On the Complexity of Iterated Insertions

Markus Holzer and Klaus-Jörn Lange

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
Sand 13, D-72076 Tübingen, Germany
email: {holzer,lange}@informatik.uni-tuebingen.de

Abstract. We investigate complexities of insertion operations on formal languages relatively to complexity classes. In this way, we introduce operations closely related to $LOG(CFL)$ and NP. Our results relativize and give new characterizations of the ways to relativize nondeterministic space.

1 Introduction

There are many close connections between the theory of formal languages and structural complexity theory [14, 17]. While it is obvious to express the complexity of classes of formal languages in terms of completeness results, it is another question to classify the complexity of operations on formal languages [4, 11, 13]. Our approach is to determine relatively to a base complexity class A. In this way, we consider two constructions: on the one hand we analyze the complexity of a single application of an operator op to A. This leads to the class $APPL(A, op)$ of all languages reducible to $op(L)$ for some $L \in A$. The drawback of this class is that it is not necessarily closed under op, even if op is idempotent. Therefore, we consider also the class $HULL(A, op)$ which is the smallest class containing A and closed under op as well as downward under logspace many-one reducibility.

In this notation, for example the relation between the Kleene star ($STAR$), nonerasing homomorphism (HOM) and the complexity classes $NSpace(\log n)$, NP, and $1DSpace(\log n)$ (the class of languages recognizable by logarithmically space bounded deterministic Turing machines with *one-way* input tape) are:

$$\begin{aligned} NSpace(\log n) &= APPL(1DSpace(\log n), STAR) \\ &= HULL(1DSpace(\log n), STAR), \end{aligned}$$

whereas for nonerasing homomorphisms we find

$$\begin{aligned} NSpace(\log n) &= APPL(1DSpace(\log n), HOM) \quad \text{and} \\ NP &= HULL(1DSpace(\log n), HOM). \end{aligned}$$

There are many more relations like this, nearly all pertaining to the classes $NSpace(\log n)$ and NP. We remark that $LOG(CFL)$, the complexity class generated by the context-free languages, has not been characterized in this way.

One of the main results of this paper will be the construction of an operation on formal languages filling this gap. The key observation to do this will be to consider operations which are iterations of simpler operations. As an example, Kleene's star operation may be regarded as the iterated application of the operation of concatenation. We will now replace concatenation by the more complex operation of *(monadic) insertion* of languages. A similar approach was made in [6, 7, 10, 15] in terms of iterated substitution. The difference is that we here are interested in complexity theoretical aspects.

Since insertion is not associative there are several possibilities to iterate the operation of insertion. One is to do it *outside-in (OI)*, i.e., to insert atomic words into composed ones, while *inside-out (IO)* iteration of insertion inserts composed words into atomic ones. It will turn out that outside-in iterated insertion characterizes NP, while inside out iterated insertion characterizes $NSpace(\log n)$. The anticipated operation characterizing $LOG(CFL)$ is now obtained by iterating the operation of *binary insertion*. Again the outside-in iteration of binary insertion characterizes NP, while the inside-out iteration now characterizes $LOG(CFL)$. In particular we obtain the following equations:

1. $NSpace(\log n) = APPL(1DSpace(\log n), IO_{MON})$
   $= HULL(NSpace(\log n), IO_{MON})$,
2. $LOG(CFL) = APPL(1DSpace(\log n), IO_{BIN}) = HULL(LOG(CFL), IO_{BIN})$,
3. $LOG(CFL) = APPL(1DSpace(\log n), OI)$,
4. $NP = APPL(DSpace(\log n), OI) = HULL(NP, OI)$, and
5. $NP = HULL(1DSpace(\log n), OI)$.

In a second part we show that all these relations relativize. It is interesting to see how the different ways to iterate insertions characterize the different ways to equip space bounded complexity classes with oracles: the two most important possibilities to relativize nondeterministic space are that of Ladner and Lynch [12] and that of Ruzzo, Simon, and Tompa [18]. These two notions carry over in a natural way to time and space bounded auxiliary pushdown automata. It turns out that the outside-in iteration of insertion corresponds to LL-relativizations while inside-out iterations pertain to RST-relativizations.

2 Preliminaries

We assume the reader to be familiar with the basics of complexity theory as contained in [1, 9, 20]. In particular, we will deal with the well-known sequence of complexity classes:

$$1DSpace(\log n) \subseteq DSpace(\log n) \subseteq NSpace(\log n) \subseteq P \subseteq NP.$$

Here $1DSpace(\log n)$, $DSpace(\log n)$, $NSpace(\log n)$, P, and NP, respectively, denote the set of all problems recognizable in one-way logarithmic space, logarithmic space, nondeterministic logarithmic space, polynomial time, and nondeterministic polynomial time, respectively.

Completeness and hardness results are always meant with respect to deterministic logspace many-one reducibilities, unless otherwise stated. $L \leq_m^{\log} M$ is used to denote the fact that L is reducible to M. For a class A let $LOG(A) := \{ L \mid \exists_{M \in A} : L \leq_m^{\log} M \}$. In addition, we use λ to denote the empty word, $|w|$ for the length of a word w, and w^R for the mirror image of w.

In the following, we will often make use of the concept of auxiliary pushdown automaton [2, 9]. Let $NauxPDA\text{-}TimeSpace(t(n), s(n))$ denote the set of all problems accepted by $O(t(n))$ time- and $O(s(n))$ space-bounded nondeterministic pushdown automata. The importance of this automaton model is demonstrated by its ability to represent the classes

$$P = NauxPDA\text{-}TimeSpace(2^{O(n)}, \log n) \quad [2] \text{ and}$$
$$LOG(CFL) = NauxPDA\text{-}TimeSpace(n^{O(1)}, \log n) \quad [19].$$

Throughout this paper, we will consider complexities of operations on formal languages. In this context, we introduce a "measure" for the complexity of an operation relative to a complexity class.

Definition 1. Let op be an operation on formal languages and A some class, then $op(A) := \{ op(L) \mid L \in A \}$.

We define $APPL(A, op)$ to be the logspace many-one closure of $op(A)$, i.e., $APPL(A, op)$ is the set $LOG(op(A))$. For iterating the APPL-operation on a class A of languages we define $APPL^0(A, op) := A$ and $APPL^{i+1}(A, op) := APPL(APPL^i(A, op), op)$.

Finally, let $HULL(A, op)$ be the smallest complexity class closed under op that contains A. In other words $HULL(A, op) := \bigcup_{i \geq 0} APPL^i(A, op)$.

Obviously $APPL(A, op) \subseteq HULL(A, op)$ and sometimes we refer to A in $APPL(A, op)$ or $HULL(A, op)$ as the *base class*.

3 Iterated Insertions

We show that several nondeterministic complexity classes can be characterized in terms of formal language theoretical operations. One of the main results of this section will be the characterization of $LOG(CFL)$. The formal language operations which will be studied in this section are natural generalizations of the concatenation operation, the so called insertion operations. Thus we define:

Definition 2. Let L_1 and L_2 be arbitrary languages. The *monadic insertion* of L_1 into L_2 is defined as $L_1 \to L_2 := \{ w_1 v w_2 \mid v \in L_1 \text{ and } w_1 w_2 \in L_2 \}$.

In contrast to operations like concatenation or shuffle, the above operation is not associative. Hence, there are several ways to iterate it. The first possibility is to insert composed words into "atomic words," i.e., to make the iteration in an *inside-out* manner. Thus, for monadic insertion we define

Inside-Out monadic insertion

1. Let $IO\textsc{mon}(L, 0) := \{\lambda\}$ and $IO\textsc{mon}(L, (i+1)) := IO\textsc{mon}(L, i) \to L$.
2. Finally set $IO\textsc{mon}(L) := \bigcup_{i \geq 0} IO\textsc{mon}(L, i)$.

The other possibility to iterate the insertion process is in a so called *outside-in* manner, i.e., to insert "atomic words" into composed ones. Thus, for the monadic insertion we define:

Outside-in monadic insertion

1. Set $OI\textsc{mon}(L,0) := \{\lambda\}$ and $OI\textsc{mon}(L,(i+1)) := L \to OI\textsc{mon}(L,i)$.
2. Finally set $OI\textsc{mon}(L) := \bigcup_{i\geq 0} OI\textsc{mon}(L,i)$.

Example 1. Let F be the finite set $\{()\}$. Then $IO\textsc{mon}(F)$ is a linear language generated by the grammar $G = (\{S\},\{(,)\},P,S)$ with the productions $P = \{S \to \lambda, S \to (S), S \to S(), S \to ()S, S \to ()\}$. On the other hand one readily verifies that $OI\textsc{mon}(F)$ equals to the Dyck set D_1.

In next subsections, we will see that the complexity of these two iterated monadic insertions lead not to the class $LOG(CFL)$, but again to $NSpace(\log n)$ and NP, only. Thus, in order to find a complete operation for $LOG(CFL)$ we have to define a more "complicated" version of insertion.

Definition 3. Let L_1, L_2, and L_3 be arbitrary languages. The *binary insertion* of L_1 and L_2 into L_3 is defined as

$$(L_1, L_2) \to L_3 := \{\, w_1uw_2vw_3 \mid u \in L_1,\ v \in L_2, \text{ and } w_1w_2w_3 \in L_3 \,\}.$$

Again, we have to possibilities to iterate the insertion process:

Inside-Out binary insertion

1. Set $IO\textsc{bin}(L,0) := \{\lambda\}$, $IO\textsc{bin}(L,1) := L$, and
   $IO\textsc{bin}(L,(i+1)) := \bigcup_{0\leq j\leq i}\big(IO\textsc{bin}(L,j), IO\textsc{bin}(L,(i-j))\big) \to L$.
2. $IO\textsc{bin}(L) := \bigcup_{i\geq 0} IO\textsc{bin}(L,i)$.

Outside-In binary insertion

1. Let $OI\textsc{bin}(L,0) := \{\lambda\}$, $IO\textsc{bin}(L,1) := L$, and
   $IO\textsc{bin}(L,(i+1)) :=$
   $$\bigcup_{0\leq j\leq 1}\big(OI\textsc{bin}(L,j), OI\textsc{bin}(L,(1-j))\big) \to OI\textsc{bin}(L,i).$$
2. $OI\textsc{bin}(L) := \bigcup_{i\geq 0} OI\textsc{bin}(L,i)$.

For the outside-in binary insertion $OI\textsc{bin}(L)$ one shows that this insertion process coincides with the outside-in monadic one. Thus, we have:

Lemma 4. *$OI\textsc{mon}(L) = OI\textsc{bin}(L)$ for arbitrary language L.*

Because of this lemma, we deal only with *one* outside-in operation in the sequel, and define $OI(L) := OI\textsc{mon}(L)$ for an arbitrary language L. Let us give a further example.

Example 2. Let F be the set of the previous example. By Lemma 4 and the definition of the OI-operation we have $OI(F) = D_1$ and an easy induction on the iteration process shows that $IO\textsc{bin}(F) = D_1$, too.

3.1 Closure under iterated insertion

In this subsection, we show that several complexity classes are closed under iterated insertion. First, we consider inside-out iterated monadic and binary insertion. In both cases, the main idea for an algorithm to check $IO_{MON}(L)$ or $IO_{BIN}(L)$ is the same. The machine that checks $IO_{MON}(L)$ membership works as follows: on input w it guesses a decomposition $w = w_1 u w_2$, checks whether $w_1 w_2 \in L$, and recursively verifies that v belongs to $IO_{MON}(L)$. Then following proposition is easy to see:

Proposition 5. *If $s(n) \geq \log n$, then $IO_{MON}(NSpace(s(n))) \subseteq NSpace(s(n))$.* □

In case of binary inside-out iterated insertion we do similarly, but now using an auxiliary pushdown automaton. On input w the machine guesses a decomposition $w_1 u w_2 v w_3$, checks whether $w_1 w_2 w_3 \in L$, and recursively verifies whether both words u and v belong to $IO_{BIN}(L)$. To do so the machine stores the begin and end of the subwords u and v on its pushdown. If the nondeterministic auxiliary pushdown automaton that accepts L is $O(t(n))$-time and $O(s(n))$ space bounded, then the machine that checks $IO_{BIN}(L)$ membership is $O(n \cdot t(n))$-time and $O(s(n))$ space bounded.

Theorem 6. *Let $s(n) \geq \log n$ and $t(n) \geq n^{O(1)}$. If L is a member of the class NauxPDA-TimeSpace$(t(n), s(n))$, then the language $IO_{BIN}(L)$ belongs to NauxPDA-TimeSpace$(n \cdot t(n), s(n))$.* □

Observe that with a little bit more advanced algorithm we can even check $OI(L)$ membership in NauxPDA-TimeSpace$(2^{O(s(n))}, s(n))$ if $L \in$ 1DSpace$(s(n))$. The only modification in the construction is, that the automaton which accepts $OI_{MON}(L)$, guesses a decomposition $u_0 w_1 u_1 w_2 u_2 \ldots u_{t-1} w_t u_{t+1}$ while the input head scans the input from left to right, and checks by simulating the one-way nondeterministic $O(s(n))$ space bounded Turing machine whether $w_1 w_2 \ldots w_t$ belongs to L. Then the machine recursively verifies—as described above—whether the words u_i, for $0 \leq i \leq t+1$, belong to $OI_{MON}(L)$.

As an immediate consequence of the characterization of $LOG(CFL)$ and P in terms of nondeterministic auxiliary pushdown automata [2, 19] we get the closure of both classes under inside-out iterated binary insertion.

Corollary 7. *$IO_{BIN}(LOG(CFL)) \subseteq LOG(CFL)$ and $IO_{BIN}(P) \subseteq P$.* □

At this point we want to mention two things: (1) The construction presented to check IO_{BIN}-membership can be generalized to IO-membership for insertions where the possible insertion points into a word is constantly bounded. Hence, e.g., $LOG(CFL)$ is also closed under *iterated inside-out ternary insertion.* (2) Moreover, we want to point out that $DSpace(\log^2 n)$ is closed under both types of inside-out iterated insertion.

Finally, we mention the closure of NP under OI-operation. This proof is straight-forward and is left to the reader.

Proposition 8. *$OI(NP) \subseteq NP$.* □

3.2 Hardness of iterated insertion

For technical reasons we introduce a notation, the so-called insertion tree, which is helpful in analyzing inside-out iterated monadic and binary insertion.

Definition 9. An *insertion tree over a terminal alphabet* T is a construct $I = (V, h, x_0, label, T)$, where

1. (V, h, x_0) is a tree rooted in $x_0 \in V$, i.e., $h : V \to V$ points every node to its father, $h(x_0) = x_0$ and for all $x \in V$ there exists an $n \geq 0$ such that $h^n(x) = x_0$.
2. $label : V \to T^*(VT^*)^*$ is the labelling function.

For an insertion tree I we define the functions

1. $word : V \to T^*$, by $word(x) := w_0 w_1 \dots w_t$, if $label(x) = w_0 x_1 w_1 \dots w_{t-1} x_t w_t$,
2. $yield : V \to T*$ inductively by $yield(x) = w_0 yield(x_1) w_1 \dots w_{t-1} yield(x_t) w_t$, if $label(x) = w_0 x_1 w_1 \dots w_{t-1} x_t w_t$.

An insertion tree I is called (1) *monadic* if the mapping *label* only takes images in $T^* \cup T^* V T^*$ and (2) *binary* if it only takes images in $T^* \cup T^* V T^* V T^*$.
Obviously, for any language we have:

Lemma 10. *Let* $L \subseteq T^*$ *and* $w \in T^*$. *The word* w *belongs to* $IO\textsc{mon}(L)$ *(*$IO\textsc{bin}(L)$, $OI(L)$, *respectively) if and only if there exists a monadic (binary, arbitrary, respectively) insertion tree* $I = (V, h, x_0, label)$ *such that* $yield(x_0) = w$ *and for all* $x \in V$ *we have* $word(x) \in L \cup \{\lambda\}$. □

Hardness of the $IO\textsc{mon}$-operation The following theorem shows close relation of $IO\textsc{mon}$ and $NSpace(\log n)$. We state it without proof, since it is very similar to that on showing the analogous results of the Kleene star operation [4, 16].

Theorem 11. *There is a language* L_M *in* $1DSpace(\log n)$ *such that* $IO\textsc{mon}(L_M)$ *is* $NSpace(\log n)$*-complete.* □

Essentially the strings in L_M are of the form $b^n\$(a^*\$b^*\$)^*\#(\$a^*\$b^*)^*\$a^n$. The Kleene closure of this language is $NSpace(\log n)$-complete. But the power of the $IO\textsc{mon}$-operation makes it necessary to extend the construction in order to avoid "wrong" insertion. The details are similar to, although less extensive than, those provided in Theorem 14. Using Proposition 5 we get:

Corollary 12. $NSpace(\log n) = APPL(1DSpace(\log n), IO\textsc{mon})$
$= HULL(NSpace(\log n), IO\textsc{mon})$. □

This implies the following equalities: $APPL(1DSpace(\log n), IO\textsc{mon}) = APPL(DSpace(\log n), IO\textsc{mon}) = HULL(1DSpace(\log n), IO\textsc{mon})$. Later we will see that the OI-operation is much more sensitive with respect to this difference.

Hardness of the IO_{BIN}-operation Before we come to one of the main results of this paper establishing a close link between iterated binary insertion and polynomially time bounded auxiliary pushdown automata we need the following lemma.

Lemma 13. *There exists a $LOG(CFL)$-complete context-free language which is generated by a context-free grammar $G = (N, T, P, S)$, with nonterminals N, terminals T, axiom S, and production set $P \subseteq N \times (T \cup TN^2)$.*

Observe, that context-free grammars which satisfy $P \subseteq N \times (T \cup TN^2)$ can only generate words of odd length. Hence such a normal-form for context-free grammars does not exist in general.

Proof. Without loss of generality one can assume that the $LOG(CFL)$-complete context-free language L is generated by a grammar $G = (N, T, P, S)$ being in 2-standard Greibach normal-form, i.e.,

$$P \subseteq N \times \big((T \cup T(N \setminus \{S\}) \cup T(N \setminus \{S\})^2\big).$$

We will use new symbols $\#$, X with subscripts which are not contained in N and T. We first modify the production set P in the following way:

$$\begin{aligned} P_1 := &\{ A \to aa \mid A \to a \in P \} \cup \{ A \to aaB \mid A \to aB \in P \} \cup \\ &\{ A \to aaBC \mid A \to aBC \in P \}. \end{aligned}$$

Observe that the language $G_1 = (N, T, P_1, S)$ is $LOG(CFL)$-complete, too. Then let us construct a grammar G_2 with $L(G_2) = L(G_1)\{\#\}$. Every word that belongs to $L(G_1)\{\#\}$ has odd length. Set

$$\begin{aligned} P_2 := &\{ X_a \to a \mid a \in T \} \cup \\ &\{ X_{Ab} \to aX_aX_b, X_{bA} \to bX_aX_a, X_{bDA} \to bX_{Da}X_a \mid A \to aa \in P_1 \} \cup \\ &\{ X_{Ab} \to aX_aX_{Bb}, X_{bA} \to bX_aX_{aB}, X_{bDA} \to bX_{Da}X_{aB} \mid \\ &\qquad\qquad\qquad\qquad\qquad\qquad A \to aaB \in P_1 \} \cup \\ &\{ X_{Ab} \to aX_{aB}X_{Cb}, X_{bA} \to bX_aX_{aBC}, X_{bDA} \to bX_{Da}X_{aBC} \mid \\ &\qquad\qquad\qquad\qquad\qquad\qquad A \to aaBC \in P_1 \} \end{aligned}$$

and let

$$G_2 = (N \cup \{X_{S\#}\} \cup \{ X_a, X_{aB}, X_{aBC} \mid a \in T \text{ and } B, C \in N \}, T \cup \{\#\}, P_2, X_{S\#}).$$

Then P_2 has the expected normal-form, and obviously $L(G_2)$ is $LOG(CFL)$-complete. ☐

Theorem 14. *There is a set L_B in $1DSpace(\log n)$ such that both $OI_{MON}(L_B)$ and $IO_{BIN}(L_B)$ are $LOG(CFL)$-complete.*

Proof. We start with a $LOG(CFL)$-complete language L_1 which is generated by a context-free grammar $G = (N, T, P, S)$ satisfying the requirement of the above lemma.

Observe that we do not require L_1 to be a hardest language in the sense of Greibach [6], but only to be $LOG(CFL)$-complete. Our construction closely follows that one of Greibach although we have to be more careful due to the nonsequential nature of iterated insertion (compared to inverse homomorphism).

In the following we will need new symbols \$, #, 0, 2, and F contained in neither N nor T. In addition, let $\bar{N} := \{\, \bar{A} \mid A \in N \cup \{F\}\,\}$ be a disjoint copy of $N \cup \{F\}$.

For an arbitrary $a \in T$ consider all productions $p_1, \ldots, p_k$ such that $p_i \in N \times (a \cup aN^2)$ for each $1 \leq i \leq k$. For each $i \geq 0$ and each $1 \leq j \leq k$ define

$$f_i^a(j) := \begin{cases} \bar{A}2CB\$^i & \text{if } p_j \text{ equals } A \to aBC^1 \\ \bar{A}0\$^i & \text{if } p_j \text{ equals } A \to a \end{cases}$$

and

$$f_i'^a(j) := \begin{cases} \lambda & \text{if } p_j \text{ equals } A \to aBC \\ \bar{A}2FF\$^i & \text{if } p_j \text{ equals } A \to a. \end{cases}$$

Further on we set

$$g_i^a := \$^i f_i^a(1) f_i^a(2) \ldots f_i^a(k) \quad \text{and} \quad g_i'^a := \$^i f_i'^a(1) f_i'^a(2) \ldots f_i'^a(k).$$

For a word $w = a_1 \ldots a_n \in T^*$ with $a_i \in T$ we define

$$h(a_1 \ldots a_n) := S\#g_1^{a_1}\#g_2^{a_2}\# \ldots \#g_{n-1}^{a_{n-1}}\#g_n'^{a_n}\#\$^{n+1}\bar{F}0\$^{n+1}\#\$^{n+2}\bar{F}0.$$

Obviously, the mapping is computable in deterministic logarithmic space.

Now we define the language L_B. First let

$$R := \{\, \bar{A}0 \mid A \in N \cup \{F\}\,\} \cup \{\, \bar{A}2BC \mid A \in N \text{ and } B, C \in N \cup \{F\}\,\}$$

and for $i \geq 0$ set $R_i := \$^i(R\$^i)^*$. Finally, define

$$L_B := \{\, A\alpha\#\beta\bar{A}c \mid \exists i \geq 1 : \alpha \in R_{i-1}, \beta \in R_i; A \in N \cup \{F\}; c \in \{0, 2\}\,\}.$$

Obviously, L_B is a member of *1DSpace*$(\log n)$.

The idea underlying this construction is to translate a derivation tree of G into an insertion tree as follows: if A is a nonterminal labelling the root of a subderivation tree $\mathcal{D}$ and B and C are the root-labels of the left and right subtrees $\mathcal{D}_L$ and $\mathcal{D}_R$, there will be three elements of L_B, namely $w_A := A\alpha\#\beta\bar{A}2$, $w_B := B\alpha'\#\beta'\bar{B}c'$, and $w_C := C\alpha''\#\beta''\bar{C}c''$. The corresponding part of the insertion tree will consist of w_A on top, w_C inserted at the very right end of w_A, and w_C inserted after the first symbol of w_C, which is the symbol C. That is left brothers become the left sons of the right brothers. This is illustrated in Figure 1.

[0] Observe the inversion of B and C.

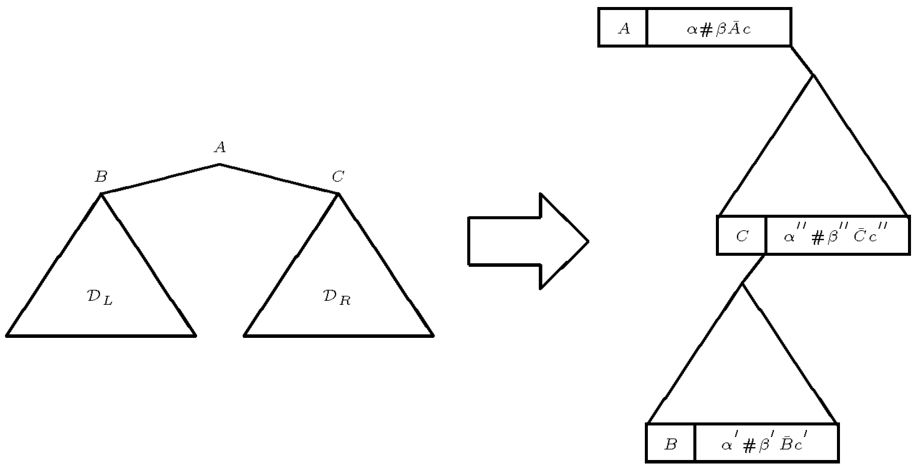

Fig. 1. The conversion of a derivation tree into an insertion tree.

Now we have to prove $w \in L_1$ if and only if $h(w) \in OI(L_B)$ if and only if $h(w) \in IO_{BIN}(L_B)$. It is easy to show that $w \in L_1$ implies $h(w) \in IO_{BIN}(L_B)$ and hence $h(w) \in OI(L_B)$. The converse makes use of the many additional features which we added to Greibach's construction [6].

Let us assume $h(w) \in OI(L_B)$. Then there exists an insertion tree $I = (V, h, x_0, label)$ with $yield(x_0) = h(w)$ and $word(x) \in L_B$ for all $x \in V$. We proceed in several stages:

Step 1 Let $x \in V$ and $label(x) = w_0 x_1 w_1 \ldots x_t w_t$. Due to the increasing length of the $\$^i$-blocks it is easy to see that for a typical element $A\alpha\#\beta\bar{A}c \in L_B$ there are only three places to perform insertion: before A, behind A, or after the c. Otherwise the resulting word could no longer be a subword of $h(w)$.

Step 2 We can rearrange I in the following way: First, I no longer has nodes inserting the empty word, and second whenever two nodes in I are directly neighboured, i.e., the concatenation of the yields is a subword of $h(w)$, the right one is inserted as a son at the very right end of the left one. The way to rearrange I is indicated in the Figures 2 and 3.

Step 3 After the rearrangement, each node x of I is either a leaf or has at most two sons, one inserted at the right end of $word(x)$ and one after the first symbol of $word(x)$. Let x be in V and $word(x) = A\alpha\#\beta\bar{A}c \in L_0$. Then we set $nonterminal(x) := A$ and $index(x) := i$ if $\beta \in R_i$ (and $\alpha \in R_{i-1}$). It is not hard to work out that $nonterminal(x_0) = S$.

Step 4 The structure of the mapping h enforces the following claim:

Claim 1. *If $x \in V$ with $word(x) = A\alpha\#\beta\bar{A}c$ possesses a right son y, inserted after the symbol c, then (1) $c = 2$ and (2) y possesses a left son z inserted after the first symbol of $word(y)$.*

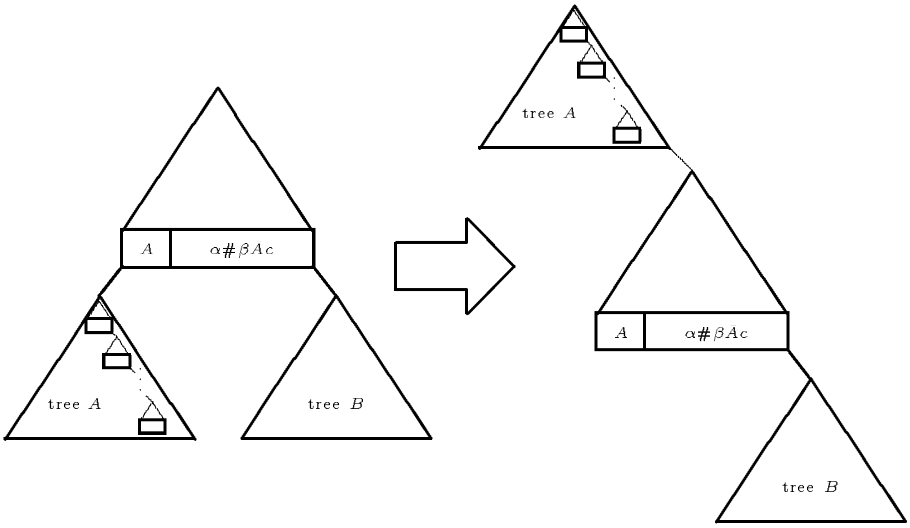

Fig. 2. First rearrangement of the insertion tree I.

Proof. If $word(y) = C\alpha''\#\beta''\bar{C}c''$, then $\bar{A}cC$ must be a subword of $h(w)$, since otherwise nothing is inserted left of C. Hence c cannot be 0, but must be 2. But then we need a second nonterminal following the symbol 2. This can be only provided by the insertion of a left son z after symbol c. □

Step 5 Inductively we define the mapping $derive : V \to T^*$ by $derive(x) := a_i$, if x does not possess a right son in I. Here $i := index(x)$. If $i \geq n+1$, we set $a_i := \lambda$. If x possesses a right son y we know by the previous step that y in turn possesses a left son z. In this case we define $derive(x) := a_i derive(z) derive(y)$. The reader may verify that $derive(x_0) = w$!

Step 6 For each $x \in V$ with $index(x) \leq n$ we have $A \Rightarrow_G^* derive(x)$. In particular $S = nonterminal(x_0) \Rightarrow_G^* w$, i.e., $w \in L_1$.

Proof. If x has no right son, then $word(x) = A\alpha\#\beta\bar{A}0$ for some $\alpha \in R_{i-1}$, $\beta \in R_i$, and $i := index(x)$. Hence, $\$^i\bar{A}0$ is a subword of $h(w)$ and $g_i^{a_i}$. This implies that $A \to a_i$ is in P. Hence $A \Rightarrow_G^1 a_1 = derive(x)$.
If x possesses a right son y with $nonterminal(y) = C$, then by Step 4 the node y has a left son z with $nonterminal(z) = B$. Then we have that $\$^i\bar{A}2CB$ is a subword of $h(w)$ and hence of $g_i^{a_i}$. This implies $A \to a_iBC$ is in P. Hence, by induction $A \Rightarrow_G^1 a_iBC \Rightarrow_G^* a_i derive(z) derive(y) = derive(x)$. □

Using Corollary 7 we get:

Corollary 15. *1.* $LOG(CFL) = APPL(1DSpace(\log n), IO_{BIN})$
$= HULL(LOG(CFL), IO_{BIN})$.
2. $LOG(CFL) = APPL(1DSpace(\log n), OI)$. □

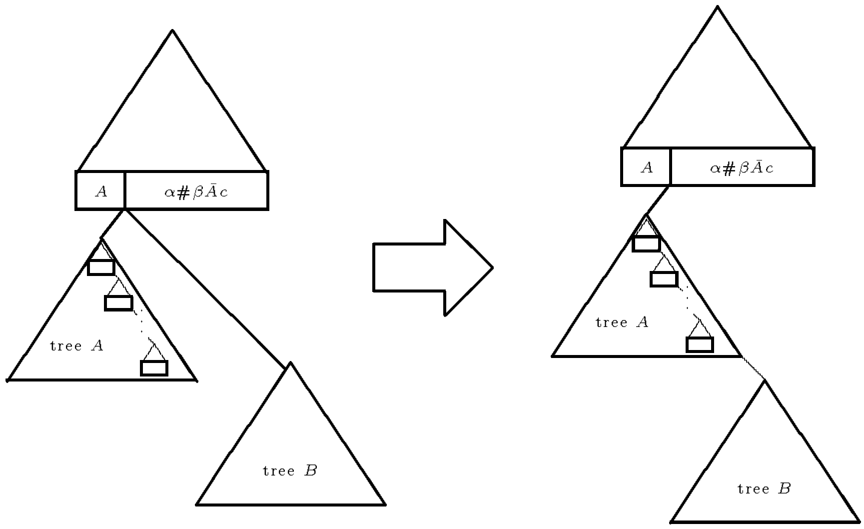

Fig. 3. Second rearrangement of the insertion tree I.

Hardness of the OI-operation In this sub-subsection, we will exhibit some crucial differences in the structural behaviour of OI compared with IO_{MON} and IO_{BIN}.

Theorem 16. $NP = APPL(DSpace(\log n), OI)$
$= HULL(1DSpace(\log n), OI) = HULL(NP, OI)$.

Proof. We first show the inclusion $NP \subseteq APPL(DSpace(\log n), OI)$. Let L_1 and L_2 bet the sets:

$$L_1 := \{\, \$a_1\$\$a_2\$\$\ldots\$\$a_k\$\#b\# \mid a_1, \ldots, a_k, b \text{ are binary numbers with } \textstyle\sum_{i=1}^{k} a_i = b \,\} \quad \text{and}$$
$$L_2 := \{\, \$a\$ \mid a \text{ is a binary number} \,\}.$$

Now set $L_{OI} := L_1 \cup L_2$. Obviously, L_{OI} belongs to $DSpace(\log n)$ and language $OI(L_{OI}) \cap \big((\$\{0,1\}^*\$)^*\#\{0,1\}^*\#\big)$ is the NP-complete subset-sum problem (see, e.g., [20]). Hence, $OI_{MON}(L_{OI})$ is NP-complete, too.

The inclusion $APPL(DSpace(\log n), OI) \subseteq HULL(1DSpace(\log n), OI)$ follows since the former class is included in $APPL^2(1DSpace(\log n), OI)$. Finally, $HULL(1DSpace(\log n), OI) \subseteq HULL(NP, OI)$ is trivial and to close the circle we use Proposition 8 and the fact that NP is closed under deterministic logarithmic space bounded reducibilities, which gives us $HULL(NP, OI) \subseteq NP$. □

We want to mention that the above given construction even works with an "unbounded" variant of insertion, i.e., the number of insertion points are not bounded any more. Moreover, if one modifies set L_2 to be $\{\, \$a_1\$\$a_2\$\$\ldots\$\$a_k\$ \mid$

$a_1, a_2, \ldots, a_k$ are binary numbers $\}$, then one obtains subset-sum with the shuffle operation (SHU). Since NP is closed under shuffle it equals the complexity class $APPL(DSpace(\log n), SHU)$. This strengthens a result in [8].

In the light of construction following Theorem 6 we should not hope to find a language L in $1DSpace(\log n)$ with an NP-complete set $OI(L)$, since this would imply $LOG(CFL) = P = NP$.

This sensitivity of the OI-operation with respect to the used base class leads to surprising phenomena: OI compared to IO_{MON} is idempotent, i.e., $OI(OI(L)) = OI(L)$ while in general $IO_{MON}(L) \subseteq IO_{MON}(IO_{MON}(L))$ for a language L. But on the other hand, we have $APPL(1DSpace(\log n), IO_{MON}) = APPL^2(1DSpace(\log n), IO_{MON})$ while $APPL(1DSpace(\log n), OI) = LOG(CFL)$ seems to be different from the class $APPL^2(1DSpace(\log n), OI) = NP$.

3.3 Relativizations

We show that all the relations found in the previous section relativize. For space bounded complexity classes, there are two main possibilities to relativize them, i.e., to equip space bounded machines with an oracle mechanism. In the approach of Ladner and Lynch [12], further called *LL-relativization*, the machine may use all of its power to generate oracle queries, while in the approach of Ruzzo, Simon, and Tompa [18], further called *RST-relativization*, the queries have to be generated deterministically. As usual, the use of parentheses is reserved for the LL-mechanism, while the use of the RST-relativization is indicated by using angles. Hence, for an arbitrary oracle set A one gets, e.g., in case of nondeterministic logspace bounded Turing machines the LL-relativized class $NSpace(\log n)^{(A)}$ and the RST-relativized version $NSpace(\log n)^{\langle A\rangle}$, respectively. Observe that in case of deterministic logspace bounded machines both relativizations coincide.

In [13] it was shown that the relations

1. $NP = APPL(DSpace(\log n), HOM) = HULL(DSpace(\log n), HOM)$,
2. $NSpace(\log n) = APPL(1DSpace(\log n), HOM)$,
3. $NP = HULL(1DSpace(\log n), HOM)$, and
4. $NSpace(\log n) = APPL(DSpace(\log n), STAR)$
   $= HULL(DSpace(\log n), STAR)$

relativize, i.e, for an arbitrary oracle set A we have:

1. $NP^{(A)} = APPL(DSpace(\log n)^{(A)}, HOM) = HULL(DSpace(\log n)^{(A)}, HOM)$,
2. $NSpace(\log n)^{(A)} = APPL(1DSpace(\log n)^{(A)}, HOM)$,
3. $NP^{(A)} = HULL(1DSpace(\log n)^{(A)}, HOM)$, and
4. $NSpace(\log n)^{\langle A\rangle} = APPL(DSpace(\log n)^{(A)}, STAR)$
   $= HULL(DSpace(\log n)^{(A)}, STAR)$.

Observe that in the fourth relation the RST- and and in the second relation the LL-relativization is used.

We will see this pattern again, when replacing nonerasing homomorphism by outside-in iterated insertion and the Kleene closure by inside-out iterated insertion. Before we can state our theorem, we need the following definition:

Definition 17. A *doubly RST-restricted* nondeterministic polynomially time bounded logspace auxiliary oracle pushdown automaton is a nondeterministic polynomially time and logspace bounded pushdown automaton equipped with an oracle mechanism (tape, query- and answer states), which is not allowed to use nondeterminism or its pushdown store while writing on its oracle tape[2].

The class of languages reducible to an oracle set A via a doubly RST-restricted nondeterministic polynomially time bounded logspace augmented oracle pushdown automaton is denoted by $\mathit{NauxPDA\text{-}TimeSpace}(n^{O(1)}, \log n)^{\langle A \rangle}$.

Theorem 18. *For an arbitrary oracle set A we have:*

1. $NP^{\langle A \rangle} = \mathit{HULL}(\mathit{1DSpace}(\log n)^{\langle A \rangle}, \mathit{OI})$.
2. $NP^{\langle A \rangle} = \mathit{APPL}(\mathit{DSpace}(\log n)^{\langle A \rangle}, \mathit{OI}) = \mathit{HULL}(\mathit{DSpace}(\log n)^{\langle A \rangle}, \mathit{OI})$.
3. $\mathit{NSpace}(\log n)^{\langle A \rangle} = \mathit{APPL}(\mathit{1DSpace}(\log n)^{\langle A \rangle}, \mathit{IO}_{MON}) = \mathit{HULL}(\mathit{1DSpace}(\log n)^{\langle A \rangle}, \mathit{IO}_{MON})$.
4. $\mathit{NauxPDA\text{-}TimeSpace}(n^{O(1)}, \log n)^{\langle A \rangle} = \mathit{APPL}(\mathit{1DSpace}(\log n)^{\langle A \rangle}, \mathit{IO}_{BIN}) = \mathit{HULL}(\mathit{1DSpace}(\log n)^{\langle A \rangle}, \mathit{IO}_{BIN})$.

Idea of Proof. In the cases 1 till 3 it is possible to put oracle queries in the sets constructed in the Theorem 12 and 16, very similar to the methods used in [13]. The idea to prove 4 is a bit more complicated since one has to deal with pushdown automata instead of grammars. That is, one has to combine the triple-construction with the inside-out iterated binary operation. □

4 Conclusions

We investigated the computational power of operations on formal languages with respect to simple complexity classes. We introduced two new operations which were closely related to $\mathit{LOG}(\mathit{CFL})$ and NP. We mention in passing that similar results can be obtained when iterating the operation of deletion, defined in correspondence to that of insertion.

There are several questions left open. An interesting aspect is the treatment of abstract storage types. Most results concerning context-free languages and pushdown automata have been shown to remain valid if we replace in the automaton the pushdown store by another arbitrary storage device. For languages this led to the notions of abstract families of automata or of automata with abstract storage [3, 5]. Essentially this led to the construction of *permissible sequences of basic instructions* of a storage type. For instance, the Dyck sets are the languages of correct computations of a pushdown store. In our framework this leads to the task to construct to an abstract storage type X a characteristic operation op_{X}, which would play for X that role which inside-out iterated binary insertion plays for the context-free languages. The advantage of this approach is that all results obtained in this way would relativize.

[2] This is equivalent to a logarithmic bound on the oracle queries, if the oracle has access to the input word.

References

1. J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1988.
2. S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the ACM*, 18:4–18, 1971.
3. J. Dassow and K.-J. Lange. Complexity of automata with abstract storages. In *Proceedings of the 8th Fundamentals of Computing Theory*, number 529 in LNCS, pages 200–209. Springer, 1991.
4. P. Flajolet and J. Steyaert. Complexity of classes of languages and operators. Rap. de Recherche 92, IRIA Laboria, 1974.
5. S. Ginsburg. *Algebraic and Automata-Theoretic Properties of Formal Languages*. North-Holland, Amsterdam, 1975.
6. S. A. Greibach. The hardest context-free language. *SIAM Journal on Computing*, 2(4):304–310, December 1973.
7. J. Gruska. A characterization of context-free languages. *Journal of Computer and System Sciences*, 5:353–364, 1971.
8. D. Haussler and M. K. Warmuth. On the complexity of iterated shuffle. *Journal of Computer and System Sciences*, 28:345–358, 1984.
9. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
10. L. Kari. Insertion operations: closure properties. *Bulletin of the European Association for Theoretical Computer Science*, 51:181–191, 1993.
11. K.-I. Ko. On some natural complete operations. *Theoretical Computer Science*, 37:1–30, 1985.
12. R. Ladner and N. Lynch. Relativization of questions about log space computability. *Mathematical Systems Theory*, 10:19–32, 1976.
13. K.-J. Lange. Decomposition of nondeterministic reductions. *Theoretical Computer Science*, 58:175–181, 1988.
14. K.-J. Lange. Complexity structure in formal language theory. In *Proceedings of the 8th Annual Structure in Complexity Theory*, pages 224–238. IEEE Computer Society Press, May 1993.
15. I. P. McWhirter. Substitution expressions. *Journal of Computer and System Sciences*, 5:629–637, 1971.
16. B. Monien. About the deterministic simulation of nondeterministic $(\log n)$-tape bounded Turing machines. In *2-te GI Fachtagung Automatentheorie und Formale Sprachen*, number 33 in LNCS, pages 118–126. Springer, 1975.
17. B. Monien and I. Sudborough. The interface between language theory and complexity theory. In R. V. Book, editor, *Formal Languages—Perspectives and Open Problems*, pages 287–324, New York, 1980. Academic Press.
18. W. L. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierarchies and probabilistic computations. *Journal of Computer and System Sciences*, 28:216–230, 1984.
19. I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25:405–414, 1978.
20. K. Wagner and G. Wechsung. *Computational Complexity*. Mathematics and its applications (East Europeans series). VEB Deutscher Verlag der Wissenschaften, Berlin, 1986.

This article was processed using the LaTeX macro package with LLNCS style