Denotationally Correct Computer Arithmetic

Atticus Kuhn

2023-07-11 13:29 GMT-8

Preliminaries

About Denotational Design

Definition

Denotational Design is a type of thinking that prioirtizes thinking about the meaning and creating precise and elegant specifications using tools from abstract algebra and category theory.

About Denotational Design

Definition

Denotational Design is a type of thinking that prioirtizes thinking about the meaning and creating precise and elegant specifications using tools from abstract algebra and category theory.

Notation

In denotational design, the function $[\![\cdot]\!]$ is used to take any object to its meaning.

Motivations

Why are we interested?

Motivations

Why are we interested? Software and hardware engineering are built upon a tower of hodge-podge and ad-hoc foundations; desirable properties such as correctness are either not checked or not even specified

Motivations

Why are we interested? Software and hardware engineering are built upon a tower of hodge-podge and ad-hoc foundations; desirable properties such as correctness are either not checked or not even specified

"That is not only not right; it is not even wrong" - Wolgang Pauli

Motivations

Why are we interested? Software and hardware engineering are built upon a tower of hodge-podge and ad-hoc foundations; desirable properties such as correctness are either not checked or not even specified

"That is not only not right; it is not even wrong" - Wolgang Pauli

Using mathematics can return elegance to computation.

Philosophy

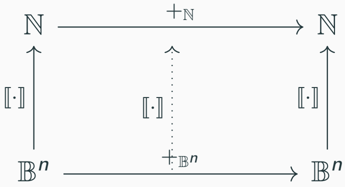

Figure 1: A Diagram Showing the Relationship Between Representations and Meanings

We care about problems in **mathematics**, but our computations take place over **physics** (electrons, circuits).

Philosophy

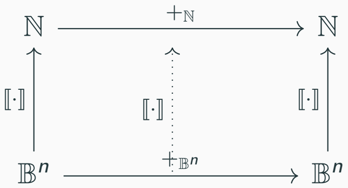

Figure 1: A Diagram Showing the Relationship Between Representations and Meanings

We care about problems in **mathematics**, but our computations take place over **physics** (electrons, circuits). The denotation function $[\![\cdot]\!]$ gives us the meaning of any representation of electrons or bits.

Philosophy

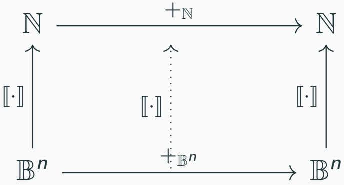

Figure 1: A Diagram Showing the Relationship Between Representations and Meanings

We care about problems in **mathematics**, but our computations take place over **physics** (electrons, circuits). The denotation function $[\![\cdot]\!]$ gives us the meaning of any representation of electrons or bits.

Question

What does it mean for a function over representations to be correct?

Philosophy

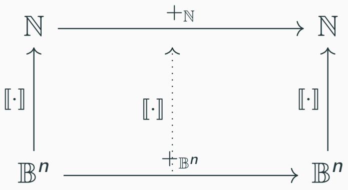

Figure 1: A Diagram Showing the Relationship Between Representations and Meanings

We care about problems in **mathematics**, but our computations take place over **physics** (electrons, circuits). The denotation function $[\![\cdot]\!]$ gives us the meaning of any representation of electrons or bits.

Question

What does it mean for a function over representations to be correct?

Theorem

*We say a function over representations is **correct** if Figure 6 commutes, i.e. if*

$$[\![A +_{\mathbb{B}^n} B]\!] = [\![A]\!] +_N [\![B]\!].$$

Why Computer Arithmetic

Reasons why I chose computer arithmetic

Why Computer Arithmetic

Reasons why I chose computer arithmetic

1. It is elementary; Most people have some exposure

Why Computer Arithmetic

Reasons why I chose computer arithmetic

1. It is elementary; Most people have some exposure
2. It is a good way to show denotational design

Why Computer Arithmetic

Reasons why I chose computer arithmetic

1. It is elementary; Most people have some exposure
2. It is a good way to show denotational design

The focus is not on any specific circuit component, but on specifications as to why it is **correct**

Simplifications

For the purposes of this talk, I simplified from my paper:

Simplifications

For the purposes of this talk, I simplified from my paper:

1. In my paper, I used the computer theorem prover language **Agda** to prove my propositions correct. You do not need to know programming for this talk.

Simplifications

For the purposes of this talk, I simplified from my paper:

1. In my paper, I used the computer theorem prover language **Agda** to prove my propositions correct. You do not need to know programming for this talk.
2. In my paper, I talked about category theory, but for the sake of this talk, just imagine everything is occuring in the category of functions.

Simplifications

For the purposes of this talk, I simplified from my paper:

1. In my paper, I used the computer theorem prover language **Agda** to prove my propositions correct. You do not need to know programming for this talk.
2. In my paper, I talked about category theory, but for the sake of this talk, just imagine everything is occuring in the category of functions.
3. In my paper, I used little endian encoding, but in this talk, I will use big endian encoding because most people are probably more familiar with big endian.

Binary Basics

We will represent binary numbers as lists of bits, where the least significant bit is on the right (big endian encoding).

As an additional preliminary, we expect the reader to be familiar with common bitwise operations, including $\cdot \oplus \cdot$, $\cdot \vee \cdot$, and $\cdot \wedge \cdot$ (see table 1).

Notation

We use N to denote our number system, we use $\mathbb{B}$ to represent a bit, and we use $\mathbb{B}^n$ to denote an n-bit representation.

$\cdot \oplus \cdot$	$\cdot \vee \cdot$	$\cdot \wedge \cdot$
$0 \oplus 0 = 0$	$0 \vee 0 = 0$	$0 \wedge 0 = 0$
$0 \oplus 1 = 1$	$0 \vee 1 = 1$	$0 \wedge 1 = 0$
$1 \oplus 0 = 1$	$1 \vee 0 = 1$	$1 \wedge 0 = 0$
$1 \oplus 1 = 0$	$1 \vee 0 = 1$	$1 \wedge 1 = 1$

Addition

Converting $\mathbb{B}^n$ to N

Anything we do is only correct modulo our meaning function $[\![\cdot]\!]$.

$$[\![b_{n-1} \cdots b_1 b_0]\!] = [\![b_0]\!] + 2[\![b_{n-1} \cdots b_1]\!] \tag{1}$$

Converting $\mathbb{B}^n$ to N

Anything we do is only correct modulo our meaning function $[\![\cdot]\!]$.

$$[\![b_{n-1} \cdots b_1 b_0]\!] = [\![b_0]\!] + 2[\![b_{n-1} \cdots b_1]\!] \tag{1}$$

$$= [\![b_0]\!] + 2[\![b_1]\!] + 4[\![b_2]\!] + \cdots + 2^{n-1}[\![b_{n-1}]\!] \tag{2}$$

Converting $\mathbb{B}^n$ to N

Anything we do is only correct modulo our meaning function $[\![\cdot]\!]$.

$$[\![b_{n-1} \cdots b_1 b_0]\!] = [\![b_0]\!] + 2[\![b_{n-1} \cdots b_1]\!] \quad (1)$$

$$= [\![b_0]\!] + 2[\![b_1]\!] + 4[\![b_2]\!] + \cdots + 2^{n-1}[\![b_{n-1}]\!] \quad (2)$$

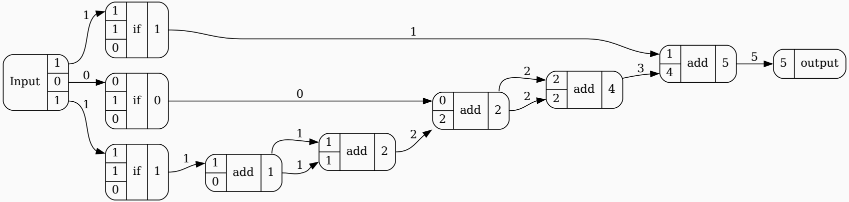

Figure 2: An Example showing $[\![101]\!] = 5$

Half-Adder Specification

A half adder is a function that adds two bits (possibly with a carry).

$$\cdot +_H \cdot : \mathbb{B} \times \mathbb{B} \to \mathbb{B}^2 \tag{3}$$

We need a correctness specification for a half-adder.

Half-Adder Specification

A half adder is a function that adds two bits (possibly with a carry).

$$\cdot +_H \cdot : \mathbb{B} \times \mathbb{B} \to \mathbb{B}^2 \tag{3}$$

We need a correctness specification for a half-adder.

$$\forall A, B \in \mathbb{B}^1 \qquad [\![A +_H B]\!] = [\![A]\!] +_N [\![B]\!] \tag{4}$$

Half-Adder Example

$$\forall A, B \in \mathbb{B}^1 \qquad A +_H B = (A \wedge B, A \oplus B) \tag{5}$$

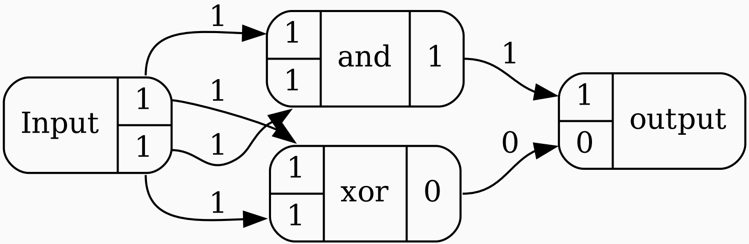

Figure 3: An Example Showing $1 +_H 1 = 10$

Full-Adder Specification

A full-adder adds 3 bits with possibly a carry.

$$+_F(\cdot,\cdot,\cdot) : \mathbb{B} \times \mathbb{B} \times \mathbb{B} \to \mathbb{B}^2 \tag{6}$$

$$\forall A, B, C \in \mathbb{B}^1 \qquad [\![+_F(A, B, C)]\!] = [\![A]\!] + [\![B]\!] + [\![C]\!] \tag{7}$$

Full-Adder Example

$$\forall A, B, C \in \mathbb{B}^1 \qquad +_F(A, B, C) = (A \wedge B \vee (A \oplus B) \wedge C, A \oplus B \oplus C) \tag{8}$$

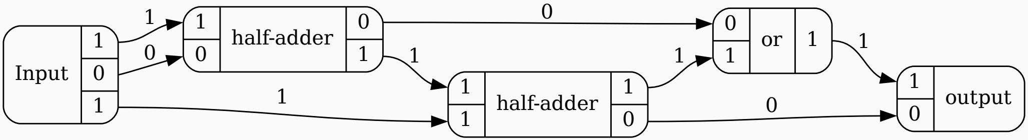

Figure 4: An Example Showing $+_F(1, 0, 1) = 10$

Ripple Adder Specification

$$\cdot +_{\mathbb{B}^n}^{\cdot} \cdot : \mathbb{B}^1 \times \mathbb{B}^n \times \mathbb{B}^n \to \mathbb{B}^{n+1} \tag{9}$$

$$\forall A, B \in \mathbb{B}^n \quad \forall C \in \mathbb{B}^1 \qquad [\![A +_{\mathbb{B}^n}^{C} B]\!] = [\![A]\!] +_N [\![B]\!] +_N [\![C]\!] \tag{10}$$

Ripple Adder Specification

[1]	[1]1	[1]0	1
+	1	1	1
1	1	0	0

Table 2: Grade-School Addition

$$a_{n-1} \cdots a_2 a_1 a_0 +_{\mathbb{B}^n}^{c_0} b_{n-1} \cdots b_2 b_1 b_0 = (a_{n-1} \cdots a_2 a_1 +_{\mathbb{B}^{n-1}}^{c_1} b_{n-1} \cdots b_2 b_1) r_0 \quad \text{where} \quad c_1 r_0 = +_F(a_0, b_0, c_0) \tag{11}$$

Ripple Adder Picture

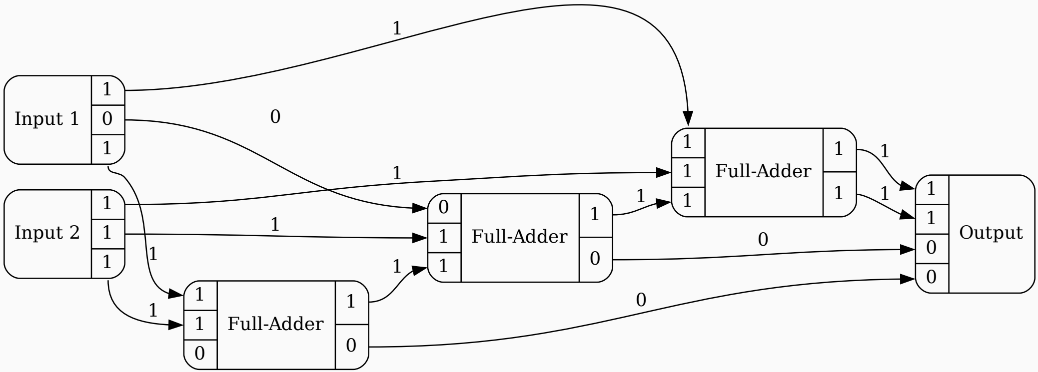

Figure 5: An Example Showin $101 +^{0}_{\mathbb{B}^3} 111 = 1100$

Ripple Adder Proof

Proof.

Induct on n. If $n = 1$, we just have a full-adder. Otherwise, let $n = n + 1$.

$$[\![a_n \cdots a_2 a_1 a_0 +^{c_0}_{\mathbb{B}^{n+1}} b_n \cdots b_2 b_1 b_0]\!] \tag{12}$$

Ripple Adder Proof

Proof.

Induct on n. If $n = 1$, we just have a full-adder. Otherwise, let $n = n + 1$.

$$\llbracket a_n \cdots a_2 a_1 a_0 +^{c_0}_{\mathbb{B}^{n+1}} b_n \cdots b_2 b_1 b_0 \rrbracket \tag{12}$$

$$= \llbracket (a_n \cdots a_2 a_1 +^{c_1}_{\mathbb{B}^n} b_n \cdots b_2 b_1) r_0 \rrbracket \tag{13}$$

Ripple Adder Proof

Proof.

Induct on n. If $n = 1$, we just have a full-adder. Otherwise, let $n = n + 1$.

$$\llbracket a_n \cdots a_2 a_1 a_0 +^{c_0}_{\mathbb{B}^{n+1}} b_n \cdots b_2 b_1 b_0 \rrbracket \tag{12}$$

$$= \llbracket (a_n \cdots a_2 a_1 +^{c_1}_{\mathbb{B}^n} b_n \cdots b_2 b_1) r_0 \rrbracket \tag{13}$$

$$= 2\llbracket a_n \cdots a_2 a_1 +^{c_1}_{\mathbb{B}^n} b_n \cdots b_2 b_1 \rrbracket + \llbracket r_0 \rrbracket \tag{14}$$

Ripple Adder Proof

Proof.

Induct on n. If $n = 1$, we just have a full-adder. Otherwise, let $n = n + 1$.

$$\llbracket a_n \cdots a_2 a_1 a_0 +^{c_0}_{\mathbb{B}^{n+1}} b_n \cdots b_2 b_1 b_0 \rrbracket \tag{12}$$

$$= \llbracket (a_n \cdots a_2 a_1 +^{c_1}_{\mathbb{B}^n} b_n \cdots b_2 b_1) r_0 \rrbracket \tag{13}$$

$$= 2\llbracket a_n \cdots a_2 a_1 +^{c_1}_{\mathbb{B}^n} b_n \cdots b_2 b_1 \rrbracket + \llbracket r_0 \rrbracket \tag{14}$$

$$= 2(\llbracket a_n \cdots a_2 a_1 \rrbracket + \llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket c_1 \rrbracket) + \llbracket r_0 \rrbracket \tag{15}$$

Ripple Adder Proof

Proof.

Induct on n. If $n = 1$, we just have a full-adder. Otherwise, let $n = n + 1$.

$$\llbracket a_n \cdots a_2 a_1 a_0 +^{c_0}_{\mathbb{B}^{n+1}} b_n \cdots b_2 b_1 b_0 \rrbracket \tag{12}$$

$$= \llbracket (a_n \cdots a_2 a_1 +^{c_1}_{\mathbb{B}^n} b_n \cdots b_2 b_1) r_0 \rrbracket \tag{13}$$

$$= 2\llbracket a_n \cdots a_2 a_1 +^{c_1}_{\mathbb{B}^n} b_n \cdots b_2 b_1 \rrbracket + \llbracket r_0 \rrbracket \tag{14}$$

$$= 2(\llbracket a_n \cdots a_2 a_1 \rrbracket + \llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket c_1 \rrbracket) + \llbracket r_0 \rrbracket \tag{15}$$

$$= 2\llbracket a_n \cdots a_2 a_1 \rrbracket + 2\llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + 2\llbracket c_1 \rrbracket + \llbracket r_0 \rrbracket \tag{16}$$

Ripple Adder Proof

Proof.

Induct on n. If $n = 1$, we just have a full-adder. Otherwise, let $n = n + 1$.

$$\llbracket a_n \cdots a_2 a_1 a_0 +^{c_0}_{\mathbb{B}^{n+1}} b_n \cdots b_2 b_1 b_0 \rrbracket \tag{12}$$

$$= \llbracket (a_n \cdots a_2 a_1 +^{c_1}_{\mathbb{B}^n} b_n \cdots b_2 b_1) r_0 \rrbracket \tag{13}$$

$$= 2\llbracket a_n \cdots a_2 a_1 +^{c_1}_{\mathbb{B}^n} b_n \cdots b_2 b_1 \rrbracket + \llbracket r_0 \rrbracket \tag{14}$$

$$= 2(\llbracket a_n \cdots a_2 a_1 \rrbracket + \llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket c_1 \rrbracket) + \llbracket r_0 \rrbracket \tag{15}$$

$$= 2\llbracket a_n \cdots a_2 a_1 \rrbracket + 2\llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + 2\llbracket c_1 \rrbracket + \llbracket r_0 \rrbracket \tag{16}$$

$$= 2\llbracket a_n \cdots a_2 a_1 \rrbracket + 2\llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket c_1 r_0 \rrbracket \tag{17}$$

Ripple Adder Proof

Proof.

Induct on n. If $n = 1$, we just have a full-adder. Otherwise, let $n = n + 1$.

$$\llbracket a_n \cdots a_2 a_1 a_0 +^{c_0}_{\mathbb{B}^{n+1}} b_n \cdots b_2 b_1 b_0 \rrbracket \tag{12}$$
$$= \llbracket (a_n \cdots a_2 a_1 +^{c_1}_{\mathbb{B}^n} b_n \cdots b_2 b_1) r_0 \rrbracket \tag{13}$$
$$= 2\llbracket a_n \cdots a_2 a_1 +^{c_1}_{\mathbb{B}^n} b_n \cdots b_2 b_1 \rrbracket + \llbracket r_0 \rrbracket \tag{14}$$
$$= 2(\llbracket a_n \cdots a_2 a_1 \rrbracket + \llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket c_1 \rrbracket) + \llbracket r_0 \rrbracket \tag{15}$$
$$= 2\llbracket a_n \cdots a_2 a_1 \rrbracket + 2\llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + 2\llbracket c_1 \rrbracket + \llbracket r_0 \rrbracket \tag{16}$$
$$= 2\llbracket a_n \cdots a_2 a_1 \rrbracket + 2\llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket c_1 r_0 \rrbracket \tag{17}$$
$$= 2\llbracket a_n \cdots a_2 a_1 \rrbracket + 2\llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket +_F(a_0, b_0, c_0) \rrbracket \tag{18}$$

Ripple Adder Proof

Proof.

Induct on n. If $n = 1$, we just have a full-adder. Otherwise, let $n = n + 1$.

$$\llbracket a_n \cdots a_2 a_1 a_0 +^{c_0}_{\mathbb{B}^{n+1}} b_n \cdots b_2 b_1 b_0 \rrbracket \tag{12}$$
$$= \llbracket (a_n \cdots a_2 a_1 +^{c_1}_{\mathbb{B}^n} b_n \cdots b_2 b_1) r_0 \rrbracket \tag{13}$$
$$= 2\llbracket a_n \cdots a_2 a_1 +^{c_1}_{\mathbb{B}^n} b_n \cdots b_2 b_1 \rrbracket + \llbracket r_0 \rrbracket \tag{14}$$
$$= 2(\llbracket a_n \cdots a_2 a_1 \rrbracket + \llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket c_1 \rrbracket) + \llbracket r_0 \rrbracket \tag{15}$$
$$= 2\llbracket a_n \cdots a_2 a_1 \rrbracket + 2\llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + 2\llbracket c_1 \rrbracket + \llbracket r_0 \rrbracket \tag{16}$$
$$= 2\llbracket a_n \cdots a_2 a_1 \rrbracket + 2\llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket c_1 r_0 \rrbracket \tag{17}$$
$$= 2\llbracket a_n \cdots a_2 a_1 \rrbracket + 2\llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket +_F(a_0, b_0, c_0) \rrbracket \tag{18}$$
$$= 2\llbracket a_n \cdots a_2 a_1 \rrbracket + 2\llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket a_0 \rrbracket + \llbracket b_0 \rrbracket + \llbracket c_0 \rrbracket \tag{19}$$

Ripple Adder Proof

Proof.

Induct on n. If $n = 1$, we just have a full-adder. Otherwise, let $n = n + 1$.

$$
\begin{aligned}
&[\![a_n \cdots a_2 a_1 a_0 +^{c_0}_{\mathbb{B}^{n+1}} b_n \cdots b_2 b_1 b_0]\!] && (12)\\
&= [\![(a_n \cdots a_2 a_1 +^{c_1}_{\mathbb{B}^n} b_n \cdots b_2 b_1) r_0]\!] && (13)\\
&= 2[\![a_n \cdots a_2 a_1 +^{c_1}_{\mathbb{B}^n} b_n \cdots b_2 b_1]\!] + [\![r_0]\!] && (14)\\
&= 2([\![a_n \cdots a_2 a_1]\!] + [\![b_{n-1} \cdots b_2 b_1]\!] + [\![c_1]\!]) + [\![r_0]\!] && (15)\\
&= 2[\![a_n \cdots a_2 a_1]\!] + 2[\![b_{n-1} \cdots b_2 b_1]\!] + 2[\![c_1]\!] + [\![r_0]\!] && (16)\\
&= 2[\![a_n \cdots a_2 a_1]\!] + 2[\![b_{n-1} \cdots b_2 b_1]\!] + [\![c_1 r_0]\!] && (17)\\
&= 2[\![a_n \cdots a_2 a_1]\!] + 2[\![b_{n-1} \cdots b_2 b_1]\!] + [\![+_F(a_0, b_0, c_0)]\!] && (18)\\
&= 2[\![a_n \cdots a_2 a_1]\!] + 2[\![b_{n-1} \cdots b_2 b_1]\!] + [\![a_0]\!] + [\![b_0]\!] + [\![c_0]\!] && (19)\\
&= 2[\![a_n \cdots a_2 a_1]\!] + [\![a_0]\!] + 2[\![b_{n-1} \cdots b_2 b_1]\!] + [\![b_0]\!] + [\![c_0]\!] && (20)
\end{aligned}
$$

Ripple Adder Proof

Proof.

Induct on n. If $n = 1$, we just have a full-adder. Otherwise, let $n = n + 1$.

$$\llbracket a_n \cdots a_2 a_1 a_0 +_{\mathbb{B}^{n+1}}^{c_0} b_n \cdots b_2 b_1 b_0 \rrbracket \tag{12}$$

$$= \llbracket (a_n \cdots a_2 a_1 +_{\mathbb{B}^n}^{c_1} b_n \cdots b_2 b_1) r_0 \rrbracket \tag{13}$$

$$= 2\llbracket a_n \cdots a_2 a_1 +_{\mathbb{B}^n}^{c_1} b_n \cdots b_2 b_1 \rrbracket + \llbracket r_0 \rrbracket \tag{14}$$

$$= 2(\llbracket a_n \cdots a_2 a_1 \rrbracket + \llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket c_1 \rrbracket) + \llbracket r_0 \rrbracket \tag{15}$$

$$= 2\llbracket a_n \cdots a_2 a_1 \rrbracket + 2\llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + 2\llbracket c_1 \rrbracket + \llbracket r_0 \rrbracket \tag{16}$$

$$= 2\llbracket a_n \cdots a_2 a_1 \rrbracket + 2\llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket c_1 r_0 \rrbracket \tag{17}$$

$$= 2\llbracket a_n \cdots a_2 a_1 \rrbracket + 2\llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket +_F(a_0, b_0, c_0) \rrbracket \tag{18}$$

$$= 2\llbracket a_n \cdots a_2 a_1 \rrbracket + 2\llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket a_0 \rrbracket + \llbracket b_0 \rrbracket + \llbracket c_0 \rrbracket \tag{19}$$

$$= 2\llbracket a_n \cdots a_2 a_1 \rrbracket + \llbracket a_0 \rrbracket + 2\llbracket b_{n-1} \cdots b_2 b_1 \rrbracket + \llbracket b_0 \rrbracket + \llbracket c_0 \rrbracket \tag{20}$$

$$= \llbracket a_n \cdots a_2 a_1 a_0 \rrbracket + \llbracket b_n \cdots b_2 b_1 b_0 \rrbracket + \llbracket c_0 \rrbracket \tag{21}$$

Multiplication

Multiplication Specification

Before we implement multiplication, we first need a specification.

Multiplication Specification

Before we implement multiplication, we first need a specification. The specification for multiplication is very similar to that of addition.

Multiplication Specification

Before we implement multiplication, we first need a specification. The specification for multiplication is very similar to that of addition.

$$\forall A \in \mathbb{B}^m \quad B \in \mathbb{B}^n \qquad [\![A \times_{\mathbb{B}^{m,n}} B]\!] = [\![A]\!] \times_N [\![B]\!]$$

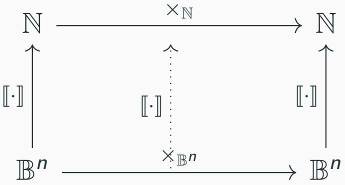

Figure 6: Specification of Multiplication

Bit Multiplier

Our first building block is multiplication by a single bit.

Bit Multiplier

Our first building block is multiplication by a single bit. The correctness specification is

$$\forall b \in \mathbb{B}^1 \quad A \in \mathbb{B}^n \qquad [\![b \times_{\mathbb{B}^{1,n}} A]\!] = [\![b]\!] \times_N [\![A]\!]$$

Bit Multiplier

Our first building block is multiplication by a single bit. The correctness specification is

$$\forall b \in \mathbb{B}^1 \quad A \in \mathbb{B}^n \qquad [\![b \times_{\mathbb{B}^{1,n}} A]\!] = [\![b]\!] \times_N [\![A]\!]$$

One implementation is

$$\cdot \times_{\mathbb{B}^{1,m}} \cdot : \mathbb{B}^1 \times \mathbb{B}^n \to \mathbb{B}^n$$
$$a \times_{\mathbb{B}^{1,m}} B = \mathit{if}(a, B, 0)$$

Shift Right

We also need the ability to double a number, which we will call $\cdot \ll 1$.

Shift Right

We also need the ability to double a number, which we will call $\cdot \ll 1$. We will have specification

$$\forall B \in \mathbb{B}^n \qquad [\![B \ll 1]\!] = 2[\![B]\!]$$

Shift Right

We also need the ability to double a number, which we will call $\cdot \ll 1$. We will have specification

$$\forall B \in \mathbb{B}^n \qquad [\![B \ll 1]\!] = 2[\![B]\!]$$

We can implement the specification by just appending a 0 to the end.

$$b_{n-1} \cdots b_1 b_0 \ll 1 = b_{n-1} \cdots b_1 b_0 0$$

Shift and Add

				1	0	1	1
			$\times$	1	1	1	0
				0	0	0	0
			1	0	1	1	
		1	0	1	1		
$+$	1	0	1	1			
1	0	0	1	1	0	1	0

Table 3: An Example shift-and-add multiplication

$$b_{n-1} \dots b_1 b_0 \times_{\mathbb{B}^{n,m}} A = b_0 \times_{\mathbb{B}^{1,m}} A + (b_{n-1}, \dots, b_1 \times_{\mathbb{B}^{n-1,m}} A) \ll 1 \tag{22}$$

Future Work

1. Carry-Lookahead Adders

Future Work

1. Carry-Lookahead Adders
2. Binary Subtraction
3. Binary Division

Key Takeaways

Key Takeaways

1. We can formally prove the correctness of software and hardware components.

Key Takeaways

1. We can formally prove the correctness of software and hardware components.
2. Homomoprhisms and category theory can give us more elegant and precise specifications.

Questions?

Ask me any questions. Or if you have any questions later

1. Email me at atticusmkuhn@gmail.com
2. On Discord at Euler#2074