The provably total NP search problems of weak second order bounded arithmetic

Leszek Aleksander Kołodziejczyk* Phuong Nguyen†
Neil Thapen†

August 19, 2009

Abstract

We define a new NP search problem, the "local improvement" principle, about labellings of an acyclic, bounded-degree graph. We show that, provably in PV, it characterizes the $\forall\Sigma^b_1$ consequences of V^1_2 and that natural restrictions of it characterize the $\forall\Sigma^b_1$ consequences of U^1_2 and of the bounded arithmetic hierarchy. We also show that over V^0 it characterizes the $\forall\Sigma^B_0$ consequences of V^1 and hence that, in some sense, a miniaturized version of the principle gives a new characterization of the $\forall\Pi^b_1$ consequences of S^1_2. Throughout our search problems are "type-2" NP search problems, which take second-order objects as parameters.

1 Introduction and results

An *NP search problem* is the problem of finding, given the parameter x, a witness y for a true $\forall\Sigma^b_1$ sentence, that is, a sentence of the form

$$\forall x \exists y < 2^{|x|^k}\ R(x, y)$$

where R is decidable in polynomial time. If such a sentence is provable in a theory T, we would like to be able to extract from the proof an algorithm to solve the problem. In this way we think of the set of NP search problems provably total in a theory, which we identify with the set of such sentences, as characterizing the algorithmic strength of the theory. This is analogous with the classification of classical fragments of PA by their provably recursive functions.

In [5], Buss defined the bounded arithmetic hierarchy S^i_2 and related it to the polynomial hierarchy. In particular (translated into our terminology) he showed that the NP search problems provably total in S^1_2 can be solved by a polynomial

*Institute of Mathematics, University of Warsaw, Banacha 2, 02-097 Warszawa, Poland, lak@mimuw.edu.pl. Partially supported by grant N N201 382234 of the Polish Ministry of Science and Higher Education, and supported on a visit to Prague by a grant from the John Templeton Foundation.

†Institute of Mathematics, Academy of Sciences of the Czech Republic, Žitná 25, CZ-115 67 Praha 1, pnguyen@cs.toronto.edu & thapen@math.cas.cz. Partially supported by Institutional Research Plan AV0Z10190503 and grant IAA100190902 of GA AV ČR, by a grant from the John Templeton Foundation, and by Eduard Čech Center grant LC505.

time machine, and conversely, given a polynomial time machine, the problem of finding its output given its input can be represented as such a problem.

The result for S^1_2 is generalized in [5] to the other theories in the hierarchy, but in such a way that the search problems considered are no longer NP search problems. Precisely, the Σ^p_{i+1} search problems provably total in S^{i+1}_2 correspond to the functions computed by polynomial time machines with Σ^p_i oracles. Essentially this is the same as the result for S^1_2, but with i levels of quantifier complexity added everywhere. Similar witnessing theorems are shown for the stronger, two-sorted theories U^1_2 and V^1_2, but concerning an even more complex kind of provable sentence, now with second-order existential quantifiers.

A more satisfying characterization would be to classify the strength of the theories of the hierarchy and beyond by describing, in a non-logical way, the class of algorithms needed to solve their provably total NP search problems, perhaps in terms of some class of machines built up out of polynomial time "atoms" and based on a simple combinatorial principle. Again a guiding analogy here is the characterization of the provably recursive functions of fragments of PA in terms of primitive recursive functions and countable ordinals (see e.g. [11]).

This was achieved for the second level of the hierarchy in [7]. It was shown that the NP search problems provably total in S^2_2 are exactly those that are reducible to PLS problems, where a PLS problem is solved by finding a node with locally minimal cost in an exponential-size graph with polynomial time cost and neighbourhood functions; a PLS problem can also be thought of as corresponding to a limited kind of exponentially-long iteration of a polynomial time function. Another result along these lines was for the theory consisting of S^1_2 together with a form of the weak pigeonhole principle, in terms of a subclass of probabilistic polynomial time machines (Wilkie, published in [13]). In [15] combinatorial and computational characterizations were given of the $\forall\Sigma^b_1$ consequences of the next two levels of the hierarchy, somewhat in the spirit of PLS. In [18] this was extended to the whole bounded arithmetic hierarchy; also see that paper for a broader introduction to this area, and in particular to the important question about improving the known separations between the theories in the relativized hierarchy to $\forall\Sigma^b_1(\alpha)$ separations. Other work dealing with these issues has appeared recently in [3] and [17]. The aim of the current paper is to extend this kind of characterization to theories beyond the bounded arithmetic hierarchy, such as the two-sorted theories U^1_2 and V^1_2 (although in this two-sorted setting the problems we consider will no longer be strictly NP search problems, since they will take second-order objects, which can be thought of as "oracles", as parameters; see below).

Our characterizations involve a "local improvement" principle, LI, which is about labellings of a directed, acyclic, bounded-degree graph. We are given a scoring function that computes a "score" for every node in the graph from the labels given to that node and its neighbours; an initial labelling which scores 0 everywhere; and an improvement function which allows us to increase the score of a node x by 1 if the score s of x under the current labelling is even and all predecessors of x already score $s+1$, or if the current score s is odd and all successors already score $s+1$. The principle says that under these conditions, we can find labellings that give arbitrarily high scores.

The principle can be thought of as an exponentially blown-up version of PLS: the solution space is the set of (exponential-sized) total labellings, and the cost of a labelling is the (exponential-sized) total assignment of scores to nodes

that it generates. The improvement function changes a labelling to give you a "better" cost. However all functions work locally, on a finite part of a labelling at a time; this is what allows the principle to be written in an $\forall\Sigma^b_1$ way.

In its full strength, LI captures the $\forall\Sigma^b_1$ consequences of V^1_2. If we restrict the score to be of at most polylogarithmic size in our parameters, and fix the graph to be an interval $[0, a)$ with the usual ordering on it, the principle captures the $\forall\Sigma^b_1$ consequences of U^1_2. If we restrict the maximum score to a finite number k, and keep the graph as an interval, it corresponds to the game induction principle of [18] and thus to the $\forall\Sigma^b_1$ consequences of T^k_2 (equivalently, of S^{k+1}_2); in particular for $k = 1$ it can be seen to correspond to the "exponentially-long iteration" version of PLS. We remark that the new principle has intuitively a different flavour from the game induction characterization of the consequences of T^k_2, since it has the feeling of constructing a witness co-operatively, rather than of something adversarial.

In this way we give new algorithmic/combinatorial characterizations of the strength of a large range of bounded arithmetic theories, in a uniform way (except, unfortunately, for the change in the topology of the underlying graph), over PV, a relatively weak base theory.

The results mentioned above concern bounded arithmetic theories with the smash function, corresponding to the polynomial hierarchy and its extensions. But by a similar construction we can show that the principle LI captures precisely the Σ^B_0 consequences of V^1 over V^0, where V^1 is a "linear" version of V^1_2 and V^0 has essentially the strength of $\mathrm{I}\Delta_0$ (see [10]). Via the RSUV translation between one-sorted theories with the smash function and two-sorted theories without it, this can be seen as a new characterization of the "$\forall\Pi^b_1$ consequences of S^1_2" (previously known characterizations have usually been based on consistency statements for extended Frege and related propositional proof systems, although these can have an elegant combinatorial nature, see e.g. [1]). We would suggest that questions about the $\forall\Pi^b_1$ consequences of S^1_2 are more naturally studied in the second-order setting. This is because using the first-order setting leads to issues about finding a suitable language (for example, a language containing all polynomial time functions is too strong) and a suitable base theory (BASIC is too weak and PV too strong). V^0 provides a robust base theory which is substantially weaker than V^1, and studying the strength of V^1 over V^0 is analogous to studying complexity classes below P using AC^0-reductions.

We will mention two possible applications of this result about V^1 and V^0. Firstly, one of the goals of research in bounded arithmetic is to characterize the minimal theories needed to prove various theorems of finite mathematics (recently [9] this has been called "bounded reverse mathematics", by analogy with the programme of reverse mathematics which has done this for theorems of infinite mathematics and strong theories of second-order arithmetic). The natural way to show that a theory is minimal is to prove its axioms from the theorem in question. However, since many theorems studied in bounded reverse mathematics are $\forall\Sigma^B_0$ and most relevant theories have higher logical complexity, the best we can hope to show is that a given theorem implies all the $\forall\Sigma^B_0$ consequences of a theory. To achieve this goal, characterizations of the $\forall\Sigma^B_0$ consequences of various theories, such as ours for V^1, are likely to be helpful.

Secondly, via translations into propositional logic, it gives some new candidate families of DNFs which may be hard to prove in the Frege proof system. One is the propositional translation of LI (which is as hard as the consistency of

extended Frege). The other two are the translations of two natural weakenings of LI, namely $\mathrm{LI}_{\log}$, in which we restrict the bound on scores to polylogarithmic size, and LLI, in which we restrict the graph to a line. If we combine these restrictions we get $\mathrm{LLI}_{\log}$, which has a short proof in the Frege system; but it is not clear where these two principles by themselves fall between Frege and extended Frege. The difficulty of the bounded arithmetic versions of these principles is similarly unclear; they may be as strong as LI, or already provable in U^1_2, or somewhere in between.

The paper is organized in the following way. We first give precise definitions of the languages and theories we will use and of our principle LI (we have been rather informal with our notation above) and summarize our results. In Section 2, we use weakenings of LI together with results from [18] to characterize the $\forall\hat{\Sigma}^{b+}_1$ consequences of the theories of the first-order bounded arithmetic hierarchy. In Section 3, we go on to characterize the $\forall\hat{\Sigma}^{b+}_1$ consequences of U^1_2; the idea is to use the principle to simulate a polynomial time algorithm which is trying to show that a certain large second-order object is not a correctly-formed PSPACE computation. In Section 4, we characterize the $\forall\Sigma^B_0$ consequences of V^1, by simulating a "linear PLS" algorithm which is trying to show that the computation of a large circuit is incorrect. The idea is similar to the previous section, but is more complicated both in the nature of the simulation and in that we are working in the less familiar setting of linear rather than polynomial resources. In Section 5, we characterize the $\forall\hat{\Sigma}^{b+}_1$ consequences of V^1_2, by describing how to adapt the argument of Section 4. Section 6 contains proofs of some unsurprising technical lemmas which were postponed from earlier sections.

1.1 Languages and theories

We describe in particular where our definitions are different from the usual ones. For more details of the usual definitions in bounded arithmetic see [13], [6] or [10].

We will work with two-sorted theories of arithmetic. Normally, these are formalized in such a way that the "first-order" sort consists of numbers and the "second-order" sort consists of bounded sets, or equivalently, sequences of bits [19]. However, we find it more convenient to deal with a second-order sort containing bounded sequences of numbers, rather than sequences of binary bits. We therefore redefine our languages and theories accordingly. All our main results can be translated to the more traditional "set" setting (see Section 6.1 below).

Our languages will use symbols $X(i)$, $|X|_l$ and $|X|_b$ for the ith element of a sequence, the length of a sequence, and the upper bound of a sequence, respectively. We take axioms guaranteeing that $X(i) = 0$ for all $i \geq |X|_l$ and that $X(i) < |X|_b$ for all i; but note that two second-order objects can have the same non-zero elements but different lengths and upper bounds. We will not include a symbol for second-order equality.

In general, when we write a tuple of parameters as $\bar{X}$ it may include a mixture of first- and second-order variables. A *first-order* formula is one which contains no second-order quantifiers; it is allowed to contain second-order variables and constants.

We will consider two kinds of theory: one with the smash function $x\#y := 2^{|x||y|}$ for the number sort (the polynomial setting), the other without this function (the linear setting).

1.1.1 The linear setting

The language L_1 contains the symbols $X(i), |X|_l, |X|_b$ and the basic language $0, 1, +, \cdot, <$ of arithmetic. We also allow a finite number of other useful, easy functions, in particular division, remainder, a pairing function $\langle x, y\rangle$ and projection functions π_1, π_2 to recover the elements of a pair. We will extend this notation to talk about larger tuples as needed, and will use the pairing function to treat second-order objects as two- or higher-dimensional tables in the usual way.

We also close the language under the following operation: for each quantifier-free formula $\phi(\bar{X})$, with all free variables shown, we add a new function symbol $\text{choose}_\phi(\bar{X}, y, z)$, which is intended to take the value y if $\phi(\bar{X})$ is true and the value z otherwise.

If $\mathcal{C}$ is a (possibly empty) set of constant symbols, a *simple* $L_1(\mathcal{C})$ *term* contains only first- and second-order free variables and constants, and the symbols $|X|_l, |X|_b, 0, 1, +, \cdot$.

We call a first-order quantifier *bounded* if it is bounded by a simple term. An E_1 formula consists of a block of bounded existential quantifiers followed by an open formula. An A_1 formula is dual to this. The class of all bounded first-order formulas is denoted Σ_0^B. The class Σ_1^B consists of formulas made up of a bounded existential second-order quantifier (i.e. one of the form $\exists X < t$, where t is a simple term, meaning that both $|X|_l$ and $|X|_b$ are bounded by t) in front of a Σ_0^B formula.

All our theories in the language of L_1 will extend the theory E_1-IND, which we define to contain induction for all E_1 formulas with parameters, together with axioms fixing the basic properties of the language as defined above. (We avoid the name IE_1, as this is associated with a weaker theory in a more algebraic language.)

V^0 and V^1 are theories consisting of the basic axioms plus the Σ_0^B or Σ_1^B comprehension schemes, respectively. These schemes express that a second-order object exists for every sequence definable by formulas from this class, with parameters. Formally, comprehension for a formula $\phi(\bar{X}, i, y)$ is the axiom

$$\begin{aligned}&\forall \bar{X} \exists W,\ |W|_l = l \wedge |W|_b = b + 1 \wedge \forall i < l \\ &\quad [(\phi(\bar{X}, i, W(i)) \wedge \forall y < W(i)\, \neg\phi(\bar{X}, i, y)) \vee (W(i) = b \wedge \forall y < b\, \neg\phi(\bar{X}, i, y))].\end{aligned}$$

1.1.2 The polynomial setting

L_2 extends L_1 to include the smash function $\#$. Furthermore it includes a term $f_e(X, x)$ for every polynomial time oracle Turing machine e, where oracle replies are allowed to be numbers (in binary), rather than just single bits. The symbol $f_e(X, x)$ is understood as standing for the output of machine e if run on input x, with queries to the oracle tape replied to according to the sequence X, and with the bounds on X available as hidden inputs to the machine (this is necessary for the machine to be able to read all bits of an oracle reply and remain

polynomial time). Via coding we can take such terms to have any number of first- or second-order inputs, in the usual way.

If $\mathcal{C}$ is a (possibly empty) set of constant symbols, a *simple* $L_2(\mathcal{C})$ *term* contains only first- and second-order free variables and constants, and the symbols $|X|_l, |X|_b, 0, 1, +, \cdot, \#$.

We will not use the normal definition of a Σ^b_1 formula from [5]. Instead we define the closely-related class of $\hat{\Sigma}^{b+}_1$ formulas, where the symbol $\hat{}$ indicates that, in terms of quantifier complexity, the formulas correspond to strict Σ^b_1 formulas and the superscript $+$ indicates that the formulas may contain second-order free variables.

A $\hat{\Sigma}^{b+}_1$ formula consists of a block of bounded, first-order existential quantifiers followed by an open L_2 formula. $\hat{\Pi}^{b+}_1$, $\hat{\Sigma}^{b+}_i$, $\hat{\Pi}^{b+}_i$ formulas are defined similarly. Σ^{b+}_∞ is the class of all such bounded first-order formulas, and $\Sigma^{1,b}_1$ is the class of formulas consisting of a bounded existential second-order quantifier (i.e. $\exists X$ with both $|X|_l$ and $|X|_b$ bounded by a simple L_2 term) followed by a Σ^{b+}_∞ formula.

The L_2 terms are a natural extension of the PV function symbols to a two-sorted world, and all our theories in this language will extend a base theory PV^+. This is the universal theory which extends PV to talk in the obvious way about the oracle machines named by our L_2 terms, where PV is our name for the usual first-order version (sometimes called PV_1) of Cook's equational theory PV [8].

We will also consider natural second-order versions $T^{1+}_2, T^{2+}_2, \ldots$ of the bounded arithmetic hierarchy, where T^{i+}_2 is PV^+ together with induction for $\hat{\Sigma}^{b+}_i$ formulas, with first- and second-order parameters.

V^1_2 is a theory containing the Σ^{b+}_∞ comprehension scheme and the induction scheme $\Sigma^{1,b}_1$-IND. The theory U^1_2 differs from V^1_2 in that it only has the "polynomial induction" scheme $\Sigma^{1,b}_1$-LIND, in which induction only holds up to logarithmically many steps. V^1_2 should be thought of as the analogue of V^1 in the polynomial setting. The reasons for differences in notation and in the style of axioms ($\Sigma^{1,b}_1$ induction as opposed to Σ^B_1 comprehension) are mainly historical.

1.1.3 Term comprehension

A type of comprehension scheme useful when dealing with sequences rather than sets is *term comprehension*. Our choice of language above, in particular the inclusion of the "choose$_\phi$" function in L_1, was partly to ensure that in a number of technical contexts term comprehension will give us all the sequences we need.

Definition 1 *Let L be L_1, L_2 or an extension of one of these. Term comprehension is the scheme*

$$\forall \bar{X}, l, b\, \exists W\, [|W|_l = l \wedge |W|_b = b + 1 \wedge \forall i{<}l\, W(i) = \min(t(\bar{X}, i), b)]$$

for L-terms t.

Let M be an L-structure. A sequence W is term definable *in M if there exist a term $t(\bar{X}, i)$, parameters $\bar{X}$ in M and a length and bound l and b in M such that*

$$|W|_l = l \wedge |W|_b = b + 1 \wedge \forall i{<}l\, W(i) = \min(t(\bar{X}, i), b).$$

Let Γ be an extension of L. A sequence is closed Γ-term definable *in M if the above holds but no parameters $\bar{X}$ are present and both l and b are given by closed Γ-terms.*

1.2 Search problems

To give our results about L_2 theories in their strongest form, we present them as reductions between NP search problems. In our two-sorted setting, these are allowed to have second-order parameters, and are sometimes called *type-2* search problems, following [2, 4].

We define a search problem to be a true sentence of the form $\forall \bar{X} \phi(\bar{X})$, where, in the setting of L_1, ϕ is an E_1 formula and we call such sentences $\forall E_1$; in the setting of L_2, ϕ is a $\hat{\Sigma}^{b+}_1$ formula and we call such sentences $\forall \hat{\Sigma}^{b+}_1$. The initial $\forall$ symbol can be thought of as indicating universal closure over all free variables of both sorts.

Definition 2 *A search problem $P = \forall \bar{X}\, \exists y < t(\bar{X})\, \phi(\bar{X}, y)$ is reducible to a search problem $Q = \forall \bar{U}\, \exists v < t(\bar{U})\, \theta(\bar{U}, v)$ if there exist polynomial time, oracle machines $\bar{F}, g$ such that*

$$\forall \bar{X}, v\, [v < t(\bar{F}(\bar{X})) \wedge \theta(\bar{F}(\bar{X}), v) \rightarrow g(\bar{X}, v) < t(\bar{X}) \wedge \phi(\bar{X}, g(\bar{X}, v))].$$

If $\mathcal{P}$ and $\mathcal{Q}$ are classes of search problems, we say that $\mathcal{P}$ is reducible to $\mathcal{Q}$, $\mathcal{P} \leq \mathcal{Q}$ (provably in a theory T), if every search problem in $\mathcal{P}$ is reducible to one in $\mathcal{Q}$ (provably in T, with the proof allowed to depend on the particular problems). We use $\mathcal{P} \equiv \mathcal{Q}$ to express that the reducibility goes in both directions.

We should explain the notation here. We are given an instance $\bar{X}$ of P, and make from it an instance $\bar{U} = \bar{F}(\bar{X})$ of Q, which we think of as locally being derived from $\bar{X}$ in polynomial time. Formally, $\bar{F}$ is a collection of PV^+ terms, that is, of polynomial time oracle Turing machines. For each first-order variable u_i in $\bar{U}$ there is a term f_i in $\bar{F}$, and for each second-order variable U_i in $\bar{U}$ there are terms F_i, F^l_i and F^b_i in $\bar{F}$. Then $\theta(\bar{F}(\bar{X}), v)$ is obtained by taking $\theta(\bar{U}, v)$ and replacing occurrences of terms of the form u_i, $U_i(z)$, $|U_i|_l$ and $|U_i|_b$ respectively with $f_i(\bar{X})$, $F_i(\bar{X}, z)$, $F^l_i(\bar{X})$ and $F^b_i(\bar{X})$. Reducibility then states that from any solution v for $\bar{F}(\bar{X})$ in θ we may compute, in polynomial time, a solution $g(\bar{X}, v)$ for $\bar{X}$ in ϕ.

1.3 The local improvement principle

We give a formal statement of our principle. To write it in a $\forall E_1$ (hence automatically also $\forall \hat{\Sigma}^{b+}_1$) way, as a type-2 search problem, we exploit the fact that the principle only needs to talk about labels on constantly many nodes of the graph at once, so can be expressed without referring to any global properties of labellings or scores that would require a second-order quantifier.

Formally, fix size, bound and score parameters a, b and c. Let G be a directed acyclic graph on $[0, a)$. Further assume that G has indegree and outdegree both bounded by a constant (which we take to be four, to fix our principle as a single sentence), and that the directions agree with the ordering, that is, if there is an edge from x to y then we must have $x < y$. We will call a node at the other end of an edge coming into x (respectively going out of x) a predecessor

(respectively successor) of x. We assume that G comes with functions which, for a node x, explicitly output its predecessors and successors.

Given a node x, call the predecessors and successors of x together the *neighbours* of x. The *neighbourhood* of x consists of x together with its neighbours, and the *extended neighbourhood* of x consists of x, its neighbours, and all of their neighbours.

An *initial global labelling* E of G is a function which associates with every node of G a label in $[0, b)$.

A *local labelling* w of G of size k is a list naming k nodes of G and associating with each of them a label in $[0, b)$.

A *scoring function* S takes as input a node x and a local labelling w of the neighbourhood of x. It either returns a number in $[0, c]$, which we call the score at x under w, or it returns a symbol $*$, in which case we say that x is not well-formed under w.

An *improvement function* I takes as input a node x and a local labelling w of the neighbourhood of x. It returns a number x' in $[0, b)$ which we think of as a new, improved label for x.

Note that I and S may be given labellings as input that cover more nodes than just the neighbourhood of x; however their output depends only on the labels in the neighbourhood of x.

Definition 3 *Let $\Theta(G, E, S, I, a, b, c)$ be the E_1 formula asserting that if G, E, S and I are as above, then the following three things cannot all be true:*

1. *All nodes x score 0 under the initial global labelling E. (Formally, for all nodes x, if w is the local labelling of the neighbourhood of x that arises naturally from E, then the score at x under w is 0).*

2. *For any node x let w be any local labelling of the extended neighbourhood of x, under which x and all its neighbours are well-formed. Let w' be w with the label of x replaced by the improvement $x' = I(x, w)$. If there is a number m such that $S(x, w) = 2m$ and $S(y, w) = 2m + 1$ for every predecessor y of x, then under the improved labelling w', $S(x, w') = 2m+1$ and all other scores are unchanged.*

3. *The dual of (2) - if the score at x under w is $2m+1$ and the score at every successor y of x is $2m + 2$, then in the improved labelling x scores $2m + 2$ and all other scores are unchanged.*

This is first-order because there is a constant bound of 65, the maximum size of an extended neighbourhood, on the size of the local labellings that are quantified over. It is existential because membership of the neighbourhood or extended neighbourhood of x can be expressed without quantifiers. The local improvement principle LI *is the sentence*

$$\forall G, E, S, I, a, b, c\ \Theta(G, E, S, I, a, b, c).$$

For a term t, the linear local improvement principle LLI_t *is the sentence*

$$\forall E, S, I, a, b, c\ \Theta(L_a, E, S, I, a, b, t(c))$$

where the scores are limited to values in $[0, t(c)] \cup \{*\}$ and the graph G is replaced with the fixed line graph L_a given by the interval $[0, a)$ with the usual predecessors and successors. We will be particularly interested in $\mathrm{LLI}_{\log}$ and LLI_k for $k \in \mathbb{N}$.

Theorem 4 LI *is true, and furthermore is provable in* V^1 *(and hence has polynomial size extended Frege proofs).*

Proof We construct an exponentially long (in $|a|$) sequence of exponential-size total labellings of the graph. Start with the initial labelling E. The first node in the graph has no predecessors and scores 0 under this labelling, which is even. So by (2) we may improve E at the first node to a new labelling in which the first node scores 1 and all the rest score 0. Now, the second node in the graph scores 0 and all its predecessors score 1, so by (2) we may again improve our labelling at the second node so that it and the first node both score 1 and the rest still score 0.

We carry on like this until we have a labelling that scores 1 everywhere. Then by (3) we may improve the score at the last node to 2, and then, using (3) repeatedly, propagate 2s back across the whole graph.

Carrying on in this way, after ca local improvements we will have a labelling which scores c everywhere. This cannot be improved at any point, since there is no way to score $c+1$. So this contradicts either (2) or (3), depending on whether c is even or odd. □

1.4 Summary of results

Note that "over PV^+, $P \equiv \forall\hat{\Sigma}^{b+}_1(T)$" means that T proves P, and each $\forall\hat{\Sigma}^{b+}_1$ sentence provable in T, viewed as a search problem, is provably in PV^+ reducible to P.

Theorem 5 *Over* PV^+*, for* $k \in \mathbb{N}$, $\mathrm{LLI}_k \equiv \forall\hat{\Sigma}^{b+}_1(T^{k+}_2)$.

This is proved in Section 2 below, by giving reductions between LLI_k and the game induction principle GI_k of [18].

Theorem 6 *Over* PV^+, $\mathrm{LLI}_{\log} \equiv \forall\hat{\Sigma}^{b+}_1(U^1_2)$.

This is proved in Section 3, essentially by showing that we can use $\mathrm{LLI}_{\log}$ to simulate first producing an exponentially long computation of a PSPACE machine and then running a polynomial time machine which has oracle access to this computation.

Theorem 7 *Over* PV^+, $\mathrm{LI} \equiv \forall\hat{\Sigma}^{b+}_1(V^1_2)$.

Theorem 8 *Over* V^0, LI *characterizes the* $\forall\Sigma^B_0$ *consequences of* V^1.

These last two theorems have similar proofs. For both of them, one direction follows immediately from Theorem 4. The other direction of Theorem 8 is proved in Section 4, essentially by using LI to simulate first producing a computation of an exponentially large circuit and then running an algorithm to solve an LLS problem (a "linear" version of PLS, which we define below) which has oracle access to this computation. The LLS algorithm here plays a similar role to the polynomial time machine simulated in the proof of Theorem 6; the extra complexity is needed because we are now dealing with two-dimensional circuits rather than one-dimensional computations of PSPACE machines. We go on in Section 5 to describe how this proof can be adapted to give a weak version of Theorem 7, where the provability is only in T^{1+}_2, and finally we show how this

can be improved to PV^+ by taking advantage of the fact that our results are given in a strong way, as reducibilities between search problems.

From Theorems 5, 6 and 7, $\forall\hat{\Sigma}^{b+}_1$ axiomatizations of the $\forall\hat{\Sigma}^{b+}_1$ consequences of these theories can easily be derived as corollaries. However because of the way our theories and principles are set up, it is necessary to be careful how to state these. For example, from Theorem 5 we get that LLI_k axiomatizes the $\forall\hat{\Sigma}^{b+}_1$ consequences of T^{k+}_2 over the theory consisting of PV^+ together with term comprehension, where term comprehension is needed to guarantee that the right instance of LLI_k exists to witness a given instance of $\forall\hat{\Sigma}^{b+}_1(T^{k+}_2)$. Alternatively, if we define $\mathrm{LLI}_k(\mathrm{PV}^+)$ to be the axiom scheme consisting of LLI_k for all graphs, scoring functions etc. that are PV^+ definable from parameters, then $\mathrm{LLI}_k(\mathrm{PV}^+)$ axiomatizes the $\forall\hat{\Sigma}^{b+}_1$ consequences of T^{k+}_2 over just PV^+.

2 The bounded arithmetic hierarchy

We prove Theorem 5, that for $k \in \mathbb{N}$, $\mathrm{LLI}_k \equiv \forall\hat{\Sigma}^{b+}_1(T^{k+}_2)$ provably in PV^+. We start by recalling the *game induction principle* GI_k. We have adapted the definition to make it suitable for the two-sorted setting (we have also renumbered the sequences to start from 0 rather than 1, as that is more convenient for this paper). A k-turn game G with moves from $[0, b)$ is formally a subset of b^k. The game is between two players A and B who alternate turns, starting with A, and each turn play a number in $[0, b)$ as their move. B wins if and only if the k-tuple of moves is in G. A game-reduction of game G to game H is a sequence of functions that tell B how to effectively adapt a winning strategy for H into a winning strategy for G. See [18] for more details.

Definition 9 *The k-turn game induction principle GI_k is a $\forall\hat{\Sigma}^{b+}_1$ sentence asserting that if we are given size parameters a and b and second-order objects G, U, V, W, and interpret W as coding a sequence $W_0, \ldots, W_{a-2}$ of game-reductions and G as coding a sequence $G_0, \ldots, G_{a-1}$ of k-turn games with moves from $[0, b)$, then the following cannot all be true:*

1. *U is a winning strategy for B in G_0;*
2. *V is a winning strategy for A in G_{a-1};*
3. *For each $i < a - 1$, W_i gives a game-reduction of G_{i+1} to G_i.*

In [18] the relativized one-sorted version of this principle was shown to be equivalent, as a class of search problems, to the $\forall\hat{\Sigma}^b_1(\alpha)$ consequences of $T^k_2(\alpha)$, provably in $\mathrm{PV}(\alpha)$. The same proof shows that the two-sorted version written above is equivalent to the $\forall\hat{\Sigma}^{b+}_1$ consequences of T^{k+}_2 in the sense of Definition 2, provably in PV^+.

We prove the two directions of Theorem 5 as separate lemmas.

Lemma 10 *For $k \in \mathbb{N}$, LLI_k is provable in T^{k+}_2. Hence LLI_k is reducible to $\forall\hat{\Sigma}^b_1(T^{k+}_2)$, trivially.*

Proof Our proof resembles (but is not, quite) a search-problem reduction from LLI_k to GI_k. We give a proof for odd k. The proof for even k is similar to this.

Using the sequences from our instance of LLI_k as oracles, we define a sequence of games $G_0, \ldots, G_{a-1}$, one for each node in the graph. Moves in the game G_i will be local labellings of the extended neighbourhood of i, under which i and all its neighbours are well-formed. The game is played between two players C and D, who take alternate turns. We will say what each player's goal is at each turn; if a player fails to make a move that achieves this, he loses the game at that point. If both players make successful moves all the way through the game, then the player who moves last wins.

- Before the first turn of the game, we think of the game board as being in an initial position, which is the local labelling given by E.
- On odd-numbered turns, C tries to relabel $i-2$, $i-1$ and i (if these exist) so that the score at i is increased by 1 but the score at $i+1$ is not changed.
- On even-numbered turns, D tries to relabel i, $i+1$ and $i+2$ (if these exist) so that the score at i is increased by 1 but the score at $i-1$ is not changed.

We will derive a contradiction by $\hat{\Sigma}^{b+}_k$ induction on i in the formula "C can always win game G_i", written in the natural way using k quantifiers.

First note that C can always win G_0. Suppose that the game has run for $2m$ turns. This means that both players have made successful moves up to this point, so under the current labelling the score at 0 must be $2m$. By part 2 of LLI_k (that is, item 2 from Definition 3), C can move successfully by improving the labelling at 0. Since k is odd, C has the last move so must win the game.

Also, D can always win G_{a-1}. As in the previous paragraph, D can always move successfully. But k is odd, so C has the last move; however if C could make this last move successfully, then the final labelling would score k at $a-1$. This is impossible, since then by part 3 of LLI_k this score could be improved to $k+1$, which contradicts the bound of k on the possible scores.

Now suppose that C can always win G_i. We will show that C can use his strategy for G_i to win G_{i+1}, as follows (essentially via a game-reduction, see [18]). To help him make moves in G_{i+1}, C will maintain a local labelling W of nodes $i-2$ to $i+3$, with the following properties at the beginning of each odd turn $2m+1$ in G_{i+1}:

- W scores $2m$ at both i and $i+1$;
- $W \restriction \{i-1, \ldots, i+3\}$ is the current position in game G_{i+1};
- $W \restriction \{i-2, \ldots, i+2\}$ would constitute a winning position for C at the beginning of turn $2m+1$ in game G_i – that is, C could win the game from here, whatever D does.

At the start of the game, the initial labelling E gives such a W, by the induction hypothesis.

Now in turn $2m+1$ in G_{i+1}, C first uses his winning strategy for turn $2m+1$ in G_i to relabel $i-2$, $i-1$ and i in W so that i scores $2m+1$ and the scores at $i+1$ and $i+2$ are not changed. He then uses part 2 of LLI_k to improve the label at $i+1$ to score $2m+1$ there without changing any other scores. He then plays $W \restriction \{i-1, \ldots, i+3\}$ as his move in game G_{i+1}.

If $2m+1$ was the last turn, then this move wins G_{i+1} for C. Otherwise, suppose D plays a good move in the next turn of G_{i+1}, turn $2m+2$. C updates W with this move, so that in the new W the scores at i and $i+1$ are $2m+1$ and $2m+2$. C then uses part 3 of LLI_k to improve W at i, to score $2m+2$ there. The changes to the labels of $i-2$ to $i+2$ in W, from the beginning of turn $2m+1$ to this point, constitute a move at turn $2m+1$ in G_i played by C according to his winning strategy, followed by a good move at turn $2m+2$ in G_i played by his opponent. Hence by the properties of W this part of W is now again in a winning position in G_i for C (at the start of turn $2m+3$), as required. □

Lemma 11 *For $k \in \mathbb{N}$, GI_k is reducible to LLI_k, provably in PV^+.*

Proof The idea is to simulate the natural exponential time algorithm which witnesses GI_k. To illustrate this, consider the picture of an instance of GI_3.

$$\begin{array}{rccccc}
G_0: & & x_1 & \rightarrow & x_2 & x_3 \\
 & & \uparrow & & \downarrow & \uparrow \\
G_1: & & x_1 & & x_2 & x_3 \\
 & & \uparrow & & \downarrow & \uparrow \\
 & & \vdots & & \vdots & \vdots \\
 & & \uparrow & & \downarrow & \uparrow \\
G_{a-1}: & \rightarrow & x_1 & & x_2 & \rightarrow \; x_3
\end{array}$$

(Note that x_j is meant to be a different number x_j^i in each row i, but here and below we have chosen not to write these superscripts.) The horizontal arrows at the bottom represent A's claimed winning strategy in G_{a-1}, the horizontal arrows at the top represent B's claimed winning strategy in G_0 and all the vertical arrows represent the game-reductions. The exponential time algorithm is to fill in all the values x_j^i, starting from the bottom left and following the arrows. Once the table is full, somewhere it must contain a witness to our instance of GI_k.

We simulate this algorithm with an instance of LLI_k. The labels that will appear on a node i in our graph will be partially completed sequences of moves in the game G_i; that is, partially completed rows of the table above. The score at a well-formed node will be the number of moves in the sequence labelling it, and an improvement will be the addition of the next move. Due to the way the two principles have been defined, it will be convenient for us to use an (equivalent) upside-down version of LLI_k, in which the roles of predecessors and successors in parts 2 and 3 of the definition of LLI_k are reversed.

The functions in our instance of LLI_k will be polynomial-time definable from the instance of GI_k, and will be set up so that the exponential procedure used to prove LI in Theorem 4 translates into the exponential procedure witnessing GI_k described above. Furthermore, we show that the properties of the simulation are "local" enough that just from the solution to the search problem LLI_k we can derive, in polynomial time, a witness for GI_k. (This pattern of argument will be repeated in later sections of this paper.)

We will do the case when k is odd. The proof for even k is similar. Let our instance of GI_k consist of games $G_0, \ldots, G_{a-1}$, a strategy U for B in G_0, a strategy V for A in G_{a-1}, and a sequence $W_0, \ldots, W_{a-2}$ of game-reductions. Let b be the bound on the size of the moves.

Our instance of LLI_k has graph $[0, a)$ and the labels are numbers smaller than $(b+1)^k$, representing partially completed sequences of moves — the $+1$ is to allow an extra symbol $\emptyset$ representing the absence of a move.

The initial labelling E labels each node with the empty sequence $\langle\rangle$.

Given a node i and a local labelling $w = (w_{i-1}, w_i, w_{i+1})$ of i and its neighbours, we need to define the score at i and the improvement at i.

We define the score first. We will list a number of forms that w can have, and will say what the score at i is in each case. If w does not have one of these forms then it is not well-formed at i and scores $*$.

A question mark indicates that any value is possible at that location; a partial row of vertical arrows indicates that these sequences of moves are matched by the appropriate initial segment of the game-reduction between these two games; a partial row of horizontal arrows indicates that we are at the top or bottom of the graph and that these moves come from an initial segment of the strategy U or V respectively.

$$\begin{array}{r|l} w_{i-1} & \emptyset \quad \ldots \quad \emptyset \\ w_i & \emptyset \quad \ldots \quad \emptyset \\ w_{i+1} & ? \quad \ldots \quad ? \end{array} \qquad S(i,w) = 0\text{; similar for } i = 0 \text{ or } a-1$$

$$\begin{array}{r|l} w_{i-1} & ? \quad ? \quad \ldots \quad ? \\ w_i & x_1 \quad \emptyset \quad \ldots \quad \emptyset \\ & \uparrow \\ w_{i+1} & x_1 \quad \emptyset \quad \ldots \quad \emptyset \end{array} \qquad S(i,w) = 1\text{; similar for } i = 0$$

$$\begin{array}{r|l} w_{a-2} & \quad ? \quad ? \quad \ldots \quad ? \\ w_{a-1} & \rightarrow x_1 \quad \emptyset \quad \ldots \quad \emptyset \end{array} \qquad S(a-1,w) = 1$$

$$\begin{array}{r|l} w_0 & x_1 \rightarrow x_2 \quad \emptyset \quad \ldots \quad \emptyset \\ & \uparrow \\ w_1 & x_1 \quad ? \quad ? \quad \ldots \quad ? \end{array} \qquad S(0,w) = 2$$

$$\begin{array}{r|l} w_{i-1} & x_1 \quad x_2 \quad \emptyset \quad \ldots \quad \emptyset \\ & \uparrow \quad \downarrow \\ w_i & x_1 \quad x_2 \quad \emptyset \quad \ldots \quad \emptyset \\ & \uparrow \\ w_{i+1} & x_1 \quad ? \quad ? \quad \ldots \quad ? \end{array} \qquad S(i,w) = 2$$

and so on, for scores up to $k-1$. The rules for scoring k have an extra condition (recall that k is odd):

$$\begin{array}{r|l} w_0 & x_1 \rightarrow x_2 \quad \ldots \quad \rightarrow x_{k-1} \quad x_k \\ & \uparrow \quad \downarrow \qquad\quad \downarrow \quad \uparrow \\ w_1 & x_1 \quad x_2 \quad \ldots \quad\quad x_{k-1} \quad x_k \end{array} \qquad S(0,w) = k \text{ if also } G_0(x_1, \ldots, x_k) \text{ is false}$$

$$
\begin{array}{r|ccccc}
w_{i-1} & x_1 & x_2 & \dots & x_{k-1} & ? \\
 & \uparrow & \downarrow & & \downarrow & \\
w_i & x_1 & x_2 & \dots & x_{k-1} & x_k \\
 & \uparrow & \downarrow & & \downarrow & \uparrow \\
w_{i+1} & x_1 & x_2 & \dots & x_{k-1} & x_k
\end{array}
\qquad S(i,w) = k \text{ if also } G_i(x_1,\dots,x_k) \text{ is false}
$$

$$
\begin{array}{r|ccccc}
w_{a-2} & x_1 & x_2 & \dots & x_{k-1} & ? \\
 & \uparrow & \downarrow & & \downarrow & \\
w_{a-1} & \rightarrow x_1 & x_2 \rightarrow & \dots & x_{k-1} \rightarrow & x_k
\end{array}
\qquad \begin{array}{l} S(a-1,w) = k \text{ if also} \\ \quad G_{a-1}(x_1,\dots,x_k) \text{ is false} \end{array}
$$

To describe the improvement function, we will give one example, for improving an even score to an odd score in the middle of the graph. The remaining cases should be clear.

Suppose that w is a local labelling of the extended neighbourhood of i, that $S(i,w) = j$ for some even j and $S(i+1,w) = j+1$, and that w is well-formed at $i-1$. Then w must look like the following picture.

$$
\begin{array}{r|cccccccc}
w_{i-2} & x_1 & x_2 & \dots & x_j & \emptyset & \emptyset & \dots & \emptyset \\
 & \uparrow & \downarrow & & \downarrow & & & & \\
w_{i-1} & x_1 & x_2 & \dots & x_j & \emptyset & \emptyset & \dots & \emptyset \\
 & \uparrow & \downarrow & & \downarrow & & & & \\
w_i & x_1 & x_2 & \dots & x_j & \emptyset & \emptyset & \dots & \emptyset \\
 & \uparrow & \downarrow & & \downarrow & & & & \\
w_{i+1} & x_1 & x_2 & \dots & x_j & x_{j+1} & \emptyset & \dots & \emptyset \\
 & \uparrow & \downarrow & & \downarrow & \uparrow & & & \\
w_{i+2} & x_1 & x_2 & \dots & x_j & x_{j+1} & \emptyset & \dots & \emptyset
\end{array}
$$

The improvement at i is to extend w_i to w_i' by calculating the correct next move x_{j+1} in G_i according to the strategy for the reduction of G_{i+1} to G_i. Notice that making this change to w_i does not change the score at any other node, and that the improvement function cannot fail to correctly increase a score j less than $k-1$.

Now suppose we are given a witness (i,w) for our instance of LLI_k. Then the witness must violate (the upside-down version of) either part 2 or part 3 of the definition LLI_k, and the score at i under w must be either $k-1$ or k.

If the score is k we must be violating part 3 of LLI_k (since k is odd) and since it is not possible to score $k+1$, i must have no predecessors (because any predecessor of i must score $k+1$, which is impossible) and so must be 0. Then from the definition of the score, we have a sequence of moves in G_0 in which B plays according to his strategy but loses.

If the score is $k-1$ then we must be violating part 2 of LLI_k. If $i = a-1$ this means that in the improved label for $a-1$, A is playing according to his strategy (by well-formedness in the original w and the rules for improvement) but loses (since otherwise the improvement would score k). If $i < a-1$ this means that in the improved labelling, we have sequences of moves for G_i and G_{i+1} which are matched by the game-reduction but in which B wins G_i but loses G_{i+1}. □

3 The $\forall\hat{\Sigma}_1^{b+}$ consequences of U_2^1

We prove Theorem 6, that $\mathrm{LLI}_{\log} \equiv \forall\hat{\Sigma}_1^{b+}(U_2^1)$, provably in PV^+. We first need a definition.

Definition 12 *Given a term $f(i,x)$, a length l and a bound b, we define an* iterator *of f to be a sequence X of numbers, of length l, with all elements smaller than b, such that $X(0) = 0$ and $X(i+1) = \min(f(i,X(i)), b)$ for $0 \leq i < l-1$.*

Let Γ be PV^+ together with the term comprehension scheme and axioms stating that for every term f (possibly with parameters, of both sorts) and every pair of numbers l and b, there exists an iterator of f as above. Note that PV^+ proves that iterators are unique, when they exist (where by "unique" we mean that they have the same length, bound and values, since we do not formally have second-order equality in our language).

The theory Γ proves that every oracle PSPACE machine has a computation, in the following sense: consider a Turing machine which takes input x and oracle A, runs for $l(x)$ steps, only uses the first $|b(x)|$ spaces on its work tape, and where if y codes the contents of the tape at step i then $g(A,y)$ codes the contents at time $i+1$, for some L_2 terms l, b and g. Then we can easily define f (which will have x as a parameter) such that an iterator for f will consist of 0 followed by all the successive tape configurations in the computation of the machine.

U_2^1 is a theory of bounded arithmetic that closely captures the properties of PSPACE [5], and in fact we have the following:

Lemma 13 *Γ and U_2^1 prove the same $\forall\Sigma_1^{1,b}$ sentences.*

Proof The proof of conservativity of U_2^1 over Γ involves showing that some standard arguments about PSPACE computations can be formalized using open induction, which over the language L_2 means induction for polynomial time formulas, which is available in PV^+. It is similar to the logical treatment of PSPACE in Buss' original work on U_2^1 in [5], with the difference that the theory we work over, PV^+, is much weaker than the theory that Buss uses. The proofs are technical and do not contain anything new, so are postponed to the final section of this paper. It should be noted, however, that it is not always true that things can be formalized in this way; in Section 4 we formalize some similar standard constructions involving circuits, and there open induction alone does not seem to be strong enough.

We will prove the other direction here, by showing that $U_2^1 \vdash \Gamma$. Given a term $f(i,x)$ and a bound b, let $\theta(W,l)$ express that W codes a $b \times l$ table of numbers, all smaller than b, and that for all $x < b$ and all $i < l-1$, $W(x,0) = x$ and $W(x,i+1) = f(i,W(x,i))$. That is, column x of W iterates f from the starting value x. If W satisfies $\theta(W,l)$ then we can define W' satisfying $\theta(W',2l)$ by term comprehension: for $i < l-1$, $W'(x,i) = W(x,i)$ and for $i \geq l$, $W'(x,i) = W(y,i-l)$ where $y = f(l-1, W(x,l-1))$. Hence, using $\Sigma_1^{1,b}$-LIND, in U_2^1 we can construct iterators of any length. □

Now we can prove one direction of Theorem 6, which also shows that $\mathrm{LLI}_{\log}$ has quasipolynomial size Frege proofs, by [14].

Lemma 14 *$U_2^1 \vdash \mathrm{LLI}_{\log}$, and hence $\mathrm{LLI}_{\log}$ is reducible to $\forall\hat{\Sigma}_1^{b+}(U_2^1)$, trivially.*

Proof Given an instance of $\mathrm{LLI}_{\log}$, let $\theta(X, j)$ express that X is a total labelling of the graph which scores j at every point. Given such an X, in U_2^1 we can construct a new labelling X' such that $\theta(X', j+1)$. This is because the point-by-point relabelling of the graph from one end to the other (as in the proof of Theorem 4) can be thought of as given by an iterator of length a which takes X and the instance as parameters. We also have a labelling E such that $\theta(E, 0)$. Hence, by $\Sigma_1^{1,b}$-LIND, for any c there is a labelling which scores $|c|$ at every point, which gives a contradiction as in Theorem 4. □

The other direction is more complicated. Suppose that a $\forall\hat{\Sigma}_1^{b+}$ sentence $\forall\bar{U}\exists v < t(\bar{U})\,\theta(u, v)$ is provable in U_2^1. Then by Lemma 13 it is provable in Γ, so we have (if we ignore term comprehension for the moment), that

$$\mathrm{PV}^+ + \{\forall\bar{E}\exists\alpha\,\mathrm{Iter}_f(\bar{E}, \alpha) : f \in \mathrm{PV}^+\} \vdash \forall\bar{U}\exists v < t(\bar{U})\,\theta(\bar{U}, v),$$

where $\mathrm{Iter}_f(\bar{E}, \alpha)$ expresses that α is an iterator of f using $\bar{E}$ as parameters. Supposing that we can reduce the number of iterated functions f to one, this can be rearranged as

$$\mathrm{PV}^+ \vdash \forall\bar{U}\exists\bar{E}\forall\alpha\exists v < t(\bar{U})\,(\neg\mathrm{Iter}_f(\bar{E}, \alpha) \vee \theta(\bar{U}, v)).$$

We would like to apply a witnessing theorem to this, to get a polynomial time algorithm which, given oracle access to an iterator α of f, outputs a witness v to θ; we would then go on to use the LLI principle to simulate constructing α and then running this algorithm. However the presence of the second-order existential quantifier over $\bar{E}$ prevents us from doing this. The purpose of Lemma 15 is to show that we do not need this quantifier. We do this by expanding the language with constants to stand for the variables $\bar{U}$, and showing that we only need to consider iterators which have all their parameters (including their length and bound) definable in this language. Note also that since we are replacing the universally quantified variables with constants, we only need to consider $\hat{\Sigma}_1^{b+}$ sentences in the expanded language where we had $\forall\hat{\Sigma}_1^{b+}$ sentences in the original language.

So let $L_2(\mathcal{C})$ be L_2 together with a collection $\mathcal{C}$ of new first- and second-order constant symbols. Take a new set $\mathcal{A}(\mathcal{C})$ of second-order constant symbols, containing a symbol $\alpha_{f,l,b}$ for each $L_2(\mathcal{C})$ term f and each pair l, b of simple, closed $L_2(\mathcal{C})$ terms. For $\alpha_{f,l,b} \in \mathcal{A}(\mathcal{C})$ let $\mathrm{Iter}_{f,l,b}(\alpha_{f,l,b})$ be the universal axiom stating that $\alpha_{f,l,b}$ names an iterator of f of length l and bound b. We will use the names Iter_f and α_f for short.

Lemma 15 *Γ is $\hat{\Sigma}_1^{b+}$ conservative over $\mathrm{PV}^+ + \{\mathrm{Iter}_g(\alpha_g) : \alpha_g \in \mathcal{A}(\mathcal{C})\}$ for $L_2(\mathcal{C})$ sentences.*

Proof Let $\exists v < r\,\phi(v)$ be any $L_2(\mathcal{C})$ formula, where ϕ is quantifier free and r is a closed $L_2(\mathcal{C})$ term. Suppose that $\Gamma \vdash \exists v < r\,\phi(v)$ and suppose for a contradiction that there is an $L_2(\mathcal{C}, \mathcal{A}(\mathcal{C}))$ structure M which is a model of $\mathrm{PV}^+ + \{\mathrm{Iter}_g(\alpha_g) : \alpha_g \in \mathcal{A}(\mathcal{C})\}$ and in which $\forall v < r\,\neg\phi(v)$. We may assume without loss of generality that every first-order element of M is smaller than some simple, closed $L_2(\mathcal{C})$ term and that the only second-order elements of M are those named by constants from L_2, $\mathcal{C}$ or $\mathcal{A}(\mathcal{C})$.

We extend M to a bigger structure M^+. The first-order part of M^+ is the same as that of M. The second-order part of M^+ consists of every sequence which is term definable in M, so that M^+ is a model of PV^+ and also of term comprehension.

Let $f(i,x)$ be any $L_2(\mathcal{C})$ term with parameters from M^+ and let l and b be any first-order elements of M^+. We will show that an iterator for f, l, b exists in M^+. It follows that M^+ is a model of Γ, but $\forall v{<}r\, \neg\phi(v)$ still holds in M^+, giving our contradiction.

We may assume that the parameters of f have the form of constants from $\mathcal{C}$ (which we will not mention from now on), a single first-order element u of M and a finite number of iterators from M named by constants $\alpha_{g_1}, \ldots, \alpha_{g_k} \in \mathcal{A}(\mathcal{C})$. By using some simple coding it is easy to iterate k functions in parallel and combine all these iterators into one iterator $\alpha_g \in \mathcal{A}(\mathcal{C})$ from which all the other ones are definable by term comprehension. So we may take the parameters of f to be simply u and α_g. We will write $f(i,x)$ as $f(i,x,u)$ and think of it as having an oracle for α_g. Let t, s be simple, closed $L_2(\mathcal{C})$ terms with $u, l, b < t$ and with $|s|$ greater than the maximum time taken by f on inputs smaller than t.

Consider the following PSPACE machine e, which runs on inputs l', b', u'. The machine first simulates $f(0,0,u')$ running for $|s|$ steps. At each step of f it assumes that f makes an oracle query of the form "$\alpha_g(y) = ?$", and answers the query by simulating the iteration which computes α_g and saving the answer that is given at the yth step of this "inner" computation. Hence this single computation of f takes exactly $|s|l_g$ steps in total (where l_g is the length parameter of α_g) and only requires memory bounded by some $L_2(\mathcal{C})$ term. Let w_1 be the output of this simulation of $f(0,0,u')$. The machine then simulates $f(1, \min(w_1, b'), u')$ in the same way, giving some output w_2, then $f(2, \min(w_2, b'), u')$, and so on for l' simulations of f. The machine then continues with dummy steps until it has run for $|s|l_g t$ steps in total.

Let d be the PSPACE machine with no parameters which simulates e for all triples $l', b', u' < t$ in turn. Then a computation of d is named by one of the constants in $\mathcal{A}(\mathcal{C})$ and so exists in M^+. We can use term-comprehension to extract from it a computation X of e on inputs l, b, u. Then by term comprehension we can construct a sequence W where $W(i) = X(|s|l_g i)$ for $i = 0, \ldots l$, recording all numbers w_i computed by our simulations of f, and from W we can recover an iterator of f.

We should note where open induction (in M^+) is used in this argument: we need it to show that every simulation of α_g produces the same values that actually appear in α_g, and that our simulations of f using our oracle replies $\alpha_g(y_1), \ldots, \alpha_g(y_{|s|})$ give the same value as actual computations of f with these oracle replies. □

Theorem 16 *$\forall\hat{\Sigma}^{b+}_1(U^1_2)$ is reducible to $\mathrm{LLI}_{\log}$, provably in PV^+.*

Proof Suppose that $U^1_2 \vdash \forall\bar{U}\exists v < t(\bar{U})\, \theta(\bar{U}, v)$, where $\theta \in \mathrm{PV}^+$ and t is a simple L_2 term. Then by using Lemma 13, introducing some new constants $\bar{C}$ and then using Lemma 15, we get that

$$\mathrm{PV}^+ + \{\mathrm{Iter}_{\alpha_f} : \alpha_f \in \mathcal{A}(\bar{C})\} \vdash \exists v{<}t(\bar{C})\, \theta(\bar{C}, v).$$

By compactness only finitely many iterators are needed in this proof, so as in the proof of the last lemma we can combine them all into one iterator $\alpha_{f,l,b}$ (where f, l, b are $L_2(\bar{C})$ terms). For simplicity we will write α for $\alpha_{f,l,b}$ and will assume that the bound b is built into the function f. So we have

$$\mathrm{PV}^+ + \forall i{<}(l-1)\,[\alpha(0) = 0 \wedge \alpha(i+1) = f(i, \alpha(i))] \vdash \exists v{<}t(\bar{C})\,\theta(\bar{C}, v).$$

We now write the expression in square brackets as $\mathrm{Def}_f(\alpha, i)$ and move it to the other side. Also we will no longer write $\bar{C}$ but will simply think of every term on the right hand side as being an $L_2(\bar{C})$ term. This gives

$$\mathrm{PV}^+ \vdash \exists i{<}(l-1)\,\neg\mathrm{Def}_f(\alpha, i) \vee \exists v{<}t\,\theta(v).$$

Now by a relativized version of Buss' witnessing theorem there are PV function g, h with oracle tapes which, on input the first-order constants in $\bar{C}$ and given oracle access to α and the second-order constants in $\bar{C}$, output numbers i and v witnessing the right hand side, provably in PV^+. Let us take g and h and replace queries to the oracle $\alpha(i)$ with queries to an $L_2(\bar{C})$ term $A_z(i)$, with a parameter z. The term A treats z as a code for a polylogarithmic-length list of pairs of oracle queries and replies, and uses these to answer queries to α. If a query i is not in this list, A_z gives an arbitrary reply (say 0). Then, assuming without loss of generality that always $g < l-1$ and $h < t$, we get

$$\mathrm{PV}^+ \vdash \forall z, \neg\mathrm{Def}_f(A_z, g(A_z)) \vee \theta(h(A_z))$$

where $g(A_z)$ and $h(A_z)$ represent g and h run using the sequence given by A_z to reply to oracle queries made to α.

We will describe an instance of $\mathrm{LLI}_{\log}$, with all first- and second-order parameters definable by L_2 terms from the constants $\bar{C}$, with the property that, in any model of $\mathrm{PV}^+(\bar{C})$, if z is a solution to our instance then $\mathrm{Def}_f(A_z, g(A_z))$ holds – that is to say, A_z looks like a correctly-formed iterator of f at the point which is found by g on input A_z. This gives our desired search-problem reduction, because now we know that if we simulate $h(A_z)$ for any such z, the output must witness $\exists v{<}t\,\theta(v)$.

We may assume that g always runs for exactly n steps and makes one oracle query "$\alpha(i) = ?$" at each step, for some $0 \le i < l$. The domain of the graph in our instance will be the interval $[0, l)$, the scores will come from $[0, 2n+1] \cup \{*\}$, and the labels of the nodes will always have one of the following five forms:

- $\langle\rangle$, an empty tuple. We call this an "initial" configuration (even length).
- $\langle\beta_i\rangle$. We call this a "first pass" configuration (odd length).
- $\langle\beta_i, q_i^1, \ldots, q_i^{k+1}, r_i^1, \ldots, r_i^k\rangle$. We call this a "going up" configuration (even length).
- $\langle\beta_i, q_i^1, \ldots, q_i^{k+1}, r_i^1, \ldots, r_i^k, ?\rangle$. We call this a "going down, waiting for reply" configuration (odd length).
- $\langle\beta_i, q_i^1, \ldots, q_i^{k+1}, r_i^1, \ldots, r_i^{k+1}\rangle$. We call this a "going down with reply" configuration (odd length).

Here β_i will stand for what the simulation locally believes the value of $\alpha_f(i)$ to be, and $\bar{q}_i$ and $\bar{r}_i$ will stand for oracle queries and replies made so far in a partial computation of g. "?" is represented by some fixed number. The index k will be at most n, the queries q_i^j will always be bounded by l, and β_i and the replies r_i^j will always be bounded by b. This gives us a bound on the size of the labels, as required.

We want to simulate the following process. Start with an initial labelling E where each label $w_0, \ldots, w_{l-1}$ is in the initial configuration. Then, beginning at the top and working down, replace w_0 with $\langle \beta_0 \rangle$ where $\beta_0 = 0$ and replace each w_{i+1} with $\langle \beta_{i+1} \rangle$ where $\beta_{i+1} = f(i, \beta_i)$. Now every label is in a "first pass" configuration.

Now beginning at the bottom and working up, calculate the first oracle query q^1 made by g and append this (the same value q^1 each time) to each w_i. Now every label is in a "going up" configuration.

Then, beginning at the top and working down, append "?" to each of $w_0, \ldots, w_{q^1-1}$ so that these are all in a "going down, waiting for reply" configuration. Then let $r^1 = \beta_{q^1}$ and, continuing down, append r^1 to each of $w_{q^1}, \ldots, w_{l-1}$ (thus correctly answering the query q^1) so that they are all in a "going down with reply" configuration.

Then, beginning at the bottom and working up, calculate the second oracle query q^2 made by g and add it to each w_i, and also replace the "?"s from the previous pass with the value r^1.

Repeat the procedure in the previous paragraphs to add more queries and replies to each w_i, until the moment we obtain somewhere a complete set of n queries and n replies, giving a full computation of g.

The important thing about this procedure is that the information we are adding to a label w_i is always available locally. Hence we can model it with an instance of the local improvement principle.

The initial labelling is simple; every element is just the empty tuple.

The score at i is the number $|w_i|$ of elements in the tuple w_i, if the triple (w_{i-1}, w_i, w_{i+1}) is well-formed. Otherwise the score is $*$. The triple is well-formed if all of the following hold (suitably adapted in the cases $i = 0$ or $i = l - 1$).

1. Each of w_{i-1}, w_i and w_{i+1} is in one of the five configurations above.

2. The sequence of lengths $\langle |w_{i-1}|, |w_i|, |w_{i+1}| \rangle$ matches one of the following, for some odd number m: $\langle m, m-1, m-1 \rangle$; $\langle m, m, m-1 \rangle$; $\langle m, m, m \rangle$; $\langle m, m, m+1 \rangle$; $\langle m, m+1, m+1 \rangle$; $\langle m+1, m+1, m+1 \rangle$. In other words, the sequence contains either only one number or two consecutive numbers, and an odd number cannot appear after an even one.

3. If w_i is in one of the two "going down" configurations, then either i is strictly less than the last query q_i^{k+1} listed in w_i and w_i is in the "going down, waiting for reply" configuration; or i is greater than or equal to q_i^{k+1} and w_i is in the "going down with reply" configuration.

4. In each of the three tuples in the triple, the oracle queries and replies agree with those in the other tuples and are consistent with a possible initial segment of a computation of g.

5. If $i = 0$, then $\beta_i = 0$. If $i > 0$, then $\beta_i = f(i-1, \beta_{i-1})$.

6. The queries and replies in w_i do not contain any witnesses that α is not correctly formed as an iterator. That is, if "$\alpha(q) = r$" and "$\alpha(q+1) = r'$" are among the oracle information in w_i, then $r' = f(q, r)$; and "$\alpha(0) = r$" only appears for $r = 0$.

7. In w_i, if i appears among the queries $\bar{q}_i$ (that is, if the query "$\alpha(i) = ?$" appears), then the reply listed in $\bar{r}_i$, if there is one, is β_i.

We think of the first four properties as "structural". Note that property (5) is exclusively about the numbers β, property (6) is exclusively about the information about the oracle α encoded in the tuples, and property (7) is the only place where the two things are connected, and there it is only the value at i which matters.

Now we need to describe the improvement function and show that it changes scores in the right way. We will deal first with improving even scores to odd scores.

Suppose $0 < i < l - 1$ and we have a local labelling in which the score at i is 0, the score at $i - 1$ is 1 and the score at $i + 1$ is not $*$. Then by properties (1) and (2) the labels around w_i must look like this:

$$\begin{aligned} w_{i-1} &= \langle \beta_{i-1} \rangle \\ w_i &= \langle \rangle \\ w_{i+1} &= \langle \rangle \end{aligned}$$

The improvement at i is to add $\beta_i := f(i-1, \beta_{i-1})$ to w_i. This clearly preserves well-formedness at $i - 1$, i and $i + 1$ so increases the score at i and does not change any other scores.

The edge cases at the top and bottom are similar – the differences are that if $i = 0$ we add $\beta_0 = 0$ to w_0; if $i = l - 1$ we do not need to worry about well-formedness at $i + 1$.

Now suppose $0 < i < l - 1$, and we have a local labelling in which the score at i is some even number $2m > 0$, the score at $i - 1$ is $2m + 1$ and the score at $i + 1$ is not $*$. Then by properties (1), (2) and (3) the tuples w_{i-1}, w_i and w_{i+1} must look like this:

$$\begin{aligned} w_{i-1} &= \langle \beta_{i-1}, q^1, \ldots, q^{k+1}, r^1, \ldots, r^k, ? \text{ or } r^{k+1} \rangle \\ w_i &= \langle \beta_i, q^1, \ldots, q^{k+1}, r^1, \ldots, r^k \rangle \\ w_{i+1} &= \langle \beta_{i+1}, q^1, \ldots, q^{k+1}, r^1, \ldots, r^k \rangle \end{aligned}$$

where $k = m - 1$ and the last element of w_{i-1} is ? if $i - 1 < q^{k+1}$ and otherwise is a reply r^{k+1}. By property (5) at i and $i + 1$ (since the sequence is well-formed at both these places), we also have that $\beta_i = f(i - 1, \beta_{i-1})$ and $\beta_{i+1} = f(i, \beta_i)$.

If $i < q^{k+1}$, then the improvement at i is to add ? to the end of w_i. This leaves the labelling well-formed at i and does not affect well-formedness anywhere else.

If $i = q^{k+1}$, then the improvement at i is to add β_i to the end of w_i, as a new oracle reply r^{k+1}. To show well-formedness we need to show that properties (6) and (7) still hold at i and that (4) still holds at $i - 1$, i and $i + 1$. (7) is clear. For (6): we have added to the tuple $\bar{q}, \bar{r}$ of queries and replies the statement that "$\alpha(i) = \beta_i$". By property (7) at $i - 1$, if the tuple already assigned a value to $\alpha(i - 1)$ then this value was β_{i-1}, which by (5) does not give a mismatch

with our new value of $\alpha(i)$. Similarly by well-formedness at $i+1$, if the tuple already assigned a value to $\alpha(i+1)$ then it must have been $\beta_{i+1} = f(i, \beta_i)$ and we do not have a mismatch. For (4), we know by well-formedness at $i-1$ and $i+1$ that the queries and replies at $i-2$ and $i+2$ are the same as those at i, hence adding the new reply at i will not clash with these.

If $i > q^{k+1}$, then w_{i-1} is in the "going down with reply" configuration and already has a value for r^{k+1}. The improvement at i is to add r^{k+1} to the end of w_i. Then properties (1) – (5) follow easily (at $i-1$, i and $i+1$). Property (6) at i in the new labelling follows from property (6) at $i-1$ in the old sequence. Property (7) must also still hold at i, since the new oracle reply added is for a query q^{k+1} with $q^{k+1} < i$.

The cases at the top and bottom, for $i = 0$ or $i = l-1$, are similar.

Now we describe how odd scores are improved to even scores.

Suppose $0 < i < l-1$, the score at i is some odd number $2m+1$, the score at $i+1$ is $2m+2$ and the score at $i-1$ is not $*$. Then by properties (1) – (4) the tuples w_{i-1}, w_i and w_{i+1} must look like this:

$$\begin{aligned} w_{i-1} &= \langle \beta_{i-1}, q^1, \ldots, q^{k+1}, r^1, \ldots, r^k, ? \text{ or } r^{k+1} \rangle \\ w_i &= \langle \beta_i, q^1, \ldots, q^{k+1}, r^1, \ldots, r^k, ? \text{ or } r^{k+1} \rangle \\ w_{i+1} &= \langle \beta_{i+1}, q^1, \ldots, q^{k+1}, q^{k+2}, r^1, \ldots, r^k, r^{k+1} \rangle \end{aligned}$$

where $k = m-1$ and w_i ends with ? if and only if $i < q^{k+1}$, and similarly for w_{i-1}.

Then the improvement at i is to replace the question mark, if there is one, with r^{k+1} and then to append q^{k+2} to the tuple $\bar{q}$ of queries.

By property (4) at $i+1$, q^{k+2} is the correct next query made by g, given the previous history of oracle queries and replies. So property (4) also holds at i in the improved labelling. Property (6) also carries across from $i+1$ to i. For property (7), either $i \geq q^{k+1}$, in which case r^{k+1} was already present and the improvement is not adding any new information about the oracle at i, or $i < q^{k+1}$, in which case the information in r^{k+1} about the value of $\alpha(q^{k+1})$ cannot directly clash with β_i.

For the edge cases, if $i = 0$ then the improvement is as above. If $i = l-1$ then w_i will not contain a question mark (since all queries q are $\leq l-1$) and the improvement is to calculate the next query q^{k+1} made by g (if there is one) and append it to the tuple $\bar{q}$ of queries.

To complete the argument, let w be any witness to the $\mathrm{LLI}_{\log}$ principle. Then w can only be a well-formed local labelling for which the improvement function does not work, and by the construction of the function this can only happen at the place mentioned in the previous paragraph; we are at the bottom of the graph, and trying to compute the next query made by g, which is impossible if $k = n$ and we have already gone through a complete computation of g. But in this case our local labelling contains all the oracle queries and replies made in this computation, and by property (6) of well-formedness these queries and replies fail to witness that α is not the iterator of f, as required. □

4 The $\forall\Sigma^B_0$ consequences of V^1 over V^0

This section has a similar shape to the last one, but for two-dimensional objects (circuits) in a linear setting, rather than one-dimensional objects (our iterators) in a polynomial setting. The low-complexity consequences of V^1 follow from V^0 together with an axiom asserting that every circuit has a computation. We show (Lemma 19) that for us it is enough to consider only definable circuits. This allows us to take a first-order sentence provable in V^1, obtain a single definable circuit C from the proof, and use a witnessing theorem to derive an algorithm (a version of PLS appropriate to the linear setting, see Definition 20) that will witness the sentence if we give the algorithm oracle access to a computation of C. We can then use the LI principle to simulate constructing the computation of C and running the algorithm on it.

The next definition is a natural generalization of the iterators used in Section 3 (except that to make some things simpler, the transition function uses i as a parameter when calculating values in column i, where an iterator would use $i-1$).

Definition 17 *We define a* circuit *to be a 4-tuple (f, l, h, b) consisting of a term $f(i, j, x_1, x_2, x_3)$ (which we will call the* transition function *of the circuit, and which may also have some other parameters) and a* length l, height h *and* bound b.

A computation *of a circuit consists of an $l \times h$ table of cells, each potentially holding a number $< b$; the value computed in cell (i, j) is $\min(f(i, j, x_1, x_2, x_3), b)$ where x_1, x_2 and x_3 are the values respectively in cells $(i-1, j-1)$, $(i-1, j)$ and $(i-1, j+1)$. A computation is coded as a second-order object in the normal way.*

Values outside the domain given by the height and width are taken to be 0. *The circuit has no input, but the term f may have other parameters which can effectively be used as inputs.*

Given a language L, we will say that a circuit is L definable if its transition function $f(i, j, x_1, x_2, x_3)$ is given by an L term (with only the free variables shown) and its length, height and bound l, h, b are given by simple, closed L terms.

Let $L_1(\mathcal{C})$ be L_1 together with a collection $\mathcal{C}$ of new first- and second-order constant symbols.

Lemma 18 *Let T be any theory in L_1 extending V^0. To show $\forall\Sigma^B_0$ conservativity of V^1 over T for L_1 sentences, it is enough to show E_1 conservativity of V^1 over T for $L_1(\mathcal{C})$ sentences.*

Proof Suppose $V^1 \vdash \forall \bar{X}\, \theta(\bar{X})$ for θ a Σ^B_0 formula from L_1. We will assume for simplicity that θ is of the form $\exists y{<}t(\bar{X})\, \forall z{<}t(\bar{X})\, \phi(\bar{X}, y, z)$ with ϕ quantifier-free, since the argument extends easily to more general formulas with more quantifiers. Let $\bar{C}$ be a tuple of constants from $\mathcal{C}$ and let α be one more second-order constant from $\mathcal{C}$. Then

$$V^1 \vdash \exists y{<}t(\bar{C})\, [\alpha(y) < t(\bar{C}) \rightarrow \phi(\bar{C}, y, \alpha(y))],$$

so E_1 conservativity of V^1 over T for $L_1(\mathcal{C})$ formulas implies that this is also provable in T, so also (since the constants $\mathcal{C}$ do not appear in T)

$$T \vdash \forall A \exists y{<}t(\bar{C})\, [A(y) < t(\bar{C}) \rightarrow \phi(\bar{C}, y, A(y))].$$

Now since T extends V^0, which has comprehension for Σ^B_0 formulas, in any structure $(M, \bar{C}) \models T$ we can take A to be the bounded sequence $t(\bar{C}) \longrightarrow t(\bar{C})$ where $A(y)$ is the least $z < t(\bar{C})$ such that $\neg\phi(\bar{C}, y, z)$ or is 0 if there is no such z. Then

$$(M, \bar{C}) \models \exists y{<}t(\bar{C})\, \forall z{<}t(\bar{C})\, \phi(\bar{C}, y, z)$$

and the lemma follows. □

Take a new set $\mathcal{A}(\mathcal{C})$ of second-order constant symbols, containing a symbol $\alpha_{f,l,h,b}$ for each $L_1(\mathcal{C})$ definable circuit (f, l, h, b). For $\alpha_{f,l,h,b} \in \mathcal{A}(\mathcal{C})$ let $\mathrm{Iter}_{f,l,h,b}(\alpha_{f,l,h,b})$ be an A_1 axiom stating that $\alpha_{f,l,h,b}$ is a computation of the circuit (f, l, h, b), that is, an $l \times h$ table with suitable values in all the cells. In what follows we will often write these as α_f and Iter_f for short.

The next lemma does a similar job to Lemma 15 in the precious section. In the proof, we will appeal to $E_1{-}\mathrm{IND}$ several times to prove essentially the following: if two circuits D and E have the same bounds and have transition functions which agree on all inputs below the bounds, and α_D is a computation of D and α_E is a computation of E, then α_D and α_E are the same everywhere. We need this to show that we can simulate one circuit as part of another; it is provable by induction on i in the bounded universal formula "the ith column of α_D agrees at all points with the ith column of α_E".

Lemma 19 *V^1 is conservative over $E_1{-}\mathrm{IND} + \{\mathrm{Iter}_f(\alpha_f) : \alpha_f \in \mathcal{A}(\mathcal{C})\}$ for E_1 sentences in the language $L_1(\mathcal{C})$.*

Proof Let $\exists y{<}t\, \phi(y)$ be any $L_1(\mathcal{C})$ sentence, where ϕ is open and t is a closed $L_1(\mathcal{C})$ term. Suppose that $V^1 \vdash \exists y{<}t\, \phi(y)$ and suppose for a contradiction that there is an $L_1(\mathcal{C}, \mathcal{A}(\mathcal{C}))$ structure M which is a model of $E_1{-}\mathrm{IND} + \{\mathrm{Iter}_f(\alpha_f) : \alpha_f \in \mathcal{A}(\mathcal{C})\}$ and in which $\forall y < t\, \neg\phi(y)$. We may assume without loss of generality that every first-order element of M is smaller than some simple, closed $L_1(\mathcal{C})$ term and that the only second-order elements of M are those named by constants.

We extend M to a bigger structure M^+. The first-order part of M^+ is the same as that of M. The second-order part of M^+ consists of every sequence which is term definable in M. Notice that given any first-order formula with parameters from M^+ we can replace all second-order parameters with their definitions, yielding an equivalent formula (with respect to first-order variables; we are treating the second-order variables as fixed) with the same quantifier complexity but in which all parameters are from M. Hence M^+ is a model both of $E_1{-}\mathrm{IND}$ and of term comprehension.

We will show that every circuit definable with parameters from M^+ already has a computation in M^+. It can then easily be shown (by constructing computations to evaluate quantifiers) that M^+ satisfies comprehension for all bounded first-order formulas. This, together with $E_1{-}\mathrm{IND}$ and the statement that every circuit has a computation, proves all the bounded first-order consequences of V^1 [16, 10]. But in M^+ we still have that $\forall y{<}t\, \neg\phi(y)$, giving a contradiction.

We first claim that for any $L_1(\mathcal{C}, \mathcal{A}(\mathcal{C}))$ term $t(x)$, with x the only free variable, and for any simple, closed $L_1(\mathcal{C})$ term c, we can construct an $L_1(\mathcal{C})$ definable circuit (f, l, h, b) such that $t(x) = \alpha_{f,l,h,b}(l-1, x)$ for all $x < c$.

We first convert t into a (finite) arithmetic circuit. This has gates for $+$, $\cdot$ and our other arithmetic operations. It has gates for $=$ and $<$, which both take two inputs and output 0 or 1 depending in the obvious way on these; these gates are used to translate the choose_ϕ terms. For each $C \in \mathcal{C}$ it has a gate labelled C with one input, intended to map y to $C(y)$, and for each $\alpha \in \mathcal{A}(\mathcal{C})$ it has a gate labelled α with two inputs, intended to map y and z to $\alpha(y, z)$. It has an input gate for x and hard-wired inputs for constants $c \in \mathcal{C}$ and for $|C|_l$ and $|C|_b$ for $C \in \mathcal{C}$. The circuit is constructed by showing inductively that there is a circuit for every term and open formula, using induction on the complexity of the terms and formulas measured by their length and by the depth of nesting of choose_ϕ functions inside the index formulas ϕ of other choose_ϕ functions.

We then show by induction on the (finite) structure of such a circuit that the value at each gate can be calculated by a $L_1(\mathcal{C})$ definable circuit as required above. The difficult case is if the outermost gate of $t(x)$ has the form $\alpha_{f',l',h',b'}(t_1(x), t_2(x))$ for some constant $\alpha_{f',l',h',b'}$ in $\mathcal{A}(\mathcal{C})$ for a circuit (f', l', h', b'). In this case, by the inductive hypothesis we have $L_1(\mathcal{C})$ definable circuits (f_1, l_1, h_1, b_1) and (f_2, l_2, h_2, b_2) such that $t_1(x) = \alpha_{f_1,l_1,h_1,b_1}(l_1 - 1, x)$ and $t_2(x) = \alpha_{f_2,l_2,h_2,b_2}(l_2 - 1, x)$ for all $x < c$. We will call these circuits $E_{f'}$, E_{f_1} and E_{f_2} for short.

Let $h = \max(h_1, h_2, h')$ and $b = cb_1b_2b'$. We will describe a circuit E_f of height ch, length $l = 1 + l_1 + l_2 + l' + 3h'$ and bound b such that $t(x) = \pi_4(\alpha_f(l-1, hx))$ (where π_4 is a projection function). From this it is easy to construct the circuit needed for the inductive step.

The circuit E_f is actually made up of c circuits $D_0, \ldots D_{c-1}$, with each D_x having height h and length l and consisting of rows xh to $(x+1)h - 1$ of E_f. These subcircuits can be thought of as operating independently and in parallel; there is no interaction between them. We will describe one such subcircuit D_x. Each cell is thought of as containing a 4-tuple (x, z_1, z_2, z_3). The first column computes and stores x (in every row). The next l_1 columns use place z_1 to simulate a computation of E_{f_1}. The next h' columns take the xth row in the output of this simulation and propagate it so that, from this point on, z_1 has this value in each of the first h' rows of the subcircuit. The next $l_2 + h'$ columns do the same for E_{f_2} in place z_2. From this point on, z_1 and z_2 will not change, so the first h' rows of the subcircuit will have access to $t_1(x)$ and $t_2(x)$ as parameters. This is provable in E_1-IND, which is needed to show that all the values in our simulations are the same as the values in α_{f_1} and α_{f_2} themselves (open induction is enough to show that the propagation works). The next l' columns simulate $E_{f'}$ in place z_3, up until the $t_1(x)$-th column of the simulation, after which they just copy the values from this column. The final h' columns take the value of z_3 in the $t_2(x)$-th row of the subcircuit and transfer it to row 0. As above, E_1-IND proves that row 0 of the last column of our subcircuit contains $\alpha_{f'}(t_1(x), t_2(x))$ in place z_3.

This completes the proof of the claim. Now take any circuit D in M^+. By the construction of M^+, D has a length, height and bound l, h, b that are first-order elements of M^+ and a transition function $f(p, i, j, x_1, x_2, x_3)$ that is an $L_1(\mathcal{C}, \mathcal{A}(\mathcal{C}))$ term with a first-order parameter p from M and no parameters except for the ones shown. By the claim, there is an $L_1(\mathcal{C})$ definable circuit

(g, l', h', b') such that $f(p, i, j, x_1, x_2, x_3) = \alpha_g(l' - 1, \langle p, i, j, x_1, x_2, x_3\rangle)$. Using this and a similar construction to that used in the claim, we can expand D to a much larger circuit D' in which each transition in D is replaced with a simulation of a computation of (g, l', h', b') and an extraction of the value of $f(p, i, j, x_1, x_2, x_3)$ from this computation. From any computation of this new circuit we can extract by term comprehension a computation of D (and use E_1−IND to show that each (g, l', h', b') simulates f correctly and thus prove that this really is a computation of D).

However we still do not know that a computation of D' exists in M^+, since the transition function and bounds are not pure $L_1(\mathcal{C})$ terms but still depend on the parameters p, l, h, b. But p, l, h and b must all be smaller than some closed $L_1(\mathcal{C})$ term t, so we can take an $L_1(\mathcal{C})$ definable circuit which simulates in parallel t^4 versions of D', one for each possible value of these parameters up to this bound. A computation of this circuit exists in M^+ and we can recover a computation of D' from it by term comprehension. □

As a technical tool, we define a natural adaptation of polynomial local search problems [7] to the L_1 setting.

Definition 20 *A* linear local search *(LLS) problem is given by a triple of L_1 terms $\langle C(\bar{X}, x), N(\bar{X}, x), s(\bar{X})\rangle$. Given input parameters $\bar{X}$, a solution to the problem is a witness to the fact that the following things cannot all be true (where $*$ is a special value that may be taken by C, and is not treated as a number):*

1. *$C(\bar{X}, 0) \neq *$;*
2. *For each $x < s(\bar{X})$ such that $C(\bar{X}, x) \neq *$, either $N(\bar{X}, x) = x$ or $N(\bar{X}, x) < s(\bar{X})$ and $C(\bar{X}, N(\bar{X}, x)) > C(\bar{X}, x)$;*
3. *There is no $x < s(\bar{X})$ such that $C(\bar{X}, x) \neq *$ and $N(\bar{X}, x) = x$.*

The $x < s(\bar{X})$ such that $C(\bar{X}, x) \neq *$ are regarded as the "possible solutions" to the problem, and $C(\bar{X}, x)$ is the cost of possible solution x. N is a neighbourhood function on possible solutions. The problem expresses the fact that if 0 is a possible solution and if the neighbour of a possible solution x always either equals x or has higher cost, then there is a possible solution with locally maximal cost, i.e. x such that $N(\bar{X}, x) = x$. Note that there is an L_1 term $m(\bar{X})$ which dominates $C(\bar{X}, x)$ for any $\bar{X}$ and any $x < s(\bar{X})$.

Lemma 21 *Let ϕ be an open L_1 formula, t an L_1 term and suppose that E_1−IND $\vdash \forall \bar{X}\, \exists y < t(\bar{X})\, \phi(\bar{X}, y)$. Then witnessing the right hand side is provably reducible to solving an LLS problem. That is, there is an LLS problem $P = \langle C, N, s\rangle$ and an L_1 term g such that*

E_1−IND $\vdash \forall \bar{X}, z\, [$*"$z$ is a solution to $P(\bar{X})$"* $\rightarrow g(\bar{X}, z) < t(\bar{X}) \wedge \phi(\bar{X}, g(\bar{X}, z))]$.

Proof Postponed until Section 6.

Lemma 22 *Let $\exists y < t\, \phi(y)$ be an E_1 sentence in $L_1(\mathcal{C})$. If E_1−IND + $\{\mathrm{Iter}_f(\alpha_f) : \alpha_f \in \mathcal{A}(\mathcal{C})\}$ proves $\exists y < t\, \phi(y)$, then there is an $L_1(\mathcal{C})$ definable circuit D, an $L_1(\mathcal{C}, \alpha)$ LLS problem P (with no parameters except ones that are definable from $\mathcal{C}$ and α, where α is a new constant symbol) and $L_1(\mathcal{C}, \alpha)$ terms g, h such that, provably in E_1−IND, if w is a solution to P, then either $g(w)$ witnesses that $\mathrm{Iter}_D(\alpha)$ is false or $h(w)$ witnesses that $\exists y < t\, \phi(y)$.*

Proof By compactness, we may assume that there is a finite sequence $C_1, \ldots, C_k$ of $L_1(\mathcal{C})$ definable circuits such that $E_1{-}\mathrm{IND} + \{\mathrm{Iter}_{C_1}(\alpha_1), \ldots, \mathrm{Iter}_{C_k}(\alpha_k)\}$ proves $\exists y{<}t\,\phi(y)$. Let D be the $L_1(\mathcal{C})$ definable circuit which simulates $C_1, \ldots, C_k$ computing in parallel. Then the computations of $C_1, \ldots, C_k$ can be recovered from a computation of D by term comprehension. So we can make any model of $E_1{-}\mathrm{IND} + \mathrm{Iter}_D(\alpha)$ into a model of $E_1{-}\mathrm{IND} + \{\mathrm{Iter}_{C_1}(\alpha_1), \ldots, \mathrm{Iter}_{C_k}(\alpha_k)\}$ by adding these computations. The computations are term-definable, so $E_1{-}\mathrm{IND}$ will still hold, and are second-order, so doing this will not change the truth of any first-order $L_1(\mathcal{C})$ sentences. Hence $E_1{-}\mathrm{IND} + \mathrm{Iter}_D(\alpha) \vdash \exists y{<}t\,\phi(y)$.

So $E_1{-}\mathrm{IND} \vdash \neg\mathrm{Iter}_D(\alpha) \vee \exists y < t\,\phi(y)$ and since $\mathrm{Iter}_D(\alpha)$ is A_1 the result follows by Lemma 21. □

We can now prove the main result of this section. The proof will take up the rest of the section.

Theorem 23 *Every $\forall\Sigma^B_0$ consequence of V^1 is already provable in $V^0 + \mathrm{LI}$.*

Proof By Lemma 18 it is enough to show E_1 conservativity of V^1 over $V^0 + \mathrm{LI}$ for $L_1(\mathcal{C})$ sentences. So let $\exists v < t\,\phi(v)$ be an E_1 sentence in $L_1(\mathcal{C})$ which is provable in V^1.

By Lemma 19, $\exists v < t\,\phi(v)$ is also provable in $E_1{-}\mathrm{IND} + \{\mathrm{Iter}_f(\alpha_f) : \alpha_f \in \mathcal{A}(\mathcal{C})\}$. So by Lemma 22, replacing the constant α a universally quantified second-order variable α, there is an LLS problem P, a tuple of $L_1(\mathcal{C})$ terms $f(i, j, x_1, x_2, x_3), l, h, b$ and an $L_1(\mathcal{C})$ term $g(\alpha, w)$ such that

$$E_1{-}\mathrm{IND} \vdash \forall\alpha, w,\ \text{“}w \text{ is a solution to } P(\alpha)\text{”} \rightarrow [\exists v{<}t\,\phi(v) \vee \neg\mathrm{Def}_f(\alpha, g(\alpha, w))]$$

where $\neg\mathrm{Def}_f(\alpha, z)$ expresses that z is a pair $\langle i, j\rangle$ witnessing that α is not a correct computation of the circuit defined by f, l, h and b, that is, that either one of the bounds is violated or

$$\alpha(i, j) \neq f(i, j, \alpha(i-1, j-1), \alpha(i-1, j), \alpha(i-1, j+1)).$$

Now suppose for a contradiction that there is an $L_1(\mathcal{C})$ structure M which is a model of $V^0 + \mathrm{LI}$ but in which $\exists v{<}t\,\phi(v)$ is false. Then we must have

$$M \vDash \forall\alpha, w,\ \text{“}w \text{ is a solution to } P(\alpha)\text{”} \rightarrow \neg\mathrm{Def}_f(\alpha, g(\alpha, w)).$$

We now follow a similar plan to the proof of Theorem 16, with the difference that we are simulating an LLS problem rather than a polynomial time function (so will need exponentially more steps, and the steps will be of a slightly different nature) and simulating oracle queries to a two-dimensional circuit computation rather than to a one-dimensional iterator. In what follows we will assume that the constants $\mathcal{C}$ are built into our language and will no longer mention them. For clarity we have broken the proof up under sub-headings.

A modification of the LLS problem. For reasons of technical convenience, we redefine the notion of an LLS problem slightly. The redefined version is easily seen to be equivalent to the official one; in particular, the witnessing Lemma 21 remains true.

Instead of being defined by a triple of terms $\langle C(\alpha, x), N(\alpha, x), s(\alpha, x)\rangle$, we assume our problem P is given by a predicate $U(\alpha, s, y)$, a function $N(\alpha, s, y)$

and a size parameter a. A solution to the problem is a pair $\langle s, y\rangle$ witnessing the following (α is suppressed):

$$[U(0,0) \wedge \forall i, x{<}a\, N(i,x) < a \wedge \forall i, x{<}a\, (U(i,x) \to U(i+1, N(i,x)))] \\ \to \exists x{<}a\, U(a,x).$$

So, if $\neg U(0,0)$, then any pair $\langle s, y\rangle$ is a solution; otherwise a solution is a pair $\langle s, y\rangle$ with $s, y < a$ such that either $N(s,y) > a$, $U(s,y) \wedge \neg U(s+1,N(s,y))$ or $U(a,y)$ holds.

Additionally, rather than thinking of U, N and g as being computed by $L_1(\alpha)$ formulas or terms, it is convenient to make an ad-hoc definition of a different computation model. A *straight-line program* consists of a sequence of n instructions. A computation of it is a sequence $z_1, \ldots, z_n$ of numbers, where each z_i is computed (according to the ith instruction in the program) either as $t(\bar{z})$, where t is an L_1 term, or as $\alpha(z_j, z_k)$. Here $\bar{z}$ or z_j, z_k are some specified earlier numbers in the sequence. The output of the program is z_n. Every $L_1(\alpha)$ term is computed by such a straight-line program of some constant, finite length.

We may assume that U, N and g are computed by three such programs, all of the same length $n \in \mathbb{N}$. The programs for N and g will output the value of the terms N and g; the program for U will output 1 ("accept") or output 0 ("reject") depending on whether U is true or false. By padding these programs out with dummy queries if necessary, this allows us to simulate U, N and g by processes which each make exactly n oracle queries of the form "$\alpha(i,j) = ?$" (with $i < l$ and $j < h$), where each query is determined in a simple way by the answers to previous queries. This simulation can be formalized in V^0, in the sense that there is a bounded first-order formula defining a function that takes as input a sequence $q_1, r_1, q_2, r_2, \ldots, q_n, r_n$ of oracle queries and replies and either outputs the value of U or returns an error if the sequence did not correspond to a computation of U, and similarly for N and g. Because a sequence of oracle queries and replies $q_1, r_1, q_2, r_2, \ldots, q_m, r_m$, for $m \leq n$, uniquely determines a (partial) computation of one of our straight-line programs, we will refer to such a sequence simply as a (partial) computation of the program. For the purposes of our proof, a partial computation is even allowed to omit the last reply r_m, or to have a question mark "?" in place of r_m (indicating that in our simulation, we are still waiting for a reply to this query).

Our goal. Working in M, we want to find a tuple $s, y, q_1, r_1, q_2, r_2, \ldots, q_{4n}, r_{4n}$ where each q_k is an oracle query and r_k is the corresponding reply, such that if we use these $\bar{q}$, $\bar{r}$ to define an "oracle" $A_{\bar{q},\bar{r}}$ (which is defined arbitrarily to be 0 at places which do not appear in $\bar{q}$), then $\langle s, y\rangle$ gives a solution to $P(A_{\bar{q},\bar{r}})$, but $\mathrm{Def}_f(A_{\bar{q},\bar{r}}, g(A_{\bar{q},\bar{r}}, s, y))$ is true, which is a contradiction.

An iterative algorithm for linear local search. First, to avoid having to make extra queries in some cases, in place of $g(\alpha, s, y)$ we will use a program $g'(\alpha, s, y)$ that first runs $g(\alpha, s, y)$, getting a pair (i, j) as output, and then queries $\alpha(i,j)$, $\alpha(i-1, j-1)$, $\alpha(i-1, j)$ and $\alpha(i-1, j+1)$. The program g' finally outputs 1 if the replies to these queries show that

$$\alpha(i,j) \neq f(i, j, \alpha(i-1, j-1), \alpha(i-1, j), \alpha(i-1, j+1)),$$

that is, if they witness that $\mathrm{Def}_f(\alpha, \langle i, j\rangle)$ is false, and outputs 0 otherwise, that

is, if they witness that $\mathrm{Def}_f(\alpha, \langle i, j\rangle)$ is true. We may assume without loss of generality that g' still runs in exactly n steps.

To obtain the desired tuple, we will simulate the following algorithm B that computes, in the obvious, iterative way, a solution to the local search problem. In our simulation, we will reply to oracle queries in a way that guarantees that g' never accepts (that is, outputs 1) on the solution that the algorithm finds. To make the simulation more uniform (and so easier to describe), we will in fact make the algorithm itself run g' on every candidate solution that turns up in the course of the computation, although this is not strictly necessary.

B starts by setting s and y to 0 and running $U(\alpha, s, y)$. If $U(\alpha, s, y) = 0$ then it runs $g'(\alpha, s, y)$ and halts. Otherwise, until $s = a$: it runs $g'(\alpha, s, y)$; then computes $y' = N(\alpha, s, y)$; then runs $U(\alpha, s + 1, y')$ and halts if the output is 0, and otherwise replaces y with y' and increments s.

Note that, although B runs for an exponentially long time, when simulating B the only oracle replies that we need to store when we increment s are those for the most recent computation of U; informally, we can discard the other replies, and give different answers the next time these queries are made, and B will still find a solution to the LLS problem. Hence the length $4n$ of $\bar{q}$ and $\bar{r}$ in our goal above is the maximum number of oracle queries that we need to store at one time during the simulation, and when B halts this information is sufficient to verify that we have a solution of the local search problem and to run g' on this solution. For example, if B halts with a solution (s, y) where $0 < s < a$ we will have computations of $U(\alpha, s, y)$, $g'(\alpha, s, y)$, $N(\alpha, s, y)$ and $U(\alpha, s+1, N(\alpha, s, y))$.

This bound on the information we need to store is important because it will allow us to store the state of our simulation of B in each label on the graph, so that it is locally accessible from everywhere, which we would not be able to do if we had to remember simultaneously all the oracle replies made so far.

The LI instance – the graph. Now we describe the instance of LI that we use to simulate B and g'. The underlying graph has lh nodes (i, j) arranged in an $l \times h$ grid, matching the form of our circuit. The node $(0, 0)$ is at the top-left corner and $(l - 1, h - 1)$ is at the bottom-right corner. We order the nodes lexicographically, that is, first by column, then by row. The edges of the graph are as follows: there is an outgoing edge from every node (i, j) to each of the nodes $(i + 1, j - 1)$, $(i + 1, j)$, $(i + 1, j + 1)$ and $(i, j + 1)$ (when these exist). The neighbourhood of a typical (i, j) is depicted in Figure 1.

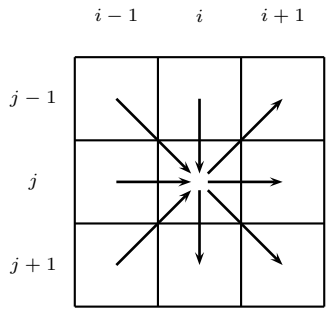

Figure 1: The neighbourhood of (i, j) (only edges touching (i, j) are shown).

The LI instance – the labels. The labels on the nodes can take the following forms:

- $\langle\rangle$ – the "initial" configuration; the initial labelling E labels every node with this.
- $\langle \beta_{i,j}, 0, 0\rangle$ – a "first pass" configuration;
- $\langle \beta_{i,j}, s, y, C\rangle$, where C stands for a sequence of length $\leq 8n$ with one of three possible forms:

  $q_1, r_1, q_2, r_2, \ldots, q_k, r_k, q_{k+1}$ – a "backwards" configuration;

  $q_1, r_1, q_2, r_2, \ldots, q_k, r_k, q_{k+1}, ?$ – a "forwards & waiting" configuration;

  $q_1, r_1, q_2, r_2, \ldots, q_k, r_k, q_{k+1}, r_{k+1}$ – a "forwards & done" configuration.

The labels have bounded size, with $\beta_{i,j} < b$, $s \leq a$, $y < a$, each $q_j < lh$ and each $r_j < b$; $n \in \mathbb{N}$ is fixed, and "?" is a symbol represented by, say, the number b.

An exponential algorithm to simulate B. We can now describe an algorithm, requiring exponential time and space, that simulates B by labelling and relabelling the graph in the style of the proof of Theorem 4 (our original proof of LI in V^1). The algorithm will only ever make use of information recorded locally in the part of the graph it is working on at the moment, so can be translated into the score and improvement functions of an instance of LI. We will define those functions formally in the next part of the proof; the exponential algorithm is included here to make the definitions easier to understand.

The algorithm repeatedly passes forwards and backwards across the graph, following the ordering. It first fills in the graph with a computation of the circuit (f, l, h, b), and then begins to simulate B. To do this, on each forwards pass it picks up an answer to the current oracle query made by B. On each backwards pass it works out the next query made by B and tidies up the labelling, to be ready for the next forward pass.

In more detail, we begin with the graph labelled according to the empty initial configuration. Then, starting from the top-left corner, we pass forwards through the graph and label each node (i, j) with a value $\langle \beta_{i,j}, 0, 0\rangle$ such that

$$\beta_{i,j} = f(i, j, \beta_{i-1,j-1}, \beta_{i-1,j}, \beta_{i-1,j+1})$$

(we are assuming, without loss of generality, that the bound b on values appearing in the circuit is already implicit in the function f). Informally, $\beta_{i,j}$ is what the graph locally (around (i, j)) believes the value of $\alpha(i, j)$ to be.

After the first forward pass, we calculate the first oracle query q_1 in the computation of U on input $(0, 0)$ and then pass through the graph backwards, replacing each label $\langle \beta_{i,j}, 0, 0\rangle$ with the backwards configuration

$$\langle \beta_{i,j}, 0, 0, q_1\rangle.$$

Suppose that q_1 is the query "$\alpha(i_1, j_1) = ?$" for some i_1, j_1. For the next forward pass we define P_{i_1,j_1} to be the set of nodes that are reachable from node (i_1, j_1). Formally

$$P_{i,j} = \{(i', j') \ : \ i \leq i' \wedge i + j \leq i' + j'\},$$

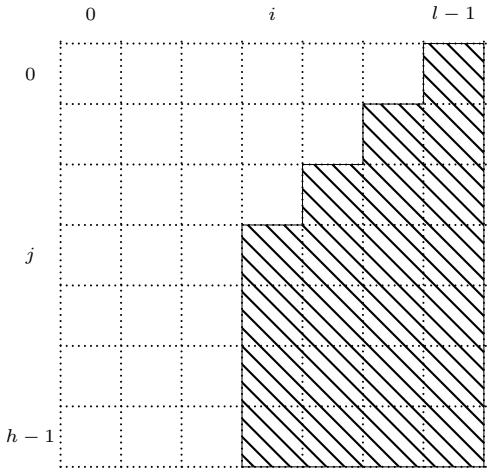

Figure 2: $P_{i,j}$.

illustrated in Figure 2. We go through the graph and replace each backwards configuration with either

$$\langle \beta_{i,j}, 0, 0, q_1, ? \rangle \text{ (forwards \& waiting)}$$

if $(i,j) \notin P_{i_1,j_1}$, or

$$\langle \beta_{i,j}, 0, 0, q_1, \beta_{i_1,j_1} \rangle \text{ (forwards \& done)}$$

if $(i,j) \in P_{i_1,j_1}$. Note that if $(i,j) = (i_1,j_1)$ then β_{i_1,j_1} is already available in the existing label of (i,j), and for other nodes $(i,j) \in P_{i_1,j_1}$ the value β_{i_1,j_1} is already available in the label of at least one predecessor of (i,j).

For the next backward pass we calculate the second oracle query q_2 for the computation of U on input $(0,0)$ and add this to the labels in the same way as in the first backward pass. We also carry out one extra task: we propagate the oracle reply to the first query q_1 to all nodes (i,j) not in P_{i_1,j_1}.

Continuing this way, after $2n+1$ passes ($n+1$ forwards and n backwards) we will have simulated U on input $(0,0)$, and labelled every node with a history of this computation (except perhaps for the last oracle reply). In the same fashion we then simulate g' on $(0,0)$, and if U did not accept $(0,0)$ we halt our algorithm at this point (on reaching the bottom-right corner). Otherwise, we simulate N on input $(0,0)$, getting some output y', and then U on $(1,y')$. If U is rejecting we then halt at the bottom-right corner; otherwise on the next backwards pass we discard the computations of U, N and g' on $(0,0)$ from all labels, and then simulate, as before, g' and N on $(1,y')$ and U on $(2,N(1,y'))$. Note that we still have the computation of U on $(1,y')$ available, since we did not discard this.

We iterate this process until a solution (s,y) is found. We know that g' will reject (s,y), since by construction we have guaranteed that it never sees a place where the circuit is not calculated correctly according to the transition function.

The LI instance – scoring. Recall that in the proof of Theorem 16 we used the length of a label to define the score function, and also as a condition

for well-formedness. Here in place of length we use the "pseudo-score" of a configuration, defined as follows: the pseudo-scores of $\langle\rangle$ and $\langle\beta_{i,j}, 0, 0\rangle$ are 0 and 1, respectively; the pseudo-score of a configuration $\langle\beta_{i,j}, s, y, C\rangle$ is

$$1 + 6sn + |C|$$

where $|C|$ denotes the length of the sequence C. Typically, for each pass through the simulation we will increase $|C|$ by 1 until $|C| = 8n$, and at that point we increase s by 1 and reduce $|C|$ to $2n + 1$. Thus the pseudo-scores will increase by 1 in each pass.

For a local labelling w which is well-formed at node (i, j), the score at (i, j) under w is defined to be the pseudo-score of the label of (i, j). The maximum score c in our instance of LI is $1 + (6a + 8)n$.

It remains to define well-formedness. A local labelling w is well-formed at (i, j) if the following conditions A1–A8 and B1–B5 hold. Conditions A1–A8 describe properties of the label at (i, j), while B1–B5 describe properties that also involve other labels in the neighbourhood of (i, j). Here $\langle\beta_{i,j}, s, y, C\rangle$ is the label at (i, j), and C is a sequence of the form $q_1, r_1, q_2, r_2, \ldots, \delta$ where the last entry δ may be a query q_k, a reply r_k or a question mark, and no other entries may be question marks. For an interval $[u, v]$ we will use $C \restriction [u, v]$ to mean the (possibly empty) subsequence of C that begins at the uth element and ends either at the vth element or when we reach the end of C.

A1. If $s = 0$ then $y = 0$. If $|C| \leq 2n$ then $s = 0$.

A2. $C \restriction [1, 2n]$ is a (partial) computation of U on (s, y).

A3. If $C \restriction [1, 2n]$ is a rejecting computation of U, then $s = 0$ and $|C| \leq 4n$.

A4. $C \restriction [2n + 1, 4n]$ is a (partial) computation of g' on (s, y).

A5. $C \restriction [4n + 1, 6n]$ is a (partial) computation of N on (s, y).

A6. $C \restriction [6n + 1, 8n]$ is a (partial) computation of U on $(s + 1, y')$ where y' is the output of N on (s, y) from the computation in A5.

A7. If C is in a "forwards & waiting" configuration

$$\langle\beta_{i,j}, s, y, q_1, r_1, q_2, r_2, \ldots, q_k, r_k, q_{k+1}, ?\rangle$$

the last query q_{k+1} must be of the form "$\alpha(i', j') = ?$", where $(i, j) \notin P_{i',j'}$. On the other hand, if C is in a "forwards & done" configuration

$$\langle\beta_{i,j}, s, y, q_1, r_1, q_2, r_2, \ldots, q_k, r_k, q_{k+1}, r_{k+1}\rangle$$

the last query q_{k+1} must be of the form "$\alpha(i', j') = ?$", where $(i, j) \in P_{i',j'}$.

A8. If, for any node (i', j'), replies to all the queries "$\alpha(i', j') = ?$", "$\alpha(i' - 1, j' - 1) = ?$", "$\alpha(i' - 1, j') = ?$" and "$\alpha(i' - 1, j' + 1) = ?$" are present in C, then they obey the transition function f.

B1. The pseudo-scores of the nodes in the neighbourhood of (i, j) must be either all the same or of two consecutive values (i.e., $\{2m + 1, 2m\}$ or $\{2m + 1, 2m + 2\}$). Furthermore a node with an even pseudo-score cannot have a successor with an odd pseudo-score.

B2. If two nodes u v in the neighbourhood of (i, j) have pseudo-scores in some set $\{2m + 1, 2m\}$ then the labels must be the same at every position, except possibly for the first position and where there is a "?".

B3. Suppose two nodes u, v in the neighbourhood of (i, j) have pseudo-scores of the form $2m + 1$ and $2m + 2$ respectively, for some m; in particular the label of u has the form

$$\langle \beta_{i',j'}, s', y', q'_1, r'_1, q'_2, r'_2, \ldots, q'_k, z \rangle$$

for some k (where z is either ? or some reply r'_k). Then either (a) $k < 4n$, in which case the labels of u and v must be the same at every position, except possibly for the first position and where there is a "?"; or (b) $k = 4n$, in which case the label of v must have the form

$$\langle \beta_{i'',j''}, s' + 1, y'', q''_1, r''_1, q''_2, r''_2, \ldots, q''_n, r''_n, q''_{n+1} \rangle$$

where

- y'' is the output of the computation $q'_{2n+1}, r'_{2n+1}, \ldots, q'_{3n}, r'_{3n}$ (of N on input (s', y'));
- $q''_1 = q'_{3n+1}, r''_1 = r'_{3n+1}, q''_2 = q'_{3n+2}, r''_2 = r'_{3n+2}, \ldots, q''_n = q'_{4n}$; and
- if z is not a ?, then $r''_n = z$.

B4. The transition function is locally correct at (i, j), that is,

$$\beta_{i,j} = f(i, j, \beta_{i-1,j-1}, \beta_{i-1,j}, \beta_{i-1,j+1}).$$

B5. At each node (i', j') in the neighbourhood of (i, j), if the query "$\alpha(i', j') =$?" appears in the label of (i', j'), then the corresponding reply, if there is one, must be $\beta_{i',j'}$.

Now we describe the improvement function. It will be the case that the only well-formed local labellings for which this function does not improve the score are certain labellings that can occur at the bottom-right corner, and from any such labelling we will be able to obtain a solution to the search problem which we know will not witness $\neg\mathrm{Def}_f$.

The LI instance – improving an even score to an odd score. This is always possible and corresponds to one step in a forward pass of our exponential algorithm; the effect it has on scores is illustrated in Figure 3. Suppose that we have a local labelling w in which (i, j) scores $2m$, all the neighbours of (i, j) are well-formed, and all of its predecessors score $2m + 1$. By condition B1 of well-formedness at (i, j), all its successors must also score $2m$. Furthermore since B1 holds at every other node in the neighbourhood of (i, j), the only possible pseudo-scores in the extended neighbourhood of (i, j) are also $2m$ and $2m + 1$. Note that B1 is still true at all nodes in the neighbourhood after an improvement.

Suppose first that $m = 0$. Then it must be the case that (i, j) and its successors are labelled with the empty sequence $\langle\rangle$, and its predecessors have labels of the form

$$\langle \beta_{i-1,j-1}, 0, 0 \rangle,\ \langle \beta_{i-1,j}, 0, 0 \rangle,\ \langle \beta_{i-1,j+1}, 0, 0 \rangle \text{ and } \langle \beta_{i,j-1}, 0, 0 \rangle.$$

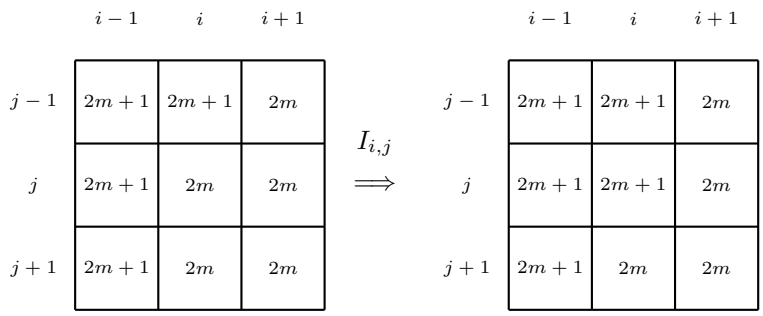

Figure 3: Improving an even score at (i, j).

The improvement function changes the label at (i, j) to

$$\langle f(i, j, \beta_{i-1,j-1}, \beta_{i-1,j}, \beta_{i-1,j+1}), 0, 0\rangle.$$

This preserves B4 and B2 at all nodes in the neighbourhood, and B3 and B5 do not apply. The improved labelling satisfies A1 at (i, j), and A2–A8 are irrelevant.

Next, suppose that $m > 0$. Suppose the label $w_{i,j}$ of (i, j) is of the form $\langle \beta_{i,j}, s, y, C\rangle$. By definition $2m = 1 + 6sn + |C|$, hence $|C|$ is odd, and it can be seen that $w_{i,j}$ must be a backwards configuration,

$$w_{i,j} = \langle \beta_{i,j}, s, y, q_1, r_1, q_2, r_2, \ldots, q_k, r_k, q_{k+1}\rangle$$

for some $k \geq 0$. Similarly, the predecessors of (i, j) are labelled with forwards configurations. Now, the improved label at (i, j) will depend on the last query q_{k+1}. Suppose that q_{k+1} is the query "$\alpha(i_1, j_1) = ?$", for some i_1, j_1. There are three cases. The first case is $(i, j) \notin P_{i_1,j_1}$. Here the improvement function changes the label at (i, j) to the "forwards & waiting" configuration

$$\langle \beta_{i,j}, s, y, q_1, r_1, q_2, r_2, \ldots, q_k, r_k, q_{k+1}, ?\rangle.$$

The second case is where $(i, j) = (i_1, j_1)$. Here the new label at (i, j) is the "forwards & done" configuration

$$\langle \beta_{i,j}, s, y, q_1, r_1, q_2, r_2, \ldots, q_k, r_k, q_{k+1}, \beta_{i,j}\rangle.$$

The third case is where $(i, j) \in P_{i_1,j_1}$ but does not equal (i_1, j_1). Then at least one predecessor of (i, j) is also in P_{i_1,j_1}, and hence by B2 and A7 this predecessor must be labelled with a "forwards & done" configuration of the form

$$\langle \beta_{i',j'}, s, y, q_1, r_1, q_2, r_2, \ldots, q_k, r_k, q_{k+1}, r_{k+1}\rangle$$

(and by B2 it does not matter which predecessor we pick). The label at (i, j) is improved to

$$\langle \beta_{i,j}, s, y, q_1, r_1, q_2, r_2, \ldots, q_k, r_k, q_{k+1}, r_{k+1}\rangle.$$

In no case are we changing s or increasing $|C|$ to $2n + 1$, so A1 is preserved. A2 and A4–A6 are preserved, because adding a reply or a question mark cannot

make a partial computation into a non-computation. For A3, we are not changing s or increasing $|C|$ to $4n+1$, and if this improvement completes a rejecting computation of U, then $|C| \leq 2n$ and hence $s = 0$ by A1. A7 is satisfied by construction. For A8, if we are in the first case, there is no problem because we are not adding any reply. If we are in the third case, then all the replies were already present and satisfying A8 in the chosen predecessor of (i, j). If we are in the second case, and A8 was true before the improvement, then we only need to check what happens if we are adding $\beta_{i,j}$ as one of the replies mentioned in A8 and the node (i', j') mentioned in A8 is in the neighbourhood of (i, j); but in this case A8 is preserved, by B4 and B5 at (i', j') in the unimproved labelling.

For B2, the first case is clear; in the second case, all predecessors of (i, j) are outside $P_{i,j}$ so have a question mark at the end of their label and are thus automatically consistent with the new reply; in the third case it follows from B2 at the chosen predecessor in the unimproved labelling. B3 is not relevant because the extended neighbourhood of (i, j) does not contain a node with an odd pseudo-score and a node with a larger even pseudo-score. B4 is unchanged and B5 is preserved in the second case and irrelevant in the other cases.

The LI instance – improving an odd score to an even score. This corresponds to one step in a backward pass of our exponential algorithm; the effect it has on scores is illustrated in Figure 4. Suppose that we have a local

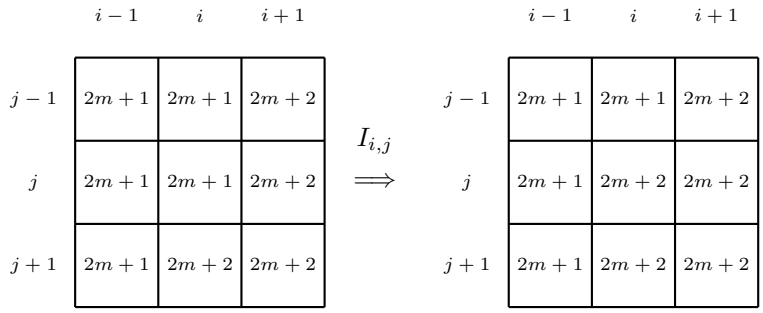

Figure 4: Improving an odd score at (i, j).

labelling w in which (i, j) scores $2m + 1$, all the neighbours of (i, j) are well-formed, and all of its successors score $2m+2$. By condition B1 of well-formedness at (i, j), all its predecessors must also score $2m+1$, and as before since B1 holds at every other node in the neighbourhood of (i, j), the only possible pseudo-scores in the extended neighbourhood of (i, j) are also $2m + 1$ and $2m + 2$. B1 is still true at all nodes in the neighbourhood after an improvement.

The label $w_{i,j}$ of (i, j) must be a forwards configuration

$$\langle \beta_{i,j}, s, y, C \rangle = \langle \beta_{i,j}, s, y, q_1, r_1, q_2, r_2, \ldots, q_k, z \rangle$$

for some $0 \leq k \leq 4n$, where the last entry z is either ? or some reply r_k (or $k = 0$, in which case C is empty).

Case 1: suppose that $s \neq 0$. By A1 and A3, $|C| > 2n$ and U accepts (s, y). There are two subcases: either $k < 4n$ or $k = 4n$.

(1a) Consider the case $k < 4n$. In this case we can always improve the score at (i, j). If (i, j) is at the bottom-right corner (i.e., $i = l - 1$, $j = h - 1$), then

by A7 and the fact that the bottom-right corner is reachable from anywhere, z must be a non-? value r_k. Let q_{k+1} be the next oracle query – depending on k, this may come from a computation of U, g' or N (but always exists, since $k < 4n$). We improve the label at (i, j) to the backwards configuration

$$\langle \beta_{i,j}, s, y, q_1, r_1, q_2, r_2, \ldots, q_k, r_k, q_{k+1} \rangle.$$

Otherwise, (i, j) has at least one successor, and its successors are labelled with backward configurations. Pick the least successor (i', j') of (i, j) and consider its label. By B3, since $k < 4n$, it must have the form:

$$\langle \beta_{i',j'}, s, y, q_1, r_1, q_2, r_2, \ldots, q_k, r_k, q_{k+1} \rangle.$$

The improvement function changes the label at (i, j) to

$$\langle \beta_{i,j}, s, y, q_1, r_1, q_2, r_2, \ldots, q_k, r_k, q_{k+1} \rangle$$

(note that r_k here might be replacing a question mark in the original label).

A1 and A3 are irrelevant here. A2 and A4–A6 either hold by construction (if we are in the corner) or carry over from the successor of (i, j). A7 is irrelevant, and A8 is either irrelevant or, if we are replacing a question mark with a reply r_k, carries over from the successor. B2 and B4 are clear. For B3, notice that before the improvement, by B1 the only pseudo-scores possible in the extended neighbourhood of (i, j) are $2m+1$ and $2m+2$. The labels with odd pseudo-score $2m + 1$ have the form $\langle \ldots, q_k, z \rangle$ with $k < 4n$, so the situation described in B3 cannot occur. For B5, if we replaced a question mark at (i, j) with a reply r_k, then by A7 the query q_k must have been about some other node than (i, j).

(1b) Consider now the case $k = 4n$. Suppose first that (i, j) is the bottom-right corner $(l - 1, h - 1)$. Then as above z is not ?, and by A2 and A4-A6 we have complete computations of U, g' and N on (s, y) as well as of U on $(s+1, y')$ where y' is the output of N on input (s, y). If U rejects $(s + 1, y')$ or $s = a$ then it is not possible to improve the score at (i, j) and in fact such a labelling will yield a witness to our instance of LI; see below. Otherwise, the score at (i, j) can be improved as follows. Let

$$q'_1 = q_{3n+1}, r'_1 = r_{3n+1}, q'_2 = q_{3n+2}, r'_2 = r_{3n+2}, \ldots, q'_n = q_{4n}, r'_n = r_{4n}$$

and let q'_{n+1} be the first oracle query that g' makes on input $(s + 1, y')$. The new label at (i, j) is defined to be

$$\langle \beta_{i,j}, s + 1, y', q'_1, r'_1, q'_2, r'_2, \ldots, q'_n, r'_n, q'_{n+1} \rangle.$$

Suppose finally that (i, j) is not the bottom-right corner. Then by B3 each successor (i', j') of (i, j) has a label of the form

$$\langle \beta_{i',j'}, s + 1, y', q'_1, r'_1, q'_2, r'_2, \ldots, q'_n, r'_n, q'_{n+1} \rangle$$

and we give (i, j) the new label

$$\langle \beta_{i,j}, s + 1, y', q'_1, r'_1, q'_2, r'_2, \ldots, q'_n, r'_n, q'_{n+1} \rangle.$$

For (i, j) in the bottom-right corner, A1 holds because in the new label, $|C| > 2n$ and we have increased s by one; A2 holds by construction; A3 holds in

the new label because the computation of U that we preserve is accepting; A4–A8 are trivial. For (i, j) not in the bottom corner, A1–A8 hold trivially because, except for the first component $\beta_{i,j}$, the new label is a copy of an existing one. In both cases, B2 is preserved because the new score at (i, j) is $2m + 2$; B3 holds by construction and by B2 in the old labelling; B4 and B5 are preserved straightforwardly.

Case 2: Suppose that $s = 0$. By A1, $y = 0$.

(2a) $C \restriction [1, 2n]$ is not yet a complete computation of $U(0, 0)$. Then we improve as in case (1a) above. We do not change s or y so A1 and A3 are preserved; the other conditions are as in (1a).

(2b) $C \restriction [1, 2n]$ is an accepting computation of $U(0, 0)$. We improve just as in case 1.

(2c) $C \restriction [1, 2n]$ is a rejecting computation of $U(0, 0)$. Then by A3, $|C| \leq 4n$. If $|C| < 4n$ we improve as in (2a) above. But if $|C| = 4n$, then we must be at the bottom-right corner, since by B3 any successor of (i, j) scoring $2m+2$ must also contain a rejecting computation of $U(0, 0)$, contradicting A3. Similarly there is no way to improve this labelling at (i, j).

Obtaining the desired sequences $\bar{q}$ and $\bar{r}$. By construction, the only possible witnesses to our instance of LI are certain well-formed local labellings of the bottom-right corner of the graph in which all nodes in the neighbourhood of $(l-1, h-1)$ have odd scores but the improvement function does not increase the score at $(l-1, h-1)$ to the next even number. This is because the improvement function increases the score correctly in all other cases.

The possible labellings of $(l-1, h-1)$ that may occur in such a witness have the following forms (where we make heavy use of the fact that our labelling is well-formed):

- $\langle \beta_{l-1,h-1}, 0, 0, q_1, r_1, q_2, r_2, \ldots, q_{2n}, r_{2n} \rangle$ where $q_1, r_1, q_2, r_2, \ldots, q_n, r_n$ is a rejecting computation of U on $(0, 0)$;
- $\langle \beta_{l-1,h-1}, s, y, q_1, r_1, q_2, r_2, \ldots, q_{4n}, r_{4n} \rangle$ where $q_{3n+1}, \ldots, r_{4n}$ is a rejecting computation of U on $(s+1, N(s, y))$, for some $s < a$ – this gives a solution to the LLS problem, since by condition A3 of well-formedness, U accepts (s, y) with the listed replies;
- $\langle \beta_{l-1,h-1}, a, y, q_1, r_1, q_2, r_2, \ldots, q_{4n}, r_{4n} \rangle$ – this has the maximum score, and always gives a solution, since $s = a$ and U accepts (a, y), again by A3.

In each case we have a pair (s, y) and a set of oracle queries and replies such that (s, y) is a solution to our LLS problem simulated with this oracle, and such that g' run on this solution with this oracle is rejecting, by A8. Hence if we define $A_{\bar{q},\bar{r}}$ to be the oracle built out of these queries and replies, we will have $\mathrm{Def}_f(A_{\bar{q},\bar{r}}, g(A_{\bar{q},\bar{r}}, s, y))$, as required. □

5 The $\forall\hat{\Sigma}^{b+}_1$ consequences of V^1_2

To prove the hard direction of Theorem 7, we need to show that $\forall\hat{\Sigma}^{b+}_1(V^1_2)$ is reducible to LI, provably in PV^+. This is proved by a similar construction to the last section. We will need to be careful about what we mean by "polynomial

time" and "polynomial size". Generally we will have some collection $\mathcal{C}$ of constants that we are using as parameters. "Polynomial" will mean of size $|c|^{O(1)}$ for some first-order constant from $\mathcal{C}$ or of size $||C|_l|^{O(1)}$ or $||C|_b|^{O(1)}$ for some second-order constant from $\mathcal{C}$.

We will use the same definition of a circuit as in Section 4, but in this section the transition function will be an L_2 term (that is, a polynomial time oracle machine) and the length, height and bound l, h, b will be given by simple L_2 terms. As before these will usually have some extra constant symbols as well. We still need some induction over universal formulas to reason about circuits, so will take T_2^{1+} to be our base theory (where in the linear case we had E_1-IND). However we will be able to weaken this to PV^+ at the end of this section.

Let $\mathcal{C}$ be a set of new first- and second-order constant symbols, and let $\mathcal{A}(\mathcal{C})$ contain a constant symbol for each circuit (f, l, h, b) definable from $\mathcal{C}$.

Lemma 24 *V_2^1 is conservative over $T_2^{1+} + \{\mathrm{Iter}_f(\alpha_f) : \alpha_f \in \mathcal{A}(\mathcal{C})\}$ for $\hat{\Sigma}_1^{b+}$ sentences in the language $L_2(\mathcal{C})$.*

Proof As for Lemma 19. Take an $L_2(\mathcal{C}, \mathcal{A}(\mathcal{C}))$ structure M which is a model of $T_2^{1+} + \{\mathrm{Iter}_f(\alpha_f) : \alpha_f \in \mathcal{A}(\mathcal{C})\}$ but not of $\exists y < t\, \phi(y)$, where ϕ is an open $L_2(\mathcal{C})$ formula, t is a simple, closed $L_2(\mathcal{C})$ term and $V_2^1 \vdash \exists y < t\, \phi(y)$. We may assume that M contains no second-order objects except those named by $\mathcal{C}$ and $\mathcal{A}(\mathcal{C})$. Extend M to a bigger structure M^+ by adding to M every sequence that is term-definable in M. Then we claim that every circuit in M^+ already has a computation in M^+.

This is shown as in the proof of Lemma 19. The important difference is that in that proof we first considered an $L_1(\mathcal{C}, \mathcal{A}(\mathcal{C}))$ term $t(x)$ with x the only free variable, and showed that it could be converted to a finite "arithmetic circuit" with gates for the constant symbols from $\mathcal{C}$ and $\mathcal{A}(\mathcal{C})$, evaluating $t(x)$ for all x less than some bound c; we then went on to replace the gates for the symbols α from $\mathcal{A}(\mathcal{C})$ with subcircuits simulating the computation of α. The equivalent thing here is to take an $L_2(\mathcal{C}, \mathcal{A}(\mathcal{C}))$ term $t(x)$ with x the only free variable, and first show that it can be converted to a polynomial size (in the above sense) circuit E with first-order constants from $\mathcal{C}$ as parameters and gates for second-order objects in $\mathcal{C}$ and $\mathcal{A}(\mathcal{C})$ – but this is clear, since such a t is just a description of a polynomial time machine with oracles for $\mathcal{C}$ and $\mathcal{A}(\mathcal{C})$. Given such an E, we can write down the transition function for a circuit F in which each gate for a constant $\alpha \in \mathcal{A}(\mathcal{C})$ in E is replaced with a complete circuit simulating a computation of α (we also replace all other gates in E with large, dummy circuits so that the structure of F is kept uniform). Then given any computation of F, we use $\hat{\Pi}_1^{b+}$-IND first to show that each simulation of an $\alpha \in \mathcal{A}(\mathcal{C})$ computes the same values as α itself does, and then that the values computed in E are matched by those at corresponding points in F.

Finally, by constructing computations to eliminate quantifiers, we can show that such an M^+ is also a model of full bounded comprehension, and it follows that it also satisfies all the bounded consequences of V_2^1. □

We will use a somewhat nonstandard, "iterative" definition of PLS. This is completely analogous to the modified version of LLS used in the proof of Theorem 23. It is easily seen to be equivalent to the usual definition.

Lemma 25 *Let $\exists y{<}t\,\phi(y)$ be an $\hat{\Sigma}^{b+}_1$ formula in $L_2(\mathcal{C})$. If $T^{1+}_2 + \{\mathrm{Iter}_f(\alpha_f) : \alpha_f \in \mathcal{A}(\mathcal{C})\}$ proves $\exists y{<}t\,\phi(y)$, then there is an $L_2(\mathcal{C})$ definable circuit D, an $L_2(\mathcal{C},\alpha_f)$ PLS problem P (with no parameters except ones that are definable from $\mathcal{C}$ and α_f) and $L_2(\mathcal{C},\alpha_f)$ terms g,h such that, provably in T^{1+}_2, if (i,x) is a solution to $P(\alpha_f)$ then either $g(i,x)$ witnesses that $\mathrm{Iter}_D(\alpha_f)$ is false or $h(i,x)$ witnesses that $\exists y{<}t\,\phi(y)$.*

Proof Just as for Lemma 22. □

Theorem 26 *$\forall\hat{\Sigma}^{b+}_1(V^1_2)$ is reducible to* LI*, provably in T^{1+}_2.*

Proof The construction of an LI instance is exactly the same as in the proof of Theorem 23, except that now the length n of our straight-line programs is a polynomial (in the logarithm of the parameters) rather than a constant. This is not a problem, since the smash function is available so we can still bound the size of the scores.

A more subtle difference is that in that theorem we just wanted to show that certain consequences of V^1 were provable from $V^0 + \mathrm{LI}$. Here we want to show something stronger, that the problem of witnessing a $\forall\hat{\Sigma}^{b+}_1$ consequence of V^1_2, considered as a search problem, is effectively reducible to solving LI, provably in T^{1+}_2.

So suppose that $V^1_2 \vdash \forall\bar{X}\exists v < t(\bar{X})\,\phi(\bar{X},v)$. Then we may replace the quantified variables $\bar{X}$ with some constant symbols $\bar{C}$ and apply Lemmas 24 and 25 to get an $L_2(\mathcal{C})$ PLS problem P, a tuple of $L_2(\mathcal{C})$ terms $f(i,j,x_1,x_2,x_3), l, h, b$ and two $L_2(\mathcal{C})$ term $g_1(\alpha,s,y)$ and $g_2(\alpha,s,y)$ such that

$$T^{1+}_2 \vdash \forall\alpha,s,y,\ \text{“}(s,y)\text{ is a solution to }P(\alpha)\text{”} \rightarrow [\neg\mathrm{Def}_f(\alpha,g_1(\alpha,s,y)) \vee \phi(\bar{C},g_2(\alpha,s,y))]$$

where $\neg\mathrm{Def}_f(\alpha,u)$ expresses that u is a pair $\langle i,j\rangle$ witnessing that α is not a correct computation of the circuit defined by f, l, h and b, and where we assume that g_2 is implicitly bounded by $t(\bar{C})$.

As in the proof of Theorem 23, we can now replace α with an “oracle” A_z defined in terms of some first-order string w of oracle queries and replies, to get, rearranging things slightly,

$$T^{1+}_2 \vdash \forall z,s,y,\ \text{“}(s,y)\text{ is a solution to }P(A_z)\text{”} \wedge \mathrm{Def}_f(A_z,g_1(A_z,s,y)) \rightarrow \phi(\bar{C},g_2(A_z,s,y)).$$

Now to get our reducibility result, it suffices to give an instance of LI which is polynomial time definable from the constants $\bar{C}$ and from any solution of which we can extract z,s,y satisfying the conjunction on the left hand side of the above implication. But this is exactly what we get from the construction in the proof of Theorem 23, adapted for L_2. □

Finally, we can use a characterization of the $\forall\hat{\Sigma}^{b+}_1$ consequences for T^{1+}_2 to weaken our base theory from T^{1+}_2 to PV^+.

Theorem 27 *$\forall\hat{\Sigma}^{b+}_1(V^1_2)$ is reducible to* LI*, provably in* PV^+.

Proof We will show how the proof of Theorem 26 can be adapted to give this stronger result. In the last paragraph of that proof, we obtained the following:

$$T_2^{1+} \vdash \forall z, s, y, \Phi(z, s, y) \to \phi(\bar{C}, g_2(A_z, s, y)).$$

where we have written $\Phi(z, s, y)$ to express the quantifier free, $L_2(\bar{C})$ formula

$$\text{“}(s, y) \text{ is a solution to } P(A_z)\text{”} \wedge \mathrm{Def}_f(A_z, g_1(A_z, s, y)).$$

This is a $\forall\hat{\Pi}_1^b$ sentence, provable in T_2^{1+}. Hence by the PLS witnessing theorem ([7] – see for example [12] for a strengthened version which works over PV), there is a PLS problem $Q = (N_Q, U_Q)$ in the language $L_2(\bar{C})$, which takes (z, s, y) as parameters, such that

$$\mathrm{PV}^+ \vdash \forall\, z, s, y, r, x, \text{“}(r, x) \text{ is a solution to } Q((z, s, y))\text{”} \wedge \Phi(z, s, y) \to \phi(\bar{C}, g_2(A_z, s, y)).$$

So to complete the proof it is sufficient to alter the LI instance constructed in the proof of Theorem 26 in such a way that from any solution w of our new instance, not only can we recover z, s and y satisfying Φ, but we can also recover a solution (r, x) to the PLS problem Q run with parameters (z, s, y).

Let us write $\mathcal{I}$ for the LI instance from Theorem 26. It has a graph G, which is an $l \times h$ grid; it has scoring, initialization and improvement functions S, E and I; the labels are bounded by b and the scores go up to c. We may assume that solving the PLS problem $Q = (N_Q, U_Q)$ consists in finding witness r, x to a sentence

$$\neg U_Q(0, 0) \vee \exists r, x{<}t\, (U_Q(r, x) \wedge \neg U_Q(r + 1, N_Q(r, x))) \vee \exists x{<}t\, U_Q(t, x)$$

(where we omit the parameter (z, s, y) of U_Q and N_Q) and that the range of N_Q is bounded by t. All numbers, functions and sets can be taken to be definable in $L_2(\bar{C})$.

We will define a new instance $\mathcal{J}$ of LI. Essentially this will be an isomorphic copy of $\mathcal{I}$, modified so that once a solution to $\mathcal{I}$ is found, this solution is propagated to every node of the graph, and then N_Q is iterated at each node separately, using the solution as a parameter, until a solution to Q is found.

The graph and labels. The underlying graph of $\mathcal{J}$ is just G. The labels are triples (ω, i, x), where $\omega < b$ has the form of a label from $\mathcal{I}$, i is from $[0, t] \cup \{-1\}$ and x is from $[0, t) \cup \{-1\}$. The labels in $\mathcal{J}$ can have one of two forms. “First-stage” labels have the form $(\omega, -1, -1)$; their pseudo-score is the pseudo-score of ω. “Second-stage” labels have the form (ω, r, x) with $r, x \geq 0$; their pseudo-score is (the pseudo-score of ω) $+ r + 1$.

The initialization function assigns to every node the label $(\langle\rangle, -1, -1)$.

Let w be a local labelling of a point (i, j) and its neighbours, and let (ω, r, x) be the label of (i, j) under w. Recall that ω has the form

$$\langle \beta, s, y, q_1, r_1, \ldots, q_k/r_k/? \rangle.$$

We will write s_ω and y_ω for the elements s and y from this sequence, and z_ω for the sequence of complete pairs of oracle queries and replies.

The scoring function. We can now give the conditions for w to be well-formed at (i, j). We will use the definitions A1-A8 and B1-B5 from the proof of Theorem 23, suitably adapted for the L_2 setting. If w is well-formed at (i, j), the score at (i, j) is just the pseudo-score of (ω, r, x).

If (ω, r, x) is first-stage, w is well-formed at (i, j) if B1 holds, A1-A8 hold (for ω), B2-B5 hold restricted to only the nodes in the neighbourhood of (i, j) with first-stage labels (for the first components ω' of their labels), and lastly the extra condition B6 holds:

B6. In the neighbourhood of (i, j), no node with a second-stage label can have a successor with a first-stage label; and no two nodes with respectively first- and second-stage labels can share the same pseudo-score.

If (ω, r, x) is second-stage, w is well-formed at (i, j) if B6 holds and ω also satisfies $\Phi(z_\omega, s_\omega, y_\omega)$ and $U((z_\omega, s_\omega, y_\omega), r, x)$.

The improvement function. We will first consider improving a first-stage label. Suppose that w is a labelling in which (i, j) scores an even number $2m$, in which all the neighbours of (i, j) are well-formed, and in which all the predecessors of (i, j) score $2m + 1$. By B1, all the successors of (i, j) score $2m$, and in fact the only pseudo-scores possible in the extended neighbourhood of (i, j) are $2m$ and $2m + 1$. Thus by B6 all labels in the extended neighbourhood are first-stage. Therefore we can improve the label at (i, j) by using the improvement function of $\mathcal{I}$, applied just to the first components of the labels; this will always improve scores correctly.

Now suppose that (i, j) scores an odd number $2m+1$, its successors all score $2m + 2$, and all its neighbours are well-formed. Then by B1 all its predecessors score $2m + 1$, and $2m + 1$ and $2m + 2$ are the only pseudo-scores appearing in the extended neighbourhood. By B6 all the predecessors of (i, j) are first-stage.

If (i, j) has no successor, or has only first-stage successors, then the improvement function is again the improvement function of $\mathcal{I}$ applied to first components. This will improve scores correctly, except in the case that these components form a local labelling for which the improvement function of $\mathcal{I}$ fails itself to improve the score in $\mathcal{I}$. But in this case, this labelling is a solution of $\mathcal{I}$, so (i, j) must be at the bottom-right corner of G and the label of (i, j) must have the form $(\omega, -1, -1)$ satisfying $\Phi(z_\omega, s_\omega, y_\omega)$ – note that although these properties were proved in Theorem 26 in the context of a stronger base theory, they really only need PV^+. The improvement in this case is to replace this label with $(\omega, 0, 0)$. If this is not a valid second-stage label, it can only be because $\neg U_Q((z_\omega, s_\omega, y_\omega), 0, 0)$, which gives us a solution to the PLS problem Q, as required.

If (i, j) has a second-stage successor, then by B6 all of its successors are second stage. The improvement is simply to replace the label of (i, j) with the label of its least successor.

Finally, the improvement function for a second-stage label (ω, r, x) with $r < t$ is to replace the label with $(\omega, r+1, x')$ where $x' = N_Q((z_\omega, s_\omega, y_\omega), r, x)$. If this improvement changes a labelling from being well-formed into not being well-formed, it must be because $U_Q((z_\omega, s_\omega, y_\omega), r, x)$ but $\neg U_Q((z_\omega, s_\omega, y_\omega), r+1, x')$, giving us a solution to the PLS problem Q. If $r = t$, then no improvement is possible, but the labelling must already contain a solution to the PLS problem. □

6 Postponed proofs

6.1 From Section 1

We sketch how our results, formulated officially in terms of theories about sequences, can be transferred to the more common framework where the second-order sort consists of sets. In this subsection, for any theory T axiomatized by one of the standard schemes mentioned above, we let T_{seq}, T_{set} denote the sequence and set versions of T, respectively.

The possibility of moving back and forth between results about T_{seq} and T_{set}, for all T we are interested in, is essentially due to the existence of suitably well-behaved interpretations set of T_{set} in T_{seq} and seq in the opposite direction. These interpretations should be chosen so that they do not change the quantifier complexity of sentences (at least on a level meaningful to us) and have the additional property that $T_{seq} \vdash \phi \leftrightarrow ((\phi)^{\mathsf{seq}})^{\mathsf{set}}$ and $T_{set} \vdash \phi \leftrightarrow ((\phi)^{\mathsf{set}})^{\mathsf{seq}}$ for all relevant ϕ.

Consider first the linear setting, i.e. Theorem 8. Here, we do not count alternations of first-order quantifiers, so we have a rather wide choice of set and seq: for example, a set is a 0-1 valued sequence, while a sequence is a set of ordered pairs with the existence and uniqueness properties. Assume that

$$V^1_{set} \vdash \forall \bar{X}\, \phi(\bar{X}),$$

where ϕ is bounded first-order. This implies

$$V^1_{seq} \vdash \left(\forall \bar{X}\, \phi(\bar{X})\right)^{\mathsf{set}},$$

and hence, by our Theorem 8 for sequence theories,

$$V^0_{seq} \vdash \left(\forall \bar{X}\, \phi(\bar{X})\right)^{\mathsf{set}} \vee \neg \mathrm{LI}.$$

We can then get

$$V^0_{set} \vdash \left(\left(\forall \bar{X}\, \phi(\bar{X})\right)^{\mathsf{set}}\right)^{\mathsf{seq}} \vee (\neg \mathrm{LI})^{\mathsf{seq}},$$

which is the same as

$$V^0_{set} \vdash \forall \bar{X}\, \phi(\bar{X}) \vee (\neg \mathrm{LI})^{\mathsf{seq}}$$

or

$$V^0_{set} + (\mathrm{LI})^{\mathsf{seq}} \vdash \forall \bar{X}\, \phi(\bar{X}).$$

The principle $(\mathrm{LI})^{\mathsf{seq}}$ is a little awkward, as there are objects naturally viewed as sets but which we have formally coded as sequences (e.g. the graphs in the original LI principle of Section 1.3), which sequences we now have to interpret as (different) sets, via seq. However, it is easy to formulate an equivalent version of LI which would be more natural in the context of V^0_{set}. This version is then our "official" principle characterizing the $\forall\Sigma^B_0$ consequences of V^1_{set} over V^0_{set}.

The polynomial setting is similar, except that now we need to take into account first-order quantifier complexity, so we have to be more careful with our choice of interpretations. Here, sequences should not be regarded as sets of ordered pairs, but referred to by means of their bit-graphs, where the bit-graph of a sequence X is a relation $\tilde{X}$ such that $\tilde{X}(i,j)$ holds if the j-th bit of $X(i)$ is 1. Contexts like $X(i) = k$ can then be translated as $\forall j < |k|\, (\tilde{X}(i,j) \leftrightarrow$

$bit(k, j) = 1$). This ensures that the translation of a $\hat{\Sigma}^{b+}_i$ formula in the language of sequences remains $\hat{\Sigma}^{b+}_i$ in the language of sets.

A separate issue is raised by our decision not to include second order equality in our languages, contrary to standard practice. In the polynomial setting, this is justified by the fact that we are not really interested in $\forall\hat{\Sigma}^{b+}_1$ with second-order $=$, because equality of second order objects is not a Δ^b_1 notion (it is Π^b_1). In the linear setting, the usual V^0 (with second-order $=$) contains the extensionality axiom, so every $\forall\Sigma^B_0$ statement with second-order $=$ is equivalent to a $\forall\Sigma^B_0$ statement without it, obtained by replacing each occurrence of $=$ by its obvious A_1 definition.

6.2 From Section 3

We prove the remaining direction of Lemma 13, that U^1_2 is $\forall\Sigma^{1,b}_1$ conservative over Γ. By a *functional* we mean a function which takes some arguments (which may be first- or second-order) and outputs a second-order object.

Definition 28 *Extend our language by adding, for each triple $f(i, x, \bar{Z})$, $l(\bar{Z})$, $b(\bar{Z})$ of L_2 terms (with only the parameters shown and with l and b simple), the second-order function symbol $\Theta_{f,l,b}(\bar{Z})$. Extend Γ by adding axioms expressing that $\Theta_{f,l,b}(\bar{Z})$ is the iterator of $f(i, x, \bar{Z})$ with length $l(\bar{Z})$ and bound $b(\bar{Z})$. Note that every PSPACE function (formalized as in the remark after Definition 12) with both input and oracles given by $\bar{Z}$ can now be written as a term $G(\bar{Z})$ of the form $\Theta_{f,l,b}(\bar{Z})(l(\bar{Z}) - 1)$. Let us call terms of this form* PSPACE terms.

Further extend the language and the theory Γ by adding, for each PSPACE term $G(\bar{Z}, i)$, a second-order function symbol $\Phi_G(\bar{Z}, a)$ with axioms $|\Phi_G(\bar{Z}, a)|_l = a$, $|\Phi_G(\bar{Z}, a)|_b = b(\bar{Z})$, where $b(\bar{Z})$ is the bound appearing in the iterator used in G, and $\Phi_G(\bar{Z}, a)(i) = G(\bar{Z}, i)$. We will call the terms $\Phi_G(\bar{Z}, a)$ the PSPACE functional terms*; note that every PSPACE functional can be written this way.*

We will call the extended theory PSF.

Note that PSF is a conservative extension of Γ and that it contains Herbrand functions for every axiom of Γ, so that any subset of a model of Γ which is closed under PSPACE functional terms is also a model of Γ.

Lemma 29 *In* PSF*, every PSPACE term $G(\bar{X})$ is equivalent to a PSPACE term defined by iterating a polynomial time function which makes exactly one oracle query to $\bar{X}$ at each step of the iteration.*

Proof Suppose G is defined by iterating a polynomial time function g with oracle $\bar{X}$ for l steps with size bound b. Let t be the maximum running time of g on inputs bounded by b. We can simulate l iterations of g using one iterator of length lt, in the obvious way, as in the proof of Lemma 15. This iterator makes at most one oracle call at each step, since g does; and we can add a dummy call if necessary.

To complete the proof, fix values for $\bar{X}$ and let W be the iterator defining G for these inputs and V be the iterator in our length lt simulation. For each $i < l$, open induction is enough to show that our simulation of g recorded in $V(it) \dots V((i + 1)t - 1)$ is a correct computation of g, and another use of open induction then shows that the values of g recorded in $V(t - 1) \dots V(lt - 1)$ correspond to the values computed in $W(1) \dots W(l)$. □

Lemma 30 *In* PSF*, the composition of two PSPACE functionals is already a PSPACE functional. That is, if $G(\bar{X}, Y)$ is a PSPACE term and we replace queries "$Y(i) = ?$" with queries "$H(\bar{X}, i) = ?$" for some PSPACE term H, then the resulting function $K(\bar{X}) = G(\bar{X}, \lambda i.H(\bar{X}, i))$ is definable by a PSPACE term.*

Proof Similar to the proof of Lemma 29. □

Lemma 31 *In* PSF*, any first-order, bounded formula in the language L_2 is equivalent to an open formula and in particular to one of the form $G(\bar{X}) = 1$ for some PSPACE term G.*

Proof Standard PSPACE argument, formalized using Lemma 30. □

Lemma 32 *In* PSF*, let $\Phi(X, a, j)$ be a parametrized family of PSPACE functionals, given by some PSPACE term $G(X, i, j)$ (there may also be some other parameters which we do not show). Let t be a simple term. Then there is a parametrized family of PSPACE functionals $\mathrm{Iter}_\Phi(W, a, j)$ such that for all sequences W of length a and for all parameters $j < |t(a)|$,*

1. $\mathrm{Iter}_\Phi(W, a, 0) = W$
2. $\mathrm{Iter}_\Phi(W, a, j+1) = \Phi(\mathrm{Iter}_\Phi(W, a, j), a, j)$.

In other words, the set of PSPACE functionals is closed under polylogarithmic iteration.

Proof This is a polylogarithmic depth version of Lemma 30. Let us write $G(X, i, j)$ as $G_j(X, i)$. We can think of this as being computed by a machine of the form described in Lemma 29, which takes a parameter $i < a$, has bound b and runs for some number l of iterations (where b and l can be taken to be the same for all the machines G_j, once we fix W and a), at each step making exactly one query to X. $\mathrm{Iter}_\Phi(W, a, j)$ has to simulate an array $G_{j-1}, G_{j-2}, \ldots, G_0$ of such machines, where at each step of a machine, any query "$X(i) = ?$" is replaced with a query to the next machine down. So for each $0 < k < j$, each step of G_k requires a full run of l steps of G_{k-1} (and so on, recursively, down the array).

We will use three numbers y, v and i to keep track of our progress in this simulation. y is uniquely decomposeable as a sum $y = l^{j-1}y_{j-1} + \ldots + ly_1 + y_0$, where each y_k is the current position in the simulation of machine G_k. v codes a tuple $v_{j-1}, \ldots, v_0$, where v_k is the value at the current step in the iterator of G_k. i codes a tuple $i_{j-1}, \ldots, i_0$, where i_k is the parameter for the current run of G_k. Note that we may bound v by b^j and i by a^j.

The simulation is given by iterating l^j times a function $f(y, \langle v, i\rangle)$ which given a configuration $\langle v, i\rangle$ of all machines at position y computes the configuration $\langle v', i'\rangle$ for position $y + 1$.

The details are standard and are left to the reader. Open induction is enough to prove that the functional has the required properties. □

Lemma 33 *U_2^1 is $\forall\Sigma_1^{1,b}$-conservative over* PSF*. Lemma 13 follows, since* PSF *is a conservative extension of Γ.*

Proof We will use a standard model-theoretic form of the witnessing argument, see e.g. [19]. Let M be any model of PSF. We will first extend M in a $\Pi^{1,b}_1$-elementary way to a structure N with the property that, for any PSPACE term G, if $N \models \forall \bar{X}\, \exists Y < t(\bar{X})\, G(\bar{X}, Y) = 1$ then this is witnessed in N by some PSPACE functional. The induction step in an instance of $\Sigma^{1,b}_1$-LIND has this form, so in N all induction steps will be uniformly witnessed by PSPACE functionals, and by Lemma 32 we can combine these to give one functional witnessing any (polylogarithmic) number of induction steps, hence showing that $N \models U^1_2$.

Enrich the language of PSF with names for every first- and second-order element of M, together with a new countable set $C_0, C_1, \ldots$ of second-order constant symbols. Enumerate as $\theta_0, \theta_1, \ldots$ all $\forall\Sigma^{1,b}_1$ sentences in this language, where each θ_i has the form $\forall \bar{X}\, \exists Y < t(\bar{X})\, G(\bar{X}, Y) = 1$, for some G, t and some arity of the tuples. To make an argument easier below, without loss of generality we assume that G does not directly query $|Y|_b$ or $|Y|_t$.

Let T_0 be the $\Pi^{1,b}_1$ diagram of M' in our enriched language (without the new constant symbols $C_0, C_1, \ldots$). We will construct N by building a chain $T_0 \subseteq T_1 \subseteq \ldots$ of universal, consistent theories. Suppose we have constructed up to T_i. For T_{i+1}, consider the sentence $\theta_i \equiv \forall \bar{X}\, \exists Y < t(\bar{X})\, G(\bar{X}, Y) = 1$. There are two cases.

Case 1: $T_i \vdash \forall \bar{X}\, \exists Y < t(\bar{X})\, G(\bar{X}, Y) = 1$. In this case, we put $T_{i+1} := T_i$.

Case 2: $T_i \nvdash \forall \bar{X}\, \exists Y < t(\bar{X})\, G(\bar{X}, Y) = 1$. In this case, we define T_{i+1} to be $T_i + \{\forall Y < t(\bar{C})\, G(\bar{C}, Y) \neq 1\}$ where $\bar{C}$ is a tuple of some of the new constant symbols $C_0, C_1, \ldots$, chosen so that none of these symbols have already appeared in $T_0, \ldots, T_i$ or in θ_i. Note that T_{i+1} is consistent and universal.

Now let T be the union of the chain $T_0 \subseteq T_1 \subseteq \ldots$. Then T is consistent, so has a model N. T is universal, so we may assume without loss of generality that all elements of N are named by closed terms in the language. Hence the first-order elements of N are precisely the closure of the elements of M and the new constants under PSPACE terms, and similarly for the second-order elements and PSPACE functionals.

Now let $G(\bar{X}, Y)$ be any PSPACE term with parameters from N and suppose $N \models \forall \bar{X}\, \exists Y < t(\bar{X})\, G(\bar{X}, Y) = 1$. By the previous paragraph, we may assume that the parameters in G are actually either from M or are named by the new constants. Hence this sentence is equivalent to one of the sentences θ_i. It is true in N and hence must be consistent with T, so in the construction of T_{i+1} we must have been in case 1. So $T_i \vdash \forall \bar{X}\, \exists Y < t(\bar{X})\, G(\bar{X}, Y) = 1$. By Herbrand's theorem and Lemma 30, since T_i is universal, there are PSPACE functionals $\Phi_{G_1}, \ldots, \Phi_{G_k}$, for some $k \in \mathbb{N}$, such that

$$T_i \vdash \forall \bar{X} \bigvee_{j=1}^{k} G(\bar{X}, \Phi_{G_j}(\bar{X}, t(\bar{X}))) = 1.$$

Let H be the PSPACE functional which tests the sequences $Y_j := \Phi_{G_j}(\bar{X}, t(\bar{X}))$ in turn for $j = 1, \ldots, k$ and outputs the first Y_j which satisfies $G(\bar{X}, Y_j)$. Then H is the PSPACE functional witnessing our $\forall\Sigma^{1,b}_1$ sentence, as required.

To show that N is a model of U^1_2, let $G(Y, j)$ be any PSPACE term (possibly with other parameters), let $a \in N$, let t be a simple term, and suppose that

$$N \models \exists Y < a\, G(Y, 0) \wedge \forall j < |t(a)|\, [\exists Y < a\, G(Y, j) \rightarrow \exists Y < a\, G(Y, j+1)].$$

Let $W < a$ be such that $G(W, 0)$ and rearrange the second conjunct as

$$\forall j < |t(a)|\, \forall Y < a\, \exists Y' < a\, [G(Y, j) \to G(Y', j+1)].$$

Then by the construction of N there is a PSPACE functional F, with parameters in N, such that in N

$$\forall Y < a\, [G(Y, j) \to G(F(Y, j, a), j+1)].$$

By Lemma 32 we may iterate this functional to get a PSPACE functional Iter_F such that $\mathrm{Iter}_F(W, 0, a) = W$ and for all $j < |t(a)|$, $\mathrm{Iter}_F(W, j+1, a) = F(\mathrm{Iter}_F(W, j, a), j, a)$.

Open induction is now enough to show that $G(\mathrm{Iter}_F(W, j, a), j)$ holds for all $j < |t(a)|$. This gives us $\Sigma_1^{1,b}$-LIND. □

6.3 From Section 4

We prove Lemma 21: Let ϕ be an open L_1 formula, t an L_1 term and suppose that E_1−IND $\vdash \forall \bar{X}\, \exists y < t(\bar{X})\, \phi(\bar{X}, y)$. Then witnessing the right hand side is provably reducible to solving an LLS problem. That is, there is an LLS problem $P = \langle C, N, s\rangle$ and an L_1 term g such that

$$E_1\text{−IND} \vdash \forall \bar{X}, z\, [\text{“z is a solution to } P(\bar{X})\text{”} \to g(\bar{X}, z) < t(\bar{X}) \wedge \phi(\bar{X}, g(\bar{X}, z))].$$

Proof The proof is a rather standard witnessing argument, based on the one relating T_2^1 and PLS in [7]. E_1−IND may be formulated as a sequent calculus with some quantifier-free initial sequents corresponding to the basic axioms, rules for bounded quantifier introduction and the E_1-induction rule:

$$\frac{\Gamma, \psi(c), \longrightarrow \psi(c+1), \Delta}{\Gamma, \psi(0) \longrightarrow \psi(t), \Delta}$$

for $\psi \in E_1$.

By free-cut elimination, every sequent $\Gamma \longrightarrow \Delta$ consisting of E_1 formulas and provable in this system has a proof in which only sequents of (at most) E_1 formulas appear. By induction on the length of a free-cut free proof, we show that $\Gamma \longrightarrow \Delta$ can be provably witnessed by an LLS problem, in the following strong sense: there is an LLS problem $P = \langle C, N, s\rangle$ such that:

- inputs to P are of the form $\langle \bar{X}, \bar{w}\rangle$ where $\bar{X}$ are the free variables of $\Gamma \longrightarrow \Delta$ and $\bar{w}$ is a tuple of the same length as the number of formulas in Γ,
- E_1−IND proves that conditions (1) and (2) from Definition 20 are always satisfied,
- E_1−IND proves that solutions v to P on input $\langle \bar{X}, \bar{w}\rangle$ have the property that if $\bar{w}$ consists of witnesses for the existential quantifiers of all formulas from $\Gamma(\bar{X})$, then v witnesses the existential quantifiers in some formula from $\Delta(\bar{X})$.

Obviously, this will suffice to prove the theorem.

The steps for initial sequents, propositional and structural rules other than cut, and the $\exists$-introduction rules are straightforward, while the $\forall$-introduction rules never appear. Hence, we sketch only the two most interesting cases: cut and induction, leaving verification of correctness in E_1-IND to the reader.

In the case where the last inference in the proof is a cut:

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

we have two LLS problems $P_1 = \langle C_1, N_1, s_1 \rangle$ and $P_2 = \langle C_2, N_2, s_2 \rangle$, where P_1 takes as input $\langle \bar{X}, \bar{w}_\Gamma \rangle$ and produces solutions v such that if $\bar{w}_\Gamma$ witnesses Γ then v witnesses ψ or some formula in Δ, while P_2 takes as input $\langle \bar{X}, \bar{w}_\Gamma, w_\psi \rangle$ and produces solutions u such that if $\bar{w}_\Gamma$ witnesses Γ and w_ψ witnesses ψ then u witnesses some formula in Δ.

We define a new problem $P = \langle C, N, s \rangle$ witnessing $\Gamma \longrightarrow \Delta$ essentially by "composing" P_1 and P_2. Possible solutions to P on input $\langle \bar{X}, \bar{w}_\Gamma \rangle$ in the nontrivial case when $\bar{w}_\Gamma$ does witness Γ will have one of the following three forms: (i) $\langle 0, x \rangle$ where x is a possible solution to P_1 on input $\langle \bar{X}, \bar{w}_\Gamma \rangle$, (ii) $\langle 1, v, x \rangle$ where x is a possible solution to P_2 on input $\langle \bar{X}, \bar{w}_\Gamma, v \rangle$ and v witnesses ψ, (iii) u witnessing Δ. Clearly, this lets us define $s(\bar{X}, \bar{w}_\Gamma)$ as an L_1 term in: $s_1(\bar{X}, \bar{w}_\Gamma)$, some common bound on $s_2(\bar{X}, \bar{w}_\Gamma, v)$ for possible witnesses v to ψ, and the maximum of the bounds on the existential quantifiers in Δ. The neighbourhood function N is defined by (the inputs $\bar{X}, \bar{w}_\Gamma$ are suppressed):

- $N(\langle 0, x \rangle)$ is $\langle 0, N_1(x) \rangle$ unless $x = N_1(x)$, in which case x witnesses ψ or Δ and $N(\langle 0, x \rangle)$ equals $\langle 1, x, 0 \rangle$ or x, respectively,
- $N(\langle 1, v, x \rangle)$ equals $\langle 1, v, N_2(v, x) \rangle$ unless $x = N_2(v, x)$, in which case x witnesses Δ and $N(\langle 1, v, x \rangle)$ is defined to be x,
- $N(u) := u$ for u of type (iii).

Let m_1 and m_2 be bounds on the cost function for P_1 and P_2 respectively. To define the cost function, fix some term $\tilde{m}(\bar{X})$ which dominates $m_2(\bar{X}, \bar{w}_\Gamma, v)$ whenever v is below the bound on the quantifier in $\psi(\bar{X})$. Then:

- $C(\langle 0, x \rangle) := C_1(x)$,
- $C(\langle 1, v, x \rangle) := m_1 + C_2(v, x)$,
- $C(u) := m_1 + \tilde{m}$ for u of type (iii).

If the last inference in the proof uses the E_1-induction rule

$$\frac{\Gamma, \psi(c) \longrightarrow \psi(c+1), \Delta}{\Gamma, \psi(0) \longrightarrow \psi(t), \Delta}$$

and $P' = \langle C', N', s' \rangle$ is the *LLS* problem witnessing the upper sequent, we construct P witnessing the lower sequent as a $t(\bar{X})$-long iteration of P'. P' takes as input $\langle \bar{X}, c, \bar{w}_\Gamma, w_c \rangle$ and produces solutions v such that if $\bar{w}_\Gamma$ witnesses Γ and w_c witnesses $\psi(c)$ then v witnesses $\psi(c+1)$ or one of the formulas in Δ. In the nontrivial case where $\bar{w}_\Gamma, w_0$ witness what they should, possible solutions to P on input $\langle \bar{X}, \bar{w}_\Gamma, w_0 \rangle$ will have one of the following forms: (i) $\langle c, w_c, x \rangle$

where $c < t(\bar{X})$, w_c witnesses $\psi(c)$ and x is a possible solution to P' on input $\langle \bar{X}, c, \bar{w}_\Gamma, w_c \rangle$, (ii) v witnessing $\psi(t)$ or Δ. The bound $s(\bar{X}, \bar{w}_\Gamma, w_0)$ is defined accordingly. The neighbourhood function is defined by (the inputs $\bar{X}, \bar{w}_\Gamma, w_0$ are suppressed):

- $N(\langle c, w_c, x \rangle) := \langle c, w_c, N'(c, w_c, x) \rangle$ unless $x = N'(c, w_c, x)$; in the latter case, x witnesses $\psi(c+1)$ or Δ, and we let $N(\langle c, w_c, x \rangle)$ equal $\langle c+1, x, 0 \rangle$ if $c+1 < t$ and x witnesses $\psi(c+1)$, or simply x if $c+1 = t$ (and hence x witnesses $\psi(t)$) or if x witnesses Δ ,
- $N(v) := v$ for v of type (ii).

To define cost, let $\tilde{m}(\bar{X})$ dominate $m'(\bar{X}, c, \bar{w}_\Gamma, w_c)$ for all $c < t(\bar{X})$ and w_c below the bound on the quantifier in $\psi(c, \bar{X})$, where m' is a bound on the cost function of P'. Then:

- $C(\langle c, w_c, x \rangle) := c\tilde{m} + C'(c, w_c, x)$,
- $C(v) := t\tilde{m}$ for v of type (ii). □

References

[1] J. Avigad. Plausibly hard combinatorial tautologies. In P. Beame and S. Buss, editors, *Proof Complexity and Feasible Arithmetics*, pages 1–12. AMS, 1997.

[2] P. Beame, S. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi. The relative complexity of NP search problems. *Journal of Computer and System Sciences*, 57(1):3–19, 1998.

[3] A. Beckmann and S. Buss. Polynomial local search in the polynomial hierarchy and witnessing in fragments of bounded arithmetic. Preprint, 2008.

[4] J. Buresh-Oppenheim and T. Morioka. Relativized NP search problems and propositional proof systems. In *IEEE Conference on Computational Complexity*, pages 54–67, 2004.

[5] S. Buss. *Bounded Arithmetic*. Bibliopolis, 1986.

[6] S. Buss. Chapter 1: An introduction to proof theory & Chapter 2: First-order proof theory of arithmetic. In S. Buss, editor, *Handbook of Proof Theory*. Elsevier, 1998.

[7] S. Buss and J. Krajíček. An application of Boolean complexity to separation problems in bounded arithmetic. *Proceedings of the London Mathematical Society*, 69:1–21, 1994.

[8] S. Cook. Feasibly constructive proofs and the propositional calculus. *Proceedings of the 7th Annual ACM Symposium on Theory of computing*, pages 83–97, 1975.

[9] S. Cook. Bounded reverse mathematics. Plenary lecture for CiE, 2007.

[10] S. Cook and P. Nguyen. *Logical Foundations of Proof Complexity*. 2009. Book, to appear.

[11] M. Fairtlough and S. Wainer. Hierarchies of provably recursive functions. In S. Buss, editor, *Handbook of Proof Theory*. Elsevier, 1998.

[12] F. Ferreira. What are the $\forall\Sigma^b_1$-consequences of T^1_2 and T^2_2? *Annals of Pure and Applied Logic*, 75(1):79–88, 1995.

[13] J. Krajíček. *Bounded Arithmetic, Propositional Logic and Computational Complexity*. Cambridge University Press, 1995.

[14] J. Krajíček. On Frege and extended Frege proof systems. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 284–319. Birkhäuser, 1995.

[15] J. Krajíček, A. Skelley, and N. Thapen. NP search problems in low fragments of bounded arithmetic. *Journal of Symbolic Logic*, 72(2):649–672, 2007.

[16] P. Nguyen. *Bounded Reverse Mathematics*. PhD thesis, University of Toronto, 2008. `http://www.cs.toronto.edu/~pnguyen/`.

[17] P. Pudlák. Fragments of bounded arithmetic and the lengths of proofs. *Journal of Symbolic Logic*, 73(4):1389–1406, 2008.

[18] A. Skelley and N. Thapen. The provably total search problems of bounded arithmetic. Preprint, 2007.

[19] D. Zambella. Notes on polynomially bounded arithmetic. *Journal of Symbolic Logic*, 61(3):942–966, 1996.