Electronic Colloquium on Computational Complexity - Reports Series 1995 - available via:
ECCC
TR95-048
FTP: ftp.eccc.uni-trier.de:/pub/eccc/
WWW: http://www.eccc.uni-trier.de/eccc/
Email: ftpmail@ftp.eccc.uni-trier.de with subject 'help eccc'

Succinct Representation and Leaf Languages

Helmut Veith
Institut für Informationssysteme, Technische Universität Wien
A-1040 Wien, Paniglgasse 16, Austria
veith@vexpert.dbai.tuwien.ac.at

Abstract

In this paper, we present stronger results in the theory of succinct problem representation and establish a close relationship between succinct problems and leaf languages. As a major tool, we use projection reductions from descriptive complexity theory.

A *succinct problem* is a problem whose high complexity stems from the fact that its instances are not given straightforward, but are themselves encoded by boolean circuits. In [Balcázar et al. 92, Papa, Yann 85] there have been developed methods to quantify the complexity leap obtained this way. We prove a strictly stronger version of this result which allows iterative application and completeness results under projection reductions.

A *leaf language* [Bovet et al 92, Bovet et al 91] is the language of words accepted by a nondeterministic Turing Machine where for acceptance the word obtained by concatenating the bits at the leaves of the computation graph must fit in a certain pattern. Complexity bounds on the leaf pattern allow to uniformly define a lot of complexity classes [Bovet et al 92, Bovet et al 91, Hertrampf et al 93, Jenner et al 94]. We show that the succinct version $s(\Pi)$ of a problem Π is complete under projection reductions for the leaf language $\mathrm{Leaf}(\Pi)$. Thus, we obtain an easy uniform description of generic problems complete under projection reductions for all classes definable as leaf languages.

In the last part, we prove that $s(\Pi)$ is not complete under *monotone* reductions and propose an alternative succinct representation model which allows completeness even under monotone projection reductions. This approach results in the definition of generic problems complete under *monotone* projection reductions for a large number of classes. Thus we positively solve a question by Stewart, posed in [Stewart 93a, Stewart 91].

1 Introduction

Among the methods developed for complexity classes beyond **NP**, we shall investigate and interrelate two such which have gained wide interest and have initiated threads of publications: leaf languages and succinct problem representation. Our main tool will be parametrized versions of projection reductions known from descriptive complexity theory.

Leaf languages were introduced in [Bovet et al 92, Bovet et al 91] as a unified approach to define and treat many common complexity classes. The crucial idea is that a computational problem can be solved in two steps. First, it is applied to an **NP** machine. Then the pattern obtained from concatenating the output of the machine in *all* its potential computation paths (the so called *leaf string*) is matched against two sets of strings, one for acception and one for rejection. It was shown in [Bovet et al 92, Bovet et al 91] that most common complexity classes have fixed characteristic pattern sets. The classes for which the pattern sets for acception and rejection are complementary were termed *syntactic* complexity classes by [Papadimitriou 94] and were further investigated in [Papadimitriou 94, Jenner et al 94, Hertrampf et al 93]. [Jenner et al 94] generalize the notion of leaf languages to **NL** machines and obtain a lot of results similar to the **NP** case. In [Hertrampf et al 93], algebraic methods involving permutation groups developed by [Barrington 89, Cai, Furst 87] were used to characterize PSPACE by leaf languages over fragments of the regular languages. Moreover, they show that completeness of a pattern set under polylogtime reductions for a class C implies that the leaf language obtained is an exponentially higher analogue of C. This result gave strong support of our intuition that syntactic leaf languages are tightly connected to succinct problems.

The *succinctness* of the representation of a computational problem is well-known to have a strong impact on the computational complexity of the problem. In [Galperin, Widgerson 83], problems were investigated whose high complexity stems from the fact their instances are not given straightforward, but are themselves encoded by boolean circuits. It was shown that the succinct version of certain such problems becomes exponentially harder. Further steps were taken in [Papa, Yann 85, Lozano, Balcázar 89, Balcázar et al. 92], where the latter provide a general upgrading theorem deriving completeness of the succinct problem under polynomial time reductions from completeness under logtime reductions for the normally represented problems. In [Eiter et al. 94] the assumption was relaxed to polylogtime completeness, and in [Veith 94] the conclusion was sharpened to logspace completeness. The methodology developed was exploited in [Eiter et al. 94, Gottlob et al 95] for proving expression complexity [Vardi 82] of database query languages and logics such as Henkin logic known from finite model theory. The main drawback so far is that logspace reductions are not known to be

subsumed by polylogtime reductions, hence there is no easy means for iterating the completeness upgrade technique up to higher complexity classes. Similarly, there is no immediate way to iterate the leaf language technique.

Projection reductions were defined in [Immerman 87] as a highly restrictive notion of logical reduction. In descriptive complexity theory, computational problems are identified with sets of finite structures. A projection reduction then is a quantifierfree logical reduction, i.e. a mapping between structures, where each tuple in the output structure depends on only one tuple in the input structure. Following Immerman's seminal paper, many problems have been shown complete for **NP** and other classes by Stewart, for instance in [Stewart 92, Stewart 93a, Stewart 94, Stewart 91]. The latter paper gives the first methodology of projection reductions. Monotone projection reductions are projection reductions where the input relations appear only positively in the reduction formulas. In [Stewart 94], several problems are shown to be complete under monotone projection reductions for *monotone* **NP**. However, there are no problems known to be complete under monotone projection reductions for other, more common complexity classes. Note that quantifierfree reductions are subsumed by logtime reductions (cf. [Eiter et al. 94, Veith 94]), hence the above results about projection reductions imply results about logtime reductions.
Since a logically defined problem can be seen as a property of structures, first order logic can be extended by generalized quantifiers which check those properties over arbitrary structures, if the problem considered is complete under logical reductions. Expressibility and normal form results for such logics were obtained for instance by [Immerman 87] and [Stewart 92, Stewart 93a, Stewart 94].

The main results of *this paper* are the following:

Section 3 We prove a strictly stronger complexity upgrading theorem for succinct problems. In particular, we show that polylogtime reducibility between two problems implies monotone projection reducibility between the succinct versions of the problems. Moreover, we show that if a problem is $C-$hard under polylogtime reductions then its succinct version is hard under projection reductions for the exponentially harder analogue of C. Note that our results are not only the first to allow iterative application, but the succinct problems are complete even under a more restrictive notion than the original problems. In fact, even concrete completeness results under projection reductions are often non-trivial, as can be seen from the papers by Immerman and Stewart cited above.

At second sight, this seemingly paradox effect turns out to redeem in a sense promises made in the early investigations of succinct representations: While the belief that the regularity of boolean circuits allows to design feasible algorithms was unfounded, boolean circuits were found to be subject to

powerful manipulation.

Nevertheless completeness proofs under projection reductions tend to be somewhat complicated, therefore the abstract construction of a projection reduction from a polylogtime reduction can be imagined to be involved. Informally speaking, the key observation for this result is that deterministic Turing machines can be simulated by highly uniform circuits which are definable by projection reductions.

Section 4 We prove that the succinct version of a problem is complete under projection reductions for the leaf language it defines, regardless of the problem itself. This basic interrelation between succinct problems and leaf languages provides a standard procedure to obtain completeness results under projection reductions for many complexity classes. Simultaneously, the leaf language of a problem gives an immediate account of its succinct complexity. Moreover, we show that a slightly weaker form of the upgrading results from Section 3 follows from the results of this section. In particular, we show that the leaf language approach gives strictly sharper bounds for the complexity of succinct problems.

Section 5 We prove that the completeness results obtained above are optimal in the sense that they cannot be sharpened to monotone projections. However, we exhibit a tricky modification of the boolean circuit model which allows to turn the projection reductions used in this paper into monotone projection reductions. Consequently, we solve the open question if there exist problems which are complete under monotone projection reductions in the positive for a large class of complexity classes including **NP**.

Section 6 We show how to obtain complete problems and capturing logics for many classes up to Kalmár elementary computability.

Acknowledgements

I would like to thank Thomas Eiter and Georg Gottlob for introducing me to the subject and for discussing an earlier version of this paper.

2 Preliminaries on Complexity and Circuits

The function $\log k$ denotes the number of digits of the binary representation of k. Given a string $w \in \{0,1\}^+$, $val(w)$ denotes the natural number whose binary representation is w. An initial segment $\{0, 1, \ldots, n-1\}$ of the natural numbers

is denoted by n. We shall use the digits $0, 1$ simultaneously as propositional constants, binary digits and natural numbers. Given a function $f : A \to B$, the size $|f|$ of f is defined as the cardinality of the graph $\{(x, f(x)) : x \in A\}$.

2.1 Descriptive Complexity

A signature is a sequence $(P_1^{a_1}, \ldots, P_k^{a_k})$ of relation symbols with associated arities $a_1, \ldots, a_k$. A *finite structure* over τ is a tuple $(n, P_1^{\mathcal{A}}, \ldots, P_k^{\mathcal{A}})$, where $P_i^{\mathcal{A}} \subseteq n^{a_i}$. n is called the *universe* of $\mathcal{A}$, and denoted $|\mathcal{A}|$. The set of all finite structures over τ is denoted by $\mathrm{Struct}(\tau)$. Let $\mathcal{A}, \mathcal{B} \in \mathrm{Struct}(\tau)$, s.t. $|\mathcal{A}| = |\mathcal{B}|$. Then $\mathcal{A} \subseteq \mathcal{B}$ if $P_i^{\mathcal{A}} \subseteq P_i^{\mathcal{B}}$ for $1 \leq i \leq k$. A *computational problem* over signature τ is a set $\Pi \subseteq \mathrm{Struct}(\tau)$, s.t. Π is closed under isomorphisms. Π is *monotone*, if $\mathcal{A} \subseteq \mathcal{B}$ and $\mathcal{A} \in \Pi$ implies $\mathcal{B} \in \Pi$. Note that the elementarily decidable computational problems coincide with the classes definable in $\omega-$order logic. $\tau^{(l)} = (P_1^{la_1}, \ldots, P_k^{la_k})$ is called the *$l-$ary variant* (or vectorization) of τ. For a problem T, let $T^{(l)}$ denote the problem T over $l-$tuples, i.e. over universe $\underbrace{n \times \cdots \times n}_{l \text{ times}}$, s.t. $l-$tuples are understood as numbers from $\{0, \ldots, n^l - 1\}$.

First Order Logic $FO_s(\tau)$ is the language of all first order sentences over signature τ with logical predicates for equality $=$ and successor $s(\_, \_)$, and two constants $0, max$ denoting the minimal and maximal element wrt to the successor relation. Given signatures τ, σ and a natural number k, a k-ary interpretation of τ into σ is a definition of the $\sigma^{(k)}$ relations in terms of τ. A $k-$ary interpretation I is written as a set of equations of the form $P(\mathbf{x}) = \phi$ where $P \in \sigma^{(k)}$ and ϕ is a formula over signature τ. For a structure $\mathcal{A} \in \mathrm{Struct}(\tau)$, let $I(\mathcal{A})$ denote the structure over $\sigma^{(k)}$ which is defined by I. Let $T \subseteq \mathrm{Struct}(\tau)$, $S \subseteq \mathrm{Struct}(\sigma)$ be problems. We say that T is $\mathcal{L}$–reducible to S if there exist an interpretation I of τ into σ, s.t. I is an $\mathcal{L}-$formula, and for all $\mathcal{A} \in \mathrm{Struct}(\tau)$, $\mathcal{A} \in S$ iff $I(\mathcal{A}) \in T^{(k)}$, with k being the arity of I. By restricting the logic for the interpretations we obtain low-level reductions: A *projection reduction* [Immerman 87] is a reduction whose defining formulas are quantifierfree and in disjunctive normal form $\bigvee_i \alpha_i$, s.t. the α_i are mutually exclusive and each α_i contains maximally one relation from τ. If all occurrences of τ relations in $\bigvee_i \alpha_i$ are unnegated, the reduction is called a *monotone projection reduction*. Projection reductions are not known to be closed under iteration, while monotone projection reductions are so. Still, we can use an easy yet important fact about the interaction between monotone projections and projections due to [Stewart 93b].

Lemma 1 *If $A \leq_{proj} B \leq_{proj}^{mon} C$ then $A \leq_{proj} C$.*

The result can be seen from the fact that substituting a projection formula for a relation occuring in a monotone projection formula yields a projection formula again. Let $\leq_X$ be some reduction relation and Π be a problem. Then $[\Pi]_X$ denotes the $\leq_X$-ideal generated by Π, i.e. $[\Pi]_X = \{L | L \leq_X \Pi\}$. For a class $\mathbf{C}$ of languages, this amounts to $[\mathbf{C}]_X = \bigcup_{\Pi \in \mathbf{C}} [\Pi]_X$. Using this notation, we can write Stewart's lemma as

$$\left[[\Pi]^{mon}_{proj}\right]_{proj} = [\Pi]_{proj}$$

Let us consider simplifications which will allow us to treat logical reductions in a more friendly way: Although it is sufficient to have one equation $P(\mathbf{x}) = \phi$ for each $P \in \sigma^{(k)}$, we shall allow for multiple equations where $P = \phi, P = \psi$ stands for $P = \phi \vee \psi$. Thus, we can write each disjunct of a projection reduction in a seperate row. Moreover, we allow for expressions involving addition and subtraction of numerical constants on both sides of the equations. The expression $y + c$ for example, can be easily eliminated by replacing it with a new variable x_c and adding the conjunct $\bigwedge_{0 \leq i < c} s(x_i, x_{i+1}) \wedge x_0 = y$ to the right hand side of the equation. The successor relation on the domain induces a quantifier-free definable lexicographical order on vectors of fixed arity. Therefore, we shall treat vectors of arity k over domain n just like ordinary numbers over domain n^k. In particular, if it is understood that x is a $k-$tuple, $x = 0$ means $x = (0, \ldots, 0)$, $x = max$ means $x = (max, \ldots, max)$, and $s(x, y)$ means that the $k-$tuple y is the lexicographical successor of x. An easy example of a projection reduction can be found in the proof of Lemma 3.

Take some signature $\tau = (P_1^{a_1}, \ldots, P_k^{a_k})$ and let $l = max\{a_1, \ldots, a_k\}$. Consider signature $\alpha = (Q^{l+1})$. One can see that the structures over τ can be embedded into Struct(α) by defining

$$Q(i-1, x_1, \ldots, x_l) \equiv P_i(x_1, \ldots, x_{a_i})$$

for $1 \leq i \leq k-1$ and

$$Q(i-1, x_1, \ldots, x_l) \equiv P_k(x_1, \ldots, x_{a_k})$$

for $i \geq k$. Thus, we use the first argument of Q to distinguish between the relations; for reasons to become clear in Lemma 2, we postulated that all argument values exceeding $k-1$ refer to predicate P_k. For each structure $\mathcal{A} \in$ Struct(τ), let $single(\mathcal{A})$ denote the corresponding Struct(α) structure. For a problem A, $single(A) = \{single(\mathcal{A}) : \mathcal{A} \in A\}$. (To be rigorous, we have to postulate that all structures over τ have domains of size at least l. If we would encode i in binary by sequences of 0 and max, we could even resort to structures of size at least 2. However, in any case only a constant number of small structures are affected which is irrelevant for complexity-theoretic considerations.) In fact, the *single* encoding provides a convenient normal form for different signatures; this is formalized by the following lemma:

Lemma 2 *Let $A \subseteq \mathrm{Struct}(\tau)$, and Π be an arbitrary problem. Then*

1. $A \equiv^{mon}_{proj} single(A)$
2. $\Pi \leq_{proj} A$ *iff* $\Pi \leq_{proj} single(A)$
3. $A \leq_{proj} \Pi$ *iff* $single(A) \leq_{\mathrm{proj}} \Pi$

Proof:

1a. $A \leq^{mon}_{proj} single(A)$: For each $1 \leq i \leq k-1$, we include the rule

$$Q(y, x_1, \ldots, x_l) \quad = \quad y = i-1, P_i(x_1, \ldots, x_{a_i})$$

Moreover, we take the rule

$$Q(y, x_1, \ldots, x_l) \quad = \quad y \geq k-1, P_k(x_1, \ldots, x_{a_k})$$

where $y \geq k-1$ is expressed by $y \neq 0, \ldots, y \neq k-2$.

1b. $single(A) \leq^{mon}_{proj} A$: For $1 \leq i \leq k-1$ we set

$$P_i(x_1, \ldots, x_{a_i}) \quad = \quad y = i-1, Q(y, x_1, \ldots, x_l)$$

Moreover, we add the rule

$$P_k(x_1, \ldots, x_{a_k}) \quad = \quad y \geq k-1, Q(y, x_1, \ldots, x_l)$$

2. Suppose that $\Pi \leq_{proj} A$. From 1a. we know that $A \leq^{mon}_{proj} single(A)$, by Lemma 1 we conclude that $\Pi \leq_{proj} single(A)$. The converse implication works analogously.

3a. $\rightarrow$: Suppose that $A \leq_{proj} \Pi$, then $single(A) \leq^{mon}_{proj} A \leq_{proj} \Pi$. Lemma 1 is not applicable here, but a closer investigation of the reduction in 1b. shows that each of the projection formulas contains only one occurence of Q. Hence, their negations remain projection formulas, and can be inserted into the projection formulas between A and Π.

3b. $\leftarrow$: Suppose that $single(A) \leq_{proj} \Pi$, then $A \leq^{mon}_{proj} single(A) \leq_{proj} \Pi$. Again, we have to resort to the reduction from 1a. When can Q become false? We have built the *single* embedding in such a way that each y must satisfy one of the left conditions in the reduction. Therefore, $Q(y, x_1, \ldots, x_l)$ can be false only if either $y = i-1, 1 \leq i \leq k-1$ and $\neg P_i(x_1, \ldots, x_{a_i})$, or $y \geq k-1$ and $P_k(x_1, \ldots, x_{a_k})$. Thus, the negation of Q can be expressed as a projection formula. The result follows. $\square$

Since monotone projection reductions are the weakest reductions considered in this paper, we may without loss of generality assume that a structure is given in the *single* encoding whenever necessary.

A structure $\mathcal{A} \in \mathrm{Struct}(\tau)$ is usually encoded by a string, built as the concatenation of the domain size $|\mathcal{A}|$ (in binary) and the truth values of $P_1^{\mathcal{A}}, \ldots, P_k^{\mathcal{A}}$ upon lexicographical enumeration of their arguments (cf. [Fagin 74, Gurevich 88]). In our context it is sufficient to take the concatenation of the truth values of $single(\mathcal{A})$ instead.

2.2 Boolean Circuits

A boolean circuit of size n can be encoded as an ordered structure over signature $\zeta = (\sqcap^3, \sqcup^3, \sim^2, \lhd^1, \square^1)$, where $\sqcap$ and $\sqcup$ denote conjunction and disjunction respectively, $\sim$ denotes negation, $\lhd$ the input gates, and $\square$ the output gate. Let for example $\mathcal{M} \in \mathrm{Struct}(\zeta)$. Then $\mathcal{M} \models \sqcap(8,5,3)$ means that the circuit encoded by $\mathcal{M}$ computes gate number 8 as the conjunction of gates 5 and 3. A structure $\mathcal{A} \in \mathrm{Struct}(\zeta)$ represents a boolean circuit if it contains an output gate and each gate is either an input gate or is computed from smaller gates. We explicitly allow that there are domain elements where no gate is situated, however all gates but the input gates must refer to other gates.

2.3 Succinct Encoding

A boolean circuit with k input gates in a natural way defines a unary relation over domain 2^k: On input of the binary representation of a tuple, the circuit outputs whether the relation contains the tuple. Consequently, a circuit with ck input gates determines a $c-$ary relation over domain 2^k (and simultaneously, a unary relation over 2^{kc}). Let $\tau = (R_1^{a_1}, \ldots, R_r^{a_r})$ be a signature and $\mathcal{A} \in \mathrm{Struct}(\tau)$. The succinct signature $s(\tau)$ is defined as ζ. The intended meaning is that a structure $C \in \mathrm{Struct}(\zeta)$ defines the relation $single(\mathcal{A})$.

Let τ be like above and $\Pi \subseteq \mathrm{Struct}(\tau)$ be a problem over τ. We say that Π is *self-embeddable* if $\mathcal{A} = (n, R_1^{\mathcal{A}}, \ldots, R_r^{\mathcal{A}}) \in \Pi$ implies $(n \cup \{u\}, R_1^{\mathcal{A}}, \ldots, R_r^{\mathcal{A}}) \in \Pi$ for all $\mathcal{A} \in \mathrm{Struct}(\Pi)$. Prima facie boolean circuits can only represent structures whose domain size is a power of 2. The notion of self-embeddability distinguishes those problems which are not affected by this restriction because their membership property is not influenced by adding isolated domain elements (for a graph problem isolated element means isolated vertex). 3-colorability is a self-embeddable property because it is not affected by adding isolated vertices. The standard encoding of Hamiltonicity, on the other hand, is not self-embeddable,

because a graph with an isolated vertice cannot have a Hamiltonian circuit.

Notice that self-embeddability still is no severe restriction, but rather a technical normalization notion: All problems can be defined in a way, s.t. they can be relativized to a unary predicate in the problem signature. More formally, if $\Pi \subseteq \text{Struct}(\tau)$ is a problem, then let the self-embedding variant Π^s be the problem obtained from including a unary predicate $D \in \tau$, s.t. for all structures $\mathcal{A} \in \text{Struct}(\tau)$ it holds that $\mathcal{A} \in \Pi$ iff $(D^{\mathcal{A}}, R_1^{\mathcal{A}} \cap (D^{\mathcal{A}})^{a_1}, \ldots, R_r^{\mathcal{A}} \cap (D^{\mathcal{A}})^{a_r}) \in \Pi$. In other words, the *active* domain elements are themselves distinguished by a unary *domain relation*. This definition arises naturally with Enderton's and Hodge's notion of logical reductions [Enderton 72, Hodges 93]. Relativizablility easily extends to the *single* encoding. Moreover, a relativizable problem is at least as hard as the original problem because it holds that $\Xi \leq_{proj}^{mon} \Xi^s$, and therefore $\Pi \leq_{proj} \Xi$ implies $\Pi \leq_{proj} \Xi^s$. In particular, if Ξ is complete for some class C, then Ξ^s is complete, too. Moreover, most problems investigated in descriptive complexity in fact are self-embeddable. Usually, it is easier to prove completeness for self-embeddable problems, because the reduction need not be surjective in that case. We, too, shall prove our results for the self-embeddable case first, and extend them afterwards.

In fact, the notion of self-embeddability is very similar to closure under padding in structural complexity theory, because self-embeddability means that we can add dummy domain elements. For the sake of simplicity, we shall usually prove the results for the self-embeddable case first, and extend them to the general case afterwards.

Now we are ready to give a formal definition of succinct encodings:

Definition Let C be a circuit, and τ be a signature. Then $gen_\tau(C) \in \text{Struct}(\tau)$ denotes the structure represented by C. If $C \in \text{Struct}(\zeta)$, but C is no circuit for syntactical reasons, let $gen_\tau(C)$ denote some fixed finite structure $S_\tau \in \text{Struct}(\tau)$ by default. □

If the context is clear, we shall skip the τ subscript. Now we can give a formal definition of succinct problem encodings:

Definition Let $\Pi \subseteq \text{Struct}(\pi)$ be a problem. The *succinct encoding* $s(\Pi)$ *of* Π is defined

$$s(\Pi) = \{C \in \text{Struct}(\zeta) : gen_\pi(C) \in \Pi\}$$

□

Remark: Note that self-embeddability is crucial here: Let $3COL \subseteq \text{Struct}(E^2)$ be the class of 3-colorable graphs, and consider $3COL^* = 3COL - \{\mathcal{A} : |\mathcal{A}| =$

$2^k, k \geq 1\}$. In other words, $3COL^*$ rejects all graphs where the number of vertices is a power of 2. Of course, $3COL^*$ is not self-embeddable because a 3-colorable graph of size $2^k - 1$ is in $3COL^*$, but its extensions by another isolated vertice is not. It immediately follows that $s(\Pi)$ cannot encode graphs from $3COL^*$, therefore $gen_E(s(\Pi)) \subseteq \{S_\tau\}$, i.e. $s(\Pi)$ becomes trivially solvable. On the other hand, each structure of a self-embeddable problem can be embedded easily into a structure of size 2^k for some k.

2.4 Models of Circuit Computation

In the field of circuit complexity, boolean circuits are often assumed to allow negation of input gates only.
For a problem Π, let $s'(\Pi) = \{C \in s(\Pi) | C$ contains input negations only$\}$. It is well-known that $s(\Pi)$ and $s'(\Pi)$ are equivalent in expressive power. Here we show the projection effectiveness of this construction, thus giving some intuition for our usage of projection reductions.

Lemma 3 *For all problems Π it holds that $s(\Pi) \equiv^{mon}_{proj} s'(\Pi)$, even under successor-free reductions.*

Proof: Trivially, $s'(\Pi) \leq^{mon}_{proj} s(\Pi)$ via the identical mapping. For the opposite direction, let $C \in \text{Struct}(\zeta)$ be a circuit. Then we construct a circuit $f(C)$ as follows: For each input gate i, we add another dual gate i^- containing the negation of i. For each conjunctive (disjunctive) gate g, we add another disjunctive (conjunctive) dual gate g^- which gets as inputs the dual input gates of g and therefore outputs the dual value. Finally, all negations are replaced by a reference to the dual value. Since we need double space the reduction has arity 2. (Note that arity 2 would allow even for quadratic space.)

In the $|C| \times |C|$ space, we use the second coordinate to distinguish between the original gates (coordinate 0) and the dual gates (coordinate max). We start by constructing the input negations:

$$\begin{aligned} \lhd(< x, 0 >) &= \lhd(x) \\ \sim(< x, max >, < x, 0 >) &= \end{aligned}$$

The $\sqcap$ and $\sqcup$ gates are constructed like described above:

$$\begin{aligned} \sqcup(< x, 0 >, < y, max >, < z, max >) &= \sqcap(x, y, z) \\ \sqcup(< x, max >, < y, 0 >, < z, 0 >) &= \sqcap(x, y, z) \\ \sqcap(< x, 0 >, < y, max >, < z, max >) &= \sqcup(x, y, z) \\ \sqcap(< x, max >, < y, 0 >, < z, 0 >) &= \sqcup(x, y, z) \end{aligned}$$

Finally, we rebuild the negations and copy the output gate:

$$\begin{aligned} \sqcup(< x, 0 >, < y, max >, < y, max >) &= \sim (x, y) \\ \sqcup(< x, max >, < y, 0 >, < y, 0 >) &= \sim (x, y) \\ \square(< x, 0 >) &= \square(x) \end{aligned}$$

Note that a gate of the form $\sqcup(x, y, y)$ serves for copying the result of gate y to gate x. This concludes the proof. □

Moreover, one can see easily that the fact that we are using conjunction, disjunction and negation to build the circuits is merely a matter of convenience. In fact, any boolean complete set of connectives would do so as well, because each connective in one circuit can be replaced by a subcircuit of constant size in the other circuit. Therefore, a monotone reduction of constant arity shows the equivalence. Suppose for example, that we restrict our circuit to NOR gates. Then we replace $\sim (x, y)$ by $NOR(x, y, y)$, $\sqcup(x, y, z)$ by $NOR(x, x', x'), NOR(x', y, z)$, and $\sqcap(x, y, z)$ by $NOR(x, y', z'), NOR(y', y, y), NOR(z', z, z)$. The space for the intermediate gates is obtained by choosing a sufficiently high arity of the reduction.

Lemma 4 *Let s_X be a circuit encoding defined by a boolean complete set X of connectives of constant arity. Then $s(\Pi) \equiv^{mon}_{proj} s_X(\Pi)$, even for successor-free reductions with constants.*

Proof: Easy. □

The following corollary expresses that projection reductions are sufficiently strong to allow switching between the circuit models:

Corollary 1 *Let s' be an alternative circuit encoding like in Lemma 3 or 4, and let C be a complexity class, closed under monotone projection reductions. If $s\Pi$ is $C-$complete under projection reductions, then $s'\Pi$ is $C-$complete under projection reductions.*

Proof: Hardness: Let $A \in C$. By assumption and by Lemmas 3 and 4 we conclude that $A \leq_{proj} s\Pi \leq^{mon}_{proj} s'\Pi$. By Lemma 1, hardness follows.
Membership: follows from $s'\Pi \leq^{mon}_{proj} s\Pi$. □

2.5 Direct Access Turing Machines and Polylogtime Reductions

Polylogtime computations like all sublinear computations are based on the notion of Direct Access Turing Machines (DTM). A DTM has three tapes: a work tape A, an input tape I and a special tape S. I does not have a conventional read head, but can be accessed indirectly via A and S: On input of length n, the first $\log n$ bits of A define a number x. At each step of the Turing Machine, the bit at position x on tape I is copied on the special tape and thus can be used by the finite control like in the usual case. Note that this definition is different to [Balcázar et al. 92], because we have neither distinguished work and address tapes nor distinguished input query states, but our DTMs and the other model mutually simulate each other with neglectable overhead.

Definition A mapping $f : \mathrm{Struct}(\pi) \to \mathrm{Struct}(\xi)$ is a many-one **PLT** reduction between $\Pi \subseteq \mathrm{Struct}(\pi)$ and $\Xi \subseteq \mathrm{Struct}(\xi)$ if for all $\mathcal{A} \in \Pi$ the domain of $f(\mathcal{A})$ is less than $2^{\log^l n}$ and there exists a polylogtime Turing machine M_f, s.t.

1. $\mathcal{A} \in \Pi$ iff $f(\mathcal{A}) \in \Xi$, and
2. for each tuple $\mathbf{x}$ and relation $R \in \xi$, M_f decides $f(\mathcal{A}) \models R(\mathbf{x})$.

$l = grow(M_f)$ is called the *growth rate* of M. □

Note that this definition is just the natural adaption of the standard notion to finite structures. A conventional problem (i.e. a class of strings) can be easily expressed over a suitable string signature. Logtime reductions reductions can be shown to subsume quantifierfree reductions, see for instance [Veith 94]. Still, quantifierfree reductions are sometimes better to use because of their simplicity.

Due to the special character of PLT reductions we may assume that the DTM which accomplishes the reduction has two accepting states, corresponding to the two possible results of a computation and that the machine loops in the accepting states. (Consequently, time bound is defined with respect to the time of reaching such a looping state.) Without loss of generality we may assume that the Turing Machine first copies the output position information on the work tape (this is possible because of its logarithmic size) and then goes on as usual, using only the information from the worktape.

Let some DTM be fixed and let w be the contents of the input tape. Then $A_w(t, i)$ denotes the bit on tape A at position i and time t on input of w. $q_w(t)$ denotes the state of the machine at step t. Let $k = \log |Q|$, where Q is the set of states of the DTM. Then each state $q_w(t)$ can be identified with a $k-$tuple of

bits. The corresponding functions are called $q_{w1}(t), \ldots, q_{wk}(t)$. The position of the TM head at time t is denoted by $a_w(t)$. The bit at the special tape at time t (i.e. the bit at the input tape which is referenced to by the address written on the first bits of the work tape) is denoted $p_w(t)$. When it is understood, we shall skip the w subscript. Let $[a(t) = i]$ denote the truth value of $a(t) = i$. Using those concepts, we can define a *glimpse* of the TM in time and space:

$$g(t,i) =< A(t,i), [a(t) = i], p(t), q_1(t), \ldots, q_k(t) >$$

Thus, a glimpse is a tuple from $\{0,1\}^{k+3}$ which describes a Turing machine locally from the point of view of a tape cell at a fixed time. A computation graph of a DTM can be seen as a 2-dimensional array of glimpes, with a time and a space axis. Each glimpse contains the information about the tape contents at position i and time t. We say that a glimpse is *active* if $a(t) = i$ holds, i.e. if the work head is situated at the tape position the glimpse describes. The TM state information provided by $q_1(t), \ldots, q_k(t)$ is relevant only for active glimpses. For better intuition, we shall sometimes imagine the time axis vertical, and the space axis horizontal. A *time level* is a horizontal slice of that array, obtained from fixing the time. The time level t evidently provides an instantenuous description of the Turing machine at time t.

A glimpse $g(t,i)$ differs from $g(t-1,i)$ only if one of its *predecessor glimpses* $g(t-1,i-1), g(t-1,i), g(t-1,i+1)$ is active. Otherwise it is equal. Since the control of the DTM is finite, $g(t,i)$ can be described by a circuit of constant size with input gates for the three predecessors and for $p(t-1)$ and with output gates for $g(t,i)$. (We neglect the case of $i,j = 0$ for the moment.)

By connecting the glimpses with copies of the finite control circuits we obtain a circuit almost simulating the Turing Machine. It still remains to simulate the direct access, i.e. to direct the input bits referenced by the worktape to the special tape bit $p(t)$. Since the input structure is provided by a boolean circuit, $p(t)$ can be computed by a copy of that circuit, s.t. its input gates are fed by the work tape bits contained in the glimpses of the previous time level. Thus, we need one copy of the input circuit between each two time levels. This proof technique dates back to [Balcázar et al. 92], for an outline refer to [Gottlob et al 95]. We shall make an effective construction in the next section.

3 The Strong Conversion Lemma

In this section we give the alluded effective construction proving a strictly stronger version of [Balcázar et al. 92]'s Conversion Lemma.

3.1 The Self-Embeddable Case

Throughout this section we shall assume that all problems are self-embeddable. In Section 3.2 we shall see how to generalize our proof.

Lemma 5 *If* $A \leq_m^{PLT} B$ *then* $sA \leq_{mon\ proj} sB$.

Proof: Let M be the DTM computing the PLT reduction $A \to B$. We assume the notation from the last section. For input size n, let $\log^m n$ be a strict upper bound for the time complexity and for $(\log n)^{grow(M)}$. (We do not need a constant factor, because we can choose m sufficiently large, though constant.)

From above we know that $g(t,i) = F(g(t-1,i-1), g(t-1,i), g(t-1,i+1), p(t-1))$ for some constant size function F depending on the finite control of the TM. Since F is of constant size, it can be described by a constant size circuit $\mathcal{F}$. $\mathcal{F}$ needs $3(k+3)+1$ input gates (corresponding to 3 predecessor glimpses and the special tape symbol) and $k+3$ output gates (corresponding to the output glimpse). Formally, $\mathcal{F} \in \text{Struct}(\zeta)$, s.t. $|\mathcal{F}| = f$, the input gates of $\mathcal{F}$ are $1, \ldots, 3(k+3)+1$ and the output gates are $f_o := f-(k+3), f_o+1, \ldots, f_o+(k+3)$. In order to prove the theorem we have to show that there exists a monotone projection reduction $\Psi : \text{Struct}(\zeta) \to \text{Struct}(\zeta)$ from sA to sB.

Let $\mathcal{A} \in \text{Struct}(\zeta)$. Recall that m was the exponent of the PLT reduction. Let $p = 2m+2$ be the arity of the projection reduction. We shall denote $p-$tuples by $< b, t, s, c >$, where $b \in N$ denotes a *block*, $c \in N$ denotes a counter, and $t, s \in N^m$ denote time and space coordinates. The blocks and counters can intuitively be identified with program blocks and local line numbers within the blocks.

Let us summarize variable usage:

$f = \lvert\mathcal{F}\rvert$	=	the size of the circuit which simulates the finite control
$k = \log \lvert Q\rvert$	=	number of bits necessary for state description
$k+3$	=	number of bits for a glimpse
$f_0 = f-(k+3)$	=	start of output part within $\mathcal{F}$
$max = \lvert\mathcal{A}\rvert$	=	size of the input structure
m	=	dimension for the computation graph
$p = 2m+2$	=	overall dimension of the reduction

Looking at the $p-$dimensional domain as an address space, we shall use the following memory allocation plan:

$$\textbf{Block 0}\left\{\begin{array}{c} <0,0,0,0> \\ \vdots \\ <0,0,max,0> \end{array}\right\} \text{ contains the input gates of the new circuit.}$$

For each s, t,

$$\textbf{Block 1}\left\{\begin{array}{c} <1,t,s,0> \\ \vdots \\ <1,t,s,f> \end{array}\right\} \text{ contains a copy of } \mathcal{F}, \text{ computing glimpse } g(t,s) \text{ from input of } p(t-1) \text{ and the glimpses } g(t-1,s-1), g(t-1,s), g(t-1,s+1).$$

$$\textbf{Block 2}\left\{\begin{array}{c} <2,t,0,0> \\ \vdots \\ <2,t,0,max> \end{array}\right\} \text{ contains a copy of } \mathcal{A}, \text{ computing the input tape information } p(t) \text{ from Block 0 and from the adress given by the glimpses } g(t-1,s), 0 \leq s \leq max.$$

$$\textbf{Block 3}\left\{\begin{array}{c} <3,0,0,0> \\ \vdots \\ <3,1,max,0> \end{array}\right\} \text{ contains a circuit to determine the output of the TM simulation.}$$

$<max, max, max, max>$ contains the output of the circuit.

Note that only block 2 depends on the input structure $\mathcal{A}$. Therefore, blocks $0, 1, 3$ will be constructed by projection reductions *without* input predicates.

Consider the following projection reduction:

Block 0

Block 0 contains the input gates:

$$\lhd(0,0,s,0) \quad =$$

Block 1

Recall that Block 1 contains the array of glimpses. The glimpses are interconnected by copies of $\mathcal{F}$, such that the output gates point to a glimpse and the input gates are wired to the corresponding predecessor glimpses and to the input structure provided by Block 2.

First we redirect the input gates of the $\mathcal{F}$ circuits:

The $p(t)$ bit comes from block 2:

$$\sqcup(<1,t,s,1>,<2,t,0,max>,<2,t,0,max>) \quad =$$

Note that gates of the form $\sqcup(x,i,i)$ serve to copy a bit from gate position i to position x.

For $s,t \neq 0$, we direct the predecessor glimpses of $g(t,s)$ to the input of an $\mathcal{F}$ circuit which computes glimpse $g(t,s)$:

Left Predecessor:

$$\begin{aligned}
\sqcup(<1,t,s,2>,<1,t-1,s-1,f_o>,<1,t-1,s-1,f_o>) &= t \neq 0, s \neq 0 \\
&\vdots \\
\sqcup(<1,t,s,k+4>,<1,t-1,s-1,f_o+k+3>,<1,t-1,s-1,f_o+k+3>) &= t \neq 0, s \neq 0
\end{aligned}$$

Middle Predecessor:

$$\begin{aligned}
\sqcup(<1,t,s,k+5>,<1,t-1,s,f_o>,<1,t-1,s,f_o>) &= t \neq 0 \\
&\vdots \\
\sqcup(<1,t,s,2k+7>,<1,t-1,s,f_o+k+3>,<1,t-1,s,f_o+k+3>) &= t \neq 0
\end{aligned}$$

Right Predecessor:

$$\begin{aligned}
\sqcup(<1,t,s,2k+8>,<1,t-1,s+1,f_o>,<1,t-1,s+1,f_o>) &= t \neq 0 \\
&\vdots \\
\sqcup(<1,t,s,3k+10>,<1,t-1,s+1,f_o+k+3>,<1,t-1,s+1,f_o+k+3>) &= t \neq 0
\end{aligned}$$

For $s=0$, we by default take $g(t-1,0)$ as the left predecessor: this cannot lead to inconsistencies, since the TM program must not crash the TM tape. Since m was defined to be a strict upper bound, the Turing machine never exceeds the other end of the tape.

Hence, we obtain similar rules for the left predecessors of the leftmost glimpses:

$$\begin{aligned}
\sqcup(<1,t,0,2>,<1,t-1,0,f_o>,<1,t-1,0,f_o>) &= t \neq 0 \\
&\vdots \\
\sqcup(<1,t,0,k+4>,<1,t-1,0,f_o+k+3>,<1,t-1,0,f_o+k+3>) &= t \neq 0
\end{aligned}$$

It remains to copy the rest of $\mathcal{F}$ to the right address: For each $i \in \{3k+11,\ldots,f\}$, if $\mathcal{F} \models \Theta(i,j,l), \Theta \in \{\sqcap,\sqcup\}$ then let

$$\Theta(<1,t,s,i>,<1,t,s,j>,<1,t,s,l>) \quad =$$

We proceed analogously with $\sim$. (Recall again that i, j, k are constants, because the finite control is constant.) We still have to initialize the machine simulation for $t = 0$: First, we define truth constants in the circuit:

$$\begin{aligned} \sim(<4,0,0,0>,<0,0,0,0>) &= \\ \sqcup(<4,0,0,1>,<4,0,0,0>,<0,0,0,0>) &= \\ \sim(<4,0,0,2>,<4,0,0,1>) &= \end{aligned}$$

This forces $<4,0,0,1>$ to compute 1 and $<4,0,0,2>$ to compute 0.

Now we can set the glimpses $g(0,s)$ corresponding to the Turing machine at time 0:

$$\begin{aligned} \sqcup(<1,0,s,x>,<4,0,0,2>,<4,0,0,2>) &= s \neq 0 \\ \sqcup(<1,0,0,x>,<4,0,0,2>,<4,0,0,2>) &= x \neq f_o+1, x \neq f_0 \\ \sqcup(<1,0,0,f_o+1>,<4,0,0,1>,<4,0,0,1>) &= \\ \sqcup(<1,0,s,f_0>,<0,0,s,0>,<0,0,s,0>) &= \end{aligned}$$

Thus we set the leftmost glimpse active, have the queried string position within the output string immediately copied from the input tape to the worktape, and set everything else 0, assuming that the starting state has binary coding 0^k.

Block 2 basically consists of copies of $\mathcal{A}$, again with the input redirected:

$$\begin{aligned} \sqcup(<2,t,0,s>,<1,t-1,f_o,s>,<1,t-1,f_o,s>) &= \lhd(s) \\ \sqcup(<2,t,0,s>,<2,t,0,i>,<2,t,0,j>) &= \sqcup(s,i,j) \\ \sqcap(<2,t,0,s>,<2,t,0,i>,<2,t,0,j>) &= \sqcap(s,i,j) \\ \sim(<2,t,0,s>,<2,t,0,i>) &= \sim(s,i) \end{aligned}$$

For **Block 3** it remains to compute the output of the circuit: For this purpose, we have to find the active glimpse of the last row and determine if the state therein is accepting and which kind of accepting state it is. (Recall that we assumed that the acception state gives the information about the output bit.) Without loss of generality, assume that in the accepting states, $q_1(t)$ expresses the result bit. In order to find what this bit looks like, we have to search among all glimpses with maximal time coordinate for a glimpse which is both active and describes an accepting state. Essentially, this can be done by a large disjunction:

$$\begin{aligned} \sqcap(<3,0,s,0>,<1,max,s,f_o+2>,<1,max,s,f_o+3>) &= \\ \sqcup(<3,1,0,0>,<3,0,0,0>,<3,0,0,0>) &= \\ \sqcup(<3,1,s,0>,<3,1,s',0>,<3,0,s,0>) &= s = s'+1 \end{aligned}$$

It remains to redirect the result bit to the output gate $<max,max,max,max>$:

$$\begin{aligned} \sqcup(<max,max,max,max>,<3,1,max,0>,<3,1,max,0>) &= \\ \square(<max,max,max,max>) &= \end{aligned}$$

Since $|\mathcal{A}| = max > \log n$ by the definition of succinct representations (compare [Balcázar et al. 92]'s proof) the bound max^m is sufficiently large for both the polylogarithmic space and time bound. This concludes the proof. □

In order to develop our methods further, we need a projection definable analogue to the operator *long* from [Balcázar et al. 92] in terms of finite structures. Recall their definition:

Definition Let $A \subseteq \{0,1\}^*$. Then we define $long(A) = \{w : \overline{|w|} \in 1 \circ A\}$. □

That is, we interpret every instance of A as a binary number x and have *all* strings of size x included in $long(A)$.

In the descriptive complexity approach, the instances are structures and for the above concept we would need a structure encoding. Note that the essential property of the *long* operator is that it takes a characteristic bit sequence of the instance (in the above case, the instance itself serves as its characteristic bit sequence) and uses it as a length description. Given a structure $single(\mathcal{A}) = (n, Q^{l+1})$, consider the tuples $(x_1, \ldots, x_{l+1}) \in n^{l+1}$ in lexicographical order with respect to the successor relation. With respect to membership in Q, the tuples define a sequence of $0, 1$ which is characteristic of $\mathcal{A}$; being read as a binary representation, the sequence in turn defines a large natural number. Let us call this number $char(\mathcal{A})$. This motivates the following definition:

Definition Let $\alpha = (Q^{l+1})$ be a signature, and let $single(\mathcal{A}) \in \mathrm{Struct}(\alpha)$. Let $\delta = (D^1)$ be a signature, where $D(x)$ typically holds for an initial sequence of natural numbers. Then we define

$$long : \mathrm{Struct}(\alpha) \to \mathrm{Struct}\,(\delta)$$

by

$$long(\mathcal{A}) = \{\mathcal{B} \in \mathrm{Struct}(\delta) \mid D^{\mathcal{B}} = 2^{|\mathcal{A}|^{l+2}} + char(\mathcal{A})\}$$

□

Like in [Balcázar et al. 92]'s definition, the leading term forces the most significant bit of its binary representation to be 1. Recall that a number corresponds to an initial sequence of the natural numbers, and thus to a unary relation.

Lemma 6 *For every* Π, $\Pi \leq_{\mathrm{proj}} s(long(\Pi))$.

Proof: Let $\Xi = single(\Pi)$, then it remains to show $\Xi \leq_{\mathrm{proj}} s(long(\Pi))$. That is: given $\mathcal{A} \in \Xi$, we have to construct a circuit $\mathcal{G}$, s.t. $\mathcal{G}$ describes a string

of size $2^{|\mathcal{A}|^{l+2}} + char(\mathcal{A})$. We shall construct a circuit, which describes a string $1^{2^{|\mathcal{A}|^{l+2}}+char(\mathcal{A})}$, i.e. a unary relation containing an initial sequence of the natural numbers.

Again, we construct a 2-dimensional array. On the first horizontal line, we project the truth values $Q(0,\ldots,0),\ldots,Q(max,\ldots,max)$ and the leading 1: Depending on the truth value of $Q(x_1,\ldots,x_l)$, we connect the corresponding gate to either the constant 0 or the constant 1 function as in **Block 4** above. Therefore, our projection will be of arity $l+1$. On the second line we situate the input gates.

Given an input, the circuit has to compare the values denoted by the two horizontal lines. This can be done easily be looking for the most significant bit where the lines differ. At this position, the larger number must have a 1. If the first line is greater, then the input position is within the string size, and the circuit outputs 1. Otherwise, the string length is exceeded, and the circuit outputs 0. Such a construction can easily be accomplished similar to **Block 3** above.

After the proof of Lemma 5, the technical details of the construction can be imagined. □

Remark: Notice that monotone projections do not seem to be sufficient for the construction in Lemma 6 : The crucial point is that the reduction is required to project the values of Q into the circuit. However, this construction naturally involves a case distinction: If Q holds, then let the gate compute the permanent 1 function, *otherwise* the permanent 0 function. A monotone rule contains only rules without such an 'otherwise' condition, hence it is virtually not sufficient to gain the wished effect. For a more concise statement of this fact, refer to Proposition 1.

Along the lines of their proof, there immediately follows a strictly stronger form of the main upgrading theorem by [Balcázar et al. 92]:

Theorem 1 Upgrading Theorem
Let C_1 and C_2 be complexity classes, s.t. $long(C_1) \subseteq C_2$. If B is C_2-hard under **PLT***-reductions then sB is C_1-hard under projection reductions.*

Proof: Let $A \in C_1$ be arbitary. By assumption $long(A) \in C_2$, therefore $long(A) \leq_m^{\mathrm{PLT}} B$. By the Strong Conversion Lemma and Lemma 6, we obtain that $A \leq_{\mathrm{proj}} s(long(A)) \leq_{\mathrm{mon\ proj}} sB$, thus $A \leq_{\mathrm{proj}} sB$ because of Lemma 1. □

3.2 Eliminating Self-Embeddability

Let us consider how to extend our above results to problems which are not self-embeddable. For this purpose, we need a slightly more complicated model to encode structures by circuits: one of two circuits describes the relations as above, while the other describes the domain. A similar approach was used already in the definition of the *long* operator.

Take some signature π and consider the set $\mathrm{Struct}_2(\pi) = \{\mathcal{A} \in \mathrm{Struct}(\pi) : \exists k : |\mathcal{A}| = 2^k\}$, i.e. the subclass of $\mathrm{Struct}(\pi)$ which is describable by boolean circuits. Further consider the signature $\delta = (D^1)$, and the problem $\Delta = \{(n, D) : D \leq n\} \subseteq \mathrm{Struct}(\delta)$. In other words, Δ contains initial sequences of the natural numbers. Then it is easy to see that for each $\mathcal{A} \in \mathrm{Struct}(\pi)$ there exist structures $\mathcal{A}' \in \mathrm{Struct}_2(\pi)$ and $\mathcal{D} \in \Delta$, s.t. $\mathcal{A}'$ restricted to the elements from $D^{\mathcal{D}}$ equals $\mathcal{A}$, in symbols $\mathcal{A}'|\mathcal{D} = \mathcal{A}$. Thus, each π structure can be described by a π structure from $\mathrm{Struct}_2(\pi)$ and a δ structure. For a pair $(C_\pi, C_\delta) \in \mathrm{Struct}(\zeta) \times \mathrm{Struct}(\zeta)$, we can define $gen_\pi(C_\pi, C_\delta) := gen_\pi(C_\pi)|gen_\delta(C_\delta)$. Consequently, $s(\Pi) = \{(C_\pi, C_\delta) : gen_\pi(C_\pi, C_\delta) \in \Pi\}$. We shall refer to this encoding as the *compact* succinct encoding. This definition enables the circuits to describe structures of arbitrary size. Essentially, we could as well use circuits with two output gates, but this would make the constructions more complicated.

To make sure that $gen(C_\delta)$ is always a Δ structure, we postulate that C_δ is subject to a certain *syntactical* restriction: C_δ consists of a typically large subcircuit C_1, which computes a tuple without using the input gates. (We shall call such a circuit the *core circuit* of C_δ.) A lexicographical comparison circuit like in Lemma 6 then determines the output of C_δ by comparing the tuple with the input gates. By a construction like in Lemma 6 we see that $C_1 \equiv^{mon}_{proj} C_\delta$, therefore constructing C_δ reduces to constructing C_1.

Let us return to Lemma 5, the Conversion Lemma. Given a PLT reduction $M : A \to B$, we have to construct a reduction $\mathrm{Struct}(\zeta) \times \mathrm{Struct}(\zeta) \to \mathrm{Struct}(\zeta) \times \mathrm{Struct}(\zeta)$ between their succinct representations. There are two steps to adopt the above proof:

First, we have to make a modification in the PLT machine simulation to handle the information provided by the additional circuit. In each step, the special tape contains two bits: the first one describes the relation like above, and the second one denotes whether the queried input tuple exceeds the domain of the structure. (This information is naturally provided by the second circuit C_δ.) Thus, the glimpses in the Turing machine description will contain another bit to carry this information.

Second, we can assume that there exists a PLT transducer M', s.t. $M' : A \to$

ω where $M'(A)$ computes the length of $M(A)$. (M' can be easily constructed from M by using binary search.) M' can be considered as a degenerated special case of a PLT reduction which behaves independently of the output position in question. Hence, applying the self-embedding Conversion Lemma to M' on input A yields a circuit which contains the tuple $M'(A)$ on the last time level, and is not dependent on the input gates. Therefore, this circuit can be used as a *core circuit*, and can be easily embedded into a C_δ circuit by a monotone projection as described above.

Thus, by applying the self-embedding Conversion Lemma for both Turing machines M, M' it follows that there exist projection reductions Φ_M, $\Phi_{M'}$, which map each circuit pair $C = (C_\pi, C_\delta)$ to a pair $(\Phi_M(C), \Phi_{M'}(C))$. The Conversion Lemma follows.

For Lemma 6, things are much easier because the circuit constructed there already is the length description we need. Hence, we just use the same reduction for the self-embeddable case.

Therefore, the upgrading theorem holds for arbitrary problems.

4 Leaf Languages

In this section, we consider the relation between succinct complexity upgrade and leaf languages, as defined in [Bovet et al 92, Bovet et al 91]. In Section 4.1, we shall treat the easier self-embeddable case.

4.1 The Self-Embeddable Case

Let M be a fixed **NP**-Turing Machine which outputs 1 on acception and 0 on rejection. On input of a word $x, |x| = n$, one can imagine that M computes simultaneously in many branches, corresponding to its nondeterministic choices. Assume that the machine is clocked and halts after exactly n^k steps and after exactly $n^l, l \leq k$ nondeterminstic decisions. (This latter condition follows naturally when we program the Turing machine by a 'guess and check' algorithm.) For a given input of size n, one can evaluate the Turing machine for all 2^{n^l} possible nondeterministic choice sequences in lexicographical order. By concatenating 0's and 1's according to rejection or acceptance we obtain a word $w \in \{0,1\}^{2^{n^l}}$. We denote this *leafstring* w by $l^M(x)$. Let Y be a language. Then we define

$$\mathrm{BLeaf}^P(Y) = \left\{ L |\ \exists M : L = \{x | l^M(x) \in Y\} \right\}$$

i.e. the class of languages whose characteristic function carries over to Y via its leafstring for some suitable Turing Machine.[1] For self-embeddable Π, $\mathrm{BLeaf}^P(\Pi)$ coincides with the notion used by [Hertrampf et al 93, Jenner et al 94].

It immediately follows that $X \subseteq Y$ implies $\mathrm{BLeaf}^P(X) \subseteq \mathrm{BLeaf}^P(Y)$. The definition generalizes to complexity classes C:

$$\mathrm{BLeaf}^P(C) = \bigcup_{Y \in C} \mathrm{BLeaf}^P(Y)$$

Again a similar monotonicity property follows from the definition.

It was shown in [Bovet et al 92, Hertrampf et al 93] that by varying the parameter in $\mathrm{BLeaf}^P(\Pi)$ or $\mathrm{BLeaf}^P(C)$, it is possible to obtain characterizations of many complexity classes. For instance, all classes in the polynomial hierarchy are definable by suitable leaf patterns [Bovet et al 92]. Moreover, it follows from [Hertrampf et al 93] that $\mathrm{BLeaf}^P(long(C)) = C$, and even $\mathrm{BLeaf}^P(REG) = PSPACE$, where REG is the class of regular languages.

Our main observation is that for any problem Π, its succinct representation is just as complex as its leaf language.

Lemma 7 *Let Π be a problem. Then $s(\Pi) \in \mathrm{BLeaf}^P(\Pi)$.*

Proof: The succinct version of Π is defined as $s(\Pi) = \{C | gen_\pi(C) \in \Pi\}$. For a circuit C, $gen_\pi(C)$ can be computed as a leafstring of the following Turing machine:

1. `Guess an assignment to the input gates of` C.
2. `Evaluate the circuit.`

It is well-known that step 2 is polynomial time computable. The result follows. □

This can be extended to completeness. To outline the proof idea, we first prove completeness under logspace reductions, and go into detail later on.

Lemma 8 *$s(\Pi)$ is complete for $\mathrm{BLeaf}^P(\Pi)$ under logspace reductions.*

[1] In the sense of [Jenner et al 94], we are using a restricted form of *balanced* Turing machines, therefore we write BLeaf. Compare Section 4.2.

Proof: It remains to show hardness. Let $L \in \mathrm{BLeaf}^P(\Pi)$. Then there exists some **NP** machine M, s.t. $L = \{x | l^M(x) \in \Pi\}$. Without loss of generality we may assume that M is a 'guess and check' machine with 4 tapes: the input tape, a tape for the guessed string, a work tape and an output tape. Consider the machine M'_x obtained by fixing the input x and providing the nondeterministic choices via the input tape. Let $|x|^l$ be an upper bound for the number of nondeterministic choices on the input tape. M'_x can be easily obtained from M by interchanging the access to the first and second tape. By well-known simulation techniques, machine M'_x is equivalent to a circuit $c(M'_x)$ of size $|x|^{O(1)}$ with $|x|^l$ input gates. $c(M'_x)$ can be easily constructed in logspace. It follows that $x \in L$ iff $c(M'_x) \in s(\Pi)$ which concludes the proof. □

In order to simplify the projection reduction proof to follow, we introduce an *ad hoc* variant of 'guess and check' **NP**-Turing Machines called *nice* machines. A nice machine by definition supports the construction of a machine like M'_x above. Let n^l be an upper bound on the number of nondeterministic choices of the machine. (Clearly, n^l is not greater than the time bound.) Then the TM proceeds as follows:

1. `If the work tape is not empty goto 5`
2. `Copy the input and` $n^l - n$ `blanks on the work tape.`
3. `Write` n^l `nondeterministic bits on the worktape.`
4. `restart without erasing the worktape`
5. `Proceed like an ordinary` **NP** `machine, with the input provided by the left half of the worktape and the nondeterministic choices provided by the right half of the worktape.`

Thus, if we provide both the input string and the nondeterministic choices on the worktape of a nice Turing Machine, then it turns into a polynomial time Turing Machine whose work tape serves as input tape, too. Otherwise, i.e. if the worktape is empty, the machine proceeds as usual. We see that a nice machine supports interchanging the input and the guess string like in the proof of Lemma 8.

Theorem 2 *$s(\Pi)$ is complete for $\mathrm{BLeaf}^P(\Pi)$ under projection reductions.*

Proof: It remains to show hardness. Let $L \in \mathrm{BLeaf}^P(\Pi)$. Then there exists some *nice* Turing machine M, s.t. $L = \{\mathcal{A} | l^M(\mathcal{A}) \in \Pi\}$. Without loss of generality we

may assume that $L \subseteq \mathrm{Struct}(\alpha)$. We have to find a reduction Ψ, s.t. $\mathcal{A} \in L$ iff $\Psi(\mathcal{A}) \in s(\Pi)$.

Consider machine M with time bound n^k and let the input stucture $\mathcal{A}$ be fixed. Let $M_{\mathcal{A}}(g)$ denote the output of the machine M, when it starts with $\mathcal{A}$, $|\mathcal{A}|^l - |\mathcal{A}|$ blanks and a guess string g of length $|\mathcal{A}|^l$ on its worktape. It is easy to see that $M_{\mathcal{A}}(g)$ outputs the bit at position $val(g)$ of $l^M(\mathcal{A})$. It follows from Theorem 5.1 in [Balcázar et al. 88] that there exists a circuit $c(M_{\mathcal{A}})$ of size $|\mathcal{A}|^{O(1)}$ which simulates $M_{\mathcal{A}}$. Therefore $gen(c(M_{\mathcal{A}})) = l^M(\mathcal{A})$ and hence $\mathcal{A} \in L$ iff $l^M(\mathcal{A}) \in \Pi$ iff $gen(c(M_{\mathcal{A}})) \in \Pi$ iff $c(M_{\mathcal{A}}) \in s(\Pi)$.

We still have to show that $c(M_{\mathcal{A}})$ can be effectively constructed from $\mathcal{A}$ by projection reductions. This construction is easier than in the Conversion Lemma, and we shall not do it in such detail again.

Like in the proof of the conversion lemma, we construct an array of glimpses. By choosing the arity sufficiently high, we can make sure that the array is large enough to simulate the computation. Unlike in the proof of the Conversion Lemma, the glimpses do not need a $p(t)$ bit any more, because we do not have direct access. However, we use that bit to distinguish the rightmost unused work tape cell. Since M is nice, all time levels of the array of glimpses for $M_{\mathcal{A}}$ except the first one are independent of $\mathcal{A}$. Therefore, they can be easily constructed by projection reductions. As for the first time level, we locate the input gates of the circuit and an appropriate number of blanks on the left side, followed by the projection of the contents of $\mathcal{A}$, and by the tape end bit. Since we want the machine to operate with the nondeterminstic bits provided from the worktape, we set the initial state of the machine equal to `5`. (See the definition of nice machines above.) This concludes the proof. □

Remark: Note that again the only operation which is not expressible by *monotone* projection reductions is projecting the truth values of $\mathcal{A}$ into the first row of the circuit.

Corollary 2 $[s(\Pi)]_{proj} = \mathrm{BLeaf}^P(\Pi)$

4.2 Eliminating Self-Embeddability by Compact Encoding

The approach of the last Section works only for self-embeddable problems for similar reasons as in Section 3.1: Consider again the problem $3COL^*$ from Section 2.3 obtained by removing all instances of exponential size. Since leafstrings are

always of exponential size, it follows that $\mathrm{BLeaf}^P(3COL^*) \subseteq \mathbf{P}$ while the original definition yields **NEXP**.

In [Jenner et al 94], the concept of a *balanced* nondeterministic Turing Machine is introduced. A balanced machine is a nondeterministic Turing machine where all computation branches have equal depth n^l like above, but with the difference that the machine need not go into all 2^{n^l} nondeterministic branches. However, if it goes into a branch, it has also has to go to all lexicographically smaller branches.[2] Thus, a leafstring may have size smaller than 2^{n^l}. It was shown in [Jenner et al 94] that this model coincides with [Hertrampf et al 93] and, for complexity classes closed under padding, it also coincides with [Bovet et al 92].

Note that the bits of the binary notation of a position within the leaftstring are equal to the sequence of nondeterministic decisions in the corresponding computation branch. Consider a balanced Turing machine M. Then there exists a polynomial time Turing machine N which on input of a word w computes the length of the leafstring $l^M(w)$, i.e. the lexicographically greatest branch of M on input w: To see this, note that for a given branch, it can be checked in polynomial time if M on input w can go along this branch. Thus, N can find the largest branch by binary search.

Therefore, we can assume without loss of generality that a balanced **NP** Turing machine is described by a nice guess and check **NP** Turing machine, and an additional **P** Turing machine which computes the length of the leafstring.

Thus, the leaf string definition for the general case becomes slightly modified: For two machines M, N, let $l^{M,N}(x)$ denote the leaf string obtained from cutting off $l^M(x)$ at position $N(x)$. Then we can define

$$\mathrm{BLeaf}^P(Y) = \left\{ L |\ \exists M, N : L = \{x | l^{M,N}(x) \in Y\} \right\}$$

almost like above.

This new definition certainly matches the compact succinct encoding from Section 3.2: In fact, Lemma 7, i.e. the membership proof, holds for this encoding: Given two circuits C_π, C_δ, we use the same **NP** machine as in the proof of Lemma 7 to obtain a leafstring equivalent to $gen_\pi(C)$. The polynomial time machine has to restrict the length of this leaftstring according to the domain size described by C_δ. This can be done by binary search, because C_δ evaluates to 1 for 'small' assignments to the input gates, and to 0 otherwise. (Recall that C_δ is syntactically restricted, compare Section 3.2.)

Let us turn to Theorem 2. The idea of the proof is the following: given a

[2]Note that a branch A is said to be lexicographically smaller than a branch B if its position in the leafstring is smaller than the position of B.

language $L \in \text{BLeaf}(\Pi)$, consider the machines M, N, s.t. $L = \{\mathcal{A} | l^{M,N}(\mathcal{A}) \in \Pi\}$, and encode them as circuits C_π, C_δ. As for the **NP** Turing machine M, the construction of C_π from $M'_\mathcal{A}$ has been described in the proof of Theorem 2 above. Since N is just a normal polynomial time Turing machine, it can be easily simulated by the *array of glimpses* technique as a boolean circuit. At the last time level of this simulation circuit we by assumption find the length of the leafstring of M in binary notation.

Thus, we construct C_δ in the following way: The *core circuit* C_1 of C_δ is the simulation array for N, where the first time level has $\mathcal{A}$ as its fixed input. Like in Section 3.2. we embed C_1 into the lexicographical comparison circuit C_δ by a monotone reduction. Therefore, C_δ decides if a given position lies within the leafstring.

4.3 An Alternative Proof of the Upgrading Theorem

In this section, we shall give an alternative proof of the Upgrading Theorem from Theorem 2, while the Strong Conversion Lemma 5 cannot be derived in this way. Note that all results of this section hold under normal succinct encoding for self-embeddable problems and under the compact succinct encoding for arbitrary problems.

Corollary 3 (Weaker Conversion Lemma) *If* $A \leq_m^{PLT} B$ *then* $sA \leq_{proj} sB$.

Proof: $A \leq_m^{PLT} B$ implies $\text{BLeaf}^P(A) \subseteq \text{BLeaf}^P(B)$ by Lemma 2.13 in [Bovet et al 92]. By Theorem 2, $[sA]_{proj} \subseteq [sB]_{proj}$ and $sA \in [sB]_{proj}$. We conclude that $sA \leq_{proj} sB$. □

Since projections are not closed under iteration, this Conversion Lemma is not strong enough to prove the Upgrading Theorem.

In fact, we can prove our intuitive notion that $s(\Pi)$ is not complete for $\text{BLeaf}^P(\Pi)$ under monotone reductions:

Proposition 1 $[s(\Pi)]^{mon}_{proj} \subset [s(\Pi)]_{proj}$

Proof: Let Ξ be a problem, s.t. $\Xi \leq^{mon}_{proj} s(\Pi)$ via a monotone reduction Φ. W.l.o.g. assume that Ξ is a graph problem and (n, E) is a graph. Then $(n, E) \in \Xi$ iff $\Phi(n, E) \in s(\Pi)$. Let (n, F) be an arbitary other graph and consider $(n, E \cup F)$.

Since Φ is monotone, $\Phi(n, E \cup F)$ is a circuit obtained from $\Phi(n, E)$ by adding some more gates on formerly undefined positions. Those gates however are irrelevant, because there is no way to change the meaning of a circuit without removing any gates. Therefore, $gen(\Phi(n, E)) = gen(\Phi(n, E \cup F))$ and consequently $(n, E) \in \Xi$ iff $(n, E \cup F) \in \Xi$. Since F was arbitrary, we conclude that Ξ must be monotone, as must be all problems, which are reducible to $s(\Pi)$ by monotone reductions. Since $\text{BLeaf}^P(\Pi) = [s(\Pi)]_{proj} \supseteq \mathbf{P}$ contains not only monotone problems, the result follows. □

Lemma 9 proves our intuition that leaf languages are a more concise measure than succinct representations.

Lemma 9 *If $long(C) \subseteq D$, then $C \subseteq \text{BLeaf}^P(D)$. Moreover, the converse implication does not hold.*

Proof: Let $L \in C$, and consider $long(L)$. By assumption, $long(L) \in D$. We show that L can be recognized by a leaf language with leaf pattern $long(L)$: On input w, the **NP** machine just has to scan w and guess a bit for each bit of w. If the guessed bits are lexicographically smaller than w, the machine accepts, otherwise it rejects. Thus, like in Lemma 6, the leaf string is a word from $\{1\}^*$, which lies in $long(L)$ iff $w \in L$.

As for the converse implication, consider a counterexample based on the results from [Hertrampf et al 93]: Let $D = REG$ be the regular languages, and let C be PSPACE. Then $long(PSPACE) = \mathbf{L} \not\subseteq REG$, but on the other hand $\text{BLeaf}^P(REG) =$PSPACE by [Hertrampf et al 93]. □

The lemma can be used to give an alternative proof of the upgrading theorem:

Corollary 4 The Upgrading Theorem, II
Let C and D be complexity classes, s.t. $long(C) \subseteq D$. If A is D-hard under PLT-reductions then sA is C-hard under projection reductions.

Proof: Suppose that $long(C) \subseteq D$, and let A be $D-$hard under PLT reductions. By Lemma 9 it follows that $C \subseteq \text{BLeaf}^P(D)$. Let $L \in D$, then $L \leq^{PLT} A$, and therefore, we conclude with Lemma 2.13, [Bovet et al 92] that $\text{BLeaf}^P(L) \subseteq \text{BLeaf}^P(A)$, and that $\text{BLeaf}^P(D) = \bigcup_{L \in D} \text{BLeaf}^P(L) \subseteq \text{BLeaf}^P(A)$. Therefore, $C \subseteq \text{BLeaf}^P(D) \subseteq \text{BLeaf}^P(A)$. Hence, $s(A)$ is C-hard under projection reductions. □

5 Completeness Results

5.1 Problems Involving Circuits

In this section, we deal exclusively with self-embeddable problems. Therefore, we can apply the encoding by a single circuit.

The problem CIRCUIT SAT is the decision problem if there exists an assignment to the input gates of a given circuit, s.t. the circuit outputs 1. CIRCUIT SAT is well-known to be **NP** complete under logspace reductions. We show that completeness even holds for projection reductions:

Corollary 5 *CIRCUIT SAT is* **NP** *complete under projection reductions.*

Proof: Let $\tau = (T^1)$, $L_{\mathrm{NP}} = \{\mathcal{A} \in \mathrm{Struct}(\tau) : |T^{\mathcal{A}}| \neq 0\}$. $s(L_{\mathrm{NP}})$ is a natural encoding of CIRCUIT SAT: A circuit C is a true instance of $s(L_{\mathrm{NP}})$ iff $T^{gen(C)}$ is not empty iff there exists an assignment to the input gates of C, s.t. C evaluates to 1. By Theorem 2, we conclude that $s(L_{\mathrm{NP}})$ is complete for $\mathrm{BLeaf}^P(L_{\mathrm{NP}})$ under projection reductions.

On the other hand, $\mathrm{BLeaf}^P(L_{\mathrm{NP}}) = \mathbf{NP}$, since as a leaf pattern, L_{NP} says that a string is accepted iff there exists at least one accepting computation path (see [Hertrampf et al 93]). This concludes the proof. □

Remark: Note that the encoding of L_{NP} as a binary string coincides with the language OR in [Jenner et al 94].

This technique applies to yet another central problem in complexity theory: Given a circuit C and an assignment to its input gates, CIRCUIT VALUE is the problem of computing the output value of the circuit.

Corollary 6 *CIRCUIT VALUE is* **P** *complete under projection reductions.*

Proof: Let $\tau = (T^1)$, $L_P = \{\mathcal{A} \in \mathrm{Struct}(\tau) : \mathcal{A} \models T(0)\}$. A circuit C is in $s(L_P)$ iff C outputs 1 upon setting all input gates to 0. $s(L_P)$ is a reasonable encoding of CIRCUIT VALUE, because other input values can be simulated by adding negation gates.

$\mathrm{BLeaf}^P(L_P) = \mathbf{P}$ because the pattern says that a string is accepted iff it is accepted by the **NP** machine in its leftmost branch where it guesses only 0s. However, such an **NP** Turing machine is a **P** Turing machine. □

Remark: By Lemma 4, it follows that even CIRCUIT VALUE with negation restricted to input gates is **P** complete under projection reductions. Essentially, this can be seen as an encoding of MONOTONE CIRCUIT VALUE, see [Papadimitriou 94].

Thus, Upgrading Theorem 1 allows us to sharpen Theorem 20.2 and its Corollary 2 from [Papadimitriou 94]:

Corollary 7 *SUCCINCT CIRCUIT SAT is* **NEXP** *complete under projection reductions.*
SUCCINCT CIRCUIT VALUE is **EXP** *complete under projection reductions.*

Of course, this upgrading can be iterated, see Section 6.

5.2 Completeness under Monotone Projections

In this section, we show the announced completeness results under monotone projection reductions. All results of this section hold for both compact and normal succinct encoding.

Let us review why the results of Sections 3 and 4 did not work out for completeness under *monotone* projection reductions:

1. In the proofs of Lemma 6 and Theorem 2 we faced the problem that a monotone reduction cannot map a relation into a boolean circuit in such a way that each tuple is mapped to a gate which evaluates to 1 (0) if the tuple is contained (not contained resp.) in the relation. The reason is that monotone reductions can use only the positive facts (i.e. the tuples which *are* contained in the relation) to construct the circuit, because checking if a tuple is not contained in the relation certainly involves a negation.

2. In Proposition 1 we have proved that completeness under monotone reductions indeed is impossible. The major argument in the proof of Proposition 1 was the fact that a circuit once constructed cannot be changed by simply adding new gates.

We can overcome those difficulties by introducing *implicit* circuits. Recall that in Section 2 we have postulated that all gates are computed from other gates. Implicit circuits, however, have a somewhat looser syntax: we allow (and in fact encourage) circuits with gates pointing to positions where no other gate is

explicitely situated. Those empty positions are assumed to compute the constant zero function.

We immediately observe that the proof idea of Proposition 1 fails, because empty positions can be changed into gates later on. In fact, it is now possible to project a relation into a circuit in the way described above. Without loss of generality, let R be a unary relation. Then we define the following arity 2 monotone mapping into a circuit:

$$\begin{aligned} \sim(<2,1>,<1,1>) &= \\ \sqcup(<3,1>,<2,1>,<1,1>) &= \\ \sqcup(<x,0>,<3,1>,<3,1>) &= R(x) \end{aligned}$$

$<3,1>$ computes the constant 1 function. It is easy to see that the gates $<x,0>$ evaluate to the truth value of $R(x)$. Thus, we can easily adapt the above proofs, such that both Lemma 6 and Theorem 2 hold for monotone projections, therefore both Conversion Lemmas coincide and we obtain a strictly stronger version of Theorem 1 (the Upgrading Theorem) with the projection reductions replaced by monotone projection reductions. Instead of $s(\Pi)$ we write $i(\Pi)$ to denote the implicit succinct version of Π.

Corollary 8 $[i(\Pi)]^{mon}_{proj} = \mathrm{BLeaf}^P(\Pi)$

Thus, we obtain stronger completeness results at the cost of losing the structural distinction between monotone projections and other projections. While $s(\Pi) \equiv^{mon}_{proj} s(\Xi)$ means that Π and Ξ are *closely* related, $i(\Pi) \equiv^{mon}_{proj} i(\Xi)$ follows already from $\Pi \equiv_{PLT} \Xi$. Thus, the $\equiv^{mon}_{proj}$ clusters of the s encoding are finer.

From Corollary 8, we can easily obtain concrete problems complete under monotone projection reductions. In fact, it was stated by Stewart in [Stewart 93a, Stewart 94] that no problems are known to be complete under monotone projections for natural TM-defined complexity classes. (For monotone **NP**, Stewart has proven completeness under monotone reductions in [Stewart 94].)

Corollary 9 *IMPLICIT CIRCUIT SAT is* **NP** *complete under monotone projection reductions.*
IMPLICIT CIRCUIT VALUE is **P** *complete under monotone projection reductions.*

While IMPLICIT CIRCUIT SAT appears to be unnatural, IMPLICIT CIRCUIT VALUE is a natural encoding of CIRCUIT VALUE where all gates without explicit description serve as input gates with value 0.

In a similar way, using the leaf string characterizations of [Bovet et al 92], one can show the following results:

Corollary 10 *The implicit version of Σ_k QUANTIFIED CIRCUIT SAT is Σ_k^P complete under monotone projection reductions. Its succinct version is* $\mathbf{NEXP}^{\Sigma_{k-1}^P}$ *complete under monotone projection reductions.*

In [Stewart 91] it has been shown that HP is not complete for **NP** under monotone reductions unless **NP**=co-**NP**. It follows that $i(L_{\mathrm{NP}}) \not\leq_{proj}^{mon} HP$ unless **NP**=co-**NP**. In fact, one can show that $i(L_{\mathrm{NP}}) \not\leq_{proj}^{mon} HP$, using the following observation:

Proposition 2 *Let $\Pi \subseteq \mathrm{Struct}(\pi)$ be a nonmonotone problem, and let Ξ be a monotone problem. Then $\Pi \not\leq_{proj}^{mon} \Xi$.*

Proof: Let $\mathcal{M}, \mathcal{M}' \in \mathrm{Struct}(\pi)$, s.t. $\mathcal{M} \in \Pi$, $\mathcal{M}' \notin \Pi$, and $\mathcal{M} \subseteq \mathcal{M}'$. Such structures must exist, because Π is nonmonotone. Let Φ be a monotone reduction from Π to Ξ. Then $\Phi(\mathcal{M}) \in \Xi$ and $\Phi(\mathcal{M}') \notin \Xi$. On the other hand, the monotonicity of Φ implies that $\Phi(\mathcal{M}) \subseteq \Phi(\mathcal{M}')$, whence the montonicity of Ξ implies that $\Phi(\mathcal{M}') \in \Xi$, contradiction. □

Thus, we can sharpen Corollary 4.3 from [Stewart 91]:

Corollary 11 *HP is not* **NP***-complete under monotone projection reductions.*

Proof: HP is monotone, hence the result follows from Proposition 2. □

In particular, we obtain

Corollary 12 *$\Pi \leq_{proj}^{mon} i(L_{\mathrm{NP}}) \not\leq_{proj}^{mon} \Pi$, for all monotone Π which are* **NP***-complete by projection reductions.*

To the end of this section, let us demonstrate how to obtain completeness results for less common complexity classes: The class parity P, i.e. $\oplus P$, consists of those languages which are accepted by an **NP** machine if it has an odd number of accepting computation paths. It is immediate to find a complete problem for $\oplus$ using our approach: Let $L_{\oplus P} = \{(n, T^1) \text{ s.t. } |\{x : T(x)\}| \text{ is odd.}\}$. Then $i(L_{\oplus P})$ is complete for $\oplus P$ under monotone projection reductions. Similary,

the class PP has a complete problem $i(L_{PP})$, where $L_{PP} = \{(n, T^1) : |\{x : T(x)\}| > |\{x : \neg T(X)\}|\}$. Note that $L_{\bigoplus P}$ is self-embeddable, therefore the normal succinct encoding suffices. On the other hand, PP needs the compact succinct encoding.

6 Iteration

Our improvement of the complexity upgrading results allows iterative application. That is, given a problem A complete under PLT reductions, we can assert the complexity of $s(A)$, $i(A)$, $s(s(A))$, $i(i(A))$, etc.

For this purpose, we define the notion of $k-$fold succinct representation: For a problem Π, its $k-$fold succint representation is $i_k(\Pi) := \underbrace{i(\ldots(i}_{k \text{ times}}(\Pi)\ldots)$. As an informal problem description, this amounts to

$k - \Pi$
Instance: A circuit C
Query: $\underbrace{gen(\ldots(gen}_{k \text{ times}}(C)\ldots) \in \Pi$?

We define the tower function

$$t(n,k) = \underbrace{2^{2^{\cdots^{2^n}}}}_{k \text{ times}}$$

It is now easy to give a lot of complexity characterizations. This is exemplified by the following few corollaries:

Corollary 13 *If Π is* **NP***-complete under PLT-reductions, then $i_k(\Pi)$ is $NTIME(t(n,k))$ complete under monotone projection reductions.*

Thus, for example $k-$CIRCUIT SAT is complete for k-**NEXP** under monotone projection reductions.

Corollary 14 *If Π is logspace-complete under PLT-reductions, then $i_k(\Pi)$ is $DSPACE(t(\log n, k))$ complete under monotone projection reductions.*

Corollary 15 *If Π is Θ_k^P complete under PLT-reductions, then $i_k(\Pi)$ is $DSPACE(t(\log n, k))^{\Sigma_{k-1}^P}$ complete under monotone projection reductions.*

The class **ELEMENTARY** is defined as $\bigcup_{i>0} \mathrm{DTIME}(t(n,i))$. Since the classes mentioned above can be easily seen to be located between the levels of **ELEMENTARY**, we obtain the following characterization for elementarily decidable problems:

Corollary 16 $\Pi \in \mathrm{Struct}(\tau)$ *is an elementarily decidable problem iff there exists a constant* k*, s.t.* $\Pi \leq^{mon}_{proj} i_k(L_P)$.

A computational problem $\Pi \subseteq \mathrm{Struct}(\pi)$ gives rise to a generalized quantifier $[\Pi](\lambda \mathbf{x} : \phi(\mathbf{x}))$ which evaluates to true iff the structure described by ϕ belongs to Π. (For exact definitions, refer to [Immerman 87].)

Consider the logic $FO_s + [\mathrm{i_k(L_P)}]$, i.e. the extension of first order logic by the generalized quantifier obtained from $i_k(L_P)$. $FO_s + [\mathrm{i^*(L_P)}]$ denotes the logic obtained by adding all such quantifiers to FO_s. Then it is easy to see the following:

Corollary 17 $FO_s + [\mathrm{i^*(L_P)}]$ *captures* **ELEMENTARY**.

It follows that both logics have normal forms, i.e. each class of structures definable in one of the above logics is definable by a sentence of the form $[\mathrm{s_k(L_P)}](\lambda \mathbf{x} : \phi(\mathbf{x}))$, where ϕ is a monotone projection formula.

7 Conclusion

The succinct representation technique has been shown to match the **NP** leaf language approach to unify the treatment of complexity classes in a fortunate manner. In particular, it was possible to find generic complete problems under very weak logical reductions for all leaf language definable syntactic complexity classes.

The notion of self-embeddability has been shown to be very helpful for proving completeness results under weak logical reductions. In fact, one can argue that an informally stated problem should be encoded in a self-embeddable manner to allow easy reducibility.

A natural question to ask is if there is a similar correspondence between circuits and **NL** leaf languages, as investigated by [Jenner et al 94].

This hardly seems to be possible using our method because a result in the style of Theorem 2 involves simulating **NL** machines by boolean circuits. This

rises the old open problem of NC versus logspace, so it seems to be a challenging question.

In future work we would like to extend the approach of this paper to the general framework presented in [Bovet et al 92, Bovet et al 91] where the pattern sets for acception and rejection are not necessarily complementary, and the boolean circuits are equipped with oracles.

References

[Balcázar et al. 88] J.L.Balcázar, J.Diaz, J.Gabarro: Structural Complexity I. Springer Verlag, 1988.

[Balcázar et al. 92] J.L.Balcázar, A.Lozano, J.Toran: The Complexity of Algorithmic Problems on Succinct Instances. Algorithms Review 2(3), 1992.

[Balcázar 95] J.Balcázar: The Complexity of Searching Implicit Graphs. In: *Proceedings of the ICALP 95.* Report LSI-95-3-R, Universitat Polytecnica de Catalunya. To appear in Siam Journal of Computing.

[Barrington 89] D.A.Barrington: Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1. Journal of Computer and System Sciences, 38, 150-164 (1989).

[Bovet et al 91] D.P.Bovet, P.Crescenci, R.Silvestri: Complexity classes and sparse oracles; *Proceedings of the 6th Structure in Complexity Theory Conference* (1991), pp.102-108.

[Bovet et al 92] D.P.Bovet, P.Crescenci, R.Silvestri: A uniform approach to define complexity classes. Theoretical Computer Science 104(1992), pp.263-283.

[Cai, Furst 87] J.-Y. Cai, M.Furst: PSPACE Survives Three-Bit Bottlenecks. International Journal of Foundations of Computer Science 2 (1991), pp.67-76.

[Eiter et al. 94] T.Eiter, G.Gottlob, H.Mannila: Adding Disjunction to Datalog. In: *Proceedings of the Twelfth ACM SIGACT SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-94)*, 1994.

[Enderton 72] H.B. Enderton: A Mathematical Introduction to Logic. Academic Press, 1972.

[Fagin 74] R.Fagin: Generalized First Order Spectra and Polynomial-Time Recognizable Sets. In: R.M.Karp, ed., Complexity of Computation, pp.43-74. AMS, 1974.

[Galperin, Widgerson 83] H.Galperin, A.Widgerson: Succinct Representation of Graphs. Information and Control 56, 183-198 (1983).

[Gottlob et al 95] G.Gottlob, N.Leone, H.Veith: Second Order Logic and the Weak Exponential Hierarchies. Technical Report CD-TR 95/80, Christian Doppler Laboratory for Expert Systems, TU Vienna. Short version in *Proceedings of the MFCS 95*, Prague, 1995.

[Gurevich 88] Y.Gurevich: Logic and the Challenge of Computer Science. In: *Trends in Theoretical Computer Science*, E.Börger, ed., Computer Science Press, 1988.

[Hertrampf et al 93] U.Hertrampf, C.Lautemann, T.Schwentick, H.Vollmer, K.W.Wagner: On the Power of Polynomial Bit-Reductions. Proceedings of the 1993 Structure in Complexity Theory conference, pp.200-207.

[Hodges 93] W.Hodges: Model Theory. Cambridge University Press, 1993.

[Immerman 87] N. Immerman: Languages that Capture Complexity Classes. SIAM J. Computation, Vol. 16, No. 4, pp.760-778, 1987.

[Jenner et al 94] B.Jenner, P.McKenzie, D.Thérien: Logspace and Logtime Leaf Languages. Report LSI-94-1-R, Universitat Polytecnica de Catalunya.

[Lozano, Balcázar 89] A.Lozano, J.L.Balcázar: The Complexity of Graph Problems for Succinctly Represented Graphs. In: G.Goos,J.Hartmanis, eds., *Graph-Theoretic Concepts in Computer Science*. Lecture Notes in Computer Science 411, Springer, 1989, 277-286.

[Papadimitriou 94] C.H. Papadimitriou, Computational Complexity, Addison Wesley, 1994.

[Papa, Yann 85] C.H. Papadimitriou and M. Yannakakis, A Note on Succinct Representations of Graphs, Information and Control, 71:181–185, 1985.

[Stewart 91] I.A.Stewart: Complete Problems Involving Boolean Labelled Structures and Projection Transactions. Journal of Logic and Computation, Vol.1 No.6, pp.861-882, 1991.

[Stewart 92] I.A. Stewart: Using the Hamiltonian Path Operator to Capture NP. Journal of Computer and System Sciences 45, pp. 127-151 (1992).

[Stewart 93a] I.A.Stewart: Methods for proving completeness via logical reductions. Theoretical Computer Science 118 (1993), 193-229.

[Stewart 93b] I.A.Stewart: On Completeness for NP via Projection Translations. Lecture Notes in Computer Science 626 (1993), pp.353-366.

[Stewart 94] I.A.Stewart : Logical Descriptions of Monotone NP Problems. J.Logic Computat., Vol. 4 No 92-18, pp. 1-21, 1994.

[Vardi 82] M.Vardi: Complexity of Relational Query Languages. In *Proceedings 14th STOC*, pp.137-146, 1982.

[Veith 94] H.Veith: Logical Reducibilities in Finite Model Theory. M.Sc. Thesis, Technical University of Vienna, Vienna, 1994.