COMPLEXITY AND COMPLETENESS OF FINDING ANOTHER SOLUTION AND ITS APPLICATION TO PUZZLES

別解問題の計算論的複雑さと完全性およびパズルへの応用

by

Takayuki YATO

八登 崇之

A Master Thesis

修士論文

Submitted to

the Graduate School of Science

The University of Tokyo

on January 2003

in Partial Fulfillment of the Requirements

for the Degree of Master of Science
in Information Science

Thesis Supervisor: Hiroshi IMAI　今井 浩
Professor of Information Science

ABSTRACT

The Another Solution Problem (ASP) of a problem Π is the following problem: for a given instance x of Π and a solution s to it, find a solution to x other than s. The notion of ASP as a new class of problems was first introduced by Ueda and Nagao. They also pointed out that polynomial-time parsimonious reductions which allow polynomial-time transformation of solutions can derive the NP-completeness of ASP of a certain problem from that of ASP of another. They used this property to show the NP-completeness of ASP of Nonogram, a sort of puzzle. Following it, Seta considered the problem to find another solution when n solutions are given. (We call the problem n-ASP.) He proved the NP-completeness of n-ASP of some problems, including Cross Sum, for any n.

In this thesis we establish a rigid formalization of n-ASPs to investigate their characteristics more clearly. In particular we introduce ASP-completeness, the completeness with respect to the reductions satisfying the properties mentioned above, and show that ASP-completeness of a problem implies NP-completeness of n-ASP of the problem for all n. Moreover we research the relation between ASPs and other versions of problems, such as counting problems and enumeration problems, and show the equivalence of the class of problems which allow enumerations of solutions in polynomial time and the class of problems of which n-ASP is solvable in polynomial time.

As Ueda and Nagao pointed out, the complexity of ASPs has a relation with the difficulty of designing puzzles. We prove the ASP-completeness of three popular puzzles: Slither Link, Number Place and Fillomino. The ASP-completeness of Slither Link is shown via a reduction from the Hamiltonian circuit problem for restricted graphs, that of Number Place is from the problem of Latin square completion, and that of Fillomino is from planar 3SAT. Since ASP-completeness implies NP-completeness as is mentioned above, these results can be regarded as new results of NP-completeness proof of puzzles.

論文要旨

問題 Π に対する別解問題 (ASP) というのは、Π のインスタンス x とそれに対する 1 つの解 s が与えられた時に、x の s 以外の解を求める問題のことである。ASP を新しい問題クラスととらえたのは Ueda と Nagao が最初であり、彼らはさらに、多項式時間 parsimonious 還元（解の間に 1 対 1 対応が存在する）で対応する解への変換も多項式時間で行えるものを用いてある問題の ASP の NP 完全性から別の問題の ASP の NP 完全性が導出できることを指摘した。そして彼らはこの性質を利用してパズルの一種である「ののぐらむ」の ASP の NP 完全性を示した。続いて Seta は n 個の解が与えられた時にもう 1 つの解を求める問題（ここでは n-ASP と呼ぶ）について考察した。そして「カックロ」などのいくつかの問題の n-ASP が任意の n について NP 完全であることを示した。

本論文では、n-ASP に対して、その性質の研究をより明確に行うための厳格な定式化を行う。特に、上述の性質を満たす還元に関する完全性である ASP 完全性について考察し、ASP 完全問題について、その全ての n についての n-ASP が NP 完全となることを示す。さらに、ASP と他の種（数え上げ問題や列挙問題）との関係について調べ、多項式時間で列挙が可能な問題のクラスと多項式時間で n-ASP が解ける問題のクラスが一致することを示す。

Ueda と Nagao が指摘するように、ASP の計算量はパズルの作成の難しさと関係がある。本論文では 3 つの有名なパズル、スリザーリンクとナンバープレース（数独）とフィルオミノについてその ASP 完全性を示す。スリザーリンクの ASP 完全性は制限されたグラフに対するハミルトン閉路問題から、ナンバープレースはラテン方陣完成問題から、フィルオミノは平面的 3SAT からの還元を用いて示される。上で述べたように、ASP 完全な問題は NP 完全でもあるので、これらの結果は、パズルの NP 完全性の証明に関する新しい結果といえる。

Acknowledgements

I am grateful to my supervisor Hiroshi Imai for his advice and encouragement. I would also like to thank Takahiro Seta, the co-author of the paper related to this thesis. I further thank Yasuhito Asano, Tsuyoshi Ito, and all other members of Imai Laboratory for their helpful advice and suggestion.

Contents

Chapter 1

Introduction

1.1 Background

1.1.1 Another Solution Problem

Since the invention of **NP**-completeness by Karp and Cook in early 1970's, the theory of computational complexity have played an important role in analyzing the difficulty of problems. In particular, the theory largely facilitated the proof that a certain problem is really difficult to some extent. For example, once a problem is proved to be **NP**-complete, the fact means that the problem cannot be solved efficiently (i. e., in polynomial time) by deterministic algorithms unless $\mathbf{P} = \mathbf{NP}$, which is not believed to happen, and thus tells people that they should consider some alternative measures, such as approximation and randomization.

In some cases special type of problems were studied: when we are given not only an instance of the problem but also a solution, find a solution other than the given one. This type of problem is of interest because there is possibility that a solution given in advance will make the problem easier. To this question some negative results are known. One example is the following problem:

> Hamiltonian Circuit Problem:
> *Input:* an undirected graph G.
> *Output:* does G have a Hamiltonian circuit?

This is a famous **NP**-complete problem. Moreover, Papadimitriou noted in his book [14] that the problem to decide whether a given graph G has a Hamiltonian circuit different from a given one is also **NP**-complete. Another example is the problem of partial Latin square completion. Colbourn [4] proved that deciding the existence

of another solution, as well as this problem itself, is **NP**-complete. (See Section 3.2 for the detail of that problem.)

On the other hand there is a positive result about the question. An undirected graph G is called *cubic graph* if the degree of all vertices is exactly three. Hamiltonian circuit problem is known to remain **NP**-complete if input graphs are restricted to cubic graphs [9]. However, Smith's theorem, which states that each edge in a cubic graph G is in an even number of Hamiltonian circuit of G, implies that

> if a cubic graph has a Hamiltonian circuit, it has a second one.

That means the problem to decide the existence of another Hamiltonian circuit when a cubic graph and its Hamiltonian circuit is given is trivial, because the answer is always 'yes'. (However it is also known that to *find* another Hamiltonian circuit is not easy. See [14].)

Ueda and Nagao introduced in their paper [22] in 1996 the notion of regarding problems to determine the existence of another solution as a new class of problems, and named such problems "Another Solution Problem (ASP)." Moreover, they also pointed out that parsimonious reductions which allow transformation of solutions in polynomial time (what we call *ASP reduction* in this thesis) is useful to study the complexity of ASPs. (A *parsimonious reduction* is a reduction which preserves the number of solutions, and is used to show the #**P**-completeness of counting problems.) It is because when there is an ASP reduction from a problem Π_1 to another problem Π_2, then the **NP**-completeness of ASP of Π_2 can be derived from that of ASP of Π_1. Following this approach they proved the **NP**-completeness of ASP of Nonogram (a kind of puzzle) by showing such a reduction from 3-dimensional matching (3DM) to Nonogram.

Following that Seta [21, 20] considered the problem to find another solution when n solutions are given. (We call the problem n-ASP.) He noted that an ASP reduction from a problem Π to 1-ASP of Π itself can be extended to an ASP reduction from Π to n-ASP of Π. Using this result he showed the **NP**-completeness of n-ASP of some problems, including a basic logic problem 1-in-3 SAT and a puzzle called Cross Sum, for any nonnegative integer n.

The reason that in the history of ASP some puzzles are mentioned is that ASP has a close relation to puzzles. Before we state the relation, let us review the computational complexity of puzzles.

Game	Complexity	Reference
Shogi	**EXPTIME**-complete	[1]
Chess	**EXPTIME**-complete	[8]
Go	**EXPTIME**-complete	[17]
Checkers	**EXPTIME**-complete	[18]
Gomoku	**PSPACE**-complete	[16]
Othello	**PSPACE**-complete	[11]

Table 1.1: Complexity of two-player games

1.1.2 Computational complexity of puzzles

Today a wide variety of puzzles are played all over the world. One of the sources of their fun is their difficulty. In fact, the reason that two-player games like go, shogi and chess keep their popularity over centuries is that these games cannot be easily analyzed, even with the aid of computers.

Many results are known about the computational complexity about games and puzzles. First, let us consider two-player perfect-information games — where both players know all information about the game-play. About such games, the problem to determine which player wins at a given game state under ideal play can be theoretically computed. The complexity of this problem about several popular games (with arbitrarily large boards) are shown in Table 1.1. It is because the problem is unrealistically difficult that the fun of these games is not lost.

Next consider one-player puzzles. In this case, problems of interest are (i) whether a given problem is solvable, and (ii) to find the solution with the fewest moves (about puzzles regarding motion). Many complexity results are known also in this field. In addition "offline" version of some one-player games which are not perfect-information in themselves. For example, the problem to determine whether a given configuration of Minesweeper with some cells uncovered (and marked with the number of adjacent mines) can be realized by some distribution of mines is proved to be **NP**-complete [12]. Some results on complexity of puzzles are shown in Table 1.2. The survey by Demaine [5] refers to more results on the computational complexity of games and puzzles.

In this thesis we focus on a certain type of puzzles, *pencil puzzles*. Pencil puzzles (or pencil-and-paper puzzles) are those offered as some figure on the paper and solved by drawing on the figure with a pencil. Here is a list of pencil puzzles which are very popular in Japan:

Game	Complexity	Reference
15-puzzle (optimal play)	**NP**-complete	[15]
Cryptarithm	**NP**-complete	[7]
Peg Solitaire	**NP**-complete	[23]
Clickomania (Same-Game)	**NP**-complete	[3]
"Offline" Minesweeper	**NP**-complete	[12]
"Offline" Tetris	**NP**-complete	[6]

Table 1.2: Complexity of one-player games.

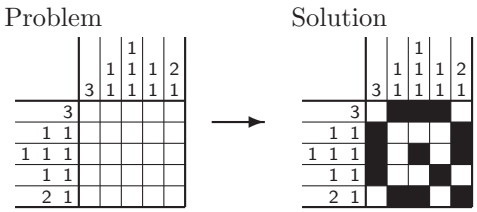

Figure 1.1: An example of Nonogram.

- Nonogram (Griddler, Paint-by-numbers)
- Slither Link
- Cross Sum (*Kakkuro*)
- Number Place (*Sudoku*)
- Nurikabe
- Heyawake

Pencil puzzles have the property that "to solve the problem is hard, but to verify the solution is easy". For example, Nonogram is the puzzle where you paint some cells of a grid black following the given hint and restore a picture. An example of Nonogram is shown in Fig. 1.1. The hint at the bottom row "2, 1" means that black cells in that row must appear (when searched from left to right) first as a length two chunk, then as a lone cell. To solve this puzzle needs some brain work, but to show that the obtained solution is correct is trivial. (For the detail of Nonogram and its complexity see [22].)

In a contrast to the vast amount of literature of computational complexity of games and puzzles, few results about pencil puzzles are known. (One reason for this is that many of them are originated in Japan and known only in Japan.) Before the author started the study the result in 1996 by Ueda and Nagao [22] that states **NP**-completeness of Nonogram was the only such result. For the present, the following results are known:

- The author proved **NP**-completeness of Slither Link in 2000 [26].
- Seta proved **NP**-completeness of Cross Sum in 2002 [20].
- Friedman proved **NP**-completeness of three (less popular) puzzles introduced on *Nikoli*, a famous puzzle magazine in Japan.

In this thesis we consider the ASP of puzzles. In the next section we state the relation between puzzles and ASPs.

1.1.3 Relation between puzzles and ASPs

Ueda and Nagao [22] pointed out that ASP has a close relation to designing puzzles in the following respect: For many sorts of puzzles, in particular pencil puzzles, the solution of a problem is strongly desired to be unique. Thus puzzle designers have to check whether the designed problem has no solutions other than the intended one. This work is exactly an instance of ASPs.

We point out that this aspect is particularly important when thinking of automatic generation of puzzle problems by computers. Designing puzzles is, as well as solving them, complicated brain work and relies heavily on human intuition and experience. Thus puzzles of high quality are not easy to obtain and thus are precious. Therefore it would be nice if a large number of puzzles could be generated automatically by computers. (Saito [19] treated the automatic generation of Slither Link.) One way to make a puzzle problem is as follows.

1° First design a figure of solution. (In the case of Nonogram, it is the black-and-white pattern of the grid.)

2° Make a problem which 'satisfies' the designed solution. Usually such a problem is easily derived from the solution. (Numbers telling the length of black chunks in the case of Nonogram,)

3° Solve the constructed problem and check if it has a solution other than the designed one. If not, the construction is completed.

When humans design puzzles, the method shown above is practical in the case of puzzles "painting a picture" like Nonogram (because the quality of the resulting picture is of great importance in such puzzles), but otherwise this method is usually not taken. Instead, a designer constructs a problem and its solution at the same time with care to preserve the uniqueness of solution by using heuristics obtained from experience. However, this method depends on human experience and not suitable for computers, and thus computers must resort to the previous "generate-and-test" method. In this situation, if the ASP of the puzzle is polynomial-time solvable and we have an efficient algorithm to solve it, then we can generate a vast number of problems. Of course the quality of the generated problem must be also considered, but we can expect that the possibility of massive generation of problems at least helps us obtain good puzzles. Investigating the ASP of puzzles is thus important also in the respect of automatic generation of problems.

In the last of this section, we list the results on the ASP of puzzles known at present.

- Ueda and Nagao [22] proved the **NP**-completeness of the 1-ASP of Nonogram.
- Seta [20] proved the **NP**-completeness of the n-ASP of Nonogram and Cross Sum for any nonnegative integer n.

1.2 Our Contribution

In this thesis we provide a rigid formalization of n-ASP in order to enable more strict argument in studying the characteristics of n-ASPs and ASP reductions. In particular we consider the completeness with respect to ASP reduction (we call it *ASP-completeness*), and prove that for any ASP-complete problem its n-ASP is **NP**-complete (as a decision problem) for any nonnegative integer n.

Moreover we investigated the relation between ASPs and other versions of problems, such as counting problems and enumeration problems. There we showed that the class of problems of which n-ASP is solvable in polynomial time is equal to the class of problems which allow enumerations of solutions in polynomial time.

As a contribution to the field of combinatorial puzzles, we prove the ASP-completeness of three popular puzzles: Slither Link, Number Place, and Fillomino. Since ASP-completeness implies **NP**-completeness these results also add new items to the list of **NP**-complete puzzles. Note also that ASP-completeness implies #**P**-completeness of counting the solutions.

1.3 Organization of This Thesis

The organization of this thesis is as follows. In Chapter 2, we review some basic concept of computational complexity and give the definition of n-ASP and ASP reduction under our formalization. Then we introduce ASP-completeness as a new concept and discuss its property. Moreover we investigate the relation between ASPs and some other versions of problems. In Chapter 3, we prove ASP-completeness of three popular puzzles: Slither Link, Number Place and Fillomino. For each of these puzzles, we first state the formal rule and then give the proof of ASP-completeness. In Chapter 4, we summarize our results and conclude the thesis with future work.

Chapter 2

Another Solution Problem (ASP)

2.1 Preliminaries

In this section we state a theory of Another Solution Problem (ASP). First of all we use the following existing formalization of function problems in order to facilitate the argument:

Definition 2.1 Let Π be a triple (D, S, σ) satisfying the following:

- D is the set of the instances of a problem.
- S is a set which includes all that can be a solution. (S may include extra things. When we think on the basis of Turing machines, S may be the set of all the strings.)
- σ is a mapping from D to 2^S. For an instance $x \in D$, $\sigma(x)$ ($\subseteq S$) is called the *solution set* to x, and an element of $\sigma(x)$ is called an *solution* to x.

Then the problem which finds a solution to an instance $x \in D$ (or simply Π itself) is called a *function problem*.

Under this formalization, the class **FNP** is described as follows:

Definition 2.2 **FNP** is a class consisting of function problems $\Pi = (D, S, \sigma)$ such that the following holds:

- There exists a polynomial p such that $|s| \leq p(|x|)$ holds for any $x \in D$ and any $s \in \sigma(x)$.
- For any $x \in D$ and $y \in S$, the proposition $y \in \sigma(x)$ can be decided in polynomial time.

Definition 2.3 Let $\Pi = (D, S, \sigma)$ be a function problem. $Y = \{x \in D \mid \sigma(x) \neq \emptyset\}$ $\Pi_{\mathrm{d}} = (D, Y)$ (that is, the pair consist of the set of all the instances and the set of all the yes-instances) is called the *decision problem induced by* Π or simply the decision problem of Π.

Note that it follows from the characteristic of **NP** that for any decision problem $\hat{\Pi} \in \mathbf{NP}$ there exists a function problem $\Pi \in \mathbf{FNP}$ satisfying $\Pi_{\mathrm{d}} = \hat{\Pi}$. (Of course, $\Pi \in \mathbf{FNP}$ implies $\Pi_{\mathrm{d}} \in \mathbf{NP}$.)

2.2 Formalization of ASP

Under the framework described in the previous section, we formalize ASPs as follows:

Definition 2.4 Let $\Pi = (D, S, \sigma)$ be a function problem. For Π and a nonnegative integer n, we call the function problem $\Pi_{[n]} = (D_{[n]}, S, \sigma_{[n]})$ constructed as follows the *n-another solution problem* of Π (or shortly n-ASP).

$$D_{[n]} = \{(x, S_x) \mid S_x \subseteq \sigma(x),\ |S_x| = n\}, \tag{2.1}$$

$$\sigma_{[n]}(x, S_x) = \sigma(x) - S_x \tag{2.2}$$

We call the decision problem of an n-another solution problem *n-another solution decision problem*.

We use the notation Π_{d} and $\Pi_{[n]}$ as the meaning mentioned above throughout this paper.

The next definition is polynomial-time ASP reduction. This is parsimonious reduction which allows polynomial-time transformation of solutions and is introduced by Ueda and Nagao [22]. (Recall that parsimonious reductions are used for proving #**P**-completeness of counting problems.)

Definition 2.5 Let $\Pi_1 = (D_1, S_1, \sigma_1)$ and $\Pi_2 = (D_2, S_2, \sigma_2)$ be function problems. We call the pair $\varphi = (\varphi_{\mathrm{D}}, \varphi_{\mathrm{S}})$ satisfying the following *polynomial-time ASP reduction* from Π_1 to Π_2:

- φ_{D} is a polynomial-time computable mapping from D_1 to D_2.
- For any $x \in D_1$, φ_{S} is a polynomial-time computable *bijection* from $\sigma_1(x)$ to $\sigma_2(\varphi_{\mathrm{D}}(x))$.

If there is a polynomial-time ASP reduction from Π_1 to Π_2, Π_1 is called to be *polynomial-time ASP reducible* to Π_2 (denoted by $\Pi_1 \preceq_{\mathrm{ASP}} \Pi_2$).

By definition, all the ASP reductions are parsimonious. Although the converse does not always hold, many parsimonious reductions involve concrete transformation of solutions and thus are ASP-reductions.

The relation $\preceq_{\mathrm{ASP}}$ is transitive, like other reducibility relations. If $\Pi_1 \preceq_{\mathrm{ASP}} \Pi_2$, then $\Pi_{1\mathrm{d}}$ is polynomial-time reducible (as a decision problem) to $\Pi_{2\mathrm{d}}$.

Seta stated some fundamental characters about ASP reductions in [20]. Since he omits their proof, we give one along the framework.

First, the relation of ASP reducibility is invariant with respect to "taking an ASP" operation.

Proposition 2.1 *Let* Π *and* Π' *be function problems. If* $\Pi \preceq_{\mathrm{ASP}} \Pi'$*, then* $\Pi_{[n]} \preceq_{\mathrm{ASP}} {\Pi'}_{[n]}$ *for any nonnegative integer* n*.*

Proof Let $\Pi = (D, S, \sigma)$ and $\Pi' = (D', S', \sigma')$, and let $\varphi = (\varphi_{\mathrm{D}}, \varphi_{\mathrm{S}})$ be a polynomial-time ASP reduction from Π to Π'.

Let $(x, \{s_1, \ldots, s_n\})$ $(\in D_{[n]})$ be an instance of $\Pi_{[n]}$, and let T denote its solution set $\sigma_{[n]}(x, \{s_1, \ldots, s_n\})$. By the definition of ASP, x is an instance of Π (i.e. $x \in D$), and its solution set $\sigma(x)$ equals $\{s_1, \ldots, s_n\} \cup T$.

We can construct a polynomial-time ASP reduction $\hat{\varphi} = (\hat{\varphi}_{\mathrm{D}}, \hat{\varphi}_{\mathrm{S}})$ from $\Pi_{[n]}$ to ${\Pi'}_{[n]}$ as follows: for an instance $(x, \{s_1, \ldots, s_n\})$ of $\Pi_{[n]}$ define $\hat{\varphi}_{\mathrm{D}}(x, \{s_1, \ldots, s_n\})$ to be $(\varphi_{\mathrm{D}}(x), \{\varphi_{\mathrm{S}}(s_1), \ldots, \varphi_{\mathrm{S}}(s_n)\})$ (let it be denoted by y), and $\hat{\varphi}_{\mathrm{S}}$ to be φ_{S} restricted onto T. Since $\sigma'(\varphi_{\mathrm{D}}(x)) = \{\varphi_{\mathrm{S}}(s_1), \ldots, \varphi_{\mathrm{S}}(s_n)\} \cup \varphi_{\mathrm{S}}(()T)$, it follows that ${\sigma'}_{[n]}(y) = \varphi_{\mathrm{S}}(()T)$. That is, $(\hat{\varphi}_{\mathrm{D}}, \hat{\varphi}_{\mathrm{S}})$ is in fact an ASP reduction. The polynomial-time computability of $\hat{\varphi}_{\mathrm{D}}$ and $\hat{\varphi}_{\mathrm{S}}$ is derived from that of φ_{D} and φ_{S}. ∎

Second, it holds that n-ASP of m-ASP is $(m+n)$-ASP (of the original problem).

Proposition 2.2 *For any function problem* Π *and nonnegative integers* m, n*,* $(\Pi_{[m]})_{[n]} \preceq_{\mathrm{ASP}} \Pi_{[m+n]}$*.*

Proof Let $\Pi = (D, S, \sigma)$. An instance $\tilde{x}$ of $(\Pi_{[m]})_{[n]}$ is of the form

$$((x, \{s_1, \ldots, s_m\}), \{t_1, \ldots, t_n\}) \quad (x \in D;\ s_1, \ldots, s_m, t_1, \ldots, t_n \in \sigma(x)).$$

For this we set $\varphi_{\mathrm{D}}(\tilde{x})$ to be $(x, \{s_1, \ldots, s_m, t_1, \ldots, t_n\})$. Then the solution set of $\tilde{x}$ is equal to that of $\varphi_{\mathrm{D}}(\tilde{x})$, and thus φ_{S} can be set to be the identity function. ∎

Combining the two propositions, we obtain the following important result.

Theorem 2.3 *Let* Π *be a function problem. If* $\Pi \preceq_{\mathrm{ASP}} \Pi_{[1]}$*, then for any nonnegative integer* n $\Pi_{[n]} \preceq_{\mathrm{ASP}} \Pi_{[n+1]}$*. (Therefore* $\Pi \preceq_{\mathrm{ASP}} \Pi_{[n]}$*,)*

Proof From $\Pi \preceq_{\mathrm{ASP}} \Pi_{[1]}$ and Proposition 2.1 we obtain $\Pi_{[n]} \preceq_{\mathrm{ASP}} (\Pi_{[1]})_{[n]}$, and from Proposition 2.2 we obtain $(\Pi_{[1]})_{[n]} \preceq_{\mathrm{ASP}} \Pi_{[n+1]}$. Thus the theorem holds because of the transitivity of $\preceq_{\mathrm{ASP}}$. ■

2.3 ASP-completeness

2.3.1 Definitions and Fundamental Results

ASP-completeness is completeness with respect to ASP reductions, and defined as follows:

Definition 2.6 A function problem Π is *ASP-complete* if and only if $\Pi \in \mathbf{FNP}$, and $\Pi' \preceq_{\mathrm{ASP}} \Pi$ for any $\Pi' \in \mathbf{FNP}$.

Proposition 2.4 *Let* Π *and* Π' *be function problems. If* Π *is ASP-complete,* $\Pi' \in \mathbf{FNP}$ *and* $\Pi \preceq_{\mathrm{ASP}} \Pi'$*, then* Π' *is ASP-complete.*

Our first ASP-complete problem is SAT. Here SAT represents the function problem of satisfiability. (The decision problem is denoted by $\mathrm{SAT}_{\mathrm{d}}$.)

Theorem 2.5 SAT *is ASP-complete.*

Proof Cook's reduction is an ASP reduction. ■

An important property of ASP-completeness is that it implies the **NP**-completeness of n-ASPs for any n. We first show this property as to SAT.

Theorem 2.6 *1.* $\mathrm{SAT} \preceq_{\mathrm{ASP}} \mathrm{SAT}_{[1]}$.

2. For any nonnegative integer n, $\mathrm{SAT}_{[n]\mathrm{d}}$ *is* **NP***-complete.*

Proof 1. We construct a polynomial-time ASP reduction $\varphi = (\varphi_{\mathrm{D}}, \varphi_{\mathrm{S}})$ from SAT to $\mathrm{SAT}_{[1]}$.

For a given instance of SAT ψ (a CNF formula), we construct a CNF formula ψ' as follows:

- We introduce a new variable w, and for each clause $l_1 \vee \cdots \vee l_r$ in ψ make a clause $l_1 \vee \cdots \vee l_r \vee w$ and add it to ψ'.
- For each variable x we add a clause $x \vee \bar{w}$ (CNF of $w \Rightarrow x_i$) to ψ'.

Then define $\varphi_{\mathrm{D}}(\psi)$ to be $(\psi', \{g\})$, where g is the assignment in which all variables are true. For a solution h to ψ, let $\varphi_{\mathrm{S}}(h)$ be h with assignment $w = \text{false}$ added.

2. From 1 and Theorem 2.3, we obtain SAT $\preceq_{\mathrm{ASP}}$ $\mathrm{SAT}_{[n]}$ for any nonnegative integer n. Thus $\mathrm{SAT}_{\mathrm{d}}$ is polynomial-time reducible to $\mathrm{SAT}_{[n]\mathrm{d}}$ and the theorem holds. ■

Using this result and Proposition 2.1 the property is shown for all ASP-complete problems.

Theorem 2.7 *For any ASP-complete function problem* Π *and any nonnegative integer* n, $\Pi_{[n]\mathrm{d}}$ *is* **NP***-complete.*

Proof Since Π is ASP-complete, SAT $\preceq_{\mathrm{ASP}}$ Π holds, thus it follows from Proposition 2.1 that $\mathrm{SAT}_{[n]} \preceq_{\mathrm{ASP}} \Pi_{[n]}$. This implies that $\mathrm{SAT}_{[n]\mathrm{d}} \preceq \Pi_{[n]\mathrm{d}}$. Thus $\Pi_{[n]\mathrm{d}}$ is shown to be **NP**-complete from Theorem 2.6. ■

Here follow the proofs of ASP-completeness of some logic problems, which are used in the later sections.

Theorem 2.8 3SAT *is ASP-complete.*

Proof We show a polynomial-time ASP reduction from SAT. Let ψ be a CNF formula given as an instance of SAT. We modify ψ as follows:

- Introduce new variables t_1, t_2, t_3 and add these 7 clauses: $t_1 \vee t_2 \vee t_3$, $t_1 \vee t_2 \vee \bar{t}_3$, $t_1 \vee \bar{t}_2 \vee t_3$, $t_1 \vee \bar{t}_2 \vee \bar{t}_3$, $\bar{t}_1 \vee t_2 \vee t_3$, $\bar{t}_1 \vee t_2 \vee \bar{t}_3$, $\bar{t}_1 \vee \bar{t}_2 \vee t_3$. (All of t_1, t_2, t_3 must be true.)
- Add $\bar{t}_1$ to each clause with two literals, and add $\bar{t}_1$ and $\bar{t}_2$ to each clause with one literal,
- Divide each clause with 4 or more literals $l_1 \vee l_2 \vee l_3 \vee \cdots \vee l_r$ as follows: $l_1 \vee l_2 \vee \bar{d}$, $\bar{l}_1 \vee d$, $\bar{l}_2 \vee d$ (these are CNF of $(l_1 \vee l_2) \equiv d$), $d \vee l_3 \vee \cdots \vee l_r$, where d is a new variable. While the last clause has 4 or more literals, repeat the division.

Let ψ' denote the modified formula. Then ψ' is satisfiable if and only if ψ is satisfiable. Moreover, for each truth assignment h satisfying ψ, there is only one assignment that can be obtained by extending h and satisfy ψ'. ■

1-in-3 SAT is the following problem: given a 3-CNF formula, find a truth assignment in which each clause contains exactly one true literal.

Theorem 2.9 *1-in-3 SAT is ASP-complete.*

Proof Seta [21] constructs a polynomial-time ASP reduction (in terminology of this paper) from 3SAT to 1-in-3 SAT. ASP-completeness of this problem follows from the result and ASP-completeness of 3SAT. ■

2.4 Relations Between ASP and Other Versions of Problems

2.4.1 Relation between ASP-completeness and #P-completeness

As is stated in the previous section, all the ASP reductions are parsimonious. which means that if a function problem is ASP-complete the counting version of the problem is #**P**-complete. Here the first question of interest related to the fact is: whether does the converse hold or not? The answer is no.

Proposition 2.10 *There is a function problem which is not ASP-complete but its counting version is #**P**-complete.*

Proof |As is known from Theorem 2.7, the decision problem of an ASP-complete problem must be **NP**-complete. Therefore the existence of a #**P**-complete problem of which the decision problem is not **NP**-complete proves the proposition. Consider the following problem, Bipartite Matching Problem: given a bipartite graph G, find a perfect matching of G. A well-known fact states that although the decision version of this problem (determining the existence of a perfect matching) is polynomial-time solvable, the counting version (finding the number of perfect matchings) is equivalent to calculating the permanent of a 0-1 matrix, which is proved to be #**P**-complete by Valiant [25]. ■

Now that we know Bipartite Matching Problem is not ASP-complete, then the next question of interest is whether the n-ASP of this problem is **NP**-complete (for some n). The answer is negative again.

Theorem 2.11 *The n-ASP of Bipartite Matching Problem is solvable in polynomial time for any nonnegative integer n.*

Proof Uno [24] presents an algorithm for enumerating all the perfect matchings in a given bipartite graph within $\mathcal{O}(|V|)$ time per a matching, where $|V|$ is the number of vertices of the graph. The n-ASP of Bipartite Matching Problem can be solved by using this algorithm in the following way:

1° For a given input bipartite graph G, generate $n+1$ perfect matchings in G by using the enumeration algorithm.

2° Among the generated matchings there must be one which is not equal to any of n matchings which are given as input. Output such one.

This procedure can be done in polynomial time with respect to the input length. Note that the input contains n matchings and thus has the length proportional to n. ■

This result suggests that difficulty of counting problems does not reflect on that of ASPs.

2.4.2 Relation between ASPs and enumeration problems

The proof of Theorem 2.11 uses the fact that the enumeration of perfect matchings can be performed efficiently.

This result can be generalized to the following relation berween ASP (as a function problem) and enumeration problem:

Theorem 2.12 *Let $\Pi = (D, S, \sigma)$ be a function problem in* **FNP**, *and suppose that there is an algorithm which enumerates all the solutions to a given instance x of Π and which takes only polynomial time (with respect to $|x|$) to output each solution. Then $\Pi_{[n]}$ is solvable in polynomial time for any nonnegative integer n.*

Proof Since $\Pi \in$ **FNP**, there exists a polynomial p such that $|s| \leq p(|x|)$ holds for any $x \in D$ and any $s \in \sigma(x)$.

Let M be a DTM which enumerates all the solutions in $p_1(|x|)$ time per a solution, where p_1 is a polynomial and $|x|$ is the input length. Just like Theorem 2.11, we can construct a DTM M' which solves an instance of $\Pi_{[n]}$ (denoted by (x, T), where $T \subseteq \sigma(x)$) by using M to output $n+1$ solutions to x and finding a solution which does not belong to T. This takes at most $(n+1)p_2(|x|)$ time (p_2 is a polynomial determined by p_1 and p). On the other hand, the input length is at least $|x| + n$. Therefore M' is polynomial time bounded. ■

The following weaker form of the converse of Theorem 2.12 also holds:

Theorem 2.13 *Let $\Pi = (D, S, \sigma)$ be a function problem in* **FNP** *such that $\Pi_{[n]}$ is solvable in polynomial time for any nonnegative integer n.*

Then there is an algorithm which enumerates all the solutions to a given instance x of Π and which takes only polynomial time with respect to $|x|$ and K to output first K solutions.

Proof For each n let M_n be a $p_3(|x|)$-time DTM for $\Pi_{[n]}$. We can construct a DTM M' which enumerates all the solutions of x ($\in D$) (an instance of Π) as follows:

1° Let k be 0 and T be the empty set.

2° Run M_k with the input (x, T). If the answer is 'no' (no other solution exists), halt.

3° Otherwise the answer is a solution s which is not in T. Output s.

4° Add s to T and add 1 to k, and go to 2°.

The length of the input to M_k is at most $|x| + k \cdot p(|x|)$ (where p is the same as in Theorem 2.12). Thus the total running time needed to output first K solutions is at most

$$\sum_{k=0}^{K-1} p_3(|x| + k \cdot p(|x|)) \leq \sum_{k=0}^{K-1} p_4(|x|, k) \leq K \cdot p_4(|x|, K),$$

(p_4 is a polynomial determined by p_3 and p). Therefore M' satisfied the given condition. ■

2.4.3 Relation between ASP-completeness and NP-completeness

We proved that ASP-completeness implies **NP**-completeness of n-ASP for all n. Then the following important problem arises:

Problem 2.1 *Let Π be a function problem such that $\Pi_{[n]\mathrm{d}}$ is **NP**-complete for any nonnegative integer n. Then can Π be said to be ASP-complete? (Does the converse of Theorem 2.7 hold or not?)*

Before the argument about this ptoblem, we note that **NP**-completeness for *infinitely many* n is necessary.

Proposition 2.14 *For any nonnegative integer k, there is a function problem Π which satisfes the following:*

> *Although $\Pi_{\mathrm{d}}, \Pi_{[1]\mathrm{d}}, \ldots, \Pi_{[k-1]\mathrm{d}}$ are all **NP**-complete, $\Pi_{[k]\mathrm{d}}$ is trivial (the answer is always 'yes').*

Proof Let $\Pi = (D, S, \sigma)$ be a function problem such that $\Pi_{[n]\mathrm{d}}$ is **NP**-complete for any nonnegative integer n (SAT for example). Then we construct a function problem $\hat{\Pi} = (D, \hat{S}, \hat{\sigma})$ for given k as follows:

- $\hat{S} = S \sqcup \{X \mid X \subseteq S, |X| = k\}$. ($\sqcup$ means disjoint union.)
- For any $x \in D$, Let $M(x)$ be $\{X \mid X \subseteq \sigma(x), |X| = k\}$ (the set of all the k-sets consisting of elements of $\sigma(x)$) and define $\hat{\sigma}(x)$ to be $\sigma(x) \sqcup M(x)$.

We prove that $\hat{\Pi}$ satisfies the condition given in the proposition.

[The proof of **NP**-completeness of $\hat{\Pi}_{[n]\mathrm{d}}$ for each n such that $0 \leq n < k$]

Since the condition of Π implies the **NP**-completeness of $\Pi_{[n]\mathrm{d}}$, we construct polynomial-time transformation φ from $\Pi_{[n]\mathrm{d}}$ to $\hat{\Pi}_{[n]\mathrm{d}}$ ($\hat{\Pi}_{[n]\mathrm{d}} \in \mathbf{NP}$ is obvious). To do so we can simply let φ be the identity. The reason is as follows. Let us consider an instance of $\Pi_{[n]\mathrm{d}}$ (which is also an instance of $\hat{\Pi}_{[n]\mathrm{d}}$) and denote it by $\hat{x} = (x, \{s_1, \ldots, s_n\})$ $(\in D_{[n]})$.

- If the answer of $\hat{x}$ as to $\Pi_{[n]\mathrm{d}}$ is 'no', then $\sigma_{[n]}(\hat{x}) = \emptyset$ and therefore $\sigma(x) = \{s_1, \ldots, s_n\}$. However, since $|\sigma(x)| = n < k$, k-sets of elements of $\sigma(x)$ does not exist, thus $\hat{\sigma}(x) = \sigma(x) = \{s_1, \ldots, s_n\}$. As a result $\hat{\sigma}_{[n]}(\hat{x}) = \emptyset$, which means the answer of $\hat{x}$ as to $\hat{\Pi}_{[n]\mathrm{d}}$ is 'no'.

- Conversely, if the answer of $\hat{x}$ as to $\Pi_{[n]\mathrm{d}}$ is 'yes', that is, there exists s such that $s \in \sigma_{[n]}(\hat{x})$, then $s \in \hat{\sigma}_{[n]}(\hat{x})$ also holds (The answer of $\hat{x}$ as to $\hat{\Pi}_{[n]\mathrm{d}}$ is 'yes').

Therefore **NP**-completeness of $\hat{\Pi}_{[n]\mathrm{d}}$ is proven.

[The proof that $\hat{\Pi}_{[k]\mathrm{d}}$ is trivial]

We show that for an instance of $\hat{\Pi}_{[k]\mathrm{d}}$ denoted by $\hat{x} = (x, \{s_1, \ldots, s_k\})$ $(\in D_{[k]})$ the set $\hat{\sigma}(x)$ has an element other than $s_1, \ldots, s_k$.

- If all of $s_1, \ldots, s_k$ $(\in \hat{\sigma}(x))$ belong to $\sigma(x)$, then $\{s_1, \ldots, s_k\} \in M(x) \subseteq \hat{\sigma}(x)$, thus the claim is satisfied.

- Suppose that some of $s_1, \ldots, s_k$ belong $M(x)$ and let one of them be $s_1 = \{t_1, \ldots, t_k\}$ without loss of generality. Then $t_1, \ldots, t_k$ are elements of $\sigma(x)$ (thus elements of $\hat{\sigma}(x)$). Moreover some of them are different from any of $s_2, \ldots, s_k$. Thus the claim is satisfied.

Therefore for any $\hat{x} \in D_{[k]}$ the answer of $\hat{\Pi}_{[k]\mathrm{d}}$ is 'yes'. ■

We have not yet solved the problem, but we conjecture the answer is negative. Here we state a strategy to prove this conjecture.

In the argument below, we use the notion of *one-way function*, a critical concept in modern cryptography.

Definition 2.7 A function $f : A \to B$ is defined to be a *one-way function* if the following hold:

- f is a bijection which is polynomially-balanced (that is, there is a positive integer k such that $|x|^{1/k} \leq |f(x)| \leq |x|^k$ for all $x \in A$).

- $f(x)$ can be computed in polynomial time with respect to $|x|$.
- There exists n_0 such that for all $n > n_0$, all $k > 0$, and all polynomial-time algorithm M,
$$\frac{|\{y \mid |y| = n \text{ and } f(M(y)) = y\}|}{|\{y \mid |y| = n\}|} \leq \frac{1}{n^k},$$
that is, there is no polynomial-time algorithm which correctly computes $f^{-1}(y)$ on a polynomial fraction of the inputs of length n.

Let us denote the satisfiability problem as SAT $= (D, G, \sigma)$, that is, G is the set of assignment vectors. The argument below is the on the assumption that there is a bijection f from G to a certain set H such that f^{-1} is one-way (that is, f is hard to compute).

We slightly modify SAT to SAT+N which can have arbitary number of trivial solutions as follows:

$$\text{SAT+N} = (D^+, G \cup \mathbf{N}, \sigma^+); \quad (\mathbf{N} \text{ is the set of nonnegative integers})$$
$$D^+ = D \times \mathbf{N}; \quad \sigma^+(x, n) = \sigma \cup \{1, \ldots, n\}.$$

That is, SAT+N is the following problem: given a pair of a CNF formula ψ and a nonnegative integer n, return either an assignment which satisfies ψ or a positive integer not greater than n. This problem SAT+N is ASP-complete because there is a trivial ASP reduction from SAT (converting ψ to $(\psi, 0)$).

We extend f to a function $\tilde{f}$ from $(G \cup \mathbf{N})$ to $(H \cup \mathbf{N})$ as follows:

$$\tilde{f}(g) = \begin{cases} f(g) & (g \in G) \\ g & (g \in \mathbf{N}) \end{cases}.$$

It is easily confirmed that $\tilde{f}$ is a bijection and $\tilde{f}^{-1}$ is also one-way.

Now we define a new problem, denoted SAT+N$/f$, as follows:

$$\text{SAT+N}/f = (D^+, H \cup \mathbf{N}, \sigma^\dagger),$$
$$\sigma^\dagger(\psi) = \{\tilde{f}(g) \mid g \in \sigma^+(\psi)\}.$$

Then there is a trivial reduction from SAT+N to SAT+N$/f$. This reduction is parsimonious, but not a polynomial-time ASP-reduction, since the one-way property of $\tilde{f}$ obstructs polynomial-time transformation of solutions. However we can prove that n-ASP of SAT+N$/f$ is **NP**-complete.

Proposition 2.15 *For any nonnegative integer k,* SAT+N$/f_{[k]\mathrm{d}}$ *is* **NP***-complete.*

Proof The membership in **NP** follows from the polynomial-time computatility of $\tilde{f}^{-1}$.

We construct a polynomial-time transformation φ from $\mathrm{SAT}_{\mathrm{d}}$ to $\mathrm{SAT{+}N}/f_{[k]\mathrm{d}}$ as follows:

$$\varphi(\psi) = ((\psi, k), \{1, \ldots, k\})$$

First we show that $\varphi(\psi)$ is really an instance of $\mathrm{SAT{+}N}/f_{[k]}$. For any ψ, the solution set of (ψ, k) as an instance of SAT+N contains $\{1, \ldots, k\}$. Since f is identity about integers, the solution set of (ψ, k) as an instance of $\mathrm{SAT{+}N}/f$ also contains $\{1, \ldots, k\}$, and thus $\varphi(\psi)$ $(\in {D^+}_{[k]})$ is a right instance of $\mathrm{SAT{+}N}/f_{[k]}$. Then using a similar argument, we can show that

$$g \in \sigma(\psi) \iff f(g) \in {\sigma^\dagger}_{[k]}(\varphi(\psi)) \quad (g \in G),$$

which means φ is a desired transformation. ∎

Then the problem left here is:

Problem 2.2 *Is it possible to show that* $\mathrm{SAT{+}N}/f$ *is* not *ASP-complete?*

Although we have not yet had a definite answer, we believe it to be unlikely that there is an ASP reduction to $\mathrm{SAT{+}N}/f$ which does not break the one-way property of f^{-1}. To solve this problem is one of the future works.

Chapter 3

ASP-completeness Results of Puzzles

Here we prove the ASP-completeness of three modern pencil puzzles widely played around the world.

3.1 Slither Link

Slither Link is one of the famous pencil puzzles originated in Japan. The rule of Slither Link is as follows:

- Each problem is given as a rectangular lattice. The length of sides of the rectangle (as to the unit length of lattice) is called the *size* of the problem.
- A 1×1 square surrounded by four points is called a *cell*. A cell may have a number out of 0, 1, 2 or 3.
- The goal is to make a loop which does not intersect or branch by connecting adjacent dots with lines, so that a number on the cell is equal to the number of lines drawn around it.

An example of problems of Slither Link is shown in Fig. 3.1.

Here we prove the ASP-completeness of Slither Link. To construct an ASP reduction, we use the following lemma.

Lemma 3.1 *To find a Hamiltonian circuit for a given planar graph with degree at most 3 is ASP-complete.*

Proof Seta [21] constructs a polynomial-time ASP reduction from 3SAT to this problem. ASP-completeness of this problem is derived from the result and Theorem 2.8 ■

We also use the following known fact (see [2]):

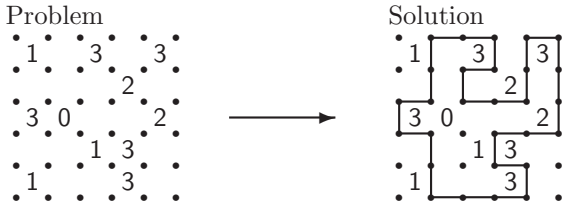

Figure 3.1: A problem of Slither Link.

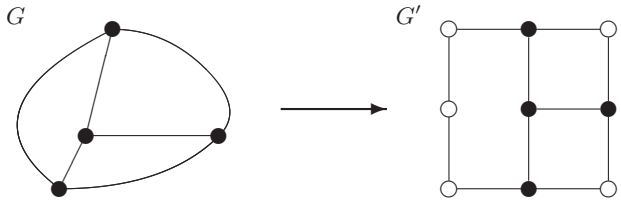

Figure 3.2: A graph embedded into a grid.

Lemma 3.2 *Any planar graph with degree at most 3 with n vertices can be embedded in an $\mathcal{O}(n) \times \mathcal{O}(n)$ grid in polynomial time in n.*

Now we are ready to state the proof.

Theorem 3.3 *To find a solution to a given instance of Slither Link is ASP-complete.*

Proof The membership in **FNP** is immediate. We construct a polynomial-time ASP reduction from the restricted Hamiltonian circuit problem to the Slither Link problem.

Using Lemma 3.2, we can transform a given instance (graph) G of the restricted Hamiltonian circuit problem into a graph G' on the grid. Note that G' has lattice points which do not correspond to any vertex of G and thus need not be visited when considering Hamiltonian circuits of G'. The degree of such points is two.

First, the gadget for a lattice point which need not be visited is the 6×6 board A shown in Fig. 3.3. Here we assume that no lines are drawn around this gadget. (This assumption will be satisfied later.) Then in this gadget lines can be drawn only on the place shown by dotted lines in the Figure (A†). Hence there are only four points, n, s, w and e, through which a line goes out of the gadget (we call them *joint points*). Consequently, the local solution is either (i) that no lines are drawn or (ii) that a path connecting two joint points is drawn as shown in A1, A2 and A3 (or their rotated or mirror images). (In fact two paths can be drawn as in A4. But this case will be

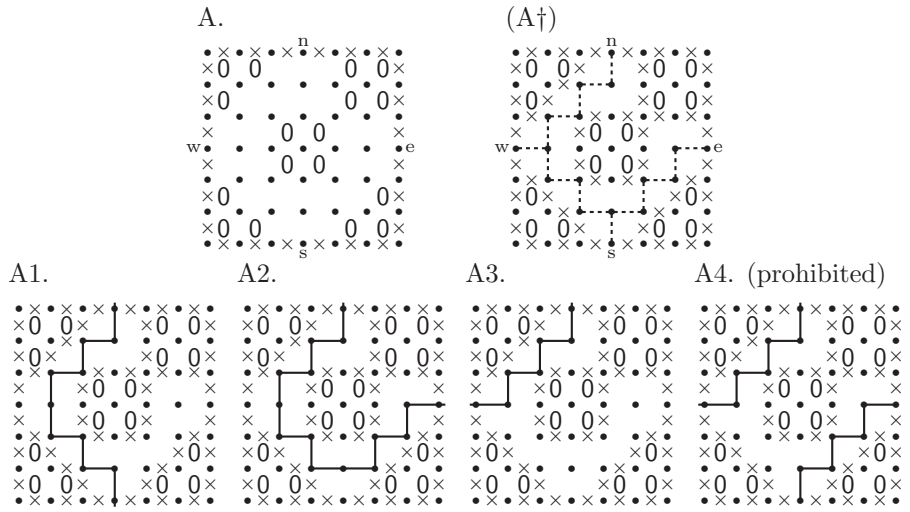

Figure 3.3: The gadget for a point in G' which does not correspond to a vertex of G, and local solutions to it.

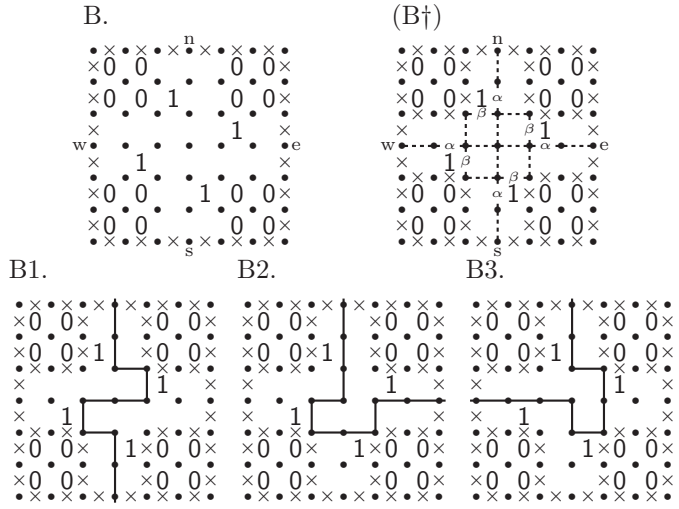

Figure 3.4: The gadget for a vertex of G' which corresponds to a vertex of G, and local solutions to it.

prohibited later by the condition on the degree.) Note that for any two joint points there is a unique way to connect them.

Next, the gadget for a lattice point which must be visited is the board B shown in Fig. 3.4. Again we assume no lines around the gadget. The place where lines can

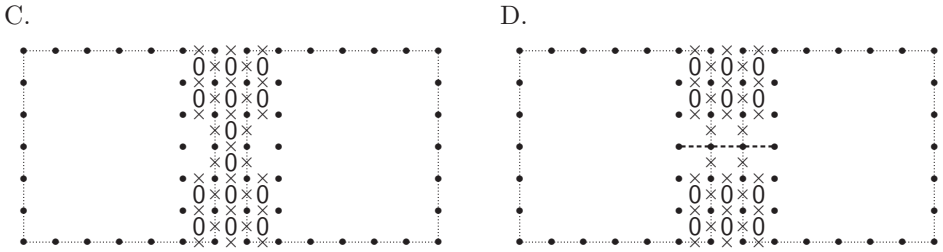

Figure 3.5: How to join gadgets.

be drawn is shown in the Figure (B†). (Here a line marked α is used only when the path goes out of the gadget straightly through the line, and a line marked β is used otherwise.) This time a path connecting two points on the boundary *must* be drawn, as in B1, B2 and B3, because this gadget has some 1's inside. Note that for any two joint points there is a unique way to connect them.

Last, we arrange these two sorts of gadgets in accordance with the grid graph G': we join two gadgets as in C of Fig. 3.5 where G' has a corresponding edge, and as in D where not. Only D allows a path connecting two gadgets. Moreover, both arrangements forbid lines on the boundary of the gadgets. (In this figure only one side seems to be forbidden to have lines. However, in the entire construction a gadget usually has adjoining gadgets in all the directions. Moreover drawing lines on the boundary of the entire board is also forbidden, since lines cannot go out of the board.) Because all vertices of G' have a degree at most three, not all of n, s, w and e of a gadget A can have a passing line. This restriction prevents two paths passing through a gadget A.

In this way, we obtain the problem of Slither Link corresponding to G' (that is, G). (An example of the overall construction is shown in Fig. 3.1.) This transformation can be done in polynomial time in the input size, and the solution of Slither Link corresponding to a Hamiltonian circuit of G is unique and also computable in polynomial time. Thus a polynomial time ASP reduction from the restricted Hamiltonian circuit problem to Slither Link is constructed. ■

3.2 Number Place

The rule of Number Place (also known as *Sudoku* in Japan) is as follows:

- A problem is given as a $n^2 \times n^2$ grid, which is divided into $n \times n$ squares with thick border lines. The value n is called *order*.

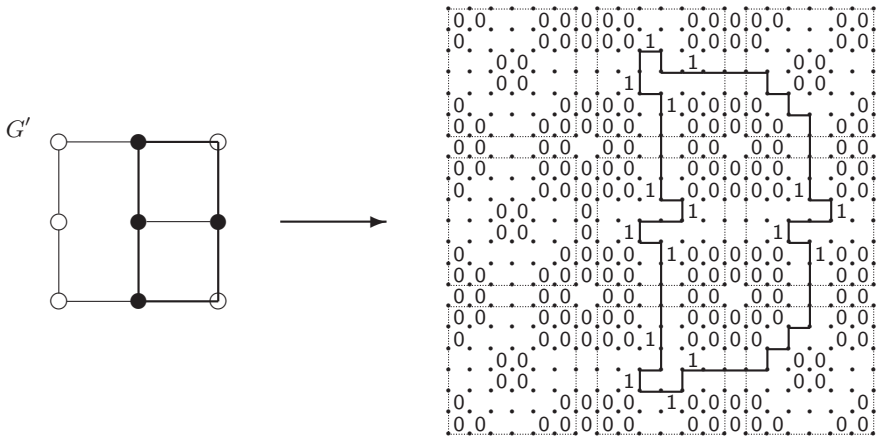

Figure 3.6: An example of the overall construction of reduction.

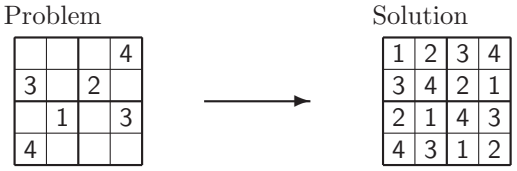

Figure 3.7: A problem of Number Place.

- Some cells are filled with an integer from 1 through n^2.
- The goal is to fill in all the blank cells so that each row, column and $n \times n$ square has each of integers from 1 through n^2 exactly once.

An example of Number Place problems is shown in Fig. 3.7.

To show the ASP-completeness of this puzzle, we use the result about Latin squares. A *Latin square* of order n is a matrix such that each row and column contains each of integers from 1 through n exactly once. (Similar to Number Place, but lacking of the "small square" condition.) A *partial Latin square* is a matrix with some blank entries such that each row and column contains each of integers from 1 through n at most once. The problem of *partial Latin square completion* is as follows: Given a partial Latin square, make a Latin square by filling in the blanks. The ASP-completeness of this problem is immediate from known results.

Theorem 3.4 *The problem of partial Latin square completion is ASP-complete.*

A0	01	02	**B0**	11	12	**C0**	21	22
D0	11	12	**E0**	21	22	**F0**	01	02
G0	21	22	**H0**	01	02	**I0**	11	12
01	02	10	11	12	20	21	22	00
11	12	20	21	22	00	01	02	10
21	22	00	01	02	10	11	12	20
02	10	11	12	20	21	22	00	01
12	20	21	22	00	01	02	10	11
22	00	01	02	10	11	12	20	21

A	B	C
D	E	F
G	H	I

Figure 3.8: Relation between Number Place and Latin square (in the case $n = 3$): Integers in Number Place are represented in base n. Although the cells with A0, ..., I0 are in actual problems blank, the lower digit of the numbers filling these cells must be 0 from the rule of Number Place. Moreover, the square on the right forms a Latin square.

Proof Colbourn [4] has proved the NP-completeness of ASP of partial Latin square completion by showing a reduction from 1-in-4 SAT to this problem. The reduction he used is what we call ASP reduction here. The ASP-completeness of 1-in-4 SAT is proven in an analogous way to that of 1-in-3 SAT. ■

More we show the next lemma which shows a relation between the two problems. In the argument below, we use integers ranged from 0 (instead of 1) as row and column numbers and contents of cells, and write $S(i, j)$ for the entry at position (i, j) of a square S ($\perp$ means a blank).

Lemma 3.5 *Let S be a Number Place problem of order n such that*

$$S(i,j) = \begin{cases} \perp & \textit{(when } (i,j) \in B\textit{)} \\ ((i \bmod n)n + \lfloor i/n \rfloor + j) \bmod n^2 & \textit{(otherwise)} \end{cases}$$

where $B = \{(i, j) \mid \lfloor i/n \rfloor = 0$ and $(j \bmod n) = 0\}$. Then a square S' obtained by filling in the blanks of S is a solution to S if and only if

- *For any $(i, j) \in B$, $S'(i, j) \bmod n$ equals 0.*
- *A square L defined by $L(i, j/n) = S'(i, j)/n$ for all $(i, j) \in B$ is a Latin square.*

(as shown in Fig. 3.8)

Proof First of all we show the following proposition:

The square S_0 defined by

$$S_0(i,j) = ((i \bmod n)n + \lfloor i/n \rfloor + j) \bmod n^2$$

forms a solution to Number Place. That is. each row, column and small square contains each of integers from 0 through $n^2 - 1$ exactly once.

Let $i_{\mathrm{l}} = (i \bmod n)$ and $i_{\mathrm{h}} = \lfloor i/n \rfloor$. Then, when i ranges over 0 through $n^2 - 1$, $(i_{\mathrm{l}}, i_{\mathrm{h}})$ takes all pairs consisting of two integers from 0 through $n - 1$. The same holds for j, and

$$S_0(i, j) = (i_{\mathrm{l}} n + i_{\mathrm{h}} + j_{\mathrm{h}} n + j_{\mathrm{l}}) \bmod n^2.$$

First we fix j, that is, look at a certain column), and then $S_0(i, j) = S_0(i', j)$ implies $i_{\mathrm{l}} n + i_{\mathrm{h}} \equiv i'_{\mathrm{l}} n + i'_{\mathrm{h}} \pmod{n^2}$, which means from the argument above $(i_{\mathrm{l}}, i_{\mathrm{h}}) = (i'_{\mathrm{l}}, i'_{\mathrm{h}})$, i. e., $i = i'$. That means no integer appeans twice in a column. The same holds for a row. In order to think of a small square we fix i_{h} and j_{h}, and then $S_0(i, j) = S_0(i', j')$ implies $i_{\mathrm{l}} n + j_{\mathrm{l}} \equiv i'_{\mathrm{l}} n + j'_{\mathrm{l}} \pmod{n^2}$, which means $(i_{\mathrm{l}}, j_{\mathrm{l}}) = (i'_{\mathrm{l}}, j'_{\mathrm{l}})$, i. e., $i = i'$, $j = j'$. Therefore the proposition is proved.

Next we consider which integer fills the blank cells (cells belonging to B) in S. For $(i, j) \in B$, $S_0(i, j)$ equals $n(i_{\mathrm{l}} + j_{\mathrm{h}}) \bmod n^2$ and thus is divisible by n. Since each integer appears as many times in S_0 as in S', integers which fill blanks of S must also be divisible by n.

Last we will show the relation to L. We only have to think of cells of B (about the other cells the condition is already satsfied). The row constraint of S' requires that

$$S'(i, j) = S'(i, j') \implies j = j'. \tag{3.1}$$

However, as far as B is concerned, $j = j' \iff j_{\mathrm{h}} = j'_{\mathrm{h}}$ and $S'(i, j) = S'(i, j') \iff L(i, j_{\mathrm{h}}) = L(i, j'_{\mathrm{h}})$. Thus (3.1) is equivalent to

$$L(i, j_{\mathrm{h}}) = L(i, j'_{\mathrm{h}}) \implies j_{\mathrm{h}} = j'_{\mathrm{h}}, \tag{3.2}$$

which is the row constraint of the Latin square L. Similarly the column constraint of S' turns out to be equivalent to the column constraint of L. The small-square constraint of S' is also equivalent to the column constraint of L. Now the proof is completed. ∎

Now we are ready to state the proof.

Theorem 3.6 *To find a solution to a given instance of Number Place is ASP-complete.*

Proof The membership in **FNP** is immediate. We show a polynomial time ASP reduction from the problem of partial Latin square completion to Number Place.

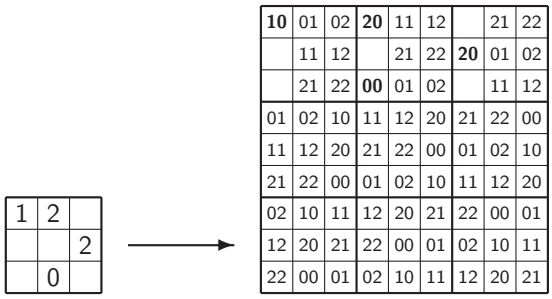

Figure 3.9: An example of the overall construction of reduction.

For a given partial Latin square L of order n, we construct a Number Place problem S as follows:

$$S(i,j) = \begin{cases} L(i, j/n) \cdot n & ((i,j) \in B,\ L(i,j/n) \neq \perp) \\ \perp & ((i,j) \in B,\ L(i,j/n) = \perp) \\ ((i \bmod n)n + \lfloor i/n \rfloor + j) \bmod n^2 & \text{(otherwise)} \end{cases}$$

(B is defined samely as in Lemma 3.5.) This construction can be done in polynomial time in the input size. Moreover, from Lemma 3.5, each solution to L has a unique corresponding solution to S, which is also polynomial-time computable. Thus the desired polynomial-time ASP reduction is obtained. ■

3.3 Fillomino

The rule of Fillomino is as follows:

- A problem is given as a rectangular grid, with some cells filled with positive integers. The length of sides of the rectangle is called the *size* of the problem.
- The goal is to fill each blank cell with a integer and divide the board into several blocks so that
  - each block consists of one or more consecutive cells (forms a *polyomino*),
  - all the cells in a block are filled with the same integer, which equals to the area of the block,
  - blocks with the same area do not share their edges.

An example of Fillomino is shown in Fig. 3.10.

In order to show the ASP-completeness of Fillomino we use a reduction from a planar version of 3SAT defined as follows:

Problem

6			2			2
	3		6		3	
3						1
	2	3		4	2	
2						3
	5		1		4	
4			3			3

Solution

6	6	6	2	2	3	2
3	3	6	6	6	3	2
3	2	3	3	1	3	1
1	2	3	5	4	2	2
2	5	5	5	4	4	3
2	5	4	1	3	4	3
4	4	4	3	3	1	3

Figure 3.10: A problem of Fillomino.

Definition 3.1 Let ϕ be a 3-CNF formula which has the variable set $X = \{x_1, \ldots, x_n\}$ and the clause set $C = \{c_1, \ldots, c_m\}$. The *bipartite graph associated with* ϕ, denoted by $G(\phi)$, is defined as follows:

$$G(\phi) = (X \cup C, E); \quad E = \{(v_i, c_j) \mid v_i \text{ or } \bar{v}_i \text{ appears in } c_j\}.$$

The problem of *planar 3SAT* is defined to be the satisfiability problem about a 3-CNF formula ϕ where $G(\phi)$ is planar.

Planar 3SAT is introduced and proved to be **NP**-complete in [13]. Moreover, ASP-completeness of thie problem can be derived from the known result.

Lemma 3.7 *Planar 3SAT is ASP-complete.*

Proof An ASP reduction from 3SAT to planar 3SAT is presented in [10]. The ASP-completeness of planar 3SAT is derived from the result and the ASP-completeness of 3SAT. ∎

The graph associated with a formula can be easily transformed to a logical circuit in the following way: replace a 'variable' vertex by an input node and a 'clause' vertex by a OR gate, and then optionally insert a NOT gate between an input node and a OR gate. If the original graph is planar, the resulting cirxuit is also planar. Moreover the output of the OR gates for an input is all 'true' if and only if the input satisfies the Boolean formula. Therefore the problem of planar 3SAT can be viewed as satisfiability problem of planar multi-output circuits where all outputs must be 'true'. Our reduction exploits this characterization. (Similar approach is taken in [12].)

Theorem 3.8 *To find a solution to a given instance of Fillomino is ASP-complete.*

Proof The membership in **FNP** is immediate. We show a polynomial time ASP reduction from planar 3SAT to Fillomino.

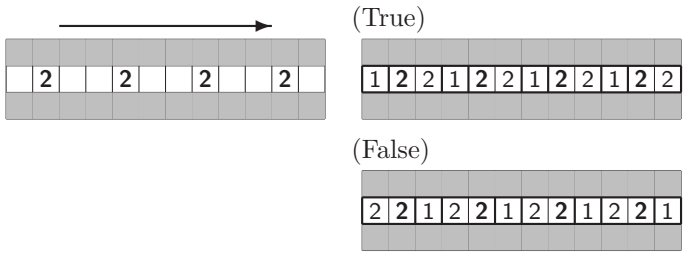

Figure 3.11: A straight wire.

As is seen from the argument above all we have to do is represent a planar multi-output circuit as an instance of Fillomino, where an input which makes all outputs 'true' corresponds to a solution of the Fillomino problem. Such a circuit can be constructed from the following components: wires carrying signals (truth values), input and output nodes, signal splitters, NOT gates and AND gates. (Since the circuit is planar, we need not consider crossover of wires.)

We will construct a gadget for each component, but before doing it we explain how to treat cells of the Fillomino problem which is *not* involved in the representation of the circuit. In the construction described below, such cells are expressed as gray cells. However, in actual construction, these cells must be filled with appropriate integers. That is, each of these cells must be filled with the number of cells contained in the connected component of cells irrelevant to the circuit including the cell in the transformed problem of Fillomino. In many cases such integers are very large. In particular we can presume that all such integers be larger than three. On the other hand as to cells which form gadgets we will only use integers not larger than three, so that no collision can occur.

Firstly, Fig. 3.11 represents a straight wire. This gadget has two local solutions, one corresponding to 'true' and one corresponding to 'false'. We decide the corespondence as follows. We first think of wires as given the orientation from input to output (shown as an arrow in the figure). Then we decide a wire to be carrying 'true' when 2's which are newly filled in the solution are put in front of the 2's given in the problem (we call them *basis 2's* and show them in bold face in the figure), and 'false' when new 2's in the solution are put behind the basis 2's.

Fig. 3.12 shows the gadget for an input node. This gadget has two local solutions. That means we can send either 'true' or 'false' from the node.

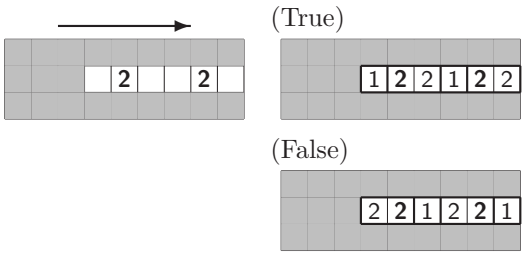

Figure 3.12: An input node.

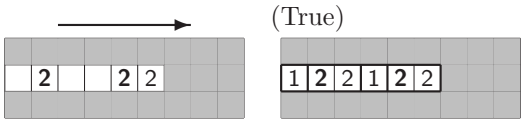

Figure 3.13: An output node.

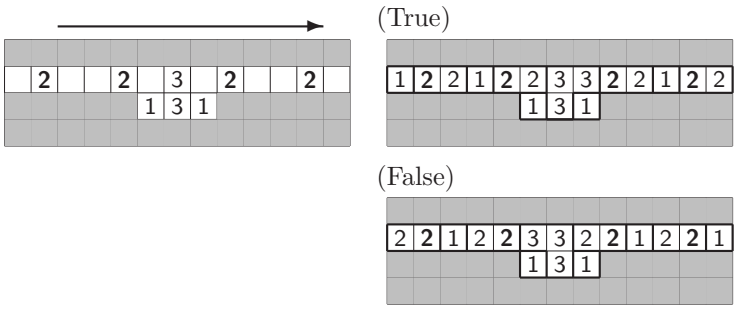

Figure 3.14: A phase changer.

Fig. 3.13 shows the gadget for an output node. The condition that all outputs must be 'true' is realized by the gadget forcing the 'true' configuration.

Next we want to construct logical gates, but there is a problem. The basis 2's in a wire appear periodically, which fact imposes unwanted restriction to arrangement of gadgets. To resolve this difficulty, we use the gadget shown in Fig. 3.14, which cancels the periodicity.

Fig. 3.15 shows the gadget to split a signal into two wires. The repeated use of this gadget allows a signal splitted into more wires. This gadget can be used to bend a single wire. In this case the unused wire is terminated by the gadget which is the same as Fig. 3.12 except that the orientation is reversed.

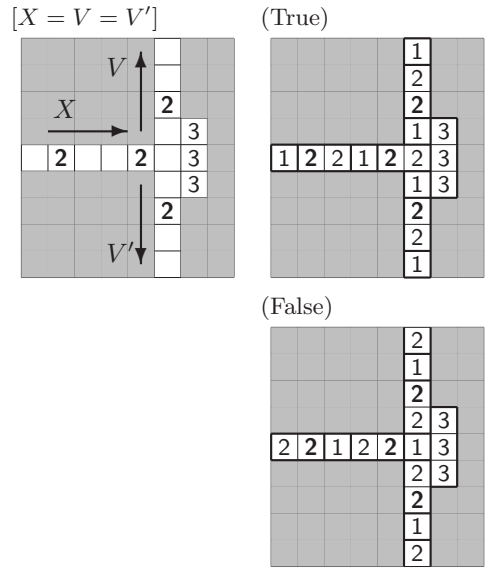

Figure 3.15: A signal splitter.

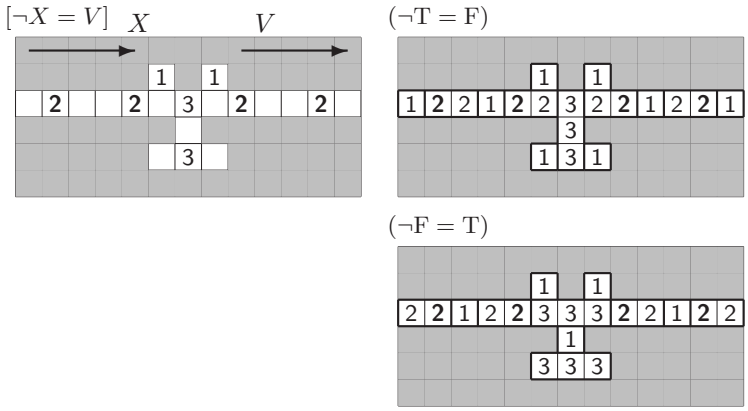

Figure 3.16: NOT gate.

Fig. 3.16 is a NOT gate. The output V is the negation of the input X.

Lastly, Fig. 3.17 is an OR gate. When either of two inputs is 'true', only three or four cells are left around the cell with 3 (including the cell itself), and that fact forces the output configuration to be 'true'. On the other hand, when both of two inputs are 'false', five cells are left around the 3, which fact makes the output 'false'.

By arranging these gadgets, we can transform an instance of planar 3SAT to a instance of Fillomino. This transformation can be done in polynomial time with respect

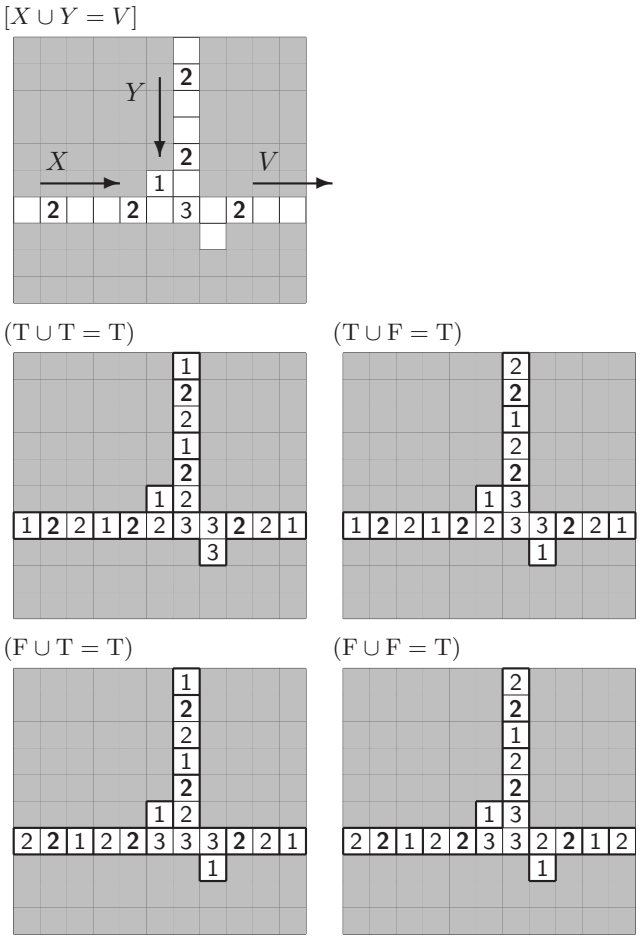

Figure 3.17: OR gate.

to the input size. Moreover, each gadget has a unique local solution correspoding to the specified input-output pattern. Thus the solution to the constructed Fillomino problem corresponding to a solution to planar 3SAT is unique and polynomial-time computable. Therefore an polynomial-time ASP reduction from planar 3SAT to Fillomoino is obtained. ■

3.4 Some Other Puzzles

Because Seta's results [20] on the **NP**-completeness of n-ASP of puzzles are based on ASP reductions, they can be easily extended to ASP-completeness, in combination with ASP-completeness of basic problems he used as the reduction source.

Theorem 3.9 *To find a solution to a given instance of Cross Sum is ASP-complete.*

Proof Seta [21] gives a polynomial-time ASP reduction from 1-in-3 SAT to Cross Sum. ASP-completeness of Cross Sum is derived from the result and Theorem 2.9. ■

Theorem 3.10 *To find a solution to a given instance of Cross Sum is ASP-complete.*

Proof Ueda and Nagao [22] gives a polynomial-time ASP reduction from 3DM to Nonogram. The reduction from 3SAT to 3DM given in [14] can be easily modified to make it a polynomial-time ASP reduction. Thus ASP-completeness of Nonogram is obtained from Theorem 2.8. ■

Chapter 4

Concluding Remarks

We formalized n-ASP, the problem to find another solution when n solutions are given, to facilitate strict investigations on its complexity. In particular, we introduced ASP-completeness, the completeness with respect to ASP reductions, and proved that ASP-completeness implies the **NP**-completeness of n-ASP for any nonnegative integer n. This result is useful in the following aspects:

- For most of the known **NP**-complete problems, their counting version is shown to be #**P**-complete.
- Moreover, in most cases, the proof of #**P**-completeness is done by constructing a bijection of solutions concretely, and thus such proof also proves ASP-completeness.
- As a result, it can be said that most of the **NP**-complete problems are ASP-complete (as a function problem) and their n-ASPs are **NP**-complete. All you have to do for most cases is inspect the proof of #**P**-completeness and check if the transformation of solutions is computable in polynomial-time.

In short, the notion of ASP-completeness provides an easy way to prove the **NP**-completeness of ASP of a problem.

Moreover we investigated the relation between ASPs and other versions of problems, namely counting problems and enumeration problems. There we showed the following:

- ASP-completeness implies #**P**-completeness but the converse does not hold.
- The class of problems of which n-ASP is solvable in polynomial time is equal to the class of problems which allow enumerations of solutions in polynomial time in a sense.

On the other hand one question of interest, whether the **NP**-completeness of n-ASP for all n implies ASP-completeness or not, was left open as future work, although we provided some strategy to prove that the proposition was false.

As an application to the field of combinatorial puzzles, we proved the ASP-completeness of three popular puzzles: Slither Link, Number Place, and Fillomino. These results indicate that designing these sorts of puzzles, as well as solving, is essentially difficult. We hope that more ASP-completeness results will appear.

References

[1] H. Adachi, H. Kamekawa, and S. Iwata. Shogi on $n \times n$ board is complete in exponential time. *Trans. IEICE*, J70-D(10):1843–1852, 1987. (in Japanese).

[2] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry*, 4(5):235–284, 1994.

[3] T. C. Biedl, E. D. Demaine, M. L. Demaine, R. Fleischer, L. Jacobsen, and J. I. Munro. The complexity of Clickomania. In R. J. Nowakowski, editor, *More Games of No Chance*, pages 389–404. Cambridge University Press, 2002. Collection of papers from MSRI Combinatorial Game Theory Research Workshop, Berkeley, California, July, 2000.

[4] C. J. Colbourn, M. J. Colbourn, and D. R. Stinson. The computational complexity of recognizing critical sets. In *Graph theory, Singapore 1983*, number 1073 in Lecture Notes in Math., pages 248–253. Springer, 1984.

[5] E. D. Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. Computing Research Repository, arXiv:cs.CC/0106009, 2002. (available at http://arXiv.org/).

[6] E. D. Demaine, S. Hohenberger, and D. Liben-Nowell. Tetris is hard, even to approximate. Computing Research Repository, arXiv:cs.CC/0106009, 2002. (available at http://arXiv.org/).

[7] D. Eppstein. On the NP-completeness of cryptarithms. *SIGACT News*, 18(3):38–40, 1987.

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

[9] M. R. Garey, D. S. Johnson, and R. Endre Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4), 1976.

[10] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, and R. E. Stearns. The complexity of planar counting problems. *SIAM Journal on Computing*, 27(4):1142–1167, 1998.

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

[12] R. Kaye. Minesweeper is NP-complete. *Mathematical Intelligencer*, 22(2):9–15, 2000.

[13] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11:329–343, 1982.

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

[15] D. Ratner and M. Warmuth. Finding a shortest solution for the $N \times N$-extension of the 15-puzzle is intractable. *J. Symb. Comp.*, 10:111–137, 1990.

[16] S. Reisch. Gobang ist PSPACE-vollständig. *Acta Informatica*, 13:59–66, 1980.

[17] J. M. Robson. The complexity of Go. In *Proceeding of the IFIP 9th World Computer Congress on Information Processing*, pages 413–417, 1983.

[18] J. M. Robson. N by N checkers is EXPTIME-complete. *SIAM Journal on Computing*, 13(2):252–267, 1984.

[19] T. Saito. An algorithm for automatic generation of the Slither Link problems. Senior thesis, Department of Infomation Science, the Faculty of Science, the University of Tokyo, 1998.

[20] T. Seta. The complexities of CROSS SUM. *IPSJ SIG Notes AL-84*, pages 51–58, 2002. (in Japanese).

[21] T. Seta. The complexities of puzzles, CROSS SUM and their another solution problems (ASP). Senior Thesis, Department of Infomation Science, the Faculty of Science, the University of Tokyo, 2002.

[22] N. Ueda and T. Nagao. NP-completeness results for NONOGRAM via parsimonious reductions. Technical Report TR96-0008, Department of Computer Science, Tokyo Institute of Technology, 1996.

[23] R. Uehara and S. Iwata. Generalized Hi-Q is NP-complete. *Trans. IEICE*, E73(2):270–273, 1990.

[24] T. Uno. Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs. In *Algorithms and Computation (Proceeding of ISAAC97)*, number 1350 in Lecture Notes in Computer Science, pages 92–101. Springer, 1997.

[25] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[26] T. Yato. On the NP-completeness of the Slither Link puzzle. *IPSJ SIG Notes AL-74*, pages 25–32, 2000. (in Japanese).

[27] T. Yato and T. Seta. Complexity and completeness of finding another solution and its application to puzzles. *Trans. IEICE*, E86-A(5), 2003. to appear.