CS/ECE 374 Spring 2023   William Cheng (shihuac2@illinois.edu) **Homework 11 Problem 1**

Solution: (a) Suppose L_{regular} is decidable and there is an algorithm DecideLRegular that correctly decides the language L_{regular}. Then we can solve the halting problem as follows:

```
DecideHalt(⟨M, w⟩):
    Encode a Turing machine M′:
        M′(x) :
            run M on input w
            return True
    if DecideLRegular(⟨M′⟩)
        return True
    return False
```

- Suppose M halts on input w. Then M' accepts every input string x, including any regular string. Then DecideLRegular accepts the encoding $\langle M' \rangle$. So DecideHalt correctly accepts the encoding $\langle M, w \rangle$.
- Suppose M doesn't halt on input w. Then M' diverges on every input string x, including any regular string. Then DecideLRegular rejects the encoding $\langle M' \rangle$. So DecideHalt correctly rejects the encoding $\langle M, w \rangle$.

However, DecideHalt cannot be correct because Halt is undecidable. Therefore, the algorithm DecideLRegular does not exist, and L_{regular} is undecidable.

(b) Suppose we have an algorithm DecideHalt($\langle M \rangle$) that correctly decides if M halts on blank input. Then we can decide L_u as follows:

```
DecideLU(⟨M, w⟩):
    Encode a Turing machine M′:
        M′ writes w on the tape and simulates M
    if DecideHalt(⟨M′⟩)
        run w on M
        if M accepts w
            return True
    return False
```

- Suppose M accepts w. Then M' will halt and DecideHalt accepts $\langle M' \rangle$. Then we run w on M and since M accepts w, DecideLU correctly accepts $\langle M, w \rangle$.
- Suppose M does not accept w. Then there are two cases: 1) M rejects w. Then M' halts and DecideHalt accepts $\langle M' \rangle$. Then we run w on M and since M rejects w, DecideLU correctly rejects $\langle M, w \rangle$. 2) M diverges on w. Then DecideHalt rejects $\langle M' \rangle$, and DecideLU correctly rejects $\langle M, w \rangle$.

Therefore, the reduction from L_u to L_{HALT} is correct.

(c) We begin by showing that $L_{\text{nonemptylang}} = \Sigma^* \setminus L_{\text{emptylang}}$ is undecidable. Then, we prove that $L_{\text{nonemptylang}}$ is recursively enumerable. Finally, we use Lemma 5 in Jeff's notes 7.3 to prove that $L_{\text{emptylang}}$ is not recursively enumerable by contradiction.

- Let $L_{\text{nonemptylang}} = \Sigma^* \backslash L_{\text{emptylang}}$. Suppose there is an algorithm DecideNonEmptyLang($\langle M \rangle$) that decides if M accepts at least one string. We can reduce the Halting problem to DecideNonEmptyLang as follows:

```
DecideHalt(⟨M, w⟩):
  Encode a Turing machine M′:
      M′(x) :
          run M on input w
          return True
  if DecideNonEmptyLang(⟨M′⟩)
      return True
  return False
```

  If M halts on w, then M' accepts every input and clearly accepts at least one string, so DecideNonEmptyLang accepts the encoding $\langle M' \rangle$ and DecideHalt correctly accepts the encoding $\langle M, w \rangle$. If M does not halt on w, then M' diverges on every input and $L(M') = \emptyset$, so DecideNonEmptyLang rejects the encoding $\langle M' \rangle$ and DecideHalt correctly rejects the encoding $\langle M, w \rangle$. We know that the Halting problem is undecidable, therefore $L_{\text{nonemptylang}}$ is undecidable.

- Let $M'(\langle M \rangle)$ be a Turing machine that operates as follows: M' enumerates over Σ^*, and for each string $w \in \Sigma^*$ checks if M accepts w. M' accepts $\langle M \rangle$ if M accepts w. It is obvious that M' will accept $\langle M \rangle$ if M accepts at least one string, and will diverge on $\langle M \rangle$ if M does not accept any string, and therefore M' accepts $L_{\text{nonemptylang}}$. So $L_{\text{nonemptylang}}$ is recursively enumerable.

- Lemma 5 in Jeff's notes 7.3: an acceptable (recursively enumerable) language is decidable if and only if its complement is acceptable. Assume $L_{\text{emptylang}}$ is recursively enumerable. Then $L_{\text{nonemptylang}}$ is decidable by Lemma. However, we have proved that $L_{\text{nonemptylang}}$ is undecidable, which is a contradiction.

Therefore, $L_{\text{emptylang}}$ is not recursively enumerable.

■

CS/ECE 374 Spring 2023 William Cheng (shihuac2@illinois.edu)

Homework 11 Problem 2

Solution: (a) Let x_e be a Boolean variable for $e \in E$. If we add e to the answer set M then $x_e = 1$. In a perfect matching, each vertex is incident to exactly one edge. We use the following functions to generate Boolean expressions that checks if a vertex v has at least one incident edge/at most one incident edge.

```
AtLeastOne(v):
  find incident edges e_1, ..., e_k of v
  if there is no incident edge return False
  return x_{e_1} ∨ ... ∨ x_{e_k}

AtMostOne(v):
  find incident edges e_1, ..., e_k of v
  if there is 0 or 1 incident edge return True
  return (¬x_{e_1} ∨ ¬x_{e_2}) ∧ ... ∧ (¬x_{e_1} ∨ ¬x_{e_k}) ∧ (¬x_{e_2} ∨ ¬x_{e_3}) ∧ ... ∧ (¬x_{e_{k-1}} ∨ ¬x_{e_k})
```

The Boolean expression in AtLeastOne evaluates to 0 if no incident edge of v is added to M, otherwise 1. The Boolean expression in AtMostOne enumerates all pairs of incident edges and checks if at least one pair of edges is in M. If there is at least a pair, it evaluates to 0. Otherwise, it evaluates to 1. Following is the algorithm to check if a perfect matching exists using an algorithm for SAT.

```
ExactlyOne(v):
  return AtLeastOne(v) ∧ AtMostOne(v)

DecidePerfectMatching(G):
  let v_1, v_2, ..., v_n be the vertices of G
  φ ← ExactlyOne(v_1) ∧ ... ∧ ExactlyOne(v_n)
  return DecideSAT(φ)
```

Correctness: If there exists a set of variables $x_{e_1}, ..., x_{e_m}$ that satisfies the Boolean expression ϕ, then every vertex in G must have exactly one incident edge: *deg*$(v) = 1$ for all $v \in V$ since it's the definition of ϕ. Let $M = \{e_i \mid e_i \in E, x_{e_i} = 1\}$. Then M is a matching of G because no two edges share a vertex (otherwise there exists v_i where *deg*$(v_i) > 1$). $|M| = \frac{\sum_{v \in V} deg(V)}{2} = \frac{|V|}{2}$, therefore M is a perfect matching.

If there exists a perfect matching M of G, then every vertex in G must have exactly one incident edge. We make $x_{e_i} = 1$ for all $e_i \in M$ and $x_{e_i} = 0$ otherwise. Then the set of variables $x_{e_1}, ..., x_{e_m}$ satisfies ϕ since ExactlyOne$(v) = 1$ for all $v \in V$ by definition.

Since an instance of PerfectMatching is a "yes" instance if and only if the reduction ϕ is a "yes" instance of SAT, the reduction is correct.

Time complexity: AtLeastOne takes $O(|E|)$ time and AtMostOne takes $O(|E|^2)$ time. We compute ExactlyOne for $|V|$ times. Therefore, computing a formula for ϕ takes $O(|V||E|^2)$ time, which is polynomial.

This does not prove that PerfectMatching is NP-complete because this shows PerfectMatching $\leq_P$ SAT. If we want to prove PerfectMatching is NP-complete we need to reduce SAT to PerfectMatching, which is SAT $\leq_P$ PerfectMatching.

(b) We begin by proving that EIGHT is in NP, and then reduce the Hamiltonian cycle problem to EIGHT to prove EIGHT is NP-hard:

- We use a subgraph as a certificate and check if the subgraph is an eight-graph. We can use the following certifier: check if there are $2k-1$ nodes in the subgraph, run DFS on an arbitrary node and get a DFS tree of the subgraph, and look for back-edges in the DFS tree to identify cycles. If there are $2k-1$ nodes in the subgraph and exactly two cycles of size k and exactly one node that is in both cycles, then return true. The certifier runs in polynomial time, therefore EIGHT is in NP.
- We can reduce Hamiltonian cycle to EIGHT by constructing a graph G': first, we add two disjoint copies of G to G'; second, we add a new vertex a to G' and connect a to all the other nodes in G'. Let $k = |V| + 1$.
  - If G has a Hamiltonian cycle, then G' has an eight-graph on $2k-1$ nodes. Let the Hamiltonian cycles of the two copies of G be C_1 and C_2. Let u_1, v_1 and u_2, v_2 be two connected nodes in C_1 and C_2. Then the graph $C_1 + C_2 - u_1v_1 - u_2v_2 + u_1a + av_1 + u_2a + av_2$ is an eight-graph on $2k-1$ nodes: it contains nodes from C_1, C_2 and a, so its size is $2|V| + 1 = 2k - 1$. It contains two cycles of size $|V| + 1 = k$: each cycle is obtained by removing an edge in the Hamiltonian cycle and adding a to the cycle. The vertex a is the only shared vertex of the two cycles since the two Hamiltonian cycles are originally disjoint.
  - If G' has an eight-graph on $2k-1$ nodes, then G has a Hamiltonian cycle. a must be the shared vertex of the two cycles because the two cycles were originally disjoint. Denote the two copies of G as G_1 and G_2. Let u_1, v_1 be the neighbors of a in G_1. Let the cycle containing vertices in G_1 and a be C_1. By definition, $|C_1| = k$. Now we remove a and its incident edges in C_1 and add the edge u_1v_1 to it to form a new cycle C_1'. Then the size of C_1' is $k - 1 = |V|$. Since C_1' is a cycle whose vertices all belong to G_1 and $|C_1'| = |V|$, we conclude that C_1' is a Hamiltonian cycle in G_1. Since G_1 is a copy of G, G must have a Hamiltonian cycle.

  Therefore, the reduction from Hamiltonian cycle to EIGHT is correct. Since Hamiltonian cycle is NP-hard, EIGHT must also be NP-hard.

Since EIGHT is in NP and EIGHT is NP-hard, it can be concluded that EIGHT is NP-complete.

■