黄羽翔 2024302061069
计算理论导引
2025 年 12 月 2 日

计算理论导引第四次作业

Question 1. 证明引理 1.8。

1. 对数空间可计算 $\rightarrow$ 隐式对数空间可计算：

如果函数 f 可以被一个标准的 **L**-变换器 $\mathbb{M}$ 计算出来，那么它自动满足隐式定义的所有条件：

多项式长度：**L**-变换器运行时间为多项式 $T(n)$，所以输出长度 $|f(x)| \leq T(n)$，满足多项式有界。

按位可计算性：要计算 $f(x)$ 的第 i 位，我们只需让一个 **L** 机器 $\mathbb{M}'$ 模拟 $\mathbb{M}$ 的完整计算过程。$\mathbb{M}'$ 在模拟过程中，只将前 $i-1$ 位写入一个虚拟输出带，并在计算到第 i 位时，将该位的值作为自己的接受/拒绝结果。

整个模拟过程的额外工作空间仍然是 $O(\log n)$，因此满足 **L** 可计算性。

2. 隐式对数空间可计算 $\rightarrow$ 对数空间可计算：

如果函数 f 是隐式可计算的，我们可以构造一个标准的 **L**-变换器 $\mathbb{M}$ 来输出整个 $f(x)$：

构造 $\mathbb{M}$：$\mathbb{M}$ 使用其 $O(\log n)$ 工作空间来存储一个计数器 i，从 $i = 1$ 迭代到 $|f(x)|$ 的最大长度。

生成输出：在每次迭代中，$\mathbb{M}$ 调用隐式定义中的 **L** 判定机，以确定 $f(x)_i$ 的值（0 或 1）。

写入：$\mathbb{M}$ 将得到的值顺序写入只写输出带。

空间分析：计数器 i 只需要 $O(\log n)$ 空间。每次调用按位判定机也只用 $O(\log n)$ 空间。因此，总工作空间始终保持在 $O(\log n)$。

Question 2. 将引理 1.9 的证明中描述的线性空间算法用程序实现。

```
vector<vector<bool>> vars;
vector<int> exist;
vector<int> forall;
/* 构造 n 个布尔变量的真值表，代表 varphi */
void construct_table(int n){
    for(int i = 0; i < (1 << n); i++){
        bool truth = rand() % 2; // 随机生成真值
        if (!truth) continue;
        vector<bool> row;
        for(int j = 0; j < n; j++){
            row.push_back((i & (1 << j)) != 0);
        }
        vars.push_back(row);
    }
}
/* 构造量词 */
void construct_quantifier(int n){
    for(int i = 0; i < n; i++){
        int quantifier = rand() % 2; // 0 代表存在量词，1 代表全称量词
        if(!quantifier){
            exist.push_back(i);
        }
        else{
            forall.push_back(i);
        }
    }
}
void judge(vector<bool>& assignment, int cnt){
    if(cnt < exist.size()){
        assignment[exist[cnt]] = true;
        judge(assignment, cnt + 1);
        assignment[exist[cnt]] = false;
        judge(assignment, cnt + 1);
```

```
    }
    else if (cnt < exist.size() + forall.size()){
        assignment[forall[cnt - exist.size()]] = true;
        judge(assignment, cnt + 1);
        assignment[forall[cnt - exist.size()]] = false;
        judge(assignment, cnt + 1);
    }
    else {
        bool satisfied = false;
        for(const auto& row : vars){
            bool match = true;
            for(int i = 0; i < assignment.size(); i++){
                if(row[i] != assignment[i]){
                    match = false;
                    break;
                }
            }
            if(match){
                satisfied = true;
                break;
            }
        }
        if(satisfied){
            cout << "Satisfiable assignment found: ";
            for(bool val : assignment){
                cout << val << " ";
            }
            puts("");
        }
    }
}
int main(){
    int n=4;
    construct_table(n);
    construct_quantifier(n);
    vector<bool> result;
    result.resize(n);
    judge(result, 0);
    return 0;
}
```

.

Question 3. 说明定理 1.15 的证明中构造的 φ_x 是对数空间可计算的。

考虑判定隐式对数空间可计算的三个条件：

- φ_x 即 $\psi_{S(|x|)}$，则 φ_x 中含有 $O(S(n))^2$ 个变量，$|\varphi_x| = O(S(n)^2)$，这符合第一个条件：$\exists c.\forall x.|f(x)| \le c|x|^c$。

- 枚举所有两种格局的组合，丢入 φ_x 判定，用一个计数器记录组合数量，就可以确定 $|\varphi_x|$，即满足 $\{\langle x, i\rangle | i \le |f(x)|\} \in \mathbf{L}$。

- φ_x 的输出只有一位，我们只要能在对数空间内计算出 $\varphi_x(x_1, x_2)$。依赖定义：$\psi_{i+1}(C', C'') = \exists C \forall X \forall Y \Big[\big((X = C' \wedge Y = C) \vee (X = C \wedge Y = C'')\big) \rightarrow \psi_i(X, Y)\Big]$，每一次递推都可以用对数空间，我们只需要 $O(\log|x|)$ 的工作空间来存储索引和层级信息，就能精确地定位和输出 φ_x 的任意一位，即满足 $\{\langle x, i\rangle | f(x)_i = 1\} \in \mathbf{L}$。

Question 4. 用程序实现萨维奇定理证明中的算法。

```
pair<int, int> edge[N];
bool reach(int u, int v, int k){
    if(!k){
        for(auto e:edge){
            if((e.first == u && e.second == v)||(e.first == v && e.second == u))return true;
        }
        return u == v;
    }
    else{
        for(int w = 0; w < n;w++){
            return reach(u, w, k - 1) && reach(w, v, k - 1);
        }
    }
    return false;
}
```

Question 5. 证明：若 $A, B \in \mathbf{NP}$，则 $A \cup B, A \cap B, AB \in \mathbf{NP}$。

设 A 和 B 分别由多项式时间的验证器 $\mathbb{M}_A, \mathbb{M}_B$ 和多项式长度的证书 u_A, u_B 判定：

- $A \cup B$：一个输入 x 满足 A 或 B 中的任意一个，构造证书 u'：$u' = \langle x, u_x \rangle$，其中 x 用于指示 $x \in A$ 还是 $x \in B$，u_x 是对应的证书 u_A 或 u_B，由于 $A, B \in \mathbf{NP}$ 所以 $A \cup B \in \mathbf{NP}$。

- $A \cap B$：一个输入 x 同时满足 A 和 B，构造证书 u'：$u' = \langle u_A, u_B \rangle$，验证器 $\mathbb{M}'(x, u')$ 为先运行 $\mathbb{M}_A(x, u_A)$，再运行 $\mathbb{M}_B(x, u_B)$，由于 $u_A + u_B$ 还是多项式长度，所以 $A \cap B \in \mathbf{NP}$。

- AB：一个输入 x 被分割为分别满足 A 和 B 的两部分 a 和 b，构造证书 u'：$u' = \langle \langle u_A, u_B \rangle, k \rangle$，设 u'_A 为 $\langle u_A, u_B \rangle$ 的前 k 部分，u'_B 为 $\langle u_A, u_B \rangle$ 的后 k 部分，验证器 $\mathbb{M}'(x, u')$ 为先运行 $\mathbb{M}_A(a, u'_A)$，再运行 $\mathbb{M}_B(b, u'_B)$，由于 $u'_A + u'_B$ 还是多项式长度，且一台猜测 u' 的非确定图灵机能在多项式时间猜到 u'，所以 $AB \in \mathbf{NP}$。