Lecture 25: Linear Bounded Automata and Undecidability for CFGs

28 April 2009

"It is a damn poor mind indeed which can't think of at least two ways to spell any word."
– Andrew Jackson

This lecture covers Linear Bounded Automata, an interesting compromise in power between Turing machines and the simpler automatas (DFAs, NFAs, PDAs). We will use LBAs to show two CFG grammar problems (equality and generating all strings) are undecidable.

In some of the descriptions we uses PDAs. However, the last part of this notes show how these PDAs can be avoided, resulting in arguably a simpler and slightly more elegant argument.

1 Linear bounded automatas

A ***linear bounded automata*** (***LBA***) is a TM whose head never moves off the portion of the tape occupied by the initial input string.

That is, an LBA is a TM that uses only the tape space occupied by the input.

An equivalent definition of an LBA is that it uses only k times the amount of space occupied by the input string, where k is a constant fixed for the particular machine. To simulate k tape cells with a single tape cell, increase the size of the tape alphabet Γ. E.g. the new tape alphabet has symbols that are k-tuples of the symbols from the old alphabet.

A lot of interesting algorithms are LBAs, because they use only space proportional to the length of the input. (Naturally, you need to pick the variants of the algorithms that use space efficiently.) Examples include A_{DFA}, A_{CFG}, E_{DFA}, E_{CFG}, $s - t$ graph reachability, and many others.

When an LBA runs, a transition off the righthand edge of the input area cause the input to be rejected. Or maybe the read head simply sticks on the rightmost input position. You can define them either way and it will not matter for what we are doing here.

1.1 LBA halting is decidable

Suppose that a given LBA T (which is a TM, naturally) has

- q states,
- k characters in the tape alphabet Γ (we remind the reader that the input alphabet $\Sigma \subseteq \Gamma$ and also the special blank character $\sqcup$),

- and the input length is n.

Then $\mathtt{T}$ can be in at most

$$\alpha(n) = \overbrace{k^n}^{\substack{\text{tape}\\ \text{content}}} * \overbrace{n}^{\substack{\text{head}\\ \text{position}}} * \overbrace{q}^{\substack{\text{controller}\\ \text{state}}} = k^n n q \tag{1}$$

configurations.

Here is the shocker: If an LBA runs more than $\alpha(n)$ steps, then it must be looping. As such, given an LBA $\mathtt{T}$ and a word w (with n characters), we can simulate it for α steps. If it does not terminate by then, then it must be looping, and as such it can never stop on its input. Thus, an LBA that stops on input w must stop in at most $\alpha(|w|)$ steps.

This implies that

$$\mathrm{A}_{\mathsf{LBA}} = \left\{ \langle \mathtt{T}, w \rangle \;\middle|\; \mathtt{T} \text{ is a } \mathsf{LBA} \text{ and } \mathtt{T} \text{ accepts } w \right\}$$

is decidable. Similarly, the language

$$\mathrm{Halt}_{\mathsf{LBA}} = \left\{ \langle \mathtt{T}, w \rangle \;\middle|\; \mathtt{T} \text{ is a } \mathsf{LBA} \text{ and } \mathtt{T} \text{ stops on } w \right\}.$$

is decidable.

We formally prove one of the above claims. The other one follows by a similar argumentation.

Claim 1.1 *The language* $\mathrm{A}_{\mathsf{LBA}}$ *is decidable.*

Proof: Indeed, our decider would receive as input $\langle \mathtt{T}, w \rangle$, where $\mathtt{T}$ is an LBA. Let $n = |w|$. By the argumentation above, if $\mathtt{T}$ accepts w, then it does it in at most $\alpha(n)$ steps (see Eq. (1)). As such, simulate $\mathtt{T}$ on the input w for $\alpha(n)$ steps, using the universal TM simulator U_{TM}. If the simulation accepts, then accept. If the simulation rejects, then reject. Now, if the simulation did not accept after simulating $\mathtt{T}$ for $\alpha(n)$ steps, then $\mathtt{T}$ is looping forever on w, and so we reject.

Of course, if during the simulation $\mathtt{T}$ decides to move past the end of the input, it's not an LBA, and as such we reject the input.[1] ∎

1.2 LBAs with empty language are undecidable

In light of the above claim, one might presume that all "natural" languages on LBAs are decidable, but surprisingly, this is not the situation. Indeed, consider the language of all empty LBAs; that is,

$$\mathsf{E}_{\mathsf{LBA}} = \left\{ \langle \mathtt{T} \rangle \;\middle|\; \mathtt{T} \text{ is a } \mathsf{LBA} \text{ and } \mathsf{L}(\mathtt{T}) = \emptyset \right\}.$$

The language $\mathsf{E}_{\mathsf{LBA}}$ is actually undecidable.

[1] For the very careful reader, Sipser handles this case slightly differently. His encoding of $\mathtt{T}$ would specify that the machine is supposed to be an LBA. Attempts to move off the input region would cause the read head to stay put.

1.2.1 A proof via verifying accepting traces

We remind the reader that configuration x of a TM $\mathtt{T}$ ***yields*** the configuration y of tm, if running $\mathtt{T}$ on x for one step results in the configuration y of $\mathtt{T}$. We denote this fact by $x \mapsto y$.

The idea. Assume we are given a general TM $\mathtt{T}$ (which we emphasize is not an LBA) and a word w. We would like to decide if $\mathtt{T}$ accepts w (which is of course undecidable).

If $\mathtt{T}$ does accept w, we can demonstrate that it does by providing a trace of the execution of $\mathtt{T}$ on w. This trace (defined formally below) is just a string. We can easily build a TM that verifies that a supposed trace is legal (e.g. uses the correct transitions for $\mathtt{T}$), and indeed shows that $\mathtt{T}$ accepts w.

Crucially, this trace verification can be done by a TM $\mathtt{Vrf}_{\mathtt{T},w}$ that just uses the space provided by the string itself. That is, the verifier $\mathtt{Vrf}_{\mathtt{T},w}$ is an LBA. The language of $\mathtt{Vrf}_{\mathtt{T},w}$ is empty if $\mathtt{T}$ does not accept w (because then there is no accepting trace). If $\mathtt{T}$ does accept w then the language of $\mathtt{Vrf}_{\mathtt{T},w}$ contains a single word: the trace showing that $\mathtt{T}$ accepts w. So, if we have a decider than decides if $\langle \mathtt{Vrf}_{\mathtt{T},w} \rangle \in \mathsf{E}_{\mathsf{LBA}}$, then we can decide if $\mathtt{T}$ accepts w.

Observe that we assumed nothing about $\mathtt{T}$ or w. The only required property is that $\mathtt{Vrf}_{\mathtt{T},w}$ is a LBA.

The verifier. A ***computation history*** (i.e., trace) for a TM $\mathtt{T}$ on input w is a string

$$\#C_1\#C_2\#\ldots\#C_k\#\#,$$

where $C_1, \ldots, C_k$ are configurations of $\mathtt{T}$, such that

(i) $C_1 = q_0 w$ is the initial configuration of $\mathtt{T}$ when executed on w, and

(ii) C_i yields C_{i+1} (according to the transitions of $\mathtt{T}$), for $i = 1, \ldots, k-1$.

The pair of sharp signs marks the end of the trace, so the algorithm knows when the trace ends.

Such a trace is an ***accepting trace*** if the configuration C_k is an accepting configuration (i.e., the accept state q_{acc} of $\mathtt{T}$ is the state of $\mathtt{T}$ encoded in C_k).[2]

Initial checks. So, we are given $\langle \mathtt{T} \rangle$ and w, and we want to build a verifier $\mathtt{Vrf}_{\mathtt{T},w}$ that checks, given a trace t as input, that this trace is indeed an accepting trace for $\mathtt{T}$ on w. As a first step, $\mathtt{Vrf}_{\mathtt{T},w}$ will verify that C_1 (the first configuration written in t) is indeed $q_0 w$. Next it needs to verify that C_k (the last configuration in t) is an accepting configuration which is also easy (i.e., just verify that q_{acc} is the state written in it). Finally, the verifier needs to make sure that the ith configuration implies the $(i+1)$th configuration in the trace t, for all i.

[2] It should also be the case that no previous configuration in this trace is either accepting or rejecting. This is implied by the fact that TM's don't have transitions out of the accept and reject states.

Verifying two consecutive configurations. So, consider the ith configuration in t, that is

$$C_i = \alpha \mathtt{a} q \mathtt{b} \beta,$$

where α and β are two strings. Naturally, C_{i+1} is the next configuration in the input trace t. Since $\mathtt{Vrf}_{\mathtt{T},w}$ has the code of $\mathtt{T}$ inside it (as a built-in constant), it knows what $\delta_{\mathtt{T}}(q, \mathtt{b})$, the transition function of $\mathtt{T}$, is. Say it knows that $\delta_{\mathtt{T}}(q, \mathtt{b}) = (q', \mathtt{c}, \mathtt{R})$. If our input is a valid trace, then C_{i+1} is supposed to be

$$C_{i+1} = \alpha \mathtt{a} \mathtt{c} q' \beta.$$

To verify that C_i and C_{i+1} do match up in this way, the TM $\mathtt{Vrf}_{\mathtt{T},w}$ goes back and forth on the tape erasing the parts of C_i and C_{i+1} that must be identical. We can not erase these symbols: we will need to keep C_{i+1} around so we can check it against C_{i+2}. So instead we translate each letter $\mathtt{a}$ into a special version of this character $\widehat{\mathtt{a}}$.[3]

After we have marked all the identical characters, we've verified this pair of configurations except for the middle two to three letters (depending on whether this was a left or right move). So the tape in this stage looks like

$$\ldots \# \overbrace{\alpha \mathtt{a} q \mathtt{b} \beta}^{C_i} \# \overbrace{\widehat{\alpha} \mathtt{a} \mathtt{c} q' \widehat{\beta}}^{C_{i+1}} \# \ldots$$

We have verified that the prefix of C_i (i.e., α) is equal to $\widehat{\alpha}$, and the suffix of C_i (i.e., β) is equal to the suffix of C_{i+1} (i.e., $\widehat{\beta}$). So only thing that remains to be verified is the middle part, which can be easily done since we know $\mathtt{T}$'s transition function.

After that, the verifier removes the hats from the characters in C_{i+1} and moves right to match C_{i+1} against C_{i+2}. If it gets to the end of the trace and all these checks were successful, the verifier $\mathtt{Vrf}_{\mathtt{T},w}$ accepts the input trace t.

Lemma 1.2 *Given a (general) TM* $\mathtt{T}$ *and a string* w*, one can build a verifier* $\mathtt{Vrf}_{\mathtt{T},w}$*, such that given an accepting trace* t*, the verifier accepts* t*, and no other string. Note, that* $\mathtt{Vrf}_{\mathtt{T},w}$ *is a decider that always stops.*

Moreover, $\mathtt{Vrf}_{\mathtt{T},w}$ *is a LBA.*

Proof: The details of $\mathtt{Vrf}_{\mathtt{T},w}$ are described above.

It is easy to see that $\mathtt{Vrf}_{\mathtt{T},w}$ never goes on the portion of the tape which are not parts of its original input t. As such, $\mathtt{Vrf}_{\mathtt{T},w}$ is a LBA. ■

Theorem 1.3 *The language* E_{LBA} *is undecidable.*

Proof: Proof by reduction from A_{TM}. Assume for the sake of contradiction that $\mathsf{E}_{\mathsf{LBA}}$ is decidable, and let $\mathsf{E}_{\mathsf{LBA}}$**−Decider** be the TM that decides it. We will build a decider $\mathtt{decider}_5$-A_{TM} for A_{TM}.

[3] We have omitted some details about how to handle moves near the right and left ends of the non-blank tape area. There details are tedious but easy to fill in, and the reader should verify that they know how to fill in the missing details.

$\texttt{decider}_5\text{-}\mathrm{A}_{\mathsf{TM}}\left(\langle \mathtt{T}, w\rangle\right)$

Check that $\langle \mathtt{T}\rangle$ is syntactically correct TM code
Compute $\langle \mathtt{Vrf}_{\mathtt{T},w}\rangle$ from $\langle \mathtt{T}, w\rangle$.
$res \leftarrow \mathsf{E}_{\mathsf{LBA}}\text{−}\mathbf{Decider}\left(\langle \mathtt{Vrf}_{\mathtt{T},w}\rangle\right)$.
if $res ==$ accept **then**
    reject
else
    accept

Since we can compute $\langle \mathtt{Vrf}_{\mathtt{T},w}\rangle$ from $\langle \mathtt{T}, w\rangle$, it follows that this algorithm is a decider. Furthermore, given $\langle \mathtt{T}, w\rangle$ such that $\mathtt{T}$ accepts w, then there exists an accepting trace t for $\mathtt{T}$ accepting w, and as such, $L(\mathtt{Vrf}_{\mathtt{T},w}) \neq \emptyset$. As such $\mathsf{E}_{\mathsf{LBA}}\text{−}\mathbf{Decider}\left(\langle \mathtt{Vrf}_{\mathtt{T},w}\rangle\right)$ rejects its input, which imply that $\texttt{decider}_5\text{-}\mathrm{A}_{\mathsf{TM}}$ accepts $\langle \mathtt{T}, w\rangle$.

Similarly, if $\mathtt{T}$ does not accept w, then $L(\mathtt{Vrf}_{\mathtt{T},w}) = \emptyset$. As such, $\mathsf{E}_{\mathsf{LBA}}\text{−}\mathbf{Decider}\left(\langle \mathtt{Vrf}_{\mathtt{T},w}\rangle\right)$ accepts its input, which imply that $\texttt{decider}_5\text{-}\mathrm{A}_{\mathsf{TM}}$ rejects $\langle \mathtt{T}, w\rangle$.

Thus $\texttt{decider}_5\text{-}\mathrm{A}_{\mathsf{TM}}$ is indeed a decider for A_{TM}, but this is impossible, and we thus conclude that our assumption, that $\mathsf{E}_{\mathsf{LBA}}$ is decidable, was false, implying the claim. ■

1.2.2 A direct proof that $\mathsf{E}_{\mathsf{LBA}}$ is undecidable

We provide a direct proof of Theorem 1.3 because it is shorter and simpler. The benefit of the previous proof is that it introduces the idea of verifying accepting traces, which we would revisit shortly.

Alternative direct proof of Theorem 1.3: We are given $\langle \mathtt{T}, w\rangle$, were $\mathtt{T}$ is a TM and w is an input for it. We will assume that the tape alphabet of $\mathtt{T}$ is Γ, its input alphabet is Σ, and assume that $\mathtt{z}$ and \$ are not in Γ. We build a new machine $\mathtt{Z}_w$ from $\mathtt{T}$ and w that gets as input a word of the form $\mathtt{z}^k\$$. The machine $\mathtt{Z}_w$ first writes w on the input tape, move the head to te beginning of the tape, and then just runs $\mathtt{T}$ on the input, with the modification that the new machine treats $\mathtt{z}$ as a space. However, if the new machine ever reaches the $T\$$ character on the input (in any state), it immediately stops and rejects.

Clearly, $\mathtt{Z}_w$ is an LBA (by definition). Furthermore, if $\mathtt{T}$ accepts w after k steps, then $\mathtt{Z}_w$ would accept the word $w\mathtt{z}^{k+1}\$$. Similarly, if $w\mathtt{z}^j\$$ is accepted by $\mathtt{Z}_w$ then $\mathtt{T}$ would accept w. We thus conclude that $\mathsf{L}(\mathtt{Z}_w)$ is not empty if and only if $w \in \mathsf{L}(\mathtt{T})$.

Going back to the proof, given $\langle \mathtt{T}\rangle$ and w the construction of $\langle \mathtt{Z}_w\rangle$ is easy. As such, assume for the sake of contradiction, that $\mathsf{E}_{\mathsf{LBA}}$ is decidable, and we are given a decider for membership of $\mathsf{E}_{\mathsf{LBA}}$, we can feed it $\langle \mathtt{Z}_w\rangle$, and if this decider accepts (i.e., $\mathsf{L}(\mathtt{Z}_w) = \emptyset$, then we know that $\mathtt{T}$ does not accept w. Similarly, if $\mathtt{Z}_w$ is being rejected by the decider, then $\mathsf{L}(\mathtt{Z}_w) \neq \emptyset$, which implies that $\mathtt{T}$ accepts w. Namely, we just constructed a decider for A_{TM}, which is undecidable. A contradiction. ■

2 On undecidable problems for context free grammars

We would like to prove that some languages involved with context-free grammars are undecidable. To this end, to reduce A_{TM} to a question involving CFGs, we somehow need to map properties of TMs to CFGs.

2.1 TM consecutive configuration pairs is a CFG

Lemma 2.1 *Given a TM* $\mathtt{T}$*, the language*

$$\mathsf{L}_{\mathtt{T}:x\mapsto y} = \left\{ x\#y^R \;\middle|\; x, y \text{ are valid configurations of } \mathtt{T} \text{ and } x \text{ yields } y \right\}$$

is a CFG.

Proof: Let Γ be the tape alphabet of $\mathtt{T}$, and Q be the set of states of $\mathtt{T}$. Let δ be the transition function of $\mathtt{T}$. We have the following rewriting rules depending on δ:

$$\begin{aligned}
&\forall \alpha, \beta \in \Gamma^* \;\; \forall \mathtt{b}, \mathtt{c}, \mathtt{d} \in \Gamma \;\; \forall q \in Q \\
&\qquad \text{if } \; \delta(q, \mathtt{c}) = (q', \mathtt{d}, \mathtt{R}) \quad \text{then } \; \alpha q \mathtt{c} \beta \mapsto \alpha \mathtt{d} q' \beta \quad \equiv \quad \alpha q \mathtt{c} \beta \mapsto \left(\beta^R q' \mathtt{d} \alpha^R\right)^R \\
&\qquad \text{if } \; \delta(q, \mathtt{c}) = (q', \mathtt{d}, \mathtt{L}) \quad \text{then } \; \alpha \mathtt{b} q \mathtt{c} \beta \mapsto \alpha q' \mathtt{b} \mathtt{d} \beta. \quad \equiv \quad \alpha \mathtt{b} q \mathtt{c} \beta \mapsto \left(\beta^R \mathtt{d} \mathtt{b} q' \alpha^R\right)^R .
\end{aligned}$$

Intuitively, $x \mapsto y$ is equivalent to saying that the string x can be very locally edited and generate y. In the above, we need to copy the α and β portions, and then do the rewriting which only involves at most 3 letters. As such, the grammar

$$\begin{aligned}
\Longrightarrow \; \mathsf{S}_1 &\to \mathsf{C} \\
\mathsf{C} &\to \mathtt{x}\mathsf{C}\mathtt{x} \quad \forall \mathtt{x} \in \Gamma \\
\mathsf{C} &\to \mathsf{T} \\
\mathsf{T} &\to q\mathtt{c}\mathsf{Z}q'\mathtt{d} \qquad \forall \mathtt{b}, \mathtt{c}, \mathtt{d} \in \Gamma \; \forall q \in Q \quad \text{such that } \delta(q, \mathtt{c}) = (q', \mathtt{d}, \mathtt{R}) \\
\mathsf{T} &\to \mathtt{b}q\mathtt{c}\mathsf{Z}\mathtt{b}\mathtt{d}q' \qquad \forall \mathtt{b}, \mathtt{c}, \mathtt{d} \in \Gamma \; \forall q \in Q \quad \text{such that } \delta(q, \mathtt{c}) = (q', \mathtt{d}, \mathtt{L}) \\
\mathsf{Z} &\to \mathtt{x}\mathsf{Z}\mathtt{x} \quad \forall \mathtt{x} \in \Gamma \\
\mathsf{C} &\to \#.
\end{aligned}$$

generates $\mathsf{L}_{\mathtt{T}:x\mapsto y}$ as can now be easily verified. ∎

Lemma 2.2 *Given a TM* $\mathtt{T}$ *and an input string* w*, the language*

$$\mathsf{L}_{\mathtt{T},w,trace} = \left\{ C_1\#C_2^R\#C_3\#C_4^R\#\ldots C_k \;\middle|\; \begin{array}{c} C_1\#C_2\#C_3\#C_4\#\ldots\#C_k \\ \text{is an accepting trace of } \mathtt{T} \text{ on } w \end{array} \right\}$$

can be written as the intersection of two context free languages.

Proof: Let L_1 be the regular language $q_0 w \# \Gamma_{\#}^*$ – these are all traces that start with the initial state of $\mathtt{T}$ on w, where q_0 is the initial state of $\mathtt{T}$, and $\Gamma_{\#} = \Gamma \cup \{\#\}$.

Let L_2 be the language of all traces, such that the configuration C_{2i}^R written in the even position $2i$ is implied by the configuration C_{2i-1} written in position $2i-1$, for all $i \geq 1$. Clearly, this is a context free grammar, by just extending the grammar of Lemma 2.1.

Using similar argument, L_3 be the language of all traces, such that the configuration C_{2i+1} written in the odd position $2i+1$ are implied by the configuration C_{2i}^R written in position $2i$. Clearly, this is a context free grammar, by just modifying and extending the grammar of Lemma 2.1.

Finally, let L_4 be the regular language of all traces, such that the last trace written on them is accepting. That is $\Gamma_{\#}^* \# \Gamma^* q_{\text{acc}} \Gamma^*$, where q_{acc} is the accepting state of $\mathtt{T}$.

Now, L_1 and L_4 are regular, and L_2 and L_3 are context free. Since context free language are closed under intersection with regular languages, it follows that the language $\mathsf{L}' = L_1 \cap L_2 \cap L_4$ is CFL. Now, the required language is clearly $L_1 \cap L_2 \cap L_3 \cap L_4 = \mathsf{L}' \cap L_3$, which is the intersection of two context free languages. ■

Theorem 2.3 *The language* $\left\{ \langle \mathcal{G}, \mathcal{G}' \rangle \;\middle|\; \mathsf{L}(\mathcal{G}) \cap \mathsf{L}(\mathcal{G}') \neq \emptyset \right\}$ *is undecidable. Namely, given two context free grammars, there is no decider that can decide if there is a word that they both generates.*

Proof: If this was decidable, then given $\langle \mathtt{T}, w \rangle$, we can decide if the language $\mathsf{L}_{\mathtt{T},w,trace}$ of Lemma 2.2 is empty or not, since it is the intersection of two context free grammars that can be computed from $\langle \mathtt{T}, w \rangle$. But if this language is not empty, then $\mathtt{T}$ accepts w. Namely, we got a decider for A_{TM}, which is a contradiction. ■

2.2 The language of a context-free grammar generates all strings is undecidable

Consider the language

$$\mathsf{ALL}_{\mathsf{CFG}} = \left\{ \langle G \rangle \;\middle|\; G \text{ is a } \mathsf{CFG}, \text{ and } L(G) = \Sigma^* \right\}.$$

This language seems like it should be decidable, since $\mathsf{E}_{\mathsf{CFG}}$ was decidable. But it is not. It is a fairly delicate matter whether questions about CFGs are decidable or not. The proof technique is similar to what we used for $\mathsf{E}_{\mathsf{LBA}}$.

2.2.1 The idea

The idea is that given $\mathtt{T}$ and w to build a verifier to an accepting traces for $\mathtt{T}$ and w. Here the verifier is going to be a CFG. The problem is, if you think about it, is that there is no way that a CFG can verify a trace, as the checks needed to be performed are too complicated to be performed by a CFG.

Luckily, we can generate a CFG $\mathtt{Vrf}\mathcal{G}_{\mathtt{T},w}$ that would accept all the traces that are ***not*** accepting traces for $\mathtt{T}$ on w. Indeed, we will build several CFGs, each one "checking" one condition, and their union would be the required grammar. As such, $L(\mathtt{Vrf}\mathcal{G}_{\mathtt{T},w})$ is the set of all strings, if and only if, $\mathtt{T}$ does not have an accepting trace for w.

The alphabet of our grammar is going to be

$$\Sigma = \Gamma \cup Q \cup \{\#\},$$

where Γ is the tape alphabet of $\mathtt{T}$, Q is the set of states of $\mathtt{T}$, and $\#$ is the special separator character.

(Or, almost. There is a small issue that needs to be fixed, but we will get to that in a second.)

2.2.2 The details of the PDA trace checker

It is easier to understand checking the trace if we build a PDA. We can then transform our PDA into an equivalent grammar.

Checking that a trace $t = \#C_1\#C_2\# \dots \#C_k\#\#$ is valid, requires checking the following:

(i) t looks syntactically like a trace

(ii) Initial check: $C_1 = q_0 w$.

(iii) Middle check: C_i implies C_{i+1}, for all i.

(iv) Final check: C_k contains q_{acc}.

It is not hard for a PDA to check that syntax and the first and last configurations are OK.

However, we can not check that the middle configurations match, because a PDA can only compare strings that are in reverse order. Furthermore, it is not clear how a PDA can perform this check for more than one pair $C_i\#C_{i+1}$. So, we need to modify the format of our traces, so that every odd-numbered configuration is written backwards. Thus, the trace would be given as

$$t = \#C_1\#C_2^R\#C_3\# \dots \#C_{k-1}^R\#C_k\#,$$

or, if there are an even number of configurations in the trace, the trace would be written as

$$t = \#C_1\#C_2^R\#C_3\# \dots \#C_{k-1}\#C_k^R\#.$$

Our basic plan is still valid. Indeed, there will be an accepting trace in this modified format if and only if T accepts w.

Verifying two consecutive configurations. Let us build a pushdown automata that reads two configurations $X\#Y^R\#$ and decides if the configuration X does not imply Y. To make things easier, let us first build a PDA that checks that the configuration X does imply the configuration Y.

The PDA P would scan X and push it as it to the stack. As it reads X and read the state written in X, it can push on the stack how the output configuration should look like (there is a small issue with having to write the state on the stack. This can be easily be done by some a few pushes and pops, but this is tedious but manageable). Thus, by the time we are done reading X (when P encounters $\#$), the stack already contains the implied (reversed) configuration of X, let use denote it by Z^R. Now, P just read the input (Y^R) and matches it to the stack content. It accepts if and only if the configuration X implies the configuration Y.

Interestingly, the PDA P is deterministic, and as such, we can complement it (this is not true of a general PDA because of the nondeterminism). Alternatively, just observe that P has a reject state that is arrived to after the comparison fails. In the complement PDA, we just

make this "hell" state into an accept state. Thus, we have a PDA $\overline{P}$ that accepts $X\#Y^R\#$ iff the configuration X does not imply the configuration Y. Now, its easy to modify this PDA so that it accepts the language

$$L_1 = \left\{ \Sigma^*\#X\#Y^R\#\Sigma^* \;\middle|\; \text{Configuration } X \text{ does not imply configuration } Y \right\},$$

which clearly contains only invalid traces. Similarly, we can build a PDA that accepts the language

$$L_2 = \left\{ \Sigma^*\#X^R\#Y\#\Sigma^* \;\middle|\; \text{Configuration } X \text{ does not imply configuration } Y \right\},$$

Putting these two PDAs together, yield a PDA that accepts all strings containing two consecutive configurations, such that the first one does not imply the second one.

Now, since we a PDA for this language, we clearly can build a CFG G_M that accepts all such strings.

Strings with invalid initial configurations. Consider all traces having invalid initial configurations. Clearly, they are generated by strings of the form

$$\left(\Sigma \setminus \{\#, q_0\}\right)^* \#\Sigma^*.$$

Clearly, one can generate a grammar G_I that accepts these strings.

Strings with invalid final configurations. Consider all traces having invalid initial configurations. Clearly, they are generated by strings of the form

$$\Sigma^*\#\left(\Sigma \setminus \{\#, q_{\text{acc}}\}\right)^*.$$

Clearly, one can generate a grammar G_F that accepts these strings.

Putting things together. Clearly, all invalid (i.e., non-accepting) traces of T on w are generated by the grammars G_I, G_M, G_F. Thus, consider the context free grammar $G_{M,w}$ formed by the union of G_I, G_M, G_F.

When T does not accept w, there is no accepting trace for T on w, so $L(G_{M,w})$ (the strings that are not accepting traces) is Σ^*. When T accepts w, there is an accepting trace for T on w, so $L(G_{M,w})$ (the strings that are not accepting traces) is not equal to Σ^*.

2.2.3 The reduction proof

Theorem 2.4 *The language*

$$\mathsf{ALL}_{\mathsf{CFG}} = \left\{ \langle G \rangle \;\middle|\; G \textit{ is a CFG, and } L(G) = \Sigma^* \right\}$$

is undecidable.

Proof: Let us assume, for the sake of contradiction, that the language $\mathsf{ALL}_{\mathsf{CFG}}$ is decidable, and let $\mathbf{deciderAll}_{\mathsf{CFG}}$ be its decider. We will now reduce A_{TM} to it, by building a decider for it as follows.

```
decider_6-A_TM(⟨M, w⟩)
    Check that ⟨M⟩ is syntactically correct TM code
    Compute ⟨G_{M,w}⟩ from ⟨M, w⟩, as described above.
    res ← deciderAll_CFG(⟨G_{M,w}⟩).
    if res == accept then
        reject
    else
        accept
```

Clearly, this is a decider, and indeed if T accepts w, then there exists an accepting trace t showing it. As such, $L(G_{M,w}) = \Sigma^* \setminus \{t\} \neq \Sigma^*$. Thus, $\mathbf{deciderAll}_{\mathsf{CFG}}$ rejects $\langle G_{M,w} \rangle$, and thus $\mathtt{decider}_6\text{-}\mathrm{A}_{\mathsf{TM}}$ accepts $\langle M, w \rangle$.

Similarly, if T rejects w then $L(G_{M,w}) = \Sigma^*$, and as such $\mathbf{deciderAll}_{\mathsf{CFG}}$ accepts $\langle G_{M,w} \rangle$. Implying that $\mathtt{decider}_6\text{-}\mathrm{A}_{\mathsf{TM}}$ rejects $\langle M, w \rangle$.

Thus, $\mathtt{decider}_6\text{-}\mathrm{A}_{\mathsf{TM}}$ is a decider for A_{TM}, which is impossible. We conclude that our assumption, that $\mathsf{ALL}_{\mathsf{CFG}}$ is decidable, is false, implying the claim. ■

Now, suppose that $\mathsf{ALL}_{\mathsf{CFG}}$ is decided by R. We construct a decider for A_{TM} as follows:

2.3 CFG equivalence is undecidable

From the undecidability of $\mathsf{ALL}_{\mathsf{CFG}}$, we can quickly deduce that

$$EQ_{\mathsf{CFG}} = \left\{ \langle G, H \rangle \;\middle|\; G \text{ and } H \text{ are } \mathsf{CFG}\text{s and } L(G) = L(H) \right\}$$

is undecidable. This proof is almost identical to the reduction of E_{TM} to EQ_{TM} that we saw in lecture 21.

Theorem 2.5 *The language EQ_{CFG} is undecidable.*

Proof: Proof by contradiction. Suppose that EQ_{CFG} is decidable and let $\mathbf{deciderEq}_{\mathsf{CFG}}$ be a TM that decides it.

Given an alphabet Σ, it is not hard to construct a grammar F_Σ that generates all strings in Σ^*. E.g. if $\Sigma = \{c_1, c_2, \ldots c_k\}$, then we could use rules:

$$S \to XS \mid \epsilon$$

$$X \to c_1 \mid c_2 \mid \ldots \mid c_k$$

As such, here is a TM $\mathbf{deciderAll}_{\mathsf{CFG}}$ that decides $\mathsf{ALL}_{\mathsf{CFG}}$.

```
deciderAll_CFG(⟨G⟩)
    Σ ← alphabet used by ⟨G⟩
    Compute F_Σ from Σ.
    res ← deciderEq_CFG(⟨G, F_Σ⟩)
    return res
```

It is easy to verify that $\textbf{deciderAll}_{\mathsf{CFG}}$ is indeed a decider. However, we have already shown that $\mathsf{ALL}_{\mathsf{CFG}}$ is undecidable. So this decider $\textbf{deciderAll}_{\mathsf{CFG}}$ can not exist. As such, our assumption that EQ_{CFG} is decidable is false. As such, EQ_{CFG} is undecidable decidable. ■

3 Avoiding PDAs

The proofs we used above are simpler when one uses PDAs. The same argumentation can be done without PDAs by slightly changing the rules. The basic idea is to interleave two configurations together. This is best imagined by thinking about each character as being a tile of two characters. Thus, the following

b	d	x	a	b	q	c	e	b	d	x
b	d	x	a	q'	b	d	e	b	d	x

describes the two configurations $x = \texttt{bdxab}q\texttt{cebdx}$ and $y = \texttt{bdxa}q'\texttt{bdebdx}$. If we are given a TM T with tape alphabet Γ and set of states Q, then the alphabet of the tiles is

$$\widehat{\Sigma} = \left\{ \begin{array}{|c|}\hline \texttt{x} \\ \hline \texttt{y} \\ \hline \end{array} \;\middle|\; x, y \in \Gamma \cup Q \right\}.$$

Note, that in the above example x yields y, which implies that except for a region of three columns the two strings are identical, see

b	d	x	a	b	q	c	e	b	d	x
b	d	x	a	q'	b	d	e	b	d	x

.

Thus, a single step of a TM is no more than a local rewrite of the configuration string.

Given two configurations x, y of T, we will refer to the string resulting from writing them together interleaved over $\widehat{\Sigma}$ as described above as ***pairing***, denoted by $\begin{array}{|c|}\hline \texttt{x} \\ \hline \texttt{y} \\ \hline \end{array}$. Note, that if one of the configurations is shorter than the other, we will pad the other configuration by introducing blanks characters (i.e., ␣) so that they are of the same length.

Lemma 3.1 *Given a TM* T*, one can construct an NFA* M*, such that* M *accepts a pairing* $\begin{array}{|c|}\hline \texttt{x} \\ \hline \texttt{y} \\ \hline \end{array}$ *if and only if* x *and* y *are two valid configurations of* T*, and* $x \mapsto y$*.*

Proof: First making sure x and y are valid configurations when reading the string $s = \begin{array}{|c|}\hline \texttt{x} \\ \hline \texttt{y} \\ \hline \end{array}$ is easy using a DFA (you verify that x contains only a single state in it, and the rest of the

characters of x are from the tape alphabet of x, one also has to do the same check for y). Let refer to the DFAs verifying the x and y parts of s as $\mathtt{M}_x$ and $\mathtt{M}_y$, respectively. Note that $\mathtt{M}_x$ (resp. $\mathtt{M}_y$) reads the string s but ignores the bottom (resp. top) part of each character of s.

As such, we just need to verify that x yields y. To this end, observe that x yields y if and only if they are identical except for three positions where the transitions happens. We build a NFA that verify that the top and bottom parts are equal, till it guess that it needs to rewrite this 3 tile region. It then guesses what is the tile that needs to be written (note, that the transition function of $\mathtt{T}$ specify all valid such tiles), it verifies that indeed thats what in the next three characters of the input, and then it compares the rest of the input. Let this NFA be $\mathtt{M}_=$.

Now, we construct a DFA that accepts the language of $\mathsf{L}(\mathtt{M}_x) \cap \mathsf{L}(\mathtt{M}_y) \cap \mathsf{L}(\mathtt{M}_=)$. Clearly, this DFA accepts the required language. ■

Similarly, it is easy to build a DFA that verifies that the pairing $\overline{\boxed{\begin{array}{|c|}\hline \mathtt{x}^{\mathtt{R}} \\ \hline \mathtt{y}^{\mathtt{R}} \\ \hline \end{array}}}$ is valid and x yields y (according to $\mathtt{T}$). Now, consider an extended execution trace

$$\overline{\boxed{\begin{array}{|c|}\hline \mathtt{C}_1 \\ \hline \mathtt{C}_2 \\ \hline \end{array}}}\boxed{\begin{array}{c}\$\\ \$\end{array}}\overline{\boxed{\begin{array}{|c|}\hline \mathtt{C}_2^{\mathtt{R}} \\ \hline \mathtt{C}_3^{\mathtt{R}} \\ \hline \end{array}}}\boxed{\begin{array}{c}\$\\ \$\end{array}}\overline{\boxed{\begin{array}{|c|}\hline \mathtt{C}_3 \\ \hline \mathtt{C}_4 \\ \hline \end{array}}}\boxed{\begin{array}{c}\$\\ \$\end{array}}\cdots\boxed{\begin{array}{c}\$\\ \$\end{array}}\overline{\boxed{\begin{array}{|c|}\hline \mathtt{C}_{k-2}^{\mathtt{R}} \\ \hline \mathtt{C}_{k-1}^{\mathtt{R}} \\ \hline \end{array}}}\boxed{\begin{array}{c}\$\\ \$\end{array}}\overline{\boxed{\begin{array}{|c|}\hline \mathtt{C}_{k-1} \\ \hline \mathtt{C}_k \\ \hline \end{array}}}.$$

We would like to verify that this encodes a valid accepting trace for $\mathtt{T}$ on the input string w. This would require verifying that following conditions are met.

(i) The trace has the right format of pairings separated with dollar tiles. Can be easily be done by a DFA.

Let L_1 be the language that this DFA accepts.

(ii) Check that $C_1 = q_0 w$ - can be done with a DFA.

Let L_2 be the language that this DFA accepts.

(iii) The last configuration C_k is an accepting configuration. Easily can be done by a DFA.

Let L_3 be the language that this DFA accepts.

(iv) The pairs $\overline{\boxed{\begin{array}{|c|}\hline \mathtt{C}_{2i-1} \\ \hline \mathtt{C}_{2i} \\ \hline \end{array}}}$ and $\overline{\boxed{\begin{array}{|c|}\hline \mathtt{C}_{2i}^{\mathtt{R}} \\ \hline \mathtt{C}_{2i+1}^{\mathtt{R}} \\ \hline \end{array}}}$ are valid pairing such that $C_{2i} \mapsto C_{2i+1}$ and $C_{2i-1} \mapsto C_{2i}$, for all i (again, according to TM. This can be done by a DFA, by Lemma 3.1.

Let L_4 be the language that this DFA accepts.

(v) Finally, we need to verify that the configurations are copied correctly from the bottom of one tile to the top of the next tile.

Let L_5 be the language of all string that their copying is valid.

Clearly, the set of all valid traces of $\mathtt{T}$ on w is the set $\mathsf{L} = \mathsf{L}_1 \cap \mathsf{L}_2 \cap \mathsf{L}_3 \cap \mathsf{L}_4 \cap \mathsf{L}_5$.

We are interested in building a CFG that recognized the complement language $\overline{\mathsf{L}}$, which is the language

$$\overline{\mathsf{L}} = \overline{\mathsf{L}_1} \cup \overline{\mathsf{L}_2} \cup \overline{\mathsf{L}_3} \cup \overline{\mathsf{L}_4} \cup \overline{\mathsf{L}_5}.$$

Now, since $\mathsf{L}_1, \ldots, \mathsf{L}_4$ it is easy to build a CFG that accepts $\overline{\mathsf{L}_1}$, $\overline{\mathsf{L}_2}$, $\overline{\mathsf{L}_3}$ and $\overline{\mathsf{L}_4}$, respectively.

The only problematic language is $\overline{\mathsf{L}_5}$ which is just all strings where there is a consecutive pair of configurations such that the copying failed. That is

$$\cdots \begin{array}{|c|}\hline \$ \\ \hline \$ \\ \hline \end{array} \begin{array}{|c|}\hline \mathtt{x}^{\mathtt{R}} \\ \hline \mathtt{y}^{\mathtt{R}} \\ \hline \end{array} \begin{array}{|c|}\hline \$ \\ \hline \$ \\ \hline \end{array} \begin{array}{|c|}\hline \mathtt{y}' \\ \hline \mathtt{z} \\ \hline \end{array} \cdots$$

where $\left(y^R\right)^R \neq y'$. But if we ignore the rest of the string and top and bottom portions of these two pairings, this is just recognized the language "not palindrome", which we know is CFG. Indeed, the grammar of not-palindrome over an alphabet Γ is

$$\begin{array}{rrcll} \Longrightarrow & \mathsf{S}_2 & \to & \mathtt{x}\mathsf{S}_2\mathtt{x} & \forall \mathtt{x} \in \Gamma \\ & \mathsf{S}_2 & \to & \mathtt{x}\mathsf{C}\mathtt{y} & \forall \mathtt{x}, \mathtt{y} \in \Gamma \text{ and } \mathtt{x} \neq \mathtt{y} \\ & \mathsf{C} & \to & \mathsf{C}\mathtt{x} \mid \mathtt{x}\mathsf{C} & \forall \mathtt{x} \in \Gamma \\ & \mathsf{C} & \to & \$. & \end{array}$$

We now extend this grammar for the extended alphabet $\widehat{\Sigma}$ as follows

$$\begin{array}{rrcll} \Longrightarrow & \mathsf{S}_3 & \to & \begin{array}{|c|}\hline \mathtt{u} \\ \hline \mathtt{x} \\ \hline \end{array} \mathsf{S}_3 \begin{array}{|c|}\hline \mathtt{x} \\ \hline \mathtt{v} \\ \hline \end{array} & \forall u, v, x \in \Gamma_{\mathtt{T}} \\ & \mathsf{S}_2 & \to & \begin{array}{|c|}\hline \mathtt{u} \\ \hline \mathtt{x} \\ \hline \end{array} \mathsf{C} \begin{array}{|c|}\hline \mathtt{y} \\ \hline \mathtt{v} \\ \hline \end{array} & \forall x, y, u, v \in \Gamma_{\mathtt{T}} \text{ and } x \neq y \\ & \mathsf{C} & \to & \mathsf{C} \begin{array}{|c|}\hline \mathtt{x} \\ \hline \mathtt{y} \\ \hline \end{array} \mid \begin{array}{|c|}\hline \mathtt{x} \\ \hline \mathtt{y} \\ \hline \end{array} \mathsf{C} & \forall x, y \in \Gamma_{\mathtt{T}} \\ & \mathsf{C} & \to & \begin{array}{|c|}\hline \$ \\ \hline \$ \\ \hline \end{array}, & \end{array}$$

where $\Gamma_{\mathtt{T}}$ is the tape alphabet of $\mathtt{T}$. Thus, the context-free language

$$\widehat{\Sigma}^* \begin{array}{|c|}\hline \$ \\ \hline \$ \\ \hline \end{array} \mathsf{L}(\mathsf{S}_3) \begin{array}{|c|}\hline \$ \\ \hline \$ \\ \hline \end{array} \widehat{\Sigma}^*$$

is exactly $\overline{\mathsf{L}_5}$. We conclude that $\overline{\mathsf{L}}$ is a context-free language (being the union of 5 context-free/regular languages). Furthermore, $\overline{\mathsf{L}} = \widehat{\Sigma}^*$ if and only if $\mathtt{T}$ rejects w. We conclude the following.

Theorem 3.2 (Restatement of Theorem 2.4.) *The language*

$$\mathsf{ALL}_{\mathsf{CFG}} = \left\{ \langle G \rangle \;\middle|\; G \text{ is a CFG, and } L(G) = \Sigma^* \right\}$$

is undecidable.