

Succinct circuit representations and leaf language classes are basically the same concept ¹

Bernd Borchert ^{a,2}, Antoni Lozano ^{b,*}

^a Universität Heidelberg, Im Neuenheimer Feld 294, D-69120 Heidelberg, Germany

^b Departament L.S.I., Universitat Politècnica de Catalunya, Pau Gargallo 5, 08028 Barcelona, Catalonia, Spain

Received 22 January 1996; revised 31 May 1996

Communicated by L.A. Hemaspaandra

Abstract

This note connects two topics of complexity theory: The topic of *succinct circuit representations* initiated by Galperin and Wigderson (1983), and the topic of *leaf language classes* initiated by Bovet et al. (1992). It will be shown for any language that its succinct version is polynomial-time many-one complete for the leaf language class determined by it.

Keywords: Computational complexity; Leaf language classes; Succinct representations; Polynomial-time many-one completeness

1. Introduction

Consider the following well-known results concerning nondeterministic polynomial-time Turing machines, polynomial-time many-one reducibility \leq_m^P , and Boolean circuits:

Let NP (PP, C = P, \oplus P, 1-NP) be the class of languages for which there is a nondeterministic polynomial-time Turing machine which accepts if and only if there exists an accepting computation path

(there is a majority of accepting computation paths, exactly half of the computation paths are accepting, there is an odd number of accepting computation paths, there is exactly one accepting path). The set of circuits for which there exists a satisfying assignment (there is a majority of satisfying assignments, exactly half of the assignments are satisfying, there is an odd number of satisfying assignments, there is exactly one satisfying assignment) is \leq_m^P -complete for the class NP (PP, C = P, \oplus P, 1-NP).

The common pattern in the results above is obvious: satisfying assignments for circuits and accepting computation paths for nondeterministic polynomial-time Turing machines seem to correspond. And not only the results are alike but also the proofs: once you know a proof of one of these results you immediately see how the others are proven. So it is no surprise that there is a general rule behind this from which all the results above can be concluded as

* Corresponding author. Email: lozano@lsi.upc.es.

¹ This result was observed in April 1995 when the first author visited the site of the second author, supported by the EC Human Capital and Mobility project CoLoReT CHRX-CT93-0415, and by DAAD Acción Integrada 322-AI-e-dr. The first presentation was given in September 1995 at the Annual Meeting of the German Mathematical Society (DMV) in Ulm.

² Email: bb@math.uni-heidelberg.de.

special cases. This general rule will be formulated as our main result. It states a strong relation between classes defined by leaf languages (all the classes above are defined by leaf languages) and succinct circuit representations (all the circuit problems above are succinct versions). More specifically, it will be shown for any given language A that its succinct circuit version is \leq_m^p -complete for the class determined by A as a leaf language.

In Section 2 and Section 3 we will motivate and define the two concepts of succinct circuit versions and leaf language classes, respectively. The main result with its corollaries will be stated in Section 4.

Veith in his paper [13] independently and simultaneously found our main result in an even stronger form: using slightly different definitions he shows completeness with respect to some stronger reducibility from finite model theory. Our paper concentrates on the main result and shows that it is easily proven by standard methods not using the concepts of finite model theory.

2. Succinct circuit representations

The topic of *succinct circuit representations* was initiated by Galperin and Wigderson [7]. The idea is the following. Instead of representing a graph having 2^n nodes by its adjacency matrix, it is represented by a circuit with two length- n vectors of variables: An assignment to a vector encodes a node, and for each assignment to the two vectors the circuit determines whether there is an edge from one node to the other. If the graph is kind of regular then there may be a circuit which is logarithmically smaller than the adjacency matrix. Therefore it is no surprise that the computational complexity of several graph problems increases significantly if the succinct circuit representation is used, see for example [7,2]. Balcázar et al. [2] showed that the concept can easily be generalized from graph problems to general word problems. This idea will be adopted in this note.

For a standard definition of *circuits* see for example [1]. It is assumed that there is a linear order $<$ on the variables occurring in circuits. Let a circuit $c(x_1, \dots, x_n)$ with n occurring variables $x_1 < \dots < x_n$ be given, it describes in a natural way a word $\text{result}(c)$ of length 2^n : Its i th letter is the value of

the i th assignment, where assignments are ordered lexicographically. In other words, $\text{result}(c)$ is the result column of the truth table representation of c . Here are two examples: for the circuit $c = c(x_1, x_2) = x_1 \wedge x_2$ the word $\text{result}(c)$ equals 0001, for the circuit $d = d(x_1, x_2, x_3) = \neg x_1 \vee (x_2 \wedge \neg x_3)$ the word $\text{result}(d)$ equals 11110010.

Definition 1 (uncut succinct version). The *uncut succinct version* $S_u(A)$ of a language A is the set of circuits c such that $\text{result}(c) \in A$.

Of course, $S_u(A)$ only depends on the words of A whose length is a power of 2. Below it will become clear why we use the attribute *uncut*. We can consider $S_u(A)$ to be a computational problem by identifying a circuit with its usual encoding as a word (if a word x does not encode a circuit then $\text{result}(x)$ is – arbitrarily – defined to be the length-1 word 0).

Examples. Let A_1 be the language consisting of the words which contain at least one letter 1. Then $S_u(A_1)$ equals the set of circuits which have a satisfying assignment, in other words, $S_u(A_1)$ equals the classical circuit satisfiability problem, which is \leq_m^p -complete for NP [6,10]. As another example, let A_2 be the language consisting of the words which contain more 1's than 0's. Then $S_u(A_2)$ equals the set of circuits which have more satisfying assignments than non-satisfying assignments. $S_u(A_2)$ is known to be complete for the class PP. The reader may easily verify that also the other circuit problems mentioned in the examples in the introduction are uncut succinct versions of obvious languages A_3, A_4, A_5 . It should be mentioned that all these languages A_1, \dots, A_5 are of a very restricted kind: membership in the language depends for a word only on the number of 1's in it.

In order to let a succinct version also be determined by words with a length not a power of 2 we have the following definition: Fix some usual polynomial-time computable pairing function $\langle \cdot, \cdot \rangle$ and consider a coded pair $\langle c, m \rangle$ of a circuit $c = c(x_1, \dots, x_n)$ and a number $m \in \mathbb{N}$ in binary. Let $\text{result}(c, m)$ be the length- m prefix of $\text{result}(c)$. Note that this implies $\text{result}(c) = \text{result}(c, 2^n)$. Here is an example: for the circuit $c = c(x_1, x_2) = x_1 \wedge x_2$ from above, $\text{result}(c, 0) = \varepsilon$, $\text{result}(c, 1) = 0$,

result($c, 2$) = 00, result($c, 3$) = 000, and result(c, m) = 0001 for every $m \geq 4$.

Definition 2 (succinct version). The *succinct version* $S(A)$ of a language A is the set of pairs $\langle c, m \rangle$ such that result(c, m) $\in A$.

Example. Let A_1 like in the example above be the language consisting of the words which contain at least one letter 1. Then $S(A_1)$ equals the set of coded pairs $\langle c, m \rangle$ such that the circuit c has a satisfying assignment among its first m assignments. This version of the satisfiability problem is also well known to be NP-complete. In fact, for all the languages A_1, \dots, A_5 from the examples above we have that the succinct version $S(A_i)$ is \leq_m^p -equivalent to the uncut succinct version $S_u(A_i)$. This is not true in general, for example if a language not in P does not contain any word whose length is a power of 2. However, it can easily be checked that $S_u(A)$ is always \leq_m^p -reducible to $S(A)$ for any language A .

Remark. The definition of $S(A)$ is nearly the same concept as the definition of the succinct version sA in [2], but here the length of the described word belongs to the problem input and is not indicated by some additional output bit of the circuit. In their paper [2] and also in [7] Balcázar et al. implicitly assumed that the circuit is consistent in its length-indicating output bit, i.e. has the following property: If any assignment i evaluates the length-indicating output bit to 0, then also every lexicographically larger assignment evaluates it to 0. But the problem whether a given circuit is consistent can easily be shown to be co-NP-complete. Therefore, in the sense of [7,2], given a circuit c with two outputs, there is no efficient way of checking if the circuit describes a graph (respectively word) at all, unless $P = NP$.

3. Leaf language classes

The topic of *leaf languages* as a way to unify the definition of complexity classes was started by Bovet, Crescenzi and Silvestri [4,5]. They show that many well-known complexity classes in the polynomial-time setting can be determined by just one language.

Several following investigations refined this approach, see for example [8,3,9].

Consider nondeterministic polynomial-time Turing machines as described in [1]. Here we have the additional requirements that (1) all nondeterministic branchings are binary, and (2) all paths of the resulting binary tree representing the branchings have the same length. In other words, on every input a nondeterministic polynomial-time Turing machine produces a balanced complete binary computation tree whose leaves are marked with 0 for rejecting and 1 for accepting. Call these machines *normalized Turing machines*. For any normalized Turing machine M and any input x let yield(M, x) be the word consisting of the bits at the leaves (in lexicographic order of the paths) of the computation tree produced by M on input x . Note that the length of the word yield(M, x) is a power of 2 because the computation trees are by our convention balanced and complete.

Definition 3 (uncut leaf language class). For a language A let the *uncut leaf language class* $C_u(A)$ be the class consisting of the languages L for which there exists a normalized Turing machine M such that

$$x \in L \Leftrightarrow \text{yield}(M, x) \in A$$

Example. Let A_1 be the language consisting of all words which contain at least one letter 1. Then it is easy to verify that $C_u(A_1) = NP$: a language L is in $C_u(A_1)$ if and only if there is machine M such that $x \in L \Leftrightarrow \text{yield}(M, x) \in A_1$, in other words, $x \in L \Leftrightarrow M$ running on input x has an accepting computation path. But this is the original definition of Karp [10] for L being in NP. Likewise, just by their original definition, the classes PP, $C = P$, $\oplus P$, and 1-NP are uncut leaf language classes characterized by the languages A_2, A_3, A_4, A_5 , respectively, from the examples in the previous section.

The next definition makes clear why we called the leaf language classes $C_u(A)$ of Definition 3 *uncut* (in [8] the next definition was shown to be equivalent to the original definition of leaf language classes in [5]).

Definition 4 (leaf language class). For a language A let the *leaf language class* $C(A)$ be the class consist-

ing of the languages L such that there exist a normalized Turing machine M and a polynomial-time computable function $l: \Sigma^* \rightarrow \mathbb{N}$ (numbers are represented in binary) such that

$$x \in L \iff \text{the length-}l(x) \text{ prefix of } \text{yield}(M, x) \text{ is in } A.$$

It can easily be seen that for the languages A_1, \dots, A_5 it holds that $C_u(A_i) = C(A_i)$. Actually, for every language A it can be seen that $C_u(A)$ is a subset of $C(A)$. But the opposite direction does not hold generally, as can be seen by a language which is not in P and does not contain any word whose length is a power of 2. However, if we consider the set of all uncut leaf language classes and the set of all leaf language classes, they can be shown to coincide and can be characterized in the following way.

Theorem 5 (Borchert [3]). *The following sets of classes are equal:*

- (1) *The set of uncut leaf language classes $C_u(\cdot)$.*
- (2) *The set of leaf language classes $C(\cdot)$.*
- (3) *The set of complexity classes which, with respect to \leq_m^P -reducibility, have a complete language and are closed downward.*

4. The main result

The following theorem connects succinct circuit representations and leaf languages, thus justifying the title of this note.

Theorem 6 (main result). *For any language A the following holds.*

- (1) *The uncut succinct version $S_u(A)$ of A is \leq_m^P -complete for the uncut leaf language class $C_u(A)$.*
- (2) *The succinct version $S(A)$ of A is \leq_m^P -complete for the leaf language class $C(A)$.*

Proof. (1) First it is proven that for a fixed language A the language $S_u(A)$ belongs to the class $C_u(A)$. We have to show that there exists a normalized Turing machine M_0 such that for all words x it holds $\text{result}(x) \in A \iff \text{yield}(M_0, x) \in A$. Define M_0 the following way: For an input x which does not encode a circuit terminate with a rejecting state.

If the input encodes a circuit $c(x_1, \dots, x_n)$ then branch nondeterministically into two computations, the left one continuing with the circuit $c(0, \dots, x_n)$, the other with $c(1, \dots, x_n)$. Do this iteratively n times until also x_n is replaced by a constant. Then each of the 2^n computation paths has a Boolean circuit in which all variables are replaced by constants. Let the computation be accepting on that path if and only if the circuit evaluates to 1. It is clear by the construction and the properties of the lexicographic order that $\text{result}(x) = \text{yield}(M_0, x)$. Therefore $\text{result}(x) \in A \iff \text{yield}(M_0, x) \in A$, i.e. $S_u(A)$ belongs to the class $C_u(A)$.

Now we show that each language in $C_u(A)$ is \leq_m^P -reducible to $S_u(A)$. Let a language L in $C_u(A)$ be given by the machine M , i.e. $x \in L \iff \text{yield}(M, x) \in A$. Consider the following polynomial-time reduction function h . On input x , first compute the depth d of the computation tree produced by M on input x , this is possible by just simulating the leftmost path of the tree. After that, consider the following function g_x on inputs of length d : for an input $y_1 \dots y_d$ the value of g_x equals 1 if and only if machine M running on input x is accepting on the path determined by $y_1 \dots y_d$. This function g_x is computable in time polynomial in d and therefore also in $|x|$. Now consider a circuit $c_x(y_1, \dots, y_n)$ such that c_x does the same job as g_x , i.e. it satisfies $c_x(y_1, \dots, y_n) = g_x(y_1 \dots y_n)$ for all words $y_1 \dots y_n$. Such a circuit can be constructed in time polynomial in the running time of g_x and therefore in time polynomial in $|x|$ (and is therefore of polynomial size). For this construction see the book of Balcázar et al. [1] who refer to a paper of Savage [12], but actually this construction is already used as a key technique in the classical papers of Cook [6] and Karp [10], see also [11]. The intended and important property of c_x is that $\text{result}(c_x) = \text{yield}(M, x)$. Finally, let h be the polynomial-time computable function which maps an input x to c_x . Then $x \in L \iff \text{yield}(M, x) \in A \iff \text{result}(c_x) \in A \iff c_x \in S_u(A)$, the last equivalence holds just by the definition of $S_u(A)$. Therefore, h is a polynomial-time many-one reduction from L to $S_u(A)$.

For part (2), the proof is just an extension of the proof of part (1). First it is proven that the language $S(A)$ belongs to the class $C(A)$, so we have to

define a machine M_0 and a function l in order to apply Definition 4. For inputs $\langle x, m \rangle$ let M_0 be the machine from part (1) running on x , and let l be the function which maps $\langle x, m \rangle$ to m . M_0 and l witness that $S(A)$ belongs to $C(A)$ because $\langle x, m \rangle \in S(A) \Leftrightarrow$ the length- m prefix of $\text{result}(x)$ is in $A \Leftrightarrow$ the length- $l(\langle x, m \rangle)$ prefix of $\text{yield}(M_0, \langle x, m \rangle)$ is in A . Finally we have to show that each language in $C(A)$ is \leq_m^p -reducible to $S(A)$. Let a language L in $C(A)$ be given according to Definition 4 by a machine M and a function l such that $x \in L \Leftrightarrow$ the length- $l(x)$ prefix of $\text{yield}(M, x)$ is in A . Consider the polynomial-time computable function which on input x computes the pair $\langle c_x, l(x) \rangle$ where the circuit c_x is defined like in part (1) using the machine M . This function \leq_m^p -reduces L to $S(A)$ because $x \in L \Leftrightarrow$ the length- $l(x)$ prefix of $\text{yield}(M, x)$ is in $A \Leftrightarrow$ the length- $l(x)$ prefix of $\text{result}(c_x)$ is in $A \Leftrightarrow \text{result}(c_x, l(x)) \in A \Leftrightarrow \langle c_x, l(x) \rangle \in S(A)$.

Note that all constructions in the proof are independent of A . \square

As a first consequence note that the standard completeness results for the classes NP, PP, $C = P$, $\oplus P$, and 1-NP mentioned in the introduction are special cases of our main result. For the case of NP this means in other words: the main theorem above is a generalization of the classical result of Cook [6] and Karp [10] about the NP-completeness of the circuit satisfiability problem. Actually, also the proof of the above theorem can be seen as a generalization of the original proof of that result.

Together with Theorem 5 the above theorem implies the following corollary.

Corollary 7. *Each polynomial-time many-one degree contains an (uncut) succinct version.*

Also it can be concluded that an inclusion problem among classes which are (uncut) leaf language classes can be turned into a \leq_m^p -problem among the (uncut) succinct versions, and vice versa.

Corollary 8. *For all languages A, B the following holds.*

- (1) $C_u(A) \subseteq C_u(B) \Leftrightarrow S_u(A) \leq_m^p S_u(B)$.
- (2) $C(A) \subseteq C(B) \Leftrightarrow S(A) \leq_m^p S(B)$.

Acknowledgements

The authors are grateful to J. Balcázar, H. Veith, and the referees for helpful comments.

References

- [1] J.L. Balcázar, J. Díaz and J. Gabarró, *Structural Complexity I* (Springer, Berlin, 1988).
- [2] J.L. Balcázar, A. Lozano and J. Torán. The complexity of algorithmic problems on succinct instances, in: R. Baeza-Yates and U. Manber, eds., *Computer Science* (Plenum Press, New York, 1992) 351–377.
- [3] B. Borchert, Predicate classes and promise classes, in: *Proc. 9th Structure in Complexity Theory Conf.* (1994) 235–241.
- [4] D.P. Bovet, P. Crescenzi and R. Silvestri, Complexity classes and sparse oracles, in: *Proc. 6th Structure in Complexity Theory Conf.* (1991) 102–108.
- [5] D.P. Bovet, P. Crescenzi and R. Silvestri, A uniform approach to define complexity classes, *Theoret. Comput. Sci.* 104 (1992) 263–283.
- [6] S.A. Cook. The complexity of theorem proving procedures, *Proc. 3rd Ann. ACM Symp. on the Theory of Computing* (1971) 151–158.
- [7] H. Galperin and A. Wigderson, Succinct representations of graphs, *Inform. and Control* 56 (1983) 183–198.
- [8] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer and K. Wagner, On the power of polynomial time bit-computations, in: *Proc. 8th Structure in Complexity Theory Conf.* (1993) 200–207.
- [9] B. Jenner, P. McKenzie and D. Thérien, Logspace and logtime leaf languages, in: *Proc. 9th Structure in Complexity Theory Conf.* (1994) 242–254.
- [10] R. Karp, Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher, eds., *Complexity of Computer Computations* (Plenum Press, New York, 1972) 85–103.
- [11] R. Ladner, The circuit value problem is log space complete for P, *SIGACT News* 7 (1975) 18–20.
- [12] J.E. Savage, Computational work and time of finite machines, *J. ACM* 19 (1972) 660–674.
- [13] H. Veith, Succinct representation, leaf languages, and projection reductions, in: *Proc. 11th Ann. IEEE Conf. on Computational Complexity* (1996) 118–126.