ECS 120 Lesson 20 – The Post Correspondence Problem

Oliver Kreylos

Wednesday, May 16th, 2001

Today we will continue yesterday's discussion of reduction techniques by looking at another common strategy, *reduction by computation chains*. We will apply this method to prove the undecidabiliy of the emptiness problem for LBAs, and later to prove the undecidability of a problem not directly related to language theory, the *Post Correspondence Problem (PCP)*.

1 Reduction Using Computation Chains

Another reduction strategy is based on computation chains for Turing Machines. If M is a Turing Machine, and w is an input word, then the computation of M on w can be simulated by a UTM. This UTM will typically store the current configuration of M on its tape, and will update it for every computation step. We can now modify the Universal Turing Machine to write a new configuration for each step instead of modifying the old one. The simulation will work the same way, but after simulation is complete, the UTM's tape will contain a valid chain of IDs for machine M and input word w. Technically, the modified UTM will write the chain as a string $\#\mathrm{ID}_0\#\mathrm{ID}_1\#\mathrm{ID}_2\#\ldots\#\mathrm{ID}_k\#$. If a computation chain leads to a halting configuration, it must be of finite length; if it does not, we will not consider it being valid. We will then use the fact that a computation chain can be checked for validity by an LBA: A computation chain can be invalid for three reasons:

1. The first configuration ID_0 is not an initial configuration,
2. the last configuration ID_k is not a halting configuration, or

3. there is a pair of configurations ID_i, ID_{i+1} that is not related by the turnstile relation: $\mathrm{ID}_i \nvdash \mathrm{ID}_{i+1}$.

All three of these conditions can be checked by a Turing Machine that does not write past its input, i. e., an LBA. Since the acceptance problem for LBAs is decidable, so is the validity of a computation chain for any Turing Machine.

1.1 The Emptiness Problem for LBAs

Though the acceptance problem for LBAs is decidable, not all problems about LBAs are. One undecidable problem is the emptiness of the language accepted by an LBA. To prove this, we will use a reduction from $\mathrm{A_{TM}}$ using computation chains. We will assume that the emptiness problem is decidable, and will construct a "helper" LBA whose language is the set of all accepting computation chains of M on input word w. If this LBA has a non-empty language $L(B)$, there exists a valid computation chain in M on w, which means that w is accepted by M. Otherwise, $L(B)$ is empty, there are no valid computation chains, and M does not accept w. This means we have reduced the problem of deciding whether a Turing Machine M accepts a word w to the problem of deciding whether an LBA B's language $L(B)$ is empty.

Algorithm $\mathrm{A_{TM}} \longrightarrow \mathrm{E_{LBA}}$ On input $\langle M, w \rangle$, where M is a Turing Machine and w is an input word,

1. Construct an LBA B that can check a computation chain of M for validity. On input $\#\mathrm{ID}_0\#\mathrm{ID}_1\#\ldots\#\mathrm{ID}_k\#$,
   (a) Check whether ID_0 is an initial configuration for M and input w.
   (b) Check whether for each $i = 1, 2, \ldots, k : \mathrm{ID}_{i-1} \vdash \mathrm{ID}_i$.
   (c) Check whether ID_k is an accepting configuration of M.
2. Run the decider E for $\mathrm{E_{LBA}}$ on input $\langle B \rangle$. If E accepts, reject; otherwise, accept.

If E accepts $\langle B \rangle$, this means that the language $L(B)$ is empty. Then there exists no valid computation chain for w in M, and the overall machine will reject. Otherwise, it will accept, because there is a valid chain and M would

accept w. Again, the machine B is never actually executed by the algorithm. It is just fed into the decider for the emptiness problem to find out whether its language is empty or not.

2 Problems About Context-Free Languages

Reductions using computation chains can be used to prove that certain problems about context-free languages are undecidable. Without proof, here is a list of undecidable questions about context-free languages:

- Is a context-free language identical to the set Σ^*; in other words, does a context-free language contain all possible strings[1]?
- Are two context-free languages identical?
- Is a context-free grammar ambiguous?
- Is a context-free language inherently ambiguous?

3 The Post Correspondence Problem

All decidable/undecidable problems we have seen so far were directly related to language theory: Acceptance of a word by a Turing Machine, emptiness of a Turing Machine's language, etc. We are now going to see an undecidable problem that is not related to languages and automata and that is a basis to show other problems undecidable. The *Post Correspondence Problem (PCP)* is related to matching strings. It can be viewed as a puzzle: The input is a set of dominos, where each domino has two strings written onto it, one on each side. A typical such domino could look like

$$\left[\frac{\texttt{ab}}{\texttt{c}}\right],$$

and a collection of dominos could look like

$$\left\{\left[\frac{\texttt{ab}}{\texttt{a}}\right], \left[\frac{\texttt{a}}{\texttt{aa}}\right], \left[\frac{\texttt{aa}}{\texttt{b}}\right]\right\}.$$

[1] The "opposite" question, whether a context-free language is empty or not, *is* decidable.

The task of the puzzle is to place dominos from the collection next to each other (repetitions are allowed), such that the two strings created by concatenating the strings on either side of the selected dominos are identical. If such a selection of dominos exists, it is called a *match*. A match for the collection given above would be

$$\left[\frac{\texttt{ab}}{\texttt{a}}\right]\left[\frac{\texttt{aa}}{\texttt{b}}\right]\left[\frac{\texttt{a}}{\texttt{aa}}\right]\left[\frac{\texttt{a}}{\texttt{aa}}\right].$$

More formally, a match for a set of dominos

$$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right] \right\}$$

is a sequence of integers $(i_1, i_2, \ldots, i_n) \in \{1, 2, \ldots, k\}^n$ such that the two strings spelled out by the two sides of the dominos are identical: $t_{i_1} t_{i_2} \ldots t_{i_n} = b_{i_1} b_{i_2} \ldots b_{i_n}$. The Post Correspondence Problem is now to detect whether a given set of dominos P has a match or not. Expressed as a language, this problem becomes $\text{PCP} = \left\{ \langle P \rangle \;\middle|\; P \text{ has a match} \right\}$.

The problem seems simple enough, and something that might actually come up in practice, but as it turns out there is no algorithm solving it – the language PCP is not decidable. To prove this claim, we reduce the acceptance problem for Turing Machines, A_{TM}, to PCP. The reduction is based on computation chains. The basic idea for this proof is simple, but there are many technical details involved. To avoid some problems later, let us first assume that machine M never attempts to move its head beyond the left end of its tape. This can be achieved by modifying the description of M in a straightforward way (marking the beginning of the tape and checking for the left end). Also, instead of looking at the PCP as stated, we will consider a modified version of the PCP where the domino starting the match is predetermined, by fixing i_1 to be one; this means any match must be started with the first domino in P. We will later remove this modification.

The input to the reduction algorithm is a word $\langle M, w \rangle$: The encoding of a Turing Machine M and an input word w. The algorithm has to decide whether M accepts w. To reduce this to (the modified version of) PCP, we will construct a set of dominos P that will have a match exactly if M accepts w. If P can be constructed, the answer of a machine deciding PCP to the set P is the answer to the acceptance problem.

We construct P in such a way that any match in P will imply the existence of a valid computation chain for w in M. We will encode computation chains

in the same way as discussed above. P will be constructed such that the first domino in P, the only domino that can begin a match, corresponds to the initial configuration of M on input word w of length $|w| = n$:

$$\left[\frac{\#}{\#q_0\ w_1 w_2 \ldots w_n \#}\right]$$

We will create all other dominos in a way that a match can only exist if the string spelled out by the top and bottom strings is a valid computation chain in M on word w. To do this, let us reconsider how the configuration of a Turing Machine can change in a single step of computation.

In every step, the Turing Machine reads the current character, transitions to a new state, writes a character to the tape, and moves the head either left or right. This means that only a very small portion of the tape is actually changed – the area right around the head. Let us assume that the current configuration is $u\ q\ av$, where $u, v \in \Gamma^*$, $a \in \Gamma$, and $q \in Q$, and that $\delta(q, a) = (q', b, \mathrm{R})$. Then the next configuration is $ub\ q'\ v$. Everything is identical, only the substring "$q\ a$" is replaced by the substring "$b\ q'$." Similarly, if the current configuration is $ua\ q\ bv$, where $b \in \Gamma$ is another tape symbol, and $\delta(q, b) = (q', c, \mathrm{L})$, then the next configuration is $u\ q'\ acv$. Again, only the substring "$a\ q\ b$" is replaced by the substring "$q\ ac$." We can create dominos for each possible transition allowed by M's transition function:

$$\left[\frac{q\ a}{b\ q'}\right] \quad , \quad \text{for every } \delta(q, a) = (q', b, \mathrm{R}), \text{ and}$$

$$\left[\frac{a\ q\ b}{q'\ ac}\right] \quad , \quad \text{for every } \delta(q, b) = (q', c, \mathrm{L}).$$

The rest of the configuration is not changed in a single step; we add dominos to P that allow "copying" the tape symbols:

$$\left[\frac{a}{a}\right] \quad , \quad \text{for every } a \in \Gamma.$$

Finally, we have to match the separator # between two adjacent configurations; sometimes, if the tape head is at the right end of the input, we also have to add one additional blank symbol at the end of the configuration. For these two cases, we add two dominos

$$\left[\frac{\#}{\#}\right], \left[\frac{\#}{\sqcup\#}\right].$$

Let us now consider an example to show how these dominos enforce the creation of a valid computation chain. Let the input word be $w = \texttt{abcd}$. The first domino we choose is the first one in the set (by modification of the PCP):

$$\left[\frac{\texttt{\#}}{\texttt{\#}q_0\ \texttt{abcd\#}}\right]$$

Let us further assume that $\delta(q_0, \texttt{a}) = (q_3, \texttt{d}, \mathrm{R})$. Then the only dominos that can be chosen to match the lower string are

$$\left[\frac{\texttt{\#}}{\texttt{\#}q_0\ \texttt{abcd\#}}\right]\left[\frac{q_0\ \texttt{a}}{\texttt{d}\ q_3}\right]\left[\frac{\texttt{b}}{\texttt{b}}\right]\left[\frac{\texttt{c}}{\texttt{c}}\right]\left[\frac{\texttt{d}}{\texttt{d}}\right]\left[\frac{\texttt{\#}}{\texttt{\#}}\right]$$

Spelled out, the current top string is $\texttt{\#}\ q_0\ \texttt{abcd\#}$, and the current bottom string is $\texttt{\#}\ q_0\ \texttt{abcd\#d}\ q_3\ \texttt{bcd\#}$. As we can see, the next configuration in the string spelled out by the lower half of the dominos is the next configuration machine M goes to when reading word w. To match up the "dangling end" of the lower string, the next dominos that must be chosen will create the third configuration of M on the bottom string, and so on. At some point, if M accepts w, the last configuration in the lower string will be an accepting configuration, i. e., a configuration including the state q_{accept}. To now match the rest of the string, we add dominos that allow the tape head to "eat up" the leftover tape contents. We add dominos

$$\left[\frac{a\ q_{\text{accept}}}{q_{\text{accept}}}\right], \left[\frac{q_{\text{accept}}\ a}{q_{\text{accept}}}\right] \quad , \quad \text{for every } a \in \Gamma.$$

and a final domino

$$\left[\frac{q_{\text{accept}}\texttt{\#\#}}{\texttt{\#}}\right]$$

For example, an accepting configuration is matched up in the following way:

$$\left[\frac{\texttt{\#}}{\texttt{\#}q_{\text{accept}}\ \texttt{bc\#}}\right]\left[\frac{q_{\text{accept}}\ \texttt{b}}{q_{\text{accept}}}\right]\left[\frac{\texttt{c}}{\texttt{c}}\right]\left[\frac{\texttt{\#}}{\texttt{\#}}\right]\left[\frac{q_{\text{accept}}\ \texttt{c}}{q_{\text{accept}}}\right]\left[\frac{\texttt{\#}}{\texttt{\#}}\right]\left[\frac{q_{\text{accept}}\texttt{\#\#}}{\texttt{\#}}\right]$$

Using this construction of P, the modified PCP exactly has a match if there is a valid computation chain for input w in machine M. In other words, if there were a decider for the modified PCP, we could then use that decider and the construction of P to decide the acceptance problem for Turing Machines, which we know to be undecidable. This contradiction proves that the modified PCP is undecidable as well.

To show the same for the original PCP[2], we have to find some way to enforce that any match starts with the domino representing the initial configuration of M on input w. We can achieve this by interleaving the characters spelling out the strings on each domino with a new symbol $\diamondsuit$. Say that $d \in P$ is any domino except the initial one and the final one. Say that the top string is $t_1 t_2 \dots t_n$ and the bottom string is $b_1 b_2 \dots b_m$. We then replace the domino

$$\left[\frac{t_1 t_2 \dots t_n}{b_1 b_2 \dots b_m}\right] \quad \text{with} \quad \left[\frac{\diamondsuit t_1 \diamondsuit t_2 \diamondsuit \dots \diamondsuit t_n}{b_1 \diamondsuit b_2 \diamondsuit \dots \diamondsuit b_m \diamondsuit}\right].$$

Furthermore, we replace the initial domino

$$\left[\frac{\#}{\#q_0\ w_1 w_2 \dots w_n \#}\right] \quad \text{with} \quad \left[\frac{\diamondsuit \#}{\diamondsuit \# \diamondsuit q_0 \diamondsuit w_1 \diamondsuit w_2 \diamondsuit \dots \diamondsuit w_n \diamondsuit \# \diamondsuit}\right]$$

and the final domino

$$\left[\frac{q_{\text{accept}} \# \#}{q_{\text{accept}} \#}\right] \quad \text{with} \quad \left[\frac{\diamondsuit q_{\text{accept}} \diamondsuit \# \diamondsuit \# \diamondsuit}{\# \diamondsuit}\right].$$

This way, the only domino that can begin a match is the first one (every other domino has top and bottom strings that differ in the first character), and the only domino that can end a match is the final one (every other domino has top and bottom strings that differ in the last character). Feeding this modified P into the original PCP will result in a match exactly if M accepts w, as intended.

4 Applications of the Post Correspondence Problem

Since the PCP is a problem about matching strings, it is not directly related to the theory of languages and automata. This allows using the PCP to prove that a variety of problems from other areas are undecidable. One example that is related to languages is the ambiguity problem for context-free grammars: The question whether a context-free grammar G has two different parse trees for the same word $w \in L(G)$ can be used to decide the PCP; therefore, it must be undecidable.

[2]Asking the original PCP about the set P we constructed would always answer yes, independent of computation of w. There is a match using only one domino: Any of those we constructed where the top and bottom strings are equal.