Ordinal Arithmetic in ACL2

Panagiotis Manolios and Daron Vroon

Georgia Institute of Technology, College of Computing, CERCS Lab
801 Atlantic Drive, Atlanta, Georgia, 30332, USA,
{manolios,vroon}@cc.gatech.edu
http://www.cc.gatech.edu/∼{manolios,vroon}

Abstract. Ordinals form the basis for termination proofs in ACL2. Currently, ACL2 uses a rather inefficient representation for the ordinals up to ϵ_0 and provides limited support for reasoning about them. We present algorithms for ordinal arithmetic on an exponentially more compact representation than the one used by ACL2. The algorithms have been implemented and numerous properties of the arithmetic operators have been mechanically verified, thereby greatly extending ACL2's ability to reason about the ordinals. We describe how to use the libraries containing these results, which are currently distributed with ACL2 version 2.7.

1 Introduction

Termination proofs play a crucial role in the mechanical verification of systems. For example, ACL2's definitional principle requires that functions be shown to terminate before they are admitted. Termination proofs are even useful in the context of *reactive systems*, non-terminating systems that are engaged in on-going interaction with an environment, as they are used to establish liveness properties by showing that the desired behavior is not postponed forever. Proving the termination of a system is accomplished by showing a relation corresponding to a decreasing "measure" of the system is well-founded. Since every well-founded relation can be extended to a total ordering that is isomorphic to an ordinal, it makes sense that systems such as ACL2 use ordinals as the basis for termination proofs.

The theory of ordinals has been studied for over 100 years, since it was introduced by Cantor as the core of his set theory [1, 2] (see also the English translation [3]). Ordinals have subsequently played an important role in logic, *e.g.*, they are routinely used to prove the consistency of logical systems. This practice was introduced by Gentzen, when he proved the consistency of Peano arithmetic using induction up to ϵ_0 [9]. In order to obtain constructive proofs, constructive ordinal notations are used [16, 20]. The general theory of these notations was initiated by Church and Kleene [4] and is reviewed in Chapter 11 of Roger's book on computability [15].

Although ordinal notations have been studied extensively by various communities for over a century, we have been unable to find a comprehensive treatment of arithmetic for ordinal notations. We define the *ordinal arithmetic problem* for a notational system denoting ordinals up to some ordinal δ as follows: given α and β, expressions in the system denoting ordinals less than δ, 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 arithmetic operations in the given notational system. While partial solutions to this problem are presented in various texts and papers [16, 6, 8, 14, 17, 20], (for example, definitions for $<$ appear in many of the above sources), we have not found any similar statement of this problem nor any comprehensive solution in previous work.

In a companion paper, we present efficient algorithms for ordinal addition, subtraction, multiplication, and exponentiation on succinct ordinal representations, prove their correctness, and analyze their complexity [13]. In this paper, we discuss the ACL2 mechanization, as we use ACL2 to define our ordinal representation and arithmetic algorithms. Our representation is exponentially more compact than the current representation in ACL2 and we prove that the ordinals in our representation are isomorphic to the ACL2 ordinals. This makes it possible for users to use our representation by issuing a single command. In addition, we discuss an ACL2 library of theorems about ordinal arithmetic. This library significantly increases the extent to which ACL2 can automatically reason about ordinals and also makes it easier to reason about ordinals from a human perspective. Previously, users were forced to define functions that explicitly constructed ordinals and had to reason about the representations. With our library, users can for the most part use the algebraic properties of the ordinals while ignoring representational issues. The library is now part of version 2.7 of the ACL2 distribution.

Our paper is organized as follows. In Section 2, we give a brief overview of the theory of ordinals and ordinal arithmetic, we compare our representation of the ordinals with the current system employed by ACL2, and we give an overview of the properties of the ordinal arithmetic operators. In Section 3 we define our algorithms for the arithmetic operators and give informal proofs of correctness. In Section 4, we give a brief overview of how to use the books in ACL2. In Section 5, we consider complexity issues. Section 6 contains our conclusions and outlines future work.

2 Ordinals

2.1 Set-Theoretic Ordinals

We briefly review the theory of ordinal numbers [7, 12, 16]. A relation, $\prec$ is said to be *well-founded* if every decreasing sequence is finite. A *well-ordering* is a total, well-founded relation.

A *woset* is a pair $\langle X, \prec\rangle$, such that $\prec$ is a well-ordered relation over X. Given a woset $\langle X, \prec\rangle$, and an element $a \in X$, we define X_a to be $\{x \in X | x \prec a\}$. An *ordinal* is a woset, $\langle X, \prec\rangle$, such that for all $a \in X$, $a = X_a$. Note that if $\langle X, \prec\rangle$ is an ordinal and $a \in X$, then a is an ordinal and $\prec$ is $\in$.

For the remainder of the paper, we will use lowercase Greek letters to denote ordinals, and $<$ and $\in$ to denote the well-ordering.

Given two wosets, $\langle X, \prec\rangle$ and $\langle X', \prec'\rangle$, we say a function $f : X \rightarrow X'$ is said to be *isomorphic* if it is a bijection and for all $x, y \in X$, $x \prec y \equiv f.x \prec' f.y$. It is a well-known result of set theory that every woset is isomorphic to a unique ordinal.

Since termination is established using well-founded relations, which can be extended to well-ordered relations, we see that the theory of ordinals is the most general setting for termination proofs. Given a woset, $\langle X, \prec \rangle$, we denote the unique ordinal to which it is isomorphic by $\mathit{Ord}(X, \prec)$.

Given an ordinal, α, its *successor*, denoted α', is the ordinal $\alpha \cup \{\alpha\}$. There is clearly a minimal ordinal, $\emptyset$, usually denoted by 0. The successor of 0 is $0' = \{0\}$, and is denoted by 1. It is the second smallest ordinal. The next smallest is $2 = 1' = \{0, 1\}$. By continuing in this manner, we obtain the natural numbers.

A *limit ordinal* is a non-zero ordinal that is not a successor. The smallest of these is the set of all natural numbers, denoted ω. We now provide definitions for arithmetic functions over the ordinals.

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

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. *$\alpha \cdot \beta = \mathit{Ord}(A, <_A)$, where $A = \bigcup_{\xi<\beta}(\{\xi\} \times \alpha)$ and $<_A$ is the lexicographic ordering on A.*

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

Using these ordinal operations, we can extend our hierarchy of ordinals:

$0, 1, 2, \ldots, \omega, \omega+1, \omega+2, \ldots, \omega \cdot 2, \omega \cdot 2+1, \ldots \omega^2, \ldots \omega^3, \ldots, \omega^\omega$, and so on. Eventually, we reach the ordinal $\epsilon_0 = \omega^{\omega^{\omega^{\cdots}}}$, which is the smallest ordinal, α such that $\alpha = \omega^\alpha$.

2.2 Representation

ACL2 currently represents ordinals up to (but not including) ϵ_0. The representation is based on a well-known theorem by Cantor:

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

This normal form is known as Cantor Normal Form (CNF). For example, $w \cdot 2 + 3 = \omega^1 + w^1 + \omega^0 + \omega^0 + \omega^0$. However, since the sum of all terms of the form w^0 is a natural number and since ϵ_0 is the smallest ordinal, α, such that $\alpha = \omega^\alpha$, we get the following corollary:

Corollary 1. *For every ordinal $\alpha \in \epsilon_0$, there are unique $n, p \in \omega$ and $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_n > 0$ such that $\alpha > \alpha_1$ and $\alpha = \omega^{\alpha_1} + \cdots + \omega^{\alpha_n} + p$.*

We can define a function, f to convert any ordinal $\alpha \in \epsilon_0$ to its ACL2 representation. Let $\omega^{\alpha_1} + \cdots + \omega^{\alpha_n} + p \in \epsilon_0$ be the CNF decomposition of α according to Corollary 1. Then,

$$f.\alpha = \begin{cases} \alpha, & \text{if } \alpha \in \omega \\ (f.\alpha_1 \quad f.\alpha_2 \quad \ldots \quad f.\alpha_n \quad . \quad p) & \text{otherwise} \end{cases}$$

For example, $f(5) = $ `5`, $f(\omega^4+\omega^4+\omega^2+\omega+3) =$ `(4 4 2 1 . 3)`, and $f(\omega^\omega+\omega^2+\omega^2)$ = `((1 . 0) 2 2 . 0)`. A further explanation of the ordinal representation used in ACL2 can be found in [10].

Our representation of the ordinals is similar to ACL2's, but uses the left distributive property of ordinal multiplication over addition to make it more compact. It is a well known result of the theory of ordinals that, for all α, β, and γ, we have $\alpha(\beta+\gamma) = \alpha\beta+\alpha\gamma$. Using this fact we can collect like terms. For example, $\omega^\alpha+\omega^\alpha+\omega^\alpha = \omega^\alpha \cdot 3$. Hence, we get a new corollary to Theorem 1:

Corollary 2. *For every ordinal $0 < \alpha < \epsilon_0$, there are unique $\alpha_1 > \alpha_2 > \cdots > \alpha_n > 0 (n \in \omega)$, $k_1, k_2, \ldots, k_n \in \omega \backslash \{0\}$, and $p \in \omega$ such that $\alpha_1 < \alpha$ and $\alpha = \sum_{i=1}^{n} \omega^{\alpha_i} k_i + p$.*

Thus, we can define another function, CNF to convert any ordinal $\alpha \in \epsilon_0$ into our representation. Let $\sum_{i=1}^{n} \omega^{\alpha_i} k_i + p$ be the CNF decomposition of α according to Corollary 2. Then,

$$\text{CNF}.\alpha = \begin{cases} \alpha, & \text{if } \alpha \in \omega \\ ((\text{CNF}.\alpha_1 \ . \ k_1) \quad (\text{CNF}.\alpha_2 \ . \ k_2) \quad \ldots \quad (\text{CNF}.\alpha_n \ . \ k_n) \quad . \quad p) & \text{otherwise} \end{cases}$$

For the same examples we used before, we get CNF(5) = `5`, CNF$(\omega^4+\omega^4+\omega^2+\omega+3)$ = `((4 . 2) (2 . 1) (1 . 1) . 3)`, and CNF$(\omega^\omega+\omega^2+\omega^2)$ = `(((1 . 0) . 1) (2 . 2) . 0)`.

Lemma 1. *The ordinal representation in Corollary 2 is exponentially more succinct than the representation in Corollary 1.*

Proof Consider $\omega \cdot k$: it requires $O(k)$ bits with the representation in Corollary 1 and $O(\log k)$ bits with the representation in Corollary 2. $\Box$

For example, $f(\omega^5 * 10) =$ `(5 5 5 5 5 5 5 5 5 5 . 0)`, while CNF$(\omega^5 * 10) =$ `((5 . 10) . 0)`. Instead of replicating `5` 10 times, we store the number 10, which is exponentially more succinct.

2.3 Properties

The following are well-known properties of arithmetic operations over the ordinals. Proofs of these properties are distributed across various texts on proof theory and set theory [16, 7, 12]. We use these properties to prove the correctness of our algorithms as follows. Recall that by Corollary 2 we have a canonical representation for the ordinals up to ϵ_0. Thus, to show that an algorithm for some (say binary) arithmetic operator, $\star$, is correct, we show that when the algorithm is given `a` and `b` as inputs, it returns `c`, such that if `a` is the ordinal representation of α and `b` is the ordinal representation of β, then `c` is the ordinal representation of $\alpha \star \beta$. Such proofs are not carried out in ACL2 as this would require formalizing ZFC. Instead, they are carried out in the meta level, using standard mathematical arguments. Detailed proofs appear in [13]; here we only give the proof ideas.

Almost all of the following properties appear as theorems in our library on ordinal representations. The four properties marked with a "†" do not appear in the library, but the proofs are simple consequences the theorems we do prove.

The ordering relation on ordinals satisfies the following properties.

$$\begin{array}{l}
\neg(\alpha < \alpha) \\
\beta < \alpha \Rightarrow \neg(\alpha < \beta) \quad \wedge \quad \neg(\alpha = \beta) \\
\alpha < \beta \quad \wedge \quad \beta < \gamma \Rightarrow \alpha < \gamma \\
\neg(\alpha < \beta) \quad \wedge \quad \neg(\alpha = \beta) \Rightarrow \beta < \alpha
\end{array}$$

Ordinal addition and subtraction satisfy the following properties.

$$\begin{array}{lll}
 & \alpha + 0 = \alpha & \\
 & 0 + \alpha = \alpha & \\
 & \alpha < \alpha + 1 & \\
\dagger & \alpha + 1 \; = \; \alpha' & \\
 & \alpha < \beta \quad \equiv \quad \alpha + 1 \leq \beta & \\
 & \alpha < \beta + 1 \quad \equiv \quad \alpha \leq \beta & \\
 & \alpha \leq \beta + \alpha & \\
 & \alpha \leq \alpha + \beta & \\
 & (\alpha + \beta) + \gamma \; = \; \alpha + (\beta + \gamma) & \text{(associativity)} \\
 & (\beta < \gamma) \; \Rightarrow \; \alpha + \beta < \alpha + \gamma & \text{(strict right monotonicity)} \\
 & (\beta < \gamma) \; \Rightarrow \; \beta + \alpha \leq \gamma + \alpha & \text{(weak left monotonicity)} \\
 & (\alpha < \omega^{\beta}) \; \Rightarrow \; \alpha + \omega^{\beta} = \omega^{\beta} & \text{(additive principal property)} \\
\dagger & (\alpha, \beta < \omega^{\gamma}) \; \Rightarrow \; \alpha + \beta < \omega^{\gamma} & \text{(closure of additive principal ordinals)} \\
 & \alpha - \alpha = 0 & \\
 & \alpha - \beta \leq \alpha & \\
 & \alpha \leq \beta \Rightarrow \alpha + (\beta - \alpha) = \beta & \\
 & \alpha + \gamma = \beta \Rightarrow \beta - \alpha = \gamma & \\
 & \alpha + \beta = \alpha + \gamma \quad \equiv \quad \beta = \gamma &
\end{array}$$

Ordinal multiplication satisfies the following properties.

$$\begin{array}{lll}
 & \alpha 0 = 0 & \\
 & 0 \alpha = 0 & \\
 & \alpha 1 = \alpha & \\
 & 1 \alpha = \alpha & \\
\dagger & n \in \omega \wedge n > 0 \; \Rightarrow \; n \cdot \omega = \omega & \\
 & (\alpha \cdot \beta) \cdot \gamma \; = \; \alpha \cdot (\beta \cdot \gamma) & \text{(associativity)} \\
 & (\beta < \gamma) \; \Rightarrow \; \alpha \cdot \beta < \alpha \cdot \gamma & \text{(strict right monotonicity 1)} \\
 & (\beta \leq \gamma) \; \Rightarrow \; \alpha \cdot \beta \leq \alpha \cdot \gamma & \text{(strict right monotonicity 2)} \\
 & (\beta < \gamma) \; \Rightarrow \; \beta \cdot \alpha \leq \gamma \cdot \alpha & \text{(weak left monotonicity)} \\
 & \alpha \cdot (\beta + \gamma) \; = \; (\alpha \cdot \beta) + (\alpha \cdot \gamma) & \text{(left distributivity)}
\end{array}$$

Ordinal exponentiation has the following properties.

$\alpha^0 = 1$
$\alpha^1 = \alpha$
$0 < \alpha \Rightarrow 0^\alpha = 0$
$1^\alpha = 1$
$\alpha^\beta \cdot \alpha^\gamma \ = \ \alpha^{\beta+\gamma}$
$(\alpha^\beta)^\gamma \ = \ \alpha^{\beta\cdot\gamma}$
$(\beta < \gamma) \ \Rightarrow \ \alpha^\beta < \alpha^\gamma$ (strict right monotonicity)
$(\beta < \gamma) \ \Rightarrow \ \beta^\alpha \leq \gamma^\alpha$ (weak left monotonicity)
† $(p \in \omega) \ \Rightarrow \ p^\omega = \omega$

Limit ordinals satisfy the following properties

$lim.\beta \Rightarrow \alpha < \beta \equiv \alpha + 1 < \beta$
$lim.\beta \ \wedge \ \alpha < \omega^\beta \Rightarrow \langle \exists \gamma :: \alpha < \gamma \ \wedge \ \gamma < \omega^\beta \rangle$

In addition to these theorems, we created counterexamples to the following conjectures in ACL2.

$\alpha + \beta = \beta + \alpha$ fails when $\alpha = 1$, $\beta = \omega$
$\beta < \gamma \Rightarrow \beta + \alpha < \gamma + \alpha$ fails when $\alpha = \omega$, $\beta = 1$, and $\gamma = 2$
$(\alpha + \beta) - \gamma = \alpha + (\beta - \gamma)$ fails when $\alpha = \omega + 1$, $\beta = 1$, and $\gamma = 2$
$\alpha\beta = \beta\alpha$ fails when $\alpha = 2$, $\beta = \omega$
$(\beta + \gamma)\alpha = \beta\alpha + \gamma\alpha$ fails when $\alpha = \omega$, $\beta = 1$, and $\gamma = 1$
$\beta < \gamma \Rightarrow \beta\alpha < \gamma\alpha$ fails when $\alpha = \omega$, $\beta = 1$, and $\gamma = 2$
$(\alpha\beta)^\gamma = \alpha^\gamma\beta^\gamma$ fails when $\alpha = 2$, $\beta = 2$, and $\gamma = \omega$
$(\beta < \gamma) \Rightarrow \beta^\alpha < \gamma^\alpha$ fails when $\alpha = \omega$, $\beta = 2$, and $\gamma = 3$

3 Ordinal arithmetic in ACL2

In this section, we explain our algorithms for the ordinal arithmetic operators given above. Given an expression in our ordinal representation, `a = ((a1 . k1) (a2 . k2) ... (an . kn) . p)`, we call each pair `(ai . ki)` a *term*. If $\sum_{i=1}^{n} \omega^{\alpha_i} k_i + p$ is the ordinal corresponding to `a` in its CNF decomposition, we can think of each term, `(ai . ki)`, as corresponding to $\omega^{\alpha_i} k_i$ (where `ai` corresponds to α_i). The *first exponent* of `a` is `a1` (the first component of the first term of `a`) and the *first coefficient* of `a` is `k1` (the second component of the first term of `a`).

We begin by defining some helper functions. The functions appearing in the library also have guards (which we verify), but such issues are not considered in this paper.

```
(defun natp (x)
  "Recognizer for natural numbers"
  (and (integerp x)
       (<= 0 x)))
```

```
(defun posp (x)
  "Recognizer for positive integers"
  (and (integerp x)
       (< 0 x)))

(defun first-exp (x)
  "The exponent of the first term of a cnf ordinal"
  (if (atom x)
      0
    (caar x)))

(defun first-coef (x)
  "The coefficient of the first term of a cnf ordinal"
  (if (atom x)
      x
    (cdar x)))

(defmacro first-term (x)
  "gives the first term of x"
  `(cons (cons (first-exp ,x) (first-coef ,x)) 0))

(defun omega-term (x k)
  "Creates (w^x)*k"
  (cons (cons x k) 0))
```

3.1 The Irreflexive Ordering

We start by defining `o<`, the ordering function for our ordinal representation.

```
(defun o< (x y)
  "The less-than relation on cnf ordinals"
  (cond ((atom x)
         (or (consp y) (< x y)))
        ((atom y) nil)
        ((not (equal (first-exp x) (first-exp y)))
         (o< (first-exp x) (first-exp y)))
        ((not (= (first-coef x) (first-coef y)))
         (< (first-coef x) (first-coef y)))
        (t (o< (cdr x) (cdr y)))))
```

In the sequel, we assume that α and β are the ordinals corresponding to `x` and `y`, respectively. We now show `(o< x y)` holds iff $\alpha < \beta$. First note that $\alpha > \omega$ iff `x` is a cons. Hence, if `x` is an atom, it is a natural number. Thus, if `y` is a cons, or `y` is a natural number greater than `x`, we return true. Otherwise, `x` is a cons, and therefore an ordinal greater than ω. Thus if `y` is an atom, it is less than `x`. Otherwise, both `x` and `y` are conses, and it follows that α is of the form $\omega^{\alpha_1}k_1 + \gamma$, where $\gamma < \omega^{\alpha_1}$, thus $\alpha < \omega^{\alpha_1}(k_1 + 1)$ Therefore, if the first exponent of y is greater than that of x, we know that x must be less than y. The same argument holds if the first exponent of x is greater than that of y. If the exponents are equal, we look at the coefficients. Again, if one of the arguments has a greater coefficient than the other, we know it is the greater of the two. Finally, if the first exponents and the first coefficients are both equal, then the first terms are equal, so we apply the strong left monotonicity of ordinal addition. By this property we know that x is less than y iff the cdr of x is less than the cdr of y. Thus, we recurse into the cdr of both arguments.

3.2 The Ordinal Predicate

Now we examine op, the predicate which determines if the argument is an ordinal in our representation.

```
(defun op (a)
  "A recognizer for cnf ordinals"
  (if (atom a)
      (natp a)
    (and (consp (car a))
         (op (first-exp a))
         (not (eql 0 (first-exp a)))
         (posp (first-coef a))
         (op (cdr a))
         (o< (first-exp (cdr a))
             (first-exp a)))))
```

The correctness of this function is easy to see, since it directly follows from Corollary 2 and the definition of CNF.

3.3 Addition

Next, we examine ordinal addition. We define here the binary function, ob+, which takes 2 ordinals as input and adds them together.

```
(defun ob+ (x y)
  "Ordinal addition"
  (let ((x0 (first-exp x)) (y0 (first-exp y))
        (x1 (first-coef x)) (y1 (first-coef y)))
    (cond ((and (atom x) (atom y)) (+ x y))
          ((or (atom x) (o< x0 y0)) y)
          ((o< y0 x0) (cons (cons x0 x1)
                            (ob+ (cdr x) y)))
          (t (cons (cons x0 (+ x1 y1))
                   (cdr y))))))
```

If both arguments are atoms, they are natural numbers and are simply added using +. Otherwise, due to the additive principal property, we know that y will smash any term in x with a smaller exponent than the first exponent of y. Hence, we walk down x, copying it, until an exponent is found that is less than or equal to the first exponent of y (the third case of the cond). If the exponents are equal (the last case of the cond), the left distributive property is employed: the first coefficients are added and the rest of y is tacked onto the end. Otherwise, all of y is tacked onto the end (the second case of the cond). In other words, y smashes all the terms of x with smaller exponents.

3.4 Subtraction

Ordinal subtraction is very similar in structure to addition.

```
(defun o- (x y)
  "Ordinal subtraction"
  (let ((x0 (first-exp x)) (y0 (first-exp y))
        (x1 (first-coef x)) (y1 (first-coef y)))
    (cond ((o< x y) 0)
          ((and (atom x) (atom y)) (- x y))
          ((or (atom y) (o< y0 x0)) x)
          ((< y1 x1) (cons (cons x0 (- x1 y1))
                           (cdr x)))
          (t (o- (cdr x) (cdr y))))))
```

If y is greater than x, we return 0. The cases where we are dealing with natural numbers are easy to see. Otherwise, if the first exponent of x is greater than or equal to that of y, we return x, as $\beta + \alpha = \alpha$ (recall that we are assuming that α and β are the ordinals corresponding to x and y, respectively). Otherwise, the first exponents are equal and we look at the coefficients; if they are different, we subtract them and tack on the end of x. Otherwise, we subtract (cdr y) from (cdr x). The correctness follows from the correctness of ob+. It is easily verified that (ob+ x (o- y x)) = y if (o< x y).

3.5 Multiplication

Next, we examine ob*.

```
(defun ob* (x y)
  "Ordinal multiplication"
  (let ((x0 (first-exp x)) (y0 (first-exp y))
        (x1 (first-coef x)) (y1 (first-coef y)))
    (cond ((or (equal x 0) (equal y 0)) 0)
          ((and (atom x) (atom y)) (* x y))
          ((atom y)
           (cons (cons x0 (* x1 y1))
                 (cdr x)))
          (t (cons (cons (o+ x0 y0)
                         y1)
                   (ob* x (cdr y)))))))
```

If either argument to `ob*` is 0, we return 0. If they are both atoms, we multiply them using `*`. If only `y` is an atom, then we are adding `x` to itself `y` times. Now, if we consider the addition algorithm, we see that we smash everything but the first term of all but the last `x`, which is equivalent to modifying `x` so that the first coefficient is `(* x1 y1)` (recall that `y1` is `y` in this case).

For the last case, we have that `y` is a cons. We consider two cases. If `x` is an atom, say representing n, then `(ob* x y)` corresponds to $n(\omega^{\beta_1}l_1 + \cdots + w^{\beta_j}l_j + q)$, which is $\omega^{\beta_1}l_1 + \cdots + w^{\beta_j}l_j + nq$, as for any additive principal ordinal $\alpha > 1$ (*i.e.*, $\alpha = \omega^{\beta}$ for some β) and natural number $p > 0$, $p \cdot \alpha = \alpha$. We return the representation of this ordinal, as `x0` is 0, thus `(o+ x0 y0)` is `y0`. In the second case, both `x` and `y` are infinite ordinals and we are adding `x` to itself an infinite number of times. First we use the left distributive property, multiplying `x` by each term in `y`. If we are multiplying by a natural number, we already know what to do. The interesting part is when we are multiplying by an infinite term, `(y0 . y1)`, representing $\omega^{y_0}y_1$. The key insight here is that `x` is less than `(cons (cons x0 (+ x1 1)) 0)`, and at least as great as `(cons (cons x0 x1) 0)`. But since a natural number multiplied by an infinite additive principal ordinal on the right is just that infinite ordinal, we see that $\omega^{x_0}x_1 \cdot \omega^{y_0}y_1 = \omega^{x_0} \cdot (x_1 \cdot \omega^{y_0}) \cdot y_1 = \omega^{x_0} \cdot \omega^{y_0}y_1 = \omega^{x_0+y_0}y_1$. By the same argument, $\omega^{x_0}(x_1+1) \cdot \omega^{y_0}y_1 = \omega^{x_0+y_0}y_1$. Therefore, we know that `x` multiplied by `(y0 . y1)` is `(cons (o+ x0 y0) y1)`. Recognizing that the exponents of `y` are decreasing and using the strong left monotonicity of addition, we can simply compute each term as shown above and cons them together to get the product.

3.6 Exponentiation

In Figure 1, we present `ob^`. First, we take care of the base cases, when `y` is 0 or `x` is 1 or `x` is 0. If `y` is an atom, we simply return the representation of $\alpha \cdot \alpha^{\beta-1}$ (recall that α, β correspond to `x`, `y`). Otherwise, `y` is a cons and we distribute the exponentiation over `x` and multiply the results, since $\alpha^{(\beta+\gamma)} = \alpha^{\beta} \cdot \alpha^{\gamma}$ for all α, β, γ. How this is done depends on whether `x` is a cons or an atom. When `x` is an atom, say representing n, we use the fact that $n^{\omega} = \omega$, *e.g.*, we use that $n^{\omega^{\alpha}} = n^{\omega^{1+\alpha-1}} = n^{\omega * \omega^{\alpha-1}} = (n^{\omega})^{\omega^{\alpha-1}} = \omega^{\omega^{\alpha-1}}$. Finally, if `x` is a cons, we multiply the first exponent of `x` by the first term of `y` to obtain the first exponent of the first component of the product; the second component of the product is obtained by raising `x` to the `(cdr y)`.

4 Ordinal Library Overview

In this section, we discuss some of the interesting aspects of our library, and give some pointers on how to use it effectively.

4.1 The Books

The ordinal library is composed of ten books. The first is `top-with-meta`, which simply includes the arithmetic books necessary to deal with operations over the natural numbers.

```
(defun ob^ (x y)
  "Ordinal exponentiation"
  (let ((x0 (first-exp x))
        (y0 (first-exp y))
        (y1 (first-coef y)))
    (cond ((or (and (atom y)
                    (not (posp y))) ;(zp y))
               (equal x 1))
           1)
          ((equal x 0)
           0)
          ((atom y)
           (o* x (ob^ x (1- y))))
          ((atom x)
           (if (equal y0 1)
               (o* (omega-term y1 1)
                   (ob^ x (cdr y)))
             (o* (omega-term (omega-term (o- y0 1)
                                         y1)
                             1)
                 (ob^ x (cdr y)))))
          (t
           (o* (omega-term (o* x0 (first-term y))
                           1)
               (ob^ x (cdr y)))))))
```

Fig. 1. Ordinal exponentiation.

The second is `ordinal-definitions`, almost all of which we have already covered. The two omissions so far are the guards and several macros defining our binary operations on an arbitrary number of arguments. The guards are rather straightforward. The macros are defined as shown below.

```
(defmacro o+ (&rest rst)
; based on the macro +
  (if rst
      (if (cdr rst)
          (xxxjoin 'ob+ rst)
        (car rst))
    0))

(defmacro o* (&rest rst)
; based on the macro *
  (cond ((null rst) 1)
        ((null (cdr rst))
         (car rst))
        (t (xxxjoin 'ob* rst))))

(defmacro o^ (&rest rst)
; based on the macro *
  (cond ((null rst) 1)
        ((null (cdr rst))
         (car rst))
        (t (xxxjoin 'ob^ rst))))
```

All of these macros apply the binary operator associated with them to an arbitrary number of arguments, associating the operator to the right. For example, `(o+ a b c d)` opens up to `(ob+ a (ob+ b (ob+ c (ob+ d))))`. This is similar to the way the macros `+` and `*` behave. In order to allow the user to reason about the more readable macros than their messy expansions into the binary operators, we included the following lines in this book.

```
(add-binop o+ ob+)
(add-binop o* ob*)
(add-binop o^ ob^)
```

The `add-binop` command tells ACL2 to write the binary operation in terms of the macro. Again, this is similar to the way ACL2 treats `+`, which is never printed as `binary-+`. This does not affect the way ACL2 reasons about the binary operations. It simply cleans up the output in order to make it more readable.

Also, note that we defined the strict (irreflexive) version of the ordering function. We chose to define this rather than the reflexive version of the ordering ($\leq$) because the irreflexive version is a well-ordering, and therefore more useful for termination proofs. In addition, the irreflexive ordering is necessary to prove our ordinals isomorphic to the ACL2 ordinals. However, in the `basic-definitions-thms` book, we prove *trichotomy*. That is, we prove for any ordinals, `a` and `b`, either `(o< a b)`, `(o< b a)`, or `(equal a b)`. Therefore, it should be easy to define a reflexive ordering and reason about it, for those users who wish to do so.

In addition to trichotomy, the `basic-definitions-thms` book contains basic theorems about the helper functions, as well as `o<` and `op` (the predicate recognizing ordinals in our representation). After these theorems are proved, we disable the function definitions. We found that this not only helped to make the ordinal representation transparent to the user, but also significantly sped up the theorems in subsequent books.

The `ordinal-addition`, `ordinal-multiplication`, and `ordinal-exponentiation` books contain the main results about the ordinal operations. We began with lists of properties about the ordinals similar to the ones presented in this paper, and proved them along with any necessary lemmas. It is, however, important to note that there are a few theorems we export from each file that are not in our list of properties. The most important of these involve building a theory about each function in terms of the helper functions we created. For each of the operators, we include theorems about the value of their first exponents, first coefficients, and `cdr`s, as well as under what circumstances they return atoms (finite ordinals) or conses (infinite ordinals). The importance of these theorems is that they allow us to reason about the ordinals algebraically rather than structurally. Once we build up this theory about the function, we can disable its definition, and still effectively use it to reason about the ordinals.

The `limits` book contains proofs about limit ordinals (in our representation, this corresponds to infinite ordinals with a 0 as their last element).

In the `ordinal-isomorphism` book we show that our representation is isomorphic to the ACL2 ordinals. More precisely, let O_c be the set of objects that correspond to ordinals in our representation, and let O_a be the set of objects representing the ACL2 ordinals. The purpose of this book is to show that O_c is isomorphic to O_a. To this end, we created two functions: `ctoa`, which maps O_c to O_a, and `atoc`, which O_a to O_c. To show that our representation is isomorphic to the ordinals in ACL2, we show the following:

1. `ctoa` is well defined. That is, $\mathtt{x} \in O_c \Rightarrow$ `(ctoa x)` $\in O_a$. Translating this to ACL2 gives us:

   ```
   (implies (op x)
            (e0-ordinalp (ctoa x)))
   ```

   We also proved the equivalent theorem for `atoc`:

   ```
   (implies (e0-ordinalp x)
            (op (ctoa x)))
   ```

2. `ctoa` is surjective. By definition, this means $\langle \forall \mathtt{x} \in O_a :: \langle \exists \mathtt{y} \in O_c ::$ `(ctoa y)` $= \mathtt{x} \rangle\rangle$. We prove this by showing that `atoc` is the inverse of `ctoa`. Thus, given $\mathtt{x} \in O_a$, we have that `(ctoa (atoc x))` $=$ `x`. In ACL2, this becomes:

   ```
   (implies (e0-ordinalp x)
            (equal (ctoa (atoc x))
                   x))
   ```

   We also proved the equivalent theorem for `atoc`:

```
(implies (op x)
         (equal (atoc (ctoa x))
                x))
```

3. `ctoa` is injective, *i.e.*, $\langle \forall \mathtt{x}, \mathtt{y} \in O_c :: $ `(ctoa x)` $=$ `(ctoa y)` $\Rightarrow$ `x` $=$ `y`$\rangle$. In ACL2:

   ```
   (implies (and (op x)
                 (op y))
            (equal (equal (ctoa x) (ctoa y))
                   (equal x y)))
   ```

   We also proved the equivalent theorem for `atoc`:

   ```
   (implies (and (e0-ordinalp x)
                 (e0-ordinalp y))
            (equal (equal (atoc x) (atoc y))
                   (equal x y)))
   ```

4. `ctoa` is homomorphic with respect to `o<` and `e0-ord-<`. That is, $\mathtt{x}, \mathtt{y} \in O_c$ such that `(o< x y)` $\Rightarrow$ `(e0-ord-< (ctoa x) (ctoa y)`. In ACL2, this is:

   ```
   (implies (and (op x)
                 (op y))
            (equal (e0-ord-< (ctoa x)
                             (ctoa y))
                   (o< x y)))
   ```

   and equivalently for `atoc`

   ```
   (implies (and (e0-ordinalp x)
                 (e0-ordinalp y))
            (equal (o< (atoc x)
                       (atoc y))
                   (e0-ord-< x y)))
   ```

Hence, the `ordinal-isomorphism` book shows our representation of the ordinals to be isomorphic to the ACL2 ordinals. As an added bonus, these theorems imply that our representation is a well-founded relation. Hence, we prove the following theorem:

```
(defthm well-founded-cnf
  (and (implies (op x) (e0-ordinalp (ctoa x)))
       (implies (and (op x)
                     (op y)
                     (o< x y))
                (e0-ord-< (ctoa x) (ctoa y))))
  :rule-classes :well-founded-relation)
```

The "`:rule-classes :well-founded-relation`" statement allows the user to set the default well-founded relation used by ACL2 to prove termination to be our ordinal representation. To do this, simply run the following command:

```
(set-well-founded-relation o<)
```

More on the `set-well-founded-relation` command can be found in the ACL2 online documentation [11]. Note that since our relation and representation are identical to the ACL2 ordinals for natural numbers, and since ACL2's automatic measure generation only generates measures that are natural numbers, changing the well-founded relation will not affect ACL2's automatic measure generation.

The final book in our library is `ordinal-counter-examples`, which contains the counter-examples listed earlier in this paper.

4.2 Using the Books

As we stated previously, one of our goals for this library is to allow the user to reason about our ordinal representation and functions algebraically rather than structurally. We have found that ACL2 is rather eager to open up the definitions of our functions, so we recommend that in most cases the user disable the definitions of `o<`, `o+`, `o-`, `o*`, and `o^`. This will force ACL2 to use the theorems we proved about these functions, making it easier for the user to follow ACL2's reasoning; this also tends to lead to significantly shorter proofs.

For this same reason, we recommend that for termination proofs the user construct ordinals algebraically rather than constructively. As a simple example, consider Exercise 6.15 in *Computer-Aided Reasoning: An Approach* [10], which asks the user to come up with a measure for Ackermann's function:

```
(defun ack (x y)
  (if (zp x)
      1
    (if (zp y)
        (if (equal x 1) 2 (+ x 2))
      (ack (ack (1- x) y) (1- y)))))
```

A measure based on the ACL2 ordinals is `(cons (1+ (nfix y)) (nfix x))`, which corresponds to the ordinal $\omega^{y+1} + x$. Using our ordinals, a solution to this exercise is `(o+ (o* (omega) (nfix y)) (nfix x))`, which corresponds to the ordinal $\omega y + x$. First note that the solution based on our ordinals is more easily readable than the solution based on the ACL2 ordinals, as in our solution the representation of the ordinals is completely hidden. Second, note that the ACL2 solution uses much larger ordinals than the solution in our representation, because the ACL2 representation makes it easier to use cons. (No matter what the values of `x` and `y` are, our measure function is guaranteed to return an ordinal less than ω^2, whereas the upper bound on the ACL2 measure is ω^ω.) Finally, note that since our functions return ordinals in our representation when passed ordinals in our representation, the user does not need to worry about whether an ordinal is constructed in all cases. In the ACL2 solution, on the other hand, the user must add 1 to `y` to make sure that the exponent in the constructed ordinal is not a 0.

Finally, we note that the user may need to enable the function definitions when proving theorems about a newly defined function that constructs ordinals directly. For example, if the user were to define a new multiplication function that constructed the ordinals in a different way and wanted to prove it equivalent to `o*`, it may be necessary to enable `o*`'s definition.

5 Complexity

When writing the algebraic operators for this library, we were more concerned with making them easy for ACL2 to reason about than with the complexity. However, we found that our algorithms were surprisingly efficient. This becomes an issue when ACL2 uses the executable counterpart of our functions to compute ground expressions involving ordinal operations or uses the definition as a rewrite rule to simplify expressions involving ordinals. Since releasing this library, we have come up with more efficient algorithms, which will appear in [13]. However, here we present the complexity of `o<` and `op`, which have the same complexity as the version that will appear in the next release of the library.

For our analysis, we assume that the integer operations have constant running time. One can account for integer operations using the fastest known algorithms. We take this approach to allow us to focus on the interesting aspects of the algorithm.

Let the function `(size a)` give the size of an ordinal expression in ACL2. We define it such that

```
(defun size (x)
   (if (atom x)
       1
     (+ (size (first-exp a))
        (size (cdr a)))))
```

Theorem 2. `(o< a b)` *runs in time* $O(\min(\texttt{(size a)}, \texttt{(size b)}))$.

The complexity of this function is bounded by the recurrence relation

$$T(\texttt{a},\texttt{b}) = \begin{cases} c, & \text{if } \texttt{(atom a)} \text{ or } \texttt{(atom b)} \\ T(\texttt{(first-exp a)}, \texttt{(first-exp b)}) + T(\texttt{(cdr a)}, \texttt{(cdr b)}) + c, & \text{otherwise} \end{cases}$$

for some constant value, c. We will show that $T(\texttt{a},\texttt{b}) \leq k \cdot \min(\texttt{(size a)}, \texttt{(size b)}) - t$ for any constants, k, t, such that $t \geq c$ and $k \geq c + t$. If `(atom a)` or `(atom b)`, then $\min(\texttt{(size a)}, \texttt{(size b)}) = 1$. Hence, we get $T(\texttt{a},\texttt{b}) = c \leq k - t = k \cdot \min(\texttt{(size a)}, \texttt{(size b)}) - t$. Otherwise, using the induction hypothesis, we get

$$\begin{aligned}
T(\texttt{a},\texttt{b}) &= T(\texttt{(first-exp a)}, \texttt{(first-exp b)}) + T(\texttt{(cdr a)}, \texttt{(cdr b)}) + c \\
&\leq k \cdot \min(\texttt{(size (first-exp a))}, \texttt{(size (first-exp b))}) - t \\
&\quad + k \cdot \min(\texttt{(size (cdr a))}, \texttt{(size (cdr b))}) - t + c \\
&\leq k \cdot \min(\texttt{(size (first-exp a))} + \texttt{(size (cdr a))}, \\
&\qquad\qquad \texttt{(size (first-exp b))} + \texttt{(size (cdr b))}) - 2t + c \\
&\leq k \cdot \min(\texttt{(size (first-exp a))} + \texttt{(size (cdr a))}, \\
&\qquad\qquad \texttt{(size (first-exp b))} + \texttt{(size (cdr b))}) - t \quad \{\ t \geq c\ \} \\
&\leq k \cdot \min(\texttt{(size a)}, \texttt{(size b)}) - t \quad \square
\end{aligned}$$

Lemma 2. $\langle \forall x \in \omega :: \binom{2x}{x} \geq 2^x \rangle$.

Proof If $x = 0$, then $1 \geq 1$, else $\binom{2x}{x} = \frac{(2x)!}{x!x!} = \frac{(2x)(2x-1)\cdots(x+1)}{x(x-1)\cdots 1} \geq 2 \cdots 2 \geq 2^x$ $\Box$

Lemma 3. $\langle \forall x, y \in \omega :: x \leq y \quad \Rightarrow \quad x^x y^y 2^x \leq (x+y)^{x+y} \rangle$

Proof $(x+y)^{x+y} \;\geq\; \{$ by the binomial theorem $\}$ $x^x y^y \binom{x+y}{x} \;\geq \{y \geq x\}$ $x^x y^y \binom{2x}{x} \;\geq\; \{$ by Lemma 2 $\}$ $x^x y^y 2^x$ $\Box$

Lemma 4. $\langle \forall x, y \in \omega :: x \leq y \quad \Rightarrow \quad x \log x + y \log y + x \;\leq\; (x+y)\log(x+y) \rangle$

Proof By Lemma 3, we know that $x^x y^y 2^x \;\leq\; (x+y)^{x+y}$. By the monotonicity of log, this implies $\log(x^x y^y 2^x) \;\leq\; \log[(x+y)^{x+y}]$, *i.e.*, $\log x^x + \log y^y + \log 2^x \;\leq\; (x+y)\log(x+y)$, *i.e.*, $x \log x + y \log y + x \;\leq\; (x+y)\log(x+y)$. $\Box$

Theorem 3. `(op a)` *runs in time* $O($`(size a)`$(\log$ `(size a)`$))$.

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

$$T(\mathtt{a}) = \begin{cases} c, \quad \text{if } \mathtt{(atom\ a)} \\ T(\mathtt{(first\text{-}exp\ a)}) + T(\mathtt{(cdr\ a)}) \\ \quad + \min(\mathtt{(size\ (first\text{-}exp\ a))}, \mathtt{(size\ (cdr\ a))}) + c, \quad \text{otherwise} \end{cases}$$

for some constant, c, by Theorem 2. We will show by induction on `(size a)`, that $T(\mathtt{a}) \leq k(\mathtt{(size\ a)})(\log \mathtt{(size\ a)}) + t$ 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 $x = \min(\mathtt{(size\ (first\text{-}exp\ a))}, \mathtt{(size\ (cdr\ a))})$ and $y = \max(\mathtt{(size\ (first\text{-}exp\ a))}, \mathtt{(size\ (cdr\ a))})$. We have:

$T(\mathtt{a})$

$= \{$ Definition of T, x $\}$

$T(\mathtt{(first\text{-}exp\ a)}) + T(\mathtt{(cdr\ a)}) + x + c$

$\leq \{$ Induction hypothesis $\}$

$kx \log x + t + ky \log y + t + x + c$

$\leq \{$ $kx \geq 2t + x$ as $k \geq 3t$ $\}$

$k(x \log x + y \log y + x) + c$

$\leq \{$ Lemma 4, $t \geq c$, $x + y = \mathtt{(size\ a)}$ $\}$

$k(\mathtt{(size\ a)})(\log \mathtt{(size\ a)}) + t$ $\Box$

6 Conclusions

We introduced a representation of ordinals up to ϵ_0 that is exponentially more succinct than the ACL2 representation. We also presented algorithms for various arithmetic operations on our representation, along with a library of theorems about these algorithms which significantly extends ACL2's ability to reason about ordinals and ordinal arithmetic. This library is included with ACL2 version 2.7, and has already been used to give a constructive proof of Dickson's Lemma [19].

Despite the fact that the theory of ordinals has been studied for over 100 years, we believe we are the first to give a full solution to the ordinal arithmetic problem [13] and the first to give mechanical proofs of correctness of algorithms for ordinal arithmetic.

We end by outlining directions for future work. In [13] we describe more efficient algorithms than the ones appearing in this paper and we plan to mechanically prove that the two are equivalent. Another possibility involves extending our representation to larger countable ordinals. Of particular interest is the countable ordinal Γ_0, which is much larger than ϵ_0 and is needed to show the termination of some term rewriting systems [5, 8]. There are notational systems for ordinals up to Γ_0 and even for much larger countable ordinals [14, 17, 18], and it would be interesting to find algorithms for manipulating these notations. A final possibility is to explore heuristics that allow ACL2 to guess measure functions based on the ordinals; the hope being that our library would automatically discharge many of the proof obligations.

References

1. G. Cantor. Beiträge zur Begründung der transfiniten Mengenlehre. *Mathematische Annalen*, xlvi:481–512, 1895.
2. G. Cantor. Beiträge zur Bgründung der transfiniten Mengenlehre. *Mathematische Annalen*, xlix:207–246, 1897.
3. G. Cantor. *Contributions to the Founding of the Theory of Transfinite Numbers.* Dover Publications, Inc., 1952. Translated by Philip E. B. Jourdain.
4. A. Church and S. C. Kleene. Formal definitions in the theory of ordinal numbers. *Fundamenta mathematicae*, 28:11–21, 1937.
5. 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.
6. N. Dershowitz and E. M. Reingold. Ordinal arithmetic with list structures. In *Logical Foundations of Computer Science*, pages 117–126, 1992.
7. K. Devlin. *The Joy of Sets: Fundamentals of Contemporary Set Theory.* Springer-Verlag, second edition, 1992.
8. 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.
9. 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.
10. M. Kaufmann, P. Manolios, and J. S. Moore. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, July 2000.
11. M. Kaufmann and J. S. Moore. ACL2 homepage. See URL `http://www.cs.utexas.edu/-users/moore/acl2`.

12. K. Kunen. *Set Theory - an Introduction to Independence Proofs*, volume 102 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1980.
13. P. Manolios and D. Vroon. Algorithms for ordinal arithmetic. In *19th International Conference on Automated Deduction (CADE)*, 2003.
14. L. W. Miller. Normal functions and constructive ordinal notations. *Journal of Symbolic Logic*, 41:439–459, June 1976.
15. H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. MIT Press, 1st paperback edition, 1987.
16. K. Schütte. *Proof Theory*. Springer-Verlag, 1977. translated by J. N. Crossley from the revised version of *Beweistheorie*, 1st edition, 1960.
17. A. Setzer. Ordinal systems. In B. Cooper and J. Truss, editors, *Sets and Proofs*, pages 301–331. Cambridge University Press, 1999.
18. A. Setzer. Ordinal systems part 2: One inaccessible. In *Logic Colloquium '98*, volume 13 of *ASL Lecture Notes in Logic*, pages 426–448, 2000.
19. M. Sustik. Proof of Dixon's lemma using the ACL2 theorem prover via an explicit ordinal mapping. In *ACL2 Workshop 2003*, 2003. Submitted.
20. A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, second edition, 2000.