Exact, abstract, certifiable computations

Yann Kieffer[1]

[1]G-SCOP lab. of sciences for conception and optimization of production systems
Grenoble, France

Dec. 15th 2009 / MAP'09 / Monastir

How difficult it sometimes is to find a title for a talk

- The COMP project: past, present and future
  (COMP means: “a Collection Of Mathematical Programs”)
- Why computing (things) the way you prove (things) is a good idea
- “Reaching for the limits of computations”
- Consctructive mathematics meet computers

There is no general accepted scheme to implement all of mathematics—not even all that is implementable.

COMP is an effort towards this goal.

“Do only maths that can be turned into programs.
Write only programs that have mathematical content.”

How difficult it sometimes is to find a title for a talk

- The COMP project: past, present and future
  (COMP means: “a Collection Of Mathematical Programs”)
- Why computing (things) the way you prove (things) is a good idea
- “Reaching for the limits of computations”
- Consctructive mathematics meet computers

There is no general accepted scheme to implement all of mathematics—not even all that is implementable.

COMP is an effort towards this goal.

“Do only maths that can be turned into programs.
Write only programs that have mathematical content.”

How difficult it sometimes is to find a title for a talk

- The COMP project: past, present and future
  (COMP means: “a Collection Of Mathematical Programs”)
- Why computing (things) the way you prove (things) is a good idea
- “Reaching for the limits of computations”
- Consctructive mathematics meet computers

There is no general accepted scheme to implement all of mathematics—not even all that is implementable.

COMP is an effort towards this goal.

“Do only maths that can be turned into programs.
Write only programs that have mathematical content.”

How difficult it sometimes is to find a title for a talk

- The COMP project: past, present and future
  (COMP means: “a Collection Of Mathematical Programs”)
- Why computing (things) the way you prove (things) is a good idea
- “Reaching for the limits of computations”
- Consctructive mathematics meet computers

There is no general accepted scheme to implement all of mathematics—not even all that is implementable.

COMP is an effort towards this goal.

“Do only maths that can be turned into programs.
Write only programs that have mathematical content.”

How difficult it sometimes is to find a title for a talk

- The COMP project: past, present and future
  (COMP means: “a Collection Of Mathematical Programs”)
- Why computing (things) the way you prove (things) is a good idea
- “Reaching for the limits of computations”
- Consctructive mathematics meet computers

There is no general accepted scheme to implement all of mathematics—not even all that is implementable.

COMP is an effort towards this goal.

“Do only maths that can be turned into programs.
Write only programs that have mathematical content.”

How difficult it sometimes is to find a title for a talk

- The COMP project: past, present and future
  (COMP means: “a Collection Of Mathematical Programs”)
- Why computing (things) the way you prove (things) is a good idea
- “Reaching for the limits of computations”
- Consctructive mathematics meet computers

There is no general accepted scheme to implement all of mathematics—not even all that is implementable.

COMP is an effort towards this goal.

“Do only maths that can be turned into programs.
Write only programs that have mathematical content.”

How difficult it sometimes is to find a title for a talk

- The COMP project: past, present and future
  (COMP means: “a Collection Of Mathematical Programs”)
- Why computing (things) the way you prove (things) is a good idea
- “Reaching for the limits of computations”
- Consctructive mathematics meet computers

There is no general accepted scheme to implement all of mathematics—not even all that is implementable.

COMP is an effort towards this goal.

“Do only maths that can be turned into programs.
Write only programs that have mathematical content.”

Nature of the work

- “Bishop’s program”
- Proof of concept; founded on the motto:
  “computable mathematics = constructive mathematics”
  (‘Bishop’s thesis’)
- Analysis of the limits on implementability (with current methods)
- Which of those limits can be overcome, and how?
- Example realization in
  “COMP phase 1: (exact) real numbers”

Nature of the work

- “Bishop’s program”
- Proof of concept; founded on the motto:
  “computable mathematics = constructive mathematics”
  (‘Bishop’s thesis’)
- Analysis of the limits on implementability (with current methods)
- Which of those limits can be overcome, and how?
- Example realization in
  “COMP phase 1: (exact) real numbers”

Nature of the work

- “Bishop’s program”
- Proof of concept; founded on the motto:
  “computable mathematics = constructive mathematics”
  (‘Bishop’s thesis’)
- Analysis of the limits on implementability (with current methods)
- Which of those limits can be overcome, and how?
- Example realization in
  “COMP phase 1: (exact) real numbers”

Nature of the work

- “Bishop’s program”
- Proof of concept; founded on the motto: “computable mathematics = constructive mathematics” (‘Bishop’s thesis’)
- Analysis of the limits on implementability (with current methods)
- Which of those limits can be overcome, and how?
- Example realization in “COMP phase 1: (exact) real numbers”

Nature of the work

- “Bishop’s program”
- Proof of concept; founded on the motto:
  “computable mathematics $=$ constructive mathematics”
  (‘Bishop’s thesis’)
- Analysis of the limits on implementability (with current methods)
- Which of those limits can be overcome, and how?
- Example realization in
  “COMP phase 1: (exact) real numbers”

How does this talk relate to its title?

- Both the whole COMP project and its phase 1 target only *exact* mathematics. Actually, just 'mathematics'.
- A natural benefit of the experience is that the code has the right shape to be *proved*, say with the help of Coq.
- In the case of real numbers, *speed* is a major concern to people presented with this research program. We show that
  - Speed is good enough to actually compute
  - The tension between provability and speed is not too strong (argument only—no proofs implemented yet!)

How does this talk relate to its title?

- Both the whole COMP project and its phase 1 target only *exact* mathematics. Actually, just 'mathematics'.
- A natural benefit of the experience is that the code has the right shape to be *proved*, say with the help of Coq.
- In the case of real numbers, *speed* is a major concern to people presented with this research program. We show that
  - Speed is good enough to actually compute
  - The tension between provability and speed is not too strong (argument only—no proofs implemented yet!)

How does this talk relate to its title?

- Both the whole COMP project and its phase 1 target only *exact* mathematics. Actually, just 'mathematics'.
- A natural benefit of the experience is that the code has the right shape to be *proved*, say with the help of Coq.
- In the case of real numbers, *speed* is a major concern to people presented with this research program. We show that
  - Speed is good enough to actually compute
  - The tension between provability and speed is not too strong (argument only—no proofs implemented yet!)

How does this talk relate to its title?

- Both the whole COMP project and its phase 1 target only *exact* mathematics. Actually, just 'mathematics'.
- A natural benefit of the experience is that the code has the right shape to be *proved*, say with the help of Coq.
- In the case of real numbers, *speed* is a major concern to people presented with this research program. We show that
  - Speed is good enough to actually compute
  - The tension between provability and speed is not too strong (argument only—no proofs implemented yet!)

How does this talk relate to its title?

- Both the whole COMP project and its phase 1 target only *exact* mathematics. Actually, just 'mathematics'.
- A natural benefit of the experience is that the code has the right shape to be *proved*, say with the help of Coq.
- In the case of real numbers, *speed* is a major concern to people presented with this research program. We show that
  - Speed is good enough to actually compute
  - The tension between provability and speed is not too strong (argument only—no proofs implemented yet!)

Today's menu

Real numbers in a computer. . . is it possible?

"To compute with something, you need a finite representation of that something—because your computer's memory is finite"

- What is a representation?
- This idea led to treat real numbers through approximated values
- but *reliability* got lost in the process!

But wait. . . an expression is also a finite representation.
Example: $\pi + 1$
Conclusion: there may be *different representations* for the same object. Let's make a good choice!

Real numbers in a computer. . . is it possible?

"To compute with something, you need a finite representation of that something—because your computer's memory is finite"

- What is a representation?
- This idea led to treat real numbers through approximated values
- but *reliability* got lost in the process!

But wait. . . an expression is also a finite representation.
Example: $\pi + 1$
Conclusion: there may be *different representations* for the same object. Let's make a good choice!

Real numbers in a computer. . . is it possible?

"To compute with something, you need a finite representation of that something—because your computer's memory is finite"

- What is a representation?
- This idea led to treat real numbers through approximated values
- but *reliability* got lost in the process!

But wait. . . an expression is also a finite representation.
Example: $\pi + 1$
Conclusion: there may be *different representations* for the same object. Let's make a good choice!

Real numbers in a computer. . . is it possible?

"To compute with something, you need a finite representation of that something—because your computer's memory is finite"

- What is a representation?
- This idea led to treat real numbers through approximated values
- but *reliability* got lost in the process!

But wait. . . an expression is also a finite representation.
Example: $\pi + 1$
Conclusion: there may be *different representations* for the same object. Let's make a good choice!

Real numbers in a computer. . . is it possible?

"To compute with something, you need a finite representation of that something—because your computer's memory is finite"

- What is a representation?
- This idea led to treat real numbers through approximated values
- but *reliability* got lost in the process!

But wait. . . an expression is also a finite representation.

Example: $\pi + 1$

Conclusion: there may be *different representations* for the same object. Let's make a good choice!

Real numbers in a computer. . . is it possible?

"To compute with something, you need a finite representation of that something—because your computer's memory is finite"

- What is a representation?
- This idea led to treat real numbers through approximated values
- but *reliability* got lost in the process!

But wait. . . an expression is also a finite representation.
Example: $\pi + 1$
Conclusion: there may be *different representations* for the same object. Let's make a good choice!

Real numbers in a computer. . . is it possible?

"To compute with something, you need a finite representation of that something—because your computer's memory is finite"

- What is a representation?
- This idea led to treat real numbers through approximated values
- but *reliability* got lost in the process!

But wait. . . an expression is also a finite representation.
Example: $\pi + 1$
Conclusion: there may be *different representations* for the same object. Let's make a good choice!

Review of proposals for implementing real numbers

- computer algebra systems - algebraic numbers
- interval arithmetic, with or without iterative recomputation - “forward chaining”
- exact - “backward chaining”
  - lambda-expressions-based (with or without unbounded computations)
  - (lazy) streams of signed digits
  - (lazy) streams of holographic operations
- extracted from prover scripts (*exact*, *bounded*)

Review of proposals for implementing real numbers

- computer algebra systems - algebraic numbers
- interval arithmetic, with or without iterative recomputation - “forward chaining”
- exact - “backward chaining”
  - lambda-expressions-based (with or without unbounded computations)
  - (lazy) streams of signed digits
  - (lazy) streams of holographic operations
- extracted from prover scripts (*exact*, *bounded*)

Review of proposals for implementing real numbers

- computer algebra systems - algebraic numbers
- interval arithmetic, with or without iterative recomputation - “forward chaining”
- exact - “backward chaining”
  - lambda-expressions-based (with or without unbounded computations)
  - (lazy) streams of signed digits
  - (lazy) streams of holographic operations
- extracted from prover scripts (*exact*, *bounded*)

Review of proposals for implementing real numbers

- computer algebra systems - algebraic numbers
- interval arithmetic, with or without iterative recomputation - “forward chaining”
- exact - “backward chaining”
  - lambda-expressions-based (with or without unbounded computations)
  - (lazy) streams of signed digits
  - (lazy) streams of holographic operations
- extracted from prover scripts (*exact*, *bounded*)

Review of proposals for implementing real numbers

- computer algebra systems - algebraic numbers
- interval arithmetic, with or without iterative recomputation - “forward chaining”
- exact - “backward chaining”
  - lambda-expressions-based (with or without unbounded computations)
  - (lazy) streams of signed digits
  - (lazy) streams of holographic operations
- extracted from prover scripts (*exact*, *bounded*)

Review of proposals for implementing real numbers

- computer algebra systems - algebraic numbers
- interval arithmetic, with or without iterative recomputation - “forward chaining”
- exact - “backward chaining”
  - lambda-expressions-based (with or without unbounded computations)
  - (lazy) streams of signed digits
  - (lazy) streams of holographic operations
- extracted from prover scripts (*exact*, *bounded*)

Review of proposals for implementing real numbers

- computer algebra systems - algebraic numbers
- interval arithmetic, with or without iterative recomputation - “forward chaining”
- exact - “backward chaining”
  - lambda-expressions-based (with or without unbounded computations)
  - (lazy) streams of signed digits
  - (lazy) streams of holographic operations
- extracted from prover scripts (*exact*, *bounded*)

Forward chaining

Consider the expression "$\pi + 1$". Suppose we want 30 digits of that.

Forward chaining:

- fix an initial precision *p*;
- compute π and 1 with precision *p*;
- compute the (approximate) value of $\pi + 1$;
- check if the prcision obtained for $\pi + 1$ is better than the expected precision; if not, restart with a better initial precision.

Backward chaining

Consider the expression "$\pi + 1$". Suppose we want 30 digits of that.

Backward chaining:

- from the precision desired for the final expression, derive precisions for the arguments of '$+$';
- 'recursively' compute arguments of '$+$' with the necessary precision;
- compute the value of the application of '$+$' to its arguments.

Forward and backward chaining contrasted

Forward chaining:

- needs less abstract structures for implementing;
- needs ‘good choices’ for initial precision, and precision augmentation step for recomputation;
- recomputations take time;
- if precision augmentation step is big enough, it might not be an issue;
- difficult to prove (and have) convergence.

Forward and backward chaining contrasted

Backward chaining:

- more involved implementation;
- only one computation, when expression is a tree, but issues with DAG-expressions;
- more abstract, closer to mathematical thinking.

Isn't that what the CCA community is about?

(**CCA** stands for: computability and complexity in analysis)

Well, sort of! But...

Most people in the CCA community are theoretical computer scientists, with a strong background in computability and/or complexity.

Example: some consider that real numbers can (should?) be labeled by integers. Is it relevant for implementation purposes?

Example 2: they proved that max of a function on $[0, 1]$ is NP-complete. Isn't that depressing?

My point of view: constructive mathematics put me in a better mood. Anyway, I learnt about constructive mathematics long before I learnt about CCA!

Isn't that what the CCA community is about?

(**CCA** stands for: computability and complexity in analysis)
Well, sort of! But...

Most people in the CCA community are theoretical computer scientists, with a strong background in computability and/or complexity.

Example: some consider that real numbers can (should?) be labeled by integers. Is it relevant for implementation purposes?

Example 2: they proved that max of a function on $[0, 1]$ is NP-complete. Isn't that depressing?

My point of view: constructive mathematics put me in a better mood. Anyway, I learnt about constructive mathematics long before I learnt about CCA!

Isn't that what the CCA community is about?

(**CCA** stands for: computability and complexity in analysis)
Well, sort of! But...

Most people in the CCA community are theoretical computer scientists, with a strong background in computability and/or complexity.

Example: some consider that real numbers can (should?) be labeled by integers. Is it relevant for implementation purposes?

Example 2: they proved that max of a function on $[0, 1]$ is NP-complete. Isn't that depressing?

My point of view: constructive mathematics put me in a better mood. Anyway, I learnt about constructive mathematics long before I learnt about CCA!

Isn't that what the CCA community is about?

(**CCA** stands for: computability and complexity in analysis)
Well, sort of! But...

Most people in the CCA community are theoretical computer scientists, with a strong background in computability and/or complexity.

Example: some consider that real numbers can (should?) be labeled by integers. Is it relevant for implementation purposes?

Example 2: they proved that max of a function on $[0, 1]$ is NP-complete. Isn't that depressing?

My point of view: constructive mathematics put me in a better mood. Anyway, I learnt about constructive mathematics long before I learnt about CCA!

Isn't that what the CCA community is about?

(**CCA** stands for: computability and complexity in analysis)
Well, sort of! But...

Most people in the CCA community are theoretical computer scientists, with a strong background in computability and/or complexity.

Example: some consider that real numbers can (should?) be labeled by integers. Is it relevant for implementation purposes?

Example 2: they proved that max of a function on $[0, 1]$ is NP-complete. Isn't that depressing?

My point of view: constructive mathematics put me in a better mood. Anyway, I learnt about constructive mathematics long before I learnt about CCA!

Constructive Mathematics: What Is It?

- A way of doing maths where existential statements are made explicit
- This sometimes requires to alter the form of the statements themselves

Example

'prime numbers form an infinite set' $\longrightarrow$
'given any finite list of prime numbers, one can build another one'

Note: no proof of that theorem avoids the work needed to prove a shorter, more precise, constructive version.

Constructive Mathematics: What Is It?

- A way of doing maths where existential statements are made explicit
- This sometimes requires to alter the form of the statements themselves

Example

'prime numbers form an infinite set' $\longrightarrow$
'given any finite list of prime numbers, one can build another one'

Note: no proof of that theorem avoids the work needed to prove a shorter, more precise, constructive version.

Constructive Mathematics: What Is It?

- A way of doing maths where existential statements are made explicit
- This sometimes requires to alter the form of the statements themselves

Example

'prime numbers form an infinite set' $\longrightarrow$
'given any finite list of prime numbers, one can build another one'

Note: no proof of that theorem avoids the work needed to prove a shorter, more precise, constructive version.

Constructive Mathematics: What Is It?

- A way of doing maths where existential statements are made explicit
- This sometimes requires to alter the form of the statements themselves

Example

'prime numbers form an infinite set' $\longrightarrow$
'given any finite list of prime numbers, one can build another one'

Note: no proof of that theorem avoids the work needed to prove a shorter, more precise, constructive version.

Constructive Mathematics: Isn't that too restricted?

- Errett Bishop's 'Foundations of Constructive Analysis'
- '$x = 0$ or $x \neq 0$' is not true anymore: no finite procedure can decide which of the two is true.
- Constructive replacement: 'given $a < b$, either $x \leq b$ or $a \leq x$'

Classical mathematics: 'shortcut mathematics'
Constructive mathematics: explicit, implementable mathematics.

Constructive Mathematics: Isn't that too restricted?

- Errett Bishop's 'Foundations of Constructive Analysis'
- '$x = 0$ or $x \neq 0$' is not true anymore: no finite procedure can decide which of the two is true.
- Constructive replacement: 'given $a < b$, either $x \leq b$ or $a \leq x$'

Classical mathematics: 'shortcut mathematics'
Constructive mathematics: explicit, implementable mathematics.

Constructive Mathematics: Isn't that too restricted?

- Errett Bishop's 'Foundations of Constructive Analysis'
- '$x = 0$ or $x \neq 0$' is not true anymore: no finite procedure can decide which of the two is true.
- Constructive replacement: 'given $a < b$, either $x \leq b$ or $a \leq x$'

Classical mathematics: 'shortcut mathematics'
Constructive mathematics: explicit, implementable mathematics.

Constructive Mathematics: Isn't that too restricted?

- Errett Bishop's 'Foundations of Constructive Analysis'
- '$x = 0$ or $x \neq 0$' is not true anymore: no finite procedure can decide which of the two is true.
- Constructive replacement: 'given $a < b$, either $x \leq b$ or $a \leq x$'

Classical mathematics: 'shortcut mathematics'
Constructive mathematics: explicit, implementable mathematics.

Constructive approaches to real numbers

Two different approaches to building real numbers in classical mathematics yield two different mathematical notions in constructive mathematics.

More precisely, being a

Dedekind real

A function $f : \mathcal{Q} \to \{0, 1\}$ such that $x \leq y \implies f(x) \leq f(y)$

is a stronger requirement than being a

Cauchy real

A function $f : \mathcal{N} \to \mathcal{Q}$ such that there is a function $\phi : \mathcal{Q}^+ \to \mathcal{N}$ such that $n, m > \phi(r) \implies |f(n) - f(m)| < r$

Implementing Constructive (Cauchy) Real Numbers

- Informally: a real number is an object one can ask rational approximations of, to any precision.
- Rational numbers can be a hassle.
- Let's take dyadic numbers instead ($\frac{p}{2^q}$)
- In COMP: the real number x is an *oracle* that given any integer k replies with an integer q such that

$$|x - \frac{q}{2^k}| < \frac{1}{2^k}$$

Implementing Constructive (Cauchy) Real Numbers

- Informally: a real number is an object one can ask rational approximations of, to any precision.
- Rational numbers can be a hassle.
- Let's take dyadic numbers instead ($\frac{p}{2^q}$)
- In COMP: the real number x is an *oracle* that given any integer k replies with an integer q such that

$$|x - \frac{q}{2^k}| < \frac{1}{2^k}$$

Implementing Constructive (Cauchy) Real Numbers

- Informally: a real number is an object one can ask rational approximations of, to any precision.
- Rational numbers can be a hassle.
- Let's take dyadic numbers instead ($\frac{p}{2^q}$)
- In COMP: the real number x is an *oracle* that given any integer k replies with an integer q such that

$$|x - \frac{q}{2^k}| < \frac{1}{2^k}$$

Implementing Constructive (Cauchy) Real Numbers

- Informally: a real number is an object one can ask rational approximations of, to any precision.
- Rational numbers can be a hassle.
- Let's take dyadic numbers instead ($\frac{p}{2^q}$)
- In COMP: the real number x is an *oracle* that given any integer k replies with an integer q such that

$$|x - \frac{q}{2^k}| < \frac{1}{2^k}$$

Constructive mathematics meets functional programming

How does one implement functions?

- (Nearly) every programming language has functions
- But that's a notion of *explicit functions*: functions you, the programmer, completely describe
- Chances are you don't want to program each real number you use as an explicit function, written from scratch!
- If functions could take functions as arguments, and return functions as values, we'd be all set!

This is the (old times) definition of 'functional programming'!

Times have changed. With such a definition, Python is as much a functional programming language as Caml, Haskell or Lisp!

Constructive mathematics meets functional programming

How does one implement functions?

- (Nearly) every programming language has functions
- But that's a notion of *explicit functions*: functions you, the programmer, completely describe
- Chances are you don't want to program each real number you use as an explicit function, written from scratch!
- If functions could take functions as arguments, and return functions as values, we'd be all set!

This is the (old times) definition of 'functional programming'!

Times have changed. With such a definition, Python is as much a functional programming language as Caml, Haskell or Lisp!

COMP real numbers (cont'd)

COMP real numbers model

the real number x is an *oracle* that given any integer k replies with an integer q such that

$$|x - \frac{q}{2^k}| < \frac{1}{2^k}$$

Is this finite data?

- The oracle can be a (finite) program
- The oracle could also be some input from the outside world
- As a decimal expansion, it won't usually look finite

COMP real numbers (cont'd)

COMP real numbers model

the real number x is an *oracle* that given any integer k replies with an integer q such that

$$|x - \frac{q}{2^k}| < \frac{1}{2^k}$$

Is this finite data?

- The oracle can be a (finite) program
- The oracle could also be some input from the outside world
- As a decimal expansion, it won't usually look finite

COMP real numbers (cont'd)

COMP real numbers model

the real number x is an *oracle* that given any integer k replies with an integer q such that

$$|x - \frac{q}{2^k}| < \frac{1}{2^k}$$

Is this finite data?

- The oracle can be a (finite) program
- The oracle could also be some input from the outside world
- As a decimal expansion, it won't usually look finite

Real Number Type and Basic Operations

First (current) implementation of COMP as a haskell library.

- type Index = Integer
- type RealN = Index -> Integer

Code

```
realNeg :: RealN -> RealN
realNeg x i = - (x i)
```

Note: The type of realNeg can also be written:
'RealN -> Index -> Integer'

Real Number Type and Basic Operations

First (current) implementation of COMP as a haskell library.

- type Index = Integer
- type RealN = Index -> Integer

Code

```
realNeg :: RealN -> RealN
realNeg x i = - (x i)
```

Note: The type of realNeg can also be written:
'RealN -> Index -> Integer'

Some more example code

Code for the sum of two real numbers:

Code

```
realSum :: RealN -> RealN -> RealN
realSum x y i =
      ((x (i+2)) + (y (i+2)) + 2) `div` 4
```

Multiplication by a power of 2:

Code

```
scaleBy :: RealN -> Index -> RealN
scaleBy x i j = x (i+j)
```

Some more example code

Code for the sum of two real numbers:

Code

```
realSum :: RealN -> RealN -> RealN
realSum x y i =
      ((x (i+2)) + (y (i+2)) + 2) `div` 4
```

Multiplication by a power of 2:

Code

```
scaleBy :: RealN -> Index -> RealN
scaleBy x i j = x (i+j)
```

Bounded Reals and Non-Null Reals

- To compute a product, we need an upper bound on the absolute value of operands
- To compute an inverse, we need a lower bound on the absolute value of the operand

- `(BR k x)` is the real number x together with the information that $|x| < 2^k$
- `(NNR h x)` is the real number x together with the information that $|x| > 2^{-h}$

Bounded Reals and Non-Null Reals

- To compute a product, we need an upper bound on the absolute value of operands
- To compute an inverse, we need a lower bound on the absolute value of the operand

- `(BR k x)` is the real number x together with the information that $|x| < 2^k$
- `(NNR h x)` is the real number x together with the information that $|x| > 2^{-h}$

More elaborate functions

- Inverse, Sqrt: using Newton iterations
- Log, Pi, Arctan: using AGM iterations
- Exp through Log with Newton
- Tan through Arctan with Newton
- Other trigonometric functions with Tan

Usage examples

Code

```
*Reals1> showRealN (realSum realPi one) 40
"4.1415926535897932384626433832795028841972"
*Reals1> showRealN (realLog (NNR 0 realPi) ) 40
"1.1447298858494001741434273513530587116473"
*Reals1> showRealN (iterateN realLogisticMap half 1000) 40
".79174674092244363768698535805863962044 99"
```

Usage examples

Code

```
*Reals1> showRealN (realSum realPi one) 40
"4.1415926535897932384626433832795028841972"
*Reals1> showRealN (realLog (NNR 0 realPi) ) 40
"1.1447298858494001741434273513530587116473"
*Reals1> showRealN (iterateN realLogisticMap half 1000) 40
".7917467409224436376869853580586396204499"
```

Usage examples

Code

```
*Reals1> showRealN (realSum realPi one) 40
"4.1415926535897932384626433832795028841972"
*Reals1> showRealN (realLog (NNR 0 realPi) ) 40
"1.1447298858494001741434273513530587116473"
*Reals1> showRealN (iterateN realLogisticMap half 1000) 40
".7917467409224436376869853580586396204499"
```

Usage examples

Code

```
*Reals1> showRealN (realSum realPi one) 40
"4.1415926535897932384626433832795028841972"
*Reals1> showRealN (realLog (NNR 0 realPi) ) 40
"1.1447298858494001741434273513530587116473"
*Reals1> showRealN (iterateN realLogisticMap half 1000) 40
".7917467409224436376869853580586396204499"
```

Usage examples

Code

```
*Reals1> showRealN (realSum realPi one) 40
"4.1415926535897932384626433832795028841972"
*Reals1> showRealN (realLog (NNR 0 realPi) ) 40
"1.1447298858494001741434273513530587116473"
*Reals1> showRealN (iterateN realLogisticMap half 1000) 40
".7917467409224436376869853580586396204499"
```

Usage examples

Code

```
*Reals1> showRealN (realSum realPi one) 40
"4.1415926535897932384626433832795028841972"
*Reals1> showRealN (realLog (NNR 0 realPi) ) 40
"1.1447298858494001741434273513530587116473"
*Reals1> showRealN (iterateN realLogisticMap half 1000) 40
".79174674092244363768698535805863962044 99"
```

Usage examples

Code

```
*Reals1> showRealN (realSum realPi one) 40
"4.1415926535897932384626433832795028841972"
*Reals1> showRealN (realLog (NNR 0 realPi) ) 40
"1.1447298858494001741434273513530587116473"
*Reals1> showRealN (iterateN realLogisticMap half 1000) 40
".7917467409224436376869853580586396204499"
```

A few words about speed

All algorithms are asymptotically best (among known algorithms).

Let $M(n)$ denote the complexity of $n-$digits multiplication.

Complexity $M(n)$: multiply, inverse, square root.

Complexity $\log(n)M(n)$: π, ln, exp, arctan, sin, cos, ...

Note: all computations reduce in the end to integer multiplies, adds, and shifts

A few words about speed

All algorithms are asymptotically best (among known algorithms).

Let $M(n)$ denote the complexity of $n-$digits multiplication.

Complexity $M(n)$: multiply, inverse, square root.

Complexity $\log(n)M(n)$: π, ln, exp, arctan, sin, cos, ...

Note: all computations reduce in the end to integer multiplies, adds, and shifts

Formal proofs for the COMP1 code?

"COMP phase 1" should be easy to prove formally - it was written with proofs in mind!

- Pure functional style
  - no side effects;
  - subexpression values do not depend on context;
  - clear semantics.
- Bounded computations only
- No 'computer science style' optimizations
- No optimizations leading to harder proofs
- Less than 500 lines of code

Summary

- "Computable Mathematics = Constructive Mathematics"
- Constructive mathematics as an effective guide to computable mathematics
- Constructive mathematics do not appear to put limits on computations
- Complexity is a different issue from computability - choosing the right algorithm is important!
- Programming doesn't have to be too different from doing mathematics!
- Abstraction and Efficiency can be reconciled (as seen on real numbers)
- Paving the way for fast, certified numerical computations

Summary

- “Computable Mathematics = Constructive Mathematics”
- Constructive mathematics as an effective guide to computable mathematics
- Constructive mathematics do not appear to put limits on computations
- Complexity is a different issue from computability - choosing the right algorithm is important!
- Programming doesn’t have to be too different from doing mathematics!
- Abstraction and Efficiency can be reconciled (as seen on real numbers)
- Paving the way for fast, certified numerical computations

Summary

- “Computable Mathematics = Constructive Mathematics”
- Constructive mathematics as an effective guide to computable mathematics
- Constructive mathematics do not appear to put limits on computations
- Complexity is a different issue from computability - choosing the right algorithm is important!
- Programming doesn’t have to be too different from doing mathematics!
- Abstraction and Efficiency can be reconciled (as seen on real numbers)
- Paving the way for fast, certified numerical computations

Summary

- “Computable Mathematics = Constructive Mathematics”
- Constructive mathematics as an effective guide to computable mathematics
- Constructive mathematics do not appear to put limits on computations
- Complexity is a different issue from computability - choosing the right algorithm is important!
- Programming doesn’t have to be too different from doing mathematics!
- Abstraction and Efficiency can be reconciled (as seen on real numbers)
- Paving the way for fast, certified numerical computations

Summary

- “Computable Mathematics = Constructive Mathematics”
- Constructive mathematics as an effective guide to computable mathematics
- Constructive mathematics do not appear to put limits on computations
- Complexity is a different issue from computability - choosing the right algorithm is important!
- Programming doesn’t have to be too different from doing mathematics!
- Abstraction and Efficiency can be reconciled (as seen on real numbers)
- Paving the way for fast, certified numerical computations

Summary

- “Computable Mathematics = Constructive Mathematics”
- Constructive mathematics as an effective guide to computable mathematics
- Constructive mathematics do not appear to put limits on computations
- Complexity is a different issue from computability - choosing the right algorithm is important!
- Programming doesn’t have to be too different from doing mathematics!
- Abstraction and Efficiency can be reconciled (as seen on real numbers)
- Paving the way for fast, certified numerical computations

Summary

- “Computable Mathematics = Constructive Mathematics”
- Constructive mathematics as an effective guide to computable mathematics
- Constructive mathematics do not appear to put limits on computations
- Complexity is a different issue from computability - choosing the right algorithm is important!
- Programming doesn’t have to be too different from doing mathematics!
- Abstraction and Efficiency can be reconciled (as seen on real numbers)
- Paving the way for fast, certified numerical computations

COMP was not the first!

Other attempts at similar goals

(Using constructive mathematics as a guide to implement mathematics)

- `Kenzo` for algebraic topology (Sergeraert)
- `Few digits` for real numbers (O'Connor)

Are there others?

Other exact real packages

- CReal (Menissier-Morain)
- IrRam (N. Müller)
- ICReals
- RealLib, Ginac, XR, and some others

Those designed for speed (like IrRam) were not put together with proofs in mind.

But they outperform COMP.

Perspectives for COMP phase 1

- Improve speed
- Detailed pencil and paper proofs
- Formal proofs
- Haskell type classes integration
- Modularity $\longrightarrow$ try other basic types
- Publish the source code
- Publish a paper about it (where?)

Perspectives for the COMP project

- Limits, series, integrals
- Complex numbers
- Matrices (exact, real and complex)
- Algebra (with approximation needs)
- Foster collaboration/put students on the project
- Turn COMP into a real mathematical experimentation platform

Questions

Questions?