ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Prof. Dr. Andreas Podelski
Matthias Heizmann
Christian Schilling

July 15th-16th, 2014

7. Lecture
Computer Science Theory

Chapter V – Non-computable functions – undecidable problems (pp. 97-122)

§2 Concrete undecidable problem: halting for Turing machines (pp. 101-107)

Short repetition: We have shown that K, the *special halting problem* for Turing machines, is undecidable.

$$K = \{bw_\tau \in B^* \mid \tau \text{ applied to } bw_\tau \text{ halts}\}$$

Note that it is important that we talk about all TMs and all input words here. Given a TM and a word, there is always a trivial deciding TM (although we may not know which one, but we are only interested in the existence).

✎ hints for exercises 1, 3 on sheet 7 ✐

The rest of the course will be centered around the following definition.

Definition 2.4 Let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be languages. Then L_1 is *reducible* to L_2, shortly $L_1 \leq L_2$, if there is a total computable function $f : \Sigma_1^* \to \Sigma_2^*$ so that for all $w \in \Sigma_1^*$ it holds that: $w \in L_1 \Leftrightarrow f(w) \in L_2$. We also write: $L_1 \leq L_2$ *using* f. We will see some examples and use the same idea in the last chapter again.

Definition 2.6 The *(general) halting problem* for Turing machines is the language

$$H = \{bw_\tau 00u \in B^* \mid \tau \text{ applied to } u \text{ halts}\}.$$

Theorem 2.7 H is undecidable.

Definition 2.8 The *blank tape halting problem* for Turing machines is the language

$$H_0 = \{bw_\tau \in B^* \mid \tau \text{ applied to the blank tape halts}\}.$$

Theorem 2.9 H_0 is undecidable. As a summary, talking about all Turing machines seems impossible. Let us restrict ourselves to one fixed Turing machine.

Definition 2.10 The *halting problem* for a given Turing machine τ is the language

$$H_\tau = \{w \in B^* \mid \tau \text{ applied to } u \text{ halts}\}.$$

For many TMs this language is decidable. But not for all of them, namely those which read and interpret TMs themselves.

Definition 2.11 A Turing machine τ_{uni} with the input alphabet B is called *universal* if for the function $h_{\tau_{uni}}$ computed by τ_{uni} the following holds:

$$h_{\tau_{uni}}(bw_\tau 00u) = h_\tau(u),$$

i.e., τ_{uni} can simulate every Turing machine τ applied to input string $u \in B^*$.

Theorem 2.13 $H_{\tau_{uni}}$ is undecidable.

Thus we have shown the following chain: $K \leq H = H_{\tau_{uni}} \leq H_0$.

hints for exercises 2, 4 on sheet 7

§3 Recursive enumerability (pp. 107-110)

We soften our notions of computation and decision in order to capture the new problems we have seen.

Definition 3.1 A language $L \subseteq \Sigma^*$ is called *recursively enumerable*, shortly *r.e.*, if $L = \emptyset$ or there exists a total (Turing-)computable function $\beta : \mathbb{N} \to \Sigma^*$ with

$$L = \beta(\mathbb{N}) = \{\beta(0), \beta(1), \beta(2), \dots\},$$

i.e., we can enumerate all elements with a Turing machine.

Definition 3.2 A language $L \subseteq \Sigma^*$ is called *semi-decidable* if the *partial characteristic function of* L

$$\psi_L : \Sigma^* \rightharpoonup \{1\}$$

is computable. The partial function ψ_L is defined as follows:

$$\psi_L(v) = \begin{cases} 1 & \text{if } v \in L \\ \text{undef.} & \text{otherwise} \end{cases}$$

Remark For all languages $L \subseteq \Sigma^*$ it holds that:

(a) L is semi-decidable $\Leftrightarrow$ L is Turing-acceptable.

(b) L is decidable $\Leftrightarrow$ L and $\overline{L}$ are semi-decidable.

Lemma 3.3 For all languages $L \subseteq \Sigma^*$ it holds that: L is recursively enumerable $\Leftrightarrow$ L is semi-decidable.

Theorem 3.4 For all languages $L \subseteq \Sigma^*$ the following statements are equivalent:

(a) L is recursively enumerable.

(b) L is the range of results of a Turing machine τ, i.e.,

$$L = \{v \in \Sigma^* \mid \exists w \in \Sigma^* \text{ with } h_\tau(w) = v\}.$$

(c) L is semi-decidable.

(d) L is the halting range of a Turing machine τ, i.e.,

$$L = \{v \in \Sigma^* \mid h_\tau(v) \text{ exists}\}.$$

(e) L is Turing-acceptable.

(f) L is Chomsky-0.

Corollary 3.5 For all languages $L \subseteq \Sigma^*$ it holds that: L is decidable (recursive) $\Leftrightarrow$ L and $\overline{L} = \Sigma^* \setminus L$ are recursively enumerable.

✎ hints for exercise 5 on sheet 7 ✐

Lemma 3.6 Let $L_1 \leq L_2$. Then it holds: If L_2 is recursively enumerable, then L_1 is also recursively enumerable.

Theorem 3.7 $H_0 \subseteq B^*$ is recursively enumerable.

Theorem 3.8 The halting problems K, H, H_0, and $H_{\tau_{uni}}$ are recursively enumerable, but not decidable. Their complementary problems are not recursively enumerable.

✎ hints for exercise 6 on sheet 7 ✐

§4 Automatic program verification (pp. 110-112)

We skip this part in the interest of time.

Summary: The program verification problem (also called model checking problem) is given as follows:
Given: program $\mathcal{P}$ and specification $\mathcal{S}$ $\quad (\mathcal{S} \subseteq \mathcal{T}_{B,B})$
Question: Does $\mathcal{P}$ satisfy the specification $\mathcal{S}$?

It is undecidable except for the trivial cases $\mathcal{S} = \emptyset$ and $\mathcal{S} = \mathcal{T}_{B,B}$.

§5 Grammar problems and Post correspondence problem (pp. 112-119)

We skip this part in the interest of time.

Summary: Another undecidable problem is introduced. It is used to prove results of the following section.

§6 Results on undecidability of context-free languages (pp. 120-122)

We skip this part in the interest of time.

Summary: For context-free languages the intersection problem, the equivalence problem, the inclusion problem, and the ambiguity problem are shown undecidable.

Prime number encoding of pairs

For the proof of Lemma 3.3 we needed a way to encode pairs into natural numbers. Here we describe how this is possible. For this we exploit the fact that every positive integer has a unique decomposition into prime numbers (see Wikipedia).
Let $w \in \Sigma^*$ be a word and $k \in \mathbb{N}$ be a natural number.

We want to know the tuple (w, k) that is encoded by some natural number n (note: not every number encodes such a pair, but this can be checked).

In other words: Given a natural number n, we want to decode it to get the pair (w, k) (or we want to know if no such pair exists).

1) In a first step, we show how we can decode a word w from a natural number.

Let $\Sigma = \{a_1, \ldots, a_m\}$ and $nr : \Sigma \longrightarrow \mathbb{N}$ be a function returning the index number of some symbol in Σ, i.e., $nr(a_i) = i$ for $i = 1, \ldots, n$.

Let p_j be the j-th prime number, i.e.,

$$p_1 = 2, p_2 = 3, p_3 = 5, p_4 = 7, p_5 = 11, \ldots$$

Let us write w as $w = w_1 w_2 \ldots w_\ell$ if w has length ℓ ($w = \varepsilon$ if $\ell = 0$).

The prime number encoding of w is the function $\pi : \Sigma^* \longrightarrow \mathbb{N}$ with

$$\begin{aligned} \pi(\varepsilon) &= 1 \\ \pi(w_1 \ldots w_\ell) &= p_1^{nr(w_1)} \cdot \ldots \cdot p_\ell^{nr(w_\ell)} = \prod_{i=1}^{\ell} p_i^{nr(w_i)} \end{aligned}$$

Example: Let $\Sigma = \{a_1, a_2, a_3, a_4\}$. The number $n = 720$ is uniquely decomposed into the prime numbers $2 \cdot 2 \cdot 2 \cdot 2 \cdot 3 \cdot 3 \cdot 5$, which can be written as $2^4 \cdot 3^2 \cdot 5^1$. Thus it encodes the word $w = a_4 a_2 a_1$, because

$$720 = 2^4 \cdot 3^2 \cdot 5^1 = p_1^4 \cdot p_2^2 \cdot p_3^1 = \pi(a_4 a_2 a_1).$$

2) Now we can decode pairs (w, k). For this we use the same idea again. We define the function

$$\pi_2 : \mathbb{N}^2 \longrightarrow \mathbb{N}$$

for which we need to first encode w into a number $\pi(w)$ (see above)

$$\pi_2(\pi(w), k) = p_1^{\pi(w)} \cdot p_2^k$$

Example: We continue the example. The number $n = 2^{720} \cdot 3^{50}$ (it is too big to write down) is already (uniquely) decomposed into prime numbers. Thus it encodes the pair (w, k) for $w = a_4 a_2 a_1$ and $k = 50$, because

$$n = 2^{720} \cdot 3^{50} = p_1^{720} \cdot p_2^{50} = \pi_2(720, 50) = \pi_2(\pi(a_4 a_2 a_1), 50).$$