Parlists – a Generalization of Powerlists (extended version)

Jacob Kornerup
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
E–mail: kornerup@cs.utexas.edu

May 29, 1997

Abstract

The powerlist notation has been very successful in specifying a number of parallel algorithms in a very elegant fashion. The major criticism of the notation was the restriction that input lengths were limited to powers of two. In this paper we present ParList, an extension of the powerlist notation to lists of arbitrary positive lengths. We use the ParList notation to describe a *prefix-sum* algorithm and to describe two addition circuits.

0 Introduction

The powerlist notation [Mis94] has proven to be a major step forward in describing parallel algorithms succinctly. It allows the programmer to work at a high level of abstraction, by avoiding indexing notations, leading towards efficient implementations on parallel architectures [Kor95]. The powerlist data structure is a list whose length is a power of two. In the powerlist notation it is possible to elegantly specify algorithms such as the Discrete Fast Fourier Transform without resorting to "index gymnastics" [Mis94]. For such algorithms this restriction on the lengths is not serious, as they are often presented this way in the literature. However, for most algorithms the restriction is unnatural. To remedy this, Jayadev Misra [Mis96] generalized the powerlist notation to lists of arbitrary length by adding constructs from linear list theory. In this paper we present an extension of the powerlist notation to lists of arbitrary positive lengths[0] and work through a number of examples. This new data structure is called "ParList", which is short for *parallel list*.

1 ParList Theory

A ParList is a non-empty list, whose elements are all of the same type, either scalars from the same base type, or (recursively) ParLists that enjoy the same property. Two ParLists are *similar* if they have the same length and their elements are similar; two scalars are similar when they are from the same base type. We categorize ParLists according to their length. The shortest ParList has length 1, it is called a *singleton*. We denote the singleton containing the scalar x by $\langle x \rangle$.

[0] The theory presented in [Mis96] was incomplete. The author of this paper has completed the theory and worked out the examples presented in this paper. This paper is submitted with permission from Jayadev Misra.

A non-singleton $\mathsf{ParList}$ v can be *deconstructed* into a single element and a $\mathsf{ParList}$ whose length is one less than that of v, using the $\triangleright$ ("cons") and the $\triangleleft$ ("snoc") operator:

$$v = a \triangleright p \ \wedge \ v = q \triangleleft b \tag{0}$$

where a, b and the elements of p and q are similar to the elements of v, and p and q are similar $\mathsf{ParLists}$. In (0) a is the first element of v and b is the last element of v. This definition corresponds to standard list theory, which is well-known from sequential, functional languages like Miranda™ [Tur86], ML [MTH90] and Haskell [HJW$^+$92].

A $\mathsf{ParList}$, p, of even length has the property that it can be deconstructed using the $\bowtie$ ("zip") and the $|$ ("tie") operator:

$$p = u \bowtie v \ \wedge \ p = r \mid s \tag{1}$$

where u is a $\mathsf{ParList}$ containing the elements of p at even positions, and v is the $\mathsf{ParList}$ containing the elements of p at odd positions. Similarly, r is the $\mathsf{ParList}$ containing the first half of p and s is the second half of p; the $\mathsf{ParLists}$ r, s, u and v are all similar.

We formalize the involved types and lengths by introducing the type function $\mathsf{ParList}$ that takes two arguments, a type and a positive integer and returns the type of all $\mathsf{ParLists}$ with elements of the given type and length equal to the given length. Let Type be the type of all types[1] and Pos the type of positive integers, we define $\mathsf{ParList}$ as the function

$$\mathsf{ParList} : \mathsf{Type} \times \mathsf{Pos} \ \longrightarrow \ \mathsf{Type} \tag{2}$$

which returns the type containing all $\mathsf{ParLists}$ with elements from Type, whose length is as specified by the second argument. Using $\mathsf{ParList}$ we can give the signature for the $\mathsf{ParList}$ operators (X is a type and n is in Pos, the positive natural numbers)

$$\langle\ \_\ \rangle \ : \ \mathsf{X} \ \longrightarrow \ \mathsf{ParList.X.1} \tag{3}$$

$$\_\triangleright\_ \ : \ \mathsf{X} \times \mathsf{ParList.X}.n \ \longrightarrow \ \mathsf{ParList.X}.(n+1) \tag{4}$$

$$\_\triangleleft\_ \ : \ \mathsf{ParList.X}.n \times \mathsf{X} \ \longrightarrow \ \mathsf{ParList.X}.(n+1) \tag{5}$$

$$\_\mid\_ \ : \ \mathsf{ParList.X}.n \times \mathsf{ParList.X}.n \ \longrightarrow \ \mathsf{ParList.X}.(2{*}n) \tag{6}$$

$$\_\bowtie\_ \ : \ \mathsf{ParList.X}.n \times \mathsf{ParList.X}.n \ \longrightarrow \ \mathsf{ParList.X}.(2{*}n) \tag{7}$$

We overload the name $\mathsf{ParList}$, by having it denote the type of all parlists (corresponding to $\mathsf{ParList.X}.n$ for all X and n) and naming the algebra we define below. We further refine the type $\mathsf{ParList}$, by introducing the subtype $\mathsf{ParList.X}$ that corresponds to all $\mathsf{ParLists}$ whose elements are taken from X. Finally, we partition the type $\mathsf{ParList.X}$ into the subtypes $\mathsf{Singleton.X}$, $\mathsf{EvenParList.X}$ and $\mathsf{OddParList.X}$, defined respectively as $\mathsf{ParList.X.1}$, $\mathsf{ParList.X}.(2{*}k)$ and $\mathsf{ParList.X}.(2{*}k{+}1)$, where k ranges over Pos. Note that powerlists is a subtype of $\mathsf{ParList}$, corresponding to the lists whose length is a power of two ($\mathsf{ParList.X}.(2^k)$).

We will only write expressions that have a correct type as defined above; e.g. when we write $p \bowtie q$ it is understood that p and q are similar $\mathsf{ParLists}$, i.e. both members of $\mathsf{ParList.X}.n$ for some X and n; when we write $a \triangleright p$ it follows that a is similar to the elements of p.

We use the proof format and notation presented by Dijkstra and Scholten [DS90], this includes writing function application using an infix dot: $f.x$. To minimize the use of parenthesis, we give different binding powers to the operators (most will be defined later in the paper) as prescribed by the table below, where the operators are grouped in decreasing order from left to right, and operators in the same group have equal binding power:

[1] This is just a name. We will not do any reasoning based on types. The worried reader should skip the following definition, as Type does not appear elsewhere in this paper

.	$\rightarrow\ \leftarrow$	$\triangleright\ \triangleleft$	$\bowtie\ \mid$	$\oplus\ +\ \odot\ \star\ \bullet\ \div$ mod	$=$	$\Rightarrow$	$\wedge\ \vee$	$\equiv$

Remark The definitions that define the constructors for ParList are similar to how one might define the function *power* : Real $\times$ Pos $\longrightarrow$ Real, that computes the value of its first argument raised to the power of its second argument, i.e. $power.x.n = x^n$. We can define *power* recursively as follows:

$$power.x.1 = x \tag{8}$$
$$power.x.(2{*}n+1) = x{*}power.x.(2{*}n) \tag{9}$$
$$power.x.(2{*}n) = (power.x.n)^2 \tag{10}$$

the choices for inductive cases where rather arbitrary, as we could equally well have chosen:

$$power.x.(2{*}n+1) = power.x.(2{*}n) * x \tag{11}$$
$$power.x.(2{*}n) = power.x^2.n \tag{12}$$

Note how (10) and (12) corresponds to (1), and (9) and (11) corresponds to (0). *End Remark*

1.0 Axioms

In the following we extend the axioms of the powerlist theory [Mis94] to an axiomatization of the ParList algebra. The ParList algebra has five constructors: $\langle\ \rangle, |, \bowtie, \triangleright$ and $\triangleleft$. They are all *isomorphisms* on their respective domains, with the following laws as consequence, where $p, q, u, v \in$ ParList.X.$n\ \wedge\ a, b, c \in$ X:

$$\langle a\rangle = \langle b\rangle \equiv a = b \tag{13}$$
$$p \mid q = u \mid v \equiv p = u\ \wedge\ q = v \tag{14}$$
$$p \bowtie q = u \bowtie v \equiv p = u\ \wedge\ q = v \tag{15}$$
$$a \triangleright p = b \triangleright q \equiv a = b\ \wedge\ p = q \tag{16}$$
$$p \triangleleft a = q \triangleleft b \equiv a = b\ \wedge\ p = q \tag{17}$$

$$(\forall t : t \in \mathsf{ParList.X}.1 : (\exists\, a :: t = \langle a\rangle)) \tag{18}$$
$$(\forall t : t \in \mathsf{ParList.X}.(2{*}n) : (\exists\, u, v :: t = u \mid v)) \tag{19}$$
$$(\forall t : t \in \mathsf{ParList.X}.(2{*}n) : (\exists\, u, v :: t = u \bowtie v)) \tag{20}$$
$$(\forall t : t \in \mathsf{ParList.X}.(n+1) : (\exists\, a, p :: t = a \triangleright p)) \tag{21}$$
$$(\forall t : t \in \mathsf{ParList.X}.(n+1) : (\exists\, b, q :: t = q \triangleleft b)) \tag{22}$$

The following axioms are from the powerlist theory:

$$\langle a\rangle \bowtie \langle b\rangle = \langle a\rangle \mid \langle b\rangle \tag{23}$$
$$(p \mid q) \bowtie (u \mid v) = (p \bowtie u) \mid (q \bowtie v) \tag{24}$$

The remaining axioms extends the powerlist algebra to define the full ParList algebra.

$$a \triangleright (p \mid q) \bowtie (u \mid v) \triangleleft b = a \triangleright (u \bowtie p) \mid (v \bowtie q) \triangleleft b \tag{25}$$
$$a \triangleright \langle b\rangle = \langle a\rangle \mid \langle b\rangle \tag{26}$$

$$\begin{array}{rcll}
\langle a\rangle \triangleleft b &=& \langle a\rangle \mid \langle b\rangle & (27)\\
a \triangleright (p \triangleleft b) &=& (a \triangleright p) \triangleleft b & (28)\\
a \triangleright (p \bowtie q) = (u \bowtie p) \triangleleft b &\equiv& a \triangleright q = u \triangleleft b & (29)\\
a \triangleright (p \mid q) = (u \mid v) \triangleleft c &\equiv& (\exists\, b :: a \triangleright p = u \triangleleft b \;\wedge\; b \triangleright q = v \triangleleft c) & (30)
\end{array}$$

Note the symmetry between $\bowtie$ and $\mid$ in axiom (24). Without an operational model the roles of $\bowtie$ and $\mid$ can be interchanged in the powerlist algebra. This is not the case when we consider the ParList algebra. If we interpret $\triangleright$ and $\triangleleft$ as prepending and appending an element to a ParList then the contrast between (29) and (30) and between (25) and (24) precisely capture the operational difference between $\bowtie$ and $\mid$.

Let $\oplus$ be a binary operator, defined on a scalar type. We lift $\oplus$ to operate on ParList over elements of that type with the following laws:

$$\begin{array}{rcll}
\langle a\rangle \oplus \langle b\rangle &=& \langle a \oplus b\rangle & (31)\\
(a \triangleright p) \oplus (b \triangleright q) &=& (a \oplus b) \triangleright (p \oplus q) & (32)\\
(p \bowtie q) \oplus (u \bowtie v) &=& (p \oplus u) \;\bowtie\; (q \oplus v) & (33)
\end{array}$$

As alternatives to (32) and (33) we could have chosen (34) and (35) as they are interchangeable:

$$\begin{array}{rcll}
(p \triangleleft a) \oplus (q \triangleleft b) &=& (p \oplus q) \triangleleft (a \oplus b) & (34)\\
(p \mid q) \oplus (u \mid v) &=& (p \oplus u) \;\mid\; (q \oplus v) & (35)
\end{array}$$

It is a worthwhile exercise to prove that (34) and (35) follows from (31), (32) and (33).

From (29) and (22) we can derive the following lemma, that is useful in proofs of properties of ParLists.

Lemma 0

$$(\forall a, p, q :: (\exists\, b, u, v \;::\; a \triangleright (p \bowtie q) = (u \bowtie v) \triangleleft b \;\wedge\; a \triangleright q = u \triangleleft b \;\wedge\; p = v)) \qquad (36)$$

$$(\forall b, u, v :: (\exists\, a, p, q \;::\; a \triangleright (p \bowtie q) = (u \bowtie v) \triangleleft b \;\wedge\; a \triangleright q = u \triangleleft b \;\wedge\; p = v)) \qquad (37)$$

Proof of (36) ((37) is similar)

true

$\equiv$ { axiom (22) }

$(\forall a, q :: (\exists\, b, u :: a \triangleright q = u \triangleleft b))$

$\equiv$ { axiom (29) }

$(\forall a, p, q :: (\exists\, b, u :: a \triangleright (p \bowtie q) = (u \bowtie p) \triangleleft b \;\wedge\; a \triangleright q = u \triangleleft b))$

$\equiv$ { one-point rule and trading }

$(\forall a, p, q :: (\exists\, b, u, v :: a \triangleright (p \bowtie q) = (u \bowtie v) \triangleleft b \;\wedge\; a \triangleright q = u \triangleleft b \;\wedge\; p = v))$

End of Proof

1.1 Functions in ParList

Functions over ParList are defined by three different cases based on the length of the argument ParList: singleton, even length and odd length. Each case is defined using pattern-matching on the argument ParList: $\langle\;\rangle$ for singletons, $\bowtie$ or $\mid$ for even length lists, and $\triangleright$ or $\triangleleft$ for odd length lists.

Subtype	Allowed Constructors
Singleton.X	$\langle\ \rangle$
EvenParList.X	$\bowtie \quad \mid$
OddParList.X	$\triangleright \quad \triangleleft$

We insist that $\triangleright$ and $\triangleleft$ only be used for ParLists of odd length in function definitions, since we want to exploit parallelism as much as possible. When the argument has an even length, the computation should be expressed using a balanced divide-and-conquer strategy. Arguments of odd length should be treated as an alignment step, introduced by necessity.

As an example, we define the function *rev* that reverses its argument.

$$rev.\langle a\rangle \quad = \quad \langle a\rangle \tag{38}$$

$$rev.(p \bowtie q) \quad = \quad rev.q \bowtie rev.p \tag{39}$$

$$rev.(a \triangleright p) \quad = \quad rev.p \triangleleft a \tag{40}$$

Note that the choice of $\bowtie$ and $\triangleright$ as destructors was arbitrary. A definition using $\mid$ and/or $\triangleleft$ in their place yields the same function. This is similar to the observation after the definition of the lifting of scalar operators to ParLists, (31) to (35). In the definition of *rev*, (39) expresses that each recursive case is independent and can be evaluated in parallel. The step described by (40) corresponds to a sequential "alignment" step, necessary before a balanced recursive step can be performed. In the case of *rev* the "alignment" step does not have to be sequential; depending on the parallel architecture (and the concrete implementation of ParList) *rev* can be evaluated in constant time. This would be the case on a CREW PRAM with the straightforward implementation of ParList.

A familiar property of *rev* is that it is its own inverse, which we prove below.

$$rev.(rev.p) = p \tag{41}$$

Proof of (41), base case:

$$\begin{array}{ll} & rev.(rev.\langle a\rangle) \\ = & \{\ rev\ (38)\ \} \\ & rev.\langle a\rangle \\ = & \{\ rev\ (38)\ \} \\ & \langle a\rangle \end{array}$$

Inductive even case:

$$\begin{array}{ll} & rev.(rev.(p \bowtie q)) \\ = & \{\ rev\ (39)\ \} \\ & rev.(rev.q \bowtie rev.p) \\ = & \{\ rev\ (39)\ \} \\ & rev.(rev.p) \bowtie rev.(rev.q) \\ = & \{\ \text{induction (41) twice}\ \} \\ & p \bowtie q \end{array}$$

Inductive odd case:

$$\begin{array}{ll} & rev.(rev.(a \triangleright (p \bowtie q))) \\ = & \{\ rev\ (40)\ \} \\ & rev.(rev.(p \bowtie q) \triangleleft a) \\ = & \{\ rev\ (39)\ \} \\ & rev.((rev.q \bowtie rev.p) \triangleleft a) \end{array}$$

Combining both sides of the odd case:

$$\begin{array}{ll} & a \triangleright (p \bowtie q) \ = \ (rev.v \bowtie rev.u) \triangleleft b \\ \equiv & \{\ \text{Axiom (29)}\ \} \\ & a \triangleright q = rev.v \triangleleft b \ \wedge\ p = rev.u \\ \equiv & \{\ rev(40)\ \} \\ & a \triangleright q = rev.(b \triangleright v) \ \wedge\ p = rev.u \end{array}$$

$$
\begin{array}{ll}
= & \{ \text{ See (42) below } \} \\
 & rev.(b \triangleright (u \bowtie v)) \\
= & \{ \ rev \ (40) \ \} \\
 & rev.(u \bowtie v) \triangleleft b \\
= & \{ \ rev \ (39) \ \} \\
 & (rev.v \bowtie rev.u) \triangleleft b
\end{array}
\qquad
\begin{array}{ll}
\equiv & \{ \text{ See (42) below } \} \\
 & a \triangleright q = rev.(rev.q \triangleleft a) \ \wedge \ p = rev.(rev.p) \\
\equiv & \{ \ rev(40), \text{ induction (41) } \} \\
 & a \triangleright q = rev.(rev.(a \triangleright q)) \\
\equiv & \{ \text{ induction (41) } \} \\
 & a \triangleright q = a \triangleright q \\
\equiv & \{ \text{ predicate calculus } \} \\
 & \mathsf{true}
\end{array}
$$

End of Proof

In the above we used Lemma 0 and axiom (29) to establish

$$(\exists\, b, u, v :: b \triangleright (u \bowtie v) = (rev.q \bowtie rev.p) \triangleleft a \ \wedge \ b \triangleright v = rev.q \triangleleft a \ \wedge \ u = rev.p) \tag{42}$$

1.2 Broadcast Sum

We turn to the definition of the function $sum : \mathsf{ParList.Y}.n \longrightarrow \mathsf{ParList.Y}.n$, that returns a list where each element is the sum of all the elements of the argument list (a broadcast sum). Here Y is a type with the property that $(\mathsf{Y}, +)$ is a semigroup. It is necessary to define the functions $last : \mathsf{ParList.X} \longrightarrow \mathsf{X}$, which returns the last element of a list, and $[a+] : \mathsf{ParList.Y}.n \longrightarrow \mathsf{ParList.Y}.n$, which returns the list where a has been added to each element of the argument list.

$$
\begin{array}{rcll}
sum.\langle a \rangle & = & a & (43) \\
sum.(a \triangleright p) & = & (a + last.t) \ \triangleright \ [a+].t, \quad \text{where } t = sum.p & (44) \\
sum.(p \bowtie q) & = & t \bowtie t, \quad \text{where } t = sum.(p+q) & (45) \\
last.\langle a \rangle & = & a & (46) \\
last.(p \triangleleft b) & = & b & (47) \\
last.(p \mid q) & = & last.q & (48) \\
{[a+]}.\langle b \rangle & = & \langle a + b \rangle & (49) \\
{[a+]}.(b \triangleright p) & = & (a+b) \ \triangleright \ [a+].p & (50) \\
{[a+]}.(p \mid q) & = & [a+].p \ \mid \ [a+].q & (51) \\
first.\langle a \rangle & = & a & (52) \\
first.(a \triangleright p) & = & a & (53) \\
first.(p \mid q) & = & first.p & (54)
\end{array}
$$

When sum is evaluated with an argument of length $2^n - 1,\ n \geq 1$ there are $n-1$ deconstructions using $\triangleright$ and $n-1$ deconstructions using $\bowtie$. Each deconstruction takes one parallel time step, in order to perform the sum. The total number of parallel steps thus becomes $2{*}n - 2$. In contrast, if the argument is of length 2^n, only n parallel steps are needed. Adding a sufficient number of dummy elements (i.e. identity elements of $+$) to a list makes it into a powerlist. Thus, functions like sum can be evaluated in parallel in fewer steps than with the original list.

1.3 Reusing Powerlist Proofs in the ParList Algebra

One of the advantages of the ParList algebra is that it is a conservative extension of the powerlist algebra. As a consequence any result proven about a function defined in the powerlist algebra holds for those parlists that are also powerlists, i.e. whose length is a power of two.

Moreover, when powerlist function definitions are extended with an odd case they become ParList functions. Inductive proofs of properties done in the powerlist algebra can be reused in the proof of the same property for the extended function in the ParList algebra. Depending on the structure of the powerlist proof, the only remaining proof obligation may be to prove the odd case. Take as an example the function *rev* defined in the powerlist algebra by (38) and (39). A proof of (41) in the powerlist algebra consisting of the base and even cases is sufficient to prove (41) in the powerlist algebra. When (40) is added to make *rev* a ParList function, the odd case is the only missing part of the proof; the two others can be reused. A requirement is that the reused proof does not use properties that are specific to powerlists, e.g. properties like

$$\mathit{length.p} \text{ is even} \quad \Rightarrow \quad \mathit{length.p} \text{ is a power of 2.}$$

1.4 Prefix Sum

Prefix sum is a fundamental parallel algorithm; it is used in many algorithms as a building block, e.g. carry lookahead addition (see Sect. 2). The prefix sum of a ParList p over a data type Y, with the property that $(\mathsf{Y}, +, 0)$ is a monoid, can be defined [Mis94] as the (unique) solution to the equation (in u):

$$u = (0 \rightarrow u) + p \tag{55}$$

where the operator $\rightarrow$ takes a element and a ParList and "pushes" a scalar into the list from the left and the rightmost element of the list is lost. $\rightarrow$ has a higher binding power than that of $\bowtie, |, \triangleright$ and $\triangleleft$; it is defined as follows:

$$a \rightarrow \langle b \rangle \quad = \quad \langle b \rangle \tag{56}$$
$$a \rightarrow (p \triangleleft b) \quad = \quad a \triangleright p \tag{57}$$
$$a \rightarrow (p \bowtie q) \quad = \quad a \rightarrow q \;\bowtie\; p \tag{58}$$

The dual operator $\leftarrow$: $\mathsf{ParList.X}.n \times \mathsf{X} \longrightarrow \mathsf{ParList.X}.n$ "pushes" a scalar into the list from the right and the leftmost element of the list is lost. $\leftarrow$ has the same binding power as $\rightarrow$; it is defined as follows:

$$\langle b \rangle \leftarrow a \quad = \quad \langle b \rangle \tag{59}$$
$$(b \triangleright p) \leftarrow a \quad = \quad p \triangleleft a \tag{60}$$
$$(p \bowtie q) \leftarrow a \quad = \quad q \;\bowtie\; p \leftarrow a \tag{61}$$

Exploring the defining equation for prefix sum (55), we can derive a scheme for computing the prefix sum, due to Ladner & Fischer [LF80]. Misra [Mis94] derived the base (62) and even (63) cases for powerlists, we present a version that is similar to his below and derive the odd case.

Even case

$\quad ps.(p \bowtie q)$

$= \quad \{$ Defining equation for ps (55) $\}$

$\quad 0 \rightarrow ps.(p \bowtie q) + p \bowtie q$

$=\ \{$ define $u, v := ps.(p \bowtie q)$ $\}$

$0 \rightarrow (u \bowtie v) + p \bowtie q$

$=\ \{ \rightarrow (58) \}$

$0 \rightarrow v \bowtie u + p \bowtie q$

$=\ \{$ Axiom (31) $\}$

$(0 \rightarrow v + p) \ \bowtie\ (u + q)$

$=\ \{$ By definition of u, v $\}$

$u \bowtie v$

Summarizing:

$u \bowtie v \ = \ (0 \rightarrow v + p) \ \bowtie\ (u + q)$

$\equiv\ \{$ Axiom (15) $\}$

$u = 0 \rightarrow v + p \ \wedge\ v = u + q$

$\Rightarrow\ \{$ Solving for v $\}$

$u = 0 \rightarrow v + p \ \wedge\ v = 0 \rightarrow v + p + q)$

$\equiv\ \{$ Defining equation for ps (55) $\}$

$u = 0 \rightarrow v + p \ \wedge\ v = ps.(p + q)$

$\Rightarrow\ \{$ Solving for u $\}$

$u = 0 \rightarrow ps.(p + q) + p \ \wedge\ v = ps.(p + q)$

$\equiv\ \{$ Definition of u, v and axiom (15) $\}$

$ps.(p \bowtie q) \ = \ 0 \rightarrow ps.(p + q) + p \ \bowtie\ ps.(p + q)$

We explore the odd case. By introducing q and b such that $ps.(p \triangleleft a) = q \triangleleft b$ we get:

$q \triangleleft b$

$=\ \{$ Defining equation for ps (55) $\}$

$0 \rightarrow (q \triangleleft b) + p \triangleleft a$

$=\ \{ \rightarrow (57) \}$

$0 \triangleright q + p \triangleleft a$

$=\ \{$ Lemma 1 (65), below $\}$

$0 \rightarrow q \triangleleft last.q + p \triangleleft a$

$=\ \{$ Axiom (32) $\}$

$(0 \rightarrow q + p) \triangleleft (last.q + a)$

Summarizing:

$q \triangleleft b \ = \ (0 \rightarrow q + p) \triangleleft (last.q + a)$

$\equiv\ \{$ Axiom (17) $\}$

$q = 0 \rightarrow q + p \ \wedge\ b = last.q + a$

$\equiv\ \{$ Defining equation for ps (55), Leibnitz Rule $\}$

$q = ps.p \ \wedge \ b = last.(ps.p) + a$

From the above along with Misra's definition , we get the following definition of Ladner and Fischer's algorithm:

$$
\begin{aligned}
ps.\langle a\rangle &= \langle a\rangle & (62)\\
ps.(p \bowtie q) &= (0{\rightarrow}t + p) \ \bowtie \ t, \quad \text{where } t = ps.(p+q) & (63)\\
ps.(p \triangleleft a) &= ps.p \triangleleft \ (last.(ps.p) + a) & (64)
\end{aligned}
$$

In the proof above we used Lemma 1

Lemma 1 $\forall a, p : a \in \mathsf{X} \ \wedge \ p \in \mathsf{ParList.X} :$

$$
\begin{aligned}
a \triangleright p &= a{\rightarrow}p \triangleleft \ last.p & (65)\\
p \triangleleft a &= first.p \ \triangleright \ p \leftarrow a & (66)
\end{aligned}
$$

Proof of (65); the proof of (66) is similar. Even inductive case:

$a \triangleright (p \bowtie q) \ = \ a{\rightarrow}(p \bowtie q) \triangleleft \ last.(p \bowtie q)$

$\equiv \ \{ \ \rightarrow (58), \ last \ (48) \ \}$

$a \triangleright (p \bowtie q) \ = \ (a{\rightarrow}q \bowtie p) \triangleleft \ last.q$

$\equiv \ \{$ Axiom (29) $\}$

$a \triangleright q \ = \ a{\rightarrow}q \triangleleft \ last.q$

$\equiv \ \{$ Induction (65) $\}$

true

Odd inductive case:

$a \triangleright (p \triangleleft b) \ = \ a{\rightarrow}(p \triangleleft b) \triangleleft b$

$\equiv \ \{ \ \rightarrow (58), \ last \ (48) \ \}$

$a \triangleright (p \triangleleft b) \ = \ (a \triangleright p) \triangleleft b$

$\equiv \ \{$ Axiom (28) $\}$

true

Base case:

$a{\rightarrow}\langle x\rangle \triangleleft \ last.\langle x\rangle$

$= \ \{ \rightarrow (56)$ and $last$ (46) $\}$

$\langle a\rangle \triangleleft x$

$= \ \{$ (26) and (27) $\}$

$a \triangleright \langle x\rangle$

End of Proof

1.5 Concatenation

A very useful operation on lists is to append one list onto another, regardless of the length of the lists. We define the concatenation operator $\diamondsuit$: $\mathsf{ParList.X}.n \times \mathsf{ParList.X}.m \longrightarrow \mathsf{ParList.X}.(n+m)$ by the following equations. Note that $\diamondsuit$ has a binding power that is between that of $\bowtie$ and $|$ and the scalar operators.

$$\langle a\rangle \diamondsuit \langle b\rangle = \langle a\rangle \bowtie \langle b\rangle \tag{67}$$

$$\langle a\rangle \diamondsuit (p\bowtie q)\triangleleft b = a\triangleright p \bowtie q\triangleleft b \tag{68}$$

$$\langle a\rangle \diamondsuit (p\bowtie q) = a\triangleright(p\bowtie q) \tag{69}$$

$$(p\bowtie q)\triangleleft a \diamondsuit \langle b\rangle = (p\triangleleft a) \bowtie (q\triangleleft b) \tag{70}$$

$$a\triangleright(p\bowtie q) \diamondsuit (u\bowtie v)\triangleleft b = a\triangleright(p\diamondsuit u) \bowtie (q\diamondsuit v)\triangleleft b \tag{71}$$

$$a\triangleright(p\bowtie q) \diamondsuit u\bowtie v = a\triangleright(p\diamondsuit u) \bowtie (q\diamondsuit v)\triangleleft b \tag{72}$$

$$p\bowtie q \diamondsuit \langle a\rangle = (p\bowtie q)\triangleleft a \tag{73}$$

$$p\bowtie q \diamondsuit (u\bowtie v)\triangleleft a = ((p\diamondsuit u) \bowtie (q\diamondsuit v))\triangleleft a \tag{74}$$

$$p\bowtie q \diamondsuit u\bowtie v = (p\diamondsuit u) \bowtie (q\diamondsuit v) \tag{75}$$

By its nature $\diamondsuit$ is a generalization of $|$, so it is no surprise that $\diamondsuit$ is defined using $\bowtie$ as the constructor. It does not appear as though $|$ can be used as the defining constructor. Note the similarity between (24) and (75); in fact, by remove the equations above where the arguments to $\diamondsuit$ have different length (68), (69), (70), (72), (73) and (74) we are left with axioms that define an operator isomorphic to $|$. restricting the type of arguments of $\diamondsuit$ to lists of equal length and only keeping those equations that make sense under this restriction ((67), (71) and (75)) we have defined an operator that is isomorphic to $|$.

Since $\diamondsuit$ is a generalization of $|$, one could ask why $\diamondsuit$ was not chosen as one of the fundamental constructors for ParList. The arguments of $|$ and $\bowtie$ are of equal length, enforcing a balanced construction, which is essential to obtaining efficient parallel implementations. Many properties that hold for $|$ hold for $\diamondsuit$ as well; however, they are more tedious to prove since there are 9 defining cases to consider. We list a few properties of $\diamondsuit$ below:

$$\mathit{first}.(p\diamondsuit q) = \mathit{first}.p \tag{76}$$

$$\mathit{last}.(p\diamondsuit q) = \mathit{last}.q \tag{77}$$

$$a{\rightarrow}(p\diamondsuit q) = a{\rightarrow}p \diamondsuit \mathit{last}.p{\rightarrow}q \tag{78}$$

$$(p\diamondsuit q){\leftarrow}a = p{\leftarrow}\mathit{first}.q \diamondsuit q{\leftarrow}a \tag{79}$$

$$[a+].(p\diamondsuit q) = [a+].p \diamondsuit [a+].q \tag{80}$$

$$\mathit{sum}.(p\diamondsuit q) = \mathit{sum}.p \diamondsuit \mathit{sum}.q \tag{81}$$

One important law that holds for $|$ but not for $\diamondsuit$ is (35), due to the ambiguity that arises when deconstructing the arguments using $\diamondsuit$.

2 Adder circuits

In [Ada94] Will Adams presented powerlist descriptions for two arithmetic circuits that perform addition on natural numbers: the *ripple carry adder* and the *carry lookahead adder*. The ripple carry adder performs addition as it is first taught in grade school; it is an inheritly sequential method, yielding a linear time method in the number of bits to be added. The carry lookahead adder uses a prefix sum calculation to propagate carries, yielding a method that is logarithmic in the number of bits to be added, in a setting where sufficient parallelism available.

Adams proved that the ripple carry circuit correctly implements addition and that the carry lookahead and the ripple carry circuits are the same function. This result was achieved in the powerlist algebra. Since the powerlist algebra only contains lists whose length are a power of two, and there are no a priori restrictions on the length of either addition circuit, these circuits should be specified as ParList functions.

In the following we extend the definition of the addition circuits and the equivalence result to the ParList algebra. The ripple carry adder takes three arguments:

$$rc : \{0,1\} \times \mathsf{ParList}.\{0,1\}.n \times \mathsf{ParList}.\{0,1\}.n \longrightarrow \mathsf{ParList}.\{0,1\}.n \times \{0,1\}$$

the first argument is the carry-in bit and the second and third argument are the two ParLists of bits that are to be added. The result is a pair; the first component of the pair is a ParList containing the result of the addition, and the second component is the carry-out bit from the addition. The following defines rc, where (82) and (83) are taken from [Ada94]:

$$rc.b.\langle x\rangle.\langle y\rangle \;=\; (\langle (x+y+b) \bmod 2\rangle, (x+y+b) \div 2) \tag{82}$$

$$\begin{aligned} rc.b.(p \mid q).(r \mid s) \;&=\; (t,d) \\ &\text{where} \quad \begin{aligned} t &= u \mid v \\ (u,c) &= rc.b.p.r \\ (v,d) &= rc.c.q.s \end{aligned} \end{aligned} \tag{83}$$

$$\begin{aligned} rc.c.(p \triangleleft a).(q \triangleleft b) \;&=\; (u \triangleleft y, x) \\ &\text{where} \quad \begin{aligned} x &= (a+b+d) \div 2 \\ y &= (a+b+d) \bmod 2 \\ (u,d) &= rc.c.p.q \end{aligned} \end{aligned} \tag{84}$$

The carry lookahead adder has the following type

$$cl : \{0,1,\pi\} \times \mathsf{ParList}.\{0,1,\pi\}.n \times \mathsf{ParList}.\{0,1,\pi\}.n \longrightarrow \mathsf{ParList}.\{0,1,\pi\}.n \times \{0,1,\pi\}$$

where π corresponds to a "propagate" action for the carry-in value to a position. To specify the carry lookahead adder, Adams introduces the associative scalar operators $\bullet$, $\star$ and $\odot$ defined by:

$$\bullet : \{0,1,\pi\} \times \{0,1,\pi\} \longrightarrow \{0,1,\pi\} \qquad x \bullet y = \begin{cases} x & \text{if } x = y \\ \pi & \text{if } x \neq y \end{cases} \tag{85}$$

$$\star : \{0,1,\pi\} \times \{0,1,\pi\} \longrightarrow \{0,1,\pi\} \qquad x \star y = \begin{cases} y & \text{if } y \neq \pi \\ x & \text{if } y = \pi \end{cases} \tag{86}$$

$$\odot : \{0,1,\pi\} \times \{0,1,\pi\} \longrightarrow \{0,1,\pi\} \qquad x \odot y = \begin{cases} x & \text{if } y \neq \pi \\ \neg y & \text{if } y = \pi \end{cases} \tag{87}$$

$$\text{where} \quad \begin{aligned} \neg 0 &= 1 \\ \neg 1 &= 0 \\ \neg \pi &= \pi \end{aligned}$$

Adams [Ada94] defines the carry lookahead adder by

$$\begin{aligned} cl.b.p.q \;&=\; (t,d) \\ &\text{where} \quad \begin{aligned} t &= s \odot r \\ d &= last.s \star last.r \\ r &= p \bullet q \\ s &= ps.(b \rightarrow r) \end{aligned} \end{aligned} \tag{88}$$

where ps is computed using the associative operator $\star$ (that has π as its neutral element). Expanding the odd case of the definition of cl we get:

$$\begin{array}{rcl} cl.c.(p\triangleleft x).(q\triangleleft y) & = & (a, w) \\ & & \text{where} \quad \begin{array}{rcl} w & = & u \odot v \\ a & = & last.u \star last.v \\ v & = & (p\triangleleft x) \bullet (q\triangleleft y) \\ u & = & ps.(b \rightarrow v) \end{array} \end{array} \tag{89}$$

Comparing this with the quantities defined by $cl.b.p.q$ (88) we get

$$\begin{array}{ll} v & \\ = \{ \ (89) \ \} & \\ p\triangleleft x \bullet q\triangleleft y & \\ = \{ \ \bullet \text{ is scalar } \} & \\ (p \bullet q)\triangleleft (x \bullet y) & \\ = \{ \ (88) \ \} & \\ r\triangleleft (x \bullet y) & \end{array}$$

$$\begin{array}{l} u \\ = \{ \ (89) \ \} \\ ps.(b \rightarrow v) \\ = \{ \ \text{calculation on the left} \ \} \\ ps.(b \rightarrow (r\triangleleft (x \bullet y))) \\ = \{ \ \rightarrow (57) \ \} \\ ps.(b \triangleright r) \\ = \{ \ \text{lemma 1 } (65) \ \} \\ ps.((b \rightarrow r)\triangleleft last.r) \\ = \{ \ ps \ (64) \ \} \\ ps.(b \rightarrow r)\triangleleft (last.(ps.(b \rightarrow r)) \star last.r) \\ = \{ \ (88) \ \} \\ s\triangleleft (last.s \star last.r) \end{array}$$

$$\begin{array}{l} a \\ = \{ \ cl \ (89) \ \} \\ last.u \star last.v \\ = \{ \ \text{calculations above} \ \} \\ last.(s\triangleleft (last.s \star last.r)) \star last.(r\triangleleft (x \bullet y)) \\ = \{ \ last \ (47) \ \} \\ last.s \star last.r \star (x \bullet y) \\ = \{ \ cl \ (88) \ \} \\ d \star (x \bullet y) \end{array}$$

$$\begin{array}{l} w \\ = \{ \ cl \ (89) \ \} \\ u \odot v \\ = \{ \ \text{calculations above} \ \} \\ (s\triangleleft (last.s \star last.r)) \odot (r\triangleleft (x \bullet y)) \\ = \{ \ \text{Axiom } (34) \ \} \\ (s \odot r)\triangleleft ((last.s \star last.r) \odot (x \bullet y)) \\ = \{ \ cl \ (88) \ \} \\ t\triangleleft (d \odot (x \bullet y)) \end{array}$$

in summary we have

$$\begin{array}{rcl} cl.c.(p\triangleleft x).(q\triangleleft y) & = & (t\triangleleft (d \odot (x \bullet y)), d \star (x \bullet y)) \\ & & \text{where} \quad cl.b.p.q = (t, d) \end{array} \tag{90}$$

We can now prove the missing case in the proof of the equivalence of the ripple carry and carry lookahead adders.

Proof

$$
\begin{aligned}
& rc.c.(p\triangleleft a).(q\triangleleft b) = cl.c.(p\triangleleft a).(q\triangleleft b) \\
\equiv \quad & \{\ rc\ (84) \text{ and } cl\ (90)\ \} \\
& \quad (s\triangleleft((a+b+d) \bmod 2), (a+b+d)\div 2) = (t\triangleleft(e\odot(x\bullet y)), e\star(x\bullet y)) \\
& \ \wedge\ (s,d) = rc.c.p.q\ \wedge\ (t,e) = cl.c.p.q \\
\equiv \quad & \{ \text{ by induction } (s,d) = (t,e)\ \} \\
& (s\triangleleft((a+b+d) \bmod 2), (a+b+d)\div 2) = (s\triangleleft(d\odot(x\bullet y)), d\star(x\bullet y)) \\
\equiv \quad & \{\ \text{equality on pairs}\ \} \\
& (a+b+d)\div 2 = d\star(x\bullet y)\ \wedge\ s\triangleleft((a+b+d) \bmod 2) = s\triangleleft(d\odot(x\bullet y)) \\
\equiv \quad & \{\ \text{Axiom (17)}\ \} \\
& (a+b+d)\div 2 = d\star(x\bullet y)\ \wedge\ s = s\ \wedge\ (a+b+d) \bmod 2 = d\odot(x\bullet y) \\
\equiv \quad & \{\ \text{(91) and (92) see below}\ \} \\
& \mathsf{true}
\end{aligned}
$$

End of Proof

In the last hint we used the following identities established in [Ada94]:

$$
\begin{aligned}
d\star(x\bullet y) &= (a+b+d)\div 2 && (91)\\
d\odot(x\bullet y) &= (a+b+d) \bmod 2 && (92)
\end{aligned}
$$

3 Related Work and Conclusion

This work is built on top of the work done on powerlists. Misra presented the theory and a number of examples [Mis94]; Adams derived and verified addition circuits [Ada94]; Kornerup presented a mapping strategy for powerlist onto hypercubic architectures [Kor95] and derived the *Odd-even sort* in the powerlist notation [Kor97].

The powerlist theory itself[Mis94] and many of Adam's results [Ada94] have been mechanically verified by Kapur and Subramaniam [KS95] using the inductive theorem prover *Rewrite Rule Laboratory*. Gamboa [Gam97] has verified many fundamental results about powerlists using the ACL2 theorem prover. His work focuses on verification of sorting algorithms.

Mou and Hudak [MH88] presented *Divacon*, a very general notation for describing divide-and-conquer algorithms in a functional manner. The Divacon notation is meant to capture the entire class of divide-and-conquer algorithms. Because of this generality it is difficult to prove the kinds of properties that have been done in the powerlist and ParList notation.

4 Conclusion

ParList appears to be an appropriate generalization of the powerlist notation. The powerlist examples presented above had straightforward extensions to the ParList algebra. The set of shared axioms makes it possible to reuse proofs of properties of the corresponding powerlist functions when proving the same properties of ParList functions.

5 Acknowledgments

The basic ideas behind the extensions presented in this paper are due to my advisor Jayadev Misra; he shared them with me and encouraged me to develop the ParList theory and to write this paper. Rajeev Joshi had many useful comments to drafts of this paper.

References

[Ada94] Will E. Adams. Verifying adder circuits using powerlists. Technical Report CS-TR-94-02, University of Texas at Austin, Department of Computer Sciences, March 1994.

[DS90] Edsger W. Dijkstra and Carel Sholten. *Predicate calculus and program semantics*. Springer Verlag, 1990.

[Gam97] Ruben A. Gamboa. Defthms about zip and tie: Reasoning about powerlists in ACL2. Technical Report CS-TR-97-02, The University of Texas at Austin, Department of Computer Sciences, January 23 1997.

[HJW+92] Paul Hudak, Simon L. Peyton Jones, Philip Wadler, et al. A report on the functional language Haskell. *SIGPLAN Notices*, 1992.

[Kor95] Jacob Kornerup. Mapping a functional notation for parallel programs onto hypercubes. *Information Processing Letters*, 53:153–158, 1995.

[Kor97] Jacob Kornerup. Odd-even sort in powerlists. *Information Processing Letters*, 61:15–24, 1997.

[KS95] D. Kapur and M. Subramaniam. Automated reasoning about parallel algorithms using powerlists. In Vangalur S. Alagar and M. Nivat, editors, *AMAST '95*, volume 936 of *LNCS*, pages 416–?? Springer-Verlag, 1995.

[LF80] Richard E. Ladner and Michael J. Fischer. Parallel prefix computation. *Journal of the ACM*, 27(4):831–838, October 1980.

[MH88] Zhijing G. Mou and Paul Hudak. An algebraic model for divide-and-conquer and its parallelism. *The Journal of Supercomputing*, 2(3):257–278, November 1988.

[Mis94] Jayadev Misra. Powerlist: A structure for parallel recursion. *ACM Transactions on Programming Languages and Systems*, 16(6):1737–1767, November 1994.

[Mis96] Jayadev Misra. Generalized powerlists. Unpublished manuscript, May 1996.

[MTH90] Robin Milner, Mads Tofte, and R. Harper. *The Definition of Standard ML*. MIT Press, 1990.

[Tur86] David Turner. An overview of Miranda. *ACM SIGPLAN Notices*, 21:156–166, 1986.

A Omitted Proofs

In this appendix we prove properties that were stated in the text above, but whose proofs are too long to be included in the main text of the paper.

Proof (31) $\wedge$ (32) $\wedge$ (33) $\Rightarrow$ (34)

Base case:

$\langle x\rangle \triangleleft a \;\oplus\; \langle y\rangle \triangleleft b$

$=$ { Axioms (27) and (26) }

$x \triangleright \langle a\rangle \;\oplus\; y \triangleright \langle b\rangle$

$=$ { Axiom (32) }

$(x \oplus y) \;\triangleright\; \langle a \oplus b\rangle$

$=$ { Axioms (26) and (27) }

$\langle x \oplus y\rangle \;\triangleleft\; (a \oplus b)$

$=$ { Axiom (31) }

$(\langle x\rangle \oplus \langle y\rangle) \;\triangleleft\; (a \oplus b)$

Inductive odd case:

$(c \triangleright p) \triangleleft a \;\oplus\; (d \triangleright q) \triangleleft b$

$=$ { Axiom (28) twice }

$c \triangleright (p \triangleleft a) \;\oplus\; d \triangleright (q \triangleleft b)$

$=$ { Axiom (32) }

$(c \oplus d) \;\triangleright\; (p \triangleleft a \oplus q \triangleleft b)$

$=$ { Induction (34) }

$(c \oplus d) \;\triangleright\; ((p \oplus q) \triangleleft (a \oplus b))$

$=$ { Axiom (28) }

$((c \oplus d) \triangleright (p \oplus q)) \;\triangleleft\; (a \oplus b)$

$=$ { Axiom (32) }

$(c \triangleright p \oplus d \triangleright q) \;\triangleleft\; (a \oplus b)$

Inductive even case:

$(p \bowtie q) \triangleleft a \;\oplus\; (u \bowtie v) \triangleleft b \;=\; (p \bowtie q \;\oplus\; u \bowtie v) \;\triangleleft\; (a \oplus b)$

$\Leftarrow$ { Predicate calculus, see (93) and (94) below }

$((p \bowtie q) \triangleleft a \;\oplus\; (u \bowtie v) \triangleleft b \;=\; (p \bowtie q \;\oplus\; u \bowtie v) \;\triangleleft\; (a \oplus b)) \;\wedge\; P \;\wedge\; U$

$\equiv$ { Leibnitz' Rule using (93) and (94) }

$(c \triangleright (r \bowtie s) \;\oplus\; d \triangleright (m \bowtie n) \;=\; (p \bowtie q \;\oplus\; u \bowtie v) \;\triangleleft\; (a \oplus b)) \;\wedge\; P \;\wedge\; U$

$\equiv$ { Axiom (32) }

$((c \oplus d) \;\triangleright\; ((r \bowtie s) \;\oplus\; (m \bowtie n)) \;=\; (p \bowtie q \;\oplus\; u \bowtie v) \;\triangleleft\; (a \oplus b)) \;\wedge\; P \;\wedge\; U$

$\equiv$ { Axiom (33) twice }

$((c \oplus d) \ \triangleright \ ((r \oplus m) \ \bowtie \ (s \oplus n)) \ = \ ((p \oplus u) \ \bowtie \ (q \oplus v)) \triangleleft \ (a \oplus b)) \ \wedge \ P \ \wedge \ U$

$\equiv$ { Axiom (29) }

$((c \oplus d) \triangleright (s \oplus n) \ = \ (p \oplus u) \triangleleft (a \oplus b)) \ \wedge \ (r \oplus m \ = \ q \oplus v) \ \wedge \ P \ \wedge \ U$

$\equiv$ { Axiom (32), inductive hypothesis (34) }

$(c \triangleright s \ \oplus \ d \triangleright n \ = \ p \triangleleft a \ \oplus \ u \triangleleft b) \ \wedge \ (r \oplus m \ = \ q \oplus v) \ \wedge \ P \ \wedge \ U$

$\equiv$ { Leibnitz' Rule using (93) and (94) }

$(p \triangleleft a \ \oplus \ u \triangleleft b \ = \ p \triangleleft a \ \oplus \ u \triangleleft b) \ \wedge \ (q \oplus v \ = \ q \oplus v) \ \wedge \ P \ \wedge \ U$

$\equiv$ { predicate calculus }

$P \ \wedge \ U$

$\Leftarrow$ { lemma 0, see below }

true

End of Proof

As mentioned in the hints above, $P \wedge U$ follows from lemma 0, where the existence of c, d, r, s, m, n satisfying $P \wedge U$ is established

$$P \equiv c \triangleright (r \bowtie s) = (p \bowtie q) \triangleleft a \ \wedge \ c \triangleright s = p \triangleleft a \ \wedge \ r = q \qquad (93)$$

$$U \equiv d \triangleright (m \bowtie n) = (u \bowtie v) \triangleleft b \ \wedge \ d \triangleright n = u \triangleleft b \ \wedge \ m = v \qquad (94)$$

Next, we prove that (35) follows from the axioms.

Proof (31) $\wedge$ (32) $\wedge$ (33) $\wedge$ (34) $\Rightarrow$ (35)

Base case:

$\langle a \rangle \mid \langle b \rangle \ \oplus \ \langle c \rangle \mid \langle d \rangle$

$=$ { Axiom (23) }

$\langle a \rangle \bowtie \langle b \rangle \ \oplus \ \langle c \rangle \bowtie \langle d \rangle$

$=$ { Axiom (33) }

$(\langle a \rangle \oplus \langle c \rangle) \ \bowtie \ (\langle b \rangle \oplus \langle d \rangle)$

$=$ { Axiom (31), twice }

$\langle a \oplus c \rangle \ \bowtie \ \langle b \oplus d \rangle$

$=$ { Axiom (23) }

$\langle a \oplus c \rangle \ \mid \ \langle b \oplus d \rangle$

$=$ { Axiom (31), twice }

$(\langle a \rangle \oplus \langle c \rangle) \ \mid \ (\langle b \rangle \oplus \langle d \rangle)$

Inductive even case:

$(p \bowtie q) \ \mid \ (u \bowtie v) \ \ \oplus \ \ (r \bowtie s) \ \mid \ (m \bowtie n)$

$=$ { Axiom (24) }

$(p \mid u) \ \bowtie \ (q \mid v) \ \ \oplus \ \ (r \mid m) \ \bowtie \ (s \mid n)$

$= \{$ Axiom (33) $\}$

$(p \mid u \;\oplus\; r \mid m) \;\bowtie\; (q \mid v \;\oplus\; s \mid n)$

$= \{$ Induction hypothesis (35), twice $\}$

$((p \oplus r) \;\mid\; (u \oplus m)) \;\bowtie\; ((q \oplus s) \;\mid\; (v \oplus n))$

$= \{$ Axiom (24) $\}$

$((p \oplus r) \;\bowtie\; (q \oplus s)) \;\mid\; ((u \oplus m) \;\bowtie\; (v \oplus n))$

$= \{$ Axiom (33) $\}$

$(p \bowtie q \;\oplus\; (r \bowtie s)) \;\mid\; ((u \bowtie v) \;\oplus (v \bowtie n))$

Inductive odd case:

$(a \triangleright (p \bowtie q) \;\mid\; (u \bowtie v) \triangleleft b) \;\oplus\; (c \triangleright (r \bowtie s) \;\mid\; (m \bowtie n) \triangleleft d)$

$= \{$ Axiom (25) twice $\}$

$(a \triangleright (q \mid v) \;\bowtie\; (p \mid u) \triangleleft b) \;\oplus\; (c \triangleright (s \mid n) \;\bowtie\; (r \mid m) \triangleleft d)$

$= \{$ Axiom (33) $\}$

$(a \triangleright (q \mid v) \oplus c \triangleright (s \mid n)) \;\bowtie\; ((p \mid u) \triangleleft b \oplus (r \mid m) \triangleleft d)$

$= \{$ Axioms (32) and (34) $\}$

$(a \oplus c) \triangleright ((q \mid v) \oplus (s \mid n)) \;\bowtie\; ((p \mid u) \oplus (r \mid m)) \triangleleft (b \oplus d)$

$= \{$ Inductive hypothesis (35) twice $\}$

$(a \oplus c) \triangleright ((q \oplus s) \;\mid\; (v \oplus n)) \;\bowtie\; ((p \oplus r) \;\mid\; (u \oplus m)) \triangleleft (b \oplus d)$

$= \{$ Axiom (25) $\}$

$(a \oplus c) \triangleright ((p \oplus r) \;\bowtie\; (q \oplus s)) \;\mid\; ((u \oplus m) \;\bowtie\; (v \oplus n)) \triangleleft (b \oplus d)$

$= \{$ Axiom (33) twice $\}$

$(a \oplus c) \triangleright ((p \bowtie q) \oplus (r \bowtie s)) \;\mid\; ((u \bowtie v) \oplus (m \bowtie n)) \triangleleft (b \oplus d)$

$= \{$ Axioms (32) and (34) $\}$

$(a \triangleright (p \bowtie q) \;\oplus\; c \triangleright (r \bowtie s)) \;\mid\; ((u \bowtie v) \triangleleft b \;\oplus\; (m \bowtie n) \triangleleft d)$

End of Proof