NP search problems in low fragments of bounded arithmetic

Jan Krajíček[1,2] * Alan Skelley[1†] Neil Thapen[1‡]

[1]Mathematical Institute
Academy of Sciences, Prague
and
[2]Faculty of Mathematics and Physics
Charles University, Prague

September 26, 2006

Abstract

We give combinatorial and computational characterizations of the NP search problems definable in the bounded arithmetic theories T_2^2 and T_2^3.

By a **search problem** in full generality we mean simply a binary relation $R(x, y)$ such that $\forall x \exists y R(x, y)$ holds. The search task is: given an **instance** x find a **solution** y such that $R(x, y)$ holds.

The most important class of search problems are **NP search problems**. Here the relation $R(x, y)$ is decidable in polynomial time and the length of

[*] The paper was written while at the Isaac Newton Institute in Cambridge (Logic and Algorithms program), supported by EPSRC grant N09176. Also supported in part by grants A1019401, AV0Z10190503, MSM0021620839, 201/05/0124, and LC505.

[†] Partially supported by grant LC505 (Eduard Čech Center) and NSERC PDF-313650-2005.

[‡] Partially supported by grants A 101 94 01 and AV0Z10190503 (AS CR).

y is bounded by a polynomial in the length of x. Problems in NP∩coNP, the PLS problems (Polynomial Local Search) of Johnson, Papadimitriou and Yannakakis [11] and various combinatorially defined problems as in Papadimitriou [20, 22] and Papadimitriou and Yannakakis [21] are all examples of NP search problems.

Any NP search problem can be defined by an open formula $\phi(x, y)$ in the language of the bounded arithmetic theory PV of Cook [6]. The totality of the problem is then expressed by sentence

$$\forall x \, \exists y < s \, \phi(x, y)$$

where s is a polynomial time function bounding solutions in terms of instances. We say that the problem is **definable in a theory** if this sentence expressing its totality is provable in the theory (and in fact we will identify the problem with this sentence, see below).

Buss' hierarchy [2] of bounded arithmetic theories

$$\mathrm{PV} \subseteq S^1_2 \subseteq T^1_2 \subseteq T^2_2 \subseteq T^3_2 \subseteq \dots$$

and various other related theories like PV+dWPHP, U^1_2 or V^1_2 offer a natural stratification of the class of all NP search problems into subclasses consisting of those problems definable in a particular theory.

It turns out that if we can define an NP search problem in a theory of bounded arithmetic, we learn something about the complexity of solving the search problem. For example, if it is definable in PV itself or in the stronger theory S^1_2 then the problem can be solved by a polynomial time algorithm. In fact, a form of the converse implication also holds: any NP search problem solvable by a polynomial time algorithm can be defined (by a particular formula $\phi(x, y)$) such that its totality is provable in both PV and S^1_2.

Similar connections exists between T^1_2 and the PLS search problems, see Buss and Krajíček [3]. In fact, various characterizations are known of the NP search problems definable in higher levels of the bounded arithmetic hierarchy (as well as of more complex search problems, with the relation R not necessarily polynomial time, but somewhere in the polynomial hierarchy).

These are based on reflection principles for fragments of quantified propositional logic in Krajíček and Pudlák [13]; on various reflection principles for a suitably modified concept of provability in T^i_2 in Krajíček and Takeuti [17]; or go via Herbrandization of induction axioms in Ferreira [7] or a more general Herbrandization in Hanika [9, 8]. However, these characterizations for T^i_2 with $i \geq 2$ appear to lack an immediate combinatorial content and they do not really illuminate the particular class of search problems definable in the theory.[1]

The aim of this paper is to give a combinatorial characterization of the class of NP search problems definable in T^2_2 and T^3_2, previously the simplest classes without such a characterization. We are more successful for T^2_2 than for T^3_2, in that our characterization in the latter case in terms of "verifiable recursion programs" (sections 3 and 4) is perhaps more computational than combinatorial. Nevertheless we feel that these programs are simpler than previous results and provide the most promising route towards an extension of this work to find natural combinatorial principles capturing the higher levels in the hierarchy; and they are interesting in their own right, particularly with the relation between deep and shallow programs.

Our approach is to sharpen and generalize a known link between constant depth Frege systems and relativized theories $T^i_2(\alpha)$ established by Paris and Wilkie [23]. In the past this link has been used mainly for proving upper bounds on lengths of proofs or for independence results for $T^i_2(\alpha)$. Here we show that the link can be used to give characterizations in terms of reflection principles for (extensions of) resolution, which we subsequently interpret combinatorially.[2]

A note on formalization. As pointed out earlier any NP property of x can be defined in the language of PV by a formula of the form $\exists y < s\phi(x, y)$, with ϕ open. It is, however, more convenient to allow **strict** Σ^b_1-formulas (denoted $\hat{\Sigma}^b_1$): a block of bounded existential quantifiers followed by a formula built using Boolean connectives and **sharply bounded** quantifiers $\exists z < |t|$ and $\forall z < |t|$. It is well-known that, over PV, any $\hat{\Sigma}^b_1$-formula is equivalent to a

[1] Recently Pudlák [26] has outlined a game-theoretic characterization of $\Sigma^b_1(T^i_2)$.

[2] We could have used the existing construction from Krajíček [15] (drawing on [14]), interpreting it carefully. Instead, however, we give a simpler, self-contained exposition.

formula of the form $\exists y < s\phi(x,y)$ with ϕ open. In fact, if we replaced PV with S^1_2 as our base theory we could allow the even larger class of (non-strict) Σ^b_1 formulas.

Let $\hat{\Sigma}^b_1(T^2_2)$ be the set of $\forall\hat{\Sigma}^b_1$ sentences provable in T^2_2. Characterizing the NP search problems definable in T^2_2 is equivalent to characterizing this set. So it is convenient for us to identify a search problem with the sentence expressing its totality. An NP **search problem class** is then simply a true $\forall\hat{\Sigma}^b_1$ theory (by "theory" we mean nothing more than a set of sentences).

Let us recall a useful concept of reducibility among NP search problems (see also Beame et.al. [1] and Hanika [9, 8]). A problem $\forall x\, \exists y < s\, \phi(x,y)$ is (many-one) **reducible** to a problem $\forall u\, \exists v < t\, \psi(u,v)$ if there are polynomial time functions f and g such that $\forall x, v,\ f(x) < s \wedge g(x,v) < t$ and $\forall x, v\ \psi(f(x), v) \to \phi(x, g(x,v))$.

A class Γ is reducible to a class Δ, written $\Gamma \leq \Delta$, if every problem in Γ is reducible to some problem in Δ. If two classes are reducible to each other then we say they are equivalent, $\Gamma \equiv \Delta$. By an instance of a class, we mean an instance (an assignment to the universal quantifier) of one of the members of the class.

We summarize our results with a table. The entries in the table are classes of search problems and in each row all the classes are equivalent.

$\hat{\Sigma}^b_1(T^1_2)$	PLS			VR(log)$-$totality
$\hat{\Sigma}^b_1(T^2_2)$	CPLS	1$-$Ref(Res)	VR$-$totality	2VR(log)$-$totality
$\hat{\Sigma}^b_1(T^3_2)$		1$-$Ref(PK$_1$)	2VR$-$totality	

PLS and CPLS are defined in section 1, 1$-$Ref(Res) in section 2, VR$-$totality and VR(log)$-$totality in section 3 and 1$-$Ref(PK$_1$), 2VR$-$totality and 2VR(log)$-$totality in section 4. In the last section we briefly discuss the relation between these and some other classes of search problems and describe some open problems.

This is a paper in bounded arithmetic and we expect that the reader is familiar at least with the well established basic facts and definitions. A background can be found in [12]. More (including bibliographical information) on search problems in connection with bounded arithmetic can be found in [4, 9, 19].

The authors are grateful to Arnold Beckmann for his comments on an earlier version of this work.

1 PLS and coloured PLS

In this section we introduce a combinatorial principle, **coloured PLS** or **CPLS**, and prove that CPLS $\leq \hat{\Sigma}_1^b(T_2^2)$ (Lemma 3). In the next section we introduce a proof-theoretic principle $1-$Ref(Res), and prove that $1-$Ref(Res) $\leq$ CPLS (Lemma 7) and $\hat{\Sigma}_1^b(T_2^2) \leq 1-$Ref(Res) (Lemma 9).

CPLS is a generalization of the class PLS, introduced in Johnson et. al. [11]. Buss and Krajíček [3] showed that, in our notation, PLS $\equiv \hat{\Sigma}_1^b(T_2^1)$.

PLS is normally defined in terms of a cost function and a neighbourhood function, but to make the connection between PLS and CPLS clear it is convenient for us to use a different (equivalent) definition, which does not use the cost function and instead brings in a polynomial time domain.

Note that we will try to describe instances of these problems in a natural, combinatorial way, rather than stick closely to our formal definition of a search problem.

Definition 1 *A* PLS *problem is given by a polynomial time set D and a polynomial time "neighbourhood function" N which also take a size parameter a as an extra (unwritten) input. In an instance of* PLS*, $D \subseteq [0, a]$ such that $a \in D$, $N : [0, a] \to [0, a]$. A solution to the instance is a witness that the following is false:*

$$\forall x \in D \, (N(x) \in D \wedge N(x) < x).$$

The minimization axiom for polynomial time sets is enough to show that every instance of PLS has a solution, since the least member of D is always a solution. Hence this is provable in T_2^1, so PLS $\leq \hat{\Sigma}_1^b(T_2^1)$. The other direction is harder; see [3].

Definition 2 *A* CPLS *problem (for Coloured PLS) is given by unary polynomial time functions N and e, a polynomial time set L and a binary polynomial time relation C, which also take a size parameter a as a third (unwritten)* *input. We think of an instance of the principle as talking about a directed graph on a set $[0, a]$ of nodes, and all the sets and functions live in the domain $[0, a]$. Each node i is associated with a set $C_i := \{x \in [0, a] : C(i, x)\}$ of colours. L defines a set of leaf nodes, and we call a the source node. (Note that although for convenience we use one parameter a to give us both the set of nodes and the set of colours, these are treated as disjoint sets of objects and it is not significant that they are the same size.)*

A solution to the instance is a witness that one of the following is false:

1. *$\forall i \notin L\ N(i) < i$ – every non-leaf node has a strictly smaller neighbour*
2. *$\forall i \notin L\ C_{N(i)} \subseteq C_i$ – the colours of the neighbour of i are a subset of the colours of i*
3. *$\forall i \in L\ e(i) \in C_i$ – every leaf has a colour*
4. *$C_a = \emptyset$ – the source does not have any colour.*

This is equivalent to PLS if we add the condition that every leaf has the same colour (just take for D the set of nodes avoiding that colour).

In PLS we are given a directed graph where the neighbour of a node is always smaller than it, and we are given a source node, and the search problem is to find any leaf. We now give several more ways of thinking of CPLS as an extension of this picture. Firstly, we can assume that 1, 2 and 4 above are true. Our search problem is then to find either evidence that one of these is actually false, or a leaf which does not have the colour it should. Such a leaf must exist, since the source has no colours and if a node has no colours then neither does its neighbour; it is as if we are trying to find a leaf on the path that starts at the source, and we can identify nodes on this path by a Π_1^b property (having no colours), or as if we had an instance of PLS with a Π_1^b domain D.

Secondly, we can assume that 1, 2 and 3 above are true. Our search problem is now to find a colour for the source. We can think of solving it (in exponential time) by following the path down from the source to a leaf, then tracing the colour of the leaf back up this path to the source.

Lemma 3 CPLS $\leq \hat{\Sigma}_1^b(T_2^2)$.

Proof It is sufficient to prove in T^2_2 that every instance of CPLS has a solution. Suppose that there is no solution. Then we can show by Π^b_2 induction that for every node i, every node $j \leq i$ has at least one colour, which is a contradiction when $i = a$. $\square$

The next theorem also follows from the relativized version of Theorem 10, since it is known that, for example, the weak pigeonhole principle for α is a $\forall\hat{\Sigma}^b_1(\alpha)$ principle which is provable in $T^2_2(\alpha)$ by Maciel, Pitassi and Woods [18] but which is not reducible to a PLS problem (see Chiari and Krajíček [4]). We sketch a direct proof, however, as it is straightforward.

Theorem 4 *In the relativized setting,* CPLS $\not\leq$ PLS*.*

Proof (sketch) We will consider instances (L, N, C, e) of CPLS given entirely by oracles. Suppose there is a PLS problem, with domain $D \subseteq [0, t(a)]$ (where t is a term and a is the size of the instance of CPLS) and neighbourhood function M, which, given access to an instance of CPLS as an oracle always outputs a solution to the instance.

Choose a sufficiently large and let F be the following set of partially defined instances of CPLS of size a: $(L, N, C, e) \in F$ if and only if, where they are defined, L, N, C, e do not violate any part of the definition of CPLS and for all i, if $N(i)$ is defined then $N(i) \geq i - \sqrt{a}$.

Let $S =_{def} \{(x, f) : f \in F \text{ and } x \in D^f\}$. Then $S \neq \emptyset$. Let $(x, f) \in S$ be such that x is minimal. We may assume that only polylog(a) bits in f are defined, since we do not need more than this to fix $x \in D^f$.

Now, f consists of partial information about neighbours and colours. In particular no path in f has length more than, say, $a/4$, and the colours that f says appear (or fail to appear) on a path are consistent with the rules of CPLS. Also at least $3a/4$ nodes are blank, in that f records no information about them at all.

Now begin a computation of the neighbourhood function M on input x. We need to find y and $g \in F$ with $g \supseteq f$ such that $y = M^g(x)$. Then by minimality of x we will have $y \geq x$, but g will not contain enough information to fix a solution to every instance of CPLS extending g, which will give our desired contradiction.

We extend f as follows. Suppose M asks for a neighbour (or a colour) of the node i at the end of the path in f which begins at the source. Then we fix $N(i)$ to be j, where j is the first blank node below i; because f is small, we know there must be such a $j \geq i - \sqrt{a}$.

Suppose M asks for a neighbour (or a colour) of some other node i (which does not already have a neighbour). Then we declare that i is a leaf. If i is already at the end of some path p in f, then since f is small at most polylog(a) colours are already ruled out from appearing in p, so there is some colour $e(i)$ we can consistently give to i.

We reply appropriately to any other queries about colours. □

2 A reflection principle for polynomial time resolution

We first define a **polynomial time resolution proof**. Informally, it is a possibly exponentially long proof all of the structure of which is, however, given by polynomial time functions and relations.

Definition 5

1. *A clause over a propositional variables is encoded by a subset of $[2a]$, saying for each literal whether or not it is in the clause (we allow both a literal and its negation to occur simultaneously in the same clause).*

   The clause is p-time if and only if the set encoding it is p-time.

2. *A set of clauses $C_1, \dots, C_b$ over a atoms is encoded by a relation $R \subseteq [b] \times [2a]$ as follows:*

$$R_i := \{j \in [2a] \mid (i, j) \in R\}$$

   encodes C_i.

   The set is p-time iff the relation is.

3. *A p-time set of clauses $C_1, \dots, C_b$ over a atoms is* narrow *if there is a p-time function g that upon receiving an index i lists all literals in C_i.*

4. *A p-time resolution derivation consists of a set of initial clauses $C_1, \dots, C_r$, a set of internal clauses $D_1, \dots, D_s$ and a set of final clauses $E_1, \dots, E_t$, encoded respectively by polynomial time relations R, S and T, as described above.*

   It also has a p-time function $f : [s+t] \rightarrow [max((r+s)^2 \cdot 2a, r+s+t)]$ giving for each member F of the sets of internal and final clauses either an earlier pair of clauses (from the initial or internal sets) from which F was inferred by resolution, as well as the resolved literal, or a single clause from which F follows by weakening (this includes the "trivial" weakening when we just rewrite a clause that has already appeared in the proof).

5. *A p-time resolution refutation is like a p-time resolution derivation, except there is no set of final clauses; instead we insist that the last internal clause is empty. We will write a refutation as a tuple (R, S, f).*

The statement that a tuple of relations R, S, T and function f is a resolution derivation is $\hat{\Pi}_1^b(R, S, T, f)$-expressible. Similarly the statement that (R, S, f) is a resolution refutation is $\hat{\Pi}_1^b(R, S, f)$-expressible.

If we have a function α giving a truth assignment to the atoms, then the **soundness** of such a resolution refutation is the statement that one of the initial clauses contains only false literals, and this is a $\hat{\Sigma}_2^b(R, S, f, \alpha)$-formula. However if the initial clauses are narrow (as witnessed by a function g), then the soundness statement is a $\hat{\Sigma}_1^b(R, S, f, g, \alpha)$-formula. If everything in the proof is polynomial time, then the statement is $\hat{\Sigma}_1^b$.

Definition 6 *An instance of* $\mathbf{1{-}Ref(Res)}$ *is given by polynomial time objects R, S, f, g, α as above, and a parameter a giving the size of the proof, and possibly some other parameters. A solution to the instance is either a witness that with these parameters R, S, f, g fail to define a resolution refutation with narrow initial clauses, or a clause in R in which all literals are false under α. By the above observations, this defines a class of NP search problems.*

We are abusing notation here, since "1-reflection" for a proof system is normally a statement about small, coded proofs, rather than our large, polynomial time proofs.[3]

Lemma 7 1−Ref(Res) $\leq$ CPLS.

Proof Suppose we are given a p-time assignment α and $\pi = (R, S, f, g)$ which claims to be a polynomial time refutation with narrow initial clauses. Suppose π has $b+1$ lines in total (the length of R plus the length of S). Label the clauses in π as $D_0, \ldots, D_b$. Suppose there are a literals (so there are $a/2$ variables). We identify the literals with the numbers $1, \ldots, a$.

We define an instance of CPLS. The nodes are the clauses of π. The source node is the D_b, the empty clause. The leaves are the clauses from R, namely the initial clauses.

The colours of a node i are the *true* literals (under α) which appear in the clause D_i (to match our formal definition of CPLS, without loss of generality we may assume $a \leq b$, by padding out the proof π if necessary).

To find the neighbour of a non-leaf node i: D_i was derived by resolution from clauses D_{j_1} and D_{j_2}, where j_1, j_2 and the name z of the variable that was resolved on are given in polynomial time by the function f. Let us say D_{j_1} is the clause Ez and D_{j_2} is the clause $F\bar{z}$, where $E, F \subseteq D_i$. Then to find the neighbour of i, we follow the direction of the false literal. That is, if z is false then the neighbour is j_1, while if $\bar{z}$ is false then the neighbour is j_2. (And if D_i was derived by weakening rather than resolution, then its neighbour is the single clause it was derived from).

We first show that properties 1, 2 and 4 of the definition of CPLS hold – strictly speaking, we show that any witness that one of these fails to hold gives a witness that (R, S, f, g) is not a valid resolution refutation, which will be a valid solution to our instance of 1−Ref(Res). 1 is clear. 4 is true because there are no literals at all in the empty clause, so there are certainly no true literals. For 2, observe that if j is the neighbour of i, then every literal in j already appeared in i, with the exception of the resolved literal; but by our choice of j we have made sure that this literal is false. So $C_j \subseteq C_i$.

Lastly we define the function e assigning colours to leaves. For each leaf i, D_i is an initial clause and the function g lists for us in polynomial time

[3] Viewing our proofs as given by sets, relations and functions we see that it is the correct concept of reflection in the second order setting of theories like $T_2(\alpha)$, U^1_2 or V^1_2.

the (polynomially many) literals appearing in that clause. We define the function e to output the first true literal in the clause, or 0 if there is no such literal (recall that 0 is not a literal).

If 1, 2 and 4 hold, the solution to this CPLS problem must be a node i such that $e(i) \notin C_i$. But this means that $e(i)$ did not find a true literal in clause D_i. So D_i must be a false initial clause, as required. □

The last thing we need to show is that $\hat{\Sigma}^b_1(T^2_2) \leq 1{-}\mathrm{Ref(Res)}$, which will require some proof-theoretic analysis of T^2_2 which may be of independent interest. Note that in the following proof we consider the relativized case. The other results in this paper relativize, but in this case we can prove something slightly stronger (that it is only the *assignment* that needs to use the oracle, and the structure of the refutation is independent of it) so we go into a little more detail.

Theorem 8 *Suppose that $T^2_2(\rho) \vdash \forall a\, \exists y < a\, \forall z < a\, \phi(a, y, z)$, where ϕ is a sharply bounded formula and ρ is an undefined relation symbol that may appear in ϕ. Then (uniformly in a) there exists a p-time set $\mathcal{C}_a$ of clauses C_i and a p-time set $\mathcal{A}_a$ of narrow clauses (as witnessed by a p-time function that we do not show explicitly in the notation) given together by a relation R, and a p-time resolution refutation of $\mathcal{C}_a \cup \mathcal{A}_a$ given by S, f, such that:*

1. *R, S and f are strictly p-time, that is, they do not use an oracle for ρ.*

2. PV *proves that the R, S and f define a resolution refutation.*

3. *There is a p-time truth assignment α with an oracle for ρ such that* PV(ρ) *proves the following two properties:*

   (a) α satisfies all clauses in $\mathcal{A}_a$.

   (b) For any $i \in [a]$, if C_i is unsatisfied by α then $\forall z < a\, \phi(a, i, z)$ is true.

Moreover, if the formula proved in $T^2_2(\rho)$ is only $\hat{\Sigma}^b_1(\rho)$ (i.e. the quantifier $\forall z < a$ is not used) then we have the additional property that clauses in $\mathcal{C}_a$ are narrow – in fact they are singleton clauses (and there is a p-time function g witnessing this).

Proof Suppose that ϕ is a sharply bounded formula with an undefined relation symbol ρ such that $T^2_2(\rho) \vdash \forall x \, \exists y \! < \! x \, \forall z \! < \! x \, \phi(x,y,z)$. We will show, given any value a for x, how to construct a p-time resolution refutation (uniform in a) such that finding a false initial clause in the refutation yields a witness y such that $\forall z \! < \! a \, \phi(a,y,z)$, provably in $\mathrm{PV}(\rho)$.

The first step is to take a free-cut-free proof Π, in the sequent calculus for $T^2_2(\rho)$, in which every formula is $\hat{\Pi}^b_2(\rho)$ and the last line of which is the sequent

$$\forall y \! < \! x \, \exists z \! < \! x \, \neg\phi(x,y,z) \longrightarrow \emptyset.$$

Then we will show how to translate $\hat{\Pi}^b_2(\rho)$ formulas into families of clauses and we will inductively construct, for each line on the proof, a polynomial time resolution derivation of the clauses on the right from the clauses of the left. We also allow ourselves to use some extra, “helper” clauses $\mathcal{A}$ (we will generally not write the parameter a) in the derivation. We insist that the set of these is describable in polynomial time, that they are small (polynomial size), and that they are true, which means that they will be satisfied in the evaluation α (see below). The derivation for the last line of Π will give us the required resolution refutation.

Let a be the value we use for x. Some term $t(a)$ gives the largest bound on any quantifier that appears in the proof Π. This is our most important parameter, which we will call t.

We describe our propositional variables. For each sharply bounded formula θ that appears in Π, and for every possible tuple $\bar{b}$ of parameters from $[0,t]$, we have one propositional variable $\langle\theta(\bar{b})\rangle$. There are only constantly many such formulas, so we may identify the set of all variables with some set $[0,s]$ (where s is given by a term in a).

The assignment α just maps every variable $\langle\theta(\bar{b})\rangle$ to the truth value of the formula $\theta(\bar{b})$.

The $\hat{\Pi}^b_2(\rho)$ formula $\forall x \! < \! b \, \exists y \! < \! c \, \theta(b,c)$ (which may contain extra parameters) now translates naturally into the family of clauses

$$\{\bigvee_{j<c} \langle\theta(i,j)\rangle : i < b\}.$$

We will use this translation for clauses in the antecedent of a sequent - we translate the succedent in a slightly different way. Note that if the existential quantifier is not used, this translation will give singleton clauses.

Our inductive hypothesis is the following: for each sequent in the proof Π, for every choice of parameters for the free variables in the sequent, there is a p-time resolution derivation (R, S, T, f) (using these parameters) with the following initial and final clauses:

Suppose the sequent is

$$\begin{aligned}&\forall x_1 < p_1\, \exists y_1 < q_1\, \phi_1(x_1, y_1), \ldots, \forall x_k < p_k\, \exists y_k < q_k\, \phi_k(x_k, y_k)\\ &\qquad\qquad \longrightarrow \forall u_1 < s_1\, \exists v_1 < t_1\, \psi_1(u_1, v_1), \ldots, \forall u_l < s_l\, \exists v_l < t_l\, \psi_l(u_l, v_l).\end{aligned}$$

Then the initial clauses R are

$$\{ \bigvee_{y_i < q_i} \langle \phi_i(x_i, y_i) \rangle : x_i < p_i, i = 1, \ldots, k \}$$

together with a polynomial time set of helper clauses, as defined above. The final clauses S are

$$\{ \bigvee_{v_1 < t_1} \langle \psi_1(u_1, v_1) \rangle \vee \ldots \vee \bigvee_{v_l < t_l} \langle \psi_l(u_l, v_l) \rangle : u_1 < s_1, \ldots, u_l < s_l \}.$$

Note that the final clauses are a translation of

$$\forall u_1 < s_1 \ldots \forall u_l < s_l,\ \exists v_1 < t_1\, \psi_1(u_1, v_1) \vee \ldots \vee \exists v_l < t_l\, \psi_l(u_l, v_l).$$

Each sequent is introduced by a bounded quantifier rule (not including sharply bounded quantifier introduction), the cut rule, an induction rule, propositional connective introduction, or sharply bounded quantifier introduction; or it may be an axiom from $BASIC$ or an equality axiom.

The last two sets of rules and the axioms are dealt with easily and justify our slightly unusual choice of literals. Because Π was a free-cut-free proof, these rules are only applied to sharply bounded formulas, which translate as single literals (or rather, singleton clauses). Since the rules are sound and only have a small number of hypotheses, we can essentially include the instance of the rule as one of our helper axioms. We can treat any instance of an axiom similarly.

We will give one example, for the right sharply-bounded universal quantifier introduction rule. This has the form:

$$\frac{x < |t|, \Gamma \longrightarrow \Delta, \theta(x)}{\Gamma \longrightarrow \Delta, \forall y \!<\! |t|\, \theta(y)}.$$

By the inductive hypothesis, for each x we have a polynomial time resolution derivation for the hypothesis of the rule. That is, it has initial clauses $\langle x < |t|\rangle$ (a singleton clause) and every clause $\gamma \in \Gamma^*$ (where Γ^* is a set of clauses arising from the translations of Γ), together with some helper clauses. It has final clauses $\delta \vee \langle\theta(x)\rangle$ (we have one such clause for every $\delta \in \Delta^*$, where Δ^* is a set of clauses arising from translations of the formulas in Δ).

Let t' be the value of $|t| - 1$. We add the clauses $\langle 0 < |t|\rangle, \ldots, \langle t' < |t|\rangle$ (all singletons) and $\langle \forall y \!<\! |t|\, \theta(y)\rangle \vee \bigvee_{i<|t|} \neg\langle\theta(i)\rangle$ to our set of helper clauses.

We now describe a derivation for the conclusion of the rule. From Γ^* and our helper clauses $\langle i < |t|\rangle$ we can derive all clauses $\delta \vee \langle\theta(i)\rangle$, for all $\delta \in \Delta^*$ and all $i < |t|$. For each δ we resolve all such clauses with our helper clause $\langle \forall y \!<\! |t|\, \theta(y)\rangle \vee \bigvee_{i<|t|} \neg\langle\theta(i)\rangle$, to get $\delta \vee \langle \forall y \!<\! |t|\, \theta(y)\rangle$, as required.

Induction rule

Suppose the last rule applied in Π is

$$\frac{\Gamma, \forall x \!<\! p\, \exists y \!<\! q\, \theta(s, x, y) \longrightarrow \Delta, \forall x \!<\! p\, \exists y \!<\! q\, \theta(s+1, x, y)}{\Gamma, \forall x \!<\! p\, \exists y \!<\! q\, \theta(0, x, y) \longrightarrow \Delta, \forall x \!<\! p\, \exists y \!<\! q\, \theta(t, x, y)}.$$

Our strategy is to take the resolution derivations for every inductive step and put them all together to get one long derivation of step t from step 0. This derivation is polynomial time because the derivations for the inductive steps are given uniformly in s. The construction is complicated by having to deal with the side formulas. In particular, we need to re-use the clauses for Γ several times, once for each induction step, which is one of the reasons why the final derivation fails to be treelike.

By the inductive hypothesis we have, for each s, a polynomial time derivation π_s for the hypothesis of the rule, with initial clauses Γ^* and $\bigvee_y \langle\theta(s, x, y)\rangle$ for each x (with suitable bounds) together with a set $\mathcal{A}_s$ of helper clauses; and final clauses $\delta \vee \bigvee_y \langle\theta(s+1, x, y)\rangle$ for each x and each $\delta \in \Delta^*$.

We define a new series of derivations, π'_s. For each δ, let $\pi_{s,\delta}$ be a copy of π_s with the disjunction δ added to every clause. By listing all these derivations

$\pi_{s,\delta}$ one after the other we can get a derivation π'_s, with initial clauses $\delta \vee A$, $\delta \vee \gamma$ and $\delta \vee \bigvee_y \langle \theta(s,x,y) \rangle$ for each δ, each $A \in \mathcal{A}_s$, each $\gamma \in \Gamma^*$ and each x; and final clauses $\delta \vee \bigvee_y \langle \theta(s+1,x,y) \rangle$ for each x and each $\delta \in \Delta^*$.

We can now string the derivations $\pi'_0, \ldots, \pi'_{t-1}$ together one after the other, giving a derivation with initial clauses $\delta \vee A$, $\delta \vee \gamma$ and $\delta \vee \bigvee_y \langle \theta(0,x,y) \rangle$ for each δ, each $A \in \mathcal{A}_s$ (for each s), each $\gamma \in \Gamma^*$ and each x; and final clauses $\delta \vee \bigvee_y \langle \theta(t,x,y) \rangle$ for each x and each $\delta \in \Delta^*$.

We add weakening steps to the beginning of this derivation to obtain each $\delta \vee A$ from just A, each $\delta \vee \bigvee_y \langle \theta(0,x,y) \rangle$ from just $\bigvee_y \langle \theta(0,x,y) \rangle$ and each $\delta \vee \gamma$ from just γ. This gives the required derivation; notice that our helper clauses are now $\bigcup_s \mathcal{A}_s$.

Cut rule

This is done in the same way as one step in the induction rule.

Bounded $\exists$ left introduction

Suppose the last rule applied in Π is

$$\frac{x < s, \theta(x), \Gamma \longrightarrow \Delta}{\exists y \!<\! s\, \theta(y), \Gamma \longrightarrow \Delta}$$

where the variable x does not occur in the conclusion; we may also assume that the formula θ is sharply bounded, since Π is free-cut-free.

By the inductive hypothesis, for each $x < s$ we have a p-time derivation π_x with initial clauses $\langle \theta(x) \rangle$, $\gamma \in \Gamma^*$ and a set $\mathcal{A}_x$ of helper clauses (we may assume that $\mathcal{A}_x$ contains $\langle x < s \rangle$, since it is a true sentence) and final clauses $\delta \in \Delta^*$.

By removing $\langle \theta(x) \rangle$ from the initial clauses and adding the negated literal $\neg \langle \theta(x) \rangle$ to each remaining clause in π_x we get a derivation π'_x with initial clauses $\neg \langle \theta(x) \rangle \vee \gamma$ and $\neg \langle \theta(x) \rangle \vee A$ for each $A \in \mathcal{A}_x$, and final clauses $\neg \langle \theta(x) \rangle \vee \delta$.

Make a new derivation by first stringing π'_0 to π'_{s-1} together. Then add every γ and every $A \in \mathcal{A}_x$ as initial clauses, together with the clause $\bigvee_{y<s} \langle \theta(y) \rangle$. With these initial clauses, we can now obtain the initial clauses of each π'_x by weakening. Finally for each $\delta \in \Delta^*$ resolve every $\neg \langle \theta(x) \rangle \vee \delta$ with $\bigvee_{y<s} \langle \theta(y) \rangle$ to obtain δ as a final clause.

Bounded $\forall$ left introduction
Suppose the last rule applied in Π is

$$\frac{\theta(r), \Gamma \longrightarrow \Delta}{r < s, \forall x{<}s\, \theta(x), \Gamma \longrightarrow \Delta}.$$

The derivation for the bottom depends on the values of r and s. If $r \geq s$ then we add $\neg\langle r < s\rangle$ as a helper clause, derive the empty clause from $\langle r < s\rangle$ and derive every δ by weakening. If $r < s$, then the initial clauses of the conclusion of the rule already contain the initial clauses of the hypothesis as a subset.

Bounded $\exists$ right introduction
This is the rule

$$\frac{\Gamma \longrightarrow \Delta, \theta(r)}{r < s, \Gamma \longrightarrow \Delta, \exists x{<}s\, \theta(x)}.$$

It is dealt with similarly to the bounded $\forall$ left introduction. If $r < s$ is true, we use weakening to obtain the bottom derivation from the top one.

Bounded $\forall$ right introduction
This is the rule

$$\frac{x < s, \Gamma \longrightarrow \Delta, \theta(x)}{\Gamma \longrightarrow \Delta, \forall y{<}s\, \theta(y)}.$$

For each $x < s$ we add $\langle x < s\rangle$ as a helper clause. This allows us to derive each $\delta \vee \langle\theta(x)\rangle$ from Γ^*; stringing all these together gives a derivation for the bottom.

For the convenience of the reader, we summarize the helper clauses that we introduced in the proof. We used translations of instances of the propositional connective and sharply bounded quantifier introduction rules; translations of instances of the *BASIC* and equality axioms; and some true clauses of the form $\langle i < j\rangle$, introduced to help with our treatment of bounded quantifiers. □

Lemma 9 $\hat{\Sigma}_1^b(T_2^2) \leq 1{-}\mathrm{Ref(Res)}$ *(provably in* PV*).*

Proof Suppose $T^2_2 \vdash \forall x\, \exists y < t\, \phi(x,y)$. By altering ϕ if necessary, we may assume that the bound t on y is just the variable x. By Theorem 8, there are polynomial time objects $R, S, f, g, \alpha, \mathcal{A}, \mathcal{C}$ such that, if we plug in a parameter a as a value for x, $\mathcal{C}$ is a set of narrow initial clauses $C_0, \ldots, C_{a-1}$ such that if any C_i is unsatisfied by α then $\phi(a,i)$ is true, $\mathcal{A}$ is a set of narrow clauses all satisfied by α, the narrowness of $\mathcal{C}$ and $\mathcal{A}$ is witnessed by g, (R,S,f) describes a resolution refutation of $\mathcal{A} \cup \mathcal{C}$, and all of this is provable in PV.

This is an instance of 1$-$Ref(Res) and the only possible solution is some i such that C_i is false under α. But such an i is a solution to our instance of $\hat{\Sigma}^b_1(T^2_2)$. □

We summarize these results as a theorem.

Theorem 10 *The set of* $\forall\hat{\Sigma}^{\,b}_{\,1}$ *consequences of* T^2_2 *is many-one equivalent to the classes* CPLS *and* 1$-$Ref(Res)*.*

Furthermore these reductions are provable in PV (we showed this explicitly in Lemma 9, but it is clear in the other two reductions), which gives as a corollary (recall that formally we identify a class of search problems with a set of sentences):

Theorem 11 *As theories of bounded arithmetic,* $\hat{\Sigma}^b_1(T^2_2) = \mathrm{PV} + \mathrm{CPLS} = \mathrm{PV} + 1{-}\mathrm{Ref(Res)}$*.*

These theorems relativize in the natural way.

Theorem 8 also gives us a new characterization of $\hat{\Sigma}^b_2(T^2_2)$, the set of $\forall\hat{\Sigma}^b_2$ consequences of T^2_2. This appears to be simpler than the characterization in Chiari and Krajíček [4] in terms of *Generalized Local Search* (GLS) problems.

Theorem 12 *Let* $\mathbf{2-Ref(Res)}$ *be the principle that for every polynomial time resolution refutation (*not *necessarily with narrow initial clauses) and every polynomial time truth assignment, one of the initial clauses contains only false literals.*

Alter the definition of CPLS *by getting rid of the function* e *and replacing condition 3, "*$\forall i \in L\ e(i) \in C_i$*," with "*$\forall i \in L\ \exists x\ x \in C_i$*". Let* $\Pi_1 - \mathrm{CPLS}$

be the principle that every instance of this new definition of CPLS *has a solution.*

Then $2-\mathrm{Ref(Res)}$*and* $\Pi_1 - \mathrm{CPLS}$ *are* $\forall\hat{\Sigma}^b_2$ *principles which axiomatize* $\hat{\Sigma}^b_2(T^2_2)$ *over* PV*.*

3 Verifiable Recursion Programs

A verifiable recursion program is roughly a sequence $R_a, \ldots, R_0$ of machines. In the course of its computation, each machine may invoke (provide input to and receive output from) other machines further down in the sequence. Each machine has an associated correctness predicate for checking its output, and the goal of the whole program is to produce a correct output for R_a.

Definition 13 *A* **verifiable recursion program***, or* **VR program***,* P *consists of polynomial time machines* R *(with recursive query ability) and* V*, both taking a size parameter* a*. An* **instance** P_a *of a verifiable recursion program* P *is specified by giving a value for the parameter* a *and comprises:*

1. *A sequence* $R_a, ..., R_0$ *of polynomial time machines with inputs from* $[0, a]$ *with recursive query ability. Formally,* R_i *is defined as follows: a computation of* R_i *on input* x *is precisely a computation of* R *on input* (a, i, x)*.*

   Each machine R_i *is only allowed to make calls to machines* R_j *with* $j < i$ *(so* R_0 *cannot make any calls).*

2. *A sequence* $V_a, ..., V_0$ *of polynomial time predicates on* $[0, a]^2$*. As above,* $V_i(x, y)$ *abbreviates* $V(a, i, x, y)$*. If* $V_i(x, y)$ *is true, we say that* y *is a* **correct reply** *for* x*, at level* i*.*

A program P_a *is* **well-defined** *if for all* i *and* x*, if the recursive calls made by machine* R_i *on input* $x \in [0, a]$ *are answered by (any) correct replies, then the output* y *of* $R_i(x)$ *satisfies* $V_i(x, y)$ *(and is thus correct).*

A program P_a is total if for every $x \in [0, a]$ there is some $y \in [0, a]$ such that $V_a(x, y)$. We can think of this as "every input to R_a has a correct output", but notice that R_a does not appear in the formal definition.

For a given VR program P, the principle that if any instance of P is well-defined then it is total is expressible as a $\forall\hat{\Sigma}_1^b$ sentence: for every a and $x \in [0,a]$, either

1. there exists $y \in [0,a]$, $k \in [0,a]$ and a computation of R_k on y with correct answers to queries but which returns incorrect output, or
2. there exists a correct output for R_a on input x.

This is a search problem, and we call the class of all such problems **VR−totality**. Note that in our formalization, VR−totality is also a first-order theory.

Lemma 14 *T_2^2 proves* VR−totality*. Hence* VR−totality $\leq \hat{\Sigma}_1^b(T_2^2)$*.*

Proof Fix a program (R,V) and a parameter a.

Firstly, even PV proves that for any $x \in [0,a]$ there is a correct reply to R_0 on input x. Now for a fixed k assume that there is a correct reply for all $x \in [0,a]$ at every level $j \leq k$. Let $z \in [0,a]$. By induction on the length of the computation of R_{k+1} on z, there exists a partial computation with each query correctly answered. Therefore by well-definedness of R and V, there is a correct output for R_{k+1}.

Now, by applying Π_2^b-IND on k, there is a correct output to any input x at level a. □

Lemma 15 CPLS $\leq$ VR−totality*. Hence* $\hat{\Sigma}_1^b(T_2^2) \equiv$ VR−totality*.*

Proof Consider a CPLS problem given by N, e, L and C and fix also the size parameter a.

Fix a suitable term $b > a$, large enough to code any solution to the CPLS problem. Our program only needs depth a, so we will describe R_i and V_i for $i \leq a$ and stipulate that for $i > a$ they are the same as R_a and V_a.

We define the correct output at level $i \leq a$ as: $V_i(x,y)$ is true if y is a colour of node i, or if y already codes a solution to the CPLS instance. Note that neither here, nor in the definition of the machines R_i, do we use the input x.

Machine R_0 returns $e(0)$ if 0 is a leaf and $e(0) \in C_0$. If one of these is not true, then this constitutes a solution to the CPLS problem and R_0 returns a witness to this instead.

For $i \leq a$, machine R_i computes the neighbour $j = N(i)$ of node i and queries the machine R_j. If the reply is a witness to the CPLS problem, then R_i returns that. Otherwise the reply should be a colour in C_j, which should also be in C_i. If so, R_i returns this colour. If however one of these conditions fails, then R_i has found a witness to the CPLS problem, which it returns. □

Notice that this proof is more complicated than it needs to be for the lemma. We have built an "exception handling" system into the program, so that, whenever it encounters a witness to the CPLS problem, this is immediately passed up to the top machine. Hence the program is provably well-defined in PV, and the only witnesses to this VR−totality problem are correct outputs of the top machine.

We may guarantee this in general: given any verifiable recursion program P there is a program P', provably well-defined in PV, such that for any a and x the correct outputs at the top level of P'_a are the union of the correct outputs at the top level of P_a with the set of witnesses (if there are any) that P_a is not well-defined. So if P_a is well-defined, the outputs of P_a and P'_a are the same. Hence we get

Corollary 16 *Let* **VR** *be the class of multifunctions F such that there is a term t and a verifiable recursion program P, provably well-defined in* PV*, such that for all x, the correct outputs of $P_{t(x)}$ on input x are precisely the values of $F(x)$. Then the $\hat{\Sigma}^b_1$ consequences of T^2_2 are witnessed precisely by multifunctions from* VR*, provably in* PV*.*

That is, if $T^2_2 \vdash \forall x\, \exists y\, \theta(x, y)$, where θ is an (implicitly bounded) PV *formula, then for some $F \in$* VR*,* PV $\vdash \forall x\, \forall y\, , \theta(x, y) \leftrightarrow F(x) = y$*; and if $F \in$* VR *then there is an (implicitly bounded)* PV *formula θ such that $T^2_2 \vdash \forall x\, \exists y\, \theta(x, y)$ and* PV $\vdash \forall x\, \forall y\, , \theta(x, y) \leftrightarrow F(x) = y$*.*

Definition 17 *A* **shallow** *verifiable recursion program is defined as in Definition 13, except that the allowed depth of recursion is only $|a|$ rather than a. That is, the inputs and outputs still come from the set $[0, a]$, but we only have machines $R_{|a|}, \ldots, R_0$ and $V_{|a|}, \ldots, V_0$.*

Let **VR(log)−totality** *be the principle "every well-defined shallow verifiable recursion program is total". This is also a class of search problems.*

Lemma 18 VR(log)−totality $\leq \hat{\Sigma}_1^b(T_2^1)$.

Proof We show $S_2^2 \vdash$ VR(log)−totality by the same argument as in Lemma 14, but using LIND rather than IND. Then the result follows by $\forall\Sigma_1^b$ conservativity of S_2^2 over T_2^2. □

We remark that there is a direct reduction of VR(log)−totality to PLS, informally as follows: let $n = |a|$, where a is the parameter giving the size of our instance of VR−totality. For a given $x \in [0, a]$, the domain of the PLS problem is the set of sequences $(c_n, \ldots, c_0, w_n, \ldots, w_0)$ where $w_n, \ldots, w_0$ is a partial computation of the entire "computation stack" of R and each c_i measures the progress made in w_i (the number of recursive queries made and correctly answered so far). These progress indicators can be verified in polynomial time by checking that the answers to recursive queries are correct, and thus the domain is a polynomial time set. The neighbourhood function examines this stack of computations, advances the computation of some machine that is not waiting for a query to be answered (the bottom one, in the worst case), and passes completed outputs back up the stack. In the resulting computation sequence, the progress $(c_n, \ldots, c_0)$ will be lexicographically larger unless the topmost machine had already finished its computation. (We would need to reverse this ordering to match our version of PLS, which searches for a minimal cost.)

Lemma 19 PLS $\leq$ VR(log)−totality. *Hence* $\hat{\Sigma}_1^b(T_2^1) \equiv$ VR(log)−totality.

Proof Given the polynomial time domain D, the neighbourhood function N and a size parameter a, define R_i and V_i as follows:

$V_k(x, y)$ returns true if $x \notin D$. Otherwise it accepts if $y \in D$ and either $N(y) = y$ or $x - y \geq 2^k$.

$R_0(x)$ computes $N(x)$. For $k > 0$, $R_k(x)$ queries $R_{k-1}(x)$, obtaining x' and then queries $R_{k-1}(x')$, returning the result.

The condition on the R and V machines – namely that correct answers to queries implies correct output – is direct from the definitions of the machines.

Now, PV proves that a correct (satisfactory to V_n) reply y to $R_{|a|}(a)$ satisfies $y \in D$ and $N(y) = y$ (since it is not possible that $x - y \geq 2^{|a|}$). □

Note that the ability to make multiple recursive calls at each level seems to be essential.

As before since all these reductions are provable in PV, we get as corollaries:

Corollary 20 *Let* **VR(log)** *be the class of multifunctions F such that there is a term t and a shallow verifiable recursion program P, provably well-defined in* PV*, such that for all x, the correct outputs of $P_{t(x)}$ on input x are precisely the values of $F(x)$. Then the Σ_1^b consequences of T_2^1 are witnessed precisely by multifunctions from* VR(log)*, provably in* PV*.*

Theorem 21 *As first order theories,* $\mathrm{PV} + \hat{\Sigma}_1^b(T_2^2) = \mathrm{PV} + \mathrm{VR}-\mathrm{totality}$ *and* $\mathrm{PV} + \hat{\Sigma}_1^b(T_2^1) = \mathrm{PV} + \mathrm{VR(log)}-\mathrm{totality}$.

4 Search principles for T_2^3

In this section we expand on the results of the previous sections to obtain a characterization of the $\hat{\Sigma}_1^b$ consequences of T_2^3 through a generalization of the verifiable recursion machines.

We obtain this by changing the verification property: namely, the verification machines V_i will have an additional "check" argument z. We say that z witnesses that a reply y to input x at level i is incorrect if $\neg V_i(x, y, z)$. In a certain informal sense, y is "correct" if $\forall z < a\, V_i(x, y, z)$, and as with the verifiable recursion programs, if a machine's queries are replied to correctly, then its output is correct. In fact, however, we do not use this universal quantifier. Instead we add Herbrand functions F_i such that from any x, y, z falsifying V_i and the queries and responses used to compute y, F_i identifies an incorrect reply and provides a z' as a witness. Formally:

Definition 22 *A* **2-verifiable recursion program** *P consists of polynomial time machines R (with recursive query ability), V and F, all taking a size parameter a. An* **instance** *P_a of P is as follows.*

1. *The sequence $R_a, ..., R_0$ is as before.*
2. *The sequence $V_a, ..., V_0$ are now predicates on $[0,a]^3$.*
3. *There is a sequence $F_a, ..., F_1$ of polynomial time Herbrand functions $[0,a]^{poly(|a|)} \to [0,a]^2$, where as usual, $F_i(x, z, ...)$ abbreviates $F(a, i, x, z, ...)$.*

A program P_a is well-defined if for all i, x and z, if the recursive calls $R_{j_1}(x_1), ..., R_{j_m}(x_m)$ made in a computation of R_i on input $x \in [0,a]$ are answered by $y_1, ..., y_m$, R_i outputs y and $F_i(x, z, \bar{y})$ outputs $\langle k, z' \rangle$, then either $V_i(x, y, z)$ is true, or $V_{j_k}(x_k, y_k, z')$ is false.

In particular, the output of R_0 must be correct for any z, as there are no recursive queries made.

Note that the property of well-definedness is $\hat{\Pi}^b_1$.

A program P_a is total if for every $x \in [0,a]$, there is some $y \in [0,a]$ such that $V_a(x, y, 0)$; that is, at the top level, we do not care about the argument z and there is no implicit quantification.

Now observe that, similarly to the previous section, the following statement **2VR–totality** is expressible as a $\forall\hat{\Sigma}^b_1$-sentence: if a 2-verifiable recursion program P is well-defined then it is total.

Lemma 23 $T^3_2 \vdash$ 2VR$-$totality*, and therefore* 2VR$-$totality $\leq \hat{\Sigma}^b_1(T^3_2)$.

Proof The proof in T^3_2 is by induction on i on the hypothesis:

$$\forall j < i \, \forall x \leqslant a \, \exists y \leqslant a \, \forall z \leqslant a \, V_i(x, y, z). \qquad \Box$$

We note a useful assumption we can make about these 2-verifiable recursion machines. If we are describing a machine R_i, we can assume that when R_i makes a query x to R_j and receives an answer y, then this answer is correct for any particular values of z that R_i can compute (i.e., $V_j(x, y, z)$ is satisfied for these z). This is because the Herbrand function F_i can retroactively check these assumptions, and provide an “excuse” for R_i to compute incorrectly if an assumption was not met.

The converse of Lemma 23 is also true:

Theorem 24 *$\hat{\Sigma}^b_1(T^3_2) \leq$ 2VR$-$totality.*

This is the main result of this section. The broad outline of the proof is as follows: we first define a propositional proof system called PK_1 that is a strengthening of resolution[4] and the associated reflection principle $1-\mathrm{Ref}(\mathrm{PK}_1)$ for polynomial time proofs in this system. We then show how to reduce the principle to 2VR$-$totality. Finally, we show how to adapt the theorem of section 2 concerning T^2_2 and resolution to reduce $\hat{\Sigma}^b_1(T^3_2)$ to this reflection principle.

Definition 25 PK_1 *is a proof system like resolution, but with the following additions:*

1. *Clauses in PK_1 (**1-clauses**) are sets containing both literals and conjunctions of literals.*
2. *The resolution rule is expanded to the **1-resolution** rule:*

   $$\frac{C \vee (\bigwedge_k l_k \wedge l') \qquad D \vee \neg l'}{C \vee D}$$

   which is ordinary resolution if the conjunction resolved upon is simply l'.
3. *There is a new rule, $\wedge$**-introduction:***

   $$\frac{C \vee \bigwedge_j l_j \qquad C \vee l'}{C \vee (\bigwedge_j l_j \wedge l')}$$

We now define a presentation of PK_1 derivations, which we call p-time if the relevant relations and functions are p-time. This is meant to be a generalization of Definition 5 for resolution proofs.

Definition 26

1. *A conjunction over a propositional variables is encoded as a subset of $[2a]$. A set of conjunctions is encoded by a relation $Q \subseteq [c] \times [2a]$.*

[4] PK_1 is a special case of a proof systems $R(f)$ defined in [15]; it is $R(id)$.

A 1-clause over a propositional variables and c conjunctions is encoded as a subset of $[2a+c]$ *describing which literals and which conjunctions appear in the 1-clause, together with a relation* Q *(as above) describing which literals appear in the conjunctions. In general* Q *will also describe many conjunctions which do not appear in the 1-clause.*

2. *A set of 1-clauses* $C_1, \ldots, C_b$ *over a atoms and c conjunctions is encoded by a relation* Q *(giving the conjunctions) and*

   $$R \subseteq [b] \times [2a+c]$$

   (giving the 1-clauses).

3. *A p-time set of 1-clauses* **has narrow conjunctions** *if there is a polynomial-time function which, given the name of any conjunction in* Q*, outputs all the literals appearing in it.*

   The set of 1-clauses **has narrow clauses** *if there is a polynomial-time function which, given a 1-clause in the set, lists the literals and the (names of) conjunctions that appear in it.*

   Finally the set is **fully narrow** *if it both has narrow conjunctions and has narrow clauses.*

4. *A derivation consists of sets* $C_1, \ldots, C_r$*,* $D_1, \ldots, D_s$ *and* $E_1, \ldots, E_t$ *of respectively initial, internal and final 1-clauses, which all share a common set of conjunctions encoded by* Q *and are themselves encoded by relations* R, S *and* T*. A function* f *identifies the hypotheses used to derive a given clause, and identifies either the conjunction and/or literal resolved upon, or the conjunctions and literal involved in a* $\wedge$*-introduction inference, as appropriate. (For the formalization to work we need to include one extra rule: that we can replace a literal in a clause with a conjunction containing only that literal.)*

5. *As before a derivation is a refutation with no final clauses and with the last internal clause empty.*

The statements that (Q, R, S, T, f) is a derivation and that (Q, R, S, f) is a refutation are $\hat{\Pi}^b_1(Q, R, S, T, f)$ expressible. For a p-time assignment α, the soundness of a refutation with a fully narrow set of initial clauses (witnessed by p-time functions g_1 and g_2) is $\hat{\Sigma}^b_1$.

Hence we can define $1-\mathrm{Ref}(\mathrm{PK}_1)$ as an NP search problem: for (Q, R, S, f, g, α) polynomial time, given a size parameter a specifying the instance, find either a witness that the PK_1 refutation with fully narrow initial clauses is improperly defined or an initial clause falsified by α.

Theorem 27 *$1-\mathrm{Ref}(\mathrm{PK}_1) \leq 2\mathrm{VR}-\mathrm{totality}$.*

Proof Consider a polynomial time refutation $\pi = (Q, R, S, f, g)$ and an assignment α. Fix the size parameter a and let $D_0, \ldots, D_b$ be the 1-clauses in π and let $A_1, \ldots, A_c$ be the conjunctions used in π.

Choose a size $d > b$ for our 2VR program large enough to code witnesses to the $1-\mathrm{Ref}(\mathrm{PK}_1)$-instance. As we will only require depth b, for $i > b$ all machines will behave as machine b. The inputs x to all machines are ignored, and we will not write them in what follows. All witnesses to the 2VR program being ill-defined will give rise to a witness that π is not well-formed.

The correctness predicate $V_i(y, z)$ (remember we are ignoring x) accepts if either y names a true literal in 1-clause D_i, or if it names a conjunction in 1-clause D_i and z does not name a false literal in the conjunction. A correct reply at the top level then clearly witnesses that the final clause of π is not empty.

By the remark before Lemma 24, we may assume that any reply y to a recursive call to level i is either a true literal in D_i or a conjunction in D_i.

Informally, machine R_i seeks a true literal or a true conjunction in D_i. If D_i is an initial clause (and therefore fully narrow), R_i examines all conjunctions and literals occurring and returns a true one. If D_i was derived by weakening from D_j, R_i calls R_j and returns the result.

Suppose D_i was derived by an instance of $\wedge$-introduction, of the form

$$\frac{C \vee \bigwedge_j l_j \qquad C \vee l'}{C \vee (\bigwedge_j l_j \wedge l')}.$$

Say that the bottom 1-clause is D_i, and the left- and right-hand top clauses are D_j and D_k.

Case (1): l' is false. Then R_i queries R_k, and gets a reply y which it returns. We may assume that y is either a true literal in D_k or a conjunction in D_k. If $\neg V_i(y,z)$, that is, z witnesses that R_i's reply is incorrect, then either y is a false literal in D_i (which is impossible) or z is a false literal in the conjunction A named by y, and A appears in D_i. But then A must be in C and so also appears in D_k. So $\neg V_k(y,z)$. Hence in this case, we put $F_i(y,z) := \langle k, z\rangle$.

Case (2) l' is true. Then R_i queries R_j, and gets a reply y, and we may assume y is either a true literal in D_j or a conjunction in D_j. If y is a literal, then R_i returns y. As above, y is a true literal appearing in D_i, so no z can witness that y is an incorrect reply for level i.

So suppose y is a conjunction A. If A appears in C, then R_i returns y. Otherwise A is the conjunction $\bigwedge_j l_j$, in which case R_i returns the name y' of the conjunction $\bigwedge_j l_j \wedge l'$ in D_i (this name is given to us in polynomial time by the function f). In this case $F_i(y,z) := \langle j, z\rangle$, which has the desired property that if $\neg V_i(y',z)$ (i.e. z is a false literal in $\bigwedge_j l_j \wedge l'$) then $\neg V_j(y,z)$ (ie. z is a false literal in $\bigwedge_j l_j$ – for z cannot be the literal l', since l' is known to be true, in this case).

The case of 1-resolution is similar. □

Observe that the 2VR program produced by the reduction makes only one recursive call at each level. This is analogous to the situation for (full depth) VR programs and in contrast to the shallow programs of both types.

All that remains to finish the proof of Theorem 24 is to show that $\hat{\Sigma}^b_1(T^3_2) \leq 1-\mathrm{Ref}(\mathrm{PK}_1)$. This follows from the following theorem, which is a generalization of Theorem 8 from section 2.

Theorem 28 *Suppose that $T^3_2(\rho) \vdash \forall a\, \exists x < a \forall y < a\, \exists z < a\, \phi(a,x,y,z)$, where ϕ is sharply bounded and the undefined relation symbol ρ may appear in ϕ. Then (uniformly in a) there exists a p-time set $\mathcal{C}_a$ of 1-clauses C_i and a p-time fully narrow set $\mathcal{A}_a$ of clauses given together by a relation R, and a p-time PK_1 refutation of $\mathcal{C}_a \cup \mathcal{A}_a$ given by S, f (all over a p-time set of conjunctions given by relation Q) such that:*

1. *Q, R, S and f are strictly p-time (i.e., do not depend on ρ).*

2. PV proves that Q, R, S and f define a PK_1 refutation.

3. There is a p-time truth assignment α to the literals in the refutation, with an oracle for ρ, such that PV(ρ) proves:

   (a) α satisfies all clauses in $\mathcal{A}_a$.

   (b) For any $i \in [a]$, if C_i is unsatisfied by α then $\forall y < a\, \exists z < a\, \phi(a, i, y, z)$ is true.

Moreover, if the formula proved is $\hat{\Sigma}^b_2(\rho)$ or $\hat{\Sigma}^b_1(\rho)$ (i.e., the quantifier $\exists z < a$ and possibly also $\forall y < a$ is not used), then the set of clauses $\mathcal{C}_a$ will have narrow conjunctions or be fully narrow, respectively.

Proof The proof follows that of Theorem 8, with several modifications that we indicate. As before fix a value for parameter a and this time take a **treelike** free-cut-free proof with end-sequent

$$\forall x < a \exists y < a \forall z < a \neg\phi(a, x, y, z) \longrightarrow \emptyset.$$

For every proper $\hat{\Pi}^b_3(\rho)$ formula $\forall x < p \exists y < q \forall z < r \theta(x, y, z)$ in Π and its ancestors that are not sharply bounded, replace $\forall z < r\theta(x, y, z)$ by $\beta_\theta(r, x, y)$, where β_θ is a new relation symbol (and possibly some extra parameters $\bar{w}$ may appear in all these formulas). The resulting Π' fails to be a proof only at the sites of former $\forall$-introduction inferences, where the new β relations now spontaneously appear.

The assumption about being treelike is needed so that, for example, a formula $\forall z < r\, \theta(e, f, z)$ will not be an ancestor of both $\forall x < p\, \exists y < q\, \forall z < r\, \theta(x, y, z)$ and $\forall u < p\, \exists v < q\, \forall z < r\, \theta(u, v, z)$, because these would get different β symbols unless we were very careful about the names of our variables.

The set of conjunctions that will appear in the PK$_1$ derivation we construct comprises $\bigwedge_{i<r} \langle \theta(i, \bar{b}) \rangle$ for every bounded quantifier $\forall i < x\, \theta(i, \bar{y})$ that we replace with a β_θ in the above fashion, and for every choice $r, \bar{b}$ of parameters from $[0, t]$ (recall that t is the value of the term $t(a)$ giving the largest bound on any quantifier in Π). This set of conjunctions is clearly given by a p-time relation Q.

The theorem is now proved by induction as before (constructing a derivation of clauses translating the succedent from clauses translating the antecedent), but we modify the induction hypothesis as follows: Whenever a literal $\langle \beta_\theta(r, x, y)\rangle$ would occur in the translation into clauses, instead the translation produces the conjunction $\bigwedge_{i<r}\langle\theta(i, x, y)\rangle$.

Introduction of a propositional connective or a sharply bounded quantifier, as well as *BASIC* and equality axioms, are as before: the helper clauses are the same and only ordinary resolution is used.

The cases of induction and cut are likewise unaffected: both of these rules entail stringing together several PK_1 derivations (in the case of induction, modified by adding extra disjuncts to all 1-clauses and some weakening steps at the beginning). No resolutions steps of any kind are added.

What remain are the (non-sharply bounded) quantifier introduction rules.

Bounded $\exists$ left and right introduction

If the formula $\exists y < s\, \theta(y)$ introduced is $\hat{\Sigma}^b_1(\rho)$ (and therefore involves no relations β_θ) then the rule is handled as before, and involves only weakening and ordinary resolution. If $\exists y < s\, \beta_\theta(r, y)$ is introduced on the right, then again the rule is as before and involves only weakening (this time adding conjunctions rather than literals to the final 1-clauses).

Finally, if $\exists y < s\, \beta_\theta(r, y)$ is introduced on the left we take a proof π_y (corresponding to the hypothesis of the rule) for each $y < s$, removing the initial clauses $\bigwedge_{i<r}\langle\theta(i, y)\rangle$ and adding the disjunction $\bigvee_{i<r}\neg\langle\theta(i, y)\rangle$ to each remaining 1-clause. We string together the resulting proofs $\pi'_0, ..., \pi'_{s-1}$ and add every $A \in \mathcal{A}_y$ and $\gamma \in \Gamma^*$ as initial clauses, as well as $\bigvee_{y<s}(\bigwedge_{i<r}\langle\theta(i, y)\rangle)$. As before each initial clause of every π'_y is now obtained by weakening. For each $\delta \in \Delta^*$, "cut" every $\bigvee_{i<r}\neg\langle\theta(i, y)\rangle\vee\delta$ with $\bigvee_{y<s}(\bigwedge_{i<r}\langle\theta(i, y)\rangle)$ to obtain δ as a final clause. (Note that we can simulate a cut

$$\frac{U \vee \bigvee_{i<r}\neg\langle\theta(i, y)\rangle \qquad V \vee \bigwedge_{i<r}\langle\theta(i, y)\rangle}{U \vee V}$$

by r applications of 1-resolution).

Bounded $\forall$ left and right introduction

If the $\forall$ introduction inference is correct in Π' (i.e., does not involve the spontaneous production of a β_θ relation), then the rule is handled as before and involves only weakening and stringing together proofs.

If the "inference" is

$$\frac{x < r, \Gamma \longrightarrow \Delta, \theta(x)}{\Gamma \longrightarrow \Delta, \beta_\theta(r)},$$

add $\langle x < r \rangle$ as helper clauses for every $x < r$. We obtain derivations of every $\delta \vee \langle \theta(x) \rangle$ from Γ^*. These are strung together and $\wedge$-introduction is applied to obtain the final 1-clauses $\delta \vee \bigwedge_{i<r} \langle \theta(i) \rangle$.

Finally if the "inference" is

$$\frac{\theta(p), \Gamma \longrightarrow \Delta}{p < r, \beta_\theta(r), \Gamma \longrightarrow \Delta},$$

and $p \geq r$ then every δ follows as in Theorem 8 from the helper clause $\neg\langle p < r \rangle$ and weakening. If $p < r$ then add the helper clause $\langle \theta(p) \rangle \vee \neg \langle \theta(p) \rangle$. The initial clauses of the hypothesis of the rule are all either initial clauses of the conclusion, or are derived from the new initial clause $\bigwedge_{i<r} \langle \theta(i) \rangle$ and the new helper clause.

Note that the helper clause introduced in the last case above is the only additional kind of helper clause beyond those produced by Theorem 8. In particular, no conjunctions appear in any helper clause. □

We could also define an extension C$_2$PLS of PLS to capture $\hat{\Sigma}^b_1(T^3_2)$. Rather than each node having a set C_i of colours, it has a 0/1 matrix D_i whose rows correspond to conjunctions in the PK_1 proof, so that a row is all 1 if the conjunction is true in that 1-clause. Consider the following contradiction: the root contains no all-1 rows, each leaf contains an all-1 row, and if j is the neighbour of i, then j contains an all-1 row only if i does. By carefully introducing p-time functions which witness all the existential quantifiers here, we can say this in a $\hat{\Pi}^b_1$ way and then take C$_2$PLS to be the negation of this. But this principle does not seem very illuminating and we will give no more details of it.

Definition 29 *A* **shallow 2-verifiable recursion program** *is defined as above, except that there are only $|a|+1$ many machines $P_{|a|}, \ldots, P_0$ (and similarly $|a|+1$ many correctness predicates and Herbrand functions) so the recursion only has logarithmic depth.*

Theorem 30 *(*PV*) The principle* $2\mathrm{VR}(\log)-\mathrm{totality}$ *that every well-defined shallow 2-verifiable recursion machine is total is provable in S^3_2 and hence T^2_2. Furthermore the search principle* CPLS *is reducible to it. So shallow 2-verifiable recursion machines witness exactly the $s\Sigma^b_1$ consequences of T^2_2.*

Proof The first part of the theorem is as for $2\mathrm{VR}-$totality, but with Σ^b_3-LIND.

We now prove the reducibility part of the theorem. The proof uses the "shortening cuts" idea of the proof of the conservativity of S^3_2 over T^2_2, and is a natural extension of the proof of Lemma 19[5].

Informally the idea is that machine P_i should output either the 2^ith iterated neighbour of x, or a colour of x.

Suppose the CPLS problem has nodes $[0, a]$. We define a particular 2VR(log) machine, with size parameter a^2 (hence of "depth" $2|a|$). Any witness to the CPLS instance is a correct reply at any level and is immediately passed up to the top whenever it is encountered, and for the sake of clarity we omit this special case from our descriptions below.

Our correctness predicate is: $V_i(x, y, z)$ is true if either (1) $y \leq x - 2^i$ and $z \notin C_y \setminus C_x$ or (2) $y \in C_x$.

Machine R_0 on input x first checks if x is a leaf. If so, and if $e(x)$ is a colour of x, then R_0 returns it and this is correct for any z (otherwise it has found a witness to the CPLS instance). If x is not a leaf, R_0 returns the neighbour of x. If this reply is incorrect for some z, violating the well-definedness of our program, then we obtain a witness to the CPLS instance.

For $i > 0$, machine R_i on input x first calls $R_{i-1}(x)$ and gets an output y'. If $y' \in C_x$, then R_i returns y'. Otherwise, we can assume that $y' \leq x - 2^{i-1}$ (to be checked by F_i). In this case R_i calls $R_{i-1}(y')$ and gets an output y''.

[5] There is a similar argument relating CPLS with Pudlák's game principle A_3 (in [26]), to which these shallow recursion machines seem to be closely related.

Now, if $y'' \in C_{y'}$ then we assume that $V_{i-1}(x, y', y'')$ accepts (to be checked by F_i). Since y' is not a colour of x, case (1) must hold here, and we have $y'' \notin C_{y'} \setminus C_x$. Hence $y'' \in C_x$. Otherwise, if $y'' \notin C_{y'}$ then we have $y'' \leq y' - 2^{i-1} \leq x - 2^i$. In either case, R_i returns y''.

F_i, given z, first checks the assumptions made above. Next, the only way R_i's answer could be incorrect for z is if it tried to return $y'' \leq x - 2^i$, and z was such that $z \in C_{y''} \setminus C_x$. F_i therefore checks $V_{i-1}(y', y'', z)$ and $V_{i-1}(x, y', z)$, and outputs according to whichever one is false. If both of these are true, then $z \notin C_{y''} \setminus C_{y'}$ and $z \notin C_{y'} \setminus C_x$, which together contradict $z \in C_{y''} \setminus C_x$.

The size parameter a^2 is large enough to code all the possible inputs and outputs described: nodes, colours, or witnesses to the CPLS instance. Finally we show how a correct output of the top machine $R_{2|a|}$ yields a colour for the source node a. Let y be such that $V_{2|a|}(a, y, 0)$. It is not possible that $y \leq a - 2^{2|a|}$, since this number is negative. So y must be a colour of a. □

Lastly, since all these reductions are provable in PV,

Theorem 31 *As first order theories,* $\mathrm{PV} + \hat{\Sigma}^b_1(T^3_2) = \mathrm{PV} + 2\mathrm{VR}{-}\mathrm{totality}$ *and* $\mathrm{PV} + \hat{\Sigma}^b_1(T^2_2) = \mathrm{PV} + 2\mathrm{VR}(\log){-}\mathrm{totality}$.

5 Concrete reducibilities and open problems

Two prominent combinatorial principles provable in T^2_2 are the **weak pigeonhole principle** WPHP, asserting that no function can injectively map a^2 into a, and the **minimization principle** MIN asserting that a partial ordering on $[0, a]$ has a minimal element. Stated in this way MIN is a $\hat{\Sigma}^b_2$ statement but it has a natural $\hat{\Sigma}^b_1$ consequence called the **generalized iteration principle** ITER. The principle ITER says that no function can be strictly decreasing in a partial ordering (see [4]). It is known [4] that when stated with oracles neither WPHP nor ITER is provable in $T^1_2(\alpha)$.

The provability of WPHP in T^3_2 (and in T^2_2 if the function is assumed to be surjective) was established by Paris, Wilkie and Woods [24]. The surjectivity assumption was later removed by Maciel, Pitassi and Woods [18]. The principle MIN, and hence ITER, is easily provable in T^2_2 by induction on a. We now give, as an illustration of how our new search problems work, direct reductions of these principles to CPLS (that is, not going via their provability in T^2_2).

Proposition 32 ITER $\leq$ CPLS.

Proof Suppose our instance of ITER is an ordering $\preceq$ on the domain $[0,a]$ and a function $g : [0,a] \rightarrow [0,a]$. A solution is a witness that $\preceq$ is not an ordering or some x with $g(x) \not\prec x$.

We define an instance of CPLS. The nodes are pairs $(x,y) \in [0,a]^2$ with $y \leq x$, ordered lexicographically (with the order reversed, so $(a,y) < (0,y)$). The root is $(0,0)$ and the leaves are the nodes of the form (a,y).

The colours of $C_{(x,y)}$ are the set $\{z \leq x : z \prec y\}$ of witnesses that y is not a minimal element of the set $[0,x]$. The neighbour of an internal node (x,y) is $(x+1,x+1)$ if $x+1 \prec y$; otherwise it is $(x+1,y)$. So if moving from a node to its neighbour introduces a new colour, this gives a witness that $\preceq$ is not an ordering. Lastly for any leaf node (a,y), we define $e((a,y)) = g(y)$. $\square$

Proposition 33 *Consider* WPHP *in the form: if f is a function $a \longrightarrow a^2$ and g is a function $a^2 \longrightarrow a$ then g is not the inverse of f.*

This can be witnessed by a shallow 2-VR program.

Proof We adapt the proof of WPHP from [24]. Let $n = |a|$ and suppose $f : x \mapsto (f_0(x), f_1(x))$. Define $F^i_s(x) = f_{s_i}(\ldots(f_{s_2}(f_{s_1}(x)))\ldots)$, where $s_1, \ldots, s_i$ are the bits of s, and let $h(x) = F^n_x(x) + 1$.

Consider a complete binary tree of height n, with each node labelled with a number $< a$. Each leaf u (at the bottom) is labelled with $h(u)$, and if the children w_0, w_1 of a node w are labelled with y_0, y_1, then w is labelled with $g(y_0, y_1)$.

For $i = 0, \ldots, n$ the intended goal of machine R_i is to take as input the address u of a node at level i of the tree (counting from the bottom up), and output what its label y would be if f really were the inverse of g. So the correctness relation $V_i(u,y,v)$ is "$|v| \neq i$ or $|u| \neq n-i$ or $F^i_v(y) = h(u \frown v)$" and machine R_i on input x calls R_{i-1} on inputs $u0$ and $u1$ and gets replies y_0, y_1 respectively, then outputs $y = g(y_0, y_1)$.

Suppose that v witnesses that y is an incorrect label for node u. Suppose that the first bit of v is 0, so v has the form $0w$. Then the Herbrand machine F_i outputs $v' = w$, which should witness that y_0 is an incorrect label for $u0$ (and similarly if the first bit of z is 1).

Suppose this fails, and $V^{i-1}(u0, y_0, w)$ is true. Then $F_w^{i-1}(y_0) = h(u0w)$, so if $y_0 = f_0(y)$ we would have $F_{0w}^i(y) = h(u0w)$ and y would be correct for u at $v = 0w$. But y is not correct, so we have $y_0 \neq f_0(y)$. But $y = g(y_0, y_1)$, so we have found a witness that g is not the inverse of f.

We have one extra machine R_{n+1}, which is never correct (except that it will accept witnesses of the above form that get passed up the machine, as described in an earlier remark). It calls R_n and gets a label y for the root of the tree. F_{n+1} outputs $v' = y$, and $V_n(\emptyset, y, y)$ must be incorrect by the definition of $h(u)$. □

To complete the reduction of WPHP to CPLS, we informally sketch a reduction of the totality of a shallow 2-VR program to an instance of CPLS (compare the remark after Lemma 18). We will only consider programs where each R_i only makes two recursive calls, and they must both be to R_{i-1}.

The domain of our instance of CPLS is the set of labelled paths through the computation tree of the recursion program, going from the root to a leaf. The elements have the form of sequences $w_n, \dots, w_1$ as follows: for each w_i, either (1) w_i is a number x, and w_{i-1} represents the first recursion call made by P_i on input x; or (2) w_i is a pair (x, y') and w_{i-1} represents the second recursion call made by R_i on input x, if the answer to the first recursion call was y'.

To order these elements, represent each singleton x as 1 and each pair (x, y') as 0, then order the sequences lexicographically.

The neighbour of an element is the next step in the computation. For example, consider a string ending $(x_3, y_3'), x_2, (x_1, y_1')$. We simulate R_1 on input x_1, after it has made its first recursion call and got reply y_1'. It makes some second call to R_0 and gets some reply y''. It then outputs y'''. We then simulate R_2 on input x_2, after it has made its first recursion call and got reply y'''. It makes a second call to R_1, with some input x'. So the neighbour of our string will be a string ending $(x_3, y_3'), (x_2, y'''), x'$.

Finally, z is a colour of a string if it witnesses that one of the replies y' made to a recursion call was incorrect. The Herbrand functions guarantee that if z witnesses that the neighbour of a string is incorrect, then some z' (possibly different) witnesses that the original string is incorrect.

Let us conclude with some open problems. The most pressing one is surely to show that $T^3_2(\alpha)$ is not $\hat{\Sigma}^b_1(\alpha)$-conservative over $T^2_2(\alpha)$. This would follow if one constructs an oracle relative to which 2VR$-$totality is not reducible to CPLS or, equivalently, to VR$-$totality or to 2VR(log)$-$totality. This is likely also to have consequences (depending on the type of argument) for the proof complexity of $R(\log)$ of [15].

A natural NP search problem not known to be reducible to CPLS is the **Ramsey principle** RAM (cf [5]): given as an instance an undirected graph on $[0, a-1]$, find a homogeneous subgraph of size at least $\log(a)/2$. This is guaranteed to exist by a version of the finite Ramsey theorem, which Pudlák [27] has shown to be provable in T^5_2, so non-reducibility (relative to an oracle) to CPLS would imply that $T^5_2(\alpha)$ is not $\hat{\Sigma}^b_1(\alpha)$-conservative over $T^2_2(\alpha)$. Using the no-gap theorem of Chiari and Krajíček [5] this would further imply non-$\hat{\Sigma}^b_2(\alpha)$-conservativity of $T^3_2(\alpha)$ over $T^2_2(\alpha)$. (We note that such a non-conservativity is known when the smash function is not present in the theories, see [10].)

A test case for any technique aimed at showing non-reducibility to CPLS is to show that the ordinary PHP is not reducible to CPLS, without using the random restriction method of [16, 25].

Another problem which may be easier (since it does not involve proving independence from CPLS, which may be as hard as proving lower bounds for $R(\log)$) is to establish whether or not ITER is equivalent to CPLS.

References

[1] Paul Beame, Stephen Cook, Jeff Edmonds, Russell Impagliazzo, and Toniann Pitassi. The relative complexity of NP search problems. *Journal of Computer and System Sciences*, 57(1):3–19, August 1998.

[2] S. Buss. *Bounded Arithmetic.* Bibliopolis, Naples, 1986.

[3] Samuel Buss and Jan Krajíček. An application of Boolean complexity to separation problems in bounded arithmetic. *Proceedings of the London Mathematical Society*, 69(3):1–21, 1994.

[4] Mario Chiari and Jan Krajíček. Witnessing functions in bounded arithmetic and search problems. *The Journal of Symbolic Logic*, 63(3):1095–1115, September 1998.

[5] Mario Chiari and Jan Krajíček. Lifting independence results in bounded arithmetic. *Archive for Mathematical Logic*, 38(2):123–138, 1999.

[6] Stephen A. Cook. Feasibly constructive proofs and the propositional calculus (preliminary version). In *Conference Record of Seventh Annual ACM Symposium on Theory of Computing*, pages 83–97, Albuquerque, New Mexico, 5–7 May 1975.

[7] F. Ferreira. What are the $\forall\Sigma_1^b$-consequences of T_2^1 and T_2^2? *Annals of Pure and Applied Logic*, 75(1):79–88, 1995.

[8] J. Hanika. Herbrandizing search problems in bounded arithmetic. *Mathematical Logic Quarterly*, 50(6):577–586, 2004.

[9] J. Hanika. *Search Problems and Bounded Arithmetic*. PhD thesis, Charles University, Prague, 2004. Available via the ECCC (http://eccc.hpi-web.de/eccc).

[10] R. Impagliazzo and J. Krajíček. A note on conservativity relations among bounded arithmetic theories. *Mathematical Logic Quaterly*, 48(3):375–377, 2002.

[11] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, August 1988.

[12] Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*, volume 60 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, 1995.

[13] Jan Krajíček and Pavel Pudlák. Quantified propositional calculi and fragments of bounded arithmetic. *Zeitschr. f. Mathematikal Logik u. Grundlagen d. Mathematik*, 36(1):29–46, 1990.

[14] Jan Krajíček. Lower bounds to the size of constant-depth propositional proofs. *Journal of Symbolic Logic*, 59(1):73–86, 1994.

[15] Jan Krajíček. On the weak pigeonhole principle. *Fundamenta Mathematicae*, 170(1-3):123–140, 2001.

[16] Jan Krajíček, Pavel Pudlák, and Alan Woods. An exponential lower bound to the size of bounded depth frege proofs of the pigeonhole principle. *Random Structures and Algorithms*, 7(1):15–39, 1995.

[17] Jan Krajíček and Gaisi Takeuti. On induction-free provability. *Annals of Mathematics and Artificial Intelligence*, 6:107–126, 1992.

[18] Alexis Maciel, Toniann Pitassi, and Alan Woods. A new proof of the weak pigeonhole principle. *Journal of Computer and System Sciences*, 64(4):843–872, 2002.

[19] Tsuyoshi Morioka. *Logical Approaches to the Complexity of Search Problems: Proof Complexity, Quantified Propositional Calculus, and Bounded Arithmetic*. PhD thesis, University of Toronto, 2005. Available via the ECCC (http://eccc.hpi-web.de/eccc).

[20] Christos Papadimitriou. On graph-theoretic lemmata and complexity classes (extended abstract). In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science (Volume II)*, pages 794–801. IEEE Computer Society, 1990.

[21] Christos Papadimitriou and Mihalis Yannakakis. Optimization, approximation and complexity classes. In *20th Annual ACM Symposium on the Theory of Computing*, pages 229–234. ACM Press, 1988.

[22] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, June 1994.

[23] J. Paris and A. Wilkie. Counting problems in bounded arithmetic. In *Methods in Mathematical Logic*, volume 1130 of *LNM*, pages 317–40. Springer-Verlag, 1985.

[24] Jeff Paris, Alex Wilkie, and Alan Woods. Provability of the pigeonhole principle and the existence of infinitely many primes. *Journal of Symbolic Logic*, 53:1235–1244, 1988.

[25] Toniann Pitassi, Paul Beame, and Russell Impagliazzo. Exponential lower bounds for the pigeonhole principle. *Computational complexity*, 3:97–308, 1993.

[26] Pavel Pudlák. On Σ_1 sentences in bounded arithmetic. In preparation.

[27] Pavel Pudlák. Ramsey's theorem in bounded arithmetic. In E. Borger, H. Kleine Buning, M. M. Richter, and W. Schonfeld, editors, *Proceedings of Computer Science Logic*, volume 553 of *LNCS*, pages 308–312. Springer-Verlag, 1992.