5, 8, 11, \ldots$ and finishes the job. Each gh is 3:3 (not 1:1). This works because the status of the cell in position p at a given time t depends only on the status of cells $p - 1$, p, and $p + 1$ at time $t - 1$.

[Here I drew a diagram showing the dependencies. The class said later it looked like three tiers of "Marching Ms." When the idea was set, I admitted there was a "tiny fib" in the statement of the theorem, and invited discussion. The "fib" is that since g_1 begins its work in cell 1, it is not really *composed* with the output of g_0. Put another way, what I drew might be called a "cascaded composition," but it is not a composition in the formal (sequential) sense. The class was challenged on the first problem set to devise a definition of "generalized homomorphism" which makes the theorem valid, and to decide certain properties of it, such as whether the class of functions so defined is closed under composition. The exercise was also intended as one in "research taste." Now t term's end, *I'm* still not sure what should be the best formal definition, though I lean toward the following: g should have the form $\phi_1 \cdot h \cdot \phi_2$ where for some numbers c, d, e with $c < d$, ϕ_1 maps Σ^c to Γ^*, h is a d:e gh, and ϕ_2 is a function of the last up-to-$d-1$ symbols left over after h is done. Probably one should also stipulate that the length of the output of ϕ_1 and ϕ_2 depends only on the length of the input. Note that this definition does not include erasing homomorphisms, so it's not really a *generalization* of homomorphisms. Good names for it are still welcome!]

3. The Chomsky Hierarchy

[Lectures 4–5, scribed by Debra Burhans and Kannan Govindarajan.]

Definition 3.1. A *Grammar* is a 4-tuple (V, T, P, S) where

1. V is a non-empty set of *variable* symbols.
2. T is a non-empty set of *terminal* symbols.
3. P is a set of *productions* of the form $(V \cup T)^+ \to (V \cup T)^*$.
4. $S \in V$ is the *start symbol.*
5. $V \cap T = \phi$.

Table 1 summarises the 4-level Chomsky hierarchy. The linear grammars are strictly more powerful than type 3 grammars, though they are not as powerful as type 2 grammars. Examples of simple linear grammars which are not regular are:

- $A \to 0B \mid \epsilon$. $B \to A1$.
  This grammar accepts the language $\{0^n 1^n | n \geq 0\}$, which is not regular.
- $A \to 0A0 \mid 1A1 \mid 0 \mid 1 \mid \epsilon$.
  This grammar generates palindromic strings over the alphabet { 0,1 }, which is a known non-deterministic CFL, and *not* a DCFL.

[1]By special dispensation, the length decreasing production $S_{00} \to \epsilon \mid S$ is permitted in case type 1 languages contain the empty string.

A Grammar is	If every production has the form	Equivalent machine class
Type 0 (RE)	$(V \cup T)^+ \to (V \cup T)^*$	TM
The Normal Form is	$V \to T$ $VV \to VV$ $V \to VV$ $V \to \epsilon$	
Type 1 (CSL)	$V \to T$ or $\alpha \to \beta$ where $\alpha, \beta \in V^+$ [1] and $\mid \alpha \mid \leq \mid \beta \mid$	NLBA
The Normal Form is	$V \to T$ $V \to VV$ $VV \to VV$	
Type 2 (CFL)	$V \to (T \cup V)^*$	NPDA
Chomsky Normal Form	$V \to VV$ $V \to T$	
Greibach Normal Form	$V \to TV^*$	Real-time NPDA
Type 3 (REG)		FA
Left Linear	$V \to VT^*$ $V \to T^*$	
Right Linear	$V \to T^*V$ $V \to T^*$	
Strongly Regular	$V \to VT$ $V \to T$	
Strongly Regular	$V \to TV$ $V \to T$	
Linear	$V \to T^*VT^*$ $V \to T*$	
Simple Linear	$V \to TV$ $V \to VT$ $V \to T$	

Table 1: The Chomsky Hierarchy

4. Known Results and Open Problems

4.1. Determinism and Non-determinism

1. Every language which is accepted by an NTM is accepted by a DTM, so for any type 0 language we need not specify whether the TM that accepts the language is deterministic or non-deterministic.
2. Every language which is accepted by an NFA is accepted by a DFA, so for any type 3 language we need not specify whether the FA that accepts the language is deterministic or non-deterministic.
3. It is known that there exist languages which are accepted by an NPDA which are not accepted by any DPDA, so the correct machine equivalent for type 2 languages is the NPDA.
4. It is not yet known whether NLBA is equivalent to DLBA. The best known result is Savitch's theorem which shows that NLBA $\subseteq$ DSPACE(n^2).

1. Every 2HDFA language is in $NC^1 \iff$ LOGSPACE= NC^1.
2. $\{w\#w \mid w \in \{0,1\}^*\}$ and $\{p\#t \mid$ p is a substring of t$\}$ are not kHDFA languages for any k.

4.2. Machine Definitions

Turing Machine

The elements of the generalised Turing Machine are:

1. The input tape alphabet Σ.
2. The work tape alphabet $\Gamma = \Sigma \cup \{B\}$.
3. A read-only input tape. Input tapes can be real-time (only right moves allowed) or one-way (no left moves allowed).
4. A read-write work tape. A *WORM* work tape is a Write Once Read Many tape, and a *monotone* work tape can write a 1 over a 0, but not vice versa. Work tapes always have one two-way head. A *stack-restricted* work tape is one in which in any left move by the head the B character must be written (POP).

One can now define the usual kinds of automata in terms of this TM model.

1. A DFA is a DTM which consists of one real-time input tape and no work tape.
2. A DPDA is a DTM which has a one-way input tape, and has a stack-restricted work-tape.

There are other generalisations (or restrictions) of the *stack-restricted* DTM depending on the kinds of operations that the DTM is allowed (or not allowed) to perform on the stack. Some of these are

1. A stack automaton can operate in two modes, a *pushdown* mode on top of the stack or a *reading* mode in the interior of the stack, where it can read the symbols on the stack but cannot write.
2. A *non-erasing* stack automaton, can push symbols but not pop symbols from the top of the stack.
3. A *checking* automaton is a *non-erasing* automaton which can push symbols on top of its stack as long as it is in the *pushdown* mode, but once it enters the *reading* mode, it cannot return to the *pushdown* mode.

5. Deterministic Pushdown Automata

[Lectures 5–6, scribed by Burhans and Govindarajan.]

Definition 5.1. A DPDA M is defined[2] as the 6-tuple (Q, Σ, Γ, δ, B, s, F) where:

1. Q is a finite set of states of the DPDA.
2. Σ is the finite "terminal" alphabet.
3. Γ is the finite "stack" alphabet.
4. δ is the transition function of the DPDA. $\delta : (Q \times \Sigma \times \Gamma) \longrightarrow (\Gamma^* \times \{R, S\} \times Q)$
5. s $\in$ Q, is the start state of the DPDA.
6. F $\subseteq$ Q, is the set of final states of M.

This definition allows δ to contain tuples of the form (p,a,A;X,D,q) where p,q $\in$ Q, a $\in \Sigma$, A $\in \Gamma$, X = $X_1X_2\ldots X_n \in \Gamma^*$ and D $\in$ {R,S}. X_1 is the new top of stack character. The kinds of transitions that a *stack-restricted* DTM might perform to simulate the above move of the DPDA would be

(n+1). pop A. If X = ϵ, go to step 0.

n. push X_n.

(n-1). push X_{n-1}.

$\vdots$

1. push X_1.

0. move input head in direction D.

[2] This definition is cosmetically different from the definition in the text [HU79]. We have left out the bottom of stack marker.

-1. go to state q.

An ϵ-move in our DPDA definition is a tuple of the form (p,a,A;X,S,q) that is an ϵ-move is characterised just by the input head not moving. This is more general than the text definition of an ϵ-move because the text puts in an ϵ instead of the 'a' in (p,a,A;X,S,q) and enforces another restriction that there should be no other "reading" tuple for p and A in the transition function. Our definition does allow tuples of the form (p,b,A;Y,R,r) for some other character b in the input alphabet, in the transition function.

Normal Forms and Acceptance conditions of DPDAs We can assume that the DPDA M, in its first transition, writes a **#** as a bottom of stack marker. M, then never writes another **#**, nor does it pop the **#** except possibly at the end.

There are two commonly accepted acceptance conditions for DPDAs. These are:

1. The "Freeze" condition. The DPDA after undergoing a transition (p,b,A;Y,R,r) where b is the last character of the input and r is a final state, just "freezes" and halts.

2. The "Musical Chairs" condition. After exhausting its input, the DPDA is allowed one transition (q,B,A;ϵ,S,r) and it accepts if and only if r is an accepting state. This accepting condition similar to the first if there were an explicit end-of-input marker $ at the end of the input string.

5.1. Normal Forms of DPDAs

Theorem 5.1. *For every* DPDA *M (under either acceptance condition), there exists a* DPDA *M', which is total and such that L(M) = L(M').*

Totality means that M' reads all of its input and halts on every input.

Kripa's question: Why should the DPDA be made to read all of its input for it to be total? Why isnt halting on all inputs enough irrespective of whether all the input has been read or not?

Proof. Suppose we are given the DPDA M = (Q, $\Sigma, \Gamma, \delta, \wedge$[3], s, F)

We first arrange for the transition function δ : Q $\times \Sigma \times \Gamma \rightarrow \Gamma^* \times \{R, S\} \times$ Q to be total by adding a dead state d in which M first pops the stack down to the $\wedge$, finishes reading the input before halting. So if the machine M halts on its input, it is clear that the DPDA M' constructed by altering the transition function will also halt and will read its input in its entirity. Also M' accepts an input X if and only if M itself does.

The DPDA M can diverge on some input if one of two things happen:

1. M loops among a set of states without progressing on the input. For example if δ has tuples like (q,0,A;B,S,r) and (r,0,B;A,S,q), M would loop among states q and r if in state q with A on top of stack it comes across a 0 in the input.

2. M could also diverge by pushing an indefinite number of symbols on the stack.

[3]When we use the $\wedge$ as an explicit bottom marker, we assume that the DPDAs dont have to explicitly push down a bottom of stack etc. In the Turing Machine formalism, the blank symbol B, is treated as the bottom of stack marker.

We now introduce a few informal notions:

1. If the machine does a "pop", we regard that as making progress.

2. A semi-ID is a triple of the form (state, input character, top of stack).

If one looks at the semi-IDs that the machine goes through while performing some computation, we immediately see that if the machine does not diverge then we can repeat semi-IDs as long as the stack is "shrinking" between consecutive appearances of the semi-ID. In any segment of a computation by the machine M, in which a stack symbol A is never popped, the computation is unaffected by symbols which are under the symbol A. We can use this fact to find semi-IDs (p,c,A) in which A is never popped, and the input head is also not moved. If we succeed in finding in such semi-IDs, then we know that M can diverge on some input.

For each c $\in \Sigma$, define the directed graph G_c by:

$$\begin{aligned} V_c &= Q \times \Gamma. \\ E_c &= \{(p, A) \rightarrow (q, B) : \delta(p, c, A) = (\overline{X}, \overline{S}, q)\} \end{aligned}$$

where $\overline{X}$ begins with the new top of stack symbol B. Note that this requires $\overline{X} \neq \epsilon$. If $\overline{X} = \epsilon$ then this is a "pop" move and M has made "progress".

Proposition 5.2. *$\forall$ X, M(X) diverges iff the computation semi-ID (p,c,A) s.t. in G_c, node (p,A) is on a directed cycle.*

It turns out that this definition of the graph G_c isnt powerful enough as it does not take into account "pop" moves, so we define the graph differently as follows:

$$\begin{aligned} V_c &= Q \times \Gamma. \\ E_c &= \{(p, A) \rightarrow (q, B) : \delta(p, c, A) = (\overline{X}, \overline{S}, q)\} \end{aligned}$$

To construct the edges in G_c, we start with some node (p,A) and we run M starting it off in semi-ID (p,c,A) and if (p,c,A) $\vdash$ (q,c,$\overset{B}{A}$), then add edge (p,A) $\rightarrow$ (q,B) to E_c, until one of the following happens:

1. M moves the input head right.

2. M reaches state p again scanning A again.

3. M repeats some other (q,B) in same manner as 2.

4. M pops the A it started with.

While none of 1-4 happens, add edges from (p,A) to every semi-ID (q,B) encountered on the way.

Claim 5.3. *At least one of 1-4 happens after atmost* $\|Q\| \times \|\Gamma\|$ *steps.*

One can now construct all the edges in the graph by doing the above for every node in the graph G_c.

Proposition 5.4 (Correct Proposition). *$\forall$ X, M(X) diverges iff, M(X) reaches a semi-ID (p,c,A), s.t. (p,A) is on a directed cycle in the graph G_c.*

Proof. [$\Leftarrow$] If M on input X reaches a semi-ID which is on a directed cycle then it obviously does not halt, and hence it diverges. ■

Proof. [$\Rightarrow$] Suppose M diverges, then $\exists$ c on the input tape such that M does not read any input symbol after c. Let p be the state and A be the symbol on top of the stack when the DPDA first scanned the character c. Since the DPDA did not progress further on the input tape nor did it pop A off the stack, in the graph G_c there is a cycle. Since each node in G_c corresponds to a semi-ID that M reaches in its computation it is clear that M reaches a semi-ID (q,c,B) such that (q,B) is in a cycle in the graph G_c. ■

The final construction that yields the machine M' which accepts the same language as M which is total consists of the following construction. For every node (q,B) in G_c which is on a cycle, change the transition function δ so that δ(q,c,B) $\rightarrow$ "dead state".

This construction will now ensure that the DPDA M' will read the entire input and halt even if M did not halt.

■

5.2. Digression: discussion of problem about reducibility relations

[Lecture 7, scribed by Daniel Boyd and Mike Smolowitz.]

Just when will a reducibility relation be reflexive?

For a reducibility relation on a class of functions $\mathcal{F}$ to be reflexive, for all languages A it would have to be true that $A \leq_{\mathcal{F}} A$. Then for all $A \subseteq \Sigma^*$, there exists a function $f \in \mathcal{F}$ such that for all $x \in A$, $x \in A \iff f(x) \in A$.

So if $A = \emptyset$, then it's easy. $\emptyset \leq_{\mathcal{F}} \emptyset$.

Well, this certainly looks like $\mathcal{F}$ has to contain the identity function, but we can fool around to remove even this condition:

Let a_1, a_2 be elements of A.

$$f(x) = \begin{cases} a_2 & \text{if } x = a_1 \\ a_1 & \text{if } x = a_2 \\ x & \text{otherwise} \end{cases}$$

Then f isn't the identity in $\mathcal{F}$, yet $A \leq_m A$ via f.

The last case is if $||A|| = 1$. We know that $A \leq_m^{\mathcal{F}} B$ via f if and only if $\bar{A} \leq_m^{\mathcal{F}} \bar{B}$ via f. So apply the last case to A.

Warning: $A \leq_m B$ via f means $A = f^{-1}(B)$, but is not the same as $f(A) = B$.

Question: Is it true that whenever, all functions in $\mathcal{F}$ are recursive, then $\leq^{\mathcal{F}}$ is reflexive if and only if $\mathcal{F}$ contains the identity?

6. Closure properties of CFLs

[Lectures 7–8, scribed by Boyd and Smolowitz.]

Fact: The DCFLs are closed under complementation, but not closed under intersection or union. For example:

$$L = \{a^i b^j c^k : i = j\}$$

L is the intersection of two DCFLs, yet L is not CF.

But DCFL is closed under intersection with a regular set. A simple sketch of this: let D be a DPDA for a CFL A, and let M be a DFA for a regular language R. Define the DPDA D' (which will accept $A \cap R$) by the following sets of states and final states:

$$\begin{aligned} Q &= Q_D \times Q_M \\ F' &= F_D \times F_M \end{aligned}$$

The machine D' runs a copy of M in parallel with the finite control of D. Whenever D makes a right move on the input tape, the simulated M gets to make a transition on the new input symbol. The combined machine accepts only when both of its components are simultaneously in accepting states.

6.1. Monoid of transformations

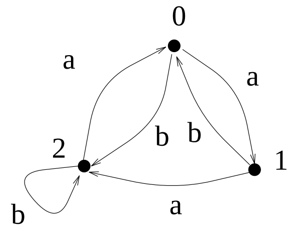
One way of talking about Σ^* is to call it the free monoid over Σ with operation $\cdot$ (concatenation). So don't be afraid of the words 'free monoid'.

Definition 6.1. Let $M = (Q, \Sigma, \delta, s, F)$ be a DFA. The monoid of transformations $\mathcal{M}$ of M is defined as the collection of mappings $g_c\colon Q \rightarrow Q$ over all $c \in \Sigma$ where $g_c(q) = \delta(q, c)$, together with the identity and with their closure under composition. ($\mathcal{M}$ is 'generated' by the g_cs.

Example 6.1. Let $\Sigma = \{a, b\}$. Consider this DFA (start and final states are unimportant.)

Then g_a is the transformation on $\{0, 1, 2\}$ performed by this machine on reading an a. If in state 0 initially, an a takes you to state 1. If in state 1, an a takes you to 2. In state 2, an a takes you to state 0. This is written as $g_a = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \end{pmatrix}$, which is standard notation for permutations.

Now note that g_b isn't a permutation, because it has two 2s.



$$\begin{aligned} g_a &= \begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \end{pmatrix} \\ g_b &= \begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 2 \end{pmatrix} \\ g_{ab} = g_b \cdot g_a &= \begin{pmatrix} 0 & 1 & 2 \\ 0 & 2 & 2 \end{pmatrix} \\ g_{aa} = g_a \cdot g_a &= \begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \end{pmatrix} \end{aligned}$$

Note: the identity need not be in the composition of the g_cs, if some state is a *source* in the graph of δ.

Now, $\mathcal{M}$ has the identity $g_\epsilon(q) = q$. $\mathcal{M}$ is closed under $\cdot$ (composition). $\cdot$ is associative and is *something else which I didn't write down.* Anyway, these properties define a monoid or a semigroup.

If for all $g \in \mathcal{M}$, there exists an $h \in \mathcal{M}$ such that $h \cdot g$ is the identity, then $\mathcal{M}$ becomes a group (leaving behind the pastures of immature monoidality. This can happen if and only if every g_c is a permutation.

Elements of $\mathcal{M}$ can be encoded over the alphabet

$$Q^Q = \{|Q| - \text{tuples of elements in} q\}$$

$g \in \mathcal{M}$ is encoded as the single combination character $(g(q_0), g(q_1), g(q_2) \dots g(q_{|Q|-1}))$.

Definition 6.2. The "annotation" of a string $w \in \Sigma^+$ by M is the string w' of the same length over alphabet $Q^Q \times \Sigma$ such that

$$\forall i, w'_i = (g_{w_i w_{i+1} \cdots w_n}, w_i).$$

This gives information about the last state reached by $M(w)$ given the state of M before reading w_i.

Example 6.2. $w = abab$

$$w' = \underbrace{\begin{pmatrix} 0 & 1 & 2 \\ 0 & 2 & 2 \end{pmatrix}}_{g_{abab}} \underbrace{\begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 2 \end{pmatrix}}_{g_{bab}} \underbrace{\begin{pmatrix} 0 & 1 & 2 \\ 0 & 2 & 2 \end{pmatrix}}_{g_{ba}} \underbrace{\begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 2 \end{pmatrix}}_{g_b}$$

This is a diagram that showed up here somewhere. Typically I include such diagrams so they can be deleted later if someone thinks they don't mean much.

The *-free regular languages are those that are representable with $\cup$, $\cap$, $\cdot$, but no Kleene*. Also, $\mathrm{NC}^0 \subset \mathrm{AC}^0$.

Notation. Let D be a DPDA. We will denote an ID of D as (q, α, x) where q is the current state of D, $\alpha = \alpha_1\alpha_2 \dots \alpha_m$ is the current contents of the stack with α_1 at the top, and x is the remaining portion of the input on the input tape.

Lemma 6.1. *(This is in [HU79] as lemma 10.5 in section 4.) Let D be a DPDA, and M be a DFA with states $\{p_0 \dots p_k\}$.*

Then D can be modified to D' such that in any ID (q, α, x) of D, the corresponding ID of D' has α' fully annotated by M. In other words, D' has on its stack

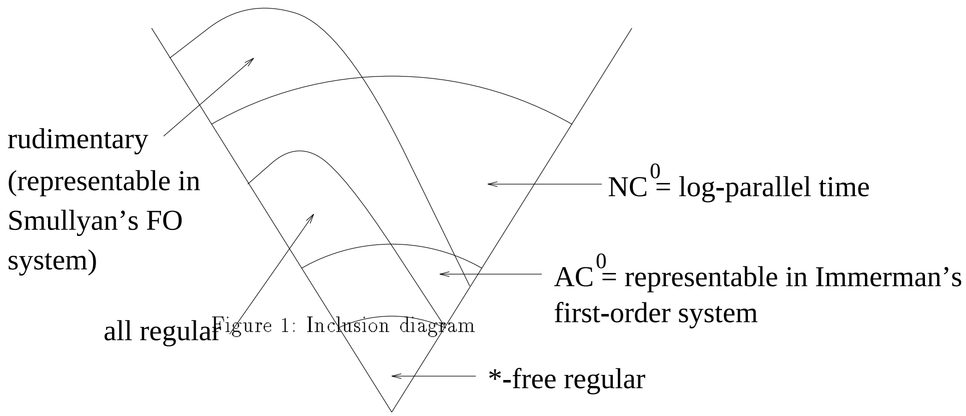


Figure 1: Inclusion diagram

$$\alpha' = \begin{pmatrix} \delta^*_M(p_0, \alpha) \\ \delta^*_M(p_1, \alpha) \\ \vdots \\ \delta^*_M(p_k, \alpha) \end{pmatrix}, \alpha_1 \Bigg) \cdots \Bigg(\begin{pmatrix} \delta^*_M(p_0, \alpha_{m-1}\alpha_m) \\ \delta^*_M(p_1, \alpha_{m-1}\alpha_m) \\ \vdots \\ \delta^*_M(p_k, \alpha_{m-1}\alpha_m) \end{pmatrix}, \alpha_{m-1} \Bigg) \Bigg(\begin{pmatrix} \delta^*_M(p_0, \alpha_m) \\ \delta^*_M(p_1, \alpha_m) \\ \vdots \\ \delta^*_M(p_k, \alpha_m) \end{pmatrix}, \alpha_m \Bigg)$$

Idea: Looking at the top stack symbol α'_1 tells what M would do on α. More concisely,

$$\alpha' = (g_\alpha, \alpha_1)(g_{\alpha_2 \cdots \alpha_m}, \alpha_2)(g_{\alpha_3 \cdots \alpha_m}, \alpha_3) \cdots (g_{\alpha_{m-1}\alpha_m}, \alpha_{m-1})(g_{\alpha_m}, \alpha_m)$$

.

Proof. Use a normal form for D in which every move has the form $(q, c, A, \epsilon, \{R, S\}, r)$ (a POP move) or $(q, c, A, BA, \{R, S\}, r)$ (a PUSH move).

Building D': Initially, stack contains $\wedge$. (We may suppose that DFA M doesn't ever run on $\wedge$ or $g(\wedge)$ is the identity on Q_M.)

Invariant: stack is nonempty and fully annotated.

POP moves preserve the invariant (nothing to do).

In PUSH moves, D' reads $\delta^*(p_i, A\alpha_2\alpha_3 \dots \alpha_m)$. D wants to write, for each state p_i,

$$\delta^*(p'_i, BA\alpha_2 \dots \alpha_m) = \delta^*(\delta(p'_i), A\alpha_2 \dots \alpha_m)$$

and this can be computed locally since the "A" info is already there. $g_{BA\alpha_2 \dots \alpha_m} = g_B \cdot g_{A\alpha_2 \dots \alpha_m}$. This works for other annotations, too. ■

Define $Min(L) = \{w \in L : (\nexists v \in \Sigma^+) wv \in L\}$. This is somewhat like forcing L to have the prefix property, by throwing away any string that has an extension that is also in L.

Theorem 6.2. *If $L \in$ DCFL, then so is $Min(L)$. (The proof prefers the "freeze" DPDA for the pruning proof.)*

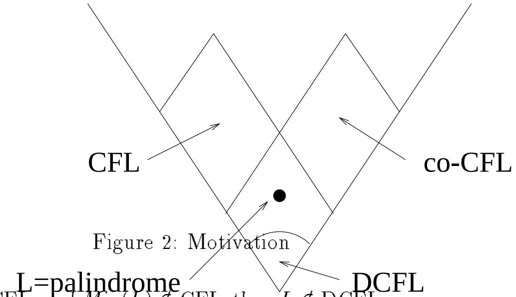


Figure 2: Motivation

Corollary 6.3. *If $L \in$ CFL and $Min(L) \notin$ CFL then $L \notin$ DCFL.*

Proposition 6.4. *If L is regular, then $Min(L)$ is regular.*

Proof. Delete all arcs out of final states of DFA for L. ∎

(Added by DFB) I think this conflicts with the given definition of Min. Consider

$$Min(a^*) = \{a^n : (\nexists a^m, m > 0)\ a^n a^m \in a^*\} = \{a^n : (\nexists m > 0)\ n + m \in \mathbb{N}\} = \emptyset.$$

In other words, $Min(a^*)$ is the set of all strings of a's such that there isn't a longer string of a's. That's the empty set.

I think something is wrong with this definition.

Answer by KWR: Touché: evidently I got $Min(L)$ and $Max(L)$ confused in my lecture (or you did). The definition in Comment 6.1 is for $Max(L)$, but the two stated results are indeed for $Min(L)$. $Max(a^*) = \emptyset$ and $Min(a^*) = \{\epsilon\}$. Pages 244-245 of [HU79] set the record straight.

7. Predicting Machine Construction

[Lecture 9, scribed by Boyd and Smolowitz.]

Theorem 7.1. *Let D be a DPDA, and let M be a DFA. Then D can be modified to D', which maintains in its top stack symbol Z the information Δ_Z. Δ_Z is the set of pairs (p, q), where $p \in Q_M$ and $q \in Q_D$, such that there is a $w \in \Sigma^*$ such that: M started in state p accepts w, and D started in state q with the whole current stack accepts w.*

The idea is that as you begin reading the input x, you update the "foreknowledge."

Corollary 7.2. *Let L be a DCFL, and let B be any regular set. Define the quotient $L/B = \{x \in \Sigma^* : (\exists w \in B)\, xw \in L\}$. Then L/B is a DCFL.*

Theorem 7.3. *If L is regular and B is arbitrary, then L/B is regular.*

Proof. *Construct a DFA M_L. Put $q \in F'$ if $\exists y \in B$ such that M started in state q on y accepts.* ∎

One point of the DCFL theorem is that when B is regular, Δ_Z can be defined constructively. The intuitive reason is that the problem:

INSTANCE: A CFL A, a regular set B.

QUESTION: Is $A \cap B \neq \emptyset$?

is decidable.

Proof. (DCFL Theorem)

Basis.

$Z = \wedge$. $\Delta_Z = \{(p, q) : L(M_p) \cap L(D_q) \neq \emptyset\}$, where $L(M_p) = \{w \in \Sigma^* : \delta^*_M(p, w) \in F_m\}$ and $L(D_q) = \{w \in \Sigma^* : D$, started in state q, accepts $w\}$.

We need to solve $||Q_M|| \cdot ||Q_D||$ instances of that decision problem to build $\Delta_\wedge$.

Invariant: Every symbol A on the stack has the correct symbol Δ_A for the substack with that symbol as its top.

Observation: Δ_A is independent of the bits of x already read, which are $x_0 \cdots x_{m-1}$.

Claim: It is sufficient to show how to update the stack in a move of type (q, c, A, BA, S, r).

Reason:

- Whether the direction is S or R doesn't matter - it's independent of the bits already used.
- POP moves automatically preserve the invariant. In the NF (normal form), either move is either "POP A" or "PUSH B".

Induction Hypothesis. Δ_A is correct.

The objective is to build Δ_B such that Δ_B depends *only* on D and M. We *can't* make Δ_B depend on the whole stack. It does so indirectly, via Δ_A. The *correctness* of Δ_B depends on the correctness of Δ_A.

We want to put (p, r) into Δ_B iff one of these two hold:

(1) There is some string u and $(p', q') \in \Delta_A$ such that: $\delta^*_M(p, u) = p'$; and D, started in state r with stack $\frac{B}{A}$, gets to state q' with stack A, with this being the first time after state q that D scans A again, and having read exactly u.

Idea: $w = uv$. If $(p', q') \in \Delta_A$, and (1) holds, then:

- There is a v such that D in state q' with A as the top-of-stack accepts V.
- There is a w such that D in state r with B as the top-of-stack accepts $w = uv$.

(2) There is a $u \in \Sigma^*$ such that: $\delta_M^*(p,u) \in F_M$; and D, started in state r with stack B, never pops that B and accepts u.

We have correctness, from (1) and (2). The construction reduces to solving instances of the same decision problem.

■

This is a top-down approach; the text does it from the bottom-up.

8. LR(k) Grammars

[Lectures 10–14, postpared by K.W. Regan.]

These notes get to the theoretical heart of the LR(k) principle with a minimum of new conceptual overhead. It does not lead to the most efficient parsers for LR(k) grammars, but provides a basis for addressing issues of efficiency later.

Let $G = (V, T, P, S)$ be any context-free grammar. Call G "start-normal" if S does not appear on the right-hand side of any production. (It is trivial to change any given CFG into one which is "start-normal" by adding a new start symbol.) Let $\Longrightarrow_r^*$ stand for " generates by a rightmost derivation." The motivation is that an LR(k) parser reading an input $x \in T^*$ works backward from x to find a derivation of x, and since the input is conventionally read left-to-right by a DPDA, the desired derivations are rightmost.

Definition 8.1. For any $k \geq 1$, an *LR(k) item* for the CFG G has form either

$$A \to \alpha \bullet \beta \text{ (on } z) \qquad \text{or} \qquad A \to \alpha \bullet \beta \text{ (on } y\$),$$

where $A \to \alpha\beta$ is a production in G, $z \in T^k$, and $y \in T^{<k}$. The item is *complete* if $\beta = \epsilon$. In an *LR(0) item*, the (on z) or (on y) part is omitted altogether.

The intuitive meaning is that the item applies if the next k input characters, including the character currently scanned, equal z, or if there are at most $k-1$ characters left and these equal y.

Definition 8.2. For any $k \geq 0$, CFG G, and string $\gamma \in (T \cup V)^*$, the set $I^k(\gamma)$ of LR(k) items which are *valid for* γ equals

$$\begin{aligned} &\{ A \to \alpha \bullet \beta \text{ (on } z) : (\exists \delta \in (T \cup V)^*, w \in T^*)\, \gamma = \delta\alpha \ \wedge\ w \sqsupseteq z \ \wedge\ S \Longrightarrow_r^* \delta A w \} \\ \cup\ &\{ A \to \alpha \bullet \beta \text{ (on } y\$) : (\exists \delta \in (T \cup V)^*)\, \gamma = \delta\alpha \ \wedge\ S \Longrightarrow_r^* \delta A y \}. \end{aligned} \tag{1}$$

A string γ is *viable* for G if $I^k(\gamma)$ is nonempty, and *completely viable* if $I^k(\gamma)$ contains a complete item. The basic idea is that the DPDA simulating G will keep (completely) viable prefixes γ which occur in a rightmost derivation from S on its stack. We show first that $I^k(\gamma)$ can be determined by a DFA as γ is read. Hence by the "first predicting machine construction," the DPDA can "annotate" its stack by this DFA and thus maintain in its top-of-stack symbol the information about exactly which (complete) items are valid for γ.

Theorem 8.1. *Let $G = (V, T, P, S)$ be any CFG, and let $k \geq 0$. Then we can build a DFA $M = (Q_M, \Sigma, \delta_M, s_M, F_M)$, where $\Sigma = T \cup V$ and each state in Q_M is some set of LR(k) items, such that for all $\gamma \in \Sigma^*$, $\delta_M(\gamma) = I^k(\gamma)$.*

The motivation is that if we define F_M to comprise all sets I of items which contain at least one complete item, then $L(M)$ equals the language of completely viable prefixes of G. M will have a dead state $\emptyset$, which corresponds to $I^k(\gamma)$ being empty.

Proof. It suffices to build an NFA$_\epsilon$ $N = (Q_N, \Sigma, \delta_N, s_N, F_N)$ such that Q_N equals the set of LR(k) items for G plus a special start state s_N, and for all $\gamma \in \Sigma^*$ and items i, $i \in \delta_N^*(s_N, i) \iff i \in I^k(\gamma)$. Then applying the sets-of-states construction to this NFA$_\epsilon$ (with F_N equal to the set of complete items) automatically produces M. For any language L and $k \geq 0$, let $\mathit{Prefs}^k(L)$ stand for $\{ x : |x| = k \ \wedge\ (\exists y \in L) x \sqsubseteq y \}$. For $\beta \in (T \cup V)*$, $L(\beta)$ stands for $\{ x \in T^* : \beta \Longrightarrow_r^* x \}$. We define N by the following three rules:

(0) For each production $S \to \beta$ in G, put $(s_N, \epsilon, S \to \bullet\beta \text{ (on } \$))$ into δ_N.

(1) For each item $i = A \to \alpha \bullet X\beta$ (on z), where $X \in T \cup V$, put $(i, X, A \to \alpha X \bullet \beta \text{ (on } z))$ into δ_N. Same with (on $y\$$) in place of (on z).

(2a) For each item $i = A \to \alpha \bullet B\beta$ (on z), where $B \in V$, and for each production $B \to \eta$ in G, put $(i, \epsilon, B \to \bullet\eta \text{ (on } z'))$ into δ_N for all $z' \in \mathit{Prefs}^k(L(\beta)\cdot z)$.

(2b) For each item $i = A \to \alpha \bullet B\beta$ (on $y\$$), with $B \to \eta$ as before, put $(i, \epsilon, B \to \bullet\eta \text{ (on } z'))$ into δ_N for all $z' \in \mathit{Prefs}^k(L(\beta)z)$, and $(i, \epsilon, B \to \bullet\eta \text{ (on } y'\$))$ into δ_N for all $y' \in L(\beta)y'$ such that $|y'| < k$.

Given this definition of N, we need to show (i) that the transition rules preserve validity, and (ii) that every valid item for a given γ can be obtained by transitions of N. In symbols, (i) states "for all γ, $\delta_N^*(\gamma) \subseteq I^k(\gamma)$, while (ii) states "for all γ, $I^k(\gamma) \subseteq \delta_M^*(\gamma)$." Both statements suggest a proof by induction, but we have to figure out what best to induct *on*. For (i) we can do a "structural induction" on δ_N itself.

(0) *Basis.* The only transitions out of the start state s_N are ϵ-moves to items $S \to \bullet\beta$ (on $\$$) for productions $S \to \beta$ in G. With reference to Definition 8.2, we have $\gamma = \epsilon$, and since $y = \epsilon$, need only check that $S \Longrightarrow_r^* \epsilon S \epsilon$, which is true. Hence all such items belong to $I^k(\epsilon)$.

(1) *Induction.* Rules (1) and (2) only come up after the initial transition by N, so they belong only to the induction step. Suppose that the item $j = A \to \alpha \bullet \beta$ (on z) is reached by a path reading γ whose last transition is of type (1). Let $\gamma = \gamma_0 X$ where $X \in T \cup V$; then the transition must be (i, X, j) where i equals $A \to \alpha_0 \bullet X\beta$ (on z) with $\alpha = \alpha_0 X$. By induction hypothesis, the item i is valid for γ_0, so there exists $\delta_0 \in (T \cup V)^*$ such that

$$\gamma_0 = \delta_0\alpha_0 \ \wedge\ (\exists w \in T^*)(w \sqsupseteq z \ \wedge\ S \Longrightarrow_r^* \delta A w). \tag{2}$$

But then the same holds for γ in place of γ_0 with α in place of α_0, with the same $\delta = \delta_0$. Hence j is valid for γ. A similar argument holds in the case of an item $A \to \alpha \bullet \beta$ (on $y\$$).

(2) Suppose j is an item obtained by a path reading γ in which the last transition is of type (2a); then j has the form $j = B \rightarrow \bullet\eta$ (on z') where $B \in V$. Then there is an item i of the form $i = A \rightarrow \alpha \bullet B\beta$ (on z), and since (i, ϵ, j) is an ϵ-transition, by the induction hypothesis i is valid for γ. Hence there exists δ_0 such that $\gamma = \delta_0\alpha$ and

$$(\exists w \in T^*)(w \sqsupseteq z \ \wedge\ S \Longrightarrow_r^* \delta_0 A w). \tag{3}$$

To show that j is valid, we need to show that

$$(\exists w' \in T^*)(w' \sqsupseteq z' \ \wedge\ S \Longrightarrow_r^* \gamma B w'). \tag{4}$$

By rule (2a), $z' \in \mathit{Prefs}^k(L(\beta)\cdot z)$. Since there exists a string $w \in T^*$ of the form $w = zv$ such that $S \Longrightarrow_r^* \delta_0 A w \Longrightarrow_r \gamma B\beta z v$, and $z' \sqsubseteq L(\beta)\cdot z$, (4) follows.

The argument for a transition of type (2b) is similar: if j has the same form as above and follows from $i = A \rightarrow \alpha \bullet B\beta$ (on $y\$$), use $S \Longrightarrow_r^* \delta_0 A y \Longrightarrow_r \gamma B\beta y$ and the fact that $z' \in \mathit{Prefs}^k(L(\beta)\cdot y$. If $j = B \rightarrow \bullet\eta$ (on $y'\$$), use this and the fact that $y' \in (L(\beta)\cdot y$ and $|y'| \le k - 1$.

For part (ii), it is natural to try an induction on the length of the string γ. But since N has ϵ-moves, this can be circular if we try to reach a given valid item j in $I^k(\gamma)$ by a path whose last transition is of type (2). Thus, let us try to induct also on the length n of the derivation(s) in the definition of $j \in I^k(\gamma)$. For the basis, we observe that no items of the form $S \rightarrow \alpha \bullet \beta$ (on z) can belong to $I^k(\epsilon)$, because S never appears on the right-hand side of a production, and the only items $S \rightarrow \alpha \bullet \beta$ (on $y\$$) in $I^k(\epsilon)$ must have $y = \epsilon$ and $\alpha = \epsilon$, with $S \rightarrow \beta$ being a production in G. All such items are reached by ϵ-moves from s_N. This gives us the result for $\gamma = \epsilon$ and $n = 0$. Now given any item in $I^k(\gamma)$ of the form $j = A \rightarrow \alpha \bullet \beta$ (on z) or $j = A \rightarrow \alpha \bullet \beta$ (on $y\$$) with $A \neq S$, we break into two cases which comprise the induction:

(1) *Case* $\alpha \neq \epsilon$. Then there exists $X \in T \cup V$ and $\alpha_0 \in (T \cup V)^*$ such that $\alpha = \alpha_0 X$. By the validity of j for γ, there exist $\delta \in (T \cup V)$ such that $\gamma = \delta\alpha$ and (etc.). Let $\gamma_0 = \delta\alpha_0$, so that $\gamma = \gamma_0 X$. Then the item $i = A \rightarrow \alpha_0 \bullet X\beta$ (on $\ldots$) is valid for γ_0 as well. By the induction hypothesis, which applies since $|\gamma_0| < |\gamma|$, i is in $\delta_N^*(\gamma_0)$. Hence j is in $\delta_N^*(\gamma)$ since it can be reached in one more step by the transition (i, X, j).

(2) *Case* $\alpha = \epsilon$. It helps notationally to re-write the first form of item j as $j = B \rightarrow \bullet\eta$ (on z'). By $j \in I^k(\gamma)$, it follows that

$$(\exists w' \in T^*)(w' \sqsupseteq z' \ \wedge\ S \Longrightarrow_r^* \gamma B w'). \tag{5}$$

Fix any such w', and consider the production which introduced B in this rightmost derivation. Without loss of generality, we can write this production as $A \rightarrow \alpha B\beta$, where for some terminal string w, $\beta w \Longrightarrow_r^* w'$. If $|w| \ge k$, let z be the first k characters of w. It follows that there exists $\delta \in (T \cup V)^*$ such that $\gamma = \delta\alpha$, and we have $w \sqsupset z$ and $S \Longrightarrow_r^* \delta A w$. Hence the item $i = A \rightarrow \alpha \bullet B\beta$ (on z) belongs to $I^k(\gamma)$. Since $\ldots$, the induction hypothesis gives us a δ_N path from s_N to i reading γ. Since $z' \in \mathit{Prefs}^k(L(\beta)\cdot z)$, δ_N has the transition (i, ϵ, j). Hence there is a δ_N path from s_N to j reading γ as well.

If $|w| < k$, then take $y = w$. Similar reasoning shows that the item $i = A \rightarrow \alpha \bullet B\beta$ (on $y\$$) is valid for γ, and using the induction hypothesis, there is a path to j reading γ via i and rule (2b).

Last, for the second form $j = B \rightarrow \bullet\eta$ (on $y'\$$), use similar reasoning. $\square$

Definition 8.3. For all $k \ge 1$, a CFG $G = (V, T, P, S)$ is $LR(k)$ if it is "start-normal" and

(a) No two complete items have the same "on" strings.

(b) For every incomplete item of the form $A' \rightarrow \alpha' \bullet c\beta'$ (on $\ldots$) where $c \in T$, no "on" string in a complete item begins with c.

For $k = 0$, G is $LR(0)$ if whenever $I^0(\gamma)$ contains a complete item, no other item is complete or has a terminal c directly to the right of the $\bullet$.

9. Proof of the equivalence to DPDAs

[Lectures 15–17, prepared by Regan.]

Theorem 9.1. *For any $k \ge 0$, if G is an $LR(k)$ grammar, then $L(G)$ is a DCFL.*

Proof. We describe our DPDA D as though whenever D is reading an input symbol c, it also sees the next $k-1$ characters, or if the c is within $k-1$ places of the end of the input x, D sees the remaining symbols plus the fact that the input ends. This is done to make it easier to describe the main *invariant* of the operation of D. Such a D can be simulated by a standard DPDA D' which remembers the last $k-1$ characters seen in its current state. When D' reaches the end of the input x, the relaxed halting conditions which we have made standard allow D' to continue simulating D by appropriate operations on its stack. (And, as shown by the "first predicting machine construction," by annotating the stack from top to bottom with the actions that D' would take after reaching the end of the input, one can simulate D' by a DPDA D'' which operates under the "Freeze!" convention, at an even greater cost in stack alphabet size and number of states.)

D annotates its stack from bottom to top with the state sequence of the DFA M which recognizes viable prefixes. Thus the top symbol of γ encodes $I^k(\gamma)$ as well. In any step, reading input character c and "looking ahead" to v (if $|cv| < k$ then think of cv as $y\$$), D does one of the following:

(a) If there is a complete item $A \rightarrow \alpha\bullet$ (on cv) in $I^k(\gamma)$, then by the definition of validity, α is atop the stack, and then D pops α, pushes A, and keeps its input head stationary.

(b) Else, if there is a valid incomplete item of the form $A \rightarrow \alpha \bullet c\beta$ (on $\ldots$), push c onto the stack and move the input head right.

(c) Else, go to a dead state and reject.

Colloquially, (a) is called "reducing α to A," and (b) is called a "shift." D accepts iff it reads all of the input and the stack holds just S. By the LR(k) property, there is at most one valid complete item which can apply in (a), and action (b) is determined even if there there is more than one incomplete item with lookahead cv. Hence D is deterministic. We need to show that D is correct, and this is the time to observe several *invariants* of the operation of D on any input x:

(1) At any step in the operation of D, let y be the portion of the input x remaining to be read (including the current symbol c), and let γ be the current contents of the stack. Then, unless D is in a dead state, $\gamma y \Longrightarrow_r^* x$, and we can call γy the sentential form *represented* by D at this step.

(2) The sentential form represented does not change after a shift move.

(3) Immediately after a reduce move, γ ends in some variable A, and D is not in a dead state. Since the complete item $A \to \alpha\bullet$ (on cv) used in the reduction was valid, there exists some $w \in T^*$ such that $S \Longrightarrow_r^* \gamma w$.

It is immediate that these three conditions are preserved in every move by D. If $D(x)$ accepts, then D has only S on its stack, and by invariant (1), $S \Longrightarrow_r^* x$. We need to prove the converse: if $x \in L(G)$, then $D(x)$ accepts. It is tempting to say that this follows immediately from invariant (3): "once D gets within $k-1$ symbols of the end of the input x, the definition of validity requires that $w = cv$, and together with (1) this implies that $S \Longrightarrow_r^* \gamma w \Longrightarrow_r^* x$, so D will accept x." Well, the "so" doesn't necessarily follow, and more to the point, what if $D(x)$ enters its dead state, and what if $D(x)$ never reads all of the input x at all? We could try to answer the latter question by applying the construction which forces D to read all of its input, but even the resulting DPDA D' does still observe the three invariants (problem: does it?), the former question holds force if and when D' goes to its dead state. First we prove the following:

Claim: If $x \in L(G)$, then for every form $\gamma y = \delta\alpha y$ in which D makes a reduce move to obtain the form $\eta_1 = \delta_1 A y$, there is no sentential form $\eta_2 \neq \eta_1$ such that $\eta_2 \Longrightarrow_r \gamma y$ and $S \Longrightarrow_r^* \eta_2$.

Suppose the claim is false. Then let $B \to \beta$ be the production used in $\eta_2 \Longrightarrow_r \gamma y$, so that there are strings $\delta_2 \in (T \cup V)^*$ and $z \in T^*$ such that $\eta_2 = \delta_2 B z$ and $\gamma y = \delta_2\beta z = \delta_1\alpha y$. Now we have several cases:

(i) $|z| > |y|$. Then there exists $u \in T^*$ such that $z = uy$. Then u is the top portion of the stack, meaning that the most recent moves by D were to shift u from input to stack. Let c be the first symbol of u, and consider the earlier move when D shifted c. Since the form γw represented during these moves did not change, at that point β was its top of stack. Now we claim that at this point, with D scanning c and the next up-to-$(k-1)$ characters after u being v, that the complete item $B \to \beta\bullet$ (on cv) was valid for $\delta_2\beta$. This is because $S \Longrightarrow_r^* \delta_2 B z$ and $z \supseteq cv$, by the definition of validity. Thus D would not have shifted c at that point; instead, it would have performed a reduction.

(ii) $|z| = |y|$. Then $z = y$. By similar reasoning, with cv now being the first up-to-k symbols of y or z, the complete item $B \to \beta\bullet$ (on cv) is valid for γ, together with the distinct item $A \to \alpha\bullet$ (on cv) used in the reduce move. But this violates the LR(k) condition that no two complete items which are valid for the same γ can have the same lookahead string.

(iii) $|z| < |y|$. Then $y = uz$ for some $u \in T^*$. Let cv be the first up-to-k symbols of y, and β ends at the same place u does. First suppose that β contains the symbol c, so that β has the form $\beta_1 c\beta_2$. Let dv' be the first up-to-k symbols of z. Then the incomplete item $B \to \beta_1 \bullet c\beta_2$ (on dv') is valid for γ. But condition (b) in the definition of LR(k) states that then no complete item in $I^k(\gamma)$ should have an "on string" which begins with c, and $A \to \alpha\bullet$ (on c) is such an item.

Finally, suppose β is completely to the right of c (this includes the subcase of (iii) where β is empty). Then there exists a string $u' \in T^*$ such that $\eta_2 = \gamma c u' B z$. Trace the purported rightmost derivation $S \Longrightarrow_r^* \eta_2$ backward until one reaches the step in which c was introduced. This step can be represented as $\delta'' B' z' \Longrightarrow_r \delta''\beta_1' c\beta_2' z'$. Here β_2' must contain a variable, since B is to the right of c in η_2, and likewise all intermediate steps between $\delta''\beta_1' c\beta_2' z'$ and η_2 have variables to the right of c. Thus $\delta''\beta_1' = \gamma$. Hence, with dv' as the first up-to$k$ symbols of z', the incomplete item $B' \to \beta_1' \bullet c\beta_2$ (on dv') belongs to $I^k(\gamma)$, causing the same contradiction as before.

This proves the claim. In particular, this proves that D can never "loop," since at some point there would have to be a branch off the loop which goes back to S. Finally, suppose D enters its dead state at some point. This can only happen if $I^k(\gamma)$ becomes empty, and this could only happen after a reduce move, producing $\delta A y$ from γy, with reference to the above. From the claim, it follows that $S \Longrightarrow_r^* \delta A y$ (note that the definition of valid item only gives $(\exists w)\, S \Longrightarrow_r^* \delta A w$). Trace the derivation backward from $\delta A y$ to S until one finds the step in which the A in introduced, and by reasoning similar to (iii) above, this yields a valid item in $I^k(\delta A)$.

This completes the proof of Theorem 9.1. ☐

Corollary 9.2. *Every LR(k) grammar is unambiguous.*

Proof. It follows from the *Claim* in the last proof that there cannot exist two different rightmost derivations from S of any terminal string x. ☐

Remarks: I also think this settles another question implicit in the text: the CFG G need not be "reduced" in any form. Deadwood symbols, unreachable symbols, ϵ-productions, even trivial productions such as $A \to A$ are all AOK. (Unless A is never used in a derivation, having $A \to A$ will automatically violate the LR(k) conditions.) It is worth scanning the proof of the main claim for airtightness under the possibility that α and/or β is empty—my way of breaking down the cases according to z vs. y seems to handle that by itself.

My proof didn't use the clause that D only shifts a character c when $I^k(\gamma)$ is nonempty and contains some item with c just to the right of the $\bullet$. The clause isn't needed to describe the operations of D. So what this proves is that when the parser runs on an input $x \in L(G)$, there is always at least one valid item that corresponds to any shift move. One in particular comes from the production which produced the shifted character in the derivation of x;

there may be others. Similarly, even if the actual input x is not in $L(G)$, so long as $I^k(\gamma)$ is nonempty, there always is at least one valid item which corresponds to the shift. These remarks prove the equivalence of parsers with slightly different programs from D above.

Now we show how to convert a given DPDA into an equivalent LR(1) grammar. This shows the equivalence of DPDAs and LR(k) grammars for all $k \geq 0$. Following Hopcroft-Ullman, we take a detour via LR(0) grammars for $L(D)\$$, where \$ is a character not in Σ.

Lemma 9.3. *For every DPDA D we can find an equivalent DPDA D' whose state set Q' can be partitioned into $Q_R \cup Q_S$, such that every arc out of a state in Q_R moves the input head right, and every arc out of a state in Q_S keeps the input head stationary and does not depend on the input character scanned.*

This is pretty much how Hopcroft-Ullman defines a DPDA to begin with: if a state has an "ϵ-move," then it has no other reading moves (ch. 5, p113). The above is a little bit stronger in that the input head's action doesn't depend on the stack symbol either. This seems needed to make the grammar constructed below LR(0). To see this intuitively, imagine further that in a state in Q_R, D' first shifts the current input symbol c to its stack before making the move. Then every distinctive action by D' is in a transition which doesn't depend on the current input symbol at all, and thus D' functions with "zero lookahead." But it is not necessary to make this latter modification explicitly in order to see the LR(0) property in the grammar.

Proof. Given $D = (Q, \Sigma, \Gamma, \delta, \wedge, s, F)$, let Q_R be a copy of Q, and take $Q_S := Q \times \Sigma$. For all states $p \in Q_R$, wherever D has a tuple (p, c, A, X, R, q), D' has the same tuple. But wherever D has (p, c, A, X, S, q), D' instead has $p, c, A, X, R, (q, c))$. Now in a state (q, c) in Q_S, wherever D has a tuple (q, c, B, Y, S, r), for all $d \in \Sigma$ D' has the same tuple $((q, c), d, B, Y, S, (r, c))$. And if D has (q, c, B, Y, R, r), which finally reads c, then for all $d \in \Sigma$, D' has $((q, c), d, B, Y, S, r)$, which signifies that now D has "caught up with" D' so that D' can go back to states in Q_R. It is clear that D and D' have essentially the same computations. ☐

Theorem 9.4. *For every DCFL L, we can find an LR(0) grammar for $L \cdot \$$, where \$ is not in the alphabet of L.*

Proof. There exists a DPDA D which accepts $L\$$, such that every accepting computation simultaneously pops the initial bottom-of-stack character $\wedge$ and goes to a unique accepting state f. For ease of exposition, we suppose that D is not only in the normal form of Lemma 9.3, but also uses only PUSH and POP moves on its stack. That is, in every tuple $(p, c, A, X, \mathrm{D}, q)$, either $X = \epsilon$ or $X = BA$ for some $B \in \Gamma$ ($B = A$ allowed).

The CFG $G = (V, T, P, S)$ has $T = \Sigma \cup \{\$\}$ and $V = \{S\} \cup (Q \times \Gamma \times Q) \cup (Q \times (\Sigma \cup \{e\}) \times \Gamma \times Q)$, where e is a "place-holder" for the empty string. The productions fall into six groups:

(0) For all $p, q \in Q$, $c \in \Sigma \cup \{\$\}$, and $A \in \Gamma$, P has $[p, c, A, q] \to c$ and $[p, e, A, q] \to \epsilon$. (This includes the productions $[p, \$, \wedge, f] \to \$$ for all $p \in Q$.)

(1) P has $S \to [s, \wedge, f]$.

(2) For each tuple $(p, c, A, \epsilon, R, q)$ in δ, P has $[p, A, q] \to [p, c, A, q]$. (This includes $[p, \wedge, f] \to [p, \$, \wedge, f]$ for all $p \in Q$.)

(3) For each tuple $(p, c, A, \epsilon, S, q)$ in δ, P has $[p, A, q] \to [p, e, A, q]$.

(4) For each tuple (p, c, A, BA, R, q) in δ, and all $r, g \in Q$, P has $[p, A, r] \to [p, c, A, q][q, B, g][g, A, r]$.

(5) For each tuple (p, c, A, BA, S, q) in δ, and all $r, g \in Q$, P has $[p, A, r] \to [p, e, A, q][q, B, g][g, A, r]$.

Then we apply to G the (polynomial-time) algorithm which eliminates all deadwood variables. The resulting grammar has the property that whenever $\gamma \in V^*$, there exists some terminal string v such that $\gamma \Longrightarrow v$.

The intuition is that in (4) and (5), g stands for a *guess* as to what state D will be in when the particular A next becomes top-of-stack. Correct guesses yield a derivation which mimics the actions of D in an accepting computation, while any one incorrect guess will be shown to kill any chance that a terminal string can be derived.

In (2) and (4), the "4-domino" $[p, c, A, q]$ records that the last move by D was the unique tuple beginning $(p, c, A, \ldots)$. The point of the normal form in Lemma 9.3 is that in (3) and (5), the "4-domino" $[p, e, A, q]$ likewise comes from a unique last move, since if D has an ϵ-move in state p with top-of-stack symbol A, this move is unique (and independent of the current input). This seems to be needed for the LR(0) property. We also call the symbols of the form $[p, A, q]$ "3-dominoes." For intuition, we consider derivations in which each step expands the rightmost 3-domino in the sentential form, but 4-dominoes are not expanded until the very end, calling this kind of derivation "rightmost" as well. Now we observe the following *invariants*:

(a) In every sentential form derivable from S (not just in a rightmost derivation), the dominoes "match up like dominoes."

(b) In a rightmost derivation, the only two 3-dominoes which can touch are the rightmost 3-domino and its neighbor.

(c) In any sentential form (not necessarily from a rightmost derivation), any chain of touching 4-dominoes represents left-to-right a portion of a valid computation by D, giving the sequence of states, input head motions, and top-of-stack symbols encountered.

Invariant (c) holds because the 4-dominoes correspond directly to tuples of D. This is analogous to Lemma 10.6 in the text, and should likewise be formally proved by induction on lengths of derivations in G, but the above seems an acceptable "handwave." The following is my attempt to do Lemma 10.7 without quite so much formal machinery:

Claim: Let $\gamma \in V^*$, and take some terminal string v such that $\gamma \Longrightarrow^* v$. Then there exists a string γ_v of 4-dominoes such that $\gamma \Longrightarrow^* gamma' \Longrightarrow^* v$, and γ_v gives a sequence of valid moves of D on input v over which it reads all of v.

The claim follows from invariants (a) and (c). First we prove that $L(G) = L(D)$. If D accepts x, then consider the unique *leftmost* derivation in which at each step, the production chosen corresponds to the next move of D, and in productions of type (4) or (5), the guess g is read off from the computation path of D. The end result is a derivation of $x\$$. Now suppose $S \Longrightarrow_r^* x\$$. Consider a derivation which is rightmost for 3-dominoes. Then one of the sentential forms α consists entirely of 4-dominoes. Since $\alpha \Longrightarrow^* x\$$, invariant (c) implies that α represents a valid accepting computation of D on input $x\$$, so D accepts $x\$$.

Now we show that D has the LR(0) property. We need to show that for all $\gamma \in (T \cup V)^*$ such that $I(\gamma)$ contains a complete item $X \to \alpha\bullet$, $I(\gamma)$ does not contain a distinct complete item $Y \to \beta\bullet$, nor an item of the form $Z \to \bullet a$, where $a \in \Sigma \cup \{\,\$, \epsilon\,\}$. We attack the Y item first, and without loss of generality, may suppose $|\alpha| \leq |\beta|$. This breaks down into three cases:

(1) Neither $X \to \alpha$ nor $Y \to \beta$ is a terminal production (type 0): Then α and β are nonempty. By validity, each is a suffix of γ, so α is a suffix of β. None of the right-hand-sides in (1)–(5) can be a suffix of any other right-hand side except itself—in particular, $[s, \wedge, f]$ cannot be a suffix of another production because only the initial step produces it. Hence $\alpha = \beta$. All five types (1)–(5) are such that the right-hand side includes all information about the left-hand side. Hence the two productions cannot be distinct. (This is a little snappier than Lemma 10.9, "Case 1" in the text, owing to my putting both states into a 4-domino.)

(2) Both productions are of type (0). If $\alpha \neq \epsilon$, then both α and β are the same terminal c. Let v be a terminal string such that $\gamma \Longrightarrow_r^* v$, and take the 4-domino chain γ_v from the Claim. Then $\gamma_v X \Longrightarrow vc$ and $\gamma_v Y \Longrightarrow vc$. By invariant (c), γ_v specifies a sequence of valid moves by D on input vc which leaves the input head scanning c. Let q be the state D winds up in and A the top-of-stack symbol after this sequence of moves. Then since $\gamma_v X$ and $\gamma_v Y$ both occur as initial segments in (non-rightmost) derivations from S, and are chains of 4-dominoes, they too must be specify valid sequences of moves. But both X and Y must be moves in state q reading c and A, and since D is deterministic, they must specify the same move. Thus $X = Y$, so once again the two productions cannot be distinct.

If both α and β are ϵ, then X and Y both have "e" as their second component. Thus with γ_v as before, D on input v has first the sequence of moves specified by γ_v, which leaves the head scanning the cell to the right of the last symbol in v, and then the move specified by X vis-a-vis Y. But since both X and Y specify ϵ-moves, our normal form for D assures that these moves are independent of whatever symbol happens to be to the right of v, and since D is deterministic, that they are the same move. Hence $X = Y$ in this case as well. Last, if $\alpha = \epsilon$ and $\beta = c$, then by similar reasoning we get a case where D on input vc simultaneously has an ϵ-move and a read-move when scanning the c, which is also ruled out by our normal form.

(3) Exactly one of the productions is of type (0). By the suffix conditions and our assumption $|\alpha| \leq |\beta|$, we must have $\alpha = \epsilon$. Then γX is a prefix of a right-sentential form, and γ ends in β. Now β can't be a right-hand side from (4) or (5), because this would violate invariant (b) for rightmost derivations. Nor can β be a single 4-domino, since then the variable X could never have been produced in a rightmost derivation. So this is impossible.

Finally, we need to show that $Z \to \bullet a$ cannot co-exist with $X \to \alpha\bullet$ in $I(\gamma)$. By validity, there exists $w \in T^*$ such that $S \Longrightarrow_r^* \gamma Z w$, and α is a suffix of γ. As in case (3) just above, α cannot be a right-hand side of a production of types (1)–(5). Nor can α be a terminal, since this is a rightmost derivation. Hence α must be ϵ, and by validity, there exists $w' \in T^*$ such that $S \Longrightarrow_r^* \gamma X w'$. Again let v be such that $\gamma \Longrightarrow_r^* v$, and let γ_v be the chain of 4-dominoes from the claim. Then $\gamma_v X$ and $\gamma_v Z$ are chains of 4-dominoes which occur in derivations from S, so invariant (c) applies to them too. Thus D on input va has the sequence of moves specified by γ_v, which leaves the head scanning a (or the cell to the right of v if $a = \epsilon$), followed by the valid move specified by X or Z. But X specifies an ϵ-move, since $\alpha = \epsilon$, and by our normal form, Z must specify the same ϵ-move. Thus $Z = X$. This completes the entire proof. $\square$

Now define the grammar G_1 from G by changing the productions $[p, \$, \wedge, f] \to \$$ in (0) to $[p, \$, \wedge, f] \to \epsilon$. Clearly if $L(G) = L\$$, then $L(G_1) = L$. What goes wrong in the above proof if we try to show that $L(G_1)$ is LR(0)? The problem is that $I(\gamma)$ can now have a new kind of complete item $X \to \alpha\bullet$ with $\alpha = \epsilon$, namely $[p, \$, \wedge, f] \to \epsilon\bullet$. This can conflict with an item of the form $Z \to \bullet c$ where Z is the 4-domino $[p, c, \wedge, q]$. With reference to the last paragraph, let $\gamma \Longrightarrow^* v$, and take γ_v as before. Then $\gamma_v X$ is a chain of 4-dominoes and $\gamma X \Longrightarrow v$ also, but Invariant (c) no longer holds for it because $[p, \$, \wedge, f]$ no longer comes from a valid move (before, it came from the tuple $(p, \$, \wedge, \epsilon, R, f)$ which read the \$). There is now no contradiction with Z specifying a valid move of D in state p with stack $\wedge$ reading some input symbol c which comes after v. Below is a concrete example for a DPDA D which accepts a language L that doesn't have the prefix property.

However, this conflict goes away if we have "(on y)" lookahead in the sole case $y = \epsilon$ (where we have been writing this as (on \$)). Since $[p, \$, \wedge, f]$ can only appear as the rightmost symbol in a derivation, the *only* valid LR(1) item corresponding to the production is $[p, \$, \wedge, f] \to \bullet$ (on \$); that is, no item $[p, \$, \wedge, f] \to \bullet$ (on c) is ever valid for $c \in \Sigma$. Hence, by the definition of an LR*(1)* grammar, there is no conflict with $[p, c, \wedge, q] \to \bullet c$ (on $\ldots$).

Theorem 9.5. *G_1 is LR(1). Thus every DCFL has an LR(1) grammar.*

Proof. The above remarks, plus the idea that for $k \geq 0$, every LR(k) grammar is also an LR($k+1$) grammar, are enough to constitute a proof.[4] $\square$

Hence a language L is a DCFL only if, as well as if, there is some k such that L has an LR(k) grammar.

[4] Hecque, it's more than the text has!

Corollary 9.6. *The above statement remains true if "LR(k) grammar" is defined by the more-restrictive condition that whenever $I^k(\gamma)$ contains a complete item, it contains no other critical item, where an item $A \to \alpha \bullet \beta$ (on ...) is defined to be "critical" if $\beta \notin V^*$.*

The following example will help shed light on these observations.

Example 9.1. Let D be the DPDA with $Q = \{0, 1, 2, 3\}$, $\Sigma = \{a, b\}$, $\Gamma = \{A, \wedge\}$, start state 0, unique final state 3, and tuples

$$\{(0, \mathrm{a}, \wedge, A\wedge, \mathrm{R}, 1), (1, \mathrm{a}, A, AA, \mathrm{R}, 1), (1, \mathrm{b}, A, \epsilon, \mathrm{R}, 2)\} \cup$$

$$\{(2, \mathrm{a}, \wedge, A\wedge, \mathrm{R}, 1), (2, \mathrm{b}, A, \epsilon, \mathrm{R}, 2), (2, \$, \wedge, \epsilon, \mathrm{R}, 3)\}.$$

Then $L(D) = \{\mathrm{a}^m\mathrm{b}^m : m \geq 1\}^+\$$. In fact, D is the "most obvious" DPDA for this language, and since it makes no ϵ-moves (i.e., runs in real time), it is automatically in the normal form of Lemma 9.3. The corresponding grammar G_1 has productions:

(0) $[p, \mathrm{a}, A, q] \to \mathrm{a}$, $[p, \mathrm{a}, \wedge, q] \to \mathrm{a}$, $[p, \mathrm{b}, A, q] \to \mathrm{b}$, $[p, \mathrm{b}, \wedge, q] \to \mathrm{b}$, and $[p, \$, \wedge, 3] \to \epsilon$, for all $p, q \in Q$.

(1) $S \to [0, \wedge, 3]$.

(2-3) $[1, A, 2] \to [1, \mathrm{b}, A, 2]$, $[2, A, 2] \to [2, \mathrm{b}, A, 2]$, and $[2, \wedge, 3] \to [2, \$, \wedge, 3]$.

(4-5)

$$\begin{aligned} [0, \wedge, r] &\to [0, \mathrm{a}, \wedge, 1][1, A, g][g, \wedge, r], \\ [1, A, r] &\to [1, \mathrm{a}, A, 1][1, A, g][g, A, r], \\ [2, \wedge, r] &\to [2, \mathrm{a}, \wedge, 1][1, A, g][g, \wedge, r], \end{aligned}$$

for all $r, g \in Q$.

Now consider the following two rightmost derivations, the first of the string ab, the second of the string abab: $S \Longrightarrow_r [0, \wedge, 3]$

$$\begin{aligned} &\Longrightarrow_r [0, \mathrm{a}, \wedge, 1][1, A, 2][2, \wedge, 3] \\ &\Longrightarrow_r [0, \mathrm{a}, \wedge, 1][1, A, 2][2, \$, \wedge, 3] \\ &\Longrightarrow_r [0, \mathrm{a}, \wedge, 1][1, A, 2] \\ &\Longrightarrow_r [0, \mathrm{a}, \wedge, 1][1, \mathrm{b}, A, 2] \Longrightarrow_r^2 \mathrm{ab}, \end{aligned}$$

and $S \Longrightarrow_r [0, \wedge, 3]$

$$\begin{aligned} &\Longrightarrow_r [0, \mathrm{a}, \wedge, 1][1, A, 2][2, \wedge, 3] \\ &\Longrightarrow_r [0, \mathrm{a}, \wedge, 1][1, A, 2][2, \mathrm{a}, \wedge, 1][1, A, 2][2, \wedge, 3] \\ &\Longrightarrow_r [0, \mathrm{a}, \wedge, 1][1, A, 2][2, \mathrm{a}, \wedge, 1][1, A, 2][2, \$, \wedge, 3] \\ &\Longrightarrow_r [0, \mathrm{a}, \wedge, 1][1, A, 2][2, \mathrm{a}, \wedge, 1][1, A, 2] \\ &\Longrightarrow_r [0, \mathrm{a}, \wedge, 1][1, A, 2][2, \mathrm{a}, \wedge, 1][1, \mathrm{b}, A, 2] \\ &\Longrightarrow_r [0, \mathrm{a}, \wedge, 1][1, A, 2][2, \mathrm{a}, \wedge, 1]\ \mathrm{b} \\ &\Longrightarrow_r [0, \mathrm{a}, \wedge, 1][1, A, 2]\ \mathrm{a\ b} \Longrightarrow_r^3 \mathrm{abab}. \end{aligned}$$

Now fix attention on the viable prefix $\gamma = [0, \mathrm{a}, \wedge, 1][1, A, 2]$. The second displayed line of the first derivation shows that the item $[2, \$, \wedge, 3] \to \bullet$ is valid for γ. But the line

$$\Longrightarrow_r [0, \mathrm{a}, \wedge, 1][1, A, 2][2, \mathrm{a}, \wedge, 1]\ \mathrm{b}$$

shows that the item $[2, \mathrm{a}, \wedge, 1] \to \bullet\mathrm{a}$ is also valid for γ. Hence G_1 is not an LR(0) grammar, though changing the last production in (0) to $[2, \$, \wedge, 3] \to \$$ makes it so. But using the LR(1) item $[2, \$, \wedge, 3] \to \bullet$ (on \$) solves the problem without introducing \$ into the language.

9.1. Some final remarks about LR(k) grammars

As remarked before, our choice of formalizing items as having single lookahead strings does not lead to the most efficient parsers if programmed literally. For one thing, it is natural to save space by grouping all lookahead strings z such that $A \to \alpha \bullet \beta$ (on z) are valid for the same γ into a single entry. There are other efficiency tricks, including so-called LALR parsing, which are described in some other texts, including Aho and Ullman's "Dragon Book" on compilers. My main interest in this line has been to come back to the question:

What is a deterministic context-free grammar?

The simple example $S \to \mathrm{a}S\mathrm{a}|\mathrm{b}S\mathrm{b}|\epsilon$, which generates the non-DCFL $\{xx^R : x \in \{a, b\}^*\}$ and is easy to parse, says intuitively that the answer "an LR(k) grammar" is too restrictive. This last grammar is unambiguous, and this suggests the answer "an unambiguous grammar." Note that $\{\mathrm{a}^i\mathrm{b}^j\mathrm{a}^k\mathrm{b}^l : i = l \vee j = k\}$ is an example of a CFL which does not have an unambiguous grammar, so this answer would still be a nontrivial restriction on the notion of a context-free *language*. But there are unambiguous CFGs which are considered difficult to parse because their parse trees grow exponentially in the size of terminal strings.

What I've had in mind is to develop a notion of CFGs that can be parsed by a *two-way* DPDA. A 2DPDA on input x has an input tape which holds $\wedge x\$$, and is allowed to make left moves of its input head. 2DPDAs can accept non-CFLs; for instance, a 2DPDA can accept $\{0^n1^n2^n\}$ by first comparing 0s against 1s, making left moves until reading a 0, and then comparing 1s against 2s. The special reason for interest in 2DPDAs is that even though an individual 2DPDA D can run for exponential time, there is an efficient general method due to Cook for simulating D by a RAM algorithm which runs in *linear* time. Oddly enough, Cook's procedure is not known to run in linear time when ported to a Turing machine. If time allows, I'll demonstrate Cook's procedure in two class hours. Two-way DPDAs begin to hit the level of power where, as with P and NP, we know hardly anything.

I am interested in whether one can formalize conditions on CFGs G which allow parsing by 2DPDAs. One might call such a parser a "shift-reduce-review" parser. Any of you who sees a good way to start doing this is welcome to pursue it as a research topic with me.

10. Rational Relations

[Lectures 18–21, scribed by D. Sivakumar.]

Let $\rho \subseteq \Sigma^* \times \Gamma^*$ be a binary relation on strings. Define the language $L_\rho \subseteq \Sigma^* \times \Gamma^*$ to be $\{\langle x,y\rangle \mid \rho(x,y) \text{ holds}\}$, where $\langle\cdot,\cdot\rangle$ denotes the standard pairing function given by $\langle x,y\rangle = \textsc{DoubleBits}(x)\cdot 10\cdot \textsc{DoubleBits}(y)$.
(*Note*: For the special case when $\Sigma = \Gamma = \{0,1\}$, if we need $\langle\cdot,\cdot\rangle$ to be one-to-one *and* onto, then we can use the pairing function described in [Reg88]. This pairing function, due to KWR, is computable by a deterministic finite-state transducer (hence in linear time) and is one-to-one and onto.)

Definition 10.1. Let $A \subseteq \Sigma^*, B \subseteq \Gamma^*$ be any two languages. We say that A *ρ-redues* to B, denoted $A \leq_\rho B$, if for all $x \in \Sigma^*$,

$$x \in A \iff (\exists y \in B)\,(\rho(x,y)).$$

Given a language B and a relation ρ, there is a unique A such that $A \leq_\rho B$, namely, $A = \rho^{\leftharpoondown}(B) := \{x \mid (\exists y \in B)\ \rho(x,y)\}$. If ρ is the graph of a total function $f : \Sigma^* \to \Gamma^*$, then $A \leq_\rho B \equiv (\forall x) x \in A \iff f(x) \in B$.

Definition 10.2. A relation ρ is said to be *$g(n)$-length bounded* if

$$(\forall x,y)\rho(x,y) \implies |y| \leq g(|x|).$$

Examples.

1. $\rho(x,y) \equiv$ "x is a prefix of y". Then for any B, $\rho^{\leftharpoondown}(B) \equiv \mathit{Prefs}(B)$.
2. For every partial multivalued function f, define the relation $\rho_f(x,y) \iff f(x) \mapsto y$, that is, $y \in$ *set-$f(x)$*, or, "y is a value of $f(x)$". Conversely, given ρ, we can define the partial multivalued function $f_\rho(x) \mapsto y \iff \rho(x,y)$. With these in hand, we can define the complexity classes NPMV and NPMV$_g$ as follows:

$$\begin{aligned} \text{NPMV} &= \{f_\rho : \rho \text{ is polynomial length bounded and } L_\rho \in \text{NP}\}, \\ \text{and NPMV}_g &= \{f_\rho : \rho \text{ is polynomial length bounded and } L_\rho \in \text{P}\}. \end{aligned}$$

We will now give the formal definition of rational relations on $\Sigma_1^* \times \Sigma_2^* \times \ldots \times \Sigma_k^*$.
Definition 10.3. A tuple $(a_1, a_2, \ldots, a_k) \in \Sigma_1^* \times \Sigma_2^* \times \ldots \times \Sigma_k^*$ is called a *singleton* if for all $i, 1 \leq i \leq k :\ a_i \in \Sigma_i \cup \{\epsilon\}$.

Definition 10.4. The *rational expressions* that describe rational relations on $\Sigma_1^* \times \Sigma_2^* \times \ldots \times \Sigma_k^*$ are inductively defined as follows:

Basis.

(0) $\emptyset$ is a rational expression that denotes the empty relation $\emptyset$.

(1) Any singleton is a rational expression that denotes the relation with one tuple.

Induction.

(2) If ρ and σ are rational expressions, then $\rho \cup \sigma := \{\vec{x} \mid \vec{x} \in \rho \ \vee\ \vec{x} \in \sigma\}$ is a rational expression that denotes the union of the two relations.

(3) If ρ and σ are rational expressions, then $\rho\cdot\sigma := \{\vec{x} \mid (\exists \vec{y} \in \rho)(\exists \vec{z} \in \sigma)\ \vec{x} = \vec{y}\cdot\vec{z}\}$ is a rational expression that denotes the concatenation of the two relations.

(4) If ρ and σ are rational expressions, then $\rho^* := \{\vec{\epsilon}\} \cup \{\vec{x} \mid (\exists n \geq 1)\ \vec{x} \in \rho_n\}$, where $\vec{\epsilon}$ is the all-ϵ k-tuple, is a rational expression that denotes the Kleene closure of the relation ρ.

Observe that $\emptyset$ is the identity for the union operation and ϵ is the identity for the concatenation operation.

Example. The rational expression $\rho := ((a,a)\cup(b,b))^*\cdot((\epsilon,a)\cup(\epsilon,b))^*$ denotes the relation $\{\langle x,y\rangle \mid x \sqsubseteq y\}$. Therefore, for any set $B \in \{a,b\}^*$, there exists a unique set $A \in \{a,b\}^*$ such that $A \leq_\rho B$, and this is just the set $A = \mathit{Prefs}(B)$.

Definition 10.5. A rational relation ρ is said to be *free* if there are regular expressions α, β such that $(\forall x,y)\ (x,y) \in \rho \iff x \in L(\alpha)\ \wedge\ y \in L(\beta)$.

Proposition 10.1. *If ρ is a free rational expression, then $L_\rho =_{\text{def}} \{x\#y \mid (x,y) \in \rho\}$ is regular.*

The converse of Proposition (10.1) is not true, however. For example, the rational relation $\rho := \{(a,a),(b,b)\}$ is not free, but L_ρ is finite and, therefore, regular. Another example is the rational relation for substrings, namely, $\pi := ((\epsilon,a)\cup(\epsilon,b))^*((a,a)\cup(b,b))^*((\epsilon,a)\cup(\epsilon,b))^*$. The language $L_\pi = L_{pat} = \{\langle x,y\rangle \mid x \text{ is a substring of } y\}$ cannot even be accepted by a k-head DFA for any fixed k.

Rational Relations and GSM$_\epsilon$ Mappings

Definition 10.6. A *transition graph with output (TGO)* is the directed graph of a finite automaton $M = (Q, \Sigma, \Gamma, \delta, s, F)$ where Q is a set of states that label the vertices of the graph, $s \in Q$ is a designated start state, $F \subseteq Q$ is a set of accepting states, Σ and Γ are some alphabets, $\delta \subseteq Q \times \mathit{RatRel}(\Sigma\times\Gamma) \times Q$, and where each arc is labelled by a rational relation $\rho \in \mathit{RatRel}(\Sigma\times\Gamma)$.

Definition 10.7. The *relation computed by M* is defined by

$$\begin{aligned} \rho_M \ =_{\text{def}}\ \{(x,y) \mid\ & (\exists q_0,\ldots,q_m \in Q, q_0 = s, q_m \in F) \\ & (\exists x_1,\ldots,x_m \in \Sigma^*)(\exists y_1,\ldots,y_m \in \Gamma^*)\ x = x_1\cdot x_2\ldots x_m, y = y_1\cdot y_2\ldots y_m \text{ and} \\ & (\forall i : 1 \leq i \leq m)(x_i,y_i) \in \rho_i \text{ where } (q_{i-1},\rho_i,q_i) \in \delta\}. \end{aligned}$$

Consider the following TGO:

This is a rational relation that is *not* a generalized homomorphism. To see this, consider a string of the form $\ldots 000\mathbf{11}\mathbf{0}10\ldots$; we cannot infer what the TGO produces on the boldfaced substring "10", without the knowledge about which of "00" or "11" precedes it.

We will now show a normal form of a TGO, and then proceed to show the equivalence of TGO's and rational relations. The idea behind the use of the normal form is that we can convert any TGO into one that is in the normal form and, consequently, it suffices to exhibit the rational relation for TGO's in the normal form. The normal form for TGO's is given by the following:

Lemma 10.2. *The rational relation corresponding to a TGO M in its normal form shown above is given by $\rho_M = (\alpha \cup \beta\gamma^*\eta)^* \cdot \beta\gamma^*$.*

Proof. By inspection. ☐

Lemma 10.3. *For any TGO M, we can find an equivalent TGO M' such that $s' = 1$ and $F' = \{f\}$ where f is numbered 2. The TGO M' is called special.*

Proof. Given M, add a new final state f, number f 2 and add arcs $(f', (\epsilon, \epsilon), f)$ for all original final states f' of M. ☐

Theorem 10.4. *For any TGO M, ρ_M is a rational relation.*

Proof. By Lemma 10.3, we may assume wlog. that the TGO M is special. Define the numerical predicate $P(n, m) \equiv$ "for all special M such that $|Q| = n$ and such that there do not exist an m' and a relation ρ which satisfy $m < m' < n \ \wedge \ (m', \rho, n) \in \delta$, ρ_M is rational". We prove the theorem by induction on n and m. The basic idea is that if m is the highest numbered node other than n which has an in-arc to node n, we can eliminate the arc from m to n and apply the induction hypothesis.

Basis ($n = 2, m = 1$). Follows from Lemma 10.2.

Induction ($n \geq 2$). Assume, as induction hypothesis (IH), that for all $n' < n$ and $m' < n'$, $P(n', m')$ holds, and that for all $m' < m$, $P(n, m')$ holds. We need to show that $P(n, m)$ is true.

Given a special TGO M with $|Q| = n$, let m be the highest numbered node with an in-arc to n. Note that $n \neq f$ and $n \neq s$, because f is numbered 2, s is numbered 1 and $n \geq 3$. For every out-neighbor q of n ($q = m$ included), consider the diagram:

Any computation that uses the arc (m, β, n) comes in from m and exits n via some other arc. Simultaneously for every q which is an out-neighbor of n, do the following:

- Replace α by $\alpha \cup \beta\gamma^*\eta$.
- Delete the arc (m, β, n).
- If (m, β, n) was the last arc into node n, delete node n. Now, $|Q| = n - 1$, and by IH $P(n - 1, any)$ is true; therefore, $P(n, m)$ follows. If (m, β, n) was not the last arc into node n, repeat the procedure for the next highest-numbered arc. Again, by IH, $P(n, m - 1)$ is true; therefore $P(n, m)$ follows. ☐

Definition 10.8. A GSM_ϵ is the special case of TGO in which every label ρ on an arc is a singleton or a finite union of singletons.

Theorem 10.5. *For every rational relation ρ, there is a GSM_ϵ M such that $\rho = \rho_M$.*

Proof. (By induction on the structure of rational relations.)

Basis.

- $\rho = \emptyset, M = \emptyset$.
- ρ is a singleton. M is one of the following four TGO's.

Induction.

- $\rho \cup \sigma$:

- $\rho \cdot \sigma$:

- $(\rho)^*$:

$\square$

Theorem 10.6. *Let ρ be a rational relation on $\Sigma^* \times \Gamma^*$. Then we can find an alphabet Δ, a regular set $R \subseteq \Delta^*$ and homomorphisms $h : \Delta \to \Gamma^*, t : \Delta \to \Sigma^*$ such that for all sets $B \subseteq \Gamma^*, \rho^-(B) = t(h^{-1}(B) \cap R)$.*

Proof. By Theorem 10.5, there is a special GSM$_\epsilon$ M such that $\rho = \rho_M$. Let $M = (Q, \Sigma, \Gamma, \delta, s, \{ f \})$ where $\delta \subseteq Q \times (\Sigma \cup \{ \epsilon \}) \times (\Gamma \cup \{ \epsilon \}) \times Q$.

Let $\Delta = \{ [p, a, b, q] \in \delta \}$ be the alphabet of "dominoes". Let $h : \Delta \to \Gamma^*$ be the erasing homomorphism defined as $h([p, a, b, q]) = b$, and let $t : \Delta \to \Sigma^*$ be the erasing homomorphism defined as $t([p, a, b, q]) = a$.

Let $R := \{\vec{c} \in \Delta^* |\ \vec{c}$ is a sequence of dominoes in δ which "match" like dominoes, such that the first domino begins with s, and the last domino ends with $f\}$.

Lemma 10.7. *R is regular.*

Proof. By inspection. $\square$

We need to show that for all x, $x \in \rho^- \iff x \in t(h^{-1}(B) \cap R)$.
For all x:

$$\begin{array}{lll}
x & \in & t(h^{-1}(B) \cap R) \\
& \iff & (\exists y \in \Delta^*)(t(y) = x \ \wedge\ y \in R \ \wedge\ y \in h^{-1}(B)) \\
& \iff & (\exists y \in \Delta^*)(\exists z \in B)(t(y) = x \ \wedge\ y \in R \ \wedge\ h(y) = z) \\
& \iff & (\exists y \in \Delta^*)(\exists z \in B)(t(y) = x \ \wedge\ h(y) = z \ \wedge\ y \\
& & \quad \text{is a valid computation sequence of } M(x) \text{ which gives output } z) \\
& \iff & (\exists z \in B)(x, z) \in \rho_M \\
& \equiv & x \in \rho^-(B).
\end{array}$$

This completes the proof. $\square$

Definition 10.9. For any $A \subseteq \Sigma^*$ and relation ρ on $\Sigma^* \times \Gamma^*$, define the *forward reduction by ρ* to be $\rho(A) =_{\text{def}} \{ y \mid (\exists x \in A)(x, y) \in \rho \}$.

Theorem 10.8. *For every rational relation ρ on $\Sigma^* \times \Gamma^*$, there exists an alphabet Δ and regular set $R \subseteq \Delta^*$ and homomorphisms $h : \Delta \to \Gamma^*, t : \Delta \to \Sigma^*$ such that for all sets $A \subseteq \Sigma^*, \rho(A) = h(t^{-1}(A) \cap R)$.*

Proof. Same R, same t, same h, same proof as Theorem 10.6!
We need to show that for all z, $z \in \rho(A) \iff z \in h(t^{-1}(A) \cap R)$.
For all z:

$$\begin{array}{lll}
z & \in & h(t^{-1}(A) \cap R) \\
& \iff & (\exists y \in \Delta^*)(h(y) = z \ \wedge\ y \in R \ \wedge\ y \in t^{-1}(A)) \\
& \iff & (\exists y \in \Delta^*)(\exists x \in A)(h(y) = z \ \wedge\ y \in R \ \wedge\ t(y) = x)
\end{array}$$

$$\begin{aligned}
&\iff (\exists y \in \Delta^*)(\exists x \in A)(h(y) = z \ \wedge \ t(y) = x \ \wedge \ y \\
&\qquad \text{is a valid computation sequence of } M(x) \text{ which gives output } z) \\
&\iff (\exists x \in A)(x, z) \in \rho_M \\
&\equiv \quad z \in \rho(A).
\end{aligned}$$

This completes the proof. □

Observations.

(1) If M makes no output on ϵ moves, then t is 1:1.

(2) If M never erases an input symbol, then h is 1:1.

(3) If M is a nondeterministic Mealy machine, both h and t are 1:1.

(4) If we want $\Delta = \Sigma = \Gamma = \{0, 1\}$, then t and h can be $r : 1$ generalized homomorphisms, where $r = 4\lceil\log_2 |Q|\rceil + 18$ is the number of symbols needed to encode $[0^k, 00, 00, 0^k]$ using the tupling function.

11. The Chomsky-Schützenberger Representation Theorem

[Lectures 22–23, scribed by Burhans and Govindarajan.]

Definition 11.1. Let Γ_n be the alphabet $\{ [_1, [_2, \ldots, [_n,]_1,]_2, \ldots,]_n \}$. The Dyck set of order n, is the language of the grammar:

$$G_n : S \rightarrow SS \mid \epsilon \mid [_1 \ S \ ]_1 \mid [_2 \ S \ ]_2 \mid \ \ldots \ \mid [_n \ S \ ]_n$$

Note that the grammar G_n is ambiguous but the language L$(G_n) = D_n$ is a DCFL.

Question: Which of the following grammars are LR(1)?

1. $S \rightarrow \epsilon \mid [_i S]_i S$
2. $S \rightarrow \epsilon \mid S[_i S]_i$

Theorem 11.1. *For every CFL L, over some alphabet Σ, we can find an $n > 0$, an alphabet Δ, a regular set $R \subseteq \Delta^*$ and homomorphisms*

1. $h : \Delta \rightarrow \Gamma_n^*$.
2. $t : \Delta \rightarrow \Sigma^*$

such that $L = t(h^{-1}(D_n) \cap R)$.

Proof. Suppose for the moment that $\epsilon \notin L$. Let G be a CFG in Chomsky Normal Form for L. Take n to be the number of type $V \rightarrow VV$ productions in G. Let $\Delta := \Sigma \cup \Gamma_n$, and homomorphisms h and t are defined as follows:

$$h(x) = \begin{cases} \epsilon & \text{if} x \in \Sigma \\ x & \text{otherwise} \end{cases}$$

$$t(x) = \begin{cases} \epsilon & \text{if } x \in \Gamma_n \\ x & \text{otherwise} \end{cases}$$

Number the type $V \rightarrow VV$ productions in G by i := 1,..., n. Let G_R be the CFG defined by:

- For each production $A \rightarrow BC$ in G, with number i, and each terminal production $D \rightarrow a$, $a \in \Sigma$ let G_R have

$$A \rightarrow [_i B \ \mid \ D \rightarrow a \ \mid \ D \rightarrow a]_i C$$

G_R is a right-linear grammar, so

R := L(G_R) is regular.

Note: $\epsilon \in D_n$, therefore, if $\epsilon \in L$, we first write the grammar G for L $\setminus \{\epsilon\}$ and at the end define $R' =$ R $\cup \{\epsilon\}$.

We need to prove: L(G) $= t(h^{-1}(D_n) \cap L(G_R))$

L(G) $\subseteq t(h^{-1}(D_n) \cap L(G_R))$ part: Let the induction hypothesis P(m) be the following: P(m) $\equiv$ For any derivation A $\Rightarrow^*$ x in G of m or fewer steps, there is a derivation in G_R of a string y such that t(y) = x, y $\in h^{-1}(D_n)$ and y ends in a symbol a $\in \Sigma$. Clearly

$$\forall m P(m) \Longrightarrow L(G) \subseteq t(h^{-1}(D_n) \cap L(G_R))$$

Basis (m=1): If A $\Rightarrow_G$ a, then since $\epsilon \in D_n$, a $\in h^{-1}(D_n)$ and G_R^A has A $\rightarrow$ a, therefore, a $\in L(G_R^A)$. (Note: G_R^A is G_R with A as the start symbol)

Induction (m $\geq$ 2): Let A $\Rightarrow_G$ BC $\Rightarrow_G{}^*$ x.
Let v, w be such that B $\Rightarrow_G^*$ v, C $\Rightarrow_G^*$ w, and vw = x.
By IH applied to B, $\exists$ a string y_v such that B $\Rightarrow_G{}^* y_v$, and t(y_v) = v, and y_v ends in a terminal symbol a. Then in G_R, there is the leftmost derivation A $\Rightarrow$ $[_i B \ \Rightarrow^* \ [_i uD \Rightarrow [_i ua = y_v$. Now do the derivation A $\Rightarrow^*$ $[_i uD \ \Rightarrow [_i ua]_i C$. By IH applied to C, $\exists$ a string y_w such that C $\Rightarrow_{G_R}^* y_w$ and t(y_w) = w, then y = $y_v.y_w$ is the desired string in L(G_R). Since y_v and $y_w \in h^{-1}(D_n)$, so is y.

L(G) $\supseteq t(h^{-1}(D_n) \cap L(G_R))$ part : Let the induction hypothesis P(m) be the following: P(m) $\equiv$ For every derivation A $\Rightarrow^*$ y in G_R such that h(y) $\in D_n$ in m or fewer steps, t(y) $\in$ L(G). Clearly

$$\forall m P(m) \Longrightarrow L(G) \supseteq t(h^{-1}(D_n) \cap L(G_R))$$

Basis (m=1) : y $\in \Sigma$, so this is true because G and G_R have the same terminal productions.

Induction (m $\geq$ 2) : Let A $\Rightarrow^m_{G_R}$ y, then this derivation starts with some non-terminal production of the form A $\Rightarrow_{G_R}$ $[_i$B and y begins with $[_i$. Note that it can't begin with A $\to$ a$]_i$C, because h(y) $\in D_n$ and can't begin with $]_i$. Since h(y) $\in D_n$, there is a matching $]_i$ in y. That is, we can write y = $[_i v]_i w$ such that h(v) $\in D_n$ and h(w) $\in D_n$.

The production which produced the $]_i$ in the derivation of y must have had the form D $\to$ a$]_i$C, where G has the production A $\to$ BC, and D $\to$ a is in both grammars. Therefore

$$B \Rightarrow^{m'}_{G_R} v, \quad m' < m \text{ and } C \Rightarrow^{m''}_{G_R} w, \quad m'' < m$$

By IH, B $\Rightarrow^*_G$ t(v) and C $\Rightarrow^*_G$ t(w). Since t is a homomorphism which erases symbols in Γ_n, in particular $[_i$ and $]_i$, t(y) = t(v).t(w). Thus we have A $\Rightarrow_G$ BC $\Rightarrow^*_G$ t(v).C $\Rightarrow^*_G$ t(v).t(w) = t(y). ■

11.1. Restating and Refining Chomsky-Schützenberger

Theorem 11.2. *For all CFLs L over Σ we can find n, $\Delta = \Gamma_n \cup \Sigma$, a regular set $R \subseteq \Delta^*$ and homomorphisms $t : \Delta \to \Sigma^*$, $h : \Delta \to \Gamma_n$, such that $L = t(h^{-1}(D_n) \cap R)$.*

Fact: With a different R, h and Δ, one can make t a 1:1 homomorphism. [Ronald V. Book, 1975, unpublished?]

Refinement 1: Suppose $\Sigma = \{c_1, \ldots, c_m\}$, then define Γ' = $\Gamma_n \cup \{ [_{n+1}, \ldots, [_{n+m},]_{n+1}, \ldots,]_{n+m} \}$. Modify G_R by replacing each terminal $c_j \in \Sigma$, by $[_{n+j}]_{n+j}$ to get $G_{R'}$ where R' = L ("new" G_R). Let t' erase Γ_n and map $[_{n+j} \to \epsilon$, $]_{n+j} \to c_j$. Therefore

$$L(G) = t'(D_{n+m} \cap R').$$

Refinement 2: Take $\Delta = \{ (,), [,] \}$.
Recode $[_i \;\equiv\; (^{i-1}[(^{n+m-i}$.
Recode $]_i \;\equiv\;)^{n+m-i}])^{i-1}$.
Adjust t' to t'' accordingly. t'' will erase any block with a [, and it will erase)...)])...) if] doesn't come in the first m places. If] comes in the j^{th} place, with j < m, then t'' maps block to c_j. So

$$L(G) = t''(D_2 \cap R'').$$

Research questions:

1. Prove that if the size of the alphabet Σ is large enough, then there does not exist a homomorphism t''' so that for all context free grammars G, L(G) = $t'''(D_2 \cap R''')$.
2. Does this hold when $\Sigma = \{0, 1\}$?

Refinement 3: With a larger Δ again, we can find homomorphisms h : $\Delta \to \{ (,), [,] \}^*$, t : $\Delta \to \Sigma^*$ and L(G) = t($h^{-1}(D_2 \cap R)$), (R $\subseteq \Delta^*$) $\in \mathcal{REG}$.

Therefore $\mathcal{CFL} \equiv$ the closure of $\{D_2\}$ under h^{-1}, h [5] and $\cap_R$.

Challenge questions: Define AFL(A) := closure of $\{ \Sigma^*, \phi, A \}$ under h, h^{-1} and $\cap_R$. We know AFL(D_2) = $\mathcal{CFL}$.

1. What is AFL($\{0^n 1^n \mid n \geq 0 \}$)?
2. What is AFL(D_1)?

12. Rational Relations Versus Regular Languages

[Lectures 24–25, scribed by Burhans and Govindarajan.]

Given a language L $\subseteq \Sigma^*$, define the relation $\sim_L$ as follows:

$$\forall x, y \in \Sigma^*, \quad x \sim_L y \;\text{ if }\; \forall z \in \Sigma^*(x.z \in L \Leftrightarrow y.z \in L).$$

For any equivalence relation $\sim$ on a set S and any D $\subseteq$ S, define the index of $\sim$ on D to be the number of elements in D which fall into different equivalence classes.

- D *shatters* $\sim$ if no two distinct elements in D are equivalent.
- Let $N_L(n) \equiv$ Index of $\sim_L$ on $\Sigma^{\leq n}$.

Theorem 12.1 (Myhill-Nerode Theorem). *L is regular if and only if $\sim_L$ has finite index on Σ^*, i.e. $N_L(n)$ is bounded as n $\to \infty$.*

Definition 12.1. Given a relation ρ on $\Sigma_1^* \times \ldots \times \Sigma_k^*$ define $(x_1, x_2, \ldots, x_k) \sim_\rho (y_1, y_2, \ldots, y_k)$ if $\forall z_1 \in \Sigma_1^*,\; z_2 \in \Sigma_2^*, \ldots,\; z_k \in \Sigma_k^*$.

$$\rho(x_1.z_1, x_2.z_2, \ldots, x_k.z_k) \;\Leftrightarrow\; \rho(y_1.z_1, y_2.z_2, \ldots, y_k.z_k)$$

A "k-ary" Myhill-Nerode theorem would read :

$$\forall \text{ relations } \rho,\ \rho \text{ is rational} \Leftrightarrow \sim_\rho \text{ has finite index on } \Sigma_1^* \times \ldots \times \Sigma_k^*.$$

$\Rightarrow$ is false: Let ρ be $\{(x_1\ ,\ x_2) \ :\ x_1 \sqsubseteq x_2\}$.
D = $\{(a\ ,\ a^n b) : n \geq 1\}$. D is infinite and D shatters $\sim_\rho$.
Let $(x_1\ ,\ x_2) = (a\ ,\ a^m b)$ and $(y_1\ ,\ y_2) = (a\ ,\ a^n b)$ and m < n. Take z = $(a^{n-1}\ ,\ a^{n-1})$. Clearly $(x_1.z_1\ ,\ x_2.z_2) \notin \rho$ and $(y_1.z_1\ ,\ y_2.z_2) \in \rho$.

[5] With $h^{1:1}$ in place of h too.

But $\Leftarrow$ is true! If $\sim_\rho$ has finite index, build a k-ary transition graph N.

- The states of N correspond to the equivalence classes of $\sim_\rho$.
- $F_N = \{[(x_1, x_2, \ldots, x_k)] \mid \rho(x_1, x_2, \ldots, x_k) holds\}$.
- Transitions are on singletons $(a_1, \ldots, a_k)$, $a_j \in \Sigma_j$ or $a_j = \epsilon$, $\delta([(x_1, \ldots, x_k)], (a_1, \ldots, a_k)) = [(x_1.a_1, \ldots, x_k.a_k)]$

Note: This is not a k-tape DFA because N can make stationary moves. We can however make N special by adding a state f with $(\epsilon, \ldots, \epsilon)$ arcs. ¿From this we can use the algorithm to construct the corresponding rational expression for ρ.

Definition 12.2. A rational relation $\rho(x_1, \ldots, x_k)$ is said to be length preserving if $|x_1| = \ldots = |x_k|$.

Research question: Characterize rational relations of finite index. Being length preserving is sufficient for a rational relation to have finite index, but is not necessary.

Note: The definition of accepting computation forces the k-tape NFA to read all of its input.

Theorem 12.2. *ρ is a k-ary rational relation if and only if there is a k-tape* NFA$_\epsilon$ *N such that N accepts ρ.*

Proof. Same as k=2 case. ∎

Theorem 12.3. *If $(\epsilon, \ldots, \epsilon) \notin \rho$ then ρ is rational if and only if there is a k-tape* NFA$_\epsilon$ *N that accepts ρ with no arcs labeled $(\epsilon, \ldots, \epsilon)$.*

12.1. A Pumping Lemma for Rational Relations

Lemma 12.4 (Pumping Lemma for Regular Languages). *If L is regular then there exists $n > 0$ such that for all $x \in L$ with $|x| \geq n$, there exist $u, v, w \in \Sigma^*$ such that $x = uvw$, $|uv| \leq n$, $v \neq \epsilon$, and for all $c \geq 0$, $uv^cw \in L$.*

Lemma 12.5 (Rational Pumping Lemma). *If ρ is rational, then there exists $n > 0$ such that for all $(x_1, \ldots, x_k) \in \rho$, if $(\exists i : |x_i| \geq n)$, then there exist k-tuples $(u_1, \ldots, u_k)$,*

$(v_1, \ldots, v_k$, and $(w_1, \ldots, w_k)$ such that for each i, $x_i = u_iv_iw_i \ \wedge \ |u_iv_i| \leq n$, and for all $c \geq 0$, $(u_1v_1^cw_1, \ldots, u_kv_k^cw_k) \in \rho$, and $(v_1, \ldots, v_k) \neq (\epsilon, \ldots, \epsilon)$.

Proof. The proof is the same proof by induction on ρ that was done for k=2.

As ρ is rational, there is a k-tape NFA$_\epsilon$ M [6] such that $\rho = \rho_M$.

Let n = number of states in M, let $\overline{c}$ be some accepting computation of M on $(x_1, \ldots, x_k)$. Some x_i has length $\geq$ n and every arc label is a singleton, therefore there must exist n transitions $\overline{c}$ of the form

$$(p_j, (a_{j_1}, \ldots, a_{j_k}), q_j) \text{ such that } a_{j_i} \neq \epsilon$$

Let S = $\{s\} \cup \{q_1, \ldots, q_j, \ldots, q'_n\}$ be the set of states encountered after reading a character in x_i where $n' = |x_i| \geq n$. Since $\|S\| \geq n+1$, there must be two repeated states. ∎

Example: Let ρ = { (p,t) : p is not a substring of t}.
Given n, let $(x_1, x_2) = (a^{n+1}, a^nba^n)$. Let $\bar{x} = \bar{u}\bar{v}\bar{w}$, $(v_1, v_2) \neq (\epsilon, \epsilon)$. By $|u_2v_2| \leq n$, v_2 comes to the left of b. There are two possibilities:

1. If $v_1 \neq \epsilon$, pump down to get $(a^m, \_ba^n)$, $m < n+1$, so $m \leq n$. a^m is a substring of $\_ba^n$. $\_$ stands for an arbitrary string of a's.
2. If $v_1 = \epsilon$ then $v_2 \neq \epsilon$, so pump up to get (a^{n+1}, a^Nba^n) with $N \geq (n+1)$.

Corollary 12.6. *Rational relations are not closed under complement.*

Question: Can one pump each component of ρ separately? In which case the pumping lemma reads as follows:

Lemma 12.7 (Modified Rational Pumping Lemma). *If ρ is rational, then $(\exists\ n > 0)$ such that $\forall (x_1, \ldots, x_k)$ such that $(x_1, \ldots, x_k) \in \rho$, $\exists i$: $|x_i| \geq n$, $\exists (u_1, \ldots, u_k, v_1, \ldots, v_k, w_1, \ldots, w_k)$ such that for each i $x_i = u_iv_iw_i \wedge$ for each i $|u_iv_i| \leq n \wedge (\forall (c_1, \ldots, c_k) > (0, \ldots, 0))$
$(u_1v_1^{c_1}w_1, \ldots, u_kv_k^{c_k}w_k) \in \rho \wedge (v_1, \ldots, v_k) \neq (\epsilon, \ldots, \epsilon)$.*

Answer: No, the rational relation $\rho = (a, b)^+$ is an example.

Exercises: Prove that complements of length preserving rational relations are also length preserving rational relations.

Footnote to the Pumping Lemma proof : Find a step m such that for some i, $|a_{i1}, a_{i2}, \ldots, a_{im}|$ = n and $\forall j \neq i$ $|a_{j1}, a_{j2}, \ldots, a_{jm}| \leq m$. m is the step at which some head has read n symbols on its tape.

If $\rho(x_1, \ldots, x_k)$ holds, and there exists some i such that $|x_i| \geq n$, then there exists an m which is the first step at which some head moved off cell n on its tape.

[6] M has no arcs labeled $(\epsilon, \ldots, \epsilon)$ except from the start state s to the final state f if $(\epsilon, \ldots, \epsilon) \in \rho$.

13. Descriptive Complexity

[Lecture 26, scribed by Boyd.]

There are three main ways to represent a language in a formal system.

1. Strings as ground objects: strings are like natural numbers. A formula Ψ with one free variable x of type string (or natural number).

$$L(\Psi) = \{x : \Psi(x) \text{is true}\}$$

2. Strings as `Array[0..n-1] of Char`, with predefined relations

$$Z(i) = \begin{cases} 1 & \text{if char } i = 1 \\ 0 & \text{if not.} \end{cases}$$

or, for alphabets Σ other than $\{0,1\}$,

$$Z_c(i) = \begin{cases} \text{true} & \text{if char } i = c \\ \text{false} & \text{if not.} \end{cases}$$

and possibly other relations — (Seeing strings as relational structures) – or a sentence σ with constants $0, 1, N$ and variables i, j of type `index into string`, and later, `index into index into string`.

$$L(\sigma) = \{x : x \models \sigma\}$$

(Read that as σ is true for x with $N := |x|$. "$\models$" (`\models`) is used to mean "satisfies".

3. Strings as bit-strings. $x = x_1 \cdot x_2 \cdots x_n$ Represent by a boolean formula Ψ with n free variables $x_1 \ldots x_n$ of type $\{0,1\}$ (or more generally of type Σ)

$$L_\Psi = \{x | \Psi(x_1, \ldots x_n) = \text{true, where } x = x_1 \ldots x_n\}$$

A boolean formula (over $\neg$, & , or) is the special case of a circuit over ($\neg$, or , &) where each gate has fan-out 1. (The lecturer sketched some examples:)

$$\phi_3 = ((x_1 \wedge x_2) \vee (x_2 \wedge x_3)) \wedge ((\neg(x_1 \wedge x_2)) \vee (\neg(x_2 \wedge x_3))) \qquad L(\phi_3) = \{110, 011\}$$

Problem: to define a *language*, you need one boolean formula for each input length; it's a non-uniform description of a language. One needs a family $\{\Phi_n | n = 0, 1, 2, \ldots\}$ of formulas.

(1) and (2) are uniform – only one object is needed to specify L for all lengths.

Example 13.1.

$$\sigma : \exists i : \neg Z(i) \wedge (\forall j : j \neq i \Longrightarrow Z(j)$$

$$L_\sigma = 1 * 01 * .$$

Implicitly, j, i range over $\{0, \ldots, N-1\}$. We are saying that some x_i equals 0 and that all other places are 1.

Alternatively (this is preferred):

$$\sigma : \exists i : Z(i) = 0 \wedge (\forall j, j = i \vee Z(j) = 1)$$

Predicates used: $Z(\cdot)$, $=$.

Example 13.2.

$$\tau = \exists i : Z(i) = 0 \wedge (\forall j : ((j < i) \Longrightarrow Z(j) = 0) \wedge ((j > i) \Longrightarrow Z(j) = 1))$$

$$L_\tau = 0 * 01$$

Predicates: Z, $=$, $<$. $=$ and $<$ can be defined in terms of $\leq$ thusly: $(i = j) \equiv (i \leq j) \wedge (j \leq i)$.

Example 13.3.

$$\rho : \exists k : \forall i, j[(2k = N \wedge i \leq k \wedge k < j \wedge \mathrm{NDIFF}(i,j)) \Longrightarrow (Z(i) = 1 \iff Z(j) = 1)]$$

where $\mathrm{NDIFF}(i,j) \equiv (j - i = N)$.

$$L_\rho = \{x : |x| \text{ is even } \forall i, x_i = x_{\lfloor \frac{n}{2} \rfloor + i} \equiv ww \text{ for } w \in \Sigma^*\}$$

Here, we used NDIFF like a subtraction function.

Claim: as long as we stay in the area delineatedby σ and τ, we get a subset of regular languages. With $\mathrm{BIT}(i,c)$ (the cth bit of i is 1), we can do anything a TM can do.

14. Operations on Rational Relations

[Lectures 27–28, scribed by Boyd and Smolowitz.]

A formula $\Psi(x_1, \ldots x_k)$ with k free variables of type `STRING` defines a k-ary relation $\rho_\Psi = \{(x_i, \ldots x_k) | \Psi(x_1, \ldots x_k)$ holds.$\}$. It also defines the language $\Theta_\Psi = \{\langle x_1 \ldots x_k \rangle | \Psi(x_i, \ldots x_k)$ holds.$\}$.

Here are some operations on k-ary relations $\rho \subseteq (\Sigma^* \times \Sigma^* \times \cdots \Sigma^*)$ (all the Σs are the same alphabet):

14.1. Set operations

Boolean $(\cup, \cap, \sim)$, which are equivalent to logical $(\vee, \wedge, \neg)$.

14.2. Explicit Transformations

14.2.1. Substitution

Let σ be an m-ary relation. Let $k > 0$, let $t : \{1 \ldots m\} \mapsto \{1 \ldots k\}$ ($m \leq k$, $m > k$ are both ok).

Define

$$\sigma_t = \{(x_1, \ldots x_k) | \rho(x_{t(1)}, \ldots x_{t(m)}) \text{ holds}\}.$$

(In logic, this would be substitution of variables for variables.)

14.2.2. Instantiation

Let σ be $(k+1)$-ary, let w be any fixed string. Define

$$\sigma_w = \{(x_1, \ldots x_k) | (x_1, \ldots, x_k, w) \in \sigma\}$$

(In logic, this is substituting a fixed value for a variable.)

Proposition 14.1. *The rational relations are not closed under explicit transformation.*

Example 14.1. Let $\Sigma = \{a, b\}$, and define

$$\rho = \{(x, y, z) | x \sqsubseteq y \text{ and } z \text{ is a suffix of } y\}$$

ρ is rational. For the rational expression, consider cases (i) (x, z) overlap in y (ii) they don't.

$$\rho = \overbrace{((a,a,\epsilon) \cup (b,b,\epsilon))^*}^{x \sqsubseteq y} \overbrace{((a,a,a) \cup (b,b,b))^*}^{\text{overlap}} \overbrace{((\epsilon,a,a) \cup (\epsilon,b,b))}^{\text{suffix}}$$

$$\cup \quad ((a,a,\epsilon) \cup (b,b,\epsilon))^* \underbrace{((\epsilon,a,\epsilon) \cup (\epsilon,b,\epsilon))^*}_{\text{no overlap}} ((\epsilon,a,a) \cup (\epsilon,b,b))$$

But now let $t(1) = 1$, $t(2) = 2$, $t(3) = 1$. $\rho_t = \{(x, y, x) \in \rho\}$. Then ρ_t is the set of all pairs (x, y) where x is a prefix of y and x is a suffix of y.

This is not a rational relation. Imagine a 3TNFA N for ρ, and try to map it over to a 2TNFA N' for ρ_t. Intuitively, N' needs a third head on the x tape, (so it can scan part of x twice in case it needs to match it twice with different parts of y).

14.2.3. A Taxonomy of Explicit Transformations

Explicit transformations can be broken down into:

1. **Permutation of k coordinates** It suffices to consider the following two transformations, as together they generate the entire symmetric group on k items:

$$\begin{aligned} \rho_{(1,2)} &= \{x_2, x_1, x_3, \ldots, x_k) | (x_1, \ldots x_k) \in \rho\} \\ \rho_{(1\ldots k)} &= \{(x_2, x_3, \ldots, x_k, x_1) | (x_1, \ldots x_k) \in \rho\} \end{aligned}$$

   Proposition 14.2. *The rational relations are closed under permutation of coordinates.*

   Proof. Relabel the tapes of the KHNFAs. ■

2. **Adding a non-quantified variable**

$$\rho' = \{x_1, \ldots x_k, x_{k+1}) | (x_1, \ldots x_k) \in \rho\}$$

   Proposition 14.3. *If ρ is rational, then so is ρ'.*

   Proof. N' reads and ignores x_{k+1}. (N' still has to guess where x_k ends.) ■

3. **Instantiating a variable x_{k+1} to a fixed string w.**

   Proposition 14.4. *Ditto.*

   Proof. Ditto. ■

4. **Identifying two variables**

$$\rho'' = \{(x_1, \ldots x_{k-1}) | (x_1, \ldots, x_{k-1}, x_{k-1}) \in \rho\}$$

Proposition 14.5. *Every explicit transformation under 14.2 is a finite sequence of operations (1, 2, 3, 4).*

Proposition 14.6. *Operation 4 is the one under which rational relations are not closed.*

Proof. Reductio ad absurdum: They aren't closed; they are closed under (1, 2, 3), so (4) is the one. ■

On the homework, (1, 2, 3) are the "innocuous" explicit transformations.

14.3. Unbounded Existential Quantification

$E(\rho)$ is the $(k-1)$-ary relation $E(\rho) := \{(x_2, \ldots x_k) | (\exists x_1 \in \Sigma^*)(x_1, \ldots x_k) \in \rho\}$. This could also be written as $\{(x_1, \ldots x_{k-1}) | (\exists x_k \in \Sigma^*)(x_1, \ldots x_k) \in \rho\}$. This is a very powerful operation.

In a sense, it could be thought of as instantaneously atomically searching the entire space of possible computations of a DGSM. – DB

14.4. Bounded Existential Quantification

Let $S(n)$ be any positive, real-valued function. Then $E_S(\rho) = \{(x_1, \ldots, x_{k-1}, z) | (\exists x_k : |x_k| \leq S(|z|)\,(x_1, \ldots x_k) \in \rho\}$. E_S is k-ary, but z is used only for its length.

Definition 14.1. The *rudimentary* relations are all those relations defined from the rational relations by

1. Booleans
2. Explicit Transformation
3. Bounded Existential Quantification where $S(n) \in O(n)$. (*i.e.* linear-length-bounded existential quantification)

The following are equivalent:

- Saying $(\exists x) : |x| \leq c|z|$ for strings
- Saying $(\exists x) : x \leq (2^{|z|})^c$ where $z = 2^k - 1$

These are essentially like "polynomial magnitude-bounded quantification".

Also polynomial-length-bounded quantification $(\exists x)|x| < |z|^c$ is equivalent to $2^{\text{polylog}(x)}$-magnitude-bounded quantification.

If $S(n) = c \log n$, then there are at most $|z|^c$ many strings x such that $|x| \leq c \log(|z|)$. A parallel RAM with n^c processors can try them all in parallel, in unit time.

15. Aside

Suppose R is any language. Define $\iota_R = \{(x, x) | x \in R\}$. If R is regular, then ι_R is rational. (Exercise.)

For any language L, $L \cap R = \iota_R(L) = \iota_R^{-1}(L)$ because

$$\begin{aligned} \iota_R(L) &= \{y | \exists x \in L, (x, y) \in \iota_R\} &= L \cap R \\ \iota_R(L)^{-1} &= \{x | \exists y \in L, (x, y) \in \iota_R\} &= L \cap R \end{aligned}$$

In the notation of Harju and Kleijn [HK93], $\mathcal{M}$ (the *marking* function) stands for the operation of concatenating \$.

If $\mathcal{C}$ is closed under h^{-1}, H and is nonempty, is $\{\$\} \in \mathcal{C}$?

Consider the rational relation $\tau = ((a, \epsilon) \cup (b, \epsilon))^*(\epsilon, \$)$. $\tau(L) = \{\$\}$. Can τ be expressed using h, h^{-1}?

Harju and Kleijn [HK93] show that for regular R, $\iota_R \in \mathcal{H}^{-1} \cdot \mathcal{H} \cdot \mathcal{H}^{-1} \cdot \mathcal{M}$. (Boyd and Smolowitz attempted to show that $\iota_R \in \mathcal{H} \cdot \mathcal{H}^{-1} \cdot \mathcal{M}$.)

Is $\iota_R \in \mathcal{GH} \cdot \mathcal{H}^{-1} \cdot \mathcal{M}$? Can the gh be made total with $[d:e]$ fixed delay?

$h^{-1}(L)$ is always defined as a language. *Watchword*: where papers refer to "linear bounded erasing", look for a fixed-delay gh.

16. Examples of Rudimentary Languages

[Lecture 29, scribed by Boyd and Smolowitz.]

Let $L = \{x \in \{a, b\}^* : \#a(x) = \#b(x)\}$. L is non-regular, but it is rudimentary. Here is how we determine it.

Define two homomorphisms: $h_1(a) = a$, $h_1(b) = \epsilon$; $h_2(a) = \epsilon$, $h_2(b) = a$. Then $\forall x \in \Sigma^*, x \in L \Leftrightarrow h_1(x) = h_2(x)$.

Define $\rho_1 := \{(x, y) : h_1(x) = y\}$ and $\rho_2 := \{(x, y) : h_2(x) = y\}$. Then $\forall x \in \Sigma^*, x \in L \Leftrightarrow (\exists y)\,[(x, y) \in \rho_1 \wedge (x, y) \in \rho_2]$.

ρ_1 and ρ_2 are both rational, so they are both rudimentary. To get L to be rudimentary, we need "$(\exists y)$" to be linear length-bounded. Note that if $h_1(x) = y$ or $h_2(x) = y$, then $|y| \leq |x|$, so adding the length constraint won't change the truth value.

Thus we may formally define

$$x \in L \leftrightarrow (\exists y : |y| \leq |x|)\,[(x, y) \in \rho_1 \cap \rho_2].$$

Use the formal definition of a relation σ obtained by length-bounded existential quantification from $\rho_1 \cap \rho_2$.

Let $\sigma(x, z) \leftrightarrow (\exists y : |y| \leq |z|)\,[(x, y) \in \rho_1 \cap \rho_2]$. ($z$ is a place-holder for σ, but sets the length bound for y.) Finally we get to L by an existential quantification of type (2d), i.e. identifying two variables:

$$x \in L \leftrightarrow (x, x) \in \sigma.$$

Since $\rho_1, \rho_2 \in \text{RUD}$, $\rho_1 \cap \rho_2 \in \text{RUD}$, and $\sigma \in \text{RUD}$, we have $L \in \text{RUD}$.

By similar means, this language is also in RUD:

$$B = \{x \in \{(,),[,]\}^* : \# \text{ of [}\text{'s in } x = \# \text{ of]'s in } x \wedge \# \text{ of ('s in } x = \# \text{ of)'s in } x\}.$$

Lemma 16.1. *If $L \in$ RUD and h is a homomorphism, then $h^{-1}(L) \in$ RUD.*

Proof. For all x, $x \in h^{-1}(L) \leftrightarrow (\exists y)\,[y \in L \wedge h(x) = y]$.
Let $c := \max\{|h(a)| : a \in \Sigma\}$. (Homomorphisms always have a finite amount of expansion that they can do.) Then whenever $h(x) = y$, necessarily $|y| \le c|x|$. Hence $\forall x, x \in h^{-1}(L) \leftrightarrow (\exists y : |y| \le c|x|)\,[y \in L \wedge h(x) = y]$.

Let $\tau(x,y) \leftrightarrow y \in L \wedge h(x) = y$.

Let $\sigma = \{(x,z) : (\exists y : |y| \le c|z|)\,\tau(x,y)\}$.

In τ's definition, "$y \in L$" is rudimentary, and "$h(x) = y$" is rational. To be completely pedantic, let $\lambda = \{(x,y) : y \in L\}$, adding an unidentified variable by operation (2b).

Thus $\sigma = \lambda \cap \rho_h$ = rudimentary $\cap$ rational, and so σ is rudimentary. Therefore $h^{-1}(L)$ is rudimentary. ■

Q: Are rudimentary languages preserved under *forward* homomorphism?
A: $y \in h(L) \leftrightarrow (\exists x)\,[x \in L \wedge h(x) = y]$.

This would be rudimentary if we could find a constant c and guarantee that $|x| \le c|y|$. But this might not hold true, if h can perform erasing. We will see that the answer to this question is "no."

Thus RUD is closed under length-preserving homomorphisms; under $d : e$ gh's; with fixed d & e (exercise); or even $d : e$ rational relations.

Recall that $\Gamma_2 = \{(,),[,]\}$.

Define
$H_) = \{x \in \Gamma_2^* :$ the shortest nonempty prefix of x which belongs to B ends in $)\}$
and $H_] = \{x \in \Gamma_2^* :$ the shortest nonempty balanced prefix of x ends in $]\}$.

Lemma 16.2. *$\forall x, x \in D_2 \Leftrightarrow$ ($x \in B$; for all ways of writing $x = u(v$, $(v \in H_)$; and for all ways of writing $x = u[v$, $[v \in H_]$).*

Proof.
Let $\pi(x) \leftrightarrow x \in B \wedge (\forall u, v \in \Gamma_2^*)\,\{\{x = u(v \rightarrow (v \in H_)\} \wedge \{x = u[v \rightarrow [v \in H_]\}\}$. (Note: curly braces were used to avoid confusion with the characters of Γ_2 that appear in the expression.)

Most of the expression for $\pi(x)$ is clearly rudimentary:

- $x \in B$ is certainly rudimentary, because $B \in$ RUD.
- The boolean operators $\wedge$ and $\rightarrow$ are rudimentary.
- Without loss of generality, we can restrict u and v to have length $|x|$ or less.
- $x = u(v$ is an instance of the concatenation operation, which is rudimentary, applied twice. The expressions $(v$, $u[v$, and $[v$ are similar.

- Note that $(\forall x : |x| \le |z|)\, Q(x) \equiv \sim (\exists x : |x| \le |z|) \sim Q(x)$. Hence length-bounded quantification is a composition of three rudimentary operations.

It remains to show that $H_)$ and $H_]$ are rudimentary. They have similar structure, so we will do it for $H_]$ only, using the definition of B. For all x, we have:

$$\begin{aligned} x \in H_] \;\leftrightarrow\;& (\exists y : |y| \le |x|)\,[[\, y \ne \epsilon \wedge y \sqsubseteq x \wedge y \in B \wedge y \text{ ends in ']'}\,] \\ & \wedge (\forall z : |z| \le |y|)\,[\,(z \ne \epsilon \wedge z \sqsubseteq y \wedge z \in B) \rightarrow z = y\,]]. \end{aligned}$$

The z part says that there is no shorter prefix than y which is nonempty and balanced.

Every constituent relation is rational, so $H_]$ is rudimentary. Thus $\pi(x)$ is rudimentary. Since $\pi(x)$ is equivalent to $x \in D_2$, D_2 is rudimentary as well. ■

Note: Technically, to complete the definition of D_2, we need to say not "$x \in H_]$", but actually "$[v \in H_]$".

Define $\rho(v) \equiv [v \in H_]$. Then $\rho(v) \leftrightarrow (\exists w : |w| \le 2|v|)\,[\,w = [v \wedge w \in H_]\,]$.

Research problem: Can one get every rudimentary language with $c = 1$? Or is there somewhere you'll always need to say "$\exists w : |w| \le |v| + 1$"?

17. RUD and CFL

[Lecture 30, scribed by Boyd and Smolowitz.]

Recall: The class RUD of rudimentary languages is closed under h^{-1}. Also, if $L \in$ RUD, then $h(L) \in$ RUD, *provided* that $(\exists c > 0)\,(\forall x \in L)\,|h(x)| \ge |x|/c$.

h is a homomorphism; it can "wipe out" strings $w \notin L$ so long as it is restrained on $x \in L$ (i.e. h leaves a constant fraction, or amount of evidence, unerased). So

$$y \in h(L) \leftrightarrow (\exists x : |x| \le c|y|)\,(h(x) = y \;\wedge\; x \in L).$$

Note that the "$\exists x$" part is linear length-bounded; the remaining relations are rudimentary. Thus $h(L)$ is rudimentary if h obeys the above condition.

Recall KWR's proof of: For all CFL's L, $\exists n, t, h, R : L = t(h^{-1}(D_n) \cap R)$. The homomorphism t was not length-preserving, but it had the property that $\forall z \in h^{-1}(D_n) \cap R, |t(z)| \ge |z|/3$.

$R = L(G_R)$, with the grammars G and G_R described as follows:

<u>G</u>

- Productions have the form $A \to BC$.
- To derive y, with $|y| = m$, use $m-1$ variable productions and m terminal productions.

<u>G_R</u>

- The corresponding productions have the form $A \to (_i B$ and $D \to)_i C$.
- This grammar gives you $m-1$ $(_i$'s and $m-1$ $)_i$'s.
- $t(z) = y$.
- $|z| \le 2(m-1) + m \le 3m = 3|y|$.

Moreover, when we had $L = t'(h_2^{-1}(D_2 \cap R'))$, then $\forall z \in h_2^{-1}(D_2 \cap R'), |t(z)| \ge |z|/3k$, where k = number of pure variable productions in the given CFG G for L. So k does not depend on $|z|$.

By KWR's inclusion of all rational relations in the rudimentary relations, $R \in$ RUD. Also, $D_2 \in$ RUD. Therefore $h_2^{-1}(D_2)$, $h_2^{-1}(D_2) \cap R$, and $t(h_2^{-1}(D_2) \cap R)$ are all in RUD. Using the Chomsky - Schützenbeger Representation Theorem, we have proved:

Theorem 17.1. *Every CFL is rudimentary.*

Proposition 17.2. *Every finite intersection of CFL's is rudimentary.*

Thus there are rudimentary non-CFL's, such as $\{ww \mid w \in \Sigma^*\}$.

18. RUD and the Arithmetic Hierarchy

Question: Where is RUD in the Arithmetic Hierarchy?

Proposition 18.1. *Every language $L \in$ RUD is a DCSL; i.e. L belongs to DSPACE[$O(n)$].*
Strengthened version: For every rudimentary relation ρ, $\Theta_\rho \in$ DSPACE[$O(n)$], where $\Theta_\rho = \{< x_1, \dots, x_k > \mid \rho(x_1, \dots, x_k)$ holds $\}$.

Proof. We will do structural induction on the way ρ is built up.
Basis. For every rational relation ρ, there is a constant C_ρ and a Turing machine M_ρ such that $L(M_\rho) = \Theta_\rho$ and M_ρ runs in space $C_\rho n$. This is true with $C_\rho \le 3$.

M_ρ has $2k$ tracks. On odd tracks i' for the ith tape, M_ρ maintains the current location of head h_i. M_ρ remembers the state of the k-head NFA in its finite control.

Let N be the k-tape NFA such that N accepts ρ. Without loss of generality, some head of N advances for each step. Also, N has at most two branches at each step. Hence every computation of $N(x_1, \dots, x_k)$ runs for at most $n := |x_1| + |x_2| + \dots + |x_k|$ steps.

M_ρ is deterministic, and keeps one more tape with n cells to cycle through all $w \in \{0,1\}^n$ which determine the different computation paths by N. Then $M_\rho(< x_1, \dots, x_k >)$ halts and accepts iff it finds an accepting computation of N.
Note: Life here is much easier if we use concat(x, y, z) as the only basis predicate.

Induction. The cases are: $\rho = \sigma \cup \tau$; $\rho = \sigma \cup \tau$; $\rho = \sim \sigma$; explicit transformations; and $\rho = \{< x_2, \dots, x_k, z > \mid \exists x_1 : |x_1| \le z, (x_1, \dots, x_k) \in \sigma\}$.

■

Therefore rudimentary relations are recursive. In the next two lectures, we will state and prove Wrathall's Theorem about the class RUD, namely, that RUD equals the linear time hierarchy.

19. The Linear Time Hierarchy

[Lectures 31–32, scribed by Sivakumar, including Debra Burhans' notes for part of lecture 31.]

Given a binary relation ρ and a length bound $S(n)$, we can write

$$E_S(\rho) = \{\, x \mid (\exists y : |y| \le S(|x|)\ \rho(x, y)\,\}.$$

Technically, $E_S(\rho)$ contains ϵ iff ρ contains (ϵ, ϵ). For any class $\mathcal{C}$ of languages, define

$$E_{lin}[\mathcal{C}] := \{\, L \mid \text{ for some } \rho \text{ with } \theta_\rho \in \mathcal{C}, \text{ and } c > 0, L = E_{cn}(\rho)\,\}.$$

Now define $\Sigma_k^{lin} 0 = \Pi_k^{lin} 0$ = DLIN, the class of languages accepted in linear time by deterministic multitape TMs, and for $k \ge 1$, $\Sigma_k^{lin} k = E_{lin}\left[\Pi_k^{lin} k - 1\right], \Pi_k^{lin} k = \text{co}-\Sigma_k^{lin} k$. Finally define

$$\text{LINH} = \bigcup_{k=1}^{\infty} \Sigma_k^{lin} k.$$

Proposition 19.1. *Let* NLIN *denote the class of languages accepted in linear time by* nondeterministic *multitape Turing Machines. Then* $\Sigma_k^{lin} 1$ = NLIN.

Proof. By definition, $\Sigma_k^{lin} 1 = E_{lin}$[DLIN]. Therefore, we will prove that E_{lin}(DLIN) = NLIN.

$\supseteq$:

Let L be a language in NLIN, and let N be a nondeterministic multitape Turing Machine such that $L(N) = L$. The idea is to simulate N by a nondeterministic multitape Turing Machine N' which behaves as follows: on input x, N' nondeterministically writes *in advance* all the guesses made by N on a fresh tape. Call this guess string y. Then N' begins to simulate N; whenever N makes a guess, N' consults the string y and continues its simulation of N. The crucial points to note are: (a) since N is a machine that runs in linear time, the length of the guess string y is at most $c|x|$ for some constant c, and (b) after writing the guess string y, the behavior of N' is completely deterministic. It is easy to see that N' accepts the language L. To wit,

$$\begin{aligned} x \in L(N') &\iff (\exists y : |y| \le c|x|)\ N' \text{ accepts } x \text{ with guess string } y \\ &\iff N \text{ accepts } x \text{ with the same guesses as prescribed by } y \\ &\iff x \in L(N). \end{aligned}$$

$\subseteq$:

Let L be a language in E_{lin}[DLIN] defined by a binary relation ρ and a constant $c > 0$ such that $x \in L \iff (\exists y : |y| \le c|x|)\ \rho(x, y)$ and $\theta_\rho \in$ DLIN. Let M be a deterministic linear time Turing Machine such that $L(M) = \theta_\rho$, that is, $(x, y) \in L(M)$ iff $|y| \le c|x| \wedge \rho(x, y)$. A nondeterministic Turing Machine N such that $L(N) = L$ behaves as follows: on input x, N guesses a string y of length $c|x|$ and simulates M on (x, y), and accepts iff M accepts. It is easy to see that $L(N) = L$. $\square$

Notation: Wrathall writes σ_k for $\Sigma_k^{lin} k$. We will use NLIN[$\mathcal{C}$] in place of $E_{lin}[\mathcal{C}]$. Wrathall's NL($\mathcal{C}$) is the same as our $\text{NLIN}^{\mathcal{C}}$, and denotes the class of languages accepted in nondeterministic linear time with an oracle set from the class $\mathcal{C}$. We will use LINH[$\mathcal{C}$] to denote NLIN[$\mathcal{C}$] $\cup$ NLIN[co$-$NLIN[$\mathcal{C}$]] $\cup$ NLIN[co$-$NLIN[co$-$NLIN[$\mathcal{C}$]]] $\cup \ldots$, etc.

Note that taking complements (co$-\mathcal{C} = \{\tilde{L} | L \in \mathcal{C}\}$) is a rudimentary operation, and E_{lin} = (identify x and z) $\circ$ (linear length bounded $\exists$). Hence

Proposition 19.2. *For any class $\mathcal{C}$, LINH[$\mathcal{C}$] is contained in the closure of $\mathcal{C}$ under rudimentary operations.*

Corollary 19.3. *If* DLIN $\subseteq$ RUD, *then* LINH $\subseteq$ RUD.

So to prove Wrathall's theorem, it suffices to show RUD $\subseteq$ LINH and DLIN $\subseteq$ RUD. For the second part, it turns out that it is easier to prove NLIN $\subseteq$ RUD. To answer the question we raised earlier about where RUD fits in the Chomsky hierarchy, we will prove a chain of results, one of which is RUD $\subseteq$ LINH. Our main conceptual tool in the chain of proofs is the alternating Turing Machine model of Chandra, Kozen and Stockmeyer (see "Structural Complexity Theory, Vol II" by Balcazar, Diaz and Gabarro, for a complete treatment of this concept). Let ATIME[$t(n)$] denote the class of languages accepted by alternating Turing Machines (to be defined) that run in time $t(n)$. Our task can now be clearly specified as follows:

(1) RUD $\subseteq$ LINH $\subseteq$ ATIME[$O(n)$] $\subseteq$ DSPACE[$O(n)$].

(2) NLIN $\subseteq$ RUD.

Since DSPACE[$O(n)$] is exactly the class of languages accepted by deterministic linear bounded automata, part (1) above lets us conclude that RUD = $\mathcal{CSL}$, the class of context-sensitive languages.

20. Alternating Turing Machines

An alternating TM M has Q, its set of states, partitioned into $Q_\exists \cup Q_\forall \cup Q_{\text{det}}$. Technically, Q_{det} is only optional, but we will include it for clarity of exposition. A configuration $I = [v(\,^q_c\,)w]$ is *existential* if $q \in Q_\exists$, and *universal* if $q \in Q_\forall$. Wlog. we may suppose that M has a unique accepting state q_{acc}, and a unique rejecting state q_{rej}, which are halting states, and it is immaterial which one of $Q_\exists$ and $Q_\forall$ they belong to. The transition function δ_M is defined as for an NTM, but the definition of acceptance is different. A halting configuration $[v(\,^{q_{\text{acc}}}_c\,)w]$ is *accepting*, and a halting configuration $[v(\,^{q_{\text{rej}}}_c\,)w]$ is *rejecting*. A non-halting configuration $[v(\,^q_c\,)w]$ with $q \in Q_\exists$ is *accepting* if at least one of the next configurations is accepting. A non-halting configuration $[v(\,^q_c\,)w]$ with $q \in Q_\forall$ is *accepting* if *all* of the next configurations are accepting. Finally, acceptance is defined by $x \in L(M) \iff [(\,^s_{x_1}\,)x_2 \ldots x_n]$ is accepting.

Observations.

1. Every NTM N is an ATM with $Q_\exists = Q_N$.
2. If $L \in$ NLIN, then $\tilde{L}$ is accepted by an ATM with only universal states in $O(n)$ time.

Definition 20.1. For any computation path $\vec{c}$, define $\text{ALT}(\vec{c})$ to be the number of alternations between $Q_\exists$ and $Q_\forall$ in $\vec{c}$.

If $s \in Q_\exists$ and for every computation path $\vec{c}$ of M, $\text{ALT}(\vec{c}) \le k$, then M is called a Σ_k machine. If $s \in Q_\forall$ and for every computation path $\vec{c}$ of M, $\text{ALT}(\vec{c}) \le k$, then M is called a Π_k machine.

Definition 20.2. The class Σ_k-TIME[$t(n)$] is defined to be the set of all languages L such that L is accepted by a Σ_k machine M. The class Π_k-TIME[$t(n)$] is defined to be the set of all languages L such that L is accepted by a Π_k machine M.

Definition 20.3. The class ATIME[$t(n)$] is defined to be the set of all languages L such that L is accepted by an ATM M which runs in time $O(t(n))$ on inputs of length n.

The following propositions are easy to prove.

Proposition 20.1. *For every k, Σ_k-TIME[$t(n)$] $\subseteq$ ATIME[$t(n)$], and the converse is not true.*

Proposition 20.2. *For every k, $\Sigma_k^{lin} k = \Sigma_k$-TIME[$O(n)$] and $\Pi_k^{lin} k = \Pi_k$-TIME[$O(n)$].*

Theorem 20.3. ATIME[$O(n)$] $\subseteq$ DSPACE[$O(n)$].

Proof. Let M be an ATM which runs in time cn on inputs of length n. We may assume wlog. that M has binary branching on its existential and universal states. We will build a deterministic TM M' that runs in DSPACE[$O(n)$] and simulates a left-to-right evaluation of the computation tree of M on any given input x. In addition to the work tapes of M, M' maintains an extra tape that contains pairs from the set $\{\langle\exists,0\rangle,\langle\exists,1\rangle,\langle\forall,0\rangle,\langle\forall,1\rangle\}$. M' performs a depth-first, left-to-right evaluation of the computation tree of M. At each point during the simulation, the extra tape of M' contains information about the states encountered in the current computation path of M. M' has a way of remembering in its internal control the following information:

1. whether it has already evaluated the left branch of a state
2. whether it can skip the right branch of a state

Whenever M' moves up the computation tree, it "replays" the computation sequence from the initial configuration upto and including the current node. M' accepts iff M accepts (according to the definition of acceptance for an ATM). Since M runs in time $O(n)$, the work tapes of M never grow beyond length $O(n)$. It remains, therefore, to prove that the extra tape of M' never grows beyond $O(n)$ length. This follows from the fact that no computation path of M is longer than $O(n)$ steps. □

Theorem 20.4. RUD $\subseteq$ LINH.

Proof. We will prove this by induction on the structure of rudimentary relations.

Basis.

(0) If ρ is rational, there is a k-tape NFA which accepts θ_ρ. Since a nondeterministic linear time TM can easily simulate a k-tape NFA for any fixed k, $\theta_\rho \in$ NLIN.

Induction.

(1a) $\rho = \sigma \cup \tau$. By induction hypothesis, there is a k such that $\theta_\sigma \in \Sigma_k^{lin} k \ \wedge \ \theta_\tau \in \Sigma_k^{lin} k$ via $\Sigma_k^{lin} k$ machines M_σ and M_τ, both of which have existential states s_σ and s_τ as their start states. Define M_ρ to be an ATM which begins with an existential state s with two branches—one leading to s_σ and the other to s_τ. Since M_σ and M_τ are both $\Sigma_k^{lin} k$ machines, all computation paths from s_σ and s_τ have at most k alternation of existential and universal states. Since s_σ and s_τ are both existential states, any path of M still has k or fewer alternation of existential and universal states. In other words, M_ρ is also a $\Sigma_k^{lin} k$ machine, and it is easy to see that $L(M_\rho) = \theta_\rho$.

(1b) $\rho = \sigma \cap \tau$. By induction hypothesis, there is a k such that $\theta_\sigma \in \Sigma_k^{lin} k \ \wedge \ \theta_\tau \in \Sigma_k^{lin} k$ via $\Sigma_k^{lin} k$ machines M_σ and M_τ, both of which have existential states s_σ and s_τ as their start states. Define M_ρ to be an ATM which begins with a universal state s with two branches—one leading to s_σ and the other to s_τ. Since M_σ and M_τ are both $\Sigma_k^{lin} k$ machines, all computation paths from s_σ and s_τ have at most k alternation of existential and universal states. Since s_σ and s_τ are both existential states, any path of M still has $k+1$ or fewer alternation of existential and universal states. In other words, M_ρ is a $\Pi_k^{lin} k$ machine, and it is easy to see that $L(M_\rho) = \theta_\rho$.

The above proof is a "lazy proof"—a less lazy proof would be to prove that $\Sigma_k^{lin} k$ is closed under intersection.

(1c) $\rho = \tilde{}\sigma$. By induction hypothesis, there is a k such that $\theta_\sigma \in \Sigma_k^{lin} k$ via a $\Sigma_k^{lin} k$ machines M_σ which has an existential state s_σ as its start state. By interchanging the accepting and rejecting states and swapping $Q_\exists$ with $Q_\forall$, we get a $\Pi_k^{lin} k$ machine M_ρ such that $L(M_\rho) = \theta_\rho$.

(2a) (Permutation of indices)
Wlog. we may suppose that $\rho(x_1, x_2, \ldots, x_k) = \sigma(x_2, x_1, \ldots, x_k)$. By appropriately relabelling the tapes of a Σ_k machine M_σ that accepts θ_σ, we can get a Σ_k machine M_ρ that accepts θ_ρ.

(2b) (Adding a dummy index) $\rho(x_1, x_2, \ldots, x_k) = \sigma(x_2, x_1, \ldots, x_{k-1})$. Obvious.

(2c) (Instantiate a component) $\rho(x_1, x_2, \ldots, x_k) = \sigma(x_2, x_1, \ldots, x_{k-1}, w)$. Modify M_σ to do the following: on input $(x_1, \ldots, x_{k-1})$, deterministically append w in time at most $2(|(x_1, \ldots, x_{k-1})| + |w|)$, and simulate M_σ.

(2d) (Identify two variables) Similar to instantiation, only this time M_σ is re-programmed to make an extra copy a component of its input.

(3) (Bounded existential quantification)
$\rho(x_1, x_2, \ldots, x_{k-1}, z) = (\exists x_k : |x_k| \leq d|z|)\sigma(x_1, x_2, \ldots, x_{k-1}, y)$. Given a Σ_k machine M_σ, we construct M_ρ as follows: on input $(x_1, x_2, \ldots, x_{k-1}, z)$, M_ρ existentially "generates" all strings y of length at most $d|z|$ and simulates M_σ on $(x_1, x_2, \ldots, x_{k-1}, y)$ on each of the leaves of the binary tree that generates all these strings. Observe that all nodes in this binary tree are existential states, and therefore, every computation path of M_ρ on $(x_1, x_2, \ldots, x_{k-1}, z)$ has no more than k alternations, that is, M_ρ is a Σ_k machine.

This completes the inductive step. By induction, we conclude that RUD $\subseteq$ LINH. □

Corollary 20.5. RUD $\subseteq$ LINH $\subseteq$ ATIME$[O(n)]$ $\subseteq$ DSPACE$[O(n)]$.

Corollary 20.6. *For any class $\mathcal{C}$, the closure of $\mathcal{C}$ under rudimentary operations contains* LINH$[\mathcal{C}]$.

Theorem 20.7. NLIN $\subseteq$ RUD.

Proof. Let N be an NTM which runs in time cn and accepts a language A. Let $\alpha = \{(x, y) \mid y \in \{L, R, S\}^*$ and y encodes the input head movements of $N(x)$, and $N(x)$ accepts $\}$. Let $A' = \theta_\alpha$. We claim that $A' \in$ NTIME$[n]$. An NTM for A' would first extract x from (x, y), then simulate $N(x)$ and verify that y correctly encodes the head movements of $N(x)$. Furthermore, N' is *real time*—it reads one character of the input at each step. Now,

$$\forall x : x \in A \iff (\exists y : |y| \leq c|x|)\ (x, y) \in \alpha.$$

In other words, $A' \in \text{RUD} \Longrightarrow A \in \text{RUD}$. Hence to prove the theorem, it suffices to prove that $A' \in RUD$.

Note that if N' has k work tapes, it can be simulated in real time by an NTM N'' which has $2k$ stacks. It follows from the lemma below that $L(N'') \in \text{RUD}$, completing the proof of Wrathall's theorem. $\square$

Lemma 20.8 ([GH69]). *There are $2k$-many CFL's $B_1, \ldots, B_{2k}$ over some alphabet Δ and a length-preserving homomorphism $h : \Delta \to \Sigma^*$ such that $L(N'') = h(B_1 \cap B_2 \cap \cdots B_{2k})$.*

Proof Sketch. Let Δ = dominoes of the form $[p, c_0, c_1, \ldots, c_{2k}, A_1, \ldots, A_{2k}, q] \in \delta_{N''}$. Assume that the input head always moves right and that N makes no ϵ-moves. The character $c_i, i \geq 1$ stands for the character on the i-th work tape of N'', c_0 stands for the character scanned on the input tape, and the A_i's stand for the action performed on the work tapes. Call a chain of dominoes *consistent on tape j* if they match and for any pair $[p, c_0, c_1, \ldots, c_{2k}, A_1, \ldots, A_{2k}, q], [q, d_0, d_1, \ldots, d_{2k}, A'_1, \ldots, A'_{2k}, r]$, d_j is what N'' sees on tape j after action A_j on c_j.

Define $B_j = \{$ chains consistent on tape j and which end in an accepting state $\}$, and define $h([p, c_0, c_1, \ldots, c_{2k}, A_1, \ldots, A_{2k}, q]) = c_0$. Clearly, each of the B_j's is a CFL since the tapes are stack-restricted. The lemma follows from the observation that $x \in L(N'') \iff x \in B_1 \wedge x \in B_2 \wedge \ldots \wedge x \in B_{2k}$. $\square$

21. Star-free Regular Languages

[Lectures 33–34, scribed by Sivakumar.]

Definition 21.1. A *star-free regular expression* is defined inductively by the following grammar:

Basis. $E \to \emptyset \mid \epsilon \mid c$ (for all $c \in \Sigma^*$).

Induction. $E \to (E \cup E) \mid (E \cap E) \mid (\tilde{}E) \mid (E \cdot E)$.

Any language $L(\alpha)$ of a star-free regular expression α is a *star-free regular language*.

Examples.

1. $\Sigma^* = \tilde{}\emptyset$.
2. $(c_1 \cup c_2 \cup \ldots \cup c_k)^* = \tilde{}((\Sigma^*) \cdot (c_{k+1} \cup \ldots \cup c_n)(\Sigma^*))$.
3. $(ab)^* = [\tilde{}(a \cup b \cup (\tilde{}\emptyset)aa(\tilde{}\emptyset) \cup (\tilde{}\emptyset)bb(\tilde{}\emptyset)) \cap ((a(\tilde{}\emptyset)b) \cup \epsilon)]$.

The basic idea is that we can't have regular expressions of the form $(aa)^*$ or $(aaa)^*$—in other words, *modular counting is disallowed*. Star-free regular expressions are the central subject of study for the monograph *Counter-free Automata* by McNaughton and Papert

(1971). In this monograph, McNaughton and Papert characterize the class of star-free regular languages in a number of interesting ways, two of which we will study. One of these characterizations, due to M.P.Schützenberger, is in terms of the *monoid of transformations* of the minimum-state DFA for the regular set. To understand this clearly, we provide below some definitions, which are standard in abstract algebra (see Birkhoff and Bartee (1970), or Ross and Wright (1991) for better treatments).

Definition 21.2. A *monoid* $(S, \circ)$ is a set S with a binary operation $\circ$ which satisfies

(1) *Associativity.* $(\forall x, y, z \in S)\ x \circ (y \circ z) = (x \circ y) \circ z$.

(2) *Identity.* $(\exists e \in S)\ (\forall x \in S)\ x \circ e = e \circ x = x$.

Definition 21.3. A *group* is a monoid $(S, \circ)$ which satisfies

(3) *Existence of inverses.* $(\forall x \in S)\ (\exists x^{-1} \in S)\ x \circ x^{-1} = x^{-1} \circ x = e$, where e is the identity element of the monoid.

Definition 21.4. A *sub-monoid* of a monoid $(S, \circ)$ is a subset $T \subseteq S$ which satisfies

(1) $e \in T$, where e denotes the identity of the monoid $(S, \circ)$.

(2) $(\forall x, y \in S)\ x, y \in T \Longrightarrow x \circ y \in T$.

Definition 21.5. A monoid $(S, \circ)$ is said to be *aperiodic* if no sub-monoid of $(S, \circ)$ forms a non-trivial group (the set $\{ e \}$ is referred to as the trivial group).

For any DFA M over an alphabet Σ, let ϕ_c denote the mapping of states induced by the character $c \in \Sigma$, that is, $\phi_c : Q \to Q$ is defined by $\phi_c(p) = q$ if $\delta_M(p, c) = q$. Let $\mathcal{M}$ denote the closure of the set $\{ \phi_c \mid c \in (\Sigma \cup \epsilon) \}$ under $\circ$ (composition of mappings). The following proposition is elementary.

Proposition 21.1. *$(\mathcal{M}, \circ)$ is a monoid (called the* monoid of transformations *of M).*

Finally, here is the promised theorem:

Theorem 21.2. *The following are equivalent:*

(1) A is a star-free regular language.

(2) A is definable in the McNaughton-Immerman formal system with $0, \mathbf{N}, Z(i), =, \leq$ only.

(3) The monoid of transformations $(\mathcal{M}, \circ)$ of the minimum-state DFA for A is aperiodic.

As an application of Theorem 21.2, we will now show that the regular language $L_1 = (aa)^*$ is not star-free. The following is a minimum-state DFA for L_1.

The monoid elements $\phi_\epsilon = \mathrm{ID}$ and $\phi_a = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}$ form a group that is isomorphic to S_2, the symmetric group on two elements.

Next consider the regular language $L_2 = (ab)^*$, whose minimum-state DFA is given above.

The monoid elements are as follows:

$$\phi_\epsilon = \mathrm{ID},$$

$$\phi_a = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 3 \end{pmatrix}, \phi_b = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 3 \end{pmatrix},$$

$$\phi_{ab} = \phi_b \circ \phi_a = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 3 \end{pmatrix}, \phi_{ba} = \phi_a \circ \phi_b = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 3 \end{pmatrix},$$

$$\phi_{aa} = \phi_a \circ \phi_a = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 3 & 3 \end{pmatrix} = d \text{ (the ``dead'' mapping)},$$

$$\phi_{bb} = \phi_b \circ \phi_b = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 3 & 3 \end{pmatrix} = d,$$

$$d \circ (any) = (any) \circ d = d,$$

$$\phi_{ab} \circ \phi_a = \phi_{ab} \circ \phi_b = \phi_b \circ \phi_{ab} = d,$$

$$\phi_a \circ \phi_{ba} = \phi_{ba} \circ \phi_b = d,$$

$$\phi_{ab} \circ \phi_{ba} = \phi_{ba} \circ \phi_{ab} = d,$$

$$\phi_a \circ \phi_{ab} = \phi_{ba} \circ \phi_a = \phi_a,$$

$$\phi_b \circ \phi_{ba} = \phi_{ab} \circ \phi_b = \phi_b.$$

Upon careful inspection, we conclude that the monoid of transformations listed above does not have any sub-monoid that forms a non-trivial group. Intuitively, the reason why $(ab)^*$ is star-free while $(aa)^*$ is not is that the latter attempts to "count" (modulo 2) the number of a's, while the former does not. By the same token, it is easy to observe that $(abab)^*$ is not star-free since it "counts" (modulo 2) the pattern ab.

Here are some more examples of regular languages that are not star-free:

1. $L_1 := \{ x \mid |x| \text{ is even} \}$.
2. $L_2 := \{ x \in \{0,1\}^* \mid x$ has exactly two 1's and the first 1 occurs in an odd position $\}$. (The set $\{ \mathrm{ID}, \phi_0 \}$ is isomorphic to S_2.)
3. $L_3 := L_2 \cap \{ x \mid$ the last 1 in x occurs in an even position $\}$. (Once again, $\{ \mathrm{ID}, \phi_0 \}$ is isomorphic to S_2.)

22. Closure properties of Star-free Languages

Let SF denote the class of all star-free regular languages. We will show that SF is not closed under 1:1 homomorphisms. To see this, consider the language $L_1 = (ab)^*$ over $\Sigma = \{ a, b \}$. We know that $L_1 \in \mathrm{SF}$; taking $h : \Sigma \to \Sigma^*$ to be the homomorphism defined by $h(a) = h(b) = a$, we get $L_2 = h(L_1)$ to be the language $(aa)^*$, which we know is not star-free.

Next we will prove a useful lemma about star-free languages that we will use later to characterize the monoid of transformations of star-free languages.

Lemma 22.1. *For any language $B \subseteq \Sigma^*$ and $c \in \Sigma$, let $B_c := \{ x \mid cx \in B \}$. If $B \in \mathrm{SF}$, then $B_c \in \mathrm{SF}$.*

Proof. We will prove this by induction on the structure of star-free regular expressions.

Basis.

(0) If B is either $\emptyset$ or ϵ, then B_c is $\emptyset$ or ϵ, respectively.

Induction.

(1) $B = B_1 \cup B_2$. $B_c = (B_1)_c \cup (B_2)_c$. By induction hypothesis applied to B_1 and B_2, and by the closure of SF under $\cup$, we conclude that $B_c \in \mathrm{SF}$.

(2) $B = B_1 \cap B_2$. $B_c = (B_1)_c \cap (B_2)_c$. By induction hypothesis applied to B_1 and B_2, and by the closure of SF under $\cap$, we conclude that $B_c \in \mathrm{SF}$.

(3) $B = D \cdot A$.

$$\begin{aligned} B_c &= \{ x \mid cx \in D \cdot A \} \\ &= \{ x \mid (\exists y, z)\ x = yz \ \wedge\ cy \in D \ \wedge\ z \in A \} \cup \{ x \mid \epsilon \in D \ \wedge\ cx \in A \} \end{aligned}$$

If $\epsilon \notin D$, then $B_c = D_c \cdot A$; if $\epsilon \in D$, then $B_c = D_c \cdot A \cup A_c$. Either way, by induction hypothesis applied to D and A, $B_c \in \mathrm{SF}$.

(4) $B = \tilde{}D$.

$$\begin{aligned} B_c &= \{ x \mid cx \notin D \} \\ &= \tilde{}\{ x \mid cx \in D \} \\ &= \tilde{}(D_c) \end{aligned}$$

By induction hypothesis applied to D and the closure of SF under complementation, we conclude that $B_c \in \mathrm{SF}$.

This completes the induction step, and by induction, we have that $B \in \mathrm{SF} \Longrightarrow B_c \in \mathrm{SF}$. □

22.1. An Instructive Example

[Lecture 35, scribed by Burhans and Govindarajan.]

Claim 22.2. *Let $L(M) = (0+1)(00)^*$. There is no *-free expression for L.*

Proof. Suppose $L \in$ SF, L_1 = { x | 1x ∈ L }. By the last theorem, $L_1 \in$ SF but L_1 = $(00)^*$ which is known not to be SF, so L_1's syntactic monoid is not aperiodic. ■

Theorem 22.3. *Let M be the minimum state DFA for a language L. Let $\mathcal{M}$ be the monoid of transformations of M. The states of M are in 1-1 correspondence with the equivalence classes under the relation $x \sim_1 y \equiv (\forall v, xv \in L \leftrightarrow yv \in L)$. And the elements of $\mathcal{M}$ are in 1-1 correspondence with the equivalence classes under $x \sim_2 y \equiv (\forall u, v) uxv \in L \leftrightarrow uyv \in L$.*

Idea behind proof: In one direction, suppose x,y are such that ∀ states p,

$$\delta^*_M(p, x) = \delta^*_M(p, y)$$

This means that ϕ_x and ϕ_y when interpreted as mapping states to states are in the same equivalence class of functions. In M in the above example, there are 4 equivalence classes under the relation $\sim_1$ corresponding to the four states of the minimum state DFA. But there are 6 equivalence classes under the relation $\sim_2$. These are:

1. ϕ_ϵ.
2. ϕ_0.
3. ϕ_1.
4. ϕ_{00}.
5. ϕ_{10}.
6. ϕ_{dead}, where *dead* is any string not in $10^* \cup 0^*$.

Choose the elements 0 and 00 and rename them as a and e, and look at the multiplication table restricted to a and e. It looks as follows.

1. aa = e.
2. ae = a.
3. ea = a.
4. ee = e.

Therefore the two elements a and e form a sub-group with the identity element being e.

		ϵ	0	1	00	10	dead
$[\begin{pmatrix}1&2&3&4\\1&2&3&4\end{pmatrix}]$	ϵ	ϵ	0	1	00	10	dead
$[\begin{pmatrix}1&2&3&4\\2&3&2&4\end{pmatrix}]$	0	0	00	dead	0	dead	dead
$[\begin{pmatrix}1&2&3&4\\2&4&4&4\end{pmatrix}]$	1	1	10	dead	1	dead	dead
$[\begin{pmatrix}1&2&3&4\\3&2&3&4\end{pmatrix}]$	00	00	0	dead	00	dead	dead
$[\begin{pmatrix}1&2&3&4\\3&4&4&4\end{pmatrix}]$	10	10	1	dead	10	dead	dead
$[\begin{pmatrix}1&2&3&4\\4&4&4&4\end{pmatrix}]$	dead	dead	dead	dead	dead	dead	dead

Table 2: Multiplication table of the equivalence classes under $\sim_2$

Eilenberg's condition: A monoid $\mathcal{M}$ contains no non-trivial group if and only if ∃ t such that $\forall a \in \mathcal{M},\ a^{t+1} = a^t$.

Corollary 22.4. *Every finite monoid $\mathcal{M}$ has the property that $\exists q \exists t \forall a \in \mathcal{M}\ \ a^{t+q} = a^t$.*

Corollary 22.5. *SF is not closed under forward 1:1 homomorphisms* [7]

Proof. Consider the forward 1:1 homomorphism: h(a) = a.
h(b) = a.
$h((ab)^*) = (aa)^*$. Therefore SF is not closed under forward 1:1 homomorphisms. ■

Research Problems:

1. What is the closure of SF under homomorphisms?
2. What is the closure of SF under d:e generalized homomorphisms.

Is the answer to any/both of the above two questions all regular languages?

Lemma 22.6. *Every language in SF can be generated by*

$$E \rightarrow \phi \mid \epsilon \mid c \mid (E \cup E) \mid (\sim E) \mid EdE$$

Proof. The only case that needs checking is C = AB, where A,B ∈ SF. But

$$B \;=\; [\bigcup_{c \in \Sigma} c.B_c], so$$

[7] As a matter of fact nor under inverse 1:1 homomorphisms.

$$A \cdot B \;=\; [\bigcup_{c\in\Sigma} A.c.B_{c\cdot}]$$

■

23. The Equivalence of SF and FOL

[Lectures 36–37, scribed by Burhans and Govindarajan.]

Definition 23.1. For strings $z_0, \ldots, z_k$ over Σ of the same length N, let $< z_0, \ldots, z_k >$ denote their shuffle, and let $[z_0, \ldots, z_k]$ denote their shuffle as a word of length N over the alphabet Σ^{k+1}.

$< z_0, \ldots, z_k >$ can be represented as a $k+1 \times N$ matrix as follows:

$$\begin{pmatrix} z_{00} & z_{01} & \cdots & z_{0N-1} \\ z_{10} & z_{11} & \cdots & z_{1N-1} \\ & & \vdots & \\ z_{k0} & z_{k1} & \cdots & z_{kN-1} \end{pmatrix}$$

$[z_0, \ldots, z_k]$ can then be thought of as the column vectors of this matrix:

$$[z_0, \ldots, z_k] = \begin{pmatrix} z_{00} \\ z_{10} \\ \vdots \\ z_{k0} \end{pmatrix} \begin{pmatrix} z_{01} \\ z_{11} \\ \vdots \\ z_{k1} \end{pmatrix} \cdots \begin{pmatrix} z_{0N-1} \\ z_{1N-1} \\ \vdots \\ z_{kN-1} \end{pmatrix}$$

Definition 23.2. Every formula ϕ in FOL (as in Barrington, McNaughton, et al) with free variables $\underline{i_1}, \ldots, \underline{i_k}$ defines the relation:

$$\{(x, i_1, \ldots, i_k) : x \models \phi(\underline{i_1}, \ldots, \underline{i_k})\}$$

Theorem 23.1. *If $A \in SF$ then the relation $substr_A = \{ < x, i, j >: x_i, \ldots, x_{j-1} \in A \}$ is definable by a FOL formula $\sigma_A(i,j)$, i.e., the SF languages are FO representable.*

Notes: The proof is based on Barrington's but has been simplified by making the following changes:

- Instead of the $prefix_A \equiv \phi$ (_,j) and $suffix_A \equiv \phi$ (j,_) relations, the $substr_A$ relation defined above is used.
- Binary numbers i are represented in unary notation as $0^{i-1}10^{N-1-i}$.

Note for purists: Since the following representations are equivalent, they are used interchangeably in the text:

- $i = 0 \equiv \forall i \;\; i \le j$
- $i = N \equiv \forall i \;\; j \le i$
- $i = 1 \equiv \forall j \;\; j = 0 \;\vee\; i \le j$
- $i \ne j \equiv \neg(i = j)$
- $i = j + 1 \equiv \forall k \;\; \neg((i < k) \;\wedge\; (k < j)) \;\wedge\; (i < j)$

Proof.
Proof is by induction on the structure of a SF expression.

Basis

	$\underline{\sigma_A}$	$\underline{\phi_A}$
$A = \emptyset$	$(i \ne i) \wedge (j \ne j)$	$(0 \ne 0)$
$A = \{\epsilon\}$	$i = j$	$N = 0$
$A = \{c\}$	$(j = i + 1) \;\wedge\; (z(i) = c)$	$(N = 1) \;\wedge$

Induction

	$\underline{\sigma_A}$	$\underline{\phi_A}$
$A = B \cup C$	$\sigma_B \vee \sigma_C$	$\phi_B \cup \phi_C$
$A = \sim B$	$\sim \sigma_B$	$\sim \phi_B$

Finally, for the case $A = B.C$ we need to define

$\{ < x, i, j >: x_i, \ldots, x_{j-1} \;\in\; B.C \} \;\Leftrightarrow\; (\exists k\; (x_i, \ldots, x_{k-1}) \in B \;\wedge\; (x_k, \ldots, x_{j-1}) \in C)$

So,

$$\begin{aligned} \sigma_A &:= \exists k\; \sigma_B(i,k) \wedge \sigma_C(k,j) \wedge (i \le k) \wedge (k \le j) \\ \phi_A &:= \exists k\; \sigma_B(0,k) \wedge \sigma_C(k,N) \wedge (0 \le k) \wedge (k \le N) \end{aligned}$$

Note: Numbering from 0 and using substr seems to avoid Barrington's use of "E → EdE"

■

Question: What is the effect of agglutination of alphabet characters in SF languages?

Note from KWR: Lecture 37, on Monday, December 6, was on the proof of FOL $\subseteq$ SF. I gave an outline of the proof. The main idea is: Given a formula ϕ of FOL, we may suppose ϕ is in *prenex normal form*; i.e., with a "quantifier block" followed by a "matrix" of atomic predicates connected by Boolean operations but without quantifiers. First let $u_1, \ldots, u_m$ be all the variables in the matrix, and work with the "agglutinated alphabet" $\Gamma_m = \{0, 1\}^{m+1}$. The first place corresponds to the string x. Values of variables in the other places are represented in *unary* notation; for instance, the number i corresponds to the string 0^i10^{n-1-i}. The first part, done in class, is to show that each of the atomic predicates $Z(u)$, $u = v$, and $u < v$, with the appropriate extensions using $m-1$ or m "dummy variables," is definable by a *-free expression over Γ_m. Hence so is the entire matrix, since Boolean operations are simulated directly by *-free operations. Now we have to add-in the quantifiers one-by-one, stepping down Γ_m to Γ_{m-1} each time. (We can order the variables so that the place quantified is always the last free place.) Because we have negation it suffices to consider an intermediate stage $\psi = (\exists u_k)\eta$, where η has free variables $u_1, \ldots, u_k$, and by the induction hypothesis, there is a *-free expression α for η over Γ_k. The goal is to find a *-free expression β for ψ over Γ_{k-1}. The method is to let M_α be the unique minimum-state DFA for α, and since we use unary encodings, to observe that every accepting computation by M_α has exactly one transition where it reads a '1' in the last place. The remainder of this proof is the first problem on the take-home final. Reason carefully about properties a *minimum-state* DFA must have. Be sure that in addition to proving the key "L_{pq} lemma," you explain how it fits into finishing this proof.

24. "Random-Access" TMs and Sub-Linear Time

[Lecture 38, prepared by Regan.]

A *random-access Turing machine* (RAM-TM) is just like an ordinary Turing machine with an input tape and some number $k \geq 2$ of worktapes, except that it has no input tape head. Instead, the first worktape (called tape 1) holds a "pointer" to the input tape, whose address is governed by the current contents of tape 1. We make the convention that tape 1 holds a binary number written with the *least* significant bit first, not counting a left endmarker $\wedge$ in cell 0 and a right endmarker \$ that may be present. (In class I said "...most significant bit first," but least is better, although if leading 0s are tolerated in addresses then it doesn't matter.) The syntax for transitions is exactly the same, and the only difference in semantics is that the input symbol c_0 is taken from the addressed input tape cell. Allender and Gore [AG91b] use the convention that tape 1 initially holds the length n of the input in binary notation (or since we number from 0, give it $n-1$). A natural restriction, which they seem not to adopt, is that the address tape must always hold a value between 0 and n. More popular is the convention that an "address out of range" returns '\$'; then it is possible to compute n deterministically in $O(\log n)$ time by binary search. There are also several differing conventions in how the input tape itslef may be accessed:

(1) The input tape may only be polled when the RAM-TM is in a special *input query state* $q_?$, whose formalism is exactly the same as for an oracle query.

(2) Same as (1), but with the draconian proviso that on transit from state $q_?$, the address tape is *entirely blanked out*.

(3) The input tape is polled in each transition, using whatever the address-tape holds even if M is in the middle of "loading" the intended address from another tape.

A RAM-TM may be deterministic or nondeterministic, or even *alternating*. Indeed, the first paper to define the random-access mechanism as such was titled "Alternation" [CKS81]. To define alternating RAM-TMs syntactically, all one needs to do is partition the state set Q into $Q_\exists$, $Q_\forall$, and Q_D; also, the deterministic part Q_D can be divided arbitrarily between $Q_\exists$ and $Q_\forall$ instead. The semantics is the same.

Example 24.1. There is a deterministic $O(\log n)$ time RAM-TM M such that *if* the input x belongs to $\{\, 0^m1^k : m, k \geq 0 \,\}$, then M can tell whether m is odd or even. Under the convention which allows "address out of range," the algorithm is to write 1, 11, 111, ... on the address tape until getting the '\$' response, and then begin a binary search which looks for the break between 0s and 1s. Note that if the input x is not in 0^*1^*, then this binary search can be deceived. It is easy to show that the *language* $A = \{\, 0^m1^k : m, k \geq 0, m \text{ is odd} \,\}$ cannot be decided by an ATM without looking at all n bits: picture an "Input Demon" which is always poised to foil you with a '1' or a '0' in the one bit of the input x that you didn't examine if you prematurely announce your decision about whether $x \in A$. (Properties which require examining all the bits are termed *elusive*, and there was some interest around 1980 in classifying exactly which graph properties are elusive—I believe there's a conjecture which hasn't been solved yet.)

In class I ditzed around with trying to do the binary search under the convention that $n-1$ is originally given and "addresses out of range" are *streng verboten*. This is a question for computational contortionists. The *real point* of this example was intended to be that having alternation quashes a lot of these difficulties. I sketched ATMs which decide both the language A and the language 0^*1^* itself.

An ATM M running under convention (3) can be looked at this way: The computation tree of M, which unfolds independently of the input under (3), is really a *formula*.[8] An existential node in the tree is a binary $\vee$ in the formula, and a universal node is an $\wedge$. After a given branch polls an input bit x_i at its leaf, it can only be one of four possible functions: accept regardless of x_i, reject regardless of x_i, accept iff $x_i = 1$, or accept iff $x_i = 0$. These correspond to the four possible entries in a formula: 1, 0, the variable x_i, or its negation $\neg x_i$, respectively. The definition of $L(M)$ for an ATM is the same as evaluating this formula on the particular assignment ot the variables which the actual input $x \in \{\, 0, 1 \,\}^*$ represents. The following lemma is helpful conceptually:

Lemma 24.1. *An ATM M which runs under convention (1) or (2) in time $t(n)$ and space $s(n)$ can be simulated by an ATM M' which runs under convention (3) in time $3t(n)$ and space $s(n)$.*

Proof. Assume M runs under convention (1). Then replace every state q by an existential state q_e, whose function is to "guess" the input bit. Give q_e two children q_0 and q_1 which are universal states and correspond to the two guesses. The left child of q_0 then polls the input and stops right away, accepting iff the bit is 0; likewise the left child of q_1 polls the input and accepts iff the bit is 1. The right child of q_0 simulates the original state

[8] Technically, two branches of M might come back together in the same ID, and the computation graph need not even be acyclic. But we can always give M an extra worktape on which it records the sequence of branchings, thus making every ID unique to its path and making the computation graph a tree. If the original M runs in $O(\log n)$ time and space, so does the new ATM.

q, but only includes the arcs of q defined for the input bit being 0. The right child of q_1 simulates the transitions of q for a '1' input. Whichever right child has its guess validated by its left sibling is the one that carries forth the corresponding branch of the original machine M. For every state q along the path, there is a detour through two new states, so the running time is tripled. The new machine M' needs no additional space.

If M runs under convention (2), then this construction only needs to be applied to state $q_?$. ■

Now recall the definition of Σ_k^{lin} as the class of languages accepted by linear-time ATMs with the property that every computation path on every input x begins with an existential state, and alternates at most $k-1$ times from existential to universal states or vice-versa. We can make the same stipulation on an ATM which runs in logarithmic time, and this defines the class $\Sigma_k^{\log}$. However, the question of which input convention is used needs to be addressed anew, because the above proof can introduce *many* new alternations. Happily, there is the following modification, which I've seen one-sentence allusions to, but have not seen worked out in detail in any sources I have:

Lemma 24.2. *An ATM M which runs under convention (1) or (2) in time $t(n)$ and space $s(n)$ and has the Σ_k alternation property can be simulated by an ATM M' which runs under convention (3) in time $3t(n)$ and space $2s(n) + t(n)$ and has the Σ_{k+1} alternation property.*

Proof. Here is how to modify the construction of the preceding lemma: If q is an existential state, then make q_e branch to two more existential states which simulate q, and have M' record along each branch the sequence of guessed input bits which it needs to check later. As soon as the original computation by M reaches a universal state q' of M, *or* reaches a halting state q', M' goes into a routine which retrieves all the guesses and enters universal nodes whose left children check them in the manner of q_0 or q_1 above. To do this only requires adding finitely many states in a loop whose formal construction I leave to the reader. On exit from this loop, M' proceeds to q'.

If q is a universal state, M' has a state q_a which branches to two more universal states q_0 and q_1; q_0 simulates the transitions on input bit 0 for q, and q_1 simulates those for 1. M' records the bits assumed along each individual branch until M reaches an existential (or halting) state q'. In place of q', the branch by M' goes to a "master existential state" q_e. The left child of q_e goes to a routine in existential states which fans out to poll each assumed input bit in some leaf, and the leaf accepts iff the tested bit was *wrong*. The right child of q_e goes to q' and proceeds with the computation, *even though this computation path may be invalid*. The reasoning is that the path through the right child of q_e is invalid only if the left child has a leaf which accepts—this makes q_e automatically return "accept" to its parent and renders the left child of q_e immaterial. Put another way, every node q_a in the universal phase expects both its children to accept. The child for the wrong input bit has the property that all branches below it have *some* wrong bit that the left child of their q_e can guess and propagate upward the required "accept." The child for the correct input bit, however, has one critical path along which all the assumed bits are correct. The q_e node for this critical path *cannot* accept via its left child, and thus the responsibility falls

on the right child of q_e to accept. Since precisely this path carries on a valid computation path of M, the construction is correct.

Because an extra level of like quantifiers is tacked on in the case where q' is a halting state of the original machine M, the new machine has $k+1$ "like-quantifier phases" instead of k along each path. The extra additive term of $s(n) + t(n)$ on space comes from storing a spare copy of the configuration at the beginning of a phase and the sequence of guesses along the branch in that phase. ■

Corollary 24.3. *The definition of* LOGH $:= \bigcup_{k=1}^{\infty} \Sigma_k^{\log}$ *in unaffected by the choice of input-tape convention, even though the individual $\Sigma_k^{\log}$ levels might be shifted by one.* □

Consensus is that for $k \geq 1$, one should use convention (3) to define $\Sigma_k^{\log}$, while for $k = 0$, i.e. $\Sigma_0^{\log} =_{\text{def}}$ DLOGTIME, convention (1) is used. The languages 0^*1^* and A above both belong to $\Sigma_2^{\log}$, since the machine sketched in class first guesses the break and then verifies the segments. The proof in the next section uses the most liberal convention (1) even for the case $k = 1$; i.e., in simulating a nondeterministic $O(\log n)$-time ATM.

25. Log-Bounded Rudimentary Relations

[Part of Lecture 39, prepared by Regan.]

The *log-bounded rudimentary relations* (LBR) [Jon75] are the closure of the following basic relations

$$\sigma(x, j, a) \equiv \text{bit } j \text{ of } x \text{ equals } a \tag{6}$$

$$P_C(x, u, v, w) \equiv |w| \leq C \log_2 |x| \wedge uv = w \quad (C \in \mathbf{N}^+) \tag{7}$$

under (1) Boolean operations, (2) explicit transformations, and (3) *log-bounded* existential quantification, viz. if $\rho \in$ LBR, then for any $C > 0$, the relation τ (of the same arity k) defined by

$$\tau(x_1, x_2, \ldots, x_{k-1}, z) := (\exists y : |y| \leq C \log_2 |z|) \rho(x_1, x_2, \ldots, x_{k-1}, y) \tag{8}$$

also belongs to LBR. Here, think of "x_1" as the input string x, and recall that explicit transformation from τ gives us $\rho'(x, x_2, \ldots, x_{k-1}) := \tau(x, x_2, \ldots, x_{k-1}, x)$, which is usually what we are interested in.

LBR is just like the (linear-bounded) rudimentary relations except for the log-bound on concatenation, and more notably, the extra predicate $\sigma(x, j, a)$. This plays the role of both the input-indexing FOL predicate $Z(j)$, and Immerman's predicate $BIT(i, j) \equiv$ bit number j of the number i in standard binary notation is a 1. Thus LBR lets one write $\sigma(i, j, a)$ and not worry about the lack of type distinction between i and x. (Well, *I'm* bothered by it! But I haven't been able to pull off a "unary trick" like with my proof of FOL $\subseteq$ SF.) Note also the following convention on semantics: if the index j is "out of range"; i.e., if $j > |x| - 1$ (recall we number from 0), then for all alphabet characters a,

$\sigma(x,j,a)$ is false, and likewise $j > |i|-1 \Longrightarrow \wedge_a \neg\sigma(i,j,a)$. The gist of how LBR simulates FOL+BIT is conveyed by two examples. First,

$$Z(i) = a \ \wedge\ BIT(i,j) \ \wedge\ i = k$$

$$\Longleftrightarrow$$

$$\sigma(x,i,a) \ \wedge\ \sigma(i,j,1) \ \wedge\ \forall j : [\sigma(i,j,1) \longleftrightarrow \sigma(k,j,1)] \ \wedge\ [\sigma(i,j,0) \longleftrightarrow \sigma(k,j,0)].$$

Note that the "$\longleftrightarrow$" is needed for both 1 and 0, because of the semantic convention. Allender and Gore [AG91b] do essentially the same thing in their definition of "the constant n" at the bottom of p93. (Yes, this bugs me too, and for another thing, I don't see the "few simplifications that result.") Also conventionally, i and k are regarded as numbers in standard binary notation, with a leading '1' except for zero, and to justify the above enforcing that i and k are equal as bit-strings, we have to disallow leading 0s, *or* (and better, I think) change the convention to i and k being written in dyadic notation. More peculiar (in the British, not-necessarily pejorative, sense of the word) is the definition of '$<$' with some $x$cess baggage carried along:

$$i < k \iff (\exists u : |u| \leq C\log_2|x|)(\exists v : |v| \leq C\log_2|x|)(\exists v' : |v'| \leq C\log_2|x|)$$

$$[P_C(x,u,v,i) \ \wedge\ P_C(x,u,v',k) \ \wedge\ \sigma(v,0,0)\sigma(v',0,1)].$$

This works because numbers are written most-significant-bit first. If FOL variables are restricted to the range $[0\ldots N]$ ($N = |x|$), and we change the definition of P_C to read "$\ldots |w| \leq C(1 + \lfloor\log_2|x|\rfloor)$," then we can take $C = 1$ as Allender and Gore do. (More "literary criticism": Their reference to "$j+1$" on p93 is one reason I didn't restrict indices to $[0\ldots N-1]$.) Also with this restriction, all quantifications of FOL variables are automatically log-bounded, and that completes the simulation of FOL+BIT by LBR.

The direct simulation of LBR by FOL is more problematic. The predicates $Z(i)$ and $BIT(i,j)$ handle the two types of uses of σ well enough, but the problem is how to represent $P_C(x,u,v,w)$. If FOL variables are regarded as having values in standard binary notation, then one can only represent instances of concatenation in which both u and v begin with 1. This can be fixed either by using dyadic notation, or by coding a string u by an FOL variable with value $1u$. The *intent* is first to define the relation $k = |u|$,[9] and then using the predicate $PLUS(i,j,k)$ defined in terms of BIT and $>$ in [BIS90], to write

$$uv = w =_{\text{def}} (\exists j)(\forall i,k)\,[j = |u| \ \wedge\ (i < j \rightarrow (BIT(u,i) \longleftrightarrow BIT(w,i)) \ \wedge$$

$$((k \geq j \ \wedge\ PLUS(i,j,k)) \rightarrow (BIT(v,i) \longleftrightarrow BIT(w,k))).$$

[9] Even here, if $BIT(u,j)$ is defined to fail either when bit j of u is 0 or when j is out of range, opposite to the convention on σ, there are annoying technical problems in defining $k = |u|$. [BIS90] get away with their equation for "L" on p291 because they (evidently) write binary numbers with the *least* significant bit first.

I'll leave further discussion of details to later, in conjunction with asking exactly where $PLUS$ and $BSUM$ are needed in the proofs. There's also the problem of what to do if $C > 1$. Implicitly, [BIS90] encode a string of length $C\log_2|x|$ by a vector of C-many variables $\langle v_1,\ldots,v_C\rangle$. This isn't necessary in LBR, and involves one technical point worth discussing.

To maximize clarity in the next section, I shall show DLOGTIME $\subseteq$ LBR instead of DLOGTIME $\subseteq$ FO (recall FO stands for languages definable in FOL+BIT). Later I will explore what technicalities need to be plowed in order to make this work in FOL+BIT. For now, let us recall Jones' trick for getting a rudimentary definition of

$$EQ(u) \equiv \#0(u) = \#1(u),$$

where we will consider u to be over the alphabet $\{0,1,2\}$. Namely, define homomorphisms h_0 and h_1 which respectively erase 0s and 1s. Then define

$$EQ(u) =_{\text{def}} (\exists v_0)(\exists v_1) : h_0(u) = v_0 \ \wedge\ h_1(u) = v_1 \ \wedge\ |v_0| = |v_1|.$$

When applying this, we will have $|u| \leq C\log_2|x|$ for some fixed constant C, and since $|v_0|, |v_1| \leq |u|$, the quantifications are log-bounded. The three conjuncts are all rational relations, and by extension of problem 6.1 on homework, definable via P_C. This is the only place where P_C is needed in the proof in the next section, and I leave it to you to examine which *instances* of P_C are needed to set up those three conjuncts.

26. LOGH is contained in LBR

[Lectures 39–40, prepared by Regan.]

Let M be a Turing machine which runs in space $s(n)$ and time $t(n)$, and which has some fixed number k of work-tapes. The description below is intended for deterministic or nondeterministic TMs with "random-access" to input which run in time and space $O(\log n)$, but it can be modified for linear-time bounded TMs as well. Indeed, the theorem DLIN $\subseteq$ RUD could have been proved this way, instead of using intersections of CFLs as before. Previously we have analyzed a TM computation in the form of a *tableau* or *time-space diagram*, namely a sequence of IDs. The problem is that this representation can have length $s(n)t(n)$, which is too long to quantify within linear length bounds (for RUD), or within logarithmic length bounds (for LBR). Instead, we can use the *trace* of a computation, which is nothing but the sequence of transitions or "dominoes"

$$[q_0, i_{00}, c_{10}, \ldots, c_{k0}; d_{10}, \ldots, d_{k0}, D_{10}, \ldots, D_{k0}, q_1]$$

$$[q_{r-1}, i_{0r}, c_{1r}, \ldots, c_{kr}; d_{1r}, \ldots, d_{kr}, D_{1r}, \ldots, D_{kr}, q_r]$$

$$[q_{t-1}, i_{0t}, c_{1t}, \ldots, c_{kt}; d_{1t}, \ldots, d_{kt}, D_{1t}, \ldots, D_{kt}, q_t]$$

Here i_{0r} stands for the input character read at step r. For standard TMs, there would also be a direction D_{0r} for the movement of the input head in step r. For RAM-TMs, I am using the convention that they can read an input character at any step (the initially blank address tape, which is tape 1, holds zero and references bit x_0). Note that each "domino" has length depending only on M, not on t or x, and we can let Δ be the finite alphabet of dominoes as before (really, Δ is just δ_M). Let $\vec{d}$ stand for computations in this trace format. The letter e will be used for individual dominoes. A helpful abbreviation is that $\wedge_{e:c_j=c}$ denotes conjunction over all dominoes whose "c_j component" equals c, and similarly for the written characters d_j and head movements D_j.

Theorem 26.1. LOGH $\subseteq$ LBR.

Proof. Suppose M runs in time $t(n) \leq C \log_2 n$, where C is a constant. We may put M in a normal form with a unique accepting state q_{acc}, and add transitions which keep M in state $qacc$ forever. Also M is not allowed to *write* the blank B. For generality, I'll make M *non*deterministic, and for cosmetic reasons, I'll build into M the following peculiar idea: At the first step, M marks the leftmost cell of each worktape with a $\wedge$, and moves every worktape head right. M continues moving the heads right until it nondeterministically chooses to stop them after some number s of steps, each writing a \$ in cell number s on its tape. Then the computation begins for real. M itself is coded so that no head can move rightward of \$ or left of $\wedge$, so that M has initially guessed how much space it will use. Moreover, all disposable cells hold non-blank characters, and since M never writes B, the blank need not be considered from here on. Once this is done, every valid trace $\vec{d}$ begins with s-many Rs in each "D_j" component. This will be a convenience in using Jones' *EQ* predicate from [Jon69] in place of *BSUM* from [BIS90].

The basic idea is to write: For all x,

$$x \in L(M) \iff (\exists \vec{d} : |\vec{d}| \leq C \log_2 |x|)(\exists s : |s| \leq |\vec{d}|):$$

[the elements of $\vec{d}$ "match like dominoes," and state q_t is q_{acc}],

& the first s steps are as described above,

& for each step $r \geq s$, the characters $i_{0r}, c_{1r}, \ldots, c_{kr}$ listed as being read in the rth domino are the ones M really sees on the tape.

The first predicate is easy to do once we fix a way to reference individual components of a domino $e \in \Delta$, as exemplified by:

$$q_t = q_{acc} =_{\text{def}} \bigvee_{e=[\ldots,q_{acc}]} \sigma(\vec{d}, t-1, e).$$

The first predicate then becomes:

$$q_t = q_{acc} \ \wedge \ (\forall r, r') : (0 \leq r \ \wedge \ r' < t \ \wedge \ r' = r+1) \rightarrow$$

$$\bigvee_{e, e' \text{ which match}} \sigma(\vec{d}, r, e) \ \wedge \ \sigma(\vec{d}, r', e').$$

For the second, we state right out that the first domino, next $s-1$ dominoes, and the one after that have the following respective forms, in which only the input symbol c scanned is unspecified, and the states q_0, p_1, p_2 are taken verbatim from the code of M:

$$\begin{aligned} e_{0c} &= [q_0, c, B, \ldots, B, \wedge, \ldots, \wedge, R, \ldots, R, q_1] \\ e_{1c} &= [q_1, c, B, \ldots, B, 0, \ldots, 0, R, \ldots, R, q_1] \\ e_{2c} &= [q_1, c, B, \ldots, B, \$, \ldots, \$, L, \ldots, L, q_2] \end{aligned}$$

(Under the convention in which tape 1 initially holds n or $n-1$ in binary, where n is the length of the input, one can "hard-wire" a different routine, left to the reader.) The second predicate is:

$$(\bigvee_{c\in\Sigma} \sigma(\hat{d}, 0, e_{0c})) \ \wedge \ (\bigvee_{c\in\Sigma} \sigma(\hat{d}, s, e_{2c})) \ \wedge$$

$$(\forall r)[(0 < r \ \wedge \ r < s) \rightarrow \bigvee_{c\in\Sigma} \sigma(\hat{d}, r, e_{1c})).$$

The rest of the proof is how to represent the third condition that enforces the "consistency" of $\hat{d}$. In top-down fashion, with suggestive names for the intermediate predicates, we define:

$$\mathit{Valid}(\hat{d}) =_{\text{def}} (\forall r : 0 \leq r < t) \mathit{InputOK}(\hat{d}, r) \bigwedge \bigwedge_{1\leq j\leq k} \mathit{TapeOK}_j(\hat{d}, r).$$

$$\mathit{InputOK}(\hat{d}, r) =_{\text{def}} (\forall I)[\bigwedge_{c\in\Sigma} \bigwedge_{e:c_0=c} (\sigma(\hat{d}, r, e) \ \wedge \ \mathit{IsAddress}(\hat{d}, r, I)) \rightarrow \sigma(x, I, c)].$$

$$\mathit{TapeOK}_j(\hat{d}, r) =_{\text{def}} (\forall I)(\forall p):$$

$$\left(\bigwedge_{c\in\Sigma} \bigwedge_{e:c_j=c} (\sigma(\hat{d}, r, e) \ \wedge \ \mathit{IsString}_j(\hat{d}, r, I) \ \wedge \ \mathit{Pos}_j(r, p) \right) \rightarrow \sigma(I, p, c).$$

$$\mathit{IsString}_j(r, I) =_{\text{def}} (\forall p)[\bigwedge_{c\in\Gamma} (\mathit{IsChar}_{jc}(\hat{d}, r, p) \longleftrightarrow \sigma(I, p, c))].$$

$$\mathit{IsAddress}(r, I) =_{\text{def}} (\forall p, p') : p' = p + 1 \rightarrow$$

$$\Big((\sigma(I, p, 0) \ \wedge \ \mathit{IsChar}_{10}(\hat{d}, r, p') \ \vee \ (\sigma(I, p, 1) \ \wedge \ \mathit{IsChar}_{11}(\hat{d}, r, p') \Big).$$

The last definition can be modified to suit one's taste about how addresses are represented; this one allows any string in $\{0, 1\}^{s-1}$ to be considered an address. One can also consider the format of addresses to be "hard-wired" by restrictions on the code of M. The only reason to define *IsAddress* separately from *IsString* is to surmount the left endmarker $\wedge$ on tape 1. Similarly, it is not necessary to build into the formula that M never moves left of $\wedge$ not right of \$; this is already taken care of by restricting "dominoes" to Δ.

Now $\mathit{Pos}_j(r, p)$ means "at step r, the head on tape j is reading the symbol in cell p on that tape," and $\mathit{IsChar}_{jc}(\hat{d}, r, p)$ means "at step r, cell p on tape j holds c." The strings I quantified over have length $O(\log n)$, which is acceptable in LBR, while p and r

are even tinier objects of length $\log\log n + O(1)$. It remains to write *IsChar* in terms of *Pos* and then write *Pos*. Owing to our hard-wiring of the initial stage, it suffices to define $IsChar_{jc}(\hat{d}, r, p)$ for characters $c \neq B$ in a way that need only be correct for $r > s$ and $p \leq s$.

$$IsChar_{jc}(\hat{d}, r, p) =_{\text{def}} (\exists r')[r' < r \ \wedge \ Pos_j(r', p) \ \wedge \ (\forall r'' \neg(r' < r'' \ \wedge \ r'' < r \ \wedge \ Pos_j(r'', p))]$$

$$\wedge \bigvee_{e:d_j = c} \sigma(\hat{d}, r, e).$$

The second predicate asserts that there was a last time cell p on tape j was visited, and that the character written then was c. Now to write *Pos*, we focus on the entries D_j giving the direction the head moves at step r. We identify a left move with 0, a right move with 1, and use 2 for "filler" in defining:

$$\begin{aligned} IsSlice_j(\hat{d}, p, r, w) =_{\text{def}} |w| = |\hat{d}| \ \wedge \ (\forall i) : \\ [(i < p \ \vee \ i \geq r) \quad &\rightarrow \quad \sigma(w, i, 2)] \ \wedge \\ (p \leq i \ \wedge \ i < r) \quad &\rightarrow \quad (\sigma(w, i, 0) \ \wedge \bigvee_{e:D_j = L} \sigma(\hat{d}, i, e)) \\ &\vee \quad (\sigma(w, i, 1) \ \wedge \bigvee_{e:D_j = R} \sigma(\hat{d}, i, e)). \end{aligned}$$

The final idea is to assert that the head on tape j is in position p at time r iff either $r = p$ (as during the startup) or $r > p$ and the movements in the D_j column from step p to step $r - 1$ have exactly as many Ls as Rs, which becomes the following:

$$Pos_j(r, p) =_{\text{def}} \mathrm{r = p} \vee [\mathrm{r} \text{ ¿ } \mathrm{p} \wedge (\exists w)(IsSlice_j(\hat{d}, p, r, w) \ \wedge \ EQ(w))].$$

That turns the trick, and leaves only the requirement of finding a definition, in LBR or in FO+BIT, of $EQ(w)$. First express the condition $J(w, v_0, v_1)$ that $v_0, v_1 \in \{0,1\}^*$, $|v_0| = |v_1| = |w|$, v_0 has a '1' precisely where w has a '0', and v_1 has a '1' precisely where w has a '1'. Barrington, Immerman, and Straubing [BIS90] essentially define:

$$EQ(w) =_{\text{def}} (\exists v_0)(\exists v_1)(\exists i)[J(w, v_0, v_1) \ \wedge \ BSUM(v_0, i) \ \wedge \ BSUM(v_1, i)].$$

Alternatively, let η be the rational relation which defines the graph of the erasing homomorphism $h(0) = \epsilon$, $h(1) = 1$; i.e., $\eta = ((0, \epsilon) \cup (1, 1))^*$. Then we can define:

$$EQ(w) =_{\text{def}} (\exists v_0)(\exists v_1)(\exists y)[J(w, v_0, v_1) \ \wedge \ \eta(v_0, y) \ \wedge \ \eta(v_1, y)].$$

Jones [Jon69] gives a definition of η which, since the strings v_0 and v_1 have length $O(\log|x|)$, is formalizable in LBR. However, his proof does so via an inductive definition of counting, and contains essentially the same details as the definition of *BSUM* in [BIS90]. This was the ulterior motive of problem 6.1 and its timing, and the class remains challenged to fix the "handwave" I've noted and to determine whether one can represent the simple rational relation η in LBR without such sophisticated counting.

Absent this final detail, that completes the proof that nondeterministic log-time RAM-TMs accept languages in LBR. The conclusion that LOGH $\subseteq$ LBR is obtained by exactly the same means as in the earlier deduction of LINH $\subseteq$ RUD from the lemma NLIN $\subseteq$ RUD. ∎

27. Summary of Remaining Notes

Going from LBR to FO+BIT was treated in the last section, and going from LBR or FO+BIT to LOGH is straightforward and similar to RUD $\subseteq$ LINH. Hence all three classes are equal. A sketch of the remaining originally-planned lectures, which I may be able to type up in January:

- While doing LBR $\subseteq$ LOGH and FO+BIT $\subseteq$ LOGH, show also how the *matrices* of the quantifier-free formulas have depth-2 unbounded fan-in circuits, and then how each $\exists$ quantifier adds a big OR gate, each $\forall$ quantifier adds an AND gate. (Do this while defining circuits in general, plus the lemma that negations can always be pushed up to the inputs.) Thus every language $L \in$ LBR has a family $[C_n]_{n=1}^{\infty}$ of constant-depth, unbounded fan-in circuits, one for each input length n. Give the definitions of the classes NC^k [respectively, AC^k] of languages which have families $[C_n]$ of circuits of bounded [resp. unbounded] fan-in, such that the circuits C_n have polynomial size and $O(\log^k n)$-depth. Thus LBR $\subseteq \mathrm{AC}^0$. Moreover, there is a *uniform* way of describing each individual C_n, in terms of the single FOL+BIT formula ϕ that defines L for all input length.
- Discuss the problem of non-uniform vs. uniform circuits in general. Give an example of a non-uniform family of depth-1 circuits which decide a non-r.e. set—the non-r.e.-ness is all in the selection of which circuit for which n. *Briefly* review the major competing criteria of uniformity, and without going into too much unrewarding detail, explain the theorem that FOL+BIT = DLOGTIME-uniform AC^0. Then prove the theorem (from [Bar89, BIS90]) that *non-uniform* AC^0 is equivalent to FOL plus *arbitrary numerical predicates* in place of BIT. (A "numerical predicate" is one which doesn't involve the input x, i.e. has no $Z(i)$ terms. $PLUS(i, j, k)$, and the graphs of multiplication and $i^j = k$, with ad-hoc conventions for handling values-out-of-bounds, are examples of numerical predicates.)
- Mention the theorem of Furst, Saxe, and Sipser [FSS84] that the regular language $Parity =_{\text{def}} \{x \in \{0,1\}^* : \#1(x) \text{ is odd}\}$ is not in AC^0; indeed, Yao [Yao85] and Hastad [Has86, Has89] obtained exponential lower bounds on the size of constant-depth circuits for *Parity*. Then define the class ACC^0 of languages with constant-depth, polynomial size circuits of unbounded fan-in AND, OR, and "Mod k" gates, where k belongs to some finite set of numbers fixed independent of n. (Negation gates are taken for granted.) Clearly *Parity* is in ACC^0. Thus $\mathrm{AC}^0 \subset \mathrm{ACC}^0 \subseteq \mathrm{NC}^1$. It is also known that $\mathrm{ACC}^0 \subset$ PP [AG91a], but it is still consistent with current knowledge that ACC^0 = NP. Discuss the problem of separating ACC^0 from NC^1, and how there are regular languages which are "NC^1-complete" under "constant-depth reducibility."
- State and prove translation theorems, some of them original to Allender and Gore, which tie open questions about these classes to the longer-standing open questions about RUD and ATIME$[O(n)]$ and DSPACE$[O(n)]$ and the DLBA vs. NLBA problem. Doing so would tie the whole course subject matter neatly together.
- Finally describe some of my own work, including a new machine characterization of uniform NC^1 (and NC^k for all $k > 1$), and interesting research problems which arise

when one tries to extend my model for the classes AC^k.

References

[AG91a] E. Allender and V. Gore. On strong separations from AC^0. In *Proc. 8th FCT*, volume 529 of *LNCS*, pages 1–15. Springer Verlag, 1991. To appear in the DIMACS Special Year volume, ed. J.-Y. Cai, AMS Publications.

[AG91b] E. Allender and V. Gore. Rudimentary reductions revisited. *Inf. Proc. Lett.*, 40:89–95, 1991.

[Bar89] D. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comp. Sys. Sci.*, 38:150–164, 1989.

[BG70] R. Book and S. Greibach. Quasi-realtime languages. *Math. Sys. Thy.*, 4:97–111, 1970.

[BIS90] D. Mix Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *J. Comp. Sys. Sci.*, 41:274–306, 1990.

[CKS81] A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *J. ACM*, 28:114–133, 1981.

[FSS81] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. In *Proc. 22nd FOCS*, pages 260–270, 1981.

[FSS84] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Sys. Thy.*, 17:13–27, 1984.

[GH69] S. Greibach and J. Hopcroft. Scattered context grammars. *J. Comp. Sys. Sci.*, 3:233–247, 1969.

[Gre81] S. Greibach. Formal languages: Origins and directions. *Ann. Hist. Comp.*, 3:14–41, 1981.

[Has86] J. Hastad. Almost optimal lower bounds for small-depth circuits. In *Proc. 18th STOC*, pages 6–20, 1986.

[Has89] J. Hastad. Almost optimal lower bounds for small-depth circuits. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 143–170. JAI Press, Greenwich, CT, USA, 1989.

[HK93] T. Harju and H.C.M. Klein. Morphisms and rational transducers. *Bull. EATCS*, 51:168–180, October 1993. Guest column, A. Salomaa ed., Formal Language Theory Column.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison–Wesley, Reading, MA, 1979.

[Imm87] N. Immerman. Languages which capture complexity classes. *SIAM J. Comp.*, 16:760–778, 1987.

[Jon69] N. Jones. Context-free languages and rudimentary attributes. *Math. Sys. Thy.*, 2:102–380, 1969. Corrigendum *MST* **11** (1978), 379–380.

[Jon75] N. Jones. Space-bounded reducibility among combinatorial problems. *J. Comp. Sys. Sci.*, 11:68–85, 1975. Corrigendum *JCSS* **15** (1977), 241.

[Lad77] R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comp.*, 6:467–480, 1977.

[Lin92] S. Lindell. A purely logical characterization of circuit uniformity. In *Proc. 7th Structures*, pages 185–192, 1992.

[Lip78] R. Lipton. Model-theoretic aspects of computational complexity. In *Proc. 19th FOCS*, pages 191–200, 1978.

[Mey69] A. Meyer. A note on star-free events. *J. ACM*, 16:220–225, 1969.

[MP71] R. McNaughton and S. Papert. *Counter-Free Automata.* MIT Press, Cambridge, MA, 1971.

[PP86] D. Perrin and J. Pin. First-order logic and star-free sets. *J. Comp. Sys. Sci.*, 32:393–406, 1986.

[PPST83] W. Paul, N. Pippenger, E. Szemerédi, and W. Trotter. On determinism versus nondeterminism and related problems. In *Proc. 24th FOCS*, pages 429–438, 1983.

[Reg88] K. Regan. The topology of provability in complexity theory. *J. Comp. Sys. Sci.*, 3:384–432, 1988.

[Ros84] H. Rose. *Subrecursion: Functions and Hierarchies.* Oxford Logic Guides. Oxford University Press, 1984.

[Smu61] R. Smullyan. *Theory of Formal Systems*, volume 47 of *Annals of Mathematics Studies.* Princeton University Press, Princeton, NJ, 1961.

[Tho82] W. Thomas. Classifying regular events in symbolic logic. *J. Comp. Sys. Sci.*, 25:360–376, 1982.

[Wra75] C. Wrathall. *Subrecursive predicates and automata.* PhD thesis, Harvard University, 1975. Reprinted and available via the Department of Mathematics, Univ. of CA at Santa Barbara.

[Wra78] C. Wrathall. Rudimentary predicates and relative computation. *SIAM J. Comp.*, 7:194–209, 1978.

[Yao85] A. Yao. Separating the polynomial-time hierarchy by oracles. In *Proc. 26th FOCS*, pages 1–10, 1985.