Theoretical Computer Science 262 (2001) 501–523

Theoretical Computer Science

www.elsevier.com/locate/tcs

Dot operators [☆]

Bernd Borchert[a], Riccardo Silvestri[b,*]

[a] *Universität Heidelberg, Im Neuenheimer Feld 294, 69120 Heidelberg, Germany*
[b] *Dipartimento di Matematica Pura ed Applicata, via Vetoio, Università de L'Aquila, Coppito, L'Aquila, Italy*

Received 23 April 1999; revised 6 June 2000; accepted 18 July 2000
Communicated by O. Watanabe

Abstract

Well-known examples of dot operators are the existential, the counting, and the BP-operator. We will generalize this notion of a dot operator so that every language A will determine an operator $A\cdot$. In fact, we will introduce the more general notion of promise dot operators for which the BP-operator is an example. Dot operators are a refinement of the leaf language concept because the class determined by a leaf language A equals $A \cdot \mathrm{P}$. Moreover, we are able to represent not only classes but reducibilities, in fact most of the known polynomial-time reducibilities can be represented by dot operators. We show that two languages determine the same dot operator if and only if they are reducible to each other by polylog-time computable monotone projections. © 2001 Elsevier Science B.V. All rights reserved.

1. Introduction

Well-known examples of dot operators are the existential operator [35], the counting operator [35], and the BP-operator [22]. For a survey see the book of Köbler et al. [17]. Such operators have been used (for example in [35, 22, 28, 29, 17, 8]) to define new classes and hierarchies, for instance, the counting hierarchy [35]. In general, dot operators have been used as a tool to study the relationship between complexity classes. In Section 2 we will generalize this notion of a dot operator so that every language A will determine an operator $A\cdot$. In fact, we will introduce the more general notion of promise dot operators for which the BP-operator is an example.

In Section 3 we will study properties of dot operators. We will see for example that dot operators turn out to be a refinement of the leaf language concept (see

[☆] A preliminary version with the title "The General Notion of a Dot-Operator" was presented at the *12th* Annual IEEE Conference on Computational Complexity (1997).

[*] Corresponding author.

E-mail addresses: bb@math.uni-heidelberg.de (B. Borchert), silvestri@dsi.uniromal.it (R. Silvestri).

[6, 33, 12, 14, 16, 21, 5], and the recent survey [34]) because the class determined by a leaf language A equals $A \cdot \mathrm{P}$. Furthermore, we show that dot operators are closed under composition, and that complementary dot operators keep the property of classes to have a many-one complete set.

In Section 4 we show that for two languages A, B it holds: $A \cdot \mathscr{C}$ is a subset of $B \cdot \mathscr{C}$ for all classes $\mathscr{C}$ if and only if A is reducible to B by polylog-time uniform monotone projections. It can be viewed as an analog of the main result (for the complementary case) of Bovet et al. [6] about the relativization of leaf language classes.

In Section 5 we show that dot operators are able to represent not only classes but reducibilities. For example, we will construct a language T such that for all classes $\mathscr{C}$ the class $T \cdot \mathscr{C}$ is the polynomial-time Turing closure $\mathrm{P}(\mathscr{C})$ of $\mathscr{C}$. In fact, we will see that most of the known polynomial-time reducibilities can be represented by dot operators. This shows that with dot operators we get a *proper* refinement of the leaf language concept. Also, this concept of representation allows to talk about the computational complexity of a reducibility.

2. The definition of dot operators

A well-known operator is the existential operator $\exists\cdot$. For a class $\mathscr{C}$ the class $\exists \cdot \mathscr{C}$ is defined the following way (where $\langle , \rangle$ is a usual pairing function).

The class $\exists \cdot \mathscr{C}$ is the set of languages L for which there is a set H in $\mathscr{C}$ and a polynomial p such that

$$x \in L \iff \text{there is a } y \text{ of size at most } p(|x|) \text{ such that } \langle x, y\rangle \in H.$$

It holds for example $\exists \cdot \mathrm{P} = \mathrm{NP}$, $\exists \cdot \mathrm{NP} = \mathrm{NP}$, and $\exists \cdot \text{co-NP} = \Sigma_2^p$. The existential operator was first defined by Wagner [35] using the symbol $\vee$, afterwards the above notation – with or without the dot – was used for example in [3, 17].

Convention. In this paper, we will assume that classes are closed downward w.r.t polynomial-time many-one reducibility $\leqslant_m^p$, i.e. from $A \leqslant_m^p B$ and $B \in \mathscr{C}$ it follows $A \in \mathscr{C}$. Or equivalently, if we have an operator applied to a class $\mathscr{C}$ then implicitly we assume that $\mathscr{C}$ stands for the downward closure $\leqslant_m^p(\mathscr{C})$ of $\mathscr{C}$. Moreover, we assume that classes are not included in $\{\emptyset, \Sigma^*\}$.

Under this assumption the above definition is equivalent to the following one. The main reason to introduce this slightly different definition – whose additional feature is the length delimiting function l – is that it is more flexible and slightly more powerful.

The class $\exists \cdot \mathscr{C}$ is the set of languages L for which there is a language H in $\mathscr{C}$ and a polynomial-time computable function $l : \Sigma^* \to \mathbf{N}$ (numbers are represented in binary) such that

$$x \in L \iff \text{there is a binary number } y,\ 1 \leqslant y \leqslant l(x) \text{ such that } \langle x, y\rangle \in H.$$

This definition can be rewritten to the following one (Hx denotes the letter 0 if $x \notin H$ and the letter 1 if $x \in H$; $\overline{0^*}$ is the language consisting of the words which contain the letter 1).

The class $\exists \cdot \mathscr{C}$ is the set of languages L for which there is a language H in $\mathscr{C}$ and a polynomial-time computable function $l : \Sigma^* \to \mathbf{N}$ (numbers are represented in binary) such that

$$x \in L \iff H\langle x,1\rangle H\langle x,2\rangle \cdots H\langle x,l(x)\rangle \in \overline{0^*}.$$

The reason to rewrite the definition of the existential operator is that now we get other well-known dot operators just by replacing in the above definition the language $\overline{0^*}$ by other languages. For example, the reader may verify that we get the counting operator C$\cdot$ (also originally from Wagner in [35], see also [20, 30]) if we replace $\overline{0^*}$ by the language which consists of the words which contain at least as many 1's as 0's. Likewise, we get the universal operator $\forall\cdot$ (the exact counting operator $\mathrm{C}_=\cdot$, the parity operator $\oplus\cdot$) after replacing $\overline{0^*}$ by the language consisting of the words which do not contain the letter 0 (the language consisting of the words which have as many 1's as 0's, the language consisting of the words which have an odd number of 1's).

By these examples, it seems reasonable to give the following general definition which assigns a dot operator $A\cdot$ to any language A.

Definition 2.1 (*complementary dot operator*). For a language A define the *complementary dot operator $A\cdot$*, which maps classes to classes, the following way. For a class of languages $\mathscr{C}$ let $A \cdot \mathscr{C}$ be the class consisting of the languages L for which there is a language $H \in \mathscr{C}$ and a polynomial-time computable function $l : \Sigma^* \to \mathbf{N}$ (numbers are represented in binary) such that for all x it holds

$$x \in L \iff H\langle x,1\rangle H\langle x,2\rangle \cdots H\langle x,l(x)\rangle \in A.$$

Examples of complementary dot operators

- Our typical example above was the existential operator $\exists\cdot = \overline{0^*}\cdot$. It holds for example, $\exists \cdot \Pi_k^p = \Sigma_{k+1}^p, \exists \cdot \Sigma_k^p = \Sigma_k^p$, and $\exists \cdot \mathrm{PSPACE} = \mathrm{PSPACE}$.
- Dually, we have the universal operator $\forall\cdot = 1^*\cdot$. For the universal operator it holds for example $\forall \cdot \Pi_k^p = \Pi_k^p, \forall \cdot \Sigma_k^p = \Pi_{k+1}^p$, and also of course $\forall \cdot \mathrm{PSPACE} = \mathrm{PSPACE}$.
- Also the well-known counting operators C$\cdot$ and $\mathrm{C}_=\cdot$ were mentioned above. It was already mentioned that $\mathrm{C}\cdot = \{w \mid \#_1(w) \geqslant \#_0(w)\}\cdot$ and $\mathrm{C}_= \cdot = \{w \mid \#_1(w) = \#_0(w)\}\cdot$, where $\#_0(w)$ ($\#_1(w)$) denotes the number of 0's (1's) in a word w. It holds for example $\mathrm{C} \cdot \mathrm{P} = \mathrm{PP}$ and $\mathrm{C}_= \cdot \mathrm{P} = \mathrm{C}_=\mathrm{P}$.
- The operators $\mathrm{MOD}_m\cdot$ are characterized by $\mathrm{MOD}_m \cdot = \{w \mid \#_1(w) = 0 \pmod m\}\cdot$. A special case is $\oplus \cdot = \mathrm{MOD}_2\cdot$. It holds for example $\mathrm{MOD}_m \cdot \mathrm{P} = \mathrm{MOD}_m\ \mathrm{P}$.
- The language $1\Sigma^*$ (the set of words starting with a 1) determines the *identity operator* id$\cdot$, it holds $1\Sigma^* \cdot \mathscr{C} = \mathscr{C}$ (by our convention all classes $\mathscr{C}$ are closed downward w.r.t. $\leqslant_m^p$). Dually, the language $0\Sigma^*$ determines the *complementation operator* co$\cdot$

(attention: the complementation operator is a complementary operator). It holds that $0\Sigma^* \cdot \mathscr{C}$ is the set of complements of the languages in $\mathscr{C}$. Note that $\forall\cdot = \mathrm{co}\cdot \circ \exists \cdot \circ\, \mathrm{co}\cdot$.

- We will define in Section 5, Theorem 5.2(a)(x) a language T that represents the Turing-reducibility $\leqslant_T^p$, that is, for all the languages L, X it holds that $L \in \mathrm{P}^X \Leftrightarrow L \leqslant_T^p X \Leftrightarrow L \in T \cdot \leqslant_m^p (X)$. We will call $T\cdot$ the Turing-operator. For example, it holds that $T \cdot \Sigma_k^p = \Delta_{k+1}^p$ and $T \cdot \Pi_k^p = \Delta_{k+1}^p$.

We did not cover by our definition the well-known BP-operator (we will see later that this is not possible). But with an extension of our concept we are able to write the definition of the BP-operator in the fashion of Definition 2.1. Let $Y (Y')$ be the language consisting of the words with at least three quarters (at most one quarter) of 1's among its letters.

The class $\mathrm{BP} \cdot \mathscr{C}$ is the set of languages L for which there is a language H in $\mathscr{C}$ and a polynomial-time computable function $l : \Sigma^* \to \mathbf{N}$ (numbers are represented in binary) such that

$$x \in L \;\Leftrightarrow\; H\langle x,1\rangle H\langle x,2\rangle \cdots H\langle x,l(x)\rangle \in Y,$$
$$x \notin L \;\Leftrightarrow\; H\langle x,1\rangle H\langle x,2\rangle \cdots H\langle x,l(x)\rangle \in Y'.$$

The special point about this operator is that it forbids some possible outcomes for the word $H\langle x,1\rangle H\langle x,2\rangle \cdots H\langle x,l(x)\rangle$, namely the outcomes with a share of 1's between $\frac{1}{4}$ and $\frac{3}{4}$. In other words, it promises not to have the forbidden possible outcomes. We will generalize this notion of a *promise dot operator* the following way. Let a *promise language* be a pair (A, B) of disjoint languages which are not both empty.

Definition 2.2 (*promise dot operator*). For a promise language (A, B) define the *promise dot operator* $(A, B)\cdot$, which maps classes to classes, the following way. For a class of languages $\mathscr{C}$ let $(A, B) \cdot \mathscr{C}$ be the class consisting of the languages L for which there is a language $H \in \mathscr{C}$ and a polynomial-time computable function $l : \Sigma^* \to \mathbf{N}$ (numbers are represented in binary) such that for all x it holds

$$x \in L \;\Leftrightarrow\; H\langle x,1\rangle H\langle x,2\rangle \cdots H\langle x,l(x)\rangle \in A,$$
$$x \notin L \;\Leftrightarrow\; H\langle x,1\rangle H\langle x,2\rangle \cdots H\langle x,l(x)\rangle \in B.$$

A complementary dot operator is a promise dot operator because $A\cdot$ equals $(A, \bar{A})\cdot$. By this fact, we will sometimes identify a language A with the promise language $(A, \bar{A})$, and we will call a promise dot operator often just a *dot operator*. We remark that according to the terminology of Papadimitriou's book [21] one would call complementary dot operators *syntactical* dot operators and promise dot operators *semantic* dot operators. We chose our terminology because the word 'promise' is quite intuitive.

Examples of promise dot operators

- The typical example of a promise dot operator is the BP-operator $\mathrm{BP}\cdot = (\{w \mid \#_1(w) \geqslant 3\#_0(w)\}, \{w \mid \#_0(w) \geqslant 3\#_1(w)\})\cdot$. Of course it holds $\mathrm{BP} \cdot \mathrm{P} = \mathrm{BPP}$.

- The RP-operator is characterized by $\mathrm{RP}\cdot = (\{w \mid \#_1(w) \geqslant \#_0(w)\}, 0^*)\cdot$. It holds $\mathrm{RP}\cdot\mathrm{P} = \mathrm{R}$.
- A U-operator could be defined as $\mathrm{U}\cdot = (\{w \mid \#_1(w) = 1\}, \{w \mid \#_1(w) = 0\})\cdot$, that way it holds $\mathrm{U}\cdot\mathrm{P} = \mathrm{UP}$.

Remark. We should mention that one could generalize Definitions 2.1 and 2.2 in order to get *function* operators like $\#\cdot$ (see [13]). Instead of taking in Definition 2.1 a *language* A (which can be considered to be a function from Σ^* to $\{0,1\}$) we let A be a function from Σ^* to some set, for example $\mathbf{N}$. For instance, if we let A be the function mapping a word w to the number of 1's in w, then we have that $A\cdot$ equals $\#\cdot$.

3. Properties of dot operators

In this section we state some properties. The first one directly follows from the definitions.

Proposition 3.1. *Dot operators are monotone, i.e. for all promise languages* (A,B) *it holds: if* $\mathscr{C} \subseteq \mathscr{C}'$ *then* $(A,B)\cdot\mathscr{C} \subseteq (A,B)\cdot\mathscr{C}'$.

By the next proposition, dot operators are a generalization of the leaf language concept. Here is the original definition of balanced leaf language classes: (see [6]). For any promise language (A,B), $\mathscr{C}(A,B)$ is the class of languages L for which there exist two polynomial-time computable functions $R : \Sigma^* \times \mathbf{N} \to \Sigma^*$ and $\ell : \Sigma^* \to \mathbf{N}$ such that, for every string x, $x \in L \Leftrightarrow R(x,1)R(x,2)\cdots R(x,\ell(x)) \in A$ and $x \notin L \Leftrightarrow R(x,1)R(x,2)\cdots R(x,\ell(x)) \in B$.

Proposition 3.2. *For any promise language* (A,B) *it holds that* $\mathscr{C}(A,B) = (A,B)\cdot\mathrm{P}$, *where* $\mathscr{C}(A,B)$ *is the polynomial-time leaf language class determined by* (A,B).

The following property of dot operators is natural but hard to prove (at least the authors did not find a shorter proof).

Theorem 3.3. *The composition of two (complementary) dot operators is still a (complementary) dot operator.*

Proof. We show that there exists a general and uniform way to define a dot operator that is equivalent to the composition of two given dot operators. Let $(A',B')\cdot$ and $(A,B)\cdot$ be any two dot operators. We define a dot operator $(\Gamma[A',A,B], \Gamma[B',A,B])\cdot$ such that

$$\forall X \quad (\Gamma[A',A,B], \Gamma[B',A,B])\cdot \leqslant_m^p(X) = (A',B')\cdot((A,B)\cdot \leqslant_m^p(X)).$$

The form of the languages $\Gamma[A',A,B]$ and $\Gamma[B',A,B]$ is quite natural

$$\begin{aligned}\Gamma[A',A,B] = \{z \mid\ & \sigma(z,1),\ldots,\sigma(z,\lambda(|z|))\\ & \in A \cup B \wedge A(\sigma(z,1))\cdots A(\sigma(z,\lambda(|z|))) \in A'\},\\ \Gamma[B',A,B] = \{z \mid\ & \sigma(z,1),\ldots,\sigma(z,\lambda(|z|))\\ & \in A \cup B \wedge A(\sigma(z,1))\cdots A(\sigma(z,\lambda(|z|))) \in B'\},\end{aligned}$$

where σ is a function from $\Sigma^* \times \mathbf{N}$ into Σ^* and λ is a function from $\mathbf{N}$ into $\mathbf{N}$. The not-easy part is the definition of σ and λ, since they must possess some special properties. Informally, $\sigma(z,1),\ldots,\sigma(z,\lambda(|z|))$ must be substrings of z and, depending on z, their lengths and positions have to be rather flexible. At the same time, each bit of $\sigma(z,i)$ must be computable in polynomial time given in input the length of z and having access to the bits of z. In order to ensure that, we introduce some notations. Let U be a universal deterministic Turing machine. That is, a machine such that for any deterministic Turing machine M there exists an index y such that $\forall x\ U(y,x) = M(x)$. Also, U can be defined so that for every machine M there exists an index y for M and a constant c such that, if the computation $M(x)$ halts within t steps then $U(y,x)$ halts within ct^2 steps. Function λ can be defined as follows, for every $n \geqslant 1$,

$$\lambda(n) = \begin{cases} U(y,\langle x\rangle) & \text{if } \exists y,x,u : n = \langle y,x,u\rangle \wedge U(y,\langle x\rangle)\\ & \quad \text{halts within } |n| \text{ steps } \wedge U(y,\langle x\rangle) \geqslant 1,\\ 1 & \text{otherwise.}\end{cases}$$

To define σ we need the following functions, for every $n \geqslant 1$ and every i with $1 \leqslant i \leqslant \lambda(n)$,

$$\alpha(n,i) = \begin{cases} U(y,\langle x,i,0\rangle) & \text{if } \exists y,x,u : n = \langle y,x,u\rangle \wedge U(y,\langle x,i,0\rangle)\\ & \quad \text{halts within } |n| \text{ steps } \wedge, 1 \leqslant U(y,\langle x,i,0\rangle) \leqslant n,\\ 1 & \text{otherwise,}\end{cases}$$

$$\beta(n,i) = \begin{cases} U(y,\langle x,i,1\rangle) & \text{if } \exists y,x,u : n = \langle y,x,u\rangle \wedge U(y,\langle x,i,1\rangle)\\ & \quad \text{halts within } |n| \text{ steps } \wedge, \alpha(n,i) + U(y,\langle x,i,1\rangle) - \leqslant n,\\ 1 & \text{otherwise.}\end{cases}$$

Now, for every z and every i with $1 \leqslant i \leqslant \lambda(|z|)$, define

$$\sigma(z,i) = z(\alpha(|z|,i))z(\alpha(|z|,i)+1)\cdots z(\alpha(|z|,i)+\beta(|z|,i)-1)$$

that is, $\sigma(z,i)$ is the substring of z of length $\beta(|z|,i)$ that begins at the $\alpha(|z|,i)$th symbol of z.

Firstly, we prove that, for every X,

$$(A',B')\cdot((A,B)\cdot \leqslant_m^p(X)) \subseteq (\Gamma[A',A,B],\Gamma[B',A,B])\cdot \leqslant_m^p(X).$$

Let L be a language in $(A',B') \cdot ((A,B) \cdot \leqslant_m^p (X))$. Then, there exists a language $H \in (A,B) \cdot \leqslant_m^p (X)$ and a polynomial-time function l such that, for every x,

$$x \in L \;\Leftrightarrow\; H\langle x,1\rangle H\langle x,2\rangle \cdots H\langle x,l(x)\rangle \in A',$$

$$x \notin L \;\Leftrightarrow\; H\langle x,1\rangle H\langle x,2\rangle \cdots H\langle x,l(x)\rangle \in B'.$$

Since $H \in (A,B) \cdot \leqslant_m^p (X)$, there exist two polynomial-time functions S and t such that, for every z,

$$z \in H \;\Leftrightarrow\; X(S(z,1))X(S(z,2)) \cdots X(S(z,t(z))) \in A,$$

$$z \notin H \;\Leftrightarrow\; X(S(z,1))X(S(z,2)) \cdots X(S(z,t(z))) \in B.$$

By combining the conditions for L with those for H, we obtain that

$$x \in L \;\Leftrightarrow\; A(w_{\langle x,1\rangle})A(w_{\langle x,2\rangle}) \cdots A(w_{\langle x,l(x)\rangle}) \in A',$$

$$x \notin L \;\Leftrightarrow\; A(w_{\langle x,1\rangle})A(w_{\langle x,2\rangle}) \cdots A(w_{\langle x,l(x)\rangle}) \in B',$$

where $w_{\langle x,i\rangle}$ denotes the string $X(S(\langle x,i\rangle,1))X(S(\langle x,i\rangle,2)) \cdots X(S(\langle x,i\rangle,t(\langle x,i\rangle)))$. For proving that $L \in (\Gamma[A',A,B],\Gamma[B',A,B]) \cdot \leqslant_m^p (X)$ it suffices to find two polynomial-time functions T and r such that, for every x, the string $z_x := X(T(x,1))X(T(x,2)) \cdots X(T(x,r(x)))$ satisfies $\lambda(|z_x|) = l(x)$ and, for every i with $1 \leqslant i \leqslant l(x)$, $\sigma(z_x,i) = w_{\langle x,i\rangle}$. To this end, we define T and r so that z_x has the form $z_x = z_x^1 z_x^2 \cdots z_x^{l(x)} u$ with $|z_x^1| = |z_x^2| = \cdots |z_x^{l(x)}| = 2^{q(|x|)}$ and $w_{\langle x,i\rangle}$ is a prefix of z_x^i, where q is a polynomial such that $|w_{\langle x,i\rangle}| \leqslant 2^{q(|x|)}$, for every x and every $i \leqslant l(x)$. We can now define T, for every x and every $j \geqslant 1$,

$$T(x,j) = \begin{cases} S(\langle x,i\rangle, j-(i-1)\cdot 2^{q(|x|)}) & \text{if } \exists i\colon 1 \leqslant i \leqslant l(x) \text{ and } (i-1)\cdot 2^{q(|x|)} \\ & \qquad < j \leqslant i \cdot 2^{q(|x|)}, \\ 0 & \text{othewise.} \end{cases}$$

The definition of the function r is more delicate. We have to ensure that $r(x)$, that is $|z_x|$, contains sufficient information so that $\alpha(|z_x|,i) = (i-1)\cdot 2^{q(|x|)}+1$ and $\beta(|z_x|,i) = t(\langle x,i\rangle)$. It is easy to see that there exists an index $\bar{y}$ and a polynomial p such that, for every x and every i with $i \leqslant l(x)$, it holds $U(\bar{y},\langle x\rangle) = l(x), U(\bar{y},\langle x,i,0\rangle) = (i-1)\cdot 2^{q(|x|)}+1, U(\bar{y},\langle x,i,1\rangle) = t(\langle x,i\rangle)$, and all the computations $U(\bar{y},\langle x\rangle), U(\bar{y},\langle x,i,0\rangle), U(\bar{y},\langle x,i,1\rangle)$ halt within $p(|x|)$ steps. Thus, we can define $r(x) = \langle \bar{y}, x, l(x) \cdot 2^{q(|x|)+p(|x|)}\rangle$, for every x. It is easy to verify that T and r witness that $L \in (\Gamma[A',A,B], \Gamma[B',A,B]) \cdot \leqslant_m^p (X)$.

Conversely, let L be a language in $(\Gamma[A',A,B], \Gamma[B',A,B]) \cdot \leqslant_m^p (X)$. Then, there exist two polynomial-time functions R and l such that

$$x \in L \;\Leftrightarrow\; X(R(x,1))X(R(x,2)) \cdots X(R(x,l(x))) \in \Gamma[A',A,B],$$

$$x \notin L \;\Leftrightarrow\; X(R(x,1))X(R(x,2)) \cdots X(R(x,l(x))) \in \Gamma[B',A,B].$$

For every x, let z_x denote the string $X(R(x,1))X(R(x,2))\cdots X(R(x,l(x)))$. Define the following language

$$H = \{\langle x,j\rangle \mid (j \leqslant \lambda(|z_x|) \wedge \sigma(z_x,j) \in A) \vee (j > \lambda(|z_x|) \wedge \sigma(z_x,1) \in A)\}.$$

Now, it is easy to show that $H \in (A,B)\cdot \leqslant_m^p(X)$. In fact, define for all x and j,

$$t(\langle x,j\rangle) = \begin{cases} \beta(|z_x|,j) & \text{if } j \leqslant \lambda(|z_x|), \\ \beta(|z_x|,1) & \text{otherwise.} \end{cases}$$

$$s(\langle x,j\rangle,i) = \begin{cases} R(x,\alpha(|z_x|,j)+i-1) & \text{if } j \leqslant \lambda(|z_x|), \\ R(x,\alpha(|z_x|,1)+i-1) & \text{otherwise.} \end{cases}$$

Since $|z_x| = l(x)$, it is clear that $\lambda(|z_x|)$, $\beta(|z_x|,j)$ and $\alpha(|z_x|,j)$ can be computed in polynomial time in the length of x. It follows that the functions t and S are polynomial-time computable. It is easy to verify that, for all x and j,

$$X(S(\langle x,j\rangle,1))X(S(\langle x,j\rangle,2))\cdots X(S(\langle x,j\rangle,t(\langle x,j\rangle))) = \begin{cases} \sigma(z_x,j) & \text{if } j \leqslant \lambda(|z_x|), \\ \sigma(z_x,1) & \text{otherwise.} \end{cases}$$

Also, since $z_x \in \Gamma[A',A,B] \cup \Gamma[B',A,B]$, it holds that $\sigma(z_x,j) \in A \cup B$ for every j with $j \leqslant \lambda(|z_x|)$. Thus, S and t witness that $H \in (A,B)\cdot \leqslant_m^p(X)$. For proving that $L \in (A',B')\cdot \leqslant_m^p(H)$ it suffices to observe that, for every x, it holds that

$$H\langle x,1\rangle H\langle x,2\rangle \cdots H\langle x,\lambda(l(x))\rangle = A(\sigma(z_x,1))A(\sigma(z_x,2))\cdots A(\sigma(z_x,\lambda(|z_x|))).$$

Finally, we observe that if (A',B') and (A,B) are complementary then also $(\Gamma[A',A,B],\Gamma[B',A,B])$ is complementary. This shows that the composition of two complementary dot operators is a complementary dot operator. $\square$

Remark. Because dot operators are mappings, their composition is associative. The following example shows that the composition is not commutative. Let A be an oracle such that $\Sigma_2^p(A) \neq \Pi_2^p(A)$ (see [4]). It holds that

$$\exists\cdot\forall\cdot\leqslant_m^p(A) = \Sigma_2^p(A) \neq \Pi_2^p(A) = \forall\cdot\exists\cdot\leqslant_m^p(A).$$

A *class* is a set of languages. We say that a class $\mathscr{C}$ is closed *downward* if it is closed downward w.r.t. polynomial-time many-one reducibility, i.e. from $A \leqslant_m^p B$ and $B \in \mathscr{C}$ it follows $A \in \mathscr{C}$. The *join* $A \oplus B$ of two languages A,B is the set $0A \cup 1B$. A set $\mathscr{C}$ of languages is *closed under join* if from $A, B \in \mathscr{C}$ it follows $A \oplus B \in \mathscr{C}$. An $\leqslant_m^p$-*ideal* is a set I of languages which is closed downward and under join. A language B is $\leqslant_m^p$-*complete* for a class $\mathscr{C}$ if $B \in \mathscr{C}$ and for each language $A \in \mathscr{C}$ it holds $A \leqslant_m^p B$. A *principal* $\leqslant_m^p$-*ideal* is a set of languages which has a $\leqslant_m^p$-complete language and is closed downward, note that closedness under join follows from this. It was proven in [6] that the promise leaf language classes $\mathscr{C}(A,B)$ are exactly the $\leqslant_m^p$-ideals, and the complementary leaf language classes $\mathscr{C}(A)$ are exactly the principal $\leqslant_m^p$-ideals. The following result can be shown with the methods of [6, Fact 2.8 and Th. 4.2] and [5, Prop. 4 and Th. 6].

Theorem 3.4. (a) *Let* (A,B) *be a promise language and let* $\mathscr{C}$ *be an* $\leqslant_m^p$*-ideal. Then* $(A,B)\cdot\mathscr{C}$ *is an* $\leqslant_m^p$*-ideal.*

(b) *Let* A *be a language and let* $\mathscr{C}$ *be a principal* $\leqslant_m^p$*-ideal. Then* $A\cdot\mathscr{C}$ *is a principal* $\leqslant_m^p$*-ideal.*

Note that Theorem 3.4(b) implies that $A\cdot\mathscr{C}$ has a $\leqslant_m^p$-complete language. Because the Polynomial Hierarchy and the Counting Hierarchy are defined, starting with the principal $\leqslant_m^p$-ideal P, by iterative use of the complementary dot operators $\exists\cdot, \forall\cdot, \mathrm{C}\cdot$, we have the following well-known corollary. (Of course, a similar result holds for all classes defined by iterative use of a given set of complementary dot operators, starting with P.)

Corollary 3.5 (Stockmeyer [27], Wrathall [36], Wagner [35]). *The classes of the Polynomial Hierarchy and the Counting Hierarchy have* $\leqslant_m^p$*-complete sets.*

Remark. Let the $\leqslant_m^p$*-degree* of a language A be the set of languages polynomial-time equivalent to A. $\leqslant_m^p$ transfers consistently to the $\leqslant_m^p$-degrees. It is easy to see that principals $\leqslant_m^p$-ideals and $\leqslant_m^p$-degrees correspond to each other by the mapping which maps a principal $\leqslant_m^p$-ideal to its set of $\leqslant_m^p$-complete languages. This mapping is an isomorphism of the partial order $\subseteq$ on the principal $\leqslant_m^p$-ideals and the partial order $\leqslant_m^p$ on the $\leqslant_m^p$-degrees. Therefore, realizing that degrees are a much better-known concept than principal $\leqslant_m^p$-ideals, we may by the above Theorem 3.4(b) also consider a complementary dot operator to be a function which maps $\leqslant_m^p$-degrees to $\leqslant_m^p$-degrees.

4. Comparison of dot operators

The natural way of comparing (dot) operators is the following.

Definition 4.1 (*partial order on operators*). Let O_1 and O_2 be any two operators mapping classes to classes. Define the partial order $\leqslant^{\mathrm{op}}$ on operators by

$$O_1 \leqslant^{\mathrm{op}} O_2 \;\Leftrightarrow\; \text{for all classes } \mathscr{C} \text{ it holds } O_1\mathscr{C} \subseteq O_2\mathscr{C}.$$

Recall that we assume all the classes are closed downward and that they are not included in $\{\emptyset, \Sigma^*\}$. Without the latter assumption, Lemma 4.4 would not be true.

We will also simply write $(A,B) \leqslant^{\mathrm{op}} (A',B')$ instead of $(A,B)\cdot \leqslant^{\mathrm{op}} (A',B')\cdot$. We will see that there is a close connection of this order on dot operators to the following definition.

Definition 4.2 (*monotone projections, Skyum and Valiant* [26]). A function $f:\Sigma^* \to \Sigma^*$ is a *monotone projection* if there exist two functions $l:\mathbf{N}\to\mathbf{N}$ and $\sigma:\mathbf{N}\times\mathbf{N}\to\mathbf{N}$ such that for every word $x_1\cdots x_m$ it holds

$$f(x_1\cdots x_m) = [m,\sigma(m,1)][m,\sigma(m,2)]\cdots[m,\sigma(m,l(m))],$$

where $[\,,]$ is the function $\mathbf{N}\times\mathbf{N}\to\{0,1\}$ defined by

$$[m,j] := \begin{cases} x_j & \text{if } 1\leqslant j\leqslant m,\\ 0 & \text{if } j=0,\\ 1 & \text{if } j>m. \end{cases}$$

Call a monotone projection *polylog-time uniform* if l and σ are polynomial-time computable (the log-shift is caused by the binary representation of numbers).

Examples of polylog-time uniform monotone projections are functions like $f(x_1x_2\cdots x_m)=x_m\cdots x_2x_1$, or $f(x_1x_2\cdots x_m)=x_1 0 x_1$.

Monotone projections induce the following reducibility. Note that by our convention $A=(A,\bar{A})$ we will have defined the reducibility also on usual languages.

Definition 4.3 (*monotone projection reducibility*). A promise language (A,B) is *polylog-time monotone projection reducible* to a promise language (A',B'), written $(A,B)\leqslant_{mp}^{plt}(A',B')$, if there is a polylog-time uniform monotone projection f such that, for every z,

$$z\in A \Rightarrow f(z)\in A',$$

$$z\in B \Rightarrow f(z)\in B'.$$

A similar kind of reducibility is studied in [32] (see also [15]). The next Lemma 4.4 represents the easy direction of Theorem 4.6 below.

Lemma 4.4. *Let (A_1,B_1) and (A_2,B_2) be two promise languages. Then, it holds that*

$$(A_1,B_1)\leqslant_{mp}^{plt}(A_2,B_2) \Rightarrow (A_1,B_1)\leqslant^{\mathrm{op}}(A_2,B_2).$$

Proof. Since $(A_1,B_1)\leqslant_{mp}^{plt}(A_2,B_2)$, there is a projection f such that, for every z, $z\in A_1 \Leftrightarrow f(z)\in A_2$ and $z\in B_1\Leftrightarrow f(z)\in B_2$. Let σ and l be two functions that, according to Definition 4.2, witness that f is a polylog-time uniform monotone projection. Let L be a language in $(A_1,B_1)\cdot\leqslant_m^p(X)$. Then, there exist two polynomial-time functions R and t such that, for every x, it holds that $x\in L\Leftrightarrow X(R(x,1))X(R(x,2))\cdots X(R(x,t(x)))\in A_1$ and $x\notin L \Leftrightarrow X(R(x,1))X(R(x,2))\cdots X(R(x,t(x)))\in B_1$. It is easy to define polynomial-time functions T and s such that, for every x,

$$f(X(R(x,1))X(R(x,2))\cdots X(R(x,t(x))))$$
$$= X(T(x,1))X(T(x,2))\cdots X(T(x,s(x))).$$

In fact, for every x and every $i \geqslant 1$,

$$s(x) = l(t(x)) \quad \text{and} \quad T(x,i) = \begin{cases} R(x,\sigma(t(x),i)) & \text{if } 1 \leqslant \sigma(t(x),i) \leqslant t(x), \\ u_1 & \text{if } \sigma(t(x),i) \geqslant t(x)+1, \\ u_0 & \text{if } \sigma(t(x),i) = 0, \end{cases}$$

where u_0 and u_1 are two fixed strings such that $X(u_0)=0$ and $X(u_1)=1$ (at this place we need the requirement that classes are not included in $\{\emptyset, \Sigma^*\}$). It is immediate to verify that T and s witness that $L \in (A_2,B_2)\cdot \leqslant_m^p (X)$. $\square$

The following result is a corollary of the above Lemma 4.4.

Corollary 4.5. *Let A be a language. If membership in A only depends on length then $A\cdot$ is a constant function. Otherwise, it holds* $\mathrm{id}\cdot \leqslant^{\mathrm{op}} A\cdot$ *or* $\mathrm{co}\cdot \leqslant^{\mathrm{op}} A\cdot$ (*or both*).

Proof. If membership in A only depends on length then it immediately follows from the definition that $A\cdot$ is a constant function.

If this is not the case then there are two strings w_1 and w_2 of the same length n such that $w_1 \in A$ and $w_2 \notin A$ and there is only one index i such that $w_1(i) \neq w_2(i)$. Consider the following projection $f(z) = w_1(1)\cdots w_1(i-1)z(1)w_1(i+1)\cdots w_1(n)$. If $w_1(i)=1$ then f shows that $1\Sigma^* \leqslant_{mp}^{plt} A$. Otherwise, f shows that $0\Sigma^* \leqslant_{mp}^{plt} A$. From Lemma 4.4 it follows that either $\mathrm{id}\cdot \leqslant^{\mathrm{op}} A\cdot$ or $\mathrm{co}\cdot \leqslant^{\mathrm{op}} A\cdot$ (or both). $\square$

The following Theorem 4.6 and its Corollary 4.7 may be considered to be the main result of this paper.

Theorem 4.6. *Let A be a language and let (C,D) be a promise language. Then, it holds that*

$$A \leqslant_{mp}^{plt} (C,D) \;\Leftrightarrow\; A \leqslant^{\mathrm{op}} (C,D).$$

Proof. One direction has been proved in Lemma 4.4. It remains to show that $A \nleqslant_{mp}^{plt} (C,D)$ implies the existence of a language H such that $A\cdot \leqslant_m^p (H) \nsubseteq (C,D)\cdot \leqslant_m^p (H)$. Assume that $A \nleqslant_{mp}^{plt} (C,D)$. It is easy to see that only the following two cases can happen:

(1) There is an $\bar{m} \geqslant 1$ such that, for every polylog-time uniform monotone projection f, there is a string z of length $\bar{m}$ for which either $z \in A$ and $f(z) \notin C$ or $z \notin A$ and $f(z) \notin D$.

(2) For every polylog-time uniform monotone projection f, there are infinitely many z such that either $z \in A$ and $f(z) \notin C$ or $z \notin A$ and $f(z) \notin D$.[1]

Consider the first case. If $A = \Sigma^*$ then it must be the case that $C = \emptyset$. Thus, it is easy to verify that $\Sigma^* \cdot \leqslant_m^p (\Sigma^*) \nsubseteq (\emptyset, D) \cdot \leqslant_m^p (\Sigma^*)$. Assume that $A \neq \Sigma^*$. For any language X, define $L_1(X) = \{0^n \mid 2^n \geqslant \bar{m} \wedge X(0^n)X(0^{n-1}1)\ldots X(\bar{m}_n) \in A\}$ where $\bar{m}_n$ is the $\bar{m}$th string of length n, in the lexicographic order. Clearly, it holds that $L_1(X) \in A \cdot \leqslant_m^p (X)$ whenever $X \neq \emptyset, \Sigma^*$. Let $\{(R_i, l_i)\}_{i \geqslant 1}$ be an enumeration of all the pairs of polynomial-time computable functions such that $R_i : \Sigma^* \times \mathbf{N} \to \Sigma^*$ and $l_i : \Sigma^* \to \mathbf{N}$. For every $i \geqslant 1$, let p_i be a polynomial such that for every n, $p_i(n)$ bounds the running time of $R_i(z,k)$ for every z of length at most n and every $k \leqslant l_i(z)$. Also, let $p_0(n) = n$ for all n. We need the following notation: for every n and for every z, let $B_{n,z}$ be the language

$$B_{n,z} = \{x \mid |x| = n \text{ and } z(j) = 1, \text{ being } x \text{ the } j\text{th string of length } n\}.$$

We construct the language H by stages.

Begin construction

Stage 0: $H_0 := \emptyset$ and $n_0 := \bar{m}$.

Stage $k \geqslant 1$: Set $n_k := p_{k-1}(n_{k-1}) + 1$. For any z, let $\mathscr{H}_z = H_{k-1} \cup B_{n_k,z}$ and let

$$g_z^k = \mathscr{H}_z(R_k(0^{n_k},1))\mathscr{H}_z(R_k(0^{n_k},2)) \cdots \mathscr{H}_z(R_k(0^{n_k}, l_k(0^{n_k}))).$$

Let u be a string of length $\bar{m}$ such that $(u \in A \wedge g_u^k \notin C) \vee (u \notin A \wedge g_u^k \notin D)$. Set $H_k := H_{k-1} \cup B_{n_k,u}$.

End construction

Let $H = \bigcup_k H_k$. If at every stage, there always exists a string u satisfying the requirements, then it is easy to see that $L_1(H) \notin (C,D) \cdot \leqslant_m^p (H)$. We have only to prove that such a string u always exists. Consider stage k and suppose by the way of contradiction that, for every z of length $\bar{m}$, it holds that

$$z \in A \Rightarrow g_z^k \in C \quad \text{and} \quad z \notin A \Rightarrow g_z^k \in D.$$

We show the existence of a polylog-time uniform monotone projection f such that $f(z) = g_z^k$ for all the strings z of length $\bar{m}$. Such a projection f can be defined by the functions σ and l below

$$\forall n \quad l(n) = \begin{cases} l_k(0^{n_k}) & \text{if } n = \bar{m}, \\ 1 & \text{otherwise.} \end{cases}$$

[1] In fact, if there is a polylog-time uniform monotone projection g such that for all z but finitely many it holds that $z \in A \Rightarrow g(z) \in C$ and $z \notin A \Rightarrow g(z) \in D$, then there exists an $\bar{m} \geqslant 1$ for which every polylog-time uniform monotone projection fails to be a reduction from A to (C,D) on strings of length $\bar{m}$. For otherwise, being $n_1, \ldots, n_k$ the finite set of lengths on which g fails to be a reduction from A to (C,D), there would exist polylog-time uniform monotone projections $f_1, \ldots, f_k$ such that f_i is, on strings of length n_i, a reduction from A to (C,D). Thus, combining g with $f_1, \ldots, f_k$, we could obtain a polylog-time uniform monotone projection witnessing $A \leqslant_{mp}^{plt} (C,D)$, contradicting the assumption.

$$\forall i \leqslant t(n) \quad \sigma(n,i) = \begin{cases} j & \text{if } n = \bar{m} \text{ and } R_k(0^{n_k}, i) \text{ is the } j\text{th string of length} \\ & \quad n_k \text{ and } j \leqslant \bar{m}, \\ \bar{m}+1 & \text{if } n = \bar{m} \text{ and } |R_k(0^{n_k}, i)| < n_k \text{ and} \\ & \quad H_{k-1}(R_k(0^{n_k}, i)) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

It follows that f is a reduction, on strings of length $\bar{m}$, from A to (C,D), contradicting the assumption.

Consider now the second case. We assume that every polylog-time uniform monotone projection fails to be a reduction from A to (C,D) on infinitely many lengths. For any language X, define $L_2(X) = \{z \mid X(0^{|z|+1})X(0^{|z|}1)\cdots X(r_z) \in A\}$ where r_z denotes the zth string of length $|z|+1$. It is clear that $L_2(X) \in A \cdot \leqslant_m^p (X)$. We construct by stages a language E such that $L_2(E) \notin (C,D) \cdot \leqslant_m^p (E)$. For every string z we denote by $\|z\|$ the length of the binary representation of $|z|$.

Begin construction

Stage 0: $E_0 := \emptyset$ and $n_0 := 1$.

Stage $k \geqslant 1$: For any y let $\mathscr{E}_y = E_{k-1} \cup B_{\|y\|+1,y}$ and let

$$h_y^k = \mathscr{E}_y(R_k(|y|,1))\mathscr{E}_y(R_k(|y|,2))\cdots\mathscr{E}_y(R_k(|y|,l_k(|y|))).$$

Let v be the least string such that $\|v\| > p_{k-1}(n_{k-1})$ and $(v \in A \wedge h_v^k \notin C) \vee (v \notin A \wedge h_v^k \notin D)$.

Set $E_k := E_{k-1} \cup B_{\|v\|,v}$ and $n_k := \|v\|$.

End construction

Let $E = \bigcup_k E_k$. If at every stage, there always exists a string v satisfying the requirements, then it is easy to see that $L_2(E) \notin (C,D) \cdot \leqslant_m^p (E)$. We have only to prove that such a string v always exists. Consider stage k and suppose by the way of contradiction that, for every y with $\|y\| > p_{k-1}(n_{k-1})$, it holds that

$$y \in A \Rightarrow h_y^k \in C \quad \text{and} \quad y \notin A \Rightarrow h_y^k \in D.$$

We show the existence of a polylog-time uniform monotone projection f such that $f(y) = h_y^k$ for every y with $\|y\| > p_{k-1}(n_{k-1})$. Such a projection can be defined by the functions σ and l below

$$\forall n \quad l(n) = l_k(n)$$

$$\forall i \leqslant t(n) \quad \sigma(n,i) = \begin{cases} j & \text{if } R_k(n,i) \text{ is the } j\text{th string of length } |n|+1 \\ & \quad \text{and } j \leqslant n, \\ n+1 & \text{if } |R_k(n,i)| < |n|+1 \text{ and } E_{k-1}(R_k(n,i)) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

It follows that, for all the strings y except at most finitely many, $y \in A \Rightarrow f(y) \in C$ and $y \notin A \Rightarrow f(y) \in D$. Hence, f is a polylog-time uniform monotone projection that reduces A to (C,D) on all the lengths except at most finitely many, contradicting the assumption. □

For the complementary case we have an isomorphism between $\leqslant_{mp}^{plt}$ and $\leqslant^{\mathrm{op}}$.

Corollary 4.7. *Let A,B be two languages. Then, it holds that*

$$A \leqslant_{mp}^{plt} B \quad \Leftrightarrow \quad A \leqslant^{\mathrm{op}} B.$$

Obviously we have the following result stated in the abstract.

Corollary 4.8. *Let A,B be two languages. Then, it holds that*

$$A \equiv_{mp}^{plt} B \quad \Leftrightarrow \quad A \equiv^{\mathrm{op}} B \quad \Leftrightarrow \quad A\cdot = B\cdot.$$

The next result shows that Theorem 4.6 cannot be extended as to hold for promise dot operators in the fashion of Corollary 4.8.

Proposition 4.9. *There exist two promise languages (V,W) and (E,F) such that*

$$(V,W) \not\leqslant_{mp}^{plt} (E,F) \quad \textit{and} \quad (V,W) \leqslant^{\mathrm{op}} (E,F).$$

Proof. Let $A_{\mathrm{NP}} = \Sigma^* - \{0\}^*$. Let $\mathscr{I}$ be an immune set (i.e. $\mathscr{I}$ is an infinite set with no infinite recursively enumerable subset). Define a promise language (V,W) and a language E as follows

$$V = \{z \mid |z| \in \mathscr{I} \wedge z \in A_{\mathrm{NP}}\}, \; W = \{z \mid |z| \in \mathscr{I} \wedge z \notin A_{\mathrm{NP}}\},$$

$$E = \{z \mid z = xy \wedge |x| = \lceil \log |z| \rceil \wedge x \in A_{\mathrm{NP}}\}.$$

Firstly, we show that $(V,W) \not\leqslant_{mp}^{plt} E$. Suppose the contrary. Let f be a polylog-time uniform monotone projection such that, for every z,

$$z \in V \Rightarrow F(z) \in E \quad \text{and} \quad z \in W \Rightarrow f(z) \notin E.$$

Let σ and l be two functions witnessing that f is a polylog-time uniform monotone projection. This implies that, for every $z, |f(z)| = l(|z|)$. Since l is polynomial-time computable, there exists a polynomial p such that $l(x) \leqslant 2^{p(|x|)}$ for every x. It follows that, for every $z, |f(z)| \leqslant 2^{p(||z||)}$. Since $\mathscr{I}$ is an infinite set, there is $\bar{n}$ such that $\bar{n} \in \mathscr{I}$ and $p(|\bar{n}|) < \bar{n} - 1$. Consider the function f on inputs of length $\bar{n}$. It holds that, for every z with $|z| = \bar{n}, |f(z)| = l(\bar{n}) \leqslant 2^{p(|\bar{n}|)} < 2^{\bar{n}-1}$. Thus, letting $\bar{m} = \lceil \log(l(\bar{n})) \rceil$, it holds that $\bar{m} \leqslant \bar{n} - 1$. For any z of length $\bar{n}$, let σ_z be the prefix of length $\bar{m}$ of $f(z)$. Since f is a projection from (V,W) to E, it must be the case that $\sigma_z \in A_{\mathrm{NP}} \Leftrightarrow z \in A_{\mathrm{NP}}$. Since $\bar{m} < \bar{n}$, there exists $\bar{k}$ with $1 \leqslant \bar{k} \leqslant \bar{n}$ such that $\sigma(\bar{n}, i) \neq \bar{k}$ for all $i = 1, \ldots, \bar{m}$. Also, if $z = 0^{\bar{n}}$ then $\sigma_z = 0^{\bar{m}}$. Thus, $\sigma(\bar{n}, i) \leqslant \bar{n}$ for all $i = 1, \ldots, \bar{m}$. It follows that if $z = 0^{\bar{k}-1}10^{\bar{n}-\bar{k}}$ then $\sigma_z = 0^{\bar{m}}$, contradicting the assumption about f.

Now, we prove that $(V,W) \leqslant^{\mathrm{op}} E$. Let $X \neq \Sigma^*$ and let L be a language in $(V,W) \cdot \leqslant_m^p (X)$. There exist two polynomial-time functions R and l such that, for every z,

$$z \in L \;\Leftrightarrow\; X(R(z,1)) \cdots X(R(z,l(z))) \in V \quad \text{and}$$

$$z \notin L \;\Leftrightarrow\; X(R(z,1)) \cdots X(R(z,l(z))) \in W.$$

This implies that, for every z, $X(R,(z,l)) \cdots X(R(z,l(z))) \in V \cup W$. Thus, letting $\mathscr{L} = \{l(z) \mid z \in \Sigma^*\}$ it holds that $\mathscr{L} \subseteq \mathscr{I}$. Since $\mathscr{I}$ is an immune set and $\mathscr{L}$ is recursively enumerable, it must be the case that $\mathscr{L}$ is a finite set. Let m be such that $l(z) \leqslant m$, for all z. Consider the functions S and t so defined

$$\forall z, i \quad t(z) = 2^m \quad S(z,i) = \begin{cases} R(z,i) & \text{if } 1 \leqslant i \leqslant l(z), \\ u_0 & \text{otherwise,} \end{cases}$$

where u_0 is a fixed string such that $X(u_0) = 0$. It is very easy to verify that S and t witness that $L \in E \cdot \leqslant_m^p (X)$. $\square$

The reader may ask what happens if in Definitions 4.2 and 4.3 one takes projections instead of monotone projections. We get a result similar to Corollary 4.7 which involves polynomial-time 1-tt reducibility. By going through the analogous proof of Theorem 4.6 we get the following theorem.

Theorem 4.10. *For all languages A, B it holds: A is polylog-time uniform projection reducible to B if and only if for all languages X*

$$A \cdot \leqslant_{1-tt}^p (X) \subseteq B \cdot \leqslant_{1-tt}^p (X).$$

Compare Theorem 4.10 with the following theorem of [6].

Theorem 4.11 (Bovet et al. [6]). *For all languages A, B it holds: A is polylog-time reducible to B if and only if for all languages X*

$$A \cdot \leqslant_T^p (X) \subseteq B \cdot \leqslant_T^p (X).$$

The $\leqslant_{mp}^{plt}$-degree structure is an upper semi-lattice: For two degrees represented by languages A and B the least upper bound is given by the language $\{x0^{|x|} \mid x \in A\} \cup \{x0^{|x|+1} \mid x \in B\}$. By Corollary 4.7 this can be transferred to $\leqslant^{\mathrm{op}}$.

Corollary 4.12. *The partial order $\leqslant^{\mathrm{op}}$ on the complementary dot operators is an upper semi-lattice.*

5. Representation of reducibilities

With dot operators it is not only possible to represent classes (like leaf languages do) but reducibilities. We will see that in fact most of the known polynomial-time reducibilities can be represented by dot operators.

Definition 5.1. An operator $(A,B)\cdot$ *represents a reducibility* $\leqslant^r$ if for all languages L and $X \neq \emptyset, \Sigma^*$ it holds $L \leqslant^r X \Leftrightarrow L \in (A,B)\cdot \leqslant_m^p (X)$.

Observe that if $(A,B)\cdot$ represents a reducibility $\leqslant^r$ then $\leqslant^r$ is transitive if and only if operator $(A,B)\cdot$ is idempotent.

We say that a promise language (A,B) represents $\leqslant^r$ if the corresponding operator $(A,B)\cdot$ represents it.

The following theorem covers most of the polynomial-time reducibilities known to the authors. An example for a polynomial-time reducibility not representable is the polynomial-time 1-1-reducibility (because it is stronger than many-one).

Theorem 5.2. (a) *The following polynomial-time reducibilities can be represented by complementary dot operators*:

(i) $\leqslant_m^p$ (*many-one reducibility*),
(ii) $\leqslant_{k-tt}^p$ (*k-truth-table* [18]),
(iii) $\leqslant_{k-c}^p$ (*k-conjunctive* [18]),
(iv) $\leqslant_{k-d}^p$ (*k-disjunctive* [18]),
(v) $\leqslant_{k-ptt}^p$ (*k-positive truth-table* [18]),
(vi) $\leqslant_{k-T}^p$ (*k-Turing*),
(vii) $\leqslant_{tt}^p$ (*truth-table* [18]),
(viii) $\leqslant_c^p$ (*conjunctive* [18]),
(ix) $\leqslant_d^p$ (*disjunctive* [18]),
(x) $\leqslant_T^p$ (*Turing*),
(xi) $\leqslant_m^{\mathrm{NP}}$ (*nondeterministic many-one* [18]),
(xii) $\leqslant_c^{\mathrm{NP}}$ (*nondeterministic conjunctive* [18]),
(xiii) $\leqslant_T^{\mathrm{NP}}$ (*nondeterministic Turing* [18]),
(xiv) $\leqslant_T^{\mathrm{PS}}$ (*PSPACE Turing [24]*),

(b) *The following reducibilities can be represented by promise dot operators but not by complementary dot operators*:

(i) $\leqslant_{btt}^p$ (*bounded-truth-table* [18]),
(ii) $\leqslant_T^{\mathrm{SN}}$ (*strong nondeterministic Turing* [19]),
(iii) $\leqslant_m^{\mathrm{RP}}$ (*random many-one* [1,31]),
(iv) $\leqslant_m^{\mathrm{BPP}}$ (*BPP many-one* [1,31]),

(c) *The following reducibilities can be represented by promise dot operators* (*but the authors do not know whether they can be represented by complementary dot operators*).

(i) $\leqslant_{ptt}^p$ (*positive truth-table* [18]),
(ii) $\leqslant_{rptt}^p$ (*locally right positive truth-table* [11]),
(iii) $\leqslant_{lptt}^p$ (*locally left positive truth-table* [11]),
(iv) $\leqslant_{pos}^p$ (*positive Tuning* [23]),
(v) $\leqslant_{rpos}^p$ (*locally right positive Turing* [11]),

(vi) $\leqslant^{p}_{lpos}$ (*locally left positive Turing* [11]),
(vii) $\leqslant^{\mathrm{SN}}_{m}$ (*strong nondeterministic many-one* [1, 19]),
(viii) $\leqslant^{\mathrm{SN}}_{c}$ (*strong nondeterministic conjunctive* [19]),
(ix) $\leqslant^{\mathrm{SN}}_{d}$ (*strong nondeterministic disjunctive* [19]),
(x) $\leqslant^{\mathrm{SN}}_{ptt}$ (*strong nondeterministic positive truth-table* [19]),
(xi) $\leqslant^{\mathrm{SN}}_{tt}$ (*strong nondeterministic truth-table* [19]),
(xii) $\leqslant^{\mathrm{RS}}_{T}$ (*robustly strong nondeterministic Turing* [9]).

Proof. (a)(x): First, we show how to represent the Turing-reducibility. This is a concrete example for illustrating the general technique that we use to provide reducibility representations. In order to represent $\leqslant^{p}_{T}$, we first need some notations. Consider a complete ordered binary tree t whose nodes are labeled by symbols in $\{0,1\}$. For such a tree t there is a special path that we can call the *correct path* of t, defined as follows: starting from the root, at any node v we go to its left son if the label of v is 0 and we go to its right son if the label is 1. The *value* of t is the value (i.e. the label) of the leaf of the correct path of t. Any complete ordered binary tree t whose nodes are labeled by symbols in $\{0,1\}$ can be encoded by a string z in $\{0,1\}^*$ which is obtained by concatenating the levels of t starting from the top level. Notice that if t has depth ℓ then $|z| = 2^{\ell} - 1$. In this way, each node u of t can be identified by an index i such that $z(i)$ is the label of u. Thus, for instance, 1 is the index of the root of t and 2 and 3 are, respectively, the indices of the left son and the right son of the root. A dot operator $T\cdot$ that represents the $\leqslant^{p}_{T}$-reducibility can be defined as follows:

$$T = \{z \mid (\exists \ell)|z| = 2^{\ell} - 1 \wedge \text{ the value of the tree encoded by } z \text{ is } 1\}.$$

We prove that $T\cdot$ represents $\leqslant^{p}_{T}$. Let $L \leqslant^{p}_{T} X (X \neq \emptyset, \Sigma^*)$ by a polynomial-time oracle Turing machine M. Without loss of generality, we can assume that, for every x, the tree $\mathscr{T}(x)$ of all the possible queries that can be made by M on input x is a complete tree. Since $\mathscr{T}(x)$ is a complete ordered binary tree we can think that each node of $\mathscr{T}(x)$ is identified by an index as explained above. Let $d(x)$ denote the depth of $\mathscr{T}(x)$ and, for every $i = 1, \dots, 2^{d(x)} - 1$, let $q(x,i)$ be the query that corresponds to the node of $\mathscr{T}(x)$ identified by i. We extend the tree $\mathscr{T}(x)$ by adding to it one more level. The extended tree $\mathscr{T}'(x)$ is such that if v is a leaf of $\mathscr{T}(x)$ then the left (respectively, the right) son of v in $\mathscr{T}'(x)$ is labeled by the output of the computation of M on input x which follows the path of queries determined by v and whose last query (i.e. the query that corresponds to v) receives answer 0 (respectively, 1). For every x and $j = 2^{d(x)}, \dots, 2^{d(x)+1} - 1$, let $m(x,j)$ be the label of the leaf of $\mathscr{T}'(x)$ identified by j. Now, we can define the following two functions

$$\forall x \quad \ell(x) = 2^{d(x)+1} - 1,$$

$$R(x,i) = \begin{cases} q(x,i) & \text{if } 1 \leqslant i \leqslant 2^{d(x)} - 1, \\ y_0 & \text{if } 2^{d(x)} \leqslant i \leqslant l(x) \text{ and } m(x,i) = 0, \\ y_1 & \text{if } 2^{d(x)} \leqslant i \leqslant l(x) \text{ and } m(x,i) = 1, \end{cases}$$

where y_0 and y_1 are two fixed strings such that $X(y_0)=0$ and $X(y_1)=1$. Moreover, let $Y=\{\langle x,i\rangle\}\mid R(x,i)\in X\}$. It is clear that $Y\in\ \leqslant_m^p(X)$ and it is easy to see that Y and l witness that $L\in T\cdot\ \leqslant_m^p(X)$. Conversely, if $L\in T\cdot\ \leqslant_m^p(X)$ it is very easy to derive a polynomial-time oracle Turing machine witnessing that $L\leqslant_T^p X$.

(a)(vi): Of course, the bounded versions $\leqslant_{k-T}^p$ of the Turing reducibility are representable by the following dot operators A_{k-T}.

$$A_{k-T}=\{z\mid |z|=2^k-1\wedge \text{ the value of the tree encoded by } z \text{ is } 1\}.$$

Now, we outline a general strategy for representing reducibilities. This strategy only provides representations by promise dot operators and thus it will be used for the reducibilities for which better (e.g. complementary) ad hoc representations do not seem exist. Consider a reducibility $\leqslant_r$. Generally, $\leqslant_r$ is defined by some kind of machines. More precisely, suppose that there exists a universal machine U_r such that, for all the languages E and F, $E\leqslant_r F$ iff there is a index y such that the machine (efficiently) simulated by U_r with index y together with oracle F satisfy some property α_r and the machine with oracle F accepts E. In most cases α_r can be decomposed into local properties, that is, the machine simulated by U_r with index y together with oracle O satisfy α_r iff for every x, the computation of U_r with index y and input x together with $O\cap\Sigma^{p(|x|)}$ (where p is a polynomial that depends on y) satisfy property β_r. To illustrate this quite abstract setting consider, for instance, the not-too-trivial case of the locally right positive Turing reducibility $\leqslant_{rpos}^p$. Recall that $E\leqslant_{rpos}^p F$ if there is a machine M witnessing that $E\leqslant_T^p F$ and such that, for every x and for every D, if $M^F(x)$ accepts then $M^{F\cup D}(x)$ accepts. In this case, the universal machine U_{rpos} simulates all the deterministic polynomial-time oracle Turing machines, that is, for each such a machine M there is an index y such that, $\forall O\ \forall x\ U_{rpos}(y,x)=M^O(x)$. Also, there is some fixed polynomial q such that if $M^O(x)$ halts within t steps then $U_{rpos}(y,x)$ halts within $q(t)$ steps. Property β_{rpos}, with regard to $U_{rpos}(y,x)$ and $O\subseteq\Sigma^{p(|x|)}$ (where p is a polynomial bounding the length of the queries made by computation $U_{rpos}(y,x)$, regardless of the oracle), requires that, for every $D\subseteq\Sigma^{p(|x|)}$, if $U_{rpos}^O(y,x)$ accepts then $U_{rpos}^{O\cup D}(y,x)$ must accept too. Returning to the general case, a dot operator $(A_r,B_r)\cdot$ which represents $\leqslant_r$ can be defined as follows.

$$A_r=\{z\mid(\exists t,x,y)[|z|=\langle 1^t,y,x\rangle\wedge U_r(y,x) \text{ halts within } t \text{ steps and together with } O_z \text{ satisfy property } \beta_r\wedge U_r^{O_z}(y,x) \text{ accepts}]\},$$

$$B_r=\{z\mid(\exists t,x,y)[|z|=\langle 1^t,y,x\rangle\wedge U_r(y,x) \text{ halts within } t \text{ steps and together with } O_z \text{ satisfy property } \beta_r\wedge U_r^{O_z}(y,x) \text{ rejects}]\},$$

where, for every z, O_z is the oracle language such that $O_z(u)=z(u)$ for $1\leqslant u\leqslant|z|$ and $O_z(u)=0$ otherwise. It is not hard to see that this strategy works for all the reducibilities listed above but the $\leqslant_{btt}^p$-reducibility.

(c)(i–xii), (b)(ii): Indeed, for some reducibilities either there are no better representations or we have not been able to find them. These are: $\leqslant^p_{ptt}$, $\leqslant^p_{rptt}$, $\leqslant^p_{lptt}$, $\leqslant^p_{pos}$, $\leqslant^p_{rpos}$, $\leqslant^p_{lpos}$, $\leqslant^{\mathrm{SN}}_m$, $\leqslant^{\mathrm{SN}}_c$, $\leqslant^{\mathrm{SN}}_d$, $\leqslant^{\mathrm{SN}}_{ptt}$, $\leqslant^{\mathrm{SN}}_{tt}$, $\leqslant^{\mathrm{RS}}_T$, and $\leqslant^{\mathrm{SN}}_T$.

As regards the other reducibilities, either complementary or simpler representations exist (as for the cases of $\leqslant^p_T$ and $\leqslant^p_{k-T}$).

(a)(i) It is immediate to verify that the dot operator $A_m\cdot = \{1\,\Sigma^*\}\cdot$ represents $\leqslant^p_m$.

Now, consider the truth-table reducibilities. First, recall their definitions (see [18]). A *tt-condition* is a string of the type $\langle \alpha, y_1, \ldots, y_k\rangle$ where α and $y_1, \ldots, y_k$ (possibly $k = 0$) are binary strings and α is supposed to be the encoding of a boolean function. A *tt-condition generator* is a recursive mapping of Σ^* into the set of tt-conditions. A *tt-condition evaluator* is a recursive mapping of the set $\{(\alpha, z) \mid \alpha, z \in \Sigma^*\}$ into $\{0, 1\}$. Let e be the tt-condition evaluator and let H be a language. A tt-condition $\langle \alpha, y_1, \ldots, y_k\rangle$ is *e-satisfied by H* if $e(\alpha, H(y_1)\cdots H(y_k)) = 1$. A language L is *polynomial-time tt-reducible* to a language H, written $L \leqslant^p_{tt} H$, if there is a polynomial-time computable generator g and a polynomial-time computable evaluator e such that $x \in L$ iff $g(x)$ is e-satisfied by H. By restricting the tt-conditions evaluators, strengthened versions of polynomial-time reducibilities may be defined. L is *polynomial-time conjunctive reducible* to $H (L \leqslant^p_c H)$ if the evaluator has the property that $e(\alpha, z) = 1$ iff $z = 1^{|z|}$. L is *polynomial-time disjunctive reducible* to $H (L \leqslant^p_d H)$ if the evaluator has the property that $e(\alpha, z) = 0$ iff $z = 0^{|z|}$. L is *polynomial-time positive tt-reducible* to $H (L \leqslant^p_{ptt} H)$ if the evaluator has the property that if $e(\alpha, z) = 1$ and y is such that $|y| = |z|$ and $z(i) = 1 \Rightarrow y(i) = 1$, for $i = 1, \ldots, |z|$, then $e(\alpha, y) = 1$. By bounding the number of queries, the above reducibilities can be further strengthened. Let k be a positive integer. A language L is *polynomial-time k-tt-reducible* to a language H, $(L \leqslant^p_{k-tt} H)$, if there is a polynomial-time computable generator g that outputs only tt-condition of arity at most k and a polynomial-time computable evaluator e such that $x \in L$ iff $g(x)$ is e-satisfied by H. Similarly, the reducibilities $\leqslant^p_{k-c}$, $\leqslant^p_{k-d}$, and $\leqslant^p_{k-ptt}$ are defined.

(a)(ii–v): The bounded versions can easily be represented. Let $f_1, f_2, \ldots, f_{2^{2^k}}$ be an enumeration of all the boolean functions of arity k. Let $A_{k-tt} = \{z0^y \mid |z| = k \wedge 1 \leqslant y \leqslant 2^{2^k} \wedge f_y(z) = 1\}$. It is easy to see that $A_{k-tt}\cdot$ represents $\leqslant^p_{k-tt}$. In a very similar way it is possible to define $A_{k-c}\cdot$, $A_{k-d}\cdot$, and $A_{k-ptt}\cdot$ that represent, respectively, $\leqslant^p_{k-c}$, $\leqslant^p_{k-d}$, and $\leqslant^p_{k-ptt}$.

(b)(i): Between the bounded and the unbounded versions there is the $\leqslant^p_{btt}$-reducibility. This is the only one that requires a non-recursive representation:

$$A_{btt} = \{z \mid (\exists k, \ell)[|z| = \langle k, \ell\rangle \wedge k \in \mathscr{I} \wedge z(1)z(2)\cdots z(\ell) \in A_{k-tt}]\},$$

$$B_{btt} = \{z \mid (\exists k, \ell)[|z| = \langle k, \ell\rangle \wedge k \in \mathscr{I} \wedge z(1)z(2)\cdots z(\ell) \notin A_{k-tt}]\},$$

where $\mathscr{I}$ is an immune set.

(a)(vii–ix): Now, consider the unbounded versions. Let U be a universal deterministic Turing machine. That is, a machine such that for any deterministic Turing machine M there exists an index i such that $\forall x\ U(i, x) = M(x)$. Also, U can be defined so that there is a constant c for which for any machine M and any index j for M, if the computation $M(x)$ halts within t steps then $U(j,x)$ halts within ct^2 steps. Now, consider the following language:

$$A_{tt} = \{z \mid (\exists k,t,y)[|z| = 1^k 0^t 1 y \wedge U(y, z(1)z(2)\cdots z(k)) \text{ halts within } t \text{ steps in an accepting state}]\}.$$

It is not hard to verify that $A_{tt}\cdot$ represents $\leqslant_{tt}^{p}$. The case of the conjunctive reducibility is easier. Let $A_c = \{z \mid (\exists k,y)[z = 1^k 0 y \wedge k = \log\lceil |z| \rceil]\}$. In a very similar way can be defined a dot operator $A_d\cdot$ that represents $\leqslant_{d}^{p}$.

(a)(xi–xii): As regards the nondeterministic reducibilities, it is immediate to verify that $\exists\cdot$ represents $\leqslant_{m}^{\mathrm{NP}}$. Also, it is easily seen that the composition of $\exists\cdot$ and $A_c\cdot$ represents $\leqslant_{c}^{\mathrm{NP}}$ and the composition of $\exists\cdot$ and $A_T\cdot$ represents $\leqslant_{T}^{\mathrm{NP}}$. Thus, from Theorem 3.3 both reducibilities can be represented by complementary dot operators.

(b)(iii, iv): The reducibility $\leqslant_{m}^{\mathrm{BPP}}$ can be represented by the BP-operator BP$\cdot$ and $\leqslant_{m}^{\mathrm{RP}}$ is represented by the RP-operator RP$\cdot$.

(a)(xiv): In order to show a dot operator representing the $\leqslant_{T}^{\mathrm{PS}}$-reducibility we need some notations. For every n we consider any string z of length $n2^n$ as describing a direct graph G_z on 2^n vertices as follows. Let $y_1, y_2, \ldots, y_{2^n}$ be such that $z = y_1 y_2 \cdots y_{2^n}$ and $|y_1| = |y_2| = \cdots = c|y_{2^n}| = n$, the vertices of G_z are all the strings of length n and there is an edge from u to v iff it holds that $v = y_j$ being u the jth string of length n (in the lexicographic order). Note that all the vertices of G_z have out-degree at most one. Define the following language

$$A_{\mathrm{PS}} = \{z \mid (\exists n)|z| = n2^n \wedge G_z \text{ has an oriented path from } 0^n \text{ to } 1^n\}.$$

We show that the dot operator $A_{\mathrm{PS}}\cdot$ represents the $\leqslant_{T}^{\mathrm{PS}}$-reducibility. Let $E \leqslant_{T}^{\mathrm{PS}} F$ via a polynomial-space bounded machine M. It is easy to see that there is a way to encode any possible instantaneous configuration (included the content of the oracle tape) of a computation of M on any input x as a string of length $q(|x|)$, for some fixed polynomial q. Also, we can assume that this encoding has the following properties: (1) any instantaneous configuration in the query state is not encoded, instead of it are encoded the two successive configurations corresponding to the two possible answers of the oracle; (2) the answer of the oracle is encoded by a specific bit of the encoding string (say the last bit); (3) the initial configuration is encoded by the string of all 0's and the accepting configuration is encoded by the string of all 1's. By using this type of encoding it is not hard to see that $E \in A_{\mathrm{PS}} \cdot \leqslant_{T}^{\mathrm{PS}}(F)$. The converse is easier and it is left to the reader.

(b): From Theorem 3.4(b) and the existence of relativized worlds in which the following classes BH [7], NP$\cap$co-NP [25], RP [25], and BPP [10] lack complete languages, it follows from the representability of $\leqslant_{T}^{p}$ that reducibilities $\leqslant_{btt}^{p}$, $\leqslant_{T}^{\mathrm{SN}}$, $\leqslant_{m}^{\mathrm{BPP}}$, and $\leqslant_{m}^{\mathrm{RP}}$ cannot be represented by complementary dot operators.

(c): For all the reducibilities of (c) but $\leqslant_{lpos}^{p}$ and $\leqslant_{rpos}^{p}$ it is easy to see that P$=$NP implies that they can be represented by complementary dot operators. This leaves open the question whether they admit complementary representations. For the strong reducibilities $\leqslant_m^{\mathrm{SN}}$, $\leqslant_c^{\mathrm{SN}}$, $\leqslant_d^{\mathrm{SN}}$, $\leqslant_{ptt}^{\mathrm{SN}}$, $\leqslant_{tt}^{\mathrm{SN}}$, and $\leqslant_T^{\mathrm{RS}}$, it seems unlikely since it would imply that $\mathrm{NP} \cap \text{co-NP}$ has complete languages. We also suspect that $\leqslant_{lpos}^p$ and $\leqslant_{rpos}^p$ cannot be represented by complementary dot operators. □

It was shown by Ambos-Spies [2] for every language A that its polynomial-time Turing closure $P(A) = \leqslant_T^p(A)$ has a $\leqslant_m^p$-complete language. By Theorem 3.4 we can generalize this to all reducibilities shown to be representable by complementary dot operators in the above Theorem 5.2(a).

Corollary 5.3. *Let A be any language. The reducibility closures $\leqslant_{k-tt}^p(A)$, $\leqslant_{k-c}^p(A)$, $\leqslant_{k-d}^p(A)$, $\leqslant_{k-ptt}^p(A)$, $\leqslant_{k-T}^p(A)$, $\leqslant_{tt}^p(A)$, $\leqslant_c^p(A)$, $\leqslant_d^p(A)$, $\leqslant_T^p(A)$, $\leqslant_m^{\mathrm{NP}}(A)$, $\leqslant_T^{\mathrm{NP}}(A)$, and $\leqslant_T^{\mathrm{PS}}(A)$ are principal $\leqslant_m^p$-ideals (this implies that they have a $\leqslant_m^p$-complete language).*

The following result shows a limit on the representability of reducibilities by dot operators. Namely: If for a reducibility there is no polynomial bound on the size of the asked questions then it is not representable by a dot operator. The proposition can be proven by straightforward diagonalization.

Proposition 5.4. *The many-one and Turing reducibilities given by* PSPACE *or* EXPTIME *computations (without restriction on the size of the questions) are not representable by dot operators.*

We have represented reducibilites by languages. This allows to apply computational complexity notions to reducibilites: Say that a reducibility $\leqslant^r$ can be represented in a complexity class $\mathscr{C}$ if $\leqslant^r$ is represented by a language in $\mathscr{C}$ (an extended definition can be given for promise languages). For example, all the reducibilities in Theorem 5.2(a) are representable in LOGSPACE. The natural way to compare reducibilities is by the refinement relation: reducibility $\leqslant^{r_1}$ *refines* reducibility $\leqslant^{r_2}$ if for all L and E, $L \leqslant^{r_1} E$ implies $L \leqslant^{r_2} E$. For instance, $\leqslant_{tt}^p$ refines $\leqslant_T^p$. The usual completeness notion for preorders can be applied to the refinement relation: A reducibility $\leqslant^r$ is *complete* w.r.t. the refinement relation for a set of reducibilities R if $\leqslant^r \in R$ and for every $\leqslant^s \in R$ it holds that $\leqslant^s$ refines $\leqslant^r$. The following proposition follows from the definitions.

Proposition 5.5. *Let $\leqslant^{r_1}$ and $\leqslant^{r_2}$ be two reducibilities represented by the languages A_1 and A_2, respectively. It holds that $\leqslant^{r_1}$ refines $\leqslant^{r_2}$ if and only if $A_1 \leqslant^{\mathrm{op}} A_2$.*

The following theorem uses Corollary 4.7.

Theorem 5.6 (completeness of reducibilities). (a)(i) *The reducibility $\leqslant_T^p$ is complete w.r.t. the refinement relation for the set of reducibilities representable in* LOGTIME, *and also for the set of reducibilities representable in* POLYLOGTIME. (ii) *Any representable reducibility that is not representable in* POLYLOGTIME *does not refine $\leqslant_T^p$.*

(b)(i) *The reducibility* $\leqslant_T^{\mathrm{PS}}$ *is complete w.r.t. the refinement relation for the set of reducibilities representable in* LOGSPACE, *and also for the set of reducibilities representable in* POLYLOGSPACE. (ii) *Any representable reducibility that is not representable in* POLYLOGSPACE *does not refine* $\leqslant_T^{\mathrm{PS}}$.

Proof. (a)(i): It is easy to see that the language T is complete for LOGTIME and POLYLOGTIME w.r.t. $\leqslant_{mp}^{plt}$, i.e. (1) T is in LOGTIME and (2) every set in POLYLOGTIME is $\leqslant_{mp}^{plt}$-reducible to T. By (1), $\leqslant_T^p$ is representable in LOGTIME, and by (2), Lemma 4.4 and Proposition 5.5 every reducibility representable in POLYLOGTIME refines $\leqslant_T^p$.

(a)(ii): It follows from the easy fact that POLYLOGTIME is closed downward w.r.t. $\leqslant_{mp}^{plt}$ together with Theorem 4.6 and Proposition 5.5.

(b): It is analogous to (a). It suffices to observe that the language A_{PS} (defined in the proof of Theorem 5.2) representing $\leqslant_T^{\mathrm{PS}}$ is complete for LOGSPACE and POLYLOGSPACE w.r.t. $\leqslant_{mp}^{plt}$, and POLYLOGSPACE is closed downward w.r.t. $\leqslant_{mp}^{plt}$. □

6. Open question

The authors could give several examples of operators and reducibilities which can be represented as dot operators according to Definitions 2.1 and 2.2. But they could not give a characterization of the set of (complementary) dot operators (like they gave for example for the set of (complementary) leaf language classes [5]).

Acknowledgements

The authors are grateful to Klaus Ambos-Spies, Andreas Eisenblätter, Wolfgang Merkle, Frank Stephan, and the referees for helpful comments.

References

[1] L.M. Adleman, K. Manders, Reducibility, randomness, and intractability, ACM Symp. on Theory of Computing, 1977, pp. 151–163.

[2] K. Ambos-Spies, A note on complete problems forcomplexity classes, Inform. Process. Lett. 23 (1986) 227–230.

[3] J. Balcazar, J. Diaz, J. Gabarro, Structural Complexity I, Springer, Berlin, 1988.

[4] T. Baker, A. Selman, A second step toward the polynomial hierarchy, Theoret. Comput. Sci. 8 (1979) 431–442.

[5] B. Borchert, R. Silvestri, A characterization of the leaf language classes, Inform. Process. Lett. 63 (1997) 153–158.

[6] D.P. Bovet, P. Crescenzi, R. Silvestri, A uniform approach to define complexity classes, Theoret. Comput. Sci. 104 (1992) 263–283.

[7] J. Cai, L.A. Hemachandra, The Boolean hierarchy: hardware over NP, Proc. 1st Structure in Complexity Theory Conf., Lecture Notes in Computer Science, vol. 223, Springer, Berlin, 1994, pp. 105–124.

[8] A. Eisenblätter, Über einige zentrale Operatoren in der Komplexitätstheorie, Diploma Thesis, Universität Heidelberg, 1994.
[9] R. Gavaldà, J. Balcázar, Strong and robustly strong polynomial time reducibilities to sparse sets, Theoret. Comput. Sci. 88 (1) (1991) 1–14.
[10] J. Hartmanis, L. Hemachandra, Complexity classes without machines: on complete languages for UP, Theoret. Comput. Sci. 58 (1988) 129–142.
[11] L.A. Hemachandra, S. Jain, On the limitations of locally robust positive reductions, Internat. J. Found. Comput. Sci. 2 (3) (1991) 237–255.
[12] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, K. Wagner, On the power of polynomial time bit-reductions, Proc. 8th IEEE Structure in Complexity Theory Conf., 1993, pp. 200–207.
[13] L. Hemaspaandra, H. Vollmer, The Satanic notations: counting classes beyond #P and other definitional adventures, SIGACT News 26 (1) (1995) 2–13.
[14] U. Hertrampf, H. Vollmer, K. Wagner, On balanced vs. unbalanced computation trees, Universität Würzburg, Tech. Rep. no. 82, May 1994.
[15] N. Immerman, Languages that capture complexity classes, SIAM J. Comput. 16 (1987) 760–778.
[16] B. Jenner, P. McKenzie, D. Thérien, Logspace and logtime leaf languages, Proc. 9th IEEE Structure in Complexity Theory Conf., 1994, pp. 242–254.
[17] J. Köbler, U. Schöning, J. Toran, The Graph Isomorphism Problem: Its Structural Complexity, Birkhäuser, Basel, 1993.
[18] R. Ladner, N. Lynch, A. Selman, A comparison of polynomial time reducibilities, Theoret. Comput. Sci. 1 (1975) 103–123.
[19] T.J. Long, Strong nondeterministic polynomial-time reducibilities, Theoret. Comput. Sci. 21 (1982) 1–25.
[20] C.H. Papadimitriou, Games against nature, J. Comput. System Sci. 31 (1985) 288–301.
[21] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, Reading, MA, 1994.
[22] U. Schöning, Probabilistic complexity classes and lowness, Proc. 2nd IEEE Structure in Complexity Theory Conf., 1987, pp. 2–8.
[23] A. Selman, Reductions on NP and P-selective sets, Theoret. Comput. Sci. 19 (1982) 287–304.
[24] I. Simon, J. Gill, Polynomial reducibilities and upward diagonalizations, Proc. 9th ACM Symp. on Theory of Computing, 1977, pp. 186–194.
[25] M. Sipser, On relativization and the existence of complete sets, Proc. 9th Internat. Colloq. on Automata, Languages and Programming (ICALP), Lecture Notes in Computer Science, vol. 140, Springer, Berlin, 1982, pp. 523–531.
[26] S. Skyum, L.G. Valiant, A complexity theory based on Boolean algebra, J. ACM 32 (1985) 484–505.
[27] L. Stockmeyer, The polynomial-time hierarchy, Theoret. Comput. Sci. 3 (1977) 23–33.
[28] S. Tang, O. Watanabe, On tally relativizations of BP-complexity classes, SIAM J. Comput. 18 (1989) 449–462.
[29] S. Toda, M. Ogiwara, Counting classes are at least as hard as the polynomial-time hierarchy, SIAM J. Comput. 21 (1992) 316–328.
[30] J. Toran, Complexity classes defined by counting quantifiers, J. ACM 38 (1991) 753–774.
[31] L.G. Valiant, V.V. Vazirani, *NP* is easy as detecting unique solutions, Theoret. Comput. Sci. 47 (1986) 85–93.
[32] H. Veith, Succinct representations and leaf languages, Proc. 11th IEEE Conf. on Computational Complexity, 1996, pp. 118–126.
[33] N.K. Vereshchagin, Relativizable and nonrelativizable theorems in the polynomial theory of algorithms, Russian Acad. Sci. Izv. Math. 42 (1994) 261–298.
[34] H. Vollmer, Uniform characterizations of complexity classes, ACM SIGACT-Newslett. 30 (1) (1999) 17–27.
[35] K. Wagner, The complexity of combinatorial problems with succinct input representations, Acta Inform. 23 (1986) 325–356.
[36] C. Wrathall, Complete sets and the polynomial-time hierarchy, Theoret. Comput. Sci. 3 (1977) 23–33.