上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

THESIS OF BACHELOR

论文题目：Ladner Theorem in Parameterized Complexity Theory

学生姓名：黄炫圭

学生学号：5100309715

专　　业：计算机科学与技术

指导教师：陈翌佳

学院(系)：电子信息与电气工程学院

参数复杂性中的 Ladner 定理

摘要

在经典复杂性中，Ladner定理证明了如果 **NP** ≠ **P**，那么存在既不是 **P** 也不是 **NP** 完备的 **NP** 问题。更进一步地，递归集的多项式度是稠密的，也就是说，对于任何的非平凡递归集 $A <^P B$，存在集合 C 使得 $A <^P C <^P B$，其中‘$\leq^P$’可以是多项式时间m-规约， 也可以是多项式时间图灵规约。

在参数复杂性中，上述定理对应的版本是这样的：如果 **FPT** ≠ **W[1]**，那么存在即不是 **FPT** 也不是 **W[1]** 完备的 **W[1]** 问题。更进一步地，在这两个类之间有无穷多的中间难度问题。在这篇学士学位论文中，我形式化地严格地证明了这个参数复杂性版本的 Ladner 定理。

在经典复杂性中，多项式时间图灵机和多项式时间规约是可以有效枚举的，所以经典的对角线方法就足以证明Ladner定理。然而，因为 **FPT** 问题和 **FPT** 规约并不可以有效枚举，在证明参数复杂性中的Ladner定理时，我们需要使用有限损害法。

关键词： Ladner 定理， 参数复杂性， 有限损害法

LADNER THEOREM

IN PARAMETERIZED COMPLEXITY THEORY

ABSTRACT

In classical complexity theory, Ladner Theorem states that if $\mathbf{NP} \neq \mathbf{P}$, then there are $\mathbf{NP}$ problems that are neither $\mathbf{NP}$-Complete nor $\mathbf{P}$. Moreover, the polynomial degrees of recursive set are dense, i.e. for any non-trivial recursive set $\mathcal{A} <^P \mathcal{B}$, there is a set C such that $\mathcal{A} <^P \mathcal{C} <^P \mathcal{B}$, where '$\leq^P$' can be either polynomial-time m-reduction or polynomial-time Turing-reduction.

The analogous fact for parameterized complexity theory would be the following: if $\mathbf{FPT} \neq \mathbf{W}[\mathbf{1}]$, then there are $\mathbf{W}[\mathbf{1}]$ problems that are neither $\mathbf{W}[\mathbf{1}]$-Complete nor $\mathbf{FPT}$, moreover, there are infinitely many intermediate problems between them. In this undergraduate thesis, I prove this parameterized version of Ladner Theorem formally and rigorously.

In classical complexity theory, polynomial time-bounded Turing Machines and polynomial time Turing reductions can be effectively enumerated, hence only classical diagonalization method is adequate. However, since $\mathbf{FPT}$ algorithms and $\mathbf{FPT}$ reductions are not effectively enumerated, we need finite injury priority method to prove parameterized version of Ladner Theorem.

Keywords: Ladner Thorem, Parameterized Complexity Theory, Finite Injury Priority Method

Contents

Chapter 1 Introduction

Theoretical computer scientists devote themselves into the cause of exploring the true meaning of computation and developing concise theory to demonstrate the true beauty of computation. In this great human process of intellectual adventures, from time to time, a variety of theories are introduced by some ambitious and wise pioneers, who may be so far beyond his/her time that few people can understand his/he genius at that time, put in front of all the scientists to receive their examination, criticism or appreciation, finally accepted by his/her competitors and cooperators, with further refinements or developments coming up, or rejected as a waste of paper, thrown into the dust of history. This is how theoretical computer science make its progress, by contributions of millions of scientists, although they may have different purposes. It is a great event in our time, across the countries, races and ideologies.

Following is a brief introduction of this great event in a perspective of parameterized computational complexity theorists, one of a small interest group of theoretical computer scientists, in order to introduce the central goal of this undergraduate thesis.

1.1 A Brief Introduction

Starting from Alan Turing's seminal paper, computability theory, or recursion theory in mathematical logic, attempt to discover the power, limitation and ultimate meaning of computation. Rigorous definition and properties of problems and programs are deduced under different computation models, which were shown to have the same capability of computation in a sense that all functions computable in one model can always be computed by another model. Therefore the famous "Church-Turing Thesis" was proposed, advocating that Turing Machine, one of the famous and useful computation models, has the power to compute all the informal and intuitive computable functions, even all the functions a physical device can compute([1], Cutland, 1980: 1.)([2], Homer and Selman, 2011: 1.). Besides, problems that are not computable in Turing Machine

are further divided into different levels of different hierachies according to their hardness with respect to different reductions or some other characterizations([3], Soare, 1987: 69.).

Besides computability, computational complexity concerns about tractability with respect to bounded resources, such as space and time([4], Papadimitriou, 2003: 1.)([5], Arora and Barak, 2009: 1.). Problems needed much space or time are considered intractable in this case. The criteria that the running time should be polynomial in the size of the input of the problem was gradually accepted as the condition for tractability. Similarly, problems are also divided into classes and levels of hierarchies according to reductions or characterizations concerning resources. Then the famous question like '$\mathbf{P} =?\mathbf{NP}$' were proposed, where $\mathbf{P}$ is the tractable class mentioned above and $\mathbf{NP}$ is also a well-known complexity class, and remain open today.

As a fairly new branch of computational complexity theory, parameterized complexity theory aims to provide a framework for a refined analysis for hard problems([6], Flum and Grohe, 2006: 1.)([7], Downey and Fellows, 1999: 1.). Numerous computational problems classified to be the same intractable class by classical complexity theory can be further divided into several hierarchies by parameterized complexity theory, exposing a more delicate structure of computational problems in the aspect of complexity. Numerous computational problems that cannot be studied comprehensively with respect to their time and space consumption by classical complexity theory can be better analysed in terms of several different input parameters of them by parameterized complexity theory.

1.2 Ladner Theorem

In classical complexity theory, Ladner Theorem([8], Ladner, 1975: 158.) reveals the density property of structures of classical complexity classes under certain plausible assumption, similar to the density property of real numbers. Between any two real numbers, there are infinitely many intermediate real numbers. Similarly, Ladner Theorem demonstrates that between the 'easiest' problems and the 'hardest' problems in the hierarchy of classical complexity classes, or even between any two problems that are

not 'equally hard', there are infinitely many intermediate problems that are not 'equally hard' with them. In detail, it says if $\mathbf{P} \neq \mathbf{NP}$, then there is an $\mathbf{NP}$ problem that is not $\mathbf{P}$ and not $\mathbf{NP}$-Complete. Moreover, the polynomial degrees of recursive set are dense, i.e. for any non-trivial recursive set $\mathcal{A} <^P \mathcal{B}$, there is a set $\mathcal{C}$ such that $\mathcal{A} <^P \mathcal{C} <^P \mathcal{B}$, where '$\leq^P$' can be either polynomial-time m-reduction or polynomial-time Turing-reduction.

Connections between classical complexity theory and parameterized complexity theory are, nevertheless, interesting and seemingly important to explore. It would be a great intellectual satisfaction if we can prove similar dense result just like Ladner Theorem, in parameterized complexity theory. An well-known analogue of the hierarchy of classical complexity classes is the W-hierarchy in parameterized complexity theory. Therefore Ladner Theorem in parameterized complexity theory states, if $\mathbf{FPT} \neq \mathbf{W}[\mathbf{1}]$, there are $\mathbf{W}[\mathbf{1}]$ problems that are neither $\mathbf{FPT}$ nor $\mathbf{W}[\mathbf{1}]$-Complete. Moreover, there are infinite intermediate problems between them.

Considerable efforts have been made to find parameterized analogues of classical complexity results to somehow find such connections([6], Flum and Grohe, 2006: 1.)([7], Downey and Fellows, 1999: 1.). Analogue of Ladner Theorem in parameterized complexity theory is claimed to be proved by Downey and Fellows as a tiny part of their ground breaking series of articles([9], Downey and Fellows, 1993: 205.). In spite of their splendid insight to develop so many significant results in this series, their proof for Ladner Theorem in parameterized complexity theory is rather brief with a few details missing, which makes it less accessible to people not well-versed in recursion theory. Consequently the central goal in this undergraduate thesis is to present a complete proof, in such a way that a reader with elementary knowledge in complexity theory would be able to understand all the details.

The parameterized version of Ladner Theorem is far more difficult to prove, compared to its original version in classical complexity theory. The latter one can be solved by classical diagonalization method, hence is not difficult due to the effective enumerability of polynomial time-bounded Turing Machines and polynomial time-bounded Turing reductions([10], Schöning, 1982: 96.). However, the former one needs a more complicated method called "Finite Injury Priority Method"([3], Soare, 1987: 110.),

which is indeed a sophisticated refinement of classical diagonalzation method, making this proof difficult and inevitably long.

In their recent book, Downey and Fellows surprisingly advocate that the above connections are not important comparing to their so-called "forward opportunities"([11], Downey and Fellows, 2013: xviii.). Nonetheless, besides suitable for a undergraduate thesis, parameterized version of Ladner Theorem is interesting and worth proving on its own, since it can deeper my understanding of not only classical and parameterized complexity theory but also finite injury priority method which is generally useful.

1.3 Structures of this thesis

In this undergraduate thesis, I give a precise proof of Ladner Theorem in parameterized complexity theory. Before doing that, I first give a short introduction to computability theory, classical and parameterized complexity theory in Chapter 2, to make this undergraduate thesis as self-contained as possible. Second, in Chapter 3, I introduce the original Ladner Theorem in classical complexity theory, and give a proof of it using the so-called "blowing holes" technique derived from R. Ladner's original paper. Then in Chapter 4, I introduce the finite injury priority method, which is first introduced in the context of recursion theory, using proofs of two interesting and inspiring theorems as examples. Finally in Chapter 5, the construction and precise proof of Ladner Theorem in paramterized complexity theory is presented.

Chapter 2 Preliminaries

In this chapter, I will introduce some basic definitions and theorems needed for the following chapter from computability theory, computational complexity and parameterized complexity.

2.1 Computability Theory

For more details of definitions and theorems in computability theory, especially proof of theorems, please refer to ([1], Cutland, 1980: 1.) and ([2], Homer and Selman, 2011: 1.).

Definition 2.1 (Problem)**.** *A* problem *is a subset of* $\{0,1\}^*$*, or equivalently* $\mathbb{N}$.

In the following part, we will use 'problem', 'language' and 'set' interchangeably.

2.1.1 Turing Machines

Definition 2.2 (Turing Machine)**.** *A* k*-tape* Turing Machine*, shorted as TM, is a system*

$$M = \langle Q, \Gamma, \Sigma, \sigma, q_0, N, q_{final} \rangle, \tag{2-1}$$

where

Q *is the finite set of states,*

Γ *is the finite tape alphabet,*

Σ *is the input alphabet,* $\Sigma \subseteq \Gamma \setminus \{B\}$,

$B \in \Gamma$ *is the blank,*

σ *is the transition function of* $Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, R, S\}^k$,

$q_o \in Q$ *is the initial state and*

q_{final} *is the final state.*

For details of how a Turing Machine works and *accepts* an input string, please refer

to. Intuitively speaking, we will say a Turing Machine *accept* an input if in the final state, the content of its output tape is 1.

Definition 2.3 (Language of Turing Machine)**.** *Let M be a Turing Machine. The language* accepted *by M is*

$$L(M) = \{w \in \Sigma^* | M \text{ accepts } w\}. \tag{2-2}$$

For a TM M, we denote $M(x) \downarrow$ if the computation of M halts on input x, and $M(x) \uparrow$ if it never stops on input x. For a function f, we also denote $f(x) \downarrow$ if $x \in Domain(f)$ and $f(x) \uparrow$ if not.

Besides, the *function computed by a Turing Machine M* is defined as the content of its output tape whenever it halts for the input, and undefined if it never stops. A function is *computable* iff it is the function computed by some Turing Machine.

Definition 2.4 (Recursive sets)**.** *A set S is* decidable*,* computable*, or* recursive*, iff its* characteristic function

$$c(x) = \begin{cases} 1, & \text{if } x \in S, \\ 0, & \text{otherwise} \end{cases} \tag{2-3}$$

is computable by a Turing Machine, denoted as the program *of set S.*

An *relativised TM M* is a TM with an oracle L, which has a magical extra tape for M, such that whenever M writes a string on this tape and goes into a special "invocation" state, then the string gets overwritten in one step by 0 or 1 depending on whether the string is in L or not.

Definition 2.5 ('$\simeq$')**.** *Let f and g be two functions. Then $f(x) \simeq g(x)$ iff $f(x) = g(x)$ whenever $f(x) \downarrow$, and $f(x) \downarrow$ iff $g(x) \downarrow$. Similarly we define $M(x) \simeq M'(x)$ for two TMs M and M'.*

Theorem 2.6 (Universal TM)**.** *There is a TM U such that $U(\alpha, x) \simeq M_\alpha(x)$, where $M_\alpha(x)$ is an TM M with α as its coding.*

Therefore we can have an effective enumeration of Turing Machines $\{M_k\}_{k\in\mathbb{N}}$ and effective enumeration of computable functions $\{\phi_k\}_{k\in\mathbb{N}}$, where ϕ_k is the function computed by M_k.

2.1.2 Undecidability and Recursively Enumerable Sets

Definition 2.7 (Halting problem)**.** *Halting problem is the problem to decide whether* $M_x(y)\downarrow$ *or not, with* x *and* y *as input.*

Definition 2.8 (Undecidability)**.** *A problem* P *is* undecidable *is it is not decidable.*

Theorem 2.9. *Halting problem is undecidable.*

Definition 2.10 (Recursively enumerable sets)**.** *A set* S *is* partial-decidable*,* partial-computable*, or* recursively enumerable *shorted as* r.e.*, iff its* partial-characteristic function

$$\chi(x) = \begin{cases} 1, & \text{if } x \in S, \\ \text{undefined}, & \text{otherwise} \end{cases} \tag{2-4}$$

is computable by a Turing Machine.

Theorem 2.11. *A set* S *is r.e. iff there is a* recursive enumeration g *of* S *such that* $S = \{g(k)\}_{k\in\mathbb{N}}$*. Moreover, if* S *is not recursive,* g *can be injective.*

2.1.3 Reductions

Definition 2.12 (m-reduction)**.** *A set* L *is* m-reducible *to a set* L'*, denoted by* $L \leq_m L'$*, if there is a computable function* $f : \mathbb{N} \to \mathbb{N}$ *such that for every* $x \in \mathbb{N}$*,* $x \in L$ *iff* $f(x) \in L'$*. Then* f *is called an* m-reduction from L to L'*, and we also denote* $L \leq_f L'$*.*

Definition 2.13 (Turing-reduction)**.** *A set* L *is* Turing-reducible *to a set* L'*, denoted by* $L \leq_T L'$*, if there is a relativised TM* M *that, given an oracle for deciding* L'*, can decide* L*. Then this relativised TM is called a* Turing-reduction from L to L'*.*

Theorem 2.14. *Let* L *and* L' *be two sets, then* $L \leq_T L'$ *iff there exists* e *such that* $L(x) \simeq \phi_e^{L'}(x)$*, where* $\phi_e^{L'}$ *is the function computed by the* e*-th relativised TM with oracle* L' *and* $L(x)$ *is the characteristic function of* L*.*

2.2 Computational Complexity Theory

2.2.1 Efficiency and Running Time

Following definitions and theorems are from Arora's textbook([5], Arora and Barak, 2009: 1.).

Definition 2.15. *We say a Turing Machine* M computes f in $T(n)$-time *if its computation on every input* x *requires at most* $T(|x|)$ *steps and output* $f(x)$.

If a Turing Machine can compute some function in $T(n)$-time where $T(n)$ is a polynomial of n, then it is a *polynomial-time TM*.

Definition 2.16 (Time-constructible functions)**.** *A function* $T : \mathbb{N} \to \mathbb{N}$ *is* time constructible *if* $T(n) \geq n$ *and there is a TM* M *that computes the function* $x \mapsto \lfloor T(|x|) \rfloor$ *in time* $T(n)$.

Examples for time-constructible functions are n, $n \log n$, n^2 and 2^n. The TM M in the above definition for time-constructible function $T(n)$ is called $T(n)$-*clocked*.

Theorem 2.17 (Efficient universal Turing Machine)**.** *There is a universal TM* U *so that if the machine it simulates halts on input* x *within* T *steps then* U *halts within* $CT \log T$ *steps, where* C *is a constant independent of* x *and depending only on the TM it simulates.*

Therefore we can have an effective enumeration of TM in the context of computational complexity.

Definition 2.18 (The class $\mathbf{DTIME}$)**.** *Let* $T : \mathbb{N} \to \mathbb{N}$ *be some function. A language* L *is in* $\mathbf{DTIME}(T(n))$ *iff there is a Turing Machine that runs in time* $c \cdot T(n)$ *for some constant* $c > 0$ *and decides* L.

Definition 2.19 (The class $\mathbf{P}$)**.** $\mathbf{P} = \cup_{c \geq 1} \mathbf{DTIME}(n^c)$.

Following definitions and theorems are from Papadimitriou's textbook([4], Papadimitriou, 2003: 1.).

Definition 2.20 (Big O notation)**.** *Let f and g be functions from $\mathbb{N}$ to $\mathbb{N}$. $f(n) = O(g(n))$ iff there are positive integers c and n_0 such that, for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$.*

Theorem 2.21 (Linear Speedup Theorem)**.** *Let $L \in \mathbf{DTIME}(f(n))$. Then for any $\epsilon > 0$, $L \in \mathbf{DTIME}(f'(n))$ where $f'(n) = \epsilon \cdot f(n) + n + 2$.*

2.2.2 Reducibility and NP-Completeness

Following definitions and theorems are from Arora's textbook([5], Arora and Barak, 2009: 1.).

Definition 2.22 (Polynomial-time m-reduction)**.** *A language L is* polynomial-time m-reducible *to a language L', denoted by $L \leq_m^{\mathbf{P}} L'$, if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$, $x \in L$ iff $f(x) \in L'$. Then f is called an* polynomial-time m-reduction from L to L'.

Definition 2.23 (Polynomial-time Turing-reduction)**.** *A language L is* polynomial-time Turing-reducible *to a language L', denoted by $L \leq_T^{\mathbf{P}} L'$, if there is a polynomial-time relativised TM M that, given an oracle for deciding L', can decide L. Then this relativised TM is called a* polynomial-time Turing-reduction from L to L'.

Theorem 2.24. $\mathbf{P}$ *is closed under polynomial-time m-reduction and polynomial-time Turing-reduction, i.e. if $L \leq^{\mathbf{P}} L'$ and $L' \in \mathbf{P}$ then $L \in \mathbf{P}$, where '$\leq^{\mathbf{P}}$' can be '$\leq_m^{\mathbf{P}}$' or '$\leq_T^{\mathbf{P}}$'.*

Definition 2.25 (The class $\mathbf{NP}$)**.** *A language L is in $\mathbf{NP}$ if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM M such that for every $x \in \{0,1\}^*$,*

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M(x,u) = 1. \tag{2-5}$$

Definition 2.26 ($\mathbf{NP}$-Hardness and $\mathbf{NP}$-Completeness)**.** *L' is* $\mathbf{NP}$-Hard *if $L \leq_m^{\mathbf{P}} L'$ for every $L \in \mathbf{NP}$. L' is* $\mathbf{NP}$-Complete *if L' is* $\mathbf{NP}$-Hard *and $L' \in \mathbf{NP}$.*

Theorem 2.27. *If language L is* $\mathbf{NP}$-Complete*, then $L \in \mathbf{P}$ iff $\mathbf{P} = \mathbf{NP}$.*

Definition 2.28 (SAT)**.** **SAT** *is the language of all satisfiable CNF formulae, where CNF formulae are boolean formula over variables* $u_1, \ldots, u_n$ *for some* n *that is in Conjunctive Normal Form.*

Theorem 2.29. **SAT** *is* **NP**-Complete.

2.3 Parameterized Complexity Theory

The following definitions and theorems are from Flum's book([6], Flum and Grohe, 2006: 1.).

2.3.1 Parameterized Tractability

Definition 2.30 (Parameterized problems)**.** *Let* Σ *be a finite alphabet.*

1. *A* parameterization *of* Σ^* *is a mapping* $\kappa : \Sigma^* \to \mathbb{N}$ *that is polynomial time computable.*

2. *A* parameterized problem *(over* Σ*) is a pair* (Q, κ) *consisting of a set* $\mathcal{Q} \subseteq \Sigma^*$ *and a parameterization* κ *of* Σ^*.

For convenience, in the following chapters we will always write a parameterized problem as $Q(\langle x, k\rangle)$ where x is the input and $k = \kappa(x)$.

Definition 2.31 (**FPT**-algorithm and fixed-parameter tractability)**.** *Let* Σ *be a finite alphabet and* $\kappa : \Sigma^* \to \mathbb{N}$ *a parameterization.*

1. *An algorithm* $\mathcal{A}$ *with input alphabet* Σ *is an* **FPT**-algorithm *(with respect to* κ*) if there is a computable function* $f : \mathbb{N} \to \mathbb{N}$ *and a polynomial* $p : \mathbb{N} \to \mathbb{N}$ *such that for every* $x \in \Sigma^*$*, the running time of* $\mathcal{A}$ *on input* x *is at most* $f(\kappa(x)) \cdot p(|x|)$.

2. *A parameterized problem* (Q, κ) *is* fixed-parameter tractable *if there is an* **FPT**-*algorithm with respect to* κ *that decides* Q.

Definition 2.32 (The class **FPT**)**.** **FPT** *denotes the class of all fixed-parameter tractable problems.*

2.3.2 Reductions under parameterized context

Definition 2.33 (**FPT** m-reduction)**.** *Let (Q, κ) and (Q', κ') be parameterized problems over the alphabet Σ and Σ', respectively. An* **FPT** m-reduction *from (Q, κ) to (Q', κ'), denoted as $(Q, \kappa) \leq_m^{\mathbf{FPT}} (Q', \kappa')$, is a mapping $R : \Sigma^* \to (\Sigma')^*$ such that:*

1. *$\forall x \in \Sigma^*$, $x \in Q \Leftrightarrow R(x) \in Q'$.*
2. *R is computable by an* **FPT***-algorithm.*
3. *There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.*

Definition 2.34 (**FPT** Turing-reduction)**.** *Let (Q, κ) and (Q', κ') be parameterized problems over the alphabet Σ and Σ', respectively. An* **FPT** Turing-reduction *from (Q, κ) to (Q', κ'), denoted as $(Q, \kappa) \leq_T^{\mathbf{FPT}} (Q', \kappa')$, is an algorithm $\mathcal{A}$ with an oracle to Q' such that:*

1. *$\mathcal{A}$ decides (Q, κ).*
2. *$\mathcal{A}$ is an* **FPT***-algorithm.*
3. *There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all oracle queries "$y \in Q'$?" posed by $\mathcal{A}$ on input xV we have $\kappa'(y) \leq g(\kappa(x))$.*

Theorem 2.35. **FPT** *is closed under* **FPT** *m-reduction and* **FPT** *Turing-reduction, i.e. if $(Q, \kappa) \leq^{\mathbf{FPT}} (Q', \kappa')$ and $(Q', \kappa') \in$* **FPT** *then $(Q, \kappa) \in$* **FPT***, where '$\leq^{\mathbf{FPT}}$' can be '$\leq_m^{\mathbf{FPT}}$' or '$\leq_T^{\mathbf{FPT}}$'.*

2.3.3 W-Hierarchy

Definition 2.36 (p-WD$_\psi$)**.** *Let $\psi(X)$ be a first-order formula with a free relation variable X of arity s. Then the parameterized problem p-WD$_\psi$ reads a structure $\mathcal{A}$ and $k \in \mathbb{N}$ as input, k as a parameter, then decides whether there is a relation $S \subseteq \mathcal{A}^*$ of cardinality $|S| = k$ such that $\mathcal{A} \models \psi(S)$.*

Definition 2.37. *Let (Q, κ) be a parameterized problem. Then we denote $[(Q, \kappa)]^{\mathbf{FPT}}$ as the set $\{(Q', \kappa') | (Q', \kappa') \leq_m^{\mathbf{FPT}} (Q, \kappa)\}$.*

Definition 2.38 (The W-Hierarchy)**.** *For every $t \geq 1$, we let $W[t] := [p\text{-}WD\text{-}\Pi_t]^{\mathbf{FPT}}$.*

Definition 2.39 ($\mathbf{W}[1]$-Hardness and $\mathbf{W}[1]$-Completeness)**.** *L' is $\mathbf{W}[1]$-Hard if $L \leq_m^{\mathbf{FPT}} L'$ for every $L \in \mathbf{W}[1]$. L' is $\mathbf{W}[1]$-Complete if L' is $\mathbf{W}[1]$-Hard and $L' \in \mathbf{W}[1]$.*

Theorem 2.40. *If language L is $\mathbf{W}[1]$-Complete, then $L \in \mathbf{FPT}$ iff $\mathbf{FPT} = \mathbf{W}[1]$.*

2.4 Miscellaneous

Definition 2.41. *For sets $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A} \oplus \mathcal{B} = \{2x | x \in \mathcal{A}\} \cup \{2x + 1 | x \in \mathcal{B}\}$.*

Definition 2.42. *For sets $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A} < \mathcal{B}$ iff $\mathcal{A} \leq \mathcal{B}$ and $\mathcal{B} \not\leq \mathcal{B}$, where the '$\leq$' relation here can be '$\leq_m$', '$\leq_T$','$\leq_m^{\mathbf{P}}$','$\leq_T^{\mathbf{P}}$','$\leq_m^{\mathbf{FPT}}$' or '$\leq_T^{\mathbf{FPT}}$', with a corresponding version of the '$<$' relation.*

Definition 2.43 (Non-triviality)**.** *A set $\mathcal{A}$ is* non-trivial *iff it is neither empty nor full.*

Definition 2.44 (Restriction of functions)**.** *Let f be a function and $S \subseteq \mathbb{N}$ be a set, then $f \upharpoonright S$ is the function defined by*

$$(f \upharpoonright S)(x) \simeq \begin{cases} f(x), & \text{if } x \in S, \\ \text{undefined}, & \text{otherwise.} \end{cases} \tag{2-6}$$

Moreover, for a number c we define $(f \upharpoonright c)(x) \simeq (f \upharpoonright S)(x)$ with $S = \{x | x \leq c\}$.

Chapter 3 Ladner Theorem

In this chapter, I will introduce the original Ladner Theorem in classical computational complexity theory and give a concise proof using "blowing holes" technique, derived from Ladner's original paper.

3.1 The Theorem

As we know, which is also included in the "Preliminaries" chapter of this thesis, problems in $\mathbf{P}$ can be considered the 'easiest' problems in $\mathbf{NP}$, while $\mathbf{NP}$-Complete problems are considered the 'hardest' one. An interesting problem comes up naturally: under the assumption that $\mathbf{P} \neq \mathbf{NP}$, are there any problems of 'intermediate hardness' in the polynomial degree? Ladner theorem gives an positive answer, and moreover, its corollary gives a much stronger answer by saying that the polynomial degree are dense([8], Ladner, 1975: 158.).

In classical complexity theory, Ladner Theorem reveals the density property of structures of classical complexity classes under plausible assumption $\mathbf{P} \neq \mathbf{NP}$, which is similar to the density property of real numbers. Between any two real numbers, there are infinitely many intermediate real numbers. Similarly, Ladner Theorem demonstrates that between the 'easiest' problems and the 'hardest' problems in the hierarchy of classical complexity classes, or even between any two problems that are not 'equally hard', there are infinitely many intermediate problems that are not 'equally hard' with them([8], Ladner, 1975: 160.).

Following is the original Ladner Theorem and its general version.

Theorem 3.1 (Ladner)**.** *Assuming* $\mathbf{P} \neq \mathbf{NP}$*, there is an* $\mathbf{NP}$ *problem that is not* $\mathbf{NP}$-Complete*, and not* $\mathbf{P}$.

Theorem 3.2. *For any non-trivial recursive sets* $\mathcal{A} <_m^P \mathcal{B}$*, there is a set* $\mathcal{C}$ *such that* $\mathcal{A} <_m^P \mathcal{C} <_m^P \mathcal{B}$*. For any recursive sets* $\mathcal{A} <_T^P \mathcal{B}$*, there is a set* $\mathcal{C}$ *such that* $\mathcal{A} <_T^P \mathcal{C} <_T^P \mathcal{B}$.

3.2 "Blowing Holes" Proof

Proof. **(of Theorem 3.1)** Fix an effective enumeration of polynomial time-bounded Turing Machines $M_0, M_1, \cdots$, with each M_k running in time $|x|^k$, and a similar effective enumeration of polynomial-time computable functions $f_0, f_1, \cdots$, with each function f_k computable in time $|x|^k$.

The problem we are going to attain is $\mathcal{A} = \{x | x \in \mathbf{SAT} \text{ and } g(|x|) \text{ is even}\}$, which means we are "blowing holes" in a $\mathbf{NP}$-Complete problem. Our construction will ensure that $g(n)$ is computable in polynomial time, therefore $\mathcal{A} \in \mathbf{NP}$.

In order to keep $\mathcal{A}$ 'harder' than any problems in $\mathbf{P}$ and 'easier' than any problems that is $\mathbf{NP}$-Complete simultaneously, we have the following requirements to meet:

$$P'_e : \mathcal{A} \neq L(M_e) \tag{3-1}$$

$$N'_e : \mathbf{SAT} \not\leq_{f_e} \mathcal{A}. \tag{3-2}$$

A detailed version of requirements in our construction is as follows:

$$\begin{aligned} P_e : \quad & ((x \in \mathbf{SAT} \quad \text{and} \quad g(|x|) \text{ is even}) \quad \text{and} \quad x \notin L(M_e)) \\ \text{or} \quad & ((x \notin \mathbf{SAT} \quad \text{or} \quad g(|x|) \text{ is odd}) \quad \text{and} \quad x \in L(M_e)) \end{aligned} \tag{3-3}$$

$$\begin{aligned} N_e : \quad & (x \in \mathbf{SAT} \quad \text{and} \quad (f_e(x) \notin \mathbf{SAT} \quad \text{or} \quad g(|f_e(x)|) \text{ is odd})) \\ \text{or} \quad & (x \notin \mathbf{SAT} \quad \text{and} \quad (f_e(x) \in \mathbf{SAT} \quad \text{and} \quad g(|f_e(x)|) \text{ is even})). \end{aligned} \tag{3-4}$$

The whole procedure is as follows: (Since we are using *delayed diagonalization*, the real *stage* in usual concept of diagonalization method is described by the value of $g(s)$ instead of s.)

'Stage' 0**,** 1 **and** 2**:** Set $g(0) = g(1) = g(2) = 0$.

'Stage' $s+1$ **(**$s \geq 2$**):** Suppose $g(s) = n$. If $(\log\log s)^n \geq \log s$, directly set $g(s+1) = n$; otherwise, we calculate $g(s+1)$ as follow:

- n *is even*: Let $e = \frac{n}{2}$. Enumerate all x s.t. $|x| \leq \log\log s$, testing whether P_e ever

holds. If so, set $g(s+1) = n+1$; otherwise, set $g(s+1) = n$.

- *n is odd*: Let $e = \frac{n-1}{2}$. Enumerate all x s.t. $|x| \leq \log\log s$, testing whether N_e ever holds. If so, set $g(s+1) = n+1$; otherwise, set $g(s+1) = n$.

We need to prove several useful lemmas first, to show that the above procedure of constructing set $\mathcal{A}$ is well-defined, computable in polynomial time and correct.

Lemma 3.3 (Well-definedness)**.** *The above construction of $g(s)$ is well-defined, i.e. $g(s)$ can be computed inductively (i.e. recursively) as described above.*

Proof. Since we will directly go to next 'stage' $s+1$ whenever $(\log\log s)^n \geq \log s$, asymptotically we have $(\log\log s)^n < \log s$. Therefore

$$|f_e(x)| \leq |x|^e \leq (\log\log s)^e \leq (\log\log s)^n < \log s < s, \tag{3-5}$$

the information we need from function $g(s)$ to calculate the s-th 'stage' lies before this stage. □

Lemma 3.4 (Polynomial-time computability)**.** *$g(s)$ is computable in polynomial time of s.*

Proof. Note that we only enumerate x with $|x| \leq \log\log s$ at each 'stage' s, and by equation 3-5, the running time to calculate $g(s)$ inductively is

$$\begin{aligned} T(s) &= T(s-1) + 2^{\log\log s}(O(2^{\log\log s}) + O((\log\log s)^e) + O(2^{(\log\log s)^e})) \\ &\leq T(s-1) + \log s(O(\log s) + O((\log\log s)^n) + O(2^{(\log\log s)^n})) \\ &\leq T(s-1) + \log s(O(\log s) + O(\log s) + O(2^{\log s})) \\ &\leq T(s-1) + O(s^2). \end{aligned} \tag{3-6}$$

$g(s)$ is thus computable in polynomial time of s. □

Lemma 3.5 (Correctness)**.** *$g(s)$ is unbounded, i.e. our construction will not get stuck in some real stage n such that there exists a 'stage' s', $\forall s \geq s'$, $g(s) = n$.*

Proof.

- The construction will not get stuck by the condition "$(\log\log s)^n \geq \log s$": for fixed n, as $\log s$ grows faster than $(\log\log s)^n$, there must be a sufficient large s such that $(\log\log s)^n < \log s$.

- It will not get stuck in n such that $n = 2e$: otherwise $\exists s$ such that $\forall |x| \geq s$, $x \in L(M_e)$ iff $x \in \mathbf{SAT}$, hence we can have a polynomial-time algorithm for $\mathbf{SAT}$, contradicting our $\mathbf{P} \neq \mathbf{NP}$ assumption.

- It will not get stuck in n such that $n = 2e+1$: otherwise $\exists s$ such that $\forall |x| \geq s$, $x \notin \mathcal{A}$ and $\forall x$, $x \in \mathbf{SAT}$ iff $f_e(x) \in \mathcal{A}$, hence we have a polynomial-time m-reduction from $\mathbf{SAT}$ to a polynomial-time computable set $\mathcal{A}$, hence $\mathbf{SAT} \in P$, contradiction our $\mathbf{P} \neq \mathbf{NP}$ assumption.

Therefore $g(s)$ is unbounded. □

By the above lemmas, all the requirements P_e's and N_e's will be checked and satisfied, hence all the P'_e's and N'_e's are satisfied, therefore $\mathcal{A} \notin \mathbf{P}$ and $\mathcal{A}$ is not $\mathbf{NP}$-Complete while $\mathcal{A} \in \mathbf{NP}$. □

3.2.1 An Alternative Proof

In the above proof, we use the fact that $\mathbf{SAT}$ can be solved in $O(2^{|x|})$ time. However, it is not essential for this kind of proof. In the original paper of Ladner, he use another approach to overcome difficulty of simulating $\mathbf{SAT}$ problem([8], Ladner, 1975: 159.), which is more general and can be used in the proof of its parameterized version.

In addition, we will use $\mathbf{P}$-time m-reductions in both sides of requirements, making the following proof more general to be used in proving Theorem 3.2.

Proof. **(of Theorem 3.1)** Similarly as above, we have an effective enumeration of polynomial time-bounded Turing Machines $\{M_k\}_{k\in\mathbb{N}}$, and an effective enumeration of polynomial-time computable functions $\{f_k\}_{k\in\mathbb{N}}$.

In addition, we have a non-trivial recursive set $\mathcal{A} \in \mathbf{P}$ and a recursive set $\mathcal{B}$ that is $\mathbf{NP}$-Complete. Then there must be x_0 such that $\mathcal{A}(x_0) = 0$ and x_1 such that $\mathcal{A}(x_1) = 1$.

The problem we are going to attain is $\mathcal{C} = \{x | x \in \mathcal{B} \text{ and } g(|x|) \text{ is even}\}$, which means we are "blowing holes" in $\mathcal{B}$. Our construction will ensure that $g(s)$ is computable in polynomial time of s, therefore $\mathcal{C} \leq_m^{\mathbf{P}} \mathcal{B}$ since $\mathcal{B}$ is obviously non-trivial, thus $\mathcal{C} \in \mathbf{NP}$.

Requirements now become the following.

$$P'_e : \mathcal{C} \not\leq_{f_e} \mathcal{A} \tag{3-7}$$

$$N'_e : \mathcal{B} \not\leq_{f_e} \mathcal{C}. \tag{3-8}$$

A detailed version of requirements in our construction is as follow:

$$P_e : \mathcal{C}_s(x) \neq \mathcal{A}_s(f_e(x)) \tag{3-9}$$

$$N_e : \mathcal{B}_s(x) \neq \mathcal{C}_s(f_e(x)), \tag{3-10}$$

where $\mathcal{A}_s$ is the function of simulating the program for $\mathcal{A}$ up to s steps, $\mathcal{C}_s(x)$ is the function obtained by either returning 0 immediately, or simulating $\mathcal{B}$ on x for s steps, according to the value of $g(|x|)$. The operator '$\neq$' will be satisfied only when its operands are converged and unequal.

The calculation of $g(s)$ is similar as in the previous proof, except that the condition for directly jump into the next 'stage' can now be relaxed to $(\log s)^n \geq s$ and we can enumerate all $|x| \leq \log s$.

Well-definedness of this construction is similar since now we have

$$|f_e(x)| \leq |x|^e \leq (\log s)^e \leq (\log s)^n < s. \tag{3-11}$$

Polynomial-time computability of this construction is also similar since we only simulate $\mathcal{A}$ and $\mathcal{B}$ for up to s steps in each stage s.

For correctness of this construction, if it is stuck in n such that $n = 2e$, then we have a $\mathbf{P}$-time m-reduction from $\mathcal{C}$ to $\mathcal{A}$, while $\mathcal{C}$ is almost the same as $\mathcal{B}$ except for finite elements at the beginning. Then we can get an m-reduction from $\mathcal{B}$ to $\mathcal{A}$ as follow: for

the beginning part of $\mathcal{B}$, directly calculates them and output x_0 if the result is 0, x_1 if it is 1; for the elements on which $\mathcal{C}$ and $\mathcal{B}$ are the same, use the above m-reduction from $\mathcal{C}$ to $\mathcal{A}$. This reduction obviously runs in polynomial time. Hence we have $\mathcal{B} \leq_m^{\mathbf{P}} \mathcal{A}$, thus $\mathcal{B} \in \mathbf{P}$, contradicting the assumption $\mathbf{P} \neq \mathbf{NP}$.

If the construction is stuck in n such that $n = 2e + 1$, then we have a $\mathbf{P}$-time m-reduction from $\mathcal{B}$ to $\mathcal{C}$, while $\mathcal{C}$ is almost empty except for finite elements at the beginning. Then we can easily get a polynomial-time algorithm for $\mathcal{B}$ as follow: calculates the image of x under the $\mathbf{P}$-time m-reduction from $\mathcal{B}$ to $\mathcal{C}$; if it lies in the beginning part of $\mathcal{C}$, then use this construction and the original program of $\mathcal{B}$ to calculate it; if it lies in the part that $\mathcal{C}$ is empty, then output 0. This reduction obviously runs in polynomial time. Besides, we can also use this algorithm to get a $\mathbf{P}$-time m-reduction from $\mathcal{B}$ to $\mathcal{A}$ simply by running it and then outputting x_0 or x_1 accordingly. Hence we also have $\mathcal{B} \leq_m^{\mathbf{P}} \mathcal{A}$ and $\mathcal{B} \in \mathbf{P}$, leading to the same contradiction.

Therefore $\mathcal{C} \not\leq_m^{\mathbf{P}} \mathcal{A}$, $\mathcal{B} \not\leq_m^{\mathbf{P}} \mathcal{C}$ and $\mathcal{C} \leq_m^{\mathbf{P}} \mathcal{B}$, thus $\mathcal{C}$ is neither $\mathbf{P}$ nor $\mathbf{NP}$-Complete while $\mathcal{C} \in \mathbf{NP}$. □

In the above proof, we don't use the fact that $\mathcal{B}$ is $\mathbf{NP}$-Complete, except for introducing contradiction to the assumption $\mathbf{P} \neq \mathbf{NP}$ and getting the finally result. However, for $\mathbf{P}$-time m-reduction part of Theorem 3.2, proving $\mathcal{B} \leq_m^{\mathbf{P}} \mathcal{A}$ as a contradiction is adequate. Besides, now we need to ensure that the constructed set $\mathcal{C}$ satisfies $\mathcal{A} \leq_m^{\mathbf{P}} \mathcal{C}$. We can easily do it by letting the desired set $\mathcal{C}$ be $\mathcal{C}' \oplus \mathcal{A}$, where $\mathcal{C}'$ is constructed by a similar method with the previous proof, with slight modifications that now N_e becomes $\mathcal{B}_s(x) \neq (\mathcal{C}'_s \oplus \mathcal{A})(f_e(x))$ and that the polynomial-time m-reduction from $\mathcal{B}$ to $\mathcal{A}$ made for contradiction in situations of getting stuck in N_e will now have two cases according to where the input is mapped to by the polynomial-time m-reduction from $\mathcal{B}$ to $\mathcal{C}' \oplus \mathcal{A}$, which can be solved easily by definition of the '$\oplus$' operator. Therefore it is straightforward to see that the above proof can also be used to prove Theorem 3.2, especially for $\mathbf{P}$-time m-reduction.

For $\mathbf{P}$-time Turing-reduction part of Theorem 3.2, slight but easy modifications are needed. All the requirements and reductions we obtained in the proof for contradiction should be changed into Turing-reduction's styles. Besides, we will restrict $\mathcal{A}_s^{\mathcal{O}}$ to run

upto s steps including steps taken by the oracle machine. Observe that there is no need to restrict the set $\mathcal{A}$ to be non-trivial, since we can directly output a $\{0,1\}$-value in Turing-reductions. Then similarly we can get this result.

3.3 Further Discussion

3.3.1 Effective Enumeration of Polynomial Time-bounded Turing Machines

It may be confusing to demonstrate that we can have an effective enumeration of polynomial time-bounded Turing Machines([2], Homer and Selman, 2011: 127.), since the following set, which contains *all* polynomial time-bounded TMs,

$$PTM = \{i| \text{ the } i\text{-th Turing Machine halts on all input in polynomial time}\} \quad (3\text{-}12)$$

is not r.e., otherwise we can solve the Halting problem.

However, we can have a set $\{M_i\}_i$ s.t. $\mathbf{P} = \cup_i\{L(M_i)\}$ and it is r.e. Of course it doesn't contain *all* polynomial time-bounded TMs, but we don't care the omitting ones, since it is sufficient for our proof. Its existence can be precisely proved as follows:

Let $\{M_i'\}_i$ be a standard effective enumeration of Turing Machines. Let $p_k = n^k + k$, which is obviously *time constructible*. Denote C_k as the p_k-clocked TMs. Then we attach C_k to each TMs, obtaining effectively $M_{\langle i,j\rangle} = M_i'||C_j$, where the '||' operator denotes simultaneous simulation([2], Homer and Selman, 2011: 90.). By certain properties of simultaneous simulation and *Linear Speedup Theorem*, we can prove that $M_{\langle i,j\rangle}$ are p_j time-bounded TMs, which makes them an effective enumeration of polynomial time-bounded TMs. In addition, since $\langle i,j\rangle \geq j$, we have M_k always run in time p_k.

Note that the additive term in p_k is necessary since for $k = 1$, by *Linear Speedup Theorem* and its corollaries we can only obtain a time bound $\mathbf{DTIME}(n+1)$ instead of $\mathbf{DTIME}(n)$ for our constructed TMs.

3.3.2 Essentials of This Proof

In the above construction, the desired set is constructed *stage by stage*. At each stage, we will either imitate some elements of the 'hard' set $\mathcal{B}$ in $\mathcal{C}$, or set 0's, which is the so-called "holes" in $\mathcal{C}$. When we are imitating $\mathcal{B}$, we attempt to make the constructed set 'hard' enough so that it cannot be reduced to the 'easy' problem, thus satisfying certain requirement. When we are "blowing holes", we attempt to make the constructed set 'easy' enough so that the 'hard' problem cannot be reduced to it, thus satisfying certain requirement. The hardness difference of the 'easy' problem and the 'hard' problem assure that this kind of attempts can always succeed. Therefore by this "blowing holes" technique the constructed set can have an immediate hardness.

In the above construction, we will try to find a x to satisfy each requirement. Such value is called a witness for each requirement. In this setting, the above construction is indeed a diagonalization method, Which was first invented by Cantor to prove that real numbers are uncountably many. As each requirement corresponds to each $\mathbf{P}$-time reduction, it is essential for this proof that polynomial time-bounded Turing Machines are effectively enumerable so that we can apply this diagonalization method. However, parameterized reductions cannot be effectively enumerated, hence more generalized diagonalization method is needed, which is the finite injury priority method in the following chapter.

Chapter 4 Finite Injury Priority Method

Finite injury priority method is used to construct a desired problem, i.e. set that have the property we want. In this method, the set is constructed *by stages* to meet certain requirements $\{R_n\}_{n\in\mathbb{N}}$. "Priority" means we will assign different priority to each requirement. Requirement R_n has higher priority than requirement R_m with $n < m$, i.e. we will try to satisfy R_n earlier than R_m with $n < m$. Action taken for satisfying R_m at stage t may be undone later at stage $s > t$ when we are satisfying R_n with $n < m$, even if R_m is already satisfied, which is called that R_m is *injured* at stage s. That is where "injury" comes. Finally, "finite" comes from the property of this method that a requirement is injured at most finite times([3], Soare, 1987: 110.).

In this chapter, I will introduce finite injury priority method using proofs of two interesting and inspiring theorems as examples.

4.1 Friedberg-Muchnik Theorem

In his fundamental paper introducing Turing degrees in recursion theory([3], Soare, 1987: 77.), Post attempted to classify the r.e. sets and r.e. degrees([12], Post *et al.*, 1944: 291.), therefore raised the question which later became known as "Post's Problem": are there more than two r.e. degrees? Friedberg and Muchnik gave a positive answer to this problem independently, using finite injury priority method. Following are the theorem they proved and a concise proof of mine adapted from Soare's book([3], Soare, 1987: 118.):

Theorem 4.1 (Friedberg-Muchnik)**.** *There exists two r.e. sets $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \not\leq_T \mathcal{B}$ and $\mathcal{B} \not\leq_T \mathcal{A}$ (incomparable).*

Proof. We can construct these two sets by the so-called "Finite Injury Priority Method."

Generally speaking, we have the following requirements to meet, in order to make $\mathcal{A}$ and $\mathcal{B}$ incomparable:

$$R_{2e} : \mathcal{A} \neq \phi_e^{\mathcal{B}} \tag{4-1}$$

$$R_{2e+1} : \mathcal{B} \neq \phi_e^{\mathcal{A}}, \tag{4-2}$$

where $\phi_e^{\mathcal{A}}$ is the partial recursive function computed by the e-th relativised Turing Machine using oracle $\mathcal{A}$.

Let $\mathcal{A} = \cup_{s\in\mathbb{N}}\mathcal{A}_s$, $\mathcal{B} = \cup_{s\in\mathbb{N}}\mathcal{B}_s$, where $\mathcal{A}_s$ and $\mathcal{B}_s$ be the sets we obtain at stage s for $\mathcal{A}$ and $\mathcal{B}$, respectively. Let disagreement witnesses $x_e = \lim_{s\to\infty} x_{e,s}$, where $x_{e,s}$ are potential witnesses for the following detailed requirement $R_{e,s}$ at each stage s:

$$R_{2e,s} : M_{e,s}^{\mathcal{B}_s}(x_{e,s}) \downarrow = 0 \tag{4-3}$$

$$R_{2e+1,s} : M_{e,s}^{\mathcal{A}_s}(x_{e,s}) \downarrow = 0, \tag{4-4}$$

where $M_{e,s}^{\mathcal{A}}$ is the e-th relativised Turing Machine using oracle $\mathcal{A}$, in the restriction of s steps of computation. In order to limit injuries, we will also calculate a restraint number $r_{e,s}$ in our construction of $\mathcal{A}$ and $\mathcal{B}$.

The whole procedure is as follows:

Stage 0**:** Set $\mathcal{A}_0 = \mathcal{B}_0 = \varnothing$, and $\forall e$, set $x_{e,0} = \langle 0, e\rangle$ (where $\langle x, y\rangle$ is the standard pairing function, in order to make witnesses for different requirements different), $r_{e,0} = -1$ (meaning they need to be checked).

Stage $s+1$**:** Let $i = min\{e | R_{e,s}$ is satisfied with $r_{e,s} = -1\}$.

- $i = 2e$: add i into $\mathcal{A}$ by setting $\mathcal{A}_{s+1} = \mathcal{A}_s \cup \{i\}$, keep $\mathcal{B}_{s+1} = \mathcal{B}_s$. Define the use function u as

$$u(\mathcal{A}; e, s, x) = \max\{y | M_{e,s}^{\mathcal{A}}(x) \text{ ask the oracle '} y \in \mathcal{A}?\text{'}\}. \tag{4-5}$$

  Calculate $r_{i,s} = u(\mathcal{B}_s; i, s, x_{i,s})$.

  – $\forall j < i$: set $r_{j,s+1} = r_{j,s}$, $x_{j,s+1} = x_{j,s}$.

– $\forall j > i$: set $r_{j,s+1} = -1$, and

$$x_{j,s+1} = min\{x | x \notin \mathcal{A}_{s+1} \cup \mathcal{B}_{s+1} \text{ and } x = \langle y, j\rangle \text{ and } y > r_{i',s}, \forall i' \leq i \text{ and } x \geq x_{j,s'}, \forall s' \leq s\}. \quad (4\text{-}6)$$

- $i = 2e + 1$: exactly the same as above, except that all things about $\mathcal{B}$ should be replaced by $\mathcal{A}$, and $\mathcal{A}$ replaced by $\mathcal{B}$.
- *No such i exists*: set $\mathcal{A}_{s+1} = \mathcal{A}_s$, $\mathcal{B}_{s+1} = \mathcal{B}_s$, $\forall e$, $x_{e,s+1} = x_{e,s}$, $r_{e,s+1} = r_{e,s}$, and move on.

Lemma 4.2. *By the above construction, $\mathcal{A}$ and $\mathcal{B}$ are recursively enumerable sets.*

Proof. Note that we only add numbers into $\mathcal{A}$ and $\mathcal{B}$, hence it is sufficient to prove each stage is indeed recursive in order to prove $\mathcal{A}$ and $\mathcal{B}$ r.e. However, it is not easy to observe their recursiveness, since the above description is more likely in a mathematician's manner, involving setting infinite ones of $x_{e,s}$ and $r_{e,s}$. Nonetheless, from a programming perspective, we don't have to update all them, which makes each stage recursive. Observing that $M_{e,s'}^{\mathcal{A}_s}(x_{e,s}) \downarrow = M_{e,s}^{\mathcal{A}_s}(x_{e,s}), \forall s' \geq s$ whenever $M_{e,s}^{\mathcal{A}_s}(x_{e,s}) \downarrow$, it is enough for our construction to consider satisfied $R_{e,s}$ with $e \leq s$ at each stage s. Therefore at each stage we only have a limited number of requirements to check, thus a limited number of $x_{e,s}$ and $r_{e,s}$ to update, by maintaining a list of 'need-to-check' tuples of $\langle e, s, x_{e,s}, r_{e,s}\rangle$ technically. □

Lemma 4.3. *All the requirements are satisfied.*

Proof. Let $r_e = \lim_{s\to\infty} r_{e,s}$. Observe that $r_{e,s}$ will only be reset to -1 by requirements $R_{e',s}$ with $e' \leq e$, which makes it 'injured' at most finite times. (Hence this method is called "Finite Injury Method") Therefore, for each e, if there is an x such that $M_e^{\mathcal{B}}(x) \downarrow = 0$, there exists s_e such that $r_e = r_{e,s_e}$ and $x_e = x_{e,s_e}$. Then we have, e.g.

$$M_{e,s_e}^{\mathcal{B}}(x_e) \simeq M_{e,s_e}^{\mathcal{B}\upharpoonright r_e}(x_e) \simeq M_{e,s_e}^{\mathcal{B}_{s_e}}(x_{s_e}) \downarrow = 0 \neq A_{s_e}(x_{e,s_e}) = A(x_e), \quad (4\text{-}7)$$

by the definition of $r_{e,s}$ (for the first equality), and since we will prevent numbers smaller that r_e entering $\mathcal{B}$ after stage s_e (for the second equality). Therefore x_e is really a disagreement witness for the original requirements that we need to care about. (Notice that some requirements R_{2e} or R_{2e+1} are satisfied automatically if there is no x such that $M_e^{\mathcal{B}}(x)\downarrow = 0$, for R_{2e} e.g.) □

Therefore, we get two r.e. sets $\mathcal{A}$ and $\mathcal{B}$ that are incomparable.

□

4.2 Splitting Theorem

Finite injury priority method can also be used to proved the following theorem, which was first proved by Sacks, generalizing two former results of Friedberg([3], Soare, 1987: 121.). Particularly, the technique in this proof is very powerful and is essential on infinite injury priority method, which is a further development of finite injury priority method([3], Soare, 1987: 129.). Although we don't need infinite injury to prove the main theorem in this thesis, it remains a good and useful exercise to prove this theorem to get familiar with finite injury priority method.

Theorem 4.4. *For any non-recursive r.e. set $\mathcal{C}$, there exists two r.e. sets $\mathcal{D}_0$ and $\mathcal{D}_1$ such that $\mathcal{D}_0$ and $\mathcal{D}_1$ are incomparable, $\mathcal{D}_0 \cup \mathcal{D}_1 = \mathcal{C}$, $\mathcal{D}_0 \cap \mathcal{D}_1 = \varnothing$.*

Proof. In this proof, we will construct $\mathcal{D}_0$ and $\mathcal{D}_1$ such that $\mathcal{C} \not\leq_T \mathcal{D}_0$, $\mathcal{C} \not\leq_T \mathcal{D}_1$, and that $\mathcal{C}$ is the disjoint union of $\mathcal{D}_0$ and $\mathcal{D}_1$. Thus we have the following negative requirements to satisfy:

$$N_{\langle e,i\rangle} : C \neq \phi_e^{\mathcal{D}_i}, i = 0, 1. \tag{4-8}$$

$\mathcal{C}$ is non-recursive r.e., hence there is a recursive enumeration without repetition g of $\mathcal{C}$, i.e. $\mathcal{C} = \{g(0), g(1), g(2), \ldots\}$. Denote $\mathcal{C}_s = \{g(0), \ldots, g(s)\}$. The positive requirements are the following:

$$P_s : g(s) \in \mathcal{D}_0 \vee g(s) \in \mathcal{D}_1. \tag{4-9}$$

Let $\mathcal{D}_0 = \cup_{s\in\mathbb{N}}\mathcal{D}_{0,s}$, $\mathcal{D}_1 = \cup_{s\in\mathbb{N}}\mathcal{D}_{1,s}$, where $\mathcal{D}_{0,s}$ and $\mathcal{D}_{1,s}$ be the sets we obtain at stage s for $\mathcal{D}_0$ and $\mathcal{D}_1$, respectively. The whole procedure is as follow:

Stage 0**:** Set $\mathcal{D}_{0,0} = \varnothing$, $\mathcal{D}_{1,0} = \varnothing$.

Stage $s+1$**:** Calculate the following two functions:

$$\text{length function: } l(i,e,s) = \{x|\forall y < x, M_{e,s}^{\mathcal{D}_{i,s}}(y)\downarrow = \mathcal{C}_s(y)\} \tag{4-10}$$

$$\text{restraint function: } r(i,e,s) = \max\{u(\mathcal{D}_{i,s};e,s,y)|y \leq l(i,e,s)\}. \tag{4-11}$$

Notice that we use "$y \leq l(i,e,s)$" in our definition of $r(i,e,s)$, hence for each requirements we will obtain $\{y|y < l(i,e,s)\}$ as *agreement* witnesses and $y = l(i,e,s)$ as a potential *disagreement* witness if possible.

Let $\langle e,i\rangle = \min\{\langle e',i'\rangle|g(s) \leq r(i',e',s)\}$, which means we will choose the negative requirement with the highest priority that will be injured by enumerating $g(s)$ in *its* side, and we say that all the $\langle e',i'\rangle$'s with $g(s) \leq r(i',e',s)$ *receive attentions* at stage s.

- *No such* $\langle e,i\rangle$ *exists*: Let $\mathcal{D}_{0,s+1} = \mathcal{D}_{0,s} \cup \{g(s)\}$, $\mathcal{D}_{1,s+1} = \mathcal{D}_{1,s}$, since we will not injure anything.
- *Otherwise*: Let $\mathcal{D}_{1-i,s+1} = \mathcal{D}_{1-i,s} \cup \{g(s)\}$, $\mathcal{D}_{i,s+1} = \mathcal{D}_{i,s}$, which means we will enumerate $g(s)$ in the *other* side.

Lemma 4.5. *The constructed sets $\mathcal{D}_0$ and $\mathcal{D}_1$ are recursively enumerable.*

Proof. Similarly as the proof of recursive enumerability of $\mathcal{A}$ and $\mathcal{B}$ in Friedberg-Muchnik Theorem. □

Lemma 4.6. *The following three identities hold for all $\langle e,i\rangle$:*

(i) The injury set $I_{e,i} = \{x|\exists s, x \leq r(i,e,s)$ and $x \in \mathcal{D}_i - \mathcal{D}_{i,s}\}$ is finite.

(ii) $\mathcal{C} \neq \phi_e^{\mathcal{D}_i}$.

(iii) $r(i,e) = \lim_{s\to\infty} r(i,e,s)$ *exists and is finite.*

Proof. We will prove them by induction on $\langle e,i\rangle$. To use induction for a $\langle e,i\rangle$ we want to prove them on, first assume that the above three identities hold for all $\langle e',i'\rangle < \langle e,i\rangle$. Define $ss(i',e') = \min\{s|\forall s' \geq s, r(i',e',s') = r(i',e')\}$ for all $\langle e',i'\rangle < \langle e,i\rangle$. Let $r_{\max} = \max\{r(i',e')|\langle e',i'\rangle < \langle e,i\rangle\} + 1$, $s_{\max} = \max\{ss(i',e')|\langle e',i'\rangle < \langle e,i\rangle\}$. Since $r_{\max}$ is finite, there exist some stage s s.t. $\mathcal{C}_s \upharpoonright r_{\max} = \mathcal{C} \upharpoonright r_{\max}$, fix such a s with $s \geq s_{\max}$ for the following proof.

For i, after stage s, there will not be numbers smaller than $r_{\max}$ entering $\mathcal{D}_0$ and $\mathcal{D}_1$, and $r(i',e',s)$ with $\langle e',i'\rangle < \langle e,i\rangle$ will not change any more, hence none of the $\langle e',i'\rangle$'s with $\langle e',i'\rangle < \langle e,i\rangle$ will receive attentions. That is to say, for stage $s' \geq s$, if receiving attentions $\langle e,i\rangle$ will be with the highest priority, thus by our construction $g(s')$ will be enumerated into the other side $\mathcal{D}_{1-i}$ instead of $\mathcal{D}_i$. Therefore $\langle e,i\rangle$ will not be injured from stage s on, i.e. the injury set is finite.

Before proving the remaining two parts, we have some useful observations: for any stage $s' \geq s$,

1. for $y < l(i,e,s')$, since $\langle e,i\rangle$ will not be hurt, $M_{e,s'}^{\mathcal{D}_{i,s'+1}}(y) \simeq M_{e,s'}^{\mathcal{D}_{i,s'}}(y)\downarrow$. Hence $M_{e,s'+1}^{\mathcal{D}_{i,s'+1}}(y) = M_{e,s'}^{\mathcal{D}_{i,s'}}(y) = C_{s'}(y)$, $u(\mathcal{D}_{i,s'+1}; e, s'+1, y) = u(\mathcal{D}_{i,s'}; e, s', y)$.

2. for $y = l(i,e,s')$, if $M_{e,s'}^{\mathcal{D}_{i,s'}}(y)\downarrow$, then $M_{e,s'}^{\mathcal{D}_{i,s'}}(y) \neq C_{s'}(y)$. Moreover, we have $M_{e,s'+1}^{\mathcal{D}_{i,s'+1}}(y) = M_{e,s'}^{\mathcal{D}_{i,s'}}(y) \neq C_{s'}(y)$ and $u(\mathcal{D}_{i,s'+1}; e, s'+1, y) = u(\mathcal{D}_{i,s'}; e, s', y)$ similarly as above.

For ii, suppose not then we can prove $\mathcal{C}$ is recursive by showing a procedure to compute $\mathcal{C}$ as follows, which leads to a contradiction: for any $q \in \mathbb{N}$, since $\lim_{s\to\infty} l(i,e,s) = \infty$ we wait until some stage $s' \geq s$ such that $l(i,e,s') > q$, output $M_{e,s'}^{\mathcal{D}_{i,s'}}(q)$. It remains to prove inductively that

$$\forall t \geq s' \ (l(i,e,t) > q \text{ and } r(i,e,t) \geq \max\{u(\mathcal{D}_{i,s'}; e, s', y)|y \leq q\}), \tag{4-12}$$

thus using observation 1, by induction $\forall t \geq s' \ (M_e^{\mathcal{D}_{i,t}}(q) = M_e^{\mathcal{D}_{i,s'}}(q))$, therefore $M_{e,s'}^{\mathcal{D}_{i,s'}}(q)\downarrow = M_e^{\mathcal{D}_{i,s'}}(q) = M_e^{\mathcal{D}_i}(q) = \mathcal{C}(q)$, i.e. we have correctly computed $\mathcal{C}(q)$

by outputting $M_{e,s'}^{\mathcal{D}_{i,s'}}(q)$.

- To prove it, assume it holds for t. Then for $t+1$, by observation 1, the computation for $y \leq q$ remains the same, hence $l(i,e,t+1) > q$ and so $r(i,e,t+1) \geq \max\{u(\mathcal{D}_{i,t+1}; e, t+1, y) | y \leq q\} = \max\{u(\mathcal{D}_{i,s'}; e, s', y) | y \leq q\}$, unless there is some $y \leq q$ s.t. $\mathcal{C}_t(y) \neq \mathcal{C}_{t+1}(y)$.

  - If so, denote the least one as q', thus $l(i,e,t+1) = q'$. Since we only add numbers into $\mathcal{C}$, it must be the situation that $\mathcal{C}_t(q') = 0$ and $\forall t' > t, \mathcal{C}_{t'}(q') = 1$. By observation 2, using induction we can prove for all $t' > t$, $M_{e,t'}^{\mathcal{D}_{i,t'}}(q') \simeq M_{e,t}^{\mathcal{D}_{i,t}}(q') \downarrow = \mathcal{C}_t(q') \neq \mathcal{C}_{t'}(q') = \mathcal{C}(q')$, hence $M_e^{\mathcal{D}_i}(q') \neq \mathcal{C}(q')$, a contradiction to our assumption.

By ii we can obtain $p = \mu y(M_e^{\mathcal{D}_i}(y) \not\simeq \mathcal{C}(y))$ for proving iii. Select a sufficient large $s' \geq s$ such that for all $s'' \geq s'$,

(a) $\forall y < p, M_{e,s''}^{\mathcal{D}_{i,s''}}(y) \downarrow = M_e^{\mathcal{D}_i}(y)$,

(b) $\forall y \leq p, \mathcal{C}_{s''}(y) = \mathcal{C}(y)$,

i.e. s' is large enough that $M_{e,s'}^{\mathcal{D}_{i,s'}}$ and $\mathcal{C}_{s'}$ will not change any more for $y < p$ and $y \leq p$, respectively. Then there are two possibilities:

- $M_e^{\mathcal{D}_i}(p) \uparrow$: then $\forall s'' \geq s'$, $l(i,e,s'') = p$, so using observation 1 by induction, $r(i,e,s'') = r(i,e,s')$. Hence $r(i,e) = r(i,e,s')$ is finite.

- $\exists t \geq s', M_{e,t}^{\mathcal{D}_{i,t}}(p) \downarrow \neq \mathcal{C}(p)$: then using observation 2 by induction, we can prove

$$\forall t' \geq t \ (M_{e,t'}^{\mathcal{D}_{i,t'}}(p) \downarrow \neq \mathcal{C}(p) \text{ and } l(i,e,t') = p \text{ and } r(i,e,t') = r(i,e,t)),$$

  thus $r(i,e) = r(i,e,t)$ is finite.

□

Therefore, by the construction we have $\mathcal{D}_0, \mathcal{D}_1 <_T \mathcal{C}$. If $\mathcal{D}_0$ and $\mathcal{D}_1$ are comparable, say e.g., $\mathcal{D}_0 \leq_T \mathcal{D}_1$, then $\mathcal{C} = \mathcal{D}_0 \otimes \mathcal{D}_1 \leq_T \mathcal{D}_1$, a contradiction. □

Chapter 5 Ladner Theorem in Parameterized Complexity Theory

Results similar with Ladner Theorem in classical complexity theory will be proven in this chapter for parameterized complexity theory, showing that the W-hierarchy in parameterized complexity is also dense, as the polynomial hierarchy in classical complexity, with respect to both $\mathbf{FPT}$ m-reduction and $\mathbf{FPT}$ Turing-reduction. Therefore intuitively we can say the structure of parameterized complexity classes and that of classical one have something in common essentially, although the former one was first defined by closure of reduction ([6], Flum and Grohe, 2006: 104.)(machine characterization was given later([13], Chen et al., 2005: 177.)) while the latter one was first defined by machine characterization([5], Arora and Barak, 2009: 39.).

5.1 The Theorem

Following is the parameterized version of Ladner Theorem and its general version.

Theorem 5.1. *Assuming* $\mathbf{FPT} \neq \mathbf{W}[1]$*, there is a parameterized problem* $\mathcal{C}(\langle x, k\rangle) \in \mathbf{W}[1]$ *that is neither* $\mathbf{FPT}$ *nor* $\mathbf{W}[1]$-Complete.

Theorem 5.2. *For any non-trivial recursive sets* $\mathcal{A} <_m^{\mathbf{FPT}} \mathcal{B}$*, there is a set* $\mathcal{C}$ *such that* $\mathcal{A} <_m^{\mathbf{FPT}} \mathcal{C} <_m^{\mathbf{FPT}} \mathcal{B}$*. For any recursive sets* $\mathcal{A} <_T^{\mathbf{FPT}} \mathcal{B}$*, there is a set* $\mathcal{C}$ *such that* $\mathcal{A} <_T^{\mathbf{FPT}} \mathcal{C} <_T^{\mathbf{FPT}} \mathcal{B}$.

5.2 Sketch of The Proof

5.2.1 Notations

In the following parts, I will use $\langle \cdot, \cdot \rangle$ for the standard pairing function, and define $\langle \cdot, \cdot, \cdot \rangle = \langle \langle \cdot, \cdot \rangle, \cdot \rangle$, $\langle \cdot, \cdot, \cdot, 0 \rangle = 2\langle \cdot, \cdot, \cdot \rangle$, $\langle \cdot, \cdot, \cdot, 1 \rangle = 2\langle \cdot, \cdot, \cdot \rangle + 1$.

$\phi_e(x)$ is the e-th Turing Machine with input x, given by standard enumeration, $\phi_{e,s}(x)$ is its computation upto s steps. $\Psi_e(\langle x, k\rangle)$ is the e-th parameterized Turing Machine with input x, parameter k.

Note that all the sets discussed here are recursive. Denote $\mathcal{A}^{(k)} = \{\langle x, k'\rangle | \langle x, k'\rangle \in \mathcal{A} \text{ and } k' = k\}$, $\mathcal{A}^{(\leq k)} = \bigcup_{k' \leq k} A^{(k')} = \{\langle x, k'\rangle | \langle x, k'\rangle \in \mathcal{A} \text{ and } k' \leq k\}$. Besides, $\mathcal{A}^{(\leq k)}$ will refuse to answer $\langle x', k'\rangle$ with $k' > k$, which can be implemented simply by returning values other than 0 and 1.

Denote $\mathcal{A}_s$ as the function of simulating the program for $\mathcal{A}$ up to s steps. The operator '$\neq$' will be satisfied only when its operands are converged and unequal.

5.2.2 Discussion

In this proof, given a non-trivial $\mathbf{FPT}$ problem $\mathcal{A}$ and a $\mathbf{W}[\mathbf{1}]$-Complete parameterized problem $\mathcal{B}$, I will construct a parameterized problem $\mathcal{C} \in \mathbf{W}[\mathbf{1}]$ such that

$$\begin{cases} \mathcal{C} & \notin \mathbf{FPT} \\ \mathcal{C} & \notin \mathbf{W}[\mathbf{1}]\text{-Complete} \end{cases}, \tag{5-1}$$

that is to say, by definition of $\mathbf{FPT}$ and $\mathbf{W}[\mathbf{1}]$-Complete, the constructed set $\mathcal{C}$ should satisfy that $\mathcal{C} \leq_m^{\mathbf{FPT}} \mathcal{B}$ and

$$\begin{cases} \mathcal{C} \not\leq_m^{\mathbf{FPT}} \mathcal{A} \\ \mathcal{B} \not\leq_m^{\mathbf{FPT}} \mathcal{C} \end{cases}. \tag{5-2}$$

For the first condition, the detailed requirements are the following:

$$R_{\langle e_1, e_2, n, 0\rangle}: \quad \exists k \exists s \exists x \text{ such that } \left(\begin{array}{l} \phi_{e_1}(k) \uparrow \\ \text{or } \left(\phi_{e_1,s}(k) \downarrow \text{ and } \left(\begin{array}{l} \Psi_{e_2}(\langle x, k\rangle) \not\downarrow \text{ in } \phi_{e_1}(k)|x|^n \text{ steps} \\ \text{or } \mathcal{A}^{(\leq \phi_{e_1}(k))}(\Psi_{e_2}(\langle x, k\rangle)) \neq \mathcal{C}(\langle x, k\rangle) \end{array} \right) \right) \end{array} \right) \tag{5-3}$$

For the second condition, the detailed requirements are the following:

$$R_{\langle e_1,e_2,n,1\rangle}: \quad \exists k \exists s \exists x \text{ such that } \left(\begin{array}{l} \phi_{e_1}(k)\uparrow \\ \text{or} \left(\begin{array}{l} \phi_{e_1,s}(k)\downarrow \text{ and } \left(\begin{array}{l} \Psi_{e_2}(\langle x,k\rangle)\not\downarrow \text{ in } \phi_{e_1}(k)|x|^n \text{ steps} \\ \text{or } \mathcal{C}^{(\leq \phi_{e_1}(k))}(\Psi_{e_2}(\langle x,k\rangle)) \neq \mathcal{B}(\langle x,k\rangle) \end{array} \right) \end{array} \right) \end{array} \right) \tag{5-4}$$

Similar as what we done in the proof of original Ladner Theorem in classical complexity theory, when construing $\mathcal{C}$ *stage by stage*, we will imitate $\mathcal{B}$ to make $\mathcal{C}$ 'harder' than $\mathcal{A}$, thus satisfying requirements $R_{\langle e_1,e_2,n,0\rangle}$, while setting 0's in $\mathcal{C}$ just like "blowing holes" on $\mathcal{B}$ to make $\mathcal{C}$ 'easier' than $\mathcal{B}$, thus satisfying requirements $R_{\langle e_1,e_2,n,1\rangle}$. However, we have two dimensions needed to treat differently in the construction of $\mathcal{C}$, say the x dimension and the k dimension, and we don't know whether $\phi_{e_1}(k)$ converges when constructing thus cannot enumerate all and only $\mathbf{FPT}$ m-reductions, which brings lots of difficulties to this proof.

The resulting proof combines the "blowing holes" technique used in the original proof and the finite injury priority method to overcome such difficulties. Roughly speaking, we will conduct construction like "blowing holes" mainly in the x dimension, while using finite injury priority method in the k dimension.

More precisely, at each stage s we will set elements $\langle x,k\rangle$ of set $\mathcal{C}$ with $|x| = s$ as in the original proof. We have a check list of requirements, with each requirement $R_{\langle e_1,e_2,n,i\rangle}$ asserting control of several consecutive rows of $\mathcal{C}$ in the k dimension between $r'(\langle e_1,e_2,n,i\rangle)$ exclusively and $r(\langle e_1,e_2,n,i\rangle)$ inclusively, according to their priorities, where $r'(\langle e_1,e_2,n,i\rangle)$ and $r(\langle e_1,e_2,n,i\rangle)$ will be maintained at each stage for each requirement $R_{\langle e_1,e_2,n,i\rangle}$. Then $\langle x,k\rangle$ is set to 0 or some element of $\mathcal{B}$ at this stage according to the requirement under which it is controlled.

$\phi_{e_1}(k)$ should be checked to converge before we do further checking of that requirement. Hence for each requirement, a counter $k(\langle e_1,e_2,n,i\rangle)$ will be maintained. Whenever $\phi_{e_1}(k(\langle e_1,e_2,n,i\rangle))$ is tested to be convergent in s steps at each stage s, this counter will be increased by one, testing convergence of ϕ_{e_1} for the next k. All the

requirements with total ϕ_{e_1} can pass this kind of checking. Although it doesn't hold conversely, it has no influence on our proof as we will just satisfy more requirements.

It is a natural idea that each requirement should control some rows of $\mathcal{C}$ in the k dimension in an attempt to make itself satisfied. $r(\langle e_1, e_2, n, i\rangle)$, the so-called restraint function, is used by each requirement, denoting the rows upto which it attempt to control. For requirements $R_{\langle e_1,e_2,n,1\rangle}$ with $i = 1$, we will try to make the first $\phi_{e_1}(k(\langle e_1, e_2, n, i\rangle))$ rows of $\mathcal{C}$ satisfy certain property. Hence if $\phi_{e_1}(k(\langle e_1, e_2, n, i\rangle))$ is tested to be convergent at that stage, we will set $r(\langle e_1, e_2, n, i\rangle)$ to it, attempting to assert control of those rows. For those with $i = 0$, we will try to make the first $k(\langle e_1, e_2, n, i\rangle)$ rows of $\mathcal{C}$ satisfy certain property. Hence if $\phi_{e_1}(k(\langle e_1, e_2, n, i\rangle))$ is tested to be convergent at that stage, we will set $r(\langle e_1, e_2, n, i\rangle)$ into $k(\langle e_1, e_2, n, i\rangle)$.

To make the constructed set $\mathcal{C} \in \mathbf{W}[\mathbf{1}]$, we have to add requirements into the check list slowly, using a similar condition with '$(\log s)^n \geq s$' that is used in the proof of the original Ladner Theorem. Then the running time of checking requirements can be bounded as we want, making $\mathcal{C} \leq_m^{\mathbf{FPT}} \mathcal{B}$, thus in $\mathbf{W}[\mathbf{1}]$.

In summary, in the construction of $\mathcal{C}$, at each stage s we will do the following phases one by one:

1. checking phase: for each requirement in the check list, we check whether it is satisfied, and remove it if so;

2. updating phase: test for each requirements in the check list that whether its corresponding $\phi_{e_1}(k)$ converges in s steps, and update its r value if so;

3. adding phase: add requirements to the check list slowly;

4. setting phase: set $\langle x, k\rangle$ of $\mathcal{C}$ with $|x| = s$, according to the requirement that controls them.

In this construction, without lost of generality we only consider requirements with strictly increasing ϕ_{e_1} by deleting the requirement if we get a $\phi_{e_1}(k(\langle e_1, e_2, n, i\rangle)) \leq \phi_{e_1}(k(\langle e_1, e_2, n, i\rangle) - 1)$ in the updating phase. Therefore if $\phi_{e_1}(k(\langle e_1, e_2, n, i\rangle))$ is

tested to converge, this requirement can assert control of more rows in the k dimension in both cases, if it has enough priority. This leads the most essential part of this proof.

The most important property of this construction is that a requirement cannot control infinite rows of $\mathcal{C}$, i.e. there must be a stage that after it this requirement cannot control more rows in the k dimension. Mathematical induction on $\langle e_1, e_2, n, i\rangle$ is needed to prove this, which is the most complicated and essential part of the proof. Intuitively speaking, if a requirement can control infinite rows of $\mathcal{C}$, then it never fail the checking phase and ϕ_{e_1} is total. There are two different cases as follow.

If we are in such a requirement that we are imitating rows of $\mathcal{B}$ in constructing $\mathcal{C}$, then we have an $\mathbf{FPT}$ m-reduction from $\mathcal{C}$ to $\mathcal{A}$ by the checking phase. If we take special attention in imitating rows of $\mathcal{B}$ so that we shift all the rows of $\mathcal{B}$ down into the region of rows of $\mathcal{C}$ that this requirement has enough priority to control, then we can deduce an $\mathbf{FPT}$ m-reduction from $\mathcal{B}$ to $\mathcal{A}$. Therefore it lead to contradiction with the assumption that $\mathbf{FPT} \neq \mathbf{W}[\mathbf{1}]$.

Otherwise we are in such a requirement that we are setting 0's in constructing $\mathcal{C}$, then we have an $\mathbf{FPT}$ m-reduction from $\mathcal{B}$ to $\mathcal{C}$ by the checking phase. For the rows of $\mathcal{C}$ it controls, we can easily get an $\mathbf{FPT}$ algorithm for calculating $\mathcal{C}$. By induction hypothesis we can have a stage such that from then on all the requirements before this one don't change the rows their control. Therefore for the rows it doesn't control, we may have a requirement of $R_{\langle e_1,e_2,n,0\rangle}$'s kind with higher priority so that it has the largest value of restraint function eventually and permanently after that stage. Then by checking phase of that requirement, we have an $\mathbf{FPT}$ m-reduction from $\mathcal{C}$ to $\mathcal{A}$ for the rows upto its restraint function's value. For the rows between them, it must be all 0's in $\mathcal{C}$ except for certain elements at the beginning. Hence by combining this three parts, we can get an $\mathbf{FPT}$ m-reduction from $\mathcal{B}$ to $\mathcal{A}$. Then it leads to the same contradiction again.

In the following section, I will present the precise proof, which is a formal and rigorous version of this sketch.

5.3 The Proof

5.3.1 Construction of $\mathcal{C}$

In the construction we will maintain a number u for slowly adding requirements, a set $\mathcal{ACT}$ recording all the requirements needed to check. For each requirement $R_{\langle e_1,e_2,n,i\rangle}$, we will maintained a counter $k(\langle e_1,e_2,n,i\rangle)$ and its restraint function's value $r(\langle e_1,e_2,n,i\rangle)$. We also maintain a value $r'(\langle e_1,e_2,n,i\rangle)$ for each requirement for convenience, calculated by the $r(\langle e_1,e_2,n,i\rangle)$'s of requirements with higher priority than it. Values of $r(\langle e_1,e_2,n,i\rangle)$ and $r'(\langle e_1,e_2,n,i\rangle)$ are in $\mathbb{N}\cup\{-1\}$.

Stage 0**:** Set $\mathcal{ACT}\leftarrow\varnothing$, $u\leftarrow 0$.

Stage s**:** Execute the following commands phase by phase:

1. *(Checking phase)*$\forall\langle e_1,e_2,n,i\rangle\in\mathcal{ACT}$,

   - *If* $i=0$*:* Enumerate all $\langle x',k'\rangle$ such that $|x'|<\log s$ and $k'<k(\langle e_1,e_2,n,i\rangle)$, check whether the following condition ever holds:

   $$\text{or}\begin{array}{c}\Psi_{e_2}(\langle x',k'\rangle)\not\downarrow \text{ in } \phi_{e_1}(k')|x'|^n \text{ steps}\\ \mathcal{A}_s^{(\leq\phi_{e_1}(k'))}(\Psi_{e_2}(\langle x',k'\rangle))\neq\mathcal{C}_s(\langle x',k'\rangle)\end{array}. \tag{5-5}$$

   If so, set $\mathcal{ACT}\leftarrow\mathcal{ACT}\setminus\langle e_1,e_2,n,i\rangle$, and we say requirement $R_{\langle e_1,e_2,n,i\rangle}$ *is removed as finitely satisfied*.

   - *If* $i=1$*:* Enumerate all $\langle x',k'\rangle$ such that $|x'|<\log s$ and $k'<k(\langle e_1,e_2,n,i\rangle)$, check whether the following condition ever holds:

   $$\begin{array}{c}\Psi_{e_2}(\langle x',k'\rangle)\not\downarrow \text{ in } \phi_{e_1}(k')|x'|^n \text{ steps}\\ \text{or}\left(\begin{array}{c}\text{and}\\ \end{array}\begin{array}{c}\Psi_{e_2}(\langle x',k'\rangle)=\langle x'',k''\rangle \text{ and } |x''|<\log s\\ \mathcal{C}_s^{(\leq\phi_{e_1}(k'))}(\langle x'',k''\rangle)\neq\mathcal{B}_s(\langle x',k'\rangle)\end{array}\right)\end{array}. \tag{5-6}$$

   If so, set $\mathcal{ACT}\leftarrow\mathcal{ACT}\setminus\langle e_1,e_2,n,i\rangle$, and we say requirement $R_{\langle e_1,e_2,n,i\rangle}$ *is removed as finitely satisfied*.

The $\mathcal{C}_s(\langle x', k'\rangle)$ in the this phase denotes the function obtained by either returning 0 immediately, or simulating $\mathcal{B}$ on the corresponding input for s steps, according to what element we set in the setting phase for $\mathcal{C}(\langle x', k'\rangle)$.

2. *(Updating phase)* Let

$$\mathcal{S}_{recv} = \{\langle e_1, e_2, n, i\rangle | \phi_{e_1,s}(k(\langle e_1, e_2, n, i\rangle)) \downarrow \text{ and } \langle e_1, e_2, n, i\rangle \in \mathcal{ACT}\}. \quad (5\text{-}7)$$

If $\mathcal{S}_{recv} \neq \varnothing$, then $\forall \langle e_1, e_2, n, i\rangle \in \mathcal{S}_{recv}$:

- *If* $\phi_{e_1}(k(\langle e_1, e_2, n, i\rangle)) \leq \phi_{e_1}(k(\langle e_1, e_2, n, i\rangle) - 1)$ *with* $k(\langle e_1, e_2, n, i\rangle) \geq 1$*:* Set $\mathcal{ACT} \leftarrow \mathcal{ACT} \setminus \{\langle e_1, e_2, n, i\rangle\}$, and we say requirement $R_{\langle e_1,e_2,n,i\rangle}$ *is removed as being bounded.*
- *Otherwise:*
  - *If* $i = 0$*:* Set $r(\langle e_1, e_2, n, i\rangle) \leftarrow k(\langle e_1, e_2, n, i\rangle)$ and $k(\langle e_1, e_2, n, i\rangle) \leftarrow k(\langle e_1, e_2, n, i\rangle) + 1$.
  - *If* $i = 1$*:* Set $r(\langle e_1, e_2, n, i\rangle) \leftarrow \phi_{e_1}(k(\langle e_1, e_2, n, i\rangle))$ and $k(\langle e_1, e_2, n, i\rangle) \leftarrow k(\langle e_1, e_2, n, i\rangle) + 1$.

3. *(Adding phase)*

- *If* $(\log s)^u > s$*:* do nothing.
- *Otherwise:* for $\langle e_1, e_2, n, i\rangle = u$, set $\mathcal{ACT} \leftarrow \mathcal{ACT} \cup \{\langle e_1, e_2, n, i\rangle\}$, $k(\langle e_1, e_2, n, i\rangle) \leftarrow 0$ and $r(\langle e_1, e_2, n, i\rangle) \leftarrow -1$. Then set $u \leftarrow u + 1$.

4. *(Setting phase)* $\forall \langle e_1, e_2, n, i\rangle \in \mathcal{ACT}$, let

$$\mathcal{S}_{down} = \{\langle e_1', e_2', n', i'\rangle | \langle e_1', e_2', n', i'\rangle < \langle e_1, e_2, n, i\rangle \text{ and } \langle e_1', e_2', n', i'\rangle\rangle \in \mathcal{ACT}\}, \quad (5\text{-}8)$$

set

$$r'(\langle e_1, e_2, n, i\rangle) \leftarrow \begin{cases} \max_{\langle e_1', e_2', n', i'\rangle \in \mathcal{S}_{down}} \{r(\langle e_1', e_2', n', i'\rangle)\} & \text{if } \mathcal{S}_{down} \neq \varnothing, \\ -1 & \text{otherwise} \end{cases}. \quad (5\text{-}9)$$

If $r'(\langle e_1, e_2, n, i\rangle) < r(\langle e_1, e_2, n, i\rangle)$, then $\forall \langle x', k'\rangle$ such that $|x'| = s$ and $r'(\langle e_1, e_2, n, i\rangle) + 1 \leq k' \leq r(\langle e_1, e_2, n, i\rangle)$,

- *If* $i = 0$*:* set $\mathcal{C}(\langle x', k'\rangle) \leftarrow \mathcal{B}(\langle x', k' - r'(\langle e_1, e_2, n, i\rangle) - 1\rangle)$.
- *If* $i = 1$*:* set $\mathcal{C}(\langle x', k'\rangle) \leftarrow 0$.

5.3.2 Proof of Correctness

The above construction is obviously well-defined and recursive.

For convenience of our proof, let $r_s(\langle e_1, e_2, n, i\rangle)$ denote the value of $r(\langle e_1, e_2, n, i\rangle)$ right after stage s, and similarly $r'_s(\langle e_1, e_2, n, i\rangle)$ of $r'(\langle e_1, e_2, n, i\rangle)$, $k_s(\langle e_1, e_2, n, i\rangle)$ of $k(\langle e_1, e_2, n, i\rangle)$, u_s of u. If the requirement $R_{\langle e_1, e_2, n, i\rangle}$ is removed at some stage, we will keep their values to all the following stages. Let $\mathcal{ACT}_s$ denote the content of the set $\mathcal{ACT}$ right after stage s. We say a requirement $R_{\langle e_1, e_2, n, i\rangle}$ is *never removed* iff $\exists$ stage s such that $\forall t \geq s, \langle e_1, e_2, n, i\rangle \in \mathcal{ACT}_t$.

Lemma 5.3. $\lim_{s\to\infty} u_s = \infty$*, i.e. we will add all the* $\langle e_1, e_2, n, i\rangle$*'s into* $\mathcal{ACT}$*.*

Proof. For fixed u, since s grows much faster than $\log s$ with respect to s, there will be stage t such that $(\log t)^u \leq t$. Therefore according to the adding phase, we will enumerate $\langle e_1, e_2, n, i\rangle$ with $\langle e_1, e_2, n, i\rangle = u$ into $\mathcal{ACT}$ at stage t and also increase u. Then by induction we can prove this lemma. □

Lemma 5.4. $\forall \langle e_1, e_2, n, i\rangle$*, we have the following statements hold:*

(i) If requirement $R_{\langle e_1, e_2, n, i\rangle}$ *is never removed, there exists a stage* $s'(\langle e_1, e_2, n, i\rangle)$ *such that* $\forall s \geq s'(\langle e_1, e_2, n, i\rangle)$, $r'_s(\langle e_1, e_2, n, i\rangle) = r'_{s'(\langle e_1, e_2, n, i\rangle)}(\langle e_1, e_2, n, i\rangle)$.

(ii) If requirement $R_{\langle e_1, e_2, n, i\rangle}$ *is removed as being bounded, then* ϕ_{e_1} *is not strictly increasing.*

(iii) $r_s(\langle e_1, e_2, n, i\rangle)$ *is non-decreasing with respect to* s*.*

(iv) $r_s(\langle e_1, e_2, n, i\rangle)$ *is finite, i.e. there exists a bound* $c(\langle e_1, e_2, n, i\rangle)$ *such that* $\forall s$*,* $r_s(\langle e_1, e_2, n, i\rangle) \leq c(\langle e_1, e_2, n, i\rangle)$.

(v) *There exists a stage* $s(\langle e_1, e_2, n, i\rangle) \geq s'(\langle e_1, e_2, n, i\rangle)$ *such that* $\forall t \geq s(\langle e_1, e_2, n, i\rangle)$, $r_t(\langle e_1, e_2, n, i\rangle) = r_{s(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle)$.

(vi) *If requirement* $R_{\langle e_1,e_2,n,i\rangle}$ *with* $i = 0$ *is never removed, then we have an* **FPT** *m-reduction from* $\mathcal{C}$ *to* $\mathcal{A}$ *for the first* $r_{s(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle)$ *rows, i.e.* $\forall k' \leq r_{s(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle)$, $\forall x'$, $\phi_{e_1}(k') \downarrow$, $\Psi_{e_2}(\langle x', k'\rangle) \downarrow$ *in* $\phi_{e_1}(k')|x'|^n$ *steps, and* $\mathcal{A}^{(\leq \phi_{e_1}(k'))}(\Psi_{e_2}(\langle x', k'\rangle)) = \mathcal{C}(\langle x', k'\rangle)$.

(vii) *Requirement* $R_{\langle e_1,e_2,n,i\rangle}$ *are satisfied, except for requirement with* ϕ_{e_1} *non-strictly-increasing.*

Proof. We will prove these statements by induction on $\langle e_1, e_2, n, i\rangle$.

INDUCTION: Suppose $\forall \langle e_1', e_2', n', i'\rangle < \langle e_1, e_2, n, i\rangle$, the above statements hold.

(i) By induction hypothesis, $\forall \langle e_1', e_2', n', i'\rangle < \langle e_1, e_2, n, i\rangle$, $s(\langle e_1', e_2', n', i'\rangle)$ exist. By definition of r', $s'(\langle e_1, e_2, n, i\rangle) = \max_{\langle e_1', e_2', n', i'\rangle < \langle e_1, e_2, n, i\rangle}(s(\langle e_1', e_2', n', i'\rangle))$. Since there are only finite many $\langle e_1', e_2', n', i'\rangle$'s, $s'(\langle e_1, e_2, n, i\rangle)$ exists.

(ii) It can only occur in the updating phase. Therefore there exists stage s with $\phi_{e_1}(k_s(\langle e_1, e_2, n, i\rangle)) \leq \phi_{e_1}(k_s(\langle e_1, e_2, n, i\rangle) - 1)$, hence ϕ_{e_1} is not strictly increasing.

(iii) In the construction, we only update $r(\langle e_1, e_2, n, i\rangle)$ to $\phi_{e_1}(k(\langle e_1, e_2, n, i\rangle))$ to a greater value in the updating phase for both cases.

(iv) If $R_{\langle e_1,e_2,n,i\rangle}$ is removed at stage s, then obviously take $c(\langle e_1, e_2, n, i\rangle) = r_s(\langle e_1, e_2, n, i\rangle)$ as a bound since $r_s(\langle e_1, e_2, n, i\rangle)$ is non-decreasing.

If $R_{\langle e_1,e_2,n,i\rangle}$ is never removed, we will prove it by contradiction. Assume $r_s(\langle e_1, e_2, n, i\rangle)$ has no upperbound, it must be the case that $\lim_{s\to\infty} k_s(\langle e_1, e_2, n, i\rangle) = \infty$. Hence ϕ_{e_1} is total and $\forall k$, $\phi_{e_1}(k) \geq k$. Therefore there is a function

$$h(k) = \min\{s \geq s'(\langle e_1, e_2, n, i\rangle) | r_s(\langle e_1, e_2, n, i\rangle) \geq k\}, \tag{5-10}$$

which is computable in k given $s'(\langle e_1, e_2, n, i\rangle)$, by simply running the above construction. Since $r_s(\langle e_1, e_2, n, i\rangle)$ is non-decreasing, $h(k)$ is non-decreasing.

Since $\mathcal{A}$ is non-trivial, there must be $\langle x_0, k_0\rangle$ such that $\mathcal{A}(\langle x_0, k_0\rangle) = 0$ and $\langle x_1, k_1\rangle$ such that $\mathcal{A}(\langle x_1, k_1\rangle) = 1$.

Then we have the following situations:

- *If $i = 0$:* according to the checking phase, since $\mathcal{A}$ and $\mathcal{C}$ are recursive sets, we have an **FPT** m-reduction from $\mathcal{C}$ to $\mathcal{A}$, i.e.

$$\begin{aligned}&\forall k', x', \phi_{e_1}(k')\downarrow, \Psi_{e_2}(\langle x', k'\rangle)\downarrow \text{ in } \phi_{e_1}(k')|x'|^n \text{ steps}\\ &\quad\text{and } \mathcal{A}^{(\leq\phi_{e_1}(k'))}(\Psi_{e_2}(\langle x', k'\rangle)) = \mathcal{C}(\langle x', k'\rangle).\end{aligned} \tag{5-11}$$

  According to the setting phase, since $h(k') \geq s'(\langle e_1, e_2, n, i\rangle)$ for all k', we have

$$\begin{aligned}&\forall k' > r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle), \forall x' \text{ such that } |x'| \geq h(k'),\\ &\quad\mathcal{C}(\langle x', k'\rangle) = \mathcal{B}(\langle x', k' - r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle) - 1\rangle).\end{aligned} \tag{5-12}$$

  Then following is an m-reduction from $\mathcal{B}$ to $\mathcal{A}$ on input $\langle x', k'\rangle$:

  - Calculate $k'' = k' + r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle) + 1$.
  - * *If $|x'| \geq h(k'')$:* output $\Psi_{e_2}(\langle x', k''\rangle)$.
    * *Otherwise:* directly run the original program computing $\mathcal{B}(\langle x', k'\rangle)$, then output $\langle x_0, k_0\rangle$ if the result is 0, and $\langle x_1, k_1\rangle$ if the result is 1.

  Correctness of the above m-reduction is obvious by equations 5-11 and 5-12, as we are shifting all rows of $\mathcal{B}$ down into region of rows in $\mathcal{C}$ starting from $r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle) + 1$.

  Assume the result computed is $\langle x''', k'''\rangle$. In the first case, k''' is bounded by $\phi_{e_1}(k'') = \phi_{e_1}(k' + r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle) + 1)$, which is a computable function in k' given $r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle)$. The running time of this case is bounded by $\phi_{e_1}(k' + r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle) + 1)|x'|^n$. Obviously the k''' computed by the second case is bounded. The running time of the second case is bounded by function computable in $h(k')$ thus also in k', since there

are only $O(2^{h(k')})$-many x''s falling into this case for each k'. Hence above is an $\mathbf{FPT}$ m-reduction from $\mathcal{B}$ to $\mathcal{A}$.

Therefore $\mathcal{B} \leq_m^{\mathbf{FPT}} \mathcal{A}$, thus $\mathcal{B} \in \mathbf{FPT}$, contradiction to the assumption that $\mathbf{FPT} \neq \mathbf{W}[\mathbf{1}]$.

- *If* $i = 1$*:* according to the checking phase, since $\mathcal{B}$ and $\mathcal{C}$ are recursive sets, we have an $\mathbf{FPT}$ m-reduction from $\mathcal{B}$ to $\mathcal{C}$, i.e.

$$\begin{aligned} &\forall k', x',\ \Psi_{e_2}(\langle x', k'\rangle)\downarrow \text{ in } \phi_{e_1}(k')|x'|^n \text{ steps} \\ &\quad \text{and } \mathcal{C}^{(\leq \phi_{e_1}(k'))}(\Psi_{e_2}(\langle x', k'\rangle)) = \mathcal{B}(\langle x', k'\rangle). \end{aligned} \tag{5-13}$$

According to the setting phase, since $h(k') \geq s'(\langle e_1, e_2, n, i\rangle)$ for all k', we have

$$\forall k' > r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle), \forall x' \text{ such that } |x'| \geq h(k'),\ \mathcal{C}(\langle x', k'\rangle) = 0. \tag{5-14}$$

Let

$$\langle e_1', e_2', n', i'\rangle = \arg \max_{\langle e_1', e_2', n', i'\rangle} \{r_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1', e_2', n', i'\rangle)\} \tag{5-15}$$

subject to conditions

$$\begin{cases} i' = 0 \\ \langle e_1', e_2', n', i'\rangle < \langle e_1, e_2, n, i\rangle \\ \forall t \geq s'(\langle e_1, e_2, n, i\rangle), \langle e_1', e_2', n', i'\rangle \in \mathcal{ACT}_t \end{cases}. \tag{5-16}$$

By definition and induction we have

$$\begin{aligned} &\forall \langle e_1', e_2', n', i'\rangle < \langle e_1, e_2, n, i\rangle, \forall k' > r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle), \\ &h(k') \geq s'(\langle e_1, e_2, n, i\rangle) \geq s(\langle e_1', e_2', n', i'\rangle) \geq s'(\langle e_1', e_2', n', i'\rangle), \end{aligned} \tag{5-17}$$

i.e. rows actually controlled by each $\langle e_1', e_2', n', i'\rangle$'s with priority higher than

$\langle e_1, e_2, n, i\rangle$ doesn't change any more after stage $h(r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle)+1)$.

If the $\langle e'_1, e'_2, n', i'\rangle$ in equation 5-15 exists, we have

$$\begin{aligned}&\forall k' \text{ such that } r_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e'_1, e'_2, n', i'\rangle) < k' \leq r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle),\\ &\forall x' \text{ such that } |x'| \geq h(r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle)+1),\\ &\mathcal{C}(\langle x', k'\rangle) = 0. \end{aligned} \tag{5-18}$$

Besides, by induction hypothesis we can also have an $\mathbf{FPT}$ m-reduction from $\mathcal{C}$ to $\mathcal{A}$ for the first $r_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e'_1, e'_2, n', i'\rangle)$ rows, i.e.

$$\begin{aligned}&\forall k' \leq r_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e'_1, e'_2, n', i'\rangle), \forall x',\\ &\phi_{e'_1}(k')\downarrow, \Psi_{e'_2}(\langle x', k'\rangle)\downarrow \text{ in } \phi_{e'_1}(k')|x'|^{n'} \text{ steps},\\ &\text{and } \mathcal{A}^{(\leq \phi_{e'_1}(k'))}(\Psi_{e'_2}(\langle x', k'\rangle)) = \mathcal{C}(\langle x', k'\rangle). \end{aligned} \tag{5-19}$$

Then following is an m-reduction from $\mathcal{B}$ to $\mathcal{A}$ on input $\langle x', k'\rangle$:

(a) Calculate $\langle x'', k''\rangle = \Psi_{e_2}(\langle x', k'\rangle)$.

(b)
- *If* $k'' > r_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e'_1, e'_2, n', i'\rangle)$*:*
  - *If* $(k'' \leq r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle)$ *and* $|x''| \geq h(r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle)+1))$ *or* $(k'' > r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle)$ *and* $|x''| \geq h(k''))$*:* output $\langle x_0, k_0\rangle$.
  - *Otherwise:* directly run the above construction computing $\mathcal{C}(\langle x', k'\rangle)$, then output $\langle x_0, k_0\rangle$ if the result is 0, and $\langle x_1, k_1\rangle$ if the result is 1.
- *Otherwise:* output $\Psi_{e'_2}(\langle x'', k''\rangle)$.

Correctness of the above m-reduction is obvious by equations 5-13, 5-14, 5-18 and 5-19. $\langle x', k'\rangle$ of $\mathcal{B}$ is first mapped into an element of $\mathcal{C}$, then mapped into $\mathcal{A}$ accordingly.

Assume the result is $\langle x''', k'''\rangle$. In the first case of (b), k''' is obviously

bounded. The worst running time of the first case is bounded by function computable in $h(k'')$ or $h(r'_{s'(\langle e_1,e_2,n,i\rangle)}(\langle e_1,e_2,n,i\rangle)+1)$ similarly as in the previous situation, thus bounded by function computable in k' since $k'' \leq \phi_{e_1}(k')$, regardless of x'. For the second case of (b), $\mathbf{FPT}$ m-reductions are close under composition. More precisely, the computed $k''' \leq \phi_{e'_1}(k'')$, and thus $k''' \leq \phi_{e'_1}(\phi_{e_1}(k'))$, which is a function computable in k', as $\phi_{e'_1}$ is strict increasing. The running time of computing $\Psi_{e'_2}(\Psi_{e_2}(\langle x', k'\rangle))$ is bounded by some $\phi(k')|x'|^m$. Hence above is an $\mathbf{FPT}$ m-reduction from $\mathcal{B}$ to $\mathcal{A}$.

If the $\langle e'_1, e'_2, n', i'\rangle$ in equation 5-15 doesn't exists, then it is straightforward to get an $\mathbf{FPT}$ m-reduction from $\mathcal{B}$ to $\mathcal{A}$ similarly.

Therefore $\mathcal{B} \in \mathbf{FPT}$, contradiction to the assumption that $\mathbf{FPT} \neq \mathbf{W}[\mathbf{1}]$.

(v) Trivial by the above two statements.

(vi) $k_s(\langle e_1, e_2, n, i\rangle)$ has an upperbound, thus $\phi_{e_1}(k_{s(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle))\uparrow$. Since the requirement is never removed as finitely satisfied, an $\mathbf{FPT}$ m-reduction from $\mathcal{C}$ to $\mathcal{A}$ on these rows can be obtained as it never fails the checking phase, similarly as in the proof of the first situation of statement iv. Notice that for requirement $R_{\langle e_1,e_2,n,i\rangle}$ with $i = 0$ we always have $r_s(\langle e_1, e_2, n, i\rangle) = k_s(\langle e_1, e_2, n, i\rangle) - 1$ for any stage s by updating phase, hence condition $k' < k(\langle e_1, e_2, n, i\rangle)$ is equivalent to condition $k' \leq r(\langle e_1, e_2, n, i\rangle)$ in all stages.

(vii) If requirement $R_{\langle e_1,e_2,n,i\rangle}$ is not removed as being bounded, then it can be the following situations:

- $R_{\langle e_1,e_2,n,i\rangle}$ *is never removed:* then we've found $k_{s(\langle e_1,e_2,n,i\rangle)}(\langle e_1, e_2, n, i\rangle)$ as a witness for non-totality of ϕ_{e_1}, hence $R_{\langle e_1,e_2,n,i\rangle}$ is satisfied.
- $R_{\langle e_1,e_2,n,i\rangle}$ *is removed as finitely satisfied:* then we've found witness $\langle x', k'\rangle$ for requirement $R_{\langle e_1,e_2,n,i\rangle}$.

BASICS: For $\langle e_1, e_2, n, i\rangle = 0$, $r'_s(\langle e_1, e_2, n, i\rangle) = -1$ for all stage s, and $s'(\langle e_1, e_2, n, i\rangle) = 0$. Then all proofs for remaining statements are similar as above. □

Lemma 5.5. *$\mathcal{C} \leq_m^{\mathbf{FPT}} \mathcal{B}$, hence $\mathcal{C} \in \mathbf{W}[1]$.*

Proof. Obviously $\mathcal{B}$ is non-trivial, hence there must be $\langle x_0, k_0\rangle$ such that $\mathcal{B}(\langle x_0, k_0\rangle) = 0$. On input $\langle x', k'\rangle$, run the construction described above to stage $|x'|$, output $\langle x', k' - r'(\langle e_1, e_2, n, i\rangle) - 1\rangle$ if $i = 0$ and $\langle x_0, k_0\rangle$ if $i = 1$. Then it is a m-reduction with output parameter bounded by computable function $\phi(k) = k$.

Observe that in the setting phase, there is no need to set all the value of $\mathcal{C}$ before stage $|x'|$. We can store the current k, r and r' values for $\langle e_1, e_2, n, i\rangle$ currently in $\mathcal{ACT}$, then reproduce values of $\mathcal{C}$ in the checking phase on demand. Hence the setting phase can be done in polynomial time. In the checking phase, at stage s for each $\langle e_1, e_2, n, i\rangle$, we only enumerate $2^{\log s} = s$ many x''s and computably many k''s, and for each x', k', we can simulate the Turing Machine in time $(\log s)^n \leq (\log s)^{u_s} \leq s$ (since we're adding $\langle e_1, e_2, n, i\rangle$'s very slowly, according to the adding phase we have $(\log s)^{u_s} \leq s$ asymptotically) times function computable in k', with logarithmic overhead. Besides, we only simulate programs for $\mathcal{A}$ and $\mathcal{B}$ for s steps. Therefore the running time of the checking phase is bounded in some $\phi(k')|x'|^m$. Obviously, the remaining phases can be done in polynomial time. Therefore the whole reduction can be done in time bounded by some $\phi(k')|x'|^m$, i.e. it is an $\mathbf{FPT}$ m-reduction from $\mathcal{C}$ to $\mathcal{B}$. □

Following is the proof of Theorem 5.1.

Proof. It is obvious that $\mathbf{FPT}$ m-reduction with restriction on strictly increasing ϕ_{e_1}'s are equivalent with the original one, in the sense that if a problem $\mathcal{A}$ can be reduced to a problem $\mathcal{B}$ using the original $\mathbf{FPT}$ m-reduction, then it can also be reduced by some $\mathbf{FPT}$ m-reduction with this restriction. Therefore $\mathcal{C} \not\leq_m^{\mathbf{FPT}} \mathcal{A}$, $\mathcal{B} \not\leq_m^{\mathbf{FPT}} \mathcal{C}$, $\mathcal{C} \leq_m^{\mathbf{FPT}} \mathcal{B}$, i.e. $\mathcal{C}$ is neither $\mathbf{FPT}$ nor $\mathbf{W}[1]$-Complete, while $\mathcal{C} \in \mathbf{W}[1]$. □

In the above proof, we don't use the fact that $\mathcal{B}$ is $\mathbf{W}[1]$-Complete, except for introducing contradiction to the assumption $\mathbf{FPT} \neq \mathbf{W}[1]$ and getting the final result. However, for $\mathbf{FPT}$ m-reduction part of Theorem 5.2, proving $\mathcal{B} \leq_m^{\mathbf{FPT}} \mathcal{A}$ as a contradiction is adequate. Besides, now we need to ensure that the constructed set $\mathcal{C}$ satisfies $\mathcal{A} \leq_m^{\mathbf{FPT}} \mathcal{C}$. We can easily do it by letting the desired set $\mathcal{C}$ be $\mathcal{C}' \oplus \mathcal{A}$, where $\mathcal{C}'$ is constructed by a similar method with the previous proof, with slight modifications that now

requirements $R_{\langle e_1,e_2,n,1\rangle}$ correspond to $\mathbf{FPT}$ m-reductions from $\mathcal{B}$ to $\mathcal{C}' \oplus \mathcal{A}$ and that the $\mathbf{FPT}$ m-reduction from $\mathcal{B}$ to $\mathcal{A}$ made for contradiction in situations that a requirement of this kind controls infinite rows will now have two cases according to where the input is mapped to by the $\mathbf{FPT}$ m-reduction from $\mathcal{B}$ to $\mathcal{C}' \oplus \mathcal{A}$, which can be solved easily by definition of the '$\oplus$' operator. Therefore it is straightforward to see that the above proof can also be used to prove Theorem 5.2, especially for $\mathbf{FPT}$ m-reduction.

For $\mathbf{FPT}$ Turing-reduction part of Theorem 5.2, slight but easy modifications are needed. All the requirements and reductions we obtained in the proof for contradiction should be changed into Turing-reduction's styles. Besides, we will restrict $\mathcal{A}_s^{\mathcal{O}}$ to run upto s steps including steps taken by the oracle machine. Observe that there is no need to restrict the set $\mathcal{A}$ to be non-trivial, since we can directly output a $\{0,1\}$-value in Turing-reductions. Then similarly we can get this result.

Chapter 6 Conclusion

Analogue of Ladner Theorem in parameterized complexity theory is claimed to be proved by Downey and Fellows as a tiny part of their article *"Fixed-parameter tractability and completeness III: Some structural aspects of the W hierarchy"*([9], Downey and Fellows, 1993: 205.). However, their proof is rather brief with a few details missing, which makes it less accessible to people not well-versed in recursion theory. I have carefully read the proof of them, justifying what they have proven, extracting their main ideas, understanding their ambiguous part clear. After considering all the details involved in this proof, I can confirm that their proof is totally correct, and their insight demonstrated in this proof is fabulous.

In this thesis, I prove the original Ladner Theorem in classical complexity theory, and introduce finite injury priority method by means of proving two interesting theorems in recursion theory. Most importantly, I present a rigorous proof of the parameterized version of Ladner Theorem, using finite injury priority method and technique from the "blowing holes" proof of the original Ladner Theorem. Besides, the proof I present in this thesis is complete in such a way that a reader with elementary knowledge in complexity theory would be able to understand all the details.

REFERENCE

[1] CUTLAND N. Computability: An introduction to recursive function theory[M].[S.l.]: Cambridge university press, 1980.

[2] HOMER S, SELMAN A L. Computability and complexity theory[M].[S.l.]: Springer, 2011.

[3] SOARE R I. Recursively enumerable sets and degrees: A study of computable functions and computably generated sets[M].[S.l.]: Springer, 1987.

[4] PAPADIMITRIOU C H. Computational complexity[M].[S.l.]: John Wiley and Sons Ltd., 2003.

[5] ARORA S, BARAK B. Computational complexity: a modern approach[M].[S.l.]: Cambridge University Press, 2009.

[6] FLUM J, GROHE M. Parameterized complexity theory[M], Vol. 3.[S.l.]: Springer, 2006.

[7] DOWNEY R G, FELLOWS M R. Parameterized complexity[M], Vol. 3.[S.l.]: Springer, 1999.

[8] LADNER R E. On the structure of polynomial time reducibility[J]. Journal of the ACM (JACM), 1975, 22(1):155–171.

[9] DOWNEY R, FELLOWS M R. Fixed-parameter tractability and completeness III: some structural aspects of the W hierarchy[C]//Complexity theory. .[S.l.]: [s.n.] , 1993:191–225.

[10] SCHÖNING U. A uniform approach to obtain diagonal sets in complexity classes[J]. Theoretical Computer Science, 1982, 18(1):95–103.

[11] DOWNEY R G, FELLOWS M R. Fundamentals of Parameterized Complexity[M].[S.l.]: Springer, 2013.

[12] POST E L, et al. Recursively enumerable sets of positive integers and their decision problems[J]. Bulletin of the American Mathematical Society, 1944, 50(5):284–316.

[13] CHEN Y, FLUM J, GROHE M. Machine-based methods in parameterized complexity theory[J]. Theoretical Computer Science, 2005, 339(2):167–199.

Acknowledgements

I am heartily thankful to my thesis advisor, Prof. Yijia Chen, who gave me guidance, suggestion and motivation on topics in parameterized complexity theory.

I would like to acknowledge all the teachers and students in BASICS lab for discussing problems and exchanging ideas.

I'm grateful to Xiao Jia for his generously sharing his LaTeX template for me to write this undergraduate thesis.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of the project.