Is forcing a computable process?

Julia Kameryn Williams
they/she

Bard College at Simon's Rock

Connecticut Logic Seminar
2024 Nov 12

Joint work with Joel David Hamkins (Notre Dame) and Russell Miller (CUNY).

Hey, you got set theory in my computable structure theory!
Hey, you got computable structure theory in my set theory!

An intuitive sketch of forcing

We want to expand our universe to add a new object G.

- A forcing poset $\mathbb{P}$ consists of possible approximations to G which. The poset grows downward, with stronger conditions being lower.
- The new object is a generic filter $\subseteq \mathbb{P}$.
  - G is upward-closed, because if p is an approximation of G then so is any weaker condition.
  - G is directed, because the approximations must be compatible.
  - G is generic: it meets every dense $D \subseteq \mathbb{P}$ (D gets below any condition).

Genericity forces $G \notin \mathrm{V}$ for nontrivial $\mathbb{P}$.

Force CH by adding an ω_1-sequence of all reals.

- Use the poset $\mathrm{Add}(\omega_1, 1)$ consisting of functions $\alpha \to 2$ for countable α, ordered by extension: stronger conditions are longer binary sequences.
- G will be an ω_1-length binary sequence, with every real coded at some point.
  - Directedness is trivial since $\mathrm{Add}(\omega_1, 1)$ is a tree: G will be a branch.
  - Genericity ensures every real is coded: for every $x : \omega \to 2$ it is dense to extend a node to code x.
  - A closure property ensures no new reals were added.

An intuitive sketch of forcing

It's not enough to add just one new object G, you need to add the rest of the forcing extension $\mathrm{V}[G]$.

- Recursively define $\mathbb{P}$-names, which describe objects in the larger universe.
- The generic G says how to interpret names: $\dot{x}^G$ is the interpretation of $\dot{x}$.
- There are definable forcing relations $p \Vdash \varphi(\dot{x}, \ldots)$ which control the behavior of $\mathrm{V}[G]$:

$$\mathrm{V}[G] \models \varphi(\dot{x}^G, \ldots) \Leftrightarrow \exists p \in G \ p \Vdash \varphi(\dot{x}, \ldots)$$

- Can check that forcing always preserves the axioms of ZFC.
- Use properties of $\mathbb{P}$ to prove more detailed facts about how V and $\mathrm{V}[G]$ relate.

An intuitive sketch of forcing

Three main parts of forcing:

- Getting a generic G;
- Interpreting the names to build the forcing extension;
- Using the forcing relations to determine satisfaction in the forcing extension.

An intuitive sketch of forcing

Three main parts of forcing:

- Getting a generic G;
- Interpreting the names to build the forcing extension;
- Using the forcing relations to determine satisfaction in the forcing extension.

Important! While $G \notin \mathrm{V}$, everything can be described within the ground model. You don't have to be a set-theoretic multiversist to make sense of forcing.

Forcing is obviously not a computable process

Forcing is obviously not a computable process

- Any computable process takes place entirely in V, so it's not possible to produce G.
- Indeed, computation is absolute, so anything we could do in $\mathrm{V}[G]$ must already be in the ground model.

Forcing is obviously not a computable process

- Any computable process takes place entirely in V, so it's not possible to produce G.
- Indeed, computation is absolute, so anything we could do in $\mathrm{V}[G]$ must already be in the ground model.
- The $\mathbb{P}$-names and forcing relations are defined by transfinite recursion, and have no hope of being computable.

Forcing is obviously not a computable process

- Any computable process takes place entirely in V, so it's not possible to produce G.
- Indeed, computation is absolute, so anything we could do in $\mathrm{V}[G]$ must already be in the ground model.
- The $\mathbb{P}$-names and forcing relations are defined by transfinite recursion, and have no hope of being computable.

If you know about the boolean algebra approach to forcing, the same problems recur.

- Building a complete boolean algebra $\mathbb{B}$ from a poset $\mathbb{P}$ and building a boolean topos $\mathrm{V}^{\mathbb{B}}$ from $\mathbb{B}$ are both infinitary processes.

For the titular question to be nontrivial we must mean something else.

Countable models of set theory

- By the Löwenheim–Skolem theorem, there are countable models of set theory.
- If M is countable and $\mathbb{P} \in M$ then $\mathbb{P}$ is countable and so the Rasiowa–Sikorski lemma implies generics for $\mathbb{P}$ exist

Countable models of set theory

- By the Löwenheim–Skolem theorem, there are countable models of set theory.
- If M is countable and $\mathbb{P} \in M$ then $\mathbb{P}$ is countable and so the Rasiowa–Sikorski lemma implies generics for $\mathbb{P}$ exist
- A countable model M of set theory can be thought of as ω equipped with a binary relation $\in^M$.
- This is an appropriate setting for computable structure theory.

Countable models of set theory

- By the Löwenheim–Skolem theorem, there are countable models of set theory.
- If M is countable and $\mathbb{P} \in M$ then $\mathbb{P}$ is countable and so the Rasiowa–Sikorski lemma implies generics for $\mathbb{P}$ exist
- A countable model M of set theory can be thought of as ω equipped with a binary relation $\in^M$.
- This is an appropriate setting for computable structure theory.

Can formulate questions.
Given $M = (\omega, \in^M)$ and a poset $\mathbb{P} \in M$:

- Can we compute a generic G?
- Can we compute a representation of the forcing extension $M[G]$?
- Can we compute the elementary diagram of $M[G]$?

Countable models of set theory

- By the Löwenheim–Skolem theorem, there are countable models of set theory.
- If M is countable and $\mathbb{P} \in M$ then $\mathbb{P}$ is countable and so the Rasiowa–Sikorski lemma implies generics for $\mathbb{P}$ exist
- A countable model M of set theory can be thought of as ω equipped with a binary relation $\in^M$.
- This is an appropriate setting for computable structure theory.

Can formulate questions.
Given $M = (\omega, \in^M)$ and a poset $\mathbb{P} \in M$:

- Can we compute a generic G?
- Can we compute a representation of the forcing extension $M[G]$?
- Can we compute the elementary diagram of $M[G]$?

Warning! No model of set theory can be computable, so we can only ask about computability relative to an oracle.

Computing a generic G

Theorem (Hamkins–Miller–W.)

Given the atomic diagram of $M = (\omega, \in^M)$ *and a poset* $\mathbb{P} \in M$ *you can compute a generic* G *for* $\mathbb{P}$*, given parameters.*

Computing a generic G

Theorem (Hamkins–Miller–W.)

Given the atomic diagram of $M = (\omega, \in^M)$ and a poset $\mathbb{P} \in M$ you can compute a generic G for $\mathbb{P}$, given parameters.

- The atomic diagram is simply the relation $\in^M$.
- Literally, $\mathbb{P}$ is an integer, not a set of conditions. Its extension is $\mathbb{P}^\in = \{n \in \omega : n \in^M \mathbb{P}\}$, and by computing G I mean as a subset of $\mathbb{P}^\in$.

Computing a generic G

Theorem (Hamkins–Miller–W.)

Given the atomic diagram of $M = (\omega, \in^M)$ and a poset $\mathbb{P} \in M$ you can compute a generic G for $\mathbb{P}$, given parameters.

- The atomic diagram is simply the relation $\in^M$.
- Literally, $\mathbb{P}$ is an integer, not a set of conditions. Its extension is $\mathbb{P}^\in = \{n \in \omega : n \in^M \mathbb{P}\}$, and by computing G I mean as a subset of $\mathbb{P}^\in$.

Proof: The usual proof of the Rasiowa–Sikorski is effective. □

Some actual details

Fix a bunch of integers: $\mathbb{P}$, $\leq_{\mathbb{P}}$, $\not\leq_{\mathbb{P}}$, $\perp_{\mathbb{P}}$, $\mathcal{D}$ the collection of dense subsets of $\mathbb{P}$.

Some actual details

Fix a bunch of integers: $\mathbb{P}$, $\leq_{\mathbb{P}}$, $\not\leq_{\mathbb{P}}$, $\perp_{\mathbb{P}}$, $\mathcal{D}$ the collection of dense subsets of $\mathbb{P}$. Can computably enumerate

$$p_0, \ldots, p_n, \ldots \qquad \text{all } p \in^M \mathbb{P}$$

$$d_0, \ldots, d_n, \ldots \qquad \text{all } d \in^M \mathcal{D}$$

Now computably enumerate a descending sequence $q_0 \geq_{\mathbb{P}} q_1 \geq_{\mathbb{P}} \cdots$

- $q_0 = p_0$;
- Given q_n, step through the p_i to find q with $\mathrm{op}(q, q_n) \in^M \leq_{\mathbb{P}}$ and $q \in^M d_n$. Set the first q you find to be q_{n+1}.

Some actual details

Fix a bunch of integers: $\mathbb{P}$, $\leq_{\mathbb{P}}$, $\not\leq_{\mathbb{P}}$, $\perp_{\mathbb{P}}$, $\mathcal{D}$ the collection of dense subsets of $\mathbb{P}$. Can computably enumerate

$$\begin{array}{ll} p_0, \ldots, p_n, \ldots & \text{all } p \in^M \mathbb{P} \\ d_0, \ldots, d_n, \ldots & \text{all } d \in^M \mathcal{D} \end{array}$$

Now computably enumerate a descending sequence $q_0 \geq_{\mathbb{P}} q_1 \geq_{\mathbb{P}} \cdots$

- $q_0 = p_0$;
- Given q_n, step through the p_i to find q with $\mathrm{op}(q, q_n) \in^M \leq_{\mathbb{P}}$ and $q \in^M d_n$. Set the first q you find to be q_{n+1}.

Then $G = \{p \in \omega : p \in^M \mathbb{P} \text{ and } \mathrm{op}(q_n, p) \in^M \leq_{\mathbb{P}} \text{ for some } q_n\}$ is computably enumerable.

But $\omega \setminus G = \{p \in \mathbb{N} : \neg(p \in^M \mathbb{P}) \text{ or } \mathrm{op}(q_n, p) \in^M \perp_{\mathbb{P}} \text{ for some } q_n\}$ is also computably enumerable. So G is computable. □

What's up with that non-uniformity?

You may not like that our algorithm required us to fix a bunch of integers. This isn't a problem for what is computable (from the atomic diagram); we may not know which of the many Turing machines happens to use the right integers, but one of them will.

What's up with that non-uniformity?

You may not like that our algorithm required us to fix a bunch of integers. This isn't a problem for what is computable (from the atomic diagram); we may not know which of the many Turing machines happens to use the right integers, but one of them will.

But this suggests there may be some non-uniformity to the computation...

We'll come back to this worry at the end.

What can we compute from the atomic diagram?

The atomic diagram is very weak, and not a sensible notion of the basic structure of a model of set theory.

Theorem (Hamkins–Miller–W.)

Let X be a subset of a model M of set theory. TFAE

- *There is a single c.e. operator which takes the atomic diagram of a presentation of M and outputs the copy of X for that presentation. (X is uniformly r.i.c.e. in the atomic diagram.)*
- *Membership $a \in X$ is witnessed by a finite pattern of $\in$ in the transitive closure of a, with the list of patterns c.e. in the atomic diagram.*

What can we compute from the atomic diagram?

The atomic diagram is very weak, and not a sensible notion of the basic structure of a model of set theory.

All of the following predicates are not uniformly r.i.c.e. in the atomic diagram.

- $x = \emptyset$
- $x \subseteq y$
- x is an ordered pair
- x is a function
- x is an ordinal
- $x = \omega$

Theorem (Hamkins–Miller–W.)

Let X be a subset of a model M of set theory. TFAE

- *There is a single c.e. operator which takes the atomic diagram of a presentation of M and outputs the copy of X for that presentation. (X is uniformly r.i.c.e. in the atomic diagram.)*
- *Membership $a \in X$ is witnessed by a finite pattern of $\in$ in the transitive closure of a, with the list of patterns c.e. in the atomic diagram.*

The Lévy hierarchy

In set theory the natural hierarchy for formulae is the Lévy hierarchy:

- The Δ_0 formulae are those whose quantifiers are all bounded: $\forall x \in y$ or $\exists x \in y$.
- Inductively build up the Σ_n and Π_n formulae by adding blocks of unbounded quantifiers.
- Δ_n means both Σ_n and Π_n.
- For $M = (\omega, \in^M)$ a model of set theory its Δ_0-diagram is the restriction of the elementary diagram to the Δ_0 formulae.
- And similar for other levels of the hierarchy.

The Lévy hierarchy

In set theory the natural hierarchy for formulae is the Lévy hierarchy:

- The Δ_0 formulae are those whose quantifiers are all bounded: $\forall x \in y$ or $\exists x \in y$.
- Inductively build up the Σ_n and Π_n formulae by adding blocks of unbounded quantifiers.
- Δ_n means both Σ_n and Π_n.
- For $M = (\omega, \in^M)$ a model of set theory its Δ_0-diagram is the restriction of the elementary diagram to the Δ_0 formulae.
- And similar for other levels of the hierarchy.

- Σ_1 properties are upward absolute: they are preserved by going up to an end-extension (an extension that doesn't add new elements to old sets).
- Π_1 properties are downward absolute.

The Lévy hierarchy

In set theory the natural hierarchy for formulae is the Lévy hierarchy:

- The Δ_0 formulae are those whose quantifiers are all bounded: $\forall x \in y$ or $\exists x \in y$.
- Inductively build up the Σ_n and Π_n formulae by adding blocks of unbounded quantifiers.
- Δ_n means both Σ_n and Π_n.
- For $M = (\omega, \in^M)$ a model of set theory its Δ_0-diagram is the restriction of the elementary diagram to the Δ_0 formulae.
- And similar for other levels of the hierarchy.

- Σ_1 properties are upward absolute: they are preserved by going up to an end-extension (an extension that doesn't add new elements to old sets).
- Π_1 properties are downward absolute.
- For each n the Σ_n-satisfaction relation is Σ_n-definable.

The Lévy diagram

- Over ω the arithmetical hierarchy of formulae is built by taking bounded quantifiers to be $\forall x \leq y$ and $\exists x \leq y$.
- Lévy Δ_0 doesn't line up with arithmetical Δ_0 over $M = (\omega, \in^M)$, as the set-theoretic bounded quantifiers are infinitary.

The Lévy diagram

- Over ω the arithmetical hierarchy of formulae is built by taking bounded quantifiers to be $\forall x \le y$ and $\exists x \le y$.
- Lévy Δ_0 doesn't line up with arithmetical Δ_0 over $M = (\omega, \in^M)$, as the set-theoretic bounded quantifiers are infinitary.

But we can make them line up by using a different diagram.

- The Lévy diagram for $M = (\omega, \in^M)$ is the atomic diagram in the signature with a relation symbol for every Lévy Δ_0 relation over M.
- Arithmetic Σ_n over the Lévy diagram is equivalent to Lévy Σ_n over the $\in$-atomic diagram.

Computing the forcing extension $M[G]$

Theorem (Hamkins–Miller–W.)

Take the Δ_0-diagram for $M = (\omega, \in^M)$ as an oracle fix a poset $\mathbb{P} \in M$. Then we can computably produce G an M-generic for $\mathbb{P}$ and a copy of $M[G]$.

More precisely, we can compute a relation $\in^G \subseteq \omega^2$ so that $M[G] \cong (\omega, \in M[^G])$ and we can compute the canonical embedding $M \hookrightarrow M[G]$.

- We already know we can compute G, and we don't need parameters because they can be computed from the Δ_0-diagram.

Computing the forcing extension $M[G]$

- We already know we can compute G, and we don't need parameters because they can be computed from the Δ_0-diagram.

- The $\mathbb{P}$-names are sets whose elements are of the form $(\dot{y}, p)$ where $\dot{y}$ is a $\mathbb{P}$-name and $p \in \mathbb{P}$.
- This is a definition by transfinite recursion, and each step in the recursion is Δ_0 so the class of $\mathbb{P}$-names is Δ_1.
- The interpretation of $\dot{x}$ by G is $\dot{x}^G = \{\dot{y}^G : \exists p \in G \ (\dot{y}, p) \in \dot{x}\}$.

Computing the forcing extension $M[G]$

- We already know we can compute G, and we don't need parameters because they can be computed from the Δ_0-diagram.
- Because the class of $\mathbb{P}$-names is Δ_1 it is computable from the Δ_0-diagram.
- Similarly we can compute from G and the Δ_0-diagram the interpretations of the names by G.

- The $\mathbb{P}$-names are sets whose elements are of the form
  $(\dot{y}, p)$ where $\dot{y}$ is a $\mathbb{P}$-name and $p \in \mathbb{P}$.
- This is a definition by transfinite recursion, and each step in the recursion is Δ_0 so the class of $\mathbb{P}$-names is Δ_1.
- The interpretation of $\dot{x}$ by G is
  $\dot{x}^G = \{\dot{y}^G : \exists p \in G \ (\dot{y}, p) \in \dot{x}\}$.

Computing the forcing extension $M[G]$

- We already know we can compute G, and we don't need parameters because they can be computed from the Δ_0-diagram.
- Because the class of $\mathbb{P}$-names is Δ_1 it is computable from the Δ_0-diagram.
- Similarly we can compute from G and the Δ_0-diagram the interpretations of the names by G.

- The $\mathbb{P}$-names are sets whose elements are of the form $(\dot{y}, p)$ where $\dot{y}$ is a $\mathbb{P}$-name and $p \in \mathbb{P}$.
- This is a definition by transfinite recursion, and each step in the recursion is Δ_0 so the class of $\mathbb{P}$-names is Δ_1.
- The interpretation of $\dot{x}$ by G is $\dot{x}^G = \{\dot{y}^G : \exists p \in G\ (\dot{y}, p) \in \dot{x}\}$.
- The following relations are Δ_1 in G:
  $\dot{x} =_G \dot{y}$ iff $\exists p \in G\ p \Vdash \dot{x} = \dot{y}$
  $\dot{x} \in_G \dot{y}$ iff $\exists p \in G\ p \Vdash \dot{x} \in \dot{y}$

Computing the forcing extension $M[G]$

- We already know we can compute G, and we don't need parameters because they can be computed from the Δ_0-diagram.
- Because the class of $\mathbb{P}$-names is Δ_1 it is computable from the Δ_0-diagram.
- Similarly we can compute from G and the Δ_0-diagram the interpretations of the names by G.
- We can compute the $=_G$ equivalence classes.
- Compute a copy of $M[G]$ by picking the least integer in each $=_G$ class.
- Compute $\in^{M[G]}$ by computing $\in_G$.

- The $\mathbb{P}$-names are sets whose elements are of the form $(\dot{y}, p)$ where $\dot{y}$ is a $\mathbb{P}$-name and $p \in \mathbb{P}$.
- This is a definition by transfinite recursion, and each step in the recursion is Δ_0 so the class of $\mathbb{P}$-names is Δ_1.
- The interpretation of $\dot{x}$ by G is $\dot{x}^G = \{\dot{y}^G : \exists p \in G \ (\dot{y}, p) \in \dot{x}\}$.
- The following relations are Δ_1 in G:
  $\dot{x} =_G \dot{y}$ iff $\exists p \in G \ p \Vdash \dot{x} = \dot{y}$
  $\dot{x} \in_G \dot{y}$ iff $\exists p \in G \ p \Vdash \dot{x} \in \dot{y}$

Computing the elementary diagram

Theorem (Hamkins–Miller–W.)

Suppose we have the elementary diagram of $M = (\omega, \in^M)$ *as an oracle and* $\mathbb{P} \in M$ *is a poset. Then we can computably produce* G *an* M*-generic for* $\mathbb{P}$ *and the elementary diagram of a copy of* $M[G]$*.*

Computing the elementary diagram

Theorem (Hamkins–Miller–W.)

Suppose we have the elementary diagram of $M = (\omega, \in^M)$ as an oracle and $\mathbb{P} \in M$ is a poset. Then we can computably produce G an M-generic for $\mathbb{P}$ and the elementary diagram of a copy of $M[G]$.

Proof:

- We already know we can compute a copy of $M[G]$.
- We can compute the elementary diagram of this copy because the forcing relations are in the elementary diagram of M.
- **Important!** The map $\varphi \mapsto$ "$p \Vdash \varphi$" sending a formula to the corresponding forcing relation is computable.

Computing the elementary diagram level by level

Theorem (Hamkins–Miller–W.)

Suppose we have the Σ_n-diagram of $M = (\omega, \in^M)$ as an oracle and $\mathbb{P} \in M$ is a poset. Then we can computably produce G an M-generic for $\mathbb{P}$ and the Σ_n-diagram of a copy of $M[G]$.
The same is true for the Δ_0-diagram.

Computing the elementary diagram level by level

Theorem (Hamkins–Miller–W.)

Suppose we have the Σ_n-diagram of $M = (\omega, \in^M)$ as an oracle and $\mathbb{P} \in M$ is a poset. Then we can computably produce G an M-generic for $\mathbb{P}$ and the Σ_n-diagram of a copy of $M[G]$.
The same is true for the Δ_0-diagram.

Proof: Because the forcing relations for Σ_n formulae are themselves Σ_n. □

Forcing is a computable procedure

Forcing is a computable procedure, with the level of information given as an oracle determining what we can compute about the extension.

- Given the atomic diagram for $M = (\omega, \in^M)$ and a poset $\mathbb{P} \in M$ we can compute a generic G for $\mathbb{P}$ (using parameters).
- Given the Δ_0-diagram we can moreover compute a copy of the extension $M[G]$ and its Δ_0-diagram.
- Given the Σ_n-diagram we can compute the Σ_n-diagram of the extension.
- Given the elementary diagram we can compute the elementary diagram of the extension.

So about that non-uniformity

- The construction of G proceeded by searching through the conditions in $\mathbb{P}$ and the dense subsets of $\mathbb{P}$.
- A different presentation of M will give a different order for the search, and produce a different G.
- In general, there will be $2^{\aleph_0}$ many possible G's, so the $M[G]$ can't all be the same.

So about that non-uniformity

- The construction of G proceeded by searching through the conditions in $\mathbb{P}$ and the dense subsets of $\mathbb{P}$.
- A different presentation of M will give a different order for the search, and produce a different G.
- In general, there will be $2^{\aleph_0}$ many possible G's, so the $M[G]$ can't all be the same.

Altogether this tells us there is a non-uniformity to the process.

Can we get uniformity by a different process?

Making the notion of uniformity precise: functoriality

For a structure M let $\mathrm{Iso}(M)$ denote the category of isomorphisms of M, with only isomorphisms as its morphisms.

- A process to interpret N in M gives a map $F : \mathrm{Iso}(M) \to \mathrm{Iso}(N)$.
- If F preserves isomorphisms then it is a functor.
- So asking for a uniform procedure to construct $M[G]$ from M amounts to asking for a functor $F : \mathrm{Iso}(M) \to \mathrm{Iso}(N)$.

Making the notion of uniformity precise: functoriality

For a structure M let $\mathrm{Iso}(M)$ denote the category of isomorphisms of M, with only isomorphisms as its morphisms.

- A process to interpret N in M gives a map $F : \mathrm{Iso}(M) \to \mathrm{Iso}(N)$.
- If F preserves isomorphisms then it is a functor.
- So asking for a uniform procedure to construct $M[G]$ from M amounts to asking for a functor $F : \mathrm{Iso}(M) \to \mathrm{Iso}(N)$.

As computable structure theorists we don't want just any functor.

- A functor F is computable if there is a Turing functional Φ which given info about an isomorphism $M \to M^*$ as an oracle will compute an isomorphism $M[G] \to M^*[G^*]$.

Making the notion of uniformity precise: functoriality

For a structure M let $\mathrm{Iso}(M)$ denote the category of isomorphisms of M, with only isomorphisms as its morphisms.

- A process to interpret N in M gives a map $F : \mathrm{Iso}(M) \to \mathrm{Iso}(N)$.
- If F preserves isomorphisms then it is a functor.
- So asking for a uniform procedure to construct $M[G]$ from M amounts to asking for a functor $F : \mathrm{Iso}(M) \to \mathrm{Iso}(N)$.

As computable structure theorists we don't want just any functor.

- A functor F is computable if there is a Turing functional Φ which given info about an isomorphism $M \to M^*$ as an oracle will compute an isomorphism $M[G] \to M^*[G^*]$.
- (HTMMM 2017) There is a computable functor $F : \mathrm{Iso}(M) \to \mathrm{Iso}(N)$ iff N is effectively interpretable in M.
- (HTMM 2018) If $F : \mathrm{Iso}(M) \to \mathrm{Iso}(N)$ is Baire-measurable then there is an infinitary interpretation $\mathcal{I}$ of N in M so that F is naturally isomorphic to $F_{\mathcal{I}}$.

Forcing is not a functorial process

Theorem (Hamkins–Miller–W.)

If ZFC *is consistent there is* $M \models \mathrm{ZFC}$ *so that there is no computable functor* $\mathrm{Iso}(M) \to \mathrm{Iso}(M[G])$.

Forcing is not a functorial process

Theorem (Hamkins–Miller–W.)

If ZFC *is consistent there is* $M \models \mathrm{ZFC}$ *so that there is no computable functor* $\mathrm{Iso}(M) \to \mathrm{Iso}(M[G])$.

Proof sketch: Take M with κ so that $\mathrm{V}_\kappa^M \prec M$. Inside M try to run the procedure Φ on the model V_κ^M.

You can't run the whole procedure, since M thinks V_κ^M is uncountable. But any decision is made from finite information. So M sees enough to know whether Φ decides $p \in G$ for each p. As such M has G as an element.

But V_κ^M is a rank-initial segment of M so it has all subsets of $\mathbb{P}$ in M. So G is generic for M, which is impossible for nontrivial G. □

Forcing is not a functorial process

Theorem (Hamkins–Miller–W.)

If ZFC *is consistent there is $M \models \mathrm{ZFC}$ so that there is no computable functor* $\mathrm{Iso}(M) \to \mathrm{Iso}(M[G])$.

Nonetheless for certain M we can achieve uniformity.

Theorem (Hamkins–Miller–W.)

If M is a pointwise-definable model of set theory there is a computable functor $\mathrm{Iso}(M) \to \mathrm{Iso}(M[G])$, *using the full diagram of M as its info.*

Proof sketch: Take M with κ so that $\mathrm{V}_\kappa^M \prec M$. Inside M try to run the procedure Φ on the model V_κ^M.

You can't run the whole procedure, since M thinks V_κ^M is uncountable. But any decision is made from finite information. So M sees enough to know whether Φ decides $p \in G$ for each p. As such M has G as an element.

But V_κ^M is a rank-initial segment of M so it has all subsets of $\mathbb{P}$ in M. So G is generic for M, which is impossible for nontrivial G. □

Forcing is not a functorial process

This result can be pushed even further.

Theorem (Schlicht & Hamkins–Miller–W.)

Suppose ZFC *is consistent. Then there is no Borel function mapping presentations of countable models of set theory to forcing extensions which preserves isomorphisms.*

Indeed, there cannot even be a Borel function mapping presentations of countable models of set theory to forcing extensions which preserves elementary equivalence.

Forcing is not a functorial process

This result can be pushed even further.

Theorem (Schlicht & Hamkins–Miller–W.)

Suppose ZFC *is consistent. Then there is no Borel function mapping presentations of countable models of set theory to forcing extensions which preserves isomorphisms.*

Indeed, there cannot even be a Borel function mapping presentations of countable models of set theory to forcing extensions which preserves elementary equivalence.

There are limits to how far it can be pushed.

Observation

Assume $\mathrm{V} = \mathrm{L}$. *Then there is a* Δ^1_2 *functor mapping presentations of countable models of set theory to forcing extensions which preserves isomorphism.*

Question

Is there an analytic (co-analytic) functorial method of producing forcing extensions?

Is forcing a computable procedure?

Positive results

- Given a presentation of a model of set theory we can compute its forcing extension.
- For special models we can do this in a functorial way.

Negative results

- But this procedure is in general dependent upon the choice of presentation.

That is, the procedure is computable in the model of set theory equipped with an ω-enumeration of its elements, not merely in the model itself.

Thank you!

- Joel David Hamkins, Russell Miller, and Kameryn J Williams, "Forcing as a computational process", *under review*.
  Preprint: arXiv:2007.00418 [math.LO].
- Matthew Harrison-Trainor, Alexander Melkinov, Russell Miller, and Antonio Montalbán, "Computable functors and effective interpretability", JSL 82.1 (2017).
- Matthew Harrison-Trainor, Russell Miller, and Antonio Montalbán, "Borel functors and infinitary interpretations", JSL 83.4 (2018).