1. **Finite automata and regular languages:**

1a. Design a finite automaton for recognizing binary sequences that start with 0. Assume that the input strings contain only symbols 0 and 1. The easiest is to have 3 states:

- the starting state s,
- the state f of all the binary sequences that start with 0,
- the error state e of all the binary sequences that start with 1.

You just need to describe transitions between these states, and which states are final. Show, step-by-step, how your automaton will accept the string 001 (that corresponds to binary number 100).

1b. Explain why in most computers binary numbers are represented starting with the lowest possible digit.

1c. On the example of the above automaton, show how the word 001 can be represented as xyz in accordance with the pumping lemma.

1d. Use a general algorithm to describe a regular expression corresponding to the finite automaton from the Problem 1a. (If you are running out of time, it is Ok not to finish, just eliminate the first state.)

1e-f. The resulting language can be described by a regular expression \(0(0 \cup 1)^*\). Use a general algorithm to transform this regular expression into a finite automaton: first a non-deterministic one, then a deterministic one.

1.6 - They are represented this way to make computations faster, saving time.

1. c

\[
\begin{array}{c|c|c|c|c}
0 & 0 & 1 & 0 & 0 \\
S & 0 & 0 & 0 & 1 \\
\end{array}
\]

Continuation on next page
\[ R'_{ij} = R_{ij} U (R_{ik} R_{kk}^* R_{jk}) \]

\[ R'_{ij} = R_{ns} s U (R_{ns} R_{ee} R_{ee}^* R_{ef}) \]
\[ = \Lambda U (\emptyset) = \Lambda \]

\[ i = ns, k = e, j = s \]

\[ R'_{ij} = R_{sf} U (R_{se} R_{ee}^* R_{ef}) \]
\[ = \emptyset U ((1) (0U1)^* \emptyset) \]
\[ = \emptyset \]

\[ i = f, k = e, j = nf \]

\[ R'_{ij} = R_{fnf} U (R_{fe} R_{ee}^* R_{enf}) \]
\[ = \Lambda U (\emptyset) \]
\[ = \Lambda \]

\[ i = ns, k = e, j = nf \]

\[ R'_{ij} = R_{nsnf} U (R_{ns} R_{ee}^* R_{enf}) \]
\[ = \emptyset U (\emptyset) \]
\[ = \emptyset \]

\[ i = ns, k = e, j = f \]

\[ R'_{ij} = R_{nsf} U (R_{ns} R_{ee}^* R_{ef}) \]
\[ = \emptyset U (\emptyset) \]
\[ = \emptyset \]

*Continue on next page*
\[ R^3 \cup R^3 \cup (R^3 \cup R^3 \cup R^3) \]

\[ i = 5, k = e, j = 3 \]

\[ = R^{ss} \cup (R^{se} \cup R^{ee} \cup R^{es}) \]

\[ = \emptyset \cup ((1 \cup 0 \cup 1) \cup \emptyset) = \emptyset \]

\[ i = 5, k = e, j = f \]

\[ = R^{ff} \cup (R^{fe} \cup R^{ee} \cup R^{ef}) \]

\[ = 0 \cup 0 \cup 0 = 0 \]

\[ i = 5, k = e, j = s \]

\[ = R^{fs} \cup (R^{fe} \cup R^{ee} \cup R^{es}) \]

\[ = \emptyset \cup 0 = \emptyset \]

\[ i = 5, k = e, j = nf \]

\[ = R^{sfnf} \cup (R^{snf} \cup R^{ee} \cup R^{sfnf}) \]

\[ = \emptyset \cup 0 \cup 0 = \emptyset \]

*Now we eliminate S*

\[
\begin{array}{ccc}
R^{ss} & \xrightarrow{0} & R^{sf} \\
\downarrow & & \nearrow \\
0 & & n^f
\end{array}
\]

\[ i = ns, k = s, j = f \]

\[ = R^{nsnf} \cup (R^{nss} \cup R^{ss} \cup R^{sf}) \]

\[ = \emptyset \cup (\emptyset \cup \emptyset) = \emptyset \]

\[ i = ns, k = s, j = nf \]

\[ = R^{nsnf} \cup (R^{nss} \cup R^{ss} \cup R^{snf}) \]

\[ = \emptyset \cup (\emptyset \cup \emptyset) = \emptyset \]

\[ i = 5, k = s, j = f \]

\[ = R^{ff} \cup (R^{fs} \cup R^{ss} \cup R^{sf}) \]

\[ = 0 \cup 0 \cup 0 = 0 \]

\[ i = 5, k = s, j = nf \]

\[ = R^{fnf} \cup (R^{fs} \cup R^{ss} \cup R^{snf}) \]

\[ = 0 \cup 0 = \emptyset \]

*Continue on next page*
eliminate f

\[ \text{ns} \quad o(ou1)^* \quad \text{nf} \]

\[ i = \text{ns}, \quad k = f, \quad j = \text{nf} \]

\[ = Rnsnf \cup (Rnsf Rff^* Rfnf) \]

\[ = \emptyset \cup ((o)(ou1)^* (\lambda)) \]

\[ = o(ou1)^* \]

• 1-e-f on next page
f. Assuming only strings with 0 and 1
1. Finite automata and regular languages:

1a. Design a finite automaton for recognizing binary sequences that start with 0. Assume that the input strings contain only symbols 0 and 1. The easiest is to have 3 states:

- the starting state \( s \),
- the state \( f \) of all the binary sequences that start with 0,
- the error state \( e \) of all the binary sequences that start with 1.

You just need to describe transitions between these states, and which states are final. Show, step-by-step, how your automaton will accept the string 001 (that corresponds to binary number 100).

1b. Explain why in most computers binary numbers are represented starting with the lowest possible digit.

1c. On the example of the above automaton, show how the word 001 can be represented as xyz in accordance with the pumping lemma.

1d. Use a general algorithm to describe a regular expression corresponding to the finite automaton from the Problem 1a. (If you are running out of time, it is Ok not to finish, just eliminate the first state.)

1e-f. The resulting language can be described by a regular expression \( 0(0 \cup 1)^* \). Use a general algorithm to transform this regular expression into a finite automaton: first a non-deterministic one, then a deterministic one.
2. Beyond finite automata: pushdown automata and context-free grammars:

2a. To overcome a presidential veto, the congress needs at least 2/3 of the votes. For example, if the voting record is YNY or YNNYN or YNNYY, where Y stands for Yes, and N for No, the veto will be overcome. Prove that the language consisting of all sequences of Y and N that lead to the overcome is not regular.

2b. Use a general algorithm to transform the finite automaton from the Problem 1a into a context-free grammar (CFG). Show, step-by-step, how this CFG will generate the word 001.

2c. For the context-free grammar from the Problem 2b, show how the word 001 can be represented as uvxyz in accordance with the pumping lemma.

2d. Use a general algorithm to translate the CFG from 2b into Chomsky normal form.

2e. Use a general algorithm to translate the CFG from 2b into an appropriate push-down automaton. Explain, step-by-step, how this automaton will accept the word 001.

2f. Use the general stack-based algorithms to show:

- how the compiler will transform a Java expression 6 - 1 - 2 into inverse Polish (postfix) notation, and
- how it will compute the value of this expression.

2a. Thm. 1. = \{ N^n Y^2^n : n = 0, 1, 2, \ldots \} is not regular.

Proof (by contradiction): Let's assume L is regular. Then by the pumping lemma, \exists p \in L (\text{len}(w) \geq p \implies w = xyz & \text{len}(y) > 0 & \text{len}(xy) \leq p \& \forall i (xy^iz \in L)). Let's take the word w = N^p Y^{2p}. Its length is 3p \geq p, so by the pumping lemma, w = xyz^i. This word starts with xy, \& \text{len}(xy) \leq p, so xy and y are among first p symbols of the word, so y consists of Ns. So when we go from xyz to xyzz, we add Ns but we don't add Ys. So in xyzz we have more than 2/3 Ns (\& not 2/3 Ys), so xyzz \notin L. But by the pumping lemma, xyzz \in L. We get a contradiction. So our assumption was wrong and L is not regular.
$\frac{6-1-2}{6-1-2} = \frac{3}{5}$
2. **Beyond finite automata: pushdown automata and context-free grammars:**

2a. To overcome a presidential veto, the congress needs at least 2/3 of the votes. For example, if the voting record is YNY or YNYNY or YYYNY, where Y stands for Yes, and N for No, the veto will be overcome. Prove that the language consisting of all sequences of Y and N that lead to the overcome is not regular.

2b. Use a general algorithm to transform the finite automaton from the Problem 1a into a context-free grammar (CFG). Show, step-by-step, how this CFG will generate the word 001.

2c. For the context-free grammar from the Problem 2b, show how the word 001 can be represented as uvxyz in accordance with the pumping lemma.

2d. Use a general algorithm to translate the CFG from 2b into Chomsky normal form.

2e. Use a general algorithm to translate the CFG from 2b into an appropriate push-down automaton. Explain, step-by-step, how this automaton will accept the word 001.

2f. Use the general stack-based algorithms to show:

   - how the compiler will transform a Java expression $6 - 1 - 2$ into inverse Polish (postfix) notation, and
   - how it will compute the value of this expression.

---

2.a -

$L = \{ Y^n N^n : n = 1, 2, \ldots \}$. Let's assume that $L$ is regular. Let's create a contradiction from this assumption. Since $L$ is a regular language by pumping lemma, there exists an integer $p$ such that every word $w^p$ from $L$ whose length $|w|$ is at least $p$ can be represented as $(xy^z)^p$, where $len(x) > 0$, $len(xy) \leq p$, $(N \ldots NY \ldots Y)$, $len(x) \leq p$. Let $w = xyz$ starts with $uxy^0$ so $uxy^p$ is in $L$. In $xxyyz$, we add $N$'s but not $Y$'s. We assumed with a word with a minimum passing of 2/3 in favor but when we added $N$'s we disrupted the 2/3 favor balance, then $xxyyz \notin L$, but by pumping lemma $xxyyz \in L$. We get a contradiction.

So our assumption that $L$ is regular language is wrong. $L$ is not regular.

---

File: //Q:/cs3350.19/final.html

Continue on next page.

4/15/2019
2.6

$\rightarrow S$

0 $\Rightarrow \emptyset$

1 $\Rightarrow 50.1$

$F \rightarrow \epsilon$

$S \rightarrow of$

$S \rightarrow 1e$

$f \rightarrow of$

$f \rightarrow 1f$

$e \rightarrow oe$

$e \rightarrow 1e$

101011
$s f f f 6$

2.6

*lowest pair

$V = 00$

$V = 1$

$x = \Lambda$

$y = \Lambda$

$z = \Lambda$
2. \[ 6 - 1 - 2 \]

\[
\begin{array}{c}
6 -1 - 2 \\
\hline
6 \quad 1 \quad 2 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
\hline
\ & \ & \ & \ & 1 & \ & 2 \\
\hline
\ & \ & \ & \ & \ & \ & \ \\
\ & \ & \ & \ & \ & \ & \ \\
\end{array}
\]

Answer is 3
2. Beyond finite automata: pushdown automata and context-free grammars:

2a. To overcome a presidential veto, the congress needs at least 2/3 of the votes. For example, if the voting record is YNY or YNYNY or YYNYY, where Y stands for Yes, and N for No, the veto will be overcome. Prove that the language consisting of all sequences of Y and N that lead to the overcome is not regular.

2b. Use a general algorithm to transform the finite automaton from the Problem 1a into a context-free grammar (CFG). Show, step-by-step, how this CFG will generate the word 001.

2c. For the context-free grammar from the Problem 2b, show how the word 001 can be represented as uvxyz in accordance with the pumping lemma.

2d. Use a general algorithm to translate the CFG from 2b into Chomsky normal form.

2e. Use a general algorithm to translate the CFG from 2b into an appropriate push-down automaton. Explain, step-by-step, how this automaton will accept the word 001.

2f. Use the general stack-based algorithms to show:

- how the compiler will transform a Java expression 6 − 1 − 2 into inverse Polish (postfix) notation, and
- how it will compute the value of this expression.

2a. theorem. The language described in 2a is not regular. Proof by contradiction. Let's assume that L is regular, then by pumping lemma there exists a number p such that for every word w from L whose length is at least p can be represented as xyz, where len(y) ≤ p, len(xy^z) = p, and for every i ≥ 0, w^i is in L.

Let's take the word w = N^p Y^p, its length is 2p + p = 3p. So, w can be represented as w^i = w in accordance with the pumping lemma, the word w^i = w as two words with xy and len(x) ≤ p, so x and y are among the first p symbols of the word. So, y consists of only N's or y when we go from x to y, we add N's, but we don't add any y's. So in bytes, the balance between N and p is disrupted. In fact, y may not be in L. But, by pumping lemma, y^2 is in L. But this contradicts, so our assumption is wrong and L is not regular.

file:///Q:/cs3350.19/final.html 4/15/2019
2d. \[ S \rightarrow SF \]
\[ S \rightarrow IE \]
\[ F \rightarrow OF \]
\[ F \rightarrow IF \]
\[ C \rightarrow CE \]
\[ C \rightarrow IE \]
\[ F \rightarrow E \]

Preliminary Step:
\[ S_0 \rightarrow S \]

Step 0:
\[ F \rightarrow O \]
\[ F \rightarrow I \]
\[ S \rightarrow O \]

Step 1:
\[ S_0 \rightarrow OF \quad S_0 \rightarrow IE \quad S_0 \rightarrow O \]

Step 2:
\[ V_0 \rightarrow G \quad S \rightarrow V_0 F \quad C \rightarrow V_0 E \]
\[ V_1 \rightarrow I \quad S \rightarrow V_1 F \quad E \rightarrow V_1 E \]
\[ F \rightarrow V_0 F \quad S_0 \rightarrow V_0 F \]
\[ F \rightarrow V_1 F \quad S_0 \rightarrow V_1 E \]

Step 3: No rules to eliminate
2f. \[
\begin{array}{c}
6 - 1 - 2 \\
\hline
6 - 1 - 2
\end{array}
\]

\[
\begin{array}{c}
6 - 1 - 2
\end{array}
\]

\[
\begin{array}{c}
\text{b b 5 2 3}
\end{array}
\]
3.a. In Houston, one third of the population is African-American (A), one third is Hispanic (H), and one third is White (W). To properly represent all the ethnic groups, the City Council must have an equal number of people from each group. So, e.g., if the council has WAWHHA, it is good. Prove that the set of all "good" ethnic combinations is not context-free and therefore, cannot be recognized by a pushdown automaton.

3b-e. Use a general algorithm to design a Turing machine that accepts exactly all sequences accepted by a finite automaton from Problem 1a. Show, step-by-step, how this Turing machine will accept the word 001. Describe, for each step, how the state of the tape can be represented in terms of states of two stacks.

3d-e. Design Turing machines for computing a + 1 in unary and in binary codes. Trace both for a = 3, i.e., for a = 111 in unary code and a = 11 in binary code.

3a. The language L of all the words of the form $w^k$ has the same number of W's, A's, and H's is not context-free.

Proof by contradiction.

Let's assume $L$ is a CFG. Then, by pumping lemma, there exists a $p$ such that for all words $w$, for all $a \leq p$, $w = uvxyz$ is a word in $L$, $|vxy| \leq p$, and $|vxy| > 0$, and for every $i \geq 0$, $uv^ixy^iz$ is a word in $L$.

The $|w| = 3p > p$, so $w = uvxyz$. Let's consider possible locations of $uvx$.

1. $uxy$ is in W's. Then, when we go from $w$ to $uv^ixy^iz$, we add W's but no A's or H's.
2. $uxy$ is in W's and A's. Then, we will add A's and no A's in $uv^ixy^iz$.

3. $uxy$ is in A's. Then, we will add A's and no A's or W's.
4. $uxy$ is in A's and H's. Then, we will add A's and H's in $uv^ixy^iz$, but no W's.
5. $uxy$ is in H's. Then, we will add H's in $uv^ixy^iz$ but no A's or W's.

In all cases, $u^i v^i w^i x^i y^i z^i$ is not in $L$, but by pumping lemma, our assumption was wrong, so $L$ is not CFG.

file://Q:/cs3350.19/final.html

4/13/2019
Automata:

\[ \text{start, } \_ \rightarrow s, R \]

\[ s, 0 \rightarrow f, R \]

\[ s, 1 \rightarrow e, R \]

\[ e, 0 \rightarrow e, R \]

\[ e, 1 \rightarrow e, R \]

\[ f, 0 \rightarrow f, R \]

\[ f, 1 \rightarrow f, R \]

\[ s, \_ \rightarrow \text{reject} \]

\[ e, \_ \rightarrow \text{reject} \]

\[ f, \_ \rightarrow \text{accept} \]

---

File: //Q:/cs3350.19/final.html

4/15/2019
3c. \( 11 \) in binary

```
1. □□□□
   ↑
   start
```

```
2. □□□□
   ↑
   add
```

```
3. □□□
   ↑
   add
```

```
4. □□□
   ↑
   add
```

```
5. □□□□
   ↑
   back
```

```
6. □□□□
   ↑
   back
```

```
7. □□□
   ↑
   back
```

```
start, _ → add, R
add, 0 → 1, back, L
add, 1 → C, add, R
back, C → L
back, _ → halt
add, _ → 1, back, L
```
3. Beyond pushdown automata: Turing machines

3a. In Houston, one third of the population is African-American (A), one third is Hispanic (H), and one third is White (W). To properly represent all the ethnic groups, the City Council must have an equal number of people from each group. So, e.g., if the council has WAWHHA, it is good. Prove that the set of all "good" ethnic combinations is not context-free and therefore, cannot be recognized by a pushdown automaton.

3b-c. Use a general algorithm to design a Turing machine that accepts exactly all sequences accepted by a finite automaton from Problem 1a. Show, step-by-step, how this Turing machine will accept the word 001. Describe, for each step, how the state of the tape can be represented in terms of states of two stacks.

3d-e. Design Turing machines for computing $a + 1$ in unary and in binary codes. Trace both for $a = 3$, i.e., for $a = 111$ in unary code and $a = 11$ in binary code.

3. A

- Theorem: $L = \{ \text{A}^n \text{H}^m \text{W}^n \mid n = 0, 1, \ldots \}$ is not CFG

Let's assume that $L$ is CFG, then by PL

- Let $w \in L$ and $\text{len}(w) \geq 2$ such that $w = uvxyz$ and $\text{len}(vxy) > 0$ and $\text{len}(uvx) \leq \ell$ for every $u, v, x, y, z \in L$.

Let's take word $w = \text{A} \ldots \text{A} \cdot \text{H} \ldots \text{H} \cdot \text{W} \ldots \text{W}$, where $\text{len}(w) = 3\ell$.

So, PL applies, the middle part cannot contain A's, H's, and W's because there would have been options:

1) $vxy$ is in A's, so we would only add A's, but not H's or W's.

2) $vxy$ is between A's and H's, so we would add A's and H's but not W's, disrupting the balance.

3) $vxy$ is in H's, we would add H's but not A's and W's, so when we go from original to $uvvxyz$ we add H's but not A's and W's.
4) \(Vxy\) is between \(H's\) and \(W's\), so when from original to \(Uvyxxyyz\) we add \(H's\) and \(W's\) but not \(A's\), \(Uvyxxyyz \& L\)

5) \(Vxy\) is in \(W's\), so when from original to \(Uvyxxyyz\) we add \(W's\) but not \(A's\) and \(H's\), so balance disrupted, \(Uvyxxyyz \& L\)

- In all cases \(Uvyxxyyz \& L\) but by PL \(Uvyxxyyz \& L\), so we get a contradiction, \(L\) is not CFG.

```
3. b-c
S   o > (1) 20.1

Start: U -> S/R
S, 0 -> f/R
S, 1 -> e/R
f, 0 or 1 -> f/R
s, 0 or 1 -> reject
f, U -> accept
e, U -> reject
```

```
trace 001

001

start

001

S

001

U

001

f

1

f

1

f

1

accept

1001

U

U

U

0
```
3. d-e

a+1 in unary and binary code

*Unary

start, U -> move, R
move, 1 -> move, R
move, U -> 1, L, back
back, 1 -> L
back, U -> halt

*Binary

start, U -> move, R
move, 1 -> 0, R
move, 0 or U -> 1, back, L
back, 0 -> back, L
back, U -> halt

*tracing a+1, a=3 in unary

*tracing a+1, a=3 in binary

file:///Q:/cs3350.19/final.html 4/15/2019
3a. In Houston, one third of the population is African-American (A), one third is Hispanic (H), and one third is White (W). To properly represent all the ethnic groups, the City Council must have an equal number of people from each group. So, e.g., if the council has WAWHHA, it is good. Prove that the set of all "good" ethnic combinations is not context-free and therefore, cannot be recognized by a pushdown automaton.

3b-e. Use a general algorithm to design a Turing machine that accepts exactly all sequences accepted by a finite automaton from Problem 1a. Show, step-by-step, how this Turing machine will accept the word 001. Describe, for each step, how the state of the tape can be represented in terms of states of two stacks.

3d-e. Design Turing machines for computing a + 1 in unary and in binary codes. Trace both for a = 3, i.e., for a = 111 in unary code and a = 11 in binary code.

3a. Theorem. \( L = \{ A^n H^n W^n : n = 0, 1, 2, \ldots \} \) is not context-free.

Proof (by contradiction): Assume \( L \) is CFG. Then by pumping lemma.

Let's consider possible locations of \( vxy \):

1) \( vxy \) is in \( A \). When we go from \( wxyz \) to \( uvwxyyz \), we add \( A \)’s but not \( H \)s or \( W \)s, so balance is disrupted and \( uv^2xyyz \notin L \).

2) \( vxy \) is between \( A \) & \( H \). When we go from \( wxyz \) to \( uvwxyyz \), we add \( A \) & \( H \) but not \( W \)s, so balance is disrupted and \( uv^2xyyz \notin L \).

3) \( vxy \) is in \( H \). Similarly, \( uv^2xyyz \notin L \).

4) \( vxy \) is between \( H \) & \( W \). Similarly, \( uv^2xyyz \notin L \).

5) \( vxy \) is in \( W \). Similarly, \( uv^2xyyz \notin L \).

In all cases, \( uv^2xyyz \notin L \), but by pumping lemma \( uv^2xyyz \in L \). This contradiction shows assumption is wrong, so \( L \) is not CFG.
\[\text{start}, - \rightarrow s, R \quad s, - \rightarrow \text{reject}\\s, 0 \rightarrow f, R \quad e, - \rightarrow \text{reject}\\s, 1 \rightarrow e, R \quad f, 0 \rightarrow R\\f, 1 \rightarrow R \quad e, 0 \rightarrow R\\e, 1 \rightarrow R\]

\[\text{3b-c}\]

\[\text{001}\]

\[\text{3d} + 1 \text{ in unary}\]

\[\text{start}, - \rightarrow \text{move, R} \quad s, - \rightarrow \text{move, R} \quad e, - \rightarrow \text{move, R} \quad f, - \rightarrow \text{move, R}\]

\[\text{move}, 1 \rightarrow R \quad \text{back}, 1 \rightarrow L \quad \text{halt}\]
4. Beyond Turing machines: computability

4a. Formulate Church-Turing thesis. Is it a mathematical theorem? Is it a statement about the physical world?

4b. Prove that the halting problem is not algorithmically solvable.

4c. Not all algorithms are feasible, but, unfortunately, we do not have a perfect definition is feasibility. Give a current formal definition of feasibility and give two examples:

- an example of an algorithm's running time which is feasible according to the current definition but not practically feasible, and
- an example of an algorithm's running time which is practically feasible but not feasible according to the current definition.

4d. Briefly describe what is P, what is NP, what is NP-hard, and what is NP-complete. Is P equal to NP?

4e. Give an example of an NP-complete problem: what is given, and what we want to find.

4f. Give definitions of a recursive (decidable) language and of a recursively enumerable (Turing-recognizable) language.

4a. Anything that can be computed on any physical device can also be computed by a Turing machine. It is a statement about the physical world.

4b. Halting problem

Proof by contradiction:

Let's assume there exists a halting checker and a program Po.

Does Po halt on Po? 

Case 1: Po halts on Po, then halting checker (Po, Po) is true, so Po won't halt on Po.

Case 2: Po doesn't halt on Po, then halting-checker (Po, Po) is false, so Po halts on Po.

So, we have a contradiction in both cases and our assumption is false.
4c. Formal definition! An algorithm $A$ is feasible if there exists a polynomial $p$, such that for every input $x$:

$$T_A(x) \leq p(\text{len}(x))$$

time it takes for $A$ to finish on input $x$

Feasible by definition. $\quad$ Not practically feasible: $\quad$ $10^{150}$

Practically feasible: $\quad$ $10^{-100}$

Not feasible by definition: $\quad$ $10^{-200}$

4d. $P$: class of all problems that can be solved in polynomial time.

$NP$: class of all problems for which once we have a candidate for a solution we can check in feasible time, whether it is indeed a solution.

$NP$-hard: A problem is $NP$-hard if every problem from the class $NP$ can be reduced to it.

$NP$-complete: A problem is $NP$-complete if it is both $NP$-hard and in the class $NP$.

No one knows if $P=NP$. It is an open problem in the CS community.

4e. The exact-change problem. In this problem, we are given a set of integers $E = e_1, e_2, e_3, \ldots, e_n$ and we have to find the values for a set of booleans $x_1, x_2, x_3, \ldots, x_n$ for which the following equation will be satisfied:

$$x_1 y_1 + x_2 y_2 + x_3 y_3 + \ldots + x_n y_n = S,$$

where $S$ is an arbitrary integer.
Recursively decidable: a language, \( A \), is recursively decidable if there is an algorithm that if given a word, decides whether the word belongs to the language.

Recursively enumerable: a language is recursively enumerable if there is an algorithm that eventually prints all elements in the language with the use of a Turing machine.