TGI WS 2016/17 Sheet 5

Sebastian Löhr, Thure Dührsen 2017-02-14

Exercise 1

Let Σ be an alphabet. Let $L_1, L_2 \subseteq \Sigma^*$.
Let L_2 be a turing decidable language and $L_1 \subseteq L_2$.

Claim

If L_2 is a Turing-decidable language and $L_1 \subseteq L_2$,
then L_1 is not – let me repeat that: NOT – necessarily Turing-decidable.

Proof

If L_1 is any undecidable language, and if $L_2 = \Sigma^*$, then $L_1 \subseteq L_2$ and L_1 is undecidable and L_2 is decidable, because a decider for L_2 simply ignores its input and halts in an accepting configuration.

tdu: Is there anything wrong about this?

seb: Good example!

FIND OUR FAILED ATTEMPTS BELOW — PLEASE DO NOT GRADE THEM! We could also compile a list of common misconceptions like these together, if the other students agree to share theirs...

It exists a turing machine M, that accepts every word $w \in L_2$ and rejects every word $v \notin L_2$. Therefore, M will accept every word $w' \in L_1 \subseteq L_2$. Therefore, L_1 is not Turing-decidable.

tdu: What is wrong about this?
seb: M will accept words that are in L_2 but not in L_1
tdu: Correct. L_1 is then a proper subset of $L(M)$, but it should be exactly $L(M)$ for the proof to be correct, if we take out the word *not* in the sentence "Therefore, L_1 is not Turing-decidable." So this first attempt is wrong in two ways: it tries to prove the opposite of what we want to show.

It exists a turing machine M, that accepts every word $w \in L_2$ and rejects every word $v \notin L_2$. Therefore, M will accept every word $w' \in L_1 \subseteq L_2$. [1]
If $L_1 \subseteq L_2$, it could be that $L_1 \subsetneq L_2$. Then the set $L_2 \setminus L_1$ is not empty. Let x be a member of $L_2 \setminus L_1$. This means $x \in L_2$, therefore M will accept x, but at the same time, x is not a member of L_1, yet it is still accepted by M.
Therefore L_1 is not Turing-decidable.

tdu: What is wrong about this?
seb: Better, why should this one be wrong?
tdu: I'll leave you hanging on this until Thursday evening. ;) Hint: Recall the definition of *Turing-decidability* and take note of the quantifiers in that definition.
seb: Not by M turing decidable, which doesn't mean it's not turing-decidable at all...
tdu: That's exactly it.

□

Exercise 2

Let $\Sigma = \{0, 1\}$ be an alphabet.
"Does a given regular expression R over Σ describe a language $L \in \Sigma^*$, such that L contains at least one string starting with 0 and ending with 1?"

Claim

a) $L' = \{v01w : v, w \in \Sigma^*\}$ **is the formal given language described by** R **over** Σ**.**

b) The language L' **is turing-decidable.**

Proof

a)

The only words we won't accept from the alphabet Σ^* are the ones, which doesn't contain the string "01"anywhere.
So the following language will do it:

$$L' = \{v01w : v, w \in \Sigma^*\}$$

is the formal given language described by R over Σ.

b)

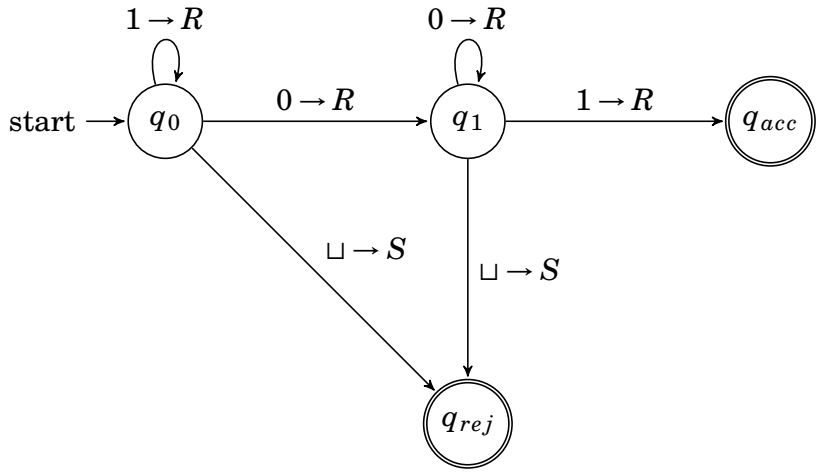
The language L' is turing-decidable, because we can makeup an turing-decider which decides it:

$$T = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$$

$$Q = \{q_0, q_1, q_{acc}, q_{rej}\}$$

$$\Gamma = \{0, 1, \sqcup\}$$

δ hold's the following graph.



L will move the write-read-head to the left.

R will move the write-read-head to the right.

S will stop the write-read-head. □

Exercise 3

Let Σ, Δ be two alphabets, $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Delta^*$ languages, where $L_1 \leq L_2$ holds. Specify which of the following properties are valid by giving a short explanation.

(a) If L_1 is Turing-decidable, then a Turing machine that decides L_1 is not necessarily able to decide L_2.

(b) If L_1 is Turing-decidable, then L_2 does not have to be.

(c) If L_1 is not Turing-decidable, then L_2 is not Turing-decidable, either.

Proof

Given that $L_1 \leqslant L_2$, choose $f : \Sigma^* \to \Delta^*$ such that

$$\forall\, w \in \Sigma^* :\ w \in L_1 \iff f(w) \in L_2 \tag{1}$$

holds.

(a) Given that L_1 is decidable, choose a Turing machine A that decides L_1.

The set Σ^* is infinite, because $\Sigma \neq \emptyset$ and Σ^* contains words of any length. In particular, $|\Sigma^*| \geqslant 2$.

Now what do we know about f? Nothing, except for (1). In particular, because $|\Sigma^*| \geqslant 2$, the function f does not have to be injective.

So consider the case that f is not injective. Then there exist w, $x \in \Sigma^*$ such that $w \neq x$ and $f(w) = f(x)$. If $w \in L_1$, then $f(w) \in L_2$, and therefore $f(x) \in L_2$. But, again, we know nothing about x except that $x \in \Sigma^*$ and $f(x) \in L_2$. In particular, we have no means to decide[2] whether or not $x \in L_1$, so that

$$x \in L_1 \iff f(x) \in L_2$$

is NOT a valid conclusion.

(b) Set L_1 as turing decidable language. L_2 is the language that holds all Diophantine equations meant like in Hilbert's tenth problem. So each element of the language L_2 describes the solvability of an Diophantine equation and the language holds all possible equations. $L_1 \leq L_2$ holds in this case and we find an bijective function f', which assigns to any word $w \in L_1$ a word $f'(w) = d \in L_2$. We well know, that the solvability of Diophantine equations is an not turing decidable problem, therefore we proved the claim.

[2] in the "true-or-false" sense, not in the "Turing-decide" sense

(c) We show the contrapositive, i.e.

$$\text{if } L_2 \text{ is decidable, then } L_1 \text{ is decidable.}$$

So let's consider the case that L_2 is decidable. Choose a decider B for L_2. Let $w \in \Sigma^*$. Then $y := f(w) \in \Delta^*$.

Now do the following:

Algorithm 1: Turing machine A that decides L_1

Input: $w \in \Sigma^*$

Output: If $y \in L_1$, halt in an accepting configuration;
if $y \notin L_1$, halt in a rejecting configuration

```
1 y ← f(w)
2 Set B to run on input y
  /*This always halts, because B is a decider for L_2          */
3 if B accepts y then
4 |   accept w
5 else
6 |   reject w
```

□

Exercise 4

Let $\Sigma := \{0, 1\}$.

Fix the language

$$HALT_{TM}^{\Sigma} := \left\{ \langle M \rangle \,:\, M \text{ is a Turing machine over } \Sigma \text{ and } L(M) = \Sigma^* \right\} \quad .$$

Claim

$HALT_{TM}^{\Sigma}$ is not Turing-decidable.

Proof

Suppose $HALT_{TM}^{\Sigma}$. We will then show that the halting problem

$$HALT_{TM}^{\varepsilon} := \{\langle M \rangle \,:\, M \text{ is a Turing machine over } \Sigma \text{ and } M \text{ halts on the empty input}\}$$

is decidable, which is a contradiction.

For every Turing machine M over Σ, let M' be the Turing machine that does the following:

(a) erase the input to M

(b) Simulate M running on empty input

(c) If M stops in an accepting configuration, accept; if M stops in a rejecting configuration, reject.

Then, M' is also a Turing machine over Σ.

Also, let N be Turing machine over Σ that does the following:

(a) erase the input to N

(b) loop forever

Define $f : \Sigma^* \to \Sigma^*$ by

$$x \mapsto \begin{cases} \langle M' \rangle & \text{if } x = \langle M \rangle \text{ for some } M \in HALT_{TM}^{\varepsilon} \\ \langle N \rangle & \text{otherwise} \end{cases}$$

for all $x \in \Sigma^*$.

Then, for all $x \in \Sigma^*$, we have

$$\begin{aligned} x \in HALT_{TM}^{\varepsilon} &\iff x = \langle M \rangle \text{ for a TM } M \text{ that halts on } \varepsilon \\ &\iff f(x) = \langle M' \rangle \text{ for a TM } M \text{ that halts on all inputs} \\ &\iff f(x) \in HALT_{TM}^{\Sigma} \end{aligned}$$

□

Exercise 5

Let $\Sigma := \{0, 1\}$ and

$$\mathscr{X} := \{\langle M \rangle : M \text{ accepts at least one word}\} \subseteq \Sigma^* \ .$$

Claim

(a) $\mathscr{X}$ is not turing-decidable.

(b) $\mathscr{X}$ is turing-recognisable.

Proof

(a) Suppose $\mathscr{X}$ is turing decidable.
In this case also $\overline{\mathscr{X}} = \{\langle M \rangle : M \text{ accepts no word}\}$ is turing decidable.[3]
But $\overline{\mathscr{X}}$ is not Turing-decidable, as was mentioned in the lecture.

(b) It suffices to show that there exists a Turing machine T that recognises $\mathscr{X}$ – that is, T must accept every word in $\mathscr{X}$, and for every word $x \in \Sigma^* \setminus \mathscr{X}$, either T rejects the word x or T does not halt.

What we are trying to do below is capture every word in Σ^*, and for every such word x we try M on input x any finite number of steps to see if it accepts. Once we have found a word that is accepted by M, we are happy. If no such word exists, we are guaranteed to not reach an accepting configuration and therefore we *recognise* the language $\mathscr{X}$.

We avoid infinite computation sequences on any particular word by enumerating pairs $(x, j) \in \Sigma^* \times \mathbb{N}_0$ – where x is the word in question and j is the number of steps M should run on x – in a way reminiscent of the way used to show the countability of $\mathbb{N}_0 \times \mathbb{N}_0$.

Let $p : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$ be the Cantor pairing function, that is,

$$\forall\ (x, y) \in \mathbb{N}_0 \times \mathbb{N}_0 : \quad p(x, y) := y + \frac{(x + y)\,(x + y + 1)}{2} \quad ,$$

and set $\varphi := p^{-1} : \mathbb{N}_0 \to \mathbb{N}_0 \times \mathbb{N}_0$, which is possible because p is bijective (cf. Otmar Spinas, Mathematische Logik, WS 2015/16).

The set Σ^* is countable. Let $E : \mathbb{N}_0 \to \Sigma^*$ be an enumeration[4] of Σ^* such that

i) $\forall\ x, y \in \Sigma^* : \quad |x| \leqslant |y| \implies E^{-1}(x) \leqslant E^{-1}(y)$ and

[3] *Sipser, Michael:* Introduction to the Theory of Computation (Thrid Edition), S. 209f §4.22
[4] that is, $E : \mathbb{N}_0 \to \Sigma^*$ is a bijection

ii) $\forall\, x, y \in \Sigma^* : \quad |x| = |y| \implies$
$\big[x$ does not come after y in dictionary order $\implies E^{-1}(x) \leqslant E^{-1}(y)\big]$

Now the following Turing machine should do what we want.

Algorithm 2: Turing machine T that accepts $\mathscr{X}$

Input: $w \in \Sigma^*$

Output: If $w \in \mathscr{X}$, halt in an accepting configuration;
if $w \notin \mathscr{X}$, either reject w or compute forever

```
1  if w is not a valid encoding of a Turing machine then
2      reject w
3  M ← the Turing machine encoded by w
4  i ← 0
5  while true do
6      (a, j) ← φ(i)
7      x ← E(a)
8      Set an instance of M to run on input x
9      if M accepts x after exactly j + 1 steps then
10         accept x
11     else
12         ignore this instance of M; also ignore your electric bill
13     i ← i + 1
```

□