CS 121 Section 8

Harvard University

Fall 2013

Overview

This week we will focus on reviewing the core concepts involved with undecidability, reducibility, Rice's theorem, incompleteness of mathematics, and so on.

1 Concept Review

1.1 Undecidability

By a cardinality argument, we know that almost all languages are undecidable. This argument, however, does not give us an explicit construction. The following theorem does just that.

Theorem 1.1. *The language* $\{\langle M, w\rangle : M \text{ accepts the input } w\}$ *is not decidable.*

Proof. Assume $\{\langle M, w\rangle : M \text{ accepts the input } w\}$ is decidable, then the language $D = \{\langle M\rangle : M \text{ accepts } \langle M\rangle\}$ is decidable, hence $\overline{D} = \{\langle M\rangle : M \text{ does not accepts } \langle M\rangle\}$ is decidable. Suppose $\overline{D}$ is decidable by M_1, then $\langle M_1\rangle \in \overline{D}$ iff M_1 accepts $\langle M_1\rangle$ iff $\langle M_1\rangle \in D$, which is a contradiction. (This is the standard diagonalization argument.) □

1.2 Reducibility

Definition 1.1. *A function* $f : \Sigma_1^* \to \Sigma_2^*$ *is computable if there is a Turing machine such that for every input* $w \in \Sigma_1^*$, *M halts with just* $f(w)$ *on its tape.*

Definition 1.2. *A reduction of* $L_1 \subseteq \Sigma_1^*$ *to* $L_2 \subseteq \Sigma_2^*$ *is a computable function* $f : \Sigma_1^* \to \Sigma_2^*$ *such that, for any* $w \in \Sigma^*$, $w \in L_1$ *if and only if* $f(w) \in L_2$, *and we write* $L_1 \leq_m L_2$.

Intuitively, L_1 reduces to L_2 means that L_1 is not harder than L_2. More formally, we can express this intution in the following lemma.

Lemma 1.1. *If* $L_1 \leq_m L_2$ *and* L_1 *is undecidable, then so it* L_2.

1.3 Rice's theorem

Theorem 1.2 (Rice's theorem)**.** *Let $\mathcal{P}$ be any subset of the class of r.e. languages such that $\mathcal{P}$ and its complement are both nonempty. Then the language $L_{\mathcal{P}} = \{\langle M \rangle : L(M) \in \mathcal{P}\}$ is undecidable.*

Intuitively, Rice's theorem states that Turing machines can not test whether another Turing machine satisfies a (nontrivial) property. For example, let $\mathcal{P}$ be the subset of the recursively enumerable languages which contains the string a. Then Rice's theorem claims that there is no Turing machine which can decide whether a Turing machine accepts a.

2 Exercises

Exercise 2.1. *Reductions can be tricky to get the hang of, and you want to avoid "going the wrong way" with them. In which of these scenarios does $L_1 \leq_m L_2$ provide useful information (and in those cases, what may we conclude)?*

(a) L_1's decidability is unknown and L_2 is undecidable

Nothing

(b) L_1's decidability is unknown and L_2 is decidable

L_1 is decidable. This is in Sipser.

(c) L_1 is undecidable and L_2's decidability is unknown

Undecidable. Corollary to the above question.

(d) L_1 is decidable and L_2's decidability is unknown

Nothing

Exercise 2.2. *Argue that $\leq_m$ is a transitive relation.*

Let f be the function that reduces A to B, i.e., $A \leq_m B$ by f, and let g be the function that reduces B to C, i.e., $B \leq_m C$ by g. Then $w \in A \iff f(w) \in B$. Furthermore, $x \in B \iff g(x) \in C$. It follows that for every $w \in A$, $g(f(w)) \in C$, and furthermore, for every $x \in C$, there exists a $y \in B$ such that $g(y) = x$, and for every $y \in B$, there exists a $w \in A$ such that $f(w) = x$, so that for every $x \in C$, there exists a $w \in A$ with $g(f(w)) = x \in C$. Thus $w \in A$ iff $g(f(w)) \in C$, so that $A \leq_m C$.

Exercise 2.3. *Determine, with proof, whether the following languages are decidable.*

(a) $L = \{\langle M, x\rangle :$ *At some point it its computation on* x, M *re-enters its start state*$\}$

Undecidable. Assume for the sake of contradiction that this language is decidable. We will show that a decider for this language can decide the halting problem. Given an input $\langle M, w\rangle$, *modify* M *so that the start state is split into two states, and the new start state has no incoming transitions except for those we will specifically add in this construction. Also, change the accept and reject states so that they are no longer accept / reject states, but just normal states that (effectively) epsilon transition to the new start state. Make new accept / reject states that are inaccessible. Now the machine re-enters its start state iff the original machine would have halted. This construction was a finite transformation, and something a TM could do, so by combining this machine with a decider for the language in question, we could decide the halting problem. Contradiction.*

(b) $L = \{\langle x, y\rangle : f(x) = y\}$ *where* f *is a fixed computable function.*

Decidable. Given $\langle x, y\rangle$, *compute* $f(x)$ *and compare it to* y.

(c) $\mathrm{CF}_{TM} = \{\langle M\rangle : L(M)$ *is context-free*$\}$

Undecidable. The subset of languages which are context-free and the subset of languages which are not are both non-empty. Apply Rice's theorem.

(d) $L = \{\langle M\rangle : M$ *calculates* $\pi\}$, *meaning that* $M : \mathbb{N} \to \mathbb{N}$, *and* $M(i)$ *is equal to the* i*-th digit in the decimal expansion of* π.

Undecidable. Assume $\{\langle M\rangle : M$ *calculates* $\pi\}$ *is decidable, then the language* $D = \{\langle M\rangle : M$ *accepts* $\langle M\rangle\}$ *is decidable, hence* $\overline{D} = \{\langle M\rangle : M$ *does not accepts* $\langle M\rangle\}$ *is decidable. Suppose* $\overline{D}$ *is decidable by* M_1, *then* $\langle M_1\rangle \in \overline{D}$ *iff* M_1 *accepts* $\langle M_1\rangle$ *iff* $\langle M_1\rangle \in D$, *which is a contradiction. (This is the standard diagonalization argument.)*

Exercise 2.4. *Show* $\{G : G$ *is a CFG generating* $x\} \leq_M \{G : G$ *is a CFG generating* $xy\}$.

Given a grammar G, *define* $f(G)$ *to be the grammar which is the same, except that it has a new start variable* S' *and a new rule* $S' \to Sy$ *(where* S *was the old start variable). If* $x \in L(G)$, *then* $x \in L(f(G))$, *since the same derivation that leads to* x *from* S *can be used to derive* xy *from* Sy. *If* $xy \in L(f(G))$, *then since* Sy *must have been a working string after the first step of the derivation,* x *must have been generated from* S, *which means* $x \in L(G)$.