Theory of Computation 13
Undecidability in Formal Languages

Frank Stephan

Department of Computer Science

Department of Mathematics

National University of Singapore

fstephan@comp.nus.edu.sg

Repetition 1

Polynomials $\mathbf{P}(\mathbb{N})$ and $\mathbf{P}(\mathbb{Z})$
A function $\mathbf{f}(\mathbf{x}, \mathbf{y_1}, \ldots, \mathbf{y_n})$ is in $\mathbf{P}(\mathbb{Z})$ if it is a sum of products of integer constants and variables $\mathbf{x}, \mathbf{y_1}, \ldots, \mathbf{y_n}$. This function is in $\mathbf{P}(\mathbb{N})$ iff all the integer constants occurring are in $\mathbb{N}$.

Diophantine Sets
A set $\mathbf{A} \subseteq \mathbb{N}$ is Diophantine iff there is $\mathbf{f} \in \mathbf{P}(\mathbb{Z})$ such that $\mathbf{x} \in \mathbf{A} \Leftrightarrow \exists \mathbf{y_1}, \ldots, \mathbf{y_n} \in \mathbb{N}\, [\mathbf{f}(\mathbf{x}, \mathbf{y_1}, \ldots, \mathbf{y_n}) = \mathbf{0}]$.

Hilbert's Tenth Problem [1900]
Construct a Decision Procedure for Diophantine sets.

Theorem [Matiyasevich 1970]
A set is Diophantine iff it is r.e.; thus it can be undecidable.

Repetition 2

Definition 12.11
A set $\mathbf{A} \subseteq \mathbb{N}$ is called arithmetic iff there is a formula using existential $(\exists)$ and universal $(\forall)$ quantifiers over variables such that all variables except for $\mathbf{x}$ are quantified and that the predicate behind the quantifiers only uses Boolean combinations of polynomials from $\mathbf{P}(\mathbb{N})$ compared by $<$ and $=$ in order to evaluate the formula.

Theorem [Church 1936; Turing 1936; Tarski 1936]
Arithmetic sets can be undecidable; the halting problem (suitably coded) is an example of such a set.

The Entscheidungsproblem

Question 1928
Hilbert asked whether one can check whether a sentence in first-order logic is valid, that is, true in all logical structures which can be defined. For example, $\forall \mathbf{x} \exists \mathbf{y} [\mathbf{y} < \mathbf{x}]$ is not valid, as it is not true in $(\mathbb{N}, <)$ and it applies to all structures which have a relation called $<$. The formula $\forall \mathbf{x}\, \forall \mathbf{y}\, [\mathbf{x} < \mathbf{y} \text{ or } \neg(\mathbf{x} < \mathbf{y})]$ is an example of a valid formula.

Theorem [Church 1936; Turing 1936]
The Entscheidungsproblem is undecidable.

This proof also works by coding up the machine computations and show that the corresponding halting set is undecidable; if σ codes the computations of machine $\mathbf{e}$ on input $\mathbf{e}$ and τ consists of sufficiently many axioms of $(\mathbb{N}, +, \cdot)$ then $\tau \rightarrow \sigma$ is valid iff the coded Turing machine $\mathbf{e}$ halts on input $\mathbf{e}$.

Repetition 3

Proving Undecidability of Arithmetics

Code machine simulation into an arithmetic formula:

- Code the run of a Registermachine in Arithmetics;
- Make sure that the configurations of the Register machine at the various steps can be decoded with an arithmetic formula;
- Construct a formula which checks for each two successive configurations that the next one is obtained by going one step in the register program from the previous configuration;
- Construct a formula which checks that the initial configuration codes the input and the last one codes the output and is in a halting line number;
- Combine all this to one arithmetic formula which is true iff there is a run and a way to code this run such that the input produces the correct output.

Repetition 4

An index set is a set for each functions either either all or no indices. It is based on acceptable numberings.

Definition 12.17: Acceptable Numbering [Gödel 1931]
A numbering $\varphi_{\mathbf{e}}$ of partial functions is a partial-recursive function $\mathbf{e}, \mathbf{x} \mapsto \varphi_{\mathbf{e}}(\mathbf{x})$. A numbering is acceptable iff for every further numbering ψ there is a recursive function $\mathbf{f}$ such that, for all $\mathbf{e}$, $\psi_{\mathbf{e}} = \varphi_{\mathbf{f}(\mathbf{e})}$.

That is, $\mathbf{f}$ translates “indices” or “programs” of ψ into “indices” or “programs” of φ which do the same.

The universal functions for register machines and for Turing machines constructed by Turing and others are actually acceptable numberings.

Repetition 5

Theorem 12.19 [Rice 1953]
Let φ be an acceptable numbering and $\mathbf{I}$ be an index set (with respect to φ).

(a) The set $\mathbf{I}$ is recursive iff $\mathbf{I} = \emptyset$ or $\mathbf{I} = \mathbb{N}$.

(b) The set $\mathbf{I}$ is recursively enumerable iff there is a recursive enumeration of finite lists $(\mathbf{x_1}, \mathbf{y_1}, \ldots, \mathbf{x_n}, \mathbf{y_n})$ of conditions such that every index $\mathbf{e}$ satisfies that $\mathbf{e} \in \mathbf{I}$ iff there is a list $(\mathbf{x_1}, \mathbf{y_1}, \ldots, \mathbf{x_n}, \mathbf{y_n})$ in the enumeration for which $\varphi_\mathbf{e}(\mathbf{x_1}) = \mathbf{y_1}$ and $\ldots$ and $\varphi_\mathbf{e}(\mathbf{x_n}) = \mathbf{y_n}$.

Corollary 12.20
Let $\mathbf{I} = \{\mathbf{e} : \forall \mathbf{x}\, [\varphi_\mathbf{e}(\mathbf{x})$ is defined$]\}$. The set $\mathbf{I}$ of indices of total functions is arithmetic and not recursively enumerable.

Repetition 6

Observation 12.21
If A, B are sets and B is recursively enumerable and if there is a recursive function g with $x \in A \Leftrightarrow g(x) \in B$ then A is also recursively enumerable. Similarly, if B is recursive then so is A.

Definition 12.22
A set A is many-one reducible to a set B iff there is a recursive function g such that, for all x, $x \in A \Leftrightarrow g(x) \in B$.

Theorem
A set is recursively enumerable iff it is many-one reducible to the halting problem. A set is recursive iff it is many-one reducible to the set of odd numbers.

One can prove undecidability of a set B by finding a set A known to be undecidable such that one can construct a many-one reduction from A to B.

Counter Automata

Counter automata are modifications of register machines.

- The counters (= registers) have much more restricted operations: One can add or subtract 1 or compare whether they are 0. Initival values of all counters is 0.
- Like a pushdown automaton, one can read one symbol from the input at a time; depending on this symbol, the automaton can go to the corresponding line. One makes the additional rule that a run of the counter automaton is only valid iff the full input was read.
- The counter automaton can either terminate in lines with the special commands “ACCEPT” and “REJECT” or signal “REJECT” by running forever.

Goal: Configurations and runs are more compatible with grammars while automaton is still able to simulate everything.

Example

Counter automaton accepts words which have, in all initial parts, at least as many $\mathbf{0}$ as $\mathbf{1}$.

Line 1: Counter Automaton Zeroone;
Line 2: Input Symbol – Symbol $\mathbf{0}$: Goto Line 3; Symbol $\mathbf{1}$: Goto Line 4; No further Input: Goto Line 7;
Line 3: $\mathbf{R_1 = R_1 + 1}$; Goto Line 2;
Line 4: If $\mathbf{R_1 = 0}$ Then Goto Line 6;
Line 5: $\mathbf{R_1 = R_1 - 1}$; Goto Line 2;
Line 6: REJECT;
Line 7: ACCEPT.

Rune of automaton with input $\mathbf{001}$ and $\mathbf{001111000}$:

```
Line:   1 2 3 2 3 2 4 5 2 7  |... 5 2 4 5 2 4 6
Input:    0   0   1     -    |...   1     1
R1:     0 0 0 1 1 2 2 2 1 1  |... 2 1 1 1 0 0 0
```

Adding on Counter Machine

Counter Automaton Translation for $\mathbf{R_1 = R_2 + R_3}$.

Line 1: ... // $\mathbf{R_4}$ is $\mathbf{0}$;
Line 2: If $\mathbf{R_1 = 0}$ Then Goto Line 4;
Line 3: $\mathbf{R_1 = R_1 - 1}$; Goto Line 2;
Line 4: If $\mathbf{R_2 = 0}$ Then Goto Line 6;
Line 5: $\mathbf{R_4 = R_4 + 1}$; $\mathbf{R_2 = R_2 - 1}$; Goto Line 4;
Line 6: If $\mathbf{R_4 = 0}$ Then Goto Line 8;
Line 7: $\mathbf{R_1 = R_1 + 1}$; $\mathbf{R_2 = R_2 + 1}$; $\mathbf{R_4 = R_4 - 1}$; Goto Line 6;
Line 8: If $\mathbf{R_3 = 0}$ Then Goto Line 10;
Line 9: $\mathbf{R_4 = R_4 + 1}$; $\mathbf{R_3 = R_3 - 1}$; Goto Line 8;
Line 10: If $\mathbf{R_4 = 0}$ Then Goto Line 12;
Line 11: $\mathbf{R_1 = R_1 + 1}$; $\mathbf{R_3 = R_3 + 1}$; $\mathbf{R_4 = R_4 - 1}$; Goto Line 10;
Line 12: ... // Continue with next command, $\mathbf{R_4}$ is $\mathbf{0}$ again;

Subtracting from Constant

Counter Automaton Translation for $\mathbf{R_1 = 2 - R_2}$.

Line 1: ... // Previous Operation
Line 2: If $\mathbf{R_1 = 0}$ Then Goto Line 4;
Line 3: $\mathbf{R_1 = R_1 - 1}$; Goto Line 2;
Line 4: $\mathbf{R_1 = R_1 + 1}$; $\mathbf{R_1 = R_1 + 1}$;
Line 5: If $\mathbf{R_2 = 0}$ Then Goto Line 10;
Line 6: $\mathbf{R_1 = R_1 - 1}$; $\mathbf{R_2 = R_2 - 1}$;
Line 7: If $\mathbf{R_2 = 0}$ Then Goto Line 9;
Line 8: $\mathbf{R_1 = R_1 - 1}$;
Line 9: $\mathbf{R_2 = R_2 + 1}$;
Line 10: ... // Continue with next command;

Quiz

Quiz 13.3

Provide counter automaton translations for the following commands:

- $\mathbf{R_1 = 2}$;
- $\mathbf{R_2 = R_2 + 3}$;
- $\mathbf{R_3 = R_3 - 2}$.

Write them in a way that a counter $\mathbf{R_k}$ having the value $\mathbf{0}$ does not perform $\mathbf{R_k = R_k - 1}$.

Exercises

Exercise 13.4
Provide a translation for a subtraction: $\mathbf{R_1 = R_2 - R_3}$. Here the result is $\mathbf{0}$ in the case that $\mathbf{R_3}$ is greater than $\mathbf{R_2}$. The values of $\mathbf{R_2, R_3}$ after the translated operation should be the same as before.

Exercise 13.5
Provide a translation for a conditional jump: If $\mathbf{R_1 \leq R_2}$ then Goto Line 200. The values of $\mathbf{R_1, R_2}$ after doing the conditional jump should be the same as before the translation of the command.

Corollary 13.6
Every language recognised by a Turing machine or a register machine can also be recognised by a counter machine. In particular there are languages recognised by counter machines for which the membership problem is undecidable.

Main Result

Theorem 13.7

If $\mathbf{K}$ is recognised by a counter machine then there are deterministic context-free languages $\mathbf{L}$ and $\mathbf{H}$ and a homomorphism $\mathbf{h}$ such that $\mathbf{K} = \mathbf{h}(\mathbf{L} \cap \mathbf{H})$. In particular, $\mathbf{K}$ is generated by some grammar.

Proof

The main idea of the proof is the following: One makes $\mathbf{L}$ and $\mathbf{H}$ to be computations such that for $\mathbf{L}$ the updates after an odd number of steps and for $\mathbf{H}$ the updates after an even number of steps is checked; furthermore, one intersects one of them, say $\mathbf{H}$ with a regular language in order to meet some other, easy to specify requirements on the computation.

Furthermore, $\mathbf{h}(\mathbf{L} \cap \mathbf{H})$ will get out the input words from the valid counter automaton computations.

Coding Configurations

The simulated counter automaton $\mathbf{M}$ has registers $\mathbf{R_1, R_2, \ldots, R_n}$ and lines $\mathbf{1, 2, \ldots, m}$. A configuration consists of the input read (if any), the line number of the current statement and the register contents before the current statement is done.

Code is $\mathbf{b \cdot 3^{LN} \cdot 4^x}$

Here $\mathbf{b} \in \{\varepsilon, \mathbf{0}, \mathbf{1}, \mathbf{2}\}$: ε – no input processed; $\mathbf{2}$ – input exhausted; $\mathbf{0}, \mathbf{1}$ – respective input bit read.

$\mathbf{4^x}$: $\mathbf{x = p_1^{R_1} \cdot p_2^{R_2} \cdot \ldots \cdot p_n^{R_n}}$ where $\mathbf{p_1, p_2, \ldots, p_n}$ are the first $\mathbf{n}$ primes.

So if $\mathbf{R_1 = 3}$, $\mathbf{R_3 = 1}$ and $\mathbf{R_4 = 1}$ and all other registers are $\mathbf{0}$ then $\mathbf{x = 2^3 \cdot 5 \cdot 7 = 280}$.

Update Set

Let $\mathbf{I}$ be the set of possible configurations,

$$\mathbf{I} = \{\varepsilon, \mathbf{0}, \mathbf{1}, \mathbf{2}\} \cdot \{\mathbf{3}, \mathbf{33}, \ldots, \mathbf{3^m}\} \cdot \{\mathbf{4}\}^+$$

where some illegal values of $\mathbf{x}$ are not checked (like $\mathbf{p^3_{n+1}}$).

Let $\mathbf{J} = \{\mathbf{v} \cdot \mathbf{w} : \mathbf{v}, \mathbf{w} \in \mathbf{I}$ and counter automaton goes from configuration $\mathbf{v}$ to configuration $\mathbf{w}$ in one step$\}$.

Next slides: $\mathbf{J}, \mathbf{J}^*, \mathbf{I} \cdot \mathbf{J}^*$ are deterministic context-free. The deterministic pushdown automaton for $\mathbf{J}$ is explained for representative cases of configurations $\mathbf{v}, \mathbf{w}$.

General outline: The pushdown automaton stores values of $\mathbf{b}, \mathbf{3^{LN}}$ in memory and counts up the 4 of $\mathbf{v}$ in stack and counts down the 4 of $\mathbf{w}$ in stack, sometimes with different speed to check multiplications / divisions by $\mathbf{p_k}$.

Increment

Line $\mathbf{i}$: $\mathbf{R_k = R_k + 1}$;
In this case, one has that the configuration update must be of the form

$$\{\mathbf{3^i}\} \cdot \{\mathbf{4}\}^{\mathbf{x}} \cdot \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \varepsilon\} \cdot \{\mathbf{3^{i+1}}\} \cdot \{\mathbf{4}\}^{\mathbf{x \cdot p_k}}$$

and the deterministic pushdown automaton checks whether the new number of $\mathbf{3}$ is one larger than the old one and whether when comparing the second run of $\mathbf{4}$ those are $\mathbf{p_k}$ times many of the previous run, that is, it would count down the stack only after every $\mathbf{p_k}$-th $\mathbf{4}$ and keep track using the state that the second number of $\mathbf{4}$ is a multiple of $\mathbf{p_k}$.

Decrement

Line $\mathbf{i}$: $\mathbf{R_k = R_k - 1}$;
In this case, one has that the configuration update must be of the form

$$\{\mathbf{3^i}\} \cdot \{\mathbf{4}\}^{\mathbf{x}} \cdot \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \varepsilon\} \cdot \{\mathbf{3^{i+1}}\} \cdot \{\mathbf{4}\}^{\mathbf{x/p_k}}$$

and the deterministic register machine checks whether the new number of $\mathbf{3}$ is one larger than the old one and whether when comparing the second run of $\mathbf{4}$ it would count down the stack by $\mathbf{p_k}$ symbols for each $\mathbf{4}$ read and it would use the state to check whether the first run of $\mathbf{4}$ was a multiple of $\mathbf{p_k}$ in order to make sure that the subtraction is allowed.

Conditional Branching

Line $\mathbf{i}$: If $\mathbf{R_k = 0}$ then Goto Line $\mathbf{j}$;
In this case, the configuration update must either be of the form

$$\{\mathbf{3^i}\} \cdot \{\mathbf{4}\}^{\mathbf{x}} \cdot \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \varepsilon\} \cdot \{\mathbf{3^j}\} \cdot \{\mathbf{4}\}^{\mathbf{x}}$$

with $\mathbf{x}$ not being a multiple of $\mathbf{p_k}$ or it must be of the form

$$\{\mathbf{3^i}\} \cdot \{\mathbf{4}\}^{\mathbf{x}} \cdot \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \varepsilon\} \cdot \{\mathbf{3^{i+1}}\} \cdot \{\mathbf{4}\}^{\mathbf{x}}$$

with $\mathbf{x}$ being a multiple of $\mathbf{p_k}$. Being a multiple of $\mathbf{p_k}$ can be checked by using the state and can be done in parallel with counting; the preservation of the value is done accordingly.

Processing Input

Line $\mathbf{i}$: If input symbol is $\mathbf{0}$ then goto Line $\mathbf{j_0}$; If input symbol is $\mathbf{1}$ then goto Line $\mathbf{j_1}$; If input is exhausted then goto Line $\mathbf{j_2}$;
Now the configuration update must be of one of the form

$$\mathbf{u} \cdot \{\mathbf{3^i}\} \cdot \{\mathbf{4}\}^{\mathbf{x}} \cdot \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \varepsilon\} \cdot \{\mathbf{3^{j_u}}\} \cdot \{\mathbf{4}\}^{\mathbf{x}}$$

for some $\mathbf{u} \in \{\mathbf{0}, \mathbf{1}, \mathbf{2}\}$ and the deterministic pushdown automaton can use the state to memorise $\mathbf{u}, \mathbf{i}$ and the stack to compare the two occurrences of $\mathbf{4^x}$. Again, if the format is not adherred to, the pushdown automaton goes into an always rejecting state and ignores all future input.

General Remarks

The pushdown automaton maintains a stack of the form $\mathbf{S}$ or $\mathbf{ST}$ or $\mathbf{STU}^+$ where the $\mathbf{T}$ and $\mathbf{U}$ are used to count the $\mathbf{4}$ and the $\mathbf{T}$ is a symbol signalling that it is just above $\mathbf{S}$ (in order to manage the state correctly). The automaton can easily be adjusted to handle $\mathbf{J}^*$ in place of $\mathbf{J}$: After each checking of $\mathbf{J}$, it has in the case of success again to have the stack $\mathbf{S}$ and the last $\mathbf{4}$ processed from the entry in $\mathbf{J}$ will follow up with a digit from $\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}$ for the next entry from the next configuration.

The language $\mathbf{I} \cdot \mathbf{J}^*$ is also recognisable by a deterministic pushdown automaton, as the entry from $\mathbf{I}$ consists of digits from $\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}$ followed by at least one $\mathbf{4}$ with the next entry from $\mathbf{J}$ again be starting with one of $\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}$.

For the regular languages $\{\varepsilon\} \cup \mathbf{I}$ and $\mathbf{R}$, it holds that $\mathbf{L} = \mathbf{J}^* \cdot (\mathbf{I} \cup \{\varepsilon\})$ and $\mathbf{H} = (\mathbf{I} \cdot \mathbf{J}^* \cdot (\mathbf{I} \cup \{\varepsilon\})) \cap \mathbf{R}$ are also deterministic context-free.

Choosing R and Conclusion

The permitted runs are from

$$\mathbf{L} \cap \mathbf{H} = (\mathbf{J}^* \cdot (\mathbf{I} \cup \{\varepsilon\})) \cap (\mathbf{I} \cdot \mathbf{J}^* \cdot (\mathbf{I} \cup \{\varepsilon\})) \cap \mathbf{R}$$

and $\mathbf{R}$ codes that the last line number is that of a line having the command ACCEPT and that the first line number is $\mathbf{1}$ and the initial value of all registers is $\mathbf{0}$ and that once a $\mathbf{2}$ is read from the input (for exhausted input) then all further attempts to read an input are answered with $\mathbf{2}$. For example, if the lines $\mathbf{5}$ and $\mathbf{8}$ carry the command ACCEPT then

$$\mathbf{R} = (\{\mathbf{34}\} \cdot \mathbf{I}^* \cdot \{\mathbf{3^5}, \mathbf{3^8}\} \cdot \{\mathbf{4}\}^+) \cap (\{\mathbf{0}, \mathbf{1}, \mathbf{3}, \mathbf{4}\}^* \cdot \{\mathbf{2}, \mathbf{3}, \mathbf{4}\}^*).$$

Furthermore, $\mathbf{h}(\mathbf{0}) = \mathbf{0}$, $\mathbf{h}(\mathbf{1}) = \mathbf{1}$, $\mathbf{h}(\mathbf{2}) = \varepsilon$, $\mathbf{h}(\mathbf{3}) = \varepsilon$, $\mathbf{h}(\mathbf{4}) = \varepsilon$. As the $\mathbf{0}, \mathbf{1}$ are only from the input read, $\mathbf{K} = \mathbf{h}(\mathbf{L} \cap \mathbf{H})$.

Exercises

Exercise 13.8

In the format of the proof before and with respect to the sample multi counter machine from the beginning of today's lecture, give the encoded version (as word from $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}\}^+$) of the run of the machine on the input $\mathbf{001}$.

Exercise 13.9

In the format of the proof before and with respect to the sample multi counter machine from the beginning of today's lecture give the encoded version (as word from $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}\}^+$) of the run of the machine on the input $\mathbf{001111000}$.

Generated by Grammar = R.E.

Theorem 13.10
A set $K \subseteq \Sigma^*$ is recursively enumerable iff it is generated by some grammar. In particular, there are grammars for which it is undecidable which words they generate.

Proof.
If K is generated by some grammar, then every word w has a derivation $S \Rightarrow v_1 \Rightarrow v_2 \Rightarrow \ldots \Rightarrow v_n$ in this grammar. An algorithm can check, by all possible substitutions, whether $v_m \Rightarrow v_{m+1}$. Thus there is a partial recursive f which on input $S \Rightarrow v_1 \Rightarrow v_2 \Rightarrow \ldots \Rightarrow v_n$ checks whether all steps of the derivation are correct and whether $v_n \in \Sigma^*$; if so, then f outputs v_n else f is undefined. K is the range of f.

For the converse direction, if K is recursively enumerable then K is recognised by a Turing machine and then K is recognised by a counter automaton and then K is generated by some grammar.

Undecidable Questions

Corollary 13.11
The following questions are undecidable:

$(\mathbf{a})$ Given a grammar and a word, does this grammar generate the word?

$(\mathbf{b})$ Given two deterministic context-free languages by deterministic push down automata, does their intersection contain a word?

$(\mathbf{c})$ Given a context-free language given by a grammar, does this grammar generate $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}\}^*$?

$(\mathbf{d})$ Given a context-sensitive grammar, does its language contain any word?

Let $\mathbf{K}$ be the halting problem with inputs from $\{\mathbf{0}, \mathbf{1}\}^*$ and $\mathbf{L}, \mathbf{H}$ simulating the computations of a fixed counter machine which recognises $\mathbf{K}$. Choosing a grammar for $\mathbf{K}$ answers item $(\mathbf{a})$.

Items (b), (c), (d)

One can compute for $\mathbf{w} \in \{\mathbf{0}, \mathbf{1}\}^*$ a deterministic pushdown automaton which recognises

$$\mathbf{H_w} = \mathbf{H} \cap (\{\mathbf{3}, \mathbf{4}\}^* \cdot \{\mathbf{b_1}\} \cdot \{\mathbf{3}, \mathbf{4}\}^* \cdot \{\mathbf{b_2}\} \cdot \ldots \cdot \{\mathbf{3}, \mathbf{4}\}^* \cdot \{\mathbf{b_n}\} \cdot \{\mathbf{2}, \mathbf{3}, \mathbf{4}\}^*)$$

so that $\mathbf{L} \cap \mathbf{H_w}$ contains all accepting computations which read $\mathbf{b_1 b_2 \ldots b_n = w}$. This gives item $(\mathbf{b})$.

Item $(\mathbf{c})$ follows from studying $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}\}^* - (\mathbf{L} \cap \mathbf{H_w})$ which has a context-free grammar which can be computed from the deterministic pushdown automata for $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}\}^* - \mathbf{L}$ and $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}\}^* - \mathbf{H_w}$, the language sought for is the union of these two languages.

The last item $(\mathbf{d})$ follows from converting the two pushdown automata for $\mathbf{L}$ and $\mathbf{H_w}$ into a context-sensitive grammar for $\mathbf{L} \cap \mathbf{H_w}$ which contains a member iff $\mathbf{w} \in \mathbf{K}$. Also this test is undecidable.

Post's Correspondence Problem

An instance of Post's Correspondence Problem is a list $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ of pairs of words. Such an instance has a solution iff there is a sequence $k_1, k_2, \ldots, k_m$ of numbers in $\{1, \ldots, n\}$ such that $m \geq 1$ and

$$x_{k_1} x_{k_2} \ldots x_{k_m} = y_{k_1} y_{k_2} \ldots y_{k_m}.$$

For example, the following pairs: (a,a), (a,amanap), (canal,nam), (man,lanac), (o,oo), (panama,a), (plan,nalp). This list has some trivial solutions like $1, 1, 1$ giving aaa for both words. It has also the famous solution $2, 4, 1, 7, 1, 3, 6$ which gives the palindrome as a solution:

```
a      man   a plan a canal panama
amanap lanac a nalp a nam   a
```

The instance (alpha,beta), (gamma,delta) has no solution.

Exercises

Exercise 13.13 and 13.14

For the following instances of Post's Correspondence Problem, determine whether it has a solution:

13.13: (23,45), (2289,2298), (123,1258), (777,775577), (1,9999), (11111,9).

13.14: (1,9), (125,625), (25,125), (5,25), (625,3125), (89,8), (998,9958).

Exercise 13.15

Use Post's Correspondence Problem to prove that it is undecidable whether the intersection of two deterministic context-free languages is non-empty. See the lecture notes for hints.

Non-deterministic machines

One way to implement non-determinism in counter and register machines is to allow multiple targets for a Goto command and the machine chooses just one of the line numbers.

- A function f computes on input x a value y iff there is an accepting run which produces the output y and every further accepting run produces the same output; rejected runs and non-terminating runs are irrelevant in this context.
- A set L is recognised by a non-deterministic machine iff for every x it holds that $x \in L$ iff there is an accepting run of the machine for this input x.

One can use non-determinism to characterise the regular and context-sensitive languages via Turing machines or register machines.

Characterisations

Theorem 13.17
A language L is context-senstive iff there is a Turing machine which recognises L and which modifies only those cells on the Turing tape which are occupied by the input iff there is a non-deterministic register machine recognising the language and a constant c such that the register machine on any run for an input consisting of n symbols never takes in its registers values larger than c^n.

Theorem 13.18
A language L is regular iff there is a non-deterministic Turing machine recognising L and numbers a, b such that the Turing machine makes for each input consisting of n symbols in each run at most $a \cdot n + b$ steps.

Note that a linear time Turing machines can modify the tape on which the input is written while a finite automaton does not have this possibility.

Example 13.19

Assume that a Turing machine has as input alphabet $\{0, 1, \ldots, 9\}$ and additional tape symbol $\sqcup$. This Turing machine does the following: For an input word , it goes four times over the word from left to right and each time it replaces the current word w by v with $3v = w$; if the division by 3 has a remainder, the Turing machine rejects. It accepts iff the final word is from $\{0\}^* \cdot \{110\} \cdot \{0\}^* \cdot \{110\} \cdot \{0\}^*$.

The language recognised by this Turing machine is $\{0\}^* \cdot \{891\} \cdot \{0\}^* \cdot \{891\} \cdot \{0\}^+$ and thus regular.

Exercises

Exercise 13.20

Assume that a Turing machine does the following: It has $\mathbf{5}$ passes over the input word $\mathbf{w}$ and at each pass, it replaces the current word $\mathbf{v}$ by $\mathbf{v/3}$. In the case that during this process of dividing by $\mathbf{3}$ a remainder different from $\mathbf{0}$ occurs for the division of the full word, then computation is aborted as rejecting. If all divisions go through and the resulting word $\mathbf{v}$ is $\mathbf{w/3^5}$ then the Turing machine adds up the digits and accepts iff the sum of digits is exactly $\mathbf{2}$.
Determine a regular expression for this language.

Exercise 13.21

A Turing machine does two passes over a word and divides it the decimal number on the tape each time by $\mathbf{7}$. It then accepts iff the remainders of the two divisions sum up to $\mathbf{10}$.
Construct a dfa for this language.

Exercise 13.22

Assume that a Turing machine works on an input word from $\{0, 1, 2\}^*$ as below.

Initialise $c = 0$ and update c to $1 - c$ whenever a 1 is read. For each symbol do the following replacement:
If $c = 0$ then $1 \to 0, 2 \to 1, 0 \to 0$;
If $c = 1$ then $1 \to 2, 2 \to 2, 0 \to 1$.

```
Before pass 0100101221010210
After pass  0011200222001220
```

The Turing machine accepts if before the pass there are an even number of 1 and afterwards there are an odd number of 1.

Explain what the language recognised by this Turing machine is and why it is regular. As a hint: interpret the numbers as natural numbers in ternary representation and analyse what the tests and the operations do.