Ordinal Arithmetic: Algorithms and Mechanization

Panagiotis Manolios (manolios@cc.gatech.edu)
College of Computing, CERCS Lab, Georgia Institute of Technology
801 Atlantic Drive, Atlanta, Georgia 30332-0280 U.S.A.
http://www.cc.gatech.edu/∼manolios

Daron Vroon (vroon@cc.gatech.edu)
College of Computing, CERCS Lab, Georgia Institute of Technology
801 Atlantic Drive, Atlanta, Georgia 30332-0280 U.S.A.
http://www.cc.gatech.edu/∼vroon

Abstract. Termination proofs are of critical importance for establishing the correct behavior of both transformational and reactive computing systems. A general setting for establishing termination proofs involves the use of the ordinal numbers, an extension of the natural numbers into the transfinite which were introduced by Cantor in the nineteenth century and are at the core of modern set theory. We present the first comprehensive treatment of ordinal arithmetic on compact ordinal notations and give efficient algorithms for various operations, including addition, subtraction, multiplication, and exponentiation.

Using the ACL2 theorem proving system, we implemented our ordinal arithmetic algorithms, mechanically verified their correctness, and developed a library of theorems that can be used to significantly automate reasoning involving the ordinals. To enable users of the ACL2 system to fully utilize our work required that we modify ACL2, *e.g.*, we replaced the underlying representation of the ordinals and added a large library of definitions and theorems. Our modifications are available starting with ACL2 version 2.8.

1. Introduction

Termination proofs are of critical importance for mechanically establishing that computing systems behave correctly. In the case of transformational systems, termination proofs allow us to go from partial correctness to total correctness [1]. Termination proofs are important even in the context of *reactive systems*, non-terminating systems that are engaged in on-going interaction with an environment (*e.g.*, network protocols, operating systems, distributed databases, pipelined machines, etc.): they are used to prove liveness properties *i.e.*, to show that some desirable behavior is not postponed forever. Proving termination amounts to showing that a relation is well-founded [2]. Any well-founded relation can be extended to a total order, giving rise to a well-ordered relation. From a basic theorem of set theory, we have that every well-ordered relation is isomorphic to an ordinal; thus, the ordinal numbers provide a general setting for establishing termination proofs.

1.1. Previous Work

The ordinal numbers were introduced by Cantor over 100 years ago and are at the core of modern set theory [10, 11, 12]. They are an extension of the natural numbers into the transfinite and are an important tool in logic, *e.g.*, after Gentzen's proof of the consistency of Peano arithmetic using the ordinal number ε_0 [21], proof theorists routinely use ordinals and ordinal notations to establish the consistency of logical theories [57, 62]. To obtain constructive proofs, constructive ordinals notations are employed. The general theory of ordinal notations was initiated by Church and Kleene [13] and is recounted in Chapter 11 of Roger's book on computability [50].

An early use of the ordinals for proving program termination is due to Alan M. Turing, who in 1949 wrote the following [64, 42].

> The checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops. To the pure mathematician it is natural to give an ordinal number. In this problem the ordinal might be $(n-r)\omega^2 + (r-s)\omega + k$.[1]

The automated reasoning community has studied the problem of formalizing the ordinals (as opposed to ordinal notations), focusing on proving known results. Dennis and Smaill studied higher-order heuristic extensions of rippling. They used ordinal arithmetic as a case study, which was implemented in λ*Clam*, a higher order proof planning system for induction. They were able to successfully plan standard undergraduate textbook problems using their system [14]. Paulson and Grabczewski have mechanized a good deal of set theory, including the proof that for any infinite cardinal, κ, we have $\kappa \otimes \kappa = \kappa$, most of the first chapter of Kunen's excellent book on set theory [30], and the equivalence of eight forms of the well-ordering theorem [49]. More recently, Paulson has mechanized the proof of the relative consistency of the axiom of choice and has proved the reflection theorem [48, 47]. Paulson and Grabczewski's efforts required reasoning about the ordinals and

[1] Readers familiar with the ordinals may suspect that Turing's measure function is not quite right, as $(i)\omega^j = \omega^j$ when i and j are positive integers. This seems to be purely a notational issue, *e.g.*, Turing uses the same convention in a paper on logics based on ordinals [63]. Using modern conventions, the measure function is written $\omega^2(n-r) + \omega(r-s) + k$.

were carried out with the Isabelle/ZF system [44, 46]. A version of the reflection theorem was also proved by Bancerek, using Mizar [3]. Another line of work is by Belinfante, who has used Otter to prove elementary theorems of ordinal number theory [4, 5, 6]. There is much more work that can be mentioned, but we end by listing some of the theorem proving systems for which there exists support for the ordinals: Nqthm [8], ACL2 [26], Coq [7], PVS [43], HOL [22], Isabelle [45], and Mizar [51].

Ordinal notations are used in the context of automated reasoning to prove termination. For example, the ordinals up to ε_0 are the basis for termination proofs in the ACL2 theorem proving system [26, 27], which has been used on some impressive industrial-scale problems by companies such as AMD, Rockwell Collins, Motorola, and IBM. ACL2 was used to prove that the floating-point operations performed by AMD microprocessors are IEEE-754 compliant [41, 54], to analyze bit and cycle accurate models of the Motorola CAP, a digital signal processor [9], and to analyze a model of the JEM1, the world's first silicon JVM (Java Virtual Machine) [24]. Termination proofs have played a key role in various projects that use ACL2 to verify reactive systems. For example, in [32], we develop a theory of refinement for reactive systems that has been used to mechanically verify protocols, pipelined machines, and distributed systems [35, 31, 60]. Ordinals have also played a key role in projects to implement polynomial orderings [39] and multiset relations [52]. The relationship between proof theoretic ordinals and term rewriting is explored in [15, 20].

1.2. The Ordinal Arithmetic Problem

Despite the fact that ordinals have been studied and used extensively by various communities for over 100 years, we have not been able to find a comprehensive treatment of arithmetic on ordinal notations. The *ordinal arithmetic problem* for a notational system denoting the ordinals up to some ordinal δ, is as follows: given α and β, expressions in the system denoting ordinals $< \delta$, is γ the expression corresponding to $\alpha \star \beta$, where $\star$ can be any of $+, -, \cdot$, exponentiation? Solving this problem amounts to defining algorithms for ordinal arithmetic on the notation system in question. The practical implications of a solution to the ordinal arithmetic problem is that it allows users of theorem proving systems such as ACL2 to think and reason about ordinals algebraically. Algebraic reasoning is more convenient and powerful than the previously available options, which required users to use the underlying representation of ordinals to both define measure functions and to reason about them.

In this paper, we present a solution to the ordinal arithmetic problem for a notational system denoting the ordinals up to ε_0. Partial solutions to this problem appear in various books and papers [57, 16, 20, 40, 58, 62], *e.g.*, it is easy to find a definition of $<$ for various ordinal notations, but we have not found any statement of the problem nor any comprehensive solution in previous work. One notable exception is the dissertation work of John Doner [19, 18]. Doner and Tarski (his adviser) study hierarchies of ordinal arithmetic operations. They give a transfinite recursive definition for binary operations O_γ for any ordinal γ. The operation O_0 corresponds to addition, O_1 corresponds to multiplication, O_2 corresponds to exponentiation (for the most part), and so on. Using this hierarchy of operations, Doner and Tarski define a generalization of the Cantor normal form. However, Doner and Tarski stop short of defining an ordinal notation. They give pseudo-algorithms for O_1, O_2, and O_3, but it is not immediately clear how to apply these to an ordinal notation to obtain algorithms. This was not within the scope of their work, as they were studying operations on the set-theoretic ordinals, not on ordinal notations.

We use a notational system that is exponentially more succinct than the one used in ACL2 before version 2.8 and we give efficient algorithms, whose complexity we analyze. A preliminary conference version of the results appeared in [36]. Here, we give a more comprehensive treatment and provide full complexity and correctness proofs.

Using the ACL2 theorem proving system, we implemented our ordinal arithmetic operations and mechanically verified the correctness of the implementations. In addition, we modified ACL2 by replacing its then current ordinal representation (in ACL2, version 2.7) with the exponentially more succinct representation we present in Section 2 [38]. (Both representations are based on the Cantor normal form.) We show that our changes do not affect the soundness of the ACL2 logic by exhibiting a bijection between our ordinal representation and the previous ACL2 representation, using ACL2 version 2.7. The modifications appear in ACL2 starting with version 2.8, which also includes a library of definitions and theorems that we engineered to significantly automate reasoning involving the ordinals. Previously, none of the ordinal arithmetic operations were defined in ACL2 and proving termination required defining functions to explicitly construct ordinals. With our library, users can ignore representational issues and can work in an algebraic setting. An example application is due to Sustik, who used a previous version of our library [37] to give a constructive proof of Dickson's lemma [61]. This is a key lemma in the proof of the termination of Buchberger's algorith for finding Gröbner bases, and Sustik made essential use of the ordinals and our library. With the version of the library described in this paper all the proof obligations involving the ordinals are discharged automatically.

There are many issues in developing such a library [38] and we discuss a number of them in this paper. As an example, note that the definitions of the ordinal arithmetic operations serve two purposes. They are used to reason about expressions in the ground (variable-free) case, by computation, and they are used to develop the various rewrite rules appearing in the library. For computation we prefer efficient algorithms, whereas for reasoning we prefer simple definitions. To satisfy these mutually exclusive requirements, we use ACL2's `defexec` mechanism, which allows us to compute efficiently in the ground case and to reason effectively in the general case, while guaranteeing soundness (see page 29). As a simple example of the kind of reasoning that can our library can perform, the following equation

$$7 < m < \omega \ \wedge \ 0 < n < \omega \quad \Rightarrow \quad 5^{(\omega^{\omega})} + \alpha \ < \ [\omega^n + w \cdot 3]^{(\omega^m \cdot 2 + \omega^5 \cdot 5)^{\omega}} + \beta$$

gets (correctly) reduced to $\alpha < \beta$.

1.3. Overview of the Paper

The paper consists of two main parts. The first part presents the theory of ordinal arithmetic on ordinal notations and absolutely no knowledge of ACL2 is assumed, as only the algorithms and their complexity are discussed. In the second section we discuss our implementation in the ACL2 theorem proving system, but we have tried to make the discussion as self-contained as possible.

The first part of the paper consists of six sections. We start in Section 2 by giving an overview of the set-theoretic ordinals and ordinal notations. In Section 3 we explain the model of computation we use for our complexity results and what theorems we prove to establish the correctness of our algorithms. The next four sections are concerned with algorithms, including correctness and complexity proofs, for comparing and recognizing ordinals, addition and subtraction, multiplication, and exponentiation, respectively.

The second part of the paper consists of Section 8, where we describe the implementations, in ACL2, of the algorithms presented in the first part of the paper. We mechanically verified the implementations, meaning that we proved that the ACL2 definitions terminate, that they satisfy the various algebraic properties satisfied by the corresponding set-theoretic analogues, that our new representation is isomorphic to the previous ACL2 representation, and so on. What we did not do is to mechanically prove the theorems in the first part of the paper, meaning that we did not prove that the implementations correspond to the set-theoretic ordinals—this would require that we embed set theory into ACL2—and we did not mechanically prove the complexity results. Our goal was to provide a library that partially automate reasoning about the ordinals, not to mechanically verify the theorems in part one of our paper. Nonetheless, some of reasoning about the ordinal arithmetic algorithms in ACL2 benefited from our analysis in the first part of the paper. We give an overview of the library of theorems we created and review the modifications we made to ACL2 version 2.8 to allow it to use our new representation and library. We conclude with Section 9, where we also outline future work.

2. Ordinals

2.1. Set Theoretic Ordinals

We review the theory of ordinals [17, 30, 57]. A relation $\prec$ is *well-founded* if every decreasing sequence is finite. A *woset* is a pair $\langle X, \prec \rangle$, where X is a set, and $\prec$ is a *well-ordering*, a total, well-founded relation, over X. Given a woset, $\langle X, \prec \rangle$, and an element $a \in X$, X_a is defined to be $\{x \in X | x \prec a\}$. An *ordinal* is a woset $\langle X, \prec \rangle$ such that for all $a \in X$, $a = X_a$. It follows that if $\langle X, \prec \rangle$ is an ordinal and $a \in X$, then a is an ordinal and that $\prec$ is equivalent to $\in$. In the sequel, we use lower case Greek letters to denote ordinals and $<$ or $\in$ to denote the ordering.

Given two wosets, $\langle X, \prec \rangle$ and $\langle X', \prec' \rangle$, a function $f : X \to X'$ is said to be an *isomorphism* if it is a bijection and for all $x, y \in X$, $x \prec y$ iff $f.x \prec' f.y$. Two wosets are said to be *isomorphic* if there exists an isomorphism between them. A basic result of set theory states that every woset is isomorphic to a unique ordinal. Given a woset $\langle X, \prec \rangle$, we denote the ordinal to which it is isomorphic as $Ord(X, \prec)$. Since every well-founded relation can be extended to a woset, we see that, as a setting for proving termination, the theory of the ordinals is as general as possible.

Given an ordinal, α, we define its *successor*, denoted α' to be $\alpha \cup \{\alpha\}$. There is clearly a minimal ordinal, $\emptyset$. It is commonly denoted by 0. The next smallest ordinal is $0' = \{0\}$ and is denoted by 1. The next is $1' = \{0, 1\}$ and is denoted by 2. Continuing in this manner, we obtain all the natural numbers. A *limit ordinal* is an ordinal > 0 that is not a successor. The set of natural numbers, denoted ω, is the smallest limit ordinal.

2.2. Ordinal Arithmetic

In this section we define addition, subtraction, multiplication, and exponentiation for the ordinals. After each definition, we list various properties; proofs can be found in texts on set theory [17, 30, 57].

Definition 1 $\alpha + \beta = Ord(A, <_A)$ where $A = (\{0\} \times \alpha) \cup (\{1\} \times \beta)$ and $<_A$ is the lexicographic ordering on A.

Ordinal addition satisfies the following properties.

$\alpha + 1 \;=\; \alpha'$	
$(\alpha + \beta) + \gamma \;=\; \alpha + (\beta + \gamma)$	(associativity)
$\beta < \gamma \;\Rightarrow\; \alpha + \beta < \alpha + \gamma$	(strict right monotonicity)
$\beta < \gamma \;\Rightarrow\; \beta + \alpha \leq \gamma + \alpha$	(weak left monotonicity)

Note that addition is not commutative, *e.g.*, $1 + \omega = \omega < \omega + 1$.

Definition 2 $\alpha - \beta$ is defined to be 0 if $\alpha \leq \beta$, otherwise, it is the unique ordinal, ξ such that $\beta + \xi = \alpha$.

Definition 3 $\alpha \cdot \beta = Ord(A, <_A)$ where $A = \beta \times \alpha$ and $<_A$ is the lexicographic ordering on A.

Ordinal multiplication satisfies the following properties.

$0 < n < \omega \;\Rightarrow\; n \cdot \omega = \omega$	
$(\alpha \cdot \beta) \cdot \gamma \;=\; \alpha \cdot (\beta \cdot \gamma)$	(associativity)
$(0 < \alpha \wedge \beta < \gamma) \;\Rightarrow\; \alpha \cdot \beta < \alpha \cdot \gamma$	(strict right monotonicity)
$\beta < \gamma \;\Rightarrow\; \beta \cdot \alpha \leq \gamma \cdot \alpha$	(weak left monotonicity)
$\alpha \cdot (\beta + \gamma) \;=\; (\alpha \cdot \beta) + (\alpha \cdot \gamma)$	(left distributivity)

Note that commutativity and right distributivity do not hold for multiplication, *e.g.*, $2 \cdot \omega = \omega < \omega \cdot 2$, and $(\omega + 1) \cdot \omega = \omega \cdot \omega < \omega \cdot \omega + \omega$.

Definition 4 Given any ordinal, α, we define exponentiation using transfinite recursion: $\alpha^0 = 1$, $\alpha^{\beta+1} = \alpha^\beta \cdot \alpha$, and for β a limit ordinal, $\alpha^\beta = \bigcup_{0<\xi<\beta} \alpha^\xi$.

Ordinal exponentiation satisfies the following properties, where an additive principal ordinal is an ordinal, β such that $\forall \alpha < \beta$, $\alpha + \beta = \beta$ (such ordinals always have the form ω^γ for some ordinal, $\gamma > 0$).

$1 < p < \omega \;\Rightarrow\; p^\omega = \omega$	
$\alpha^\beta \cdot \alpha^\gamma \;=\; \alpha^{\beta+\gamma}$	
$(\alpha^\beta)^\gamma \;=\; \alpha^{\beta \cdot \gamma}$	
$\alpha < \omega^\beta \;\Rightarrow\; \alpha + \omega^\beta = \omega^\beta$	(additive principal property)
$\alpha, \beta < \omega^\gamma \;\Rightarrow\; \alpha + \beta < \omega^\gamma$	(closure of additive principal ordinals)
$1 < \alpha \wedge \beta < \gamma \;\Rightarrow\; \alpha^\beta < \alpha^\gamma$	(strict right monotonicity)
$\beta < \gamma \;\Rightarrow\; \beta^\alpha \leq \gamma^\alpha$	(weak left monotonicity)

Using the ordinal operations, we can construct a hierarchy of ordinals: $0, 1, 2, \ldots, \omega, \omega+1, \omega+2, \ldots, \omega \cdot 2, \omega \cdot 2+1, \ldots, \omega^2, \ldots, \omega^3, \ldots,$ $\omega^\omega, \ldots,$ and so on. The ordinal $\omega^{\omega^{\omega^{\cdots}}}$ is called ε_0, and it is the smallest ordinal, α, for which $\omega^\alpha = \alpha$; such ordinals are called ε-ordinals. Our representation deals with the ordinals less than ε_0. It is based upon the Cantor Normal Form for ordinals [57], which we now define.

Theorem 1 For every ordinal $\alpha \neq 0$, there are unique $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_n (n \geq 1)$ such that $\alpha = \omega^{\alpha_1} + \cdots + \omega^{\alpha_n}$.

For every $\alpha \in \varepsilon_0$, we have that $\alpha < \omega^\alpha$, as ε_0 is the smallest ε-ordinal. Thus, we can add the restriction that $\alpha > \alpha_1$ for these ordinals. This is essentially the representation of ordinals used in ACL2 [26, 27]. However, as $\omega^\alpha \cdot k + \omega^\alpha = \omega^\alpha \cdot (k+1)$ and $n \in \omega$, we can collect like terms and rewrite the normal form as follows.

Corollary 1 *(Cantor Normal Form)* For every ordinal $\alpha \in \varepsilon_0$, there are unique $n, p \in \omega, \alpha_1 > \cdots > \alpha_n > 0$, and $x_1, \ldots, x_n \in \omega \backslash \{0\}$ such that $\alpha > \alpha_1$ and $\alpha = \omega^{\alpha_1} x_1 + \cdots + \omega^{\alpha_n} x_n + p$.

By the *size* of an ordinal under a representation, we mean the number of bits needed to denote the ordinal in that representation.

Lemma 1 The ordinal representation in Corollary 1 is exponentially more succinct than the representation in Theorem 1.

Proof Consider $\omega \cdot k$: it requires $O(k)$ bits with the representation in Theorem 1 and $O(\log k)$ bits with the representation in Corollary 1. $\square$

2.3. Representation

We use nested triples to represent our ordinals. These triples are denoted by square brackets, with commas delimiting the elements in the triple. Thus the triple containing `a`, `b`, and `c` appears as [`a`, `b`, `c`]. $\mathrm{CNF}(\alpha)$ denotes our representation of the ordinal α. If $\alpha \in \omega$, then $\mathrm{CNF}(\alpha) = \alpha$. Otherwise, α has a unique decomposition, $\alpha = \sum_{i=1}^{n} \omega^{\alpha_i} x_i + p$. When this is the case,

$$\mathrm{CNF}(\alpha) = [\mathrm{CNF}(\alpha_1), x_1, \mathrm{CNF}(\sum_{i=2}^{n} \omega^{\alpha_i} x_i + p)]$$

We now define several basic functions for manipulating ordinals in our notation. Some of our functions are partial, *i.e.*, they are not specified for all inputs. In such cases, we never use them outside of their intended domain. **finp** returns true if `a` is a natural number, and false if it is an infinite ordinal. **fe**, **fco**, and **rst** return the first exponent, first coefficient, and rest of an ordinal, respectively. If $\mathbf{finp}(\mathsf{a})$, $\mathbf{fe}(\mathsf{a}) = 0$, $\mathbf{fco}(\mathsf{a}) = \mathsf{a}$, and **rst** is not used on a. For an infinite ordinal of the form $[\mathsf{a}, \mathsf{b}, \mathsf{c}]$, $\mathbf{fe}([\mathsf{a}, \mathsf{b}, \mathsf{c}]) = \mathsf{a}$, $\mathbf{fco}([\mathsf{a}, \mathsf{b}, \mathsf{c}]) = \mathsf{b}$, and $\mathbf{rst}([\mathsf{a}, \mathsf{b}, \mathsf{c}]) = \mathsf{c}$.

3. Correctness and Complexity Concerns

In the following sections, we define algorithms for ordinal arithmetic and analyze their correctness and complexity. In this section, we provide a high-level overview and explain what exactly is entailed and what assumptions we make.

Taken together, the correctness proofs establish that the structure consisting of the set-theoretic ordinals up to ε_0 with the usual arithmetic operations, is isomorphic to the structure consisting of E_0, the set of expressions corresponding to ordinals in our representation, along with the corresponding arithmetic operations (for which we provide algorithms). The set-theoretic structure is $\langle \varepsilon_0, cmp, +, -, \cdot, exp \rangle$, where exp is ordinal exponentiation and cmp is a function that orders ordinals: given ordinals α and β, it returns lt if $\alpha < \beta$, gt if $\alpha > \beta$ and eq if $\alpha = \beta$. The other structure is $\langle E_0, \mathbf{cmp_o}, +_o, -_o, \cdot_o, \mathbf{exp_o} \rangle$, where the intended meaning of the functions should be clear. Showing that the two structures are isomorphic involves first exhibiting a bijection between ε_0 and E_0; a trivial consequence of results in the previous section is that CNF is such a bijection. Secondly, the proof requires showing that the corresponding functions are equivalent. To this end, we show that: $cmp(\alpha, \beta) = \mathbf{cmp_o}(\mathrm{CNF}(\alpha), \mathrm{CNF}(\beta))$, $\mathrm{CNF}(\alpha \star \beta) = \mathrm{CNF}(\alpha) \star_o \mathrm{CNF}(\beta)$, where $\star$ ranges over $\{+, -, \cdot\}$, and, lastly, $\mathrm{CNF}(exp(\alpha, \beta)) = \mathbf{exp_o}(\mathrm{CNF}(\alpha), \mathrm{CNF}(\beta))$.

Note that these proofs are not mechanically verified. To do so would require using a theorem prover that can reason both about ACL2 and set theory. But, we implement the algorithms in ACL2 and reason about the implementations. For example, we prove that the implementations terminate and that they satisfy the numerous properties that their set-theoretic counterparts satisfy. We also develop a powerful library for reasoning about the ordinals. The details are in Section 8.

For our complexity analysis, we assume that integers require constant space and that integer operations have constant running time. One can later account for the integer operations by using the fastest known algorithms. This approach allows us to focus on the interesting aspects of our algorithms, namely the aspects pertaining to the ordinal representations. To make explicit that arithmetic operations are be-

```
|a|  {the length of a}
    finp(a)   :   0
    true      :   1 + |rst(a)|

#a  {the size of a}
    finp(a)   :   1
    true      :   #fe(a) + #rst(a)
```

Figure 1.: The length and size functions used for complexity analysis.

ing applied to integers, we refer to the usual arithmetic operations on integers as $<_\omega$, $+_\omega$, $-_\omega$, $\cdot_\omega$, and exp_ω.

The complexity of the ordinal arithmetic algorithms is given in terms of the functions in Figure 1. In the figure, we use a sequence of *condition* : *result* forms to define functions: the conditions should be read from top to bottom until a condition that holds is found and then the corresponding result is returned. Note that the `true` condition always holds. We use this format for definitions throughout the paper.

4. Comparing and Recognizing Ordinals

In this section we present and analyze functions that recognize and compare ordinals. The definitions are given in Figure 2.

In the sequel, the ordinals α and β have the following Cantor normal form decompositions $\alpha = \sum_{i=1}^{n} \omega^{\alpha_i} x_i + p$ and $\beta = \sum_{i=1}^{m} \omega^{\beta_i} y_i + q$; in addition, $\mathtt{a}$, $\mathtt{a}_i$, $\mathtt{b}$, and $\mathtt{b}_j$ denote $\mathrm{CNF}(\alpha)$, $\mathrm{CNF}(\alpha_i)$, $\mathrm{CNF}(\beta)$, and $\mathrm{CNF}(\beta_j)$, respectively, for all $1 \leq i \leq n$ and $1 \leq j \leq m$.

We start with $\mathbf{cmp_o}$, the comparison function for ordinals corresponding to *cmp*. In Figure 2 we also define $<_o$, $\leq_o$, and $=_o$. These functions are not needed and in fact, are not used in the implementation, where for efficiency reasons we use $\mathbf{cmp_o}$ exclusively. The definition of $-_o$ (page 15) provides a nice example of where this is useful, as instead of computing both $\mathbf{fe}(\mathtt{a}) <_o \mathbf{fe}(\mathtt{b})$ and $\mathbf{fe}(\mathtt{a}) >_o \mathbf{fe}(\mathtt{b})$, we only compute $\mathbf{cmp_o}(\mathbf{fe}(\mathtt{a}), \mathbf{fe}(\mathtt{b}))$. The reason for including $<_o$, $\leq_o$, and $=_o$ is to make the presentation clearer, and we also use $>_o$ and $\geq_o$ where we find them useful.

Theorem 2 For all $\alpha, \beta \in \varepsilon_0$, $\mathbf{cmp_o}(\mathtt{a}, \mathtt{b}) = cmp(\alpha, \beta)$.

Proof The proof is by induction on the sizes of $\mathtt{a}$ and $\mathtt{b}$. The base case, where $\mathbf{finp}(\mathtt{a})$ or $\mathbf{finp}(\mathtt{b})$ holds, is straightforward.

```
cmp_ω(p,q)   {ordering on naturals}
   p <_ω q     :  lt
   q <_ω p     :  gt
   true        :  eq

cmp_o(a,b)   {ordering on ordinals}
   finp(a)  ∧  finp(b)              :  cmp_ω(a,b)
   finp(a)                          :  lt
   finp(b)                          :  gt
   cmp_o(fe(a),fe(b)) ≠ eq          :  cmp_o(fe(a),fe(b))
   cmp_ω(fco(a),fco(b)) ≠ eq        :  cmp_ω(fco(a),fco(b))
   true                             :  cmp_o(rst(a),rst(b))

a <_o b   {< for ordinals}
   cmp_o(a,b) = lt   :  true
   true              :  false

op(a)   {ordinal recognizer}
   finp(a)   :  a ∈ ω
   true      :  ¬finp(first(a))
                ∧ fco(a) ∈ ω
                ∧ 0 <_ω fco(a)
                ∧ op(fe(a))
                ∧ op(rst(a))
                ∧ fe(rst(a)) <_o fe(a)

a ≤_o b   {≤ for ordinals}
   cmp_o(a,b) = gt   :  false
   true              :  true

a =_o b   {= for ordinals}
   cmp_o(a,b) = eq   :  true
   true              :  false
```

Figure 2.: The ordinal ordering and recognizer algorithms.

For the induction step, we have that $\#\mathtt{a}$, $\#\mathtt{b} > 1$ and for all γ, δ if $\#\mathrm{CNF}(\gamma) < \#\mathtt{a}$ and $\#\mathrm{CNF}(\delta) < \#\mathtt{b}$, then $\mathbf{cmp_o}(\mathrm{CNF}(\gamma),\ \mathrm{CNF}(\delta)) = cmp(\gamma,\ \delta)$. There are 3 cases.

In the first, $\mathbf{cmp_o}(\mathtt{a}_1, \mathtt{b}_1) \neq eq$. If $\mathbf{cmp_o}(\mathtt{a}_1, \mathtt{b}_1) = lt$, then by the induction hypothesis, $\alpha_1 < \beta_1$. Thus $\omega^{\alpha_1} < \omega^{\beta_1}$. Thus, since $\sum_{i=2}^{n} \omega^{\alpha_i} x_i < \omega^{\alpha_1}$, $\alpha < \omega^{\beta_1} \leq \beta$ by the closure of additive principal ordinals under addition. Therefore, $cmp(\alpha,\beta) = lt = \mathbf{cmp_o}(\mathtt{a},\ \mathtt{b})$. By a similar argument, $\mathbf{cmp_o}(\mathtt{a}_1, \mathtt{b}_1) = gt \quad \equiv \quad cmp(\alpha,\beta) = \mathbf{cmp_o}(\mathtt{a},\mathtt{b}) = gt$.

In the next case, we have that $\mathbf{cmp_o}(\mathtt{a}_1, \mathtt{b}_1) = eq \ \wedge\ \mathbf{cmp}_\omega(x_1, y_1) \neq eq$. By induction hypothesis, $\alpha_1 = \beta_1$. Suppose that $\mathbf{cmp}_\omega(x_1, y_1) = lt$.

Again, we note that $\sum_{i=2}^{n} \omega^{\alpha_i} x_i < \omega^{\alpha_1}$. Thus $\alpha < \omega^{\alpha_1} x_1 + \omega^{\alpha_1}$, by the strict right monotonicity of ordinal addition. But then we have

$$\omega^{\alpha_1} x_1 + \omega^{\alpha_1} = \omega^{\alpha_1}(x_1 + 1) = \omega^{\beta_1}(x_1 + 1) \leq \omega^{\beta_1} y_1$$

Hence, $\alpha < \beta$, so $\mathbf{cmp_o}(\mathtt{a}_1, \mathtt{b}_1) = lt = cmp(\alpha, \beta)$. A similar argument establishes the case where $\mathbf{cmp}_\omega(x_1, y_1) = gt$.

In the final case, we have that $\mathbf{cmp_o}(\mathtt{a}_1, \mathtt{b}_1) = eq \ \wedge \ \mathbf{cmp}_\omega(x_1, y_1) = eq$. By the induction hypothesis, this means $\alpha_1 = \beta_1$. If $\mathbf{cmp_o}(\mathbf{rst}(\mathtt{a}), \mathbf{rst}(\mathtt{b})) = eq$, then by the induction hypothesis, $\sum_{i=2}^{n} \omega^{\alpha_i} x_i + p = \sum_{i=2}^{n} \omega^{\beta_i} y_i + q$ and we have $cmp(\alpha, \beta) = eq = \mathbf{cmp_o}(\mathtt{a}_1, \mathtt{b}_1)$.

If $\mathbf{cmp_o}(\mathbf{rst}(\mathtt{a}), \mathbf{rst}(\mathtt{b})) = lt$, then by the induction hypothesis, $\sum_{i=2}^{n} \omega^{\alpha_i} x_i + p < \sum_{i=2}^{n} \omega^{\beta_i} y_i + q$; hence we have $\alpha < \beta$. Therefore, $cmp(\alpha, \beta) = lt = \mathbf{cmp_o}(\mathtt{a}_1, \mathtt{b}_1)$. A similar argument establishes the case where $\mathbf{cmp_o}(\mathbf{rst}(\mathtt{a}), \mathbf{rst}(\mathtt{b})) = gt$. $\square$

Theorem 3 $\mathbf{cmp_o}(\mathtt{a}, \mathtt{b})$ runs in time $O(\min(\#\mathtt{a}, \#\mathtt{b}))$.

Proof In the worst case we simultaneously recur down `a` and `b`. In more detail, the complexity of this function is bounded by the recurrence relation

$$T(\mathtt{a}, \mathtt{b}) = \begin{cases} c, & \text{if } \mathbf{finp}(\mathtt{a}) \text{ or } \mathbf{finp}(\mathtt{b}) \\ T(\mathtt{a}_1, \mathtt{b}_1) + T(\mathbf{rst}(\mathtt{a}), \mathbf{rst}(\mathtt{b})) + c, & \text{otherwise} \end{cases}$$

for some constant value, c. It now follows by induction on the size of `a` and `b` that $T(\mathtt{a}, \mathtt{b}) \leq k \cdot \min(\#\mathtt{a}, \#\mathtt{b}) - t$ for any constants, k, t, such that $t \geq c$ and $k \geq c + t$. $\square$

We now analyze the complexity of **op**. At first glance it seems that the complexity is quadratic as **op** recurs both down the rest of `a` ($\mathbf{op}(\mathbf{rst}(\mathtt{a}))$) and into the exponent ($\mathbf{op}(\mathtt{a}_1)$). However, a closer examination reveals the following.

Theorem 4 $\mathbf{op}(\mathtt{a})$ runs in time $\mathrm{O}(\#\mathtt{a}(\log \#\mathtt{a}))$.

Proof The running time is bounded by the (non-linear) recurrence relation

$$T(\mathtt{a}) = \begin{cases} c, & \text{if } \mathbf{finp}(\mathtt{a}) \\ T(\mathtt{a}_1) + T(\mathbf{rst}(\mathtt{a})) + \min(\#\mathtt{a}_1, \#\mathbf{rst}(\mathtt{a})) + c, & \text{otherwise} \end{cases}$$

for some constant, c, by Theorem 3. We show by induction on $\#\mathtt{a}$, that $T(\mathtt{a}) \leq k(\#\mathtt{a})(\log \#\mathtt{a}) + t$ where k, t are constants such that $t \geq c$ and $k \geq 3t$. In the base case, we have $T(\mathtt{a}) = c \leq t$. For the induction step, let $x = \min(\#\mathtt{a}_1, \#\mathbf{rst}(\mathtt{a}))$ and $y = \max(\#\mathtt{a}_1, \#\mathbf{rst}(\mathtt{a}))$. Note

$$
\begin{array}{llll}
\multicolumn{4}{l}{\texttt{a} +_o \texttt{b} \quad \{\textit{ordinal addition}\}} \\
& \mathbf{finp}(\texttt{a}) \;\wedge\; \mathbf{finp}(\texttt{b}) & : & \texttt{a} +_\omega \texttt{b} \\
& \mathbf{fe}(\texttt{a}) <_o \mathbf{fe}(\texttt{b}) & : & \texttt{b} \\
& \mathbf{fe}(\texttt{a}) =_o \mathbf{fe}(\texttt{b}) & : & [\mathbf{fe}(\texttt{a}),\ \mathbf{fco}(\texttt{a}) +_\omega \mathbf{fco}(\texttt{b}),\ \mathbf{rst}(\texttt{b})] \\
& \mathbf{true} & : & [\mathbf{fe}(\texttt{a}),\ \mathbf{fco}(\texttt{a}),\ \mathbf{rst}(\texttt{a}) +_o \texttt{b}]
\end{array}
$$

***Figure 3.*:** The ordinal addition algorithm.

that $x + y = \#\texttt{a}$. We have:

$$
\begin{array}{lll}
& & T(\texttt{a}) \\
= & \{\text{Definition of } T\} & T(\texttt{a}_1) + T(\mathbf{rst}(\texttt{a})) + x + c \\
\leq & \{\text{Induction Hypothesis}\} & kx \log x + t + ky \log y + t + x + c \\
\leq & \{kx \geq 2t + x \text{ as } k \geq 3t\} & k(x \log x + y \log y + x) + c \\
= & \{\text{Log}\} & k \log(x^x y^y 2^x) + c \\
\leq & \{\langle \forall z \in \omega :: 2^z \leq \binom{2z}{z} \rangle\} & k \log(x^x y^y \binom{2x}{x}) + c \\
\leq & \{x \leq y\} & k \log(x^x y^y \binom{x+y}{x}) + c \\
\leq & \{\text{Binomial Theorem}\} & k \log(x+y)^{x+y} + c \\
= & \{t \geq c,\ x + y = \#\texttt{a}\} & k(\#\texttt{a}) \log(\#\texttt{a}) + t \ \square
\end{array}
$$

5. Ordinal Addition and Subtraction

The algorithm for ordinal addition is given in Figure 3. The main idea of the algorithm is to traverse `b` until an exponent is found that is $\leq$ the first exponent of `a`. We now prove the correctness of the algorithm and analyze its complexity.

Theorem 5 For all $\alpha, \beta \in \varepsilon_0$ $\mathrm{CNF}(\alpha + \beta) = \texttt{a} +_o \texttt{b}$.

Proof The proof is by induction on α. The key insight here (as it was for the proof of Theorem 2) is that $\sum_{i=2}^{n} \omega^{\alpha_i} x_i < \omega^{\alpha_1}$.

If $\alpha, \beta \in \omega$, then $\mathrm{CNF}(\alpha + \beta) = \texttt{a} +_o \texttt{b}$. Now suppose that $\beta > \omega$ and either $\alpha \in \omega$ or $\alpha_1 < \beta_1$. Then:

$$
\begin{array}{lll}
& & \alpha + \beta \\
= & \{\text{Definition of } \beta, \text{ arithmetic}\} & \alpha + \omega^{\beta_1}(1 + y_1 - 1) + \sum_{i=2}^{m} \omega^{\beta_i} y_i + q \\
= & \{\text{left distributivity}\} & \alpha + \omega^{\beta_1} + \omega^{\beta_1}(y_1 - 1) + \sum_{i=2}^{m} \omega^{\beta_i} y_i + q \\
= & \{\text{Additive principal property}\} & \omega^{\beta_1} + \omega^{\beta_1}(y_1 - 1) + \sum_{i=2}^{m} \omega^{\beta_i} y_i + q \\
= & \{\text{Definition of } \beta\} & \beta
\end{array}
$$

Next, suppose that $\alpha, \beta > \omega$ and $\alpha_1 = \beta_1$. Then:

$$\begin{array}{ll}
 & \alpha+\beta \\
= \{\text{Definition of } \alpha\} & \omega^{\alpha_1}x_1+\sum_{i=2}^{n}\omega^{\alpha_i}x_i+p+\beta \\
= \{\text{Additive principal property}\} & \omega^{\alpha_1}x_1+\beta \\
= \{\text{Definition of } \beta\text{, distributivity}\} & \omega^{\alpha_1}(x_1+y_1)+\sum_{i=2}^{m}\omega^{\beta_i}y_i+q
\end{array}$$

Note that $\mathrm{CNF}(\omega^{\alpha_1}(x_1+y_1)+\sum_{i=2}^{m}\omega^{\beta_i}y_i+q) = [\mathtt{a}_1, \mathtt{x}_1 +_o \mathtt{y}_1, \mathbf{rst}(\mathtt{b})]$, which matches the definition of $+_o$.

In the final case, we have that $\mathtt{a}_1 <_o \mathtt{b}_1$ and $\neg\mathbf{finp}(\mathtt{a})$. Now, $\alpha+\beta = \omega^{\alpha_1}x_1+\delta$, where $\delta = \sum_{i=2}^{n}\omega^{\alpha_i}x_i+p+\beta$. Since $\sum_{i=2}^{n}\omega^{\alpha_i}x_i+p < \omega^{\alpha_1}$ and $\beta < \omega^{\alpha_1}$, $\delta < \omega^{\alpha_1}$. Letting $\sum_{i=1}^{k}\omega^{\delta_i}z_i+r$ be the Cantor normal form decomposition of δ, we see that $\mathrm{CNF}(\alpha+\beta) = \mathrm{CNF}(\omega^{\alpha_1}x_1 + \sum_{i=1}^{k}\omega^{\delta_i}z_i+r)$, which by the induction hypothesis is $[\mathtt{a}_1, \mathtt{x}_1, \mathbf{rst}(\mathtt{a}) +_o \mathtt{b}]$. $\square$

Theorem 6 $\mathtt{a} +_o \mathtt{b}$ runs in time $\mathrm{O}(\min(\#\mathtt{a},\ |\mathtt{a}|\cdot\#\mathtt{b}_1))$.

Proof The running time of $\mathtt{a} +_o \mathtt{b}$ is given by the recurrence relation

$$T(\mathtt{a},\mathtt{b}) = \begin{cases} c, & \text{if } \mathbf{finp}(\mathtt{a}) \\ T(\mathbf{rst}(\mathtt{a}),\mathtt{b}) + k_1\min(\#\mathtt{a}_1,\#\mathtt{b}_1)+c, & \text{otherwise}\end{cases}$$

for some constants c and k_1, using Theorem 3. We use induction to show that $T(\mathtt{a},\mathtt{b}) \le k\cdot\min(\#\mathtt{a},|\mathtt{a}|\cdot\#\mathtt{b}_1)+c$, where k is a constant such that $k \ge k_1+c$. In the base case, $T(\mathtt{a},\mathtt{b}) = c = k\cdot\min(\#\mathtt{a},|\mathtt{a}|\cdot\#\mathtt{b}_1)+c$, since $|\mathtt{a}| = 0$. Otherwise, using the induction hypothesis, we have:

$$\begin{array}{l}
T(\mathtt{a},\mathtt{b}) \\
= \{\text{Definition of } T\} \\
T(\mathbf{rst}(\mathtt{a}),\mathtt{b}) + k_1\min(\#\mathtt{a}_1,\#\mathtt{b}_1)+c \\
\le \{\text{Inductive Hypothesis}\} \\
k\cdot\min(\#\mathbf{rst}(\mathtt{a}),|\mathbf{rst}(\mathtt{a})|\cdot\#\mathtt{b}_1)+c+k_1\min(\#\mathtt{a}_1,\#\mathtt{b}_1)+c \\
\le \{\text{Arithmetic, } k \ge k_1+c\} \\
k\cdot\min(\#\mathtt{a}_1+\#\mathbf{rst}(\mathtt{a}),|\mathbf{rst}(\mathtt{a})|\cdot\#\mathtt{b}_1+\#\mathtt{b}_1)+c \\
= \{\text{Definition of } \#\} \\
k\cdot\min(\#\mathtt{a},|\mathtt{a}|\cdot\#\mathtt{b}_1)+c \ \square
\end{array}$$

We now turn our attention to ordinal subtraction; our algorithm is given in Figure 4. Recall that $\alpha-\beta$ is defined to be 0 if $\alpha<\beta$ and otherwise to be the unique ordinal, ξ such that $\beta+\xi=\alpha$. One must be careful to avoid silly mistakes when subtraction involves infinite ordinals, *e.g.*, note that $(\omega+1)-1 \ne \omega$.

Theorem 7 For all $\alpha,\beta\in\varepsilon_0$, $\mathrm{CNF}(\alpha-\beta) = \mathtt{a} -_o \mathtt{b}$.

Proof It is easy to prove, using induction, that if $\alpha<\beta$, then $\mathtt{a} -_o \mathtt{b} = 0$.

When $\alpha\ge\beta$, the proof amounts to showing that $\mathtt{b} +_o (\mathtt{a} -_o \mathtt{b}) = \mathtt{a}$ and $\mathtt{a} -_o \mathtt{b}$ is in proper CNF form, and is by induction on $\#\mathtt{a}$ and $\#\mathtt{b}$.

```
a −o b  {ordinal subtraction}
   finp(a) ∧ finp(b) ∧ a ≤ω b   :  0
   finp(a) ∧  finp(b)           :  a −ω b
   fe(a) <o fe(b)               :  0
   fe(a) >o fe(b)               :  a
   fco(a) <ω fco(b)             :  0
   fco(a) >ω fco(b)             :  [fe(a), fco(a) −ω fco(b), rst(a)]
   true                         :  rst(a) −o rst(b)
```

Figure 4.: The ordinal subtraction algorithm.

If $\beta = \alpha$ this is trivial, since $\mathtt{b} +_o (\mathtt{a} -_o \mathtt{b}) = \mathtt{a}$. We now focus on the case where $\beta < \alpha$.

If $\mathbf{finp}(\mathtt{a})$ and $\mathbf{finp}(\mathtt{b})$, then $\mathtt{b} +_o (\mathtt{a} -_o \mathtt{b}) = \mathtt{a}$. If $\mathtt{b}_1 < \mathtt{a}_1$, then $\mathtt{b} +_o (\mathtt{a} -_o \mathtt{b}) = \mathtt{b} +_o \mathtt{a} = \mathtt{a}$. If $\mathtt{b}_1 = \mathtt{a}_1$ and $y_1 < x_1$, we have:

$$
\begin{array}{lll}
 & & \mathtt{b} +_o (\mathtt{a} -_o \mathtt{b}) \\
= & \{\text{Definition of } -_o\} & \mathtt{b} +_o [\mathtt{a}_1, x_1 - y_1, \mathbf{rst}(\mathtt{a})] \\
= & \{\text{Definition of } +_o\} & [\mathtt{a}_1, x_1 - y_1 + y_1, \mathbf{rst}(\mathtt{a})] \\
= & \{\text{Arithmetic}\} & [\mathtt{a}_1, x_1, \mathbf{rst}(\mathtt{a})] \\
= & \{\text{Definition of } \mathtt{a}\} & \mathtt{a}
\end{array}
$$

Also, note that $\mathbf{op}(\mathtt{a} -_o \mathtt{b})$ since $\mathbf{op}(\mathtt{a})$ and $x_1 > y_1$; hence, $x_1 - y_1 > 0$.

Finally, suppose $\mathtt{b}_1 = \mathtt{a}_1$ and $y_1 = x_1$; then $\mathbf{rst}(\mathtt{b}) < \mathbf{rst}(\mathtt{a})$. We now have:

$$
\begin{array}{lll}
 & & \mathtt{b} +_o (\mathtt{a} -_o \mathtt{b}) \\
= & \{\text{Definition of } -_o, +_o\} & [\mathtt{b}_1, y_1, \mathbf{rst}(\mathtt{b}) +_o (\mathbf{rst}(\mathtt{a}) -_o \mathbf{rst}(\mathtt{b}))] \\
= & \{\text{Ind. hypothesis, } x_1 = y_1,\ \mathtt{a}_1 = \mathtt{b}_1\} & \mathtt{a}\ \Box
\end{array}
$$

Theorem 8 $\mathtt{a} -_o \mathtt{b}$ runs in time $\mathrm{O}(\min(\#\mathtt{a}, \#\mathtt{b}))$.

The recursion relation for the complexity of this function is

$$
T(\mathtt{a}, \mathtt{b}) = \begin{cases} c, & \text{if } \mathbf{finp}(\mathtt{a}) \text{ or } \mathbf{finp}(\mathtt{b}) \\ k_1 \cdot \min(\#\mathtt{a}_1, \#\mathtt{b}_1) + T(\mathbf{rst}(\mathtt{a}), \mathbf{rst}(\mathtt{b})) + c, & \text{otherwise} \end{cases}
$$

for some constants, k_1, c. The proof that $T(\mathtt{a}, \mathtt{b}) \leq k \cdot \min(\#\mathtt{a}, \#\mathtt{b})$ is almost identical to that of Theorem 3. $\Box$

6. Ordinal Multiplication

A first attempt at defining multiplication is given in Figure 5. Later in this section we derive a more efficient (and more complicated) algo-

```
a *_o b   {ordinal multiplication}
   a = 0   ∨  b = 0      :  0
   finp(a)  ∧  finp(b)   :  a ·_ω b
   finp(b)               :  [fe(a), fco(a) ·_ω b, rst(a)]
   true                  :  [fe(a) +_o fe(b), fco(b), a *_o rst(b)]
```

Figure 5.: Ordinal multiplication, take one.

rithm, but its correctness depends on the correctness of $*_o$, which we now consider.

Lemma 2 For all $\alpha, \beta \in \varepsilon_0$, $x, y \in \omega$ such that $\beta, x > 0$, $\omega^\alpha x \cdot \omega^\beta y = \omega^{\alpha+\beta} y$.

Proof $\omega^\alpha x \cdot \omega^\beta y \ = \ \omega^\alpha (x \cdot \omega^\beta) y \ = \ \omega^\alpha \cdot \omega^\beta y \ = \ \omega^{\alpha+\beta} y$ $\square$

Theorem 9 For all $\alpha, \beta \in \varepsilon_0$, $\text{CNF}(\alpha \cdot \beta) = \mathtt{a} *_o \mathtt{b}$.

Proof The proof is by (transfinite) induction on β. The cases where $\alpha = 0$, $\beta = 0$, or $\alpha, \beta \in \omega$ are obvious. The remainder of the proof consists of two cases: $\beta < \omega$ holds or it does not.

If $\beta < \omega$, then $\beta > 0$ and $\alpha > \omega$. The base case, where $\beta = 1$ is straightforward. For the induction step we have:

$$
\begin{array}{lll}
 & & \text{CNF}(\alpha \cdot \beta) \\
= & \{\text{Subtraction, distributivity}\} & \text{CNF}(\alpha + (\alpha \cdot (\beta - 1))) \\
= & \{\text{Theorem 5, induction hypothesis}\} & \mathtt{a} +_o (\mathtt{a} *_o \text{CNF}(\beta - 1)) \\
= & \{\text{Definition of } *_o,\ \beta < \omega\} & \mathtt{a} +_o [\mathtt{a}_1, x_1 \cdot_\omega (\beta -_\omega 1), \mathbf{rst}(\mathtt{a})] \\
= & \{\text{Definition of } +_o\} & [\mathtt{a}_1, x_1 + (x_1 \cdot_\omega (\beta - 1)), \mathbf{rst}(\mathtt{a})] \\
= & \{\text{Distributivity of } \cdot_\omega\} & [\mathtt{a}_1, x_1 \cdot_\omega \beta, \mathbf{rst}(\mathtt{a})] \\
= & \{\text{Definition of } *_o\} & \mathtt{a} *_o \mathtt{b}
\end{array}
$$

For the final case we have that $\beta \geq \omega$ and $\alpha > 0$. First, we note that $\alpha \cdot \omega^{\beta_1} y_1 = \omega^{\alpha_1 + \beta_1} y_1$:

$$
\begin{array}{lll}
 & & \alpha \cdot \omega^{\beta_1} y_1 \\
\leq & \{\text{Weak left monotonicity of multiplication}\} & \omega^{\alpha_1}(x_1 + 1) \cdot \omega^{\beta_1} y_1 \\
= & \{\text{Lemma 2}\} & \omega^{\alpha_1 + \beta_1} y_1 \\
= & \{\text{Lemma 2}\} & \omega^{\alpha_1} x_1 \cdot \omega^{\beta_1} y_1 \\
\leq & \{\text{Weak left monotonicity of multiplication}\} & \alpha \cdot \omega^{\beta_1} y_1
\end{array}
$$

We now have:

$$
\begin{array}{lll}
 & & \mathrm{CNF}(\alpha \cdot \beta) \\
= & \{\,\text{Distributivity}\,\} & \mathrm{CNF}(\omega^{\alpha_1+\beta_1} y_1 + (\alpha \cdot \sum_{i=2}^{m} \omega^{\beta_i} y_i)) \\
= & \{\,\text{Theorem 5}\,\} & \mathrm{CNF}(\omega^{\alpha_1+\beta_1} y_1) +_o \mathrm{CNF}(\alpha \cdot \sum_{i=2}^{m} \omega^{\beta_i} y_i) \\
= & \{\,\text{Def. of CNF, ind. hyp.}\,\} & [\mathrm{CNF}(\alpha_1+\beta_1), y_1, 0] +_o \mathtt{a} *_o \mathrm{CNF}(\sum_{i=2}^{m} \omega^{\beta_i} y_i) \\
= & \{\,\text{Def. of } \mathbf{rst}\text{, Theorem 5}\,\} & [\mathtt{a}_1 +_o \mathtt{b}_1, y_1, 0] +_o \mathtt{a} *_o \mathbf{rst}(\mathtt{b}) \\
= & \{\,\mathbf{fe}(\mathtt{a} *_o \mathbf{rst}(\mathtt{b})) <_o \mathtt{a}_1 + \mathtt{b}_1\,\} & [\mathtt{a}_1 +_o \mathtt{b}_1, y_1, \mathtt{a} *_o \mathbf{rst}(\mathtt{b})] \\
= & \{\,\text{Definition of } *_o\,\} & \mathtt{a} *_o \mathtt{b} \ \square
\end{array}
$$

The problem with this definition is its running time. Note that this algorithm walks down $\mathtt{b}$, adding $\mathtt{a}_1$ to each exponent of $\mathtt{b}$. This is equivalent to adding some ordinal, $\mathtt{c}$ to a decreasing sequence of ordinals ($\mathtt{d}_1$, $\mathtt{d}_2$, ..., $\mathtt{d}_n$). Using the addition algorithm, we find that for each $\mathtt{d}_i$, $\mathbf{fe}(\mathtt{d}_i)$ is compared to each exponent of $\mathtt{c}$ until the first exponent of $\mathtt{c}$ such that $\mathbf{fe}(\mathtt{d}_i)$ is $\geq$ this exponent is found. But since the $\mathtt{d}_i$'s are decreasing, we know that $\mathbf{fe}(\mathtt{d}_i) \geq \mathbf{fe}(\mathtt{d}_{i+1})$. Therefore, if the jth exponent of $\mathtt{c}$ is $> \mathbf{fe}(\mathtt{d}_i)$, we know that it is $> \mathbf{fe}(\mathtt{d}_{i+1})$. This means that simply adding each element of the decreasing sequence to $\mathtt{c}$ is inefficient. If we can keep track of how many exponents of $\mathtt{c}$ we went through before adding $\mathtt{d}_i$, we can just skip over those when we add $\mathtt{d}_{i+1}$. These observations lead to the definitions in Figure 6, which provide a quicker way to compute multiplication.

Lemma 3 $\mathtt{d} <_o \mathtt{b} \quad\Rightarrow\quad \mathbf{c1}(\mathtt{a}, \mathtt{b}) \leq \mathbf{c1}(\mathtt{a}, \mathtt{d})$.

Proof The proof is by induction on $\mathbf{c1}(\mathtt{a}$,$\mathtt{b})$. If $\mathbf{c1}(\mathtt{a}, \mathtt{b}) = 0$, then this is trivially true since $\mathbf{c1}$ always returns a natural number. In the induction step, we have that $\mathbf{c1}(\mathtt{a}, \mathtt{b}) > 0$ and $\mathtt{d} <_o \mathtt{b}$. Thus, $\mathbf{fe}(\mathtt{d}) \leq_o \mathtt{b}_1 <_o \mathtt{a}_1$ by the definitions of $<_o$ and $\mathbf{c1}$. Finally, by the induction hypothesis, $\mathbf{c1}(\mathtt{a}, \mathtt{b}) = 1 + \mathbf{c1}(\mathbf{rst}(\mathtt{a}), \mathtt{b}) \leq 1 + \mathbf{c1}(\mathbf{rst}(\mathtt{a}), \mathtt{d}) = \mathbf{c1}(\mathtt{a}, \mathtt{d})$. $\square$

Lemma 4 $n \leq \mathbf{c1}(\mathtt{a}, \mathtt{b}) \quad\Rightarrow\quad \mathbf{c1}(\mathtt{a}, \mathtt{b}) = \mathbf{c2}(\mathtt{a}, \mathtt{b}, n)$.

Lemma 5 $\mathbf{padd}(\mathtt{a}, \mathtt{b}, \mathbf{c1}(\mathtt{a}, \mathtt{b})) = \mathtt{a} +_o \mathtt{b}$.

Proof The proof is by induction on $\mathbf{c1}(\mathtt{a}$,$\mathtt{b})$. If $\mathbf{c1}(\mathtt{a}, \mathtt{b}) = 0$, $\mathtt{b}_1 \leq_o \mathtt{a}_1$, so the lemma is clearly true. In the induction step, $\mathbf{c1}(\mathtt{a}, \mathtt{b}) > 0$, so $\mathtt{b}_1 <_o \mathtt{a}_1$ by the definition of $\mathbf{c1}$. Hence, $\mathbf{c1}(\mathbf{rst}(\mathtt{a}), \mathtt{b}) < \mathbf{c1}(\mathtt{a}, \mathtt{b})$. By the induction hypothesis, we know that $\mathbf{padd}(\mathtt{d}, \mathtt{b}, \mathbf{c1}(\mathtt{d}, \mathtt{b})) = \mathtt{d} +_o \mathtt{b}$ for all $\mathtt{d} <_o \mathtt{a}$. Thus

$$
\begin{array}{lll}
 & & \mathtt{a} +_o \mathtt{b} \\
= & \{\,\text{Definition of } +_o\,\} & [\mathtt{a}_1, \mathtt{x}_1, \mathbf{rst}(\mathtt{a}) +_o \mathtt{b}] \\
= & \{\,\text{Induction Hypothesis}\,\} & [\mathtt{a}_1, \mathtt{x}_1, \mathbf{padd}(\mathbf{rst}(\mathtt{a}), \mathtt{b}, \mathbf{c1}(\mathbf{rst}(\mathtt{a}), \mathtt{b}))] \\
= & \{\,\text{Definition of } \mathbf{c1}\,\} & [\mathtt{a}_1, \mathtt{x}_1, \mathbf{padd}(\mathbf{rst}(\mathtt{a}), \mathtt{b}, \mathbf{c1}(\mathtt{a}, \mathtt{b}) - 1)] \\
= & \{\,\text{Definition of } \mathbf{padd}\,\} & \mathbf{padd}(\mathtt{a}, \mathtt{b}, \mathbf{c1}(\mathtt{a}, \mathtt{b})) \ \square
\end{array}
$$

```
restn(a,n)   {n is a natural number}
   n = 0   :   a
   true    :   restn(rst(a),n-1)

c1(a,b)   {finds the index of the first exponent of a that is ≤ b1}
   fe(a) >o fe(b)   :   1 +ω c1(rst(a),b)
   true             :   0

c2(a,b,n)   {skips over the first n elements of a and then calls c1}
   true   :   n + c1(restn(a,n),b)

padd(a, b, n)   {skips over the first n elements of a and adds the rest to b}
   n = 0   :   a +o b
   true    :   [fe(a), fco(a), padd(rst(a),b,n - 1)]

pmult(a,b,n)   {pseudo-multiplication}
   a = 0  ∨  b = 0       :   0
   finp(a)  ∧  finp(b)   :   a ·ω b
   finp(b)               :   [fe(a), fco(a) ·ω b, rst(a)]
   true                  :   [padd(fe(a),fe(b),m),
                              fco(b),
                              pmult(a,rst(b),m)]
                                  where m = c2(fe(a),fe(b),n)

a ·o b   {quicker ordinal multiplication}
   true   :   pmult(a,b,0)
```

Figure 6.: An efficient algorithm for ordinal multiplication.

Theorem 10 $n \leq \mathbf{c1}(\mathtt{a}_1, \mathtt{b}_1) \quad \Rightarrow \quad \mathbf{pmult}(\mathtt{a}, \mathtt{b}, n) = \mathtt{a} *_o \mathtt{b}$.

Proof The proof is by induction on $|\mathtt{b}|$. If $\mathbf{finp}(\mathtt{b})$, then this is clearly true. For the induction step, we know $\neg(\mathbf{finp}(\mathtt{b}))$. The induction hypothesis tells us that for all $\mathtt{d}$ such that $|\mathtt{d}| < |\mathtt{b}|, n \leq \mathbf{c1}(\mathbf{fe}(\mathtt{d}), \mathtt{b}_1) \Rightarrow \mathbf{pmult}(\mathtt{d}, \mathtt{b}, n) = \mathtt{d} *_o \mathtt{b}$. Suppose $n \leq \mathbf{c1}(\mathtt{a}_1, \mathtt{b}_1)$. Then if we let $m = \mathbf{c2}(\mathtt{a}_1, \mathtt{b}_1, n)$, we have that $m = \mathbf{c1}(\mathtt{a}_1, \mathtt{b}_1)$ by Lemma 4. Therefore, we also know that $m \leq \mathbf{c1}(\mathtt{a}_1, \mathbf{fe}(\mathbf{rst}(\mathtt{b})))$ by Lemma 3. Thus, by the Induction Hypothesis we have the following.

$$
\begin{array}{lll}
 & & \mathbf{pmult}(\mathtt{a}, \mathtt{b}, n) \\
= & \{\text{Definition of } \mathbf{pmult}\} & [\mathbf{padd}(\mathtt{a}_1, \mathtt{b}_1, m), \mathtt{y}_1, \mathbf{pmult}(\mathtt{a}, \mathbf{rst}(\mathtt{b}), m)] \\
= & \{\text{Lemma 5}\} & [\mathtt{a}_1 +_o \mathtt{b}_1, \mathtt{y}_1, \mathbf{pmult}(\mathtt{a}, \mathbf{rst}(\mathtt{b}), m)] \\
= & \{\text{Induction hypothesis}\} & [\mathtt{a}_1 +_o \mathtt{b}_1, \mathtt{y}_1, \mathtt{a} *_o \mathbf{rst}(\mathtt{b})] \\
= & \{\text{Definition of } *_o\} & \mathtt{a} *_o \mathtt{b} \ \Box
\end{array}
$$

Corollary 2 For all $\alpha, \beta \in \varepsilon_0$, $\mathrm{CNF}(\alpha \cdot \beta) = \mathtt{a} \cdot_o \mathtt{b}$.

Proof Follows directly from Theorems 9 and 10. □

We now turn our attention to complexity issues. For the next lemmas and theorem, let $\mathtt{a}_1 = [\mathtt{d}_1, z_1, [\mathtt{d}_2, z_2, \ldots [\mathtt{d}_k, z_k, \mathtt{r}] \ldots]]$.

Lemma 6 $\mathbf{c1}(\mathtt{a}, \mathtt{b})$ takes time $\mathrm{O}(\sum_{i=1}^{\mathtt{c1(a,b)}+1} \min(\#\mathtt{a}_i, \#\mathtt{b}_1))$.

Proof In the worst case, we traverse $\mathtt{a}$, comparing $\mathtt{a}_i$ with $\mathtt{b}_1$. By Theorem 3, this takes $\mathrm{O}(\sum_{i=1}^{\mathtt{c1(a,b)}+1} \min(\#\mathtt{a}_i, \#\mathtt{b}_1))$ time. □

Lemma 7 $\mathbf{c2}(\mathtt{a}, \mathtt{b}, s)$ takes time $\mathrm{O}(s + \sum_{i=s+1}^{\mathtt{c1(a,b)}+1} \min(\#\mathtt{a}_i, \#\mathtt{b}_1))$.

Lemma 8 $\mathbf{padd}(\mathtt{a}, \mathtt{b}, s)$ runs in time $\mathrm{O}(\min(\#\mathbf{fe}(\mathbf{restn}(\mathtt{a}, s)), \#\mathtt{b}_1) + s)$ when $s \geq \mathbf{c1}(\mathtt{a}, \mathtt{b})$.

Proof Note that $\mathbf{fe}(\mathbf{restn}(\mathtt{a}, s)) \leq \mathtt{b}_1$ since the exponents of $\mathtt{a}$ are decreasing and $s \geq \mathbf{c1}(\mathtt{a},\mathtt{b})$. Hence, $\mathbf{restn}(\mathtt{a}, s) +_o \mathtt{b}$ requires one comparison and creates an answer in constant time. Therefore, by Theorem 3, **padd** takes $O(\min(\#\mathbf{fe}(\mathbf{restn}(\mathtt{a}, s)), \#\mathtt{b}_1) + s)$ time. □

Theorem 11 $\mathbf{pmult}(\mathtt{a}, \mathtt{b}, s)$ runs in time $\mathrm{O}(|\mathtt{a}_1||\mathtt{b}| + \#\mathbf{restn}(\mathtt{a}_1, s) + \#\mathtt{b})$ if $s \leq \mathbf{c1}(\mathtt{a}_1, \mathtt{b}_1)$.

Proof Let $m = \mathbf{c2}(\mathtt{a}_1, \mathtt{b}_1, s)$; then $m = \mathbf{c1}(\mathtt{a}_1, \mathtt{b}_1)$ by Lemma 4. Thus, using Lemmas 7 and 8, we can construct the following recurrence relation to bound the running time of **pmult**:

$$T(\mathtt{a}, \mathtt{b}, s) = \begin{cases} d, \quad \text{if } \mathbf{finp}(\mathtt{b}) \quad \vee \ \mathtt{a} = 0 \\ T(\mathtt{a}, \mathbf{rst}(\mathtt{b}), m) + k_1(s + \sum_{i=s+1}^{m+1} \min(\#\mathtt{d}_i, \#\mathbf{fe}(\mathtt{b}_1))) \\ \ +k_2(\min(\#\mathbf{fe}(\mathbf{restn}(\mathtt{a}_1, m)), \#\mathtt{b}_1) + m) + d, \quad \text{otherwise} \end{cases}$$

for some constants k_1, k_2, and d. We use induction on $|\mathtt{b}|$ to show that $T(\mathtt{a}, \mathtt{b}, s) \leq k \cdot (|\mathtt{a}_1||\mathtt{b}| + \#\mathbf{restn}(\mathtt{a}_1, s) + \#\mathtt{b})$ where $k \geq k_1 + k_2 + d$. This is true in the base case, because $\#\mathtt{b} = 1$ and $k \geq d$. For the induction step, we first note the following.

$$\begin{array}{l} k_1[s + \sum_{i=s+1}^{m+1} \min(\#\mathtt{d}_i, \#\mathbf{fe}(\mathtt{b}_1))] + k_2[\min(\#\mathbf{fe}(\mathbf{restn}(\mathtt{a}, m)), \#\mathbf{fe}(\mathtt{b}_1)) + m] + d \\ \leq \ \{ \text{Arithmetic, } m \geq s \} \\ k_1(m + \sum_{i=s+1}^{m} \#\mathtt{d}_i + \#\mathbf{fe}(\mathtt{b}_1)) + (k_2 + d)(\#\mathbf{fe}(\mathtt{b}_1) + m) \\ \leq \ \{ k \geq k_1 + k_2 + d \} \\ k(m + \sum_{i=s+1}^{m} \#\mathtt{d}_i + \#\mathbf{fe}(\mathtt{b}_1)) \end{array}$$

```
exp₁(k,a)   {raising a positive integer to an infinite ordinal power}
   fe(a) =ₒ 1      :   [fco(a), exp_ω(k,rst(a)), 0]
   finp(rst(a))    :   [[fe(a) −ₒ 1, fco(a), 0], exp_ω(k,rst(a)), 0]
   true            :   [[fe(a) −ₒ 1, 1, fe(c)], fco(c), 0]
                         where c = exp₁(k,rst(a))
```

Figure 7.: Ordinal exponentiation: raising a positive integer to an infinite power.

Combining this with the recurrence relation and using the induction hypothesis, we have:

$T(\mathtt{a}, \mathtt{b}, s)$
$\leq$ { Definition of T, earlier reasoning }
$T(\mathtt{a}, \mathbf{rst}(\mathtt{b}), m) + k(m + \sum_{i=s+1}^{m} \#\mathtt{d}_i + \#\mathbf{fe}(\mathtt{b}_1))$
$\leq$ { Induction Hypothesis }
$k(|\mathtt{a}_1||\mathbf{rst}(\mathtt{b})| + \#\mathbf{restn}(\mathtt{a}_1, m) + \#\mathbf{rst}(\mathtt{b}) + m + \sum_{i=s+1}^{m} \#\mathtt{d}_i + \#\mathbf{fe}(\mathtt{b}_1)) + d$
$\leq$ { Arithmetic, $m \leq |\mathtt{a}_1|$ }
$k(|\mathtt{a}_1||\mathtt{b}| + \#\mathbf{restn}(\mathtt{a}_1, m) + \sum_{i=s+1}^{m} \#\mathtt{d}_i + \#\mathbf{rst}(\mathtt{b}) + \#\mathbf{fe}(\mathtt{b}_1)) + d$
$=$ { Definitions of $\#$, **restn** }
$k(|\mathtt{a}_1||\mathtt{b}| + \#\mathbf{restn}(\mathtt{a}_1, s) + \#\mathbf{rst}(\mathtt{b}) + \#\mathbf{fe}(\mathtt{b}_1)) + d$
$<$ { $\#\mathbf{fe}(\mathtt{b}_1) < \#\mathtt{b}_1$ }
$k(|\mathtt{a}_1||\mathtt{b}| + \#\mathbf{restn}(\mathtt{a}_1, s) + \#\mathtt{b}) + d$ $\square$

Corollary 3 $\mathtt{a} \cdot_o \mathtt{b}$ runs in time $O(|\mathtt{a}_1||\mathtt{b}| + \#\mathtt{a}_1 + \#\mathtt{b})$.

7. Ordinal Exponentiation

Ordinal exponentiation is more complex than ordinal multiplication, and in an effort to increase clarity, we define exponentiation ($\mathbf{exp_o}$) using four helper functions: $\mathbf{exp_1}$, $\mathbf{exp_2}$, $\mathbf{exp_3}$, and $\mathbf{exp_4}$. We introduce them one at a time, proving the correctness and complexity of each before moving on to the next. The correctness and complexity of $\mathbf{exp_o}$ come at the end and follow directly from the results proved for the helper functions. Before reading further, the reader may want to try a few examples; a particularly revealing class of examples is $(\omega+1)^{\omega^\omega+k}$, where k ranges over the naturals.

The first helper function, $\mathbf{exp_1}$, is defined in Figure 7, and it is used to raise a positive integer to an infinite ordinal power. We proceed by proving that it is correct and analyzing its complexity.

Lemma 9 $\forall k, x \in \omega, \alpha \in \varepsilon_0$ such that $\alpha > 0$, $x > 0$, and $k > 1$, $k^{\omega^\alpha x} = \omega^{\omega^{\alpha-1} x}$

Proof $k^{\omega^\alpha x} = k^{\omega^{1+\alpha-1}x} = k^{\omega\cdot\omega^{\alpha-1}x} = (k^\omega)^{\omega^{\alpha-1}x} = \omega^{\omega^{\alpha-1}x}$ □

Lemma 10 $\forall k \in \omega, \alpha \in \varepsilon_0$ such that $\alpha > 0$ and $k > 1$, $k^\alpha = (\omega^{\sum_{i=1}^n \omega^{\alpha_i - 1} x_i}) k^p$

Proof Recall that the Cantor normal form decomposition of α is $\sum_{i=1}^n \omega^{\alpha_i} x_i + p$. The proof follows from Lemma 9. □

Theorem 12 For all $k \in \omega, \alpha \in \varepsilon_0$ such that $\alpha \geq \omega$ and $k > 1$, $\mathrm{CNF}(k^\alpha) = \mathbf{exp_1}(k, \mathsf{a})$.

Proof The proof is by induction on α. If $\alpha_1 = 1$, then $\alpha = \omega \cdot x_1 + p$ for some $x_1, p \in \omega$. Thus, we have:

$$
\begin{array}{lll}
 & & \mathrm{CNF}(k^\alpha) \\
= & \{\text{Definition of } \alpha\} & \mathrm{CNF}(k^{\omega\cdot x_1 + p}) \\
= & \{\text{Property of exponentiation}\} & \mathrm{CNF}(k^{\omega\cdot x_1} \cdot k^p) \\
= & \{\text{Lemma 9}\} & \mathrm{CNF}(\omega^{x_1} \cdot k^p) \\
= & \{\text{Theorem 9}\} & \mathrm{CNF}(\omega^{x_1}) \cdot_o k^p \\
= & \{\text{Definitions of CNF}, \mathbf{exp}_\omega\} & [x_1, 1, 0] \cdot_o \mathbf{exp}_\omega(k, p) \\
= & \{\text{Definition of } \cdot_o\} & [x_1, \mathbf{exp}_\omega(k, p), 0] \\
= & \{\text{Definition of } \mathbf{exp_1}\} & \mathbf{exp_1}(k, \mathsf{a})
\end{array}
$$

Likewise, if $\alpha_1 > 1$ and $\mathbf{finp}(\mathbf{rst}(\mathsf{a}))$, then $\alpha = \omega^{\alpha_1} x_1 + p$ for some $\alpha_1 \in \varepsilon_0$, $x_1, p \in \omega$. We now have:

$$
\begin{array}{lll}
 & & \mathrm{CNF}(k^\alpha) \\
= & \{\text{Definition of } \alpha\} & \mathrm{CNF}(k^{\omega^{\alpha_1} x_1 + p}) \\
= & \{\text{Property of exponentiation}\} & \mathrm{CNF}(k^{\omega^{\alpha_1} x_1} \cdot k^p) \\
= & \{\text{Lemma 9}\} & \mathrm{CNF}(\omega^{\omega^{\alpha_1 - 1} x_1} \cdot k^p) \\
= & \{\text{Corollary 2}\} & \mathrm{CNF}(\omega^{\omega^{\alpha_1 - 1} x_1}) \cdot_o k^p \\
= & \{\text{Definitions of CNF}, \mathbf{exp}_\omega\} & [[\mathsf{a}_1 -_o 1, x_1, 0], 1, 0] \cdot_o \mathbf{exp}_\omega(k, p) \\
= & \{\text{Definition of } \cdot_o\} & [[\mathsf{a}_1 -_o 1, x_1, 0], \mathbf{exp}_\omega(k, p), 0] \\
= & \{\text{Definition of } \mathbf{exp_1}\} & \mathbf{exp_1}(k, \mathsf{a})
\end{array}
$$

In the final case $\neg\mathbf{finp}(\mathbf{rst}(\mathsf{a}))$ holds and by the induction hypothesis $\forall \xi < \alpha$, $\mathrm{CNF}(k^\xi) = \mathbf{exp_1}(k, \mathrm{CNF}(\xi))$. Now, letting $\mathsf{c} = \mathbf{exp_1}(k, \mathbf{rst}(\mathsf{a}))$, we have:

$$
\begin{array}{lll}
 & & \mathrm{CNF}(k^\alpha) \\
= & \{\text{Lemma 10}\} & \mathrm{CNF}((\omega^{\sum_{i=1}^m \omega^{\alpha_i - 1} x_i}) k^p) \\
= & \{\text{Ordinal arithmetic}\} & \mathrm{CNF}(\omega^{\omega^{\alpha_1 - 1} x_1} (\omega^{\sum_{i=2}^m w^{\alpha_i - 1} x_i}) k^p) \\
= & \{\text{Lemma 10}, \cdot_o\} & \mathrm{CNF}(\omega^{\omega^{\alpha_1 - 1} x_1}) \cdot_o \mathrm{CNF}(k^{\sum_{i=2}^m w^{\alpha_i} x_i + p}) \\
= & \{\text{Ind. hyp., CNF}\} & [[\mathsf{a}_1 -_o 1, \mathsf{x}_1, 0], 1, 0] \cdot_o \mathbf{exp_1}(k, \mathbf{rst}(\mathsf{a})) \\
= & \{\mathsf{c} = [\mathbf{fe}(\mathsf{c}), \mathbf{fco}(\mathsf{c}), 0]\} & [[\mathsf{a}_1 -_o 1, \mathsf{x}_1, 0], 1, 0] \cdot_o [\mathbf{fe}(\mathsf{c}), \mathbf{fco}(\mathsf{c}), 0] \\
= & \{\text{Definition of } \cdot_o\} & [[\mathsf{a}_1 -_o 1, \mathsf{x}_1, 0] +_o \mathbf{fe}(\mathsf{c}), \mathbf{fco}(\mathsf{c}), 0] \\
= & \{\mathsf{a}_1 -_o 1 > \mathsf{a}_2 -_o 1\} & [[\mathsf{a}_1 -_o 1, \mathsf{x}_1, \mathbf{fe}(\mathsf{c})], \mathbf{fco}(\mathsf{c}), 0] \\
= & \{\text{Definition of } \mathbf{exp_1}\} & \mathbf{exp_1}(k, \mathsf{a}) \ \square
\end{array}
$$

```
exp2(a,k)  {raising a limit ordinal to a positive integer}
   true   :   [fe(a) ·o (k - 1), 1, 0] ·o a

natpart(a)  {returns the natural part of an ordinal}
   finp(a)   :   a
   true      :   natpart(rst(a))

limitp(a)  {returns true if a represents a limit ordinal}
   true   :   op(a)   ∧  ¬finp(a)   ∧  natpart(a) = 0

limitpart(a)  {returns the greatest ordinal, b, such that limitp(b) and b <o a}
   finp(a)   :   0
   true      :   [fe(a), fco(a), limitpart(rst(a))]
```

Figure 8.: Ordinal exponentiation: raising a limit ordinal to a positive integer.

Theorem 13 $\mathbf{exp_1}$ runs in time $\mathrm{O}(|\mathtt{a}|)$.

Proof Note that by Theorem 8, computing $\mathtt{a}_1 -_o 1$ takes constant time. The proof is now straightforward. $\square$

We now consider the second helper function, $\mathbf{exp_2}$, which is shown in Figure 8 and is used to raise a limit ordinal to a positive integer.

Lemma 11 For all $\mathtt{a}, \mathtt{b}$ such that $\mathbf{op}(\mathtt{a})$, $\mathbf{op}(\mathtt{b})$, $\mathbf{natpart}(\mathtt{b}) = 0$ and $\neg\mathbf{finp}(\mathtt{a})$, $\mathtt{a} \cdot_o \mathtt{b} = [\mathtt{a}_1, 1, 0] \cdot_o \mathtt{b}$.

Proof The proof is by induction on $|\mathtt{b}|$. If $\mathbf{finp}(\mathtt{b})$, then $\mathtt{b} = 0$ and $\mathtt{a} \cdot_o \mathtt{b} = 0 = [\mathtt{a}_1, 1, 0] \cdot_o \mathtt{b}$. For the induction step we have:

$$
\begin{array}{lll}
 & & \mathtt{a} \cdot_o \mathtt{b} \\
= & \{\,\text{Definition of } \cdot_o\,\} & [\mathtt{a}_1 + \mathtt{b}_1, \mathtt{y}_1, \mathtt{a} \cdot_o \mathbf{rst}(\mathtt{b})] \\
= & \{\,\text{Induction Hypothesis}\,\} & [\mathtt{a}_1 + \mathtt{b}_1, \mathtt{y}_1, [\mathtt{a}_1, 1, 0] \cdot_o \mathbf{rst}(\mathtt{b})] \\
= & \{\,\text{Definition of } \cdot_o\,\} & [\mathtt{a}_1, 1, 0] \cdot_o \mathtt{b} \ \square
\end{array}
$$

Theorem 14 For all $\alpha \in \varepsilon_0, k \in \omega$ such that $\alpha \geq \omega$, $\mathbf{limitp}(\mathtt{a})$, and $k > 1$, $\mathrm{CNF}(\alpha^k) = \mathbf{exp_2}(\mathtt{a}, k)$.

Proof The proof is by induction on k. If $k = 2$, then $\mathrm{CNF}(\alpha^k) = \mathrm{CNF}(\alpha^2) = \mathrm{CNF}(\alpha \cdot \alpha) = \mathtt{a} \cdot_o \mathtt{a}$. Applying Lemma 11, we get $[\mathtt{a}_1, 1, 0] \cdot_o$

```
exp3h(a,p,n,k)   {helper function for exp3}
   k = 1   :   (a ·o p) +o p
   true    :   padd(exp2(a,k) ·o p, exp3h(a,p,n,k-1), n)

exp3(a,k)   {raising an infinite ordinal to a positive integer power}
   k = 1       :   a
   limitp(a)   :   exp2(a,k)
   true        :   padd(exp2(c,k),
                          exp3h(c,natpart(a),n,k-1),
                          n)
                   where c = limitpart(a) and n = |a|
```

Figure 9.: Ordinal exponentiation: raising an infinite ordinal to a positive integer power.

$\mathtt{a} = \mathbf{exp_2}(\mathtt{a}, k)$. For the induction step we have:

$$
\begin{array}{lll}
 & & \mathrm{CNF}(\alpha^k) \\
= & \{\text{Ordinal arithmetic, } \cdot_o\} & \mathtt{a} \cdot_o \mathrm{CNF}(\alpha^{k-1}) \\
= & \{\text{Induction hypothesis}\} & \mathtt{a} \cdot_o \mathbf{exp_2}(\mathtt{a}, k-1) \\
= & \{\text{Definition of } \mathbf{exp_2}\} & \mathtt{a} \cdot_o [\mathtt{a}_1 \cdot_o (k-2), 1, 0] \cdot_o \mathtt{a} \\
= & \{\text{Definition of } \cdot_o\} & [\mathtt{a}_1 +_o (\mathtt{a}_1 \cdot_o (k-2)), 1, 0] \cdot_o \mathtt{a} \\
= & \{\text{Distr., Theorem 5, Corollary 2}\} & [\mathtt{a}_1 \cdot_o (k-1), 1, 0] \cdot_o \mathtt{a} \\
= & \{\text{Definition of } \mathbf{exp_2}\} & \mathbf{exp_2}(\mathtt{a}, k) \ \square
\end{array}
$$

Theorem 15 $\mathbf{exp_2}(\mathtt{a}, k)$ runs in time $\mathrm{O}(|\mathtt{a}_1||\mathtt{a}| + \#\mathtt{a})$

Proof Note that $\mathtt{a}_1 \cdot_o (k-1)$ takes constant time, since $k-1$ is of size 1. Also, note that $\#(\mathtt{a}_1 \cdot_o (k-1)) = \#\mathtt{a}_1$ and $|\mathtt{a}_1 \cdot_o (k-1)| = |\mathtt{a}_1|$. So, by Corollary 3, we have that the running time is $O(|\mathtt{a}_1||\mathtt{a}| + \#\mathtt{a})$. $\square$

The third helper function, $\mathbf{exp_3}$, is defined in Figure 9. It is used to raise an infinite ordinal to a positive integer power. The complexity analysis of $\mathbf{exp_3}$ will reveal that the running time depends on the positive integer power, *i.e.*, it is exponential in the number of bits needed to represent the integer. As a result, the complexity of our algorithm for exponentiation is exponential. We will show at the end of this section that we cannot do much better.

Lemma 12 For all $\mathtt{a}$ such that $\mathbf{op}(\mathtt{a})$ and $\alpha \geq \omega$, $\mathbf{exp_3h}(\mathtt{c}, p, |\mathtt{a}|, k) = (\sum_{i=0}^{k-1} \mathbf{exp_2}(\mathtt{c}, k-i) \cdot_o p) +_o p$ where $\mathtt{c} = \mathbf{limitpart}(\mathtt{a})$ and $p = \mathbf{natpart}(\mathtt{a})$ (the summation is with respect to $+_o$).

Proof Let $\mathtt{c} = \mathbf{limitpart}(\mathtt{a}) = [\mathtt{a}_1, x_1, [\mathtt{a}_2, x_2, \ldots [\mathtt{a}_n, x_n, 0] \ldots]]$. The proof is by induction on k. Clearly, by the definition of $\cdot_o$, the lemma holds when $k = 1$. For the induction step we have:

$$
\begin{array}{l}
\mathbf{exp_3h}(\mathsf{c}, p, n, k) \\
= \{\text{Definition of } \mathbf{exp_3h}\} \\
\mathbf{padd}(\mathbf{exp_2}(\mathsf{c}, k) \cdot_o p, \mathbf{exp_3h}(\mathsf{a}, p, n, k-1), n) \\
= \{\text{Induction hypothesis, arithmetic}\} \\
\mathbf{padd}(\mathbf{exp_2}(\mathsf{c}, k) \cdot_o p, (\sum_{i=1}^{k-1} \mathbf{exp_2}(\mathsf{c}, k-i) \cdot_o p) +_o p, n) \\
= \{\text{See immediately below}\} \\
(\sum_{i=0}^{k-1} \mathbf{exp_2}(\mathsf{c}, k-i) \cdot_o p) +_o p
\end{array}
$$

We justify the last step of the above proof by noting that:

$$\mathbf{exp_2}(\mathsf{c}, k) \cdot_o p = [\mathsf{a}_1 \cdot_o x +_o \mathsf{a}_1,\ x_1 \cdot p,\ [\mathsf{a}_1 \cdot_o x +_o \mathsf{a}_2,\ x_2,\ \ldots [\mathsf{a}_1 \cdot_o x +_o \mathsf{a}_n,\ x_n,\ 0] \ldots]]$$

where $x = k - 1$. That is, by the definition of $\mathbf{exp_3h}$, we have that $\mathbf{fe}(\mathbf{exp_3h}(\mathsf{a}, p, n, k-1)) = (\mathsf{a}_1 \cdot_o (k-1))$; thus, every exponent in $\mathbf{exp_2}(\mathsf{c}, k) \cdot_o p$ is greater than the first exponent of $\mathbf{exp_3h}(\mathsf{a}, p, n, k-1)$. $\square$

Theorem 16 For all $\alpha \in \varepsilon_0$, $k \in \omega$ such that $\alpha \geq \omega$ and $k > 0$, $\mathrm{CNF}(\alpha^k) = \mathbf{exp_3}(\mathsf{a}, k)$.

Proof Recall that $\alpha = \sum_{i=1}^{n} \omega^{\alpha_i} x_i + p$ and let $\delta = \sum_{i=1}^{n} \omega^{\alpha_i} x_i$. Note that $\mathrm{CNF}(\delta) = \mathbf{limitpart}(\mathsf{a})$. The case where $k = 1$ or $p = 0$ follows from Theorem 14. Otherwise, with the aid of Lemma 12, we can show that $\mathbf{exp_3}(\mathsf{a}, k) = \mathrm{CNF}(\delta^k + (\sum_{j=1}^{k-1} \delta^{k-j})p + p)$ and what remains is to show that $\alpha^k = \delta^k + (\sum_{j=1}^{k-1} \delta^{k-j})p + p$ for all $k > 0$. We do so by induction on k; note that the base case has already been addressed. For the induction step we have the following, where $\gamma = \sum_{j=1}^{k-2} \delta^{k-1-j}$ and $\xi = \delta^{k-1} + \gamma p + p$.

$$
\begin{array}{ll}
 & \alpha^k \\
= \{\text{Exponentiation}\} & \alpha^{k-1} \cdot \alpha \\
= \{\text{Induction hypothesis, def. of } \delta\} & \xi(\delta + p) \\
= \{\text{Distributivity}\} & \xi\delta + \xi p \\
= \{\text{Lemma 11, def. } \xi\} & \delta^{k-1}\delta + (\delta^{k-1} + \gamma p + p)p \\
= \{\delta^{k-1} > \gamma p, \text{ additive principal property}\} & \delta^k + \delta^{k-1}p + \gamma p + p \\
= \{\text{Distributivity, def } \gamma\} & \delta^k + (\sum_{j=1}^{k-1} \delta^{k-j})p + p \ \square
\end{array}
$$

We now analyze the complexity of $\mathbf{exp_3}$.

Lemma 13 $\mathbf{exp_3h}(\mathsf{a}, p, n, k)$ runs in time $\mathrm{O}(k(|\mathsf{a}_1||\mathsf{a}| + \#\mathsf{a}))$ when $\neg\mathbf{finp}(\mathsf{a})$, $\mathbf{limitp}(\mathsf{a})$, $p \in \omega, n = |\mathsf{a}|$, and $0 < k < \omega$.

Proof Note that for any c, the following hold: $\mathsf{c} \cdot_o p$ takes $O(1)$ time (by the definition of $\mathbf{pmult}$); $\mathsf{c} +_o p$ takes $O(|\mathsf{c}|)$ time (by Theorem 6); $|\mathbf{exp_2}(\mathsf{c}, k) \cdot_o p| = |\mathsf{c}|$ (see Theorem 15); and $\#(\mathbf{exp_2}(\mathsf{c}, k) \cdot_o p) = \#\mathsf{c}$

```
exp4(a,b)  {raising an infinite ordinal to an infinite power}
   true   :   [fe(a) ·o limitpart(b), 1, 0] ·o exp3(a,natpart(b))

expo(a,b)  {ordinal exponentiation (raises a to the b power)}
   b = 0   ∨  a = 1      :   1
   a = 0                 :   0
   finp(a)  ∧  finp(b)   :   expω(a,b)
   finp(a)               :   exp1(a,b)
   finp(b)               :   exp3(a,b)
   true                  :   exp4(a,b)
```

Figure 10.: The ordinal exponentiation algorithm.

(again, see Theorem 15). Now, $\mathbf{exp_3h}$ gets called $O(k)$ times and each call requires $O(|\mathsf{a}_1||\mathsf{a}| + \#\mathsf{a} + |\mathsf{a}|)$ time. Therefore, by Theorem 15 and the above observations, the total time is $O(k(|\mathsf{a}_1||\mathsf{a}| + \#\mathsf{a}))$. $\square$

Theorem 17 $\mathbf{exp_3}(\mathsf{a}, k)$ runs in time $\mathrm{O}(k \cdot (|\mathsf{a}_1||\mathsf{a}| + \#\mathsf{a}))$.

Proof This is a straightforward consequence of Lemma 13 and Theorem 15. $\square$

The fourth and final helper function, $\mathbf{exp_4}$, and $\mathbf{exp_o}$ are defined in Figure 10. We use $\mathbf{exp_4}$ to raise an infinite ordinal to an infinite power. The exponentiation function, $\mathbf{exp_o}$, is now simple to define, as all that is required is to invoke the appropriate helper function. We start by showing the correctness of $\mathbf{exp_4}$, after which the correctness of $\mathbf{exp_o}$ is immediate. We end the section by analyzing the complexity of $\mathbf{exp_4}$ and $\mathbf{exp_o}$, and we show that even though the complexity of $\mathbf{exp_o}$ is exponential, it is of the same order as the size of the resulting ordinal.

Lemma 14 For all $\alpha, \xi, z \in \varepsilon_0$ such that $a \geq \omega$ and $\xi > 0$, $\alpha^{\omega^\xi z} = \omega^{\alpha_1 \omega^\xi z}$.

Proof $\alpha^{\omega^\xi z} \leq (\omega^{\alpha_1+1})^{\omega^\xi z} = \omega^{(\alpha_1+1)\omega^\xi z} = \omega^{\alpha_1 \omega^\xi z} = (\omega^{\alpha_1})^{\omega^\xi z} \leq \alpha^{\omega^\xi z}$ $\square$

Theorem 18 For all $\alpha, \beta \in \varepsilon_0$, such that $\alpha, \beta \geq \omega$, $\mathrm{CNF}(\alpha^\beta) = \mathbf{exp_4}(\mathsf{a}, \mathsf{b})$.

Proof

$$\begin{array}{lll}
 & & \mathrm{CNF}(\alpha^{\beta}) \\
= & \{\,\text{Def. of } \beta\,\} & \mathrm{CNF}(\alpha^{\sum_{i=1}^{m}\omega^{\beta_i}y_i+q}) \\
= & \{\,\text{Ordinal arithmetic}\,\} & \mathrm{CNF}(\prod_{i=1}^{m}\alpha^{\omega^{\beta_i}y_i}\cdot\alpha^{q}) \\
= & \{\,\text{Lemma 14}\,\} & \mathrm{CNF}(\prod_{i=1}^{m}\omega^{\alpha_1\cdot\omega^{\beta_i}y_i}\cdot\alpha^{q}) \\
= & \{\,\text{Property of exponentiation}\,\} & \mathrm{CNF}(\omega^{\alpha_1\cdot\sum_{i=1}^{m}\omega^{\beta_i}y_i}\cdot\alpha^{q}) \\
= & \{\,\text{Theorem 16, Corollary 2}\,\} & [\mathtt{a}_1\cdot_o \mathbf{limitpart}(\mathtt{b}),1,0]\cdot_o \mathbf{exp_2}(\mathtt{a},q) \\
= & \{\,\text{Definition } \mathbf{exp_4}\,\} & \mathbf{exp_4}(\mathtt{a},\mathtt{b})\ \Box
\end{array}$$

Theorem 19 For all $\alpha, \beta \in \varepsilon_0, \mathrm{CNF}(\alpha^{\beta}) = \mathbf{exp_o}(\mathtt{a}, \mathtt{b})$.

Proof The proof follows from Theorems 12, 16, and 18. $\Box$

Lemma 15 $\#(\mathtt{a} +_o \mathtt{b}) \leq \#\mathtt{a} + \#\mathtt{b}$

Proof The proof is by induction on $\#\mathtt{a}$. $\Box$

Lemma 16 $\mathbf{limitp}(\mathtt{b}) \quad\Rightarrow\quad |\mathtt{a} \cdot_o \mathtt{b}| = |\mathtt{b}|$

Proof The proof is by induction on $|\mathtt{b}|$. $\Box$

Lemma 17 $\mathbf{limitp}(\mathtt{b}) \quad\Rightarrow\quad \#(\mathtt{a} \cdot_o \mathtt{b}) \leq \#\mathtt{a}_1|\mathtt{b}| + \#\mathtt{b}$

Proof The proof is by induction on the size of $\mathtt{b}$. $\Box$

Theorem 20 $\mathbf{exp_4}(\mathtt{a}, \mathtt{b})$ runs in time $\mathrm{O}(\mathbf{natpart}(\mathtt{b})[|\mathtt{a}||\mathtt{b}| + |\mathtt{a}_1||\mathtt{a}| + \#\mathtt{a}] + \#\mathbf{fe}(\mathtt{a}_1)|\mathtt{b}| + \#\mathtt{b})$.

Proof There are 3 operations that $\mathbf{exp_4}$ calls that take more than constant time. The first is $\mathbf{exp_3}(\mathtt{a}, \mathbf{natpart}(\mathtt{b}))$, which we showed runs in time $O(\mathbf{natpart}(\mathtt{b}) \cdot (|\mathtt{a}_1||\mathtt{a}| + \#\mathtt{a}))$. The second is $\mathtt{a}_1 \cdot_o \mathbf{limitpart}(\mathtt{b})$, which takes time $O(|\mathbf{fe}(\mathtt{a}_1)||\mathtt{b}| + \#\mathbf{fe}(\mathtt{a}_1) + \#\mathtt{b})$. The final operation is $[\mathtt{a}_1 \cdot_o \mathbf{limitpart}(\mathtt{b}), 1, 0] \cdot_o \mathbf{exp_3}(\mathtt{a}, \mathbf{natpart}(\mathtt{b}))$. If we let $\mathtt{c} = [\mathtt{a}_1 \cdot_o \mathbf{limitpart}(\mathtt{b}), 1, 0]$ and $\mathtt{d} = \mathbf{exp_3}(\mathtt{a}, \mathbf{natpart}(\mathtt{b}))$, we obtain a time bound of $O(|\mathbf{fe}(\mathtt{c})||\mathtt{d}| + \#\mathbf{fe}(\mathtt{c}) + \#\mathtt{d})$. By Lemmas 16 and 17, among others, we have that $|\mathbf{fe}(\mathtt{c})| = |\mathtt{b}|$, $\#\mathbf{fe}(\mathtt{c}) = \#\mathbf{fe}(\mathtt{a}_1)|\mathtt{b}| + \#\mathtt{b}$, $|\mathtt{d}| = |\mathtt{a}| \cdot \mathbf{natpart}(\mathtt{b})$, and $\#\mathtt{d} = \#\mathtt{a} \cdot \mathbf{natpart}(\mathtt{b})$.

Hence, the complexity of this operation is $O(|\mathtt{b}|(|\mathtt{a}|\mathbf{natpart}(\mathtt{b})) + \#\mathbf{fe}(\mathtt{a}_1)|\mathtt{b}| + \#\mathtt{b} + \#\mathtt{a} \cdot \mathbf{natpart}(\mathtt{b}))$, which gives an overall complexity for the algorithm of

$$\begin{aligned}
O(&\mathbf{natpart}(\mathtt{b}) \cdot (|\mathtt{a}_1||\mathtt{a}| + \#\mathtt{a}) \\
&+ |\mathbf{fe}(\mathtt{a}_1)||\mathtt{b}| + \#\mathbf{fe}(\mathtt{a}_1) + \#\mathtt{b} + |\mathtt{b}|(|\mathtt{a}|\mathbf{natpart}(\mathtt{b})) \\
&+ \#(\mathbf{fe}(\mathtt{a}_1))|\mathtt{b}| + \#\mathtt{b} + \#\mathtt{a} \cdot \mathbf{natpart}(\mathtt{b}))
\end{aligned}$$

By gathering like terms and noting that $\#(\mathbf{fe}(\mathtt{a}_1))|\mathtt{b}| > |\mathbf{fe}(\mathtt{a}_1)||\mathtt{b}|$, we obtain a time bound of $O(\mathbf{natpart}(\mathtt{b})[|\mathtt{a}||\mathtt{b}|+|\mathtt{a}_1||\mathtt{a}|+\#\mathtt{a}]+\#\mathbf{fe}(\mathtt{a}_1)|\mathtt{b}|+\#\mathtt{b})$. □

Theorem 21 $\mathbf{exp_o}(\mathtt{a}, \mathtt{b})$ runs in time $\mathrm{O}(\mathbf{natpart}(\mathtt{b})[|\mathtt{a}||\mathtt{b}| + |\mathtt{a}_1||\mathtt{a}| + \#\mathtt{a}] + \#\mathbf{fe}(\mathtt{a}_1)|\mathtt{b}| + \#\mathtt{b})$.

Proof This follows directly from Theorems 13, 17, and 20. □

An obvious question is whether we can improve the exponential running time of $\mathbf{exp_o}$. Given our representation of the ordinals, the answer is no, as the following class of examples shows. Fix a to be [1, 1, 1], which corresponds to the ordinal $\omega + 1$ and let $\mathtt{b}_k$ be [[1, 1, 1], 1, k], which corresponds to the ordinal $\omega^\omega + k$. For this infinite class of examples, $\#\mathbf{exp_o}(\mathtt{a}, \mathtt{b}_k)$ is exactly equal to the complexity of $\mathbf{exp_o}$. That is, simply constructing the ordinal corresponding to $\mathbf{exp_o}(\mathtt{a}, \mathtt{b}_k)$ takes as long as this function takes to run in the worst case. Therefore, this algorithm is as efficient as can be expected.

8. Implementation

In this section, we describe the implementation of the ordinal arithmetic algorithms in the ACL2 theorem proving system. There are several facets to this work. In Section 8.1, we give a brief overview of ACL2 and describe the ACL2 implementation of the ordinal arithmetic algorithms. In Section 8.2, we discuss the mechanical verification of the implementations. The proof scripts are part of the ACL2 distribution (as of version 2.8) and are described in greater detail elsewhere [37]. Our main goal in formalizing ordinal arithmetic on ordinal notations was to enable ACL2 to automatically prove complex theorems involving the ordinals. In Section 8.3, we give an overview of the library we engineered into a tool for reasoning about termination and the ordinals in ACL2, and we give an example of their use. Our treatment of the ordinals provides many advantages over the treatment in ACL2 v2.7; thus, we decided to update the representations, definitions, theorems, and documentation of the ordinals [38]. These modification are present in ACL2 version 2.8 and include a more efficient ordinal representation and a greatly extended ability to reason about ordinals. This aspect of our work is described in Section 8.4.

8.1. Implementation in ACL2

"ACL2" stands for "A Computational Logic for Applicative Common Lisp." It is the name of a programming language, a first-order mathematical logic based on recursive functions, and a mechanical theorem prover for that logic [26, 27, 25].

As a programming language, ACL2 can best be thought of as an applicative—side-effect free or purely functional—subset of Lisp. ACL2 is executable: terms composed entirely of defined functions and constants can be reduced to constants by Lisp calculation. This is very important to many applications. For example, ACL2 models of commercial floating-point designs have been executed on millions of test cases to "validate" the models against industrial design simulation tools, before subjecting the ACL2 models to proof [56]. ACL2 models of microprocessors have been executed at 90% of the speed of comparable C simulation models [23].

As a mathematical logic, ACL2 may be thought of as first-order predicate calculus with equality, recursive function definitions, and mathematical induction. The primitives of applicative Common Lisp are axiomatized, as are the basic data types, including natural numbers, integers, rationals, complex rationals, ordered pairs, symbols, characters, and strings. ACL2 includes a representation of the ordinals up to ε_0 and the principle of mathematical induction, in ACL2, is stated as a rule of inference that allows induction up to ε_0. A principle of definition is also provided, by which the user can extend the axioms by the addition of equations defining new function symbols. To admit a new recursive definition, the principle requires the identification of an ordinal measure function and a proof that the arguments to every recursive call decrease according to this measure. Only terminating recursive definitions can be so admitted under the definitional principle. (However, "partial functions" can be axiomatized; see [33, 34].)

As a theorem prover, ACL2 is an industrial-strength version of the Boyer-Moore theorem prover [8]. Of special note is its "industrial-strength," *e.g.*, it has been used to prove some of the largest and most complicated theorems ever proved about commercially designed digital artifacts [41, 54, 53, 55, 56, 9, 24]. The theorem prover is an integrated system of ad hoc proof techniques that include simplification, generalization, induction, and many other techniques. Simplification is the main technique and includes: (1) the use of evaluation (*i.e.*, the explicit computation of constants when, in the course of symbolic manipulation, certain variable-free expressions, like `(expt 2 32)`, arise), (2) conditional rewrite rules (derived from previously proved lemmas), (3) definitions (including recursive definitions), (4) propositional calculus (implemented both by the normalization of if-then-else expressions and the use of BDDs), (5) a linear arithmetic decision procedure for the rationals, (6) user-defined equivalence and congruence relations, (7) user-defined and mechanically verified simplifiers (meta-reasoning), (8) a user-extensible type system, (9) forward chaining, (10) an interactive loop for entering proof commands, and (11) various means to control and monitor these features including heuristics, interactive features, and user-supplied functional programs. See [26, 25] or the documentation, source code and examples at the URL [27] for details.

An example of ACL2 definitions appears in Figure 11, where we give our definition of ordinal multiplication (`ob*`). Functions are defined with the `defun` construct, *e.g.*, consider the definition of `count1`. The first argument, in this case, `count1`, is the name of the function. The second argument is a list of its parameters. The last argument is the body of the function. Once ACL2 admits the function, an axiom is added stating that `(count1 x y)` is equal to the body of `count1`. The body of `count1` refers to several functions not defined in Figure 11, *e.g.*, `finp` is the predicate that recognizes if an ordinal is finite and corresponds to **finp** from the first part of the paper. The functions `fe`, `fco` (not used in `count1`), and `rst` correspond to the **fe**, **fco**, and **rst** functions, respectively. The `op` function in the declaration corresponds to **op**.

The `declare` statement in between the parameters and body of `count1` is an optional argument that does not affect the meaning of the `defun`, but allows the user to inform ACL2 about various pragmatic issues. In this case, we declare the expected type of the inputs, using a `guard` declaration. Guards are used by the compiler to generate efficient code. A guard can be any predicate over the function parameters; in the case of `count1`, we require that both arguments are ordinals. When verifying the guards of a function, ACL2 must demonstrate that when the guards conditions for the function hold, the guard conditions for all functions called within the body also hold. For example, we must prove that whenever `count2` calls `count1`, `(lastn n x)` and `y` are both ordinals. If a top level expressions satisfies its guards, then we are guaranteed that no guard violations can occur during execution, and ACL2 is free to execute the efficient version of the definition.

Another notable feature of ACL2 is `defexec`, which the definition of `ob*` takes advantage of. Using `defexec` the user can specify two definitions of a function: the `:logic` definition and the `:exec` definition. ACL2 is then required to prove that these two definitions are equivalent when the guards hold, which allows ACL2 to use the simpler `:logic` definition when performing symbolic manipulation, but to use the more efficient `:exec` definition during execution. In our case, we use the

```
(defun count1 (x y)
  (declare (xargs :guard (and (op x) (op y))))
  (cond ((finp x) 0)
        ((o< (fe y) (fe x))
         (+ 1 (count1 (rst x) y)))
        (t 0)))

(defun count2 (x y n)
  (declare (xargs :guard (and (op x) (op y) (natp n))))
  (+ n (count1 (lastn n x) y)))

(defun padd (x y n)
  (declare (xargs :guard (and (op x) (natp n)
                              (op y) (<= n (count1 x y)))))
  (if (or (finp x) (zp n))
      (o+ x y)
    (make-ord (fe x) (fco x) (padd (rst x) y (1- n)))))

(defun pmult (x y n)
  (declare (xargs :guard (and (op x) (natp n)
                              (op y) (<= n (count1 (fe x)
                                                   (fe y))))))
  (let* ((fe-x (fe x)) (fco-x (fco x))
         (fe-y (fe y)) (fco-y (fco y))
         (m (count2 fe-x fe-y n)))
    (cond ((or (equal x 0) (equal y 0)) 0)
          ((and (finp x) (finp y)) (* x y))
          ((finp y) (make-ord fe-x (* fco-x fco-y) (rst x)))
          (t (make-ord (padd fe-x fe-y m)
                       fco-y
                       (pmult x (rst y) m))))))

(defexec ob* (x y)
  (declare (xargs :guard (and (op x) (op y))))
  (mbe :logic (let ((fe-x (fe x)) (fco-x (fco x))
                    (fe-y (fe y)) (fco-y (fco y)))
                (cond ((or (equal x 0) (equal y 0)) 0)
                      ((and (finp x) (finp y)) (* x y))
                      ((finp y) (make-ord fe-x
                                          (* fco-x fco-y)
                                          (rst x)))
                      (t (make-ord (o+ fe-x fe-y)
                                   fco-y
                                   (ob* x (rst y))))))
        :exec (pmult x y 0)))
```

Figure 11.: ACL2 definitions of ordinal addition and multiplication.

inefficient version of multiplication (given in Section 6) for the `:logic` definition and the efficient version for the `:exec` definition. We also use `defexec` to define ordinal exponentiation.

8.2. Mechanical Verification

The mechanical verification of the ordinal arithmetic algorithm implementations involved proving two different classes of theorems, beyond the guard conjectures and termination proofs mentioned in the previous section. The first class deals with the algebraic properties of the operations. We proved that each function has the same algebraic properties as its corresponding set-theoretic operation. For example, we proved that `ob+`, `ob-`, `ob*`, and `ob^` have all the properties we listed for ordinal addition, subtraction, multiplication, and exponentiation in Section 2.

The second class of theorems are about the notation and involve helper functions, such as `make-ord`, `fe`, `fco`, and `rst`, and how they interact with the algebraic functions. An example of this is the following theorem.

```
(defthm o+-fe-1
  (implies (o< (fe a)
               (fe b))
           (equal (fe (o+ a b))
                  (fe b))))
```

Recall that all of these theorems are part of the ACL2 distribution. Also, note that these theorems deal with ordinal notations and the implementations in ACL2 of the ordinal arithmetic algorithms. That is, we do not mechanically establish any connection with the set-theoretic definitions on which our algorithms are based, as our goal was not to formalize set-theory in ACL2. Instead, we focused on using our results about arithmetic on ordinal notations to extend ACL2's ability to reason about termination. However, many of the paper and pencil proofs in the first part of this paper turned out to be quite useful, as they provided the key insights required to complete the ACL2 proofs.

8.3. Library for Automatic Verification

Enabling ACL2 to effectively and automatically reason about the ordinals and termination requires more than proving the correctness of the implementations, the topic of the previous section. It requires carefully constructing a library that makes effective and efficient use of the various types of mechanisms that ACL2 provides to control the way in which theorems are used. A complete description of the issues is beyond the scope of this paper, but see [38]. Instead, we discuss a few important considerations that went into engineering a useful library.

The first consideration involves a concept in ACL2 known as "rule classes." When ACL2 proves a theorem, it gets entered into a database so that it can be used in subsequent proof attempts. By default, theorems are entered as rewrite rules. Rewrite rules can be conditional and are triggered when a goal contains an expression matching the left hand size of the rule's consequent. When this happens, ACL2 attempts to establish the antecedents of the rule via backchaining, and if successful, it rewrites the expression, using the right hand side of the rewrite rule. For example, consider the following rule.

```
(defthm |~(a=0) /\ b>1  <=>  a < ab|
  (implies (and (op a)
                (op b))
           (equal (o< a (o* a b))
                  (and (not (equal a 0))
                       (not (equal b 0))
                       (not (equal b 1))))))
```

After proving this theorem, ACL2 enters it into the database of rules as a rewrite rule. Subsequently, when ACL2 sees an expression of the form `(o< e1 (o* e1 e2))`, where `e1` and `e2` are arbitrary ACL2 expressions, it will try to determine if `e1` and `e2` are `op`s. If so, ACL2 will rewrite `(o< e1 (o* e1 e2))` to `(and (not (equal e1 0)) (not (equal e2 0)) (not (equal e2 1)))`. Notice that although `(o< e1 (o* e1 e2))` is smaller in size than `(and (not (equal e1 0)) (not (equal e2 0)) (not (equal e2 1)))`, it contains `o<` and `o*`, which are relatively complex functions. It is important to orient rewrite rules so that they reduce expressions containing complex functions into expressions containing simpler functions. It is also important to take into account how much effort will be expended trying to discharge the hypotheses, and rules should be written in a way that forces expressions into "canonical" forms.

While rewrite rules are the most widely used rule class, there are other types of rules, *e.g.*, forward chaining rules are triggered when all of the antecedents are known to be true. When this happens, the consequent is added to the "context," the collection of known facts. When dealing with large libraries such as ours, the interaction between rules of different classes can become quite complicated and the decisions made about which rules to put in which classes therefore has a significant impact on the effectiveness and efficiency of a library of theorems. There are general guidelines as to which rule classes to use [27, 26], but engineering an efficient library requires a good dose of experimentation and profiling.

It is also important to distinguish between the theorems that one wants to export versus the intermediate lemmas that are used to prove such theorems. For example, to prove the left distributive property of multiplication over addition, we had to prove several lemmas which correspond to special cases of the theorem. The distributive property theorem should be exported (made visible when the library is loaded into ACL2), but the supporting lemmas should not. This is accomplished with ACL2's `local` form. Sometimes a lemma can also cause problems within a library and in this case, one can use the `encapsulate` form, which provides a way of hiding `local` theorems from the rest of the library (and much more).

Another concern is deciding when to use macros and when to use functions. In ACL2 macros are simply syntactic sugar and are expanded away before theorem proving begins. Thus, ACL2 does not reason about macros. In designing our library, we used macros in two ways. The first was to simplify the class of theorems needed to reason about the ordinals. For example, we made `o<=` a macro such that `(o<= a b)` expands to `(not (o< b a))`. This greatly simplified our library, because we did not need to develop rewrite rules to reason about expressions involving `o<=`. The problem with this approach is that the output generated by ACL2 is with respect to `o<`, so we altered ACL2 to print `(o<= a b)` instead of `(not (o< b a))`. This leads to improved readability.

The second way we used macros was to create polyadic versions of our binary functions. For example, `o*` is a macro and `(o* x y z)` macro expands to `(ob* x (ob* y z))`. We also include similar macros for addition and exponentiation. To improve readability, ACL2 can be instructed to print `ob*` in terms of `o*` with the command `(add-binop o* ob*)`. Thus, users are under the illusion that they are reasoning about polyadic functions, while all reasoning is really with respect to the binary functions. In summary, macros provide not only syntactic extensions, but also provide limited support for maintaining the illusion that users can reason about these extensions, thereby simplifying the interface between theorem prover and user.

The final consideration we address here is the structure of the library. The library is divided into files of ACL2 theorems and definitions, called "books." Dividing the theorems properly between the books adds logical coherence and modularity to the library. This maximizes efficiency through code reuse and makes the books easier to understand for users. The structure of this library is illustrated in Figure 12, where the rectangles represent books, and the arrows represent the dependencies between books. For example, the arrow from `ordinal-isomorphism` to `e0-ordinal` indicates that the results of `e0-ordinal` rely on the results of `ordinal-isomorphism`. A short description of the contents of the books can be found in Table I. The total size of the books is

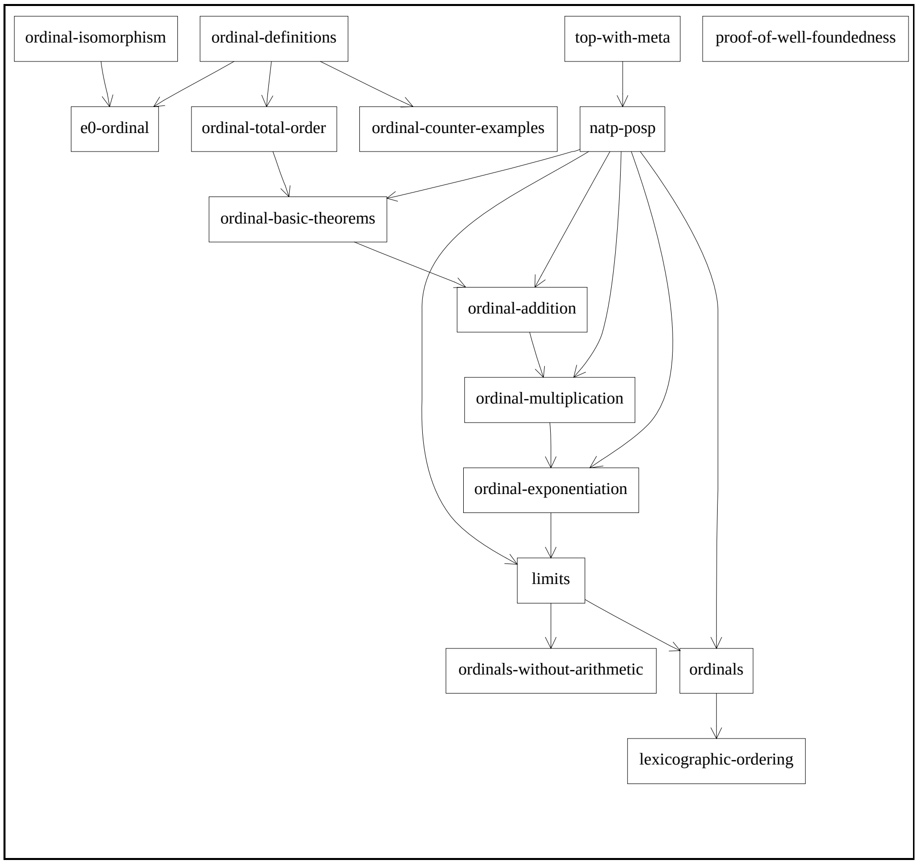

Figure 12.: The Ordinal Library.

181K and they consist of about 5,455 lines of definitions, theorems, and comments.

To use the ordinal library, the user loads either the `ordinals` or the `ordinals-without-arithmetic` book, depending on whether she wishes to include results about integer arithmetic. The integer arithmetic book, `top-with-meta`, was essential for the creation of the library, but can sometimes interfere with other books.

The library has already been used extensively to perform termination proofs in ACL2. When using it, simple termination proofs go through without any input from the user, and even relatively complicated proofs usually require only the specification of a measure function. In addition, we have used the library to define new well-founded relations for termination proofs. This is accomplished in ACL2 with the `set-well-founded-relation` command, which requires proving that a set and a relation over that set can be embedded in the ordinals in an order-preserving way. In the `lexicographic-ordering` book we

Book	Description
`top-with-meta`	A link to the arithmetic books
`natp-posp`	Theorems about natp and posp
`ordinal-definitions`	The function definitions
`ordinal-total-order`	Theorems about the behavior of `o<`
`ordinal-basic-thms`	Basic theorems about the helper functions
`ordinal-addition`	Theorems about `o+` and `o-`
`ordinal-multiplication`	Theorems about `o*`
`ordinal-exponentiation`	Theorems about `o^`
`ordinal-isomorphism`	Proof of isomorphism of our ordinals & ACL2's
`e0-ordinal`	Exports major results of `ordinal-isomorphism`
`limits`	Theorems about limit ordinals
`ordinal-counter-examples`	Counter-examples, *e.g.*, commutativity
`ordinals-without-arithmetic`	Exports ordinal thms without integer arithmetic
`ordinals`	Exports ordinal thms with integer arithmetic
`proof-of-well-foundedness`	Part of proof of well-foundedness of our ordinals
`lexicographic-ordering`	Proves well-foundedness of a lexicographic order

Table I.: The Ordinal Library.

illustrate how this is done by proving that the lexicographic ordering over natural numbers is well-founded. The lexicographic order over the natural numbers is much simpler to explain to students and is what we now use when teaching ACL2.

Our library has also been used by Sustik to give a constructive proof of Dickson's Lemma in ACL2 [61]. Dickson's Lemma states that for any infinite sequence of monomials (over some fixed number of variables), $m_0, m_1, m_2, \ldots$, there exists $i, j \in \mathrm{N}$ such that $i < j$ and m_i divides m_j. This is a key lemma in the proof of the termination of Buchberger's algorithm, which finds a Gröbner basis of a polynomial ideal. Sustik's argument involves mapping initial segments of the monomial sequence into the ordinals such that if Dickson's lemma fails, the ordinal sequence will be decreasing. Thus, the existence of an infinite sequence of monomials such that no monomial divides a later monomial implies the existence of an infinite decreasing sequence of ordinals, which is not possible due to the well-foundedness of the ordinals. Sustik's proof relies heavily on ordinal addition and exponentiation and is an example of the kind of termination proof that would be quite difficult to fully automate. Nonetheless, his proof requires no theorems about ordinals beyond those provided by our library.

8.4. Modifying ACL2

After creating the ordinal arithmetic library, we decided to modify ACL2, replacing its ordinal representation by our own, so that it could take full advantage of our work. The modifications included updating the documentation and modifying the ACL2 sources and consisted of about 1,750 lines of code and documentation. We submitted the changes to Kaufmann and Moore, the authors of ACL2, and they have incorporated the changes into the ACL2 version 2.8 [27].

It is worth noting that our changes do not affect the soundness of the ACL2 logic. In the `ordinal-isomorphism` book of our library, we exhibit a bijection between our ordinal representation and the previous ACL2 representation (see Corollary 1). This proof was carried out in ACL2 version 2.7, thus guaranteeing that soundness is not affected.

We now discuss some of the issues we confronted in modifying ACL2. First, the ordinals are needed in ACL2's *ground-zero theory*, the initial theory encountered when starting an ACL2 session. Proving theorems, defining functions, including books, etc. all result in extensions to the ground-zero theory, and we wanted to keep it as clean and simple as possible. Therefore, we did not want to add our entire library of definitions and theorems to the ground-zero theory. Instead, we included only the basic constructors and destructors (`make-ord`, `fe`, `fco`, `rst`), the functions necessary for `op` and `o<` (`natp`, `posp`, `infp`, `finp`, `o<`, `op`), and a few macros based on `o<` (`o>`, `o<=`, `o>=`). The arithmetic functions and theorems remain in the library.

After replacing the old ordinals with our new representation, we had to deal with legacy issues, including backward compatibility for the books included with ACL2, as many of these books referenced the old ordinal representation. The key to fixing these references was the theorems proved in the `ordinal-isomorphism` and `e0-ordinal` books. The main result in the books is a proof that there exists a bijection between the new and old ordinal representations. This result allowed us to switch the well-founded relation used by ACL2 to the version 2.7 relation (for the admission of the troublesome books only). That fixed most of the problems, however, some books used the old ordinals to prove more than just termination. Again, by using the bijection proof, we were able to transfer results about the old ordinals to the new ordinals, which resolved the remaining problems.

9. Conclusion

We presented efficient algorithms for ordinal addition, subtraction, multiplication, and exponentiation on succinct ordinal representations, proved their correctness, and analyzed their complexity. We implemented the algorithms in the ACL2 system, mechanically verified the correctness of the implementations, and developed a library of theorems that can be used to significantly automate reasoning involving the ordinals. We modified ACL2 so that it directly supports our representation of the ordinals and our libraries; the modifications are part of ACL2 version 2.8. While the theory of the ordinal numbers has been studied by various research communities for over 100 years, we believe that we are the first to give algorithms for ordinal arithmetic on ordinal notations.

Our work can be extended by replacing ε_0 with larger countable ordinals. A simple counting argument shows that no ordinal notation can represent all countable ordinals, but there are well known notations that can represent ordinals up to Γ_0 (which is needed to show termination of some term rewrite systems [15, 20]) and further into the Veblen hierarchies [65] and further still [40, 58, 59]. Another possible extension is to define additional operations on ordinals, *e.g.*, division, taking logs, etc. Finally, a promising direction for future work is to use our library and ACL2 as a proof checker for termination tools that guess measure functions in some heuristic fashion and to apply these tools to systems written in common imperative languages such as C and Java.

Acknowledgments
We would like to thank Matt Kaufmann and J Strother Moore for integrating our modifications into ACL2 version 2.8. We would also like to thank Mátyás Sustik for using our library and making many suggestions on how we can improve it.

References

1. K. R. Apt and E.-R. Olderog. *Verification of Sequential and Concurrent Programs.* Springer-Verlag, New York, 1991.
2. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
3. G. Bancerek. The reflection theorem. *Journal of Formalized Mathematics*, 2, 1990. See URL `http://megrez.mizar.org/mirror/JFM/Vol2/zf_refle.html`.
4. J. G. Belinfante. Computer proofs in Gödel's class theory with equational definitions for composite and cross. *Journal of Automated Reasoning*, 22(3):311–339, 1999.
5. J. G. F. Belinfante. On computer-assisted proofs in ordinal number theory. *Journal of Automated Reasoning*, 22(3):341–378, 1999.
6. J. G. F. Belinfante. Reasoning about iteration in Gödel's class theory. In F. Baader, editor, *Automated Deduction - CADE-19, Proceedings of the 19th International Conference on Automated Deduction*, volume 2741 of *LNAI*, pages 228–242. Springer-Verlag, 2003.
7. Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development, Coq'Art: the calculus of inductive constructions.* Texts in Theoretical Computer Science. Springer-Verlag, May 2004.
8. R. S. Boyer and J. S. Moore. *A Computational Logic Handbook.* Academic Press, second edition, 1997.

9. B. Brock, M. Kaufmann, and J. S. Moore. ACL2 theorems about commercial microprocessors. In M. Srivas and A. Camilleri, editors, *Formal Methods in Computer-Aided Design (FMCAD'96)*, pages 275–293. Springer-Verlag, 1996.
10. G. Cantor. Beiträge zur Begründung der transfiniten Mengenlehre. *Mathematische Annalen*, xlvi:481–512, 1895.
11. G. Cantor. Beiträge zur Bgründung der transfiniten Mengenlehre. *Mathematische Annalen*, xlix:207–246, 1897.
12. G. Cantor. *Contributions to the Founding of the Theory of Transfinite Numbers.* Dover Publications, Inc., 1952. Translated by Philip E. B. Jourdain.
13. A. Church and S. C. Kleene. Formal definitions in the theory of ordinal numbers. *Fundamenta mathematicae*, 28:11–21, 1937.
14. L. A. Dennis and A. Smaill. Ordinal arithmetic: A case study for rippling in a higher order domain. In R. Boulton and P. Jackson, editors, *Theorem Proving in Higher Order Logics: 14th International Conference, TPHOLs 2001*, volume 2152 of *LNCS*, pages 185–200. Springer-Verlag, 2001.
15. N. Dershowitz and M. Okada. Proof-theoritic techniques for term rewriting theory. In *3rd IEEE Symp. on Logic in Computer Science*, pages 104–111, 1988.
16. N. Dershowitz and E. M. Reingold. Ordinal arithmetic with list structures. In *Logical Foundations of Computer Science*, pages 117–126, 1992.
17. K. Devlin. *The Joy of Sets: Fundamentals of Contemporary Set Theory.* Springer-Verlag, second edition, 1992.
18. J. Doner. Definability in the extended arithmetic of ordinal numbers. *Dissertationes Mathematicae*, 96, 1972.
19. J. Doner and A. Tarski. An extended arithmetic of ordinal numbers. *Fundamenta Mathematicae*, 65:95–127, 1969.
20. J. H. Gallier. What's so special about Kruskal's theorem and the ordinal Γ_0? A survey of some results in proof theory. *Annals of Pure and Applied Logic*, pages 199–260, 1991.
21. G. Gentzen. Die Widerspruchsfreiheit der reinen Zahlentheorie. *Mathematische Annalen*, 112:493–565, 1936. English translation in M. E. Szabo (ed.), The Collected Works of Gerhard Gentzen, pp. 132-213, North Holland, Amsterdam, 1969.
22. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, 1993.
23. D. Greve, M. Wilding, and D. Hardin. High-speed, analyzable simulators. In Kaufmann et al. [25], pages 113–135.
24. D. A. Greve. Symbolic simulation of the JEM1 microprocessor. In *Formal Methods in Computer-Aided Design – FMCAD*, LNCS. Springer-Verlag, 1998.
25. M. Kaufmann, P. Manolios, and J. S. Moore, editors. *Computer-Aided Reasoning: ACL2 Case Studies.* Kluwer Academic Publishers, June 2000.
26. M. Kaufmann, P. Manolios, and J. S. Moore. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, July 2000.
27. M. Kaufmann and J. S. Moore. ACL2 homepage. See URL `http://www.cs.-utexas.edu/users/moore/acl2`.
28. M. Kaufmann and J. S. Moore, editors. *Proceedings of the ACL2 Workshop 2000.* The University of Texas at Austin, Technical Report TR-00-29, November 2000.

29. M. Kaufmann and J. S. Moore, editors. *Fourth International Workshop on the ACL2 Theorem Prover and Its Applications (ACL2-2003)*, July 2003. See URL `http://www.cs.utexas.edu/users/moore/acl2/workshop-2003/`.
30. K. Kunen. *Set Theory - An Introduction to Independence Proofs*, volume 102 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1980.
31. P. Manolios. Correctness of pipelined machines. In W. A. Hunt, Jr. and S. D. Johnson, editors, *Formal Methods in Computer-Aided Design–FMCAD 2000*, volume 1954 of *LNCS*, pages 161–178. Springer-Verlag, 2000.
32. P. Manolios. *Mechanical Verification of Reactive Systems*. PhD thesis, University of Texas at Austin, August 2001. See URL `http://www.cc.gatech.edu/~manolios/publications.html`.
33. P. Manolios and J. S. Moore. Partial functions in ACL2. In Kaufmann and Moore [28].
34. P. Manolios and J. S. Moore. Partial functions in ACL2. *Journal of Automated Reasoning*, 31(2):107–127, 2003.
35. P. Manolios, K. Namjoshi, and R. Sumners. Linking theorem proving and model-checking with well-founded bisimulation. In N. Halbwachs and D. Peled, editors, *Computer-Aided Verification–CAV '99*, volume 1633 of *LNCS*, pages 369–379. Springer-Verlag, 1999.
36. P. Manolios and D. Vroon. Algorithms for ordinal arithmetic. In F. Baader, editor, *19th International Conference on Automated Deduction – CADE-19*, volume 2741 of *LNAI*, pages 243–257. Springer–Verlag, July/August 2003.
37. P. Manolios and D. Vroon. Ordinal arithmetic in ACL2. In Kaufmann and Moore [29]. See URL `http://www.cs.utexas.edu/users/moore/acl2/workshop-2003/`.
38. P. Manolios and D. Vroon. Integrating reasoning about ordinal arithmetic into ACL2. In *Formal Methods in Computer-Aided Design: 5th International Conference – FMCAD-2004*, LNCS. Springer–Verlag, November 2004.
39. I. Medina-Bulo, F. Palomo-Lozano, and J. A. Alonso-Jimenez. Implementation in ACL2 of well-founded polynomial orderings. In M. Kaufmann and J. S. Moore, editors, *Proceedings of the ACL2 Workshop 2002*. 2002.
40. L. W. Miller. Normal functions and constructive ordinal notations. *Journal of Symbolic Logic*, 41:439–459, June 1976.
41. J. S. Moore, T. Lynch, and M. Kaufmann. A mechanically checked proof of the AMD5_K86 floating-point division program. *IEEE Trans. Comp.*, 47(9):913–926, September 1998.
42. F. Morris and C. Jones. An early program proof by Alan Turing. *IEEE Annals of the History of Computing*, 6(2):139–143, April–June 1984.
43. S. Owre, J. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, pages 748–752. Lecture Notes in Artificial Intelligence, Vol 607, Springer-Verlag, June 1992.
44. L. C. Paulson. Set theory for verification: I. From foundations to functions. *Journal of Automated Reasoning*, 11(3):353–389, 1993.
45. L. C. Paulson. *Isabelle: A Generic Theorem Prover*. Springer-Verlag LNCS 828, 1994.
46. L. C. Paulson. Set theory for verification: II. Induction and recursion. *Journal of Automated Reasoning*, 15(2):167–215, 1995.

47. L. C. Paulson. The reflection theorem: a study in meta-theoretic reasoning. In A. Voronkov, editor, *18th International Conf. on Automated Deduction: CADE-18*, number 2392 in LNAI, pages 377–391. Springer-Verlag, 2002.
48. L. C. Paulson. The relative consistency of the axiom of choice mechanized using isabelle. *LMS Journal of Computation and Mathematics*, 6:198–248, 2003.
49. L. C. Paulson and K. Grabczewski. Mechanizing set theory: cardinal arithmetic and the axiom of choice. *Journal of Automated Reasoning*, 17:291–323, 1996.
50. H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* MIT Press, 1st paperback edition, 1987.
51. P. Rudnicki. An overview of the MIZAR project. In *1992 Workshop on Types for Proofs and Programs*, 1992.
52. J.-L. Ruiz-Reina, J.-A. Alonso, M.-J. Hidalgo, and F.-J. Martin. Multiset relations: A tool for proving termination. In Kaufmann and Moore [28].
53. D. M. Russinoff. A mechanically checked proof of correctness of the AMD5_K86 floating-point square root microcode. *Formal Methods in System Design Special Issue on Arithmetic Circuits*, 1997.
54. D. M. Russinoff. A mechanically checked proof of IEEE compliance of a register-transfer-level specification of the AMD-K7 floating-point multiplication, division, and square root instructions. *London Mathematical Society Journal of Computation and Mathematics*, 1:148–200, December 1998.
55. D. M. Russinoff. A mechanically checked proof of correctness of the AMD-K5 floating-point square root microcode. *Formal Methods in System Design*, 14:75–125, 1999.
56. D. M. Russinoff and A. Flatau. RTL verification: A floating-point multiplier. In Kaufmann et al. [25], pages 201–231.
57. K. Schütte. *Proof Theory.* Springer-Verlag, 1977. Translation from the German by J. N. Crossley. The book is a completely rewritten version of *Beweistheorie*, Springer-Verlag, 1960.
58. A. Setzer. Ordinal systems. In B. Cooper and J. Truss, editors, *Sets and Proofs*, pages 301–331. Cambridge University Press, 1999.
59. A. Setzer. Ordinal systems part 2: One inaccessible. In *Logic Colloquium '98*, volume 13 of *ASL Lecture Notes in Logic*, pages 426–448, 2000.
60. R. Sumners. An incremental stuttering refinement proof of a concurrent program in ACL2. In Kaufmann and Moore [28].
61. M. Sustik. Proof of Dixon's lemma using the ACL2 theorem prover via an explicit ordinal mapping. In Kaufmann and Moore [29]. See URL `http://-www.cs.utexas.edu/users/moore/acl2/workshop-2003/`.
62. A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory.* Cambridge University Press, second edition, 2000.
63. A. M. Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, 45(2):161–228, 1939. See URL `http://www.-turingarchive.org/`.
64. A. M. Turing. Checking a large routine. In *Report of a Conference on High Speed Automatic Calculating Machines*, pages 67–69. Universtity Mathematical Laboratory, Cambridge, June 1949.
65. O. Veblen. Continuous increasing functions of finite and transfinite ordinals. *Transactions of the American Mathematical Society*, 9:280–292, 1908.