CS21004 Formal Languages and Automata Theory, Spring 2019–2020

A Note on Reductions and Undecidability

This article clarifies some potential misconceptions about reduction proofs, and about undecidable and unsemidecidable problems.

1. *That a problem* Π *is undecidable does not mean that all instances of* Π *are undecidable.*

In order to see what this means, consider the halting problem HP. We have proved by a diagonalization argument that no decider for HP can exist. However, special cases can be handled by a decider. One such special case is the class of Turing machines which never move their heads left. If M is such a machine, and w is the input to M, then one can simulate M for a finite number of steps. If M does not halt by that time, this implies M has already entered a loop, and there is no point continuing the simulation any further.

As a very specific example, consider the Turing machine K with the following specifications. $Q = \{s,t,r\}$ (with the states having the usual meanings). $\Sigma = \{0,1\}$. $\Gamma = \{\triangleright, 0, 1, \square\}$. The transition function has only the following entries. Since t and r are halting states, there are no transitions from these states.

$$\begin{aligned}
\delta(s,\triangleright) &= (s,\triangleright,R),\\
\delta(s,0) &= (t,0,R),\\
\delta(s,1) &= (r,1,R),\\
\delta(s,\square) &= (r,\square,R).
\end{aligned}$$

It is clear that

$$\mathscr{L}(K) = \{w \in \{0,1\}^* \mid w \text{ starts with } 0\}.$$

Indeed, K is a total TM. It halts on all inputs.

In general, for a machine M with no left movement of the head, its behavior on an input w can be decided as follows. Run M for $1+|w|+|Q|$ steps. If M halts by that time, we are done. If not, M must have entered a loop, because after the first $1+|w|$ steps, the head of M enters the blank portion of the tape, and running M for another $|Q|$ steps ensures the repetition of a state.

These are solvable (easy) instances of HP. But Turing machines are allowed to move left, so this decider cannot be applied on all Turing machines. Indeed, the diagonalization argument establishes that no matter how *intelligent* a decider is, it cannot solve *all* instances of HP.

2. *What if a reduction* HP $\leqslant_m \Pi$ *converts all instances of* HP *to only easy* (*solvable*) *instances of* Π?

This is impossible. If all converted instances are easy (decidable by a decider), then the reduction mechanism followed by running that decider gives a decider for HP. But HP is proved to be undecidable.

3. *How can* VALCOMP(M,w) *be infinite if* M *is a DTM and halts on* w?

There are two reasons. First, we are allowed to repeat the halting configuration (at the end) as many times as we want. Moreover, in each configuration, we can append as many blank symbols ($\begin{smallmatrix}\square\\ \_\end{smallmatrix}$ to be more precise) as we want.

4. *Does this infiniteness of* VALCOMP(M,w) *makes it a non-CFL? Is* VALCOMP(M,w) *always a non-CFL?*

Of course, for VALCOMP(M,w) to be a non-CFL, it has to be infinite. But VALCOMP(M,w) is not necessarily a non-CFL. Consider the machine K (in the above example) and the input 0 for K. I show that VALCOMP$(K,0)$ is not only a CFL, it is also regular. Its initial configuration is $C_0 = \begin{matrix}\triangleright\ 0\\ s\ -\end{matrix}\begin{pmatrix}\square\\ -\end{pmatrix}^*$. After one step, the configuration changes to $C_1 = \begin{matrix}\triangleright\ 0\\ -\ s\end{matrix}\begin{pmatrix}\square\\ -\end{pmatrix}^*$. The next configuration is a halting configuration $C_2 = \begin{matrix}\triangleright\ 0\ \square\\ -\ -\ t\end{matrix}\begin{pmatrix}\square\\ -\end{pmatrix}^*$. It follows that VALCOMP$(K,0)$ is the language of the regular

expression $\#\, C_0\, \#\, C_1\, \#\, C_2\, \#\, (C_2\, \#\,)^*$. In fact, it is easy to see that VALCOMP(K,w) is regular (and so a CFL) for all w.

5. *But then what does it mean to say that* VALCOMP(M,w) *is not a CFL, in general*?

This is a very crucial question. Indeed, it is mandatory for a non-empty VALCOMP(M,w) to be a non-CFL in certain proofs, like in the proof of undecidability of whether the complement of a CFL is a CFL. I now propose a generic construction that, given any M, generates an NTM N such that $\mathscr{L}(M) = \mathscr{L}(N)$, and M halts on w if and only if N has finite valid computation histories.

Mark the accept state t and the reject state r of M as non-halting. Add a new accept state t' and a new reject state r'. When M reaches t, it keeps on staying in that state, keeps moving right after replacing every tape cell by a new symbol ■ until it nondeterministically switches to the new accept state t' and halts. The behavior at state r and the final nondeterministic transition to r' are analogous.

6. VALCOMP(N,w) *is a non-CFL for any M and w.*

The proof is based on a strong version of Ogden's lemma. Let L be a CFL, and k a pumping-lemma constant (this constant is specific to this version of the pumping lemma). Take a string $z \in L$ such that

(a) z contains e excluded positions,
(b) z contains $d \geqslant k(e+1)$ distinguished positions.

There may be positions which are neither distinguished nor excluded. Then, z has a decomposition of the form

$$z = uvwxy,$$

such that

(1) vx contains at least one distinguished position but no excluded positions,
(2) vwx contains at most d distinguished positions,
(3) $z_i = uv^iwx^iy \in L$ for all $i \geqslant 0$.

(The version of Ogden's lemma covered in the class corresponds to $e = 0$.)

Let M halt on w. Assume that VALCOMP(N,w) is a CFL. We use the notations of the above lemma. Now, N has halting histories. However, these histories may be arbitrarily long depending upon for how long N plans to stay in the state t or r. Take a sufficiently long valid computation history $\#C_0\#C_1\#C_2\#\ldots\#C_L\#$ of N, mark the #'s as excluded positions, and all other positions as distinguished. Also assume that there are no trailing blank symbols in each configuration (unless the head is there). That is, $\begin{array}{c}\square\\ q\end{array}$ may be the last symbol in a configuration for any state $q \neq t', r'$, but $\begin{array}{c}\square\\ \_\end{array}$ may not be. We choose L sufficiently large so that the condition $d \geqslant k(e+1)$ is satisfied. It is easy to verify that z_0 or z_2 contains (at least) one configuration that has an incorrect number of states or is not consistent with the preceding or the following one. So $z_0, z_2 \notin$ VALCOMP(N,w), a contradiction. ◄

The implication of this result is that in a reduction proof which requires a non-empty $\overline{\text{VALCOMP}(M,w)}$ to be a non-CFL, we may assume that we first convert M to N and generate the CFG for $\overline{\text{VALCOMP}(N,w)}$.

Similar comments can be made for the non-CFL-ness of VALCOMP-ALT(M,w).