Provably Total Functions in Bounded Arithmetic Theories R_3^i, U_2^i and V_2^i

Samuel R. Buss[*] [†]

Department of Mathematics

University of California, San Diego

Jan Krajíček[*] [‡]

Mathematics Institute

Czechoslovakian Academy of Sciences

Prague, Czechoslovakia

Gaisi Takeuti[*] [§]

Department of Mathematics

University of Illinois, Urbana-Champaign

July 11, 2002

[*] All three authors supported in part by NSF-ČSAV grant INT-8914569.

[†] Supported in part by NSF Grant DMS-8902480. Email address: sbuss@ucsd.edu.

[‡] Work performed while visiting San Diego and on leave at Urbana-Champaign. Email address: krajicek@csearn.bitnet

[§] Supported in part by NSF grant DMS-8800314.

Abstract

This paper investigates the provably total functions of fragments of first- and second-order Bounded Arithmetic. The (strongly) Σ_i^b-definable functions of S_3^{i-1} and R_3^i are precisely the (strong) $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ functions. The $\Sigma_i^{1,b}$-definable functions of V_2^{i-1} and U_2^i are the $\mathrm{EXPTIME}^{\Sigma_{i-1}^{1,p}}[wit, poly]$ functions and the $\Sigma_i^{1,b}$-definable functions of V_2^i are the $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$-functions. We give witnessing theorems for these theories and prove conservation results for R_3^i over S_3^{i-1} and for U_2^i over V_2^{i-1}.

1 Introduction

This paper discusses the Σ_i^b-definable functions of the first-order theories R_3^i and S_3^{i-1} and the closely related $\Sigma_i^{1,b}$-definable functions of U_2^i and V_2^{i-1}. In addition, we characterize the $\Sigma_i^{1,b}$-definable functions of V_2^i. We give new witnessing theorems for the appropriate fragments of these theories and prove several conservation results.

Buss [2] provided a characterization of the Σ_i^b-definable functions of S_2^i as the set of functions which are polynomial time computable with an oracle from the class Σ_{i-1}^p of the polynomial time hierarchy. Later, he characterized the Σ_i^b-definable functions of T_2^{i-1} by showing that S_2^i is conservative over T_2^{i-1} with respect to $\forall\Sigma_i^b$-consequences [3]. In this paper we establish similar characterizations of the Σ_i^b-definable functions of theories S_3^{i-1} and R_3^i. Recall that the theory R_3^i was introduced in various forms by Allen [1], Clote-Takeuti [7] and Takeuti [18]. In analogy with earlier results, we show that R_3^i and S_3^{i-1} have the same Σ_i^b-definable functions and that R_3^i is conservative over S_3^{i-1} with respect to $\forall\Sigma_i^b$-consequences.

The above characterization of provably total functions of R_3^i uses the witness function method but also requires the introduction a new notion of oracle computation: we define a *witness oracle* to be an oracle which when presented with an existential question, either responds 'No' or responds 'Yes' and provides a witness to the truth of the question; i.e., provides a instance or value of the existential quantifier proving that the answer is 'Yes'.[1] It is shown that the Σ_i^b-definable functions of R_3^i and S_3^{i-1} are precisely the functions that can be computed in time $2^{(\log n)^{O(1)}}$ time with witness oracle from the class Σ_{i-1}^p of the polynomial time hierarchy. In addition, we consider a notion of 'strongly Σ_i^b-definable' functions and also characterize the strongly Σ_i^b-definable functions of R_3^i and S_3^{i-1} as being precisely the functions that can be "strongly" computed in time $2^{(\log n)^{O(1)}}$ time with witness oracle from the class Σ_{i-1}^p of the polynomial time hierarchy. Unfortunately, we have not been able to accomplish a similar result for R_2^i with polynomial time computations; it is open whether such a theorem holds. For S_2^{i-1} there is such a theorem known: Krajicek [11] shows that the Σ_i^b-definable functions of S_2^{i-1} are precisely the functions which can be computed in polynomial time with a witness oracle from Σ_{i-1}^p.

It turns out that this investigation of first-order systems is entirely analogous to investigating the $\Sigma_i^{1,b}$-definable functions of V_2^{i-1} and U_2^i. For these systems, we prove a old conjecture of the first author [2] regarding the class of functions with first-order values which can be $\Sigma_i^{1,b}$-defined by U_2^i and V_2^i; however, the method of proof is rather different from what the first author had in mind when making the conjecture. In addition we characterize the $\Sigma_i^{1,b}$-definable functions of these theories that have second-order values.

The outline of this paper is as follows: in section 2, we introduce the computational complexity classes using witness oracles and prove several fundamental closure properties for them. In section 3, we briefly review the fragments of Bounded Arithmetic needed and prove that various complexity classes of functions can be defined in these theories. In section 4, we review the witness predicate and prove the witnessing lemma and various corollaries for the first-order systems. Section 5 is a translation of the results of section 4 to the second-order systems. The reader who is interested primarily in first order-systems may safely omit all the sections that pertain to second-order objects and second order-systems (sections 2.2, 3.3 and 5). However, the reader interested in second-order systems must read the entire paper since we frequently omit proofs in the second-order case. A summary of the main results can be found in sections 4.3 and 5.3. A couple of open questions are

[1] Our use of witness oracles is closely related to Kreisel's nocounterexample interpretation as well as to the use of Herbrand's theorem in [12, 15, 10]. See also [4, 9] for recent applications of the the use of witness oracles to Peano arithmetic.

also mentioned in section 4.3.

2 Computational Complexity

2.1 Witness Oracles and Function Complexity Classes

Complexity classes such as P, NP, the polynomial time hierarchy classes Σ_i^p and Π_i^p, PSPACE, EXPTIME, and LOGSPACE are classes of predicates; i.e., are classes of problems which are computed by resource-bounded Turing machines which provide Yes/No answers. In this section we define related classes of functions. The inputs and outputs of our functions are integers which, by standard coding methods, is equivalent to using strings of characters over a finite alphabet. The length of an integer x is the length of its binary representation and is denoted $|x|$.

The classes of functions we define below are computed with Turing machines with *witness oracles*. A witness oracle is a generalized form of an oracle: when a witness oracle is asked an existential question "$(\exists x)\varphi(x)$?", it responds either with the answer "No" or with a value for x making $\varphi(x)$ true. Since there may be multiple x's making $\varphi(x)$ true this allows the witness oracle of a degree of non-determinism. Because of this non-determinism we shall allow our functions to be multivalued. A multivalued k-ary function is a relation on $\mathbb{N}^k \times \mathbb{N}$; we write $f(\vec{x}) = y$ for $(\vec{x}, y) \in f$; we shall always assume f is total.

To motivate these complexity classes, let's consider a couple of examples of functions that use a witness oracle for an NP predicate. First, let $f(x)$ be the following multi-valued function of values x coding propositional formulas:

$$f(x) \;=\; \begin{cases} y & \text{if } y \text{ codes a satisfying assignment for } x \\ 0 & \text{if } x \text{ is not satisfiable} \end{cases}$$

The function $f(x)$ can be easily computed with a single call to a witness oracle for SAT (the set of satisfiable propositional formulas). Second, let $g(x)$ be defined to the multivalued function such that if x codes a graph G then $g(x)$ codes a clique of maximal size in G. To compute $g(x)$, find the maximal clique size using binary search with $O(\log |x|)$ many queries to an NP predicate; then ask a witness oracle for NP for a clique of that maximal size. Both $f(x)$ and $g(x)$ are in the class $\mathrm{FP}^{\Sigma_1^p}[wit, log]$ defined next.

Loosely speaking, the class $\mathrm{FP}^{\Sigma_i^p}[wit, log]$ contains the functions which are polynomial time computable with a witness oracle for Σ_i^p and with the restriction that the oracle may be queried only $O(\log n)$ times. Recall that Σ_i^p and Π_i^p are classes in the polynomial time hierarchy with $\Sigma_0^p = \Pi_0^p = P$ and $\Sigma_1^p = \mathrm{NP}$ and $\Pi_1^p = \mathrm{coNP}$, etc.

Definition $\mathrm{FP}^{\Sigma_i^p}[wit, log]$ is the class of multivalued functions f for which there is a Turing machine M such that the following hold:

(1) M has an input x of length n and M runs in polynomial time. The value x may be a single integer or a vector of integers.

(2) M has Σ_i^p witness oracle for (w.l.o.g.) a predicate of the form

$$\Omega(q) \Leftrightarrow (\exists z, |z| < |q|^k) R(z, q)$$

where $R \in \Pi_{i-1}^p$ and k is a constant. The oracle is accessed with a query tape, an oracle response tape, a query state and oracle accept and reject states. When M enters the query state with q written on the query tape, the next configuration of M is either (i) in the oracle reject state if $\Omega(q)$ is false or (ii) in the oracle accept state with a value z written on the oracle response tape such that $R(z, q)$ holds and such that $|z| < |q|^k$. In case (1) the response tape is blank and in case (2) the tape head is at the leftmost symbol of z and the rest of the tape is blank.

(3) M makes only $O(\log n)$ many queries in any computation.

(4) At the end of the computation $M(x)$ outputs a value y such that $f(x) = y$. However, it is not necessarily the case that for any value $y = f(x)$ there is some sequence of valid oracle answers such that $M(x)$ outputs y.

The restriction that the witness oracle only be called $O(\log n)$ many times is necessary for the use of witness oracles to be meaningful: if polynomially (i.e., arbitrarily) many calls to the witness oracle were allowed, then M could use an ordinary (non-witness) oracle to get witnesses by asking a witness value one bit at a time.

Remark 1: That condition (4) allows $f(x) = y$ even if it is impossible for $M(x)$ to output y may seem surprising at first — especially since this allows the relation $f(x) = y$ to be non-recursive.[2] However, one should think of the problem of computing $f(x)$ as being the problem of searching for a y such that $f(x) = y$. From this point of view, it makes sense to say that M can compute $f(x)$, i.e., solve the search problem, even though M may not have the potential of outputting each y such that $f(x) = y$.

Let $func_M$ be the multivalued problem defined by $func_M(x) = y$ if and only if $M(x)$ *can* output y. An alternative definition of $\mathrm{FP}^{\Sigma_i^p}[wit, log]$ is that it is the class of functions f such that $f \supseteq func_M$ for M satisfying (1)-(3). It is also useful to consider the class of functions of the form $func_M$; accordingly we define:

Definition A function f is in *strong*-$\mathrm{FP}^{\Sigma_i^p}[wit, log]$ if and only if there is a Turing machine satisfying conditions (1)-(3) such that $f = func_M$.

Remark 2: It is possible to modify the definition of $\mathrm{FP}^{\Sigma_i^p}[wit, log]$ so that the witness oracle does not provide a witness until the final oracle call. This would not change the power of the witness oracle since M as defined above can be simulated by a Turing machine M' which runs the following algorithm:

[2] To construct a non-recursive f, pick A to be any non-recursive set and let $f(x) = 0$ hold for all x and let $f(x) = 1$ hold iff $x \in A$. The function f is clearly in $\mathrm{FP}^{\Sigma_i^p}[wit, log]$ since M need merely output 0 on all inputs.

Input: x

For $k = 1, \ldots, c \cdot \log n$ /* $c \cdot \log n$ = max. number of queries */

Ask oracle: "Is there a valid computation of M such that M's first $k-1$ queries are answered by $\alpha_1, \ldots, \alpha_{k-1}$ and such that the k-th query is answered 'Yes'?"

If so, set $\alpha_k =$'Yes',

Else set $\alpha_k =$'No'.

End for

Ask oracle for a witness to the true statement "There is a computation of M in which the oracle answers are $\alpha_1, \ldots, \alpha_{c \cdot \log n}$".

Output the value y which is on the output tape of the final configuration of the computation of M returned by the witness oracle.

To properly understand the above algorithm we must see why the oracle queries are Σ_i^p queries. Suppose that $\alpha_{i_1}, \ldots, \alpha_{i_r}$ are the ones among $\alpha_1, \ldots, \alpha_{k-1}$ that are equal to 'Yes'. Then the query for a computation of M can be phrased as the following Σ_i^p query about x and $i_1, \ldots, i_r$:

> "Is there (an encoding of) a computation of M such that during the computation M asks some oracle queries $q_1, \ldots, q_{k-1}, q_k$ and receives 'Yes' answers for exactly the queries q_{i_j} for $j = 1, \ldots r$ such that the witness responses β_{i_j} to the queries with 'Yes' answers satisfy $|\beta_{i_j}| < |q_{i_j}|^k$ and $R(\beta_{i_j}, q_{i_j})$?"

This is a Σ_i^p query since the predicate $R(-,-)$ is a Π_{i-1}^p property. Note that the oracle does not have to check if the negative responses by the oracle are correct (this would would make the query too complex anyway) since the α_j's are chosen greedily to be 'Yes' if possible. This is because the sequence $\alpha_1, \ldots, \alpha_{k-1}$ is the lexicographically largest possible sequence of Yes/No oracle answers (taking 'Yes' as greater than 'No' for the lexicographic ordering); hence if the 'Yes' answers are correct with correct witnesses β_{i_j} then the 'No' answers are necessarily correct.

One consequence of this remark is that the machine M may be restricted to use only $O(\log n)$ space until the final witness oracle query (for the purposes of measuring space, the query tape is write-only, is erased after each query, and is not counted in the space computation); the response to the final query is a polynomial size witness which w.l.o.g. contains the output of M as a substring. Thus M may operate in $O(\log n)$ space until its final query, at which point it merely copies the output from (part of) the response tape to the output tape. Accordingly, another possible name for this function complexity class is $FL^{\Sigma_i^p}[wit, log]$. It can be shown using well-known techniques that the restriction that only $O(\log n)$ queries may be made to the witness oracle may be dropped for FL function classes, and thus this class is the same as $FL^{\Sigma_i^p}[wit]$ (see [8, 6, 19] for these techniques).

It is interesting to note that $func_{M'}$ may not be the same as $func_M$; since not every valid computation of M receives the lexicographically largest sequence of possible Yes/No answers. Part of our reason for using the class $\mathrm{FP}^{\Sigma_i^p}[wit, log]$ instead of strong-$\mathrm{FP}^{\Sigma_i^p}[wit, log]$ is to make Remark (2) hold.

Remark 3: It is also possible to allow M unlimited queries to a Σ_{i-1}^p oracle without changing the class $\mathrm{FP}^{\Sigma_i^p}[wit, log]$. This is because a polynomial time computation with unlimited queries to Σ_{i-1}^p may be simulated by making a single query to a witness oracle for Σ_i^p; namely, ask the witness oracle if there is a correct computation of M with correct answers to the Σ_{i-1}^p queries; of course, there is always a unique correct computation and the witness oracle returns it on its response tape.

Remark 4: It would be possible to define a class of predicates $P^{\Sigma_i^p}[wit, log]$ by considering the class of 0/1-valued functions in $\mathrm{FP}^{\Sigma_i^p}[wit, log]$. However, using the method of Remark 2, it is easy to see that this would be the class $P^{\Sigma_i^p}[log]$ which uses a regular (non-witness) oracle. This is the class of predicates polynomial time truth-table reducible to Σ_i^p (see Krentel [14], Buss-Hay [6], Wagner [19]). Krajíček [11] shows that these are precisely the predicates Δ_{i+1}^b-definable in S_2^i.

Similar considerations show that if a function in $\mathrm{FP}^{\Sigma_i^p}[wit, log]$ is constrained to output only values of length $O(\log n)$ bits, then it is in the class $\mathrm{FP}^{\Sigma_i^p}[log]$ which is defined as above but with a (non-witness) oracle for Σ_i^p.

Remark 5: Krentel [14] gave the original definition of the class $\mathrm{FP}^{\Sigma_i^p}[log]$ of functions. Our function class of strong-$\mathrm{FP}^{\Sigma_i^p}[wit, log]$ with witness oracles provides a seemingly different and possibly more natural function class. For example, the multivalued function defined by $f(x) = y$ if and only if either $y = 0$ or x codes a Boolean formula with y a satisfying assignment, is clearly in strong-$\mathrm{FP}^{\mathrm{NP}}[wit, log]$ since M can ask the witness oracle for a satisfying assignment of x. However, Krentel showed that this function is in $\mathrm{FP}^{\mathrm{NP}}[log]$ if and only if $\mathrm{P} = \mathrm{NP}$. On the other hand, our function class defined in terms of witness oracles seems to have the inherent disadvantage of containing multivalued functions.

For use with the theory R_3^i, we need to a slight modification of the above function class to reflect the presence of the $\#_3$ function in the language:

Definition $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ is the class of multivalued functions defined in exactly the same way as $\mathrm{FP}^{\Sigma_i^p}[wit, log]$ except that the runtime of the Turing machine is bounded by $2^{(\log|x|)^{k_1}}$ and the number of oracle queries is bounded by $(\log|x|)^{k_2}$ for some constants k_1, k_2.

The runtime bound $2^{(\log n)^{O(1)}}$ on inputs of length n is called "$\#_3$ time".

2.2 Functions of second-order objects

We next consider higher-order analogues of $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$. There are essentially two modifications: first, the computational complexity will be exponential time or polynomial space and, second, there will two different kinds (called *orders*) of inputs — first-order inputs of length n and second-order inputs of length $2^{n^{O(1)}}$. In our applications to second-order theories U_2^i and V_2^i, these two kinds of inputs correspond to first- and second-order variables.

The class $\Sigma_i^{1,p}$ is the class of predicates which can be defined by a $\Sigma_i^{1,b}$ formula; in terms of Turing machines, this is the class of the predicates that can be recognized by a $2^{n^{O(1)}}$-time Turing machine which has i blocks of existential and universal alternations beginning with an existential block.

Definition $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$ is the class of single-valued functions f which are computed by a Turing machine M such that

1. M has first-order input x of length n (x may be a vector of values, in which case n is the total length of the first-order inputs). And M has second-order inputs $\vec{\varphi}$. Each second-order input is a string of symbols written on its own input tape.

2. M has run time bounded by 2^{n^k} for some constant k. Thus M can access only 2^{n^k} many squares of the second-order input tapes and M can ask oracle queries of length up to 2^{n^k} symbols.

3. M has a (nonwitness) oracle for a predicate in Σ_i^p. Since exponentially long queries are allowed, this corresponds to asking $\Sigma_i^{1,p}$ queries about the first-order inputs.

4. M outputs either a second-order value or a first-order value. Any first-order output must have length bounded by $n^{k'}$ for some constant k'.

The computational power $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$ would not be significantly changed if M was allowed to use a witness oracle; this is because the number of oracle queries by M is not restricted and M can ask repeated oracle queries to obtain witnesses one bit at a time.

Definition $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$ is a class of multivalued functions; it is defined to be the set of functions such that there is a Turing machine satisfying the conditions 1.-4. above, except that, firstly, the third condition is modified so that M has a witness oracle for Σ_i^p and M may only make n^{k_0} queries to the witness oracle for some constant k_0, and secondly, if $M(x, \vec{\varphi})$ can output y or ψ then $f(x, \vec{\varphi}) = y$ or $f(x, \vec{\varphi}) = \psi$ (but not necessarily conversely).

Strong-$\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$ is the class of functions $func_M$ for M satisfying the modified conditions 1.-4.

The remarks above about $\mathrm{FP}^{\Sigma_i^p}[wit, log]$ above apply also to $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$. For example, the Turing machine M for $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$ may be restricted so that only its final oracle query returns a witness; likewise M may be restricted to use only polynomial space until after the final witness query it copies part of the response tape to the output tape. As before, the write-only query tape is erased after each query and is not considered in the space computation. Also, an $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$ Turing machine M may make unrestricted queries to a Σ_{i-1}^p oracle. If M outputs only first-order values then a usual oracle for $\Sigma_i^{1,b}$ suffices and the use of the witness oracle is unnecessary.

Finally, it should be noted that $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$ and $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$ may be regarded as being the same classes as $\mathrm{FP}_3^{\Sigma_i^p}$ and $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ if the second-order inputs are reinterpreted as being first order inputs. This is because $\#_3$-time of inputs of length $2^{n^{O(1)}}$ is the same thing as time $2^{n^{O(1)}}$. This is related to the 'RSUV isomorphism' discussed in the next section.

3 Fragments of Bounded Arithmetic

3.1 Preliminaries

In this section we review the fragments of Bounded Arithmetic that are used in this paper. We shall assume familiarity with Buss [2] and shall need some theorems from Buss [3]. The systems we shall deal with are R_3^i, S_3^i, S_2^i, T_2^i, U_2^i and V_2^i. The subscript indicates the growth rate of function symbols in the language; the subscript 2 indicates that 0, S, $+$, $\cdot$, $\#$, $\lfloor\frac{1}{2}x\rfloor$ and $|x|$ are function symbols and the subscript 3 indicates that the $\#_3$ function is also in the language where $x\#_3y = 2^{|x|\#|y|}$. The function $\#_2$ has polynomial growth rate and the growth rate of $\#_3$ is superpolynomial and subexponential: the $\#_2$ function allows formation of terms with growth rate $2^{|x|^{O(1)}}$ and the $\#_3$ function allows formation of terms with growth rate $2^{2^{|(|x|)|^{O(1)}}}$. In addition R_3^i has function symbols $\dot{-}$ and MSP for subtraction and "most significant part". (This choice of functions symbols avoids problem with bootstrapping and makes R_3^0 a useful theory. We shall not discuss the details of bootstrapping in this paper; since we are dealing primarily with R_3^i with $i > 1$, it is only necessary to show that R_3^i contains S_3^{i-1} and then the well-known bootstrapping for S_3^1 applies.)

The definition of the classes Σ_i^b and Π_i^b of bounded formulas is as usual, counting alternations of bounded quantifiers $(Qx \leq t)$ but ignoring sharply bounded quantifiers of the form $(Qx \leq |t|)$. The terms in bounded quantifiers may contain the $\#_3$ function if it is in the language. Recall that Σ_i^b and Π_i^b formulas represent precisely the predicates in the corresponding level of the polynomial time hierarchy when the language does not contain $\#_3$; if the language does contain $\#_3$ then these formulas can represent precisely the predicates in the corresponding levels of the $\#_3$-time hierachy.

S_2^i and S_3^i are axiomatized with the Σ_i^b-PIND rule and T_2^i is axiomatized with the the Σ_i^b-IND rule. The definitions of R_2^i and R_3^i are based on the work of Allen [1] who dealt with a theory D^i which is equivalent to R_2^i and on the independent work of Clote-Takeuti [7]. This paper uses the definition for R_2^i and R_3^i from Takeuti [18]. The axioms of R_2^i and R_3^i are the BASIC axioms which define the function symbols and the Σ_i^b-LBIND rules:

$$\frac{A(\lfloor\frac{1}{2}a\rfloor),\Gamma\longrightarrow\Delta,A(a)}{A(0),\Gamma\longrightarrow\Delta,A(|t|)}$$

where Γ and Δ are arbitrary cedents of formulas and $A\in\Sigma_i^b$ and the *eigenvariable* a must not occur in the lower sequent. This is equivalent to the Σ_i^b-LLIND rule

$$\frac{A(a),\Gamma\longrightarrow\Delta,A(Sa)}{A(0),\Gamma\longrightarrow\Delta,A(||t||)}$$

Allen [1] showed that R_2^i and R_3^i prove the Δ_i^b-comprehension axioms. Clote-Takeuti [7] and Takeuti [18] showed that R_2^i and R_3^i prove the Π_i^b-separation axioms. Takeuti and Allen also showed that R_3^i contains the theory S_2^{i-1}, for $i\geq 1$ (the method of proof is similar to the proof that S_2^i contains T_2^{i-1}). The next theorem, due to Allen, generalizes these three results since Σ_i^b-replacement implies S_2^{i-1} is shown by Buss [2] and since it is easy to see directly that Σ_i^b-replacement implies Π_i^b-separation.

Theorem 1 (Allen [1]) *Σ_i^b-replacement is a consequence of R_2^i and R_3^i.*

Proof Recall that the Σ_i^b-replacement axioms can be stated as

$$\begin{aligned}&(\forall x\leq|t|)(\exists y\leq s)A(x,y)\rightarrow\\&\qquad(\exists w\leq SqBd(s,t))(\forall x\leq|t|)(A(x,\beta(Sx,w))\wedge\beta(Sx,w)\leq s)\end{aligned}$$

where $A(x,y)$ is a Σ_i^b-formula, possibly with other free variables besides x and y, and where w.l.o.g. the term s does not contain x. Here $SqBd(s,t)$ is a term that bounds the size of a minimal Gödel number of a sequence of $|t|+1$ numbers of values $\leq s$ [2]. Let X and Y be the hypothesis and the conclusion (respectively) of the above replacement axiom and let $Z(j)$ be the formula

$$\begin{aligned}&(\forall u\leq|t|)(\exists w\leq SqBd(s,t))(\forall x\leq|t|)\\&\qquad[(x\leq j\wedge u+x\leq|t|)\rightarrow A(u+x,\beta(Sx,w))\wedge\beta(Sx,w)\leq s].\end{aligned}$$

Now it is trivial that R_2^i and R_3^i can prove $X\rightarrow Z(0)$ and it is not hard to see that they also prove $Z(\lfloor\frac{1}{2}j\rfloor)\rightarrow Z(j)$ and also that they prove $Z(|t|)\rightarrow Y$. By Σ_i^b-LBIND on Z, they prove $Z(0)\rightarrow Z(|t|)$ and hence R_2^i and R_3^i prove the Σ_i^b-replacement axiom. $\Box$

U_2^i and V_2^i are second-order systems axiomatized with the comprehension rule for bounded first-order ($\Sigma_0^{1,b}$) properties and with induction rules $\Sigma_i^{1,b}$-PIND and $\Sigma_i^{1,b}$-IND, respectively [2]. It is easy to see that $U_2^{i+1} \supseteq V_2^i$ by the well-known methods (analogously to the proof that S_2^{i+1} contains T_2^i, or more precisely, to the proof that R_3^{i+1} contains S_3^i). There is a sharp analogy between the second-order theories U_2^i and V_2^i and the first-order theories R_3^i and S_3^i, respectively. This analogy is called the "RSUV isomorphism and is developed by [17, 18]. The basic idea of the RSUV isomorphism is that bounded second-order objects in one of the second-order theories correspond to first-order objects in the first-order theory and that first-order objects in the second-order theory correspond to lengths of objects in the appropriate first-order theory. By a bounded second-order object, we mean a predicate φ on the integers $< x$ for some first-order object x; to make the correspondence between a bounded second-order object φ and a first-order object y, we interpret the truth values of $\varphi(i)$ for $i < x$ as the bits in the binary representation of y. This makes a second-order quantifier correspond to a (bounded) first-order quantifier and makes a bounded first-order quantifier correspond to a sharply bounded quantifer. Since the second-order theory has $\#_2$, the corresponding first-order theory has the *lengths* of integers closed under $\#_2$, i.e., the first-order theory must have integers closed under $\#_3$. In addition, the $\Sigma_i^{1,b}$-PIND axioms of the theory U_2^i correspond to Σ_i^b-LBIND axioms of R_3^i. Similarly, the $\Sigma_i^{1,b}$-IND axioms of the theory U_2^i correspond to Σ_i^b-LIND axioms of S_3^i. The analogies are summarized by the table below:

Second-order Theory	First-order Theory
Second-order object (predicate)	First-order object (integer)
First-order object (integer)	Length of an integer
$\#_2$ on first-order objects	$\#_3$ on first-order objects
$\Sigma_i^{1,b}$	Σ_i^b
PIND/LIND	LBIND/LLIND
IND	LIND/PIND
$\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$	$FP_3^{\Sigma_i^p}$
$\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$	$\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$
U_2^i	R_3^i
V_2^i	S_3^i

The Σ_i^b-definable functions of some of these first-order theories can be characterized: For S_2^i and T_2^{i-1}, the Σ_i^b-definable functions are precisely the $\square_i^p$ functions, i.e., the $\mathrm{FP}^{\Sigma_{i-1}^b}$ functions [2, 3]. For R_2^1 and $i = 1$, they are precisely the NC functions, see Allen [1] and Clote-Takeuti [7]. The $\Sigma_1^{1,b}$-definable functions of U_2^1 and V_2^1 were shown by Buss [2] to be precisely the polynomial space and exponential time computable functions, respectively. Buss [2] made a conjecture about the $\Sigma_i^{1,b}$-definable first-order-valued functions of U_2^i and V_2^i. We prove this conjecture, and we also characterize the functions with second-order values that can be $\Sigma_i^{1,b}$-defined by U_2^i and V_2^i; namely, they are the $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$ and $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$ functions, respectively.

Also in this paper we characterize the Σ_i^b-definable functions of S_3^{i-1} and R_3^i as being precisely the $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ functions, for $i > 1$.

The prior characterizations for R_2^1, S_2^i, U_2^1 and V_2^1 concerned the definability of *single-valued* functions. However, to work with the classes $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ we also require a notion of Σ_i^b-definition of a multivalued function.

Definition Let f be a multivalued function (i.e. a relation). Then we say f is Σ_i^b-defined by a theory T if and only if for some Σ_i^b-formula $A(x, y)$,

(1) $T \vdash (\forall x)(\exists y)A(x, y)$, and

(2) Whenever $A(n, m)$ is true (in the standard model), then $f(n) = m$ is true.

The fact that condition (2) of the definition of Σ_i^b-definability is an "if ... then ..." is perhaps somewhat surprising; however, this matches the condition (4) of the definition of $\mathrm{FP}^{\Sigma_{i-1}^p}[wit, log]$. There is also a "strong" version of Σ_i^b-definability:

Definition A multivalued function f is *strongly* Σ_i^b*-definable* iff f is Σ_i^b-definable and for all $m, \vec{n}$, $A(\vec{n}, m)$ holds iff $f(\vec{n}) = m$.

We shall prove that the class of functions strongly Σ_i^b-definable by R_3^i (and by S_3^{i-1}) is precisely strong-$\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$.

3.2 Some Σ_i^b-Definitions of Functions in R_3^i and S_3^{i-1}

As part of characterizing the Σ_i^b-definable functions of R_3^i and S_3^{i-1} we need to give some intensional Σ_i^b-definitions of functions in these two theories. Later, we shall prove that all functions Σ_i^b-definable in R_3^i and S_3^{i-1} are captured by the next theorem. We say that M is an explicit $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ Turing machine if M has built-in 'clocks' that limit the run time to $2^{(\log n)^k}$ steps and the number of oracle queries to $\leq (\log n)^k$ on inputs of length n, for some constant k. This means that if M exceeds the runtime or oracle query limits then M aborts and outputs some constant (say 0). Given a description of a Turing machine with a witness oracle for a Σ_i^b-predicate Ω the property "w codes a valid execution of M on input x" can be expressed as a $\Sigma_{i+1}^b \cap \Pi_{i+1}^b$ formula $Run_M(x, w)$. Loosely speaking $Run_M(x, w)$ states that w completely codes a computation of $M(x)$ and that for each query q made to the witness oracle ω either (1) $\Omega(q)$ is false and M entered the oracle reject state after the query, or (2) M entered the oracle accept state with the response tape containing a value z witnessing the truth of $\Omega(q)$.

Theorem 2 *Fix $i \geq 1$. Let M be an explicit* $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ *Turing machine.*

(a) $R_3^{i+1} \vdash (\forall x)(\exists w) Run_M(x, w)$.

(b) $S_3^i \vdash (\forall x)(\exists w) Run_M(x, w)$.

Corollary 3 *Let $i > 1$. R_3^i and S_3^{i-1} can each Σ_i^b-define every* $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ *function.*

It is an open question whether R_3^1 can Σ_1^b-define all $\#_3$-time functions (this would be $i = 1$ of part (a) of the theorem). What is known is that R_3^1 can Σ_1^b-define exactly the polylog-space computable functions; this follows from the 'RSUV isomorphism' and from the characterization of the provably total functions of U_2^1 as the polynomial space functions.

Proof Since $R_3^{i+1} \vdash S_3^i$ it will suffice to prove (b). We fix M an $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ machine, which has a fixed oracle $\Omega \in \Sigma_i^p$ and runs in time $2^{(\log n)^k}$ and asks $(\log n)^k$ queries on inputs of length n. The oracle $\Omega(q)$

is of the form $(\exists z \leq t(q))B(z,q)$ for some $B \in \Pi^p_{i-1}$ and some term t in the language of S^i_3. We reason inside S^i_3. We say that w codes a *precomputation* of M if w is a sequence of configurations of M's execution respect to an unspecified oracle; that is, w being a precomputation implies nothing about whether the oracle answers in w are correct. A Q-computation is a precomputation in which all the 'Yes' answers are correct for the oracle Ω (but the 'No' answers may be incorrect). Let $QComp_M(w,x,v)$ be the following formula which states that w codes a Q-computation of M on input x such that all the 'Yes' answers are correct with a correct response tape contents and such that, for all j, the j-th most significant bit of v is a "1" if and only if the j-th query of $M(x)$ to the witness oracle yields a 'Yes' answer.

$$
\begin{aligned}
QComp_M(w,x,v) \;\Leftrightarrow\; & \\
& w \text{ codes an precomputation of } M(x) \text{ and} \\
& (\forall j \leq (\log|x|)^k)[YesAns(w,j) \rightarrow Bit((\log|x|)^k - j, v) = 1 \text{ and} \\
& (\forall j \leq (\log|x|)^k)[Bit((\log|x|)^k - j, v) = 1 \rightarrow CorrectYes(w,j)]
\end{aligned}
$$

where the formula $YesAns(w,j)$ asserts the j-th oracle query in the precomputation w receives a 'Yes' answer and the formula $CorrectYes(w,j)$ asserts that the j-th oracle query in the precomputation w yields a 'Yes' answer and that $B(z,q) \wedge z \leq t(q)$ holds, where q is the j-th oracle query in w and z is the response tape contents after the j-th oracle query. It is easy to see that $CorrectYes$ and $QComp_M$ are Π^b_{i-1} formulas, since B is (unless $i=1$ in which case, they are Δ^b_1 formulas). $YesAns$ is, of course, always a Δ^b_1-formula.

By coding precomputations efficiently, it can be presumed that any precomputation w for $M(x)$ can be bounded by some term $r(x)$; this is because M runs in $\#_3$ time. Likewise the v in $QComp_M(w,x,v)$ has length $\leq (\log|x|)^k$ and hence $v \leq |s(x)|$ for some term s in the language of S^i_3. Thus the formula $(\exists w)QComp(w,x,v)$ is (equivalent to) a Σ^b_i-formula. Since S^1_3 can prove that deterministic, non-oracle, $\#_3$-time Turing machines always halt, it follows that S^i_3 proves that $(\exists w)QComp(w,x,0)$; namely, S^i_3 proves that there is precomputation of $M(x)$ with all the oracle queries answered 'No'. And since S^i_3 admits the "length maximization axioms" Σ^b_i-LMAX, S^i_3 can prove that there exists a maximum $v < |s(x)|$ such that $(\exists w)QComp(w,x,v)$. Let v now denote this maximum value; it follows that S^i_3 can prove that if $QComp(w,x,v)$ then w codes a computation with all witness oracle answers correct. To prove this in S^i_3, one argues that all 'Yes' answers must be correct since $QComp_M$ holds and that, for any j, if the j-th 'No' answer were incorrect v would not be maximum since one could obtain a larger value v' by changing the j-th most significant bit of v to "1" and setting all lower bits to zero. Then from $(\exists w)QComp(w,x,v)$, S^1_3 proves that $(\exists w')QComp(w',x,v')$ by letting w' code the precomputation corresponding to w up to the j-th query, then coding a 'Yes' answer with a valid witness on the response tape for the j-th query and subsequently coding M's computation with all oracle queries returning 'No' answers.

Thus S^i_3 can prove that $M(x)$ always has at least one valid computation and S^i_3 can Σ^b_{i+1}-define the function which M computes. The formula which Σ^b_{i+1}-defines the function $M(x)=y$ is the formula

$$(\exists y)(\exists w)[Run_M(x,w) \text{ and computation } w \text{ outputs } y]$$

which asserts that there is a w encoding a valid computation of $M(x)$ which outputs the value y. Note that w and y can be bounded by terms involving x. Q.E.D. Theorem 2

Corollary 4 *Let $i>1$. R^i_3 and S^{i-1}_3 can each strongly Σ^b_i-define every strong $\mathrm{FP}_3^{\Sigma^p_{i-1}}[wit, log^{O(1)}]$ function.*

Proof This follows immediately from Theorem 2 since the Σ^b_i-definition of $func_M$,

$$(\exists y)(\exists w)[Run_M(x,w) \text{ and computation } w \text{ outputs } y],$$

is also a strong Σ^b_i-definition of $func_M$. □

It should be noted that the proof of Theorem 2 involved formalizing, in S^i_3, the algorithm of Remark 2 above.

It is easy to see that R^i_3 and S^{i-1}_3 prove that the class $\mathrm{FP}_3^{\Sigma^p_i}[wit, log^{O(1)}]$ is closed under composition. Next we show that R^i_3 and S^{i-1}_3 are also able to prove that the class $\mathrm{FP}_3^{\Sigma^p_i}[wit, log^{O(1)}]$ is closed under *limited logarithmic recursion on notation*. We say that f is *defined from g and h by limited logarithmic recursion on notation with bound k* if and only if f is defined by the following:

$$f(x,\vec{y}) \;=\; f^*(|x|,\vec{y})$$

where

$$f^*(0,\vec{y}) \;=\; g(\vec{y})$$

$$f^*(z,\vec{y}) \;=\; \min\{h(z,\vec{y},f^*(\lfloor\tfrac{1}{2}z\rfloor,\vec{y})),k(\vec{y})\} \qquad \text{for } z \neq 0.$$

The definition of limited logarithmic recursion on notation is phrased so that the only purpose of k is to bound the size of the values of $f^*(z,\vec{y})$. Thus we shall henceforth consider only bounds $k(\vec{y})$ of the form $k_c(\vec{y}) = 2^{2^{|(|\vec{y}|)|^c}}$ where c is a constant.

Theorem 5 *Let $i \geq 1$. Suppose M_g and M_h are explicitly $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ Turing machines which compute multivalued functions g and h and suppose $c > 0$. Then there is a canonical, explicitly $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ Turing machine M computing the function f obtained by limited logarithmic recursion from g and h with bound k_c such that S_3^i can prove that the function computed by M_f satisfies the defining equations for f in terms of g and h.*

Proof It is easy enough for S_3^i to construct M_f from M_g, M_h and c so that M_f computes f in the obvious straightforward manner. It is also easy for S_3^i to prove that M_f has $\#_3$-runtime and only asks $(\log(|x|+|\vec{y}|))^{O(1)}$ oracle queries, since M_g and M_h also satisfy such bounds and the computation of f consists of computing g once and iterating h only $(\log|x|)$ many times. $\Box$

Next we show that $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ is closed under a form of parallel computation.

Definition Suppose $n \geq 1$ and f is an n-ary multivalued function. Then $\overline{f}$ is the multivalued function defined by:

$$\overline{f}(m,\vec{x}) \;=\; \langle f(0,\vec{x}), f(1,\vec{x}), \ldots, f(|m|-1,\vec{x})\rangle. \tag{1}$$

Theorem 6 *Fix $i \geq 1$.*

(a) *Suppose f is a $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ function. Then $\overline{f}$ is also a $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ function.*

(b) *Let M_f be an explicitly* $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ *Turing machine computing a function f. Then there is a canonical* $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ *Turing machine $M_{\overline{f}}$ computing the function $\overline{f}$ such that S_3^i can prove that for every value of $func_{M_{\overline{f}}}(m, \vec{x})$ there are values of $f(0, \vec{x}), \ldots, f(|m|-1, \vec{x})$ that satisfy equation (1).*

Remark: Theorem 6 can be strengthened by additionally requiring in (b) that S_3^i can prove that for all values of $f(0, \vec{x}), \ldots, f(|m|-1, \vec{x})$, there is a value of $func_{M_{\overline{f}}}(m, \vec{x})$ which makes equation (1) true. An indirect way to prove this is to use Theorem 20 below and the fact that R_3^i proves Σ_i^b-replacement.

Proof (The main ideas of this proof may be found already in [8, 5, 19].) We shall prove (a) and leave it to the reader to show that for fixed M_f, the proof can be formalized in S_3^i. Let M_f be an $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ Turing machine that runs in time $2^{(\log n)^k}$ and asks $(\log n)^k$ queries to a Σ_i^p witness oracle $\Omega(q) = (\exists z \le t(q))B(q, z)$ on inputs of length n. Let $\overline{M}$ be the Turing machine which computes $\overline{f}$ in the straightforward way by running M on the inputs $0, \vec{x}$, the inputs $1, \vec{x}$, etc., up to $|m|-1, \vec{x}$. The runtime of $\overline{M}$ is $O(|m| \cdot 2^{(\log n)^k})$ which is $O(2^{(\log n)^{k+1}})$; however, the number of witness oracle queries made by $\overline{M}$ is only bounded by $|m| \cdot (\log n)^k$ which is $O(n(\log n)^k)$. So $\overline{M}$ has the desired runtime but makes far too many oracle queries and we need to devise a more sophisticated Turing machine $M_{\overline{f}}$ which is in $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ and computes the same function as $\overline{M}$ (or, more correctly, $func_{M_{\overline{f}}} \subseteq func_{\overline{M}}$).

By Remark 2 we may assume that M_f ignores the contents of its oracle response tape until after the final query to the witness oracle. It follows that, for any fixed input values, there are at most $2^{(\log n)^k} - 1$ many possible oracle queries which can be made in any precomputation of M on those inputs (recall that a precomputation need not have correct oracle answers). This is because the i-th query of M will depend only on the inputs and on the prior Yes/No answers of the oracle and because M asks at most $(\log n)^k$ many oracle queries. Thus, for an fixed input values $m, \vec{x}$, there is a set containing at most $|m| \cdot (2^{(\log n)^k} - 1)$ queries which contains all the queries that $\overline{M}$ may ask in any precomputation. This set of queries can be indexed by pairs (i, j) where $i < |m|$ and $0 < j < 2^{(\log n)^k}$; namely, let $\ell = |j| - 1$ and define $q_{i,j}$ to be the $(\ell+1)$-st query in a precomputation of $M(i,\vec{x})$ if, for all $k<\ell$, the $(k+1)$-st query in the pre-computation was answered 'Yes' iff $Bit(k,j)=1$. In other words, $q_{i,j}$ is the next query $M(i,\vec{x})$ will ask if the prior queries were answered as specified by the bits of the binary representation of j. Form the array

$$\left(q_{i,j} : 0 \le i < |m|, 0 < j < 2^{(\log n)^k}\right)$$

of all possible queries in any precomputation of $\overline{M}(m,\vec{x})$.

We are now ready to describe the Turing machine $M_{\overline{f}}$. First, $M_{\overline{f}}$ computes all the entries (queries) $q_{i,j}$ in the array. Second, $M_{\overline{f}}$ uses a binary search procedure to find the number of entries q in the array such that $\Omega(q)$ holds. This is accomplished by asking queries of the following form, for p an integer:

"Do there exist at least p many entries q such that $\Omega(q)$?"

Since $\Omega \in \Sigma_i^p$, these queries are also Σ_i^p properties (of p and the array of queries). The response to such a query either is a 'No' answer or is a 'Yes' answer and an array of values z which witness at least p of the queries satisfying Ω. After $|m|+(\log n)^k$ many queries, $M_{\overline{f}}$ has ascertained the precise number of entries which for which $\Omega(q)$ is true and on the response tape there is any array of values $z_{i,j}$ which either indicate that $\Omega(q_{i,j})$ is false or which are a witness to the truth of $\Omega(q_{i,j})$. Third, $M_{\overline{f}}$ simulates the execution of $\overline{M}$ except that whenever $\overline{M}$ would query the oracle, $M_{\overline{f}}$ instead looks up the answer (which is already on $M_{\overline{f}}$'s response tape).

That completes the proof of part (a). The reader should convince him- or herself that this argument can be formalized in S_3^i. It might be useful to remember that S_3^i admits PIND for $\Sigma_{i+1}^b \cap \Pi_{i+1}^b$ predicates [3]. For instance, the formula expressing the property that there are $\ge p$ and $< p'$ many entries $q_{i,j}$ which satisfy $\Omega(q_{i,j})$ is a Boolean combination of Σ_i^b-formulas and S_3^i admits PIND for such a formula.
Q.E.D. Theorem 6

Theorem 6 is quite useful in a variety of situations. As one application, let $P(x,\vec{y},z)$ be a predicate; we say that $h(\vec{y},z)$ is defined by *length-bounded minimization* from P if

$$h(\vec{y},z) = \begin{cases} \text{the least } x \le |z| \text{ such that } P(x,\vec{y},z) \text{ if such an } x \text{ exists} \\ |z|+1 \quad \text{otherwise} \end{cases}$$

We use $h(\vec{y},z) = (\mu x \le |z|)P(x,\vec{y},z)$ as a compact notation for definition by length-bounded minimization.

Theorem 7 *$(i \ge 1)$ Let $P(x,\vec{y},z)$ be a Π_i^p-predicate. Then the function $h(\vec{y},z)$ defined from P by length-bounded minimization is in $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$. Furthermore, there is a canonical, explicitly $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ Turing machine M such that S_3^i can prove the the function h computed by M satisfies the above defining equation for length-bounded minimization.*

Proof Let $f(x,\vec{y},z)$ be the function which is equal to 1 if $P(x,\vec{y},z)$ holds and is equal to 0 otherwise. Clearly f is in $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ since it can be computed with a single (non-witness) oracle query to the Π_i^p-predicate P. By Theorem 6, it follows that the function $\overline{f}$ is also in $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$. And h can be easily computed in polynomial time without any further oracle queries from $\overline{f}(2z+1,\vec{x},z)$ since $|2z+1| = |z|+1$. $\Box$

3.3 Some Function Definitions in U_2^i and V_2^i

In this section we shall begin the investigation of the functions which are $\Sigma_{i+1}^{1,b}$-definable in the three theories V_2^i, U_2^{i+1} and V_2^{i+1} where $i \ge 1$. It is shown in this section that every $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$ function is $\Sigma_{i+1}^{1,b}$-definable in V_2^{i+1} and that every $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$ function is $\Sigma_{i+1}^{1,b}$-definable in the theories V_2^i and U_2^{i+1}. Recall that Buss [2] has shown that the $\Sigma_1^{1,b}$-definable functions of U_2^1 and V_2^1 are precisely the polynomial space and exponential time computable functions, respectively (in [2], this was only shown for functions with first-order values, but the methods immediately extend to functions with second-order values). Because of the "RSUV isomorphism", the theories U_2^i and V_2^i are in some sense equivalent to R_3^i and S_3^i; thus we shall often just outline or omit proofs because they will precisely parallel the proofs already given for R_3^i and S_3^i.

A Turing machine M is said to be an explicit $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$ Turing machine or an explicit $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$ Turing machine if it has builtin 'clocks' that limit the run time to 2^{n^k} step and, in the latter case, the number of oracle queries to n^k for some constant k. Here n is the total length of the first-order inputs to M; these runtime bounds will limit M to accessing at most the first 2^{n^k} symbols (i.e., bits) of its oracles; thus if a second-order input has a length $> 2^{n^k}$ the excess symbols are ignored.

If M has a Σ_i^p (witness) oracle let the formula $Run_M(x, \alpha, \zeta)$ state that the second-order object ζ codes a correct computation of M on input x, α: it is easy to see that Run_M is a $\Sigma_{i+1}^{1,b} \cap \Pi_{i+1}^{1,b}$ formula. (In the sequel, x and α may be vectors of first- and second-order objects, respectively.)

Theorem 8 *For $i \geq 0$ and M an explicit* $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$ *Turing machine,*

$$V_2^{i+1} \vdash (\forall x)(\forall \alpha)(\exists \zeta) Run_M(x, \alpha, \zeta)$$

Hence V_2^{i+1} can $\Sigma_{i+1}^{1,b}$-define every $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$*-function.*

The proof of Theorem 8 is entirely analogous to the proof of the the fact that S_3^{i+1} can Σ_{i+1}^b-define every function $\#_3$-time computable with an oracle for Σ_i^p. To prove this directly for V_2^{i+1}, one can modify the proof of Theorem 10.1 of [2].

Theorem 9 *Let $i \geq 1$ and M be an explicit* $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$ *Turing machine. Then*

(a) $V_2^i \vdash (\forall x)(\forall \alpha)(\exists \zeta) Run_M(x, \alpha, \zeta)$

(b) $U_2^{i+1} \vdash (\forall x)(\forall \alpha)(\exists \zeta) Run_M(x, \alpha, \zeta)$

Hence V_2^i and U_2^{i+1} can $\Sigma_{i+1}^{1,b}$-define every $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$*-function.*

The proof of Theorem 9 is analogous to the proof of Theorem 2. It suffices to prove the theorem for V_2^i since it is a subtheory of U_2^{i+1}. The notion of a precompuation of M is defined analogously as before and likewise a formula $QComp_M(\zeta, x, \alpha, v)$ is defined which says that ζ codes a precomputation of M in which the i-th oracle query returns a (correct) yes answer iff the i-th most significant bit of the binary representation of v is a "1". Note that since only polynomially many queries may be made by M, v is a first-order object. Now V_2^i can prove that there exists a maximum value for v such that $(\exists \zeta) M(\zeta, x, \alpha, v)$; this maximum value must give a true, correct computation of M. We leave it to the reader to supply the rest of the details of the proof.

It is obvious that the classes $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$ and $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$ are closed under composition, and provably so in the theories V_2^{i+1} and the theories V_2^i and U_2^{i+1}, respectively. We next discuss the closure of these classes under limited forms of primitive recursion.

Definition Let g and h be (possibly multivalued) functions which have second-order values. We say that f is defined from g and h by *first-order recursion* iff f is defined by

$$\begin{aligned} f(0, \vec{y}, \vec{\alpha}) &= g(\vec{y}, \vec{\alpha}) \\ f(x+1, \vec{y}, \vec{\alpha}) &= h(x, \vec{y}, \vec{\alpha}, f(x, \vec{y}, \vec{\alpha})) \end{aligned}$$

We say f is defined by *first-order recursion on notation* iff f is defined by

$$\begin{aligned} f(0, \vec{y}, \vec{\alpha}) &= g(\vec{y}, \vec{\alpha}) \\ f(x, \vec{y}, \vec{\alpha}) &= h(x, \vec{y}, \vec{\alpha}, f(\lfloor \tfrac{1}{2}x \rfloor, \vec{y}, \vec{\alpha})) \end{aligned}$$

Because the second-order objects have length exponential in the length of the first-order objects, one can think of "first-order" as a synonym for "logarithmic"; hence the notion of *first-order recursion on notation* is analogous to the notion of *logarithmic recursion on notation*. One important thing to note about the definition of f by first-order recursion (on notation) is that g and h and hence f must take on second-order values. Because of this there is no need to limit to growth rate of the values of f by a function k as we did in the definition of limited logarithmic recursion on notation. For this, it is important that the runtime bounds and number of queries bounds for computation in $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$ and $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$ are in terms of the total length n of the first-order inputs.

Theorem 10 *Let $i \geq 0$. Suppose M_g and M_h are explicit* $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$ *Turing machines computing functions of the appropriate number of first- and second-order arguments with second-order outputs. Then there is a canonical explicit* $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$ *Turing machine M_f computing the function f defined from g and h by first-order recursion such that V_2^{i+1} proves that the function computed by M_f satisfies the defining equation for f in terms of g and h.*

Theorem 11 *Let $i \geq 0$. Suppose M_g and M_h are explicit* $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$ *Turing machines computing multivalued functions g and h of the appropriate number of first- and second-order arguments with second-order outputs. Then there is a canonical explicit* $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$ *Turing machine M_f computing the multivalued function f defined from g and h by first-order recursion on notation such that V_2^i proves that the function computed by M_f satisfies the defining equation for f in terms of g and h.*

The proofs of Theorems 10 and 11 are based on the fact that the straightforward computation of f in terms of g and h satisfies the proper runtime bounds and, for the second theorem, makes only polynomially many oracle queries.

Next we shall define a new version of $\overline{f}$ and prove an analogue of Theorem 6. Since f will generally have second-order values, we need to define a notion of sequences of second-order object; a sequence of second-order objects can be coded by a single second-order object using the $\not\!\!\boldsymbol{\beta}$ and $\langle \cdots \rangle$ conventions from [2]. Recall that $\not\!\!\boldsymbol{\beta}(a, \alpha)$ is the abstract $\{x\}\alpha(\langle a, x\rangle)$ and that if α_i are second-order objects then

$$\langle \alpha_1, \ldots, \alpha_n \rangle$$

is the second-order object such that $\not\!\!\boldsymbol{\beta}\,(a, \langle \vec{\alpha} \rangle)$ is α_a for all $a \leq n$.

Definition Let $f(x, \vec{y}, \vec{\alpha})$ be a function which takes second-order values. The function $\overline{f}$ is defined by

$$\overline{f}(x, \vec{y}, \vec{\alpha}) \;\; = \;\; \langle f(0, \vec{y}, \vec{\alpha}), f(1, \vec{y}, \vec{\alpha}), \ldots, f(x, \vec{y}, \vec{\alpha}) \rangle. \qquad (2)$$

Note that $\overline{f}$ may be multivalued if f is.

Theorem 12 *Fix $i \geq 0$.*

(a) *Suppose f is a* $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$*-function. Then $\overline{f}$ is also a* $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$*-function.*

(b) *Suppose M_f is an explicit* $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$ *Turing machine computing a function f. There there is a canonical* $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$ *Turing machine that computes the function $\overline{f}$ such that V_2^{i+1} can prove that the function computed by the Turing machine satisfies equation (2).*

Theorem 12 is proved by noting that $\overline{f}$ can be computed from f by straightforwardly computing each requisite value of f — this can be easily formalized in V_2^{i+1}.

Theorem 13 *Fix $i \geq 1$.*

(a) *Suppose f is a $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$-function (multivalued). Then $\overline{f}$ is also a $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$-function (multivalued).*

(b) *Suppose M_f is an explicit $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$ Turing machine computing a function f. There there is a canonical $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$ Turing machine that computes the function $\overline{f}$ such that V_2^i can prove that the function computed by the Turing machine satisfies equation (2).*

Theorem 13 is proved by a rather complicated construction analogous to the proof of Theorem 6. We omit this proof.

Finally we consider *first-order minimization*. Let $P(x, \vec{y}, z, \vec{\alpha})$ be a predicate where $x, \vec{y}, z$ are first-order arguments; $h(\vec{y}, z, \vec{\alpha})$ is defined by *first-order minimization* from P if

$$h(\vec{y}, z, \vec{\alpha}) = \begin{cases} \text{the least } x \leq z \text{ such that } P(x, \vec{y}, z, \vec{\alpha}) \text{ if such an } x \text{ exists} \\ z+1 \quad \text{otherwise} \end{cases}$$

Note the function h defined by first-order minimization has a first-order value. We use $h(\vec{y}, z, \vec{\alpha}) = (\mu x \leq z) P(x, \vec{y}, z, \vec{\alpha})$ as a compact notation for definition by first-order minimization.

Theorem 14 *($i \geq 1$) Let $P(x, \vec{y}, z, \vec{\alpha})$ be a $\Pi_i^{1,p}$-predicate. Then the function $h(\vec{y}, z, \vec{\alpha})$ defined from P by first-order minimization is in $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}[wit, poly]$. Furthermore, there is a canonical, explicitly $\mathrm{FP}_3^{\Sigma_i^p}[wit, log^{O(1)}]$ Turing machine M such that V_2^i can prove the the function h computed by M satisfies the above defining equation for first-order minimization.*

Theorem 14 is proved similarly to Theorem 7; namely, apply Theorem 13 to the characteristic function of P.

4 The Σ_i^b-Definable Functions of R_3^i and S_3^{i-1}

In this section, the Σ_i^b-definable functions of R_3^i and S_3^{i-1} are characterized. We have already shown that every $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ function is Σ_i^b-definable in these theories. It will suffice to establish the converse for the stronger theory R_3^i: we shall do this by proving a 'witnessing theorem' which states that every sequent of Σ_i^b-formulas provable in R_3^i is 'witnessed' by a $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ function (provably in S_3^i). This not only characterizes the Σ_i^b-definable functions of R_3^i and S_3^{i-1}, but also gives a conservation result between the two theories and proves that S_3^{i-1} admits Δ_i^b-PIND.

4.1 The Witness Formula

We next review briefly a definition from [2] which is necessary for the the characterization of the Σ_i^b-definable functions of S_3^{i-1} and R_3^i. For the rest of this section, $i \geq 1$ will be a fixed integer; the applications in this paper only need $i > 1$. Let $A(\vec{a})$ be a Σ_i^b-formula. A formula $Witness_A^{i,\vec{a}}(w, \vec{a})$ is defined which has limited quantifier complexity and which states that w is a number 'witnessing' the truth of $A(\vec{a})$.

Definition Suppose $A(\vec{a}) \in \Sigma_i^b$ and $\vec{a}$ is a vector of variables including all those free in A. The formula $Witness_A^{i,\vec{a}}$ is defined below, inductively on the complexity of A:

(1) If $A \in \Pi_{i-1}^b$ then $Witness_A^{i,\vec{a}}$ is just A itself.

(2) If A is $B \wedge C$ then define

$$Witness_A^{i,\vec{a}}(w, \vec{a}) \iff Witness_B^{i,\vec{a}}(\beta(1, w), \vec{a}) \wedge Witness_C^{i,\vec{a}}(\beta(2, w), \vec{a}).$$

(3) If A is $B \vee C$ then define

$$Witness_A^{i,\vec{a}}(w, \vec{a}) \iff Witness_B^{i,\vec{a}}(\beta(1, w), \vec{a}) \vee Witness_C^{i,\vec{a}}(\beta(2, w), \vec{a}).$$

(4) If A is $B \to C$ and is not in $\Pi_{i-1}^{1,b}$ then we define

$$Witness_A^{i,\vec{a}}(w, \vec{a}) \iff Witness_{\neg B}^{i,\vec{a}}(\beta(1, w), \vec{a}) \vee Witness_C^{i,\vec{a}}(\beta(2, w), \vec{a}).$$

(5) If $A \notin \Pi^b_{i-1}$ and $A(\vec{a})$ is $(\forall x \leq |s(\vec{a})|)B(\vec{a},x)$ then define

$$\begin{aligned} \mathit{Witness}^{i,\vec{a}}_A(w,\vec{a}) \Longleftrightarrow\ & \mathit{Seq}(w) \wedge \mathit{Len}(w) = |s(\vec{a})| + 1 \wedge \\ & \wedge (\forall x \leq |s(\vec{a})|)\, \mathit{Witness}^{i,\vec{a},b}_{B(\vec{a},b)}(\beta(x+1,w),\vec{a},x). \end{aligned}$$

In words, w witnesses $A(\vec{a})$ if $w = \langle w_0, \ldots, w_{|s|}\rangle$ and each w_i witnesses $B(\vec{a},i)$. The formula $\mathit{Seq}(w)$ says w is a valid Gödel number of a sequence and $\mathit{Len}(w)$ is a function giving the number of entries in the sequence w.

(6) If $A \notin \Pi^b_{i-1}$ and A is $(\exists x \leq t(\vec{a}))B(\vec{a},x)$ then define

$$\begin{aligned} \mathit{Witness}^{i,\vec{a}}_A(w,\vec{a}) \Longleftrightarrow\ & \mathit{Seq}(w) \wedge \mathit{Len}(w) = 2 \wedge \beta(1,w) \leq t(\vec{a}) \wedge \\ & \wedge\, \mathit{Witness}^{i,\vec{a},b}_{B(\vec{a},b)}(\beta(2,w),\vec{a},\beta(1,w)). \end{aligned}$$

So w witnesses $A(\vec{a})$ if $w = \langle n, v\rangle$ where $n \leq t(\vec{a})$ and v witnesses $B(\vec{a},n)$.

(7) If $A \notin \Pi^b_{i-1}$ and A is $\neg B$ then use prenex operations to push the negation sign into the formula so that it can be handled by cases (1)–(6).

The purpose of defining *Witness* is to give a canonical way of verifying that $A(\vec{a})$ is true. It is easy to see that $(\exists w)\, \mathit{Witness}^{i,\vec{a}}_A(w,\vec{a})$ is equivalent to $A(\vec{a})$. The next propositions express some properties of *Witness*; these are proved mostly by induction on the complexity of A.

Proposition 15 *For $i \geq 2$, and $A \in \Sigma^b_i$, $\mathit{Witness}^{i,\vec{a}}_A$ is a Π^b_{i-1}-formula.*

Proposition 16 $(i \geq 1)$. *Let $A(\vec{a})$ be a Σ^b_i-formula. Then*

$$S^{i-1}_3 \vdash \mathit{Witness}^{i,\vec{a}}_A(w,\vec{a}) \rightarrow A(\vec{a})$$

and there is a term $t_A(\vec{a})$ such that

$$S^{i-1}_3 + \Sigma^b_i\text{-replacement} \vdash A(\vec{a}) \leftrightarrow (\exists w \leq t_A)\, \mathit{Witness}^{i,\vec{a}}_A(w,\vec{a}).$$

Also there is a Σ^b_1-defined function $g_A(w)$ such that

$$S^1_2 \vdash \mathit{Witness}^{i,\vec{a}}_A(w,\vec{a}) \rightarrow \mathit{Witness}^{i,\vec{a}}_A(g_A(w),\vec{a}) \wedge g_A(w) \leq t_A.$$

Proposition 16 also holds for S_2^{i-1} and is proved by induction on the complexity of A exactly as in the proofs of the corresponding theorems in Buss [2, 3].

Proposition 17 $(i \geq 1)$. *Let A be a Σ_i^b-formula. The predicate $Witness_A^{i,\vec{a}}$ is a Π_{i-1}^p-predicate.*

4.2 The Witnessing Theorem for S_3^{i-1} and R_3^i

In this section we give the proof that every Σ_i^b-definable function of S_3^{i-1} and R_3^i is in $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$. The proof is a proof-theoretic 'witnessing theorem' and hence is constructive; however, it uses cut elimination and is not feasibly constructive (since cut-elimination involves superexponential growth rate). We now consider the theories of Bounded Arithmetic formalized in a Gentzen-style sequent calculus: each line in a proof is a *sequent* of the form

$$A_1, \ldots, A_k \longrightarrow B_1, \ldots, B_\ell$$

where each A_j and B_j is a formula. The intended meaning of this sequent is that the conjunction of the *antecedent* $A_1, \ldots, A_k$ implies the disjunction of the *succedent* $B_1, \ldots, B_\ell$. Note that the sequent connective symbol $\longrightarrow$ is distinct from the logical connective $\rightarrow$. Capital Greek letters $\Gamma, \Delta, \Pi, \Lambda, \ldots$ will be used to denote a series of formulae separated by commas, these are called *cedents*.

There are about 23 rules of inference for the sequent calculus; in addition, there are induction rules which replace the induction axioms. The *initial sequents* (i.e., axioms) of a sequent calculus proof must be equality axioms, logical axioms or non-logical axioms. The theories S_3^{i-1} and R_3^i each have a finite set of open (i.e., quantifier free) sequents as initial sequents. There are no induction axioms as initial sequents since induction rules are used instead. An important theorem (due to Gentzen, but see also Takeuti [16]) concerning the sequent calculus is that many instances of the cut rule may be eliminated from proofs — more precisely, all *free cuts* may be eliminated from a proof. Rather than define precisely what a free cut is, let us merely say that for a proof of a Σ_i^b-formula in a theory S_3^{i-1} or R_2^i, we may assume that every formula appearing in the proof is a Σ_i^b- or a Π_i^b-formula. For more information on the sequent calculus see Takeuti [16] and for information on the sequent calculus for theories of Bounded Arithmetic, consult chapter 4 of [2].

If Γ is a cedent we write $\bigwedge\Gamma$ and $\bigvee\Gamma$ to denote the conjunction and disjunction, respectively, of the formulae in Γ. Conjunction and disjunction associate from right to left; for example, if Γ is A, B, C then $\bigwedge\Gamma$ denotes $A \wedge (B \wedge C)$.

The theory R^i_3 has the function and predicate symbols β, Seq, and Len which manipulate Gödel numbers of sequences ($i > 1$). These function symbols can be Σ^b_1-defined by S^{i-1}_3, for $i > 1$, and w.l.o.g. we assume that the language of S^{i-1}_3 is enlarged to contain these function symbols. We use $\langle a_1, \ldots, a_n\rangle$ to denote the Gödel number of the sequence $a_1, \ldots, a_n$. Also, $*$ is a binary function defined so that

$$\langle a_1, \ldots, a_n\rangle * a_{n+1} = \langle a_1, \ldots, a_n, a_{n+1}\rangle.$$

Finally $\langle\!\langle a_1, \ldots, a_n\rangle\!\rangle$ is equal to $\langle a_1, \langle a_2, \ldots, \langle a_{n-1}, a_n\rangle \ldots\rangle\,\rangle$.

These conventions allow us to conveniently discuss witnessing a cedent. For example, suppose Γ is $A_1, \ldots, A_n$ and that $w = \langle\!\langle w_1, \ldots, w_n\rangle\!\rangle$. Then $Witness^{i,\vec{a}}_{\bigwedge\Gamma}(w, \vec{a})$ holds if and only if $Witness^{i,\vec{a}}_{A_j}(w_j, \vec{a})$ holds for all $1 \le j \le n$.

Theorem 18 (The Witnessing Lemma for S^{i-1}_3 and R^i_3)
Fix $i > 1$. Suppose the sequent $\Gamma, \Pi \longrightarrow \Delta, \Lambda$ is a theorem of R^i_3 and each formula in $\Gamma \cup \Delta$ is Σ^b_i and each formula in $\Pi \cup \Lambda$ is Π^b_i. Let $c_1, \ldots, c_p$ be the free variables in the sequent and let G and H be the formulae

$$G = \left(\bigwedge\Gamma\right) \wedge \bigwedge\{\neg C : C \in \Lambda\}$$

and

$$H = \left(\bigvee\Delta\right) \vee \bigvee\{\neg C : C \in \Pi\}.$$

Then there is a $\mathrm{FP}_3^{\Sigma^p_{i-1}}[wit, log^{O(1)}]$ function f which is Σ^b_i-defined by S^{i-1}_3 such that

$$S^{i-1}_3 \vdash Witness^{i,\vec{c}}_G(w, \vec{c}) \rightarrow Witness^{i,\vec{c}}_H(f(w, \vec{c}), \vec{c}).$$

Furthermore, S^{i-1}_3 defines f as being equal to $func_M$ for some explicit $\mathrm{FP}_3^{\Sigma^p_{i-1}}[wit, log^{O(1)}]$ Turing machine.

Proof

A formula is said to be in *negation normal form* if every negation sign ($\neg$) has an atomic formula in its scope. Since any formula is logically equivalent to a formula in negation normal form, we may, without loss of generality, restrict our attention to proofs in which every formula is in negation normal form. In particular, the induction formulas and every formula in the endsequent are restricted to being in negation normal form. This simplifies the notation considerably since now Π and Λ may, without loss of generality, be presumed to be the empty cedent.

By the free-cut elimination theorem there is a R_3^i-proof P of $\Gamma \longrightarrow \Delta$ such that every formula in the proof is in negation normal form and every cut in P has a Σ_i^b principal formula and such that P is in free variable normal form (see [2] for definitions). The proof of Theorem 18 is by induction on the number of sequents in the proof P.

To begin, consider the case where P has no inferences and consists of a single sequent. This sequent must be a nonlogical axiom of R_3^i or a logical axiom or an equality axiom. In any event, it contains only atomic formulae and is also a consequence of S_3^{i-1}. For atomic formulae A, $Witness_A^{i,\vec{c}}$ is just A itself; hence this case is completely trivial.

The argument for the induction step splits into thirteen cases depending on the final inference of P. Since the general form of this induction is by now quite familiar, we shall omit the easier cases and discuss only the more difficult cases of the induction step.

Case (1): ($\vee$:left) Suppose the last inference of P is

$$\frac{B, \Gamma^* \longrightarrow \Delta \qquad C, \Gamma^* \longrightarrow \Delta}{B \vee C, \Gamma^* \longrightarrow \Delta}$$

Let D be the formula $B \wedge (\bigwedge \Gamma^*)$ and let E be $C \wedge (\bigwedge \Gamma^*)$ and let F be $(B \vee C) \wedge (\bigwedge \Gamma^*)$. By the induction hypothesis, there are $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ functions g and h such that

$$S_3^{i-1} \vdash Witness_D^{i,\vec{c}}(w, \vec{c}) \rightarrow Witness_{\bigvee \Delta}^{i,\vec{c}}(g(w, \vec{c}), \vec{c})$$

and

$$S_3^{i-1} \vdash Witness_E^{i,\vec{c}}(w, \vec{c}) \rightarrow Witness_{\bigvee \Delta}^{i,\vec{c}}(h(w, \vec{c}), \vec{c}).$$

Let the function k be defined by

$$k(v,w,\vec{c}) = \begin{cases} v & \text{if } \mathit{Witness}^{i,\vec{c}}_{\bigvee\Delta}(v,\vec{c}) \\ w & \text{otherwise} \end{cases}$$

By Proposition 17, $k(w,a,b,\vec{c})$ can be computed with a single call to a Σ^p_{i-1}-oracle; thus k is a $\mathrm{FP}_3^{\Sigma^p_{i-1}}[wit, log^{O(1)}]$ function. Let f be the function

$$f(w,\vec{c}) = k(g^*(w,\vec{c}), h^*(w,\vec{c}), \vec{c})$$

where

$$g^*(w,\vec{c}) = g(\langle\beta(1,\beta(1,w)),\beta(2,w)\rangle,\vec{c})$$

and

$$h^*(w,\vec{c}) = h(\langle\beta(2,\beta(1,w)),\beta(2,w)\rangle,\vec{c}).$$

Since f is defined as the composition of $\mathrm{FP}_3^{\Sigma^p_{i-1}}[wit, log^{O(1)}]$ functions, f is itself in $\mathrm{FP}_3^{\Sigma^p_{i-1}}[wit, log^{O(1)}]$. Clearly f can be intensionally defined by S_3^{i-1} and the conditions of Theorem 18 are satisfied.

Case (2): ($\wedge$:right). Suppose the last inference of P is

$$\frac{\Gamma \longrightarrow B, \Delta^* \qquad \Gamma \longrightarrow C, \Delta^*}{\Gamma \longrightarrow B \wedge C, \Delta^*}$$

The argument for this case is similar to that of case (1). Let D be the formula $B \vee (\bigvee \Delta^*)$, let E be $C \vee (\bigvee \Delta^*)$ and let F be $(B \wedge C) \vee (\bigvee \Delta^*)$. The induction hypothesis is that there are $\mathrm{FP}_3^{\Sigma^p_{i-1}}[wit, log^{O(1)}]$ functions g and h so that

$$S_3^{i-1} \vdash \mathit{Witness}^{i,\vec{c}}_{\bigwedge\Gamma}(w,\vec{c}) \rightarrow \mathit{Witness}^{i,\vec{c}}_{D}(g(w,\vec{c}),\vec{c})$$

and

$$S_3^{i-1} \vdash \mathit{Witness}^{i,\vec{c}}_{\bigwedge\Gamma}(w,\vec{c}) \rightarrow \mathit{Witness}^{i,\vec{c}}_{E}(h(w,\vec{c}),\vec{c}).$$

Let k be the function such that

$$k(v,w,\vec{c}) = \begin{cases} v & \text{if } \mathit{Witness}^{i,\vec{c}}_{\bigvee\Delta^*}(v,\vec{c}) \\ w & \text{otherwise.} \end{cases}$$

By Proposition 17, $\mathit{Witness}^{i,\vec{c}}_{\bigvee\Delta^*}$ is a Π^b_{i-1}-predicate; hence k is a $\mathrm{FP}_3^{\Sigma^p_{i-1}}[wit, log^{O(1)}]$ function. Let f be the function

$$f(w,\vec{c}) = \langle\, \langle\beta(1,g(w,\vec{c})),\beta(1,h(w,\vec{c}))\rangle, k(\beta(2,g(w,\vec{c})),\beta(2,h(w,\vec{c})),\vec{c})\rangle.$$

As in case (1), it is clear that f satisfies the desired conditions.

Case (3): ($\exists\le$:left). Suppose the last inference of P is

$$\frac{a \le s, B(a), \Gamma^* \longrightarrow \Delta}{(\exists x \le s)B(x), \Gamma^* \longrightarrow \Delta}$$

The free variable a is the *eigenvariable* and appears only as indicated. Let D be the formula $a \le s \wedge (B(a) \wedge (\bigwedge \Gamma^*))$ and let E be $(\exists x \le s)B(x) \wedge (\bigwedge \Gamma^*)$. By the induction hypothesis, there is a $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ function g such that

$$S_3^{i-1} \vdash Witness_D^{i,\vec{c},a}(w, \vec{c}, a) \rightarrow Witness_{\bigvee\Delta}^{i,\vec{c}}(g(w, \vec{c}, a), \vec{c}).$$

(Note that the variable a can be omitted from the right hand side of the implication since it does not appear free in Δ.)

We wish to define a function $h(w, \vec{c})$ which produces a value for a such that $B(a)$ holds: we can apply the function g to this to get a witness for $\bigvee\Gamma$. To define h, we must consider three subcases: first, if $(\exists x \le s)B$ is in $\Sigma_i^b \setminus \Pi_{i-1}^b$, let $h(w, \vec{c}) = \beta(1, \beta(1, w))$; if w is a witness for E then $h(w, \vec{c})$ is a value for a such that $B(a)$ holds and such that $\beta(2, \beta(1, w))$ is a witness for $B(a)$. Second, if $(\exists x \le s)B \in \Sigma_{i-1}^b \cap \Pi_{i-1}^b$, $h(w, \vec{c})$ is the (possibily multivalued) function which is computed by asking a witness oracle for $(\exists x \le s)B(x)$ for a value for x; i.e., $h(w, \vec{c}) = a$ iff $a \le s$ and $B(a)$ holds. Third, if $(\exists x \le s)B \in \Pi_{i-1}^b \setminus \Sigma_{i-1}^b$, then the quantifier $(\exists x \le s)$ must be sharply bounded with $s = |r|$ for some term r. Define h by $h(w, \vec{c}) = (\mu x \le |r|)B(x, \vec{c})$; by Theorem 7, h is a $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ function definable by S_3^{i-1}.

In each case, we have that

$$S_3^{i-1} \vdash Witness_E^{i,\vec{c}}(w, \vec{c}) \rightarrow B(h(w, \vec{c}), \vec{c}) \wedge h(w, \vec{c}) \le s(\vec{c})$$

and, indeed, that

$$S_3^{i-1} \vdash Witness_E^{i,\vec{c}}(w, \vec{c}) \rightarrow Witness_B^{i,\vec{c},a}(\beta(2, \beta(1, w)), \vec{c}, h(w, \vec{c})).$$

The desired $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ function $f(w, \vec{c})$ is given by

$$f(w, \vec{c}) = g(\langle\!\langle 0, \beta(2, \beta(1, w)), \beta(2, w)\rangle\!\rangle, \vec{c}, h(w, \vec{c}))$$

and it is can easily be checked that all the conditions of Theorem 18 hold.

Case (4): ($\forall \leq$:right). Suppose the last inference of P is:

$$\frac{a \leq s, \Gamma \longrightarrow B(a), \Delta^*}{\Gamma \longrightarrow (\forall x \leq s) B(x), \Delta^*}$$

(As usual this is one of the hardest cases.) The free variable a is the *eigenvariable* and must appear only as indicated. Let D be the formula $a \leq s \wedge (\bigwedge \Gamma)$, let E be $B(a) \vee (\bigvee \Delta^*)$ and let F be $(\forall x \leq s)B(x) \vee (\bigvee \Delta^*)$. The induction hypothesis is that there is a $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ function g such that

$$S_3^{i-1} \vdash Witness_D^{i,\vec{c},a}(w, \vec{c}, a) \rightarrow Witness_E^{i,\vec{c},a}(g(w, \vec{c}, a), \vec{c}, a).$$

First, consider the case where $(\forall x \leq s)B(x)$ is in Π_{i-1}^b. We shall define a function f such that

$$S_3^{i-1} \vdash Witness_D^{i,\vec{c}}(w, \vec{c}) \rightarrow Witness_F^{i,\vec{c}}(f(w, \vec{c}), \vec{c}).$$

by informally describing how to compute f. It will be clear that f is a $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ function since g is. To compute $f(w, \vec{c})$, first ask a Σ_{i-1}^p witness oracle if there exists a value $a \leq s(\vec{c})$ such that $\neg B(a, \vec{c})$ holds. If such an value exists, the oracle returns a value a, and $f(w, \vec{c}) = g(\langle 0, w \rangle, \vec{c}, a)$. If no such value exists, then $f(w, \vec{c}) = \langle 0, 0 \rangle$. In the latter case, $\langle 0, 0 \rangle$ is a witness for $F(\vec{c}, s)$ since 0 is a witness for the true Π_{i-1}^b formula $(\forall x \leq s)B(x)$.

Second, consider the case where $(\forall x \leq s)B(x)$ is in $\Sigma_i^b \setminus \Pi_{i-1}^b$. Then it must be that $(\forall x \leq s)$ is sharply bounded and $s = |r|$ for some term r. Let k be the function defined by

$$k(w, \vec{c}, a) = \begin{cases} \langle 0, \beta(1, g(w, \vec{c}, a)) \rangle & \text{if } Witness_B^{i,\vec{c},a}(\beta(1, g(w, \vec{c}, a)), \vec{c}, a) \\ \langle 1, \beta(2, g(w, \vec{c}, a)) \rangle & \text{otherwise.} \end{cases}$$

To understand the defintion of k; note that the function $g(w, \vec{c}, a)$ provides a witness for E; such a witness is an ordered pair $\langle v_1, v_2 \rangle$ such that either v_1 is a witness for $B(a)$ or v_2 is a witness for $\bigvee \Delta^*$. By definition, $k(w, \vec{c}, a)$ is equal to $\langle 0, v_1 \rangle$ in the former case and to $\langle 1, v_2 \rangle$ otherwise. It is clear that k is a $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ function since g is and by Proposition 17. Now let

$\overline{k}(w,\vec{c},a)$ be the $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ function defined from k as in Theorem 6. And define the function $f(w,\vec{c})$ in terms of $\overline{k}$ by letting

$$f(w,\vec{c}) = \begin{cases} \langle\langle w_0,\ldots,w_{|s|}\rangle,0\rangle & \text{if } a_i = 0 \text{ for all } i \\ \langle 0, w_i\rangle & \text{for the least } i \text{ s.t. } a_i = 1, \text{ otherwise} \end{cases}$$

where $\overline{k}(2s(\vec{c})+1,\vec{x},a) = \langle\langle a_0,w_0\rangle,\ldots,\langle a_{|s|},w_{|s|}\rangle\rangle$. Clearly f is defined by a polynomial function of the value of $\overline{k}$ and, by construction, f satisfies the desired conditions for Theorem 18.

Case (5): (Σ_i^b-LLIND). Suppose the last inference of P is

$$\frac{B(\lfloor\frac{1}{2}a\rfloor),\Gamma^*\longrightarrow B(a),\Delta^*}{B(0),\Gamma^*\longrightarrow B(|t|),\Delta^*}$$

where a is the *eigenvariable* and must not appear in the lower sequent.

Let D be the formula $B(\lfloor\frac{1}{2}a\rfloor)\wedge(\bigwedge\Gamma^*)$, let $E(\vec{c},a)$ be $B(a)\vee(\bigvee\Delta^*)$, let F be $B(0)\wedge(\bigwedge\Gamma^*)$ and let A be $B(|t|)\vee(\bigvee\Delta^*)$. The induction hypothesis is that there is a $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ function g such that

$$S_2^{i-1} \vdash \mathit{Witness}_D^{i+1,\vec{c},a}(w,\vec{c},a) \rightarrow \mathit{Witness}_E^{i+1,\vec{c},a}(g(w,\vec{c},a),\vec{c},a).$$

By Proposition 16 there is a polynomial time computable function g_E which is Σ_1^b-definable in S_3^{i-1} and a term t_E such that S_3^{i-1} can prove that if w is a witness for $E(\vec{c},a)$ then $g_E(w)$ is $\le t_E(\vec{c},a)$ and is also a witness for $E(\vec{c},a)$. Define the function h by

$$\begin{aligned} h(0,w,\vec{c}) &= g_E(\langle\beta(1,w),0\rangle) \\ h(a,w,\vec{c}) &= \begin{cases} h(\lfloor\frac{1}{2}a\rfloor,w,\vec{c}) & \text{if } \mathit{Witness}_E^{i,\vec{c},a}(h(\lfloor\frac{1}{2}a\rfloor,w,\vec{c}),\vec{c},a) \\ g_E(g(\langle\beta(1,h(\lfloor\frac{1}{2}a\rfloor,w,\vec{c})),\beta(2,w)\rangle,\vec{c},a)) & \text{otherwise} \end{cases} \end{aligned}$$

for $a > 0$. Also define $f(w,\vec{c}) = h(|t|,w,\vec{c})$. Because of the use of the function g_E, the values of $h(a,w,\vec{c})$ are bounded by $t_E(\vec{c},a)$ and thus f is defined by limited logarithmic recursion on notation from functions in $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ and by Theorem 5, f is also in $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ and S_2^{i-1} can prove that f can be computed by a canonical Turing machine.

Thus, S_3^{i-1} can prove that, for any computation of the value of $f(w,\vec{c})$, there is a sequence of values

$$h(0,w,\vec{c}),\cdots,h(\lfloor |t|/2^j \rfloor, w,\vec{c}),\cdots,h(|t|,w,\vec{c})$$

obtained during the computation. Now it is easy for S_3^{i-1} to prove that $h(\lfloor |t|/2^j \rfloor, w,\vec{c})$ is a witness for $E(\vec{c},\lfloor |t|/2^j\rfloor)$ by Σ_{i-1}^b-LBIND since the formula $Witness_E^{i,\vec{c},a}$ is Π_{i-1}^b (in fact, Σ_{i-1}^b-LIND is available). Thus

$$S_3^{i-1} \vdash Witness_F^{i,\vec{c}}(w,\vec{c}) \rightarrow Witness_A^{i,\vec{c}}(f(w,\vec{c}),\vec{c}).$$

Q.E.D. Theorem 18

4.3 Some Corollaries

Theorem 19 $(i>1)$ *The Σ_i^b-definable functions of R_3^i and of S_3^{i-1} are precisely the* $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ *functions.*

Proof It is immediate from Theorem 18 that every Σ_i^b-definable function of R_3^i is an $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ function. Since $R_3^i \vdash S_3^{i-1}$, the same is true of every function Σ_i^b-definable function of S_3^{i-1}. Finally, by Theorem 3, every $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ function is Σ_i^b-definable by S_3^{i-1}. □

Theorem 20 *The functions which are strongly Σ_i^b-definable by R_3^i (or, by S_3^{i-1}) are precisely the strong* $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ *functions.*

Proof Corollary 4 gives one direction. Conversely, suppose $R_3^i \vdash (\forall \vec{x})(\exists y)A(\vec{x},y)$ where $A\in\Sigma_i^b$ and let f be the function such that $f(\vec{x})=y$ iff $A(\vec{x},y)$. We need to show that f is in strong $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$. By Theorem 18, there is an explicit $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ Turing machine M such that $S_3^{i-1}\vdash(\forall\vec{x})A(\vec{x},func_M(\vec{x}))$. Without loss of generality, assume $A(\vec{x},y)$ is of the form $(\exists z\le t')B(\vec{x},y,z)$ with $B\in\Pi_{i-1}^b$ and with t' a term in the variables $\vec{x}$ only. Construct a Turing machine M' which runs the following algorithm:

Input: $\vec{x}$

(1) Simulate $M(\vec{x})$ until an output y_0 is obtained.

(2) Ask the witness oracle query "$(\exists y \le t)(y = y)$?";
Then ask the witness oracle query "$(\exists z \le t')(z = z)$?"
Let y_1 and z_1 be the oracle responses.

(3) Ask the Σ^b_{i-1}-query "$\neg B(\vec{x}, y_1, z_1)$?"
If answer is Yes (so B is false), then output y_0 and halt.
Otherwise, answer is No: output y_1 and halt.

Clearly M' is explicitly $\mathrm{FP}_3^{\Sigma^p_{i-1}}[wit, log^{O(1)}]$ since M is. Step (2) of the algorithm consists of asking known-to-be-true queries for the sole purpose of generating nondeterministically values for y_1 and z_1. If these values for y_1 and z_1 happen to witness the truth of $(\exists y \le t)A$, then y_1 is output, otherwise y_0 is output. Having y_0 ensures that at least one value of $f(\vec{x})$ can be found; and nondeterministically guessing y_1 and z_1 makes it possible for any $y = f(\vec{x})$ to be output. $\Box$

Theorem 21 $(i > 1)$ *R^i_3 is $\forall\Sigma^b_i$-conservative over S^{i-1}_3.*

Proof This is also immediate from Theorem 18: if R^i_3 proves a $(\forall\vec{x})A(\vec{x})$ with $A \in \Sigma^b_i$, then R^i_3 proves the sequent $\longrightarrow A(\vec{c})$ and, by Theorem 18, S^{i-1}_3 proves $(\exists w)\, Witness^{i,\vec{c}}_A(w, \vec{c})$. Thus, Proposition 16, S^{i-1}_3 proves $(\forall\vec{x})A(\vec{x})$. $\Box$

The class of formulas which are Boolean combinations of Σ^b_i-formulas is denoted $\mathcal{B}(\Sigma^b_i)$; and $\forall\mathcal{B}(\Sigma^b_i)$ is the set of universal generalizations of Boolean combinations of Σ^b_i-formulas.

Theorem 22 *R^i_3 is $\forall\mathcal{B}(\Sigma^b_i)$-conservative over $S^{i-1}_3 + \Sigma^b_i$-replacement.*

Recall that $R^i_3 \supseteq S^{i-1}_3 + \Sigma^b_i$-replacement; however, we have no indication as to whether these theories are distinct.

Proof It suffices, of course, to show that any Boolean combination of Σ^b_i-formulas (with free variables) provable by R^i_3 is provable in $S^{i-1}_3 + \Sigma^b_i$-replacement. Since any Boolean combination can be written in conjunctive normal form, it suffices to show that any disjunction of Σ^b_i- and Π^b_i-formulas provable in R^i_3 is provable in $S^{i-1}_3 + \Sigma^b_i$-replacement. By rewriting a disjunction as an implication in the form $\bigwedge A_j \to \bigvee B_j$ with each

A_j and B_j in Σ_i^b, it suffices to show that if R_3^i proves a sequent $A \longrightarrow B$ with $A, B \in \Sigma_i^b$ then $S_3^{i-1} + \Sigma_i^b$-replacement proves the sequent too. If the sequent is provable by R_3^i then by Theorem 18, S_3^{i-1} proves

$$(\exists w)\, \mathit{Witness}_A^{i,\vec{c}}(w, \vec{c}) \longrightarrow (\exists w)\, \mathit{Witness}_B^{i,\vec{c}}(c, \vec{c}).$$

By Proposition 16, $S_3^{i-1} + \Sigma_i^b$-replacement proves $(\exists x)\, \mathit{Witness}_A^{i,\vec{c}}(w, \vec{c})$ is equivalent to $A(\vec{c})$ and likewise for B. Thus $S_3^{i-1} + \Sigma_i^b$-replacement proves $A \rightarrow B$. $\Box$

Recall that a formula is Δ_{i+1}^b with respect to a theory if and only if the theory proves that the formula is equivalent to a Σ_{i+1}^b-formula and to a Π_{i+1}^b-formula.

Theorem 23 $(i \geq 1)$ *S_3^i admits Δ_{i+1}^b-PIND.*

Proof For any formula A which is Δ_{i+1}^b w.r.t. S_3^i, the induction axiom for A is a $\forall\Sigma_{i+1}^b$-sentence. Since R_3^{i+1} proves Δ_{i+1}^b-PIND, it follows that S_3^i does too. $\Box$

Actually there is another proof of the previous theorem which also applies to S_2^i; this gives the following stronger result which answers a question from Buss [3].

Theorem 24 $(i \geq 1)$ *S_2^i admits Δ_{i+1}^b-PIND.*

Proof Based on a theorem of Ressayre it was shown in [3] that $S_2^i + \Sigma_{i+1}^b$-replacement is Σ_{i+1}^b-conservative over S_2^i. Since the Δ_{i+1}^b-PIND axioms are equivalent to $\forall\Sigma_{i+1}^b$-formulas, it suffices to show that $S_2^i + \Sigma_{i+1}^b$-replacement proves the Δ_{i+1}^b-PIND axioms. It is easy to see that Σ_{i+1}^b-replacement implies 'comprehension' for Δ_{i+1}^b formulas; i.e., if $A(u)$ is Δ_{i+1}^b then $S_2^i + \Sigma_{i+1}^b$-replacement proves

$$(\exists w)(\forall u \leq |t|)(Bit(u, w) = 1 \leftrightarrow A(u))$$

where t may be any term not involving u, w and $A(u)$ may have other variables besides u as parameters. Now LIND for A follows easily from the existence of w (just do LIND on $Bit(u, w) = 1$). $\Box$

We conclude this section with a couple open questions regarding the theories above results.

First, it would be interesting to know if any Σ_i^b-defined function of R_3^i (equivalently, S_3^{i-1}) can be strenthened to be a provably single-valued function. That is, if R_3^i proves $(\forall x)(\exists y)A(x,y)$ with $A \in \Sigma_i^b$, must there be a formula $A^*(x,y) \in \Sigma_i^b$ such that R_3^i proves $A^*(x,y) \to A(x,y)$ and such that R_3^i proves $(\forall x)(\exists! y)A^*(x,y)$? It would also be nice to know this for S_3^{i-1} as well (this does not seem to automatically follow from the statement for R_3^i). Note that this capability of strenthening a function definition does exist for prior-studied theories such as S_2^i.

Second, it is open whether the conservation results of Theorems 21 and 22 hold for the theories R_2^i and S_2^{i-1} in place of R_3^i and S_3^{i-1}. Likewise, it would be quite interesting to know what the Σ_i^b-definable functions of R_2^i are. It seems that R_3^i is somehow a much more natural theory than R_2^i; as if the growth rate of the $\#_3$ function is somehow naturally linked to the LLIND and LBIND axioms (at least at our present state of knowledge). Note that Krajíček [11] has shown that the Σ_i^b-definable functions of S_2^{i-1} are precisely the functions in $\mathrm{FP}^{\Sigma_i^p}[wit, log]$.

5 The $\Sigma_i^{1,b}$-Definable Functions of Three Second-Order Systems

In this section, the $\Sigma_i^{1,b}$-definable functions of the theories V_2^{i-1}, U_2^i and V_2^i are characterized. We have already shown that every $\mathrm{EXPTIME}^{\Sigma_{i-1}^{1,p}}[wit, poly]$ function is $\Sigma_i^{1,b}$-definable in these theories in the first two theories and that every $\mathrm{EXPTIME}^{\Sigma_i^{1,p}}$ function is $\Sigma_i^{1,b}$-definable in the third theory. To prove the converses, we shall prove 'witnessing theorems' regarding sequents of $\Sigma_i^{1,b}$-formulas provable in these theories.

5.1 The Witness Formula

In [2] a predicate *Witness2* was defined for the purpose of witnessing $\Sigma_1^{1,b}$-formulas; we must generalize this definition to handle witnessing $\Sigma_i^{1,b}$-formulas. For the rest of this section, $i \geq 1$ will be a fixed integer; the applications in this paper only need $i > 1$. Let $A(\vec{a}, \vec{\alpha})$ be a $\Sigma_i^{1,b}$-formula.

A formula $\mathit{Witness2}_A^{i,\vec{a},\vec{\alpha}}(\gamma,\vec{a},\vec{\alpha})$ is defined which has limited quantifier complexity and which states that γ is a second-order object 'witnessing' the truth of $A(\vec{a},\vec{\alpha})$.

Definition Suppose $A(\vec{a},\vec{\alpha}) \in \Sigma_i^{1,b}$ and $\vec{a},\vec{\alpha}$ is a vector of variables including all those free in A. The formula $\mathit{Witness2}_A^{i,\vec{a},\vec{\alpha}}$ is defined below, inductively on the complexity of A:

(1) If $A \in \Pi_{i-1}^{1,b}$ then $\mathit{Witness2}_A^{i,\vec{a},\vec{\alpha}}$ is just A itself.

(2) If A is $B \wedge C$ then define

$$\begin{aligned}&\mathit{Witness2}_A^{i,\vec{a},\vec{\alpha}}(\gamma,\vec{a},\vec{\alpha}) \iff \\ &\qquad \mathit{Witness2}_B^{i,\vec{a},\vec{\alpha}}(\boldsymbol{\beta}(1,\gamma),\vec{a},\vec{\alpha}) \wedge \mathit{Witness2}_C^{i,\vec{a},\vec{\alpha}}(\boldsymbol{\beta}(2,\gamma),\vec{a},\vec{\alpha}).\end{aligned}$$

(3) If A is $B \vee C$ then define

$$\begin{aligned}&\mathit{Witness2}_A^{i,\vec{a},\vec{\alpha}}(\gamma,\vec{a},\vec{\alpha}) \iff \\ &\qquad \mathit{Witness2}_B^{i,\vec{a},\vec{\alpha}}(\boldsymbol{\beta}(1,\gamma),\vec{a},\vec{\alpha}) \vee \mathit{Witness2}_C^{i,\vec{a},\vec{\alpha}}(\boldsymbol{\beta}(2,\gamma),\vec{a},\vec{\alpha}).\end{aligned}$$

(4) If A is $B \to C$ and is not in $\Pi_{i-1}^{1,b}$, then we define

$$\begin{aligned}&\mathit{Witness2}_A^{i,\vec{a},\vec{\alpha}}(\gamma,\vec{a},\vec{\alpha}) \iff \\ &\qquad \mathit{Witness2}_{\neg B}^{i,\vec{a},\vec{\alpha}}(\boldsymbol{\beta}(1,\gamma),\vec{a},\vec{\alpha}) \vee \mathit{Witness2}_C^{i,\vec{a},\vec{\alpha}}(\boldsymbol{\beta}(2,\gamma),\vec{a},\vec{\alpha}).\end{aligned}$$

(5) If $A \notin \Pi_{i-1}^{1,b}$ and $A(\vec{a},\vec{\alpha})$ is $(\forall x \le s)B(x,\vec{a},\vec{\alpha})$ then define

$$\begin{aligned}&\mathit{Witness2}_A^{i,\vec{a},\vec{\alpha}}(\gamma,\vec{a},\vec{\alpha}) \iff \\ &\qquad (\forall x \le s)\, \mathit{Witness2}_{B(b,\vec{a},\vec{\alpha})}^{i,b,\vec{a},\vec{\alpha}}(\boldsymbol{\beta}(x+1,\gamma),x,\vec{a},\vec{\alpha}).\end{aligned}$$

(6) If $A \notin \Pi_{i-1}^{1,b}$ and $A(\vec{a},\vec{\alpha})$ is $(\exists x \le s)B(x,\vec{a},\vec{\alpha})$ then define

$$\mathit{Witness2}_A^{i,\vec{a},\vec{\alpha}}(\gamma,\vec{a},\vec{\alpha}) \iff (\exists x \le s)\, \mathit{Witness2}_{B(b,\vec{a},\vec{\alpha})}^{i,b,\vec{a},\vec{\alpha}}(\gamma,x,\vec{a},\vec{\alpha}).$$

(7) If $A \notin \Pi^{1,b}_{i-1}$ and A is $(\exists\varphi)B(\vec{a},\vec{\alpha},x)$ where φ is a unary second-order predicate, then define

$$\mathit{Witness2}^{i,\vec{a},\vec{\alpha}}_{A}(\gamma,\vec{a},\vec{\alpha}) \Longleftrightarrow \mathit{Witness2}^{i,\vec{a},\vec{\alpha},\varphi}_{B(b,\vec{a},\vec{\alpha})}(\not{\boldsymbol{\beta}}(2,\gamma),\vec{a},\vec{\alpha},\not{\boldsymbol{\beta}}(1,\gamma)).$$

The assumption that φ is unary is sufficient for proving the witnessing lemmas below. For k-ary predicates we would replace $\not{\boldsymbol{\beta}}(1,\gamma)$ with $ARY_k(\not{\boldsymbol{\beta}}(1,\gamma))$ as in [2].

(8) If $A \notin \Pi^{1,b}_{i-1}$ and A is $\neg B$ then use prenex operations to push the negation sign "into" the formula so that it can be handled by cases (1)–(6).

The purpose of defining *Witness2* is to give a canonical way of verifying that $A(\vec{a},\vec{\alpha})$ is true. It is easy to see that $(\exists\gamma)\,\mathit{Witness2}^{i,\vec{a},\vec{\alpha}}_{A}(\gamma,\vec{a},\vec{\alpha})$ is equivalent to $A(\vec{a},\vec{\alpha})$. The next propositions express some properties of *Witness*; these are are analogous to Propositions 15-17 and are proved similarly.

Proposition 25 *For $i \geq 1$, and $A \in \Sigma^{1,b}_{i}$, $\mathit{Witness2}^{i,\vec{a},\vec{\alpha}}_{A}$ is a $\Pi^{1,b}_{i-1}$-formula.*

Proposition 26 $(i \geq 1)$. *Let $A(\vec{a},\vec{\alpha})$ be a $\Sigma^{1,b}_{i}$-formula. Then*

$$V^{i-1}_{2} \vdash \mathit{Witness2}^{i,\vec{a},\vec{\alpha}}_{A}(\gamma,\vec{a}) \rightarrow A(\vec{a})$$

and

$$V^{i-1}_{2} + \Sigma^{1,b}_{i}\text{-replacement} \vdash A(\vec{a},\vec{\alpha}) \leftrightarrow (\exists\gamma)\,\mathit{Witness2}^{i,\vec{a},\vec{\alpha}}_{A}(\gamma,\vec{a},\vec{\alpha}).$$

Recall that $V^{i-1}_{2} + \Sigma^{1,b}_{i}$-replacement is a subtheory of U^{i}_{2} (by Theorem 9.16 of [2]).

Proposition 27 $(i \geq 1)$. *Let A be a $\Sigma^{1,b}_{i}$-formula. The predicate represented by $\mathit{Witness2}^{i,\vec{a},\vec{\alpha}}_{A}$ is a $\Pi^{1,p}_{i-1}$-predicate.*

5.2 Two Witnessing Theorems for Second-Order Theories

In this section we state two witnessing theorems regarding the definability and computability of witness functions for sequents of $\Sigma_i^{1,b}$-formulas. The first, for V_2^i, is the easiest to prove since it does not depend on the use of witness oracles.

Theorem 28 (The Witnessing Lemma for V_2^i)
Fix $i \geq 1$. Suppose the sequent $\Gamma, \Pi \longrightarrow \Delta, \Lambda$ is a theorem of V_2^i and each formula in $\Gamma \cup \Delta$ is $\Sigma_i^{1,b}$ and each formula in $\Pi \cup \Lambda$ is $\Pi_i^{1,b}$. Let $\vec{c}, \vec{\alpha}$ be the free variables in the sequent and let G and H be the formulae

$$G = \left(\bigwedge \Gamma\right) \wedge \bigwedge \{\neg C : C \in \Lambda\}$$

and

$$H = \left(\bigvee \Delta\right) \vee \bigvee \{\neg C : C \in \Pi\}.$$

Then there is a $\mathrm{EXPTIME}^{\Sigma_{i-1}^{1,p}}$ function f which is $\Sigma_i^{1,b}$-defined by V_2^i such that

$$V_2^i \vdash \mathit{Witness2}_G^{i,\vec{c},\vec{\alpha}}(w, \vec{c}, \vec{\alpha}) \rightarrow \mathit{Witness2}_H^{i,\vec{c},\vec{\alpha}}(f(w, \vec{c}, \vec{\alpha}), \vec{c}, \vec{\alpha}).$$

The case $i = 1$ of Theorem 28 is already proved in Buss [2]; the proof of the general case is exactly the same as the proof of the case $i = 1$ except that all exponential time computations are relative to an oracle which is a complete predicate of Σ_{i-1}^p.[3] Another way to think about Theorem 28 is to use the 'RSUV isomorphism' to see that the theorem corresponds to a witnessing theorem for S_3^i — the witnessing theorem for S_3^i is completely idential to the witnessing theorem for S_2^i except that $\#_3$-time is used in place of polynomial time. Because the proof of Theorem 28 is so similar to earlier proofs, we omit it here.

Theorem 29 (The Witnessing Lemma for V_2^{i-1} and U_2^i)
Fix $i > 1$. Suppose the sequent $\Gamma, \Pi \longrightarrow \Delta, \Lambda$ is a theorem of U_2^i and each

[3] Recall that exponentially long queries may be made to the Σ_{i-1}^p-oracle so, effectively, the computation may ask any $\Sigma_i^{1,p}$-query.

formula in $\Gamma \cup \Delta$ *is* $\Sigma_i^{1,b}$ *and each formula in* $\Pi \cup \Lambda$ *is* $\Pi_i^{1,b}$. *Let* $\vec{c}, \vec{\alpha}$ *be the free variables in the sequent and let* G *and* H *be the formulae*

$$G = \left(\bigwedge \Gamma\right) \wedge \bigwedge\{\neg C : C \in \Lambda\}$$

and

$$H = \left(\bigvee \Delta\right) \vee \bigvee\{\neg C : C \in \Pi\}.$$

Then there is a $\mathrm{EXPTIME}^{\Sigma_{i-1}^{1,p}}[wit, poly]$ *function* f *which is* $\Sigma_i^{1,b}$*-defined by* V_2^{i-1} *such that*

$$V_2^{i-1} \vdash \mathit{Witness2}_G^{i,\vec{c},\vec{\alpha}}(w, \vec{c}, \vec{\alpha}) \to \mathit{Witness2}_H^{i,\vec{c},\vec{\alpha}}(f(w, \vec{c}, \vec{\alpha}), \vec{c}, \vec{\alpha}).$$

The proof of Theorem 29 is entirely analogous to the proof of Theorem 18. For the case of $\forall$*:left* inferences, we use the definability of first-order minimization in place of length-bounded minimization; and for the case of $\Sigma_i^{1,b}$-PIND, we use the closure of $\mathrm{EXPTIME}^{\Sigma_{i-1}^{1,p}}[wit, poly]$ under first-order recursion on notation in place of the closure of $\mathrm{FP}_3^{\Sigma_{i-1}^p}[wit, log^{O(1)}]$ under limited, logarithmic recursion on notation. We omit the details. It is also possible to prove Theorem 29 as a corollary of Theorem 18 using the RSUV isomorphism.

5.3 Main Results for Second-Order Theories

The following theorems are all corollaries of the witnessing lemmas; these are proved similarly to the corresponding theorems for first-order theories in section 4.3 above.

Theorem 30 (See [2]) $(i \geq 1)$ *The* $\Sigma_i^{1,b}$*-definable functions of* V_2^i *are precisely the* $\mathrm{EXPTIME}^{\Sigma_{i-1}^{1,p}}$ *functions.*

Theorem 31 $(i > 1)$ *The* $\Sigma_i^{1,b}$*-definable functions of* U_2^i *and of* V_2^{i-1} *are precisely the* $\mathrm{EXPTIME}^{\Sigma_{i-1}^{1,p}}[wit, poly]$ *functions.*

The strongly $\Sigma_i^{1,b}$*-definable functions of* U_2^i *and of* V_2^{i-1} *are precisely the strong* $\mathrm{EXPTIME}^{\Sigma_{i-1}^{1,p}}[wit, poly]$ *functions.*

The $\Sigma_i^{1,b}$*-definable functions of* U_2^i *and of* V_2^{i-1} *which have first-order values are precisely the functions in* $\mathrm{PSPACE}^{\Sigma_{i-1}^{1,p}}$ *(equivalently, in* $\mathrm{EXPTIME}^{\Sigma_{i-1}^{1,p}}[poly]$*).*

Theorem 32 *($i > 1$) U_2^i is $\forall\Sigma_i^{1,b}$-conservative over V_2^{i-1}.*

Theorem 33 *U_2^i is $\forall\mathcal{B}(\Sigma_i^{1,b})$-conservative over $V_2^{i-1} + \Sigma_i^{1,b}$-replacement.*

A formula is $\Delta_{i+1}^{1,b}$ with respect to a theory if and only if the theory proves that the formula is equivalent to a $\Sigma_{i+1}^{1,b}$-formula and to a $\Pi_{i+1}^{1,b}$-formula.

Theorem 34 *($i \geq 1$) V_2^i admits $\Delta_{i+1}^{1,b}$-PIND.*

Theorem 34 follows from Theorem 32 since a $\Delta_{i+1}^{1,b}$-formula is equivalent to a $\forall\Sigma_{i+1}^{1,b}$-formula.

The *bounded $\Delta_{i+1}^{1,b}$ comprehension* axioms are the formulas

$$(\forall a)(\exists\varphi)(\forall x \leq a)\,(\varphi(x) \leftrightarrow A(x))$$

where A is $\Delta_{i+1}^{1,b}$ and may contain other free variables in addition to x.

Theorem 35 *($i \geq 1$) $V_2^i \vdash$ bounded $\Delta_{i+1}^{1,b}$ comprehension.*

Theorem 35 improves a result of Takeuti that V_2^i proves $\Sigma_i^{1,b}$-bounded comprehension [17].

Proof First we show that, for $i \geq 0$, U_2^{i+1} proves bounded $\Delta_{i+1}^{1,b}$ comprehension (this extends Theorem 2.1 of [13]). Let A be $\Delta_{i+1}^{1,b}$ with respect to U_2^{i+1} and let $B(z,a)$ be the formula

$$(\forall x \leq a)(\exists\varphi)(\forall y \leq z)(\varphi(x+y) \leftrightarrow A(x+y));$$

note that B is a $\Sigma_{i+1}^{1,b}$-formula. Now we can use the usual "doubling trick" (see Theorems 2.22 or 9.16 of [2]): it is easy to see that $U_2^{i+1} \vdash B(\lfloor \frac{1}{2}z \rfloor, a) \rightarrow B(z,a)$, from whence, by $\Sigma_{i+1}^{1,b}$-PIND, $U_2^{i+1} \vdash B(a,a)$, which readily implies that U_2^{i+1} proves the bounded $\Delta_{i+1}^{1,b}$-comprehension axiom for $A(x)$.

Now suppose $i \geq 1$ and $A(x)$ is $\Delta_{i+1}^{1,b}$ with respect to V_2^i. The bounded $\Delta_{i+1}^{1,b}$ comprehension axiom for A is a $\forall\Sigma_{i+1}^{1,b}$-formula and is provable in U_2^{i+1}. Hence, by Theorem 32, it is provable also by V_2^i.

References

[1] B. Allen, *Arithmetizing uniform NC*, Annals of Pure and Applied Logic, 53 (1991), pp. 1–50.

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

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

[4] ——, *The witness function method and fragments of Peano arithmetic*, in Logic, Methodology and Philosophy of Science IX, D. Prawitz, B. Skyrms, and D. Westerståhl, eds., Amsterdam, 1994, North-Holland, pp. 29–68.

[5] S. R. Buss and L. Hay, *On truth-table reducibility to SAT and the difference hierarchy over NP*, in Proceedings of the Structure in Complexity Conference, June 1988, pp. 224–233.

[6] ——, *On truth-table reducibility to SAT*, Information and Computation, 91 (1991), pp. 86–102.

[7] P. Clote and G. Takeuti, *Bounded arithmetics for NC, ALOGTIME, L and NL*, Annals of Pure and Applied Logic, 56 (1992), pp. 73–117.

[8] L. A. Hemachandra, *The strong exponential hierarchy collapses*, in Proceedings 19-th Annual ACM Symposium on Theory of Computing, 1987, pp. 110–122. To appear in JCSS.

[9] R. Kaye, *Using Herbrand-type theorems to separate strong fragments of arithmetic*, in Arithmetic, Proof Theory and Computational Complexity, P. Clote and J. K. cek, eds., Oxford, 1993, Clarendon Press (Oxford University Press).

[10] J. Krajíček, *No counter-example interpretation and interactive computation*, in Logic From Computer Science: Proceedings of a Workshop

held November 13-17, 1989, Mathematical Sciences Research Institute Publication #21, Springer-Verlag, 1992, pp. 287–293.

[11] ——, *Fragments of bounded arithmetic and bounded query classes*, Transactions of the A.M.S., 338 (1993), pp. 587–598.

[12] J. Krajíček, P. Pudlák, and G. Takeuti, *Bounded arithmetic and the polynomial hierarchy*, Annals of Pure and Applied Logic, 52 (1991), pp. 143–153.

[13] J. Krajíček and G. Takeuti, *On bounded Σ^1_1 polynomial induction*, in Feasible Mathematics: A Mathematical Sciences Institute Workshop, Ithaca, June 1989, Birkhäuser, 1990, pp. 259–280.

[14] M. W. Krentel, *The complexity of optimization problems*, Journal of Computer and System Sciences, 36 (1988), pp. 490–509.

[15] P. Pudlák, *Some relations between subsystems of arithmetic and the complexity of computations*, in Logic From Computer Science: Proceedings of a Workshop held November 13-17, 1989, Mathematical Sciences Research Institute Publication #21, Springer-Verlag, 1992, pp. 499–519.

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

[17] ——, *S^i_3 and $\mathring{V}^i_2(BD)$*, Archive for Math. Logic, 29 (1990), pp. 149–169.

[18] ——, *RSUV isomorphisms.* Typeset manuscript, 1991.

[19] K. W. Wagner, *Bounded query classes*, SIAM Journal on Computing, 19 (1990), pp. 833–846.