A Formally-Proved Algorithm to Compute the Correct Average of Decimal Floating-Point Numbers

Sylvie Boldo, Florian Faissole, and Vincent Tourneur\textsuperscript{1}

\textsuperscript{1}Thanks to the IEEE for the student travel award.

ARITH-25 - June 26th
FP arithmetic: IEEE-754.

IEEE-754 2008 revision adds radix-10 formats (decimal32, decimal64).

Many algorithms designed for radix-2 FP numbers are not valid anymore.

Goal: Adapt an existing algorithm from radix-2 FP numbers literature to radix-10.
Average of two FP numbers

Compute the **correct rounding** of the average of two FP numbers:

$$\circ \left( \frac{a + b}{2} \right)$$ with $$\circ$$ a rounding to nearest

with as few tests as possible.
Outline

1. Radix-2 Average Algorithms
2. Unsuccessful Radix-10 Average Algorithm
3. Radix-10 Average Algorithm
4. Formal Proof with Coq and Flocq
5. Conclusion
1. Radix-2 Average Algorithms

2. Unsuccessful Radix-10 Average Algorithm

3. Radix-10 Average Algorithm

4. Formal Proof with Coq and Flocq

5. Conclusion
FP Average in Radix 2

- Studied by Sterbenz (1974):
  - \((a \oplus b) \oslash 2\): accurate, but may overflow when \(a\) and \(b\) share the same sign.
  - \((a \oslash 2) \oplus (b \oslash 2)\): accurate, except when underflow.
  - \(a \oplus ((b \ominus a) \oslash 2)\): less accurate, but does not overflow. when \(a\) and \(b\) share the same sign.

- A corresponding algorithm has been proved by Boldo to guarantee accuracy. This is a long program, since a full sign study is required to choose the correct formula.
Studied by Sterbenz (1974):

- \((a \oplus b) \oslash 2\): accurate, but may overflow when \(a\) and \(b\) share the same sign.

- \((a \oslash 2) \oplus (b \oslash 2)\): accurate, except when underflow.

- \(a \oplus ((b \ominus a) \oslash 2)\): less accurate, but does not overflow. when \(a\) and \(b\) share the same sign.

A corresponding algorithm has been proved by Boldo to guarantee accuracy. This is a long program, since a full sign study is required to choose the correct formula.
A simpler algorithm that computes the correctly-rounded average is formally proved by Boldo (2015). Using radix-2 binary64 FP numbers:

```c
double average(double C, double x, double y) {
    if (C <= abs(x))
        return x/2+y/2;
    else
        return (x+y)/2;
}
```

C is a constant that can be chosen between $2^{-967}$ and $2^{970}$. 
In radix 2, dividing by 2 is exact (except when underflow).

In radix 10, there are 2 different cases:

- If the mantissa is even or small: the result is exact.
- Otherwise, the mantissa is odd and the result is a midpoint.
Outline

1. Radix-2 Average Algorithms
2. Unsuccessful Radix-10 Average Algorithm
3. Radix-10 Average Algorithm
4. Formal Proof with Coq and Flocq
5. Conclusion
In this section, we use the following FP format:

- Radix: 10.
- Mantissa size: 4 digits.
- Unbounded exponent range.
- Rounding to nearest, tie-breaking to even.
Algorithms based on \((a + b)/2\)

**Algorithm:** \((a \oplus b) \odot 2\)

Counter-example of correct rounding:

\[
a = 3001 \times 10^{10}, \ b = 1000 \times 10^0
\]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>3001</td>
</tr>
<tr>
<td>(b)</td>
<td>1000</td>
</tr>
<tr>
<td>(a + b)</td>
<td>30010000001</td>
</tr>
<tr>
<td>(a \oplus b)</td>
<td>3001</td>
</tr>
<tr>
<td>((a \oplus b)/2)</td>
<td>1500</td>
</tr>
<tr>
<td>((a \oplus b) \odot 2)</td>
<td>1500</td>
</tr>
<tr>
<td>((a + b)/2)</td>
<td>150050000005</td>
</tr>
<tr>
<td>(\circ((a + b)/2))</td>
<td>1501</td>
</tr>
</tbody>
</table>

\(a/2\) is a midpoint, but \(b\) is positive, so the rounding should have been towards \(+\infty\).
Algorithms based on \((a + b)/2\)

Algorithm: \((a \oplus b) \odot 2\)

Counter-example of correct rounding:

\[
a = 3001 \times 10^{10}, \quad b = 1000 \times 10^0
\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>3001</td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>(a + b)</td>
<td>30010000001</td>
<td></td>
</tr>
<tr>
<td>(a \oplus b)</td>
<td>3001</td>
<td></td>
</tr>
<tr>
<td>((a + b)/2)</td>
<td>15005</td>
<td></td>
</tr>
<tr>
<td>((a \oplus b) \odot 2)</td>
<td>1500</td>
<td></td>
</tr>
<tr>
<td>((a + b)/2) d</td>
<td>150050000005</td>
<td></td>
</tr>
<tr>
<td>((a + b)/2)</td>
<td>1501</td>
<td></td>
</tr>
</tbody>
</table>

\(a/2\) is a midpoint, but \(b\) is positive, so the rounding should have been towards \(+\infty\).
Algorithms based on \((a + b)/2\)

Algorithm: \((a \oplus b) \odot 2\)

Counter-example of correct rounding:

\[
a = 3001 \times 10^{10}, \quad b = 1000 \times 10^{0}
\]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>3001</td>
</tr>
<tr>
<td>(b)</td>
<td>1000</td>
</tr>
<tr>
<td>(a + b)</td>
<td>30010000001</td>
</tr>
<tr>
<td>(a \oplus b)</td>
<td>3001</td>
</tr>
<tr>
<td>((a \oplus b)/2)</td>
<td>15005</td>
</tr>
<tr>
<td>((a \oplus b) \odot 2)</td>
<td>1500</td>
</tr>
<tr>
<td>((a + b)/2)</td>
<td>150050000005</td>
</tr>
<tr>
<td>(\odot((a + b)/2))</td>
<td>1501</td>
</tr>
</tbody>
</table>

\(a/2\) is a midpoint, but \(b\) is positive, so the rounding should have been towards \(+\infty\).
Algorithms based on \((a/2) + (b/2)\)

Algorithm: \((a \otimes 2) \oplus (b \otimes 2)\)

Counter-example of correct rounding: (same)

\[ a = 3001 \times 10^{10}, \quad b = 1000 \times 10^{0} \]

<table>
<thead>
<tr>
<th>(a)</th>
<th>(3001)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b)</td>
<td>(1000)</td>
</tr>
<tr>
<td>(a/2)</td>
<td>(1500)</td>
</tr>
<tr>
<td>(a \otimes 2)</td>
<td>(1500)</td>
</tr>
<tr>
<td>(b/2)</td>
<td>(5000)</td>
</tr>
<tr>
<td>((a \otimes 2) + (b \otimes 2))</td>
<td>(15000000005)</td>
</tr>
<tr>
<td>((a \otimes 2) \oplus (b \otimes 2))</td>
<td>(1500)</td>
</tr>
<tr>
<td>((a + b)/2)</td>
<td>(150050000005)</td>
</tr>
<tr>
<td>(\odot((a + b)/2))</td>
<td>(1501)</td>
</tr>
</tbody>
</table>

Same issue, \(a/2\) is a midpoint, and is rounded before taking into account the value of \(b\).
Algorithms based on \( (a/2) + (b/2) \)

Algorithm: \((a \odot 2) \oplus (b \odot 2)\)

Counter-example of correct rounding: (same)

\[
a = 3001 \times 10^{10}, \quad b = 1000 \times 10^0
\]

\[
\begin{array}{l|l}
\hline
a & 3001 \\
\hline
b & 1000 \\
\hline
a/2 & 1500.5 \\
\hline
a \odot 2 & 1500 \\
\hline
b/2 & 5000 \\
\hline
(a \odot 2) + (b \odot 2) & 1500.00000005 \\
\hline
(a \odot 2) \oplus (b \odot 2) & 1500 \\
\hline
\end{array}
\]

Same issue, \( a/2 \) is a midpoint, and is rounded before taking into account the value of \( b \).
Algorithms based on \((a/2) + (b/2)\)

Algorithm: \((a \odot 2) \oplus (b \odot 2)\)

Counter-example of correct rounding: (same)

\[
a = 3001 \times 10^{10}, \quad b = 1000 \times 10^0
\]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3001</td>
</tr>
<tr>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>1500</td>
<td>5</td>
</tr>
<tr>
<td>1500</td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td></td>
</tr>
<tr>
<td>1500</td>
<td>000000005</td>
</tr>
<tr>
<td>1500</td>
<td></td>
</tr>
<tr>
<td>1500</td>
<td>500000005</td>
</tr>
<tr>
<td>1501</td>
<td></td>
</tr>
</tbody>
</table>

Same issue, \(a/2\) is a midpoint, and is rounded before taking into account the value of \(b\).
Algorithms based on \((a/2) + (b/2)\), using FMA

Algorithm: \(\circ(a \times 0.5 + (b \odot 2))\)

There is one rounding less thanks to the FMA operator.

Counter-example of correct rounding:

\[a = 2001 \times 10^{10}, b = 2001 \times 10^{8}\]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2001</td>
</tr>
<tr>
<td>b</td>
<td>2001</td>
</tr>
<tr>
<td>b/2</td>
<td>1000 5</td>
</tr>
<tr>
<td>b \odot 2</td>
<td>1000 0</td>
</tr>
<tr>
<td>a × 0.5</td>
<td>1000 5</td>
</tr>
<tr>
<td>a × 0.5 + (b \odot 2)</td>
<td>1010 5</td>
</tr>
<tr>
<td>(\circ(a \times 0.5 + (b \odot 2)))</td>
<td>1010</td>
</tr>
<tr>
<td>((a + b)/2) (\circ)</td>
<td>1010 505</td>
</tr>
<tr>
<td>((a + b)/2)</td>
<td>1011</td>
</tr>
</tbody>
</table>
Algorithms based on \((a/2) + (b/2)\), using FMA

Algorithm: \(\circ(a \times 0.5 + (b \odot 2))\)
There is one rounding less thanks to the FMA operator.

Counter-example of correct rounding:

\[
a = 2001 \times 10^{10}, \ b = 2001 \times 10^{8}
\]

<table>
<thead>
<tr>
<th></th>
<th>(a)</th>
<th>2001</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(b)</td>
<td>2001</td>
</tr>
<tr>
<td></td>
<td>(b/2)</td>
<td>1000.5</td>
</tr>
<tr>
<td></td>
<td>(b \odot 2)</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>(a \times 0.5)</td>
<td>1000.5</td>
</tr>
<tr>
<td></td>
<td>(a \times 0.5 + (b \odot 2))</td>
<td>1010.5</td>
</tr>
<tr>
<td></td>
<td>(\circ(a \times 0.5 + (b \odot 2)))</td>
<td>1010</td>
</tr>
<tr>
<td></td>
<td>((a + b)/2)</td>
<td>1010505</td>
</tr>
<tr>
<td></td>
<td>(\circ((a + b)/2))</td>
<td>1011</td>
</tr>
</tbody>
</table>
Algorithms based on \((a/2) + (b/2)\), using FMA

Algorithm: \(\circ(a \times 0.5 + (b \odot 2))\)
There is one rounding less thanks to the FMA operator.

Counter-example of correct rounding:

\[
a = 2001 \times 10^{10}, \quad b = 2001 \times 10^8
\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2001</td>
</tr>
<tr>
<td></td>
<td>10005</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>a \times 0.5</td>
<td>10005</td>
<td></td>
</tr>
<tr>
<td>a \times 0.5 + (b \odot 2)</td>
<td>10105</td>
<td></td>
</tr>
<tr>
<td>(\circ(a \times 0.5 + (b \odot 2)))</td>
<td>1010</td>
<td></td>
</tr>
<tr>
<td>(a + b)/2</td>
<td>1010505</td>
<td></td>
</tr>
<tr>
<td>(\circ((a + b)/2))</td>
<td>1011</td>
<td></td>
</tr>
</tbody>
</table>
Outline

1. Radix-2 Average Algorithms
2. Unsuccessful Radix-10 Average Algorithm
3. Radix-10 Average Algorithm
4. Formal Proof with Coq and Flocq
5. Conclusion
TwoSum \((x, y)\) computes (with 6 flops) the sum of \(x\) and \(y\), and the rounding error. It works in radix-10 and returns the rounding and the error of an FP addition (always representable exactly by an FP number).

\[(a, b) = \text{TwoSum}(x, y) \implies x + y = a + b \land |b| \leq \frac{\text{ulp}(a)}{2}\]
Function Average10(x, y)

(a, b) = TwoSum(x, y)

if ◦(a × 0.5 − (a ⊗ 2)) = 0 then
    return ◦(b × 0.5 + (a ⊗ 2))
else
    return ◦(a × 0.5 + b)

- The if checks whether a/2 is a FP number.
- If $a/2 ∈ \mathbb{F}$, we have $a ⊗ 2 = a/2$. So the computations of line 4 are exact until the last rounding.
- In the other case, $a/2$ is a midpoint and we rely on the following lemma.
- In the other case, $b$ is not divided by 2 contrary to intuition.
Function `Average10(x, y)`

1. \((a, b) = \text{TwoSum}(x, y)\)
2. \(\text{if } o(a \times 0.5 - (a \odot 2)) = 0 \text{ then}\)
3. \(\quad \text{return } o(b \times 0.5 + (a \odot 2))\)
4. \(\text{else}\)
5. \(\quad \text{return } o(a \times 0.5 + b)\)

- The `if` checks whether \(a/2\) is a FP number.
- If \(a/2 \in \mathbb{F}\), we have \(a \odot 2 = a/2\). So the computations of line 4 are exact until the last rounding.
- In the other case, \(a/2\) is a midpoint and we rely on the following lemma.
- In the other case, \(b\) is not divided by 2 contrary to intuition.
Function Average10(x, y)
(a, b) = TwoSum(x, y)
if ◦(a × 0.5 − (a ⊙ 2)) = 0 then
    return ◦(b × 0.5 + (a ⊙ 2))
else
    return ◦(a × 0.5 + b)

- The if checks whether a/2 is a FP number.
- If a/2 ∈ F, we have a ⊙ 2 = a/2. So the computations of line 4 are exact until the last rounding.
- In the other case, a/2 is a midpoint and we rely on the following lemma.
- In the other case, b is not divided by 2 contrary to intuition.
The if checks whether $a/2$ is a FP number.

If $a/2 \in \mathbb{F}$, we have $a \odot 2 = a/2$. So the computations of line 4 are exact until the last rounding.

In the other case, $a/2$ is a midpoint and we rely on the following lemma.

In the other case, $b$ is not divided by 2 contrary to intuition.
1 Function Average10(x, y)
2 \[(a, b) = \text{TwoSum} (x, y)\]
3 \textbf{if} \(\circ(a \times 0.5 - (a \odot 2)) = 0\) \textbf{then}
4 \hspace{1em} \textbf{return} \(\circ(b \times 0.5 + (a \odot 2))\)
5 \textbf{else}
6 \hspace{1em} \textbf{return} \(\circ(a \times 0.5 + b)\)

- The \textbf{if} checks whether \(a/2\) is a FP number.
- If \(a/2 \in \mathbb{F}\), we have \(a \odot 2 = a/2\). So the computations of line 4 are exact until the last rounding.
- In the other case, \(a/2\) is a midpoint and we rely on the following lemma.
- In the other case, \(b\) is not divided by 2 contrary to intuition.
Lemma (Midpoint)

Let \( m = g + \frac{\text{ulp}(g)}{2} \) with \( g \in \mathbb{F} \), \( m > 0 \) and \( 0 < e \leq \frac{\text{ulp}(g)}{2} \).

- \( m \oplus e = g \)
- \( m \ominus e = \text{succ}(g) \)
Outline

1. Radix-2 Average Algorithms
2. Unsuccessful Radix-10 Average Algorithm
3. Radix-10 Average Algorithm
4. Formal Proof with Coq and Flocq
5. Conclusion
The Coq proof assistant

Floating-Point numbers library: Flocq (Boldo-Melquiond), which provides an FP numbers formalization and many results.

There are several FP formats in Flocq, defined as subsets of reals numbers $\mathbb{R}$. All formats depend on a radix ($\beta$).

- FLX: unbounded exponent range.
- FLT: exponent has a minimal value (gradual underflow).

<table>
<thead>
<tr>
<th>Format</th>
<th>Parameters</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLX</td>
<td>$\beta, p$</td>
<td>$</td>
</tr>
<tr>
<td>FLT</td>
<td>$\beta, p, e_{\text{min}}$</td>
<td>$</td>
</tr>
</tbody>
</table>

A real number is a FP number if equal to $m \times \beta^e$
We define our algorithm in Coq:

```coq
Definition average10 (x y : R) :=
  if (Req_bool (round (x/2 - round (x/2))) 0)
    then round (y/2 + round (x/2))
    else round (x/2 + y).
```

We assume that this function is called with the output of TwoSum.
Main Theorem

This is the main theorem, stating the correctness of the algorithm:

**Theorem** average10_correct :

\[
\forall a \ b, \ \text{format} \ a \ \rightarrow \ \text{format} \ b \ \rightarrow \\
\text{Rabs} \ b \ \leq \ (\text{ulp} \ a) / 2 \ \rightarrow \\
\text{average10} \ a \ b = \text{round} \ ((a + b) / 2).
\]

- \text{format} \ x \ means \ that \ x \ \in \ \mathbb{F}. \ We \ define \ it \ depending \ on \ the \ chosen \ format.
- \text{round} \ x \ is \ o(x). \ It \ also \ depends \ on \ the \ format, \ and \ is \ a \ rounding \ to \ nearest, \ with \ an \ arbitrary \ tie.
- \text{ulp} \ x \ is \ \text{ulp}(x).
Main Theorem

This is the main theorem, stating the correctness of the algorithm:

**Theorem** average10_correct :

\[
\forall a, b, \text{format } a \rightarrow \text{format } b \rightarrow \text{Rabs } b \leq \frac{\text{ulp } a}{2} \rightarrow \text{average10 } a \ b = \text{round } \left(\frac{(a + b)}{2}\right).
\]

- **format** \(x\) means that \(x \in \mathbb{F}\). We define it depending on the chosen format.
- **round** \(x\) is \(\circ(x)\). It also depends on the format, and is a rounding to nearest, with an arbitrary tie.
- **ulp** \(x\) is \(\text{ulp}(x)\).
We first prove it with an unbounded exponent range (FLX).

We then prove that it holds with gradual underflow (FLT).

- The test $\circ(a \times 0.5 - (a \ominus 2)) = 0$ is not equivalent to $a/2 \in \mathbb{F}$.
- In the else case, one must compute $\circ(a \times 0.5 + b)$, instead of $\circ(a \times 0.5 + b \ominus 2)$ (both would work in FLX).

We generalize it for any even radix.
Outline

1  Radix-2 Average Algorithms
2  Unsuccessful Radix-10 Average Algorithm
3  Radix-10 Average Algorithm
4  Formal Proof with Coq and Flocq
5  Conclusion
Summary:

- The algorithm computes the correct rounding (to nearest) of the average of two FP numbers: $\circ((a + b)/2)$.
- It holds with gradual underflow.
- It holds with any tie-breaking rule.
- It is formally-proved.
- It has been generalized to any even radix.

We have problems with spurious overflows (due to TwoSum).

We showed that it may not be straightforward to adapt some existing algorithms from radix-2 literature to radix-10.
Summary:

- The algorithm computes the correct rounding (to nearest) of the average of two FP numbers: \((a + b)/2\).
- It holds with gradual underflow.
- It holds with any tie-breaking rule.
- It is formally-proved.
- It has been generalized to any even radix.

We have problems with spurious overflows (due to TwoSum).

We showed that it may not be straightforward to adapt some existing algorithms from radix-2 literature to radix-10.