Computational Back-and-Forth Arguments in Constructive Type Theory

Dominik Kirst
Saarland University, Saarland Informatics Campus, Germany

Abstract

The back-and-forth method is a well-known technique to establish isomorphisms of countable structures. In this proof pearl, we formalise this method abstractly in the framework of constructive type theory, emphasising the computational interpretation of the constructed isomorphisms. As prominent instances, we then deduce Cantor's and Myhill's isomorphism theorems on dense linear orders and one-one interreducible sets, respectively. By exploiting the symmetry of the abstract argument, our approach yields a particularly compact mechanisation of the method itself as well as its two instantiations, all implemented using the Coq proof assistant. As adequate for a proof pearl, we attempt to make the text and mechanisation accessible for a general mathematical audience.

2012 ACM Subject Classification Theory of computation → Constructive mathematics; Theory of computation → Type theory; Theory of computation → Logic and verification

Keywords and phrases back-and-forth method, computable isomorphisms, Coq

Digital Object Identifier 10.4230/LIPIcs.ITP.2022.20

Supplementary Material `https://www.ps.uni-saarland.de/extras/back-and-forth`

1 The Informal Argument

We begin by explaining the back-and-forth method informally to provide some background and intuition, using the example of Cantor's isomorphism theorem on countable dense linear orders. Actually established by Cantor himself via an alternative but related method [2], his result is often considered the origin of the back-and-forth argument and textbook presentations typically follow this particular strategy (Hausdorff's "Grundzüge der Mengenlehre" [8] may be the earliest example).[1] An important consequence of Cantor's isomorphism theorem is that the first-order theory of unbounded dense linear orders is complete and therefore decidable, which is the reason why the back-and-forth method was brought to fruition with a variety of similar results in model theory (see for instance [13], especially Chapter 6). While we do not include more of these model-theoretic constructions in this paper, as a second instance of an explicitly computational flavour we will consider Myhill's isomorphism theorem [12], stating that one-one interreducible sets are recursively isomorphic, at a later point.

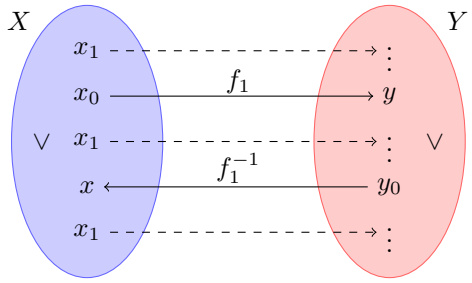
So to get started on Cantor's isomorphism theorem, recall that (strict) linear orders may be characterised as irreflexive and transitive binary relations $(X, <)$ satisfying trichotomy $(x < y \vee x = y \vee y < x)$. Such a linear order is called dense if whenever $x < y$, then there is some z with $x < z < y$. These requirements alone do not yet fully determine the structure, as two orderings might differ regarding their endpoints. This leaves four possible cases depending on the existence of left and/or right endpoints, we focus on the unbounded version:

$$\forall x.\, \exists y y'.\, y < x < y'$$

Now given two unbounded dense linear orders $(X, <)$ and $(Y, <)$ that are countable, so assuming that $X = \{x_0, x_1, x_2, \dots\}$ and $Y = \{y_0, y_1, y_2, \dots\}$, we want to establish a

[1] We refer the interested reader to [17] for further historic references as well as a comparison of Cantor's actual proof to the back-and-forth method.

Figure 1 Illustration of the three cases how a finite stage of the order-isomorphism of $(X, <)$ and $(Y, <)$ can be extended. By construction, the stage f_1 already contains matchings for x_0 and y_0. For the construction of f_2 we want to find a matching for the next element x_1, which either lies in between the domain of f_1 or completely above or below it. In any case we find a matching partner using density, upwards unboundedness, or downwards unboundedness of Y.



bijection $F : X \to Y$ respecting the order, i.e. satisfying $x < x'$ iff $F(x) < F(x')$. This order-isomorphism can be constructed in finite stages, meaning that we begin with the empty partial isomorphism f_0 and iteratively extend to f_1, f_2, ... such that x_n is in the domain of f_{n+1}. To make sure that in this process no element of Y is missed, we simultaneously take care that y_n is in the range of f_{n+1}. Once the stages f_n are defined, we simply set $F(x_n) := f_{n+1}(x_n)$ and observe that the expected properties transport from the stages to F. Borrowing common set-theoretic notation, the construction can be summarised as

$$f_0 := \emptyset$$
$$f_{n+1} := \{(x_n, y), (x, y_n)\} \cup f_n$$
$$F := \bigcup_{n \in \mathbb{N}} f_n$$

where the matching partners y and x for x_n and y_n, respectively, remain to be determined.

To this end, we begin with a forwards step treating x_n. If $f_n(x_n)$ is already defined, then there is nothing to do. Otherwise, we need to find a suitable partner y not yet in the range of f_n which we can safely add as image of x_n while preserving the orderings. We distinguish three possible cases, which are also depicted in Figure 1:

- If there are x and x' in the domain of f_n such that $x < x_n < x'$, then (since f_n is finite) we can assume that x is the greatest such element while x' is the smallest. By the density of $(Y, <)$ there is y with $f_n(x) < y < f_n(x')$ which is then a suitable match for x.
- If the whole domain of f_n is below x_n, then using the unboundedness of $(Y, <)$ we choose y such that the whole range of f_n is below y. This makes y the desired match for x.
- In the remaining case where x_n is below the domain of f_n, we find a match y below the range of f_n again by unboundedness.

In any case we set $f'_{n+1}(x_n) := y$ as an intermediate extension of f_n and continue with the backwards step treating y_n, again only necessary if y_n is not yet in the range of f'_{n+1}. This step is completely symmetric to the forward step, that is, we find a suitable match x for y_n by distinguishing the same three cases regarding the position of y_n relative to the range of f'_{n+1}. Once a matching partner x is found, we extend f'_{n+1} by setting $f_{n+1}(x) := y_n$.

This concludes the construction of f_{n+1} and therefore of the order-isomorphism F in the case of unbounded orders. We just remark that the constructions work analogously if $(X, <)$ and $(Y, <)$ were to contain endpoints of the same characteristic.

In the next section, we continue by extracting the abstract idea of the above argument into a general isomorphism theorem, which is then instantiated to Cantor's result in Section 4. To illustrate the generality, in Section 5 we provide a second instantiation to Myhill's isomorphism theorem, crucially relying on the computational perspective outlined in Section 3.

All results of these technical sections are formalised in constructive type theory and mechanised in a single Coq file of about 500 lines of code.[2] While (even constructive) mechanisations of Cantor's isomorphism theorem were already given Giese and Schönegge [6] as well as Marzion [10], and while we closely follow Forster, Jahn, and Smolka [3] in the instantiation to Myhill's isomorphism theorem, we are not aware of an abstract formulation in constructive type theory, let alone a mechanisation with a proof assistant, of the back-and-forth method and its computational interpretation as such.

2 The Abstract Argument

Now switching to the formal framework of constructive type theory, we represent the two sets X and Y subject to the back-and-forth argument by types in the universe $\mathbb{T}$. Since the method only applies to countable structures, we could fix X and Y to be the type $\mathbb{N}$ of natural numbers. To explicitly include finite types, however, we just assume that X and Y are retracts of $\mathbb{N}$, so for instance X comes with an injection $i_X : X \hookrightarrow \mathbb{N}$ with explicitly given surjective left-inverse $r_X : \mathbb{N} \twoheadrightarrow X$, where we just write x_n for $r_X\, n$.[3] In order to investigate the argument abstractly, we need to axiomatise three ingredients: the particular structure of each instance, the corresponding notion of structure-preserving isomorphism, and a procedure to extend a partial isomorphism by one step.

The Abstract Assumptions

We first abstract over the required structure relating both sides of the potential isomorphism by an operation $\mathcal{A} : \mathbb{T} \times \mathbb{T} \to \mathbb{T}$. For Cantor's isomorphism theorem, an element $\mathcal{S}_{X,Y}$ of $\mathcal{A}(X, Y)$ will independently equip both X and Y with a dense unbounded linear order. Myhill's isomorphism theorem, however, imposes shared structure via mutual one-one reductions, making a binary operation necessary. To still enable the full usage of symmetry, we further assume a function associating to every structured pair $\mathcal{S}_{X,Y}$ its inverse $\mathcal{S}^{-1}_{X,Y}$ in $\mathcal{A}(Y, X)$ with

$$(\mathcal{S}^{-1}_{X,Y})^{-1} = \mathcal{S}_{X,Y}. \tag{1}$$

The next ingredient is the abstract isomorphism property we want to establish. It suffices to express this property locally, i.e. to relate pairs (x, x') in $X \times X$ and (y, y') in $Y \times Y$, written $(x, x') \sim (y, y')$, relative to $\mathcal{S}_{X,Y}$. The idea is to let $(x, x') \sim (y, y')$ express that x relates to x' in the structure over X exactly like y relates to y' in the structure over Y. In particular, we require that the structural similarity extends to equality, formally

$$(x, x') \sim (y, y') \;\to\; (x = x' \leftrightarrow y = y') \tag{2}$$

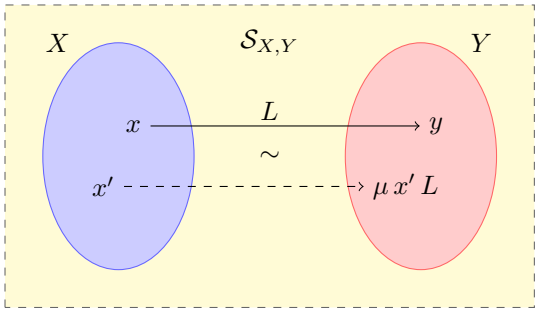
and that inverting the structure $\mathcal{S}_{X,Y}$ to $\mathcal{S}^{-1}_{X,Y}$ maintains the matching, formally

$$(x, x') \sim (y, y') \;\to\; (y, y') \sim (x, x') \tag{3}$$

[2] The Coq file can be obtained from the URL listed as supplementary material and an HTML version is accessible by simply clicking on any of the highlighted statements in this PDF.

[3] Note that for simplicity we exclude empty types as retracts of $\mathbb{N}$ by the requirement that r_X be total. For our purpose, we deem this choice reasonable as empty structures are usually trivially isomorphic.

Figure 2 Illustration of the abstract assumptions. Two types X and Y are related by a surrounding structure $\mathcal{S}_{X,Y}$. If a partial isomorphism L connects an element $x : X$ with $y : Y$, then for x' the step function yields an element $\mu\, x'\, L$ relating to y like x' relates to x, yielding $(x, x') \sim (y, \mu\, x'\, L)$.



where the premise and conclusion are relative to $\mathcal{S}_{X,Y}$ and $\mathcal{S}^{-1}_{X,Y}$, respectively. The desired isomorphism can then be characterised as a function $F : X \to Y$ with inverse $F^{-1} : Y \to X$ such that $(x, x') \sim (F\, x, F\, x')$ for all x and x'.

Before we get to the final third ingredient, we need to model the notion of (finite) partial isomorphisms approximating F and F^{-1}. A simple choice is to take lists[4] $L : \mathcal{L}(X \times Y)$ of pairs $(x, y) : X \times Y$ and call them partial isomorphisms, written $\mathcal{S}_{X,Y}(L)$ to emphasise the dependency on $\mathcal{S}_{X,Y}$, if $(x, x') \sim (y, y')$ whenever $(x, y) \in L$ and $(x', y') \in L$. By flipping the components of each pair in L, we can turn L into $L^{-1} : \mathcal{L}(Y \times X)$ with the obvious property that $\mathcal{S}^{-1}_{X,Y}(L^{-1})$ whenever $\mathcal{S}_{X,Y}(L)$. We denote by $\mathsf{dom}(L)$ and $\mathsf{ran}(L)$ the domain and range of L, respectively, and write $L[x]$ for the (first) value of $x \in \mathsf{dom}(L)$.

So finally concluding the abstract ingredients, we assume a polymorphic step function

$$\mu \;:\; \forall XY.\, \mathcal{A}(X, Y) \to X \to \mathcal{L}(X \times Y) \to Y$$

finding a suitable partner y for x not yet included in L, meaning that if L is a partial isomorphism and $x \notin \mathsf{dom}(L)$, then $(x, \mu\, x\, L) :: L$ is a partial isomorphism:[5]

$$x \notin \mathsf{dom}(L) \;\to\; \mathcal{S}_{X,Y}(L) \;\to\; \mathcal{S}_{X,Y}((x, \mu\, x\, L) :: L) \tag{4}$$

To sum up, we illustrate the abstract setup graphically in Figure 2 and provide an overview of all assumptions as stated in Coq in Figure 3.

The Abstract Construction

As in the informal argument, the idea is to start with the empty partial isomorphism and then to extend in step $n + 1$ by matching up both x_n and y_n with suitable partners (if not yet accommodated), thereby ensuring that both structures are exhausted. So we first extend μ to a polymorphic function μ' adding a new match for x to L if no existing match is found:

$$\mu' \;:\; \forall XY.\, \mathcal{A}(X, Y) \to X \to \mathcal{L}(X \times Y) \to \mathcal{L}(X \times Y)$$

$$\mu'\, x\, L \;:=\; \begin{cases} (x, \mu\, x\, L) :: L & \text{if } x \notin \mathsf{dom}(L) \\ (x, L[x]) :: L & \text{if } x \in \mathsf{dom}(L) \end{cases}$$

[4] Constructed from the empty list [] and the cons operation $x :: L$ for $x : X$ and $L : \mathcal{L}(X)$.
[5] We treat the first three arguments of μ as implicit and will do so for all similar functions.

Figure 3 Abstract assumptions regarding the structure (`structure`), the corresponding notion of isomorphism (`iso`), and the extension procedure (`find`) as formulated in Coq. Note that the former two take their arguments in curried form as opposed to the cartesian representation on paper.

```
structure : Type -> Type -> Type
srev : forall X Y, structure X Y -> structure Y X
srev_invol : forall X Y (S : structure X Y), srev (srev S) = S

iso : forall X Y, structure X Y -> X -> X -> Y -> Y -> Prop
iso_eq : forall X Y (S : structure X Y) x x' y y', iso S x x' y y' -> x = x' <-> y = y'
iso_rev : forall X Y (S : structure X Y) x x' y y', iso S x x' y y' -> iso (srev S) y y' x x'

find : forall X, structure X Y -> X -> list (X * Y) -> Y
find_iso : forall X Y (S : structure X Y) L x, x ∉ dom L -> tiso S L -> tiso S ((x, find S x L) :: L)
```

Note that in the second case, where a match for x was already present in L, we could as well just return L, but the choice to repeat the match slightly simplifies the (in any case straightforward) argument for the desired property $x \in \mathsf{dom}(\mu'\, x\, L)$.

Now since μ' is polymorphic and therefore usable in both directions, the iterative process at the heart of the back-and-forth argument can be defined exploiting the structural symmetry:

$$L\_ \;:\; \forall XY.\, \mathcal{A}(X,Y) \to \mathbb{N} \to \mathcal{L}(X \times Y)$$

$$\begin{aligned} L_0 &:= [] \\ L_{n+1} &:= (\mu'\, y_n\, (\mu'\, x_n\, L_n)^{-1})^{-1} \end{aligned}$$

So in the second case, we first extend L_n with a match for x_n by the inner application of μ' (regarding $\mathcal{S}_{X,Y}$) and then flip the result to obtain a partial isomorphism in the opposite direction. This is then extended with a match for y_n by the outer application of μ' (regarding $\mathcal{S}^{-1}_{X,Y}$), after which the result is flipped again to yield a partial isomorphism in the original direction.

Since we have $x_n \in \mathsf{dom}(L_{n+1})$ and $y_n \in \mathsf{dom}(L_{n+1})^{-1} = \mathsf{ran}(L_{n+1})$ by the above remark that $x \in \mathsf{dom}(\mu'\, x\, L)$ for all x, we can define the isomorphism $F : X \to Y$ and its inverse $F^{-1} : Y \to X$ on a given structured pair $\mathcal{S}_{X,Y}$ simply by

$$F\, x_n \;:=\; L_{n+1}[x_n] \qquad \text{and} \qquad F^{-1}\, x_n \;:=\; L^{-1}_{n+1}[y_n]$$

where we implicitly use that for every $x : X$ there is $n : \mathbb{N}$ with $x = x_n$, analogously for $y : Y$.

The Abstract Verification

To conclude the abstract back-and-forth argument, we show that the function $F : X \to Y$ is an isomorphism for $\mathcal{S}_{X,Y}$. We first formally state that flipping and extending with μ' preserves partial isomorphisms.

▶ **Lemma 1**. *If* $\mathcal{S}_{X,Y}(L)$ *then* $\mathcal{S}^{-1}_{X,Y}(L^{-1})$ *and* $\mathcal{S}_{X,Y}(\mu'\, x\, L)$.

Proof. Straightforward by Assumptions (3) and (4), respectively. ◀

Then in particular every L_n is a partial isomorphism.

▶ **Lemma 2**. $\mathcal{S}_{X,Y}(L_n)$

Proof. By induction on n, the case of $L_0 = []$ is trivial. For L_{n+1} we assume $\mathcal{S}_{X,Y}(L_n)$ as inductive hypothesis and apply both parts of Lemma 1 in both directions to obtain $(\mathcal{S}^{-1}_{X,Y})^{-1}(L_{n+1})$, which is the same as $\mathcal{S}_{X,Y}(L)$ by Assumption (1). ◀

We can now prove the abstract isomorphism theorem with a simple argument.

▶ **Theorem 3** (Isomorphism). *F satisfies $(x, x') \sim (F\,x, F\,x')$ and is inverted by F^{-1}.*

Proof. For the first claim, let $x = x_n$, $x' = x_m$, and w.l.o.g $n \leq m$. Since then $L_{n+1} \subseteq L_{m+1}$, we have $(x, F\,x) = (x_n, L_{n+1}[x]) \in L_{m+1}$ and $(x', F\,x') = (x_m, L_{m+1}[x]) \in L_{m+1}$, from which conclude $(x, x') \sim (F\,x, F\,x')$ since $\mathcal{S}_{X,Y}(L_{m+1})$ by Lemma 2.

For the second claim, note that for some n big enough, we have both $(x, F\,x) \in L_n$ (since x is eventually handled by some forwards step) and $(F^{-1}(F\,x), F\,x) \in L_n$ (since $F\,x$ is eventually handled by some backwards step). By Lemma 2 we therefore obtain $(x, F^{-1}(F\,x)) \sim (F\,x, F\,x)$, from which $x = F^{-1}(F\,x)$ follows by Assumption (2). Analogously, we establish $y = F\,(F^{-1}\,y)$ and thus conclude the claim that F^{-1} inverts F. ◀

3 The Computational Argument

Before we consider instantiations of our abstract back-and-forth argument, we take a break to compare two quite different readings of the isomorphism theorem which are available in our constructive setting. Conventionally, Theorem 3 could be summarised as:

> *Countable structures with a structure-preserving extension function are isomorphic.*

Here the view is that both the input extension function and the output isomorphism are ordinary (set-theoretic) functions. Moreover, the embedding of the structures into the natural numbers is a mere limitation to countable cardinality.

The alternative view is that we consider all involved functions to be computable, relying on the fact that all functions definable in constructive type theory are such.[6] This allows for a synthetic approach to computability [16, 1, 4] disposing of any reference to a formal model of computation like Turing machines, especially easing the mechanisation of otherwise quite laborious arguments regarding undecidability [5] and incompleteness [9]. Also in situations like the back-and-forth method with its set-theoretic reading covering up the inherent computational content, this approach admits an elegant combination of both perspectives.

So adopting the computational view, the input extension function provides a matching algorithm μ which we use to implement μ', L_n, and ultimately F. Since we are able to implement these functions without classical assumptions,[7] their computability is guaranteed. The only price we have to pay is that the classically trivial case distinction on $x \in \mathsf{dom}(L)$ in the definition of μ' needs to be constructively justified. Fortunately, implementing the necessary decision procedure for list membership on a type with decidable equality is a simple exercise, especially if we instead were to hastily use classical assumptions at the cost of having to program Turing machines for computational results. Finally observing that a computable embedding into natural numbers replaces countability with effective enumerability and discreteness (i.e. decidable equality), we may reinterpret Theorem 3:

> *Computational interpretation:* ***enumerable*** *and* ***discrete*** *structures with a structure-preserving extension* ***algorithm*** *are* ***computably*** *isomorphic.*

[6] An intuitive justification for this fact is that constructive type theory is nothing but a dependently typed programming language. A more formal justification applicable for many constructive systems is to exhibit a realisability model with the desired properties.

[7] So for instance without appeal to the excluded middle ($\forall P.\, P \vee \neg P$), or even stronger, the axiom of choice. Both principles can be consistently added to constructive type theory but, depending on the formulation, already the former would allow the definition of uncomputable functions.

This means that in each of the upcoming instantiations we actually obtain two results: a classical and a computational isomorphism. Moreover, both readings are meaningful in their own right and it is one of the nice features of constructive mathematics and synthetic computability that we obtain them both simultaneously after doing the work just once.[8]

4 Cantor's Isomorphism Theorem

We now start reaping what we sowed in Section 2 and establish Cantor's result with our general isomorphism theorem. To admit the computational view, we represent unbounded dense linear orders $(X, <)$ with explicit functions d, l, and g for density and unboundedness:

$$x < d\,x\,y < y \qquad\qquad l\,x < x < g\,x$$

Moreover, we assume a characteristic function $f_< : X \to X \to \mathbb{B}$ using the inductive type $\mathbb{B}$ of boolean values such that $f_<\,x\,y = \mathsf{true}$ iff $x < y$.[9] In the computational view, this means that the witnesses for density and unboundedness can be computed and that the relation is decidable, while in the classical view using these functions is a mere symbolic convenience.

We now fix two countable unbounded dense linear orders $(X, <)$ and $(Y, <)$ and bring the informal idea outlined in Section 1 into the shape matching the abstract argument in Section 2. Given two pairs $(x, x') : X \times X$ and $(y, y') : Y \times Y$, we define

$$(x, x') \sim_C (y, y') \;:=\; (x = x' \leftrightarrow y = y') \wedge (x < x' \leftrightarrow y < y')$$

and call $L : \mathcal{L}(X \times Y)$ a partial order-isomorphism if $(x, x') \sim_C (y, y')$ for all $(x, y), (x', y') \in L$. By virtue of the abstract isomorphism theorem, we just need to show how a partial order-isomorphism can be extended by one step in one direction.

▶ **Definition 4**. *We define a function $\mu_C : X \to \mathcal{L}(X \times Y) \to Y$ exactly following the three cases distinguished in Section 1. So given $x \notin \mathsf{dom}(L)$, first use $f_<$ to determine which of the three situations relating x to $\mathsf{dom}(L)$ is accurate. In the case where x is in between elements of $\mathsf{dom}(L)$, use d to find a match in Y. In the two other cases where x is below or above all elements of $\mathsf{dom}(L)$, use l and g, respectively. If $x \in \mathsf{dom}(L)$, we just return a dummy value.*

▶ **Lemma 5**. *If $x \notin \mathsf{dom}(L)$ and L is a partial order-isomorphism, then so is $((x, \mu_C\,x\,L) :: L)$.*

Proof. The only non-trivial obligation is to show $(x, x') \sim_C (\mu_C\,x\,L, y')$ for $(x', y') \in L$:

- That $x = x'$ iff $\mu_C\,x\,L = y'$ is straightforward since both are actually false. The former is ruled out by $x \notin \mathsf{dom}(L)$ and the latter since in that case $\mu_C\,x\,L \notin \mathsf{ran}(L)$.
- That $x < x'$ iff $\mu_C\,x\,L < y'$ follows by analysing the cases underlying the definition of μ_C. We only consider the more complicated case where x lies in between elements of $\mathsf{dom}(L)$, the other two cases are similar and easier. In this situation, there are x_{max} and x_{min} in $\mathsf{dom}(L)$ with $x_{\mathsf{max}} < x < x_{\mathsf{min}}$ with no domain element in between. Since L is already a partial order-isomorphism and since then $\mu_C\,x\,L = d\,L[x_{\mathsf{max}}]\,L[x_{\mathsf{min}}]$, we have $L[x_{\mathsf{max}}] < \mu_C\,x\,L < L[x_{\mathsf{min}}]$ in Y with no range element in between. We then deduce:

$$x < x' \;\Leftrightarrow\; x_{\mathsf{min}} \le x' \;\Leftrightarrow\; L[x_{\mathsf{min}}] \le L[x'] \;\Leftrightarrow\; \mu_C\,x\,L < y' \qquad \blacktriangleleft$$

Now Cantor's isomorphism theorem in its classical formulation can be derived directly.

[8] As already done in Section 2 we continue to try and use neutral formulations avoiding a bias into one of the readings, although wordings like "we construct" or "we compute" impose themselves very often.

[9] Note that in Coq we prefer to formulate these assumptions with informative types, for instance d is expressed by $\forall xy.\,\Sigma z.\,x < z < y$ and $f_<$ could be expressed as $\forall xy.\,(x < y) + (x \not< y)$.

Figure 4 Coq proof script for Cantor's isomorphism theorem. After using the general isomorphism theorem `back_and_forth`, the first eight goals are arranged like the assumptions listed in Figure 3. The only non-trivial ones are the latter two where `partner` is μ_C and `step_morph` is Lemma 5. The next five goals instantiate to the given orders and the last goal is the actual derivation of the claim.

```
Theorem Cantor X Y (OX : dulo X) (OY : dulo Y) (RX : retract X nat) (RY : retract Y nat) :
{ F : X -> Y & { G | inverse F G /\ forall x x', x < x' <-> (F x) < (F x') } }.
Proof.
  unshelve edestruct back_and_forth as [F[G[H1 H2]]].
  - intros A B. exact (dulo A * dulo B).                                    (* structure *)
  - intros A B [OA OB]. exact (OB, OA).                                     (* srev *)
  - cbn. intros A B [OA OB]. reflexivity.                                   (* srev_invol *)
  - intros A B [OA OB] a a' b b'. exact ((a = a' <-> b = b') /\ (a < a' <-> b < b')).  (* iso *)
  - cbn. tauto.                                                             (* iso_eq *)
  - cbn. tauto.                                                             (* iso_rev *)
  - cbn. intros A B [OA OB] f a. exact (partner OA OB f a).                 (* find *)
  - cbn. intros A B [OA OB] f x. unfold step, tiso. now apply step_morph.   (* find_iso *)

  - exact X.
  - exact Y.
  - exact (OX, OY).
  - exact RX.
  - exact RY.

  - cbn in *. exists F, G. split; try apply H1. apply H2.
Qed.
```

▶ **Theorem 6** (Cantor). *All countable unbounded dense linear orders are isomorphic.*

Proof. We instantiate Theorem 3 as follows, see also Figure 4:

- As structure $\mathcal{A}(X, Y)$ we require that X and Y are unbounded dense linear orders.
- The inversion operation turning elements of $\mathcal{A}(X, Y)$ to $\mathcal{A}(Y, X)$ is obvious.
- As isomorphism property we choose $(x, x') \sim_C (y, y')$.
- As polymorphic one-step extension function we choose μ_C.
- Assumptions (1)–(3) are trivial.
- Assumption (4) was proven in Lemma 5.

This yields a function $F : X \to Y$ with inverse F^{-1} such that $(x, x') \sim_C (F\,x, F\,x')$. The latter contains $x < x' \leftrightarrow F\,x < F\,x'$, so F is an order-isomorphism. ◀

The corresponding computational result we obtain for free then reads:

▶ **Theorem 7** (Computational Cantor). *All decidable linear orders over enumerable domain with computable witnesses for density and unboundedness are computably isomorphic.*

5 Myhill's Isomorphism Theorem

As a second instance of our general isomorphism theorem, we tackle Myhill's result that one-one interreducible sets of natural numbers are recursively isomorphic.[10] To first provide some intuition, recall that a set $X \subseteq \mathbb{N}$ many-one reduces to a set $Y \subseteq \mathbb{N}$ if there is a computable function $f : \mathbb{N} \to \mathbb{N}$ such that $f(x) \in Y$ implies $x \in X$ and $f(x) \notin Y$ implies $x \notin Y$. As a typical usage of such a reduction f, a decision procedure for $x \in X$ can be obtained by a decider for Y by first computing $f(x)$ and then determining whether $f(x) \in Y$. If f happens to be injective (bijective), it is called one-one reduction (recursive isomorphism).

[10] We simply adopt the Coq mechanisation given by Forster et al. [3] to our abstract setting and explicitly link up the corresponding definitions and theorems in the forthcoming text.

Given that interreducible sets and recursive isomorphisms are algorithmic notions, we now fully rely on the synthetic approach to computability as explained in Section 3 and take simple type-theoretic definitions representations of these notions:

- Sets of natural numbers are more generally represented as unary predicates $p : X \to \mathbb{P}$ on enumerable and discrete types X (that is, retracts of $\mathbb{N}$).
- A many-one reduction from $p : X \to \mathbb{P}$ to $q : Y \to \mathbb{P}$ is represented as a function $f : X \to Y$ such that $p\,x$ iff $q\,(f\,x)$, which we denote by $f : p \preceq q$.
- A one-one reduction (recursive isomorphism) is an injective (bijective) many-one reduction.

Recall that if we were to use classical logic, we would spoil the computational interpretation and instead would have to explicitly require reductions to be computable in a formal model like Turing machines. This would turn at least the mechanisation level into a mess, especially since then the recursive isomorphism obtained by the back-and-forth method must be shown computable, too.

That said, we now fix two predicates $p : X \to \mathbb{P}$ and $q : Y \to \mathbb{P}$ on retracts X and Y of $\mathbb{N}$, together with one-one reductions $f : p \preceq q$ and $g : q \preceq p$. A recursive isomorphism $F : X \to Y$ will satisfy both $F : p \preceq q$ and $F^{-1} : q \preceq p$, which yields a strict correspondence of both predicates. Expressing this correspondence locally for pairs $(x, x') : X \times X$ and $(y, y') : Y \times Y$, we therefore want to establish that $p\,x$ iff $q\,y$ (by symmetry the same follows for x' and y'), along with the usual requirement that the correspondence be one-to-one:

$$(x, x') \sim_M (y, y') \;:=\; (x = x' \leftrightarrow y = y') \wedge (p\,x \leftrightarrow q\,y)$$

We call $L : \mathcal{L}(X \times Y)$ a partial (recursive) isomorphism if $(x, x') \sim_M (y, y')$ for all $(x, y), (x', y') \in L$. Again by virtue of the abstract isomorphism theorem, we just need to show how a partial isomorphism can be extended by one step in one direction.

▶ **Definition 8** (Lemma 55 in [3]). *We first define a function $\mu_M^X : X \to \mathcal{L}(X \times Y) \to X$ such that given x and L it is safe to add $f\,(\mu_M^X\,x\,L)$ as value for x to L. So determine whether $f\,x \in \mathsf{ran}(L)$, if not we can just return x. If otherwise $(x', f\,x) \in L$ return $\mu_M^X\,x'\,L'$, where L' is L with $(x', f\,x)$ removed, so we keep searching until we find some element x_0 with $f\,x_0 \notin \mathsf{ran}(L)$ while the search along f ensures that $p\,x$ iff $p\,x_0$.*[11]

We then obtain a function $\mu_M : X \to \mathcal{L}(X \times Y) \to Y$ by $\mu_M\,x\,L := f\,(\mu_M^X\,x\,L)$.

▶ **Lemma 9**. *If $x \notin \mathsf{dom}(L)$ and L is a partial isomorphism, then so is $((x, \mu_M\,x\,L) :: L)$.*

Proof. We first verify that $\mu_M\,x = f\,(\mu_M^X\,x\,L)$ indeed is a suitable match for x:

$$(p\,x \leftrightarrow p\,(\mu_M^X\,x\,L)) \;\wedge\; \mu_M^X\,x\,L \in x :: \mathsf{dom}(L) \;\wedge\; f\,(\mu_M^X\,x\,L) \notin \mathsf{ran}(L) \qquad (*)$$

We apply induction following the execution trace of μ_M^X. In the terminating case where $f\,x \notin \mathsf{ran}(L)$, we have $\mu_M^X\,x\,L = x$ and hence the claim is trivial. In the recursive case we have $\mu_M^X\,x\,L = \mu_M^X\,x'\,L'$, where L' is L without $(x', f\,x)$, and assume the claim for x' and L'.

- We deduce $p\,x \leftrightarrow p\,(\mu_M^X\,x'\,L')$ using $p\,x \leftrightarrow q\,(f\,x)$, since f is a reduction, $q\,(f\,x) \leftrightarrow p\,x'$ since L is a partial isomorphism, and $p\,x' \leftrightarrow p\,(\mu_M^X\,x'\,L'))$ by the inductive hypothesis.
- We already have $\mu_M^X\,x'\,L' \in \mathsf{dom}(L)$ since by induction $\mu_M^X\,x'\,L' \in x' :: \mathsf{dom}(L')$ and we have $x' :: \mathsf{dom}(L') = \mathsf{dom}(L)$ by construction of L'.

[11] Since this algorithm is not structurally recursive but descends on the length of L, we use the Equations package [18] to implement it in Coq.

Figure 5 Coq proof script for Myhill's isomorphism theorem. After using the general isomorphism theorem `back_and_forth`, the first eight goals are arranged like the assumptions listed in Figure 3. The only non-trivial ones are the latter two where `mstep` is μ_M and `step_corr` is Lemma 9. The next five goals instantiate to the given orders and the last goal is the actual derivation of the claim.

```
Theorem Myhill X Y (SXY : bireduction X Y) (RX : retract X nat) (RY : retract Y nat) :
{ F : X -> Y & { G | inverse F G /\ reduction p q F } }.
Proof.
  unshelve edestruct back_and_forth as [F[G[H1 H2]]].
  - intros A B. exact (bireduction A B).                                    (* structure *)
  - intros A B S. cbn in *. apply (@Build_bireduction B A eY eX q p g f); apply S.   (* srev *)
  - cbn. intros A B []. reflexivity.                                        (* srev_invol *)
  - intros A B S a a' b b'. cbn in S. exact ((a = a' <-> b = b') /\ (p a <-> q b)).  (* iso *)
  - cbn. tauto.                                                             (* iso_eq *)
  - cbn. tauto.                                                             (* iso_rev *)
  - cbn. intros A B S C a. exact (mstep f eX eY C a).                       (* find *)
  - cbn. intros A B S C a. unfold step, tiso. apply step_corr; apply S.     (* find_iso *)

  - exact X.
  - exact Y.
  - exact SXY.
  - exact RX.
  - exact RY.

  - cbn in *. exists F, G. split; try apply H1. intros x. now apply H2.
Qed.
```

- We assume $f\,(\mu_M^X\,x'\,L') \in \mathsf{ran}(L)$, so given that $\mathsf{ran}(L) = f\,x :: \mathsf{ran}(L')$ we have either $f\,(\mu_M^X\,x'\,L') \in \mathsf{ran}(L')$ or $f\,(\mu_M^X\,x'\,L') = f\,x$. Since the former contradicts the inductive hypothesis the latter must be the case, so $\mu_M^X\,x'\,L' = x$ by the injectivity of f. Since the second part of the inductive hypothesis then yields $x \in x' :: \mathsf{dom}(L')$, we obtain a contradiction to the assumption $x \notin \mathsf{dom}(L)$.

Now returning to the actual goal, we show $(x, x') \sim_M (\mu_M\,x\,L, y')$ for $(x', y') \in L$.

- That $x = x'$ iff $\mu_M\,x\,L = y'$ is trivial using $x \notin \mathsf{dom}(L)$ and $\mu_M\,x\,L \notin \mathsf{ran}(L)$, the former being an assumption and the latter part three of $(*)$.
- That $p\,x$ iff $q\,(\mu_M\,x\,L)$ is immediate using that $q\,(\mu_M\,x\,L)$ iff $p\,(\mu_M^X\,x\,L)$ since f is a reduction, and $p\,(\mu_M^X\,x\,L)$ iff $p\,x$ by part one of $(*)$. ◀

Now Myhill's isomorphism theorem follows for all enumerable and discrete types.

▶ **Theorem 10** (Myhill, Theorem 58 in [3]). *One-one interreducible unary predicates on enumerable and discrete types are recursively isomorphic.*

Proof. We instantiate Theorem 3 as follows, see also Figure 5:

- As structure $\mathcal{A}(X, Y)$ we require a pair of one-one interreducible predicates.
- The inversion operation turning elements of $\mathcal{A}(X, Y)$ to $\mathcal{A}(Y, X)$ is obvious.
- As isomorphism property we choose $(x, x') \sim_M (y, y')$.
- As polymorphic one-step extension function we choose μ_M.
- Assumptions (1)–(3) are trivial.
- Assumption (4) was proven in Lemma 9.

This yields a function $F : X \to Y$ with inverse F^{-1} such that $(x, x') \sim_M (F\,x, F\,x')$. The latter contains $p\,x \leftrightarrow q\,(F\,x)$, so F is a recursive isomorphism. ◀

To conclude, observe that if we forget about the computational view, we have just proven something quite different: the Cantor-Bernstein theorem restricted to countable sets. This restriction happens to be provable constructively (exactly due to the connection to Myhill's isomorphism theorem), while the general result is equivalent to the excluded middle [14].

▶ **Theorem 11** (Countable Cantor-Bernstein, Theorem 59 in [3]). *Countable sets X and Y with given injections $f : X \to Y$ and $g : Y \to X$ also admit a bijection $h : X \to Y$.*

Proof. If we choose as p and q the empty predicates on X and Y, respectively, then f trivially satisfies the defining property of a reduction from p to q and analogously so does g for q and p. Thus Theorem 10 yields a bijection (with a trivial additional property regarding p and q that we just ignore). ◀

6 Conclusion

In this proof pearl, we have isolated an abstract and computational approach to the back-and-forth method and considered two prominent instantiations. As the aim was a condensed and pointed presentation, we refrained from providing more instantiations in other domains for now. Nevertheless, this brief concluding section is meant to sketch a few more applications that our approach might be usable for.

First of all, for Cantor's isomorphism theorem we only discuss the characteristic of dense linear orders without endpoints, as it is the most familiar case due to its applicability to $\mathbb{Q}$. However, the three other cases of one (left or right) or two endpoints are completely analogous and can be obtained by simple modification of our proof. Of course one could also abstract over the exact characteristic, yielding all four versions simultaneously.

Secondly, regarding other domains, we already mentioned the crucial use of back-and-forth arguments in model theory, as described in Chapter 7 of [13]. Two further examples are the isomorphism of countably infinite Boolean algebras [7] as well as the uniqueness of the Rado graph as the result of a particular method to generate random graphs [15]. We believe that all these examples can be described in our abstract framework and so for a succinct Coq mechanisation one would just need to represent the corresponding structure and verify a suitable step function.

Thirdly, not immediately concerning further instantiations, the computational content of our axiom-free mechanisation could be made even more explicit by using Coq's extraction mechanism. With this feature one would obtain the algorithms implemented in the constructive proofs as executable programs in external programming languages. These algorithms could then be analysed regarding their efficiency and compared with respect to the necessary computational primitives. For instance, while the back-and-forth construction itself just follows the enumeration of the domains, the step function in the case of Cantor's isomorphism theorem may exhibit quite different complexity, depending on whether the witnesses for density and unboundedness are provided as efficient functions or need to be computed with linear search. For this example and a more general analysis of the back-and-forth method regarding different notions of computation, we refer the reader to [11].

References

1 Andrej Bauer. First steps in synthetic computability theory. *Electronic Notes in Theoretical Computer Science*, 155:5–31, 2006.

2 Georg Cantor. Beiträge zur Begründung der transfiniten Mengenlehre. *Mathematische Annalen*, 46(4):481–512, 1895.

3 Yannick Forster, Felix Jahn, and Gert Smolka. A Constructive and Synthetic Theory of Reducibility: Myhill's Isomorphism Theorem and Post's Problem for Many-one and Truth-table Reducibility in Coq (Full Version). working paper or preprint, February 2022. URL: `https://hal.inria.fr/hal-03580081`.

4 Yannick Forster, Dominik Kirst, and Gert Smolka. On synthetic undecidability in Coq, with an application to the entscheidungsproblem. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 38–51, 2019.

5 Yannick Forster, Dominique Larchey-Wendling, Andrej Dudenhefner, Edith Heiter, Dominik Kirst, Fabian Kunze, Gert Smolka, Simon Spies, Dominik Wehr, and Maximilian Wuttke. A Coq library of undecidable problems. In *CoqPL 2020 The Sixth International Workshop on Coq for Programming Languages*, 2020.

6 Martin Giese and Arno Schönegge. *Any Two Countable, Densely Ordered Sets Without Endpoints are Isomorphic: A Formal Proof with KIV*. Univ., Fak. für Informatik, 1995.

7 Paul Halmos and Steven Givant. *Introduction to Boolean algebras*. Springer, 2009.

8 Felix Hausdorff. Grundzüge der Mengenlehre, 1914.

9 Dominik Kirst and Marc Hermes. Synthetic undecidability and incompleteness of first-order axiom systems in Coq. In *12th International Conference on Interactive Theorem Proving (ITP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

10 Evan Marzion. Visualizing Cantor's theorem on dense linear orders using Coq. Blog post. URL: `https://emarzion.github.io/Cantor-Thm/`.

11 Alexander G Melnikov and Keng Meng Ng. The back-and-forth method and computability without delay. *Israel Journal of Mathematics*, 234(2):959–1000, 2019.

12 John Myhill. Creative sets. *Journal of Symbolic Logic*, 22(1), 1957.

13 Bruno Poizat. *A course in model theory: an introduction to contemporary mathematical logic*. Springer Science & Business Media, 2012.

14 Pierre Pradic and Chad E. Brown. Cantor-Bernstein implies excluded middle. *arXiv preprint arXiv:1904.09193*, 2019.

15 Richard Rado. Universal graphs and universal functions. *Acta Arithmetica*, 9(4):331–340, 1964.

16 Fred Richman. Church's thesis without tears. *The Journal of symbolic logic*, 48(3):797–803, 1983.

17 Charles L. Silver. Who invented Cantor's back-and-forth argument? *Modern Logic*, 4(1):74–78, 1994.

18 Matthieu Sozeau and Cyprien Mangin. Equations reloaded: High-level dependently-typed functional programming and proving in Coq. *Proceedings of the ACM on Programming Languages*, 3(ICFP):1–29, 2019.