Improved Witnessing and Local Improvement Principles for Second-Order Bounded Arithmetic

Arnold Beckmann*†
Department of Computer Science
Swansea University
Swansea SA2 8PP, UK
a.beckmann@swansea.ac.uk

Samuel R. Buss*‡
Department of Mathematics
University of California, San Diego
La Jolla, CA 92093-0112, USA
sbuss@math.ucsd.edu

March 10, 2012

Abstract

This paper concerns the second order systems U_2^1 and V_2^1 of bounded arithmetic, which have proof theoretic strengths corresponding to polynomial space and exponential time computation. We formulate improved witnessing theorems for these two theories by using S_2^1 as a base theory for proving the correctness of the polynomial space or exponential time witnessing functions. We develop the theory of nondeterministic polynomial space computation in U_2^1. Kołodziejczyk, Nguyen, and Thapen have introduced local improvement properties to characterize the provably total NP functions of these second order theories. We show that the strengths of their local improvement principles over U_2^1 and V_2^1 depend primarily on the topology of the underlying graph, not the number of rounds in the local improvement games. The theory U_2^1 proves the local improvement principle for linear graphs even without restricting to logarithmically many rounds. The local improvement principle for grid graphs with only logarithmically rounds is complete for the provably total NP search problems of V_2^1. Related results are obtained for local improvement principles with one improvement round, and for local improvement over rectangular grids.

*The authors thank the John Templeton Foundation for supporting their participation in the CRM Infinity Project at the Centre de Recerca Matemàtica, Barcelona, Catalonia, Spain in February 2011, during which this project was instigated.

†This research was partially done while the author was a visiting fellow at the Isaac Newton Institute for the Mathematical Sciences in the programme "Semantics & Syntax".

‡Supported in part by NSF grant DMS-1101228 and by a grant from the Simons Foundations (#208717 to Sam Buss).

1 Introduction

The theories U_2^1 and V_2^1 of bounded arithmetic are well-known to have proof theoretic strengths corresponding to polynomial space and exponential time computation. It is a fundamental and famous open question in computer science whether these two complexity classes are equal, a question which is related to the P versus NP question. Likewise, it is a fundamental open problem whether U_2^1 and V_2^1 are distinct. The difference in working with bounded arithmetic theories instead directly with computational classes is that the theories may possibly be shown to be distinct by combining logical considerations of provability along with computational complexity considerations. In this paper, we give improved characterizations of the multifunctions and the total NP search problems which are definable in U_2^1 and V_2^1. Our main results give improved theorems about the strengths of local improvement principles in U_2^1 and V_2^1. Along the way, we show that Savitch's theorem on non-deterministic polynomial space computation is formalizable in U_2^1, and we give improved "new-style" witnessing theorems.

A "multifunction" is a function which can have multiple values, namely a total relation. NP search problems are multifunctions f which have polynomial growth rate and whose graph is polynomial-time recognizable. The provably total NP search problems of a theory T of bounded arithmetic are the multifunctions which have polynomial time graph $G_f(x, y)$ such that T proves $(\forall x)(\exists y)G_f(x, y)$. If G_f is instead a Σ_i^b-formula, then f is a Σ_i^b-definable multifunction of T. The provably total NP search problems of T and the Σ_1^b-definable multifunctions of T are essentially the same, as the latter can be defined as projections of the former.

There have been a series of recent results giving new characterizations of the provably total NP search problems for theories of bounded arithmetic, and more generally the Σ_i^b-definable multifunctions of these theories. The most recent work in this direction includes [13, 11, 1, 2, 8]. The first four of these papers give characterizations of the Σ_i^b-definable functions of T_2^k for all $1 \leq i \leq k$. Skelley and Thapen [13] introduced k-round game principles, GI_k, which characterize the provably total NP search problems of T_2^k. Beckmann and Buss [1, 2] used an extension of polynomial local search (PLS) along with Skolemization techniques to characterize the Σ_i^b-definable multifunctions of T_2^k for $1 \leq i \leq k$. Pudlák and Thapen [11] gave another quite different characterization of the Σ_i^b-definable multifunctions of T_2^k based on alternating min-max principles. The fifth paper, Kołodziejczyk, Nguyen, and Thapen [8], extended the idea of the game principles to a "local improvement" principle and applied this to characterize the Σ_1^b-definable

multifunctions of the second order theories U^1_2 and V^1_2.

As we explain below in more detail, the present paper extends the results of [8] in several ways. The first part of the paper describes U^1_2 and V^1_2 and extends the bootstrapping of U^1_2 to show that U^1_2 can define nondeterministic polynomial space (NPSPACE) computations and can prove Savitch's theorem about the equivalence of deterministic and nondeterministic polynomial space. We then present improved witnessing theorems for U^1_2 and V^1_2. The final part of the paper improves the results of [8] that characterize the Σ^b_1-definable multifunctions in terms of local improvement (LI) and linear local improvement (LLI) principles. Our new results include that U^1_2 can prove the principle LLI, and that the $\mathrm{LI}_{\log}$ principle is (provably) many-one complete for the total NP search problems of V^1_2. This improves results from [8], who had proved weaker versions of these results with $\mathrm{LLI}_{\log}$ and LI in place of LLI and $\mathrm{LI}_{\log}$, respectively. We also characterize the strengths of the LLI_1 principle, and the rectangular local improvement principles, RLI, $\mathrm{RLI}_{\log}$, and RLI_1.

The original witnessing theorems [4] for bounded arithmetic followed the following general template. These witnessing theorems were formulated to apply to a theory T, a formula class Φ, and a complexity class $\mathcal{C}$. In most cases, the complexity class $\mathcal{C}$ has complete problems, and the functions in the complexity class $\mathcal{C}$ can be enumerated by specifying an algorithm for the function that uses specified computational resources. A function that is specified in such a way is said to be "explicitly $\mathcal{C}$". The witnessing theorem then states that if $\phi \in \Phi$ and $T \vdash (\forall \vec{x})(\exists y)\phi(\vec{x}, y)$, then there is an explicitly-$\mathcal{C}$ function f such that (a) T proves the totality of f and (b) T proves $(\forall \vec{x})\phi(\vec{x}, f(\vec{x}))$. For this, T does not need to have a function symbol for f, rather there is a formula G_f defining the graph of f, and condition (a) actually means that T proves $(\forall \vec{x})(\exists y)G_f(\vec{x}, y)$. Likewise, condition (b) means that T proves $(\forall \vec{x})(\forall y)[G_f(\vec{x}, y) \supset \phi(\vec{x}, y)]$. Buss [4, 5] established these kinds of results for the theories S^k_2, T^k_2, U^1_2, and V^1_2, and for function classes such as polynomial time, levels of the polynomial hierarchy, polynomial space, and exponential time. Buss and Krajíček [6] proved a witnessing theorem for T^1_2 and PLS. And, various authors have established a wide range of additional witnessing theorems; many of these are reported in a modern form in Cook-Nguyen [7].

In many cases, the witnessing theorem also includes a "uniqueness condition" that f is a function rather than a multifunction; namely, that T proves $(\forall \vec{x})(\exists! y)G_f(\vec{x}, y)$. However, there are some notable exceptions, namely those related to witnessing with PLS and game principles: these include (among others) [6, 1, 2, 8]. In these cases, the explicitly-$\mathcal{C}$ functions are

conjectured to be inherently multifunctions rather than functions, so the (conjectured!) failure of the uniqueness condition is unavoidable.

In nearly every case, the witnessing theorem is accompanied with a converse result stating that every explicitly-$\mathcal{C}$ function is provably definable in T with its graph G_f a formula from Φ.

Some recent witnessing theorems have followed an improved paradigm, which provides an extension of the template described above. These "new-style" witnessing theorems were used implicitly in [13] and more explicitly in [1, 2, 8, 15]. For the improved paradigm, the condition (b) of a witnessing theorem is replaced with

(b') S^1_2 proves $(\forall \vec{x})(\forall y)[G_f(\vec{x}, y) \supset \phi(\vec{x}, y)]$.

That is, the correctness of the witnessing function f is now proved in the (weaker) theory S^1_2 rather than in T.[1] Of course, in these situations, it is generally conjectured that S^1_2 does not necessarily prove the totality of f; thus (b') includes the existence of $y = f(\vec{x})$ as a hypothesis. We shall prove two such new-style witnessing theorems for U^1_2 and V^1_2 in Section 4.

Section 2 reviews quickly the definitions of the theories U^1_2 and V^1_2. We presume, however, that the reader has basic familiarity with the bounded arithmetic theories S^i_2 and T^i_2 and the syntactic classes Σ^b_i and Π^b_i. This section also introduces an alternate sequent calculus formulation of U^1_2 that will be useful for establishing normal forms for free-cut free proofs in U^1_2.

Section 3 shows that U^1_2 can formalize nondeterministic polynomial space computations. This is based on a formalization of Savitch's theorem that NSPACE(n) is contained in SPACE(n^2) and that hence PSPACE equals NPSPACE. The formalization of Savitch's theorem in U^1_2 is completely straightforward, but some care must be taken to show that it is possible for U^1_2 to pick out a particular nondeterministic computation path, including, for instance, the lexicographically first one. This construction is used in a crucial way for the proof of Theorem 15.

Section 4 establishes the two new-style witnessing theorems of U^1_2 and V^1_2. Of course, the two theories already have witnessing theorems linking them to polynomial space and exponential time computation, respectively. The new witnessing theorems use S^1_2 as a base theory as in (b') above, or more precisely, a conservative extension of S^1_2 to include second order variables. To formulate the witnessing theorem, we define a notion of what it

[1] So far, new-style witnessing theorems have been proved only for theories T that contain S^1_2. It should be straightforward to extend these results to use even weaker theories than S^1_2. No new-style witnessing theorems have been proved yet for theories $T \subseteq S^1_2$.

means for a second order object (or, "predicate") to "canonically verify" the truth of a bounded ($\Sigma_0^{1,b}$) formula. We then prove two witnessing lemmas, over the base theory S_2^1, about the witnessing of sequents of $\Sigma_1^{1,b}$ formulas that are provable in U_2^1 or V_2^1, using polynomial space or exponential time (respectively) computable predicates.

Kołodziejczyk, Nguyen, and Thapen [8] already proved new-style witnessing theorems for U_2^1 and V_2^1 using closure under certain types of iteration. The results of Section 4 use a more straightforward definition for polynomial space and exponential time computation, along with the notion of canonical verification. In addition, Theorems 11 and 12 for V_2^1 use S_2^1 as a base theory, rather than the ostensibly stronger theory T_2^1 which was used by [8]. This improvement of using S_2^1 as the base theory will be crucial later for the proof of Theorem 16. (Since S_2^1 is conservative over PV, we could equally well use PV as the base theory.)

Section 5 discusses the local improvement principles of [8]. Loosely speaking, a local improvement principle uses a directed acyclic graph G: the vertices in the graph G are assigned labels with scores. Initially all labels have score value equal to zero, but a mechanism is provided to make local updates to labels which increment scores by one. This local update proceeds by sweeping across the graph, and is well-defined since the graph is acyclic. In essence, the local improvement principle states that the scores can be incremented for a certain number, c, of rounds. (The actual formulation of the local improvement principles will be as a set of contradictory assertions, which yields an NP search problem.)

There are two kinds of local improvement principles: the principle LI has underlying graph G on N vertices with constantly bounded in- and out-degrees, and LLI uses a linearly ordered set of N points as its underlying graph. (The value N will be first order, but not a length.) The principles LI and LLI both use $c = N^{O(1)}$ many rounds of score increases. Limiting the number of rounds to instead be $c = O(\log N)$ gives the $\mathrm{LI}_{\log}$ and $\mathrm{LLI}_{\log}$ principles. When using exactly $c = 1$ rounds, the principles are called LI_1 and LLI_1.

Prior work [8] proved, for T the theory U_2^1 (respectively, V_2^1), that the $\mathrm{LLI}_{\log}$ principle (respectively, the LI principle) is provable in T, and is many one complete for the provably total NP search problems of T, provably in S_2^1. Section 5 concludes with new improved results; namely, that U_2^1 proves the LLI principle, and that the $\mathrm{LI}_{\log}$ principle is many-one complete for the provably total functions of V_2^1, provably in S_2^1. In fact, it follows that LLI and $\mathrm{LLI}_{\log}$ are equivalent over S_2^1, and that LI and $\mathrm{LI}_{\log}$ are equivalent

over S^1_2. Consequently, the strength of these local improvement principles depends on the underlying topology of the directed graph G, not on whether the number c of rounds is logarithmic or polynomial.[2]

The *rectangular* local improvement principles, RLI, are the versions of LI where the graph G is a grid graph. We prove that the RLI and $\mathrm{RLI}_{\log}$ principles are equivalent to each other and to LI and $\mathrm{LI}_{\log}$, over S^1_2. For local improvement principles with two rounds, we prove that, over S^1_2, the LI_1 principle is equivalent to the last four mentioned principles, and that RLI_1 is equivalent to LLI and $\mathrm{LLI}_{\log}$. However, the strength of RLI_k for constant $k \geq 2$ remains an open question.

Sections 5.2 through 5.4 present the proofs of our results on the local improvement properties.

We thank Leszek Kołodziejczyk and Neil Thapen for useful discussions, comments, and corrections.

2 Preliminaries for U^1_2 and V^1_2

We assume the reader is familiar with the essentials of bounded arithmetic, for which see [4, 9]; however, we give a quick review to establish notation. Most of the paper is concerned with second order theories in the form defined in Chapter 9 of [4]. Since these second order theories are less well-known, we describe them below in a bit more detail. Our theories all use the non-logical language $0, S, +, \cdot, |\cdot|, \lfloor \cdot/2 \rfloor, \#, \leq$. Quantifiers of the form $(\exists x \leq t)$ and $(\forall x \leq t)$ are called *bounded quantifiers*. If the term t is of the form $|s|$, the quantifier is *sharply bounded*. The classes Σ^b_i and Π^b_i are defined by counting alternations of bounded quantifiers, ignoring sharply bounded quantifiers. The theories S^i_2 are axiomatized with a set, BASIC, of open axioms defining the non-logical symbols plus the Σ^b_i-PIND induction, namely polynomial induction, or equivalently, length induction. The theories T^i_2 are axiomatized with the axioms of BASIC plus Σ^b_i-IND, namely the usual induction axioms. Restricting to the case of $i = 1$, the main witnessing theorems for S^1_2 and T^1_2 state that S^1_2 can Σ^b_1-define precisely the polynomial time functions [4], and that T^1_2 can Σ^b_1-define precisely the PLS (polynomial local search) multifunctions [6].

Second order theories of bounded arithmetic extend the first order theories by adding second order variables, $X, Y, Z, \ldots$, intended to range over sets, also called "predicates". The membership $\in$ symbol is added to the language as well; the formula $t \in X$ denotes that t is in X. We often write

[2] Again, by conservativity, the same results hold over the base theory PV.

$X(t)$ instead of $t \in X$. It is convenient to now let the classes Σ_i^b and Π_i^b involve free second order variables (but no quantified second order variables). Thus, a *bounded quantifier* is a bounded, first order quantifier; a *bounded formula* is a formula with no unbounded first order quantifiers and no second order quantifiers. We also let S_2^i and T_2^i now be defined with second order variables allowed to appear in formulas, including as free variables in induction axioms. But again, second order quantifiers are not allowed in induction formulas for S_2^i and T_2^i. (Sometimes these extensions of S_2^i and T_2^i to second order logic are denoted S_2^{i+} and T_2^{i+}, but since there is no chance of confusion, we prefer to omit the superscript "+". Likewise, we eschew the notations Σ_i^{b+} and Π_i^{b+}.)

We reserve lower-case letters $a, b, c, \ldots$ and $z, y, x, \ldots$ for first order variables, and upper-case letters $A, B, C, \ldots$ and $Z, Y, X, \ldots$ for second order variables. Occasionally, we use Greek letters α, β, γ for second order variables as well. We use ϕ, ψ, and χ for formulas.

Second order bounded formulas are classified with the classes $\Sigma_i^{1,b}$ and $\Pi_i^{1,b}$ by counting the alternations of second order quantifiers, ignoring any first order quantifiers. The class $\Sigma_0^{1,b}$ is the set of bounded formulas, namely the set of formulas with no second order quantifiers but with arbitrary (first order) bounded quantifiers. The class $\Sigma_1^{1,b}$ is the set of formulas with all second order quantifiers essentially existential (that is, existential after negations are pushed inward), and arbitrary bounded first order quantifiers.

The theories U_2^1 and V_2^1 both contain all of T_2, plus the $\Sigma_0^{1,b}$-comprehension axioms, namely

$$(\forall \vec{x})(\forall \vec{X})(\exists Z)(\forall y \leq t)[y \in Z \leftrightarrow \phi(y, \vec{x}, \vec{X})] \tag{1}$$

for every bounded formula ϕ and term t. This axiom states that any set (on a bounded domain) defined by a bounded formula ϕ with parameters is coded by some second order object Z.[3] The theory U_2^1 has in addition the $\Sigma_1^{1,b}$-PIND axioms. The theory V_2^1 has instead the $\Sigma_1^{1,b}$-IND axioms. It is known that $V_2^1 \vdash U_2^1$.

Note that the $\Sigma_0^{1,b}$-comprehension axiom above is a $\Pi_2^{1,b}$-sentence; or,

[3]The original definition [4] of U_2^1 used an unbounded version of the comprehension axiom; namely, the bounded quantifier $(\forall y \leq t)$ was replaced with the unbounded quantifier $(\forall y)$. In the present paper, we are interested in only $\Sigma_i^{1,b}$-consequences of U_2^1, and, by Parikh's theorem, the unbounded version of comprehension gives no additional $\Sigma_i^{1,b}$-consequences. See alternately the discussion of the theories U_2^1(BD) and U_2^1(BD) in [4]. At any rate, subsequent authors have preferred the bounded versions of comprehension (e.g., [7, 9, 10]), perhaps because it is better behaved model-theoretically.

stripping off leading universal quantifiers, it is a $\Sigma_1^{1,b}$-formula, in fact a strict $\Sigma_1^{1,b}$-formula, as will be defined momentarily.

As a side remark, we note that the second order systems can be conservatively extended to include second order function variables which range over functions with a specified polynomial growth rate. Then, the $\Sigma_0^{1,b}$-comprehension implies the following $\Sigma_0^{1,b}$ function comprehension axiom for a function symbol δ with growth rate bounded by the term s:

$$(\forall \vec{x})(\forall \vec{X})(\exists \delta)(\forall y \leq t)[(\exists z \leq s)\phi(y, z, \vec{x}, \vec{X}) \supset \delta(y) \leq s \wedge \phi(y, \delta(y), \vec{x}, \vec{X})]$$

where ϕ is a $\Sigma_0^{1,b}$-formula [4, p. 164]. The $\Sigma_0^{1,b}$ function comprehension axiom is effectively subsumed by $\Sigma_0^{1,b}$-comprehension, since $\Sigma_0^{1,b}$-comprehension can define the bit graph of δ so that $\delta(y)$ equals the least $z \leq s$ satisfying $\phi(y, z, \vec{x}, \vec{X})$, if any such z exists.

However, for simplicity and without loss of generality, we formulate U_2^1 and V_2^1 with only second order predicate symbols and without second order function symbols.[4]

A function $f(\vec{x})$ is said to be $\Sigma_1^{1,b}$-*defined* by a theory T provided that T proves $(\forall \vec{x})(\exists y)\phi(\vec{x}, y)$ where $\phi(\vec{x}, y)$ defines the graph of f and $\phi \in \Sigma_1^{1,b}$. The original witnessing theorems of [4, Ch.10] for U_2^1 and V_2^1 characterize their $\Sigma_1^{1,b}$-defined functions in terms of computational complexity. Namely, U_2^1 can $\Sigma_1^{1,b}$-define precisely the polynomial growth rate functions which are computable by polynomial space Turing machines, and V_2^1 can $\Sigma_1^{1,b}$-define precisely the polynomial growth rate functions which are computable in exponential time (that is, time $2^{n^{O(1)}}$). A formula $\psi(\vec{x}, \vec{X})$ is said to be $\Delta_1^{1,b}$-definable by T provided that T proves ψ is equivalent to both a $\Sigma_1^{1,b}$-formula and a $\Pi_1^{1,b}$-formula. A corollary to the witnessing theorems for U_2^1 and V_2^1 states that the $\Delta_1^{1,b}$-predicates of U_2^1 (respectively, V_2^1) are precisely the polynomial space predicates (respectively, the exponential time predicates). In addition, U_2^1 can prove the $\Delta_1^{1,b}$-IND and $\Delta_1^{1,b}$-MIN principles (see Theorem 16 of Chapter 9 of [4]). This means that U_2^1 can use polynomial space predicates and functions freely for induction and minimization. In short, U_2^1 can carry out a range of arguments about polynomial space predicates and functions.

We now define the notion of "strict" $\Sigma_1^{1,b}$-formula in analogy with similar notions for bounded formulas. A $\Sigma_1^{1,b}$-formula is *strict* provided that it contains at most one second order existential quantifier, and this quantifier

[4]Theorem 5 of Chapter 9 of [4] proves the conservativity between the theories with and without function symbols.

is the outermost connective. That is, a formula is strict $\Sigma_1^{1,b}$ provided either it is either a bounded formula, or it has the form $(\exists X)\phi$ where ϕ is bounded. By comparison, a non-strict $\Sigma_1^{1,b}$-formula may have connectives and bounded quantifiers in front of the second order quantifiers. We shall sometimes use the notation $\mathrm{s}\Sigma_1^{1,b}$ to denote the class of strict $\Sigma_1^{1,b}$-formulas.

It is useful to restrict proofs to contain only *strict* $\Sigma_1^{1,b}$-formulas as this will considerably simplify the proofs of the new-style witnessing theorems of Section 4. Both U_2^1 and V_2^1 can prove that any $\Sigma_1^{1,b}$-formula is equivalent to a strict $\Sigma_1^{1,b}$-formula by using $\Sigma_1^{1,b}$-replacement principles, which are theorems of both U_2^1 and V_2^1 (see Theorem 16 of Chapter 9 of [4]). Thus it is reasonable to assume that any free-cut free U_2^1- or V_2^1-proof of a strict $\Sigma_1^{1,b}$ formula could be restricted to contain only strict $\Sigma_1^{1,b}$-formulas.

For V_2^1 this works readily. It is easy to check that V_2^1 can prove the $\Sigma_1^{1,b}$-replacement principles, and more generally prove the equivalence of any given $\Sigma_1^{1,b}$-formula to a strict $\Sigma_1^{1,b}$ formula, while using induction only on strict $\Sigma_1^{1,b}$ formulas. Thus, the usual free-cut elimination theorem (c.f., [3]) gives the following.

Theorem 1 *Suppose V_2^1 proves a sequent $\Gamma \longrightarrow \Delta$ of strict $\Sigma_1^{1,b}$-formulas. Then there is a V_2^1-proof of $\Gamma \longrightarrow \Delta$ in which every formula is strict $\Sigma_1^{1,b}$.*

For U_2^1, the situation is less simple. The known proof in U_2^1 that every $\Sigma_1^{1,b}$-formula is equivalent to a strict $\Sigma_1^{1,b}$-formula uses induction on non-strict $\Sigma_1^{1,b}$-formulas.[5] Thus, free-cut elimination seemingly cannot be used with the usual formulation of U_2^1 to obtain proofs containing only strict $\Sigma_1^{1,b}$-formulas as cut formulas.

We can instead use a trick, and work with a slightly reformulated version of the theory U_2^1 called U_2^{1*}.

Definition An $\mathrm{s}\Sigma_1^{1,b}$-repl-$\forall$ inference is an inference of the form

$$\frac{a \leq t, \Gamma \longrightarrow \Delta, (\exists X)\phi(X, a)}{\Gamma \longrightarrow \Delta, (\exists Y)(\forall x \leq t)\phi(\{z\}Y(\langle x, z\rangle), x)}$$

where ϕ is a $\Sigma_0^{1,b}$-formula, a is an eigenvariable, and $\langle x, z\rangle$ is the usual pairing function used for bounded arithmetic. The notation $\{z\}Y(\langle x, z\rangle)$ denotes an *abstract* in the sense of Takeuti [14]. An abstract is akin to a lambda term but is not a syntactic part of the language; instead it is removed by the

[5] The proof of $\Sigma_1^{1,b}$-replacement in U_2^1 given for Theorem 16 of Chapter 9 of [4] uses a doubling trick that seems to depend essentially on the use of non-strict $\Sigma_1^{1,b}$-formulas.

process of substitution. Namely, $\phi(\{z\}Y(\langle x, z\rangle), x)$ is the formula obtained from $\phi(X, a)$ by replacing every occurrence of the variable a with x, and every occurrence of any subformula $X(s)$ with $Y(\langle x, s\rangle)$.

Definition The theory U_2^{1*} is defined to be U_2^1, but with $\mathrm{s}\Sigma_1^{1,b}$-PIND instead of $\Sigma_1^{1,b}$-PIND, and with $\mathrm{s}\Sigma_1^{1,b}$-repl-$\forall$ as an additional rule of inference.

It is clear that the $\mathrm{s}\Sigma_1^{1,b}$-repl-$\forall$ inference is a derived rule of inference for U_2^1, although proving this in U_2^1 involves a cut on a non-strict $\Sigma_1^{1,b}$-formula. Therefore, U_2^1 proves all theorems of U_2^{1*}.

Theorem 2 *In U_2^{1*}, every $\Sigma_1^{1,b}$-formula can be proved equivalent to a strict $\Sigma_1^{1,b}$-formula.*

The theorem is straightforward to prove. The proof is by induction on the complexity of formulas and uses the $\mathrm{s}\Sigma_1^{1,b}$-repl-$\forall$ rule to handle the hard case of moving a bounded quantifier past a second order quantifier.

As a corollary, U_2^{1*} admits PIND induction on all $\Sigma_1^{1,b}$-formulas. This immediately implies the equivalence of U_2^1 and U_2^{1*}.

Corollary 3 *U_2^1 and U_2^{1*} have the same consequences.*

The difference between U_2^1 and U_2^{1*} is only that they have different formalizations for sequent calculus proofs. The sequent calculus is formalized in a standard way. It uses conventional rules for weak inferences, for cut, and for first order connectives. In place of induction axioms, it uses induction rules with side formulas. The theory U_2^{1*} admits the $\mathrm{s}\Sigma_1^{1,b}$-repl-$\forall$ rule of inference. The comprehension axiom (1) becomes the initial sequents

$$\longrightarrow (\exists Z)(\forall y \leq t)[y \in Z \leftrightarrow \phi(y, \vec{x}, \vec{X})].$$

This allows the second order $\exists$:right and $\forall$:right axioms to be formulated with only second order variables (instead of the more general substitution of abstracts). Namely, the two rules for second order existential quantifiers are:

$$\exists\text{:right}\ \frac{\Gamma \longrightarrow \Delta, \phi(A)}{\Gamma \longrightarrow \Delta, (\exists X)\phi(X)} \qquad \text{and} \qquad \exists\text{:left}\ \frac{\phi(A), \Gamma \longrightarrow \Delta}{(\exists X)\phi(X), \Gamma \longrightarrow \Delta}$$

where, for the $\exists$:left rule, the second order variable A is a eigenvariable and does not appear in the lower sequent of the inference. Dual rules are used for second order universal quantifiers. (However, second order universal quantifiers are never needed in our free-cut free proofs.)

Eliminating free-cuts from U_2^{1*}-proofs gives the following theorem.

Theorem 4 *Suppose U_2^1 proves a sequent $\Gamma \longrightarrow \Delta$ of strict $\Sigma_1^{1,b}$-formulas. Then there is a U_2^{1*}-proof of $\Gamma \longrightarrow \Delta$ in which every formula is strict $\Sigma_1^{1,b}$.*

It will be convenient to work with U_2^{1*} instead of U_2^1 for our witnessing constructions in Section 4. The downside of having $\mathrm{s}\Sigma_1^{1,b}$-repl-$\forall$ as an additional inference is more than offset by the convenience of working with only strict $\Sigma_1^{1,b}$-formulas in the proof of the witnessing lemmas.

3 Nondeterministic polynomial space in U_2^1

We next formalize, in U_2^1, Savitch's theorem [12] that nondeterministic polynomial space is equal to polynomial space. An important consequence for us is that this means that U_2^1 can use induction (IND) and minimization (MIN) on NPSPACE predicates. It turns out that Savitch's argument can be carried out inside U_2^1 without complications; nonetheless, it is useful to check the details of exactly how it is formalized.

Figure 1 shows the usual algorithm behind Savitch's theorem. We assume that M is a nondeterministic Turing machine, running on input w, with explicit polynomial space bound $p(n)$ where $n = |w|$. For convenience, we use the convention that the input w is written on a read only input tape. A *configuration* of $M(w)$ is a complete description of M's tape contents, head positions, and current state at a given instant of time.

Let C_{init} be the initial configuration of $M(w)$. We may assume without loss of generality that if there is an accepting computation for $M(w)$, then it ends with a known configuration C_{end} after a known number of steps t_{end}. Then, to determine if $M(w)$ has an accepting computation, one merely invokes

$$\textsc{Reachable}(\ w,\ C_{\text{init}},\ 0,\ C_{\text{end}},\ t_{\text{end}}\ ). \tag{2}$$

It is well-known that Savitch's algorithm uses only polynomial space. In fact, it is straightforward to formalize Savitch's algorithm in U_2^1 as an explicitly polynomial space bounded computation.

With efficient coding, a configuration of $M(w)$ can be written out with $d{\cdot}p(n)$ many bits; thus a configuration can be coded by a number $C < 2^{d\cdot p(n)}$. For convenience, we shall use $\mathrm{Bd}_M(n)$ to denote the term $d \cdot p(n)$ bounding the lengths of codes of configurations of M. It is not particularly important how configurations C are coded, but it is important that it be done in a straightforward matter so that information about the tape contents, tape head positions, current state, etc., can be extracted by polynomial functions of C, and so that our base theory S_2^1 can prove elementary properties about

```
REACHABLE( w, C_1, t_1, C_2, t_2 )

// C_1 and C_2 are configurations, and t_1 < t_2.
if t_2 = t_1 + 1 then
   if (C_2 follows from C_1 by one step of M) then
      return TRUE
   else
      return FALSE
   end if
else
   set t := ⌊(t_1 + t_2)/2⌋
   set C := 0
   loop while C < 2^{Bd_M(|w|)}
      if ( C codes a valid configuration of M(w)
               and REACHABLE(w, C_1, t_1, C, t)
               and REACHABLE(w, C, t, C_2, t_2) )
         Mark C as the identified configuration for time t.
         return TRUE
      end if
      set C := C + 1
   end loop
   return FALSE
end if
```

Figure 1: Savitch's algorithm is a recursively invoked procedure that does a depth first, divide-and-conquer, search for an accepting computation. It determines whether, starting in configuration C_1 at time t_1, the Turing machine M with input string w can reach configuration C_2 at time $t_2 > t_1$ by some nondeterministic computation.

configurations, including whether one configuration succeeds another, or what the possible next moves are from a given configuration.

Note that the algorithm in Figure 1 has a line for marking a configuration C as being "identified" as the time t configuration. It can certainly happen that more than one configuration C is identified for a particular time t. Indeed, suppose a recursive call REACHABLE(w, C_1, t_1, C, t) returns true. Then certainly some configuration is identified for each time $t' \in (t_1, t)$. If, however, the next call REACHABLE(w, C, t, C_2, t_2) returns false, then the Savitch algorithm proceeds to the next value of C, and retries the calls with the new value for C. This of course, can cause new configurations to be identified for the times $t' \in (t_1, t)$, etc.

Accordingly, when a particular call (2) to REACHABLE returns `TRUE`, we are interested in the *last* configuration that is identified as the time t configuration. Let $C[t]$ denote this last such configuration. We claim that the sequence of configurations $C[0]$, $C[1]$, $C[2]$, ..., $C[t_{\text{end}}]$ is in fact an accepting computation for the Turing machine M on input w, where $C[0]$ and $C[t_{\text{end}}]$ are C_{init} and C_{end}. We shall call this sequence of configurations the "Savitch computation" of $M(w)$.

A computation of $M(w)$ consists of $t_{\text{end}} + 1$ many configurations, each coded by a string of $d \cdot p(n)$ bits. Accordingly, the entire computation can be coded by $(t_{\text{end}} + 1) \cdot d \cdot p(n)$ many bits, where $n = |w|$. Since t_{end} is exponentially bounded in n, an entire computation of $M(w)$ can be coded, in U_2^1, by a second order object X. Namely, by letting $X(i)$ have truth value equal to the i-th bit of the computation, for $i < (t_{\text{end}} + 1) \cdot d \cdot p(n)$.

The claim is that U_2^1 can prove that if REACHABLE$(w, C_{\text{init}}, 0, C_{\text{end}}, t_{\text{end}})$ returns `TRUE`, then there is an X coding the entire Savitch computation of $M(w)$. A sketch of the proof is as follows. First note that there must be some second order object Z coding the entire computation of the call to REACHABLE. Consequently, $C[t]$ is computable in polynomial space (from w and t), namely by examining Z. (In fact, w.l.o.g., $C[t]$ is computable in polynomial time from Z.) The execution of REACHABLE as coded by Z contains many invocations of REACHABLE(w, C_1, t_1, C_2, t_2). Using either IND on the depth of the recursive calls, or PIND on the values $t_2 - t_1$, it can be proved that for any such invocation REACHABLE(w, C_1, t_1, C_2, t_2) which returns `TRUE`, the sequence C_1, $C[t_1{+}1]$, ..., $C[t_2{-}1]$, C_2 identified during the invocation is a valid computation for $M(w)$ starting in configuration C_1 and ending at C_2. The base case of the induction argument is trivial, and the induction step is immediate.

By this argument, we get the following theorem.

Theorem 5 *Let M be an explicitly polynomial space nondeterministic Turing machine. Then U_2^1 proves: "If there is a Y coding an accepting computation of $M(w)$, then* REACHABLE$(w, C_{\text{init}}, 0, C_{\text{end}}, t_{\text{end}})$ *returns* `TRUE`*. Conversely, if* REACHABLE$(w, C_{\text{init}}, 0, C_{\text{end}}, t_{\text{end}})$ *returns* `TRUE`*, then there exists an X coding the entire Savitch computation, and this is an accepting computation of $M(w)$."*

The first part of the theorem is proved by noting that the REACHABLE algorithm cannot fail to accept when it reaches the computation coded by Y.

Theorem 5 implies further that U_2^1 can prove natural properties about the existence of nondeterministic polynomial space computations. An example of this is that U_2^1 can prove it is possible to concatenate two partial computations. To formalize this, we can extend the notion of a Savitch computation to talk about the Savitch computation that starts at configuration C_1 at time t_1 and ends at configuration C_2 at time t_2. Then, we claim that U_2^1 can prove that if there are Savitch computations X and Y, one from C_1 at time t_1 to C_2 at time t_2 and the other from C_2 at time t_2 to C_3 at time t_3, then there is a Savitch computation from C_1 at time t_1 to C_3 at time t_3. Of course, the two computations X and Y cannot be merely concatenated to give a Savitch computation, since they may have different lengths, so their divide-and-conquer splitting points do not line up. Instead, however, their concatenation does give a (non-Savitch) computation, and then Theorem 5 implies the existence of the desired Savitch computation from C_1 to C_3.

Savitch computations provide a kind of canonical accepting computation; that is, if there is some accepting computation, then the Savitch computation exists and is unique. However, Savitch computations are a bit unnatural since they depend on the divide-and-conquer algorithm. An arguably more natural notion of canonical computation is a "lex-first" computation, which is defined as follows. We assume that each configuration has exactly two possible successor computations that can be reached in a single step. These two successors can be called the 0-successor and the 1-successor, say according to the order they appear in the transition relation table. In other words, we think of a nondeterministic algorithm of choosing exactly one random bit in each step, and moving according to that bit. A string Z of t_{end} many bits then fully specifies a computation. A *lex-first accepting computation* is defined to be the computation that arises from the lexicographically first Z that gives an accepting computation. Note that the string Z is exponentially long, and thus is represented in U_2^1 by the values of a second order object.

Of course the property that Z gives rise to a lex-first computation can

be expressed as a $\Pi_1^{1,b}$-property since it states that there does not exist a Z' lexicographically preceding Z which specifies an accepting computation. However, U_2^1 can also express this as a $\Delta_1^{1,b}$-property. To see this, let $C_Z[i]$ be the configuration reached after making i steps according to Z, and let $C'_Z[i]$ be the computation reached after making $i-1$ steps according to Z but making the i-step with the choice opposite to Z. Then Z gives rise to a lex-first computation if and only if, for each value i such that $Z(i) = 1$, there is no computation from $C'_Z[i]$ to the accepting configuration. The last condition is an coNPSPACE property, hence PSPACE; so the entire condition is $\Delta_1^{1,b}$.

Theorem 6 *Let M be an explicitly polynomial space nondeterministic Turing machine. Then U_2^1 proves: "If there is a Y coding an accepting computation of $M(w)$, then there exists a lex-first accepting computation of $M(w)$."*

The idea of the proof of Theorem 6 is the following: The Turing machine M, including its nondeterministic choices, is simulated step-by-step by a deterministic PSPACE algorithm M'. At each step, M' invokes a PSPACE algorithm to check whether there exists an accepting computation starting from the 0-successor of the current configuration. If so, M' selects the 0-successor as the next configuration of M. Otherwise, the 1-successor is selected. It is obvious that M' selects the lex-first accepting computation of M if there is any accepting computation. It is furthermore straightforward to show U_2^1 proves this.

4 Improved witnessing theorems for U_2^1 and V_2^1

This section states and proves the improved, new-style witnessing theorems for U_2^1 and V_2^1. First, we need to define what it means for a polynomial space or exponential time computation to output either a first order or second order object. Second, in Section 4.1, we define what it means for a (polynomial space) computation to "canonically evaluate" the truth of a $\Sigma_0^{1,b}$- or $\Sigma_1^{1,b}$-formula. The intuition behind this is simple: in order to canonically verify the truth of such a formula, the PSPACE algorithm does a brute force evaluation by considering all possible values for the first order quantified variables. However, the unexpected aspect is that all this must be formalizable in the weak base theory S_2^1, since the new-style witnessing theorems use S_2^1 as the base theory.

Sections 4.2 and 4.3 then state and prove the two new witnessing theorems and their associated witnessing lemmas.

We first establish some further conventions on how Turing machine computations are coded by second order objects and how they produce outputs. The previous section already discussed how configurations and complete computations are coded for polynomial space computations. This notion needs to be extended to handle exponential time computations. Suppose that M is a Turing machine, with input w of length n, and that M is either explicitly polynomial space or explicitly exponential time. The running time of M is bounded by a term t_{end} with value $t_{\text{end}} < 2^{q(n)}$ for some polynomial q. Configurations of $M(w)$ are to be coded in some straightforward way by a string of length $\leq \text{Bd}_M(w)$. For M in PSPACE, $\text{Bd}_M(w)$ equals $p'(n)$ for some polynomial p'. For M exponential time, $\text{Bd}_M(w)$ equals $2^{p'(n)}$, again for p' a polynomial. For a polynomial space computation, a configuration of $M(w)$ could be coded by a first order object $C < 2^{p'(x)}$. For exponential time machines however, a configuration is too large and must be coded as a second order object C, where $C(i)$ gives the i-th bit of the configuration. In either case, an entire computation of M can be coded by a string of $\text{Bd}_M(w) \cdot (t_{\text{end}} + 1)$ bits using a second order object X. The object X can code the computation by merely concatenating the codes $C[0], \ldots, C[t_{\text{end}}]$. As before, the exact details of the encoding are not important, however, S^1_2 must be able to define polynomial time functions that extract information about the states, tape head positions, and tape contents at any given time. Furthermore S^1_2 must be able to express other combinatorial properties about the computation; in particular, the condition that X codes a correct computation must be expressible by a Π^b_1-predicate in S^1_2.

If X codes a complete computation, $\mathit{out}(X)$ denotes the first order object output by the computation (if any). By encoding the computation of M in X appropriately, we can ensure that $\mathit{out}(X)$ is computable in polynomial time relative to X. We sometimes allow a polynomial space or exponential time Turing machine M to also output a second order object, and use $\mathit{Out}(X)$ to denote the second order object output (if any). The encoding X must allow the second order object $\mathit{Out}(X)$ to be polynomial time computable, in that the value of $\mathit{Out}(X)(i)$ is computable in polynomial time relative to X. For an exponential time machine, which has exponentially large configurations, this can be done by using a separate output tape for the second order output. For a polynomial space machine, this can be done by requiring M to write each value $\mathit{Out}(X)(i)$ at a special tape location at a prespecified time that is easily computed from i. This permits configurations of M to be coded by first order objects in spite of the fact that M outputs an exponentially large second order object. It also permits $\mathit{Out}(X)(i)$ to be computed in polynomial time relative to X.

4.1 Canonical evaluation and canonical verification

We now define the notion of how a second order object α can "canonically evaluate" or "canonically verify" a bounded formula. It is important for later developments that these notions make sense over the base theory S^1_2.

Let $\phi(\vec{x}, \vec{X})$ be a first order bounded formula with all free variables indicated. Without loss of generality, the formula ϕ is in prenex form and, for notational convenience, we also assume that the quantifiers are alternating existential and universal, and that they all use the same bounding term $t(\vec{x})$. (These assumptions can be made without loss of generality in any event since we are only concerned that formulas are bounded, but not concerned about what Σ^b_i or Π^b_i class they are in.) Thus, we can assume ϕ has the form

$$(\exists y_1 \leq t)(\forall y_2 \leq t)(\exists y_3 \leq t) \cdots (Q_k y_k \leq t)\psi(\vec{y}, \vec{x}, \vec{X}),$$

with ψ quantifier-free. Here we are temporarily adopting the notation that, for $i \leq k$, Q_i is "$\exists$" if k is odd, and "$\forall$" otherwise. We next define what it means for α to *canonically evaluate* $\phi(\vec{x}, \vec{X})$. An input to α will be interpreted as a tuple of the form $\langle a_1, a_2, \ldots, a_\ell \rangle$ where $0 < \ell \leq k+1$ and where $0 \leq a_i \leq t$ for each i. Any standard sequence encoding may be used for coding tuples.

The intuition is that if $\ell \leq k$ then $\alpha(\langle a_1, \ldots, a_\ell, t \rangle)$ is true if and only if

$$(Q_{\ell+1} y_{\ell+1} \leq t) \cdots (Q_k y_k \leq t)\psi(a_1, \ldots, a_\ell, y_{\ell+1}, \ldots, y_k, \vec{x}, \vec{X})$$

is true. Note the final value in the tuple is t. However, more generally, the intuition is that if $\ell < k$, then $\alpha(\langle a_1, \ldots, a_\ell, a_{\ell+1} \rangle)$ is true if and only if

$$(Q_{\ell+1} y_{\ell+1} \leq a_{\ell+1})(Q_{\ell+2} y_{\ell+2} \leq t) \cdots (Q_k y_k \leq t)\psi(a_1, \ldots, a_\ell, y_{\ell+1}, \ldots, y_k, \vec{x}, \vec{X})$$

is true. We make these intuitions formal by setting the following conditions on α.

(a) For all $a_1, \ldots, a_k, b \leq t$, we have

$$\alpha(\langle \vec{a}, b \rangle) \;\leftrightarrow\; \psi(\vec{a}, \vec{x}, \vec{X}).$$

Note that the value b is just a placeholder and is not actually used.

(b) For all odd $\ell \leq k$ and all $a_1, \ldots, a_\ell \leq t$,

$$\alpha(\langle \vec{a} \rangle) \;\leftrightarrow\; [(a_\ell > 0 \wedge \alpha(\langle a_1, \ldots, a_{\ell-1}, a_\ell - 1 \rangle)) \vee \alpha(\langle \vec{a}, t \rangle)].$$

(c) For all even $\ell \leq k$ and all $a_1, \ldots, a_\ell \leq t$,

$$\alpha(\langle \vec{a} \rangle) \;\leftrightarrow\; [(a_\ell > 0 \supset \alpha(\langle a_1, \ldots, a_{\ell-1}, a_\ell - 1 \rangle)) \wedge \alpha(\langle \vec{a}, t \rangle)].$$

Definition The second order object α *canonically evaluates* $\phi(\vec{x}, \vec{X})$ provided that all the conditions (a)-(c) above hold. And, α *canonically verifies* $\phi(\vec{x}, \vec{X})$ provided that α canonically evaluates $\phi(\vec{x}, \vec{X})$ and $\alpha(\langle t \rangle)$ is true.

Note that "α canonically evaluates $\phi(\vec{x}, \vec{X})$" and "α canonically verifies $\phi(\vec{x}, \vec{X})$" are expressible as Π^b_1 formulas.

We extend the definitions of canonical evaluation and verification to $\Sigma^{1,b}_1$-formulas as follows.

Definition Let ϕ be a strict $\Sigma^{1,b}_1$-formula of the form $(\exists Y)C(\vec{x}, \vec{X}, Y)$, and let β be a second order object. Then α *canonically verifies that* β *witnesses* ϕ if and only α canonically verifies $C(\vec{x}, \vec{X}, \beta)$.

Theorem 7 *For $\phi(\vec{x}, \vec{X})$ a $\Sigma^{1,b}_0$-formula, S^1_2 proves*

"If α canonically verifies $\phi(\vec{x}, \vec{X})$, then $\phi(\vec{x}, \vec{X})$ is true."

For $\phi(\vec{x}, \vec{X})$ a strict $\Sigma^{1,b}_1$-formula $(\exists Z)\psi(\vec{x}, \vec{X}, Z)$, S^1_2 proves

"If α canonically verifies that β witnesses $\phi(\vec{x}, \vec{X})$, then $\psi(\vec{x}, \vec{X}, \beta)$ is true."

Proof (Sketch) This is proved using induction (outside S^1_2) on the number of quantifiers k. For $k = 0$, it is immediate by condition (a). For $k > 0$, suppose $\alpha(\langle t \rangle)$ holds. Arguing in S^1_2, use binary search or Δ^b_1-minimization to find the least $a_1 \leq t$ such that $\alpha(\langle a_1 \rangle)$ holds. By (b), this implies $\alpha(\langle a_1, t \rangle)$ holds. By the (dual of the) induction hypothesis, applied to the negations of α and the negation of $(\forall y_2 \leq t) \cdots (Q_k y_k \leq t)\psi(a_1, \vec{y}, \vec{x}, \vec{X})$, we have $\phi(\vec{x}, \vec{X})$ is true with y_1 set equal to a_1. □

We shall also need second order objects to canonically evaluate or canonically verify formulas ϕ that are not in prenex form. (In particular, such formulas seem to be unavoidable in the comprehension axioms.) For this, suppose ϕ is a non-prenex formula; for the next theorem, we use ϕ^* to denote any prenex form of ϕ; that is, ϕ^* is obtained from ϕ by pulling out quantifiers using prenex operations. We claim that S^1_2 is able to prove that the canonical verifications give the same results no matter what prenex form is used. The following theorem partially formalizes this claim.

Theorem 8 *Let ϕ and ψ be in prenex form with canonical evaluations given by α and β. Suppose γ is a canonical evaluation of $(\phi \wedge \psi)^*$, or of $(\phi \vee \psi)^*$, or of $(\neg\phi)^*$. Then γ canonically verifies the truth of the formula if and only α and β canonically verify ϕ and ψ, or one of α or β canonically verifies ϕ or ψ, or α does not canonically verify ϕ (respectively).*

Furthermore, for any fixed choice of formulas, this statement is provable in S^1_2.

The proof of the theorem is straightforward, as the canonical verification γ is expressible in terms of α and β in a very explicit way, based on the order in which prenex operations were applied. We omit the details.

4.2 The new-style witnessing theorems

Theorem 9 (Witnessing Theorem for U^1_2.)

a. *Suppose U^1_2 proves $(\exists y)\phi(y, \vec{a}, \vec{A})$ for ϕ a $\Sigma^{1,b}_0$-formula. Then there is a* PSPACE *oracle Turing machine M such that S^1_2 proves "If Y encodes a complete computation of $M^{\vec{A}}(\vec{a})$, then $\phi(out(Y), \vec{a}, \vec{A})$ is true."*

b. *Suppose U^1_2 proves $(\exists Z)\phi(Z, \vec{a}, \vec{A})$ for ϕ a $\Sigma^{1,b}_0$-formula. Then there is a* PSPACE *oracle Turing machine M such that S^1_2 proves "If W encodes a complete computation of $M^{\vec{A}}(\vec{a})$, then $Out(W) = \langle Y, Y'\rangle$ where Y canonically verifies that Y' witnesses $\exists Z\phi(x, Z, X)$ is true."*

The notation $M^{\vec{A}}(\vec{a})$ denotes that the machine M has as inputs the first order objects $\vec{a}$, and has oracle access to the second order objects $\vec{A}$. The notation $W = \langle Y, Y'\rangle$ is the ordinary pairing on second order objects, namely it means that $W(i)$ is true precisely for those i's of the form $\langle 0, y\rangle$ with $y \in Y$ or of the form $\langle 1, y'\rangle$ such that $y' \in Y'$.

The proof of Theorem 9 is based on the following witnessing lemma.

Theorem 10 (Witnessing Lemma for U^1_2.) *Suppose U^{1*}_2 proves a sequent $\Gamma \longrightarrow \Delta$ of strict $\Sigma^{1,b}_1$-formulas with free variables $\vec{a}, \vec{A}$. Let Γ be $\phi_1, \ldots, \phi_k$ and Δ be $\psi_1, \ldots, \psi_\ell$ with each ϕ_i equal to $(\exists Y_i)\phi'_i(\vec{a}, \vec{A}, Y_i)$ and each ψ_i equal to $(\exists Z_i)\psi'_i(\vec{a}, \vec{A}, Z_i)$. (Some of the quantifiers may be omitted.) Then there is a* PSPACE *oracle machine M such that S^1_2 proves:*

> *"If U_i canonically verifies that Y_i is a witness for ϕ_i for $i = 1, \ldots, k$, and if W encodes a complete computation of $M^{\vec{U},\vec{Y},\vec{A}}(\vec{a})$, then this computation of M outputs a first order $j = out(W) \in$*

$\{1, \ldots, \ell\}$ and encodes a second order output $Out(W) = \langle V, Z_j \rangle$ such that V canonically verifies that Z_j is a witness for Ψ_j."

Theorem 9 is an immediate consequence of Theorems 4, 7, and 10. The proof of Theorem 10, given in Section 4.3 below, uses induction on the number of lines in a U_2^{1*} sequent calculus proof which contains only strict $\Sigma_1^{1,b}$-formulas.

The witnessing theorem and lemma for V_2^1 are completely analogous to those for U_2^1.

Theorem 11 (Witnessing Theorem for V_2^1.)

a. *Suppose V_2^1 proves $(\exists y)\phi(y, \vec{a}, \vec{A})$ for ϕ a $\Sigma_0^{1,b}$-formula. Then there is an exponential time oracle Turing machine M such that S_2^1 proves "If Y encodes a complete computation of $M^{\vec{A}}(\vec{a})$, then $\phi(out(Y), \vec{a}, \vec{A})$ is true."*

b. *Suppose V_2^1 proves $(\exists Z)\phi(Z, \vec{a}, \vec{A})$ for ϕ a $\Sigma_0^{1,b}$-formula. Then there is an exponential time oracle Turing machine M such that S_2^1 proves "If W encodes a complete computation of $M^{\vec{A}}(\vec{a})$, then $Out(W) = \langle Y, Y' \rangle$ where Y canonically verifies that Y' witnesses $\exists Z \phi(x, Z, X)$ is true."*

Theorem 12 (Witnessing Lemma for V_2^1.) *Suppose V_2^1 proves a sequent $\Gamma \longrightarrow \Delta$ of strict $\Sigma_1^{1,b}$-formulas with free variables $\vec{a}, \vec{A}$. Let Γ be $\phi_1, \ldots, \phi_k$ and Δ be $\psi_1, \ldots, \psi_\ell$ with each ϕ_i equal to $(\exists Y_i)\phi_i'(\vec{a}, \vec{A}, Y_i)$ and each ψ_i equal to $(\exists Z_i)\psi_i'(\vec{a}, \vec{A}, Z_i)$. (Some of the quantifiers may be omitted.) Then there is an exponential time oracle machine M such that S_2^1 proves:*

"If U_i canonically verifies that Y_i is a witness for ϕ_i for $i = 1, \ldots, k$, and if W encodes a complete computation of $M^{\vec{U}, \vec{Y}, \vec{A}}(\vec{a})$, then this computation of M outputs a first order $j = out(W) \in \{1, \ldots, \ell\}$ and encodes a second order output $Out(W) = \langle V, Z_j \rangle$ such that V canonically verifies that Z_j is a witness for Ψ_j."

Theorem 11 follows from Theorems 7 and 12. Theorem 12 is proved in Section 4.3 below.

4.3 Proofs of the witnessing lemmas for U_2^1 and V_2^1

We now prove Theorem 10. Assume P is a U_2^{1*}-proof containing only strict $\Sigma_1^{1,b}$-formulas. The proof of Theorem 10 uses induction on the number of steps in the proof P, and splits into cases depending on the final inference

of P. There are two base cases where P consists a single sequent, with no inferences. The first is where P is a single initial sequent of the form $A \longrightarrow A$ where, w.l.o.g., A is atomic. This case is completely trivial of course. The second base case is when P consists of a single $\Sigma_0^{1,b}$-comprehension axiom of the form

$$\longrightarrow (\exists Z)(\forall y \leq t(\vec{x}))[y \in Z \leftrightarrow \phi(y, \vec{x}, \vec{X})]. \tag{3}$$

for ϕ bounded. We must describe a polynomial space Turing machine M that computes Z and a second order object V that canonically verifies that Z witnesses the truth of the comprehension axiom. We shall use informal arguments to describe M, but it will be clear that S_2^1 can formalize them in the sense that S_2^1 can prove that if a second order object encoding a complete computation of M is given, then the outputs $out(M)$ and $Out(M)$ correctly provide a canonical verification of the sequent (3). There is only a single formula, so $\ell = 1$ and of course $out(M) = 1$. The deterministic polynomial space algorithm for M is straightforward: for each value $y < t(\vec{x})$, $M^{\vec{X}}(\vec{x})$ computes the predicate V_y such that V_y canonically evaluates the truth of $\phi(y, \vec{x}, \vec{X})$. If V_y indicates $\phi(y, \vec{x}, \vec{X})$ is true, then $Z(y)$ is determined to be true; otherwise, $Z(y)$ is determined to be false. For each fixed value y, the V_y can be straightforwardly converted into a canonical verification (of a prenex form) of $y \in Z \leftrightarrow \phi(y, \vec{x}, \vec{X})$. Combining all these gives a canonical verification of (3).

The cases where the final inference of P is a weakening inference or an exchange inference are trivial. The cases where the final inference is a propositional inference are also rather trivial, but we do the case of $\wedge$:right to illustrate this. Suppose the final inference of P is

$$\frac{\phi_1, \ldots, \phi_k \longrightarrow \psi_1, \ldots, \psi_{\ell-1}, \psi'_\ell \qquad \phi_1, \ldots, \phi_k \longrightarrow \psi_1, \ldots, \psi_{\ell-1}, \psi''_\ell}{\phi_1, \ldots, \phi_k \longrightarrow \psi_1, \ldots, \psi_{\ell-1}, \psi'_\ell \wedge \psi''_\ell}$$

Note there are no second order quantifiers in ψ'_ℓ or ψ''_ℓ since P is free-cut free and thus all formulas in the proof are strict $\Sigma_1^{1,b}$. The induction hypothesis gives two Turing machines M' and M'' which satisfy Theorem 10 for the two upper sequents. We describe how to form a Turing machine M that fulfills the same condition for the lower sequent. The machine M has first order inputs $\vec{a}$ and uses oracles $\vec{U}, \vec{Y}, \vec{A}$. The machine M starts by forming a canonical evaluation of a prenex form of $\psi'_\ell \wedge \psi''_\ell$: this uses only inputs $\vec{y}$ and $\vec{A}$, and involves looping through all possible values for the bounded quantifiers in this formula, and uses polynomial space. If the canonical evaluation shows that $\psi'_\ell \wedge \psi''_\ell$ is true, M halts outputting the first order value ℓ

indicating the ℓth formula of the antecedent is true, and also outputting a second order object Z that canonically verifies $\psi'_\ell \wedge \psi''_\ell$. Otherwise, M canonically evaluates both ψ'_ℓ and ψ''_ℓ. By Theorem 8, at least one of these two formulas will be found to be false. Suppose, w.l.o.g., that ψ'_ℓ is false. In this case, M simulates M' and outputs whatever it outputs. Note that M' cannot report that ψ'_ℓ is true, so it must instead output some $j < \ell$, some Z_j, and some V which canonically verifies that Z_j is a witness for ψ_j. It is clear that S^1_2 can simulate this argument sufficiently well so as to prove that, if a complete computation of M is given as a second order object W, then it gives a canonical verification either for $\psi'_\ell \wedge \psi''_\ell$ or for some ψ_j with $j < \ell$.

Now suppose the final inference of P is a bounded first order $\exists$:right inference:

$$\frac{\Gamma \longrightarrow \Delta, \psi(s)}{s \leq t, \Gamma \longrightarrow \Delta, (\exists x \leq t)\psi(x)}$$

Note that ψ again has no second order quantifiers. The proof idea is somewhat similar to the case of $\wedge$:right just done. The induction hypothesis gives a Turing machine M' satisfying the witnessing conditions for the upper sequent. The desired Turing machine M acts as follows. It first builds a canonical evaluation for $(\exists x \leq t)\psi(x)$. If this finds the formula to be true, it outputs this fact along with the canonical verification. (As an alternate construction, it would also be enough to do this only if $\psi(s)$ is true. It must be the case that $s \leq t$ since the input to M includes a canonical evaluation of this atomic formula.) Otherwise, M continues to simulate M'. The output of M' must produce an index j for a formula in Δ along with a Z_j and a V which together witness and canonically verify the truth of the jth formula of Δ. Again, S^1_2 can prove that a complete computation by M produces the desired output.

Suppose the final inference of P is a bounded first order $\exists$:left inference:

$$\frac{a_0 \leq t, \phi_0(a_0), \Gamma \longrightarrow \Delta}{(\exists x \leq t)\phi_0(x), \Gamma \longrightarrow \Delta}$$

Here a_0 is an eigenvariable and does not occur in the lower sequent. Of course, ϕ_0 does not have any second order quantifiers. Let M' be given by the induction hypothesis. The desired machine M has among its inputs a canonical verification U_0 of the formula $(\exists x \leq t)\phi_0(x)$. M starts by extracting the least value for x for which U_0 has found that $\phi(x)$ is true, and sets a_0 equal to this value. (M can readily find a_0 either a polynomial

space linear search through all values of x, or by a polynomial time binary search as in the proof of Theorem 7.) Once a value for a_0 is determined, M continues by simulating M' and using its outputs.

The case where the final inference of P is a bounded first order $\forall$:right inference

$$\frac{a_0 \leq t, \Gamma \longrightarrow \Delta, \psi(a_0)}{\Gamma \longrightarrow \Delta, (\forall x \leq t)\psi(x)}$$

is similar to the previous two cases. Namely, the Turning machine M for the lower sequent starts by forming a canonical evaluation of $(\forall x \leq t)\psi(x)$. If this is found to be true, this is output by M. Otherwise, M finds a value for $a_0 \leq t$ that makes $\psi(a_0)$ false, and M continues by simulating the machine M' for the upper sequent with this value for a_0.

Suppose the final inference of P is a bounded first order $\forall$:left inference

$$\frac{\phi_0(s), \Gamma \longrightarrow \Delta}{s \leq t, (\forall x \leq t)\phi_0(x), \Gamma \longrightarrow \Delta}$$

The Turing machine M for the lower sequent is given among its inputs a second order object U_0 (an oracle) that canonically evaluates $(\forall x \leq t)\phi_0(x)$. It is easy to extract from U_0 another second order object U_0' that canonically evaluates $\phi_0(s)$. This is because $s \leq t$ must be true, and since we can define $U_0'(\langle a_1, \ldots, a_j \rangle)$ to equal $U_0(\langle s, a_1, \ldots, a_j \rangle)$. Let M' be the polynomial space Turing machine given by the induction hypothesis. The machine M acts by simulating M' using U_0' as the canonical verification for $\phi_0(s)$.

Suppose the final inference of P is a second order $\exists$:right inference

$$\frac{\Gamma \longrightarrow \Delta, \psi(A)}{\Gamma \longrightarrow \Delta, (\exists Z)\psi(Z)}$$

The second order variable A is not an eigenvariable, and so, w.l.o.g., appears in the lower sequent. Thus the desired machine M for the lower sequent takes the same inputs as the polynomial space machine M' given by the induction hypothesis for the upper sequent. The machine M' will output a canonical verification either of a formula in Δ or of $\psi(A)$. In the former case, M gives the same output as M'. In the latter case, M sets Z equal to A and outputs the canonical verification of $\psi(Z)$.

Suppose the final inference of P is a second order $\exists$:left

$$\frac{\phi(A), \Gamma \longrightarrow \Delta}{(\exists Y)\psi(Y), \Gamma \longrightarrow \Delta}$$

where now A is an eigenvariable and does not appear in the lower sequent. Let M' be the polynomial space machine given by the induction hypothesis; we must define the machine M for the lower sequent. One of the inputs to M is a second order Y along with a canonical verification U of $\phi(Y)$. The machine M runs by letting this Y be the value of the input A to M', using U as the canonical verification of $\phi(A)$, and then just running M'.

Now suppose the final inference of P is an $\mathrm{s}\Sigma^{1,b}_1$-repl-$\forall$ inference,

$$\frac{a \leq t, \Gamma \longrightarrow \Delta, (\exists X)\psi(X, a)}{\Gamma \longrightarrow \Delta, (\exists Z)(\forall x \leq t)\psi(\{z\}Z(\langle x, z\rangle), x)}$$

where a is an eigenvariable and may not occur in the lower sequent. The induction hypothesis gives a polynomial space Turing machine M' for the upper sequent. We form a new machine M which has the same inputs as M' except that a is not an input to M. The machine M runs as follows: it loops through all values of $a \leq t$, and simulates M' with each of these values for a. If, for any value a, M' indicates that a formula ψ_j in Δ is true and gives a witness Z_j and a canonical verification V for ψ_j, then M halts and outputs the same values j, Z_j and V. Otherwise, for each value of a, M' produces a second order X_a and a canonical verification V_a showing that X_a witnesses $(\exists X)\psi(X, a)$. When this happens for all values of a, the second order X_a's can be combined into a single second order Z defined so that $Z(a, z)$ holds iff $X_a(z)$ holds; furthermore, the canonical verifications V_a can be straightforwardly combined to give a canonical verification that Z is a witness for $(\exists Z)(\forall x \leq t)\psi(\{z\}Z(\langle x, z\rangle), x)$. It is clear that M is polynomial space bounded, since M' is.

Suppose the final inference of P is a cut inference,

$$\frac{\Gamma \longrightarrow \Delta, \chi \qquad \chi, \Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta}$$

Let M_1 and M_2 be the Turing machines given by the induction hypothesis for the left and right upper sequents, respectively. The machine M for the lower sequent is constructed as follows. It begins by running machine M_1, which takes the identical inputs as M. If M_1 finishes with a witness for one of the formulas in Δ, then M halts producing the same first- and second order outputs as M_1. Otherwise, M_1 outputs a pair of second order objects V and $Z_{\ell+1}$ such that V canonically verifies that $Z_{\ell+1}$ is a witness for χ. In this case, M then invokes M_2 with the intent of using V and $Z_{\ell+1}$ as inputs to M_2 that provide a witness and a canonical verification for the occurrence of χ in the antecedent of the upper right sequent. The only catch is that

M is allowed to use only polynomial space, and this is not sufficient space for M to save the exponentially long values of V and $Z_{\ell+1}$. Instead, as M simulates M_2, it recomputes the values of V and $Z_{\ell+1}$ as needed by running machine M_1 again. Since M_1 is deterministic, this always yields consistent values for V and $Z_{\ell+1}$. This allows M to use only polynomial space as, at any given point in time, M needs to remember only one configuration of M_1 and one configuration of M_2.

Finally, suppose the last inference of P is an $\mathrm{s}\Sigma_1^{1,b}$-LIND induction,

$$\frac{\chi(a_0), \Gamma \longrightarrow \Delta, \chi(a_0+1)}{\chi(0), \Gamma \longrightarrow \Delta, \chi(|t|)}$$

(It is slightly more convenient to use LIND instead of PIND, but the argument is essentially the same either way.) Let M' be the Turing machine given by the induction hypothesis for the upper sequent. The intuition is that we handle the induction hypothesis by treating it as $|t|-1$ many cuts, on the formulas $\chi(1)$, $\chi(2)$, ..., $\chi(|t|-1)$. This means that M is iterating computations of M'; however, the iterations are nested only to a depth $|t|$, so M needs to remember at most $|t|$ many configurations of M' at any given point in time. Since $|t|$ is polynomially bounded in terms of the lengths of the first order free variables, this means M uses only polynomial space.

For a bit more detail, let Δ have $\ell-1$ formulas. M starts by computing M' with a_0 set equal to 0, which we denote $M'[a_0 := 0]$, and potentially continues for $a_0 = 1, 2, \ldots, |t|-1$. If $M'[a_0 := i]$ yields a first order output $< \ell$, a witness of a formula in Δ has been obtained, and M can output this. Otherwise, $M'[a_0 := i]$ outputs first order output ℓ along with a witness for $\chi(|t|)$. If this happens with $i = |t|$, then the desired output has been obtained. For $i < |t|-1$, M must instead invoke $M'[a_0 := i{+}1]$ using the output of $M'[a_0 := i]$ as the second order witness and canonical verification for $\chi(i)$. As in the case of cut, the output of $M'[a_0 := i]$ is exponentially large, and cannot be written out in polynomial space. Instead, whenever, $M'[a_0 := i{+}1]$ queries its second order inputs for $\chi(i)$, M interrupts the computation of $M'[a_0 := i{+}1]$ and re-simulates the entire computation of $M'[a_0 := i]$. These recomputations must be carried out recursively, but only to a depth of $|t|$. At any given point in time, M needs to remember at most configurations for one invocation of each of $M'[a_0 := i]$, for $i = 0, \ldots, |t|$. It is straightforward to show that S_2^1 proves that a complete computation the algorithm M gives the correct output.

The above completes the proof of Theorem 10. The proof of Theorem 12 is mostly identical. The various cases, based on the final inference of

V_2^1-proof P, are essentially identical to the cases described above for Theorem 10. The cases of cut and induction merit more discussion however. In the setting of V_2^1, the exponential time machine M is allowed to use exponential space and this allows a simplification to be made in the construction of M. For the case where the final inference of P is cut, the output of the machine M_1 can be written down completely in M's memory as this requires 'only' exponential time and space. It is thus unnecessary to redo the computation of M_1 every time M_2 needs a value of M_1's second order output. Similar considerations apply to the case where the final inference of the V_2^1 is an IND induction inference. M now needs to do an exponentially long iteration; however, instead of recomputing values, M can just store them all in memory.

5 Local improvement principles

5.1 Definitions and theorems

The local improvement principles were defined by Kołodziejczyk, Nguyen, and Thapen [8] as an extension of the game principles of Skelley and Thapen [13]. The local improvement principle is specified by a set of contradictory conditions, so the local improvement principle states that it is always possible to find a counterexample to one of the conditions. Our definition of the local improvement principles below includes a minor, inessential change to the definition of [8] so as to make the score values a function of a single label instead of a function of the labels in a neighborhood.

Definition An instance of the local improvement principle consists of a specification of a directed acyclic graph G with domain $[a] := \{0, 1, 2, \ldots, a-1\}$ and polynomial time computable edges, an upper bound $b > 0$ on labels, an upper bound $c > 0$ on scores, an initial labeling function E, a wellformedness predicate *wf*, and a local improvement function I. These satisfy the following conditions.

a. The directed graph G is consistent with the usual $<$-ordering of its domain $[a]$, and has in- and out-degrees bounded by a fixed constant. The edges of G are specified by a polynomial time neighborhood function f. For each vertex $x \in [a]$, $f(x)$ outputs a set of vertices $y \in [a]$: the vertices $y < x$ (respectively, $y > x$) are the predecessors (respectively, the successors) of the vertex x. (The function f is constrained to output a valid of set predecessors and successors including respecting the degree bound, say by taking x to be an isolated point whenever

$f(x)$ gives an invalid output.) The *neighborhood* of x is the set containing x together with its successors and predecessors. The *extended neighborhood* of x is the union of the neighborhoods of the neighbors of x.

b. Vertices in G will be assigned a series of labels. A label is in the range $[0, b)$ and includes a *score* value s in the range $[0, c]$. The score value associated with the label on vertex x is polynomial time computable as a function of the label on x.[6] The polynomial time predicate *wf* determines whether a labeling of a neighborhood of x is *wellformed*. The inputs to the predicate *wf* are the vertices in the neighborhood and their labels. A labeling of vertices is *extended-wellformed* around x if it is wellformed on the neighborhood of every vertex y in the neighborhood of x.

c. The two functions E and I provide methods of assigning labels to vertices. To initialize the labels, the polynomial time function $E(x)$ assigns labels to vertices x with score 0 so that all neighborhoods have wellformed labelings. The improvement function I provides a method to replace a label with a label with a higher score value: I takes as input a vertex x and a wellformed labeling of the neighborhood of x, and provides a new label for x. Specifically, suppose s is even and that every predecessor of x has a label with score $s+1$ and that x and every successor of x has a label with score s; then I provides a new label for x with score $s+1$. Dually, suppose s is odd and that every successor of x has a label with score $s+1$ and that x and every predecessor of x has a label with score s; then I provides a new label for x with score $s+1$. In other cases, the I function is undefined. Furthermore, whenever I is defined and the labeling is extended-wellformed around x, then the labeling obtained by replacing the label on x with the the new label given by I is still extended-wellformed around x.

The intuition behind the local improvement function is that it provides labels with higher score values. Initially, all labels have score 0, but then sweeping forward through G allows scores to increase from even to odd values, and sweeping backwards allows scores to increase from odd to even values. The preservation of the extended-wellformed properties implies that

[6] This is slightly different from the convention of [8] which makes the score value a function of the labels in the neighborhood of x. They let the score value equal "$*$" if the labels do not constitute a wellformed local labeling. The difference in how scores are defined makes no difference to the complexity of the local improvement principle.

scores can increase without bound. This, however, contradicts the property that score values are $\leq c$. Thus, the local improvement conditions listed above are contradictory.

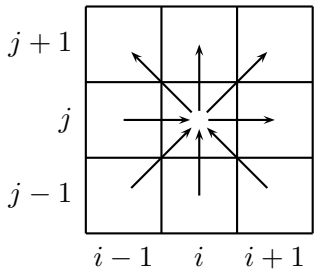
Definition A *solution* to an instance of the local improvement property consists of either: (a) An extended-wellformed labeling of a vertex x and its extended neighborhood where the local improvement function is defined but fails to provide a new label for x with the correct score value that preserves the extended-wellformed property, or (b) a neighborhood of a vertex x where the initialization function E fails to provide an extended-wellformed labeling with scores all equal to zero.

Note that any solution to the local improvement property is polynomial time checkable.

Definition An instance of the local improvement principle is formalized in bounded arithmetic by a constant (finite) degree bound for G, by first order values a, b, and c, and by explicitly polynomial time functions which describe G and compute the functions s, E, I and wf: it consists of the Σ^b_1 formula (with free variables a, b and c) that asserts that a solution exists. The notation LI denotes the set of Σ^b_1-formulas obtained from all instances of the local improvement principle. We use $\mathrm{LI}_{\log}$ to denote instances LI where c is a length, that is where $c = |c'|$ for some term c'. And, LI_k denotes instances of LI where $c = k$.

The *linear local improvement* principles LLI, $\mathrm{LLI}_{\log}$ and LLI_k are defined in the same way, but with G restricted to be a linear graph. That is, G has vertices $[a]$, and the edges of G are the directed edges $(i-1, i)$, for $0 < i < a$.

It is also useful to define "rectangular" local improvement principles. These are instances of LI or $\mathrm{LI}_{\log}$ where the underlying graph G has domain $[a] \times [a]$, each vertex (i, j) has up to four incoming edges, namely from the vertices $(i-1, j)$, $(i-1, j-1)$, $(i, j-1)$, and $(i+1, j-1)$. Thus, the edges involving (i, j) are as pictured:



except that any edges that would involve vertices outside the domain of G are omitted. We shall call instances of LI and $\mathrm{LI}_{\log}$ based on these rectangular graphs RLI and $\mathrm{RLI}_{\log}$. (These rectangular graphs were used by [8], although they did not use this terminology.)

Definition An NP search problem Q is specified by a first order sentence $(\forall x)(\exists y \leq t)\phi(y,x)$ with ϕ a Δ_1^b-formula w.r.t. S_2^1. A *solution* to $Q(x)$ is a value $y \leq t$ such that $\phi(y,x)$ holds. We denote this condition by $y = Q(x)$; note there may be multiple solutions y for a single input x.

The NP search problem Q is *total* provided that every x has at least one solution. It is *provably total* in a theory T provided $T \vdash (\forall x)(\exists y \leq t)\phi(y,x)$.

Any instance of the local improvement principle has a solution. This fact can be expressed as a $\forall\Sigma_1^b$-formula, and any solution can be verified in polynomial time. Thus the local improvement principles are total NP search problems.

Definition Suppose that $(\forall x)(\exists y \leq t)\phi(y,x)$ and $(\forall x)(\exists y \leq s)\psi(y,x)$ specify NP search problems, denoted Q_ϕ and Q_ψ. A *many-one reduction* from Q_ϕ to Q_ψ consists of a pair of polynomial time functions g and h such that whenever $y = Q_\psi(g(x))$, we have $h(y,x) = Q_\phi(x)$. We write $Q_\phi \leq_m Q_\psi$ to denote that there is a many-one reduction from Q_ϕ to Q_ψ.

A theory proves that $Q_\phi \leq_m Q_\psi$ provided that it proves

$$(\forall x)(\forall y)[y = Q_\psi(g(x)) \supset h(y,x) = Q_\phi(x)].$$

We can now state the results of [8] about the local improvement principles and the provably total NP search problems of U_2^1 and V_2^1.

Theorem 13 ([8]) *U_2^1 proves the linear, logarithmic local improvement principle* $\mathrm{LLI}_{\log}$. *Furthermore,* $\mathrm{LLI}_{\log}$ *is many-one complete, provably in S_2^1, for the provably total* NP *search problems of U_2^1; namely, if Q is a provably total* NP *search problem of U_2^1, then S_2^1 can prove that Q is many-one reducible to an* NP *search problem in* $\mathrm{LLI}_{\log}$.

Theorem 14 ([8]) *V_2^1 proves the local improvement principle* LI. *Furthermore,* LI *is many-one complete, provably in S_2^1, for the provably total* NP *search problems of V_2^1; namely, if Q is a provably total* NP *search problem of V_2^1, then S_2^1 can prove that Q is many-one reducible to an* NP *search problem in* LI.

The same results hold for RLI *in place of* LI.

We shall improve these results below by proving the following two theorems. The first theorem states that U_2^1 can also prove the LLI formulas. This is a somewhat surprising and unexpected result, since the straightforward algorithmic way to prove the local improvement principle LLI would be to iteratively define labels with increasing score values by sweeping back and forth across the linear graph G. If this is done deterministically, this could simulate c steps of a Turing machine computation, that is to say, it could simulate exponential time algorithms. This is (conjecturally) beyond the power of U_2^1 which can only define polynomial space predicates. However, as we shall see in Section 5.2, the LLI principle can instead be proved using only (nondeterministic) polynomial space computations.

Theorem 15 *U_2^1 proves the linear local improvement principle* LLI. *Furthermore,* LLI *is many-one complete, provably in S_2^1, for the provably total* NP *search problems of U_2^1; namely, if Q is a provably total* NP *search problem of U_2^1, then S_2^1 can prove that Q is many-one reducible to an* NP *search problem in* LLI.

The second part of Theorem 15 follows already from Theorem 13 since LLI contains $\mathrm{LLI}_{\log}$ as a special case. The proof of first part of Theorem 15 is given in Section 5.2 below.

Our new result for V_2^1 states that $\mathrm{LI}_{\log}$ is already strong enough to be many-one complete for the set of provably total NP search problems of V_2^1, and that the many-one completeness is provable over the base theory S_2^1.

Theorem 16 *V_2^1 proves the local improvement principle* $\mathrm{LI}_{\log}$. *Furthermore,* $\mathrm{LI}_{\log}$ *is many-one complete, provably in S_2^1, for the provably total* NP *search problems of V_2^1; namely, if Q is a provably total* NP *search problem of V_2^1, then S_2^1 can prove that Q is many-one reducible to an* NP *search problem in* $\mathrm{LI}_{\log}$.

The same results hold for $\mathrm{RLI}_{\log}$ *in place of* $\mathrm{LI}_{\log}$.

Theorem 16 will be proved in Section 5.4, using the rectangular local improvement principle $\mathrm{RLI}_{\log}$ for the many-one completeness. Of course, the first part of Theorem 16 follows already from Theorem 14.

It is interesting to observe that scores can be restricted further to a constant, for the price that the underlying graph structure will be more general than the linear structure in case of U_2^1, or than the rectangular structure in case of V_2^1. The best bound which we can obtain on scores is $c = 1$, namely with one round of improvement.

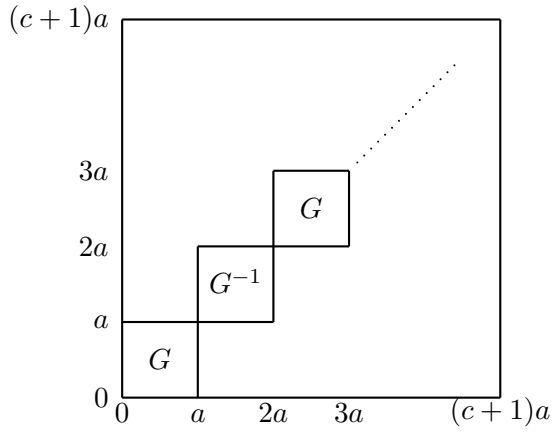


Figure 2: Structure of LI_1 game simulating an RLI game.

Theorem 17 (a) LI_1 *is many-one complete, provably in* S^1_2*, for the provable total* NP *search problems of* V^1_2*; in particular, if* Q *is a provably total* NP *search problem of* V^1_2*, then* S^1_2 *can prove that* Q *is many-one reducible to an* NP *search problem in* LI_1.

(b) RLI_1 *is many-one complete, provably in* S^1_2*, for the provable total* NP *search problems of* U^1_2*; namely,* U^1_2 *proves* RLI_1*, and if* Q *is a provably total* NP *search problem of* U^1_2*, then* S^1_2 *can prove that* Q *is many-one reducible to an* NP *search problem in* RLI_1.

Proof For part (a), using Theorem 14, it suffices to describe how to turn an RLI problem into an equivalent LI_1 problem. Let an RLI problem be given by $a, b, c, s(\cdot), wf, E(\cdot), I(\cdot)$, that is, the underlying graph G has domain $[a] \times [a]$, and each vertex (i, j) has up to four incoming edges, namely from the vertices $(i-1, j)$, $(i-1, j-1)$, $(i, j-1)$, and $(i+1, j-1)$. We think of G aligned in a way that the origin $(0, 0)$ is at the lower left corner. A simulating LI_1-problem can be constructed as follows: Let G^{-1} be formed from G by reversing the direction of all edges in G, and rotating the resulting graph 180 degrees. The lower left corner $(0, 0)$ of G becomes the upper right corner of G^{-1}, and the edges are again pointing upwards and (mostly) rightwards. We create $c+1$ many copies $G_1, \ldots, G_{c+1}$, alternating between G and G^{-1}, starting with G, and place them in ascending order on the diagonal of a $[a \cdot (c{+}1)] \times [a \cdot (c{+}1)]$ grid as shown in Figure 2.

The idea of the simulation is that instead of computing initial wellformed labels of score 0, and then sweeping back and forth to compute new well-

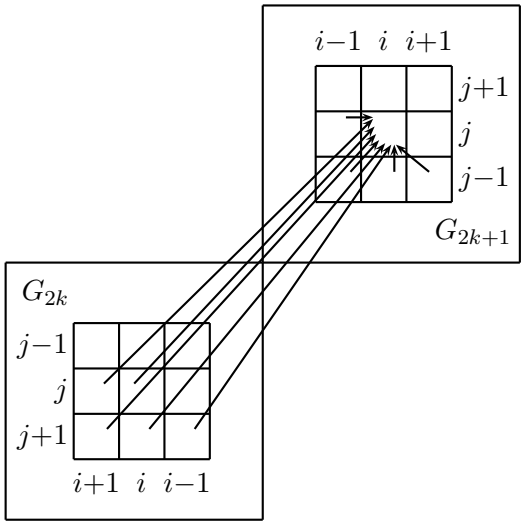


Figure 3: Structure of additional edges between G_{2k} and G_{2k+1} in LI_1 game which simulates an RLI game. Edges inside G_{2k+1} are unchanged.

formed labels of higher scores using I, we will just sweep once over the grid and produce a wellformed labeling so that G_k gets labels that encode the score k labels of the original RLI problem. For G_1 we use the initial labeling given by E, together with one round of local improvement under I. The initial labels for the graphs G_k with $k \geq 1$ may just be set to equal 0, where w.l.o.g. the label 0 is used only for these initial labels.

The LI_1 local improvement function replaces the initial labels 0 in the graph G_{k+1} with labels of score 1 which encode score $k+1$ labels as assigned in the instance of RLI. For this purpose, there are new edges added between G_k and G_{k+1} as shown in Figures 3 and 4, so that each node in G_{k+1} has new predecessors in G_k. The new local improvement function I for the instance of LI_1 will compute labels for nodes in G_{k+1} by simulating the (old) local improvement function from the instance of RLI using already computed labels of neighboring nodes from both G_k and G_{k+1}.

The additional edges between G_k and G_{k+1} are called *new* edges, and predecessors and successors based on them are called *new predecessors*, resp. *new successors*. Notions based on existing edges are dubbed *old*.

For setting labels in G_{2k+1}, see Figure 3. Suppose that every old predecessor of (i, j) in G_{2k+1}, that is $(i-1, j-1)$, $(i, j-1)$, $(i+1, j-1)$,

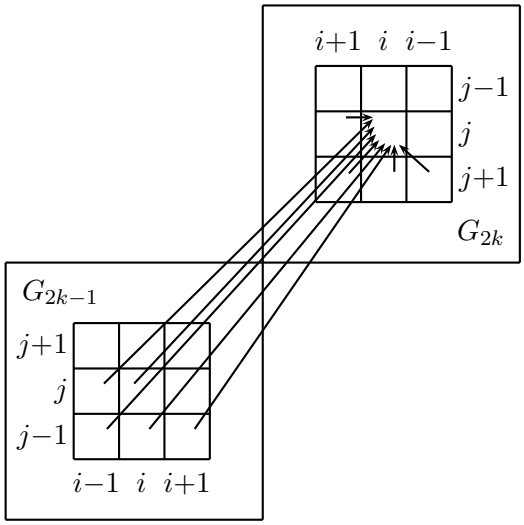


Figure 4: Structure of additional edges between G_{2k-1} and G_{2k} in LI_1 game which simulates an RLI game. Edges inside G_{2k} have been reversed for the instance of LI_1.

and $(i-1,j)$ in G_{2k+1}, has a label with score $2k+1$, and that every new predecessor of (i,j) in G_{2k+1}, that is (i,j), $(i+1,j)$, $(i-1,j+1)$, $(i,j+1)$, and $(i+1,j+1)$ in G_{2k}, has a label with score $2k$. Then I provides a new label for (i,j) in G_{2k+1} with score $2k+1$. Thus, the neighborhood on which I bases its computation, is formed from $(i-1,j-1)$, $(i,j-1)$, $(i+1,j-1)$, $(i-1,j)$ in G_{2k+1}, and (i,j), $(i+1,j)$, $(i-1,j+1)$, $(i,j+1)$, $(i+1,j+1)$ in G_{2k}.

Labels in G_{2k} are set dually; see in Figure 4. Consider a node (i,j) in G_{2k}. Suppose that every predecessor of (i,j) in the same graph G_{2k}, that is $(i+1,j+1)$, $(i,j+1)$, $(i-1,j+1)$, and $(i+1,j)$ in G_{2k}, has a label with score $2k$ — these are nodes that were (old) successors of (i,j) in the instance of RLI, but have become (new) predecessors of (i,j) in the instance of LI_1. Also suppose that every new predecessor of (i,j) in G_{2k}, that is (i,j), $(i-1,j)$, $(i+1,j-1)$, $(i,j-1)$, and $(i-1,j-1)$ in G_{2k-1}, has a label with score $2k-1$. Then I provides a new label for (i,j) in G_{2k} with score $2k$. Here, the neighborhood on which I bases its computation, consists of $(i+1,j+1)$, $(i,j+1)$, $(i-1,j+1)$, $(i+1,j)$ in G_{2k}, and (i,j), $(i-1,j)$, $(i+1,j-1)$, $(i,j-1)$, $(i-1,j-1)$ in G_{2k-1}.

The wellformedness predicate *wf* for the new LI_1 problem is defined with the aid of the predicate *wf* for the instance of RLI. On G_1 we use *wf* directly based just on old edges. Consider a vertex (i, j) in G_{2k+1}; we call this vertex x. The vertex x has the nine incoming edges as shown in Figure 3, namely, one from (i, j) in G_{2k} plus eight additional edges. In addition, there are the corresponding nine outgoing edges. If any of the predecessors of x have label 0, then the neighborhood of x is defined to be wellformed provided that x, and all of its successors also have label 0. Otherwise, the predecessors of x all have labels different from 0. Then, if x itself has label 0, then the neighborhood of x is wellformed provided that the labels on the vertex (i, j) in G_{2k} and the eight other predecessors of x are wellformed according to the criteria of the RLI instance. On the other hand, if x does not have label 0, then the neighborhood of x is wellformed provided that the labels on x and its eight predecessors are wellformed according to the criteria of the RLI instance. The wellformedness predicate is defined similarly for vertices in G_{2k}.

It is not hard to verify in S_2^1 that the above gives a faithful translation from the RLI problem to an LI_1 problem.

We now turn to part (b). By Theorem 19 proved below, RLI_1 is provable in U_2^1. For the completeness of RLI_1 under many-one reductions, in light of Theorem 13, it suffices to describe how to turn an LLI problem into an equivalent RLI_1 problem. Let an LLI problem be given by $a, b, c, s(\cdot), wf, E(\cdot), I(\cdot)$, that is, the underlying graph G has vertices $[a]$, and the edges of G are the directed edges $(i-1, i)$ for $0 < i < a$. We think of G as a horizontal line with 0 to the left, and edges pointing to the right. A simulating RLI_1 problem can be constructed as follows: Let G^{-1} be the inverse of G, that is 0 is now to the right and edges are pointing to the left.

The main idea is to again create $c + 1$ many copies alternating between G and G^{-1}, and stack them vertically to form a $[a] \times [c + 1]$ rectangular grid. Instead of computing initial wellformed labels of score 0, and then sweeping back and forth on G to compute new wellformed labels of higher scores using I, we want to just sweep once over the grid and produce a wellformed labeling so that the k-th copy of G is given labels that encode the score k labels of the LLI problem. However, as we alternate between the stacked copies of G and G^{-1}, we need, between any two alternations, $a - 1$ many additional lines which allow us to invert the positions of previously computed labels. Thus, the resulting graph has dimension $[a] \times [1 + a \cdot c]$.

The idea for inverting label positions using additional lines is as follows – see Figure 5 for an example. We can think of the original line of label

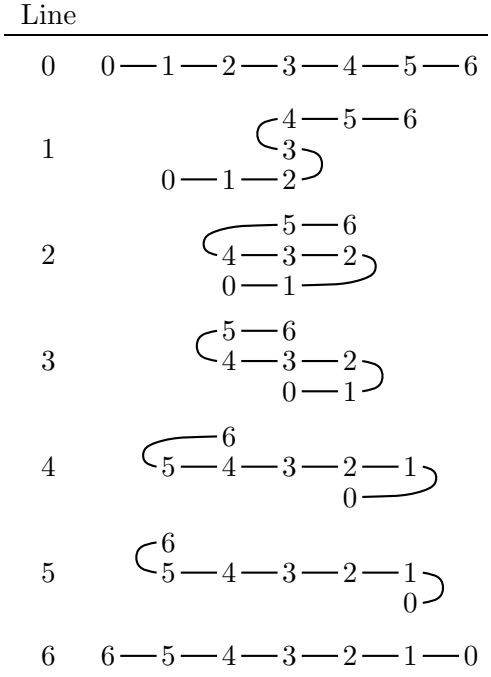


Figure 5: Inverting a line with 7 label positions using 6 additional lines.

positions as a rope stretching out horizontally in the plane. We transform the rope to first form a little "s" in the middle. Then, keeping the position of the middle point of the rope fixed, we stretch the two curves of the "s" horizontally to the sides, until eventually the rope is stretched out again, this time in the opposite direction.

During this process, a vertical line at an arbitrary horizontal position will intersect the rope at most three times. Thus, the labels at additional lines in our game graph, when imitating the just described transformation, will have to store at most 3 pieces of label information: one for those labels who have found their new position; one for those who are moving left to right, and one for those who are moving right to left. In addition, it is convenient to store a score value which measures how far the inversion has progressed.

It is obvious that a rectangular structure on the new $[a] \times [1 + a \cdot c]$ grid is sufficient to imitate the above described process. It is then straightforward to define *wf*, $E(\cdot)$ and $I(\cdot)$ which exactly describe this process – details are left to the reader. □

Corollary 18 *Over the base theory* S_2^1*,*

(a) *The principles* LLI*,* $\mathrm{LLI}_{\log}$ *and* RLI_1 *are equivalent.*

(b) *The principles* LI*,* $\mathrm{LI}_{\log}$*,* LI_1*,* RLI*, and* $\mathrm{RLI}_{\log}$ *are equivalent.*

Proof Part (a) is an immediate consequence of Theorems 13, 15, and 19, due to the fact that the LLI conditions are NP search problems. Part (b) is likewise an immediate consequence of Theorems 14, 16, and 17 and the fact that only $\mathrm{RLI}_{\log}$ will be used for the proof of Theorem 16. □

5.2 Proof of Theorem 15

The intuition behind the proof of LLI is based on the following exponential time algorithm: First set all vertices $x \in [a]$ in the linear directed graph to have the initial labels with score zero given by $E(x)$. Then, sweep back-and-forth through the vertices in linear order, alternating scans in left-to-right order (from 0 to $a - 1$) with scans in right-to-left order (from $a - 1$ to 0). Each time a vertex x is processed, its prior label, with score s, is replaced by a new label, with score $s + 1$. For even values of s, this occurs while sweeping left-to-right and for odd values of s, it occurs while sweeping from right-to-left. Up to c scans are performed, by which time a contradiction to the local improvement conditions must be found; namely, either by reaching

a point where the improvement function I fails to produce an appropriate value or by obtaining a score value $> c$.

This algorithm calculates a values of $E(x)$ and invokes the improvement function $a \cdot c$ times. Since a and c are arbitrary first order objects (not lengths), this takes exponential time. Worse, the algorithm stores the current label values for all $x \in [a]$ and this requires exponential space. Thus, the algorithm is in exponential time but not in polynomial space, and the theory U^1_2 cannot formalize the algorithm directly, unless PSPACE equals exponential time. To circumvent this barrier, we will use a non-deterministic polynomial space algorithm instead. The idea behind the NPSPACE algorithm is simple: rather than storing the labels on all vertices $x \in [a]$, it merely nondeterministically guesses them as needed. This of course does not give the "correct" labels; nonetheless, it will be sufficient to prove the theorem.

We start by describing the NPSPACE algorithm M. The algorithm M sweeps alternately from left-to-right and right-to-left setting labels on vertices x. When M is about to process the vertex numbered x, during a left-to-right sweep, it knows labels for the vertices $x-2$, $x-1$, x, $x+1$, and $x+2$ with score values $s+1$, $s+1$, s, s, and s respectively. Since it is a left-to-right sweep, the value s is even. In addition, the five known labels are wellformed around the three vertices $x-1$, x, and $x+1$; namely, according to the predicate *wf*, the labels are wellformed in the neighborhood of $x-1$, in the neighborhood of x, and in the neighborhood of $x+1$. Since the graph is linear, each neighborhood contains three vertices; for example, the neighborhood of $x-1$ contains the vertices $x-2, x-1, x$. M uses the local improvement function I to obtain a new label for vertex x with score value $s+1$. If I produces a label with score value unequal to $s+1$ or if the new label for x causes any of the three vertices $x-1$, x, and $x+1$ to no longer have neighborhoods with wellformed labels, then M halts in a rejecting state. Otherwise, M needs to step one vertex rightward, and for this M discards (forgets) the label for $x-2$ and needs to set a label value for $x+3$. If $s=0$, the label for $x+3$ is set to equal $E(x+3)$. For $s>0$, M merely non-deterministically guesses a label for $x+3$ with score value s. If this label for $x+3$ does not have score value s, or it makes the labels of the vertices $x+1, x+2, x+3$ in the neighborhood of $x+2$ not be wellformed, then M halts in a rejecting state.[7] Otherwise, M has finished processing vertex x and it proceeds to $x+1$, now with labels for $x-1$, x, $x+1$, $x+2$, and

[7] As we shall see, this is the "bad" case that we are trying to avoid. It would not happen if M remembered labels from the previous scan instead of just guessing them.

x+3.

The algorithm for sweeping right-to-left is entirely dual. In this case, s is odd. When updating the label for vertex x, M knows labels for the vertices x−2, x−1, x, x+1, and x+2 with score values s, s, s, s+1, and s+1. In the next step, to update the label for vertex x−1, M forgets the label for x+2 and has nondeterministically chosen a label for x−3.

At the ends of the linear order, the obvious modifications are made. If $x = a - 1$ is the rightmost vertex, then there is no vertex x+1 or x+2. Or if $x = a$−2, there is no vertex x+2. Likewise at $x = 0$, there is no vertex x−2 or x−1, and at $x = 1$, no vertex x−2. These missing vertices cause no problem: there are fewer neighborhoods in which labels must be wellformed, and their labels are not needed by the improvement function. When reaching $x = a$−2 in a left-to-right scan, M acts purely deterministically as there is no new vertex x+3 which needs a label. When reaching $x = a$−1, M initially knows labels for a−3, a−2, and a−1 with score values s+1, s+1, and s. It updates the label on vertex a−1 to have score s+1 (unless it rejects), and switches the scan order to right-to-left while staying at the same vertex $x = a$−1. In the next step, as the first step in the right-to-left scan, it invokes I to update the label of a−1 to have score value s+2, or rejects if I fails to provide such a label. M then rejects if s+2 $> c$. Otherwise it nondeterministically chooses a label for a−4: if this has the wrong score or fails the wellformedness property, M rejects; otherwise, it proceeds one vertex leftward to update the label on vertex $x = a$−2.

The vertices $x = 0$ and $x = 1$ at the end of right-to-left scan are handled dually.

As defined, any execution of M leads to rejection. There are three possible reasons for rejection: (a) The local improvement function I or the initialization function E may give a label with an incorrect score value or which violates the wellformedness property. (b) The nondeterministic guess of the next vertex's label (on vertex x+3 or x−3 for rightward or leftward scans, resp.) may give an incorrect score or violate the wellformedness property. (c) A score value may increase to $> c$. In either case (a) or (c) occurs, then M has found a point where the LLI conditions are falsified; that is, it has found a solution to the LLI problem. In case (b), no such solution is found. Our goal, thus, is to prove (arguing in U_2^1) that M has some computation that fails for reason (a) or (c).

The steps of a (nondeterministic) computation of M can be indexed with pairs $\langle x, s\rangle$, where x is the vertex number and s the score value. The evenness/oddness of s determines if the sweep is currently left-to-right or right-to-left. A pair $\langle x, s\rangle$ is *M-reachable* if there is some computation of M

that reaches the point where it is considering $\langle x, s\rangle$ and trying to find a new label for x with score $s+1$. The property of $\langle x, s\rangle$ being reachable is an NPSPACE, and thus a PSPACE, property. By induction (IND) on the PSPACE property of reachability, there must be some maximum value s_0 such that some $\langle x, s_0\rangle$ is reachable. If $s_0 = 0$ or $s_0 = c$, then M rejects at this step for one of the reasons (a) or (c). So we may assume $0 < s_0 < c$. Without loss of generality, s_0 is odd, so M is currently scanning right-to-left. Again using IND induction, there must be some minimum x_0 such that $\langle x_0, s_0\rangle$ is reachable.

Any computation of M that reaches $\langle x_0, s_0\rangle$ ends up with labels for $x_0{-}2$, $x_0{-}1$, x_0, $x_0{+}1$, $x_0{+}2$ with scores s_0, s_0, s_0, $s_0{+}1$, $s_0{+}1$. By choice of $\langle x_0, s_0\rangle$, M rejects while executing this step. If this happens because the improvement function fails to produce a new label for x_0 with score $s_0{+}1$ which satisfies the wellformedness properties, then it is the desired failure of type (a). Otherwise, M successfully finds a new label for x with score $s{+}1$ and with the necessary wellformedness properties, but there is no possible value for a label on $x-3$ with score s such that the labeling around $x-2$ is wellformed. We need to prove that this latter case, (b), can be avoided.

Definition We continue to assume s_0 is odd. A (non-deterministic) computation of M is s_0-*consistent at* x provided that during the computation of M, there are label values u and v for vertices $x-2$ and $x-1$ such that u and v both have score s_0, and such that the vertices $x-2$ and $x-1$ have the score s_0 labels u and v at both step $\langle x, s_0{-}1\rangle$ and step $\langle x, s_0\rangle$.

In other words, the same labels u and v are used for $x-2$ and $x-1$ in the right-to-left scan that raises score values from s_0 to $s_0{+}1$ as in the previous left-to-right scan that raised scores from $s_0{-}1$ to s_0.

Clearly, any computation of M that reaches the s_0 scan is s_0-consistent at $x = a{-}1$, since the labels on $a{-}2$ and $a{-}3$ do not change when switching over from a left-to-right scan to a right-to-left scan.

For a given vertex x, the question of whether there exists a computation of M which is s_0-consistent at x can be answered by an NPSPACE, hence a PSPACE, algorithm. Thus, by induction (IND), there is a minimum value x_1 such that there is a computation U of M which is s_0-consistent at x_1. If the computation U rejects because of reason (a) at step $\langle x_1, s_0\rangle$, then we are done. Otherwise, we claim that M can continue with the computation U for an additional step so as to be s_0-consistent at $x_1{-}1$. Namely, after obtaining an appropriate new label for x_1 with score $s{+}1$, M existentially chooses the label on $x_1{-}3$ to be exactly the same label as in the previous scan (except

this is not done if $x_1 < 3$). By choice of x_1, the vertices $x_1{-}2$ and $x_1{-}1$ already have the same label as in the previous scan, thus the labels in the neighborhood of $x_1{-}2$ are again well formed since they were well formed in the previous scan. This contradicts the choice of x_1, and completes the proof of Theorem 15.

5.3 RLI_1 is provable in U_2^1

A similar argument to the one given above shows that RLI_1 is provable in U_2^1.

Theorem 19 $U_2^1 \vdash \mathrm{RLI}_1$.

Proof (Sketch) The idea for the NPSPACE algorithm solving RLI_1 is to do a similar thing as for LLI, where row numbers in RLI_1 now play the role of scores in LLI. The algorithm M sweeps always left-to-right: it starts setting labels in the first row from left to right, then in the second row from left to right, etc. It always guesses all necessary "previously computed" labels from the previous rows. Thus, when M is about to process vertex $\langle x, y\rangle$, it knows labels for vertices $\langle x{-}2, y\rangle$, $\langle x{-}1, y\rangle$, $\langle x{-}2, y{-}1\rangle$, $\langle x{-}1, y{-}1\rangle$, $\langle x, y{-}1\rangle$, $\langle x{+}1, y{-}1\rangle$, $\langle x{+}2, y{-}1\rangle$, $\langle x{-}2, y{-}2\rangle$, $\langle x{-}1, y{-}2\rangle$, $\langle x, y{-}2\rangle$, $\langle x{+}1, y{-}2\rangle$, and $\langle x{+}2, y{-}2\rangle$, all with score 1. These labels together with the labels of score 0 given by E for the other vertices in the extended neighborhood of $\langle x, y\rangle$ are extended wellformed. M uses the local improvement function I to obtain a new label for vertex $\langle x, y\rangle$ with score 1. If I produces a label with score value unequal to 1 or if the new label for x causes the labels of the extended neighborhood of $\langle x, y\rangle$ to not be extended wellformed, then M halts in a rejecting state. Otherwise, M needs to step to the next vertex, either one vertex to the right, or to the leftmost vertex of the next row. In the former case, i.e. when $x{+}1 < a$, M discards (forgets) labels for $\langle x{-}2, y\rangle$, $\langle x{-}2, y{-}1\rangle$ and $\langle x{-}2, y{-}2\rangle$, and guesses labels for $\langle x{+}3, y{-}1\rangle$ and $\langle x{+}3, y{-}2\rangle$. In the latter case, M discards (forgets) all labels and guesses labels for $\langle 0, y\rangle$, $\langle 1, y\rangle$, $\langle 2, y\rangle$, $\langle 0, y{-}1\rangle$, $\langle 1, y{-}1\rangle$, and $\langle 2, y{-}1\rangle$. If any of these labels do not have score 1, or they make the labels in the neighborhood of $\langle x + 1, y\rangle$ in the former case, resp. the neighborhoods of $\langle 0, y + 1\rangle$ in the latter case, not be extended wellformed, M halts in a rejecting state.

The notion of $\langle x, y\rangle$ being *M-reachable* is defined as before. A computation being *consistent* at $\langle x, y\rangle$ is also defined similarly; namely, the labels that are used when setting the label on vertex $\langle x, y\rangle$ must coincide with the labels that were guessed or computed when setting labels in row $y - 1$.

Then a similar argument as before shows that this computation leads to a rejection according to (a) or (c), provably in U^1_2. □

We have not been able to characterize the strength of RLI_2 or, more generally, the strength of RLI_k for constant $k \geq 2$. In particular, we do not know if they are provable in U^1_2, nor if they are many-one complete for the provably total NP search problems of V^1_2. It is also possible they are intermediate in strength.

5.4 Proof of Theorem 16

Our proof of Theorem 16 is based on the constructions of [8], but avoids using T^1_2 as a base theory, and correspondingly avoids a detour through polynomial local search (PLS) problems. We use instead the Witnessing Theorem 11 which has S^1_2 as a base theory and thus use a polynomial time computation in place of a PLS computation.

We need to prove that $\mathrm{RLI}_{\log}$ is many-one complete for the provably total NP search problems of V^1_2. We will prove this in the following strong form, that applies to "type-2" NP search problems that have a second order input X in addition to a first order input x.

Theorem 20 *Suppose ϕ is Δ^b_0 and $(\forall x)(\forall X)(\exists y)\phi(y, x, X)$ is provable in V^1_2. Then, there is a many-one reduction from the* NP *search problem defined by $(\exists y)\phi(y, x, X)$ to an instance of* $\mathrm{RLI}_{\log}$*. Furthermore, the many-one reduction is provably correct in S^1_2.*

Proof Theorem 11 implies that there is an exponential time oracle Turing machine M such that S^1_2 proves:

(A) If Y encodes a complete computation of $M^X(x)$, then $\phi(out(Y), x, X)$ is true.

We make some simplifying assumptions about how $M^X(x)$ runs; namely, we assume that M uses a single tape, and that this tape contains three "tracks": the first track is read-only and holds the input x padded with blanks, the second track is also read-only and holds the input X, and the third track is read-write, initially blank. (Equivalently, M has three tapes, but the three tape heads move in lockstep.) Also w.l.o.g., Y encodes the computation in some simple, direct fashion; namely, Y can be taken to be the bit-graph of the function H that maps a pair $\langle p, t\rangle$ to the tape contents at position p at time t, the head position at time t, and the state of the machine at time t. With these conventions, writing (A) out in more detail gives that S^1_2 proves

(B) $\phi(out(Y), x, X)$ $\vee$ ("$\exists$ a place in Y where Y fails to satisfy the local conditions of being a correct computation of $M^X(x)$").

Or even more explicitly, S^1_2 proves

(B') $(\exists y)(y = out(Y) \wedge \phi(y, x, X))$, or
$(\exists p)(\exists t)$[the values given by Y for $H(p, t+1)$, $H(p-1, t)$, $H(p, t)$, and $H(p+1, t)$ do not code consistent information for the computation], or
$(\exists p)[H(p, 0)$ does not equal valid initial tape contents and state for position p at time 0].

Therefore, by the relativized witnessing theorem for S^1_2, there is a polynomial time function f, which takes x as input and uses X and Y as oracles, and produces values for y, p and t satisfying one of the disjuncts of (B'). Without loss of generality, the function f is computed by a clocked Turing machine, so S^1_2 proves its runtime is polynomially bounded. Because of the assumption that Y codes the bit-graph of H, we can view f as asking queries to the function H. That is, rather than querying truth values of $Y(i)$, f makes queries $q = \langle p, t\rangle$ to H and receives for an answer the value $r = H(p, t)$ giving the tape contents at position p at time t, the state at time t, and the tape head position at time t. W.l.o.g., S^1_2 proves that if f outputs values p, t satisfying (B'), then f has actually queried the four values $H(p, t+1)$, $H(p-1, t)$, $H(p, t)$, and $H(p+1, t)$, and that if it outputs a value p, then it has queried $H(p, 0)$.

We will use the computation of $f^{X,Y}(x)$ to set up an instance of $\mathrm{RLI}_{\log}$. We are particularly interested in tracking the queries that f makes to H. For fixed x, X, let $q_i = \langle p_i, t_i\rangle$ be the i-th query made during the computation of $f^{X,Y}(x)$ and $r_i = H(p_i, t_i)$ be the answer received. Since f is polynomial time, w.l.o.g., $i = 1, 2, \ldots, p(|x|)$ for some polynomial p. Note that $p(|x|)$ counts only queries to Y, and we do not count the queries made to X.

We next define the instance of $\mathrm{RLI}_{\log}$. The intent is that, provably in S^1_2, any solution to the $\mathrm{RLI}_{\log}$ problem will give a computation of f satisfying (B'). S^1_2 will be able to prove that there are no witnesses p, t or p satisfying the second or third disjunct of (B'); hence, the only possibility for a solution is a value y such that $\phi(y, x, X)$ holds. This will suffice to prove Theorem 16.

Our proof uses an amalgamation of techniques from [8]. Set $c := 2p(|x|)+1$ equal to one more than twice the number of queries f makes to H. Let P be the space used by $M^X(x)$, and T the time. The directed graph G will be the

rectangular graph $[P] \times [T]$. The edges of G are as described earlier, namely, the up to four edges incoming to (p,t) come from the vertices $(p-1,t)$, $(p-1,t-1)$, $(p,t-1)$, and $(p+1,t-1)$.

The vertices of G will be labeled with sequences. The initialization function E labels with each vertex (p,t) with the empty sequence $\langle\rangle$. This has score value 0. The empty sequence is the only label with score 0. The valid labels for a vertex (p,t) with odd score $2s+1$ are sequences of the form

$$\langle \beta_{p,t}, q_1, r_1, q_2, r_2, \ldots, q_s, ? \rangle$$

and

$$\langle \beta_{p,t}, q_1, r_1, q_2, r_2, \ldots, q_s, r_s \rangle.$$

Here the first entry $\beta_{p,t}$ is intended to equal the value of $H(p,t)$. The q_i values are intended to be queries to H, so $q_i = \langle p_i, t_i \rangle$; the r_i values are intended to be answers to the queries, except the special symbol ? indicates that r_i is not known yet. Note that in general p_i and t_i are unequal to p and t.

The valid labels for (p,t) with even score $2s$ have the form

$$\langle \beta_{p,t}, q_1, r_1, q_2, r_2, \ldots, q_s \rangle.$$

Note that the score associated with a label is always the length of the sequence coded by the label. The score values are $\leq 2p(|x|)+1$ and the sizes of the intended entries in the sequences are polynomially bounded, hence there is a term $b = b(x)$ bounding the values of labels.

The wellformedness property, *wf*, applies to labels in the neighborhood N of a vertex (p_0,t_0). In order to be wellformed, the following four sets of conditions must be satisfied.

First the labels in the neighborhood N must agree on q_j and r_j values, and each q_j must be the j-th query that is made during the computation $f^{X,Y}(x)$ when the r_j's are the query responses. Specifically, if a non-? value for r_j appears in a label, then there cannot be a different non-? value for r_j in any label in N. Since the values of r_i are the same in all labels in N, the values of q_i must also be the same in all labels in N. This is because the q_i values are determined by the computation of $f^{X,Y}(x)$ when queries to H are answered with the values r_i.

Second, the q_j, r_j values must be consistent with the $\beta_{p,t}$ values in the following way: if a vertex in the neighborhood N has coordinates $\langle p', t' \rangle$ and thus its label has first entry $\beta_{p',t'}$, and if some (possibly different) vertex $x = \langle p'', t'' \rangle$ in N has an entry $q_j = \langle p', t' \rangle$ then, if r_j is present in x's label, it must satisfy:

- If r_j is the last entry of x's label, and if there is no path in the directed graph G from (p', t') to (p'', t''), then r_j equals ?.
- Otherwise r_j must equal $\beta_{p',t'}$.

Third, the $\beta_{p,t}$ values should, at least locally, behave like valid values for $H(p,t)$; that is to say, the $\beta_{p,t}$ values must be consistent with some potential computation of $M^X(x)$. Except for labels with score zero, all nine vertices in the neighborhood have $\beta_{p,t}$ values, and these values must represent some locally consistent computation of M: in particular, they should contain the correct values in the x- and X-tracks of the tape, and the values for the tape head position, the current state, and the contents of the third tape track must be consistent with the transition relation of the Turing machine M.

Fourth, the wellformedness property *wf* requires a somewhat subtle restriction on the values r_i that prevents them from recording a solution to the second or third disjunct of (B'). Namely, there must not be any $q_i = \langle p, 0\rangle$ where r_i does not equal the correct value $H(p,0)$ describing the initial tape configuration at position p at time 0. In addition, there must not be four query values $q_{i_1} = \langle p-1, t-1\rangle$, $q_{i_2} = \langle p, t-1\rangle$, $q_{i_3} = \langle p+1, t-1\rangle$, and $q_{i_4} = \langle p, t\rangle$ such that r_{i_1}, r_{i_2}, r_{i_3}, and r_{i_4}, when interpreted as values for $H(p-1,t-1)$, $H(p,t-1)$, $H(p+1,t-1)$, and $H(p,t)$, give values that would witness the second conjunct of (B').

This completes the definition of the wellformedness condition *wf*. Note that *wf* is a polynomial time computable function of the (up to) nine labels in a neighborhood, the values of p_0 and t_0 and the input x, using X as an oracle.

We next define the local improvement function I. There are four cases to consider. First, when increasing scores from 0 to 1, the function I must compute the value $\beta_{p,t}$. For t greater than zero, this is determined from $\beta_{p-1,t-1}$, $\beta_{p,t-1}$, $\beta_{p+1,t-1}$ by M's transition relation. For t equal to zero, $\beta_{p,t}$ is just the initial configuration of M for tape cell p; the appropriate values for the read-only tracks are computed by using the input x or the oracle X. Second, consider the case where scores are raised from an even value $2s > 0$ to an odd value $2s+1$. This represents the case where we are trying to load the answer r_s to the query $q_s = \langle p_s, t_s\rangle$. If (p,t) is not reachable from (p_s, t_s) in the directed graph G, then I just sets r_s equal to ?, leaving the rest of the entries in the label unchanged. If $(p,t) = (p_s, t_s)$, I sets r_s to equal $\beta_{p,t}$. Otherwise, r_s is merely copied from a non-? r_s value of $(p-1,t)$ or $(p+1,t-1)$; by wellformedness, these two values will agree if they are both present and not equal to ?. Third, when increasing a score value from an odd value $2s-1$ to an even value with $p < P$ or $t < T$, the function I

is merely propagating a query value q_s backwards through G: the value q_s for the label on (p, t) is just copied from the q_s value of either $(p + 1, t)$ or $(p - 1, t + 1)$. Fourth, and finally, we define the local improvement function for updating the upper right vertex $(P-1, T-1)$ from an odd value $2s - 1$ to $2s$. For this vertex, the local improvement function I simulates $f^{X,Y}(x)$: Whenever f makes its i-th query to $H(p, t)$ with $i < s$, this query equals q_i, and the value r_i is used as the query answer. If there is an s-th query $\langle p, t\rangle$, then I sets the new q_s value equal to $\langle p, t\rangle$ leaving the rest of the label entries for $(P-1, T-1)$ unchanged. This q_s value will be propagated back and forth across G in the next scans in order to find the answer to the query. On the other hand, if $f^{X,Y}(x)$ halts without making any new query to H, then the local improvement function gives the invalid answer b: this constitutes a solution to the instance of $\mathrm{RLI}_{\log}$.

It should be clear that the local improvement function I is polynomial time in all cases. It uses the oracle X when increasing scores from 0 to 1, and also when increasing the score for $(P-1, T-1)$ to $2s$ while simulating the function $f^{X,Y}(x)$ querying X.

This completes the definition of the $\mathrm{RLI}_{\log}$ instance. Suppose (still arguing in S^1_2) that we have a solution to this instance. There are three possible ways to have a solution: (1) The initialization function could produce a value giving a non-wellformed neighborhood; (2) The local improvement function could produce a value giving a non-wellformed neighborhood; or (3) a score value could exceed $c = 2p(|x|) + 1$. Option (3) is impossible since this can happen only at the vertex $(P-1, T-1)$, and the function f is constrained to ask fewer than $p(|x|)$ queries to H. Option (1) is likewise impossible, just from the definition of the function E. Likewise, from the definition of the function I, the only way option (2) can occur is at vertex $(P-1, T-1)$ for the function $f^{X,Y}(x)$ to successfully halt.

This means that the only possible answer is a place where $f^{X,Y}(x)$ halted successfully while the improvement function I was attempting to update the label on vertex $(P-1, T-1)$. Because of the fourth wellformedness condition, this can happen only if $f^{X,Y}(x)$ outputs a value y which satisfies $\phi(y, x, X)$. Q.E.D. Theorems 20 and 16. □

References

[1] A. Beckmann and S. R. Buss, *Polynomial local search in the polynomial hierarchy and witnessing in fragments of bounded arithmetic*, Journal of Mathematical Logic, 9 (2009), pp. 103–138.

[2] ——, *Characterization of Definable Search Problems in Bounded Arithmetic via Proof Notations*, Ontos Verlag, 2010, pp. 65–134.

[3] ——, *Corrected upper bounds for free-cut elimination*, Theoretical Computer Science, 412 (2011), pp. 5433–5445.

[4] S. R. Buss, *Bounded Arithmetic*, Bibliopolis, 1986. Revision of 1985 Princeton University Ph.D. thesis.

[5] ——, *Axiomatizations and conservation results for fragments of bounded arithmetic*, in Logic and Computation, proceedings of a Workshop held Carnegie-Mellon University, 1987, vol. 106 of Contemporary Mathematics, American Mathematical Society, 1990, pp. 57–84.

[6] S. R. Buss and J. Krajíček, *An application of Boolean complexity to separation problems in bounded arithmetic*, Proc. London Math. Society, 69 (1994), pp. 1–21.

[7] S. A. Cook and P. Nguyen, *Foundations of Proof Complexity: Bounded Arithmetic and Propositional Translations*, ASL and Cambridge University Press, 2010. 496 pages.

[8] L. A. Kołodziejczyk, P. Nguyen, and N. Thapen, *The provably total NP search problems of weak second-order bounded arithmetic*, Annals of Pure and Applied Logic, 162 (2011).

[9] J. Krajíček, *Bounded Arithmetic, Propositional Calculus and Complexity Theory*, Cambridge University Press, Heidelberg, 1995.

[10] ——, *Forcing with Random Variables and Proof Complexity*, Cambraidge University Press, 2011.

[11] P. Pudlák and N. Thapen, *Alternating minima and maxima, Nash equilibria and bounded arithmetic*, Annals of Pure and Applied Logic, 163 (2012), pp. 604–614.

[12] W. J. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, Journal of Computer and System Sciences, 4 (1970), p. 177192.

[13] A. Skelley and N. Thapen, *The provably total search problems of bounded arithmetic*, Proceedings of the London Mathematical Society, 103 (2011), pp. 106–138.

[14] G. Takeuti, *Proof Theory*, North-Holland, Amsterdam, 2nd ed., 1987.

[15] N. Thapen, *Higher complexity search problems for bounded arithmetic and a formalized no-gap theorem*, Archive for Mathematical Logic, 50 (2011), pp. 665–680.