IEICE TRANS. INF. & SYST., VOL.E91–D, NO.4 APRIL 2008

PAPER

Recursion Theoretic Operators for Function Complexity Classes*

Kenya UENO[†a)], ***Student Member***

SUMMARY We characterize the gap between time and space complexity of functions by operators and completeness. First, we introduce a new notion of operators for function complexity classes based on recursive function theory and construct an operator which generates $FPSPACE$ from FP. Then, we introduce new function classes composed of functions whose output lengths are bounded by the input length plus some constant. We characterize FP and $FPSPACE$ by using these classes and operators. Finally, we define a new notion of completeness for $FPSPACE$ and show a $FPSPACE$-complete function.

key words: *recursive function theory, function complexity classes, operators for complexity classes*

1. Introduction

Recursive function theory expresses the computable function class as the smallest class containing some initial functions and closed under operators that generate new functions. Similarly, it is known that many function complexity classes are characterized by recursion theoretic scheme [3], [12]. Grzegorczyk [5] introduced the hierarchical classes $\mathcal{E}^n$ by composition and an operator called bounded recursion. Later, Ritchie [11] showed that $\mathcal{E}^2$ is equal to the linear space computable function class. Cobham [4] characterized FP by composition and bounded recursion on notation. Thompson [15] characterized $FPSPACE$ by composition and bounded recursion.

Whereas these studies expanded various fields [2] and have many interesting applications, there have been no attempts to study the relation among function complexity classes from a recursion theoretic viewpoint. The current studies are only concerned with how to characterize function complexity classes simpler way or fewer initial functions. However, this direction is useless when we set our goal separation of complexity classes. It is the relation among them that is the most important thing for separation. From this perspective, we extend the notion of operators to operators that act on general function complexity classes.

Operators for complexity classes are useful to clarify the relation among them [10], [17], [18]. Toda's theorem $PH \subseteq P^{PP}$ [16] makes use of some properties of operators. There are many operators studied like $\exists$ and $\forall$, which are the operators that generates NP and $coNP$ from P respectively. Many other operators that express the gaps between P and complexity classes like RP, BPP, PP and $\oplus P$ have been also studied. Then, what about the case of P and $PSPACE$? Is there any appropriate one that simply expresses the gap between P and $PSPACE$? In fact, it is hard to construct it because operators like $\exists$ and $\forall$ are defined according to the number of accepting configurations. However, in the case of function complexity classes, we can say that there exists an appropriate one.

Function complexity classes related with NP are studied widely [13]. However, there are few studies about $FPSPACE$. In this paper, we clarify the structure of $FPSPACE$. The importance of this study is that we show "functions can do what languages cannot do". This is one of the few examples which clearly indicate merits of studies on function complexity classes compared with language complexity classes.

We construct an operator which exactly expresses the gap between FP and $FPSPACE$. We introduce new operators $Comp^*$ and $BRec$ and show

$$FPSPACE = Comp^*(BRec(FP)).$$

This result can be generalized. For example, this operator generates the linear space computable function class from the linear time computable function class as

$$FSPACE(n) = Comp^*(BRec(FTIME(n))).$$

Moreover, we introduce an operator $RecN$ and new function complexity classes $AGTIME$ and $AGSPACE$. We show the structures of FP and $FPSPACE$ by showing

$$FP = Comp^*(RecN(AGTIME))$$

and

$$FPSPACE = Comp^*(RecN(AGSPACE)).$$

The case of FP is proven by simulating a polynomial time bounded deterministic Turing machine by using a function called "*step*", which receives a configuration and returns the next configuration. To prove the case of $FPSPACE$, we introduce a new function called "*next*" and substitute it for the function *step*. It simulates partial works of a deterministic Turing machine by manipulating configurations within limited space.

Manuscript received August 27, 2007.
Manuscript revised November 30, 2007.

[†]The author is with the Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, 113–8656 Japan.

*This paper was presented at The 16th Annual International Symposium on Algorithms and Computation (ISAAC 2005), Sanya, Hainan, China, December, 2005.

a) E-mail: kenya@is.s.u-tokyo.ac.jp

DOI: 10.1093/ietisy/e91–d.4.990

The function *next* contains a further implication. We introduce completeness for $FPSPACE$, which is a new notion while there is a study about completeness for polynomial space counting classes [9], and show that this function is $FPSPACE$-complete under FP Turing reductions. It is also $FSPACE(n)$-complete under $FTIME(n)$ Turing reductions. Thus, we can say that *next* exactly expresses the gap between time and space.

2. Function Complexity Classes

We assume that the readers are familiar with the basic notion of the Turing machine, resource bounded computations and the definitions of complexity classes such as P and $PSPACE$. A configuration is a representation of the whole state of a Turing machine at a certain time.

Functions discussed in this paper are partial and single-valued functions from natural numbers to a natural number. If we say that a Turing machine computes a function, it means that natural numbers are represented by strings of the binary notation and it converts strings into a string.

First, we define the following function complexity classes.

Definition 2.1.

- *$FTIME(t(n))$ is defined as the class of functions computable in $c \cdot t(n)$ time for some constant c by an offline multitape deterministic Turing machine.*
- *$FSPACE(s(n))$ is defined as the class of functions computable in $c \cdot s(n)$ space for some constant c by an offline multitape deterministic Turing machine.*

Output lengths of functions in $FSPACE(s(n))$ are bounded by $c \cdot s(n)$ for some constant c.

Then, FP and $FPSPACE$ are defined as follows.

Definition 2.2.

- $FP = \bigcup_{c \geq 1} FTIME(n^c)$.
- $FPSPACE = \bigcup_{c \geq 1} FSPACE(n^c)$.

Output lengths of functions in $FPSPACE$ are also bounded by polynomial in the input length.

These function complexity classes are closely related with language complexity classes.

Theorem 2.3 ([1], [14]).

$$P = PSPACE \Leftrightarrow FP = FPSPACE.$$

This theorem is proven by the prefix searching method, which appears in [1] and [14]. This theorem implies that there is little difference whether we study languages or functions for separating complexity classes. So, we may choose an appropriate one according to the case.

Furthermore, we introduce new function classes, which are composed of functions whose output lengths are bounded by the input length plus some constant. We call them additive growth functions.

Definition 2.4.

- *$AGTIME$ is defined as the class of functions computable in $n + c$ time by an offline multitape deterministic Turing machine for some constant c where n is the input length.*
- *$AGSPACE$ is defined as the class of functions computable in $n + c$ space by an offline multitape deterministic Turing machine for some constant c where n is the input length.*

The output lengths of functions contained in $AGSPACE$ are bounded by $n + c$ for some constant c.

3. Recursion Theoretic Operators

Now, we define the operators based on recursive function theory. Let C be an arbitrary function complexity class and we define operators by applying them to C. We write multiple argument function as $f(\vec{x}) = f(x_1, \cdots, x_n)$ for ease. Then, the operator *Comp* is defined as follows.

Definition 3.1.

$$Comp(C) = C \cup \{f \mid f(\vec{x}) = g(h_1(\vec{x}), \cdots, h_n(\vec{x})), \\ g, h_1, \cdots, h_n \in C\}.$$

In addition, we define $Comp^*(C)$ as the smallest class that contains C and is closed under *Comp*.

Bounded recursion is recursion which restricts output values of functions by a certain function. The operator *BRec* is defined as follows. *BRec** is defined similarly as the case of *Comp**.

Definition 3.2.

$$BRec(C) = C \cup \{f \mid f(0, \vec{y}) = g(\vec{y}), \\ f(x, \vec{y}) = h(x, \vec{y}, f(x-1, \vec{y})), \\ \forall x, \forall \vec{y}, f(x, \vec{y}) \leq k(x, \vec{y}), \\ g, h, k \in C\}.$$

More precisely, $f(x, \vec{y})$ is constructed from $g, h, k \in C$ as $g(\vec{y})$ when $x = 0$ and $h(x, \vec{y}, f(x-1, \vec{y}))$ otherwise and it is bounded by $k(x, \vec{y})$.

Recursion on notation is recursion that decreases length instead of value whenever it carries out recursion. The operator *RecN* is defined as follows.

Definition 3.3.

$$RecN(C) = C \cup \{f \mid f(0, \vec{y}) = g(\vec{y}), \\ f(x, \vec{y}) = h(x, \vec{y}, f(\lfloor \frac{x}{2} \rfloor, \vec{y})), \\ g, h \in C\}.$$

More precisely, $f(x, \vec{y})$ is constructed from $g, h \in C$ as $g(\vec{y})$ when $x = 0$ and $h(x, \vec{y}, f(\lfloor \frac{x}{2} \rfloor, \vec{y}))$ otherwise.

4. Closure Properties

Before proceeding the main result, we describe some basic properties related with operators defined above. First, we see the closure properties related with *Comp*.

Proposition 4.1. *$FTIME(n)$, $FSPACE(n)$, FP and $FPSPACE$ are closed under Comp.*

Next, we show the closure properties related with *BRec*.

Lemma 4.2. *If $s(n) \geq n$,*

$$BRec(FSPACE(s(n))) \subseteq FSPACE(s(n + s(n))).$$

Proof. We consider a function $f \in BRec(FSPACE(s(n)))$, which is obtained by applying *BRec* to $g, h, k \in FSPACE(s(n))$. The program computing f is described as follows:

input $x, \vec{y}$
$z = g(\vec{y})$;
for($i = 1$ to x){
  $z = h(i, \vec{y}, z)$;
}
output z .

We can reuse space many times and the length of z is bounded by $s(|x|+|y|)$ in the repetition because f is bounded by k. Moreover, space required for i is $|x|$. Thus, we have

$$\begin{aligned} s'(|x| + |y|) &\leq \max_z\{s(|x| + |y| + z) + |x|\} \\ &\leq s(|x| + |y| + s(|x| + |y|)) + |x| \\ &\leq 2 \cdot s(n + s(n)) \end{aligned}$$

because $s(n) \geq n$. Here, $s'(n)$ is the space complexity of this program. □

By this lemma, the following proposition holds.

Proposition 4.3. *$FSPACE(n)$ and $FPSPACE$ are closed under BRec, respectively.*

5. Time, Space and Bounded Recursion

In this section, we show that *BRec* characterizes the relation between time and space. First, we prove the following lemma.

Lemma 5.1. *If $s(n) \geq n$,*

$$FSPACE(s(n)) \subseteq Comp^*(BRec(FTIME(s(n)))).$$

Proof. Let f be an arbitrary function computable in $k \cdot s(n)$ space for some constant k. We define the following functions for the Turing machine computing f.

1. $init(x)$ is a function that returns the initial configuration on input x.
2. $step(c)$ is a function that returns a configuration when the Turing machine goes to the next step from the configuration c.
3. $out(c)$ is a function that returns the output of the configuration c.

These functions are all computable in linear time of each input. Thus, they are also in $FTIME(s(n))$ on assumption of $s(n) \geq n$.

Then, there exists a function $T \in FTIME(s(n))$ that satisfies $\forall x,\ |T(x)| > k \cdot s(|x|)$. There also exists a function $S \in FTIME(s(n))$ that satisfies $\forall t, x,\ |S(t, x)| > |T(x)| + k'$ for any constant k' that depends only on f. This function is defined in order to represent all configurations in $|S(t, x)|$ space.

Applying *BRec* to these functions, we construct the following function $F \in BRec(FTIME(s(n)))$.

$$\begin{aligned} &F(0, x) = init(x), \\ &F(t, x) = step(F(t - 1, x)), \\ &\forall t, \forall x, F(t, x) \leq S(t, x). \end{aligned}$$

Remark that $k \cdot s(x)$ space bounded computations are simulated by $2^{k \cdot s(x)}$ time bounded computations. Thus, the computation of f is simulated as $f(x) = out(F(T(x), x))$ because recursion is carried out $2^{|T(x)|}$ times, which is larger than $2^{k \cdot s(x)}$. □

Thus, we can obtain an operator that expresses the gap between FP and $FPSPACE$.

Theorem 5.2.

$$\begin{aligned} FPSPACE &= Comp^*(BRec(FP)) \\ &= Comp^*(BRec^*(FP)). \end{aligned}$$

Proof. $Comp^*\ (BRec^*\ (FP)) \subseteq FPSPACE$ because $FPSPACE$ is closed under *Comp* and *BRec*. Moreover, $FPSPACE \subseteq Comp^*(BRec(FP))$ by Lemma 5.1. □

Corollary 5.3. *FP is closed under BRec if and only if $P = PSPACE$.*

As we can see from the proof, it does not seem to hold $Comp^*(BRec(C)) = FPSPACE$ where C is any function complexity class smaller than FP. To include $FPSPACE$ by $Comp^*(BRec(C))$, it is necessary for C to be stronger than FP. So, we can say that it is an operator that exactly expresses the gap between FP and $FPSPACE$.

By the same way, this operator expresses the gap between $FTIME(n)$ and $FSPACE(n)$.

Theorem 5.4.

$$\begin{aligned} FSPACE(n) &= Comp^*(BRec(FTIME(n))) \\ &= Comp^*(BRec^*(FTIME(n))). \end{aligned}$$

6. Structure of Polynomially Bounded Functions

In this section, we show that a combination of the operator *RecN* and additive growth functions expresses polynomially bounded functions.

Lemma 6.1. *There is a function*

$$p \in Comp^*(RecN(AGTIME))$$

that satisfies $\forall x,\ |p(x)| > c \cdot |x|^k$ *for any constants c and k.*

Proof. To prove the existence of such a function p, we consider the function $cat(x, y) \in AGTIME$ that returns concatenation of the inputs x, y. By the definition, $|cat(x, y)| = |x| + |y|$ holds. By applying *RecN* to this function, we construct the following function $\sigma_1 \in RecN(AGTIME)$.

$$\sigma_1(0) = 0,$$
$$\sigma_1(x) = cat\left(x, \sigma_1\left(\left\lfloor \frac{x}{2} \right\rfloor\right)\right).$$

This function satisfies the following equation.

$$|\sigma_1(x)| = |x| + (|x| - 1) + (|x| - 2) + \cdots + 1 = \frac{|x| \cdot (|x| + 1)}{2}.$$

Moreover, if we define $\sigma_2(x) = cat(\sigma_1(x), \sigma_1(x))$, then we obtain the function σ_2 that satisfies $|\sigma_2(x)| > |x| \cdot (|x| + 1)$. Then, we construct the function σ_3 that satisfies $\sigma_3(x) > |x|^k$ for any constant k by applying *Comp* to σ_2 enough times as $\sigma_3(x) = \sigma_2(\cdots(\sigma_2(x)))$. Consequently, we can obtain the function $p \in Comp^*(RecN(AGTIME))$ as $p(x) = cat(\sigma_3(x), cat(\cdots cat(\sigma_3(x), \sigma_3(x))))$. □

Now, we show the theorem by using this lemma.

Theorem 6.2.

$$FP = Comp^*(RecN(AGTIME)).$$

Proof. First, we show $Comp^*(RecN(AGTIME)) \subseteq FP$. Since FP is closed under *Comp*, it is sufficient to show $RecN(AGTIME) \subseteq FP$. To show this, we consider a function $f \in (RecN(AGTIME))$, which is obtained by applying *RecN* to $g, h \in AGTIME$. The program computing f is described as follows:

input $x, \vec{y}$
$z = g(\vec{y})$;
for(i = 1 to $|x|$){
  x_i = from first bit to ($|x| - i$) th bit of x
  $z = h(x_i, \vec{y}, z)$;
}
output z .

In this program, the length of z increases at most the sum of the length of $x, \vec{y}$ plus some constant at each repetition because $h \in AGTIME$. Since both the number of repetitions and increase at each repetition is polynomially bounded, the length of z through the execution of the program is also bounded by the polynomial in the sum of the length of $x, \vec{y}$. Consequently, this program can compute f in polynomial time.

Next, we prove $FP \subseteq Comp^*(RecN(AGTIME))$. To simulate the computation of $f \in FP$, we use the three functions *init*, *step* and *out* described in Lemma 5.1 defined for the Turing machine computing f. Here, we can assume that the Turing machine consists of one tape without loss of generality because the assumption $f \in FP$ remains valid. Thus, it is easy to see that these functions are in $AGTIME$.

By applying the operator *RecN* to *init* and *step*, we obtain the following function $F \in RecN(AGTIME)$.

$$F(0, x) = init(x),$$
$$F(t, x) = step\left(F\left(\left\lfloor \frac{t}{2} \right\rfloor, x\right)\right).$$

Whenever it carries out recursion once, the length of t decreases one. So, the length of t corresponds to the time of the computation. We can construct the time bounding function $p \in Comp^*(RecN(AGTIME))$ for f by Lemma 6.1. Namely, $f(x)$ is computable in $|p(x)|$ time.

Then, the computation of f can be simulated as $f(x) = out(F(p(x), x))$, which returns the output of the final configuration after applying *step* $|p(x)|$ times to $init(x)$. □

This result can be extended to other function complexity classes related to polynomial like $FPSPACE$. To achieve this extension, we define the following function.

Definition 6.3. *$next(e, c)$ is a function that returns a configuration when the computation begins from the configuration c and the Turing machine's head comes the next position for the first time from the position at the time of c, for the Turing machine represented by e (if there is no such configuration, then return the final configuration, in the other case undefined) .*

We describe the work of *next* below. It may simulate only one step of a Turing machine's transition. On the other hand, it may simulate exponentially many steps of a Turing machine's transition by a zigzag path. In the figure, the area of oblique lines means the computational space used till the time which corresponds to the configuration after m-th call of *next*. So, $s(n)$ space bounded computations can be simulated by calling *next* at most $s(n)$ times. Thus, polynomial space computations are simulated by calling *next* polynomial times.

The function *next* partially simulates the work of a deterministic Turing machine. This is done within limited space.

Proposition 6.4. *next* $\in AGSPACE$.

Proof. *next* simulates only the input length plus some constant space bounded computations. Moreover, the length of the configuration representation increases at most constant during the computation of *next*. Thus, *next* is in

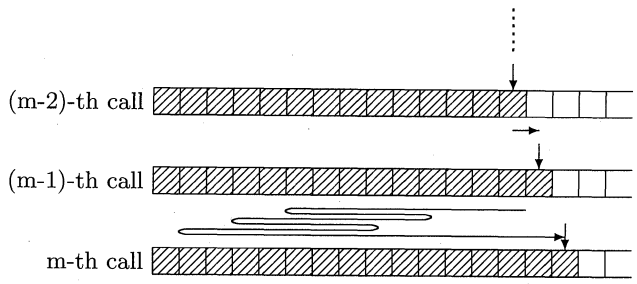


Fig. 1 The work of the function *next*.

$AGSPACE$. □

By using this function, we can obtain the following theorem. The proof is similar to the case of Theorem 6.2, but we should substitute *next* for *step*.

Theorem 6.5.

$$FPSPACE = Comp^*(RecN(AGSPACE)).$$

7. FPSPACE-Completeness

In this section, we define $FPSPACE$-completeness by using an offline multitape deterministic oracle Turing machine and show some results on $FPSPACE$-completeness.

An oracle Turing machine has two special states INPUT and OUTPUT and a special work tape called oracle tape. We consider the case when the Turing machine given a function f as oracle transfers into the state INPUT at some step of computation. Now, let x be the string of the oracle tape. Then, it transfers into the state OUTPUT and writes down the output of the function $f(x)$ on the oracle tape at one step.

We denote FP^f as the class of functions computable in polynomial time by a deterministic Turing machine with oracle tape computing a function f. Then, we define $FPSPACE$-completeness as follows.

Definition 7.1. *If we have*

$$FPSPACE \subseteq FP^f \ \wedge \ f \in FPSPACE,$$

then f is $FPSPACE$-complete.

Then, we have the following proposition.

Proposition 7.2. *For any $FPSPACE$-complete function f, we have*

$$f \in FP \Leftrightarrow FP = FPSPACE.$$

Proof. If $f \notin FP$, then $FP \subsetneq FPSPACE$ because $f \in FPSPACE$. Conversely, if $f \in FP$, then it is easy to see that $FP^f = FP$ because the computation of oracle function f can be substituted for the query of oracle. The number of query to oracle is polynomial and the simulation of the oracle computation is done in polynomial time. Thus, all functions in FP^f can be simulated in polynomial time and this implies $FPSPACE \subseteq FP^f \subseteq FP$. □

We can prove that all characteristic functions of $PSPACE$-complete languages are $FPSPACE$-complete by the prefix searching method.

Moreover, we show the following theorem.

Theorem 7.3. *next is $FPSPACE$-complete.*

Proof. From the property of *next*, polynomial space bounded computations can be simulated by calling the *next* oracle polynomial times.

More precisely, we can obtain the following polynomial time algorithm computing a function f by using *next* as oracle:

```
input x
c = the initial configuration on input x;
for(i = 0 to p(|x|)){
  c = next(e, c)
}
z = the output string of c;
output z .
```

Here, e is a representation of the Turing machine computing f in polynomial space and $p(n)$ is a polynomial function corresponding to the space complexity of e computing f.

Because $next \in FPSPACE$, we conclude that *next* is $FPSPACE$-complete. □

Corollary 7.4. $next \in FP \Leftrightarrow P = PSPACE$.

One interesting point is that *next* is $FPSPACE$-complete and at the same time it is in $AGSPACE$.

8. Concluding Remarks

We proved

$$FSPACE(s(n)) \subseteq Comp^*(BRec(FTIME(s(n)))$$

and showed that the operator *BRec* expresses the gap between FP and $FPSPACE$. Although whether $PSPACE$ strictly includes P is still an open problem, it is known that $DSPACE(s(n))$ strictly includes $DTIME(s(n))$ [7]. From this result, we have $FTIME(s(n)) \neq FSPACE(s(n))$ and thus we can observe $FTIME(t(n))$ is not closed under *Comp* or *BRec*.

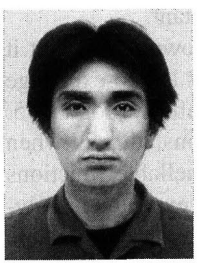
We also showed a complete function for $FPSPACE$ under FP Turing reductions. This result can be generalized. That is, the function *next* is $FSPACE(n)$-complete under $FTIME(n)$ Turing reductions. Thus, it is not in $FTIME(n)$ because $FTIME(n) \neq FSPACE(n)$.

Acknowledgements

The author is grateful to Hiroshi Imai for his valuable advice. The author thanks anonymous referees for their helpful comments. This work was partially supported by Research Fellowship for Young Scientists from Japan Society for the Promotion of Science (JSPS) and Grant-in-Aid for JSPS Fellows.

References

[1] J. Balcázar, J. Díaz, and J. Gabarró, Structural Complexity I, 2nd ed., Springer, 1994.
[2] S. Bellantoni and S. Cook, "A new recursion-theoretic characterization of the polytime functions," Computational Complexity, vol.2, no.2, pp.97–110, 1992.
[3] P. Clote and E. Kranakis, Boolean Functions and Computation Models, Springer, 2002.
[4] A. Cobham, "The intrinsic computational difficulty of functions," Proc. 1964 Congress for Logic, Methodology, and the Philosophy of Science, pp.24–30, North-Holland, 1964.
[5] A. Grzegorczyk, "Some classes of recursive functions," Rozprawy Mathematyczne, vol.4, pp.xx–xx, 1953.
[6] M. Hofmann, "Type systems for polynomial-time computation," Technical Report, University of Edinburgh, 1998.
[7] J.E. Hopcroft, W. Paul, and L.G. Valiant, "On time versus space," J. ACM, vol.24, pp.332–337, 1977.
[8] S.C. Kleene, "General recursive functions of natural numbers," Mathematical Annals, vol.112, pp.727–742, 1936.
[9] R.E. Ladner, "Polynomial space counting problems," SIAM J. Comput., vol.18, no.6, pp.1087–1097, Dec. 1989.
[10] M. Ogiwara and L.A. Hemachandra, "A complexity theory for feasible closure properties," Structure in Complexity Theory Conference, pp.16–29, 1991.
[11] R.W. Ritchie, "Classes of predictably computable functions," Transactions of the American Mathematical Society, vol.106, pp.139–173, 1963.
[12] H.E. Rose, Subrecursion: Functions and Hierarchies, Claredon Press, 1984.
[13] A.L. Selman, "Much ado about functions," Proc. Eleventh Annual IEEE Conference on Computational Complexity, pp.198–212, 1996.
[14] A.L. Selman, M.-R. Xu, and R.V. Book, "Positive relativizations of complexity classes," SIAM J. Comput., vol.12, no.3, pp.565–579, Aug. 1983.
[15] D.B. Thompson, "Subrecursiveness: Machine-independent notions of computability in restricted time and storage," Mathematical Systems Theory, vol.6, no.1, pp.3–15, 1972.
[16] S. Toda, "PP is as hard as the polynomial-time hierarchy," SIAM J. Comput., vol.20, no.5, pp.865–877, Oct. 1991.
[17] H. Vollmer and K. Wagner, "Classes of counting functions and complexity theoretic operators," Technical Report, Universitat Wurzburg, 1991.
[18] S. Zachos and A. Pagourtzis, "Combinatory complexity: Operators on complexity classes," Proc. Panhellenic Logic Symposium, 2003.



Kenya Ueno was born on October 20, 1982. He received the Bachelor of Science and Master of Information Science and Technology degrees from The University of Tokyo in 2005 and 2007, respectively. He is currently a student of Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo. His research interests include algorithms and computation theory.