# Moessner's Theorem: an exercise in coinductive reasoning in Coq

Robbert Krebbers
Joint work with Louis Parlant and Alexandra Silva

Radboud University Nijmegen

November 12, 2013 @ COIN, CWI, Amsterdam

# Moessner's construction ($n = 4$)

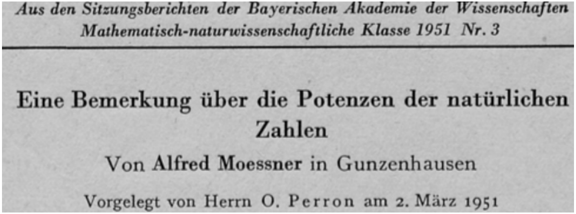
| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | ~~4~~ | 5 | 6 | 7 | ~~8~~ | 9 | 10 | 11 | ~~12~~ | 13 | 14 | 15 | ~~16~~ |
| 1 | 3 | ~~6~~ | | 11 | 17 | ~~24~~ | | 33 | 43 | ~~54~~ | | 67 | 71 | ~~96~~ | |
| 1 | ~~4~~ | | | 15 | ~~32~~ | | | 65 | ~~108~~ | | | 175 | ~~256~~ | | |
| 1 | | | | 16 | | | | 81 | | | | 256 | | | |
| $1^4$ | | | | $2^4$ | | | | $3^4$ | | | | $4^4$ | | | |

## Theorem (Moessner's Conjecture/Theorem)

*This construction gives* $1^n, 2^n, 3^n, \ldots$ *starting with any* $n \in \mathbb{N}$

# History

1951 Moessner conjectures it



*Aus den Sitzungsberichten der Bayerischen Akademie der Wissenschaften*
*Mathematisch-naturwissenschaftliche Klasse 1951 Nr. 3*

**Eine Bemerkung über die Potenzen der natürlichen Zahlen**

Von Alfred Moessner in Gunzenhausen

Vorgelegt von Herrn O. Perron am 2. März 1951

1952 Perron proves it

1952 Paasche and Salié generalize it

1966 Long generalizes it

2010 Niqui & Rutten present a new and elegant proof using coinduction

2013 This talk: Niqui & Rutten's proof formalized in Coq and extended to Long and Salié's generalization

# Niqui & Rutten's proof in a nutshell

Reduce the problem to equivalence of functional programs

- Describe Moessner's construction using stream operations

$$\text{Moessner } n := \Sigma\,\mathrm{D}_2^1\,\Sigma\,\mathrm{D}_3^2 \cdots \Sigma\,\mathrm{D}_n^{n-1}\,\texttt{nats}$$

- The stream $\texttt{nats}^{\langle n \rangle}$ is also a functional program

## Theorem (Moessner's Theorem)

*We have* Moessner $n = \texttt{nats}^{\langle n \rangle}$ *for all* $n \in \mathbb{N}$

## Proof.

Using the coinduction principle □

# Streams in Coq

```
CoInductive Stream (A : Type) : Type :=
  SCons : A → Stream A → Stream A.
Arguments SCons {_} _ _.
Infix ":::" := SCons.
```

Coinductive types are similar to inductive types:

- The above defines `Stream` $A$ as the *greatest fixpoint* of $A \times -$ (whereas `list` $A$ is the *least fixpoint* of $1 + A \times -$)
- Terms of coinductive types can represent infinite objects
- Computation with coinductive types is lazy

# Pattern matching

The destructors are implemented using pattern matching

```
Definition head {A} (s : Stream A) : A :=
  match s with x ::: _ ⇒ x end.
Definition tail {A} (s : Stream A) : Stream A :=
  match s with _ ::: s ⇒ s end.
Notation "s ‘" := (tail s).
```

# Corecursive definitions

How to define the constant stream:

$$\overline{x} = (x, x, x, \dots)$$

Using the `CoFixpoint` command:

```
CoFixpoint repeat {A} (x : A) : Stream A := x ::: #x
where "# x" := (repeat x).
```

Such `CoFixpoint` definitions should satisfy certain rules

# Productivity

To ensure logical consistency:

- Recursive definitions should be *terminating*
- Corecursive definitions should be *productive*

Intuitively this means that terms of coinductive types should always produce a constructor

The definition:

```
CoFixpoint repeat {A} (x : A) : Stream A := x ::: #x
where "# x" := (repeat x).
```

always produces the constructor `x ::: #x`

But, here this would not be the case:

```
CoFixpoint bad : Stream False := bad.
```

Problem: productivity is undecidable

# Guard condition

Since productivity is undecidable:

- Corecursive definitions should satisfy the *guard condition*, a stronger decidable syntactical criterion
- Over simplified, this means that a `CoFixpoint` definition should have the following shape (with $0 < n$):

```
CoFixpoint f p⃗ : Stream A :=
  x₀ ::: x₁ :::  ... ::: xₙ ::: f q⃗.
```

- Not guarded:

```
CoFixpoint bad : Stream False := bad.
```

- Guarded:

```
CoFixpoint repeat {A} (x : A) : Stream A := x ::: #x
where "# x" := (repeat x).
```

# Stream equality?

We wish to prove that two streams "are equal"

- Coq's notion of intensional Leibniz equality = only equates streams defined using "the same algorithm"
- For example, `#(f x) = map f (#x)` is not provable

We use bisimilarity $\equiv$ instead

```
CoInductive equal {A} (s t : Stream A) : Prop :=
  make_equal : head s = head t → s‘ ≡ t‘ → s ≡ t
where "s ≡ t" := (@equal _ s t).
```

Bisimilarity is a congruence, and we use Coq's setoid machinery to enable rewriting using it

# Ring operations

We define a ring structure using element-wise operations:

```
Infix "⊕ " := (zip_with Z.add).
Infix "⊖":= (zip_with Z.sub).
Infix "⊙ " := (zip_with Z.mul).
Notation "⊖ s":= (map Z.opp s).
```

Register that $(\overline{0}, \overline{1}, \oplus, \odot, \ominus)$ is indeed a ring:

```
Lemma stream_ring_theory :
  ring_theory (#0) (#1) (zip_with Z.add) (zip_with Z.mul)
    (zip_with Z.sub) (map Z.opp) equal.
Add Ring stream : stream_ring_theory.
```

The `ring` tactic can now solve ring equations over streams:

```
Lemma foo s t u :
  (#1 ⊙ t ⊕ u) ⊙ s ≡ (t ⊙ s) ⊕ (s ⊙ u) ⊕ #0 ⊙ u.
Proof. ring. Qed.
```

# Summing

Niqui & Rutten define the *partial sums*

$$\Sigma\, s = (s(0), s(0) + s(1), s(0) + s(1) + s(2), \dots)$$

by $(\Sigma\, s)(0) = s(0)$ and $(\Sigma\, s)' = \overline{s(0)} \oplus \Sigma\, s'$

The Coq definition

```
CoFixpoint Ssum (s : Stream Z) : Stream Z :=
  head s ::: #head s ⊕ Σ s‘
where "'Σ' s" := (Ssum s).
```

does not satisfy the guard condition due to the call `#head s ⊕ _`

Our definition uses an accumulator:

```
CoFixpoint Ssum (i : Z) (s : Stream Z) : Stream Z :=
  head s + i ::: Ssum (head s + i) (s‘).
Notation "'Σ' s" := (Ssum 0 s).
Lemma Ssum_tail s : (Σ s)‘ ≡ #head s ⊕ Σ s‘.
```

# Dropping

The *drop operators* $\mathsf{D}^i_k\, s$, with for example

$$\mathsf{D}^1_3\, s = (s(0), s(2), s(3), s(5), s(6), s(8), \dots)$$

are defined as:

```
CoFixpoint Sdrop {A} (i k : nat) (s : Stream A) : Stream A :=
  match i with
  | O ⇒ head (s‘) ::: D@{k-2,k} s‘‘
  | S i ⇒ head s ::: D@{i,k} s‘
  end
where "D@{ i , k } s" := (Sdrop i k s).
```

Dropping combined with summing:

```
Definition Ssigma (i k : nat) (s : Stream Z) : Stream Z :=
  ΣD@{i,k} s.
Notation "Σ@{ i , k } s" := (Ssigma i k s).
```

# Formalizing Moessner's Theorem

Niqui & Rutten formalize Moessner's Theorem as:

$$\Sigma^1_2\,\Sigma^2_3\cdots\Sigma^n_{n+1}\,\overline{1} = \mathtt{nats}^{\langle n\rangle}$$

Informally, this works because:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | ~~1~~ | 1 | 1 | 1 | 1 | ~~1~~ | 1 | 1 | 1 | 1 | ~~1~~ |
| 1 | 2 | 3 | ~~4~~ | | 5 | 6 | 7 | ~~8~~ | | 9 | 10 | 11 | ~~12~~ | |
| 1 | 3 | ~~6~~ | | | 11 | 17 | ~~24~~ | | | 33 | 43 | ~~54~~ | | |
| 1 | ~~4~~ | | | | 15 | ~~32~~ | | | | 65 | ~~108~~ | | | |
| 1 | | | | | 16 | | | | | 81 | | | | |
| $1^4$ | | | | | $2^4$ | | | | | $3^4$ | | | | |

# Formalizing Moessner's Theorem in Coq

Niqui & Rutten formalize Moessner's Theorem as:

$$\Sigma_2^1\, \Sigma_3^2 \cdots \Sigma_{n+1}^n\, \overline{1} = \mathtt{nats}^{\langle n \rangle}$$

In Coq this becomes:

```
Fixpoint Ssigmas (i k n : nat) (s : Stream Z) : Stream Z :=
  match n with
  | O ⇒ Σ @{i,k} s
  | S n ⇒ Σ @{i,k} Σ @{S i,S k,n} s
  end
where "Σ @{ i , k , n } s" := (Ssigmas i k n s).

Theorem Moessner n : Σ @{1,2,n} #1 ≡ nats ^^ S n.
```

# The coinduction principle

Niqui & Rutten use the coinduction principle:

```
Definition bisimulation {A} (R : relation (Stream A)) : Prop :=
  ∀ s t, R s t → head s = head t ∧ R (s‘) (t‘).
Lemma bisimulation_equal {A} (R : relation (Stream A)) s t :
  bisimulation R → R s t → s ≡ t.
```

Bisimilarity, and not Leibniz equality

So, we just need to find a bisimulation `R` with:

```
R (Σ @{1,2,n} #1) (nats ^^ S n)
```

# The bisimulation

```
Inductive Rn : relation (Stream Z) :=
  | Rn_sig1 n : Rn (Σ @{1,2,n} #1) (nats ^^ S n)
  | Rn_sig2 n : Rn (Σ @{0,2,n} #1) (nats ⊙ (#1 ⊕ nats) ^^ n)
  | Rn_refl s : Rn s s
  | Rn_plus s1 s2 t1 t2 :
     Rn s1 t1 → Rn s2 t2 → Rn (s1 ⊕ s2) (t1 ⊕ t2)
  | Rn_mult n s t : Rn s t → Rn (#n ⊙ s) (#n ⊙ t)
  | Rn_eq s1 s2 t1 t2 :
     s1 ≡ s2 → t1 ≡ t2 → Rn s1 t1 → Rn s2 t2.
```

The clause `Rn_sig1` is the theorem, the others are needed for `Rn` to be closed under tails

Differences with Niqui & Rutten:

- Indexes that count from 0 instead of 1
- Need to close `Rn` under bisimilarity
- Need to close `Rn` under scalar multiplication (for the generalization)

# The bisimulation

Need to show that `Rn s t` implies `head s = head t`

- By induction on the structure of `Rn`
- Straightforward proofs by induction for each case

Need to show that `Rn s t` implies `Rn (s‘)(t‘)`

- Also by induction on the structure of `Rn`
- Niqui & Rutten relate the tails to finite sums involving binomial coefficients
- These proofs require non-trivial induction loading
- Details absent in the *pen-and-paper* proof

# Long and Salié's generalization ($n = 4$)

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $a$ | $a+d$ | $a+2d$ | ~~$a+3d$~~ | $a+4d$ | $a+5d$ | $a+6d$ | ~~$a+7d$~~ | $a+8d$ |
| $a$ | $2a+d$ | ~~$3a+3d$~~ | | $4a+7d$ | $5a+12d$ | ~~$6a+18d$~~ | | $7a+26d$ |
| $a$ | ~~$3a+d$~~ | | | $7a+8d$ | ~~$12+20d$~~ | | | $19a+46d$ |
| $a$ | | | | $8a+8d$ | | | | $27a+54d$ |
| $a$ | | | | $(a+d)8$ | | | | $(a+2d)27$ |

**Theorem (Long and Salié's generalized Moessner's Theorem)**

*Starting from* $(a, d+a, 2d+a, \dots)$*, the Moessner construction gives* $(a \cdot 1^{n-1}, (d+a) \cdot 2^{n-1}, (2d+a) \cdot 3^{n-1}, \dots)$ *for any* $n \in \mathbb{N}$

# Proof of the generalization

We use streams of integers so we have:

$$\Sigma\,(a ::: \overline{d}) \equiv (a, d + a, 2d + a, \ldots) \equiv \overline{d} \odot \mathtt{nats} \oplus \overline{a - d}$$

Now the generalization is a corollary of the original theorem:

$$\begin{aligned}
&\Sigma_2^1 \cdots \Sigma_{m+2}^{m+1}\,\Sigma\,(a ::: \overline{d}) \\
\equiv\ &\Sigma_2^1 \cdots \Sigma_{m+2}^{m+1}\,(\overline{d} \odot \mathtt{nats} \oplus \overline{a - d}) \\
\equiv\ &\overline{d} \odot \Sigma_2^1 \cdots \Sigma_{m+2}^{m+1}\,\mathtt{nats} \oplus \overline{a - d} \odot \Sigma_2^1 \cdots \Sigma_{m+2}^{m+1}\,\overline{1} \\
\equiv\ &\overline{d} \odot \mathtt{nats}^{\langle 2+m \rangle} \ \oplus\ \overline{a - d} \odot \mathtt{nats}^{\langle 1+m \rangle} \\
\equiv\ &(\overline{d} \odot \mathtt{nats} \oplus \overline{a - d}) \odot \mathtt{nats}^{\langle 1+m \rangle} \\
\equiv\ &\Sigma\,(a ::: \overline{d}) \odot \mathtt{nats}^{\langle 1+m \rangle}
\end{aligned}$$

# Wiedijk's the *De Bruijn factor* of our proof

| | LaTeX | Coq |
|---|---|---|
| Lines of text | 882 | 758 |
| Compressed size (gzip) | 6272 bytes | 6409 bytes |

The *De Bruijn factor*

$$\texttt{moessner\_all.tex.gz} \times 1.02 = \texttt{moessner\_all.v.gz}$$

The typical De Bruijn factor for formalization of mathematics is 4

# Conclusions

Coq's support for coinduction seems different from the textbook approach, . . . but only at first sight

- Standard reasoning principles can easily be proven
- Setoid and ring support help a lot
- Most definitions are accepted without modifications
- Coq proofs are relatively short

No factual errors in Niqui & Rutten's paper

- They did a good job on presenting definitions and lemmas
- Most proofs were hidden under the carpet

Non-trivial proofs by coinduction can be done in Coq

# Questions

Sources: `http://github.com/robbertkrebbers/moessner/`