Formal marginalia in computability theory

Bjørn Kjos-Hanssen

—

ASL Special Session on Computability Theory

May 15, 2024, 3-3:20pm
ASL Annual Meeting, Iowa State University

Abstract

Formal marginalia are formal proofs of small parts of a mathematical theorem or publication.

Abstract

Formal marginalia are formal proofs of small parts of a mathematical theorem or publication.Research in computability theory makes extensive use of Church's thesis, making full formalization laborious.

Abstract

Formal marginalia are formal proofs of small parts of a mathematical theorem or publication.Research in computability theory makes extensive use of Church's thesis, making full formalization laborious. I present some examples of formal marginalia in `Lean` for two recent papers:

- my paper *A tractable case of the Turing automorphism problem*, 2024
- Kenneth Gill, *Probabilistic automatic complexity of finite strings*, 2024

Turing automorphism problem

- Does a nontrivial automorphism of D_T exist? (How about D_{tt}?)

Turing automorphism problem

- Does a nontrivial automorphism of D_T exist? (How about D_{tt}?)
- Does a "simple" nontrivial automorphism exist such as $A \mapsto \overline{A}$?

My claims

Announced	Published	Result
2014	2018	no bijection of ω induces a nontrivial automorphism of D_T
2019	2024	no bi-uniformly E_0-invariant Cantor homeomorphism induces a nontrivial automorphism of D_T[1]

[1] It now seems that the claim should be D_{tt} not D_T.

Key lemma for 2019/2024 result

Key lemma for 2019/2024 result

Lemma

If $\Theta : 2^\omega \to 2^\omega$ *is a homeomorphism and* $S^*A(n) = A(n+1)$, *and* $\Theta \circ S^* \circ \Theta^{-1}$ *is computable, then* Θ *is computable.*

Key lemma for 2019/2024 result

Lemma

If $\Theta : 2^\omega \to 2^\omega$ is a homeomorphism and $S^* A(n) = A(n+1)$, and $\Theta \circ S^* \circ \Theta^{-1}$ is computable, then Θ is computable.

- Proof idea: if $\Theta \circ S^* = \Phi \circ \Theta$ and $G(A) = \Theta^A(0)$ then $\Theta^A(n) = G(\Phi^n A)$.

Key lemma for 2019/2024 result

Lemma

If $\Theta : 2^\omega \to 2^\omega$ is a homeomorphism and $S^* A(n) = A(n+1)$, and $\Theta \circ S^* \circ \Theta^{-1}$ is computable, then Θ is computable.

- Proof idea: if $\Theta \circ S^* = \Phi \circ \Theta$ and $G(A) = \Theta^A(0)$ then $\Theta^A(n) = G(\Phi^n A)$.
- "Ergodic" interpretation

Another key idea

- A function $F : 2^\omega \to 2^\omega$ is E_0-invariant if $A =^* B \implies F(A) =^* F(B)$.

Another key idea

- A function $F : 2^\omega \to 2^\omega$ is E_0-invariant if $A =^* B \implies F(A) =^* F(B)$.
- Define *uniform* E_0-invariance as well.

Another key idea

- A function $F : 2^\omega \to 2^\omega$ is E_0-invariant if $A =^* B \implies F(A) =^* F(B)$.
- Define *uniform* E_0-invariance as well.
- Need this to extend claims using Baire Category from $[\sigma]$ to 2^ω.

New claims

Let $\sigma \searrow X$ be X with the first few bits replaced by σ.

New claims

Let $\sigma \searrow X$ be X with the first few bits replaced by σ.
The proof in 2019/2024 can be generalized from uniformly E_0-invariant functions to *sea-reducible* functions.

Definition

$F : 2^\omega \to 2^\omega$ is sea-reducible if for each σ there is an e such that for all X,

$$F(X) = [e]^{F(\sigma \searrow X) \oplus X}.$$

Here $[e]$ is the eth truth-table functional. "Sea-reducible" calls to mind both the south-east arrow ($\searrow$) and a ship at sea that floats from one $[\sigma]$ to the next.

The sea-reducible functions include the *tt-uniform* automorphisms, i.e., those induced by functions $F : 2^\omega \to 2^\omega$ such that

$$F([a]^X) = [f(a)]^{F(X)}$$

for some computable f.

Proof.

Indeed, given a string τ let $[a_\tau]^X = \tau \searrow X$. Now, given σ and X, let $\tau = X \upharpoonright |\sigma|$. So $[a_\tau]^{\sigma \searrow X} = X$, and

$$F(X) = F([a_\tau]^{\sigma \searrow X}) = [f(a_\tau)]^{F(\sigma \searrow X)}$$

Then we can let $[e]^{Y \oplus X} = [f(a_\tau)]^Y$. $\square$

No nontrivial automorphism of the truth-table degrees is induced by an automorphism of the Scott domain $2^{\leq\omega}$.

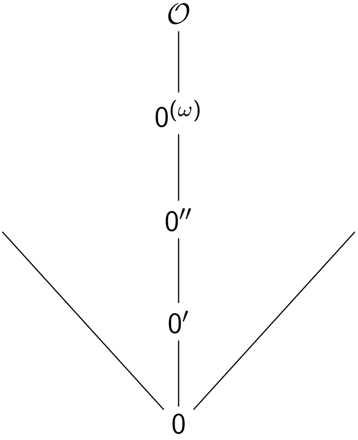


Figure: The truth-table degrees.

Scott domain

Consists of strings $2^{<\omega}$ and reals 2^{ω}.

Scott domain

Consists of strings $2^{<\omega}$ and reals 2^{ω}. These are ordered by extension: $\sigma \prec X$, $\sigma \preceq \tau$.

Scott domain

Consists of strings $2^{<\omega}$ and reals 2^{ω}. These are ordered by extension: $\sigma \prec X$, $\sigma \preceq \tau$. Automorphisms are given by bijections of ω together with bijections of 2 (one for each $n \in \omega$).

Scott domain

Consists of strings $2^{<\omega}$ and reals 2^{ω}. These are ordered by extension: $\sigma \prec X$, $\sigma \preceq \tau$. Automorphisms are given by bijections of ω together with bijections of 2 (one for each $n \in \omega$).

Example

Complementation is given by the identity function on ω and the nontrivial bijection of 2 at each n.

$$\begin{tikzcd}
 & K \oplus \overline{K} & \\
K \arrow[ur] \arrow[rr, phantom, "\longleftrightarrow_{\pi}"] & & \overline{K} \arrow[ul]
\end{tikzcd}$$

Example

Complementation is

- a nontrivial automorphism of the m-degrees,

$$
\begin{tikzcd}
 & K \oplus \overline{K} & \\
K \arrow[ur] \arrow[rr, phantom, "\longleftrightarrow_{\pi}"] & & \overline{K} \arrow[ul]
\end{tikzcd}
$$

Example

Complementation is

- a nontrivial automorphism of the m-degrees,
- a nontrivial automorphism of the p-degrees,

$$\begin{tikzcd} & K \oplus \overline{K} & \\ K \arrow[ur] \arrow[rr, phantom, "\longleftrightarrow_{\pi}"] & & \overline{K} \arrow[ul] \end{tikzcd}$$

Example

Complementation is

- a nontrivial automorphism of the m-degrees,
- a nontrivial automorphism of the p-degrees,
- tt-uniform,

$$
\begin{tikzcd}
 & K \oplus \overline{K} & \\
K \arrow[ur] \arrow[rr, phantom, "\longleftrightarrow_{\pi}"] & & \overline{K} \arrow[ul]
\end{tikzcd}
$$

Example

Complementation is

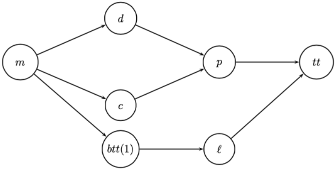
- a nontrivial automorphism of the m-degrees,
- a nontrivial automorphism of the p-degrees,
- tt-uniform,
- an automorphism of the Scott domain,

$$\begin{tikzcd}
 & K \oplus \overline{K} & \\
K \arrow[ur] & \longleftrightarrow_{\pi} & \overline{K} \arrow[ul]
\end{tikzcd}$$

Example

Complementation is

- a nontrivial automorphism of the *m*-degrees,
- a nontrivial automorphism of the *p*-degrees,
- tt-uniform,
- an automorphism of the Scott domain,
- an isomorphism of the *d*- and *c*-degrees.



Uniformly E_0-invariant implies sea-invariant

Proof.

Suppose F is uniformly E_0-invariant. Let σ be given and $a = |\sigma|$. By uniform invariance we have a b. The action of F on $n < b$ for all X is given by a finite database which is incorporated into $[e]$. For $n \geq b$, for all X, $F(X)(n) = F(\sigma \searrow X)(n)$ so we let

$$[e]^{Y \oplus X}(n) = \begin{cases} Y(n) & n \geq b, \\ F(X)(n) & n < b. \end{cases}$$

$\square$

In fact this is a bit stronger in that only a fixed finite amount of X needs to be queried.

Example

A function that is sea-reducible but not E_0-invariant. Consider

$$F(X)(n) = \begin{cases} X(0), & n = 0 \\ X(n) + X(0) + Z(n), & n > 0 \end{cases}$$

for a fixed real Z. Here $F(X)$ and $F(\sigma \searrow X)$ differ on almost all inputs if $X(0) \neq \sigma(0)$, but we can use

$$[e]^{Y \oplus X}(n) = \begin{cases} \begin{cases} Y(n) + (\sigma(0) + X(0)), & n \geq |\sigma|, \\ F(X)(n), & n < |\sigma|. \end{cases} & |\sigma| > 0 \\ Y(n) & |\sigma| = 0 \end{cases}$$

(For any F, if $\sigma = \emptyset$ we can just take $[e]^{Y \oplus X} = Y$.)

Computability theory in Lean

- A library with Turing machines, tape heads etc. exists

Computability theory in Lean

- A library with Turing machines, tape heads etc. exists
- “By Church’s thesis” — too tedious

Computability theory in Lean

- A library with Turing machines, tape heads etc. exists
- "By Church's thesis" — too tedious
- A positive notion of computability — `[Decidable]` — allows to prove computability but not to prove noncomputability.

Computability theory in Lean

- A library with Turing machines, tape heads etc. exists
- “By Church’s thesis” — too tedious
- A positive notion of computability — `[Decidable]` — allows to prove computability but not to prove noncomputability.
- `by decide` and `#eval` extends proving computability to actually computing

Forcing in Cantor space in Lean

Forcing in Cantor space in Lean

- In computability theory papers a condition is often $\sigma \in 2^{<\omega}$ (domain $|\sigma|$)

Forcing in Cantor space in Lean

- In computability theory papers a condition is often $\sigma \in 2^{<\omega}$ (domain $|\sigma|$)
- In `Lean` a condition is a function $U : \mathbb{N} \to \mathcal{P}(2)$ where $U(n) = 2$ for all but finitely many n. This makes conditions infinite objects.

Forcing in Cantor space in Lean

- In computability theory papers a condition is often $\sigma \in 2^{<\omega}$ (domain $|\sigma|$)
- In `Lean` a condition is a function $U : \mathbb{N} \to \mathcal{P}(2)$ where $U(n) = 2$ for all but finitely many n. This makes conditions infinite objects.

A compromise:

Forcing in Cantor space in Lean

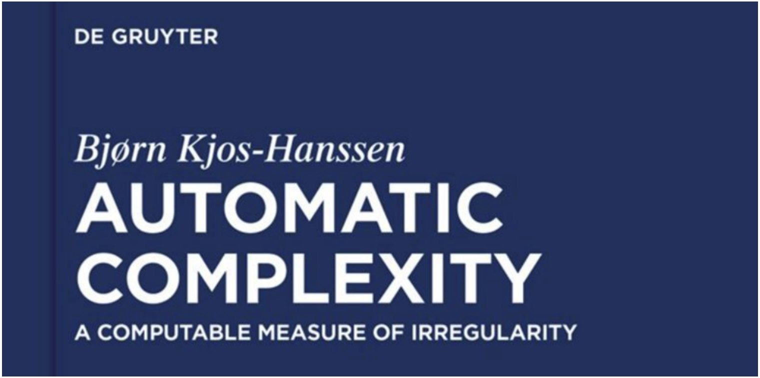
- In computability theory papers a condition is often $\sigma \in 2^{<\omega}$ (domain $|\sigma|$)
- In `Lean` a condition is a function $U : \mathbb{N} \to \mathcal{P}(2)$ where $U(n) = 2$ for all but finitely many n. This makes conditions infinite objects.

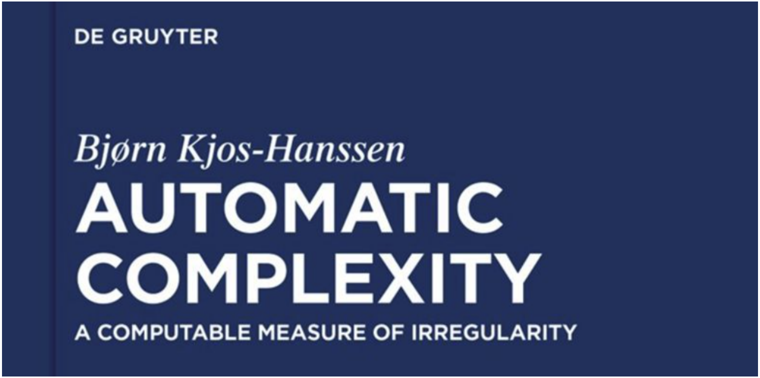
A compromise:

```
def condition {a : Type} := Σ I : Finset ℕ, I → Set a
```

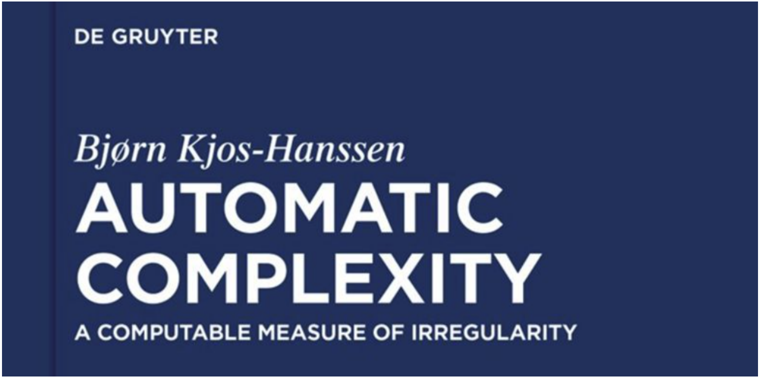
Sample theorem

```
theorem bounded_use_principle₀ {α : Type} [TopologicalSpace α] [DiscreteTopology α]
[CompactSpace (ℕ → α)] (F : (ℕ → α) → (ℕ → α)) (hF : Continuous F) (n:ℕ):
∃ (t : Finset { τ : condition // ∀ Y₁ Y₂ : (ℕ → α), τ ≼ Y₁ → τ ≼ Y₂ → F Y₁ n = F Y₂ n}),
  (Set.univ : Set (ℕ → α))
  ⊆ ⋃ σ ∈ t, {X : (ℕ → α) | σ ≼ X}
```

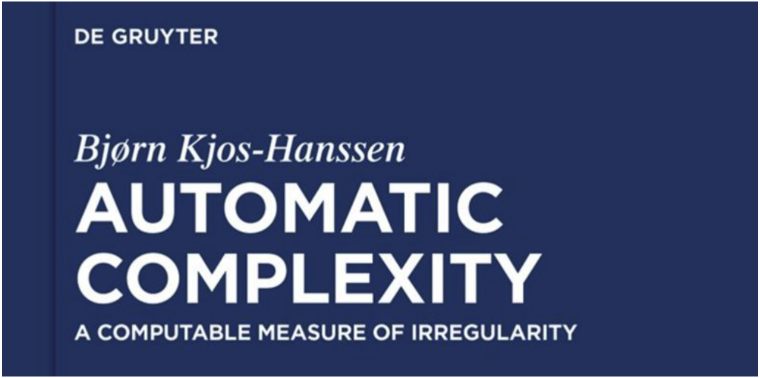

DE GRUYTER
Bjørn Kjos-Hanssen
AUTOMATIC
COMPLEXITY
A COMPUTABLE MEASURE OF IRREGULARITY



- The automatic complexity $A(x)$ of $x \in \{0,1\}^*$ is the minimum number of states of a DFA M such that $L(M) \cap \{0,1\}^{|x|} = \{x\}$. (Shallit, Wang 2001)



- The automatic complexity $A(x)$ of $x \in \{0,1\}^*$ is the minimum number of states of a DFA M such that $L(M) \cap \{0,1\}^{|x|} = \{x\}$. (Shallit, Wang 2001)
- I worked on the nondeterministic version $A_N(x)$ since 2009 culminating in a book (2024).



- The automatic complexity $A(x)$ of $x \in \{0,1\}^*$ is the minimum number of states of a DFA M such that $L(M) \cap \{0,1\}^{|x|} = \{x\}$. (Shallit, Wang 2001)
- I worked on the nondeterministic version $A_N(x)$ since 2009 culminating in a book (2024).
- The probabilistic version $A_P(x)$ uses probabilistic DFAs. (Gill, 2024)

Formalizing Gill's paper in Lean

One step of computation is matrix multiplication:

```
def step {n q:ℕ} (w : Fin n → Fin 2)
    (A : Fin 2 → Fin q → Fin q → ℚ) (i : Fin n)
            (M : Fin q → Fin q → ℚ) :
                (Fin q → Fin q → ℚ) :=
  Matrix.mul M (A (w i))
```

Formalizing Gill's paper in Lean

Lean's `Fin.foldr` allows us to consider sequences of multiplications corresponding to a word:

Formalizing Gill's paper in Lean

Lean's `Fin.foldr` allows us to consider sequences of multiplications corresponding to a word:

```
def fold_step {n q:ℕ} (w : Fin n → Fin 2) (A : Fin 2 →
    Fin q → Fin q → ℚ) : Fin q → Fin q → ℚ
  := Fin.foldr n (step w A) (fun i j ↦ ite (i=j) 1 0)

def acceptance_probability  {n q:ℕ} (w : Fin n → Fin 2)
  (A : Fin 2 → Fin q → Fin q → ℚ) (q₀ q₁ : Fin q) : ℚ :
    = by
  let Q := Matrix.mul (fold_step w A) (fun i : Fin q ↦
    fun j : Fin 1 ↦ ite (i=q₀) 1 0)
  let R := Matrix.mul (fun i : Fin 1 ↦ fun j : Fin q ↦
    ite (j=q₁) 1 0) Q
  exact R 0 0
```

Formalizing Gill's paper in Lean

Lean's `Fin.foldr` allows us to consider sequences of multiplications corresponding to a word:

```
def fold_step {n q:ℕ} (w : Fin n → Fin 2) (A : Fin 2 →
    Fin q → Fin q → ℚ) : Fin q → Fin q → ℚ
  := Fin.foldr n (step w A) (fun i j ↦ ite (i=j) 1 0)

def acceptance_probability  {n q:ℕ} (w : Fin n → Fin 2)
  (A : Fin 2 → Fin q → Fin q → ℚ) (q₀ q₁ : Fin q) : ℚ :
    = by
  let Q := Matrix.mul (fold_step w A) (fun i : Fin q ↦
    fun j : Fin 1 ↦ ite (i=q₀) 1 0)
  let R := Matrix.mul (fun i : Fin 1 ↦ fun j : Fin q ↦
    ite (j=q₁) 1 0) Q
  exact R 0 0
```

We can now both prove things and use Lean as a calculator.