EXTENSION OF THE IDEA OF ANOTHER SOLUTION PROBLEMS AND THEIR COMPLEXITY ANALYSIS SCHEME OUTSIDE FNP

別解問題の計算量解析手法の FNP 以外への拡張

by

SETA Takahiro

瀬田 剛広

A Master Thesis

修士論文

Submitted to

the Graduate School of the University of Tokyo

on February 1st, 2005

in Partial Fulfillment of the Requirements

for the Degree of Master of Information Science and Technology

in Computer Science

Thesis Supervisor: IMAI Hiroshi　今井 浩

Professor of Information Science

ABSTRACT

Another Solution Problem (ASP) is the class of problems of finding another solution than given known solutions to a given problem instance.

One of the main interests with ASP is whether ASP can be easily solved by using the known solutions, that has relation to the difficulty of checking the correctness of a puzzle problem with an intended solution, for the necessity of the uniqueness of the solution. In FNP, many complete puzzles have been proved to have their ASPs also complete, indeed.

The main technique to analyze the complexity of ASP is ASP-reduction, rather than Karp-reduction, and the completeness with it is called ASP-completeness. This completeness, or the reduction itself, is another interest related to ASP, mainly from the theoretical aspect. Many functional complete puzzles have been proved to be ASP-complete in FNP, too.

However, the ASP is mainly studied with problems in NP or FNP so far, though the definition looks independent from any complexity classes, and there also are problems out of NP, the ASPs of which have some importance like Yozume problems. For this situation, the extension of ASP is studied in this thesis, and the following results are obtained.

First, it is shown that the scheme and techniques used for the complexity analysis of ASP in NP are applicable to other classes, in spite of the worry of the effect of exponential explosion of solution lengths. And, the result is applied to problems in P, PSPACE and EXP. For puzzles, Sokoban is proved to have its ASP as a PSPACE-complete problem.

Second, necessary conditions for function problems to be ASP-complete in the corresponding complexity class are given, and it is checked what kind of problems various function problem classes corresponding to P, NP, PSPACE, EXP and NEXP have as ASP-complete problems. For FP, which corresponds to P, and for NPMV, which corresponds to NP, it is proved that the standard functional complete problems are not ASP-complete. This shows that ASP-reduction divides classes differently to the standardly used functional reduction. For PSPACE, EXP and NEXP, corresponding function class is defined and some ASP-complete problem in those classes are also defined, The ASP-complete problems contain a problem which naturally represent interesting problems in real application, and also contain Sokoban, which shows the probability of the application to analysis of puzzle problems.

Last, as a by-product, an open problem asking the equivalence of ASP-completeness in FNP and NP-completeness of the decision version of k-ASP are solved negatively with an assumption of $\mathrm{P} \neq \oplus\mathrm{P}$, by constructing a counter-example.

論文要旨

別解問題（Another Solution Problems, ASP）は、問題文とその既知の解とが与えられたとき、既知の解以外の解を探す問題のクラスである。

別解問題に関する大きな興味の 1 つは、それが与えられた解の情報を使うことで簡単に解けるのか、ということである。これは、作成された想定解つきのパズルの問題が、解の一意性を要求に対し、正しいかを検証する難しさと関連があることが指摘されている。実際、FNP においては、多くの関数完全なパズルについて、その別解問題もまた関数完全になることが示されている。

別解問題の計算量を解析するのには、Karp 還元ではなく、ASP 還元が使われる。そして、ASP 還元に関する完全性のことは ASP 完全性と呼ばれる。この ASP 完全性、または ASP 還元そのものについて調べるのが ASP に関する興味の、特に理論的なものの 1 つである。FNP においては多くの関数完全なパズルについて、それらは ASP 完全になることが示されている。

しかし、その定義が計算量クラスと関係無いように見えること、そして、NP の外にも余詰問題等、ASP が意味を持つ問題があることにも関わらず、これまで別解問題は主に NP または FNP においてのみ研究されている。そこで、本論文では ASP 及びその計算量解析手法の拡張が研究された。主な結果は次のとおりである。

まず、NP において利用された計算量解析の枠組みと手法は、解の長さの指数爆発等の影響等が懸念されたが、NP 以外のクラスでも利用可能であることが示された。そして、P、PSPACE、EXP に対してその手法が実際に適用された。パズルへの応用としては、Sokoban に対し、別解問題の PSPACE 完全性が示された。

次に、関数問題が ASP 完全になるための必要条件が与えられ、P、NP、PSPACE、EXP 及び NEXP に対応する、いくつかの関数問題クラスについて、そのクラスの ASP 完全問題がどのようになるかが調べられた。P に対する関数問題クラス FP、及び、NP に対する NPMV に対しては標準的な完全問題は ASP 完全にならないことが示された。これは ASP 還元が関数還元とは違ったクラスの分け方をすることを表す。PSPACE、EXP、NEXP については、対応する多価関数問題クラスを定義し、そのクラスにおける ASP 完全問題が与えられた。その完全問題は実用上興味深い問題を自然に表現できるもの、及び、Sokoban を含むことから、パズルに対する応用面でも有用と予想される。

最後に、副産物として、FNP における ASP 完全性と k-ASP の判定版の NP 完全性の同一性に関する予想に対し、$\mathrm{P} \neq \oplus\mathrm{P}$ を仮定することで反例を与え、否定的な結論を示した。

Acknowledgements

The author must and wants to thank Mr. Ito and Mr. Yato, for giving helpful and important comments to him, showing interesting topics to him and other things. The author also thanks other members of Imai laboratory and related people, for holding interesting seminars and talks.

The author wants to thank his parents and his brother, for supporting him mentally and economically.

Contents

Chapter 1

Introduction

Another Solution Problem, or ASP for short, is the class of the problems which ask to find an unknown solution for a given instance and known solutions for it, or to determine if there exist such solutions, which is introduced as a class of problems by Ueda and Nagao [28].

ASP naturally appears in various situations, or many problems can be regarded as ASP. As a trivial example, primality check can be regarded as ASP, because an integer is prime if and only if the integer does not have another factor than 1 and itself.

There are not only trivial examples of ASP. The problem of finding a string which mapped by a hash or digest function to a given string is one of the important problems for real use. The ASP of this problem becomes the problem of finding a collision example with the original string given. In order to use hash functions for signatures, we need the hardness of the above two problems, not only the original one but also the ASP. This can be regarded as one important example ASP plays a very important role. In this example, to consider the hardness of the ASP, we must need special treatment for ASP. If a hash functions are designed to make it difficult to find an inverse, there exist possibilities that we can easily generate another inverse from the known one.

Optimization can be considered to relate to ASP, too. In optimization problems, it is a frequently appearing situation that one feasible solution, possibly not so good one, is found easily, and then one can search for better solutions by modifying solutions starting the firstly found one. This can be regarded as ASP, which requires us to find another solution which has a better or equal score to the known ones. Some heuristic algorithms like Hill-Climbing are these type of algorithms. The assumption used here is that solutions locates concentrated. Some problems may have that property, and there is possibility to solve ASP easier with some local search method.

There also exist problems, the ASP of which are considered long. The ASP for Hamilton Cycle problem for cubic graphs is one of the most impressive ASP, known as Another, or Second, Hamiltonian Cycle. It is known the fact that there always exists another solution for any instance with one solution (Smith's Theorem, see [18], [25] or [26]), that means the answer of the ASP for it is always"YES" as opposed to the NP-completeness of the original problem. This example shows that there exist problems, the computational complexity of the ASP of which becomes lower, and becomes a part of the motivation to study the complexity of ASP. Note: from another point of view, the Another Hamilton Cycle problem is also interesting because polynomial-time algorithm to find such cycle is not known, though the existence is assured with no computation, see [27].

As seen above, complexity analysis of ASP has important meaning in some case, and in [28], the relationship between ASP and puzzle design is also pointed out, that is, ASP appears in designing instances of puzzles, for the requirement of uniqueness of solutions.

Much more important contribution of [28] is that a general technique to analyse complexity of ASP is introduced in it. The technique is parsimonious reductions which also have the polynomial-time reducibility between solutions, which we call as ASP-reduction in this thesis. If we have an ASP-reduction of problem Π_1 to problem Π_2, and the ASP of Π_1 is NP-hard, we can conclude the ASP of Π_2 is NP-hard. We do not need to construct another reduction to prove the NP-hardness of the ASP of Π_2.

The properties or usefulness of ASP-reduction is studied more in [32]. k-ASP, the problem of finding another solution than given k solutions, is introduced in it, and it is also shown that ASP-reduction is used to prove the NP-hardness of k-ASP. Just having ASP-reduction from Π_1 to the 1-ASP of Π_1 and the NP-hardness of Π_1 is enough to prove the NP-hardness of k-ASP of Π_1, and having an ASP-reduction from Π_1 to Π_2 additionally is enough for proving the NP-hardness of k-ASP of Π_1.

In [32], the idea of ASP-completeness is also introduced, and the ASP-completeness of some standard FNP-complete function problems like FSAT is proved, and it is also proved that, in FNP, ASP-completeness induces the NP-completeness of k-ASP. As applications to analysis of complexity of puzzles, the ASP-completeness of SLITHER LINK, CROSS SUM, and NUMBER PLACE are proved in [32], too.

The meaning of [28] and [32] is that they constructed a general scheme to analyse the complexity of ASP. However, it seems to be shown to work generally just in class NP or class FNP, so far. On the other hand, as seen above, ASP appears generally in any classes or independently of any complexity classes.

There seems to be some reasons that we did not have studies of ASP in classes other than NP or FNP. One may be the absence of motivations. One reason ASP is considered in NP or FNP is that there exists puzzles with a requirement of uniqueness of solutions, which may not appear in other classes. However, there do exist puzzle or game like problems, the ASP of which is important.

For example, Tsume-Shogi, which is a kind of puzzle on the problem Shogi, there is a requirement of uniqueness of solutions, which is Yozume inhibition requirement. And Shogi and Tsume-Shogi themselves are interesting problem to consider the complexities of them (the EXP-completeness of them is proved, indeed [2, 34]), it is natural to be interested in the complexity analysis of ASP of Shogi, or Yozume problem of Shogi. (The complexity analysis of this problem is done in [33], using the result of this thesis partly.) There also be same Yozume problem for Tsume-Go, and the complexity of original Go problem is known to be EXP-complete [19], and for Chess Problem, the original problem of which is EXP-complete, too [10].

As another example, puzzle problems in PSPACE, there also seems to be the importance of finding another solutions. Many motion planning type puzzles like Sokoban, or Rush Hour, are proved to be PSPCE-complete. And in [5], the Sliding Block Puzzle, which is a kind of motion planning type puzzle, is generated automatically, and as a criterion of generation, the number of solutions, or paths to reach the final state, is used. Fewer solutions mean the hardness of the problem and it is desirable. Then, it is natural to consider the ASP of motion planning puzzles.

Another reason that we did not have studies of ASP in classes other than NP or FNP may be that for PSPACE or higher complexity classes, the solutions are potentially being exponential length. If the known solutions are exponentially long, then the input length for the ASP becomes exponentially long to the original problem. Thus, if the ASP is solved in polynomial time to the input, it takes exponential time to the original problem length, and we can not conclude that the ASP is easily solved compared to the original problem. This means that the analysis technique used for NP or FNP may not work. However, as seen after in this thesis, it is possible to use the same technique to analyse some problems in higher classes. And it is expected that we can understand ASP-reduction more deeply than considering it just in NP and FNP by trying to extend the analysis techniques to other classes. Indeed, as seen after, there are much more difficulty when considering ASP-completeness, not only when extending to higher classes but also when extending lower classes.

So, in this thesis, we consider the extension of ASP and the analysis technique of it inside and outside NP or FNP, having the target of knowing much about ASP and

complexity analysis technique.

1.1 Organization of This Thesis

In the second chapter, some definitions of ideas and terminology, and checks on some properties of the defined ideas are done. Various kinds of definitions of classes of function problems and reductions between function problems consist the main part. A review of the previous result especially related to analysing ASP in NP or FNP is also done, which is a short review of [28] and [32].

The third chapter is the main chapter of this thesis. In the first part of the chapter, the possibility of applying the techniques of [28] and [32] to problems in P, PSPACE and EXP is discussed, and it is proved that some problems in those classes have the k-ASP as complete problems defining solutions in natural way. In the second part of the third chapter, ASP-completeness is mainly discussed. First, some necessary conditions for problems to be ASP-complete are stated. And then, it is discussed whether various function classes have the standard functional-complete problems as an ASP-complete problems. The target classes are FP, NPMV and function version class of PSPACE, EXP and NEXP. For FP and NPMV, it is proved that standard problems are not ASP-complete in those classes. For the other classes, corresponding function classes are defined and some examples of ASP-complete problems in each class are shown. Last of the part, the applicability of ASP-completeness to the complexity analysis of ASP, or usefulness of ASP-completeness, further extendibility of ASP-completeness, and open problems are discussed.

The fourth chapter is chapter of topics which is related to ASP but not related to extension of ASP and its analysis, and it consists of three topics. In the first part, a solution to the open problem raised in [32] is given. In the second part, an idea of defining a function class corresponding to P is discussed. In the third part, a result on re-defining ASP-reduction stronger way is shown.

The fifth chapter is concluding chapter. We summarize the results and discuss on what is understood about ASP and its analysis technique.

1.2 Related Regions or Works

Here we mention on the regions or works related to this thesis.

The most related works to this thesis is of course those on ASP and ASP-reductions, like [28], [32] and [33]. (The last one is a work which is done in parallel with this work.)

One of the other strong related regions is complexity analysis of puzzles and games, like [8], [34] and so on. Most of those puzzles need uniqueness of solutions as stated above, they relate to ASP. Of course, whole complexity theory is important for this work, the results and the techniques used in previous works play important role in this thesis.

And, we also need to mention on the relation between the studies on ♯P-completeness like [30], Since the ASP-reduction is a stronger reduction than parsimonious reduction, ASP-completeness induces completeness of corresponding counting problems. And, parsimonious reductions used for proving ♯P-completeness also works as an ASP-reduction in most case. For counting problems, there are studies which extend the idea of counting problems to PSPACE [21] or LOGSPACE [4], though the target of this work is different to those studies, there also are relationships between ASP-completeness and completeness of counting problem. Indeed, a problem used in this work appears in [21].

We also mention to the relationship to studies on enumeration like [15] and [11]. The complexity analysis of ASP can be regarded as complexity analysis of enumeration problems. The difference is that most studies on enumeration targets to prove the easiness of enumeration or to show an algorithm for enumeration, for problems solvable in polynomial time, and, on the other hand, studies of ASP targets to prove the hardness of enumeration.

Chapter 2

Preliminaries

2.1 Definitions

First, we begin with definitions and see some easy propositions.

The next two definitions are the basic definitions to define where we do discussions.

Definition 2.1 (Function problem and decision problems). A **function problem**, or a **search problem** Π is a problem defined by a binary relation $R(x, y)$. For a given input x, the problem requests to find some y that satisfies the relation R. The input x is called instance or problem instance when we need to make clear that we are considering is the individual problem $x \in \Pi$ not the problem Π as a set of many inputs. The output y is called solution.

In contrast, a **decision problem** Π' is a problem defined by a (unary) predicate $P(x)$. For a given instance x, the problem requests to answer whether $P(x)$ holds or not.

Definition 2.2 (Class of problems). A **class** $\mathcal{C}$ of problems is a set, or family, of problems, defined for both function and decision problems. A class consists of function or decision problems is called function class or decision class respectively.

Definition 2.3 (k-ASP). Let Π be a function problem. Then the **k-ASP** of Π, for integer $k \geq 0$, is defined to be the following function problem:

Input: An instance x of Π and k of its solutions.

Solutions: The solutions of x as an instance of Π, which is different from any of the k solutions in the input.

Here, 0-ASP represents just the original problem. We sometimes call 1-ASP just as ASP, and we also call general k-ASP as ASP. And in addition, we sometimes treat ASP as an operation to function problems, which maps original problems to the 1-ASP.

The next two definitions and one proposition is about operations between problems or classes, which will be used to make proofs simpler in the following chapters.

Definition 2.4. The **decision version** of a function problem defined by a binary relation $R(x, y)$ is the problem defined by the predicate $P(x)$ which holds if and only if there exists some y that satisfies $R(x, y)$. We say a function problem Π_f is a **function version** of a decision problem Π_d, if and only if the decision version of Π_f is Π_d. The **trivial functionalization** of a decision problem defined by a predicate $P(x)$ is the problem defined by the relation $R(x, y)$ which holds if and only if $P(x)$ holds and $y =$ yes. The decision version of a function class $\mathcal{C}_f$ is defined by a class which consists of the all decision version problems of the function problems in $\mathcal{C}_f$, and the function versions of a decision class $\mathcal{C}_d$ are classes, the decision version of which coincides with $\mathcal{C}_d$.

Proposition 2.5. The trivial functionalization of a decision problem Π is one of the function versions of Π.

Definition 2.6 (Operations on function problems). Let Π_1 and Π_2 be problems, and denote the solution set of each instances p_1 and p_2 of them by s_1 and s_2 respectively.

Then, we define new problems $\Pi_1 \oplus \Pi_2$ and $\Pi_1 \otimes \Pi_2$ of input in form (p_1, p_2) as problems with solution set as $s_1 \oplus s_2$ (direct sum of the two solution sets) and $s_1 \otimes s_2$ (direct product of the two solution sets) respectively, and define $\Pi_1 \cup \Pi_2$ as a problem of input in form $(1, p_1)$ or $(2, p_2)$ (that is direct sum of instances), and solutions s_1 for input $(1, p_1)$ and s_2 for input $(2, p_2)$.

We also define Π_1+1 as a problem with input same as Π_1 and with solutions almost same as Π_1 but with an extra unconditional solution, that is $s_1 \oplus \{0\}$, and define $\Pi_1 \times 2$ with input same as Π_1 and with solution $s_1 \otimes \{0, 1\}$. Π_1+k or $\Pi_1 \times k$ for any k is defined in the same way.

The following several definitions are about reductions and completeness. Some of them are very popular (seen in [18, 35], for example), however, we must carefully define them, because we want to study on ASP-reduction.

Definition 2.7 (Reduction). A **reduction** f between two decision problems Π_1

and Π_2 (from Π_1 to Π_2) is set of transformations or functions between instances or solutions of Π_1 and Π_2.

If all functions in f are computable in polynomial time to their input length, we say f is poly-time reduction, if logarithmic space, log-space reduction, and omit those prefixes, poly-time or log-space, if it is not important.

Definition 2.8 (Karp-reduction). A **Karp-reduction** f is a reduction between two decision problems Π_1 and Π_2 defined by predicates P_1 and P_2, which consists of just 1 function f_{inst} and satisfies the next properties:

- $P_1(x) \Leftrightarrow P_2(f_{\text{inst}}(x))$.

Definition 2.9 (functional reduction). A **functional reduction** f is a reduction between two function problems Π_1 and Π_2 defined by relations R_1 and R_2, which consists of 2 functions f_{inst} and f_{bsol} and satisfies the next properties:

- f_{inst} is a Karp reduction from the decision version of Π_1 to the decision version of Π_2.
- $R_2(f_{\text{inst}}(x), y') \Rightarrow R_2(x, f_{\text{bsol}}(y'))$.

Definition 2.10 (parsimonious reduction). A **parsimonious reduction** f is a reduction between two function problems Π_1 and Π_2 defined by relations R_1 and R_2 which consists of 1 function f_{inst} and satisfies the next properties:

- The number of y' which satisfies $R_2(f_{\text{inst}}(x), y')$ equals to the number of y which satisfies $R_1(x, y)$.

Note: A parsimonious reduction is also a Karp-reduction between corresponding decision versions.

Definition 2.11 (Levin-reduction). A **Levin-reduction** f is a reduction between two function problems Π_1 and Π_2 defined by relations R_1 and R_2, which consists of 3 functions f_{inst}, f_{sol} and f_{bsol} and satisfies the next properties:

- The pair (f_{inst}, f_{bsol}) works as a functional reduction from Π_1 to Π_2.
- If $R_1(x, y)$ then $R_2(f_{\text{inst}}(x), f_{\text{sol}}(x, y))$.

Definition 2.12 (ASP-reduction [28, 32]). An **ASP-reduction** f is a reduction between two function problems Π_1 and Π_2 defined by relations R_1 and R_2, which consists of 2 functions f_{inst} and f_{sol} and satisfies the next properties:

- f_{sol} maps the solution set of $x \in \Pi_1$ to $f_{\mathrm{inst}}(x) \in \Pi_2$ in 1-to-1 and onto manner. ($\lambda y.\ f_{\mathrm{sol}}(x, y)$ works as a bijection between the two solution set.)

Proposition 2.13. The reducibility of the above reductions, Karp, functional, parsimonious, Levin or ASP reductions, are transitive, that is, if there exists reductions from Π_1 to Π_2 and Π_2 to Π_3, there also exists the same kind of reduction from Π_1 to Π_3.

Note: The version for the log-space reductions of the above proposition is not so obvious, the way to prove that appears in [18] for example.

Definition 2.14 (Completeness related to reductions). Let $\mathcal{C}$ be a complexity class, Π be a problem and T is a reduction type, (that is, for example, Karp for Karp-reduction or functional for functional reduction). A problem Π is called **T-complete** in $\mathcal{C}$, if the following two conditions hold:

- $\Pi \in \mathcal{C}$.
- There exists some T-reduction f from Π' to Π for any $\Pi' \in \mathcal{C}$.

If the first condition does not hold, we say Π is **T-hard** for $\mathcal{C}$.

We omit T, and say just complete in $\mathcal{C}$, if T is known from the context. For decision problems, in this thesis, we use only Karp-reduction and thus we always omit the prefix, Karp, and we also say Π is $\mathcal{C}$-complete for decision classes.

To classify problems with complexity, we need some computation paradigm, and we use Turing machines and transducers. For those definitions, see, for example, [18]. We refer Turing machines by TM (DTM for deterministic one and NTM for nondeterministic one) and Turing transducers by Trans (or DTrans, NTrans). In some proofs we also use Alternating TM (ATM) as seen in [23]. Using prefixes like poly-time DTM or poly-space DTM, we refer DTM with bounds of such time or space.

The next several definitions and propositions are problem class definitions. For decision classes like P, NP, PSPACE, NPSPACE, EXP, NEXP and APSPACE, we use the standard definitions which appears in, for example, [18, 1, 23], and omit those definitions. For function classes, like FP, FNP and NPMV, though we also use the standard definitions, since there are some equivalent definitions to those classes or different definitions, and we will define some function classes after, we do define them here. And also, we check some function class definitions related to enumeration.

Definition 2.15 (FNP). FNP is the class of function problems defined by the relation $R(x, y)$ which satisfies the following conditions:

- y is at most polynomial length to $|x|$.
- There exists a poly-time DTM M which accepts (x, y) if and only if $R(x, y)$ holds.

Definition 2.16 (NPMV [6]). NPMV is the class of function problems defined by the relation $R(x, y)$ which satisfies the following conditions:

- y is at most polynomial length to $|x|$.
- There exists a poly-time NTrans M, the set of possible outputs of which, with input x, equals to the set $\{y|R(x, y)\}$.

Definition 2.17 (FP). FP is the class of function problems defined by the relation $R(x, y)$ which satisfies the following conditions:

- R satisfies the conditions of FNP.
- There exists a poly-time DTrans M, which outputs some $y \in \{y|R(x, y)\}$ on input x if $\{y|R(x, y)\}$ is not empty, and halts with rejecting state if $\{y|R(x, y)\}$ is empty.

Remark . There are definitions of FP as a single-valued function class, sometimes denoted as PF, however, in this thesis we are interested in ASP and thus there is nothing to do with single valued function classes.

Proposition 2.18 (Equivalent Definition for NPMV). The class of function problems defined by the relation $R(x, y)$ which satisfies the following conditions equals to NPMV.

- y is at most polynomial length to $|x|$.
- There exists a poly-time NTM M which accepts (x, y) if and only if $R(x, y)$ holds.

Proposition 2.19 (Equivalent Definition for FNP). The class of function problems defined by the relation $R(x, y)$ which satisfies the following conditions equals to FNP.

- y is at most polynomial length to $|x|$.
- There exists a poly-time NTrans M which satisfies next conditions.
  - The set of possible outputs of M equals to the set of y that satisfies $R(x, y)$.

– For any nondeterministic transition of M, M outputs a different character for each other transition candidates. As an exception of this rule, if the transition goes to the accepting final state, M does not need to output any character. (Or we may say M outputs a special character which represents that the output terminates.)

Definition 2.20 (P-enumerable [29, 30]). P-enumerable [29, 30], or **polynomial total time** [15], is the class of function problems defined by the relation $R(x, y)$ which satisfies the following conditions:

- There exists a DTrans M, which outputs the whole set $\{y|R(x, y)\}$ in any order and in time $O(poly(|x|) \cdot \sharp\{y|R(x, y)\})$, where $\sharp\{y|R(x, y)\})$ represents the size of the set $\{y|R(x, y)\}$.

Definition 2.21 (Incremental Polynomial Time [15]). Incremental Polynomial Time is the class of function problems, k-ASP decision version of which is in P for all k.

Definition 2.22 (Polynomial Delay [15]). Polynomial Delay is the class of function problems defined by the relation $R(x, y)$ which satisfies the following conditions:

- There exists a DTrans M, which outputs $y \in \{y|R(x, y)\}$ one by one, in any order and in time polynomial to input size $|x|$ for each.

Last for this section, we mention to some problems used after, discussing ASP. We use some words for problems and formulas without definitions. For example, CNF (Conjunctive Normal Form), DNF (Disjunctive Normal Form), SAT, HornSAT, QBF, Halting or Bounded Halting, all of which are very popular ideas and problems (see [12], [18] or any other books) are used without deffinitions. And, we use Sokoban [8], which is proved to be PSPACE-complete, too. We also use the next two problems $\mathrm{G_1}$ and $\mathrm{G_3}$ defined in [23], which are also popularly used, but the next definition has a little loosened conditions for inputs, or a little different form to the original. The EXP-completeness is kept in this definition.

Definition 2.23 ($\mathbf{G_1}$ and $\mathbf{G_3}$). $\mathrm{G_1}$ and $\mathrm{G_3}$ are a decision problem to decide whether the first player A wins the next game named $\mathrm{G_1}$ or $\mathrm{G_3}$ respectively.

Input: Two DNF logical formulas A-LOSE $= \{A_i\}$ and B-LOSE $= \{B_i\}$, the all variables in which are those in A-VAR $= \{a_j\}$ and B-VAR $= \{b_j\}$, variables in any of A-VAR and B-VAR can be used in A-LOSE (B-LOSE).

Rules: The rules are as follows:

- Player A and player B alternately changes the assignment in corresponding variable set to each player, that is A-VAR for A and B-VAR for B. For G_1 any number of variables can be changed, and for G_3 the assignment of just one variable can be changed.
- The game ends if the losing condition of the opposite player is satisfied before the current player's turn starts. If the A-LOSE is satisfied before B changes his variable, then B wins, and if the B-LOSE is satisfied before A changes his variable, then A wins.
- The all variables are set to 0 initially and the game starts with A's turn.

For games and logical circuits, we will use the next idea of trace as a definition of solutions, which is defined in [23].

Definition 2.24 (Trace). A **trace** of a single output logical circuit is a minimal subcircuit represented by a set of connecting lines, which is necessary to determine the output value, constructed in the following recursive way. To determine the output of an AND node to *true*, we must verify that all the inputs are *true*, so we construct the trace of current AND node with output *true* with all the traces of the input nodes and lines to them. In contrast, to determine the output of an OR node to *true*, we just need to verify one of the inputs is *true*, so we construct the trace of current OR node with output *true* by choosing one input with value *true* and merging its trace with the line to the node. We define trace of nodes with output *false* in same way, one of the inputs of value *false* for AND node and all the inputs for OR node. Thenm for NOT node, the trace is consists of the trace of the input and the input line. For the inputs to the circuit, the trace is just itself.

We also define traces for $\forall$-$\exists$ trees or two-player game trees in the same way. In such case, the input node does not exist, and a $\forall$-node with no children works as an input node with value *true*, and an $\exists$-node with no children works as an input node with value *false*.

This definition has a little difference to the definition in [23], in [23], the subcircuit is not necessarily minimal, that is, for example, trace of OR node with output *true* can be constructed with traces of more than two inputs, and the subcircuit is represented by a set of nodes rather than a set of lines. The idea of solutions appeared in [3] might be nearer.

2.2 Previous Results for ASP-reduction

The following are theorems or propositions related ASP-reduction and k-ASP

Theorem 2.25 ([28, 32]). Let Π_1 and Π_2 be problems, and f be an ASP-reduction from Π_1 to Π_2, then there exists an ASP-reduction from k-ASP of Π_1 to k-ASP of Π_2 for any k, which is naturally constructed from f.

Proposition 2.26 ([32]). Let Π be a problem, then the k-ASP of l-ASP of Π becomes $(k+l)$-ASP of Π.

Theorem 2.27 ([32]). Let Π be a problem and f is an ASP-reduction from Π to 1-ASP of Π, then there also exists an ASP-reduction from Π to k-ASP of Π for any $k \geq 0$.

Theorem 2.28 ([32]). If a function problem Π is ASP-complete in FNP, then decision version of k-ASP of Π is NP-complete for any $k \geq 0$.

As a proof of Theorem 2.28, there also exists another simpler way than [32]. This shows the usefulness of ASP-completeness for the purpose of analysing the complexity of k-ASP.

Technique 2.29. Let Π be ASP-complete in FNP. Then, since class FNP is closed with $+1$ operation, and $\Pi{+}k$ is in FNP, there exists an ASP-reduction from $\Pi{+}k$ to Π and thus k-ASP of $\Pi{+}k$ to k-ASP of Π. Combining that k-ASP of $\Pi{+}k$ includes original Π and that FNP is closed with ASP operation, we get k-ASP of Π is ASP-complete, and the decision version is NP-complete.

And, in FNP, some ASP-complete problems are found.

Proposition 2.30. FSAT, F-3SAT, F-1-in-3-SAT, Slither Link, Cross Sum and Number Place are all ASP-complete in FNP.

Chapter 3

Extension of ASP and ASP-completeness

As mentioned in the introduction chapter, there is worry that the solution length may cause some troubles analysing the complexities of ASP in classes over or inside NP (FNP).

Indeed, by considering the extension of ASP or analysis technique for ASP, we find that NP (FNP) was the most suitable class to consider and analyse ASP in it. In this chapter, we will see the difficulties which appear when extending the idea of ASP, the analysis schemes and techniques for ASP, and the idea of ASP-completeness, and how we can overcome some of such difficulties.

3.1 k-ASP in Various Complexity Classes

3.1.1 General Discussions

There are some difficulties on extending the idea of ASP and analysis technique of complexities of ASP.

The main difficulty is about the length of solutions. If the length of solutions are super-polynomial to the original input length, the input for the ASP will be super-polynomial to the original input length, then even if the ASP is solved in polynomial time to the ASP input length, it does not mean that ASP can be solved easily, for it may take super-polynomial time to the original input length.

The solutions of problems in classes over NP naturally have super-polynomial length. For any PSPACE-complete motion planning problems like Sokoban, it is easily known that there must be instances which needs exponentially long moves, or it is proved $NP = PSPACE$.

We may define the length of solutions to be at most polynomial as it is done in

some definition of FPSPACE and other function classes [21], but if we do so, we might need much time or space to verify the given known solutions of ASP instance are really solutions. Therefore, if we prove the ASP decision version of such a function problem are hard for some complexity class, we may not be able to conclude that finding another solution of the problem is difficult, because the hardness may come from the hardness of verification of given solution.

It is possible to introduce some promise that the given solutions are certainly solutions like [33], but we choose not to introduce such promise, here. The problem will not be solved just by adding promise, if the verification is not necessary we may need to much time to extract informations from the given solution.

It may be possible to change the scheme of analysing the complexity of ASP to measure with just respect to original instance length. However, this will not solve the difficulty; it takes super-polynomial time to read the given known solutions, which means it takes much time to verify them or drawing information from them. This idea has the same problem as the idea of above polynomial length restriction has.

As seen above it might seem to be impossible to analyse the complexity of ASP in classes over NP like PSPACE or EXP with the same techniques as used in NP (FNP). However, it appears that we can apply the completely same technique to analyse the complexity of ASP in such classes.

First, we see that the theorems or techniques for class NP or FNP can be used in other classes, and check the requirement to apply such techniques.

Theorem 2.25 and the Theorem 2.27 are independent from any complexity classes, these holds in any complexity classes. (Of course the definition of the reduction requires some complexity like poly-time ASP-reduction, but it does not cause problems here.) However, we may be a little careful to use those theorems. There is some requirements to the solutions for existence of ASP-reduction f from Π_1 to Π_2. One of the most important requirements is that comes from the condition of f_{inst}. The length of $f_{\mathrm{inst}}(x)$ is at most polynomial to the length of x, for f must be computed in at most polynomial time to length of x. This mean we can apply the technique induced from Theorem 2.27 only when the length of the known solution of the reduced 1-ASP problem is at most polynomial to the original instance length.

The above observation may seem to tell us the same thing that is concerned above, however, being careful, it does not tell us the necessity for all solutions to be short, but the sufficiency of being short of just the given solutions to the ASP. So, we can use the same techniques as used in NP, if we manage to construct ASP-reductions. In many case, it is easy to define problems to have solutions with polynomial length to

the instance length, which is also be checked easily, and we can prove the hardness of the ASP with such given solutions as the hardness of finding another solution itself.

Note: We may also need to mention that the condition of f_{sol} plays important part. Because the length of $f_{\mathrm{sol}}(x, y)$ is at most polynomial to the length of x and y, we conclude that all given k solutions to the k-ASP generated by $f \circ f \circ \cdots \circ f$ is at most polynomial length to the original instance length, together with the observation above.

3.1.2 k-ASP for P

For P, solutions are naturally defined to be polynomial length, and this means the input for ASP of the problem, that is pair of an original problem instance and some known solutions, is at most polynomial length to the original problem. Then, no problem happens with the length of input to ASPs.

However, here is a problem. if the verification that the given solutions are certainly solutions takes polynomial time, it may be impossible to say that the P-hardness of an ASP comes from the part of finding another solution just because the ASP is proved to be P-hard. So, we must take care on the complexity of verification of solution, and with that, the idea of ASP and analysis of it is easily extended, as shown below.

As an example, we prove that Circuit Value, which is a standard P-complete problem, has its k-ASP decision version as P-complete problem with a natural definition of solutions for any k.

Definition 3.1 (F-Circuit-Value). F-Circuit-Value is a function problem defined by the following way:

> Input: Same as Circuit Value, that is a single output logical circuit and input to it.
>
> Solutions: Traces which show the output of the circuit is *true*.

Proposition 3.2. The decision version of k-ASP of F-Circuit-Value is P-complete for any $k \geq 0$

Proof. It is easy to verify that k-ASP decision version of F-Circuit-Value is in P. It is possible to enumerate all solutions of F-Circuit-Value using a variant of depth first search method with polynomial time for each solution. Furthermore, it is also possible to count the number of solutions in polynomial time. It is also easily seen that verification of solutions are done in polynomial time.

The rest of the proof, that is proof of (log-space) P-hardness, is by constructing (log-space) ASP-reduction from original problem to 1-ASP. The ASP-reduction is easily constructed by adding a trivial solution as it is done in class NP. The additional trivial solution is added by sticking a binary-input OR node to the output of original circuit, setting the other input of the OR node to *true* and defining the new output of reduced circuit to be the output of the OR node.

It is also seen the additional solution or the solution to be given to 1-ASP is easily verified in logarithmic space. □

Corollary 3.3. k-ASP of F-Circuit-Value is FP-complete for any $k \geq 0$.

Proof. The above reduction used to prove Proposition 3.2 is also be a functional reduction, and k-ASP of F-Circuit-Value is also proved to be in FP in the same way to the proof. □

As another example, we prove that Horn-SAT also has its k-ASP decision version as P-complete.

Definition 3.4 (F-Horn-SAT). F-Horn-SAT is a function problem defined by the following way:

> Input: Same as Horn-SAT, that is a CNF logical formula, the clause of which has at most one positive literal.
>
> Solutions: Assignments of variables which satisfy the input formula.

Proposition 3.5. The decision version of k-ASP of F-Horn-SAT is P-complete for any $k \geq 0$

Proof. First, we need to prove that k-ASP decision version of F-Horn-SAT is in P for any k. This is easily proved by the possibility of enumeration of solutions in polynomial time each and the possibility of log-space verification of solutions. The enumeration is done by just repeating the computation of minimum set of variables to be assigned true and addition of one clause made of a single positive literal.

The rest of this proof is by constructing (log-space) ASP-reduction from original problem to 1-ASP, as similar to other problem. This can be done completely similar way to SAT by adding a new variable x, a literal $\overline{x}$ to all original clauses and clauses of form $x \vee \overline{a}$ for all original variable as. The verification is done in log-space. □

Corollary 3.6. k-ASP of F-Horn-SAT is FP-complete for any $k \geq 0$.

Proof. The above reduction used to prove Proposition 3.5 is also a functional reduction. And k-ASP of F-Horn-SAT is also proved to be in FP in the same way to the proof. □

In the above proofs, we checked apparently the possibility of the log-space verification. However, since the reduction is done in log-space, and the answer is generated by the reduction, we can verify the solution in log-space, by applying the reduction to the original problem. Thus, it is not necessary to check the possibility of the log-space verification.

3.1.3 k-ASP for EXP

For EXP, it is known that many two-player games like Chess [10] and Shogi [2, 34] is EXP-complete, and it is also known that such games need exponential moves to end the game, or if not so it must be solved in PSPACE. So, as stated above we need to find problems or instances of them which have short solutions. This is easily found. The above problems have instances with short solutions and long solutions, and it is also found that the standard EXP-complete problem G_3 also has such instances. Here, we prove that a function version of G_3 have its ASP as EXP-complete problems.

Definition 3.7 (FG_3). FG_3 is a function version of two-player game G_3 and defined as following way:

Input: Same as G_3.

Solutions: Traces of G_3 which prove that the first player wins.

Proposition 3.8. The decision version of k-ASP of FG_3 is EXP-complete for any $k \geq 0$

Proof. It is easily checked that k-ASP of FG_3 is in EXP, just by a search method as it is done in the case of F-Circuit SAT.

The rest part of proof is by constructing an ASP-reduction from FG_3 to 1-ASP of FG_3. This is done as following way:

First, we prepare new variables $a, a' \in A$-VAR and $b, b' \in B$-VAR, all of which are assigned to 0 initially. Second, we modify all $A_i \in A$-LOSE to $A_i \wedge b'$ and add $\overline{a} \wedge \overline{a'}$, $\overline{a} \wedge b$ and $\overline{a'} \wedge b$ to A-LOSE. Last, we modify all $B_i \in B$-LOSE to $B_i \wedge a$ and add $a \wedge \overline{b}$, $a \wedge \overline{b'}$, $a' \wedge \overline{b}$

It is verified as the following way, that the above reduction is certainly an ASP-reduction:

In the first turn, which is A's turn, A must change a or a', otherwise A loses for the clause $\overline{a} \wedge \overline{a'}$ is satisfied. If A changes a first, then the two clauses $a \wedge \overline{b}$ and $a \wedge \overline{b'}$ in B-LOSE become satisfied and no clauses in A-LOSE becomes satisfied. B can not make unsatisfied both $a \wedge \overline{b}$ and $a \wedge \overline{b'}$ by changing variables in B-VAR, therefore B loses. This is one trace or solution, which is added by ASP-reduction. Next, let us consider the other possibility, the case A changes a' first. Then, just one clause $\overline{a'} \wedge b$ in B-LOSE is satisfied and the game continues $b \rightarrow a \rightarrow b'$. And then, the game becomes same to the original one. All of the new variables are assigned to 1; all of the new clauses become unsatisfied, and the effect of additional part $\wedge b'$ or $\wedge a$ is cleared from all of the other clauses. And, in addition, both A and B can not change any of the new variables because of the clauses $\overline{a} \wedge b$, $\overline{a'} \wedge b$, $a \wedge \overline{b}$ and $a \wedge \overline{b'}$. □

Though we don't write this as a corollary or a proposition, we mention that we can say k-ASP of FG_3 is as hard as FG_3, for the reduction constructed above in the proof is also a functional reduction.

The above is application of Theorem 2.27, and the almost same result is used in [33] to prove the EXP-completeness of Yozume of Generalized Tsume-Shogi, as an application of Theorem 2.25.

3.1.4 k-ASP for PSPACE

For PSPACE, QBF is one of the most standard complete problems. So, here we define function version of QBF and prove that ASP of the problem is PSPACE-complete.

Definition 3.9 (F-Non-Prenex-QBF-TRACE). F-Non-Prenex-QBF-TRACE is a function version of QBF defined as follows:

Input: Quantified Boolean Formula, not necessarily in prenex normal form, and not necessarily in DNF or CNF form. That is, the quantifiers are not necessarily located the head of the formula, but is possibly located in middle of the formula.

Solutions: Traces of the formula which prove that the answer of the QBF is *true*. Here, we define the trace as the minimal subtree of the $\forall$-$\exists$-$\wedge$-$\vee$ tree, which consists of 5 types of nodes; not only $\forall$ and $\exists$, but also $\wedge$, $\vee$ and $\neg$. The additional three nodes come from the corresponding operators in the QBF.

Note: As it is seen from the above definition. We here define the function version of QBF with QBF not restricted to prenex normal form, though generally used QBF definitions are prenex normal form. This is because the trace of prenex normal form

must assign all variables and thus the trace becomes exponentially long to the number of variables, which is not desirable. If we do not restrict QBF to be prenex, we can write formulas like $\exists x\ x \vee (\overline{x} \wedge (\forall y \cdots))$, and it is verified that this QBF holds just only assign x to *true* (and selecting the first sub-formula for the $\vee$), without seeing the contents of the parentheses.

Proposition 3.10. k-ASP of F-Non-Prenex-QBF-TRACE is PSPACE-complete for any $k \geq 0$.

Proof. It is easy to see that k-ASP of F-Non-Prenex-QBF-TRACE is in PSPACE. We can count the number in polynomial space by multiplying the number of traces of descending nodes at $\forall$-node and adding at $\exists$-node, and thus it is possible to answer k-ASP.

The rest part, the construction of ASP-reduction from F-Non-Prenex-QBF-TRACE to 1-ASP of that, is also very easy. It is just done by adding a new variable a with an existential quantifier to the top of the original formula F, like $\exists a\ a \vee (\overline{a} \vee F)$. It is obvious that this is an ASP-reduction. □

We also mention here, that the reduction used above also works as functional reduction, as well as FG_3, and this leads that the k-ASP of F-Non-Prenex-QBF-TRACE is functional-complete in FNPSPACE-TRANS(exp, poly out) as defined later.

As an application to problems which appear in real world, we can also prove that the natural functionalization of Sokoban is PSPACE-complete. (The proof here is not ASP-reduction technique though.)

Definition 3.11 (FSokoban). FSokoban is a function version of Sokoban defined as follows:

Input: Same as Sokoban. (See [8] for the definition of Sokoban.)

Solutions: Sequence of moves of boxes which satisfies the following conditions.

- The final state is reached with it.
- It satisfies the rule of Sokoban.
- It has no subsequence consists of one box, in which the first position and the last position of the box are same and the player can move without moving any boxes (we call this sequence as redundant moves).
- The length is at most 3^n, where n is the size of the input board. (3^n is chosen because there the number of configurations is bounded by it. One block of the board can be occupied by the player or a box or nothing.)

Proposition 3.12. k-ASP of FSokoban is P-complete for any $k \geq 0$.

Proof. It is understood that k-ASP decision version of FSokoban is in NPSPACE = PSPACE. We can solve it by nondeterministically move the boxes to the final state, checking whether any difference to each given solution exists and whether redundant move does not appears. The redundancy is checked in polynomial space because we just need to see the sequence of moves of only one box and therefore recording moves with length at most square of board size (position of the box and the player) is enough, and because solving a connectivity or reachability problem is enough to check the redundancy.

The rest of proof is modifying the reduction of [8] to that for k-ASP. This is done by just adding a k shortcut paths from the starting position to the gadget which represents the accept state of the emulated DTM. The shortcut path can be made by two 'OneWay' gadgets connected in series. □

3.2 ASP-completeness in Various Complexity Classes

3.2.1 General Discussions

First, before the discussions for individual classes, we begin with general discussions.

In the previous section, we constructed problems whose k-ASP becomes complete in corresponding decision class for any k. There, especially for PSPACE or EXP, we designed the function problems to have some solutions of short length. The definitions of solutions were not so unnatural, but we may also be sorry that traces are not used as solutions of standard (prenex normal from) QBF for the exponential length. For ASP-completeness, we find the next result to find a necessary condition of a problem to be ASP-complete, the statement of which is almost expectable and obvious from the discussions which already appears, but the proof is complicated.

Proposition 3.13. If function class $\mathcal{C}$ is closed with the $+1$ operation, and the decision version class of it is not included in P, then the ASP-complete problem of $\mathcal{C}$ must have instances with polynomially short length solutions.

Proof. Let $\Pi \in \mathcal{C}$ be the ASP-complete problem in $\mathcal{C}$. This assumption and the fact $\Pi + 1 \in \mathcal{C}$ for the closeness of $\mathcal{C}$ with respect to $+1$ operation leads there exists an ASP-reduction from $\Pi{+}1$ to Π. By the definition of $+1$, any instance of $\Pi{+}1$ has a trivial solution and it is polynomially short length, and we can transduce the instance and the short solution to Π's instance and solution using the ASP-reduction. The

reduction is done in polynomial time, thus the short solution must be transduced to polynomial (to the original instance length) length solution.

Just from the conditions of ASP-reduction, there also remains a possibility the transduced solution is not polynomially short because the instance length is shortened. In such case, however, we can prove the existence of instances with short solutions.

Assume the instance length is shortened and the above transduced solution is not bounded by any polynomial of the length of transduced instance. Then we can choose some length N the longer instances are transduced shorter than factor $1/2$, since a polynomial of N is also a polynomial of $N/2$. Therefore, we can reduce the ASP-complete problem instance to another instance of the problem with at most length N by applying the $+1$ operation and the reduction repeatedly for polynomial times. The constant length problems are solved in constant time and the number of solutions are known in constant time. This mean we get the number of solutions of the original problem, and thus the decision version of Π is solved in polynomial time. □

Note that the necessity of the closeness with $+1$ operation needed for the above proposition is satisfied in most classes like FP, FNP, and NPMV. For the inclusion of trivial problems, almost all class must be closed with $+1$ operation.

Another necessary condition is an almost obvious condition, which is a little weakened version of Theorem 2.28

Proposition 3.14. The ASP-complete problem of $\mathcal{C}$ must be a function version of a complete problem in the decision version class of $\mathcal{C}$.

Proof. An ASP-reduction works as a Karp-reduction between the decision versions, and the proposition follows. □

The above are the conditions for problems to be ASP-complete, and we also find desirable conditions if we design function classes corresponding to some decision class. One is about closeness of the class with ASP operation, and the other is about defining function classes with transducer. We will see those conditions in the following sections considering FP and NPMV, which are the classes that seem to be studied and recognized relatively well.

3.2.2 ASP-completeness in FP

The class FP is the most natural class to start with if we try to find an ASP-complete problem in it. FP is studied relatively well, for the relationship to FNP, and there is a well-used definition of it as seen in Chapter 2, the preliminary definition chapter.

(Of course, we use the multi-valued definition, not the single-valued definition, as we defined it in Chapter 2) However, as seen below, we will find that the ASP-complete problems in FP are not those, we must have expect, no natural problems solvable in P will be ASP-complete in FP.

Proposition 3.15. The class FP has ASP-complete problems, and the 1-ASPs of those problems are ASP-complete in FNP.

Proof. The proof is by constructing problems.

First, we prove that if there exist ASP-complete problems, the 1-ASPs of those problems must be ASP-complete in FNP.

Let us consider the problem FSAT+1. This problem is easily known to be in FP by checking the conditions in the definition:

- The additional solution is trivially calculated in P.
- The solutions of FSAT are polynomially balanced and are able to be checked in polynomial time, because FSAT is in FNP, and so is constant one.

Then, if a problem Π is ASP-complete in FP, there must exist an ASP-reduction from FSAT+1 to Π. By Theorem 2.25, the reduction works as an ASP-reduction from the 1-ASP of FSAT+1 to the 1-ASP of Π. And the 1-ASP of FSAT+1 is ASP-complete in FNP, because it is in FNP and it contains FSAT, which is ASP-complete in FNP, so 1-ASP of Π must also be ASP-complete in FNP.

Note: We may use (FSAT+1) $\otimes \Pi_{\mathrm{FP}}$, where Π_{FP} is any single valued function problem in FP, if we dislike that the decision version of FSAT+1 becomes a trivial problem, answer of which is always YES.

Next, we construct an example of ASP-complete problem in FP.
Prepare some (log-space) function-complete function problem Π_1 in FP, an deterministic polynomial time algorithm A_1 for it, which exists because Π_1 is in FP, and some (log-space) ASP-complete problem Π_2 in FNP. Here, we define Π_1 as a single-valued function, which is possible because we can define the unique solution of Π_1 as the solution computed by A_1. We can also check such complete problems certainly exists: we can pick (single-valued) F-Horn-SAT for Π_1 and F-Circuit-SAT for Π_2 for example.

Then, define Π_3 as following way:

Input: A combination of the following six:

I_1: An input for Π_1

I_2: An input for Π_2

I_3**:** A DTM M_1

I_4**:** A DTM M_2

I_5**:** A string in Σ^*, used as a part of input to M_1 and M_2

I_6**:** A string in form 0^t, which has no meaning but represents the maximum time to run the DTMs above

Solutions: The solution s of Π_1 with input I_1, if exists, or the pairs of s and solutions of Π_2 with input I_2 which is different from $M_2(s, M_1(I_5, s))$

Π_3 is certainly in FP. We can find a solution (or conclude inexistence of solutions) by solving Π_1 part, which is done in polynomial time, and the existences of solutions of Π_1 part and Π_3 match up.

The rest part of proof, the construction of (log-space) ASP-reductions from any problems Π_4 in FP is done as following way. We show how an instance x_4 of Π_4 can be (log-space) ASP-reduced to an instance of Π_3. First, we check that, by assumptions, there exists a (log-space) function-reduction $f_1 = (f_{1,\mathrm{inst}}, f_{1,\mathrm{bsol}})$ from Π_4 to Π_1 and a (log-space) ASP-reduction $f_2 = (f_{2,\mathrm{inst}}, f_{2,\mathrm{sol}})$ from Π_4 to Π_2. Next, we check that there exists polynomials $p_{1,\mathrm{inst}}, p_{2,\mathrm{inst}}$ for bounding the length of instances like $|f_{1,\mathrm{inst}}(x_4)| \leq p_{1,\mathrm{inst}}(|x_4|)$ and $|f_{2,\mathrm{inst}}(x_4)| \leq p_{2,\mathrm{inst}}(|x_4|)$, a polynomial $p_{1,\mathrm{algo}}$ to bound the running time of A_1 and the length of the solution computed by it, a polynomial $p_{1,\mathrm{bsol}}$ to bound the running time of $f_{1,\mathrm{bsol}}$ and the length of the solutions reduced by it, and a polynomial $p_{2,\mathrm{sol}}$ which bounds the running time of $f_{2,\mathrm{sol}}$.s Then, the final reduced instance x_3 of Π_3 is constructed as following:

$$(f_{1,\mathrm{inst}}(x_4), f_{2,\mathrm{inst}}(x_4), f_{1,\mathrm{bsol}}, f_{2,\mathrm{sol}}, x_4, 0^{p_{2,\mathrm{sol}}(|x_4|, p_{1,\mathrm{bsol}}(|x_4|, p_{1,\mathrm{algo}}(p_{1,\mathrm{inst}}(|x_4|))))})$$

Note that I_6 is just polynomial length to $|x_4|$ and computable in log-space to $|x_4|$, and the reason we do not apply A_1 together in this reduction is, that takes poly-time rather than log-space.

The fact that this reduction maps solutions of x_4 to the solutions of x_3 in 1-to-1 manner holds because f_2 is an ASP-reduction. □

Note: The example of ASP-complete problems is a little complicated, for the purpose of the proof is to show that the ASP-complete problem in FP must be almost ASP-complete in FNP, however, we can construct an ASP-complete problem more simply, using TM which represents the definition of FP.

Proposition 3.16. The function problem defined as following way is ASP-complete.

Input: A DTrans M_1, A DTM M_2, a string $x \in \Sigma^*$, and another string in form 0^t.

Solutions: Let s_1 be the (singleton) set of output y of M_1 with input x if M_1 accepts in time t, otherwise the empty set, and let s_2 be any string $y \in \Sigma^*$ that M_2 accepts (x, y) in time t. Then, the solution is defined to be $s_1 \otimes (s_1 \cup s_2)$.

M_2 represents the second condition of FP, and M_1 represents the third condition of FP. The definition of solutions is a little more complicated than just defined to be s_2, which is the naive translation of the definition of FP, but to do so, we need a promise that s_2 includes s_1. Without the promise, the problem does become functional-complete and ASP-complete in FNP. To avoid such a promise, we chose the definition above.

We state the next as a corollary to Proposition 3.15

Corollary 3.17. F-Circuit-Value or F-Horn-SAT (or any other naturally defined problems in FP) is not ASP-complete in FP, unless P = NP.

Proof. 1-ASPs of F-Circuit-Value and F-Horn-SAT are in FP, and thus the decision versions are in P, however the decision versions of 1-ASP of any ASP-complete problems in FP are NP-complete. □

Observing the above result, we found the problem or unsuitability of FP for considering ASP in it. it is the non-closeness of FP with respect to ASP operation, that is, if a problem Π is known to be in FP, the ASP of it is not ensured to be in FP. This comes from the definition of FP, in the definition, the condition of computability of a solution in poly-time, which specifies FP, is required for just one solution. The condition to all other solutions are same to FNP, this causes the problem. Then, if we want to discuss ASP or ASP-completeness purely in FP, we need to re-define FP as a class closed with ASP operation. One candidate must be definitions from enumeration complexity like P-enumerable [30]. As we used the easiness of enumeration in proof of Proposition 3.2 and many other proofs before, and as Definition 2.21 defines so, the enumeration complexity is useful to ensure the closeness of classes with ASP operation. We discuss on such definitions later in Section 4.2.

3.2.3 ASP-completeness in NPMV

The class NPMV is the another function class than FNP corresponding to NP, and known to have functional-complete problems in FNP, like FSAT, as its functional-complete problem. So, the candidate of the example of ASP-complete problems must

be FSAT and other FNP-complete problems. Furthermore, for FNP $\subset$ NPMV, the FNP-complete problems must have ASP-reductions from any problems in FNP to it. Therefore, we may narrow the area of candidates to ASP-complete problems in FNP, if we only consider problems in FNP. However we get the next result.

Proposition 3.18. Problems in FNP is not ASP-complete in NPMV if FNP $\neq$ NPMV, or in other words, P $\neq$ NP. The set of ASP-complete problems in FNP equals to the set of ASP-complete problems in NPMV, if and only if P = NP. The set of ASP-complete problems in FNP and the set of ASP-complete problems in NPMV have no intersection, if and only if P $\neq$ NP.

Proof. First, we prove that is FSAT is ASP-complete in NPMV, then SAT is solved in P, and therefore P = NP.

Let FSAT be ASP-complete in NPMV. By the definition of NPMV, the trivial functionalization of SAT, $\mathrm{SAT}_{\mathrm{TF}}$, is in it, and therefore, by the above assumption, there exists an ASP reduction $f = (f_{\mathrm{inst}}, f_{\mathrm{sol}})$from $\mathrm{SAT}_{\mathrm{TF}}$ to FSAT. Then, we can solve any instance x of SAT, by applying f to $\mathrm{SAT}_{\mathrm{TF}}$ instance x and its solution candidate YES. Because an ASP-reduction works as a Karp-reduction, so does f, and f never reduces instances with no solutions to instances with solutions. If the original x is satisfiable, then the reduced instance $f_{\mathrm{inst}}(x)$ must have solutions and one of them is $f_{\mathrm{sol}}(x, \mathrm{YES})$, for FSAT $\in$ FNP, the verification of $f_{\mathrm{sol}}(x, \mathrm{YES})$ is done in polynomial time to $|f_{\mathrm{inst}}(x)|$ and $|x|$, therefore we can verify that the satisfiability of x in polynomial time. Otherwise, if the original x is not satisfiable, then the solution candidate YES is not a solution. It is not defined how f_{sol} works with a non-solution, however, by assumption, $f_{\mathrm{inst}}(x)$ is unsatisfiable and no assignment including $f_{\mathrm{sol}}(x, \mathrm{YES})$ is never verified to be solution or satisfying assignment. Therefore, we can solve SAT in polynomial time and, for the NP-completeness of SAT, P = NP holds.

Then, because the ASP-completeness of FSAT in NPMV leads P = NP, and any problem in FNP can be ASP-reduced to FSAT, we can conclude the ASP-completeness of any FNP problem in NPMV leads P = NP.

It is obvious that if P = NP, or therefore FNP = NPMV, then problems which is ASP-complete in FNP is also ASP-complete in NPMV. □

This result tells us that in NPMV, almost all function-complete problems we know are not ASP-complete. So, we may say if we define classes with transducers and prefer naturally appearing problems, the definition like Proposition 2.19, which defines FNP using a transducer, is better. Of course, this does not mean there is no ASP-complete

problem in NPMV, there exists ASP-complete problem indeed. The Bounded Halting type problems are the first examples, as shown in below.

Proposition 3.19. The problem defined as following way is functional-complete and ASP-complete in NPMV.

Input: An NTM M, a string $x \in \Sigma^*$, and another string in form 0^t

Solutions: Strings $y \in \Sigma^*$, (x, y) is accepted by M in at most time t.

or

Input: An NTrans M, a string $x \in \Sigma^*$, and another string in form 0^t

Solutions: Strings output by M with input x in at most time t.

Proof. Obvious. The above definition of problems or their solutions are just the similar to those of NPMV. t gives time bounds and therefore the above problems are in NPMV. And t is enough to be polynomially bounded to the length of the problem instance which is reduced to this problem and computed in polynomial time. □

And, we also have a kind of function version of SAT as ASP-complete problems in NPMV.

Definition 3.20 (FSAT-PARTIAL-ASSIGNMENTS).
FSAT-PARTIAL-ASSIGNMENTS, or FSAT-PA for short, is a function problem defined as following way:

Input: Input x to SAT, which is CNF logical formula, and a subset V of variables which appears in x.

Solutions: Assignments of variables in V, from which x is satisfied by assigning all other variables in some proper way.

Note: this is not a function version in our definition for the extra input V.

Definition 3.21 (FSAT-HALF-ASSIGNMENTS).
FSAT-HALF-ASSIGNMENTS, or FSAT-HA for short, is a function version of SAT defined as following way:

Input: Input x to SAT.

Solutions: Assignments of variables in V, from which x is satisfied by assigning all other variables in some proper way. where V is a first half (in the lexicographic order, the appearing order or any other order) of the variables which appear in x. If the number m of variables is odd, then the first half is defined to be $\lfloor m/2 \rfloor$

Proposition 3.22. FSAT-PA and FSAT-HA is functional-complete and ASP-complete in NPMV.

Proof. First, we prove the functional-completeness and ASP-completeness of FSAT-PA in NPMV by constructing a reduction from any problem $\Pi \in$ NPMV to FSAT-PA.

This is done by the same way as Cook's reduction. By the definition of NPMV, there exists an NTM that verifies (x, y) for any instance $x \in \Pi$ and the solution y to it. Then, we reduce this NTM to CNF logical formula as the way of Cook's reduction, and set the latter part y of the input to the NTM to V.

For FSAT-HA, we just need to add some redundant variables that must be assigned to be true in order to make the V of FSAT-PA be the first half of the variables. □

3.2.4 ASP-completeness in Functional NPSPACE

For NPSPACE, we seem to have no popular expressed definition of corresponding function class. (Here, though the result PSPACE = NPSPACE [20] is widely known and it is popular to use PSPACE to refer the class, we use NPSPACE to refer it, because the nondeterminism plays very important role in defining multi-valued function class. In addition, we name the classes defined below as same way as FNP or NPMV, which include the keyword (or the key-character) N. And, in some case we also refer the class as PSPACE if we do not need nondeterminism at all.) Therefore we first define some function classes which are possibly named to be FNPSPACE. The name used here will be different from those the readers use.

The definitions which are most natural and relatively widely used (but sometimes without definition) must be those defined by transducers, as the Church's thesis says. These definitions can be understood as the extension of NPMV (Definition 2.16). Especially for single-valued functions, this type of definition is widely used as FPSPACE as FP is defined so.

Definition 3.23. We define NPSPACEMV(poly) and NPSPACEMV(exp) as the classes of function problems defined by the relation $R(x, y)$ which satisfies the following conditions:

- y is at most polynomial length to $|x|$, for NPSPACEMV(poly)
  and exponential length for NPSPACEMV(exp).
- There exists an NTrans M with polynomial working space to the input, the set of possible outputs of which with input x equals to the set $\{y|R(x, y)\}$.

Note: Most of the readers may call one of them as FNPSPACE, and may name the class without the first condition as FNPSPACE. The first condition to NPSPACEMV(exp) can be replaced by the condition that the Ntrans M always halts. If we do not use NTrans but NTM, that is, if we count the space required by output to the space bounded by the polynomial, just the second condition becomes enough to represent NPSPACEMV(poly).

We can choose the next type of definitions, which are extensions of the definition of FNP (Definition 2.15). In [18], these types of definitions seem to be imagined.

Definition 3.24. We define FNPSPACE-VER(poly) and FNPSPACE-VER(exp) as the classes of function problems defined by the relation $R(x, y)$ which satisfies the following conditions:

- y is at most polynomial length to $|x|$, for FNPSPACE-VER(poly) and exponential length for FNPSPACE-VER(exp).
- There exists a DTM M with polynomial working space to $|x|$, which accepts (x, y) if and only if $R(x, y)$ holds.

The following definition as extensions of the second definition of FNP (Definition 2.19) is also possible.

Definition 3.25. We define FNPSPACE-TRANS(poly) and FNPSPACE-TRANS(exp) as the classes of function problems defined by the relation $R(x, y)$ which satisfies the following conditions:

- y is at most polynomial length to $|x|$, for FNPSPACE-TRANS(poly) and exponential length for FNPSPACE-TRANS(exp).
- There exists an NTrans M with polynomial working space to the input, which satisfies the same conditions in the second definition of FNP as shown in Proposition 2.19.

We also define here the next class which is used to design ASP-complete problems later.

Definition 3.26. We define FNPSPACE-TRANS(exp, poly out) as the class of function problems defined by the relation $R(x, y)$ which satisfies the following conditions:

- Same as the first condition of FNPSPACE-TRANS(exp).
- Same as the second condition of FNPSPACE-TRANS(exp).

- The output in the second condition is done in the polynomial time to the length of output y.

Note that the conditions of this class make the solutions of any problems in it able to be verified in polynomial time to the length of the corresponding instance and its own length. (However, it is not known whether substitutions of weaker conditions like poly-time verifiability for third one or poly-time outputtability by standard NTrans for the second one gives equivalent definitions.)

Next, we check whether the classes defined above is a function version of NPSPACE. If this condition is not satisfied, it is not desirable to call the class as FNPSPACE.

Proposition 3.27. NPSPACEMV(poly), NPSPACEMV(exp), FNPSPACE-VER(poly), FNPSPACE-TRANS(poly), FNPSPACE-TRANS(exp) and FNPSPACE-TRANS(exp, poly out) are function versions of NPSPACE, on the other hand, FNPSPACE-VER(exp) becomes function version of NEXP.

Proof. We prove the first part of this proposition by showing that the decision versions of any function problems in those classes are included in NPSPACE, and that all of those class includes some function version of any decision problems in NPSPACE.

First, it is easily understood that NPSPACEMV(poly), NPSPACEMV(exp), FNPSPACE-TRANS(poly), FNPSPACE-TRANS(exp) and FNPSPACE-TRANS(exp, poly out) have their decision versions in NPSPACE, because we can construct NTMs which solve the decision version problem by omitting the output tape of the transducer in the definitions of each class. And it is also easily understood that FNPSPACE-VER(poly) has its decision version in NPSPACE, because we can construct a poly-space NTM which solves the decision version problem as an NTM which outputs poly-length solution candidates nondeterministically to its own work tape and verify it by the DTM in the definition of FNPSPACE-VER(poly).

Second, we can check that NPSPACEMV(poly), NPSPACEMV(exp), FNPSPACE-VER(poly),
FNPSPACE-TRANS(poly) and FNPSPACE-TRANS(exp) include the trivial functionalization of any decision problems in NPSPACE. For NPSPACEMV(poly) and NPSPACEMV(exp), this is obvious. For FNPSPACE-VER(poly), it is shown because PSPACE = NPSPACE holds and we can verify the solution candidate "YES" by the DTM which solve the decision problem and exists by the definition of PSPACE. For FNPSPACE-TRANS(poly) and FNPSPACE-TRANS(exp), it is shown also because PSPACE = NPSPACE holds and thus we can output the answer "YES" with no nondeterministic transitions. For FNPSPACE-TRANS(exp, poly out), unfortunately

the class does not include the trivial functionalization of any decision problems in NPSPACE, however, for example, it is easily seen that F-Non-Prenex-QBF-TRACE is contained, for the solution is at most exponential length and can be output just by tracing the $\forall$-$\exists$ tree.

Next, we prove the second part. It is obvious that the decision version of FNPSPACE-VER(exp) is contained by NEXP. We can construct an exp-time NTM which solves the decision version of any problem in FNPSPACE-VER(exp), as an NTM which outputs solution candidates nondeterministically to its work tape and verify it. For the opposite direction, we can see that a function version of SUCCINCT SAT, which is an NEXP-complete decision problem, is contained in FNPSPACE-VER(exp). (For the definition of SUCCINCT SAT, see [18].) If we define the solution of SUCCINCT SAT same as that of FSAT, which is the satisfying assignment of variables, the verification is done in polynomial space to the input length, just by seeing all clauses one by one checking that the focused clause is satisfied by the assignment. □

So, we may conclude that all classes defined above except FNPSPACE-VER(exp) can be regarded as a kind of FNPSPACE, and we forget the class FNPSPACE-VER(exp) from here. And next, we check the relationship between the classes above and see some of the classes above represent the same class.

Proposition 3.28. NPSPACEMV(poly), FNPSPACE-VER(poly) and FNPSPACE-TRANS(poly) are all the same class. (We refer the class as FNPSPACE-VER(poly) from here.)
FNPSPACE-VER(poly) $\subset$ FNPSPACE-TRANS(exp) $\subset$ NPSPACEMV(exp),
and FNPSPACE-TRANS(exp, poly out) $\subset$ FNPSPACE-TRANS(exp) hold.

Proof. We prove just the first part, the statement in the second part are obvious if the first part holds, and the third part is also obvious.

If Π defined by a relation $R(x, y)$ is in NPSPACEMV(poly), then there exists a poly-space NTrans M which outputs the solutions of Π. Then, we can construct a poly-space NTM M' witch verifies whether (x, y) satisfies $R(x, y)$ just by comparing y to the (nondeterministic) output of M on input x. Because PSPACE = NPSPACE, the poly-space NTM M' can be modified to poly-space DTM, and $\Pi \in$ FNPSPACE-VER(poly) holds.

If Π is in FNPSPACE-VER(poly), then there exists a poly-space NTM M witch verifies whether (x, y) satisfies $R(x, y)$. Then, we can construct a poly-space NTrans M which satisfies the conditions in the definition of FNPSPACE-TRANS(poly), just

by nondeterministically outputting the solution candidate y and verifying it by M, and therefore, $\Pi \in$ FNPSPACE-TRANS(poly)

If Π is in FNPSPACE-TRANS(poly), it is obvious that Π is in NPSPACEMV(poly), because the transducer for the definition of FNPSPACE-TRANS(poly) can be used for the transducer for NPSPACEMV(poly). □

Next, we see some easily understood relationships to other function classes or to relationships in decision classes.

Proposition 3.29. The following holds.

- FNP $\subset$ FNPSPACE-VER(poly), or more strongly, NPMV $\subset$ FNPSPACE-VER(poly).

  Proof. By comparing the definitions (for FNP and NPMV, we need to use the definition using verification). The bounds of solution length are all same and the power of verification is strongest in the definition of FNPSPACE-VER(poly). □

- FNPSPACE-VER(poly) $\subset$ FNP $\Rightarrow$ NPSPACE $\subset$ NP, or more strongly, FNPSPACE-VER(poly) $\subset$ NPMV $\Rightarrow$ NPSPACE $\subset$ NP.

  Proof. By the fact that the trivial functionalization of QBF is included in FNPSPACE-VER(poly), FNPSPACE-VER(poly) $\subset$ NPMV leads that QBF is solved by poly-time NTM. □

- FNPSPACE-VER(poly) $\subset$ NPMV $\Leftarrow$ NPSPACE $\subset$ NP.
  Stronger relation FNPSPACE-VER(poly) $\subset$ FNP $\Leftarrow$ NPSPACE $\subset$ NP holds if and only if NPSPACE $\neq$ NP (the case the pre-condition is not satisfied) or P = NP.

  Proof. If NPSPACE $\subset$ NP, the definition of FNPSPACE-VER(poly) becomes same to that of NPMV, thus the first part holds. The second part comes from the fact that FNPSPACE-VER(poly) $\subset$ FNP leads NPMV = FNP, or same condition P = NP, with the first fact of this proposition and if P = NP the relation becomes same to the first part of this statement. □

- FNP $\subset$ FNPSPACE-TRANS(exp, poly out), and NPMV $\subset$ FNPSPACE-TRANS(exp).
  Stronger relation NPMV $\subset$ FNPSPACE-TRANS(exp, poly out) holds if and only if P = NP.

Proof. The first one is because all problems in FNP can be output by an NTM which nondeterministically outputs the poly-length solution candidates to the work and output tapes, and verifying it deterministically in polynomial time, which is possible because the problem is in FNP. The second one uses the definition of NPMV from verification. Because NP $\subset$ PSPACE, the verification is done in PSPACE, and it is also possible to construct an NTM which satisfies the conditions in definition of FNPSPACE-TRANS(exp) as it is constructed for the first part. The last part is because NPMV $\subset$ FNPSPACE-TRANS(exp, poly out) and the fact that the trivial functionalization of SAT is in NPMV lead that SAT $\in$ P. The opposite direction is because P = NP leads FNP = NPMV and the post condition becomes the same to the first part of this statement. □

- FNPSPACE-TRANS(exp, poly out) $\not\subset$ FNP, or more strongly,
  FNPSPACE-TRANS(exp, poly out) $\not\subset$ NPMV

  Proof. Obvious. Problems in FNPSPACE-TRANS(exp, poly out) may have exponential length solutions. □

  Note: From this fact, it is meaningless to consider the relationships like FNPSPACE-TRANS $\subset$ FNP $\Rightarrow$ NPSPACE $\subset$ NP.

Now, we start to discuss the main part, whether the class defined above have ASP-complete problem, and if so what such problems are. Similarly to the other classes, those classes have kinds of Bounded Halting problems as ASP-complete problems.

Proposition 3.30. The classes FNPSPACE-VER(poly), FNPSPACE-TRANS(exp), NPSPACEMV(exp) and FNPSPACE-TRANS(exp, poly out) have ASP-complete problems in them.

Proof. For FNPSPACE-VER(poly) and FNPSPACE-TRANS(exp, poly out), we construct ASP-complete problems in them later, and we do not prove this statement. (We can construct ASP-complete problems same way as other classes.) For NPSPACEMV(exp) and FNPSPACE-TRANS(exp), we can construct ASP-complete problems using Ntrans as following way:

Input: An NTrans M, a string $x \in \Sigma^*$, an integer t in binary, and another string 0^s. For FNPSPACE-TRANS(exp), M must satisfies the second condition of the definitions of FNPSPACE-TRANS(exp).

Solutions: y which is output by M on input x with a computation within time t and space s.

By the existence of s and t, and the fact the second condition of the definitions of FNPSPACE-TRANS(exp) can be checked easily in polynomial time, it is easily seen that this problems are in NPSPACEMV(exp) or FNPSPACE-TRANS(exp). And the ASP-hardness of those problems are obvious, for the above problems can have the solutions y completely same to the original problem which is reduced to the problems. So, these problems are ASP-complete in each corresponding class. □

We also find problems without TM, which are ASP-complete in FNPSPACE-VER(poly) and FNPSPACE-TRANS(exp, poly out). Some of them are technical and very complicated for the necessity of shortening of the solution length though, others are natural. A two-player game type problem (QBF) and a motion planning problem (Sokoban) are shown for each class.

Definition 3.31 (FQBF-FIRSTMOVES). FQBF-FIRSTMOVES is a function problem defined as follows:

Input: Quantified Boolean Formula in prenex normal form, same as standard QBF. The formula can be restricted to either DNF or CNF or any other form.

Solutions: Assignment to the ∃-quantified variables appearing out of any of ∀-quantifier, from which the rest part of QBF holds.

Note: In [16], ♮QBF is defined to count the solutions of FQBF-FIRSTMOVES and it is proved that ♮QBF is ♮PSPACE-complete.

Definition 3.32 (FQBF-COMPRESSEDTRACE).
FQBF-COMPRESSEDTRACE is a function problem defined as follows:

Input: Quantified Boolean Formula in DNF and prenex normal form.

Solutions: Trace of the ∀-∃ tree, but for all node from which the sequence of ∃ node starts, a special character "FALSE TO ALL" is output, when the following case happens. (We call this as compression.)

- The descending ∃-quantified variable is assigned to be all *false* independently from the assignment of ∀-quantified variables.
- If the clauses generated by assigning all the ∃-quantified variable before ∀-quantified variables are assigned can be proved to hold by simplifying just finding clauses consist of one literal.

Definition 3.33 (FSokoban-FIRSTMOVES).
FSokoban-FIRSTMOVES is a function problem defined as follows:

Input: Pair of Sokoban instance and an integer t represented in unary form like 0^t.
(See [8] for the definition of Sokoban.)

Solutions: First t moves of solutions of FSokoban.

Definition 3.34 (F-Nonplanar-Sokoban). F-Nonplanar-Sokoban is a function problem defined as follows:

Input: Board of Sokoban which has under-passes as mentioned in [8].

Solutions: Same as FSokoban.

Proposition 3.35. FQBF-FIRSTMOVES is ASP-complete in FNPSPACE-VER(poly).

Proof. It is easily checked that FQBF-FIRSTMOVES is in FNPSPACE-VER(poly). The solutions are at most linear to the number of variables, thus polynomial. The verification can be done in polynomial space because it is done by solving the QBF generated by applying the partial assignment in the solution.

The rest of proof is by constructing ASP-reduction from any problem in FNPSPACE-VER(poly) to FQBF-FIRSTMOVES. The construction is a minor change of that used to prove the PSPACE-completeness of QBF [24].

Let problem Π be in FNPSPACE-VER(poly). Then, by the definition of FNPSPACE-VER(poly), there exists a poly-space DTM M which verifies solutions. Let triplet $[t_1, t_2, s]$ denotes configuration of M, where t_1 and t_2 represent the contents of the tape and the position of the head and s represents the state of the controller. Then, using the technique of [24], the configuration of DTM with input x and y is represented like below. Here, x represents the instance of Π, and y represents the solution that is going to be verified.

$\exists t_1, t_2$ isReachable$([x, y, s_{\text{init}}], [t_1, t_2, s_{\text{accept}}])$

(x and y implicitly represent the state in which head is located in the first block.)

Next, we modify this to the next.

$\exists y \forall z \exists t_1, t_2$ isReachable$([x, y, s_{\text{init}}], [t_1, t_2, s_{\text{accept}}])$

(Here, z is a new variable.)

This reduction works as ASP-reduction and also as function-reduction, because the solution y is preserved before and after this reduction. □

Proposition 3.36. FQBF-COMPRESSEDTRACE is ASP-complete in FNPSPACE-TRANS(exp, poly out).

Proof. First, we check that FQBF-COMPRESSEDTRACE is in FNPSPACE-TRANS(exp, poly out).

If we do not compress the solution, it is easily checked that the problem is in FNPSPACE-TRANS(exp, poly out) as we did in the proof of Proposition 3.27 for the inclusion of F-Non-Prenex-QBF-TRACE.

Then, we just need to check the effect of the compression, and it is possible to output by an NTrans which satisfies the conditions in the definition of FNPSPACE-TRANS(exp, poly out) and move as following way: For each $\exists$-node from which a burst of $\exists$-nodes starts, the NTrans first checks whether it is possible to proof that the descending QBF holds by setting the all $\exists$-quantified variable to be false and using the naive single literal finding algorithm. If not proved, then it is impossible to compress and the NTrans just selects to assign the variable on the current node and to continue to go down the tree to assign the descending variables. If proved, then the NTrans nondeterministically select to output "FALSE TO ALL" and to return to the ancestor node or to move same as the above not proved case. In case proved but selected to go down, the NTrans records that the fact compression was possible at the node. When assigning the variable in each $\exists$-node, if selected to assign to *true*, the NTrans rewrite to the compression possibility information record the fact that *true* assignment is done. And last, when returned from the descending nodes, the NTrans choose to halt in rejecting state if compression was possible at the node and no *true* assignment is done to the descending $\exists$-quantified variable.

Next, we show how to construct an ASP-reduction from any problem $\Pi \in$ FNPSPACE-TRANS(exp, poly out). Because $\Pi \in$ FNPSPACE-TRANS(exp, poly out) holds, there exists NTrans M which satisfies the conditions of the definition of FNPSPACE-TRANS(exp, poly out). Then, we construct a QBF which represents M using the method of [24] like the following:

$$\text{isReachable}([t_{\text{init}}, s_{\text{init}}], [t_{\text{accept}}, s_{\text{accept}}])$$

Here, $[t, s]$ represents the configuration of NTrans, in detail, t represents the contents of input tape and work tape, and the position of the head, s represents the state of the NTrans, and the output tape does not appear in it. And, we make the all variables in the accepting state to be false, and for the purpose, we make a special transition rule of the NTrans: All contents of tapes are deleted and the head is set to the false position when transiting the final state.

It is checked that the above reduction works as an ASP-reduction as following way: The solutions of Π corresponds to the NTrans transition sequence 1-to-1 by the second condition of the FNPSPACE-TRANS(exp, poly out), and the reduction maps the transition sequence to the trace of QBF 1-to-1. The length of reduced solution and time required is at most polynomial to the length of instance $x \in \Pi$ and its solution y, because the transition sequence of the NTrans before reaching the accepting state is at most polynomial to x and y for the third condition of the definition of FNPSPACE-TRANS(exp, poly out), and the transition after reaching the accepting state that must be compressed to polynomial length for we made the configuration represent by all false assignment and it is proved that QBF holds just by using the single-literal-clause-finding algorithm by the form of the DNF formula.

In detail, the form of the DNF formula is the following form.

$$\begin{aligned}\text{isReachable}(A, B, 2^t) = \\ \exists M \forall X, Y \ (X \neq A \wedge X \neq M) \vee (X \neq A \wedge Y \neq B) \\ \vee (Y \neq M \wedge X \neq M) \vee (Y \neq M \wedge Y \neq B) \\ \vee \text{isReachable}(X, Y, 2^{t-1})\end{aligned}$$

In this formula A and B and M are assigned to all false by assumption. So, if we try to unsatisfy the first clause $(X \neq A \wedge X \neq M) = (X \neq (\text{ALL FALSE}))$, we must set X to all false. this can be seen by the naive single-literal-clause-finding algorithm, and thus we can conclude that the part after reaching the accepting state is compressed. □

Proposition 3.37. FSokoban-FIRSTMOVES is ASP-complete in FNPSPACE-VER(poly).

Proof. It is checked that FSokoban-FIRSTMOVES is in FNPSPACE-VER(poly). The solution can be verified in polynomial space by solving the problem from the state applying the move sequence of the solution and by checking the non-redundancy.

Then, it is enough to show how to construct an ASP-reduction from any problem Π in FNPSPACE-VER(poly). This is done by representing the DTM verifying solution or the NTrans used to proof FNPSPACE-VER(poly) $\subset$ FNPSPACE-TRANS(poly) by Sokoban, as minor modification of [8]. [1]

The modification is as follows: First, in [8] the tape block gadgets are designed not to need to be 'open' every time, and to make it possible to be 'closed' if there exists

[1] It must be necessary for the readers to see [8] in order to understand this proof for Proposition 3.37 or that for Proposition 3.38. And when seeing [8], they might need to be careful that in Figure 9 of [8], the middle one of the five vertically lined boxes in the middle of Device 3 is locating one block lower than the correct position.

arbitrary number of 'open' tape sell. However for the requirement of 1-to-1 relation of solutions, we make the tape must be 'open' every time, by designing last closing part to pass all 'PassReset' gadget of tape. 'Reverser', 'PassReset' without 'reset', and 'OneWay' is used for this modification. Second, to make the FIRSTMOVES to represent the solutions, we make the Sokoban instance must requires to visit 'PassReset' gadgets representing tape blocks of solution area first and set the solution to be verified. This is also modified as same way as the second modification. Last, add enough 'OneWay' gadgets to the last or inside of solution setting part, in order to make the number of moves of boxes for all solution candidates same, or in order to make solution setting part take at least t moves, where t is the input of FSokoban-FIRSTMOVES. (We can make the numbers of moves same, because just two moves are required to pass a 'OneWay' gadget and the difference of the numbers of moves must be even, for we need even number of moves to move a box to the original position again.)

Checking that the above reduction is ASP-reduction may be a little tiresome. It is easy that solutions of Π can be reduced to that of Sokoban in polynomial time. The 1-to-1 relation can be understood by careful check of each gadget of [8]. Devices except 'Crossover' have just one pattern of moves to pass, by the prohibition of redundant moves. The 'Crossover' gadget has a serious problem that we do not need to 'unlock' the gadget if the gadget is visited again, as stated in [8]. However, the 'Crossover' gadgets in the solution setting part are never visited again. So, the possibly 'unlocked' problem does not happen, and together with the prohibition of redundant moves, the 1-to-1 relation holds. □

Note: We can remove the additional input t, by defining FSokoban-FIRSTMOVES as a problem to show the first moves of same number to the boxes in the problem, for example. In such case, the number of boxes is arbitrarily assigned, by adding a sufficiently wide region, which is impossible to get out.

Proposition 3.38. F-Nonplanar-Sokoban is ASP-complete in FNPSPACE-TRANS(exp, poly out).

Proof. F-Nonplanar-Sokoban $\in$ FNPSPACE-TRANS(exp, poly out) holds, because all nondeterministic moves appear in the moves of boxes. The redundant moves are also can be checked, and an NTrans can be designed to halt in the rejecting state if it comes to do a sequence of redundant moves.

Constructing ASP-reduction is by representing the NTM M which is made by omitting the output tape of the NTrans appearing in the definition of FNPSPACE-

TRANS(exp, poly out), which is done in the same way as [8] or the proof for Proposition 3.37. The necessary modification to the reduction of [8] is the first one of the proof for Proposition 3.37 and modification from DTM to NTM, which is easy by 'Junction' gadgets.

Because F-Nonplanar-Sokoban has underpass and we do not need 'Crossover', the possibly 'unlocked' problem does not happen. Thus, the sequence of moves of boxes with prohibition of redundant moves corresponds the computation path of M in 1-to-1 way. And, as the NTrans appearing in the definition of FNPSPACE-TRANS(exp, poly out) have its output 1-to-1 for the computation path, we can conclude that this reduction works as an ASP-reduction. □

Among the above problems, FQBF-COMPRESSEDTRACE seems to be too complicated, and FSokoban-FIRSTMOVES type problems may not appear in real life, and these results have almost only theoretical meaning. However, F-Nonplanar-Sokoban, which also be regarded as F-3D-Sokoban (3-Dimensional version of Sokoban), and the same type of motion planning problems can appear in real use. And, FQBF-FIRSTMOVES appears naturally in real problems.

For example, two-player game Othello is known to be in PSPACE and is PSPACE-complete [14], and assume we design a computer game of Othello. Then, what the program must answer will be just next moves. So, the solution will be the next move. This case will happen in many two-player games, and since QBF is used to represent such games in logical formula, FQBF-FIRSTMOVES will naturally appear.

As another example, the problems of Tsume-Go usually requires just the first move as a solution, and the uniqueness condition is applied to just the first move. Though Go is known to be EXP-complete [19], this tells us the naturalness of considering first moves as solutions.

Therefore we may say in PSPACE, if we consider the ASP and want to prove ASP-completeness, it is better to use FIRSTMOVES type definition of solutions on functionalization for two-player game type problems, and to use the solutions which describes the move sequence on functionalization for motion-planning type problems.

There also is a question of whether NPSPACEMV(exp) and FNPSPACE-TRANS(exp) have ASP-complete problems, which can appear in natural. But, they are not found in this research.

3.2.5 ASP-completeness in Functional NEXP

For NEXP the function version of it will be defined same way as NP, FNP or NPMV, but there seems no well-used clearly expressed definitions as same as PSPACE. Here, as an example, we define a function version class of NEXP, as an extended definition to FNP, and see an ASP-complete problems in it.

Definition 3.39 (FNEXP-VER(exp, exp ver)). We define FNEXP-VER(exp, exp ver) as the classes of function problems defined by the relation $R(x, y)$ which satisfies the following conditions:

- y is at most exponential length to $|x|$
- There exists a DTM M which accepts (x, y) in exponential time to $|x|$ if and only if $R(x, y)$ holds.

For NEXP, decision problem SUCCINCT CIRCUIT SAT is known to be complete in it (see [18] for the definition), and we can prove a function version of SUCCINCT CIRCUIT SAT is ASP-complete in FNEXP-VER(exp, exp ver).

Definition 3.40 (F-SUCCINCT-CIRCUIT-SAT). F-SUCCINCT-CIRCUIT-SAT is a problem defined as following way:

Input: Same as SUCCINCT CIRCUIT SAT.

Solutions: Input to the input circuit which makes the output of the circuit to **true**, in run length encoding compressed form.

Note: Since the instance is given in a compressed form, it is natural to define the solution to be represent in compressed form, and run length encoding is one of the very simple compression schemes.

Proposition 3.41.
F-SUCCINCT-CIRCUIT-SAT is ASP-complete in FNEXP-VER(exp, exp ver).

Proof. This proposition is proved in same way that the ASP-completeness of F-CIRCUIT-SAT is proved.

It is obvious that F-SUCCINCT-CIRCUIT-SAT is in FNEXP-VER(exp, exp ver), for the verification of a solution is just decompress it and also decompress the input succinct circuit, and check that the output of circuit is true.

The construction of an ASP-reduction from any problem π in FNEXP-VER(exp, exp ver) is same to that for F-CIRCUIT-SAT, Cook's reduction. If the original solution

is short, most blocks of the initial tape of the verifier DTM are blank, most of the resulting reduced inputs to the circuit are *false* (we assume the blank is encoded only with *false*) and the part is compressed to polynomial length by run length encoding, and it is easy to apply run length encoding to non blank part, too. Therefore, solutions are mapped 1-to-1 in polynomial time with respect to the length of the original instance and solutions. □

We mention that the other classes defined like PSPACE also have ASP-complete problems. Bounded Halting type problems are the examples.

3.2.6 ASP-completeness in Functional EXP

For EXP, seeing the results for FP, it seems not to desirable to define FEXP as an extension of FP as a multi-valued function class. So, we do not choose such definition, but a definition like FNPSPACE-VER(poly), and show that we can design a function version of standard P-complete problem to be ASP-complete is the class.

Definition 3.42 (FEXP-VER(poly)). We define FEXP-VER(poly) as the classes of function problems defined by the relation $R(x, y)$ which satisfies the following conditions:

- y is at most polynomial length to $|x|$.
- There exists a DTM M which accepts (x, y) in exponential time to $|x|$ if and only if $R(x, y)$ holds.

Definition 3.43 ($\mathbf{FG_1}$-FIRSTMOVES). FG_1-FIRSTMOVES is a function version of G_1 defined as following way:

Input: Same as G_1

Solutions: The first moves of player A, or the assignment of A-VAR after A's first turn, from which A wins no matter how B moves.

Proposition 3.44. FG_1-FIRSTMOVES is ASP-complete in FEXP-VER(poly).

Proof. The proof is done in the same way as the proof of Proposition 3.35. There, we use the technique of [23] and the fact that APSPACE = EXP [7].

It is easily checked that FG_1-FIRSTMOVES is in FEXP-VER(poly). It is possible to verify just solving the G_1 instance generated by applying the first moves.

The construction of ASP-reduction is as follows: First, we construct a poly-space ATM M which verifies the solution y. Next, we add the special transition rule to

M, that M can and only can write the area of the tape, on which the solution to be verified is wrote, to anyhow M likes at the first move. Last, represent M with logical formulas as it is done in [23]. □

FG_1-FIRSTMOVES is as natural as FQBF-FIRSTMOVES, and there will be game type problems, the FIRSTMOVES functionalization of which become ASP-complete in FEXP-VER(poly). However, unfortunately we can prove FG_3-FIRSTMOVES or FSHOGI-FIRSTMOVES as defined the same way becomes not ASP-complete in FEXP-VER(poly), by showing that those problems have at most polynomial number of first moves and that there are problems which have exponentially many solutions in FEXP-VER(poly).

3.2.7 Relationship to Complexity Analysis of ASP and Discussions

Last of this section, let us do some general discussions a little more, like listed below:

- Whether the ASP-completeness can be used to prove the completeness of k-ASP decision version.
- Whether the above techniques to define classes or compressing solutions can be used much higher classes.
- What the open problems are and what are desired to be studied more.

Application of ASP-completeness to the Analysis of the Complexity of k-ASP

In FNP, one of the importance or usefulness of ASP-completeness is that ASP-completeness in FNP leads the completeness of the decision version of k-ASP of the problem for any k (Theorem 2.28). Then, it is natural to consider whether such theorem holds in the classes above. The answer is YES except for FP like below:

Theorem 3.45. If a function class $\mathcal{C}$ is closed with respect to ASP operation and $+1$ operation, and a problem Π is ASP-complete in $\mathcal{C}$, then the decision version of k-ASP of Π is ASP-complete in $\mathcal{C}$. If $\mathcal{C}$ is not closed with respect to ASP operation, the decision version of k-ASP of Π is ASP-hard for $\mathcal{C}$.

Proof. Let Π' be any problem in $\mathcal{C}$. Then, by assumption of the closeness for $+1$ operation, $Pi'+k$ is in $\mathcal{C}$, and there exists an ASP-reduction from $Pi'+k$ to Π by the ASP-completeness of Π, which also works ASP-reduction from k-ASP of $Pi'+k$

to k-ASP of Π by Theorem 2.25. Combining the fact that k-ASP of $Pi'+k$ contains problems same to Pi', we get an ASP-reduction from Pi' to k-ASP of Π. □

Corollary 3.46. For NPMV, FNPSPACE-VER(poly), NPSPACEMV(exp), FNPSPACE-TRANS(exp), FNPSPACE-TRANS(exp, poly out), FNEXP-VER(exp, exp ver), FEXP-VER(poly), if a problem Π is ASP-complete in each class, the decision version of k-ASP of Π is complete in the corresponding decision version class.

From the above result, we can say that the k-ASP decision version of FQBF-COMPRESSEDTRACE is PSPACE-complete. Then, how about the case defining the solutions of QBF (of prenex normal form) by its traces as it is generally done. The analysis result of FQBF-COMPRESSEDTRACE will answer this question. The compressed trace and original trace corresponds 1-to-1, and k-ASP decision version of FQBF-COMPRESSEDTRACE are PSPACE-complete. So, we can conclude the hardness of k-ASP decision version of FQBF comes from the hardness of the part of finding another solution, not from reading the input or comparing to the known solutions. Though compression of solutions makes problems away from the natural definition, analysis results can be used like this way.

We can also say the k-ASP decision version of FQBF-FIRSTMOVES is PSPACE-complete. But, here is a problem as we saw before in Section 3.1. By the definition of FNPSPACE-VER(poly), the solution needs polynomial space to be verified, and the hardness may come from the verification part, Unfortunately it seems impossible to solve this problem just with this result, however, if we use nonprenex form as it is done in Section 3.1, it becomes possible to add solutions represented just by first moves and easily verified, and the same results as 3.10 may be acquired.

Further Extendibility

We extended the idea and analysis techniques of ASP to NEXP. Then is it possible to extend them further classes like NEXPSPACE or DEXP. Though we can not say certainly, we can expect the possibility. First, the Bounded Halting type problems is expected to be ASP-complete in almost all class, as far as a class is defined by TM or transducer as defined above. Second, the limitation of the solutions to polynomial length is expected to work. For motion planning type problems, the solution, which is a sequence of moves, will represent the sequence of transition of TM, and therefore, they are expected to ASP-complete as well as Bounded Halting type problems. Third, the compression of solutions may not work in same way. It might be difficult for solutions in doubly-exponential length to be compressed to polynomial length. However, it may

be enough to compress it to exponential length, if we want to show the hardness of the ASP of problems over NEXP.

Open Problems

One most important problem to be studied is to make chains of ASP-reductions. One of the usefulness about ASP-completeness is that ASP-completeness can be shown by the chain of ASP-reduction. In NP, for example, we prove the NP-completeness of SAT by representing TM as Cook's reduction does, then 3SAT by constructing a Karp-reduction from SAT, then HC again by a Karp-reduction from 3SAT, and so on. We need to check reducibility from any problem just for SAT, and it is enough for other problems to show reducibility from problems already known to be complete. However, the results above are just the first SAT part; the chain of ASP-complete problems does not appear. So, it is desired that such chain is constructed. Without them, the usefulness of ASP-completeness does not fully works.

It is also interesting problem to see whether NPSPACEMV(ver) have some natural function version of any PSPACE-complete decision problem as it is negatively answered for NPMV.

For FNPSPACE-VER(poly) and FEXP-VER(poly), we showed that the problems to find the first moves of some logical games are ASP-complete, and stated that those problems naturally appears to represent two player games, but we do not find any function problems corresponding two player games which are ASP-complete in those classes, so far. One reason was that the number of the candidate of the first move of some two player games is at most polynomial, then how about function problems which require to answer first few moves or logarithmic moves, or how about two player games in which a player can move polynomial number of pieces in one turn. This problem must be worth considerable.

Chapter 4

Other Results not Related to Extension of ASP

In this chapter, we discuss some interesting problem related to ASP, which does not have strong relationship to extending ASP to the other classes than NP or FNP. The topics are listed below:

- Problem of whether NP-completeness of all k-ASP decision versions lead ASP-completeness. The opposite direction of Theorem 2.28.
- Re-defining functional P using enumeration.
- Strengthening ASP-reduction.

4.1 Relationship between Completeness of k-ASP and ASP-completeness

In FNP, as Theorem 2.28 says, if a problem Π is ASP-complete in FNP, k-ASP decision version of Π is NP-complete for all $k \geq 0$. Then, how about the opposite direction? This is a problem raised in [32]

Problem 4.1 ([32]). Let Π be a function problem in FNP, and k-ASP of Π is NP-complete for all $k \geq 0$. Then, is it always hold that Π is ASP-complete in FNP?

In [31], the above problem is expected to have negative result, and a strategy to show that is stated using one-way functions, but the problem was not solved.

And, here, we give a negative answer to Problem 4.1, using a very simple technique. The assumption used in the solution is P $\neq \oplus$P, which is weaker assumption than

P $\neq$ UP, the equivalent assumption to the assumption of the existence of one-way functions.

Proposition 4.2. If P $\neq \oplus$P, there exists problem Π in FNP, the all k-ASP of which is FNP-complete but which is not ASP-complete in FNP.

Proof. Let us consider the problem $\Pi = \text{SAT}\times 2 \cup \text{SAT}\times 2{+}1$.

Then, it is verified that k-ASP decision version of Π is NP-complete for any $k \geq 0$. l-ASP decision version of SAT is NP-complete for any $l \geq 0$ as proved in [32]. Then, for even $k = 2l$, the decision version of k-ASP, which is $2l$-ASP, of SAT$\times$2 is NP-complete for any l, thus for any even k and, for odd $k = 2l + 1$, the same thing holds for k-ASP of SAT$\times$2+1. From this observation, we get the NP-completeness of for any k-ASP decision version of Π.

On the other hand, we can show that if there exist an ASP-reduction from FSAT to Π we can solve $\oplus$SAT in polynomial time, just by applying the ASP-reduction to any instance x of $\oplus$SAT (as an instance of FSAT). If x is reduced to a problem in SAT$\times$2, then we can answer NO to x as an instance of $\oplus$SAT, and if reduced to a problem in SAT$\times$2+1, YES. This means P $\supset \oplus$P holds. □

4.2 Functional P Defined from Enumeration Complexity

As seen in Section 3.2.2, and stated in Corollary 3.17, natural FP-complete problems like F-Horn-SAT is not ASP-complete in FP unless P = NP, against expectations. And this comes from the fact that FP is not closed with ASP operation, or that the poly-time computable condition does not bound all solutions.

Then, is the definition of FP as a multi-valued function class is suitable? One meaning of the definition of FP must be the result FP = FNP if and only if P = NP. However, as seen below we can get same result by defining FP in another way, and this kind of definition seems to cause another undesirable thing defining class Functional EXP in same way. So, in this section, we discuss on defining Functional P from Enumeration Complexity. And, discuss whether natural FP-complete problems like F-Horn-SAT become ASP-complete in such class.

4.2.1 Problem on Defining Functional EXP as an Extension of FP

We defined a class named FEXP-VER(poly), which can be regarded Functional EXP, as an extension of FNP in Section 3.2.6. On the other hand, it is also possible and may be more natural to define Functional EXP as an extension of FP, as a class defined

by adding a condition of exp-time computability of one solution to some definition of Functional NEXP.

However, those definitions seem to cause an undesirable situation, a statement like following will hold:

If FNPSPACE = FEXP, then FNPSPACE = FNEXP.

This comes from that the definitions of FNPSPACE will not have a special condition to one solution. The +1 operation is useful to prove it. Since the decision version of the above statement is not believed to hold, this must be an undesirable situation.

4.2.2 Functional P Defined from Enumeration Complexity

There are some definitions of classes from enumeration complexity, like P-enumerable[29, 30], Incremental Polynomial Time, and Polynomial Delay [15]. P-enumerable may be not enough to ensure the closeness of Functional P with ASP operation though, Incremental Polynomial Time is enough to ensure it, and Polynomial Delay, which is stronger than the other two, is also enough.

Then, if we need another Functional P than FP for considering ASP in it, it is probably good idea to use Polynomial Delay or FNP $\cap$ Incremental Polynomial Time. Here, we choose to select Polynomial Delay and start discussion.

First, we state the next proposition.

Proposition 4.3 (Equivalent Definition to Polynomial Delay). The class of function problems defined by relation $R(x, y)$ which satisfies the following conditions equals to Polynomial Delay.

- y is at most polynomial length to $|x|$.
- There exist a polynomial p and a DTrans M, which receive x and an integer N in unary form 0^N as input, and M outputs size n subset of $\{y|R(x, y)\}$ or whole $\{y|R(x, y)\}$ in time $N \cdot p(x)$.

Proof. It is obvious that above class includes Polynomial Delay.

The proof of the other direction of inclusion is done in the following way: Let Π be a problem in the above class, and let x be an instance of it. Then, there exists p and M, and M outputs $N = 1, 2, 4, 8, 16, \cdot$ solutions in time $N \cdot p(|x|)$ each. Now, we construct a DTrans M' which outputs $y \in \{y|R(x, y)\}$ one by one as following way.

1. Let $N = 1$, and $S = \{\}$,

2. Run M with input x and 0^N, and let T be the output set of M.

3. Output $\lceil N/2 \rceil$ of $T \setminus S$ and add them to S, keeping the condition that S is sorted. If $T \setminus S$ has not enough elements, then output whole $T \setminus S$ and go to halting state.

4. Multiply N by 2, and return to 2.

It is obvious that M' outputs $y \in \{y|R(x,y)\}$ one by one until whole $\{y|R(x,y)\}$ is output. And the time used in the second part of M', until the N-th of $\{y|R(x,y)\}$ is output, is $(1+2+4+\cdots+2^t+2^{t+1})p(|x|) \leq 4N \cdot (|x|)$, here t is the integer which satisfies $2^t \leq N < 2^{t+1}$. The time used in the third part of M' is, using the fact that S is sorted, $O(2^t \cdot \log 2^t)$ for each loop indexed by 2^{t+1}, and in total until N-th of $\{y|R(x,y)\}$ are output, the time is atmost $O(N \log N)$. Here, we use the fact that the solution length is at most polynomial to $|x|$, by the fact, we can bound the size of $\{y|R(x,y)\}$ by an exponential of $|x|$, and thus $\log N$ by some polynomial p' of $|x|$. Therefore, we can conclude that the third part requires at most $O(N) \cdot p'(x)$ until $N-1$ solutions are computed. Combining the above observations, we conclude that M' satisfies the condition of Polynomial Delay, and the class defined above in the proposition statement is included in Polynomial Delay.

To be more precise, we need to design M' to buffer the set S, or wait outputting until each solution is required to be output. □

This equivalent definition is expected to be useful if we try to prove ASP-completeness of some problems in Polynomial Delay. In the original definition there may exist exponential number of solutions, and the DTrans may run exponential time in total. So, it is impossible to represent whole move of the transducer. However in the equivalent definition, the number of solutions which is required to output is in the input, we do not need to worry about preparing exponential size of variables. (It may be necessary to re-define ASP-reduction or ASP-completeness for the additional input N, though,)

Last for this section, let us discuss the possibility of natural FNP functional-complete problems to be ASP-complete in Polynomial Delay. (Note: All functional-complete problems in FNP are functional-hard for Polynomial Delay, and complete if included in Polynomial Delay, by the definitions.)

First, we can see F-Circuit-Value is not ASP-complete in Polynomial Delay, though it is functional-complete.

Proposition 4.4. F-Circuit-Value is not ASP-complete in Polynomial Delay, unless $\sharp\mathrm{P} \subset \mathrm{FP}$.

Proof. Polynomial Delay contains problems, the counting problem corresponds to which is ♯P-complete, like F-Horn-SAT or the FSAT+$2^{|V|}$ as defined in Proposition 4.6. On the other hand, the number of solutions of F-Circuit-Value is computable in polynomial time. Combining the above observation and the fact that ASP-reduction works as a parsimonious reduction, it is understood that the ASP-completeness of F-Circuit-Value induces ♯P $\subset$ FP. □

Then, we also got the next corollary, which is the Polynomial Delay version of the Proposition 4.2.

Corollary 4.5. If ♯P $\not\subset$ FP, there exists a problem, the k-ASP of which is functional-complete in Polynomial Delay for any $k \geq 0$, and which is not ASP-complete in Polynomial Delay.

The next example may tell us that it is unlikely expected there exist natural problems which are ASP-complete in Polynomial Delay.

Proposition 4.6. The problem FSAT+$2^{|V|}$ is in Polynomial Delay, where FSAT+$2^{|V|}$ is defined as following:

Input: Same as FSAT.

Solutions: The direct sum of solutions of FSAT and $\{0,1\}^{|V|}$, where V represents the set of all variables in the input.

Proof. The solution can be enumerated just by searching all assignments to V. For any assignment, at least one solution from $\{0,1\}^{|V|}$ corresponds and is output, and if the assignment satisfies the input CNF, one more solution corresponds and is output. □

To reduce this FSAT+$2^{|V|}$ to some problem Π in Polynomial Delay in ASP manner, it must be necessary to match the FSAT part to some part of Π, which seems to be impossible for the FNP functional-completeness of FSAT. However, as ♯Horn-SAT is proved to be ♯P-complete in [30], there also exists possibility of existing some ASP-reduction. So, the question of whether there exist natural FP-complete problems which are ASP-complete remains open.

4.3 Strengthening ASP-reduction

In our definition and definitions in [28] or [32], ASP-reduction is defined as a special version of parsimonious reduction not functional reduction. However, on the other

hand, the ASP-reductions constructed so far were those which work also as functional or Levin reduction. Indeed, they did have the property that the inverse of solution map is computable, which is stronger property than functional reduction. Then, it may be also interesting re-define ASP-reduction as the following way. (This definition is used only in this section, if we compare the two ASP-reductions, we refer the original one as weak one, and the new one as strong one.)

Definition 4.7 (re-definition of ASP-reduction). An **ASP-reduction** f is a reduction between two function problems Π_1 and Π_2 defined by relations R_1 and R_2 consists by 3 functions f_{inst}, f_{sol} and f_{bsol} which satisfies the next properties:

- f_{sol} maps the solution set of $x \in \Pi_1$ to $f_{\mathrm{inst}}(x) \in \Pi_2$ in 1-to-1 and onto manner. ($\lambda y.\ f_{\mathrm{sol}}(x, y)$ works as a bijection between the two solution set.)
- $\lambda y'.\ f_{\mathrm{bsol}}(x, y')$ is the inverse of $\lambda y.\ f_{\mathrm{sol}}(x, y)$.

Using this definition we can prove the next proposition.

Theorem 4.8. FP, FNP, NPMV, and the other functional class defined so far is not closed with either of functional, Levin, parsimonious or weak ASP-reduction. In contrast, all of them are closed with strong ASP-reduction. A class is said to be closed with reduction, no problem outside the class is reduced to a problem in the class,

Proof. The first part is constructing counter examples of reductions like following.

- From a problem which always have two solution 0 and very long one (doubly exponential length is enough) to a problem with two solution 0 and 1. This example is for the case, the class bounds solution length to the instance length.
- From a problem which always have two solutions 0 and YES/NO of very difficult problem to a problem with two solutions 0 and 1. Note that we define 'NO' as a solution. This example is for the case, the class requires verifiability of solution or computability of solutions.

The second part is done by just checking the definitions. We show them by the example. The above counter examples are avoidable.

As notations, Let the reduction reduces (x, y) to (x', y') where x represents an instance and y represents a solution, and let the class in consideration is $\mathcal{C}$.

The case $\mathcal{C}$ has a bound on solution length: Let $\mathcal{C}$ has bound on solution length like $|y'| \leq poly(|x'|)$, then $|y'|$ is bounded by $poly(|x'|)$, $|x'| \leq poly(|x|)$ for poly-time computability of f_{inst}, and $|y| \leq poly(|y'|)$ for poly-time computability of f_{bsol}. Combining these, we get $|y| \leq poly(|x|)$.

The case $\mathcal{C}$ requires verifiability: Let $\mathcal{C}$ has a condition like y' is verified in polytime to x', then, since each direction of transformation of $(x, y) \leftrightarrow (x', y')$ is possible, y is a solution to x if and only if y' is a solution for x'. (If y is not a solution then we can check it by transforming (x, y) to (x', y'), verifying (x', y'), inverse-transforming (x', y') to (x'', y''), and comparing (x, y) to (x'', y''). The verification may success, however in such case, (x'', y'') must be different from (x, y).) Thus the verification of (x, y) is possible.

The case $\mathcal{C}$ requires computability by some transducer: Let $\mathcal{C}$ is defined with polytime NTran with input x' and output y', then, y' can be computed from x', x' is computable from x with f_{inst}, y is computable from y' with f_{bsol}. Combining these, we get the computability of y from x. Here, using the property of 1-to-1, we also say any y can be output if any y' can be output. □

The above theorem has an important meaning. It shows that the (strong) ASP-reduction succeeded to divide the functional complexity class, which is impossible for the reductions which are widely used. The closeness of classes with reduction plays an important role in decision classes. For example, we can say, "If SAT is solved in polynomial-time, P = NP", by the fact that P is closed with Karp-reduction. And from the proposition, we can prove the strong ASP-reduction version of Proposition 3.18.

Proposition 4.9. Problems in FNP is not (strong) ASP-complete in NPMV if FNP $\neq$ NPMV.

Proof. If any problem Π in FNP is (strong) ASP-complete in NPMV, FNP $\supset$ NPMV holds by the closeness of FNP with (strong) ASP-reduction. □

Chapter 5

Conclusion

We discussed on extending ASP and the analysis technique of it so far. In summary, the results we got are the following:

- On considering ASP and analysing the complexity of the decision version in classes other than NP (Section 3.1):
  - The applicability of the analysis technique of Theorem Th:ASP-reduction and Theorem 2.27 to the classes which potentially have exponentially long solutions is pointed out. (Section 3.1.1.)
  - Some function problems are proved to have their k-ASP decision versions as complete problems in their corresponding decision class, using the technique of Theorem 2.27.
    - F-Circuit-Value and F-Horn-SAT for P. (Proposition 3.2 and Proposition 3.5.)
    - FG_3 for EXP. (Proposition 3.8.)
    - Non-Prenex-QBF for PSPACE. (Proposition 3.10.)
  - For PSPACE, a popular puzzle problem F-Sokoban is proved to have the same property, though Theorem 2.27 is not used. The proof is just by constructing Karp-reduction to each k-ASP. (Proposition 3.12.)
- On considering ASP-completeness in other classes than FNP (Section 3.2.):
  - Necessary conditions for a function problem to be ASP-complete in the corresponding function class is stated.

- For FP, it is stated that k-ASP decision version of any ASP-complete problems must be NP-complete, and thus any standard problems is not ASP-complete in that class. (Proposition 3.15 and Corollary 3.17.)
- For NPMV, it is proved that no problems in FNP is ASP-complete in NPMV, though the standard functional-complete problems in NPMV are those in FNP. (Proposition 3.18.)
- An example of ASP-complete problem in NPMV is shown, that is FSAT-HA, which is a candidate of standard ASP-complete problems. (Proposition 3.22.)
- As functional classes for PSPACE, some class definitions are tested, and some problems are proved to be ASP-complete in those classes. All of those are proved by representing the TM or Trans which appears in the definition of each class.
  - Bounded Halting type problems. (Proposition 3.30.)
  - FQBF-FIRSTMOVES and FSokoban-FIRSTMOVES for FNPSPACE-VER(poly). (Proposition 3.35 and Proposition 3.37.)
  - FQBF-COMPRESSEDTRACE and F-Nonplanar-Sokoban in FNPSPACE-TRANS(exp, poly out). (Proposition 3.36 and Proposition 3.38.)
- F-SUCCINCT-CIRCUIT-SAT is proved to be ASP-complete in FNEXP-VER(exp,exp ver), which is a functional class corresponding to NEXP. (Proposition 3.41.)
- FG_1-FIRSTMOVES is proved to be ASP-complete in FEXP-VER(poly), which is a functional class corresponding to EXP. (Proposition 3.44.)

- The open problem raised in [32] is solved. (Proposition 4.2.)
- Some function classes corresponding to P and defined with enumeration complexity are discussed.
  - An equivalent definition to Polynomial Delay is constructed, which is expected to be useful to consider ASP-completeness in Polynomial Delay.
  - It is stated that F-Circuit-Value is not ASP-complte in Polynomial Delay, and the open problem raised in [32] is solved for Polynomial Delay version.
  - A problem, named FSAT+$2^{|V|}$, which seems difficult to be ASP-reduced to natural FP problems is shown.

- A strengthened definition of ASP-reduction is considered, and it is shown that all function classes in this thesis become closed with the reduction, which does not happen with the other popular reductions.

Comparing the first target of this work, it is possible to say that we succeeded to extend the idea of ASP and its analysis technique in many part. And, from those we failed to extend them, we got some knowledge about ASP-reduction which must be not able to be found if we consider ASP just in NP or FNP, as it is also being expected.

We succeeded to show that it works just defining ASP and the complexity analysis scheme in completely same way as it is done in FNP. Before this work, the effect of the exponential explosion of solution length may be too much afraid of, however, at least defining ASP and its complexity analysis scheme, we did not need to suffer from the exponential explosion. As a result, we succeeded to prove the hardness of finding another solution of a motion planning puzzle problem Sokoban, which does not come from the hardness of verifying the given solutions or extracting information from it.

We also stated that the techniques for complexity analysis of ASP, which are Theorem Th:ASP-reduction and Theorem 2.27, works in higher complexity classes, and even lower complexity classes, and applied Theorem 2.27 to some standard problems in each class.

Talking on the extension of the idea of ASP-completeness, we showed the unsuitability of FP for considering ASP-completeness in it. We might have thought that the idea of ASP-completeness must obviously extended to lower class, because there is no worry about solution length, however, the ASP-complete problems in FP is those we will never see, if we only consider functional-reduction. And, those problems tell us the current definition of FP is far from our image or recognition of FP, as a class of poly-time solvable problems; the problems are almost FNP-complete problems indeed. Therefore we may expect a function version class of P may fit better to our image, if some naturally appearing problem become ASP-compete in the class.

We also considered ASP-completeness in NPMV, another candidate than FNP which may be called Functional NP. As a result, we got to divide the complete problems of FNP and NPMV with no intersection, which is impossible by standard functional-reduction. Though the division is possible with some other reductions like Levin-reduction, this is an interesting property of ASP-reduction, and is never found just considering ASP-reduction in FNP.

We considered ASP-completeness in higher classes, which is the first target. In those classes we seem to have many problems left, but some problems are solved. The

exponential explosion of solution lengths had been worried for the ASP-completeness, as well as extending the analysis scheme, and we clarified the worry as a necessary condition for ASP-completeness. Then, we showed two techniques to design ASP-complete problems, and constructed problems which are ASP-complete in each corresponding class, one is FIRSTMOVES type definition and the other is compression of solutions. So we can say we extended the idea of ASP-completeness to higher classes, and the first target is attained. Though some results are on very complicated problem definitions, by the necessary condition, we can say it is unavoidable for some definitions to be complicated.

As reviewed above we got many positive results, but there also remains problems too.

The most important problem is the lack of the chain of ASP-reduction, as already mentioned in the last discussion part of Chapter 3. The chain is necessary not only for considering on ASP-completeness, but also just considering complexity of ASP. Theorem 2.25 is the theorem for that purpose. Until such chain is constructed, we can not say the extension of analysis technique is completely done. And also, we can expect that we will find much more knowledge about ASP and ASP-reduction by constructing such chain, as we met many discoveries doing this work.

As an example which show we can use such chain in real problems, [33] shows an application of Theorem 2.25 to the Yozume problem of Tsume-Shogi, and proved the EXP-completeness of the problem. Since there are similar problems for Go or Chess, it is expected that such chain of problems are constructed.

And there is a Karp-reduction from QBF in [13] for Sokoban. Though the reduction does not work as an ASP-reduction and seems to be impossible to be modified to ASP-reduction, such kind of re-construction of proof of completeness is expected to help to construct the chain.

Also in PSPACE, a two-player game Generalized Geography is proved to be PSPACE-complete by a Karp-reduction from QBF [17]. Though the reduction does not map the traces of these two problems parsimoniously and does not work as an ASP-reduction, it may be possible to modify it to ASP-reduction, and a chain will be constructed. (We can easily construct ASP-reduction from Generalized Geography to its 1-ASP, defining trace as solutions, indeed.) The fact that Generalized Geography is used to prove the PSPACE-completeness of Othello is also remarkable. Though the reduction used there does not work as parsimonious or ASP-reduction unfortunately either, we can expect it is modified to an ASP-reduction, and the another chain will constructed.

Other problems of this work, will be cleared if the chain is constructed. For example, one problem must be the shortage of applications. The applications we have now are that to Sokoban and that to Tsume-Shogi. But if the above done, we will get many application like Tsume-Othello and Tsume-Go.

So, the main future work is to construct chains of reductions.

References

[1] Aaronson, S. The complexity zoo. URL: `http://www.cs.berkeley.edu/~aaronson/zoo.html`.

[2] Adachi, H., Kamekawa, H., and Iwata, S. Shogi on $n \times n$ board is complete in exponential time (in japanese). *Transactions of the IEICE J70-D*, 10 (October 1987), 1843–1852.

[3] Aida, S., Crasmaru, M., Regan, K. W., and Watanabe, O. Games with uniqueness properties. *Theory of Computing Systems 37*, 1 (January 2004), 29–47.

[4] Àlvarez, C., and Jenner, B. A very hard log-space counting class. *Theoretical Computer Science* (1993), 3–30.

[5] Bednarek, M. Automatic generation of sliding block puzzles, 2003.

[6] Book, R. V., Long, T. J., and Selman, A. L. Quantitative relativizations of complexity classes. *SIAM Journal on Computing 13* (August 1984), 461–487.

[7] Chandra, A. K., Kozen, D. C., and Stockmeyer, L. J. Alternation. *Journal of the ACM 28* (1981), 114–133.

[8] Culberson, J. C. Sokoban is PSPACE-complete. Tech. Rep. TR97-02, University of Alberta, 1997.

[9] Do, D.-Z., and Ko, K.-I. *Theory of Computational Complexity.* Wiley Interscience, 2000.

[10] Fraenkel, A. S., and Lichtenstein, D. Computing a perfect strategy for $n \times n$ chess requires time exponential in n. *Journal of Combinatorial Theory A*, 31 (1981), 199–214.

[11] Fukuda, K., and Matsui, T. Finding all the perfect matchings in bipartite graphs. *Applied Mathematics Letters 7* (1994), 15–18.

[12] Garey, M. R., and Johnson, D. S. *Computers and Intractability. A Guide to the Theory of NP-completeness.* W. H. Freeman, 1979.

[13] Hearn, R. A., and Demaine, E. D. The nondeterministic constraint logic model of computation: Reductions and applications. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP 2002)* (2002), pp. 401–413.

[14] Iwata, S., and Kasai, T. The Othello game on an $n \times n$ board is PSPACE-complete. *Theoretical Computer Science 123* (1994), 329–340.

[15] Johnson, D. S., Yannakakis, M., and Papadimitriou, C. H. On generating all maximal independent sets. *Information Processing Letters 27* (1988), 119–123.

[16] Ladner, R. Polynomial space counting problems. *SIAM Journal on computing* (1989), 1087–1097.

[17] Lichtenstein, D., and Sipser, M. GO is polynomial-space hard. *Journal of the ACM 27*, 2 (April 1980), 393–401.

[18] Papadimitriou, C. H. *Computational Complexity.* Addison-Wesley, 1994.

[19] Robson, J. M. The complexity of Go. In *Proceedings of the IFIP 9th World Computer Congress on Information Processing (IFIP'83)* (1983), pp. 413–417.

[20] Savitch, W. J. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences 4*, 2 (1970), 177–192.

[21] Selman, A., Mei-Rui, X., and Book, R. Positive relativizations of complexity classes. *SIAM journal on computing* (1983), 565–579.

[22] Seta, T. The complexities of puzzles, cross sum and their another solution problems (ASP). Senior Thesis, University of Tokyo, 2002.

[23] Stockmeyer, L. J., and Chandra, A. K. Provably difficult combinatorial games. *SIAM Journal on Computing 8*, 2 (May 1979), 151–174.

[24] Stockmeyer, L. J., and Meyer, A. R. Word problem requiring exponential time. In *Proceedings of Fifth Annual ACM Symposium on Theory of Computing* (1973), ACM, pp. 1–9.

[25] Thomassen, C. Hamiltonian paths in squares of infinite locally finite blocks. *Annals of Discrete Mathematics 3* (1978), 269–277.

[26] Tutte, W. T. On hamiltonian circuits. *Journal of the London Mathematical Society 21* (1946), 98–101.

[27] Tuza, Z. Unsolved cobinatorial problems part I. Lecture Series LS-01-1, BRICS, Department of Computer Science, University of Aarhus, May 2001.

[28] Ueda, N., and Nagao, T. NP-completeness results for NONOGRAM via parsimonious reductions. Tech. Rep. TR96-0008, Tokyo Institute of Technology, May 1996.

[29] Valiant, L. G. The complexity of computing the permanent. *Theoretical Computer Science 8* (1979), 189–201.

[30] Valiant, L. G. The complexity of enumeration and reliability problems. *SIAM Jurnal on Computing 8* (1979), 410–421.

[31] Yato, T. Complexity and comleteness of finding another solution and its application to puzzles. Master Thesis, University of Tokyo, 2003.

[32] Yato, T., and Seta, T. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E86-A*, 5 (2003), 1052–1060.

[33] Yato, T., Seta, T., and Ito, T. Finding yozume of generalized tsume-shogi is exptime-complete. Tech. Rep. 21, IEICE, June 2004.

[34] Yokota, M., Tsukiji, T., Kitagawa, T., Morohashi, G., and Iwata, S. Exptime-completeness of generalized tsume-shogi (in japanese). *Transactions of the IEICE J84-D-I*, 3 (2001), 239–246.

[35] PlanetMath.org encyclopedia. URL: `http://planetmath.org/encyclopedia/`.