Predicative Recursion and Computational Complexity

Stephen J. Bellantoni

30 September 1992

A thesis submitted in conformity with the requirements of the Degree of Doctor of Philosophy, Graduate Department of Computer Science, in the University of Toronto

Abstract

The purpose of this thesis is to give a "foundational" characterization of some common complexity classes. Such a characterization is distinguished by the fact that no explicit resource bounds are used. For example, we characterize the polynomial time computable functions without making any direct reference to polynomials, time, or even computation. Complexity classes characterized in this way include polynomial time, the functional polytime hierarchy, the logspace decidable problems, and NC.

After developing these "resource free" definitions, we apply them to redeveloping the feasible logical system of Cook and Urquhart, and show how this first-order system relates to the second-order system of Leivant. The connection is an interesting one since the systems were defined independently and have what appear to be very different rules for the principle of induction. Furthermore it is interesting to see, albeit in a very specific context, how to retract a second order *statement*, ("induction holds up to x"), into first order *type* information.

Based on this analysis we give a general discussion of "functional impredicativity", and introduce the idea of attributing impredicativity to certain specific integers, relative to a computation which is being performed or a proof which is being carried out.

Acknowledgements

I feel a deep gratitude towards Stephen Cook for working with me, for his unfailing help and exemplary professionalism. He has given unselfishly and in great measure.

Alasdair Urquhart's unique perspective helped me in many ways. Thanks for the conversation.

Sam Buss certainly deserves applause for the time and effort he spent reading and commenting on the thesis. Thanks for taking an interest in it. Professor Charles Rackoff provided many insightful comments and corrections. Professors Faith Fich, Al Borodin, Hector Levesque, and Bill Weiss have provided indispensable help. Thanks to Professor Dyer for chairing the defense.

I am thankful for the support from the Department of Computer Science and staff, the Connaught Foundation, the University of Toronto, and the Ontario Graduate Scholarship program; without them this work would have been impossible.

Toniann Pitassi has been a great friend and colleague, sharing some of the best times. There are many others – I don't think I could make a complete list. Thanks to each of you. Gara Pruesse deserves special mention for pointing out a problem, and for then being willing to help fix it.

Bruce Kapron, as well as Jim Otto, Stephen Bloch, and others, have my thanks for being there in the community and for welcoming this research.

Anne is discovered again here.

Saving the best for last... Thanks mom & dad!! It is hard to express my deep feeling of love for each of you and gratitude for your tremendous sacrifice and enduring gifts.

Contents

Notation

In this text all scalar values are non-negative integers.

x, y, z, a, b, c: integers, or integer-valued inputs or variables. (x, y, z: normal or tier 1; a, b, c: safe or tier 0).

u, v, w: integer-valued variables (Part II) or tier 2 inputs (Chapter 7).

i, j, k: usually, constants in $\{0, 1\}$, although sometimes also used subscripts for vectors.

$x[i]$ is the ith bit of integer x; this binds looser than () function application, so that $f(x)[i]$ is the ith bit of $f(x)$. However, xi or $[x]i$ is the integer x followed by the bit i; that is, $s_i(x)$.

Q, S: set-valued variables.

s, t: usually, terms.

f, g, h: function symbols or functions. When we want to refer to the inputs by name, we write e.g. $f(\overline{x}; \overline{a})$ for the function, even though strictly speaking this expression has integer type.

$\mathrm{op}(\overline{x})$ for unary op: either $\mathrm{op}(x_1), \ldots, \mathrm{op}(x_n)$ or else $\mathrm{op}(x_1) \wedge \cdots \wedge \mathrm{op}(x_n)$.

$|u|$: binary length $\lceil \log_2(u + 1) \rceil$ of u.

ui: successor function, $2u + i$, for $i \in \{0, 1\}$. Binds tighter than other functions, e.g. $xi \,\underline{\mathrm{mod}}\, yj = (xi) \,\underline{\mathrm{mod}}(yj)$.

pu: predecessor, $\lfloor u/2 \rfloor$. Binds tighter than other function symbols, e.g. $pu \div v = (pu) \div v$. In the first Part of the thesis it is a primitive function $p(; a)$, in the second Part it is $p(x;)$ defined by Predicative Notational Recursion.

$u \div v$: drop, $u/2^{|v|}$.

$a \,\underline{\mathrm{mod}}\, x$, or $\underline{\mathrm{mod}}(x; a)$: length modulus, $a \bmod 2^{|x|}$.

$u \oplus v$: concatenation, $u \cdot 2^{|v|} + v$.

$u \boxplus v$: pad, $u \cdot 2^{|v|}$.

$\#(u;)$: declining smash, $u \oplus pu \oplus ppu \oplus \cdots \oplus 1$. The function $\#(u;;)$ defined in Chapter 7 is different from $\#(u;)$ but has approximately the same length.

$u\#v$: smash, $2^{|u|\cdot|v|}$.

$\textsc{parity}(u)$: parity, $u \bmod 2$.

$\textsc{ones}(u)$: ones, $2^{|u|} - 1$.

$\textsc{omax}(x, y)$: ones' maximum, $\textsc{ones}(\max(x, y))$.

$\bullet(x, y)$: ones' catenation, $\textsc{ones}(x) \oplus \textsc{ones}(y)$.

$\sqcup$: blank space character.

$\mathbf{R}$: recursor, type 2 functional for Predicative Notational Recursion.

$\alpha[\overline{x}; \overline{a}]$, $\alpha[\overline{t}, \overline{Q}]$: formulas with all free variables substituted by the indicated expressions.

$W(u)$: formula of $L_2^{\mathrm{P}}(A)$, asserts u is a (standard) number.

$\mathrm{Cl}(Q)$: formula of $L_2^{\mathrm{P}}(A)$, asserts Q is an inductive set.

In propositional contexts, 0 is used for true and 1 for false.

$\langle \mathcal{I}, \mathcal{A} \rangle$: an $L_2^{\mathrm{P}}(A)$ model consisting of a structure $\mathcal{I}$ and an assignment $\mathcal{A}$.

$\langle \mathcal{I}', \mathcal{A}' \rangle$: a QPV model consisting of a structure $\mathcal{I}$ and an assignment $\mathcal{A}$.

$\mathbf{Q}$, $\mathbf{R}$, $\mathbf{S}$: subsets of a universe of a model.

$\overline{\mathbf{u}}$, $\overline{\mathbf{v}}$, $\overline{\mathbf{w}}$: elements of a universe of a model.

Chapter 1

Introduction

In 1964 Cobham [C] gave the first recursion-theoretic characterization of polynomial time, using a recursion whose depth was the binary length of the number being recursed on, and using explicit polynomial bounds to control the size of the output. This characterization has been used in many contexts, including, for example, recent work in feasible arithmetic [B], [C], [CU]. See [So] or [R] for a discussion of recursion theory. Examining Cobham's result, we can observe that the use of recursion on notation is not in itself enough to restrict the class of defined functions: recursion on notation can simulate primitive recursion in the absence of any size bounds.[1] From this perspective we can fit Cobham's characterization into the pattern of most (if not all) definitions of complexity classes: one defines a general model of computation and then cuts off the computation when certain resource bounds are exceeded.

The purpose of this thesis is to give a more "foundational" characterization of some common complexity classes. Specifically, we would like to define a model of computation whose full natural power is exactly that of, say, polynomial time. Of course, the model cannot simply be one which manifestly imposes resource bounds: the model must have some feature which reflects in a more fundamental way the structure of the complexity class in question.

The thesis is structured in two parts: the first part provides new recursion-theoretic characterizations of several common complexity classes; the second uses this develop-

[1] Define $x - 1$, then use notational recursion on an exponentially large value to simulate primitive recursion.

ment to explore issues in feasible logic. In addition to the general discussion given here, each of the two parts is preceded by an introduction and mention of relevant prior research.

We first develop "resource free" models for functional polynomial time, the functional polytime hierarchy, and the logspace decidable problems. Based on the work presented here, some of which was published as [BC], Bloch [Bl] gave a similar characterization of NC^1; we indicate how to characterize NC based on the work of Clote [Cl]. After developing these "resource free" definitions, we apply them to redeveloping the feasible logical system of [CU], and show how this first-order system relates to the second-order system of Leivant [L]. The connection is an interesting one since the systems were defined independently and have what appear to be very different rules for the principle of induction. Furthermore it is interesting to see, albeit in a very specific context, how to retract the second order *statement*, "induction holds up to x", into first order *type* information. Based on this analysis we give a general discussion of "functional impredicativity", and introduce the idea of attributing impredicativity to certain specific integers, relative to a computation which is being performed.

The existence of a "resource free" model of computation is suggested by the recent work of Leivant [L1], [L2] on impredicativity, complexity, and logic. Stated briefly, his program is to develop feasible second order logics by restricting the comprehension scheme.[2] The discussion of comprehension and impredicativity goes back at least to the time of Russell's paradox; see Feferman [F] for an overview. Later, Nelson [N] took this approach to constructive (although not complexity-bounded) mathematics. The idea now is to push radically beyond constructive mathematics and obtain feasible mathematics. Leivant does this by restricting comprehension to quantifier-free positive formulas. While there is no direct technical connection between the comprehension schemes introduced by Leivant and the "predicative recursion" scheme we use to obtain polynomial time functions, the prototypical example of a proof in Leivant's system gave the insight which led to the predicative recursion scheme.

In making the characterizations, we draw the philosophical conclusion that the impredicativity of the integers $\mathbb{N}$, discussed by Leivant and Nelson as a source of computational explosion, can be attributed directly to specific members of $\mathbb{N}$ during the course of a

[2]Comprehension is, roughly: $\exists R.\forall x.R(x) \leftrightarrow \varphi$, where φ is a formula with free variables including x but not R. In other words, comprehension lets us define sets from logical formulas, possibly with parameters.

computation. In the words of Leivant [L],

> "... a definition of a set X is *impredicative* if it refers to a collection of which X is an element. Uncontrolled impredicativity leads to contradictions, as in Russell's Paradox. However, impredicative definitions abound in Mathematical Analysis, where real numbers (i.e. subsets of $\mathbb{N}$ [...]) are defined in terms of quantification over all reals. We normally expect no contradiction to arise, because we implicitly assume that the power set of $\mathbb{N}$, $\mathcal{P}\mathbb{N}$, is given as a completed totality prior to the definition of any particular members thereof. In predicative systems of analysis one refrains from assuming the power set of $\mathbb{N}$ as given, albeit $\mathbb{N}$ is assumed as a completed totality. [...] An argument raised by Nelson [1986] is that the definitions of $\mathbb{N}$ are also circular: the generative (inductive) definition, as a set constructed by repeated application of the successor function, presupposes an understanding of $\mathbb{N}$ itself (specifically when Induction is proposed as the formal justification of the process).

We can turn the last assertion around to observe that the principle of Induction is "as impredicative as" the definition of $\mathbb{N}$. This principle, admitted in a constructive version of analysis where the basic objects of study are sets of integers, must be restricted or eliminated if we want a highly constructive (i.e. feasible) system whose basic objects of study are the integers themselves.

Now turning to a definition of a function by recursion, say[3] $f(xi) = h_i(x, f(x))$, we notice that we are defining a set $\{y : \exists x. y = f(x)\}$. This definition is impredicative: in order to compute $h_i(x, f(x))$ we need to know that $f(x)$ is defined, and since we do not have at hand a complete definition of the function f, we can only obtain $f(x)$ by appealing to induction, an impredicative concept. Alternatively we can just accept *prima facia* that the recursion equation can only be understood as an impredicative definition. (For a composition $f(x) = h(x, r(x))$ this issue does not arise because we already have a prior definition of r; we already have the set $\{y : \exists x. y = r(x)\}$ available as a completed totality.)

In effect, if we are to stick to predicative principles then the computation $h(x, f(x))$ should not have the ability to access the "impredicative" value $f(x)$ as freely as it

[3] Here and in the rest of the thesis, xi abbreviates $s_i(x)$.

can access concretely presented inputs such as x. Obviously, if we do not allow h to access $f(x)$ at all then we obtain constant time computable functions. If we allow h to access $f(x)$ in the same way as x then we obtain the primitive recursive functions. Striking a middle path, we can allow h to access $f(x)$ but only using certain primitive operations. In this way we will obtain polynomial time. Technically, this can be achieved by the mechanism of having two "sorts" of variables. Note that the "impredicativity" of the value $f(x)$ here is not an absolute property of the number, but is relative to the computation being performed. In other words it is a property contingent on the current computation; we can think of a homunculus in the computation who would have to use an impredicative concept in order to know the value.

Regarding minimization, which is normally used to define the partial recursive functions, we can observe a different use of impredicativity than arises in recursion: the definition of minimization refers directly to the integers $\mathbb{N}$. We discuss a combination of minimization and predicative recursion which leads to the functions $\square_i^p$ of the functional polytime hierarchy.

Returning to the logical setting, the second order systems defined by Leivant [L], [L3], [L4] connect to traditional issues of feasible computation as follows: the totality of a function can be proven in the given logic if and only if the function is in the given complexity class. Specifically, the logic in [L] corresponds to polytime in this way. These systems do not have an axiom of induction but instead use an axiom of comprehension.

The relationship between feasible computation and mathematical logic was approached from a quite different direction by Buss [Bu], Cook [C], Cook and Urquhart [CU], Pitassi [Pi], and Allen [Al], among others. In this line of research, feasible first-order arithmetic systems are obtained by restricting induction to formulas of the form $\exists x \leq t.\alpha[x]$ for quantifier-free α. Proofs in these systems are witnessed by feasible functions.

We prove a connection between these two lines of research, using formulations based on the predicative recursion developed earlier. It is satisfying that the first-order development, which we can describe as above in terms of "functional impredicativity", corresponds in a precise way to the logic which Leivant obtained by following predicative principles directly in second order logic.

In our first order logic (QPV) it is sufficient to allow induction on one of the sorts, since we only define functions by recursion on one of the sorts. A less technical way to think of this is to observe that induction is impredicative and we would like to restrict its application as much as possible. Furthermore we prefer to assert as little as possible about the "impredicative" sort. On the other hand, following the line of Leivant, the axioms of our second order logic ($L_2^{\mathrm{P}}(A)$) do not distinguish between values which are presented concretely and values which require an impredicative concept.

This creates a certain asymmetry between the systems: concerning values which are "not known to be predicative", $L_2^{\mathrm{P}}(A)$ asserts a lot of axioms which do not correspond to theorems in the first order system. The systems further differ in that $L_2^{\mathrm{P}}(A)$ has an explicit second order *statement* $W(x)$ which corresponds to *type* information in QPV, that x is a normal (not impredicative) value. Despite this asymmetry we achieve the interesting goal of showing a direct correspondence between the systems.

Part I

Predicative Recursion

Chapter 2

Introduction to Part I

2.1 Discussion of Part I

In the first part of the thesis we show how to characterize several complexity classes without explicit resource bounds, using "predicative recursion". We give characterizations of the polytime functions, the polytime function hierarchy, linear space functions, and logspace decidable problems. We also comment on a generalization to a higher-type class.

These formulations are remarkable because they do not mention any ordinary computational complexity features or resource measures. The initial functions are simple, being reminiscent of the initial functions used to define large classes like the primitive recursive functions. The closure operations involve simple syntactic considerations which do not have any obvious connection to bounding computational complexity. There is nothing in the definitions which directly suggests polytime or other complexity classes. Yet, the classes generated are traditional complexity classes.

In discussing our characterization of the polytime functions, we may start with the following observation: if we remove the size bound from Cobham's characterization of the polytime functions, we obtain all the primitive recursive functions. How then, can we alter this definition in a way which does not introduce explicit bounds but which still produces polynomial time? Reflecting on the examples which arise naturally in the provability characterization of Leivant, we find a syntactic manipulation which guarantees polynomial time. A similar construction using the unary successor instead of binary successors gives linear space.

Having viewed our polytime class as analogous to the primitive recursive functions, it is natural to ask what happens if we add a form of unbounded minimization. In the ordinary recursive context this produces the partial recursive functions; in our context it produces the polytime function hierarchy.

Turning to traditional computation models, a machine class is defined corresponding to the recursive definition. This machine model is used to prove a correspondence between a new recursion theoretic class and logspace decidable problems. The formulation is of interest because this logspace class is obtained by simply omitting one of the initial functions from the polytime class.

After reading our results on polytime, Bloch [Bl] adapted the NC^1 characterization of Allen [Al] to a similar setting. We review other recursion-theoretic NC characterizations, focussing on the "sequential" ones.

Finally we contribute to the dialog on higher type computational complexity by defining a higher type class which computes a certain functional discussed in the community.

2.2 Background.

The insight for the results proved in this part lies in Leivant's paper [L] giving a second order logic such that a function f is polytime if and only if the convergence of f is provable in the logic. The startling aspect of Leivant's result is that polynomial bounds are not mentioned in the definition of the logic; it is a "foundational" characterization. See Part II for further discussion of the logic.

The results here stand independently of Leivant's logic. A more direct use of previous work is the reliance on Cobham's characterization [Cob] of polynomial time by a class of recursive functions bounded explicitly by a polynomially large value. For further work on recursion theory one may refer to Rose [R] or Tourlakis [T]. or more tangentially Soare [So] or Lerman [Ler]. The recursion on notation used by Cobham was defined as early as 1962 ([Ben], as referred to by Compton and LaFlamme [CF]).

The notion of Predicative Notational Recursion developed here is comparable to Leivant's "tiered recurrence", appearing in [L2], but was developed independently. That paper characterizes the much smaller class of *extended polynomials*. There are close connections between the use of tiers and the use of restricted comprehension in complexity-bounded logics; and both are reminiscent of the stratified definitions of sets which form the basis of ramified analysis (see the overview by Feferman, [F] p. 100).

Based on the presentation of the current work as appearing in [BC], Leivant and Marion ([LM1], [LM2]) have recently extended Leivant's earlier work on tiered recurrence to characterize polytime, NC^1 (see [C5] for a definition), and the Kalmar elementary functions (see [R] for a definition). The characterization of NC^1 also draws on the work of Bloch [Bl]. The main points of these papers are to present a unified lambda-calculus formulation and to characterize the Kalmar Elementary functions. Although it appears on the surface that they eliminate the initial functions p and C, this is accomplished by subsuming them under the general scheme of tier 1 recursion.

Also based on the presentation in [BC], Bloch [Bl] adapted the recursion defined by Allen [Al] to obtain a "resource free" characterization of uniform log-depth circuits. Bloch also gives a characterization of the functions computed in polylog space (equivalently, logtime-uniform polylog depth circuits of size $2^{\log^k n}$).

A category theory translation of the two-sorted technique has given by Otto [O], again based on [BC].

After seeing a presentation of the work in [BC], the result there was re-proved by Handley [Ha], who independently developed a machine characterization not too unlike that in Chapter 6. Handley also independently proved exactly the same characterization of linear space as we give in Chapter 5; his proof uses more details since it is based on the machine characterization. He further gives a loop-program characterization of a predicative class defined using simultaneous recursion. Based on our results in Chapter 6, it seems likely that Handley's loop programs are equivalent in power to our non-simultaneous class BC.

In the subject of program synthesis and automatic theorem proving, Fegaras, Sheard and Stemple have formulated a recursion scheme which distinguishes between values which are being computed or accumulated during a recursion, and values on which the recursion is being performed. The distinction seems analogous to our notion of safe and normal inputs. This work [FeShSt] was performed independently and at approximately the same time as the work presented here. They do not prove an upper bound on the complexity of the functions defined in their paper. Thus it is open whether the functions defined by their "uniform traversal combinators" are different from the primitive recursive functions.

The characterization here must be contrasted with the "finite model theory" approach of Immerman [I], [I2], [I3] and others such as Gurevich [Gu]; and also Compton and LaFlamme [CF]. That line of research differs from the work presented here: in many cases the work in finite model uses relations rather than functions, which circumvents the issue of controlling output size; and in the functional setting ("global functions") the fact that the model size n is fixed determines *a priori* that the output — a tuple of elements of the model — will be at most polynomially large. In some cases (second half of [CF], theorems 4,5 of [Gu]) the size bound is explicit. In a later chapter we discuss [CF], and other papers bearing on parallel complexity such as Clote's [Cl] and Allen's [Al], in more detail.

The work here also differs substantially from the earlier work of Clote and Takeuti [CT] which uses logical sorts to distinguish the size of terms; there the explicit construction 2^t creates a term whose logical sort is one greater than the logical sort of t. For example the Key Lemma in [CT], like other contemporary work, uses an explicit bound on the recursively defined function.

Below we discuss the class $\Box_i^p$, the ith level of the polytime hierarchy; this class appears in Buss [B] and is discussed further in the context of bounded arithmetic by Krajicek, Pudlak, and Takeuti [KPT] and Buss [B3]. For a discussion of the linear space class, $\mathcal{E}^2$ of the Grzegorczyk hierarchy, see Ritchie [Ri]; or Rose [R] p. 116.

For a discussion of higher-type feasible computation classes, see Cook [C2], [C4], Cook and Urquhart [CU], Cook and Kapron [CK], Kapron [K], Kapron and Cook [KC], and Seth [S]. Recently Leivant has informally proposed another definition of feasible higher type computation, namely $\vdash \forall f.(\forall x.W(x) \rightarrow W(f(x))) \rightarrow W(F(f))$ in a suitable second order system.

In an earlier draft of this work we used the functions $D(a;b) \equiv P(b,a)$ and $E(a;b) \equiv P(b,P(b,a))$ as initial functions; Sam Buss pointed out to us [B2] that these two can be eliminated in favor of the single predecessor function, p. See the comments near the end of Part II for a discussion of how D and E are still of interest.

Chapter 3

Polytime

As a motivating example, consider the definition of the $\odot$ function (similar to the smash function of Buss and others) in [L]. The definition uses recursion on binary strings as follows: $\odot(w, sv) = \oplus(w, \odot(w, v))$ for $s \in \{0, 1\}$, where $\oplus$ (concatenation) has been defined by recursion *on its first input, not its second.* Leivant shows that convergence of the $\odot$ function is provable[1] in $L_2(QF^+)$. On the other hand, if we take the natural definition of the exponential function, $2^{sx} = \oplus(2^x, 2^x)$, we see that the recursive term 2^x is substituted in the position which was used for recursively defining $\oplus$. Convergence of the exponential function is not provable in $L_2(QF^+)$.

This motivates the definition below of a class B of functions. In this class, one does not allow recursive terms to be substituted into a position which was used for an earlier definition by recursion.

3.1 The class B

Each input to a function in B will be either a "normal" input or a "safe" input; we write the normal inputs to the left and separate them from the safe inputs using a semicolon, as follows: $f(\overline{x}; \overline{y})$. The terminology "safe" and "normal" owes to the development of the first Part of the text, in which these describe the input *positions*; a "safe" position is one where it is safe to input an unusual (impredicative) value. We use this terminology

[1] See Part II for exact definitions.

even in the second Part where it describes the input and output *values* themselves.

We formulate the result using computation on non-negative integers, but the same proof carries over to computation on binary strings as in Leivant [L], replacing 0 with ϵ. Although we are working over integers, we use binary successor functions defined as follows: $ai = 2a + i$, for $i \in \{0, 1\}$. We write $|x|$ for the binary length $\lceil \log_2(x + 1) \rceil$ of integer x; and the terms "predecessor", "successor" refer to binary notation. If $\overline{x}$ is a vector of n integers we write $|\overline{x}|$ for the vector $|x_1|, \ldots, |x_n|$ and write $\overline{f}(\overline{x})$ for $f_1(\overline{x}), \ldots, f_m(\overline{x})$.

Definition of B

> B is the smallest class of functions containing functions `i`-`v` and closed under `vi`, `vii` below:

`i.` **(Constant)** 0 (a zero-ary function).

`ii.` **(Projection)** $\pi_j^{n,m}(x_1 \ldots x_n; x_{n+1} \ldots x_{n+m}) = x_j$, for $1 \leq j \leq n + m$.

`iii.` **(Successors)** $s_i(; a) = 2a + i = ai$, for $i \in \{0, 1\}$.

`iv.` **(Predecessor)** $p(; 0) = 0, \quad p(; ai) = a$.

`v.` **(Conditional)**

$$C(; a, b, c) = \begin{cases} b & \text{if } a \bmod 2 = 0 \\ c & \text{otherwise} \end{cases}$$

`vi.` **(Predicative Notational Recursion)** Define new function f by, for $i \in \{0, 1\}$,

$$\begin{aligned} f(0, \overline{x}; \overline{a}) &= g(\overline{x}; \overline{a}) \\ f(yi, \overline{x}; \overline{a}) &= h_i(y, \overline{x}; \overline{a}, f(y, \overline{x}; \overline{a})) \quad \text{for } yi \neq 0 \end{aligned}$$

where h_i and g are in B.

`vii.` **(Safe Composition)** Define new function f by

$$f(\overline{x}; \overline{a}) = h(\overline{r}(\overline{x};); \overline{t}(\overline{x}; \overline{a}))$$

where h, $\overline{r}$, and $\overline{t}$ are in B.

The polynomial time functions will be exactly those functions in B which have all normal inputs; *i.e.* no safe inputs.

Functions in B can perform any (polytime) operation on their normal inputs, but can only apply a restricted set of operations to their safe inputs. In particular, the operations allowed on safe inputs do not increase the length by more than an additive constant. Adding a function to B which operates on safe inputs would in general be more powerful than adding the same function on normal inputs, because of the asymmetry of the composition scheme.

To understand Safe Composition, suppose $f(\overline{x};\overline{a})$ is given by a single expression in $\overline{x}$ and $\overline{a}$. Then $f(\overline{x};\overline{a})$ can be defined by Safe Composition (and Projection) from the function symbols occurring in the expression, provided that each sub-expression $g(\overline{s};\overline{t})$ has no a_i appearing in the terms $\overline{s}$.

Observe that in defining a function by recursion, the recursive value $f(z,\overline{x};\overline{a})$ is substituted into a safe position rather than a normal position of h. The safe composition scheme ensures that this recursive value will stay in a safe position and will not be copied into a normal position. Intuitively, this means that the depth of sub-recursions which h_i performs on y or $\overline{x}$ cannot depend on the value being recursively computed. This mechanism seems to have the effect of preventing the uncontrolled impredicativity which Leivant discussed as the cause of a blowup in complexity.

In concrete terms, we can think of safe positions as input positions where it is safe to substitute a large value without greatly increasing the output size of the function. In contrast, the output size may increase polynomially in the size of the normal inputs. See Lemma 4.1 below. Intuition relating the safe and normal sorts to computation time is made precise in a later chapter. In philosophical terms, we can think of safe positions as input positions where it is safe to substitute an “impredicatively defined” value.

3.2 B contains PTIME

To prove that every polytime function is in B, we use the Cobham characterization [Cob] of P.

Definition of Cobham's class, FP

> FP is the least class of functions containing the Constant, Projection, Successor functions, and the smash function $2^{|x|\cdot|y|}$; and closed under ordinary composition $h(\overline{g}(\overline{x}))$ and the following rule of Bounded Recursion on Notation.

Bounded Recursion on Notation

> If h_i, g, and j are in the class then so is f, where
>
> $$\begin{aligned} f(0,\overline{x}) &= g(\overline{x}) \\ f(yi,\overline{x}) &= h_i(y,\overline{x},f(y,\overline{x})) \ \text{ for } yi \neq 0 \end{aligned}$$
>
> $(i \in \{0,1\})$, provided that $f(y,\overline{x}) \leq j(y,\overline{x})$ for all $y,\overline{x}$.

See Rose [R] for a proof that these Cobham functions are exactly the polynomial time functions, or see [T] for the same result formulated over binary strings. In Chapter 4 we will relativize this definition, using initial functions as oracles.

In particular, for each Cobham function f there is a length-bounding monotone polynomial b_f such that $|f(\overline{x})| \leq b_f(\overline{|x|})$.

To prove that B contains every PTIME function f, we first show how to compute the value of $f(\overline{x})$ assuming that we already have a value w which is big enough. Intuitively speaking, w has to have a length at least as great as the maximum depth of recursion used in computing $f(\overline{x})$.

Recursion Simulation Lemma

> Let f be any polytime function. There is a function f' in B and a monotone polynomial p_f such that $f(\overline{a}) = f'(w;\overline{a})$ for all $\overline{a}$ and all w with $|w| \geq p_f(\overline{|a|})$.

Proof. The proof is by induction on the length of the derivation of f as a function in the Cobham class. If f is a constant, projection, or successor function then f' is trivially defined using the Constant, Projection or Successor functions of B. In these cases, let $p_f = 0$.

If f is defined by composition, $f(\overline{x}) = h(\overline{g}(\overline{x}))$, then define f' by $f'(w;\overline{x}) = h'(w;\overline{g}'(w;\overline{x}))$. Since the functions $\overline{g}$ are in the Cobham class, they have length-bounding polynomials

$\overline{b_g}$. Define p_f such that

$$p_f(\overline{|x|}) = p_h(\overline{b_g}(\overline{|x|})) + \sum_j p_{g_j}(\overline{|x|}).$$

The induction hypothesis can be applied to h and $\overline{g}$ using the facts implied by $|w| \geq p_f(\overline{|x|})$. The desired properties of f' and p_f then follow easily.

The next case is when f is $2^{|x|\cdot|y|}$. This function has a definition using recursion on notation with length-bounding polynomials $b_g(|x|,|y|) = |x| + |y|$ and $b_f(|x|,|y|) = |x| \cdot |y| + 1$. Namely: $g(0,y) = y$; $g(xi,y) = g(x,y)0$; $f(0,y) = 1$; and $f(xi,y) = g(y, f(x,y))$ (where $xi \neq 0$). In this case one can apply the same method as for bounded recursion on notation.

The difficult case is when $f(y,\overline{x})$ is constructed by bounded recursion on notation as in the Definition above. Then g', h'_0 and h'_1 in B are given by the induction hypothesis. Of course we cannot define $f'(w;y,\overline{x})$ by recursion on y, since y is not in normal position. Instead we introduce a parameter z, and use recursion on z to simulate recursion on y. The recursion parameter z is initialized to w, and the part of the recursion on $|z|$ from $|w|$ down to $|w| - |y|$ corresponds to recursion on y.

To define f' first define, using Predicative Notational Recursion and Safe Composition, the functions

$$\begin{array}{llll}
 & & \textsc{parity}(;a) & = C(;a,0,1) \\
P(0;b) & = b & Y(z,w;y) & = P(P'(z,w;);y) \\
P(ai;b) & = p(;P(a;b)) & I(z,w;y) & = \textsc{parity}(;Y(z1,w;y)) \\
P'(a,b;) & = P(a;b). & & \\
 & & \vee(0;a) & = \textsc{parity}(;a) \\
 & & \vee(xi;a) & = C(;\vee(x;a),\textsc{parity}(;P(xi;a)),1)
\end{array}$$

These are explained as follows. The function $P(a;b)$ takes a predecessors of b. The function $Y(z,w;y)$ produces an initial segment of y, namely y with $|w| - |z|$ rightmost (low-order) bits deleted. As z varies in length from $|w|$ down to $|w|-|y|$, the output of Y varies from y down to the trivial initial segment, 0. Recursion depths with $|z|$ below $|w|-|y|$ will be irrelevant. The function $I(z,w;y)$ satisfies $Y(zj,w;y) = s_{I(z,w;y)}Y(z,w;y)$, for $zj \neq 0$. At each step of the recursion on z we use the function I to look into y and see which step function h_I should be applied. The function $\vee(x;a)$ computes the logical

OR of the rightmost $|x|$ bits of a. To continue formally, define

$$
\begin{aligned}
\hat{h}(w; i, a, \bar{b}, c) \quad &= C(;\ i, h_0'(w; a, \bar{b}, c), h_1'(w; a, \bar{b}, c)) \\
\hat{f}(0, w; y, \overline{x}) \quad &= 0 \\
\hat{f}(zj, w; y, \overline{x}) \quad &= C(;\ \vee(w; Y(z1, w; y)), \\
&\qquad g'(w; \overline{x}), \\
&\qquad \hat{h}(w; I(z, w; y), Y(z, w; y), \overline{x}, \hat{f}(z, w; y, \overline{x})) \\
f'(w; y, \overline{x}) \quad &= \hat{f}(w, w; y, \overline{x}).
\end{aligned}
$$

Since f is in the Cobham class, there is a monotone polynomial b_f such that $|f(y, \overline{x})| \leq b_f(|y|, \overline{|x|})$. Letting $p_h = p_{h_0} + p_{h_1}$, define p_f such that

$$
p_f(|y|, \overline{|x|}) \quad \equiv p_h(|y|, \overline{|x|}, b_f(|y|, \overline{|x|})) + p_g(\overline{|x|}) + |y| + 1.
$$

Fixing y and $\overline{x}$, let w satisfy $|w| \geq p_f(|y|, \overline{|x|})$. We prove below by induction on $|u|$ that: for $|w| - |y| \leq |u| \leq |w|$, $\hat{f}(u, w; y, \overline{x}) = f(Y(u, w; y), \overline{x})$. Since $Y(w, w; y) = y$, we have $f'(w; y, \overline{x}) = f(y, \overline{x})$ as desired for the main induction of the lemma.

Consider any u with $|w| - |y| \leq |u| \leq |w|$. Since $|w| - |y| \geq 1$ there is z and $j \in \{0, 1\}$ such that $u = zj$. Also note $|zj| = |z1|$ and $Y(z1, w; y) = Y(zj, w; y)$. Since $|w| \geq |Y(z1, w; y)|$, the expression $\vee(w; Y(z1, w; y))$ is 0 if $Y(z1, w; y) = 0$ and is 1 if $Y(z1, w; y) \neq 0$.

First, if $|u| = |zj| = |w| - |y|$ then $Y(z1, w; y) = 0$. Using the definition of $\hat{f}$ and the main induction hypothesis on g, we immediately have $\hat{f}(zj, w; y, \overline{x}) = g'(w; \overline{x}) = f(0, \overline{x}) = f(Y(zj, w; y), \overline{x})$.

Second, if $|zj| > |w| - |y|$ we can assume $\hat{f}(z, w; y, \overline{x}) = f(Y(z, w; y), \overline{x})$. Using monotonicity of p_f and p_{h_i},

$$
\begin{aligned}
|w| \quad &\geq p_f(|y|, \overline{|x|}) \\
&\geq p_{h_i}(|Y(z, w; y)|, \overline{|x|}, b_f(|Y(z, w; y)|, \overline{|x|})) \\
&\geq p_{h_i}(|Y(z, w; y)|, \overline{|x|}, |f(Y(z, w; y), \overline{x})|).
\end{aligned}
$$

This allows us to apply the main induction hypothesis for h_i,

$$
\begin{aligned}
h_i'(w; Y(z, w; y), \overline{x}, \hat{f}(z, w; y, \overline{x})) \quad &= h_i'(w; Y(z, w; y), \overline{x}, f(Y(z, w; y), \overline{x})) \\
&= h_i(Y(z, w; y), \overline{x}, f(Y(z, w; y), \overline{x})).
\end{aligned}
$$

The condition $|w| - |y| < |zj| \leq |w|$ implies $y \neq 0$ and $Y(z1, w; y) \neq 0$, so by the definition of $\hat{f}$ and the fact $Y(zj, w; y) = s_{I(z,w;y)} Y(z, w; y)$ we have

$$\begin{aligned}
\hat{f}(zj, w; y, \overline{x}) &= h'_{I(z,w;y)}(w; Y(z, w; y), \overline{x}, \hat{f}(z, w; y, \overline{x})) \\
&= h_{I(z,w;y)}(Y(z, w; y), \overline{x}, f(Y(z, w; y), \overline{x})) \\
&= f(Y(zj, w; y), \overline{x}),
\end{aligned}$$

as desired. ∎

Theorem: B is as powerful as FP

Let $f(\overline{x})$ be a polytime function. Then $f(\overline{x};)$ is in B.

Proof. Let p_f and f' be obtained using the preceding Lemma. We will construct a function $b(\overline{x};)$ in B such that $|b(\overline{x};)| \geq p_f(\overline{|x|})$. Then setting $f(\overline{x};) = f'(b(\overline{x};); \overline{x})$ finishes the theorem.

First define concatenation of k strings (in reverse order) using one safe position:

$$\begin{aligned}
\oplus^2(0; y) &= y \\
\oplus^2(xi; y) &= s_i(; \oplus^2(x; y)) \text{ for } xi \neq 0 \\
\oplus^k(x_1, \ldots, x_{k-1}; x_k) &= \oplus^2(x_1; \oplus^{k-1}(x_2, \ldots, x_{k-1}; x_k))
\end{aligned}$$

Next define the "declining smash" function using recursion in the safe position of $\oplus^2$ as follows,

$$\begin{aligned}
\#(0;) &= 0 \\
\#(zi;) &= \oplus^2(zi; \#(z;))
\end{aligned}$$

The length of $\#z$ is $\Omega(|z|^2)$.

Let a, c be such that $(\sum_j |x_j|)^a + c \geq p_f(\overline{|x|})$ for all $\overline{x}$. Composing $\#$ onto itself a constant number of times and then concatenating a constant to the end gives a function $b^1(z;)$ such that $|b^1(z;)| \geq (|z|)^a + c$. Using safe composition again gives $\oplus(\overline{x};) = \oplus^k(x_1 \ldots x_{k-1}; x_k)$ and $b(\overline{x};) = b^1(\oplus(\overline{x};);)$ having length at least $p_f(\overline{|x|})$ as desired. ∎

3.3 PTIME contains B

To prove that all functions in B are polytime, we first derive a bound on the length of the computed value. Then it is easy to observe that the Predicative Notational Recursion operator preserves polytime if the output is length-bounded.

Polymax Bounding Lemma for B

> Let f be a function in B. There is a monotone polynomial q_f such that $|f(\overline{x};\overline{y})| \le q_f(\overline{|x|}) + \max_i |y_i|$ for all $\overline{x}$, $\overline{y}$.

Proof. The proof is by induction on the derivation of f in B. If f is a Constant, Projection, Successor, Predecessor, or Conditional function, then $|f(\overline{x};\overline{y})| \le 1+\sum_i |x_i| + \max_i |y_i|$ and therefore we can just take $q_f(\overline{|x|}) = 1 + \sum_i |x_i|$.

If f is defined by Predicative Notational Recursion then by the induction hypothesis we have q_g, q_{h_0} and q_{h_1} bounding g, h_0 and h_1 respectively. Letting $q_h = q_{h_0} + q_{h_1}$, we have

$$\begin{aligned}
|f(0,\overline{x};\overline{y})| &\le q_g(\overline{|x|}) + \max_i |y_i| \\
|f(zi,\overline{x};\overline{y})| &\le q_h(|z|,\overline{|x|}) + \max(\max_i |y_i|, |f(z,\overline{x};\overline{y})|)
\end{aligned}$$

Define q_f such that

$$q_f(|z'|,\overline{|x|}) = |z'| \cdot q_h(|z'|,\overline{|x|}) + q_g(\overline{|x|}).$$

We have trivially $|f(0,\overline{x};\overline{y})| \le q_f(|0|,\overline{|x|}) + \max_i |y_i|$ by the induction hypothesis on g and monotonicity of q_h. Now assuming $|f(z,\overline{x};\overline{y})| \le q_f(|z|,\overline{|x|}) + \max_i |y_i|$ and $|zi| \ne 0$ (hence $|zi| = |z| + 1$), we have

$$\begin{aligned}
|f(zi,\overline{x};\overline{y})| &\le q_h(|z|,\overline{|x|}) + \max(\max_i |y_i|, |f(z,\overline{x};\overline{y})|) \\
&\le q_h(|z|,\overline{|x|}) + \max(\max_i |y_i|, q_f(|z|,\overline{|x|}) + \max_i |y_i|) \\
&\le q_h(|z|,\overline{|x|}) + q_f(|z|,\overline{|x|}) + \max_i |y_i| \\
&\le q_h(|z|,\overline{|x|}) + \Big(|z| \cdot q_h(|z|,\overline{|x|}) + q_g(\overline{|x|})\Big) + \max_i |y_i| \\
&\le |zi| \cdot q_h(|z|,\overline{|x|}) + q_g(\overline{|x|}) + \max_i |y_i| \\
&\le |zi| \cdot q_h(|zi|,\overline{|x|}) + q_g(\overline{|x|}) + \max_i |y_i| \\
&\le q_f(|zi|,\overline{|x|}) + \max_i |y_i|.
\end{aligned}$$

Therefore a simple induction on $|z'|$ shows that $|f(z',\overline{x};\overline{y})| \le q_f(|z'|,\overline{|x|}) + \max_i |y_i|$ for all z', as desired.

Finally, if f is defined by Safe Composition then using the induction hypothesis on h, $\overline{r}$, and $\overline{t}$,

$$\begin{aligned} |f(\overline{x};\overline{y})| &= |h(\overline{r}(\overline{x};);\overline{t}(\overline{x};\overline{y}))| \\ &\leq q_h(\overline{|r(\overline{x};)|}) + \max_i |t_i(\overline{x};\overline{y})| \\ &\leq q_h(\overline{q_r}(\overline{|x|})) + \max_i |t_i(\overline{x};\overline{y})| \\ &\leq q_h(\overline{q_r}(\overline{|x|})) + \max_i \left(q_{t_i}(\overline{|x|}) + \max_j |y_j|\right) \\ &\leq q_h(\overline{q_r}(\overline{|x|})) + \textstyle\sum_i q_{t_i}(\overline{|x|}) + \max_i |y_i|. \end{aligned}$$

Therefore, choosing q_f such that $q_f(\overline{|x|}) \equiv q_h(\overline{q_r}(\overline{|x|})) + \sum_i q_{t_i}(\overline{|x|})$ finishes the result. ∎

Theorem: FP is as powerful as B

Let $f(\overline{x};\overline{y})$ be a function in B. Then $f(\overline{x},\overline{y})$ is polytime.

Proof. The initial functions are all clearly computable in polynomial time. For Safe Composition observe that the composition of two polynomial time functions is a polynomial time function.

With regard to Predicative Notational Recursion, it is well known that a recursion on notation can be executed in polynomial time if the result of the recursion is polynomially length-bounded and the step and base functions are polytime. In our case, the length-bound follows from the preceding lemma.

Alternatively we can observe that the bounding polynomials of the Lemma are computable in Cobham's class and that therefore every instance of Predicative Notational Recursion is an occurrence of Bounded Recursion on Notation; this gives a reduction of derivations in B to derivations in the Cobham class. ∎

3.4 Discussion

3.4.1 Philosophical comments

Philosophically we think of normal values as those which are known in totality, and safe values as those which are "impredicative" in the sense that their definition refers to members of the set N of integers other than those which are immediately known. As Nelson [N] points out and Leivant [L] emphasizes, definitions of N are impredicative because they are justified by inductive means, which presuppose an understanding of N. In other words, the validity of defining a function by recursion requires an impredicatively defined concept, namely N. The current work projects the idea of an impredicatively defined set down onto specific members of the set — relative to the fact that we have certain values given to us in their totality, references to other members of N are references to "impredicative values". They are values which you only know exist because you assume the existence of an impredicatively defined set, N. We control this impredicativity by isolating such values in safe input positions, and performing only operations on them which are constant-time with respect to their size.[2] Thus we have developed a functional analogue of highly predicative reasoning.

From this viewpoint, it is interesting to investigate the complexity classes obtained with predicative recursion by allowing various amounts of information to cross over from the safe inputs to the normal inputs; or, by allowing various amounts of computation to be performed on the safe inputs. This project could be stated technically as follows: explain, for various syntactically defined and highly restricted function classes R, the effect of adding to B the composition scheme $f(\overline{x};\overline{y}) = h(\overline{r}(\overline{x};\overline{y});\overline{t}(\overline{x};\overline{y}))$ where $r \in R$. As mentioned above, the function $r(;y) = |y|$ is permissible without exceeding polytime.

3.4.2 Bounds on running time in terms of derivation length

One additional observation to make as regards the characterization B is that if we implement the functions of B on a stack machine in the obvious way (with the low-order bits at the top of the input stacks and charging unit cost for the initial functions), then the running time can be bounded in terms of the length of the derivation of the function in B. Note that the running time is constant with respect to the size of the safe inputs. For example, if $f(y,\overline{x};\overline{a})$ is defined by predicative recursion on y from g and h, then the running time of f is $O(yT_h(y,\overline{x},\overline{a}) + T_g(\overline{x},\overline{a}))$ where T_g is a monotone bound on the running time of g and T_h is a monotone bound on the running times of h_0 and h_1. If f is defined by composition then the running time is the composition of the running times of the composed functions. It follows that the running time of a function with a derivation of size k is bounded by $n^{2^{O(k)}}$.

Another way to think of the relationship between derivations and computation time is

[2]Here we are thinking of a model of computation in which a value is represented as a linked list of bits with a pointer for accessing the low-order bit. Copying values only requires copying the pointer, a constant time operation. See the following chapters for a Turing Machine model.

to observe that with k occurrences of Predicative Recursion on Notation, one obtains the functions computable in time $O(n^k)$.

For a more detailed analysis of how the functions in B correspond to machine computations, see chapter 4.

3.4.3 Robustness of the idea.

The notion of obtaining computationally bounded classes is robust under changes to the details of the recursion scheme and initial functions. As evidence of this we may of course point to the several characterizations given in this thesis, and in [Bl] and [O].

Additional evidence of robustness is obtained by verifying that the technique works for other recursion schemes which have already been explored in the subrecursion literature. As an example, the following recursion scheme can be used while still remaining within polynomial time.

$$\begin{aligned} f(0,\overline{x};\overline{y}) &= g(\overline{x};\overline{y}) \\ f(zi,\overline{x};\overline{y}) &= h^i(z,\overline{x};\overline{y},f(z,\overline{x};\overline{j}^i(z,\overline{x};\overline{y}))) \end{aligned}$$

Here, the functions $\overline{j}^i$ must be of the same number and type as the inputs $\overline{y}$. Effectively, this allows the application of a Safe Composition in safe position, *during* a recursion. For example, by using p for j, we can directly simulate simultaneous recursion on several variables.

This scheme does not violate the output size bound proved above. The functions $\overline{j}^i$ are being composed just zi times; each time the size goes up by at most a polynomial, $\sum q_j$, which we can tack onto the bounding polynomial q_f. For example, in reproving the Polymax Bounding Lemma for a function defined by recursion, we can define

$$q_f(|z|,\overline{|x|}) = |z| * \left(q_h(|z|,|\overline{x}|) + \sum q_{j^i}(|z|,|\overline{x}|)\right) + q_g(\overline{|x|}).$$

The polymax bound then follows easily, along the same lines as before.

Chapter 4

The Polytime Function Hierarchy

In this chapter we characterize the polytime function hierarchy by adding unbounded minimization over safe inputs. This is analogous to the use of unbounded minimization to define the partial recursive functions from the primitive recursive functions. See Soare [Soare's book on degrees] for a discussion of recursion.

The Polychecking Lemma for B, which starts our discussion below, also follows from the machine characterization discussed in the next chapter. However, we prove it here for both B and C in terms of the algebraic characterizations. In any case we need the algebraic characterization in order to discuss the polytime hierarchy, as we do not have a direct machine characterization of the class which uses predicative minimization.

Definition of C

The class C is defined in the same way as B, but adding the following closure rule:

`viii` **(Predicative Minimization)** Define new function f by

$$f(\overline{x};\overline{a}) = \begin{cases} s_1(\mu b.\ h(\overline{x};\overline{a},b) \bmod 2 = 0) & \text{if there is such } b \\ 0 & \text{otherwise ,} \end{cases}$$

where h is already in the class.

Where B is defined by analogy with the primitive recursive functions, the class C is defined by analogy with the partial recursive functions. Therefore it may seem surprising that we obtain the value 0 when there is no b such that $h(\overline{x};\overline{a},b)=0$: in the ordinary recursive setting the search for b is a non-halting computation. The obtaining of output 0 in the safe context is justified by the lemma below, which forms the basis of our discussion.

4.1 A closer examination of B and C

4.1.1 Polychecking Lemma for B and C

We would like to make more precise the intuition that a function in C only examines a polynomial number of rightmost bits of its safe inputs. It seems evident that the output of a function in C consists of a polynomial number of "new" rightmost bits plus all the leftmost bits of some one of the safe inputs. Indeed this follows from the machine characterization given in a later chapter. However, this stronger property is more complicated to state than the following, which is sufficient for our purposes.

For the discussion in this section we need the following function: $a \,\underline{\mathrm{mod}}\, x \equiv a \bmod 2^{|x|}$. This is computable in B with a safe and x normal, using the helper function $N(x,y;a)$ which computes the $|y|$ high-order (i.e. leftmost) bits of $a \,\underline{\mathrm{mod}}\, x$:

$$
\begin{aligned}
b \div 0 &= b \\
b \div zk &= p(;b \div z) \quad \text{for } zk \neq 0 \\
N(x,0;a) &= 0 \\
N(x,yj;a) &= C(;a \div (x \div yj), N(x,y;a)0, N(x,y;a)1) \quad \text{for } yj \neq 0 \\
a \,\underline{\mathrm{mod}}\, x &= N(x,x;a)
\end{aligned}
$$

There are only one or two places in the discussion where we need the fact that $\underline{\mathrm{mod}}$ is computed in B; usually it is used as an ordinary mathematical function rather than as a member of the subject classes B and C. These functions are discussed further in Part II.

The discussion opens with the following basic definition.

Definition of polynomial checking function

Function $f(\overline{x},\overline{a})$ is a *polynomial checking function on* $\overline{x}$ if there is a monotone polynomial $q(|\overline{x}|)$ such that for all $\overline{x}$, $\overline{a}$, w and v satisfying $|v| \geq q(|\overline{x}|) + |w|$,

$$f(\overline{x}; \overline{a}) \,\underline{\mathrm{mod}}\, w = f(\overline{x}; \overline{a} \,\underline{\mathrm{mod}}\, v) \,\underline{\mathrm{mod}}\, w.$$

Any such polynomial q is a *threshold* for f.

It is immediate from this definition that $f(\overline{x}; \overline{a} \,\underline{\mathrm{mod}}\, v) \,\underline{\mathrm{mod}}\, w = f(\overline{x}; \overline{a} \,\underline{\mathrm{mod}}\, v') \,\underline{\mathrm{mod}}\, w$ whenever $|v|, |v'| \geq q(|\overline{x}|) + |w|$.

The important fact about polynomial checking functions, from the point of view of the μ operator, is the following:

Bounded Minimization Fact

If h is a polynomial checking function on $\overline{x}$ with threshold q, then for any $\overline{x}$ and $\overline{a}$,

$$(\exists b.\ h(\overline{x}; \overline{a}, b) \bmod 2 = 0) \rightarrow (\exists b.\ (|b| \leq q(|\overline{x}|) + 2) \wedge (h(\overline{x}; \overline{a}, b) \bmod 2 = 0)).$$

Proof. Assuming the antecedent and letting $|v| = q(|\overline{x}|) + 2$ and $b' = b \,\underline{\mathrm{mod}}\, v$, we have

$$\begin{aligned} 0 &= h(\overline{x}; \overline{a}, b) \bmod 2 \\ &= h(\overline{x}; \overline{a} \,\underline{\mathrm{mod}}\, v, b \,\underline{\mathrm{mod}}\, v) \bmod 2 \\ &= h(\overline{x}; \overline{a} \,\underline{\mathrm{mod}}\, v, b' \,\underline{\mathrm{mod}}\, v) \bmod 2 \\ &= h(\overline{x}; \overline{a}, b') \bmod 2 \end{aligned}$$

∎

Recall from section 3.2 that FP denotes the class of functions with derivations as defined by Cobham [C].

Definition of FP^n

FP^n is the class of functions in FP which are polynomial checking functions on their first n inputs. Note $FP = \cup_n FP^n$. If $\overline{x}$ is a vector of n variables, then write $FP^{\overline{x}}$ for FP^n.

We will be working with relativized versions of the classes, defined as follows.

Definition of $FP(\Phi)$, $B(\Phi)$, and $C(\Phi)$

For any set of functions Φ, the classes $FP(\Phi)$ and $FP^n(\Phi)$ are those defined as were FP and FP^n (using the Cobham formulation) but with Φ added as initial functions. Similarly, if Φ is a set of functions with an assignment of safe or normal to the input positions, then $B(\Phi)$ and $C(\Phi)$ are the classes of functions defined as were B and C, but adding Φ as initial functions.

Cobham's original theorem is formulated for the non-relativized class FP, but it can be extended to the relativized class $FP(\Phi)$: a function is computable in $FP(\Phi)$ if and only if it is computable by a polynomial time Turing machine with queries to oracles in Φ.[1]

Now we can prove the main lemma regarding C. Here is a definition of the size bound that will be proved for C.

Definition of Polymax Bounded

A function $f(\overline{x};\overline{a})$ is *polymax bounded by q on $\overline{x}$* if q is a monotone polynomial such that $|f(\overline{x};\overline{a})| \leq q(|\overline{x}|)+\max_i |a_i|$ for all $\overline{x}$ and $\overline{a}$. The function is *polymax bounded* if it is polymax bounded by some monotone polynomial q.

Polychecking Lemma for B and C

Let Φ be any set of polymax-bounded polynomial checking functions. If $f(\overline{x};\overline{a})$ is in $C(\Phi)$, then f is a polymax-bounded polynomial checking function on $\overline{x}$.

Proof. The proof is by induction on the definition of f. We can assume w.l.o.g. that the polynomial of the size bound and the polynomial of the checking threshold are identical, by summing the two together.

For the initial functions zero, projection, predecessor, C, and successors, the polynomial is at most 1 and the statement of the lemma is easily verified. Of course it is trivial for the initial functions Φ.

The proof that the length bound holds in the induction cases is the same as in the Bounding Lemma for B, except in the case of Predicative Minimization discussed below.

[1] We can take the proofs in Chapter 6 as a guideline of a proof for Cobham's theorem. In one direction, simply add oracle calls in the Turing Machine in correspondence with occurrences of the initial functions of Φ in the derivation. In the other direction, when computing the contents of a work tape at time $t+1$, we can check whether the state entered at time t is a oracle query state for that tape; if so, then we can use the value of the corresponding initial function rather than computing the value from the tape contents at time t.

If f is defined by Safe Composition from h, $\overline{r}$, and $\overline{t}$, then define $q_f(|\overline{x}|) = q_h(q_{\overline{r}}(|\overline{x}|)) + \sum_t q_t(|\overline{x}|)$. One has $q_f(|\overline{x}|) \geq q_h(|\overline{r}(\overline{x};)|) + \sum_t q_t(|\overline{x}|)$ because of the length bound $q_{\overline{r}}(|\overline{x}|) \geq |\overline{r}(\overline{x};)|$ obtained by the induction hypothesis (and using monotonicity of the polynomial q_h). Fixing any $\overline{x}$ and $\overline{a}$, let w and v satisfy $|v| \geq q_f(|\overline{x}|) + |w|$. Let v' satisfy $|v'| = q_h(|\overline{r}(\overline{x};)|) + |w|$; then $|v| \geq q_t(|\overline{x}|) + |v'|$ for all $t \in \overline{t}$. Using the induction hypothesis on h and $\overline{t}$,

$$
\begin{aligned}
f(\overline{x};\overline{a}) \,\underline{\text{mod}}\, w &= h(\overline{r}(\overline{x};);\overline{t}(\overline{x};\overline{a})) \,\underline{\text{mod}}\, w \\
&= h(\overline{r}(\overline{x};);\overline{t}(\overline{x};\overline{a}) \,\underline{\text{mod}}\, v') \,\underline{\text{mod}}\, w \\
&= h(\overline{r}(\overline{x};);\overline{t}(\overline{x};\overline{a} \,\underline{\text{mod}}\, v) \,\underline{\text{mod}}\, v') \,\underline{\text{mod}}\, w \\
&= h(\overline{r}(\overline{x};);\overline{t}(\overline{x};\overline{a} \,\underline{\text{mod}}\, v)) \,\underline{\text{mod}}\, w \\
&= f(\overline{x};\overline{a} \,\underline{\text{mod}}\, v) \,\underline{\text{mod}}\, w.
\end{aligned}
$$

If f is defined by Predicative Notational Recursion, let $q_h = q_{h_0} + q_{h_1}$ and define $q_f(|y|, |\overline{x}|) = |y| \cdot q_h(|y|, |\overline{x}|) + q_g(|\overline{x}|)$. Now we show by induction on y that for each fixed y, the function $\lambda\overline{x}, \overline{a}.\ f(y, \overline{x}; \overline{a})$ is a polynomial checking function with threshold $\lambda|\overline{x}|.\ q_f(|y|, |\overline{x}|)$. This implies that f is a polynomial checking function with threshold q_f.

- By the main induction hypothesis on g,

$$
\begin{aligned}
f(0,\overline{x};\overline{a}) \,\underline{\text{mod}}\, w &= g(\overline{x};\overline{a}) \,\underline{\text{mod}}\, w \\
&= g(\overline{x};\overline{a} \,\underline{\text{mod}}\, v) \,\underline{\text{mod}}\, w \\
&= f(0,\overline{x};\overline{a} \,\underline{\text{mod}}\, v) \,\underline{\text{mod}}\, w
\end{aligned}
$$

  for all $\overline{x}$ and $\overline{a}$ and v, w with $|v| \geq q_f(0, |\overline{x}|) + |w|$.

- When $yi \neq 0$, assume that $\lambda\overline{x}, \overline{a}.\ f(y, \overline{x}; \overline{a})$ is a polynomial checking function. Fix any $\overline{x}$, $\overline{a}$ and v, w with $|v| \geq q_f(|yi|, |\overline{x}|) + |w|$. Let v' satisfy $|v'| = q_h(|y|, |\overline{x}|) + |w|$; then (using monotonicity of the polynomials)

$$
\begin{aligned}
|v| &\geq q_f(|yi|, |\overline{x}|) + |w| \\
&\geq q_h(|yi|, |\overline{x}|) + |y| \cdot q_h(|yi|, |\overline{x}|) + q_g(|\overline{x}|) + |w| \\
&\geq q_h(|y|, |\overline{x}|) + q_f(|y|, |\overline{x}|) + |w| \\
&\geq q_f(|y|, |\overline{x}|) + |v'|.
\end{aligned}
$$

By the main induction hypothesis on h,

$$\begin{aligned}
f(yi,\overline{x};\overline{a})\,\underline{\mathrm{mod}}\,w &= h_i(y,\overline{x};\overline{a},f(y,\overline{x};\overline{a}))\,\underline{\mathrm{mod}}\,w\\
&= h_i(y,\overline{x};\overline{a}\,\underline{\mathrm{mod}}\,v',f(y,\overline{x};\overline{a})\,\underline{\mathrm{mod}}\,v')\,\underline{\mathrm{mod}}\,w\\
&= h_i(y,\overline{x};\overline{a}\,\underline{\mathrm{mod}}\,v',f(y,\overline{x};\overline{a}\,\underline{\mathrm{mod}}\,v)\,\underline{\mathrm{mod}}\,v')\,\underline{\mathrm{mod}}\,w\\
&= h_i(y,\overline{x};(\overline{a}\,\underline{\mathrm{mod}}\,v)\,\underline{\mathrm{mod}}\,v',f(y,\overline{x};\overline{a}\,\underline{\mathrm{mod}}\,v)\,\underline{\mathrm{mod}}\,v')\,\underline{\mathrm{mod}}\,w\\
&= h_i(y,\overline{x};\overline{a}\,\underline{\mathrm{mod}}\,v,f(y,\overline{x};\overline{a}\,\underline{\mathrm{mod}}\,v))\,\underline{\mathrm{mod}}\,w\\
&= f(yi,\overline{x};\overline{a}\,\underline{\mathrm{mod}}\,v)\,\underline{\mathrm{mod}}\,w.
\end{aligned}$$

This finishes the proof in the case of Predicative Recursion.

The final case is when f is defined by Predicative Minimization from h, i.e.

$$f(\overline{x};\overline{a}) = \begin{cases} s_1(\mu b.\ h(\overline{x};\overline{a},b) \bmod 2 = 0) & \text{if there is such a } b\\ 0 & \text{otherwise .}\end{cases}$$

From the Bounded Minimization Fact and the induction hypothesis on h we have

$$|(\mu b.\ h(\overline{x};\overline{a},b) \bmod 2 = 0)| \le q_h(|\overline{x}|) + 2$$

if there is any such b. Thus, setting $q_f(|\overline{x}|) = q_h(|\overline{x}|) + 2$ is sufficient to obtain the polymax size bound. Next, fix any $\overline{x}$, $\overline{a}$ and w, v with $|v| \ge q_f(|\overline{x}|) + |w|$. We have $|v| \ge q_h(|\overline{x}|) + |2|$, and applying the induction hypothesis on h we obtain for all b such that $|b| \le q_h(|\overline{x}|) + 2 \le |v|$,

$$h(\overline{x};\overline{a},b) \bmod 2\ = h(\overline{x};\overline{a}\,\underline{\mathrm{mod}}\,v, b\,\underline{\mathrm{mod}}\,v) \bmod 2\ = h(\overline{x};\overline{a}\,\underline{\mathrm{mod}}\,v, b) \bmod 2.$$

By the Bounded Minimization Fact it follows that a least b satisfying $h(\overline{x};\overline{a},b) \bmod 2 = 0$ is identically a least b satisfying $h(\overline{x};\overline{a}\,\underline{\mathrm{mod}}\,v,b) \bmod 2 = 0$. Therefore $f(\overline{x};\overline{a})\,\underline{\mathrm{mod}}\,w = f(\overline{x};\overline{a}\,\underline{\mathrm{mod}}\,v)\,\underline{\mathrm{mod}}\,w$. ∎

4.1.2 Correspondence of B and FP relative to oracles

The next theorem summarizes the important facts about B. First we write a definition.

Definition of β^*

For function $\beta(\overline{a})$, define $\beta^*(v;\overline{a}) = \beta(\overline{a}\,\underline{\mathrm{mod}}\,v)$. For set of functions Φ, define $\Phi^* = \{\beta^* : \beta \in \Phi\}$.

Fact. Suppose β has an output size bounded by a polynomial in the sizes of its inputs. Then β^* is a polymax bounded polynomial checking function on its normal input, as required for the Lemma above. Furthermore, as in the Bounding Lemma for B, there is a monotone polynomial p_β such that $\beta(\overline{a}) = \beta^*(v;\overline{a})$ whenever $|v| \geq p_\beta(|\overline{a}|)$.

The point of the following theorem is to generalize our earlier result about the relationship between B and FP. We need to consider computations relative to an oracle in order to discuss the polytime function hierarchy. While the characterization given here is imperfect because it only discusses the low-order bits of the output, it is still strong enough for most purposes.

Theorem: correspondence between $B(\Phi^*)$ and $FP(\Phi)$

> Let Φ be a set of functions whose output sizes are bounded by polynomials in the sizes of their inputs, and let q be any monotone polynomial. Then for any f
>
> $$\left(f(\overline{x},\overline{a}) \bmod 2^{q(|\overline{x}|)}\right) \in FP^{\overline{x}}(\Phi) \Leftrightarrow \left(f(\overline{x};\overline{a}) \bmod 2^{q(|\overline{x}|)}\right) \in B(\Phi^*).$$

Proof. Define $f_q(\overline{x};\overline{a}) = f(\overline{x};\overline{a}) \bmod 2^{q(|\overline{x}|)}$.

Suppose that $f_q(\overline{x},\overline{a}) \in FP^{\overline{x}}(\Phi)$. By adding trivial base cases for Φ, one extends the Recursion Simulation Lemma to the classes $FP(\Phi)$ and $B(\Phi^*)$, thereby showing that there is $f'_q(w;\overline{x},\overline{a}) \in B(\Phi^*)$ and monotone polynomial p_f such that $f_q(\overline{x},\overline{a}) = f'_q(w;\overline{x},\overline{a})$ for all $\overline{x}$, $\overline{a}$ and w satisfying $|w| \geq p_f(|\overline{x}|,|\overline{a}|)$. Since $f_q(\overline{x},\overline{a})$ is a polynomial checking function on $\overline{x}$,

$$f_q(\overline{x},\overline{a}) = f_q(\overline{x},\overline{a}) \bmod 2^{q(|\overline{x}|)} = f_q(\overline{x},\overline{a} \bmod 2^{q'(|\overline{x}|)}) \bmod 2^{q(|\overline{x}|)} = f_q(\overline{x},\overline{a} \bmod 2^{q'(|\overline{x}|)})$$

for some monotone polynomial $q'(|\overline{x}|)$. Therefore $f_q(\overline{x},\overline{a}) = f'_q(w;\overline{x},\overline{a} \bmod 2^{q'(|\overline{x}|)})$ whenever $|w| \geq p_f(|\overline{x}|,|\overline{a} \bmod 2^{q'(|\overline{x}|)}|)$. We can write this condition on w as $|w| \geq p'_f(|\overline{x}|)$ for an appropriate monotone polynomial p'_f. Computing $P_f \in B$ such that $|P_f(\overline{x};)| \geq p'_f(|\overline{x}|)$ and computing Q' such that $|Q'(\overline{x};)| \geq q'(|\overline{x}|)$ (using the declining smash function # as usual), we use Safe Composition to compute $f_q(\overline{x};\overline{a}) = f'(P_f(\overline{x};);\overline{x},\overline{a} \,\underline{\bmod}\, Q'(\overline{x};))$ in $B(\Phi^*)$.

In the other direction, suppose $f_q(\overline{x};\overline{a}) \in B(\Phi^*)$. By the Polychecking Lemma for B, we have that $f_q(\overline{x};\overline{a})$ is a polynomial checking function of $\overline{x}$; so if it is in $FP(\Phi)$ then it is in $FP^{\overline{x}}(\Phi)$. As in section 3.3 above, we observe that the initial functions of

B are in polytime; that Safe Composition keeps us in polytime; and that the length-bound required in order to simulate Predicative Notational Recursion with Cobham's Bounded Recursion on Notation is implied by the Polychecking Lemma above. The only additional fact needed is the trivial observation that functions in Φ^* can be computed in polynomial time from functions in Φ. Then the derivation of f_q in $B(\Phi^*)$ can be transformed into a Cobham-style derivation of f_q in $FP(\Phi)$. ∎

As was mentioned, this characterization of $B(\beta^*)$ is somewhat incomplete since it only discusses the outputs modulo a polynomial. This is sufficient for most purposes. A more complete characterization would require a sharper definition of FP^n since as it stands we have $\textsc{ones}(;a)$ in FP^0 but not in B. A sharper definition would refer to the computation time with respect to the first n inputs.

Corollary: $FP(\Phi)$ = normal part of $B(\Phi^*)$

Let Φ be a set of 0/1-valued functions. Then

$$f(\overline{x}) \in FP(\Phi) \Leftrightarrow f(\overline{x};) \in B(\Phi^*).$$

Proof. This is implied by the preceding theorem using two easy facts: that if $f(\overline{x}) \in FP(\Phi)$ then $|f(\overline{x})| \leq q(|\overline{x}|)$ for some polynomial q; and similarly if $f(\overline{x};) \in B(\Phi^*)$ then by the Polychecking Lemma for C, $|f(\overline{x};)| \leq q(|\overline{x}|)$ for some polynomial q. ∎

4.2 C characterizes P^{PH}

4.2.1 The polytime function hierarchy

Definition of the Polytime Function Hierarchy

The polynomial-time hierarchy is $\text{PH} = \cup_i \Sigma_i^p$, where Σ_i^p is the class of problems expressible as a polytime predicate preceded by i alternating quantifiers starting with $\exists$, each quantifier ranging over integers bounded by a polytime function of the free variables. Similarly Π_i^p is the class of problems expressible using i bounded quantifiers starting with $\forall$, over a polytime function. We use 1 for TRUE and 0 for FALSE and write these predicates as total 0/1 valued functions of their free variables. Next define

$$\begin{aligned} \Box_{i+1}^p &= FP(\Sigma_i^p) \\ \Box^p &= \cup_i \Box_i^p \end{aligned}$$

We may observe that $\Box^p_{i+1} = FP(\Sigma^p_i) = FP(\Pi^p_i) = FP(\Sigma^p_i \cup \Pi^p_i)$ since we can interchange $\exists$ and $\forall$ by applying the function $1 - x$ which computes a negation; at the base level, the negation of a polytime function is still polytime. Thus $\Box^p_{i+1}$ consists of the functions computable in polytime with an oracle expressed as i bounded quantifiers with a polytime predicate at the base. See Buss [B], p. 14+ for a discussion and alternate definition.

The effect of the preceding Theorem is to carry over the polytime function hierarchy $\Box^p$ to a hierarchy of functions with safe and normal inputs. We would like to show this hierarchy is exactly C; then the normal part of C will characterize $\Box^p$.

The following technical lemmas will be useful.

Meyer/Stockmeyer/Wrathall Lemma

> Let $PB\exists FP(\Phi)$ be the predicates obtained by poly-bounded $\exists$ quantification over 0/1 valued functions in $FP(\Phi)$, and analogously define $PB\forall FP(\Phi)$. Then $PB\exists FP(\Pi^p_k) = \Sigma^p_{k+1}$ and $PB\forall FP(\Sigma^p_k) = \Pi^p_{k+1}$.

Proof. A proof of this appears as Theorem 7 in section 1.5 of Buss's thesis ([B]. p. 17). ∎

Corollary about $\Sigma^p_k \cup \Pi^p_k$ and $\Sigma^p_{k+1} \cap \Pi^p_{k+1}$

> If g is a 0/1-valued function in $FP(\Sigma^p_k \cup \Pi^p_k)$, then g is a decision problem in $\Sigma^p_{k+1} \cap \Pi^p_{k+1}$.

Proof. Let g be a 0/1-valued function in $FP(\Sigma^p_k \cup \Pi^p_k)$, equivalently in $FP(\Sigma^p_k)$. Then g is trivially in $PB\forall FP(\Sigma^p_k)$; by the preceding Lemma this is Π^p_{k+1}. Similarly, since g is in $FP(\Pi^p_k)$ it is in $PB\exists FP(\Pi^p_k)$ and therefore in Σ^p_{k+1}. ∎

We will prove below that i unbounded minimizations over safe position are equivalent to oracles of level i. The point of this is to eliminate the bounds from the characterization, giving an apparently resource-free algebraic characterization of the polytime function hierarchy. Thus it is of interest to first state clearly the bounded version of the characterization. To our knowledge the following simple characterization of the polytime hierarchy has not appeared in the literature. First we need some formalities about derivation dags.

A derivation dag is a directed acyclic graph whose leaves (in-degree 0 nodes) are labeled with initial functions and whose interior (in-degree ¿ 0) nodes are labeled with derivation rules. Of course the fan-in of an interior node has to match the number of subfunctions used in the derivation; predicative composition and predicative recursion each have fan-in 2, predicative minimization has fan-in 1, and bounded minimization has fan-in 2. By "induction on the derivation" we mean induction on the height (maximum path length) of the derivation dag.

Definition: number of applications of a derivation rule

> Given a derivation dag for a function f, the number of applications of a given rule is the maximum number of nodes labeled with the rule on any leaf-to-root path in the derivation dag.

Derivation Rule: Bounded Minimization

> Given h and t, define f by
>
> $$f(\overline{x}) = \begin{cases} s_1(\mu y \leq t(\overline{x}).\ h(\overline{x}, y) \bmod 2 = 0) & \text{if there is such } y \\ 0 & \text{otherwise ,} \end{cases}$$

Within polytime computations, this rule is equivalent to one which omits the "s_1", since if the result were zero we could still test "$h(\overline{x}, 0) \bmod 2 = 0$" in polynomial time.

Lemma: Characterizing PH using bounded minimization

> Let μFP_i be the Cobham class augmented by allowing i applications of Bounded Minimization. Then $f \in \square^p_{i+1} \Leftrightarrow f \in \mu FP_i$.

Proof. By induction on i, the base case $i = 0$ being Cobham's theorem.

In the $\Rightarrow$ direction, the proof is by a minor induction on the derivation of f as a member of $\square^p_{i+1} \equiv FP(\Sigma^p_i)$. When f is defined by any rule other than the initial functions Σ^p_i then (using the minor induction hypothesis as appropriate to obtain any required subfunctions such as h and g in the composition $h(g(x))$), f is defined by the same rule in μFP_i. The remaining case is when f is a Σ^p_i oracle ($i > 0$), say $f(\overline{x}) \equiv \exists u \leq t(\overline{x}).\ \psi(\overline{x}, u)$ where ψ is in Σ^p_{i-1}. In this case we use the major induction hypothesis to obtain that ψ is in μFP_{i-1}. Then f is computed in μFP_i by testing the low-order bit of the output from Bounded Minimization over ψ.

In the "$\Leftarrow$" direction, suppose $f \in \mu FP_{i+1}$ is defined by Bounded Minimization from h. Since h is in μFP_i, by the induction hypothesis h is in $\square^p_{i+1}$, i.e. computable from level i oracles. By the Lemma above, $h(\overline{x}, y) \bmod 2$ is a decision problem in $\Sigma^p_{i+1} \cap \Pi^p_{i+1}$. Therefore the problem $E(z, \overline{x}) \equiv \exists y \leq z.\ (h(\overline{x}, y) \bmod 2 = 0)$ is in Σ^p_{i+1} too. Now we compute $f(\overline{x})$ using the oracle E as follows: compute $E(t(\overline{x}), \overline{x})$ to see whether $f(\overline{x}) = 0$; if $f(\overline{x}) \neq 0$ then use binary search in the range 0 to $t(\overline{x})$ to find the least z such that $E(z, \overline{x})$, and return $y1$. ∎

While we do not use the statement of this lemma directly, we use a similar proof idea below, in the more complicated two-sorted setting.

4.2.2 Characterizing it with C_i

Definition: C_i

> C_i is the set of functions derivable in C using at most i applications of Predicative Minimization.

The main point of this section is to prove the following result. In combination with the Theorem on the correspondence of $B(\Phi^*)$ and $FP(\Phi)$, (choose $\Phi = \Sigma^p_i \cup \Pi^p_i$) the next Theorem gives a characterization of the polytime function hierarchy.

Theorem: $B(\Sigma^{p*}_i \cup \Pi^{p*}_i)$ **equals** C_i

$$f(\overline{x}; \overline{a}) \in B(\Sigma^{p*}_i \cup \Pi^{p*}_i) \Leftrightarrow f(\overline{x}; \overline{a}) \in C_i$$

Proof. The proof is by induction on i. For $i = 0$ we have $C_0 = B = B(\Sigma^{p*}_0 \cup \Pi^{p*}_0)$ since a Σ^{p*}_0 or Π^{p*}_0 oracle, $\sigma^*(v; \overline{a})$, is just a polytime test on $\overline{a} \,\underline{\mathrm{mod}}\, v$ and hence is in B. Suppose the statement holds for i.

For the "$\Rightarrow$" direction, we proceed by induction on the derivation of f in $B(\Sigma^{p*}_{i+1} \cup \Pi^{p*}_{i+1})$; all the cases are trivial except for computing Σ^{p*}_{i+1} (and Π^{p*}_{i+1}) predicates in C_{i+1}.

Consider any Σ^{p*}_{i+1} predicate

$$\begin{aligned} \sigma^*(v; \overline{a}) &\equiv \exists b \leq t(\overline{a} \,\underline{\mathrm{mod}}\, v).\ \psi(\overline{a} \,\underline{\mathrm{mod}}\, v, b) \\ &\Leftrightarrow \exists b \leq t(\overline{a} \,\underline{\mathrm{mod}}\, v).\ \psi(\overline{a} \,\underline{\mathrm{mod}}\, v, b \,\underline{\mathrm{mod}}\, t(\overline{a} \,\underline{\mathrm{mod}}\, v)) \end{aligned}$$

where $\psi \in \Pi^p_i$ is 0/1-valued and t is polytime.

The function

$$\psi^\star(v,\overline{a},b) \equiv \psi(\overline{a}\,\underline{\mathrm{mod}}\,v, b\,\underline{\mathrm{mod}}\,t(\overline{a}\,\underline{\mathrm{mod}}\,v))$$

is a polynomial checking function of its first input. To see this, use as a threshold any monotone $q(|v|)$ which bounds $\max_{\overline{a}}\{|v|, |t(\overline{a}\,\underline{\mathrm{mod}}\,v)|\}$. There is such a q because there is a polynomial in $|v|$ which bounds the computation time for $t(\overline{a}\,\underline{\mathrm{mod}}\,v)$ independent of $\overline{a}$. Whenever $|V| \geq q(|v|)$ we have

$$\begin{aligned}
\overline{a}\,\underline{\mathrm{mod}}\,v \quad &= (a\,\underline{\mathrm{mod}}\,V)\,\underline{\mathrm{mod}}\,v \\
b\,\underline{\mathrm{mod}}\,t(\overline{a}\,\underline{\mathrm{mod}}\,v) \quad &= (b\,\underline{\mathrm{mod}}\,V)\,\underline{\mathrm{mod}}\,t((a\,\underline{\mathrm{mod}}\,V)\,\underline{\mathrm{mod}}\,v).
\end{aligned}$$

Noting that ψ is 0/1 valued, we have $\psi^\star(v,\overline{a},b)\,\underline{\mathrm{mod}}\,w = \psi^\star(v,\overline{a}\,\underline{\mathrm{mod}}\,V, b\,\underline{\mathrm{mod}}\,V)\,\underline{\mathrm{mod}}\,w$ for all nontrivial w and $|V| \geq q(|v|)$. This proves $\psi^\star(v,\overline{a},b) \in FP^1(\Pi_i^p)$. By the Theorem above on the correspondence of $B(\Phi^*)$ and $FP(\Phi)$, we have $\psi^\star(v;\overline{a},b) \in B(\Pi_i^{p*})$. Using the induction hypothesis, $\psi^\star(v;\overline{a},b) \in C_i$.

We can compute $\neg(;d) = C(;d,1,0)$ and $d \vee e = C(;d,C(;e,0,1),1)$ in C_0, where d and e are 0/1-valued. Also, the function $\leq_t^q(v,\overline{a},b) \equiv (b \bmod 2^{q(|v|)} \leq t(\overline{a}\,\underline{\mathrm{mod}}\,v))$, where q is a monotone polynomial and t is polytime, is in FP^1 and hence is in C_0. Now we can compute

$$\sigma^*(v;a) = \exists b.\ b \leq t(\overline{a}\,\underline{\mathrm{mod}}\,v) \wedge \psi^\star(v;\overline{a},b)$$

with one more minimization, as follows. First let q be the threshold for $\psi^\star$ and replace the condition "$b \leq t(\overline{a}\,\underline{\mathrm{mod}}\,v)$" by "$b \bmod 2^{q(|v|)+2} \leq t(\overline{a}\,\underline{\mathrm{mod}}\,v)$". In the context of the definition of σ^*, this is an equivalent test because of the Bounded Minimization Fact applied to $\psi^\star$, and this test is computable in B (i.e. C_0). Then use the parity of the value obtained by Predicative Minimization to test the existential condition.[2]

To compute a Π_{i+1}^{p*} predicate we use a similar construction except introducing negations around the $\exists$.

In the "$\Leftarrow$" direction, suppose $f(\overline{x};\overline{a}) \in C_{i+1}$. We proceed by an induction on the derivation of f in C, all the cases being trivial except when f is defined by Predicative Minimization from some $h \in C_i$. By the main induction hypothesis we have $h \in B(\Sigma_i^{p*} \cup \Pi_i^{p*})$; and using the Theorem above on the correspondence of $B(\Phi^*)$ and $FP(\Phi)$, this implies $h \in FP(\Sigma_i^p \cup \Pi_i^p)$.

By the Polychecking Lemma for C, h is a polynomial checking function; so by the

[2]The fact that $\psi^\star$ is 0/1-valued is needed to compute the $\wedge$.

Bounded Minimization Fact, $|f(\overline{x};\overline{a})| \leq q(|\overline{x}|)$ for some monotone polynomial q. Therefore $f(\overline{x};\overline{a}) = f(\overline{x};\overline{a}) \bmod 2^{q(|\overline{x}|)}$ and by the Theorem above it suffices to show $f(\overline{x},\overline{a}) \in FP^{\overline{x}}(\Sigma^p_{i+1} \cup \Pi^p_{i+1})$ in order to conclude $f(\overline{x};\overline{a}) \in B(\Sigma^{p*}_{i+1} \cup \Pi^{p*}_{i+1})$. That f is a polynomial checking function on $\overline{x}$ follows from the Polychecking Lemma for C, since $f \in C$. Therefore, all that remains is to show $f(\overline{x},\overline{a}) \in FP(\Sigma^p_{i+1} \cup \Pi^p_{i+1})$. We have at our disposal the fact "$h(\overline{x},\overline{a},b) \bmod 2 \in FP(\Sigma^p_i \cup \Pi^p_i)$," since $h \in FP(\Sigma^p_i \cup \Pi^p_i)$ as mentioned in the previous paragraph.

By the Lemma above, $h(\overline{x},\overline{a},b) \bmod 2$ is a decision problem in $\Sigma^p_{i+1} \cap \Pi^p_{i+1}$. Therefore the problem $E(y,\overline{x},\overline{a}) \equiv \exists b \leq y.\ (h(\overline{x},\overline{a},b) \bmod 2 = 0)$ is in Σ^p_{i+1}. By the Bounded Minimization Fact applied to $h \in C_i$, we can compute $f(\overline{x},\overline{a})$ in $FP(\Sigma^p_{i+1} \cup \Pi^p_{i+1})$ as follows: compute $E(2^{q(|x|)},\overline{x},\overline{a})$ to see whether $f(\overline{x},\overline{a}) = 0$; if $f(\overline{x},\overline{a}) \neq 0$ then use binary search in the range 0 to $2^{q(|\overline{x}|)}$ to find the least y such that $E(y,\overline{x},\overline{a})$, and return $y1$. ∎

Corollary: normal part of C is $\square^p$.

$$f(\overline{x};) \in C_i \Leftrightarrow f(\overline{x}) \in \square^p_{i+1}.$$

Proof. This is immediate from the two Theorems above, using the existence of polynomial length bounds on $f(\overline{x})$. ∎

Chapter 5

Linear Space

5.1 Introduction

Here we modify the FP characterization to give a characterization of linear space, taking as a guideline Ritchie's linear space characterization. The proofs are quite analogous to the proofs for polytime, replacing the inequalities $|w| \geq p(|\overline{x}|)$ and $|f(\overline{x};\overline{a})| \leq p(|\overline{x}|) + \max_i |a_i|$ with $w \geq p(\overline{x})$ and $f(\overline{x};\overline{a}) \leq p(\overline{x}) + \max_i a_i$.

Definition of A

> A is the smallest class of functions containing functions `i`-`vi` below and closed under `vii`, `viii`:

`i`. **(Constant)** 0 (a zero-ary function).

`ii`. **(Projection)**

$$\pi_j^{n,m}(x_1 \dots x_n; a_1 \dots a_m) = \begin{cases} x_j & \text{if } j \leq n \\ a_{j-n} & \text{if } n < j \leq n+m \end{cases}$$

`iii`. **(Successor)** $s(;a) = a + 1$.

`v`. **(Predecessor)** $p(;0) = 0, \quad p(;a+1) = a$.

vi. **(Conditional)**[1]

$$\mathcal{C}(;a,b,c) = \begin{cases} b & \text{if } a = 0 \\ c & \text{otherwise} \end{cases}$$

vii. **(Predicative Primitive Recursion)** Define new function f by, for $i \in \{0,1\}$,

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

where h and g are in A.

viii. **(Safe Composition)** Define new function f by $f(\overline{x};\overline{a}) = h(\overline{r}(\overline{x};);\overline{t}(\overline{x};\overline{a}))$ where h, $\overline{r}$, and $\overline{t}$ are in A.

The linear space functions will be exactly those functions in A which have all normal inputs; *i.e.* no safe inputs.

To prove that every linear space function is in A, we use the characterization of linear space as the least class of functions containing the Zero, Projection, Successor and Multiplication functions, and closed under ordinary composition $h(\overline{g}(\overline{x}))$ and the following rule:

Bounded Primitive Recursion

If h, g, and j are in the class then so is f, where

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

$(i \in \{0,1\})$, provided that $f(y,\overline{x}) \leq j(y,\overline{x})$ for all $y,\overline{x}$.

This class is $\mathcal{E}^2$ of the Grzegorczyk hierarchy and was proved to be equal to linear space by Ritchie [Ri]. See Rose [R], p. 116.

In particular, for each $\mathcal{E}^2$ function f there is a bounding monotone polynomial b_f such that $f(\overline{x}) \leq b_f(\overline{x})$.

[1] Notice this differs from the binary case.

5.2 A contains linear space

Recursion Simulation Lemma for A

> Let f be any linear space function. There is a function f' in A and a monotone polynomial p_f such that $f(\overline{a}) = f'(w;\overline{a})$ for all $\overline{a}$ and all w with $w \geq p_f(\overline{a})$.

Proof. The proof is by induction on the length of the derivation of f as a function in $\mathcal{E}^2$. If f is the zero, projection, or successor function then f' is trivially defined using the Constant, Projection or Successor functions of A. In these cases, let $p_f = 0$.

If f is defined by composition, $f(\overline{a}) = h(\overline{g}(\overline{a}))$, then define f' by $f'(w;\overline{a}) = h'(w;\overline{g}'(w;\overline{a}))$. Since the functions $\overline{g}$ are in $\mathcal{E}^2$, they have bounding monotone polynomials $\overline{b_g}$. Define

$$p_f(\overline{a}) = p_h(\overline{b_g}(\overline{a})) + \sum_j p_{g_j}(\overline{a}).$$

The induction hypothesis can be applied to h and $\overline{g}$ using the facts implied by $w \geq p_f(\overline{a})$. The desired properties of f' and p_f follow easily.

The next case is when f is multiplication, $a \cdot b$. This function has a definition using primitive recursion with bounding polynomials $b_g(a,c) = a + c$ and $b_f(a,c) = a \cdot c$. Namely: $g(0,c) = y$; $g(a+1,c) = g(a,c)+1$; $f(0,c) = 0$; and $f(a+1,c) = g(c, f(a,c))$. In this case one can apply the same method as for bounded primitive recursion.

The difficult case is when $f(c,\overline{a})$ is constructed by bounded primitive recursion on c, as in the Definition above. Then g' and h' in A are given by the induction hypothesis. Of course we cannot define $f'(w;c,\overline{a})$ by recursion on c, since c is not in normal position. Instead we introduce a parameter z, and use recursion on z to simulate recursion on c. The recursion parameter z is initialized to w, and the part of the recursion on z from w down to $w - c$ corresponds to recursion on c.

To define f' first define, using Predicative Primitive Recursion and Safe Composition, the functions

$$\begin{aligned}
P(0;b) &= b \\
P(x+1;b) &= p(;P(x;b)) \\
P'(x,b;) &= P(x;b) \\
&= P(\pi^2_1(x,b;);\pi^2_2(x,b;)).
\end{aligned}
\qquad
Y(z,w;c) = P(P'(z,w;);c)$$

These are explained as follows. The function $P(x;b)$ computes $b-x$ by taking x predecessors of b. The function $Y(z,w;c)$ produces $c-(w-z)$. As z varies in from w down to $w-c$, the output of Y varies from c down to 0. Recursion depths with z less than $w-c$ will be irrelevant.

To continue formally, define f' in A by

$$\begin{aligned}
\hat{f}(0,w;c,\overline{a}) &= 0\\
\hat{f}(z+1,w;c,\overline{a}) &= \mathcal{C}(;\ Y(z+1,w;c), g'(w;\overline{a}), h'(w;Y(z,w;c),\overline{a},\hat{f}(z,w;c,\overline{a})))\\
f'(w;c,\overline{a}) &= \hat{f}(w,w;c,\overline{a}).
\end{aligned}$$

Since f is in $\mathcal{E}^2$, there is a monotone polynomial b_f such that $f(c,\overline{a}) \le b_f(c,\overline{a})$. Define

$$p_f(c,\overline{a}) \quad \equiv p_h(c,\overline{a},b_f(c,\overline{a})) + p_g(\overline{a}) + c + 1.$$

Fixing c and $\overline{a}$, let w satisfy $w \ge p_f(c,\overline{a})$. We prove below by induction on u that: for $w-c \le u \le w$, $\hat{f}(u,w;c,\overline{a}) = f(Y(u,w;c),\overline{a})$. Since $Y(w,w;c) = c$, we have $f'(w;c,\overline{a}) = f(c,\overline{a})$ as desired for the main induction of the lemma.

Consider any u with $w-c \le u \le w$. Since $w-c \ge 1$ there is z such that $u = z+1$.

First, if $u = z+1 = w-c$ then $Y(z+1,w;c) = 0$. Using the definition of $\hat{f}$ and the main induction hypothesis on g, we immediately have $\hat{f}(z+1,w;c,\overline{a}) = g'(w;\overline{a}) = f(0,\overline{a}) = f(Y(z+1,w;c),\overline{a})$.

Second, if $u = z+1 > w-c$ we can assume $\hat{f}(z,w;c,\overline{a}) = f(Y(z,w;c),\overline{a})$. Using monotonicity of p_f and p_h with the fact $Y(z,w;c) \le c$,

$$\begin{aligned}
w &\ge p_f(c,\overline{a})\\
&\ge p_h(Y(z,w;c),\overline{a},b_f(Y(z,w;c),\overline{a}))\\
&\ge p_h(Y(z,w;c),\overline{a},f(Y(z,w;c),\overline{a})).
\end{aligned}$$

This allows us to apply the main induction hypothesis for h,

$$\begin{aligned}
h'(w;Y(z,w;c),\overline{a},\hat{f}(z,w;c,\overline{a})) &= h'(w;Y(z,w;c),\overline{a},f(Y(z,w;c),\overline{a}))\\
&= h(Y(z,w;c),\overline{a},f(Y(z,w;c),\overline{a})).
\end{aligned}$$

The condition $w-c < z+1 \le w$ implies $c \ne 0$ and $Y(z+1,w;c) \ne 0$, so by the definition of $\hat{f}$ we have

$$\begin{aligned}\hat{f}(z+1,w;c,\overline{a}) &= h'(w;Y(z,w;c),\overline{a},\hat{f}(z,w;c,\overline{a}))\\ &= h(Y(z,w;c),\overline{a},f(Y(z,w;c),\overline{a}))\\ &= f(Y(z+1,w;c),\overline{a}),\end{aligned}$$

as desired. ∎

Theorem: A as strong as $\mathcal{E}^2$

> Let $f(\overline{x})$ be a linear space function. Then $f(\overline{x};)$ is in A.

Proof. Using the successor function '+1' we can easily define addition in A with one safe input. From this we easily obtain multiplication with two normal inputs, as in $x\cdot(y+1) = x + (x\cdot y)$. Using composition we can then construct arbitrary polynomials to form the result $f(\overline{x};) = f'(p_f(\overline{x});\overline{x})$ where f' is defined using the preceding Lemma. ∎

5.3 Linear space contains A

To prove that all functions in A are linear space, we first derive a polynomial bound on the computed value.

Bounding Lemma for A

> Let f be a function in A. There is a monotone polynomial q_f such that $f(\overline{x};\overline{a}) \leq q_f(\overline{x}) + \max_i a_i$ for all $\overline{x}$, $\overline{a}$.

Proof. The proof is by induction on the derivation of f in A. If f is a Constant, Projection, Successor, Predecessor, or Conditional function, then $f(\overline{x};\overline{a}) \leq 1 + \sum_i x_i + \max_i a_i$ and therefore we can just take $q_f(\overline{x}) = 1 + \sum_i x_i$.

If f is defined by Predicative Primitive Recursion then by the induction hypothesis we have q_g and q_h bounding g and h. Thus

$$\begin{aligned}f(0,\overline{x};\overline{a}) &\leq q_g(\overline{x}) + \max_i a_i\\ f(z+1,\overline{x};\overline{a}) &\leq q_h(z,\overline{x}) + \max(\max_i a_i, f(z,\overline{x};\overline{a}))\end{aligned}$$

Define

$$q_f(z', \overline{x}) = z' \cdot q_h(z', \overline{x}) + q_g(\overline{x}).$$

We have trivially $f(0, \overline{x}; \overline{a}) \leq q_f(0, \overline{x}) + \max_i a_i$ by the induction hypothesis on g and monotonicity of q_h. Now assuming $f(z, \overline{x}; \overline{a}) \leq q_f(z, \overline{x}) + \max_i a_i$, we have

$$\begin{aligned} f(z+1, \overline{x}; \overline{a}) &\leq q_h(z, \overline{x}) + \max(\max_i a_i, f(z, \overline{x}; \overline{a})) \\ &\leq q_h(z, \overline{x}) + q_f(z, \overline{x}) + \max_i a_i \\ &\leq q_h(z, \overline{x}) + (z \cdot q_h(z, \overline{x}) + q_g(\overline{x})) + \max_i a_i \\ &\leq (z+1) \cdot q_h(z, \overline{x}) + q_g(\overline{x}) + \max_i a_i \\ &\leq (z+1) \cdot q_h(z+1, \overline{x}) + q_g(\overline{x}) + \max_i a_i \\ &\leq q_f(z+1, \overline{x}) + \max_i a_i. \end{aligned}$$

Therefore a simple induction on z' shows that $f(z', \overline{x}; \overline{a}) \leq q_f(z', \overline{x}) + \max_i a_i$ for all z', as desired.

Finally, if f is defined by Safe Composition then using the induction hypothesis on h, $\overline{r}$, and $\overline{t}$, then

$$\begin{aligned} f(\overline{x}; \overline{a}) &= h(\overline{r}(\overline{x};); \overline{t}(\overline{x}; \overline{a})) \\ &\leq q_h(\overline{r}(\overline{x};)) + \max_i t_i(\overline{x}; \overline{a}) \\ &\leq q_h(\overline{q_r}(\overline{x})) + \max_i t_i(\overline{x}; \overline{a}) \\ &\leq q_h(\overline{q_r}(\overline{x})) + \max_i (q_{t_i}(\overline{x}) + \max_j a_j) \\ &\leq q_h(\overline{q_r}(\overline{x})) + \textstyle\sum_i q_{t_i}(\overline{x}) + \max_i a_i. \end{aligned}$$

Therefore, choosing $q_f(\overline{x}) \equiv q_h(\overline{q_r}(\overline{x})) + \sum_i q_{t_i}(\overline{x})$ finishes the result. ∎

Theorem: $\mathcal{E}^2$ is as powerful as A

Let $f(\overline{x}; \overline{a})$ be a function in A. Then $f(\overline{x}, \overline{a})$ is computable in linear space.

Proof. The initial functions are all clearly computable in linear space. For Safe Composition observe that the composition of two linear space functions is a linear space function.

The bounding polynomials of the preceding Lemma are computable in $\mathcal{E}^2$, and therefore every instance of Predicative Primitive Recursion is an occurrence of Bounded Primitive Recursion; this gives a reduction of derivations in A to derivations in $\mathcal{E}^2$. ∎

5.4 Acknowledgement

I would like to thank Stephen Cook for suggesting that this result holds.

Chapter 6

A Machine Model, and Logspace Decidability

In this chapter we show that when s_0 is omitted from B, then the resulting class is, more or less, logspace. "More or less" here refers to the fact that the characterization is somewhat lopsided owing to the output being in an unusual format; a side effect is that we only characterize logspace functions which have small output. Furthermore, we use multiple recursion.

6.1 Definitions

6.1.1 Multiple Recursion

The class L.

L is the class of functions defined identically to B, but omitting s_0.

For the proofs in this section it will be convenient to be able to define functions by multiple recursion. Therefore we introduce a new rule, obtaining B^* and L^* as follows.

Multiple Predicative Recursion on Notation

If $\overline{h}^0$, $\overline{h}^1$, and $\overline{g}$ are in the class, then define the functions $\overline{f} = \langle f_1, \ldots, f_n \rangle$

by

$$
\begin{aligned}
f_k(0,\overline{x};\overline{a}) &= g_k(\overline{x};\overline{a})\\
f_k(yi,\overline{x};\overline{a}) &= h_k^i(y,\overline{x};\overline{a},\overline{f}(y,\overline{x};\overline{a}))
\end{aligned}
$$

for $k \in \{1,\ldots,n\}$.

Definition of B^* and L^*

> B^* is the class obtained from B by replacing Predicative Recursion with Multiple Predicative Notational Recursion. L^* is the class of functions defined identically to B^*, but omitting s_0.

We postpone discussing the exact relationship between the multiple-recursion classes B^*, L^* and their single-recursion correlates. At the end of this chapter we show that multiple recursion can be eliminated, i.e. $B = B^*$ and $L = L^*$.

6.1.2 Predicative Modulus Composition Rule

In the Recursion Simulation Lemma, we are simulating Cobham-style recursions using predicative recursions. A similar recursion simulation allows us to simulate predicative recursions within the predicative recursion class itself, and in fact we can simulate multiple predicative recursions: if $h(\overline{x};\overline{a})$ is defined in B^* (respectively, L^*) then there is $h'(w;\overline{x},\overline{a})$ defined in B^* (respectively, L^*) and there is a monotone polynomial q such that $h'(w;\overline{x},\overline{a}) = h(\overline{x};\overline{a})$ whenever $|w| \geq q(|\overline{x}|)$. The proof is essentially the same as for the simulation of Cobham recursions. It is sufficient to have the polynomial value $q(|\overline{x}|)$ depend on only the normal inputs $\overline{x}$, since recursion is only allowed on normal variables in the simulated class. We carry out a proof along these lines in Chapter 7, in the more complicated context of a two-tier system. The result for L^* uses the additional fact that we only use s_0 in simulating function definition if s_0 appeared in the original function definition.

Now, using the function $\underline{\text{mod}}$ defined in the previous chapter, we derive the following rule:

Safe Modulus Composition Rule

> From h and $\overline{r}$, $\overline{t}$, and m in B^*, obtain $f(\overline{x};\overline{a}) = h(\overline{r}(\overline{x};\overline{a})\,\underline{\text{mod}}\,m(\overline{x};);\overline{t}(\overline{x};\overline{a}))$.
> in B^*.

The derivation uses recursion simulation as follows: compute $\overline{r}(\overline{x};\overline{a})\,\underline{\mathrm{mod}}\,m(\overline{x};)$, then let $f(\overline{x};\overline{a}) = h'(Q(m(\overline{x};)); \overline{r}(\overline{x};\overline{a})\,\underline{\mathrm{mod}}\,m(\overline{x};), \overline{t}(\overline{x};\overline{a}))$ where Q is a suitable unary polynomial and h' is the function in B^* which simulates h.

A suitably modified version of Modulus Composition holds in L^*. It requires replacing $\underline{\mathrm{mod}}$ with the function $\underline{\mathrm{mod}}'$ defined like $\underline{\mathrm{mod}}$ but with s_0 changed to s_1 (see discussion in proofs below).

6.2 A machine model for B^*

For this discussion, a Turing machine has tapes which are infinite to the left, each tape having a read head and a write head. The alphabet is $\{0, 1, \sqcup\}$ and the tape contents are delimited by a $\sqcup$ which can be scanned but which is never scanned past. The tape contents are regarded as representing an integer, with the least significant bit being nearest the fixed (righthand) end of the tape. There are read-only tapes for the input. The output tape is one of the work tapes. Neither the input nor the output is restricted to "one-way" access. Any constant number of work tapes are allowed.

The first main point of this chapter is to show that a modification of this ordinary model can be used to characterize B.

Projective Turing Machine

> A *projective* Turing machine (PTM) is an ordinary Turing machine with the following addition:
>
> > Each halting state indicates a number which is either zero or else corresponds to one of the input tapes. When the halting state is entered, a value is copied onto the output tape starting at the current output head position and proceeding in the "infinite" (high-order) direction of the tape; this process is terminated by writing a $\sqcup$. The copied value is zero if the halting state was zero; otherwise it is taken bitwise from the input tape indicated by the halting state, starting at the bit at the current read head position and proceeding in the infinite (high-order) direction. This "projective halt" operation is conceptualized as an oracle operation which costs only 1 time unit and which does not require the machine itself to access the input or output tapes.

In discussing projective halts we simplify the notation by identifying a pseudo input tape, tape number 0 containing value 0, with the projective halting state zero.

Notation: Projective Halts

> "$\langle \xi, \alpha \rangle \to \beta$" denotes the projective halt consisting of copying the bits starting at position α on input tape number ξ, to the output tape starting at position β. Suppose the computed function is f; then we refer to the copied bits $\lfloor f(u_1, \ldots, u_n)/2^\beta \rfloor$ as the *high order bits of the output*, and to the unchanged bits $f(u_1, \ldots, u_n) \bmod 2^\beta$ as the *low order bits of the output.*

We may informally claim that if X is a reasonable class of Turing machines which includes the identity function and which is closed under composition, then we can modify the definition of X to be a class of projective Turing machines. Thus, Projective Turing Machines are essentially the same as ordinary Turing Machines. However, the distinction becomes interesting when we categorize the inputs (i.e. the input tapes) as either safe or normal. The input used in a projective halt may be either normal or safe. Since the projective halt only costs time 1, we can have a nontrivial class in which the time complexity is a constant with respect to the safe inputs. In particular, we can get machine classes corresponding exactly to B^* and L^*.

Safe Turing Machines

> A *safe* Turing machine (STM) is a projective Turing machine in which each input tape has been labeled as *safe* or *normal.* An STM is *polytime* if it runs in time polynomial in the lengths of the normal inputs, independently of the length of the safe inputs; and it is *logspace* if it runs in space logarithmic in the lengths of the normal inputs.
>
> A *unary* STM is an STM which only writes 1's onto the work and output tapes (except for the projective halt, where the oracle rather than the machine puts 0's on the output tape). *The output of a unary machine is understood to represent a number n by a string of zeros and ones of length n.*
>
> We use a semicolon to separate safe and normal inputs to the function computed by an STM, just as we did for B.

Lemma: $B^* \subseteq$ polytime STM's

> If $f(\overline{x}; \overline{a}) \in B^*$ then $f(\overline{x}; \overline{a})$ is computed by some polytime STM.

Proof. The proof is by induction on the derivation of f.

The projection functions, and zero, can be computed by simply entering one of the projective halting states. The other initial functions (s_0, s_1, p, $\bmod 2$ and C) are similarly computed after performing a constant number of operations.

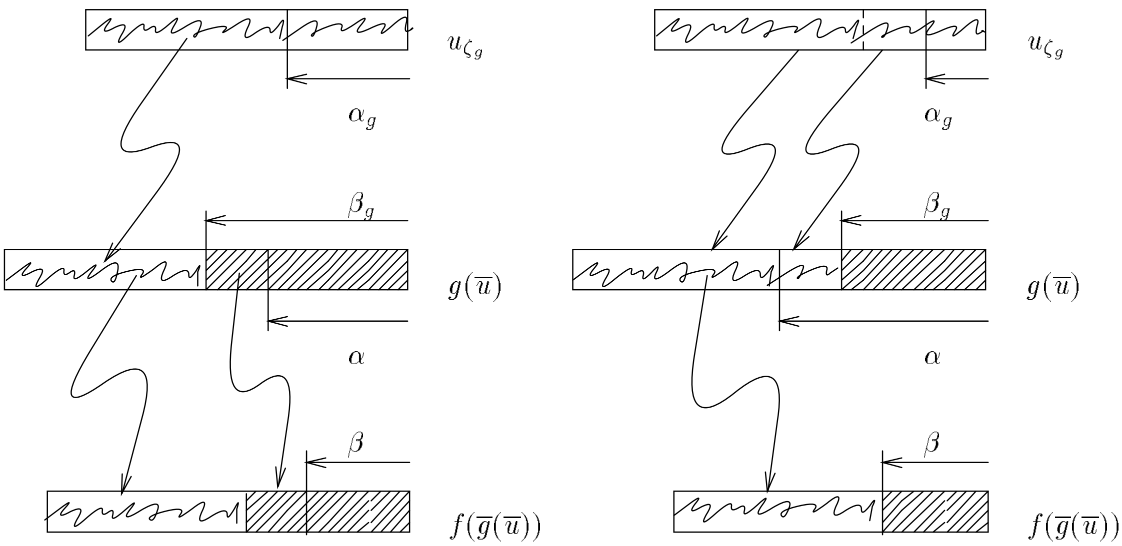
Suppose $f(\overline{x};\overline{a}) = h(\overline{r}(\overline{x};);\overline{t}(\overline{x};\overline{a}))$ is defined by Safe Composition; by the induction hypothesis we have polytime safe machines M_h, $M_{\overline{r}}$ and $M_{\overline{t}}$ computing h, $\overline{r}$ and $\overline{t}$. Let $q(|\overline{x}|)$ be the composition of the polynomials bounding the running times of M_h and $M_{\overline{r}}$; this is a bound on the number and position of the input bits accessed by M_h on input $(\overline{r}(\overline{x};);\overline{t}(\overline{x};\overline{a}))$. To compute f, we first simulate $M_{\overline{r}}(\overline{x};)$ and $M_{\overline{t}}(\overline{x};\overline{a})$ in polynomial time in $\overline{x}$ using the usual Turing machine manipulations; but we omit simulating their projective halts. Thus we have tapes containing the low-order part of the values output by $M_{\overline{r}}$ and $M_{\overline{t}}$. Now we simulate M_h, obtaining as input the low order bits of $\overline{r}(\overline{x};)$ and $\overline{t}(\overline{x};\overline{a})$ from these tapes. The high order bits of $\overline{r}(\overline{x};)$ and $\overline{t}(\overline{x};\overline{a})$ can be obtained directly from the tapes of the inputs projected by $M_{\overline{r}}$ and $M_{\overline{t}}$. Since $q(|\overline{x}|)$ is a polynomial bound on the position of any such bit, accessing the high-order input bits requires only polynomial overhead. The result of simulating M_h is a worktape containing the low-order bits of $h(\overline{r}(\overline{x};);\overline{t}(\overline{x};\overline{a}))$; we also have available the projective halts made by $M_{\overline{r}}$, $M_{\overline{t}}$, and M_h on the appropriate inputs.[1] The computation of f ends with a projective halt computed as follows.

- If the projective halt of h is $\langle 0, \alpha\rangle \rightarrow \beta$, then move the output head to position β and project 0.
- Otherwise, the halt of h is $\langle \zeta, \alpha\rangle \rightarrow \beta$ where $\zeta \geq 1$ indexes some function g in $(\overline{r}, \overline{t})$ with inputs $\overline{u}$ in $(\overline{x};\overline{a})$, and g makes a halt $\langle \zeta_g, \alpha_g\rangle \rightarrow \beta_g$ on that input. First, move to position β on the output tape, and copy $\max(0, \beta_g - \alpha)$ of the bits of $g(\overline{u})$ starting from position α. All of these copied bits are low-order bits of the output $g(\overline{u})$, hence are available on a work tape from the simulation of M_g. Next, complete the output by performing the projective halt
$$\langle \zeta_g,\ \alpha_g + \max(0, \alpha - \beta_g)\rangle \rightarrow \beta + \max(0, \beta_g - \alpha).$$

This projective halt might be more easily understood from the following diagrams, in

[1] The halt $\langle \zeta, \alpha\rangle \rightarrow \beta$ can be stored with ζ encoded in the finite state control and α and β on work tapes. In fact we can store α and β in unary notation, since their values are bounded by the running time.

which shaded areas are low-order parts of computed values.



Case where $\beta_g \geq \alpha$. Case where $\beta_g \leq \alpha$.

If $f(y, \overline{x}; \overline{a})$ is defined by Predicative Notational Recursion from h_0, h_1, and g then we simply need to perform $|y|$ compositions as described above. Each time, we obtain the low-order bits of the previous simulation from the worktape, and the high-order bits directly from the input tapes, each time keeping track of the projective halt made by the previous simulation. Iteration over $|y|$ can be performed by writing $|y|$ in unary on an extra work tape. The time required is $O(|y|)$ times the time required by the simulations of h_i and g, so the whole computation is polynomial time in the length of the normal inputs. For f defined by Multiple Predicative Notational Recursion we proceed similarly, using an extra set of work tapes for each of the recursively defined functions. ∎

Lemma: polytime STM's $\subseteq B^*$

If $f(\overline{x}; \overline{a})$ is computed by some polytime STM then $f(\overline{x}; \overline{a}) \in B^*$.

Proof. In this proof we use the encoding $0 \Rightarrow 1$, $1 \Rightarrow 11$, and $\sqcup \Rightarrow 111$; and 1111 indicates that no value is written.

First define functions W_i in B, for $i \in \{1, 11\}$, such that if $|w|$ is large enough (polynomially larger than $|b| + |c|$) then

$$W_i(w; b, c) = s_{p(;i)}(\lfloor b/2^{|c|} \rfloor) \oplus (b \bmod 2^{|c|});$$

also define $W_{111}(w; b, c) = b \bmod 2^{|c|}$ and $W_{1111}(w; b, c) = b$. The function W_i will be used to simulate the writing of bit i into position $|c|$ on the worktape; b represents the current worktape contents. The functions W_i are computable in B from normal inputs using $\dot{-}$, $\oplus$, and $\underline{\text{mod}}$; therefore by the Recursion Simulation Lemma we have W_i defined in B.

For reading, similarly define

$$R(w; b, c) = \begin{cases} 11 & \text{if } \text{PARITY}(; b \dot{-} c) = 1 \\ 111 & \text{if } b \dot{-} c = c \dot{-} b = 0 \\ 1 & \text{otherwise} \end{cases}$$

for polynomially large w.

Next define functions σ and δ_k, ω_k for each tape k, such that if $\overline{r}$ are inputs in $\{1, 11, 111\}$ representing the value in $\{0, 1, \sqcup\}$ currently scanned by the tape heads, and $\overline{c}$ is a vector of values of an equal number as $\overline{r}$ indicating (in unary) the tape head positions, and s is a unary value indicating the current machine state, then

$$\sigma(; s, \overline{r}) = \text{ the new state}$$

$$\delta_k(; \overline{c}, s, \overline{r}) = \begin{cases} p(; c_k) & \text{if head } k \text{ moves right} \\ c_k 1 & \text{if head } k \text{ moves left} \\ c_k & \text{if head } k \text{ does not move} \end{cases}$$

$$\omega_k(; s, \overline{r}) = \begin{cases} 1 & \text{if 0 is written on tape } k \\ 11 & \text{if 1 is written on tape } k \\ 111 & \text{if } \sqcup \text{ is written on tape } k \\ 1111 & \text{if no write occurs on tape } k \end{cases}$$

These are easily computed without recursion, using conditional C (and p) in a construction determined by the finite control of the machine. The projective halt states are treated as follows: if s is a state indicating a projective halt, then $\delta_k(; \overline{c}, s, \overline{r}) = c_k$ (stay at same tape position), $\sigma(; s, \overline{r}) = s$ (stay in same state) and $\omega_k(; s, \overline{r}) = 1111$ (don't write). We assume the initial state is **I**.

Now we can define the value $f_k(\cdots)$ of tape k at time t by Multiple Predicative Notational Recursion: for $1 \le k \le n$ = number of tapes,[2]

$$\begin{aligned}
s(0, w; \overline{x}, \overline{a}) &= \mathbf{I} \\
c_k(0, w; \overline{x}, \overline{a}) &= 0 \\
f_k(0, w; \overline{x}, \overline{a}) &= \pi_k(\overline{x}, \overline{a}, \overline{0}) \\
s(ti, w; \overline{x}, \overline{a}) &= \sigma(; s(t, w; \overline{x}, \overline{a}), \overline{r}) \\
c_k(ti, w; \overline{x}, \overline{a}) &= \delta_k(; \overline{c}(t, w; \overline{x}, \overline{a}), s(t, w; \overline{x}, \overline{a}), \overline{r}) \\
f_k(ti, w; \overline{x}, \overline{a}) &= \mathrm{W}_{\omega_k(; s(t, w; \overline{x}, \overline{a}), \overline{r})}(f_k(t, w; \overline{x}, \overline{a}), c_k(t, w; \overline{x}, \overline{a}))
\end{aligned}$$

where

$$\overline{r} \equiv \mathrm{R}(w; f_0(t, w; \overline{x}, \overline{a}), c_0(t, w; \overline{x}, \overline{a})), \ldots, \mathrm{R}(w; f_n(t, w; \overline{x}, \overline{a}), c_n(t, w; \overline{x}, \overline{a})).$$

Here we have avoided writing in some arguments formally required by the multiple predicative recursion scheme but not actually used in the computation.

To obtain a suitable value for w, let $q(|\overline{x}|)$ be a monotone polynomial bounding the running time of the machine; this is a bound on the lengths of the tape values and head positions input to R and W_i above. Hence there is another monotone polynomial q' such that the outputs of R and W_i above are correct whenever $|w| \ge q'(q(|\overline{x}|))$. We can compute unary $Q(\overline{x};)$ with $|Q(\overline{x};)| \ge q'(q(|\overline{x}|))$ using s_1 and Predicative Notational Recursion in $\overline{x}$ (see the definition of the declining smash function "#" given earlier).

The result of the construction is $f(\overline{x}; \overline{a}) \equiv f_n(Q(\overline{x};), Q(\overline{x};); \overline{x}, \overline{a})$, the value appearing on the output tape n before the final projective halt. We also have $\beta(\overline{x}; \overline{a}) \equiv c_n(Q(\overline{x};), Q(\overline{x};); \overline{x}, \overline{a})$, the position in unary of the head on the output tape. Similarly we have in unary the position $\alpha(\overline{x}; \overline{a}) \equiv c_{\zeta(\overline{x}; \overline{a})}(Q(\overline{x};), Q(\overline{x};); \overline{x}, \overline{a})$ of the head of the input tape (possibly zero) indicated for the projective halt, which is tape number $\zeta(\overline{x}; \overline{a}) \equiv s(Q(\overline{x};), Q(\overline{x};); \overline{x}, \overline{a})$.

Finally we can define the output of the machine by

$$M(\overline{x}; \overline{a}) = \begin{cases} f(\overline{x}; \overline{a}) \,\underline{\mathrm{mod}}(\beta(\overline{x}; \overline{a}) \,\underline{\mathrm{mod}}\, Q(\overline{x};)) & \text{if } \zeta(\overline{x}; \overline{a}) = 0 \\ (f(\overline{x}; \overline{a}) \,\underline{\mathrm{mod}}\, Q(\overline{x};)) \oplus [\pi_{\zeta(\overline{x}; \overline{a})}(\overline{x}, \overline{a}) \dot{-} (\alpha(\overline{x}; \overline{a}) \,\underline{\mathrm{mod}}\, Q(\overline{x};)] & \text{otherwise .} \end{cases}$$

[2]In writing functions, we allow the notation $f(\overline{u}) = h_{g(\overline{u})}(\cdots)$ when there are a constant number of possible values for $g(\overline{u})$; such an expression can be computed by repeated applications of the conditional C. Similarly we can perform the test $g(\overline{u}) = 0$ if there are only a constant number of possible values for $g(\overline{u})$.

Here we use Safe Modulus Composition with "$\underline{\mathrm{mod}}\, Q(\overline{x};)$" to raise safe values into normal position. The test "$\zeta(\overline{x};\overline{a}) = 0$" is computable using C, since there are only a constant number of possible values for ζ. ∎

Summarizing these two lemmas, we have proved

Corollary: B^* = polytime STM's

$$f \in B^* \Leftrightarrow f \text{ is computed by a polytime STM.}$$

6.3 A machine model for L^*.

The simulations above were carefully chosen to have the nice property that they will continue to work under the following restrictions: the STM only writes 1's (and blanks) onto the work tapes; and the function in B^* does not use s_0.

Essentially, a unary machine outputs a value in unary: although a projective halt may copy a 0 to the output tape directly from an input, intuitively this is not a significant computation. Since the output value is given by the length of the written string, the output value must have binary length $O(\log n)$. This will enter into the statements of the theorems below. First, though, we need to establish that the simulation still works for L^*.

Reviewing the simulations above, we obtain the following.

Lemma: L^* = unary polytime STM's

$$f \in L^* \Leftrightarrow f \text{ is computed by a unary polytime STM}$$

Proof. Notice that the simulation of a function in B^* by an STM does not require copying bits from the input to the work tapes, even though the final projective halt can copy an input to the output tape. For example, the composition of two machines works just as well if all the machines are restricted to write only 1's on the work tapes. The simulation of Multiple Predicative Recursion on Notation also works, keeping $|y|$ in unary on an extra work tape. The only initial function which requires writing 0 to a work or output tape is s_0. Therefore if the function being simulated is one derived with no uses of s_0, then the values written on the work tapes will consist only of 1's.

As regards the simulation of an STM by a function in B^*, note that while the definitions of W_{11} (write a '1') and W_{111} (write a '␣') could involve use of s_0, we can define them without s_0 if we only need W_{11} to be correct on unary inputs. In particular we can define $\oplus'$, F', and $\underline{\text{mod}}'$ similar to $\oplus$, F, and $\underline{\text{mod}}$ above but only using s_1. Since the Recursion Simulation Lemma does not introduce s_0, and all the relevant state and tape head position values are maintained in unary notation, the simulation carries through when these modified functions are used. Specifically, correctness of W_i and $\underline{\text{mod}}'$ on unary inputs is sufficient for a simulation in which the work and output tapes only contain 1's. The function W_{1111} does not require s_0. The rest of the construction does not use s_0 provided the simulated machine cannot write 0 onto a tape. Therefore we can carry out the construction without using s_0, provided the simulated machine is a unary STM. ∎

6.4 Logspace Decidability

Now it is a simple matter to pass from unary polytime STM's to logspace STM's.

Lemma: unary polytime STM's = small-output logspace TM's

> Let $f(\overline{x})$ be a function whose output size is $O(\log n)$, where n is the sum of the input sizes.
>
> Then $f(\overline{x};)$ is computed by a unary polytime STM if and only if $f(\overline{x})$ is computed by a logspace Turing Machine.

Proof. In one direction, we can simulate unary polytime STM's in logspace by replacing the unary values on the tapes with the corresponding binary values. The projective halt is easily simulated in logspace.

In the other direction, note that a logspace Turing machine only requires polynomial time, and therefore can be simulated by a polytime Turing machine which replaces the logarithmic-size binary values on the work (and output) tapes with polynomial-size unary values. In performing this transformation we are relying on the fact that we can easily compute the unary functions corresponding to reading and writing binary digits on the worktapes of the logspace machine. ∎

We can summarize the logspace results so far:

Theorem: L^* = small-output logspace

> Let $f(\overline{x})$ be a function whose output size is $O(\log n)$, where n is the sum of the input sizes.
>
> Then $f(\overline{x})$ is computed by a logspace Turing Machine iff $f(\overline{x};)$ is computed in L^*.

Proof. Immediate from the preceding two lemmas. ∎

For example, the 0/1 valued functions of L^* are exactly the logspace-decidable problems.

It seems possible that a slightly more regular characterization of logspace might be obtained by improving the output capabilities of the class; for example, by adding a version of Concatenation Recursion on Notation as defined by Lind [Project MAC] (See also Clote [Cl] and Gurevich [Gu]). This would not solve the problem of having 0's mixed into the unary output, only the problem of having output size $O(\log n)$. However, the complications involved in such a move would obscure the main point of this chapter, that simply omitting s_0 from the natural polytime characterization results, in some sense, in a logspace characterization. As usual, all of the output issues disappear when we restrict attention to decision problems.

The second main point of this chapter is the following corollary.

Corollary 6.4.1 *Let f be a 0/1-valued function. Then $f(\overline{x})$ is computable in logspace if and only if $f(\overline{x};)$ is computed in L.*

Proof. Follows immediately from the Theorem above that L^* = small-output logspace, and the Theorem below that $L = L^*$. ∎

6.5 Eliminating the Multiplicity of Recursion.

The purpose of this section is to eliminate the use of Multiple Recursion.

From our previous results, and using pairing functions defined in the Cobham class FP, it is easy to see that $f(\overline{x};) \in B \Leftrightarrow f(\overline{x};) \in B^*$. Unfortunately, the pairing functions used in proving this equivalence require s_0, and therefore a similar equivalence between L and L^* does not follow.

However, we can reduce the "multiplicity of recursion" to a constant in both B^* and L^*, as follows. We focus on B^*, the argument for L^* is similar.

Suppose $\overline{f}(\overline{x};\overline{a})$ is computed by multiple recursion in B^*. As discussed in the proofs above, the computation of any of the functions, $f_k(\overline{x};\overline{a})$, is achieved by a polytime STM; this machine's worktapes are space-bounded by a polynomial in the lengths of the normal inputs. By ordinary methods we can simulate this machine using just one work tape. (We include the output on the work tape). Now turning to the other Lemma, the function computed by this machine can be computed in the class B^* — and examining the proof we see that the definition in B^* requires defining at most three functions by multiple recursion, since the machine only has one tape. Thus, no matter how "large" the multiple recursions are in the original derivation computing f, they can all be replaced by a multiple recursion in which only three functions are defined.

Actually, we can press this argument further, eliminating multiple recursion entirely, as follows. First note that the three functions defined by multiple recursion are the functions giving: the current machine state, the contents of the work/output tape, and the position (in unary) of the work/output tape head. All of these values are length-bounded by a monotone polynomial q in the lengths of the normal inputs, because the machine runs in time polynomial in the lengths of the normal inputs. (The projective halt is handled separately, without multiple recursion). Furthermore, in the L^* simulation, the values have binary representations containing only 1's.

Now, in B we can compute the function $\Pi_B(w;a,b)$ which pairs together $|w|$ low-order bits of a and b by reversing and interleaving them, as in $\langle 6,5\rangle = \langle 110,101\rangle = 011011 = 27$:

$$\begin{aligned}
&\Pi_B(0;a,b) && = 0\\
&\Pi_B(wi;a,b) && = C(;a \dot{-} w,\\
&\quad C(;b \dot{-} w, \Pi(w;a,b)00, \Pi(w;a,b)01),\\
&\quad C(;b \dot{-} w, \Pi(w;a,b)10, \Pi(w;a,b)11)).
\end{aligned}$$

And in L we can compute the function $\Pi_L(w;a,b)$ whose output has a length equal to $[|a \,\underline{\mathrm{mod}}\, w|, |b \,\underline{\mathrm{mod}}\, w|]$, where $[u,v]$ is the familiar pairing function $v + \Sigma_{i=1}^{u+v} i$. For large enough w, this Π_L function is a pairing function in the sense that it provides a unary representation of the pairing of the values represented in unary by a and b. It is computed in L by:

$$\begin{aligned}
\Sigma(x, y;) &= \#'(x \oplus' y;) \\
\Pi_L(w; a, b) &= (b \,\underline{\mathrm{mod}}'\, w) \oplus' \Sigma(a \,\underline{\mathrm{mod}}'\, w, b \,\underline{\mathrm{mod}}'\, w;)
\end{aligned}$$

Here we have used Modulus Composition for the definition of Π_L. Also we have used the functions $\#'$ and $\oplus'$ which are defined the same as $\#$ and $\oplus$ except replacing all occurrences of s_0 with s_1. As in the proof of the Lemma on simulations in L^*, these functions are adequate when the inputs contain only 1's in their binary representations. It is an exercise to show how to compute the corresponding unpairing functions Π^1 and Π^2 in B and L.

To finish: the replacement of the three functions defined by multiple recursion by one function defined by single recursion is accomplished by using these pairing functions, substituting a large enough value $Q(\overline{x})$ for w. Such a function Q can be computed using $\#'$ since we have the monotone polynomial q bounding the lengths of the three functions' outputs. Omitting some details, the replacement of a multiple recursion defining f_1, f_2, and f_3 by a single function $\hat{f}$ is accomplished roughly as in:

$$\begin{aligned}
\hat{f}(0;) &= \langle g_1(), g_2(), g_3() \rangle \\
\hat{f}(yi;) &= \langle h_1(y; \Pi^1 f(y;), \Pi^2 f(y;), \Pi^3 f(y;)), \\
&\qquad h_2(y; \Pi^1 f(y;), \Pi^2 f(y;), \Pi^3 f(y;)), \\
&\qquad h_3(y; \Pi^1 f(y;), \Pi^2 f(y;), \Pi^3 f(y;)) \rangle \\
\hat{f}_i(y;) &= \Pi^i \hat{f}(y;)
\end{aligned}$$

From this discussion we can conclude the following:

Theorem: equivalence of multiple and single recursion in L and B.

$f(\overline{x}; \overline{a}) \in B \Leftrightarrow f(\overline{x}; \overline{a}) \in B^*$, and $f(\overline{x}; \overline{a}) \in L \Leftrightarrow f(\overline{x}; \overline{a}) \in L^*$.

The fact that we can eliminate multiple recursion in this setting suggests that it is a different usage of multiple recursion than appears in Leivant and Marion's recharacterization [LM2] of polynomial time.

I would like to acknowledge Charlie Rackoff for our conversation on reducing the "multiplicity of recursion" to a constant.

Chapter 7

Parallel Classes

7.1 Discussion

As discussed in section 2.2, Bloch [Bl], after reading [BC], gave a 'predicative recursion' definition of NC^1 based on the recursion defined by Allen [Al].[1] Even more recently, this result was further refined by Leivant and Marion [LM2]. We survey several other characterizations at a general level in the hope that this will be useful in further research, and then give some detail about how to convert Clote's characterization of NC.

Clote [Cl], [Cl2] showed that NC is defined from the initial functions 0, s_0, s_1, π_i^n, $\textsc{bit}(x,i)$, #, and $|x|$ plus two recursion schemes. The recursions are, given g, h_i, and r already defined, define f by either of the following:

$$\begin{array}{lll} (CRN) & f(0,\overline{v}) & = g(\overline{v}) \\ & f(ui,\overline{v}) & = s_{h_i(u,\overline{v})} f(u,\overline{v}) \ \text{ for } ui \neq 0 \end{array}$$

$$\begin{array}{lll} (WBRN) & F(0,\overline{v}) & = g(\overline{v}) \\ & F(ui,\overline{v}) & = h_i(u,\overline{v},F(u,\overline{v})) \ \text{ for } ui \neq 0 \\ & & \text{provided that } F(u,\overline{v}) \leq r(u,\overline{v}) \\ & f(u,v) & = F(|u|,v) \end{array}$$

[1] Recall that $NC = \cup_k NC^k$ where NC^k is the class of functions computed by (uniform) $O(\log^k n)$ depth, fan-in 2 circuits. $AC = \cup_k AC^k$ where AC^k functions are those computed by (uniform) $O(\log^k n)$ depth, unbounded fan-in circuits. See Cook [C5].

This is a "sequential" characterization of NC in that the recursion is not a branching recursion and the initial functions are thought of as sequential ones.[2]

Below we show how to translate Clote's characterization into a two-sorted class defined without the bounding term $F(u,\overline{v}) \leq r(u,\overline{v})$. Essentially, Clote's work in combination with our results about B indicate that NC is obtained from $\#$ on normal inputs, Clote's other initial functions on safe inputs, CRN, and a predicative version of WBRN in which $F(u,\overline{v})$ is in safe position. There is no obvious way to omit $\#$ since our previous construction for computing it fails in this setting.

Allen [Al] gave a different recursion-theoretic characterization of NC. Allen uses a form of CRN plus the following *bounded branching recursion* (BBR): given g and h_i in the class, and given polynomial p, define f by

$$\begin{aligned}
FrontHalf(u) &= u \bmod 2^{\lceil |u|/2 \rceil} \\
BackHalf(u) &= u \dot{-} FrontHalf(u)
\end{aligned}$$

$$\begin{aligned}
f(0,\overline{v}) &= g(\overline{v}) \\
f(u,\overline{v}) &= h(u,\overline{v}, f(FrontHalf(u),\overline{v}), f(BackHalf(u),\overline{v})) \;\; \text{for } u \neq 0, \\
&\quad \text{provided that } |f(u,\overline{v})| < p(|u|,|\overline{v}|) \text{ for polynomial } p
\end{aligned}$$

Bloch [Bl] modified this scheme to obtain a predicative recursion characterization of polylog space, following the ideas in $[BC]$. Aside from the indirect equivalence between BBR and WBRN which must hold because both Allen and Clote characterize NC, Allen comments directly [Al, p. 13] on the equivalence of BBR and the following "sequential" rule similar to WBRN.

Given g and h in the class, and given polynomials p and q, define f by

$$\begin{aligned}
F(u,\overline{v}) &= g(\overline{v}) \\
F(u+1,\overline{v}) &= h(u,\overline{v},F(u,\overline{v})) \;\; \text{for } u \neq 0 \\
&\quad \text{provided that } |F(u,\overline{v})| \leq q(|\overline{v}|) \text{ for all } u \leq |p(|\overline{v}|)| \\
f(\overline{v}) &= F(|p(|\overline{v}|)|,\overline{v})
\end{aligned}$$

The equivalence, of course, holds in the presence of Allen's initial functions, which

[2]The "parallel" function TREE in [Cl] is only needed to characterize NC^k as distinct from AC^k and AC^{k-1}; it is not needed to characterize NC, AC, or AC^k.

include a version of CRN and bit operators. Allen also defined a fragment of bounded arithmetic corresponding to NC.

Compton and Laflamme [CL] adopt a finite-structures setting (as in the work of Immerman [I], Gurevich [G], and others) in giving a recursion-theoretic characterization of NC^1 and a logic for NC^1. The effect of their setting on the recursion theoretic characterization is to introduce a fixed N for the maximum input value; relative to this they define, for each constant k, initial functions $N^k \rightarrow N^k$. The initial functions[3] are all linear time in the same sense as the functions of B. Although the BIT function required in the logical setting is absent, intuitively the notion of domain ordering is implicit in the successors. Of course, these definitions are uniform in N. Their functional class is defined using *Upwards Tree Recursion*, which we can write as the following: given g and h, and recalling that N is fixed, let k be any constant and define[4]

$$f(x,\overline{y}) \quad = \begin{cases} g(x,\overline{y}) & \text{if } x \geq N^k \text{ (or } x = 0) \\ h(x,\overline{y}, f(x0,\overline{y}), f(x1,\overline{y})) & \text{otherwise} \end{cases}$$

A size bound on the recursion ($f(x,\overline{y}) \leq N^c$ for some constant c) would be redundant since in this setting it follows trivially from the explicit bound on the initial functions. The constants c and k depend only on the derivation of g and h in the class. Thus the recursion schemes are effectively linear-size bounded. Compton and Laflamme further show that recursion on notation, again in their setting linear-size bounded, is simulatable by UTR.

After presenting a functional class equivalent to NC^1, Compton and Laflamme give a logic whose definable relations are exactly NC^1. Viewing predicates as 0-1 valued functions they give a recursion rule "Relational Primitive Recursion" (RPR) which is essentially simultaneous definition of functions by primitive recursion with the requirement that the defined functions be 0-1 valued. The recursion depth is 2^n for input size n; and the primitive functions are the BIT relation and first-order expressible predicates.[5] This result is interesting because the depth of recursion is now x rather than $|x|$ as in LRN or $||x||$ as in WBRN. This is traded off for constant width as in Barrington: the

[3] 0, $\pi(x) = x$, C, p, EQ, $s_0(x) \underline{\bmod} x$, and $s_1(x) \underline{\bmod} x$ where x is a vector of k inputs viewed as a single base-N value.

[4] Technically, the expression x here stands for a vector of k inputs in $\{0, \ldots, N\}$, which is viewed as a base N representation of a single value. The output of the functions are also k vectors of values in $[N]$, for some k.

[5] FO = predicates checkable in constant time on a CRAM, see Immerman [Immerman's short course].

recursion is constant-width instead of polynomial width as in WBRN.

7.2 Adapting Clote's NC characterization.

This section elaborates on the earlier comment that Clote's characterization of NC can be reworked using the idea of predicative recursion. In doing so we remove the external size bound which has previously been required for NC characterizations. The main detraction from the new characterization is the presence of $\#'$, which can be criticised as being a reference to polynomials. It is an open problem to eliminate $\#$ from the characterization. One might begin by disposing of the projection functions.

Recall that Clote's class of functions, which we here call $\textsc{clo}$, is a one-sorted class defined by the initial functions 0, s_0, s_1, π_i^n, $\textsc{bit}(x,i)$, $\#$, and $|x|$ plus three closure rules: composition and the following. Given g, h_i, and r already defined, define f by either of:

$$
\begin{array}{lll}
(CRN) & f(0,\overline{v}) & = g(\overline{v}) \\
 & f(ui,\overline{v}) & = s_{h_i(u,\overline{v})} f(u,\overline{v}) \ \text{ for } ui \neq 0 \\
 & & \\
(WBRN) & F(0,\overline{v}) & = g(\overline{v}) \\
 & F(ui,\overline{v}) & = h_i(u,\overline{v},F(u,\overline{v})) \ \text{ for } ui \neq 0 \\
 & & \text{provided that } F(u,\overline{v}) \leq r(u,\overline{v}) \\
 & f(u,v) & = F(|u|,v)
\end{array}
$$

NC is defined as the class of functions computable by a uniform family of circuits — one for each input size n — over $\{\wedge,\vee,\neg\}$ with size polynomial in n and depth polylog n. These are functions computed on binary strings rather than integers. Since we are discussing all of NC, we can use U_E^* uniformity, logspace uniformity, or first-order uniformity interchangeably; for AC^0 the uniformity condition would be first-order uniformity. Formally, to compare NC and the recursive function class A, we require translating between functions computed over strings in $\{0,1\}^+$ and functions computed over integers; this is done in an ordinary way by ignoring leading 0's. See Clote [Cl] for details.

Clote's Theorem

$\textsc{clo} = NC$.

For a proof see [Cl]. Clote further shows that FO (like AC^0 but with Immerman's uniformity; it equals the logtime hierarchy) is equivalent to the class defined as above but omitting WBRN. Indeed the depth of nestings of WBRN corresponds directly the level of the AC^k hierarchy.

The discussion is simplified a little bit by introducing two functions. Define $H(u) = \lfloor u/(2^{\lceil |u|/2 \rceil}) \rfloor$ and $P(u) = \lfloor u/2 \rfloor$. The function H is pronounced "half" and gives a value which is about half as long as u. The least number of times which H can be iterated on u before reaching zero is $||u||$.

It is evident that the same class as defined by Clote will be obtained if one replaces $WBRN$ with the following rule:[6]

$$\begin{array}{lll} (WBRN') & f(0,\overline{v}) & = g(\overline{v}) \\ & f(u,\overline{v}) & = h(u,\overline{v},f(H(u),\overline{v})) \ \text{ for } u \neq 0 \\ & & \text{provided that } f(u,\overline{v}) \leq r(|u|,\overline{v}) \end{array}$$

In fact the bound $r(|u|,\overline{v})$ can be changed to $r(u,\overline{v})$ without altering the defined class. To see this, observe that we can simply place a copy of u among the inputs $\overline{v}$ (using projection and composition). In the following we will work with $WBRN'$ rather than $WBRN$.

The following two-sorted class differs from CLO in not having an external bound on the growth rate.

Definition of 2CLO

> 2CLO (two-tier NC) is the smallest class of functions containing functions `i-vii` and closed under `viii-x` below.

`i.` **(Constant)** 0 (a zero-ary function).

`ii.` **(Projection)** $\pi_j^{n,m}(x_1 \dots x_n; x_{n+1} \dots x_{n+m}) = x_j$, for $1 \leq j \leq n+m$.

`iii.` **(Successors)** $s_0(;a) = 2a; \quad s_1(;a) = 2a+1$

[6] $WBRN$ is simulated by $WBRN'$ as follows: given h_i and g, define $h(x,\overline{v},z)$ by "if $|x| \bmod 2 = 0$ then $h_0(|x|-1,\overline{v},z)$ else $h_1(|x|-1,\overline{v},z)$"; then $WBRN'$ on h and g computes the same function as $WBRN$ on h_i and g. In the reverse direction, given h and g define $h_i(u,x,\overline{v},z) = h(E(u,x),\overline{v},z)$ where $E(u,x)$ is the low-order u bits of x (assuming $u \leq |x|$); obtaining $\hat{f}(u,x,\overline{v})$ from $WBRN$ on h_i, the function computed by $WBRN'$ on h is given by $f(u,\overline{v}) = \hat{f}(u,u,\overline{v})$. The functions required to carry out these simulations are in the required classes.

iv. **(Conditional)** $C(;a,b,c) = \begin{cases} b & \text{if } a \bmod 2 = 0 \\ c & \text{otherwise} \end{cases}$

v. **(Bit Test)** $\text{BIT}(;a,b) = \lfloor a/2^b \rfloor \bmod 2 = b$th bit of a

vi. **(Length)** $|a| \equiv L(;a) = \lceil \log(a+1) \rceil$

vii. **(Smash)** $\#'(w;a,b) = 2^{|a|\cdot|b|} \,\underline{\bmod}\, 2^{|w|\cdot|w|}$.

viii. **(Safe Composition)** Define new function f by $f(\overline{x};\overline{a}) = h(\overline{r}(\overline{x};);\overline{t}(\overline{x};\overline{a}))$ where h, $\overline{r}$, and $\overline{t}$ are in 2CLO.

ix. **(Concatenation Recursion)** Define new function f by

$$\begin{aligned} f(\overline{x};0,\overline{a}) &= 0 \\ f(\overline{x};b,\overline{a}) &= s_{h(\overline{x};b,\overline{a}) \bmod 2} f(\overline{x};P(b),\overline{a}) \quad \text{for } b \neq 0 \end{aligned}$$

where h and g are in 2CLO.

x. **(Log Recursion)** Define new function f by

$$\begin{aligned} f(0,\overline{x};\overline{a}) &= g(\overline{x};\overline{a}) \\ f(y,\overline{x};\overline{a}) &= h(y,\overline{x};\overline{a},f(H(y),\overline{x};\overline{a})) \quad \text{for } y \neq 0 \end{aligned}$$

where h and g are in 2CLO.

The set of initial function chosen here reflects the initial functions of Clote. We have modified smash for reasons that will become clear in the theorem on the equivalence of NC and 2CLO; the ordinary smash function can be computed from the modified one above.

The conditional is required because Concatenation recursion only uses one step function h instead of two. Notice that Concatenation Recursion differs from CRN in not allowing an arbitrary function in the base case. This is required in order to prevent the concatenation of two safe inputs; if we could concatenate two safe inputs then, by repeated doubling, we could define an exponentially-growing function. An alternative to omitting the arbitrary function in the base case of CRN would be to have three tiers, with log recursion on tier 2, CRN on tier 1, and no recursion on tier 0.

Concatenation recursion can be used to test against zero, to reverse $|a|$ low-order bits of a number b, to to compare the lengths of two numbers, to compare a number to zero, and to obtain a number whose length is the difference of the lengths of two other numbers. Following the style of Kleene, 0 is used for TRUE and 1 for FALSE.

$$
\begin{aligned}
\textsc{ones}(;0) &= 0\\
\textsc{ones}(;a) &= s_1(;\textsc{ones}(;P(a))) \quad \text{for } a \neq 0\\
C'(;a,b,c) &= C(;\textsc{ones}(a),b,c)\\
\\
\textsc{rev}(;0,b) &= 0\\
\textsc{rev}(;a,b) &= s_{\mathrm{BIT}(;b,|a|)}\textsc{rev}(;P(a),b) \quad \text{for } a \neq 0\\
\\
b \,\underline{\mathrm{mod}}\, c &= \textsc{rev}(;c,\textsc{rev}(;c,b))\\
\\
|a| \leq |b| &\equiv \textsc{bit}(;\textsc{ones}(;a),|b|)\\
|a| < |b| &\equiv |s_1 a| \leq |b|\\
a = 0 &\equiv |a| \leq |0|\\
\\
0 \dot{-} b &= 0\\
a \dot{-} b &= s_{\mathrm{BIT}(a,|b|)}(P(a) \dot{-} b) \quad \text{for } a \neq 0
\end{aligned}
$$

Similar functions were defined in [Cl], partly based on Lind [Lin].

Based on the experience of our previous classes, it is evident that the growth rate of this class is at most polynomial. In fact this class seems to be contained in B; let's prove the same bound.

Bounding Lemma for 2CLO.

> Let f be a function in 2CLO. Then there is a monotone polynomial q_f such that $|f(\overline{u};\overline{x};\overline{a})| \leq q_f(|\overline{u}|) + \max\{|\overline{a}|\}$.

Proof. The result is trivial for the functions 0, s_0, s_1, C, π, BIT, $|\cdot|$, and $\#$. The length of the output from Concatenation Recursion is no greater than the length of the safe input to which it is applied, which is at most $\max_i |a_i|$.

As in the Bounding Lemma for B, safe composition of functions leads to composition of the bounding polynomials.

Suppose f is defined by Log Recursion,

$$\begin{aligned} f(0,\overline{x};\overline{a}) &= g(\overline{x};\overline{a}) \\ f(y,\overline{x};\overline{a}) &= h(y,\overline{x};\overline{a},f(H(y),\overline{x};\overline{a})) \ \text{ for } y \neq 0 \end{aligned}$$

where h and g are in 2CLO. Obtaining q_h and q_g by the induction hypothesis, define q_f such that

$$q_f(|y|,|\overline{x}|) = |y| \cdot q_h(|y|,|\overline{x}|) + q_g(|y|,|\overline{x}|);$$

this is a gross over-estimate since in reality we will only have $||y||$ iterations. Proceeding by induction on $|y|$ one shows $|f(y,\overline{x};\overline{a})| \leq q_f(|y|,|\overline{x}|) + \max_i |a_i|$; the essential step is

$$\begin{aligned} f(y,\overline{x};\overline{a}) &\leq q_h(|y|,|\overline{x}|) + \max\{|\overline{a}|, |f(H(y),\overline{x};\overline{a})|\} \\ &\leq q_h(|y|,|\overline{x}|) + \max\{|\overline{a}|, q_f(|H(y)|,|\overline{x}|) + \max\{|\overline{a}|\}\} \\ &\leq q_h(|y|,|\overline{x}|) + q_f(|H(y)|,|\overline{x}|) + \max\{|\overline{a}|\} \\ &\leq q_h(|y|,|\overline{x}|) + |H(y)| q_h(|H(y)|,|\overline{x}|) + q_g(|H(y)|,|\overline{x}|) + \max\{|\overline{a}|\} \\ &\leq q_f(|y|,|\overline{x}|) + \max\{|\overline{a}|\}. \end{aligned}$$

∎

The following is the main theorem of this section.

Theorem: 2CLO is equivalent to NC

$f(\overline{u}) \in NC$ if and only if $f(\overline{u};) \in$ 2CLO.

Proof. The easier direction is that if $f \in$ 2CLO then $f \in NC$. For this, just note that all the initial functions are functions in CLO, and composition and Concatenation Recursion are rules in CLO. Instances of Log Recursion in 2CLO are identically instances of $WBRN'$ since the required bounding term is provided by the Bounding Lemma.

To prove that $f(\overline{u}) \in NC \Rightarrow f(\overline{u};) \in$ 2CLO, we prove the stronger fact that if $f(\overline{u}) \in NC$ then there is $f'(w;\mathbf{\overline{u}}) \in$ 2CLO and a polynomial p_f such that $f(w;\mathbf{\overline{u}}) = f(\mathbf{\overline{u}})$ whenever $|w| \geq p_f(|\mathbf{\overline{u}}|)$. This implies the result since, using $\#'$ and safe composition, we can define a function $W(\overline{u};)$ in 2CLO such that $|W(\overline{u};)| \geq p_f(|\overline{u}|)$ for all $\overline{u}$; then $f(\overline{u};) = f'(W(\overline{u};);\overline{u})$ is definable in 2CLO. The proof of the strengthened statement is by induction on the definition of f as a member of CLO.

If f is 0, s_0, s_1, π_i, BIT, or $|\cdot|$ then we can define f' directly in 2CLO using safe composition and projection. If f is $\#$ then $\#'$ serves in 2CLO.

If f is defined by composition, $f(\overline{u}) = h(\overline{t}(\overline{u}))$, then obtain $\overline{t}'$ and h' by the induction hypothesis and define $f'(w; \overline{\mathbf{u}}) = h'(w; \overline{t}'(w; \overline{\mathbf{u}}))$. Obtaining bounding polynomials $q_{\overline{t}}$ by the lemma above, and using monotonicity, the result follows with $p_f(|\overline{\mathbf{u}}|) = p_h(q_{\overline{t}}(|\overline{\mathbf{u}}|)) + \Sigma p_{\overline{t}}(|\overline{\mathbf{u}}|)$.

If f is defined by CRN, say

$$\begin{aligned} f(0, \overline{v}) &= g(\overline{v}) \\ f(ui, \overline{v}) &= s_{h_i(u,\overline{v})} f(u, \overline{v}) \ \text{ for } ui \neq 0 \end{aligned}$$

then we have h_i' and g' in 2CLO by the induction hypothesis. The problem is that Concatenation Recursion does not allow g' to be directly present. The plan is to use w as a template value into which we will poke the bits of $f(u, \overline{v})$. Note that the low $|u|$ bits of $f(u, \overline{v})$ are determined by h, while the high bits are given by g. Using Concatenation Recursion, define

$$\begin{aligned} h'(w; \mathbf{u}, \overline{\mathbf{v}}) &= C(; \mathbf{u},\ h_0'(w; p\mathbf{u}, \overline{\mathbf{v}}),\ h_1'(w; p\mathbf{u}, \overline{\mathbf{v}})) \\ \hat{f}(w; 0, \mathbf{u}, \overline{\mathbf{v}}) &= 0 \\ \hat{f}(w; c, \mathbf{u}, \overline{\mathbf{v}}) &= s_{C(;|c| \leq |\mathbf{u}|,\ h'(w; \mathbf{u} \,\underline{\mathrm{mod}}\, c, \overline{\mathbf{v}}),\ \mathrm{BIT}(; g'(w; \overline{\mathbf{v}}), |c \dot{-} h'(w; \mathbf{u}, \overline{\mathbf{v}})|)} \hat{f}(w; P(c), \mathbf{u}, \overline{\mathbf{v}}) \ \text{ for } c \neq 0 \end{aligned}$$

It follows that if $|w|$ is large enough for g' and h', then $\hat{f}(w; c, \mathbf{u}, \overline{\mathbf{v}})$ is the $|c|$ low-order bits of $f(\mathbf{u}, \overline{\mathbf{v}})$. Since g is in NC, g is length-bounded by a polynomial q_g; there are at most $|u| + |g(\overline{v})|$ bits in $f(\mathbf{u}, \overline{\mathbf{v}})$. Defining $f'(w; \mathbf{u}, \overline{\mathbf{v}}) = \hat{f}(w; w, \mathbf{u}, \overline{\mathbf{v}})$ and $p_f(|u|, |\overline{v}|) = p_g(|\overline{v}|) + p_h(|u|, |\overline{v}|) + (|u| + q_g(|u|, |\overline{v}|))$ finishes this case.

The last and most difficult case is when f is defined by $WBRN'$, say

$$\begin{aligned} f(0, \overline{x}) &= g(\overline{x}) \\ f(y, \overline{x}) &= h(y, \overline{x}, f(H(y), \overline{x})) \ \text{ for } y \neq 0 \\ f(y, \overline{x}) &\leq r(y, \overline{x}). \end{aligned}$$

Again we have g' and h', and also r', in 2CLO by the induction hypothesis. Of course we cannot define $f'(w; \mathbf{y}, \overline{\mathbf{x}})$ by recursion on $\mathbf{y}$ since $\mathbf{y}$ is in safe position. Instead we follow the proof of the recursion simulation lemma for B.

In the following we define a function $V(\zeta, w; \mathbf{y})$ which plays the same role as Y does in the recursion simulation lemma for B. V applies the step function H on y repeatedly, returning $H^{(k)}(\mathbf{y})$ where k is the least number satisfying $|H^{(k)}(w)| \leq |\zeta|$. In fact the equality $H^{(k)}(w) = \zeta$ will hold in the context in which V is used. Thus, the difference between w and ζ determines how many times the step function is to be applied to $\mathbf{y}$. Recall $H(u) = \lfloor u/(2^{\lceil |u|/2 \rceil})\rfloor$; it is computed by HALF defined below.

$$
\begin{aligned}
&\text{HALFONES}(;0,b) && = 0 \\
&\text{HALFONES}(;a,b) && = s_{C(;|b \dot{-} a|<|a|,1,0)}\text{HALFONES}(;P(a),b) \ \text{ for } a \neq 0 \\
&\text{HALF}(;b) && = b \dot{-} \text{HALFONES}(;b,b)
\end{aligned}
$$

$$
\begin{aligned}
&V(\zeta, 0; \mathbf{y}) && = \mathbf{y} \\
&V(\zeta, w; \mathbf{y}) && = C(;|w| \leq |\zeta|,\ \mathbf{y},\ \text{HALF}(;V(\zeta, H(w); \mathbf{y}))) \ \text{ for } w \neq 0
\end{aligned}
$$

$$
\begin{aligned}
&\hat{f}(w, 0; \mathbf{y}, \overline{\mathbf{x}}) && = 0 \\
&\hat{f}(w, \zeta; \mathbf{y}, \overline{\mathbf{x}}) && = C(;V(\zeta, w; \mathbf{y}) = 0, \\
& && \qquad g'(w; \overline{\mathbf{x}}), \\
& && \qquad h'(w; V(\zeta, w; \mathbf{y}), \overline{\mathbf{x}}\hat{f}(w, H(\zeta); \mathbf{y}, \overline{\mathbf{x}})) \\
& && \quad) \ \text{ for } \zeta \neq 0
\end{aligned}
$$

$$
f(w; \mathbf{y}, \overline{\mathbf{x}}) = \hat{f}(w, w; \mathbf{y}, \overline{\mathbf{x}})
$$

Since r is in NC there is a polynomial q_r which length-bounds r and therefore f. Define $p_f(|\overline{x}|, |y|) \equiv 2|y| + p_h(|\overline{x}|, q_r(|y|, |\overline{x}|)) + p_g(|\overline{x}|)$, and let w satisfy $|w| \geq p_f(|\overline{x}|, |y|)$.

Analogously to the proof of the recursion simulation lemma for B, we prove by induction on ζ that $\hat{f}(w, \zeta; \mathbf{y}, \overline{\mathbf{x}}) = f(V(\zeta, w; \mathbf{y}), \overline{\mathbf{x}})$ for appropriate values of ζ. In this induction we fix $\mathbf{y}$ and $\overline{\mathbf{x}}$, fix w such that $|w| \geq p_f(|\overline{\mathbf{x}}|, |\mathbf{y}|)$, and consider only values of ζ in the set $\{H^{(k)}(w) :\ 0 \leq k \leq ||\mathbf{y}||\}$. By the definition of V, the values $V(\zeta, w; \mathbf{y})$ corresponding to these values of ζ are exactly the values $\{H^{(k)}(\mathbf{y}) :\ 0 \leq k \leq ||\mathbf{y}||\}$. We start at $k = ||y||$ ($\zeta = H^{||y||}(w)$, $V(\zeta, w; \mathbf{y}) = 0$) and work towards $k = 0$ ($\zeta = w$, $V(\zeta, w; \mathbf{y}) = y$). The result will follow since these are the only values of ζ that occur in the recursion defining the value $f(w; \mathbf{y}, \overline{\mathbf{x}})$.

The base case of the induction is $\zeta = H^{(||\mathbf{y}||)}(w)$, equivalently $V(\zeta, w; \mathbf{y}) = H^{(||\mathbf{y}||)}(\mathbf{y}) = 0$. This condition implies $\zeta \neq 0$ because $|w| \geq 2|\mathbf{y}|$. Hence we do not need to consider $\zeta = 0$ in the definition of $\hat{f}$. Instead, the base case of the induction follows from the occurrence of the conditional and g' in the case $\zeta \neq 0$. (It is easy to see that w is large enough that g' computes the same value as g in this context).

The step case of the induction is $V(\zeta, w; \mathbf{y}) = H^k(\mathbf{y}) \neq 0$ with $k < ||y||$. This case follows by using the fact $V(H(;\zeta), w; \mathbf{y}) = H(V(\zeta, w; \mathbf{y}))$ in the other clause of the conditional. The induction hypothesis is

$$\hat{f}(w, H(\zeta); \mathbf{y}, \overline{\mathbf{x}}) = f(V(H(\zeta), w; \mathbf{y}), \overline{\mathbf{x}}) = f(H(V(\zeta, w; \mathbf{y})), \overline{\mathbf{x}}),$$

and using $V(\zeta, w; \mathbf{y}) \neq 0$ gives

$$\begin{aligned} \hat{f}(w, \zeta; \mathbf{y}, \overline{\mathbf{x}}) &= h'(w; V(\zeta, w; \mathbf{y}), \overline{\mathbf{x}}, \hat{f}(w, H(\zeta); \mathbf{y}, \overline{\mathbf{x}})) \\ &= h'(w; V(\zeta, w; \mathbf{y}), \overline{\mathbf{x}}, f(H(V(\zeta, w; \mathbf{y}), \overline{\mathbf{x}})). \end{aligned}$$

The case is finished by observing that

$$\begin{aligned} |w| &\geq p_h(|\mathbf{y}|, |\overline{\mathbf{x}}|, q_r(|\mathbf{y}|, |\overline{\mathbf{x}}|)) \\ &\geq p_h(|V(\zeta, w; \mathbf{y})|, |\overline{\mathbf{x}}|, q_r(|H(V(\zeta, w; \mathbf{y}))|, |\overline{\mathbf{x}}|)) \\ &\geq p_h(|V(\zeta, w; \mathbf{y})|, |\overline{\mathbf{x}}|, |f(H(V(\zeta, w; \mathbf{y})), \overline{\mathbf{x}})|) \end{aligned}$$

and therefore h' computes the same value as h in this context. This gives

$$\hat{f}(w, \zeta; \mathbf{y}, \overline{\mathbf{x}}) = h(V(\zeta, w; \mathbf{y}), \overline{\mathbf{x}}, f(H(V(\zeta, w; \mathbf{y})), \overline{\mathbf{x}})) = f(V(\zeta, w; \mathbf{y}), \overline{\mathbf{x}})$$

as required. ∎

Chapter 8

A Higher Type Class

This chapter defines a new higher-type class using the ideas developed in the preceding chapters. The purpose is, first, to show that the use of sorts is robust and carries over to higher type; and second, to define a natural class which contains the Basic Feasible Functionals of Cook and Kapron [CK], [KC] and which also computes the Well-Quasi Ordering functional discussed by Cook [C2]. For further discussion on the current state of higher type feasibility, see the closing section of this chapter.

8.1 The class BC.

In this section we define a class of functions BC using the same ideas as were used for B. The class BC is better suited to the purposes of this chapter as well as the subsequent chapters on logic; however, definitions in BC are somewhat more circuitous since they use the function mod.

In a two-sorted formulation we ascribe sort ν ("normal") and σ ("safe") to the outputs as well as the inputs of functions. Analogously with B, we use the convention that x, y, z have normal sort, and a, b, c have safe sort.

Definition of BC

The class BC is the smallest class which contains the following initial functions and is closed under the following operations:

`(i-iv)` Initial functions 0, s_0, and s_1 with safe output and safe input. Initial functions $\pi_i^{n,m}$ (projection) with with safe output and both normal and safe inputs. Initial function $\underline{\mathrm{mod}}(x;a) = a \bmod 2^{|x|}$ with normal output.

(v) Closure under ordinary composition $f(\overline{x};\overline{a}) = h(\overline{r}(\overline{x};\overline{a});\overline{t}(\overline{x};\overline{a}))$ (respecting the sorts). Here f has the same sort of output as h.

(vi) Closure under predicative notational recursion; the defined function f has safe output.

(vii) Closure under *raising*: from $f(\overline{x};)$ with safe output obtain $f^\nu(\overline{x};)$ with normal output. Notice f must have only normal inputs.

Using the construction $\pi_1^{1,0}(x)$ we are free to use normal values x as if they were safe, and we do so without special discussion.

The class BC is not the same as B; for example, $p(;a)$ is in B but not in BC. However, BC is as good as B for characterizing polynomial time: $f(\overline{x};) \in BC$ iff $f(\overline{x}) \in FP$. In one direction, we can show that functions in BC are length-bounded by a polynomial in the length of the inputs:

Bounding Lemma for BC

1. If f is in BC then there is a monotone polynomial q_f such that
$$|f(\overline{x};\overline{a})| \leq q_f(|\overline{x}|) + \max_i |a_i|.$$
2. If f has type ν output, then in fact
$$|f(\overline{x};\overline{a})| \leq q_f(|\overline{x}|).$$

The proof of this is left as an exercise; it is similar to the proof of the bounding lemma for $BC(|\cdot|)$ given below, but simplified by omitting the terms $||\overline{a}||$.

In the other direction, we can simulate Cobham recursions in BC. Without getting into details: if $f(xi,a) = h_i(x, f(x))$ is defined with $f(x) \leq t(x)$ in the Cobham class, we can define $f(xi;) = h_i(x, f(x;)\,\underline{\mathrm{mod}}\,t(x;)1;)$ in BC.

We could also define an appropriate two-sorted class with Raising by replacing initial function $\underline{\mathrm{mod}}$ with $p(;a)$ and $C(;a,b,c)$ (safe output) as in the class B. However, the presence of $\underline{\mathrm{mod}}$ is helpful in getting rid of the Raising rule, which we would like to do in order to simplify the definition of the higher type class. Furthermore, we can "simulate" B functions in BC in the sense that for every B function $f(\overline{x};\overline{a})$ there is a BC function

$\hat{f}(w; \overline{x}, \overline{a})$ such that $\hat{f}(w; \overline{x}, \overline{a}) = f(x; \overline{a})$ for large enough w. We omit a proof since this is not central to our discussion.

In terms of showing that our class computes the polytime functions, the proof for BC is simpler than the proof for B; however, this is somewhat offset by the fact that $\underline{\text{mod}}$ is not a constant-time operation as are the initial functions of B. Furthermore, one can object to $\underline{\text{mod}}(x; a)$ on the basis that it says: we can use the value a provided we have a bounding term x for a. Thus a use of $\underline{\text{mod}}$ is arguably a point at which the polynomial growth features of predicative recursion are used to bound some term.

Nevertheless, we use $\underline{\text{mod}}$ in order to technically simplify the exposition of the second part of the thesis. See the discussion at the end of Part II for a further explanation and a sketch of how the use of $\underline{\text{mod}}$ could be eliminated.

8.2 Extending BC, and defining BC^{ω}

In this section we discuss the addition of an extra initial function to the two-sorted class BC. First we would like to dispose of the Raising rule.

Replacing Raising

> The functions in BC are also obtained by replacing the Raising rule with the initial function $(x\#y)^{\nu}$ with normal output and normal inputs.

(We can simulate the Raising of $f(\overline{x};)$ by the composition $\underline{\text{mod}}(Q(\overline{x};); f(\overline{x};))$ where Q is an appropriately large function defined using smash. In the other direction, BC can define $(x\#y)^{\nu}$ using Raising, by the theorem that it can define any polytime function.)

In what follows we use the formulation of BC without Raising. This is more convenient for a lambda calculus formulation.

Write $BC(|\cdot|)$ to refer to the class generated by adding $\lambda a.|a|^{\nu}$ (safe input, normal output) as an initial function in the definition of BC. The point of this section is to prove the following.

Lemma: $BC(|\cdot|)$ is not stronger than BC.

> The normal function of normal inputs defined by $BC(|\cdot|)$ are the same as the normal functions of normal inputs defined by BC.

The statement follows by proving a suitable size bound on the functions in $BC(|\cdot|)$: (1) the polytime functions continue to be definable as in BC and (2) since $|\cdot|$ is polytime and the outputs are polynomially length-bounded (see below), derivations in $BC(|\cdot|)$ are equivalent to derivations in Cobham's class. Hence the normal part of $BC(|\cdot|)$ is FP, as is the normal part of BC.

The new size bound for Predicative Notational Recursion will use the following lemma.

Lemma: Bounding p^*.

Let $p_h(n, \overline{m}, c)$ and $p_g(\overline{m})$ be any monotone polynomials, and define p^* by

$$\begin{aligned} p^*(0, \overline{m}) &= p_g(\overline{m}) \\ p^*(n+1, \overline{m}) &= p_h(n, \overline{m}, |p^*(n, \overline{m})|) \end{aligned}$$

Then there is a monotone polynomial p_f such that $p^*(n, \overline{m}) \leq p_f(n, \overline{m})$ for all n, $\overline{m}$.

Proof. Consider any $v \geq 1$ and constant w. Then $\forall x.\ |x| \leq \sqrt[w]{(q(v)+x)/v}$, where q is a polynomial. Here v can depend on any values other than x. (There is a constant k depending on w such that $\forall z. |z^{k+1}| \leq \sqrt[w]{z^k}$; define $q(v) = v^{k+1}$. To prove the fact, consider cases $x \leq q(v)$ and $x \geq q(v)$; in the second case, note $\sqrt[w]{x/v} - |x|$ is positive and monotone increasing for $x > v^{k+1}$.)

For the purpose of proving the theorem one can increase $p_h(n, \overline{m}, c)$ so that it has the form $c^w v + p'_h(n, \overline{m})$, where $v \geq 1$ depends only on n and $\overline{m}$, and w is a constant (and of course v and p'_h are monotone polynomials). Defining q as above,

$$\begin{aligned} p^*(0, \overline{m}) &\leq p_g(\overline{m}) \\ p^*(n+1, \overline{m}) &\leq p_h(n, \overline{m}, |p^*(n, \overline{m})|) \\ &\leq p_h(n, \overline{m}, \sqrt[w]{(q(v) + p^*(n, \overline{m}))/v}) \\ &\leq p_h(n, \overline{m}, 0) + (q(v) + p^*(n, \overline{m})) \end{aligned}$$

Solving this recurrence using the monotonicity of p_h we obtain $p^*(n, \overline{m}) \leq (n+1) \cdot (p(n, \overline{m}, 0) + q(v)) = (n+1)(p(n, \overline{m}, 0) + v^{w+1}$, where $p = p_h + p_g$. ∎

The following lemma replaces the bounding lemma which holds for the original definition of BC.

Bounding Lemma for $BC(|\cdot|)$

1. If f is in $BC(|\cdot|)$ then there is a monotone polynomial q_f such that
$$|f(\overline{x};\overline{a})| \leq q_f(|\overline{x}|, ||\overline{a}||) + \max_i |a_i|.$$

2. If f has type ν output, then in fact
$$|f(\overline{x};\overline{a})| \leq q_f(|\overline{x}|, ||\overline{a}||).$$

Proof. The proof of this is based on the proof of the bounding lemma for B; we note the following changes. For the initial function $\lambda a.|a|^\nu$ it is trivial.

For a function defined by composition, $f(\overline{x};\overline{a}) = h(\overline{r}(\overline{x};\overline{a});\overline{t}(\overline{x};\overline{a}))$, we have the bound

$$\begin{aligned} |f(\overline{x};\overline{a})| &\leq q_h(|\overline{r}(\overline{x};\overline{a})|, ||\overline{t}(\overline{x};\overline{a})||) + \max_i |t_i(\overline{x};\overline{a})| \\ &\leq q_h(q_r(|\overline{x}|, ||\overline{a}||), |\,(q_t(|\overline{x}|, ||\overline{a}||) + \max_i |a_i|)\,|) + \max(q_t(|\overline{x}|, ||\overline{a}||) + \max_i |a_i|) \\ &\leq q_h(q_r(|\overline{x}|, ||\overline{a}||), q_t(|\overline{x}|, ||\overline{a}||) + \textstyle\sum_i ||a_i||) + \sum q_t(|\overline{x}|, ||\overline{a}||) + \max_i |a_i| \end{aligned}$$

This proves the required bound when f has safe output. If f has normal output, then we must prove the tighter bound (2); it follows because h must have normal output and therefore by (2) we can omit the trailing terms stemming from "$+\max_i |t_i(\overline{x};\overline{a})|$" above.

For Predicative Notational Recursion, define a monotone function q_f^* by

$$\begin{aligned} q_f^*(|0|, |\overline{y}|; ||\overline{a}||) &= q_g(|y|, ||a||) \\ q_f^*(|xi|, |\overline{y}|; ||\overline{a}||) &= |xi| \cdot q_h(|xi|, |\overline{y}|, ||a||, |q_f^*(|x|, |\overline{y}|, ||\overline{a}||)| + \Sigma_i ||a_i||) + q_g(|y|, ||a||). \end{aligned}$$

This definition fits the form required by the lemma above. Now one proves by induction on $|x|$ that $|f(x,\overline{y};\overline{a})| \leq q_f^*(|x|, |\overline{y}|, ||\overline{a}||) + \max_i \overline{a}_i$:

$$\begin{aligned} |f(0,\overline{y};\overline{a})| &\leq q_g(|\overline{y}|, ||\overline{a}||) + \max_i |a_i| \\ &\leq q_f^*(|0|, |\overline{y}|, ||\overline{a}||) + \max_i |a_i| \end{aligned}$$

$$
\begin{aligned}
|f(xi,\overline{y};\overline{a})| &\le q_h(|x|,|\overline{y}|,||\overline{a}||,||f(x,\overline{y};\overline{a})||)\\
&\quad + \max(|f(x,\overline{y};a)|,\max_i|a_i|)\\
&\le q_h(|x|,|\overline{y}|,||\overline{a}||,|(q_f^*(|x|,|\overline{y}|,||\overline{a}||)+\max_i|a_i|)|)\\
&\quad + q_f^*(|x|,|\overline{y}|,||a||)+\max_i|a_i|\\
&\le q_h(|x|,|\overline{y}|,||\overline{a}||,|q_f^*(|x|,|\overline{y}|,||\overline{a}||)|+\Sigma_i||a_i||)\\
&\quad + q_f^*(|x|,|\overline{y}|,||a||)+\max_i|a_i|\\
&\le q_h(|x|,|\overline{y}|,||\overline{a}||,|q_f^*(|x|,|\overline{y}|,||\overline{a}||)|+\Sigma_i||a_i||)\\
&\quad + |x|\cdot q_h(|x|,|\overline{y}|,||\overline{a}||,|q_f^*(|x|,|\overline{y}|,||\overline{a}||)|+\Sigma_i||a_i||)+q_g(|\overline{y}|,||\overline{a}||)+\max_i|a_i|\\
&\le q_f^*(|xi|,|\overline{y}|,||\overline{a}||)+\max_i|a_i|
\end{aligned}
$$

(In obtaining the fourth inequality for $|f(xi,\overline{y};\overline{a})|$, we use the monotonicity of q_f^* and q_h, and consider cases $x=0$, $x\neq 0$). Now applying the lemma above gives the required monotone polynomial q_f bounding q_f^*. ∎

8.3 Defining BC^ω

From the formulation above, we have that $BC(|\cdot|)$ is the class of functions defined by ordinary (sorted) composition and Predicative Notational Recursion from the initial functions 0, projection, <u>mod</u>, s_0, s_1, and $|\cdot|$.

We now introduce R of type level 2, a recursor defined by $R(g,h_0,h_1)=f$ where f is the function obtained by Predicative Notational Recursion from g, h_0, and h_1.

Definition of BC^ω

> BC^ω is the class of functions obtained by lambda abstraction and application from 0, <u>mod</u>, s_0, s_1, # $|\cdot|$, and R.

Using the strong normalization theorem, the type 1 closed terms of BC^ω compute exactly the functions in $BC(|\cdot|)$.

It is common to assign types to functionals as follows: integers are type 0; functions with inputs of type α and output of type β are said to have type $\alpha\to\beta$. Any such type can be written uniquely as $\alpha_0\to(\alpha_1\cdots(\alpha_n\to 0)\cdots)$, which is abbreviated as $(\alpha_0,\ldots,\alpha_n,0)$ (see [CU]). In the two-sorted context, there are two base types, σ and τ, instead of just 0. The type of a function is *hereditarily normal* if it is type ν or if it is type $\mathbf{a}\to\mathbf{b}$ where $\mathbf{a}$ and $\mathbf{b}$ are hereditarily normal. We write hereditarily-normal inputs to the left of the semicolon, and all others to the right. The *genus* of ν and σ is 0, and the *genus* of $\mathbf{a} \to \mathbf{b}$ is $\mathbf{a}' \to \mathbf{b}'$ where $\mathbf{a}'$ is the genus of $\mathbf{a}$ and $\mathbf{b}'$ is the genus of $\mathbf{b}$.

In defining a function f using R, we informally refer directly to the equations for Predicative Notational Recursion rather than introduce extra symbols for g, h_0, and h_1.

8.4 Computing WQO.

Define the "well-quasi-ordering functional", L, as follows. $L(f)$ is the least i such that $f(i) \leq_2 f(i+1)$, where $<_2$ is the ordering

$$\langle a_0, b_0 \rangle \leq_2 \langle a_1, b_1 \rangle \Leftrightarrow (|a_0| < |a_1|) \vee (|a_0| = |a_1| \wedge |b_0| \leq |b_1|)$$

Here $\langle \cdot, \cdot \rangle$ is any polytime pairing function such that $|\langle a, b \rangle| \geq |a| + |b|$; define the projections Π_1 and Π_2 such that $\Pi_1\langle a, b \rangle = a$ and $\Pi_2\langle a, b \rangle = b$ for all a and b.

As discussed by Cook, [C2], the WQO functional L is not basic feasible (i.e. not denoted by a term in PV^ω). However, we can compute it in BC^ω with hereditarily normal type $(\nu \to \nu) \to \nu$.

By the results characterizing polytime with BC^ω, we have all polytime type 1 functions, such as Π_k^ν, s_1^ν, $>^\nu$, $\leq_2^\nu$, and $-^\nu$, with normal inputs and output. (In the following we omit the superscript ν). Recall we have string concatenation $\oplus$ with one safe input. The function $x \div y$ returns $\lfloor x/2^{|y|} \rfloor$.

$|E(f;a)|$ is the index z of the first position after position $|a|$, at which $|\Pi_1 f(z;)| > |\Pi_1 f(z+1;)|$ or $f(z;) \leq_2 f(z+1;)$. Note that this index z is at most $|s_1 \Pi_2 f(|a|;)|$ because there can be at most this many $(\Pi_2 f(|a|;))$ lengthwise-decreasing values appearing in the second coordinate after position $|a|$.

$$\begin{aligned}
\hat{E}(f, 0; a) &= 0 \\
\hat{E}(f, yi; a) &= \begin{cases} E(f, y; a) & \text{if } |E(f, y; a)| \neq 0 \\ & \text{else,} \\ a \oplus yi & \text{if } (\Pi_1 f(|a \oplus y|;)) \div \Pi_1 f(|a \oplus yi|;) \neq 0 \\ a \oplus yi & \text{if } f(|a \oplus y|;) \leq_2 f(|a \oplus yi|;) \\ 0 & \text{otherwise} \end{cases} \\
E(f; a) &= \hat{E}(f, s_1 \Pi_2 f(|a|;); a)
\end{aligned}$$

The uses of $|\cdot|$ above require the new initial function, because the input is safe yet the output is required to be normal. Note $|E(f;a)| \geq 1$ for all a.

$|\Sigma(f,x;)|$ is the position immediately after the end of the first $|x|$ rows of f, or else it is $L(f;)+1$, whichever is smaller:

$$\begin{aligned}
\Sigma(f,0;) &= 1 \\
\Sigma(f,xi;) &= \begin{cases} \Sigma(f,x;) & \text{if } f(|\Sigma(f,x) \dot{-} 1|;) \leq_2 f(|\Sigma(f,x)|;) \\ E(f;\Sigma(f,x)) & \text{otherwise} \end{cases}
\end{aligned}$$

Now we can compute $L(f;) = |\Sigma(f, \Pi_2 f(0;);)| - 1$.

8.5 Discussion

The choice of $\lambda a.|a|^\nu$ above seems somewhat arbitrary. For example we might think of strengthening the class further by using some other function $\lambda a. \upharpoonleft a \upharpoonright^\nu$ such that

1. $\upharpoonleft a \upharpoonright$ is polytime computable.
2. $\upharpoonleft a \upharpoonright = o(\sqrt[k]{a})$ for each constant k.
3. $||a||^k \leq |\upharpoonleft a \upharpoonright|$ for each constant k.
4. $\upharpoonleft a + b \upharpoonright \leq \upharpoonleft a \upharpoonright + \upharpoonleft b \upharpoonright$.
5. if there is a polynomial q such that $\upharpoonleft f(x) \upharpoonright \leq q(\upharpoonleft x \upharpoonright)$ then there is a polynomial q' such that $|f(x)| \leq q(|x|)$.

The second condition is required for the idea of the proof presented here. The third condition is appropriate since, already having $|a|^\nu$ in the class, we could use $\underline{\text{mod}}$ and smash to compute any type $\sigma \rightarrow \nu$ PTIME function f such that $|f(;a)| \leq ||a||^k$. The fourth condition is used in the case of composition.

The fifth condition, a "niceness" condition, derives from observing that the form of the induction hypothesis would be

$$\upharpoonleft f(\overline{x};\overline{a}) \upharpoonright \leq q_f(\upharpoonleft \overline{x} \upharpoonright, \upharpoonleft\upharpoonleft \overline{a} \upharpoonright\upharpoonright) + \max_i \upharpoonleft a_i \upharpoonright,$$

rather than

$$|f(\overline{x};\overline{a})| \leq q_f(|\overline{x}|, \lceil |\overline{a}| \rceil) + \max_i |a_i|$$

or

$$|f(\overline{x};\overline{a})| \leq q_f(|\overline{x}|, |\lceil \overline{a} \rceil|) + \max_i |a_i|.$$

The first statement of a bound on $|f(\overline{x};\overline{a})|$ fails for the base case (unless $q'(\lceil |a| \rceil) \geq |\lceil a \rceil|$ for some polynomial q'); the second fails in the analysis of recursion.

In obtaining a bound on $\lceil f(\overline{x};\overline{a}) \rceil$, the induction step for Predicative Notational Recursion would require a more careful definition of q_f^* to consider whether or not $\lceil xi \rceil = \lceil x \rceil$.

8.5.1 Questions and Comments

Question Is there a recursively presentable set S of functions of type $\sigma \to \nu$ such that the type $\sigma \to \nu$ functions of $BC(S)$ are exactly the functions satisfying conditions such as those above?

Question Let S be the largest class of functions of type $\sigma \to \nu$ satisfying conditions such as those above. How does $BC(S)$ compare to the class C_2 of Seth [S]?

Comment The class obtained when $\lambda a.|a|^\nu$ is *not* present seems to be equivalent to the BFF's. We have not explored this direction since there are already several characterizations of the BFF's, for example based on Cobham's class, and on Mehlhorn's polynomial class [M]; Also our main thesis of "resource free" computation is already well illustrated with respect to the more traditional type 1 classes.

Comment The diagonalization result of Seth [S] showing that there is a non-recursively-presentable class C_2 of total functions satisfying the desiderata of Cook [C2] can be taken as an indication that one of the following two situations holds: (1) There are further desirable properties to be required of a higher type class; or (2) further development of higher type computability should be in the direction of partial feasible functionals.

In the direction (2), a class of partial computable type 2 functionals has been defined by Seth. However, since the definition also involves a class of "small" functions like S above, and since this appears to be non-R.E., more work would be required for this direction to be fruitful. In particular we may note that Seth's class of partial functionals is "minimally partial" in the sense that the type 1 section consists only of total functions – a type 2 function is only partial if its computation runs forever but at each step has only used an amount of time polynomial in the (type 0 inputs and the) oracle results to date. This leaves open the possibility of a different class of functionals, which are "polytime" only when their output is defined.

In the direction (1), Seth recently claimed that the WQO functional, although it is in some sense intuitively feasible, does not preserve the class of Kalmar elementary functions; and he conjectures that it also does not preserve the primitive recursive functions. Therefore, together with Seth and Cook, I think we can add the following to the desiderata of [C2]:

> In order for Φ to be a class of feasible type 2 functionals, it must be that whenever Γ is a natural type 1 complexity class, then the type 1 functions denoted by lambda terms over $\Phi \cup \Gamma$ are equal to Γ.

Here a "natural type 1 complexity class" is a robust class arising from other complexity considerations; of course it must be closed under composition. We can think of "preserves type 1 polytime" and "preserves oracle polytime" requirements as special cases of this desiderata.

Whichever direction (1) or (2) is taken, the class presented here is not a final candidate for a feasible higher type class. It is thought of as simply a more intuitive route to computing the WQO than is offered by the addition of ad-hoc recursions to the BFF's as discussed in the correspondence between Cook, Takeuti, and Kapron.

Part II

Three Logical Systems

Chapter 9

Introduction to Part II

9.1 Discussion

The results of the first part of this thesis bear on the logic PV ([C], [CU]), an equational theory with a function symbol for each polytime function together with defining equations for the function based on Cobham's characterization. An alternative development of PV, based on the theorems of Part I rather than Cobham's theorem, should be simpler.

On the other hand, the results can also be used within the second-order logical system defined by Leivant [L] to simplify certain results.

Introducing such a function class into each of these two logical systems allows us to define and prove a direct correspondence between the two. Thus we are able to unite two lines of investigation into logic and feasibility which historically developed from separate lines of thought. This correspondence is not obvious since the rules of Leivant's system do not simulate the induction rule of PV in a direct way.

In the following chapters we develop three logics, $\mathrm{PV}_{\mathrm{BC}}$, QPV, and $L_2^{\mathrm{P}}(A)$. The first is a quantifier-free system which, like PV of [CU], arguably captures the notion of "feasible proof". The second is a quantified version of this, with a quantifier-free axiomatic basis. The third is a second-order system which is essentially a subsystem of Leivant's $L_2(QF^+)$ (see [L]) containing function symbols for the polytime functions.

We discuss the correspondence between $\mathrm{PV}_{\mathrm{BC}}$ and PV, and that QPV is conservative over $\mathrm{PV}_{\mathrm{BC}}$. We prove a "two sorted" version of a simplified witnessing theorem for QPV, using the Herbrand theorem. This gives a derived rule of notational induction for safe-existential formulas in QPV.

Next we map statements of QPV into statements of $L_2^{\mathrm{P}}(A)$ by relativizing quantifiers, and prove a direct connection between the 'feasible' logic QPV and Leivant's second-order provability characterization of polytime. This requires proving, in $L_2^{\mathrm{P}}(A)$, the relativized version of the QPV induction scheme.

In addition, we prove that $L_2^{\mathrm{P}}(A)$ proves convergence of all polytime functions. This proof, which could be carried out in $L_2(QF^+)$ where function symbols are available for non-polytime functions, is an elegant proof of one direction of Leivant's result.

In the Appendix we demonstrate how to use the systems $\mathrm{PV}_{\mathrm{BC}}$ and QPV by proving some elementary facts.

I would like to acknowledge Stephen Cook for laying the foundations of this part, and particularly for the use of the Herbrand Theorem.

9.2 Background

One of the two lines of research into feasible logics examined below, that of bounded arithmetic, is exemplified in the work of Paris and Wilkie [PW], Buss [B], [B3]; Cook and Urquhart [CU] and, earlier, the propositional work of Cook and Reckhow [CR]. Further work in this area is by Krajicek, Pudlak, and Takeuti, for example [KPT], [KT]; by Allen [Al]; Clote [CT]; and Pitassi [PBI], [PU]. See also Ferreira [Fe] for earlier work in this area. We specifically follow the work of Cook and Urquhart [CU].

The other line of research into feasibility and logic explored here is essentially the project of of Leivant, [L], [L3], [L4] in which comprehension is identified as the source both of computational complexity and impredicativity.[1] The philosophical motivation for this line of research is found in Nelson's predicative arithmetic [N], and before that in the ramified analysis of Russell, Weyl, Spector, and many others; see Feferman [F] for a discussion. These earlier works differ from Leivant's in being concerned only with the basic

[1] The axiom scheme of comprehension is $\exists R \forall u. R(u) \leftrightarrow \gamma[u, \overline{v}, \overline{Q}]$; it allows you to define a set R from a formula γ. In a second-order logic we may, for example, try to prove that R is closed, or use it to instantiate a universally quantified second-order variable.

issue of constructibility (computability) rather than with computational complexity. Explaining the intuition of why increased comprehension leads to increased computational complexity of the provable functions, Ignjatovic commented: in the presence of lots of comprehension, we can express more computationally complex functions using logical formulas; this lets you penetrate deeper into computations with your logic.[2]

Writing $L_2(C)$ for Leivant's second order logic with comprehension allowed over formulas in the class C, characterizations by Leivant include the following. Here "provable" is defined as $W(x) \rightarrow W(f(x))$; see the detailed definitions below.

- $f \in$ PSPACE $\Leftrightarrow$ f is provable in L_2(positive universal formulas).
- $f \in$ PTIME $\Leftrightarrow$ f is provable in L_2(positive existential formulas) = provable in L_2(positive quantifier-free formulas).
- $f \in \mathcal{E}^3$ (Kalmar Elementary) f is provable in L_2(first order formulas).

Following the line of Leivant, Ignjatovic [Ig] defined the system $BASIC(\Sigma_i^b)$ and proved that a function f is in $\square_i^p$ (level i of the polytime function hierarchy) if and only if it is provable in $BASIC(\Sigma_i^b)$. We refer to the model-theoretic ideas of Ignjatovic [Ig] in one of the proofs below.

The presence of the surjectivity principle in these second order systems is a binary-successors analogue to starting with Robinson's small subsystem Q of Peano Arithmetic [Ro]; as discussed by Nelson [N], this system has no induction rule. Buss comments [B, p. 210] that Wilkie and Nelson independently proved that the bounded arithmetic systems S_2^i are predicative in the sense that they can be interpreted in Q.

The work of Leivant shares a general idea with much work carried out in bounded arithmetic: that it is useful to characterize a function class by giving a logic whose provably total functions are exactly those in the class. These bounded arithmetics are usually first order systems whose induction axiom has been restricted to apply only to formulas of a certain form. For example, we can list the following correspondences, using "f is provable" to mean that $\forall x \exists y . y = f(x)$ is provable:

- f provable in Peano Arithmetic $\Leftrightarrow$ f defined by a λ-term built from 0, +1, and Gödel's higher type primitive recursion (see [Gi]).

[2] Paraphrased from his talk at the Workshop on Feasible Mathematics II, Cornell University, May 1992.

- f provable in Peano Arithmetic using induction on existential formulas $\Leftrightarrow$ f is primitive recursive (see [LM], [CT], for references).

- f is provable in $I\Delta_0 \cup \{\forall x \exists y. x^{|x|} = y\}$ $\Leftrightarrow$ f is in $\cup_i \Box_i^p$ ([B], cited in [CT]).

- $\forall x \exists y. \chi_f(x, y)$ is provable in IS_2^i $\Leftrightarrow$ f is in $\Box_i^p$ ([B]); in particular, provability in IPV corresponds to polytime (see [CU]).[3]

- f is provable in $I\Delta_0 \cup \{\forall x \exists y. 2^x = y\}$ $\Leftrightarrow$ f is Kalmar elementary (folklore, noted in [CT]).

Although Buss [B] does consider weak second-order systems, the development is in a different direction from that of Leivant; the induction axiom of S_2^i is retained and the primary focus is definability of functions.

At a more general level, the idea of introducing feasibility considerations into logic is a radical extension of the program of constructive mathematics [H], [Bi], [TD]. Nelson's work itself is in the "nominalist" philosophy which is highly formalist and, while not finitist, is much stricter than the intuitionistic view. For a general background in logic one can refer to [GG] or [BM].

[3] We use the graph of the function here because these systems only have a restricted set of function symbols.

Chapter 10

The system $\mathrm{PV}_{\mathrm{BC}}$

10.1 Defining $\mathrm{PV}_{\mathrm{BC}}$

In this section, a quantifier-free classical logical system, $\mathrm{PV}_{\mathrm{BC}}$, is developed using the class BC over the two types ν (normal) and σ (safe). This system is a two-sorted analogue of the system PV developed by Cook [C77] and Cook & Urquhart [CU].

Terms and function symbols of $\mathrm{PV}_{\mathrm{BC}}$

Assume that $\overline{\alpha}$, β are type symbols in $\{\nu, \sigma\}$.

S1. There are infinitely many variables of types ν and σ, and each variable is a term (of its given type).

S2. The constants 0^σ and 0^ν are terms (of type σ and ν respectively).

S3. If f is a function symbol (of type $(\overline{\alpha}, \beta)$) and $t_1 \ldots t_n$ are terms (of types $\overline{\alpha}$) then $f(t_1, \ldots, t_n)$ is a term (of type β).

S4. s_0^σ and s_1^σ are function symbols (of type $\sigma \to \sigma$).

S5. $\underline{\mathrm{mod}}^\nu$ and EQ^ν are function symbols (of types (ν, σ, ν) and (σ, σ, ν), respectively.

S6. If t is a term (of type β) and $\overline{x}, \overline{a}$ is a list including all the variables in t (types $\overline{\nu}$ and $\overline{\sigma}$), then $[\lambda \overline{x}, \overline{a}.(t)]^\beta$ is a function symbol (of type $(\overline{\nu}, \overline{\sigma}, \beta)$.)

S7. If g, h_0 and h_1 are function symbols (of types $(\overline{\alpha}, \sigma)$, $(\nu, \overline{\alpha}, \sigma, \sigma)$, and $(\nu, \overline{\alpha}, \sigma, \sigma)$ respectively) then $\mathbf{R}[g, h_0, h_1]^\sigma$ is a function symbol (of type $(\nu, \overline{\alpha}, \sigma)$.)

S8. If f^ν is a function symbol of type $(\overline{\alpha}, \nu)$ then $(f^\nu)^\sigma$ is a function symbol of type $(\overline{\alpha}, \sigma)$.

S9. If f^σ is a function symbol of type $(\overline{\nu}, \sigma)$ then $(f^\sigma)^\nu$ is a function symbol of type $(\overline{\nu}, \nu)$.

The function symbols defined above are in exact correspondence with the functions of BC, although we have given a slightly different definition. Notice that in the last two rules above, the "raising" of f^σ to obtain f^ν requires that f have all normal-type inputs, whereas the "lowering" of f^ν to obtain f^σ can occur regardless of the input types.

We omit the superscript σ or ν when it can be inferred from the context. When a type ν term t is used in a context requiring a type σ value, we mean to use the construction $[\lambda x^\nu.(x)]^\sigma(t;)$. This is an equivalent construction, by axiom A5 below. As in B we write xi for $s_i(;x)$ and use ";" to separate safe inputs from normal inputs. We can assume the existence of successor functions with normal inputs and outputs, using the construction $s_i^\nu(;x)$.

The *formulas* of $\mathrm{PV_{BC}}$ are the formulas constructed using the predicate "=" and logical connectives $\wedge$, $\vee$, $\neg$, $\rightarrow$ in the usual way. Note that = is extensional in the sense that $t_1 = t_2$ can be asserted even if t_1 and t_2 have different sorts. For example, the transitivity of equality will include $x = a \wedge a = y \rightarrow x = y$ for x, y normal and a safe. On the other hand, whenever $f(\overline{t})$ appears for function symbol f and terms $\overline{t}$, the sorts of the terms $\overline{t}$ must match the sorts of the inputs to f.

Axioms A1-A5 of $\mathrm{PV_{BC}}$

A1 All axioms of the classical quantifier-free predicate calculus with the sole predicate being equality ("="), with axioms for substitutivity and equality.

A2 The axioms

$$\begin{array}{ll} s_0(0;) = 0, & 0 \neq s_1(0;) \\ a \,\underline{\mathrm{mod}}\, 0 = 0, & x \neq 0 \rightarrow a \,\underline{\mathrm{mod}}\, x0 = a \,\underline{\mathrm{mod}}\, x1 \\ a0 \,\underline{\mathrm{mod}}\, x1 = (a \,\underline{\mathrm{mod}}\, x)0, & (a1 \,\underline{\mathrm{mod}}\, u1) = (a \,\underline{\mathrm{mod}}\, x)1 \\ a = b \rightarrow \mathrm{EQ}(;a,b) = 0 & a \neq b \rightarrow \mathrm{EQ}(;a,b) = 1 \end{array}$$

A3 For function symbol f^σ, the axiom $(f^\sigma)^\nu(\overline{x};) = f^\sigma(\overline{x};)$. For function symbol f^ν, the axiom $(f^\nu)^\sigma(\overline{x};\overline{a}) = f^\nu(\overline{x};\overline{a})$.

A4 For function symbol $[\lambda\overline{x},\overline{a}.(t)]$, the axiom $[\lambda\overline{x},\overline{a}.(t)](\overline{x};\overline{a}) = t$.

A5 For function symbol $\mathbf{R}[g, h_0, h_1]$, the axioms

$$\begin{aligned} & \mathbf{R}[g,h_0,h_1](0,\overline{y};\overline{a}) = g(\overline{y},\overline{a}) \\ & \mathbf{R}[g,h_0,h_1](x1,\overline{y};\overline{a}) = h_1(x,\overline{y};\overline{a},\mathbf{R}[g,h_0,h_1](x,\overline{y};\overline{a})) \\ x \neq 0 \rightarrow \quad & \mathbf{R}[g,h_0,h_1](x0,\overline{y};\overline{a}) = h_0(x,\overline{y};\overline{a},\mathbf{R}[g,h_0,h_1](x,\overline{y};\overline{a})) \end{aligned}$$

In the following we use infix notation $a \,\underline{\mathrm{mod}}\, x$ for $\underline{\mathrm{mod}}(x; a)$, and $a \,\mathrm{EQ}\, b$ for $\mathrm{EQ}(; a, b)$.

In keeping with the philosophy of the earlier development of the functional polytime hierarchy, we would like all of the initial functions defined above to be computable by performing a constant-time modification on one of the safe inputs. We do not achieve this here, because we are using a function class formulated with <u>mod</u> (this also requires EQ as primitive). See the end of this Part of the thesis for a discussion of how the use of <u>mod</u> and EQ could be eliminated.

In order to simplify the proof that $L_2^{\mathrm{P}}(A)$ proves convergence of all polytime functions, we have kept the functions computed on safe inputs as simple as possible. In particular, we have omitted the $+1$ and predecessor functions on safe input. Considering $\mathrm{PV}_{\mathrm{BC}}$ in isolation, it seems that adding $p(; a)$ would be a good idea, however this would substantially complicate later proofs concerning $L_2^{\mathrm{P}}(A)$, and we are able to do without it.

Inference rules of $\mathrm{PV}_{\mathrm{BC}}$

R1 (Logical Rules) All rules of the classical quantifier-free predicate calculus with the sole predicate being equality ("="), and with rules for substitutivity and equality.

R2 (Nonlogical Rule) The rule of *induction on normal variables*,

$$A(0) \wedge (A(x) \rightarrow A(x0)) \wedge (A(x) \rightarrow A(x1)) \vdash_{\mathrm{PV}_{\mathrm{BC}}} A(x),$$

where x is type ν.

Applying the axiom $s_0(0) = 0$ and $x = 0 \vee x \neq 0$, we can harmlessly write the step condition as

$$A(x) \rightarrow A(x1) \wedge (x \neq 0 \rightarrow (A(x) \rightarrow A(x0))$$

instead of what is written above. Since $x0 \neq 0 \rightarrow x \neq 0$ and $x1 \neq 0$ is easily proved by induction on normal x, we can abbreviate the induction condition as

$$\bigwedge_{i\in\{0,1\}} xi \neq 0 \rightarrow (A(x) \rightarrow A(xi)).$$

For the rules and axioms of the classical quantifier-free predicate calculus we can use the rule of propositional implication and the rule of instantiation defined as follows. For formula α, write α^P for the proposition obtained by uniformly replacing atomic formulas of α with propositional letters.

(Propositional Implication) If $\overline{\alpha}^P \models \beta^P$ and $\vdash \alpha$, then conclude β. Here $\models$ is propositional entailment.

(Instantiation) If $\vdash \alpha[\overline{x};\overline{a}]$, then $\vdash \alpha[\overline{r};\overline{t}]$ for any terms $\overline{r}$ and $\overline{t}$ of the appropriate sort.

10.1.1 $\mathrm{PV}_{\mathrm{BC}}$ preliminaries

We frequently introduce function symbols as in $S7$ and simultaneously state the relevant predicative recursion axioms directly as written in $A6$, avoiding the cumbersome notation $\mathbf{R}[g, h_0, h_1]$. For example, introduce the function symbol p with the axioms

$$\begin{array}{rll} & p(0;) & = 0 \\ & p(x1;) & = x \\ x \neq 0 \rightarrow & p(x0;) & = x. \end{array}$$

Surjectivity in $\mathrm{PV}_{\mathrm{BC}}$

$\vdash_{\mathrm{PV}_{\mathrm{BC}}} x = 0 \lor x = p(x;)0 \lor x = p(x;)1.$

Proof. By induction on normal variable x, the two induction steps being

$$(x = 0 \lor x = p(x;)0 \lor x = p(x;)1) \rightarrow (xi = 0 \lor xi = p(xi;)0 \lor xi = p(xi;)1)$$

for $i \in \{0,1\}$. The induction steps are easily proved from the axioms for p and A2. ∎

Comment. Using this Fact, it is convenient to write the axioms of a function f

introduced using **R** in the form

$$\begin{aligned} f(0,\overline{y};\overline{a}) &= g(\overline{y};\overline{a}) \\ f(xi,\overline{y};\overline{a}) &= h_i(x,\overline{y};\overline{a},f(x,\overline{y};\overline{a})) \quad \text{for } xi \neq 0 \end{aligned}$$

and analyse $f(z,\overline{y};\overline{a})$ in $\mathrm{PV_{BC}}$ by an induction on z, considering the classical cases $z = 0$ and $z = s_i p(z;) \neq 0$, where i is either 0 or 1.

The conditional

Define the conditional

$$\begin{aligned} \mathrm{cond}(0;b,c) &= b \\ \mathrm{cond}(x1;b,c) &= c \\ x \neq 0 \to \mathrm{cond}(x0;b,c) &= \mathrm{cond}(x;b,c) \\ C(;a,b,c) &= \mathrm{cond}(a \,\underline{\mathrm{mod}}\, 1;b,c) \end{aligned}$$

We have that $\vdash_{\mathrm{PV_{BC}}} (z = 0 \to \mathrm{cond}(z;b,c) = b) \wedge (z \neq 0 \to \mathrm{cond}(z;b,c) = c)$ by an easy proof inducing on z. Replacing z with $a \,\underline{\mathrm{mod}}\, 1$ gives the fact

$$\vdash_{\mathrm{PV_{BC}}} (a \,\underline{\mathrm{mod}}\, 1 = 0 \to C(;a,b,c) = b) \wedge (a \,\underline{\mathrm{mod}}\, 1 \neq 0 \to C(;a,b,c) = c).$$

In particular, by classical reasoning we have $C(;a,b,c) = b \vee C(;a,b,c) = c$; and using A2 we have $C(;0,b,c) = b$ and $C(;1,b,c) = c$.

Drop ($\div$) function

We can also introduce the 'drop' function on normal inputs by

$$\begin{aligned} x \div 0 &= x \\ x \div yj &= p((x \div y) \,\underline{\mathrm{mod}}\, x;) \quad \text{for } yj \neq 0 \end{aligned}$$

Here we are using $\underline{\mathrm{mod}}$ to change the type of $x \div y$ from safe to normal. Facts about $\div$, such as $y \div x = 0 \leftrightarrow y = y \,\underline{\mathrm{mod}}\, x$, are proved in the Appendix.

10.2 Equivalence of $\mathrm{PV_{BC}}$ and PV.

The system $\mathrm{PV_{BC}}$ defined above is equal in power to the system PV discussed by Cook & Urquhart [CU]. That is, whenever α is a formula in the language of PV, there is a formula α' of $\mathrm{PV_{BC}}$ with all normal variables, such that PV proves α if and only if $\mathrm{PV_{BC}}$ proves α'.

To state the first theorem properly we need to define for each function symbol f of PV a corresponding $\mathrm{PV_{BC}}$ function symbol f' with both normal inputs and normal outputs. For the initial functions of PV (namely, $\lfloor \frac{1}{2}x \rfloor$, cond, $\dot{-}$, $\boxplus$, PARITY, and #) this is easily done using predicative recursion. (In the case of # an auxiliary function symbol is used; it is o.k. that this involves safe values provided the final result has only normal inputs). The slightly harder case is function symbols defined in PV using the construction $R[g,h,k]$; in PV the corresponding axiom is [CU, p. 20]

$$\begin{aligned} R[g,h,k](x,\overline{y}) \quad &= \mathrm{cond}(x, g(\overline{y}), \mathrm{cond}(t \dot{-} k(x,\overline{y}), t, k(x,\overline{y}))) \\ \text{where } t \quad &\equiv h(x, \overline{y}, R[g,h,k](\lfloor \tfrac{1}{2}x \rfloor, \overline{y})) \end{aligned}$$

Definition of mapping from PV to $\mathrm{PV_{BC}}$

Given hereditarily normal g', h', and k' in $\mathrm{PV_{BC}}$ corresponding to functions g, h, and k in PV, define f' corresponding to the PV function $R[g,h,k]$ by using **R** to obtain the following predicative recursion axioms:

$$\begin{aligned} f'(0,\overline{y};) \quad &= g'(\overline{y}) \\ f'(xi,\overline{y};) \quad &= C(; t \,\mathrm{EQ}\, t \,\underline{\mathrm{mod}}\, k'(xi,\overline{y};),\ t,\ k'(xi,\overline{y};)) \\ \text{where } t \quad &\equiv h'(xi, \overline{y}, f(x,\overline{y};) \,\underline{\mathrm{mod}}\, k'(xi,\overline{y};)) \end{aligned}$$

Now given formula α in PV we obtain α' in $\mathrm{PV_{BC}}$ by replacing each function symbol f in α with f', and making all free variables normal.

Lemma: $\mathrm{PV_{BC}}$ **is as strong as** PV

$\vdash_{PV} \alpha \Rightarrow \vdash_{\mathrm{PV_{BC}}} \alpha'$

Proof. Supposing $\vdash_{PV} \alpha$, we proceed by induction on the length of the proof of α. First, the axioms concerning the initial functions of PV are easily proved in $\mathrm{PV_{BC}}$ from the axioms of the corresponding functions in $\mathrm{PV_{BC}}$. For example, we have $\vdash_{\mathrm{PV_{BC}}}$ $\mathrm{cond}(s_0(x), y, z;) = \mathrm{cond}(x, y, z;)$ by the equations for cond in $\mathrm{PV_{BC}}$.

Secondly, if f was defined by $R[g,h,k]$ in PV, we need to prove the image of the PV axiom:

$$\begin{aligned} f'(x,\overline{y}) \quad &= \mathrm{cond}(x, g'(\overline{y};), \mathrm{cond}(t \dot{-} k'(x,\overline{y};), t, k'(x,\overline{y};))) \\ \text{where } t \quad &\equiv h'(x, \overline{y}, f'(p(x;), \overline{y};);) \end{aligned}$$

Working in $\mathrm{PV_{BC}}$ we use the definition of cond and consider cases $x = 0$ or $x \neq 0$. If $x = 0$ the result follows by the $\mathrm{PV_{BC}}$ axiom $f'(0,\overline{y};) = g'(\overline{y};)$. If $x \neq 0$ the result follows from the fact

$$\begin{aligned} f'(x,\overline{y};) &= C(;t \,\mathrm{EQ}\, t \,\underline{\mathrm{mod}}\, k'(x,\overline{y};),\ t,\ k'(x,\overline{y};)) \\ \text{where } t &\equiv h'(x,\overline{y},f'(p(x;),\overline{y};) \,\underline{\mathrm{mod}}\, k'(x,\overline{y};)) \end{aligned}$$

(the axiom for f' with $p(x;)$ substituted for x). To get the equality we use the axiom of EQ and the fact $t \dot{-} k'(x,\overline{y}) = 0 \leftrightarrow t = t \,\underline{\mathrm{mod}}\, k'(x,\overline{y})$ proved in lemma ($2\,\underline{\mathrm{mod}}$) of the Appendix.

Finally, if α follows by one of the inference rules (equality rules, induction rule) of PV then we use the corresponding rule of $\mathrm{PV_{BC}}$; for example, the image of the induction rule $R5$ of PV is an easy consequence of Induction On Normal Variables in $\mathrm{PV_{BC}}$. ∎

For the other direction, we set up a reverse mapping from function symbols f in $\mathrm{PV_{BC}}$ to function symbols f'' in PV. In performing this mapping we omit the sort information from the variables in $\mathrm{PV_{BC}}$.

A thornier problem is presented by function symbols defined with predicative recursion (**R** in $\mathrm{PV_{BC}}$). In PV we have to explicitly obtain the bounding term in order to define these functions using R. This issue cannot be evaded since even if the $\mathrm{PV_{BC}}$ proof is a proof of α' for some α in the language of PV, it might be the case that intermediary steps in the proof of α' use new function symbols defined with **R**. The solution requires formalizing in PV the Bounding Lemma for BC.

Definition of mapping from $\mathrm{PV_{BC}}$ to PV

For each initial function f of $\mathrm{PV_{BC}}$, let f'' be a PV function symbol which computes the same function (ignoring sort information). For f defined by safe composition in $\mathrm{PV_{BC}}$, f'' is defined by the corresponding composition in PV.

Given g'', h''_0, and h''_1 in PV corresponding to the function symbols g, h_0 and h_1 of $\mathrm{PV_{BC}}$, define f'' in PV using R as follows:

$$\begin{aligned} f''(x,\overline{y},\overline{a}) &= \mathrm{cond}(x, \\ &\qquad g''(\overline{y},\overline{a}), \\ &\qquad \mathrm{cond}(t \dot{-} k(x,\overline{y},\overline{a}), t, k(x,\overline{y},\overline{a}))) \\ \text{where } t &\equiv \mathrm{cond}(\mathrm{PARITY}(x), \\ &\qquad h''_0(\lfloor \tfrac{1}{2}x \rfloor, \overline{y}, \overline{a}, f''(\lfloor \tfrac{1}{2}x \rfloor, \overline{y}, \overline{a})), \\ &\qquad h''_1(\lfloor \tfrac{1}{2}x \rfloor, \overline{y}, \overline{a}, f''(\lfloor \tfrac{1}{2}x \rfloor, \overline{y}, \overline{a}))) \end{aligned}$$

and where $k(x,\overline{y},\overline{a})$ is a function such that $|k(x,\overline{y},\overline{a})| = q_f(|x|,|\overline{y}|) + \max_i |a_i|$ for the polynomial q_f defined in the Bounding Lemma for BC.

The PV image α'' of a $\mathrm{PV}_{\mathrm{BC}}$ formula α is obtained by replacing each function symbol f with its image f''.

The computations required for the initial functions are easily carried out using the results of [CU]; for example, we can use multi-variable recursion to define $\underline{\mathrm{mod}}$ and then use multi-variable induction to prove facts about it.

Lemma: PV is as strong as $\mathrm{PV}_{\mathrm{BC}}$

$$\vdash_{\mathrm{PV}_{\mathrm{BC}}} \alpha \Rightarrow \vdash_{PV} \alpha''$$

Proof. As discussed above, the initial functions present no difficulty since the development in [CU] is quite extensive. The deduction rules of $\mathrm{PV}_{\mathrm{BC}}$ are also unproblematic since we have equality and induction rules in PV. To prove A5 when f is defined by $\mathbf{R}$, the essential need is to prove in PV the bound $|f(x, \overline{y}, \overline{a})| \leq |k(x, \overline{y}, \overline{a})|$ in order to conclude that $t \dot{-} k(x, \overline{y}, \overline{a}) = 0$. The proof of this bound in PV is a direct formalization of the proof of the Bounding Lemma; it is an induction on x using simple arithmetic facts. ∎

The theorems above do not say much by themselves: as far as the statements of the theorems are concerned, it could be that the images α' and β'' are trivially true sentences irrespective of α and β. Of course this is not the case since $\models \alpha$ iff $\models \alpha'$, and $\models \beta$ iff $\models \beta''$.

To formalize this statement we would need to discuss models for the two-sorted system $\mathrm{PV}_{\mathrm{BC}}$. We could use the ordinary notion of models for a two-sorted logic, assuming a normal interpretation for $=$ in the sense that $a = x$ asserts that the safe object denoted by a is identically the normal object denoted by x. Since we have $(x^\nu)^\sigma = x^\nu$ as an axiom for each normal variable x^ν, it must be the case that every normal model for $\mathrm{PV}_{\mathrm{BC}}$ has sets N and S interpreting the normal and safe sorts such that $N \subseteq S$. A standard $\mathrm{PV}_{\mathrm{BC}}$ model is one in which both sorts are interpreted by the non-negative integers.

Rather than continue model-theoretically, it is perhaps clearer to give a proof-theoretic statement.

Correctness of round-trip translations

Let α be a formula of PV. Then

$$\vdash_{PV} \alpha \leftrightarrow (\alpha')''.$$

Let β be a formula of $\mathrm{PV_{BC}}$. Then

$$\vdash_{\mathrm{PV_{BC}}} \beta \leftrightarrow (\beta'')'.$$

The proofs, which are left as an exercise, essentially consist of checking that the mappings $'$ and $''$ do what they seem to do. For example, considering $R[g,h,k]$ in PV one may write down the PV definitions of $\textsc{eq}''$, C'' and $\underline{\mathrm{mod}}''$ corresponding to the $\mathrm{PV_{BC}}$ functions $\textsc{eq}$, C and $\underline{\mathrm{mod}}$, and then verify that PV proves

$$x \neq 0 \rightarrow \mathrm{cond}(x,\, g(\overline{y}),\, \mathrm{cond}(t \dot{-} k(x,\overline{y}),\, t,\, k(x,\overline{y}))) = C(t'' \overset{''}{\textsc{eq}}\, t'' \overset{''}{\underline{\mathrm{mod}}}\, k(x,\overline{y}),\, t'',\, k(x,\overline{y}))$$

$$\begin{aligned} \text{where} \quad t &\equiv h(x,\overline{y},r) \\ t'' &\equiv h(x,\overline{y},r)\, \underline{\mathrm{mod}}''\, k(x,\overline{y}) \end{aligned}$$

(Here r is a free variable replacing the recursion term $R[g,h,k](\lfloor \frac{1}{2}x \rfloor, \overline{y})$. We have assumed by the induction hypothesis that $(k'(x,\overline{y}))'' = k(x,\overline{y})$ and $(h'(x,\overline{y},r))'' = h(x,\overline{y},r)$ for all $x, \overline{y}, r$.)

Chapter 11

The system QPV

11.1 The system QPV.

Here we define a quantified version of $\mathrm{PV_{BC}}$. The purpose of this conservative extension is to provide a framework for discussing induction rules other than the quantifier-free induction of $\mathrm{PV_{BC}}$, and to provide a system which can be mapped into a Leivant-style second order system.

Rather than repeat the rules of $\mathrm{PV_{BC}}$ and include an axiom of induction, we prefer to start with all the theorems of $\mathrm{PV_{BC}}$. This allows a quantifier-free axiomatization, which is needed for the Herbrand argument below.

Definition of QPV

- QPV1. The function symbols and terms of QPV are the function symbols and terms of $\mathrm{PV_{BC}}$. The formulas of QPV are built up in the usual way from $\vee$, $\neg$, $\forall$, (, and).
- QPV2. The logical axioms and rules of QPV are those of the two-sorted predicate logic with equality between sorts.
- QPV3. The nonlogical axioms of QPV are all the theorems of $\mathrm{PV_{BC}}$.
- QPV4. There are no nonlogical inference rules.

When we write $\forall\mathrm{QPV} \vdash \alpha$ and $\forall\mathrm{QPV} \models \alpha$ we mean it in the ordinary sense of deduction and entailment in the two-sorted predicate calculus, from the set of formulas $\forall\mathrm{QPV} = \{$

universal closure of axioms of QPV }. This is identical to the notion of proof in the system QPV. When necessary we use the symbol $\vdash_{\sigma\nu}$ to indicate proof in the two sorted first order logic with equality.

Models for QPV are ordinary two-sorted models with the additional requirement of normal equality between sorts, similar to the models described above for $\mathrm{PV}_{\mathrm{BC}}$. In particular, a model satisfying the axioms for π will be a model in which the domain N of the normal sort is contained in the domain S of the safe sort.

The Henkin construction showing the completeness of the predicate calculus (by adding a witnessing constant for each existential formula; see Bell and Machover [BM] p. 117ff) can be carried out with trivial modifications for the two-sorted calculus. This gives us the following theorem.

Soundness and Completeness of QPV

$$\forall \mathrm{QPV} \models A \Leftrightarrow \forall \mathrm{QPV} \vdash A.$$

As usual, this statement is relative to the expressiveness of the system.

The line of reasoning used in proving completeness also gives the Compactness theorem for the two-sorted predicate logic, which gives the following:

Herbrand's Theorem

> Let Φ be a set of quantifier-free formulas of QPV. Then the universal closure $\forall\Phi$ is unsatisfiable if and only if some finite set of instances $\phi[t_1/x_1, \ldots, t_n/a_n]$ is truth-functionally unsatisfiable; where $\phi \in \Phi$ and $\phi[t_1/x_1, \ldots, t_n/a_n]$ has no free variables.

Proof. (1) Any assignment satisfying $\forall\Phi$ also satisfies all ground instances, hence all finite sets of ground instances.

(2) If all finite sets of ground instances are satisfiable then by compactness there is an assignment satisfying all ground instances simultaneously. The ground instances contain neither free nor quantified variables, and therefore we can identify the universe of the assignment with (a subset of) the terms appearing in the ground instances; in other words, we can assume w.l.o.g. that the universe of the assignment is the Herbrand term universe. Then the universal formulas Φ are satisfied by the assignment since every binding of the universally quantified variables corresponds to a ground instance. ∎

The fact of QPV being a conservative extension of PV_{BC} is easy.

Conservativity of QPV

QPV is a conservative extension of PV_{BC}.

Proof. We have used the theorems of PV_{BC} as the axioms of QPV, and the predicate logic used as a basis for QPV is conservative over the quantifier-free logic used as a basis for PV_{BC}. ∎

Notice that the axioms of QPV can all be stated without quantifiers, i.e. as universal formulas. In particular the axiom for induction on normal variables is conspicuously absent. However, it is a theorem of QPV.

Induction Theorem in QPV

For any quantifier free formula α containing free normal variable x,

$$\forall \mathrm{QPV} \vdash (\alpha[0, \overline{y}; \overline{a}] \wedge \forall x.\alpha[x, \overline{y}; \overline{a}] \rightarrow (\alpha[x0, \overline{y}; \overline{a}] \wedge \alpha[x1, \overline{y}; \overline{a}])) \rightarrow \forall x.\alpha[x, \overline{y}; \overline{a}].$$

Proof. Although the corresponding induction *rule* is a rule of PV_{BC}, this does not help us directly.[1]

Instead, using the propositional terms defined in the Appendix together with the function EQ, let χ_α be a function symbol such that $\vdash_{PV_{BC}} (\chi_\alpha(x, \overline{y}; \overline{a}) = 0) \leftrightarrow \alpha[x, \overline{y}; \overline{a}]$. For notational convenience we suppress $\overline{y}$ and $\overline{a}$ in the following. We use χ_α to define the following function $S(x, \overline{y}; \overline{a})$ in PV_{BC}:

$$\begin{aligned} S(0) &= \quad 0 \\ xi \neq 0 \rightarrow S(xi) &= \quad \begin{cases} x & \text{if } \alpha[x] \wedge \neg\alpha[xi] \\ S(x) & \text{otherwise} \end{cases} \end{aligned}$$

Now using the induction rule in PV_{BC} we prove that

$$\vdash_{PV_{BC}} (\alpha[0] \wedge \neg\alpha[x]) \rightarrow (\alpha[S(x)] \wedge \neg\alpha[S(x)0]) \vee (\alpha[S(x)] \wedge \neg\alpha[S(x)1]).$$

[1] The deduction theorem does hold in QPV, because proof in QPV is the same as proof in the ordinary two-sorted predicate calculus. Further, if it happens that $\vdash_{PV_{BC}} \alpha[0]$ and $\vdash_{PV_{BC}} \alpha[x] \rightarrow (\alpha[x0] \wedge \alpha[x1])$, then $\alpha[x]$ is an axiom of QPV and therefore $\forall \mathrm{QPV} \vdash \gamma \rightarrow \forall x.\alpha[x]$ for any formula γ, implying the statement of the Lemma for this particular α. However, if $\alpha[x]$ happens to not be provable in PV_{BC} then by the conservativity of QPV over PV_{BC} it is not provable in QPV. In this case it is sufficient to show the negation of $\alpha[0] \wedge \forall x.(\alpha[x] \rightarrow \alpha[x0] \wedge \alpha[x1])$ in order to conclude the induction statement. We do this by constructing the witnessing function S.

The base case $x = 0$ is trivial since the antecedent is classically false in that case. To prove the induction step $A(x) \rightarrow A(xi)$ where $i \in \{0,1\}$ and A is the statement above, also assume $xi \neq 0$ since otherwise $x0 = 0 = x$ and the step is trivial. (Note $x0 = 0 \rightarrow x = 0$ using the QPV axiom $p(;0) = 0$ and $(1p)$ from the Appendix.)

Assume the antecedents $A(x)$ and $(\alpha[0] \wedge \neg\alpha[xi])$. If $\neg\alpha[x]$ then $S(xi) = S(x)$ and applying the induction hypothesis $A(x)$ finishes it. Otherwise $\alpha[x]$ holds; we already have $\neg\alpha[xi]$, so $S(xi) = x$. Then the required conclusion $(\alpha[S(xi)] \wedge \neg\alpha[S(xi)i])$ is immediate.

From the statement $A(x)$ and the antecedents

$$(\alpha(0) \wedge \forall x.\alpha[x] \rightarrow (\alpha[x0] \wedge \alpha[x1]))$$

we can quickly conclude $\forall x.\alpha[x]$ as follows: if $\exists x.\neg\alpha[x]$ then introduce $S(x)$ to contradict $\alpha[S(x)] \rightarrow (\alpha[S(x)1] \wedge \alpha[S(x)0])$ using $A(x)$.

This proves the Lemma. ∎

Statman is reported to have used a similar type of argument, but with binary search rather than linear search, to give a feasible proof of the ordinary induction scheme,

$$[\alpha[0] \wedge \forall x.\alpha[x] \rightarrow \alpha[x+1]] \rightarrow \forall x.\alpha[x].$$

Cook carried out this proof for QPV.

Alternate Characterization of QPV

> The system obtained by replacing (QPV3) with (QPV3*) below is identical to QPV.
>
> QPV3*. The nonlogical axioms of QPV are the axioms A2-A5 of $\mathrm{PV}_{\mathrm{BC}}$ together with, for each quantifier free formula α containing free normal variable x, the axiom of induction on normal variables: $(\alpha[0,\overline{y};\overline{a}] \wedge \forall x.\alpha[x,\overline{y};\overline{a}] \rightarrow (\alpha[x0,\overline{y};\overline{a}] \wedge \alpha[x1,\overline{y};\overline{a}])) \rightarrow \forall x.\alpha[x,\overline{y};\overline{a}]$.

Proof. It follows from the preceding Lemma that QPV contains this system. In the other direction, we need to prove that if α is a theorem of $\mathrm{PV}_{\mathrm{BC}}$ (i.e. an axiom under (QPV3)) then it is a theorem of the alternate characterization. This is easy since in QPV we have the axioms and rules of $\mathrm{PV}_{\mathrm{BC}}$ except for the $\mathrm{PV}_{\mathrm{BC}}$ rule of induction on normal variables, which follows from (QPV3*). We do not need A1 in (QPV3*) explicitly, since we have it via (QPV2). ∎

11.2 Witnessing and $\exists^\sigma$ induction in QPV

Here we show a simplified "Buss theorem" about QPV. Interestingly, we show directly how to use unbounded $\exists$ quantification over safe variables rather than bounded $\exists$ quantification over normal variables. In comparison, Buss makes an analogy with Parikh's theorem in order to change unbounded existential quantifiers into bounded ones.

The results of Buss and the results in our earlier chapter on the functional polytime hierarchy suggest that this theorem holds for bounded existential quantification over normal inputs. Indeed this is the case as can be shown by a one-sorted argument with size bounds, of which the following is a two-sorted unbounded variation. This helps confirm the intuition that in some cases bounds which were present in *PV* can be eliminated in favor of statements concerning safe variables.

Witnessing functions for existential QPV proofs

> If $\forall$QPV $\vdash \forall \overline{x}, \overline{a} \exists \overline{b}. \alpha[\overline{x}; \overline{a}, \overline{b}]$, where $\overline{x}$, $\overline{a}$, and $\overline{b}$ are all the variables in the quantifier free formula α, then there are function symbols $\overline{f}$ such that $\vdash_{\mathrm{PV_{BC}}}$ $\alpha[\overline{x}; \overline{a}, \overline{f}(\overline{x}; \overline{a})]$.

Proof. Assuming $\forall$QPV $\vdash \forall \overline{x}, \overline{a} \exists \overline{b}. \alpha[\overline{x}; \overline{a}, \overline{b}]$, by soundness $\forall$QPV $\cup \{\neg \forall \overline{x}, \overline{a} \exists \overline{b}. \alpha[\overline{x}; \overline{a}, \overline{b}]\}$ is unsatisfiable. Introducing new constants $\zeta_{\overline{x}}$ and $\zeta_{\overline{a}}$ for the $\neg\forall$, we have that the set $\forall$QPV $\cup \{\forall \overline{b}. \neg\alpha[\zeta_{\overline{x}}; \zeta_{\overline{a}}, \overline{b}]\}$ is unsatisfiable, and by Herbrand's theorem there is a finite set of closed instances Q of $\forall$QPV and terms $\overline{t}^i(\zeta_{\overline{x}}; \zeta_{\overline{a}})$ with no free variables, such that

$$\{Q_i\} \cup \{\neg\alpha[\zeta_{\overline{x}}; \zeta_{\overline{a}}, \overline{t}^1(\zeta_{\overline{x}}; \zeta_{\overline{a}})] \ldots \neg\alpha[\zeta_{\overline{x}}; \zeta_{\overline{a}}, \overline{t}^k(\zeta_{\overline{x}}; \zeta_{\overline{a}})]\}$$

is not satisfiable. The negation of the conjunction of these is valid and by completeness, QPV proves it; namely

$$\forall\mathrm{QPV} \vdash \bigvee_i \neg Q_i \vee \neg\neg\alpha[\zeta_{\overline{x}}; \zeta_{\overline{a}}, \overline{t}^1(\zeta_{\overline{x}}; \zeta_{\overline{a}})] \vee \ldots \vee \neg\neg\alpha[\zeta_{\overline{x}}; \zeta_{\overline{a}}, \overline{t}^k(\zeta_{\overline{x}}; \zeta_{\overline{a}})]$$

The instances Q_i are theorems of QPV and therefore we can eliminate them; then we can introduce new variables for the arbitrary constants ζ, giving

$$\forall\text{QPV} \vdash \alpha[\overline{x};\overline{a},\overline{t}^1(\overline{x};\overline{a})] \vee \ldots \vee \alpha[\overline{x};\overline{a},\overline{t}^k(\overline{x};\overline{a})]$$

Now using EQ and the propositional terms of the Appendix to define a function symbol χ_α such that $\vdash_{\text{PV}_{\text{BC}}} (\chi_\alpha(\overline{x};\overline{a},b)=0) \leftrightarrow \alpha[\overline{x};\overline{a},b]$, define $\overline{f}$ by

$$\overline{f}(\overline{x};\overline{a}) \quad = \begin{cases} \overline{t}^1(\overline{x};\overline{a}) & \text{if } \chi_\alpha(\overline{x};\overline{a},\overline{t}^1(\overline{x};\overline{a})) \\ \overline{t}^2(\overline{x};\overline{a}) & \text{if } \chi_\alpha(\overline{x};\overline{a},\overline{t}^2(\overline{x};\overline{a})) \\ \ldots & \\ \overline{t}^k(\overline{x};\overline{a}) & \text{otherwise} \end{cases}$$

The proof $\vdash_{\text{PV}_{\text{BC}}} \alpha[\overline{x};\overline{a},\overline{f}(\overline{x};\overline{a})]$ required for the Lemma is obtained by classical cases on which of the formulas $\alpha[\overline{x};\overline{a},\overline{t}^i(\overline{x};\overline{a})]$ holds. (The proof is in PV_{BC} by conservativity of QPV over PV_{BC}). ∎

Of course, in the proof above we could not define χ_α if α were not quantifier free.

A similar result does hold when all variables are normal. In fact the existential $\exists b$ could be replaced by an existential over a normal variable even when $\overline{a}$ is present — the proof is identical to that above but changing b to a normal variable. Since the terms t^i in the proof above are QPV terms and therefore correspond to functions in BC, the Bounding Lemma for BC implies that all the t^i are length-bounded by polynomials. In this sense, the witnessing theorem above implies a theorem like Parikh's: unbounded existential quantifiers can be replaced with poly-bounded quantifiers.

The following shows that $\exists^\sigma$-PIND is a derived rule of QPV.

$\exists^\sigma$ induction in QPV

Let $\alpha[x,\overline{y};\overline{a}]$ be a formula of the form $\exists b.\beta[x,\overline{y};\overline{a},b]$. If

$$\forall\text{QPV} \vdash \alpha[0,\overline{y};\overline{a}] \wedge \forall x.(\alpha[x,\overline{y};\overline{a}] \rightarrow (\alpha[x0,\overline{y};\overline{a}] \wedge \alpha[x1,\overline{y};\overline{a}]))$$

then

$$\forall\text{QPV} \vdash \forall x.\alpha[x,\overline{y};\overline{a}].$$

Proof. Consider $\alpha[x,\overline{y};\overline{a}] \equiv \exists b.\beta[x,\overline{y};\overline{a},b]$ and assume the antecedent.

Applying the preceding Theorem to the clause $\forall\text{QPV} \vdash \exists b.\beta[0,\overline{y};\overline{a},b]$, we have function g such that $\vdash_{\text{PV}_{\text{BC}}} \beta[0,\overline{y};\overline{a},g(\overline{y};\overline{a})]$.

For $i \in \{0,1\}$, from $\forall x.\alpha[x,\overline{y};\overline{a}] \rightarrow \alpha[xi,\overline{y};\overline{a}]$ we prove in QPV the equivalent form

$$\forall b^1 \exists b^2.(\beta[x,\overline{y};\overline{a},b^1] \rightarrow \beta[xi,\overline{y};\overline{a},b^2]).$$

Using the preceding Theorem on this form we obtain function h_i such that

$$\vdash_{\mathrm{PV_{BC}}} \beta[x,\overline{y};\overline{a},b^1] \rightarrow \beta[xi,\overline{y};\overline{a},h_i(x,\overline{y};\overline{a},b^1)].$$

Now define f by Predicative Notational Recursion in $\mathrm{PV_{BC}}$,

$$\begin{array}{ll} f(0,\overline{y};\overline{a}) & = g(\overline{y};\overline{a}) \\ f(xi,\overline{y};\overline{a}) & = h_i(x,\overline{y};\overline{a},f(x,\overline{y};\overline{a})) \text{ for } xi \neq 0 \end{array}$$

We have directly from the statements above (and conservativity of QPV over $\mathrm{PV_{BC}}$) that $\vdash_{\mathrm{PV_{BC}}} \beta[0,\overline{y};\overline{a},f(0,\overline{y};a)]$ and $\vdash_{\mathrm{PV_{BC}}} \beta[x,\overline{y};\overline{a},f(x,\overline{y};a)] \rightarrow \beta[xi,\overline{y};\overline{a},f(x,\overline{y};\overline{a})]$ (for $i \in \{0,1\}$). By induction on the normal variable x, we have $\vdash_{\mathrm{PV_{BC}}} \beta[x,\overline{y};\overline{a},f(x,\overline{y};a)]$. This implies $\forall\mathrm{QPV} \vdash \forall x,\overline{y},\overline{a}.\exists b.\beta[x,\overline{y};\overline{a},b]$, as required. ∎

Chapter 12

The system $L_2^{\mathrm{P}}(A)$

12.1 A second-order system, $L_2^{\mathrm{P}}(A)$.

In this section we develop a second-order system akin to Leivant's $L_2(QF^+)$ (see [L]). The main difference between $L_2^{\mathrm{P}}(A)$ and $L_2(QF^+)$ is that in $L_2^{\mathrm{P}}(A)$ we have function symbols only for the polytime functions, where in $L_2(QF^+)$ we have function symbols for all computable functions. This is also the difference between Ignjatovic's $L_2^p(\overline{\Sigma}_0^b)$ and Leivant's system; however our $L_2^{\mathrm{P}}(A)$ differs from $L_2^p(\overline{\Sigma}_0^b)$ in the primitive relation symbols and in being a numeric rather than a binary string system.

The axioms for the polytime function symbols are derived from the Predicative Notational Recursion scheme. Nonetheless, $L_2^{\mathrm{P}}(A)$ is formulated over just one sort.[1]

Our system further differs from Leivant's in that we only need a weaker comprehension scheme (discussed below) and we are working over a numeric basis instead of a term algebra of binary strings ($s_0(0) = 0$ instead of $s_0(\epsilon) = 0$). We still use Leivant's W and Ignjatovic's L_2^p notation although these now have a slightly different meaning because of the difference between numbers and binary strings.

We are using comprehension only over atomic formulas, rather than all the quantifier-free positive formulas. Atomic comprehension is sufficient to simulate comprehension

[1]Formally, the set symbols of a second-order system form a second sort of variable, the first sort being the ordinary object variables. We use "type" (or, "type 1", "type 0") to refer to these different sorts of variables, reserving the word "sort" for the safe and normal sorts developed earlier.

over any quantifier-free formula which does not contain type 1 variables, because for any such formula α we can construct the characteristic function χ_α and use atomic comprehension on $\chi_\alpha(\overline{x};\overline{a}) = 0$.[2] We are primarily concerned with two types of comprehension: over atomic formulas $Q(t)$ for some term t, and over quantifier-free formulas α not containing type 1 variables. For the latter, we allow ourselves to act as though we have comprehension over these formulas without discussing the details of the characteristic function.

Definition of $L_2^{\mathrm{P}}(A)$

> The language of $L_2^{\mathrm{P}}(A)$ consists of variables $u, v, w, \ldots$ of type 0 (integers) and $Q, R, S, \ldots$ of type 1 (sets) plus quantifier symbols for both types, and the usual logical connectives plus equality and "$\in$". The function symbols of $L_2^{\mathrm{P}}(A)$ are the same as the function symbols of $\mathrm{PV}_{\mathrm{BC}}$ (the sort information is ignored). See S4-S9 in Chapter 10 for a list.
>
> The (type 0) terms of $L_2^{\mathrm{P}}(A)$ are defined by: 0 is a term; variables are terms; if f is a function symbol of the appropriate arity and $\overline{t}$ are terms then $f(\overline{t})$ is a term.
>
> Formulas are defined by: $t = u$ is an atomic formula for type 0 terms t and u; and $Q(t)$ (technically, "$t \in Q$") is an atomic formula for Q a type 1 variable and t a type 0 term; and close under $\neg$, $\wedge$ as usual.
>
> The logical basis of $L_2^{\mathrm{P}}(A)$ is the pure second-order classical predicate calculus with equality between type 0 terms.[3] We add the following axioms.

Axioms LQF1-LQF7 of $L_2^{\mathrm{P}}(A)$

LQF1-5. Axioms A1-A5 of $\mathrm{PV}_{\mathrm{BC}}$ are axioms of $L_2^{\mathrm{P}}(A)$ — in re-reading them one ignores the sort information, replacing ";" with ",". (Note A1 is redundant since we already have the 2nd order calculus).

LQF6. For each atomic formula $\gamma[u, \overline{v}, \overline{Q}]$ not containing variable R, the comprehension scheme

$$\exists R \forall u. R(u) \leftrightarrow \gamma[u, \overline{v}, \overline{Q}].$$

is an axiom of $L_2^{\mathrm{P}}(A)$.

[2]Is general QF comprehension equivalent to atomic comprehension in a system which has higher-type function symbols?

[3]Technically, the second order calculus can be formulated as follows: it is the first order calculus with two sorts, one for the objects and one for the sets of objects; function symbols operate on objects and $\in$ is a relation between objects and sets. We have not added comprehension yet; it is stated in LQF6.

LQF7. Let N be the function symbol defined by the following predicative notational recursion (c.f. section 4.11):

$$\begin{aligned} C(0,v,w) &= v \\ C(ui,v,w) &= w \ \text{ for } ui \neq 0 \\ N(u,0,w) &= 0 \\ N(u,vj,w) &= C(w \dot{-} (u \dot{-} vj) \,\underline{\mathrm{mod}}\, 1, N(u,v,w)0, N(u,v,w)1) \ \text{ for } vj \neq 0 \end{aligned}$$

where $\dot{-}$ is the function symbol defined as before. Then we add the following *extra equations* as axioms.

$$\begin{aligned} \underline{\mathrm{mod}}(u,w) &= N(u,u,w) \\ p(u \dot{-} v) &= pu \dot{-} v \\ \textsc{surj} & \quad \text{defined below} \end{aligned}$$

The addition of the extra equations is justified by two points. First, they can be written as pure equations; thus in Leivant's system we could tack them on for free whenever a functional equation appears. Second, they are statements which are provable in $\mathrm{PV_{BC}}$ (see the Appendix). Like surjectivity, the proofs of these equations in $\mathrm{PV_{BC}}$ use induction in a way whose corresponding proof in $L_2^{\mathrm{P}}(A)$ would introduce the undesirable antecedent $W(v)$. Clearly then, the significance of these equations and of SURJ is their content for numbers which are not in W.

As with QPV, we regard $L_2^{\mathrm{P}}(A)$ as a set of axioms and have $\forall L_2^{\mathrm{P}}(A) \vdash \alpha$ iff $\vdash_{L_2^{\mathrm{P}}(A)} \alpha$, where $\vdash$ (or, when necessary for clarity, $\vdash_2$) is proof in the pure second order classical predicate calculus with equality. Hence the usual theorems of second-order logic hold. Structures interpreting such a system consist of a universe (for the type 0 objects) and an arbitrary set of subsets of the universe (for the type 1 objects), plus of course interpretations of the function symbols.

Notice that the structure of the function symbols of $L_2^{\mathrm{P}}(A)$ is defined in a way which refers to the sort information of $\mathrm{PV_{BC}}$, but in $L_2^{\mathrm{P}}(A)$ itself we only have one sort. The terms of $L_2^{\mathrm{P}}(A)$ are formed by the rules S1-S3 with sort information omitted; and rules S4-S8 only say how to construct function symbols, not terms. Dividing the variables of $L_2^{\mathrm{P}}(A)$ into two countably infinite sets $u^\nu, \overline{v}^\nu, w^\nu, \ldots$ and $u^\sigma, \overline{v}^\sigma, w^\sigma, \ldots$, fix a 1-1 onto mapping of these with the normal variables $x, y, z, \ldots$ and the safe variables $a, b, c, \ldots$ respectively of QPV. Via this mapping, every term in QPV can be viewed as a term in $L_2^{\mathrm{P}}(A)$; and for every such $L_2^{\mathrm{P}}(A)$ term we can say that it "was normal in QPV" or "was safe in QPV" depending on the sort of the corresponding QPV term.

Despite this correspondence, we are not constrained in $L_2^{\mathrm{P}}(A)$ to obey sorts the way we were in QPV; for example $p(t)$ is a term even if t was safe in QPV. An essential fact is that a recursion equation such as $f(ui) = h(f(u))$ holds for *all* values of u: even though u was required to be normal in $\mathrm{PV}_{\mathrm{BC}}$, we can use the equation in $L_2^{\mathrm{P}}(A)$ without requiring any hypothesis such as $W(u)$ defined below.

Obviously we could extend $L_2^{\mathrm{P}}(A)$ by adding additional function symbols and axioms which correspond to non-safe recursions, and by widening atomic comprehension to be quantifier-free positive comprehension. It is evident that taking these steps would give a numeric system equivalent to the term algebra system $L_2(QF^+) \cup [P] \cup \textsc{surj}$ where $[P]$ are equations for all polytime functions, and in this sense we are working in a subsystem of $L_2(QF^+) \cup [P] \cup \textsc{surj}$. Later we will show that $L_2^{\mathrm{P}}(A)$ proves the convergence of all polytime functions.

We do not have induction as a general rule or axiom in $L_2^{\mathrm{P}}(A)$, but we can express that induction holds up to a specific number, u.

Definition of Cl(Q) and $W(u)$

$$
\begin{aligned}
\mathrm{Cl}(Q) &\equiv Q(0) \wedge \forall u. Q(u) \rightarrow (Q(u0) \wedge Q(u1)) \\
W(u) &\equiv \forall Q. \mathrm{Cl}(Q) \rightarrow Q(u) \\
W(\overline{u}) &\equiv W(u_1) \wedge \cdots \wedge W(u_n)
\end{aligned}
$$

These definitions are the same as those in Leivant's work (except over numbers instead of a term algebra). The first, $\mathrm{Cl}(Q)$, asserts that Q is inductive; it is the antecedent of an induction statement.

The second, $W(u)$, says that whenever Q is inductive then Q holds at u; in other words, u is reached by every inductive process. The statement $\forall u. W(u)$ is a statement of induction; but of course this is not a theorem of $L_2^{\mathrm{P}}(A)$.

We have $p0 = 0$ and $ui \neq 0 \rightarrow p(ui) = u$ as theorems from A5; however, the proof of surjectivity in $\mathrm{PV}_{\mathrm{BC}}$ was by induction and the corresponding proof $L_2^{\mathrm{P}}(A)$ would require $W(u)$ as an antecedent. We should have surjectivity without this antecedent, and therefore we add it explicitly, as Leivant does.

Surjectivity principle for $L_2^{\mathrm{P}}(A)$

$$\textsc{surj} \equiv \forall u \exists v.\ (u = v1) \vee (u = v0)$$

In connection with surjectivity we frequently use classical cases to consider $u = 0$ or $u \neq 0$; and in the latter case one uses the fact $\forall u, v.\ (u \neq 0 \wedge u = v0) \rightarrow (v \neq 0)$ which is proved briefly as: assume $v = 0$ then $u = v0 = 0$ because of the axiom $s_0(0) = 0$; contradiction. An equivalent way of stating surjectivity is "$\forall u.\ u = (pu)0 \vee u = (pu)1$". One also has $\forall u.\ p(u0) = p(u1) = u$ by considering cases $u0 = 0$ and $u0 \neq 0$.

As in the induction statements of QPV and $\mathrm{PV}_{\mathrm{BC}}$, we can harmlessly replace the step condition of $\mathrm{Cl}(Q)$ with

$$(Q(u) \rightarrow Q(u1) \wedge (u \neq 0 \rightarrow (Q(u) \rightarrow Q(u0));$$

if $u = 0$ then $u0 = 0$ and trivially $Q(u) \rightarrow Q(ui)$.

12.2 Discussion.

Intuitively, we think of $W(u)$ as asserting that u is a standard integer. In the "ordinary" model theory for second order logics, which has second-order objects for all possible subsets of the universe, $W(x)$ does in fact assert that x is a standard number. Of course second order logic is not complete for these structures, and we instead use Henkin structures as a tool to prove some of our results. In a Henkin structure, the "standardness" asserted by $W(u)$ is relative to the universe of sets which are available to interpret Q. Nonetheless, we assert that the "real meaning" of $W(x)$ is that x is standard, and that our use of Henkin structures is transparent. We are interested in the statement $W(x)$ because it has a direct technical correspondence to the normal values of QPV, as we will later demonstrate in the relativized induction theorems.

Following the intuition that $W(u)$ asserts "standardness" of u, the surjectivity principle says that the low-order bit of u is either 0 or 1 even when u is not a standard number. We can explain this as asserting that if u is a nonstandard number, it isn't because of the low-order bit. In fact surjectivity implies $u \div w \neq 0 \rightarrow \exists v.(u \div w = v0 \vee u \div w = v1)$: even if u and v are nonstandard, the v'th bit of u is either 0 or 1. In other words, we think of a nonstandard number as having a binary representation consisting of a nonstandard number of perfectly ordinary bits.

We can explain the significance of the other extra equations as follows. Consider a function f defined by notational recursion. Although f is defined in a way which naturally suggests a standard recursion depth, it is still possible to substitute a nonstandard number for the recursion variable.

For example, the function $N(u, v, w)$ takes v bits from the high-order (leftmost) end of $w \,\underline{\mathrm{mod}}\, u$. When u and w are nonstandard, the assertion $w \,\underline{\mathrm{mod}}\, u = N(u, u, w)$ amounts to saying that if we apply a notional-recursion step *a nonstandard number of times* to obtain u rightmost bits of w, and then apply a second infinite recursion process to obtain u leftmost bits of *that*, then we do in fact get exactly the 'correct' u bits. Intuitively, this is an example in which two recursions have exactly the same recursion depth for each standard u and are asserted to also have the same recursion depth for nonstandard u.

We can understand the second equation, $p(u \dot{-} v) = pu \dot{-} v$, in a similar way. Intuitively it implies for example the law $(|u| - |v|) - 1 = (|u| - 1) - |v|$ for infinite values of $|u|$ and $|v|$.

In the two-sorted system it was not necessary to have axioms which discussed the result of performing nonstandard notational recursion, because recursion was only allowed on normal values — these correspond to standard values. But when we remove the sort information, it suddenly becomes possible to substitute non-normal (in $L_2^{\mathrm{P}}(A)$, nonstandard) numbers into the recursion position, and we must have some minimal axioms which guarantee that the result is reasonable.

In the higher-order system, standardness of x corresponds to the *statement* in the logic, $W(x)$; in the first-order sorted system, it corresponds to the *type information* that x is normal. Hence in obtaining a correspondence between the two, we are essentially mapping certain explicit higher-order statements into simple type information in a first-order system.

Whereas the normal sort information used in $\mathrm{PV_{BC}}$ is available throughout a proof — and in particular is available when proving an induction step — the hypothesis $W(u)$ is only directly available for use in drawing the conclusion $Q(u)$ from the already proved statement $\mathrm{Cl}(Q)$. The natural formulation of $L_2^{\mathrm{P}}(A)$ requires us to prove the step case $Q(v) \to Q(vi)$ without the information that v is "normal" i.e. satisfies $W(v)$. Hence it is an interesting result to discover that in general we *can* assume $W(v)$ in proving the step case (see the Relativized Induction Theorem below). This allows the mapping of $\mathrm{PV_{BC}}$ proofs into $L_2^{\mathrm{P}}(A)$ proofs.

12.3 Preliminary facts about $\dot{-}$ in $L_2^{\mathrm{P}}(A)$.

Our first goal is to prove a few simple facts about the $\dot{-}$ function. These are numbered in accordance with the corresponding $\mathrm{PV_{BC}}$ facts in the Appendix. Significantly, these are proved using only the antecedent $W(u)$ for one of the operands u. This differs from the proofs in $\mathrm{PV_{BC}}$, which are allowed to use induction on either operand.

Recalling the characteristic functions χ defined in $\mathrm{PV_{BC}}$ using EQ and the propositional terms from the Appendix, observe that $\forall L_2^{\mathrm{P}}(A) \vdash \chi_\alpha(\overline{u}) = 0 \leftrightarrow \alpha[\overline{u}]$ for quantifier-free first-order α. The point here is that the proof of equivalence does not require the antecedent $W(\overline{u})$. Specifically, we can prove the important fact about the conditional by using surjectivity, without induction up to u.

Fact $(1C)$ **in** $L_2^{\mathrm{P}}(A)$

$$\begin{array}{l}\forall L_2^{\mathrm{P}}(A) \vdash (C(0,u,v) = u \wedge C(1,u,v) = v)\\ \forall L_2^{\mathrm{P}}(A) \vdash (C(w,u,v) = u \vee C(w,u,v) = v)\end{array}$$

Proof. Recall $C(w,u,v) = \mathrm{cond}(w \,\underline{\mathrm{mod}}\, 1, u, v)$, where cond was defined by predicative recursion on the first argument. Using SURJ on w we can analyse cases in $L_2^{\mathrm{P}}(A)$, using the equations for $\underline{\mathrm{mod}}$ to obtain $w \,\underline{\mathrm{mod}}\, 1 = 0 \vee w \,\underline{\mathrm{mod}}\, 1 = 1$. Then we can analyse these two cases to prove the required facts about $\mathrm{cond}(w \,\underline{\mathrm{mod}}\, 1, u, v)$. ∎

This justifies our using comprehension over quantifier-free first-order formulas to "get off the ground" in $L_2^{\mathrm{P}}(A)$.

Recall that we have the following defining axioms for $\dot{-}$:

$$\begin{array}{rl} & u \dot{-} 0 = u\\ & u \dot{-} v1 = p((u \dot{-} v) \,\underline{\mathrm{mod}}\, u;)\\ v \neq 0 \rightarrow & u \dot{-} v0 = p((u \dot{-} v) \,\underline{\mathrm{mod}}\, u)\end{array}$$

Fact $(1\,\underline{\mathbf{mod}})$ **in** $L_2^{\mathrm{P}}(A)$

$$\forall L_2^{\mathrm{P}}(A) \vdash p(v \,\underline{\mathrm{mod}}\, u) = pv \,\underline{\mathrm{mod}}\, pu$$

Proof. If $u = 0$ then it is trivial from the axioms of $\underline{\mathrm{mod}}$ and $p0 = 0$. If $u = wk \neq 0$ then consider v; if $v = v'j \neq 0$ then $p(v'j \,\underline{\mathrm{mod}}\, wk) = p((v' \,\underline{\mathrm{mod}}\, w)j) = v' \,\underline{\mathrm{mod}}\, w =$

$p(v'j)\,\underline{\mathrm{mod}}\,p(wk)$. We have used the condition $w \neq 0$ obtained from SURJ to apply the axioms for $\underline{\mathrm{mod}}$ in the case where $k = 0$. The case of $v = 0$ is similar using the identity $0 = s_0(0)$. ∎

Fact $(1\dot{-})$ **in** $L_2^{\mathrm{P}}(A)$.

$$\forall L_2^{\mathrm{P}}(A) \vdash \bigwedge_{k\in\{0,1\}} (wk \dot{-} v = (w \dot{-} v)0) \vee (wk \dot{-} v = (w \dot{-} v)1)$$

Proof. Applying SURJ, $wk \dot{-} v = (p(wk \dot{-} v))l$ for either $l = 0$ or $l = 1$. By the extra equation for p, have $(p(wk \dot{-} v))l = (p(wk) \dot{-} v)l = (w \dot{-} v)l$. ∎

Fact $(2\dot{-})$ **in** $L_2^{\mathrm{P}}(A)$

$$\forall L_2^{\mathrm{P}}(A) \vdash 0 \dot{-} v = 0$$

Proof. If $v = 0$ then it's trivial by the axioms for $\dot{-}$. If $v = ui \neq 0$ then $0 \dot{-} ui = p((0 \dot{-} u)\,\underline{\mathrm{mod}}\,0) = p0 = 0$ using the axioms for $\underline{\mathrm{mod}}$ and p. ∎

Fact $(4\dot{-})$ **in** $L_2^{\mathrm{P}}(A)$

$$\forall L_2^{\mathrm{P}}(A) \vdash W(u) \to ((u \dot{-} v)\,\underline{\mathrm{mod}}\,u) = u \dot{-} v$$

Proof. Let $Q(u) \equiv ((u \dot{-} v)\,\underline{\mathrm{mod}}\,u) = u \dot{-} v$, and prove $\mathrm{Cl}(Q)$ as follows. $Q(0)$ is trivial from the equations for $\underline{\mathrm{mod}}$ and $(2\dot{-})$.

Assuming $Q(u)$ and using $(1\dot{-})$ we have, say, $ui \dot{-} v = (u \dot{-} v)k$. By $Q(u)$ this equals $((u \dot{-} v)\,\underline{\mathrm{mod}}\,u)k$; by the equations for $\underline{\mathrm{mod}}$, it is $(u \dot{-} v)k\,\underline{\mathrm{mod}}\,ui$, identically $((ui \dot{-} v)\,\underline{\mathrm{mod}}\,ui)$.
∎

Now we have simplified things by eliminating $\underline{\mathrm{mod}}$ from the axioms for $\dot{-}$.

Fact $(5\dot{-})$ **in** $L_2^{\mathrm{P}}(A)$

$$\forall L_2^{\mathrm{P}}(A) \vdash W(u) \to \begin{array}{l} u \dot{-} 0 = u \\ u \dot{-} v1 = p(u \dot{-} v) \\ v \neq 0 \to u \dot{-} v0 = p(u \dot{-} v) \end{array}$$

Proof. This is trivial from $(4\dot{-})$ and the axioms for $\dot{-}$. ∎

Fact $(6\mathbin{\underset{\cdot}{-}})$ **in** $L_2^{\mathrm{P}}(A)$

$$\forall L_2^{\mathrm{P}}(A) \vdash W(u) \rightarrow u \mathbin{\underset{\cdot}{-}} u = 0.$$

Proof. Let $Q(u) \equiv u \mathbin{\underset{\cdot}{-}} u = 0$. Then $\mathrm{Cl}(Q)$ is trivial from $(5\mathbin{\underset{\cdot}{-}})$ and the extra equation for p. ∎

The reason we're so concerned with $\mathbin{\underset{\cdot}{-}}$ is because of the following property. This property will enable us to simulate, in $L_2^{\mathrm{P}}(A)$, a recursion in which the step case $Q(w) \rightarrow Q(wi)$ is proved under the assumption $W(w)$. It also is central to the fact that $L_2^{\mathrm{P}}(A)$ proves the convergence statement $W(\overline{x}) \rightarrow W(f(\overline{x}, \overline{a}))$ for all function symbols f; that is, for all polytime functions.

Convergence property of $\mathbin{\underset{\cdot}{-}}$

$$\forall L_2^{\mathrm{P}}(A) \vdash W(w) \rightarrow W(w \mathbin{\underset{\cdot}{-}} v).$$

Proof. Fixing w, v such that $W(w)$, introduce any Q such that $\mathrm{Cl}(Q)$. Letting $Q'(u) \equiv Q(u \mathbin{\underset{\cdot}{-}} v)$ by comprehension, we show $\mathrm{Cl}(Q')$. Applying $W(w)$ then implies $Q'(w) \equiv Q(w \mathbin{\underset{\cdot}{-}} v)$ as desired. First, $Q(0) \Rightarrow Q(0 \mathbin{\underset{\cdot}{-}} v) \Rightarrow Q'(0)$, by $(2\mathbin{\underset{\cdot}{-}})$. Second, suppose $Q'(u)$, and prove $Q'(ui)$ as follows:

$$Q'(u) \Rightarrow Q(u \mathbin{\underset{\cdot}{-}} v) \Rightarrow Q(p(ui) \mathbin{\underset{\cdot}{-}} v) \Rightarrow Q(p(ui \mathbin{\underset{\cdot}{-}} v)) \Rightarrow Q(ui \mathbin{\underset{\cdot}{-}} v) \Rightarrow Q'(ui).$$

The second and third '$\Rightarrow$'s are by the axioms for p and the extra equation for p respectively; the fourth is by $\mathrm{Cl}(Q)$ and SURJ on $ui \mathbin{\underset{\cdot}{-}} v$. ∎

12.4 $L_2^{\mathrm{P}}(A)$ proves relativized induction.

The relationship of interest between QPV formulas and $L_2^{\mathrm{P}}(A)$ formulas is given by the following *relativizing* injection.

Relativizing injection from QPV to $L_2^{\mathrm{P}}(A)$

> For α a formula of QPV, the formula α^W of $L_2^{\mathrm{P}}(A)$ is obtained by applying the mapping
>
> $$\begin{array}{lcl} (\forall x^\nu.\beta)^W & \Rightarrow & \forall x.W(x) \to \beta^W \\ (\exists x^\nu.\beta)^W & \Rightarrow & \exists x.W(x) \wedge \beta^W \\ (\forall a^\sigma.\beta)^W & \Rightarrow & \forall a.\beta^W \\ (\exists a^\sigma.\beta)^W & \Rightarrow & \exists a.\beta^W \\ (\alpha \wedge \beta)^W & \Rightarrow & \alpha^W \wedge \beta^W \\ (\neg\alpha)^W & \Rightarrow & \neg(\alpha^W) \\ (s = t)^W & \Rightarrow & s = t \end{array}$$
>
> **and then adding a prefix "$W(x) \to$" for each free normal variable x of α.** Essentially this relativizes the normal variables to lie in W. After these relativizations we must omit the sort information from the variables, in order to obtain a formula of $L_2^{\mathrm{P}}(A)$. We write $\forall x \in W$ and $\exists x \in W$ for the relativized quantifiers.

In the following we lay the groundwork for showing that the provability of a formula in QPV implies the provability of its image in $L_2^{\mathrm{P}}(A)$; this is an interesting result since the image of QPV induction on Q does not involve $\mathrm{Cl}(Q)$ but rather the seemingly weaker statement obtained by relativizing the quantifier of $\mathrm{Cl}(Q)$ (see last paragraph of Discussion section above). The reverse direction follows by a model-theoretic argument; the idea of using model theory for this is from Ignjatovic [Ig].

Definition of $W^W(u)$

$$\begin{array}{ll} \mathrm{Cl}(Q)^W & \equiv (Q(0) \wedge \forall w \in W,\ Q(w) \to Q(w0) \wedge Q(w1)) \\ W^W(u) & \equiv (\forall Q,\ \mathrm{Cl}(Q)^W \to Q(u)). \end{array}$$

$\mathrm{Cl}(Q)^W$ is pronounced, "Q is closed in W". The formula $W^w(u)$ is essentially $W(u)$ relativized to W.

We would like to prove that the relativized induction scheme follows from ordinary induction in $L_2^{\mathrm{P}}(A)$. To do this we use the following idea: if we have $W(v)$ and $\mathrm{Cl}(Q)^W$ then we have $Q(w) \to Q(wk)$ for all w which are initial segments of v; these w are all in W since v is. To get at these different values of w, we use the $\dot{-}$ function discussed above.

Closure of $Q(u \dot{-} (u \dot{-} v))$

$$\forall L_2^{\mathrm{P}}(A) \vdash W(v) \wedge \mathrm{Cl}(Q)^W \rightarrow \mathrm{Cl}(\lambda u.Q(v \dot{-} (v \dot{-} u))).$$

Proof. Proceeding in $L_2^{\mathrm{P}}(A)$, fix any v such that $W(v)$. Fix any Q such that $\mathrm{Cl}(Q)^W$, i.e. $Q(0)$ and $\forall w \in W,\ Q(w) \rightarrow Q(w0) \wedge Q(w1)$. Using atomic comprehension, let $Q'(u) = Q(v \dot{-} (v \dot{-} u))$. By the convergence property of $\dot{-}$, we have $\forall L_2^{\mathrm{P}}(A) \vdash \forall u,\ W(v) \rightarrow W(v \dot{-} (v \dot{-} u))$.

To prove $Q'(0)$, use $(5\dot{-})$ and $(6\dot{-})$ to get $v \dot{-} (v \dot{-} 0) = (v \dot{-} v) = 0$. Therefore $Q(0)$ implies $Q'(0)$.

Assume $Q'(u)$, with $u \neq 0$ or $i = 1$.

If $v \dot{-} (v \dot{-} ui) = 0$, then easily $Q(0) \rightarrow Q(v \dot{-} (v \dot{-} ui)) \rightarrow Q'(ui)$.

If $v \dot{-} u = 0$, then $(5, \dot{-})$ gives $v \dot{-} (v \dot{-} ui) = v \dot{-} p(v \dot{-} u) = v \dot{-} p0 = v \dot{-} 0 = v \dot{-} (v \dot{-} u)$. In this case, $Q'(u) \Rightarrow Q(v \dot{-} (v \dot{-} u)) \Rightarrow Q(v \dot{-} (v \dot{-} ui)) \Rightarrow Q'(ui)$, by substitutivity of equals.

Otherwise, by SURJ, $v \dot{-} (v \dot{-} iu) = wk \neq 0$ for some $k \in \{0, 1\}$ and some w (with $w \neq 0$ if k is 0); and $v \dot{-} u = xl \neq 0$ for some $l \in \{0, 1\}$ and some x ($x \neq 0$ if l is 0). Then by $(5, \dot{-})$ and substitutivity of equals, $w = p(v \dot{-} (v \dot{-} ui)) = p(v \dot{-} p(v \dot{-} u)) = p(v \dot{-} x) = v \dot{-} xl = v \dot{-} (v \dot{-} u)$. In this case, $Q'(u) \Rightarrow Q(v \dot{-} (v \dot{-} u)) \Rightarrow Q(w)$. Now, we have $W(v)$ and therefore $W(v \dot{-} (v \dot{-} u))$, i.e. $W(w)$. Recalling that Q has the property $\mathrm{Cl}(Q)^W$, we have $Q(w) \Rightarrow Q(wk) \Rightarrow Q(v \dot{-} (v \dot{-} iu)) \rightarrow Q'(iu)$.

This proves $\mathrm{Cl}(Q')$. ∎

Relativized Induction Theorem

$$\forall L_2^{\mathrm{P}}(A) \vdash W(v) \leftrightarrow W^W(v).$$

Proof. For the $\rightarrow$ direction, we need to show $W(v) \rightarrow (\forall Q.\ \mathrm{Cl}(Q)^W \rightarrow Q(v))$ in $L_2^{\mathrm{P}}(A)$. Fixing v and Q such that $W(v)$ and $\mathrm{Cl}(Q)^W$, applying the preceding theorem gives $\mathrm{Cl}(\lambda u.Q(v \dot{-} (v \dot{-} u)))$. Now using $W(v)$ gives $Q(v \dot{-} (v \dot{-} v))$, which by $(6, \dot{-})$ and $(5, \dot{-})$ is $Q(v)$ as desired.

The direction $W^W(v) \rightarrow W(v)$ is trivial since $\mathrm{Cl}(Q) \rightarrow \mathrm{Cl}(Q)^W$. ∎

For good measure we demonstrate that, at least relative to W, closure of Q under successors is as good as closure of Q under both successors and predecessors.

First we define closure under both successors and predecessors.

$$\begin{aligned}
\mathrm{Cl}^*(Q) \quad &\equiv Q(0) \wedge \forall v, Q(v) \to (Q(0v) \wedge Q(1v)) \\
&\qquad \wedge \forall v, (Q(0v) \vee Q(1v)) \to Q(v) \\
\mathrm{Cl}^*(Q)^W \quad &\equiv Q(0) \wedge \forall v \in W,\ Q(v) \to (Q(0v) \wedge Q(1v)) \\
&\qquad \wedge \forall v \in W,\ (Q(0v) \vee Q(1v)) \to Q(v). \\
W^{*W}(v) \quad &\equiv \forall Q,\ \mathrm{Cl}^*(Q)^W \to Q(v).
\end{aligned}$$

Lemma: Cl$(Q)^W$ implies Cl$^*(Q)^W$

$$\forall L_2^{\mathrm{P}}(A) \vdash \mathrm{Cl}(Q)^W \to \mathrm{Cl}^*(Q)^W.$$

Proof. Fix Q such that $\mathrm{Cl}(Q)^W$. We need to prove the second clause of $\mathrm{Cl}^*(Q)^W$, namely that $\forall w \in W,\ (Q(w0) \vee Q(w1)) \to Q(w)$. Fix w such that $W(w)$ and assume $Q(wi)$ for some $i \in \{0, 1\}$.

If $w = 0$ and i is 0 then $\vdash wi = w$ so we immediately have $Q(w)$ as required. Therefore, consider that $w \neq 0$ or i is 1.

$W(wi)$ follows easily from $W(w)$, so by the Lemma above we have $\mathrm{Cl}(\lambda u,\ Q(wi \dot{-} (wi \dot{-} u)))$. Applying $W(w)$ gives $Q(wi \dot{-} (wi \dot{-} w))$.

An easy proof by induction (i.e. by $W(w)$) shows that

$$(w1 \dot{-} w = 1) \wedge (w \neq 0 \to w0 \dot{-} w = 1)$$

Using $(5, \dot{-})$ we have $wi \dot{-} 1 = w$. Therefore $Q(wi \dot{-} (wi \dot{-} w)) \to Q(w)$. So we have $Q(w)$. ∎

By now we have passed from considering v satisfying all closed Q, to v satisfying all Q which are upwards-and-downwards closed inside W.

Bidirectional Relativized Induction

$$\forall L_2^{\mathrm{P}}(A) \vdash W(v) \leftrightarrow W^{*W}(v).$$

Proof. We already have $W(v) \leftrightarrow W^W(v)$. The statements $W^W(v)$ and $W^{*W}(v)$ are identical except $\mathrm{Cl}(Q)^W$ is replaced by $\mathrm{Cl}^*(Q)^W$. The Lemma above shows that $\mathrm{Cl}(Q)^W$ implies $\mathrm{Cl}^*(Q)^W$; the reverse implication is trivial from the definition of $\mathrm{Cl}^*(Q)^W$ as compared to $\mathrm{Cl}(Q)^W$. ∎

12.5 Convergence of polytime functions in $L_2^{\mathrm{P}}(A)$

To prove convergence of polytime functions, we strengthen the induction hypothesis; this lets the induction go through for Predicative Notational Recursion, using the Relativized Induction Theorem. If we wanted a less elegant proof which avoided the Relativized Induction Theorem, we could consider each f defined by predicative recursion, say $f(xi) = h(x, f(x))$, and add equations such as $\hat{f}(ui, x) = h(x \dot{-} (x \dot{-} u), \hat{f}(u, x))$ and $f(x) = \hat{f}(x, x)$ to the system.

Theorem: Strong Convergence of Polytime Functions

> Let $t[\overline{u}, \overline{v}]$ be an $L_2^{\mathrm{P}}(A)$ term where $\overline{u}$ are the images of normal QPV variables and $\overline{v}$ are the images of safe QPV variables. Then
>
> $$\texttt{(i)} \quad \forall L_2^{\mathrm{P}}(A) \vdash \forall \overline{u}, \overline{v}, Q.\ (W(\overline{u}) \wedge \mathrm{Cl}(Q) \wedge Q(\overline{v})) \rightarrow Q(t[\overline{u}, \overline{v}]).$$
>
> Furthermore, if t was normal sort in QPV, then
>
> $$\texttt{(ii)} \quad \forall L_2^{\mathrm{P}}(A) \vdash \forall \overline{u}, \overline{v}, Q.\ (W(\overline{u}) \wedge \mathrm{Cl}(Q)) \rightarrow Q(t[\overline{u}, \overline{v}]).$$

Proof. The proof is by induction on the structure of terms, defined by S1-S9. Throughout this proof we speak of normal and safe terms, referring of course to whether the term was normal or safe sort in QPV.

For S1 (variables) and S2 (constant 0) it is trivial.

The only other rule which constructs a term is S3, where t is $f(t_1, \ldots, t_n)$ for function symbol f. We consider two cases. The first case is if one or more of the t_i's is not simply a variable; in this case we use the induction hypothesis on the t_i's and on the term $f(x_1, \ldots, x_k, a_{k+1}, \ldots, a_n)$ for variables $\overline{x}$ and $\overline{a}$ which were normal or safe respectively in QPV. Write $\overline{r}$ for $t_1, \ldots t_k$ and $\overline{s}$ for $t_{k+1}, \ldots, t_n$. Working in $L_2^{\mathrm{P}}(A)$, fix $\overline{u}$, $\overline{v}$, and Q. The terms $\overline{r}[\overline{u}, \overline{v}]$ must be normal, and by the induction hypothesis `(ii)` we have a proof of $W(\overline{u}) \rightarrow W(\overline{r}[\overline{u}, \overline{v}])$. To prove the conclusion of `(ii)` in the case where f is normal: we have by the induction hypothesis `(ii)` on $f(\overline{x}, \overline{a})$ that

$$(W(\overline{r}[\overline{u}, \overline{v}]) \wedge \mathrm{Cl}(Q)) \rightarrow Q(f(\overline{r}[\overline{u}, \overline{v}], \overline{s}[\overline{u}, \overline{v}])),$$

so $W(\overline{u}) \wedge \mathrm{Cl}(Q) \rightarrow Q(f(\overline{r}[\overline{u}, \overline{v}], \overline{s}[\overline{u}, \overline{v}])$ follows by modus ponens. The induction hypothesis on $\overline{s}$ implies that we have a proof of $(W(\overline{u}) \wedge \mathrm{Cl}(Q) \wedge Q(\overline{v})) \rightarrow Q(\overline{s}[\overline{u}, \overline{v}])$. To prove the conclusion of `(i)`: we have by the induction hypothesis `(i)` on $f(\overline{x},\overline{a})$ that

$$(W(\overline{r}[\overline{u},\overline{v}]) \wedge \mathrm{Cl}(Q) \wedge Q(\overline{s}[\overline{u},\overline{v}])) \rightarrow Q(f(\overline{r}[\overline{u},\overline{v}],\overline{s}[\overline{u},\overline{v}])),$$

so $W(\overline{u}) \wedge \mathrm{Cl}(Q) \wedge Q(\overline{v}) \rightarrow Q(f(\overline{r}[\overline{u},\overline{v}],\overline{s}[\overline{u},\overline{v}])$ by modus ponens.

The second case to consider for rule S3 is when the term is $f(\overline{u},\overline{v})$. Here we must consider by which of the rules S4-S9 the function symbol f was formed, and apply the axioms LQF1-LQF5 (A1-A5) and LQF7 (the extra equations). The case of S9 (raising of f) is easy since in that case f has only normal inputs: we have `(i)` for the term $f^{\sigma}(\overline{u})$ and with A3 this is exactly the required `(ii)` (and also `(i)`) for $(f^{\sigma})^{\nu}(\overline{u})$. The case of S8 (lowering of f) is trivial using A3, since we have `(i)` and `(ii)` for the normal term $f^{\nu}(\overline{u},\overline{v})$ and we only need `(i)` for the safe term $(f^{\nu})^{\sigma}(\overline{u},\overline{v})$. The case of S6 ($t \equiv [\lambda\overline{u},\overline{v}.(T)](\overline{u},\overline{v})$) is trivial using A4 and the induction hypothesis on $T[\overline{u},\overline{v}]$. The successor functions s_0 and s_1 of S4 trivially have the property $\mathrm{Cl}(Q) \wedge Q(a) \rightarrow Q(s_i(a))$ by the definition of $\mathrm{Cl}(Q)$. The function EQ of S5 is also easily disposed of by A2, since the output is always either 0 or $s_1(0)$, both of which are in W. This leaves the function symbol <u>mod</u> of S5, and function symbols defined by Predicative Notational Recursion (S7).

For <u>mod</u>, we have $v\,\underline{\mathrm{mod}}\,u = N(u,u,v)$ (see LQF7). Working in $L_2^{\mathrm{P}}(A)$, fix any u, v, and Q such that $W(u) \wedge \mathrm{Cl}(Q)$, and define $R(x) \equiv Q(N(u,x,v))$. We prove $\mathrm{Cl}(R)$; by $W(u)$ this implies $Q(N(u,u,v))$ as required. First, $R(0) \Leftrightarrow Q(N(u,0,a)) \Leftrightarrow Q(0)$, and $Q(0)$ holds by $\mathrm{Cl}(Q)$. Second, suppose $Q(N(u,x,v))$. For $i=1$, or $i=0$ and $x \neq 0$, the equation for N is $N(u,xi,v) = C(\textsc{parity}(v \dot{-} (u \dot{-} xi)), N(u,x,v)0, N(u,x,v)1)$ and by the fact $C(a,b,c) = b \vee C(a,b,c) = c$, we have $N(u,xi,v) = N(u,x,v)0 \vee N(u,xi,v) = N(u,x,v)1$. Taking classical cases, $\mathrm{Cl}(Q)$ implies $Q(N(u,x,v)) \rightarrow Q(N(u,x,v)j)$ for $j \in \{0,1\}$ and by modus ponens this implies $R(xi)$ as required.

When $f(w,\overline{u},\overline{v}) \equiv \mathbf{R}[g,h_0,h_1](w,\overline{u},\overline{v})$ is defined by Predicative Notational Recursion on w (see A5), we rely on the Relativized Induction Theorem. Here w and $\overline{u}$ are normal, and $\overline{v}$ is safe. Working in $L_2^{\mathrm{P}}(A)$, fix w, $\overline{u}$, $\overline{v}$, and Q such that $W(w,\overline{u})$, $\mathrm{Cl}(Q)$, and $Q(\overline{v})$. Defining $R(x) \equiv Q(f(x,\overline{u},\overline{v}))$, we prove $\mathrm{Cl}(R)^{W}$; since $W^{W}(w)$ this implies $R(w)$.

To prove $R(0)$, we have $Q(g(\overline{u},\overline{v}))$ by the main induction hypothesis on the term $g(\overline{u},\overline{v})$ and by the facts $W(\overline{u})$, $\mathrm{Cl}(Q)$, and $Q(\overline{v})$. Now fix $x \in W$; we need to prove $R(x) \rightarrow R(xi)$ for $i \in \{0,1\}$. If i is 0 and $x=0$ it is trivial because $R(0)$. Otherwise, we prove $R(xi)$ using $Q(f(xi,\overline{u},\overline{v})) \Leftrightarrow Q(h_i(x,\overline{u},\overline{v},f(x,\overline{u},\overline{v})))$. The main induction hypothesis on the term $h_i(x, \overline{u}, \overline{v}, a)$ is

$$\forall x, \overline{u}, \overline{v}, a, Q.\ (W(x, \overline{u}) \wedge \mathrm{Cl}(Q) \wedge Q(a, \overline{v})) \rightarrow Q(h_i(x, \overline{u}, \overline{v}, a)).$$

Introducing x, $\overline{u}$, $\overline{v}$ and introducing $f(x, \overline{u}, \overline{v})$ for a, we have the required antecedents, including $R(x) \equiv Q(f(x, \overline{u}, \overline{v}))$ and $W(x)$. Here is where we are using the condition $x \in W$ which holds since we only need to prove relativized closure. By modus ponens we have the conclusion $Q(h_i(x, \overline{u}, \overline{v}, f(x, \overline{u}, \overline{v})))$ as required. ∎

In this way we obtain a more direct proof of one direction of Leivant's theorem.

Corollary: Convergence of polytime functions

> Let $t[\overline{u}, \overline{v}]$ be an $L_2^{\mathrm{P}}(A)$ term where $\overline{u}$ are the images of normal QPV variables and $\overline{v}$ are the images of safe QPV variables. Then
>
> $$\texttt{(i)} \quad \forall L_2^{\mathrm{P}}(A) \vdash \forall \overline{u}, \overline{v}.\ W(\overline{u}, \overline{v}) \rightarrow W(t[\overline{u}, \overline{v}]).$$
>
> Furthermore, if t was normal sort in QPV, then
>
> $$\texttt{(ii)} \quad \forall L_2^{\mathrm{P}}(A) \vdash \forall \overline{u}, \overline{v}.\ W(\overline{u}) \rightarrow W(t[\overline{u}, \overline{v}]).$$

Proof. This is just a restatement of Strong Convergence, weakening `(i)` slightly by replacing $Q(\overline{v})$ with $W(\overline{v})$. ∎

Chapter 13

Interrelationships

13.1 Further Discussion

The Relativized Induction Theorem allows the mapping of $\mathrm{PV_{BC}}$ proofs into $L_2^{\mathrm{P}}(A)$ proofs. Specifically we will prove that QPV proves a fact α containing only normal variables if and only if $L_2^{\mathrm{P}}(A)$ proves α^W.

On the part of the logic which is not "normal", a weaker correspondence holds. More exactly, we show the following, using "nonstandard" as synonymous with "$W(x)$ is not known in $L_2^{\mathrm{P}}(A)$", and "impredicative" as synonymous with "safe in QPV": (1) if a QPV fact can be proved about all impredicative values then it can be proved in $L_2^{\mathrm{P}}(A)$ about all nonstandard values; and (2) if a fact can be proved in $L_2^{\mathrm{P}}(A)$ about all nonstandard values then it can be proved in QPV about all normal (standard) values.

Thus the correspondence between "x is normal" and "$W(x)$ is known" is *not* mirrored by a correspondence between "a is safe" and "$W(a)$ is not known". In $L_2^{\mathrm{P}}(A)$ we know a lot about values of which W is *not* asserted, where in QPV we know relatively little about them. In particular, in $L_2^{\mathrm{P}}(A)$ such statements as the recursion axioms A5 and the extra axioms LQF7 hold even in the absence of W. Evidently, this is intrinsic to the second-order system as formulated by Leivant, since these statements in $L_2^{\mathrm{P}}(A)$ seem to be required to prove convergence of all polytime functions. In QPV the recursion axioms A5 do assert facts about safe values — for example, we have weak facts such as $a \oplus 01 = a01$, and the inferences deduced without induction from such facts, which prevent the safe values from being arbitrary. What we do not have in QPV is that statements requiring induction for their proof hold at safe values.

We can speculate that if one were to weaken the axioms A5 of $L_2^{\mathrm{P}}(A)$ to include the antecedent W, and omit the extra equations, then one would obtain a system in which the provable statements not involving W correspond to QPV statements on safe inputs. However, in such a system, focussing on the statements not involving W would exclude most of arithmetic. Therefore a correspondence between QPV and a weakened $L_2^{\mathrm{P}}(A)$ still would not indicate that the "impredicative" values denoted by safe variables are "nonstandard" integers.

Complementary to the idea of weakening $L_2^{\mathrm{P}}(A)$, we might think of strengthening QPV to a new system QPV$^+$ which asserts more about the safe values, and in this way force the "impredicative" values to be in fact nonstandard values in every QPV$^+$ model. For example we might allow induction on safe values. However, this approach seems artificial since the assertions about safe values would not have any grounding in the definition of the functional classes by predicative recursion.

Thus I conclude that the "impredicative" values denoted by the safe variable are a distinct kind of number: they satisfy some functional equations, but are not known to be nonstandard. In particular we do not know that they are reached by any instances of induction.

13.2 Equivalence of $L_2^{\mathrm{P}}(A)$ and QPV.

Here we use the alternate characterization of QPV and the relativized induction theorem to show that if a formula is provable in QPV (or $\mathrm{PV_{BC}}$) then its relativization is provable in $L_2^{\mathrm{P}}(A)$. We also write down a proof of the reverse direction using a modification of Ignjatovic's model-theoretic argument [Ig].

An interpretation consists of a structure $\mathcal{I}$ (universes for the objects and sets of object, and interpretations for the function symbols and the constant 0) together with an assignment $\mathcal{A}$ for the variables of the language.

Corresponding models

> We identify the variables of QPV with the type 0 variables of $L_2^{\mathrm{P}}(A)$ as discussed in section 12.1, and say that an assignment $\mathcal{A}$ for the $L_2^{\mathrm{P}}(A)$ variables and an assignment $\mathcal{A}'$ for the QPV variables are *corresponding* if they assign

> the same values to all type 0 variables. A structure $\mathcal{I}$ with universe U for the language of $L_2^{\mathrm{P}}(A)$ and a structure $\mathcal{I}'$ with universes $N \subseteq S$ for the language of QPV are *corresponding* if $\mathcal{I}(W) = N$, $U = S$, and $(\mathcal{I}'f)(\overline{\mathbf{x}};\overline{\mathbf{a}}) = (\mathcal{I}f)(\overline{\mathbf{x}},\overline{\mathbf{a}})$ for all $\overline{\mathbf{x}} \in N, \overline{\mathbf{a}} \in S$. Of course, equality has to be interpreted normally. Two interpretations correspond if their structures and assignments correspond.

For each QPV assignment there is a corresponding $L_2^{\mathrm{P}}(A)$ assignment, but not vice versa: QPV assignments are restricted to assign normal values to type ν variables.

Lemma on correspondence of satisfaction

> If $\langle\mathcal{I},\mathcal{A}\rangle$ and $\langle\mathcal{I}',\mathcal{A}'\rangle$ are corresponding interpretations, then for all formulas γ in the language of QPV, $\langle\mathcal{I}',\mathcal{A}'\rangle \models \gamma \Leftrightarrow \langle\mathcal{I},\mathcal{A}\rangle \models \gamma^W$.

Proof. The proof is by structural induction on γ.

The base case is $\gamma \equiv t[\overline{x};\overline{a}] = s[\overline{x};\overline{a}]$ for terms s and t. That $\langle\mathcal{I}',\mathcal{A}'\rangle \models \gamma \Leftrightarrow \langle\mathcal{I},\mathcal{A}\rangle \models \gamma^W$ is trivial since $(\mathcal{I}f) = (\mathcal{I}'f)$, $\mathcal{A}(\overline{x},\overline{a}) = \mathcal{A}'(\overline{x};\overline{a})$, and the condition $W(\overline{x})$ appearing in γ^W is satisfied by $\mathcal{A}'(\overline{x}) \in N \equiv \mathcal{I}'(W)$.

Next consider $\gamma \equiv \forall x.\beta$ (where x is normal). It is trivial if x is not free in β. Otherwise, $\langle\mathcal{I}',\mathcal{A}'\rangle \models \forall x.\beta$ if and only if $\forall\mathbf{x} \in N.\langle\mathcal{I}',\mathcal{A}'^x_{\mathbf{x}}\rangle \models \beta$. By the induction hypothesis on each pair $(\mathcal{A}^x_{\mathbf{x}},\mathcal{A}'^x_{\mathbf{x}})$, this holds if and only if $\forall\mathbf{x} \in \mathcal{I}(W).\langle\mathcal{I},\mathcal{A}^x_{\mathbf{x}}\rangle \models \beta^W$. Since β^W is of the form $W(x) \to \alpha$ and $\langle\mathcal{I},\mathcal{A}^x_{\mathbf{x}}\rangle \models \neg W(x)$ for $\mathbf{x} \in U\backslash\mathcal{I}(W)$, we have $\forall\mathbf{x} \in U.\langle\mathcal{I},\mathcal{A}^x_{\mathbf{x}}\rangle \models \beta^W$. Therefore $\langle\mathcal{I},\mathcal{A}\rangle \models \forall x.(\beta^W)$, which is identically $(\forall x.\beta)^W$.

The cases of $\neg$, $\wedge$, and $\forall a$ (for safe a) are easier and are omitted. ∎

A simpler, one-sorted version of the following can be found in Bell and Machover [BM] pages 102-104. We need some additional manipulations to make sure the sorts are treated correctly, and we use the Convergence Theorem.

It is a general result inasmuch as it holds for relativizations to any predicate R provided $R(\overline{u}) \to R(f(\overline{u}))$; that is, provided an appropriate convergence theorem holds.

Theorem: if $\forall$QPV $\vdash \alpha$ then $\forall L_2^{\mathrm{P}}(A) \vdash \alpha^W$

> If α is a formula of QPV and $\forall$QPV $\vdash \alpha$, then $\forall L_2^{\mathrm{P}}(A) \vdash \alpha^W$.
> If α is a formula of $\mathrm{PV_{BC}}$ and $\vdash_{\mathrm{PV_{BC}}} \alpha$, then $\forall L_2^{\mathrm{P}}(A) \vdash \alpha^W$.

Proof. The second part of this theorem follows directly from the first part, since QPV is an extension of $\mathrm{PV_{BC}}$.

The relativizations of the axioms A2-A5 (in QPV3*) are implied by the fact that A2-A5 themselves are explicitly in $L_2^{\mathrm{P}}(A)$. This is true because they are all universal axioms, hence e.g. the image of $\beta[\overline{z};\overline{b}]$ is $W(\overline{z}) \to \beta[\overline{z};\overline{b}]$ and follows from β in $L_2^{\mathrm{P}}(A)$.

This leaves the axiom of induction on normal variables, whose image in $L_2^{\mathrm{P}}(A)$ is

$$W(\overline{y}) \to (\alpha[0,\overline{y},\overline{a}] \wedge \forall x \in W.\alpha[x,\overline{y},\overline{a}] \to (\alpha[x0,\overline{y},\overline{a}] \wedge \alpha[x1,\overline{y},\overline{a}])) \to \forall x \in W.\alpha[x,\overline{y},\overline{a}],$$

for each quantifier-free first-order α. In $L_2^{\mathrm{P}}(A)$ we have $\mathrm{Cl}(Q)^W \to \forall x \in W^W.Q(x)$ by the definition of $W^W(x)$. By the relativized induction theorem, $W^W(x) \leftrightarrow W(x)$, so this is $\mathrm{Cl}(Q)^W \to \forall x \in W.Q(x)$. Defining $Q(x) \equiv \alpha[x,\overline{y};\overline{a}]$ by atomic comprehension with χ_α, substitute Q into this statement. Expanding the definition of $\mathrm{Cl}^W(Q)$ and using $Q(x) \leftrightarrow \alpha[x,\overline{y};\overline{a}]$ gives

$$(\alpha[0,\overline{y},\overline{a}] \wedge \forall x \in W.\alpha[x,\overline{y},\overline{a}] \to (\alpha[x0,\overline{y},\overline{a}] \wedge \alpha[x1,\overline{y},\overline{a}])) \to \forall x \in W.\alpha[x,\overline{y},\overline{a}],$$

which implies the image of the axiom of induction on normal variables.

Although the second order logic used for $L_2^{\mathrm{P}}(A)$ can perform all the logical inferences of the first order logic used for QPV, this does not directly show that $\forall\mathrm{QPV}^W \vdash_2 \beta^W$ whenever $\forall\mathrm{QPV} \vdash_{\sigma\nu} \beta$. Instead, we use a model-theoretic statement.

Assume $\forall\mathrm{QPV} \vdash \beta$. By compactness and the deduction theorem, $\vdash_{\sigma\nu} \alpha \to \beta$, where α is a conjunction of $\forall$QPV axioms. We prove $\forall L_2^{\mathrm{P}}(A) \vdash \alpha^W \to \beta^W$. This will complete the Theorem since as shown above α^W is provable in $L_2^{\mathrm{P}}(A)$.

Note that $\vdash_2 (\alpha^W \to \beta^W) \leftrightarrow (\alpha \to \beta)^W$, because α has no free variables.

Let $\mathcal{I}$ be any structure for the language of $L_2^{\mathrm{P}}(A)$ such that $\mathcal{I} \models \forall L_2^{\mathrm{P}}(A)$. Define a corresponding structure $\mathcal{I}'$ for the language of QPV as follows:

(1) The universe S for the safe sort is the universe of $\mathcal{I}$.

(2) The universe $N \subseteq S$ for the normal sort is $\mathcal{I}(W)$.

(3) Equality is interpreted normally in $\mathcal{I}'$; this is the same as in $\mathcal{I}$.

(4) The interpretation of function symbol f in $\mathcal{I}'$ is the same as in $\mathcal{I}$, but with normal inputs restricted to $\mathcal{I}(W)$.

Since $\forall L_2^{\mathrm{P}}(A) \vdash W(0)$, $N = \mathcal{I}(W) \neq \emptyset$. For (4) to be a meaningful definition, we must verify that if function $f(\overline{x};\overline{a})$ has ν sort output then for $\overline{\mathbf{x}} \in N$ and $\overline{\mathbf{a}} \in S$, $(\mathcal{I}'f)(\overline{\mathbf{x}};\overline{\mathbf{a}}) \in N$. This follows from $\mathcal{I}(W) = N$ and the Convergence Theorem (together with $L_2^{\mathrm{P}}(A)$ soundness and the fact that $\mathcal{I}$ satisfies $\forall L_2^{\mathrm{P}}(A)$): convergence implies $\overline{\mathbf{x}} \in \mathcal{I}(W) \Rightarrow (\mathcal{I}f)(\overline{\mathbf{x}},\overline{\mathbf{a}}) \in \mathcal{I}(W)$.

Fixing any assignment $\mathcal{A}$ for the variables of $L_2^{\mathrm{P}}(A)$, we consider two cases. If $\mathcal{A}$ assigns $\mathbf{u} \notin \mathcal{I}(W)$ to some free normal variable x of β^W then $\langle \mathcal{I},\mathcal{A}\rangle$ satisfies $(\alpha \to \beta)^W$ because it satisfies $\neg W(x)$. Otherwise, let $\mathcal{A}_*$ be an assignment which is identical to $\mathcal{A}$ on the free variables of β and which assigns members of $\mathcal{I}(W)$ to $L_2^{\mathrm{P}}(A)$ variables which were normal in QPV and which do not appear in β. Then $\langle \mathcal{I},\mathcal{A}_*\rangle \models (\alpha \to \beta)^W$ iff $\langle \mathcal{I},\mathcal{A}\rangle \models (\alpha \to \beta)^W$, and $\mathcal{A}_*$ has a corresponding QPV assignment $\mathcal{A'}_*$. By the soundness and that $\vdash_{\sigma\nu} \alpha \to \beta$, we have $\langle \mathcal{I}',\mathcal{A'}_*\rangle$ satisfies $\alpha \to \beta$. Therefore by the preceding Lemma, $\langle \mathcal{I},\mathcal{A}\rangle \models \alpha^W \to \beta^W$.

This shows that $\alpha^W \to \beta^W$ is satisfied by all models of $\forall L_2^{\mathrm{P}}(A)$, and by completeness $\forall L_2^{\mathrm{P}}(A) \vdash \alpha^W \to \beta^W$. ∎

The following Theorem is in the opposite direction from that above, and is based on the model-theoretic argument of Ignjatovic. As discussed earlier, the statement is weakened by the fact that in $L_2^{\mathrm{P}}(A)$ we can prove a lot of facts about numbers not known to be in W; in QPV we do not have these facts and therefore we confine our attention to normal values.

The normal translation, α^ν

> For formula or term α in QPV, define α^ν by replacing each safe variable a^σ with a normal variable x_a^ν. In doing this, preserve sorts in composition (S3) by using the lowering operation $(x_a^\nu)^\sigma$. A variable of α^ν *was safe* if it is x_a^ν for some safe variable a in α.

Theorem: if $\forall L_2^{\mathrm{P}}(A) \vdash \alpha^W$ then $\forall$QPV $\vdash \alpha^\nu$.

> If α is a formula of QPV and $\forall L_2^{\mathrm{P}}(A) \vdash \alpha^W$, then $\forall$QPV $\vdash \alpha^\nu$. If α is a formula of $\mathrm{PV_{BC}}$ and $\forall L_2^{\mathrm{P}}(A) \vdash \alpha^W$, then $\vdash_{\mathrm{PV_{BC}}} \alpha^\nu$.

Proof. The $\mathrm{PV_{BC}}$ part of this theorem follows from the QPV statement because QPV is conservative over $\mathrm{PV_{BC}}$.

Suppose $\forall L_2^{\mathrm{P}}(A) \vdash \alpha^W$.

Relying on the completeness of QPV, we show that any model $\langle \mathcal{I}', \mathcal{A}' \rangle$ of QPV satisfies α^ν. Fix $\langle \mathcal{I}', \mathcal{A}' \rangle$ satisfying $\forall$QPV, with safe universe S and normal universe $N \subseteq S$.

Without loss of generality we can assume $S = N$ by the following argument: α^ν has only normal variables in it and therefore (using the Convergence Theorem) the restriction of $\langle \mathcal{I}', \mathcal{A}' \rangle$ to N satisfies α^ν iff $\langle \mathcal{I}', \mathcal{A}' \rangle$ does; furthermore the restriction satisfies $\forall$QPV since the original $\langle \mathcal{I}', \mathcal{A}' \rangle$ does and $\forall$QPV has no free variables. (In performing this restriction, we can assign any values in N to safe variables which $\mathcal{A}'$ mapped outside of N — the choice is irrelevant since there are no free safe variables).

Now define $\langle \mathcal{I}, \mathcal{A} \rangle$, an interpretation for the language of $L_2^{\mathrm{P}}(A)$, as follows.

(0) $\mathcal{A}$ is any assignment corresponding to $\mathcal{A}'$. (As in the earlier proofs, we are only interested in the assignment which $\mathcal{A}$ makes to type 1 variables.)

(1) The universe U of $\mathcal{I}$ is S, identically N.

(2) Equality is interpreted normally by $\mathcal{I}$; this is the same as in $\mathcal{I}'$.

(3) The interpretation $\mathcal{I}f$ of $L_2^{\mathrm{P}}(A)$ function symbol $f(\overline{u}, \overline{v})$, where inputs $\overline{u}$ were normal and $\overline{v}$ were safe in QPV, is defined by $(\mathcal{I}f)(\overline{\mathbf{u}}, \overline{\mathbf{v}}) = (\mathcal{I}'f)(\overline{\mathbf{u}}, \overline{\mathbf{v}})$ for $\overline{\mathbf{u}} \in N$, $\overline{\mathbf{v}} \in S$. This is a complete definition since $U = N = S$.

(4) For the subsets of U interpreting the second-order variables of $L_2^{\mathrm{P}}(A)$, use the subsets

$$\mathbf{Q}_{\chi, \overline{\mathbf{v}}} \equiv \{\mathbf{u} \in U : \ (\mathcal{I}\chi)(\mathbf{u}, \overline{\mathbf{v}}) = (\mathcal{I}0)\}$$

for each set of values $\overline{\mathbf{v}} \in U$ and each function symbol $\chi(u, \overline{v})$ of $L_2^{\mathrm{P}}(A)$.

$\langle \mathcal{I}', \mathcal{A}' \rangle$ is a model of $\forall$QPV and QPV contains the axioms A2-A5 and proves the extra equations of LQF7 on normal inputs (See Appendix). Therefore by (3), $\mathcal{I}$ satisfies LQF2-5 and LQF7 on $U = N = S$.

The only remaining axiom is LQF6 (comprehension). To see that it is satisfied, consider any atomic formula $\gamma[u, \overline{v}, \overline{Q}]$ in $L_2^{\mathrm{P}}(A)$. If $\gamma \equiv t = s$ then $\overline{Q}$ is the empty vector. Define the characteristic function $\chi(u, \overline{v}) \equiv t(u, \overline{v}) \,\mathrm{EQ}\, s(u, \overline{v})$; we have $\forall L_2^{\mathrm{P}}(A) \vdash \chi(u, \overline{v}) = 0 \leftrightarrow \gamma[u, \overline{v}]$. Then the set witnessing comprehension for χ was explicitly added to $\mathcal{I}$, in (4). Otherwise, $\gamma \equiv Q(t[u, \overline{v}])$ for some second-order variable Q; choosing any set $\mathbf{Q}_{\chi, \overline{\mathbf{w}}}$ for

Q and any $\overline{\mathbf{v}} \in U$ for $\overline{v}$, we have

$$\begin{aligned}
\mathbf{u} \in \mathcal{I}^{\overline{v},Q}_{\overline{\mathbf{v}},\mathbf{Q}_{\chi,\overline{\mathbf{w}}}}(Q(t[u,\overline{v}])) &\Leftrightarrow (\mathcal{I}t)(\mathbf{u},\overline{\mathbf{v}}) \in \mathbf{Q}_{\chi,\overline{\mathbf{w}}} \\
&\Leftrightarrow (\mathcal{I}\chi)((\mathcal{I}t)(\mathbf{u},\overline{\mathbf{v}}),\overline{\mathbf{w}}) = \mathcal{I}0 \\
&\Leftrightarrow (\mathcal{I}\lambda u,\overline{v},\overline{w}.(\chi(t[u,\overline{v}],\overline{w})))(\mathbf{u},\overline{\mathbf{v}},\overline{\mathbf{w}}) = \mathcal{I}0 \\
&\Leftrightarrow \mathbf{u} \in \mathbf{Q}_{\lambda u,\overline{v},\overline{w}.(\chi(t[u,\overline{v}],\overline{w})),\overline{\mathbf{v}},\overline{\mathbf{w}}}
\end{aligned}$$

Therefore the set $\mathbf{Q}_{\lambda u,\overline{v},\overline{w}.(\chi(t[u,\overline{v}],\overline{w})),\overline{\mathbf{v}},\overline{\mathbf{w}}}$ witnesses comprehension for $Q(t[u,\overline{v}])$ when $\overline{v}$ is bound to $\overline{\mathbf{v}}$ and Q is bound to $\mathbf{Q}_{\chi,\overline{\mathbf{w}}}$.

Hence $\langle \mathcal{I}, \mathcal{A}\rangle$ is a model of $\forall L_2^{\mathrm{P}}(A)$. We outline the rest of the proof before continuing. Any model $\langle \mathcal{I}, \mathcal{A}\rangle$ satisfying (0)-(3) and $\mathcal{I}(W) = N$ is a model corresponding to $\langle \mathcal{I}', \mathcal{A}'\rangle$. We will show that (4) implies $\mathcal{I}(W) = N$. Hence, by a Lemma above, we have $\langle \mathcal{I}, \mathcal{A}\rangle \models \alpha^W \Rightarrow \langle \mathcal{I}', \mathcal{A}'\rangle \models \alpha$. Since $N = S$, this is equivalent to $\langle \mathcal{I}, \mathcal{A}\rangle \models \alpha^W \Rightarrow \langle \mathcal{I}', \mathcal{A}'\rangle \models \alpha^\nu$. We have $\langle \mathcal{I}, \mathcal{A}\rangle \models \alpha^W$ by soundness since $\forall L_2^{\mathrm{P}}(A) \vdash \alpha^W$ and $\langle \mathcal{I}, \mathcal{A}\rangle$ is a model of $\forall L_2^{\mathrm{P}}(A)$; hence $\langle \mathcal{I}', \mathcal{A}'\rangle \models \alpha^\nu$. Since $\langle \mathcal{I}', \mathcal{A}'\rangle$ was an arbitrary model of $\forall$QPV, completeness implies $\forall\mathrm{QPV} \vdash \alpha^\nu$. This finishes the main part of the Theorem.

All that remains is to prove $\mathcal{I}(W) = N$. Trivially $\mathcal{I}(W) \subseteq N$ since each set added to $\mathcal{I}$ is a subset of $U = N$. In the other direction, consider any $\mathbf{u} \in N = U$; we must show $\mathbf{u}$ is in every closed set $\mathbf{Q}_{\chi,\overline{\mathbf{v}}}$ of $\mathcal{I}$. By QPV3* we have $\forall\mathrm{QPV} \vdash \chi(0,\overline{v}) \wedge (\forall w.\chi(w,\overline{v}) = 0 \rightarrow \chi(s_0(w),\overline{v}) = 0 \wedge \chi(s_1(w),\overline{v}) = 0) \rightarrow (\forall u.\chi(u,\overline{v}) = 0)$. Using soundness, the corresponding semantic statement holds in $\langle \mathcal{I}', \mathcal{A}'^{\overline{v}}_{\overline{\mathbf{v}}}\rangle$. By the definition of $\mathbf{Q}_{\chi,\overline{\mathbf{v}}}$ this implies that $\mathbf{u} \in \mathbf{Q}_{\chi,\overline{\mathbf{v}}}$. when $\mathbf{Q}_{\chi,\overline{\mathbf{v}}}$ is closed. We are using $\mathbf{u} \in N$ because in the QPV conclusion $\forall u.\chi(u,\overline{v})$, the type of u is ν. ∎

13.3 Comments on an alternative development

Ideally we would liked to have carried out the development of $\mathrm{PV}_{\mathrm{BC}}$ using as a basis the functions generated by (a two-sorted version of) B. Recall B has functions p and C instead of mod. Indeed this seems to be the best way to carry out the program of giving a "resource-free" characterization of a feasible logic, insofar as that is possible. For example, B is a richer class than BC in that it computes some functions, such as p, on safe inputs which BC does not.

A further advantage of using B instead of BC would be the elimination of EQ as a "built-in" function, giving a system in which all the basic functions are constant-time computable. The elimination would go as follows: (1) do not permit atomic formulas in which equality ("=") is asserted between two safe variables, only allow it between a safe and a normal variable or between two normal variables; (2) show how to compute the function $\textsc{eq}(x;a)$ in B and prove the facts which are currently stated as axioms (A2). Then it appears that all the results proved above go through without change. Evidently, this elimination cannot be carried out for the system as formulated above with BC, because there is no obvious way to compute $\textsc{eq}(x;a)$ in BC.

However, as mentioned at the beginning of the last chapter, the use of B would introduce a substantial technical complication in proving that the corresponding subsystem of Leivant's logic has certain desired properties.

The complication is essentially this: replacing $\underline{\mathrm{mod}}$ with p and C invalidates the statement of the strong convergence theorem because $\mathrm{Cl}(Q)$ and $Q(a)$ do not imply $Q(p(;a))$. Still, the ordinary statement of convergence is true for p, and this is our ultimate concern. Assuming we did use p and C to formulate $\mathrm{QPV}(p, C)$, the problem could be worked around by replacing $p(;a)$ in $L_2^{\mathrm{P}}(A)$ with initial functions $D(x;a) = x \dot{-} a$ and $E(x;a) = a \dot{-} (a \dot{-} x)$, obtaining $L_2^{\mathrm{P}}(A)(D,E)$. These two functions do have a very strong convergence property and permit the definition $\hat{p}(w;a) \equiv E(D(x;s_1(D(x;a));a)$, which is a function equal to $p(;a)$ when w is polynomially larger than a. Given $\mathrm{QPV}(p, C)$ formula α, let α^{pW} in $L_2^{\mathrm{P}}(A)(D,E)$ be obtained by changing p to $\hat{p}$ in α^W; the resulting formula has new free variables $\overline{w}$. Then prove strong convergence for all subterms of α^{pW}. Formalizing the proof of the "Bounding Lemma for B" in the system $L_2^{\mathrm{P}}(A)(D,E)$ shows that we only need to define polynomially large values to substitute for $\overline{w}$, thus effectively giving $\hat{p} = p$. Of course such a substitution doesn't preserve sorts, but that's o.k. since we are in $L_2^{\mathrm{P}}(A)(D,E)$ instead of $\mathrm{QPV}(p, C)$.

This approach is not only messier to prove than that above, but also leads to messier statements of the theorems since a translation of $L_2^{\mathrm{P}}(A)$ terms would be required.

Part III

Closing

Chapter 14

Conclusion

The traditional definitions of computational complexity classes are obtained from a very general class of functions, such as all Turing-computable functions, by bounding a computational resource, such as space, time, or processor-time product. It is a difficult research problem to distinguish, within such a class, some structure which will help to differentiate the class from others.

This dissertation approaches the problem from the reverse direction: we define a class of functions using additional structuring mechanisms — such as type theory — and subsequently to investigate the complexity of the resulting class. We succeed in giving fundamentally new characterizations of several well known complexity classes. One of the characterizations is applied to contemporary research into feasible mathematics and the connection between logic and complexity theory, by demonstrating a direct connection between two previously unrelated lines of investigation. Specifically, we define a variant of a common first-order bounded arithmetic along the lines of Buss and Cook/Urquhart, and a variant of Leivant's second-order system, and prove that these two have equal power.

This research is related philosophically to the program of nominalist mathematics and the use of related ideas in feasible mathematics, because of our examination of predicative and impredicative definitions. We introduce a new idea to the discourse on predicativity and computational complexity, by attributing impredicativity to specific members of $\mathbb{N}$ rather than to subsets of $\mathbb{N}$, and doing this relative to a given computation. Relative to the fact that a computation has certain values given in their totality, references to other members of $\mathbb{N}$ are references to "impredicative values". They are values whose existence can only be known by using an impredicative concept, induction. This "relative impredicativity" applies in the setting of a computation.

In subsequently developing this idea of predicative (normal) versus impredicative (normal or safe) values in the context of feasible logic, we rely on traditional logic and model-theoretic techniques. We have variables which refer to possibly impredicative values, and in the corresponding models we identify such values as a distinct category of integers. These impredicative values, which are not known to satisfy any instances of induction but which do satisfy certain axioms, are contrasted with the nonstandard integers, which satisfy many instances of induction.

This discussion, in Part II, of a category of values which are *a priori* impredicative must be reconciled with the assertion in Part I that impredicativity is attributed to values *relative to* a particular computation and a particular input to the computation. The reconciliation might be made by thinking of the category of impredicative values as those which are impredicative relative to all computations and all inputs. This accords with the logical discussion in which all safe values satisfy the axioms A5 for all functions in the class and all values of the normal variables. However, this explanation misses a deeper point. Safe variables can also denote normal values ($N \subseteq S$ in the model) and in fact we think of the "ordinary" model as one in which $S = N$; other models are "extraordinary".[1] Thus, the appearance of a safe variable in the logic does not necessarily indicate a value which is impredicative *in itself*, but indicates that in the process of logical reasoning we may have used an impredicative concept and therefore we only have impredicative knowledge of the actual value. In this sense we can think of the occurrence of such a variable as indicating that the value is only impredicatively known *relative to the proof which has been carried out so far.* Hence we still have a sense of "relative impredicativity" in the logical setting.

In closing, I would like to speculate on an alternative line of investigation. First, we would like to make more explicit the notions of "predicatively known", and of knowledge relative to a specific computation. Second, we would like to investigate other tools besides type theory for structurally defining a class of functions or a feasible logic. It seems possible that the introduction of modality could accomplish both of these aims. Specifically, it might be possible to formally develop an adequate notion $\Box x$ indicating

[1]This distinction differs from that between standard and nonstandard models of arithmetic. Indeed, the model N of the normal sort could be either standard or nonstandard.

that value x is known to us through predicative investigation. The notion of applying modalities directly to objects has been investigated somewhat [Li] but remains sufficiently controversial and undeveloped that a major research program would be required to carry out this suggestion. On the logical end, one try to interpret $\Box\,\alpha$ as meaning that α is known to us through feasible (or predicative) reasoning. Even the ordinary use of modality in formulas containing first-order variables [Ga] is complex enough to raise tricky questions of intuition. Nevertheless, the idea of analyzing computational complexity by directly formalizing feasible or predicative knowledge has a powerful appeal.

Chapter 15

Appendix

In this appendix we prove facts in $\mathrm{PV_{BC}}$. The purpose is to demonstrate how to actually use the systems and to support certain claims made in the main text. We show that the extra equations stated in LQF7 are theorems of $\mathrm{PV_{BC}}$ when all the variables are made normal; that multivariate recursion and induction hold in QPV; and that there are QPV terms computing the propositions expressed by QPV formulae.

In some cases we allow ourselves to work in QPV. Not only is QPV a conservative extension of $\mathrm{PV_{BC}}$ but considering the "Alternate Characterization" of QPV in the main text, the translation from the proofs we give to true $\mathrm{PV_{BC}}$ proofs is usually mechanical. For proofs by induction, say proving α by induction on x, we write the two cases "$x = 0$" and "$xi \neq 0$", leaving implicit that in the latter case we are proving $\alpha(x) \to \alpha(xi)$ where either i is 1 or else i is 0 and $x \neq 0$. This method is discussed in the introductory section on $\mathrm{PV_{BC}}$. We use a similar notation when SURJ is applied to a variable, using classical reasoning.

Frequently the exact condition "i is 1 or else i is 0 and $x \neq 0$" is used implicitly in applying the axioms A5 for a particular function. We also leave out routine verbiage invoking the induction axiom of QPV (or induction rule, in $\mathrm{PV_{BC}}$). We also skip explanations of how to derive things in the predicate calculus.

Drop ($\dot{-}$) function

The 'drop' function was defined by:

$$\begin{aligned} x \div 0 &= x \\ x \div yj &= p((x \div y) \,\underline{\mathrm{mod}}\, x;) \quad \text{for } yj \neq 0 \end{aligned}$$

Here we have used $\underline{\mathrm{mod}}$ to convert the safe value $(x \div y)$ so that it can be input to the normal position of p.

Notation

In carrying out proofs in $\mathrm{PV}_{\mathrm{BC}}$, we adopt the convention that x, y, z, are normal variables and a, b, c are safe sort variables. The variables u, v, w are either normal (for QPV or $\mathrm{PV}_{\mathrm{BC}}$) or else are variables of $L_2^{\mathrm{P}}(A)$. The letters i, j, k, l, d are not symbols in the logic but are meta-symbols standing in for constant values in $\{0, 1\}$. As before, xi abbreviates $s_i(x)$.

When we write $\alpha[\overline{r}; \overline{t}]$ for a formula α, we mean that the terms $\overline{r}$ and $\overline{t}$ are substituted for the free variables of α (and all the free variables of α are so replaced). $\forall A$ is the closure of formulas in A by universal quantifiers.

The successor functions and p are assumed to bind tighter than other functions; for example $pu \div vi$ means $(pu) \div (vi)$.

$(1p)$ ______________________

$$p(xi) = x$$

Proof. If i is 1 then it's trivial by the axioms for p. If i is 0 then use SURJ on x: if $x = 0$ then by A2 $p(x0) = p0 = 0 = x$, if $x \neq 0$ then $p(x0) = x$ by the equations for p. ∎

(1 $\underline{\mathbf{mod}}$) ______________________

$$p(y \,\underline{\mathrm{mod}}\, x;) = py \,\underline{\mathrm{mod}}\, px$$

Proof. By surjectivity on y we have $y = (py)j$ for some $j \in \{0, 1\}$ (this hold even if $y = 0$).

Now use surjectivity on x. If $x = 0$, then $p(y \,\underline{\mathrm{mod}}\, x;) = p0 = 0 = (py \,\underline{\mathrm{mod}}\, 0) = (py \,\underline{\mathrm{mod}}\, px)$ as required. If $x \neq 0$ and $x = (px)0$, then

$$\begin{aligned} p(y \,\underline{\mathrm{mod}}\, x;) &= p((py)j \,\underline{\mathrm{mod}}(px)0;) \\ &= p((py)j \,\underline{\mathrm{mod}}(px)1;) && \text{A2} \\ &= p((py \,\underline{\mathrm{mod}}\, px)j) && \text{A2} \\ &= (py \,\underline{\mathrm{mod}}\, px) && (1p) \end{aligned}$$

∎

We use these facts without further comment or reference.

15.1 Lemmas for $\dot{-}$.

(1$\dot{-}$) ______________________

$$\begin{array}{ll} & yk \dot{-} x = (y \dot{-} x)0 \\ \vee & yk \dot{-} x = (y \dot{-} x)1 \end{array}$$

Proof.

$x = 0$: $yk \dot{-} 0 = yk = (y \dot{-} 0)k$.

$xi \neq 0$: By the induction hypothesis, and taking classical cases, we assume $yk \dot{-} x = (y \dot{-} x)j$ for some j. Using surjectivity, and again taking classical cases, we have $(y \dot{-} x) \,\underline{\mathrm{mod}}\, y = [p((y \dot{-} x) \,\underline{\mathrm{mod}}\, y;)]l$ for some l. Then

$$\begin{array}{lll} yk \dot{-} xi & = p((yk \dot{-} x) \,\underline{\mathrm{mod}}\, yk;) & \\ & = p((y \dot{-} x)j \,\underline{\mathrm{mod}}\, yk;) & \text{ind hyp} \\ & = p((y \dot{-} x)j;) \,\underline{\mathrm{mod}}\, p(yk;) & (1M) \\ & = (y \dot{-} x) \,\underline{\mathrm{mod}}\, y & \\ & = [p((y \dot{-} x) \,\underline{\mathrm{mod}}\, y)]l & \textsc{surj} \\ & = (y \dot{-} xi)l & \mathrm{A5}, \dot{-} \end{array}$$

∎

(2$\dot{-}$) ______________________

$$0 \dot{-} x = 0$$

Proof

$x = 0$: trivial.

$xi \neq 0$:

$$\begin{array}{lll} 0 \dot{-} xi & = p((0 \dot{-} x) \,\underline{\mathrm{mod}}\, 0) & \\ & = p0 & \text{A2 for } \underline{\mathrm{mod}} \\ & = 0 & \end{array}$$

∎

(3$\dot{-}$) ________________________________

$$p(y \dot{-} x) = py \dot{-} x.$$

Proof.

$y = 0$: $p(0 \dot{-} x) = p0 = 0 = 0 \dot{-} x = p0 \dot{-} x$ using (2$\dot{-}$).

$yj \neq 0$:

$$\begin{aligned} p(yj \dot{-} x) &= p((y \dot{-} x)0) && (1\dot{-}),\text{ case }(1) \\ &= y \dot{-} x && (1p) \\ &= p(yj) \dot{-} x && (1p) \end{aligned}$$

Case 2, $yj \dot{-} x = (y \dot{-} x)1$, is similar. ∎

(4$\dot{-}$) ________________________________

$$x \dot{-} x = 0$$

Proof.

$x = 0$: trivial.

$xi \neq 0$:

$$\begin{aligned} xi \dot{-} xi &= p(xi \dot{-} x) \\ &= p(xi) \dot{-} x && (4\dot{-}) \\ &= x \dot{-} x \end{aligned}$$

∎

(5$\dot{-}$) ________________________________

$$(y \dot{-} x)\,\underline{\mathrm{mod}}\,y = y \dot{-} x$$

Proof.

$y = 0$: $(0 \dot{-} x)\,\underline{\mathrm{mod}}\,0 = 0 = 0 \dot{-} x$ using (2$\dot{-}$).

$yj \neq 0$: Case 1:

$$\begin{aligned} yj \dot{-} x &= (y \dot{-} x)0 && (1\dot{-}) \\ &= ((y \dot{-} x)\,\underline{\mathrm{mod}}\,y)0 && \text{ind. hyp.} \\ &= ((y, \dot{-} x)0\,\underline{\mathrm{mod}}\,yj) && \text{A2 for }\underline{\mathrm{mod}} \\ &= (yj \dot{-} x)\,\underline{\mathrm{mod}}\,yj && \text{case }(1) \end{aligned}$$

Case 2, $yj \dot{-} x = (y \dot{-} x)1$, is similar. ∎

(6$\dot{-}$) ______________________

$$\begin{array}{llll} & & y \dot{-} x1 & = p(y \dot{-} x) \\ \wedge & x \neq 0 \rightarrow & y \dot{-} x0 & = p(y \dot{-} x) \end{array}$$

Proof. This is immediate from A5 for $\dot{-}$ and (5$\dot{-}$). ∎

At this point we have established that the natural equations for $\dot{-}$ hold when mod is omitted, and the extra equations (3$\dot{-}$): $p(y \dot{-} x) = py \dot{-} x$ and (4$\dot{-}$): $x \dot{-} x = 0$. We use these without further comment.

(7$\dot{-}$) ______________________

$$x \neq 0 \rightarrow y \dot{-} x = py \dot{-} px$$

Proof.

$x = 0$: trivial.

$xi \neq 0$:

$$\begin{aligned} y \dot{-} xi &= p(y \dot{-} x) \\ &= py \dot{-} x \\ &= py \dot{-} p(xi) \quad (1p) \end{aligned}$$

∎

(8$\dot{-}$) ______________________

$$x \neq 0 \rightarrow x \dot{-} px \neq 0$$

Proof.

$x = 0$: trivial.

$xi \neq 0$: Case 1, $x = 0$:

$$\begin{aligned} xi \dot{-} p(xi) &= xi \dot{-} x \\ &= xi \\ &\neq 0 \end{aligned}$$

Case 2, $x \neq 0$:

$$\begin{aligned} xi \dot{-} p(xi) &= xi \dot{-} x \\ &= p(xi) \dot{-} px && (7\dot{-}) \\ &= x \dot{-} px \\ &\neq 0 && \text{ind. hyp.} \end{aligned}$$

▮

$(9\dot{-})$ ______________________

$$(xi \dot{-} yj \neq 0 \wedge yj \neq 0) \rightarrow zk \dot{-} (xi \dot{-} yj) = z \dot{-} (x \dot{-} yj).$$

Proof. Since $yj \neq 0$ we can apply the axioms A5 for $\dot{-}$ to get $xi \dot{-} yj = p(xi \dot{-} y) = p(xi) \dot{-} y = x \dot{-} y$. Therefore $x \dot{-} y \neq 0$; using surjectivity, $x \dot{-} y = p(x \dot{-} y)d = (x \dot{-} yj)d \neq 0$ for some d in $\{0, 1\}$. Then we have

$$\begin{aligned} zk \dot{-} (xi \dot{-} yj) &= zk \dot{-} ((x \dot{-} yj)d) && \text{above} \\ &= p(zk \dot{-} (x \dot{-} yj)) && \text{A5 for } \dot{-} \\ &= p(zk) \dot{-} (x \dot{-} yj) && (3\dot{-}) \\ &= z \dot{-} (x \dot{-} yj) \end{aligned}$$

▮

15.2 Proving the extra equations (LQF7) in $\mathrm{PV_{BC}}$

In the main text we claimed that the extra equations stated in LQF7 are theorems of $\mathrm{PV_{BC}}$ when all the variables are made normal. We already proved surjectivity in the main text, and we proved $p(x \dot{-} y) = px \dot{-} y$ above; what remains is $y \,\underline{\mathrm{mod}}\, x = N(x, x, y)$, where N is defined by

$$\begin{aligned} \textsc{parity}(u) &= u \,\underline{\mathrm{mod}}\, 1 \\ N(u, 0, w) &= 0 \\ N(u, vj, w) &= C(\textsc{parity}(w \dot{-} (u \dot{-} vj)), N(u, v, w)0, N(u, v, w)1) \quad \text{for } vj \neq 0 \end{aligned}$$

$(1N)$ ______________________

$$xi \dot{-} y \neq 0 \rightarrow N(xi, y, zk) = N(x, y, z)$$

Proof.

$y = 0$: $N(xi, 0, zk) = 0 = N(x, 0, z)$.

$yj \neq 0$: Assume the antecedent $xi \dot{-} yj \neq 0$. Then

$$
\begin{aligned}
N(xi, yj, zk) &= N(xi, y, zk)\textsc{parity}(zk \dot{-} (xi \dot{-} yj)) && \\
&= N(xi, y, zk)\textsc{parity}(z \dot{-} (x \dot{-} yj)) && (9\dot{-}) \\
&= N(x, y, z)\textsc{parity}(z \dot{-} (x \dot{-} yj)) && \text{ind. hyp.} \\
&= N(x, yj, z) && \text{def'n}
\end{aligned}
$$

∎

(2N) ______________________

$$xi \neq 0 \rightarrow N(xi, x, zk) = N(x, x, z)$$

Proof. This follows from (1N) using the fact $xi \neq 0 \rightarrow xi \dot{-} x \neq 0$ ($8\dot{-}$). ∎

LQF7

$$z \,\underline{\mathrm{mod}}\, x = N(x, x, z)$$

Proof.

$x = 0$: $z \,\underline{\mathrm{mod}}\, 0 = 0 = N(x, 0, z)$

$xi \neq 0$:

$$
\begin{aligned}
N(xi, xi, zk) &= N(xi, x, zk)\textsc{parity}(zk \dot{-} (xi \dot{-} xi)) && \text{def'n} \\
&= N(xi, x, zk)\textsc{parity}(zk \dot{-} 0) && (4\dot{-}) \\
&= N(xi, x, zk)\textsc{parity}(zk) && \\
&= N(xi, x, zk)k && \text{def'n } \textsc{parity} \\
&= N(x, x, z)k && (2N) \\
&= (z \,\underline{\mathrm{mod}}\, x)k && \text{ind. hyp.} \\
&= (zk \,\underline{\mathrm{mod}}\, x1) && \text{A2} \\
&= (zk \,\underline{\mathrm{mod}}\, xi) && \text{A2}
\end{aligned}
$$

∎

15.3 Multivariate Induction.

15.3.1 Preliminary Facts.

Definition of ONES, $\bullet$, and OMAX.

$$
\begin{array}{ll}
\text{ONES}(0;) & = 0 \\
\text{ONES}(xi;) & = \text{ONES}(x;)1 \ \text{ for } xi \neq 0 \\
 & \\
y \bullet 0 & = \text{ONES}(y;) \\
y \bullet xi & = (y \bullet x)1 \\
 & \\
\text{OMAX}(x, y;) & = (x \dot{-} y) \bullet y
\end{array}
$$

The following are all proved quite directly using induction on x. (For (5$\bullet$), the base case is (2$\bullet$) and the step case follows from (3$\bullet$) or (4$\bullet$).)

(1$\bullet$) - (7$\bullet$)

$$
\begin{array}{lrl}
(1\bullet) & \text{ONES}(\text{ONES}(x)) & = \text{ONES}(x) \\
(2\bullet) & 0 \bullet x & = \text{ONES}(x) \\
(3\bullet) & y1 \bullet x & = (y \bullet x)1 \\
(4\bullet) & y \neq 0 \rightarrow y0 \bullet x & = (y \bullet x)1 \\
(5\bullet) & y \bullet x & = x \bullet y \\
(6\bullet) & y \dot{-} x & = y \dot{-} \text{ONES}(x;) \\
(7\bullet) & x \dot{-} (y \bullet x) & = 0 \\
(1\text{ONES}) & p(\text{ONES}(x)) & = \text{ONES}(p(x))
\end{array}
$$

(1OMAX)

$$x \dot{-} \text{OMAX}(x, y;) = 0$$

Proof.

$x = 0$: trivial by (2$\dot{-}$).

$xi \neq 0$: First consider the case $xi \dot{-} y = 0$. Then $\text{OMAX}(x, y;) = (xi \dot{-} y) \bullet y = \text{ONES}(y;)$ and $xi \dot{-} \text{ONES}(y;) = 0$ by (6$\bullet$) above. Otherwise,

$$
\begin{array}{ll}
xi \dot{-} \text{OMAX}(xi, y;) & = xi \dot{-} ((xi \dot{-} y) \bullet y) \quad \text{def} \\
 & = xi \dot{-} ((x \dot{-} y)k \bullet y)
\end{array}
$$

for some $k \in \{0,1\}$, by ($1\dot{-}$). Note we have $(x \dot{-} y)k \neq 0$ by the case considered above. This lets us apply the appropriate fact ($3\bullet$) or ($4\bullet$) above, to continue with

$$
\begin{aligned}
&= xi \dot{-} ((x \dot{-} y) \bullet y)1 \\
&= p(xi \dot{-} ((x \dot{-} y) \bullet y)) \\
&= p(xi) \dot{-} ((x \dot{-} y) \bullet y) \\
&= x \dot{-} ((x \dot{-} y) \bullet y) \\
&= 0 && \text{ind. hyp.}
\end{aligned}
$$

∎

(2OMAX) ______________________

$$y \dot{-} \text{OMAX}(x, y;) = 0$$

Proof.

$y = 0$: trivial by ($2\dot{-}$).

$yj \neq 0$: We have $(x \dot{-} yj) \bullet yj = p(x \dot{-} y) \bullet yj = (p(x \dot{-} y) \bullet y)1$. Therefore, setting $u \equiv p(x \dot{-} y)$,

$$
\begin{aligned}
yj \dot{-} ((x \dot{-} yj) \bullet yj) &= yj \dot{-} (u \bullet y)1 \\
&= p(yj \dot{-} (u \bullet y)) \\
&= p(yj) \dot{-} (u \bullet y) \\
&= y \dot{-} (u \bullet y) \\
&= 0 && (7\bullet)
\end{aligned}
$$

∎

(3OMAX) ______________________

$$p(\text{OMAX}(x, y)) = \text{OMAX}(px, py)$$

Proof.

$y = 0$:

$$
\begin{aligned}
p(\textsc{omax}(x, y)) &= p((x \dot{-} 0) \bullet 0) \\
&= p(x \bullet 0) \\
&= p(\textsc{ones}(x)) \\
&= \textsc{ones}(px) \qquad (1\textsc{ones}) \\
&= px \bullet 0 \\
&= \textsc{omax}(px, 0) \\
&= \textsc{omax}(px, py)
\end{aligned}
$$

$yj \neq 0$:

$$
\begin{aligned}
p(\textsc{omax}(x, yj)) &= p((x \dot{-} yj) \bullet yj) \\
&= p(((x \dot{-} yj) \bullet y)1) \\
&= (x \dot{-} yj) \bullet y \\
&= p(x \dot{-} y) \bullet y \\
&= (px \dot{-} y) \bullet y \\
&= (px \dot{-} p(yj)) \bullet p(yj) \\
&= \textsc{omax}(px, p(yj))
\end{aligned}
$$

∎

15.3.2 Propositional Terms.

It will be convenient to identify terms in $\mathrm{PV}_{\mathrm{BC}}$ which compute propositions.

The conditionals, $C(; a, b, c)$ and $\mathrm{cond}(x; b, c)$ were defined in the main text, and a couple of basic facts were proved there.

Define *propositional terms* by:

(i) 0 and 1 are propositional terms.

(ii) If P, Q, R are propositional terms, and a is a variable, then $C(; a, Q, R)$ and $C(; P, Q, R)$ are propositional terms.

It is easy to see that all propositions can be expressed this way; for example $(a \vee b) \wedge c \equiv C(; a, C(; b, 0, C(; c, 0, 1)), C(; c, 0, 1))$.

Lemma 15.3.1 *For every propositional term P,*

$$\vdash_{\mathrm{PV_{BC}}} (\textsc{parity}(;a) = 0 \rightarrow P = P[0/a]) \wedge (\textsc{parity}(;a) \neq 0 \rightarrow P = P[1/a])$$

Proof. The proof is by structural induction on P. It is trivial if P is 0 or 1. If $P \equiv C(;Q,R,S)$, or if $P \equiv C(;b,R,S)$ where b is not the same variable as a, then it follows by ordinary predicate logic from the induction hypothesis on Q, R, and S using substitutivity of equals (A1) in the term $C(;b,c,d)$. If $P \equiv C(;a,R,S)$ then by the facts about C plus the induction hypothesis on R and S, we have

$$\begin{array}{ll} \vdash_{\mathrm{PV_{BC}}} & (\textsc{parity}(;a) = 0 \rightarrow C(;a,R,S) = R = R[0/a] = C(;0,R[0/a],S[0/a])) \\ \wedge & (\textsc{parity}(;a) \neq 0 \rightarrow C(;a,R,S) = S = S[1/a] = C(;1,R[1/a],S[1/a])). \end{array}$$

∎

Lemma 15.3.2 *For any propositional terms P and Q such that P and Q are equal for all assignments of* 0 *and* 1 *to their variables,* $\vdash_{\mathrm{PV_{BC}}} P = Q$.

Proof. The proof is by induction on the number of free variables in P and Q.

To prove the induction step, suppose a is a variable in P or in Q; by the induction hypothesis $\mathrm{PV_{BC}} \vdash P[1/a] = Q[1/a]$ and $\mathrm{PV_{BC}} \vdash P[0/a] = Q[0/a]$. The lemma above gives

$$\begin{array}{rl} \mathrm{PV_{BC}} & \vdash (\textsc{parity}(;a) = 0 \rightarrow P = P[0/a] = Q[0/a] = Q) \\ & \wedge(\textsc{parity}(;a) \neq 0 \rightarrow P = P[1/a] = Q[1/a] = Q). \end{array}$$

Then the classical axiom $\textsc{parity}(;a) = 0 \vee \textsc{parity}(;a) \neq 0$ finishes it.

The base case, in which there are no free variables in P and Q, is proved by structural induction on P and Q (i.e. on the length of P and Q as strings). When P and Q are both 0 or 1 the result is trivial. Otherwise, one of them (say P) is $C(;R,S,T)$ where R, S and T have no free variables. Since R is either equal to 1 on all assignments or equal to 0 on all assignments, we can either apply the induction hypothesis on R and 0 to get $\mathrm{PV_{BC}} \vdash P = C(;0,S,T) = S$, or we can apply the induction hypothesis on R and 1 to get $\mathrm{PV_{BC}} \vdash P = C(;1,S,T) = T$. Taking for example $\mathrm{PV_{BC}} \vdash P = S$, we have S equal to Q on all assignments; by the induction hypothesis $\mathrm{PV_{BC}} \vdash S = Q$ giving $\mathrm{PV_{BC}} \vdash P = Q$. ∎

The propositional terms just described can be used to define characteristic functions:

for each formula α with free variables $\overline{x}$ and $\overline{a}$, there is a function symbol χ_α such that $\vdash_{\mathrm{PV_{BC}}} \chi_\alpha(\overline{x};\overline{a}) = 0$ iff α is true. It is obtained by constructing a propositional term for α and substituting $\textsc{eq}(;s,t)$ where α has the atomic formula $s = t$.

15.3.3 Multivariate Recursion and Induction.

For the proofs in this section we would like to make formal the fact we have been using implicitly in our notation for writing down proofs, namely that the condition "$xi \neq 0$" is equivalent to the condition "either i is 1, or i is 0 and $\vdash x \neq 0$." This will let us write down $\mathrm{PV_{BC}}$ proofs which have the condition "$xi \neq 0$" in the statement of the theorem itself, rather than simply as a meta-notation used to abbreviate the proof.

$(2p)$ ________________________

$$(x1 \neq 0) \wedge (x \neq 0 \leftrightarrow x0 \neq 0)$$

Proof. For the first conjunct, if $x1 = 0$ then $x1 = s_1(p(x1)) = s_1(0) \neq 0$, contradiction.

For the second conjunct, $p(x0) = x$ by $(1p)$, so $x0 = 0 \to x = p(x0) = p(0) = 0$. In the other direction, $x = 0 \to x0 = s_0(0) = 0$ by A2. ∎

With this fact, we can pass back and forth between the condition "either i is 1, or i is 0 and $\vdash x \neq 0$" which holds in the induction steps of our proofs and the condition "$\vdash xi \neq 0$" used below.

Multivariate Recursion

Let g and h_{ij} be function symbols ($i, j \in \{0,1\}$). Then there is a function symbol f such that, for all $i, j \in \{0,1\}$,

(i) $\vdash_{\mathrm{PV_{BC}}} \quad (x = 0 \vee y = 0) \to \quad f(x,y,\overline{z};\overline{a}) = g(x,y,\overline{z};\overline{a})$
(ii) $\vdash_{\mathrm{PV_{BC}}} \quad (xi \neq 0 \wedge yj \neq 0) \to \quad f(xi,yj,\overline{z};\overline{a}) = h_{ij}(x,y,\overline{z};\overline{a},f(x,y,\overline{z};\overline{a}))$

Proof. In the following, we omit the parameters $\overline{z}$ and $\overline{a}$ to simplify the notation. Define

$$
\begin{aligned}
&\texttt{(a)} \quad \hat{f}(w,0,x,y;) && = g(0,0) \\
&\texttt{(b)} \quad \text{for } vk \neq 0, && \\
&\qquad \hat{f}(w,vk,x,y) && = \begin{cases} g(x',y';) & \text{if } x'=0 \vee y'=0 \\ h_{\mathrm{PARITY}(x'),\mathrm{PARITY}(y')}(px',py';\hat{f}(w,v,x,y;)) & \text{otherwise} \end{cases} \\
&\qquad\qquad \text{where} && \\
&\qquad\qquad\qquad x' && \equiv x \dot{-} (w \dot{-} vk) \\
&\qquad\qquad\qquad y' && \equiv y \dot{-} (w \dot{-} vk) \\
&\texttt{(c)} \qquad f(x,y) && = \hat{f}(\textsc{omax}(x,y),\textsc{omax}(x,y),x,y)
\end{aligned}
$$

Here we are using a characteristic function to evaluate the condition $x' = 0 \vee y' = 0$. The expression $h_{\mathrm{PARITY}(x'),\mathrm{PARITY}(y')}$ is shorthand for the obvious construction $C(;\textsc{parity}(x'),\cdots)$.

The intention is to have $\hat{f}(w,v,x,y) = f(x \dot{-} (w \dot{-} v), y \dot{-} (w \dot{-} v))$.

`(i)`. Reasoning in QPV, assume $x = 0 \vee y = 0$.

Case 1: If $x = 0 \wedge y = 0$ then $\textsc{omax}(x,y) = \textsc{omax}(0,0) = 0$ and
$f(x,y) = \hat{f}(\textsc{omax}(x,y),0,x,y) = g(0,0) = g(x,y)$.

Case 2: If $x \neq 0 \vee \overline{y} \neq 0$ then $\textsc{omax}(x,y) \neq 0$. (Proof: $x = uk \neq 0 \wedge v = 0$ implies $\textsc{omax}(x,y) = (uk \dot{-} 0) \bullet 0 = uk \bullet 0 = (u \bullet 0)1 \neq 0$ Similarly if $y = uk \neq 0$, again using $(1\bullet)$-$(6\bullet)$.) Therefore equation `(b)` applies with $w = vk = \textsc{omax}(x,y)$. Noting $x \dot{-} (\textsc{omax}(x,y) \dot{-} \textsc{omax}(x,y)) = x$ and similarly for y, we have
$f(x,y) = \hat{f}(\textsc{omax}(x,y),\textsc{omax}(x,y),x,y) = g(x,y)$.

`(ii)`. Reasoning in QPV, assume $xi \neq 0 \wedge yj \neq 0$. We have $\textsc{omax}(xi,yj) = (xi \dot{-} yj) \bullet yj = ((x \dot{-} y) \bullet y)1 \neq 0$, so `(b)` above applies. As in Case 2 above,
$xi \dot{-} (\textsc{omax}(xi,yj) \dot{-} \textsc{omax}(xi,yj)) = xi$ and similarly for yj; neither of these is 0 and by the equations for $\textsc{parity}$, they have parities i and j respectively. Hence

$$
\begin{aligned}
f(xi,yj) &= \hat{f}(\textsc{omax}(xi,yj),\textsc{omax}(xi,yj),xi,yj) \\
&= h_{ij}(p(xi),p(yj);\hat{f}(\textsc{omax}(xi,yj),p\textsc{omax}(xi,yj),xi,yj;))
\end{aligned}
$$

We have $\textsc{omax}(xi,yj) \neq 0$, so by $(8 \dot{-})$ we have $\textsc{omax}(xi,yj) \dot{-} p(\textsc{omax}(xi,yj)) \neq 0$. By the Lemma below, this implies

$$\hat{f}(\text{OMAX}(xi, yj), p\text{OMAX}(xi, yj), xi, yj, \overline{z}; \overline{a}) = \hat{f}(p\text{OMAX}(xi, yj), p\text{OMAX}(xi, yj), p(xi), p(yj), \overline{z}; \overline{a}).$$

Continuing the equalities above we now have

$$\begin{aligned}
&= h_{ij}(p(xi), p(yj); \hat{f}(p\text{OMAX}(xi, yj), p\text{OMAX}(xi, yj), p(xi), p(yj);)) \\
&= h_{ij}(x, y; \hat{f}(\text{OMAX}(p(xi), p(yj)), \text{OMAX}(p(xi), p(yj)), x, y;)) && (3\text{OMAX}) \\
&= h_{ij}(x, y; \hat{f}(\text{OMAX}(x, y), \text{OMAX}(x, y), x, y;)) \\
&= h_{ij}(x, y; f(x, y;))
\end{aligned}$$

∎

Lemma Required for Multiple Recursion

$$\vdash_{\mathrm{PV_{BC}}} w \dot{-} v \neq 0 \rightarrow \hat{f}(w, v, x, y, \overline{z}; \overline{a}) = \hat{f}(pw, v, px, py, \overline{z}; \overline{a})$$

Proof. Again we omit $\overline{z}$ and $\overline{a}$ for clarity. The proof is by induction on v.

$v = 0$: $\hat{f}(w, 0, x, y) = g(0, 0;) = \hat{f}(pw, 0, x, y)$.

$vk \neq 0$: In $\mathrm{PV_{BC}}$ assume the antecedent $w \dot{-} vk \neq 0$. Define

$$\begin{aligned}
x' &\equiv x \dot{-}(w \dot{-} vk) \\
x'' &\equiv px \dot{-}(pw \dot{-} vk) \\
y' &\equiv y \dot{-}(w \dot{-} vk) \\
y'' &\equiv py \dot{-}(pw \dot{-} vk)
\end{aligned}$$

We have $x'' = px \dot{-} p(w \dot{-} vk)$ by $(3\dot{-})$, and since $w \dot{-} vk \neq 0$ we can use $(7\dot{-})$ to get $px \dot{-} p(w \dot{-} vk) = x \dot{-}(w \dot{-} vk)$; therefore $x'' = x'$. Similarly $y'' = y'$ in $\mathrm{PV_{BC}}$. Now by substitutivity of equals and the induction hypothesis,

$$\begin{aligned}
&\hat{f}(w, vk, x, y;) \\
&\quad = \begin{cases} g(x', y';) & \text{if } x' = 0 \vee y' = 0 \\ h_{\text{PARITY}(x';),\text{PARITY}(y')}(px', py'; \hat{f}(w, v, x, y;)) & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} g(x'', y'';) & \text{if } x'' = 0 \vee y'' = 0 \\ h_{\text{PARITY}(x'';),\text{PARITY}(y'')}(px'', py''; \hat{f}(w, v, x, y;)) & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} g(x'', y'';) & \text{if } x'' = 0 \vee y'' = 0 \\ h_{\text{PARITY}(x'';),\text{PARITY}(y'')}(px'', py''; \hat{f}(pw, v, px, py;)) & \text{otherwise} \end{cases} \\
&\quad = \hat{f}(pw, vk, px, py;)
\end{aligned}$$

∎

Multivariate Induction

Let $\alpha[x, y, \overline{z}; \overline{a}]$ be a formula with all variables replaced by $x, y, \overline{z}, \overline{a}$ as shown.

$$\begin{array}{lll}
\text{If } (\texttt{i}) \ \vdash_{\text{PV}_{\text{BC}}} & (x = 0 \vee y = 0) \rightarrow & \alpha[x, y, \overline{z}; \overline{a}] \\
\text{and } (\texttt{ii}) \ \vdash_{\text{PV}_{\text{BC}}} & (xi \neq 0 \wedge yj \neq 0) \rightarrow & \alpha[x, y, \overline{z}; \overline{a}] \rightarrow \alpha[xi, yj, \overline{z}; \overline{a}] \\
\text{then } (\texttt{iii}) \ \vdash_{\text{PV}_{\text{BC}}} & \alpha[x, y, \overline{z}; \overline{a}] &
\end{array}$$

Proof. As before we omit $\overline{z}$ and $\overline{a}$ for clarity.

Define $\hat{\alpha}[v, x, y;] \equiv \alpha[x \dot{-} (\text{OMAX}(x, y) \dot{-} v), y \dot{-} (\text{OMAX}(x, y) \dot{-} v)]$.

We prove $\vdash_{\text{PV}_{\text{BC}}} \text{OMAX}(x, y)1 \dot{-} v \neq 0 \rightarrow \hat{\alpha}(v, x, y)$.

Substituting $\text{OMAX}(x, y)$ for v then finishes the lemma, since one can easily prove $\text{OMAX}(x, y)1 \dot{-} \text{OMAX}(x, y) \neq 0$ by induction on y.

Write "w" for "$\text{OMAX}(x, y)$". Working in PV_{BC}, we prove $w1 \dot{-} v \neq 0 \rightarrow \hat{\alpha}[v, x, y]$ by induction on v.

$v = 0$: (Assume the antecedent $w1 \dot{-} v \neq 0$). Note $x \dot{-} (w \dot{-} v) = x \dot{-} w = 0$ by (1OMAX); similarly $y \dot{-} (w \dot{-} v) = 0$. Thus, in PV_{BC},

$$\begin{aligned}
\hat{\alpha}[0, x, y] \quad &\leftrightarrow \alpha[x \dot{-} (w \dot{-} v), y \dot{-} (w \dot{-} v)] \\
&\leftrightarrow \alpha[0, 0]
\end{aligned}$$

Using (i), we conclude $\alpha[0, x, y]$.

$vk \neq 0$: Assume the antecedent, $w1 \dot- vk \neq 0$; we need to prove $\hat{\alpha}[vk, x, y]$. Define

$$\begin{aligned} x' &\equiv x \dot- (w \dot- vk) \\ y' &\equiv y \dot- (w \dot- vk), \end{aligned}$$

so that $\hat{\alpha}[vk, x, y] \equiv \alpha[x', y']$. Still reasoning in $\mathrm{PV}_{\mathrm{BC}}$, we consider classical cases.

Case 1, $x' = 0 \vee y' = 0$. Using (i), we obtain $\alpha[x', y']$, which is identically the required conclusion $\hat{\alpha}[vk, x, y]$.

Case 2, otherwise, $x' \neq 0 \wedge y' \neq 0$. Define

$$\begin{aligned} x'' &\equiv x \dot- (w \dot- v) \\ y'' &\equiv y \dot- (w \dot- v) \end{aligned}$$

We have $vk \neq 0$ so by $(7\dot-)$ we have $w1 \dot- vk = p(w1) \dot- p(vk) = w \dot- v$; therefore $w1 \dot- vk \neq 0$ implies $w \dot- v \neq 0$. This lets us apply $(7\dot-)$ again:

$$\begin{aligned} x'' &= x \dot- (w \dot- v) \\ &= px \dot- p(w \dot- v) && (7\dot-) \\ &= px \dot- (w \dot- vk) \end{aligned}$$

Surjectivity implies $x = (px)l$ for some l either 0 or 1. Using $(1\dot-)$ there is an i in $\{0, 1\}$ such that

$$\begin{aligned} x' &= x \dot- (w \dot- vk) \\ &= (px)l \dot- (w \dot- vk) \\ &= [px \dot- (w \dot- vk)]i && (1\dot-) \\ &= x''i && \text{above} \end{aligned}$$

A similar line of reasoning implies $y' = y''j$ for some j in $\{0, 1\}$.

Neither $x''i$ nor $y''j$ is 0 because of the statement of Case 2, and applying (ii) we have

$$\begin{aligned} \hat{\alpha}[v, x, y] &\rightarrow \alpha[x'', y''] && \text{def'n} \\ &\rightarrow \alpha[x''i, y''j] && \texttt{(ii)} \\ &\rightarrow \alpha[x', y'] && \text{above} \\ &\rightarrow \hat{\alpha}[vk, x, y] && \text{def'n} \end{aligned}$$

We proved $w \dot- v \neq 0$, and the induction hypothesis is $w \dot- v \neq 0 \rightarrow \hat{\alpha}[v, x, y]$. Therefore we can conclude $\hat{\alpha}[vk, x, y]$ as required to complete the induction step. ∎

An example of the use of multivariate induction is the following proof required by section 10.2.

(2 **mod**) ______________________

$$y \dot{-} x = 0 \leftrightarrow y = y \,\underline{\mathrm{mod}}\, x$$

Proof. First show $y = 0 \vee x = 0 \rightarrow (y \dot{-} x = 0 \leftrightarrow y = y \,\underline{\mathrm{mod}}\, x)$. Taking classical cases, suppose $y = 0$, then $y \dot{-} x = 0 \dot{-} x = 0$ by $(2 \dot{-})$; also $y \,\underline{\mathrm{mod}}\, x = 0 \,\underline{\mathrm{mod}}\, x = 0 = y$ by the axiom (A2) for $\underline{\mathrm{mod}}$. Otherwise, $x = 0 \wedge y \neq 0$, in which case $y \dot{-} x = y \dot{-} 0 = y \neq 0$ by the equations for $\dot{-}$; also $y \,\underline{\mathrm{mod}}\, x = y \,\underline{\mathrm{mod}}\, 0 = 0$ by LQF7 and the equations for N.

Next show $xi \neq 0 \wedge yj \neq 0 \rightarrow ((y \dot{-} x = 0 \leftrightarrow y = y \,\underline{\mathrm{mod}}\, x) \rightarrow (yj \dot{-} xi = 0 \leftrightarrow yj = yj \,\underline{\mathrm{mod}}\, xi))$. Again using classical reasoning and the deduction theorem, assume $xi \neq 0 \wedge yj \neq 0$ and $y \dot{-} x = 0 \leftrightarrow y = y \,\underline{\mathrm{mod}}\, x$. By $(1 \,\underline{\mathrm{mod}})$ one has $p(yj \,\underline{\mathrm{mod}}\, xi) = y \,\underline{\mathrm{mod}}\, x$, and by the axioms (A2) of $\underline{\mathrm{mod}}$, $yj \neq 0 \rightarrow (y \,\underline{\mathrm{mod}}\, x)j = yj \,\underline{\mathrm{mod}}\, xi$. Therefore $y = y \,\underline{\mathrm{mod}}\, x \leftrightarrow yj = yj \,\underline{\mathrm{mod}}\, xi$. Furthermore $yj \dot{-} xi = 0 \leftrightarrow p(yj \dot{-} x) = 0 \leftrightarrow p(yj) \dot{-} x = 0 \leftrightarrow y \dot{-} x = 0$, by the facts $(1 \dot{-}) - (6 \dot{-})$. The required conclusion follows by classical reasoning since we have $yj \dot{-} xi = 0 \leftrightarrow y \dot{-} x = 0 \leftrightarrow y = y \,\underline{\mathrm{mod}}\, x \leftrightarrow yj = yj \,\underline{\mathrm{mod}}\, xi$. ∎

Chapter 16

References

[Al] B. Allen, "Arithmetizing Uniform NC", *Annals of Pure and Applied Logic*, v. 53, 1991, North Holland.

[BM] J. Bell, M. Machover, *A Course in Mathematical Logic*, North-Holland (Elsevier Science Publishers) 1977, 1986.

[BC] S. Bellantoni and S. Cook, "A New Recursion-Theoretic Characterization of the Polytime Functions", to appear in *computational complexity.* Extended Abstract appeared in *Proc. 24th Symposium on the Theory Of Computing*, 1992.

[Ben] J. Bennett, *On Spectra*, Ph. D. Thesis, Princeton University, 1962.

[Bl] S. Bloch, "Functional Characterizations of Uniform Log-depth and Polylog-depth Circuit Families", in *Annual Conference on Structure in Complexity Theory, 1992* (to appear).

[Bl2] S. Bloch, *Divide and Conquer in Parallel Complexity and Proof Theory*, Ph.D. Thesis, Department of Mathematics, University of California at San Diego, 1992.

[Bi] E. Bishop, "Mathematics as a numerical language", in *Intuitionism and Proof Theory*, eds Myhill et al., North-Holland, 1970.

[B] S. Buss, "Bounded Arithmetic", Ph.D. Dissertation, Princeton University, 1985; reprinted Bibliopolis, Naples, 1986.

[B2] S. Buss, private communication, Hotel Mazanka, Prague, July 1991.

[B3] S. Buss, "The polynomial hierarchy and intuitionistic bounded arithmetic", *Annual Conference on Structure in Complexity Theory*, Springer-Verlag Lecture Notes in Computer Science 223, 1986.

[Cl] P. Clote, "Sequential, machine-independent characterizations of the parallel complexity classes *AlogTIME*, AC^k, NC^k and NC." in *MSI Workshop on Feasible Mathematics*, Birkhauser, 1989. An earlier exposition appears as "A Sequential Programming Language for Parallel Complexity Classes", Technical Report BCCS-88-07, Department of Computer Science, Boston College, 1988.

[CT] P. Clote, G. Takeuti, "Exponential Time and Bounded Arithmetic", in *Annual Conference on Structure in Complexity Theory*, v. 1, 1986.

[Cob] A. Cobham, "The intrinsic computational difficulty of functions". In Y. Bar-Hillel ed., *Proc. of the 1964 International Congress for Logic, Methodology, and the Philosophy of Science*, p. 24-30. North Holland, Amsterdam, 1964.

[CF] K. Compton, C. LaFlamme, "An Algebra and a Logic for NC^1", in *Information and Computation* v. 87, p. 241-263, 1990.

[C] S. Cook, "Feasibly constructive proofs and the propositional calculus", in *Proc. 7th Symposium on the Theory Of Computing*, 1975.

[C2] S. Cook, "Computability and complexity of higher type functions." In *Proc. MSRI Workshop on Logic from Computer Science*, Y. Moschovakis, ed., Springer Verlag, 1990.

[C3] S. Cook, "The complexity of theorem proving procedures", *Proc. 3rd ACM Symposium on the Theory of Computing*, 1971.

[C4] S. Cook, "Computational Complexity of Higher Type Functions", *Proceedings of the International Congress of Mathematicians*, 1990.

[C5] S. Cook, "A Taxonomy of Problems with Fast Parallel Algorithms", in *Information and Control*, v. 64, n. 1-3, p. 2, 1985.

[CK] S. Cook and B. Kapron, "Characterizations of the Basic Feasible Functionals of Finite Type", in *Feasible Mathematics*, S. R. Buss and P. J. Scott eds., Birkhauser, 1990.

[CR] S. Cook, R Reckhow, "The Relative Efficiency of Propositional Proof Systems", *Journal of Symbolic Logic*, v. 44, n. 1, 1979.

[CU] S. Cook and A. Urquhart, "Functional interpretations of feasibly constructive arithmetic", *Annals of Pure and Applied Logic* (to appear). Extended abstract appears in *Proc. 21st STOC (1989)*.

[FeShSt] L. Fegaras, T. Sheard, D. Stemple, "Uniform Traversal Combinators: Definition, Use and Properties", in *11th International Conference on Automated Deduction*, June 1992.

[F] S. Feferman, "Systems of Predicative Analysis", in *The Philosophy of Mathematics*, J. Hintikka ed., series: "Oxford Readings in Philosophy", Oxford University Press, 1969.

[Fe] F. J. I. Ferreira, *Polynomial Time Computable Arithmetic and Conservative Extensions*, Ph.D. Thesis, Department of Mathematics, Pennsylvania State University, 1988.

[GG] D. Gabbay, F. Guenthner (eds.), *Handbook of Philosophical Logic* v. I, II, and III; D. Reidel Publishing Company, 1984.

[Ga] J. W. Garson, "Quantification in Modal Logic", in [GG] v.II.

[Gi] Girard, *Proof theory and logical complexity*, series: Studies in Proof Theory, Monographs; Bibliopolis, 1987.

[Gu] Y. Gurevich, "Algebras of Feasible Functions", *Proc. 24th IEEE Conference on Foundations of Computer Science*, p. 210-214, 1983.

[Ha] W. G. Handley, "Bellantoni and Cook's Characterization of Polynomial Time Functions", typescript, Department of Pure Mathematics, University of Leeds, August 1992.

[H] A. Heyting, *Intuitionism: an introduction*, North-Holland, 1966.

[Ig] A. Ignjatovic, "Delineating Classes of Computational Complexity via Second Order Theories with Weak Set Existence Principles (I)", manuscript, Department of Philosophy, Carnegie Mellon University, November 1991.

[I] N. Immerman, "Descriptive and Computational Complexity", AMS Short Course in Complexity, 1988.

[I2] N. Immerman, "Languages That Capture Complexity Classes", *SIAM Journal of Computing*, p. 760-778, v. 16 (1987).

[I3] N. Immerman, "Expressibility as a Complexity Measure", *Annual Conference on Structure in Complexity Theory*, **2**, 1987.

[K] B. Kapron, "Feasible Computation in Higher Types", Technical Report 249/91, Department of Computer Science, University of Toronto, 1991.

[KC] B. Kapron and S. Cook, "A New Characterization of Mehlhorn's Polynomial Time Functionals", in *Proceedings of the 32nd IEEE Conference on Foundations of Computer Science*, 1991.

[KPT] J. Krajicek, P. Pudlak, G. Takeuti, "Bounded Arithmetic and the Polynomial Hierarchy", *Annals of Pure and Applied Logic*, 1989.

[KT] J. Krajicek, G. Takeuti, "On Bounded Σ^1_1 Polynomial Induction", in *Feasible Mathematics*, S. Buss and P. Scott, eds., Birkhauser 1990.

[L] D. Leivant, "A foundational delineation of computational feasibility", in *Sixth Annual IEEE Symposium on Logic in Computer Science*, 1991. (Revised and expanded as "A foundational delineation of poly-time", submitted for publication, December 1991.)

[L2] D. Leivant, "Subrecursion and lambda representation over free algebras (Preliminary summary)", in *Feasible Mathematics*, S. Buss and P. Scott, eds., Birkhauser 1990.

[L3] D. Leivant, "Semantic characterizations of number theories", Technical Report CMU-CS-90-169, School of Computer Science, Carnegie Mellon University, 1990.

[L4] D. Leivant, "Characterization of Complexity Classes in Higher-Order Logic", in *Second Annual IEEE Symposium on Logic in Computer Science*, 1987. (?)

[LM1] D. Leivant, J.Y. Marion, "Capturing poly-time in a typed λ-calculus", Manuscript, Department of Computer Science, Indiana University, April 1992.

[LM2] D. Leivant, J.Y. Marion, "Purely applicative characterizations of complexity classes", Manuscript, Department of Computer Science, Indiana University, April 1992.

[Ler] M. Lerman, *Degrees of Unsolvability*, Series: Perspectives in Mathematical Logic, Springer-Verlag, 1983.

[Li] V. Lifschitz, "Calculable Natural Numbers" in *Intensional Mathematics*, S. Shapiro, ed., series: Studies in Logic v. 113, North-Holland, 1985.

[Lin] J. Lind, "Computing in logarithmic space", Project MAC Technical Report, 52, 1974, Massachusetts Institute of Technology.

[M] K. Mehlhorn, "Polynomial and abstract subrecursive classes", *JCCS* v.12, p.147-178, 1976.

[N] E. Nelson, *Predicative Arithmetic*, Princeton University Press, Princeton, 1986.

[O] J. Otto, "Categorical Characterization of Ptime I", manuscript, Dept. of Mathematics, McGill University, 1991.

[Pa] R. Parikh, "Existence and feasibility in arithmetic", *Journal of Symbolic Logic* v. 36, 1971.

[PW] J. Paris, A. Wilkie, "On the Scheme of Induction for Bounded Arithmetic Formulas", *Annals of Pure and Applied Logic* 35, **3**, 1987.

[P] R. Peter, *Recursive Functions*, Academic Press, 1967.

[Pi] T. Pitassi, "The Efficiency of Weak Formal Systems", Ph.D. Thesis, Department of Computer Science, University of Toronto, 1992.

[PBI] T. Pitassi, P. Beame, R. Impagliazzo, "Exponential Lower Bounds for the Pigeonhole Principle", TR 257/91, Department of Computer Science, University of Toronto.

[PU] T. Pitassi, A. Urquhart, "The Complexity of the Hajos Calculus", presented at the Workshop on Feasible Mathematics II, Mathematical Sciences Institute, Cornell University, May 1992.

[Ri] R. W. Ritchie, "Classes of Predictably Computable Functions", *Transactions of the American Mathematical Society*, v. 106, 1963.

[Ro] R. M. Robinson, "An essentially undecidable axiom system", *Proceedings of the International Congress of Mathematicians* v. 1, Cambridge, MA, 1950.

[R] H. E. Rose, *Subrecursion: functions and hierarchies*, Oxford Logic Guides 9, Clarendon Press, Oxford, 1984.

[S] A. Seth, "There is no Recursive Axiomatization for Feasible Functionals of Type 2" in *Proceedings of Logic in Computer Science*, 1992 (to appear).

[So] R. Soare, *Recursively Enumerable Sets and Degrees*, Series: Perspectives in Mathematical Logic, Springer-Verlag, 1987.

[T] G. J. Tourlakis, *Computability*, Reston, 1984.

[TD] A. Troelstra, D. van Dalen, *Constructivism in Mathematics: an introduction*, Series: Studies in Logic and the Foundations of Mathematics, v. 121, ed. Barwise et al., North-Holland.