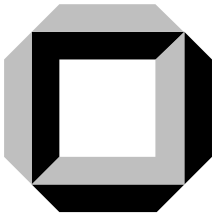
Any two countable, densely ordered sets without endpoints are isomorphic — a formal proof with KIV

Martin Giese Arno Schönegge

Technical Report No. 50/95

December 1995



Universität Karlsruhe

Fakultät für Informatik

Institut für Logik, Komplexität und Deduktionssysteme

Any two countable, densely ordered sets without endpoints are isomorphic — a formal proof with KIV*

Martin Giese Arno Schönegge

Institut für Logik, Komplexität und Deduktionssysteme
Universität Karlsruhe
D-76128 Karlsruhe, Germany
email: schoenegge@ira.uka.de
WWW: http://i11www.ira.uka.de/~schoeneg/

Abstract

Georg Cantor in 1895 gave the first (informal) proof for the fact that *any two countable, densely ordered sets without endpoints are isomorphic* [1]. Here we report on a fully formal proof of this fact constructed interactively with the KIV system [5].

*The presented work was supported under grants no. Me 672/6-2,3 by the Deutsche Forschungsgemeinschaft as part of the focus program "Deduktion".

1 Introduction

This paper reports on a formal proof — completely carried out within the KIV system (Karlsruhe Interactive Verifier) [5] — of a model-theoretic theorem, first shown by Georg Cantor about one hundred years ago [1]. We have done this work for three reasons:

- Firstly, in our opinion mechanically checkable proofs provide a strong argument in the social process that determines ones confidence in a theorem. The use of deduction systems permits to construct substantially more accurate proofs than can be done with current hand methods. This increase in accuracy means an increase in reliability.

- Secondly, our work gives once more an example illustrating the power of current reasoning systems and their range of application.

- Thirdly, the presented case-study demonstrates that software verification techniques can be successfully applied in (some areas of) mathematics, too. As we will argue, the proof task tackled here can be reduced to a problem of proving certain properties of programs. Thus, it can be dealt with in the KIV system which was originally designed for development of verified software.

The paper is organized as follows. The next section presents the theorem we are going to prove. In section 3 several approaches for doing this proof are discussed (with respect to their tractability). The formalization of the problem (section 4) prepares the way to the formal proof we have carried out within the KIV system. Outline and statistics of it are given in section 5. Finally, in the last section, we draw some conclusions.

2 The Theorem to Prove

In [1] Georg Cantor shows that

"Hat man eine einfach geordnete Menge M, welche die drei Bedingungen erfüllt:

1) $\overline{\overline{M}} = \aleph_0$,

2) M hat kein dem Range nach niedrigstes und kein höchstes Element,

3) M ist überalldicht,

so ist der Ordnungstypus von M gleich η:

$$\overline{M} = \eta.$$"

A translation is

> "Any totaly ordered set M with
>
> 1) M is countable and infinite,
> 2) M has no endpoints, and
> 3) M is dense
>
> is of order type η."

In modern nomenclature, "is of order type η" means, that there is an isomorphism (with respect to the order relation) from M to the interval $(0,1)$ of rational numbers.[1] This result is obviously equivalent to the statement, that

> *any two countable, densely ordered sets*
> *without endpoints are isomorphic*

which is a basic fact appearing in textbooks on model theory (e.g. [2], PROPOSITION 1.4.2).

For the following it is convenient to put it in algebraic terms.

Theorem (Cantor)
Let $\mathcal{A} = (A, <_{\mathcal{A}})$ and $\mathcal{B} = (B, <_{\mathcal{B}})$ two structures (algebras) with A and B non-empty[2], countable sets and $<_{\mathcal{A}}$ and $<_{\mathcal{B}}$ total orderings such that

> *no endpoints:*
> *for all $a \in A$ there are some $a', a'' \in A$ with $a' <_{\mathcal{A}} a <_{\mathcal{A}} a''$*
> *for all $b \in B$ there are some $b', b'' \in B$ with $b' <_{\mathcal{B}} b <_{\mathcal{B}} b''$*
>
> *density:*
> *for all $a', a'' \in A$ with $a' <_{\mathcal{A}} a''$ exists some $a \in A$ with $a' <_{\mathcal{A}} a <_{\mathcal{A}} a''$*
> *for all $b', b'' \in B$ with $b' <_{\mathcal{B}} b''$ exists some $b \in B$ with $b' <_{\mathcal{B}} b <_{\mathcal{B}} b''$.*

Then there is a total, bijective function $I : A \to B$ with

$$a_1 <_{\mathcal{A}} a_2 \quad \textit{iff} \quad I(a_1) <_{\mathcal{B}} I(a_2) \quad \textit{for all } a_1, a_2 \in A.$$

Countability and non-emptiness of A and B may equivalently be expressed by the existence of surjective functions $a : \mathbb{N} \to A$ and $b : \mathbb{N} \to B$ enumerating A and B, respectively.

[1] By a slight extension one obtains that any countable, densely ordered set is isomorphic to one of the intervals $(0,1)$, $[0,1)$, $(0,1]$, or $[0,1]$ of rational numbers.

[2] An ordered set with no endpoints is infinite iff it is non-empty

3 How to Do the Proof

In this section we discuss three approaches for proving the theorem above.

The proof given by Cantor [1] is elegant and short — at least at the level of abstraction chosen. It is based on a stepwise construction of an isomorphism $I : A \to B$ following the enumeration $a(0), a(1), a(2), \ldots$ as follows:

$$\begin{array}{lcll} I(a(k)) & := & b(k') & \text{where } k' \text{ is minimal such that for all } i < k \\ & & & I(a(i)) < b(k') \text{ iff } a(i) < a(k) \text{ and} \\ & & & I(a(i)) > b(k') \text{ iff } a(i) > a(k) \end{array}$$

In the order of the enumeration of A, each $a(k)$ is mapped to the the first $b(k')$ in the enumeration of B, which satisfies the isomorphism property with respect to the already fixed part of the mapping. This process starts with mapping $a(0)$ to $b(0)$.

For a human reader, with an *intuitive* access to the concepts of linear ordering and density, the existence of such a k' seems obvious. (The absence of endpoints is needed here, too!). Based on this it is quite easy to show, that I is an injective homomorphism. Surjectivity is harder to see and requires induction. Cantor's proof of the surjectivity of I also strongly relies on some familiarity with the notion of linear order.[3]

Our first approach was to formalize Cantor's proof, i.e. to break down the quite intuitive arguments into (a lot of) small, mechanically checkable proof steps. However — especially in the proof of the existence of the k' mentioned above and in the proof of the surjectivity of I — this turned out to be more complex than we had expected.

These difficulties made us look for other proofs. One alternative, which at least avoids a complicated surjectivity argument appears in [4]. Instead of defining an isomorphism following the enumeration of A, as in Cantor's proof, the first unmapped elements in the enumerations of A *and* B are taken *alternately* and are mapped to an acceptable element of the other set in a way similar to that in Cantor's proof. This process of switching from one enumeration to the other assures that the mapping is surjective as well as total.

However, we have not adopted this approach for two reasons. Firstly, the process of alternating assignment would have resulted in a rather complicated formal description. And secondly, we thought of an even more appealing approach. The idea is to reduce the problem to a more concrete one by

[3] Ernst Zermelo comments on a step in the proof given by Cantor as follows ([1], Anmerkung 13): "... wo es heißt "wie man sich *leicht* überzeugt", findet der Leser eine gewisse Schwierigkeit. ...". This might give an impression about the level of abstraction chosen in Cantor's proof.

adding further information. To show, that any two of a certain class of algebras are isomorphic, it is sufficient to show that all algebras of this class are isomorphic to *one fixed algebra.*

Theorem (Cantor, version 2)
There is an algebra $\mathcal{A}_0 = (A_0, <_{\mathcal{A}_0})$ *such that for all algebras* $\mathcal{A} = (A, <_{\mathcal{A}})$ *with* A *a non-empty, countable set and* $<_{\mathcal{A}}$ *a total ordering such that*

> *no endpoints:*
> *for all* $a \in A$ *there are some* $a', a'' \in A$ *with* $a' <_{\mathcal{A}} a <_{\mathcal{A}} a''$
>
> *density:*
> *for all* $a', a'' \in A$ *with* $a' <_{\mathcal{A}} a''$ *exists some* $a \in A$ *with* $a' <_{\mathcal{A}} a <_{\mathcal{A}} a''$.

there exists a total, bijective function $I_{\mathcal{A}} : A \to A_0$ *with*

$$a_1 <_{\mathcal{A}} a_2 \quad \textit{iff} \quad I(a_1) <_{\mathcal{A}_0} I(a_2) \quad \textit{for all } a_1, a_2 \in A.$$

Thus, proving Cantor's theorem can be done in three steps:

(1) providing an appropriate algebra $\mathcal{A}_0$,

(2) constructing a function $I_{\mathcal{A}} : A \to A_0$ for each $\mathcal{A}$, and

(3) proving that $I_{\mathcal{A}}$ is an isomorphism.

The information added in step (1) can help to make the problems to be solved in steps (2) and (3) more concrete and therefore more tractable.

The most obvious choice for $\mathcal{A}_0$ is the set of all rational numbers in the interval $(0, 1)$, i.e. $\{\frac{n}{m} \mid n, m \in \mathbb{N}, m > n > 0\}$ with their usual order on it.

It turns out, that the rationals in their usual representation as fractions are not very well suited to our purpose, as we are not intersted in arithmetic properties, but rather in properties of the ordering. We shall nevertheless use them to motivate our final choice for $\mathcal{A}_0$.

The property of rationals in $(0, 1)$ we will use for the construction of the isomorphism is that they can be sorted into an infinite binary tree such that

- each node has exactly two children,
- all nodes in the left subtree of a node labeled with r are marked with rationals smaller than r,
- all nodes in the right subtree of a node labeled with r are marked with rationals larger than r, and
- all rationals from the interval $(0, 1)$ occur in the tree.

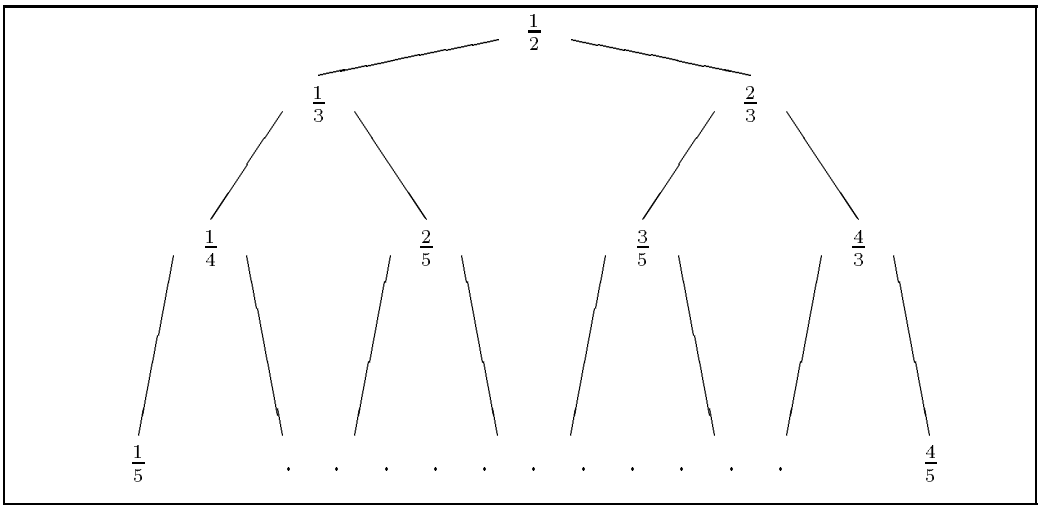


Figure 1: Sorting rationals from $(0, 1)$ into a binary tree.

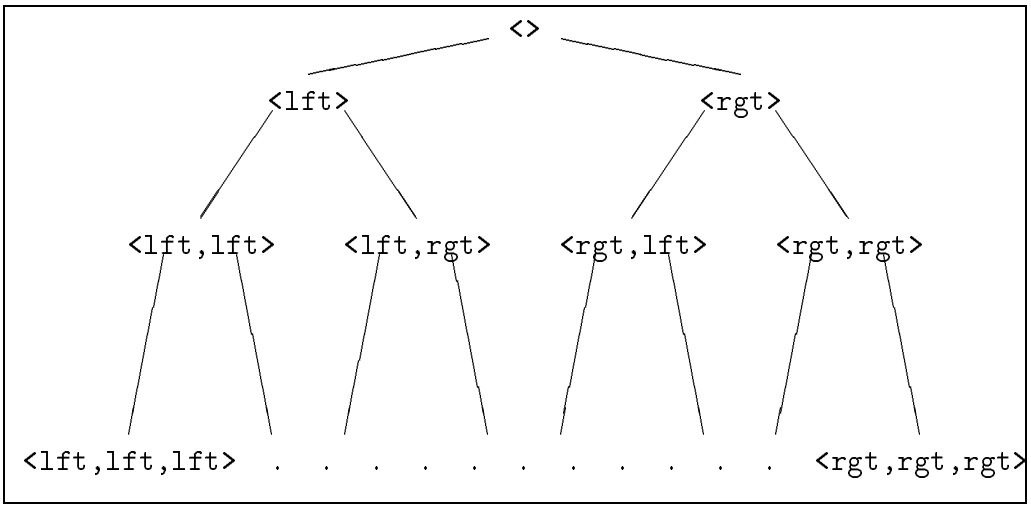


Figure 2: Correspondence between lists over $\{\texttt{lft}, \texttt{rgt}\}$ and tree nodes.

Figure 1 shows an example.

For the purpose of reasoning about the ordering, the rational numbers in $(0,1)$ can be identified with their (uniquely determined!) position (path) in the tree. We encode these paths from the root to the nodes by lists over the alphabet $\{\texttt{lft}, \texttt{rgt}\}$, as illustrated in figure 2. The ordering $\prec$ on the nodes is defined like that in a usual binary search tree, e.g. is[4]

$$\begin{aligned}
&\texttt{<lft,lft>}\\
&\quad\prec\ \texttt{<lft>}\\
&\quad\prec\ \texttt{<lft,rgt>}\\
&\quad\prec\ \texttt{<>}\\
&\quad\prec\ \texttt{<rgt,lft>}\\
&\quad\prec\ \texttt{<rgt>}\\
&\quad\prec\ \texttt{<rgt,rgt>}.
\end{aligned}$$

Since the formalization of the ordering $\prec$ on $(\texttt{lft}, \texttt{rgt})^*$ is much more straightforward than it would be for the rationals, we decided to choose

$$\mathcal{A}_0 := \Big((\texttt{lft}, \texttt{rgt})^*, \prec\Big).$$

Notice that from a theoretical point of view the choice of $\mathcal{A}_0$ does not matter (as long as it is a countable, densely ordered set without endpoints), however from a practical point of view there are important differences. We believe that our choice is a good one because it enables quite natural formalizations of the ordering and the isomorphism.

The idea for constructing an isomorphism $I_{\mathcal{A}} : A \to A_0$ is to sort the elements of A into a binary search tree following the enumeration $a : \mathbb{N} \to A$ of A, and to map each element $a(k)$ onto the corresponding path in A_0. Thus, $a(0)$ is mapped to the empty path `<>`, the first $a(k)$ in the enumeration with $a(k) <_{\mathcal{A}} a(0)$ to `<lft>`, etc. Figure 3 shows an example. Intuitively, it is obvious that this construction yields an isomorphism. An outline of a formal proof is given in section 5.

4 Formalization of the Problem

To prove Cantor's theorem formally, we have to describe the non-empty countable sets with a dense linear order without endpoints within a formal language. We use algebraic first-order specifications in the style of [6]; see the specifications A-SPEC in figure 4 which uses the specification NAT-SPEC in figure 5.

[4]Notice, that $\prec$ is not a lexical ordering.

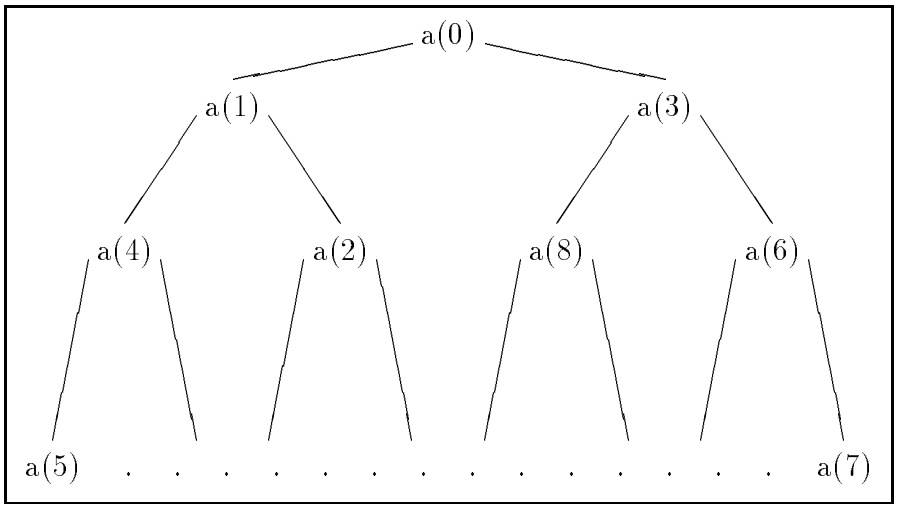


Figure 3: Correspondence between elements of A and tree nodes on the example $a(5) <_{\mathcal{A}} a(4) <_{\mathcal{A}} a(1) <_{\mathcal{A}} a(2) <_{\mathcal{A}} a(0) <_{\mathcal{A}} a(8) <_{\mathcal{A}} a(3) <_{\mathcal{A}} a(6) <_{\mathcal{A}} a(7)$.

specification A-SPEC

using	NAT-SPEC	
sort	s	
function	$a : nat \to s$	
predicate	$\cdot < \cdot : (s, s)$	
axioms		
	$\exists n.\, a(n) = x$	(surjectivity)
	$x < y \wedge y < z \to x < z$	(transitivity)
	$\neg\ x < x$	(irreflexivity)
	$x \neq y \to x < y \vee y < x$	(totality)
	$x < y \to \exists z.\, (x < z \wedge z < y)$	(density)
	$\exists y.\, x < y$	(no right endpoint)
	$\exists y.\, y < x$	(no left endpoint)

end specification

Figure 4: Algebraic specification of countable, densely ordered sets without endpoints.

specification NAT-SPEC

sort nat

functions $0 :\rightarrow nat$

$\cdot + 1 : nat \rightarrow nat$

axioms

nat **freely generated by** $0, +1$

end specification

Figure 5: Algebraic specification of natural numbers. ($+1$ denotes the successor function, written postfix.)

Here, "nat **freely generated by** $0, +1$" is a kind of higher-order axiom which restricts the class of models of the specification to those in which each object of the domain can be *uniquely* denoted by a constructor term, i.e. by a ground term built from the constructors $a, 0, +1$. In particular, this ensures countability of the model for nat, which is transferred to s via the surjectivity axiom. The restrictions[5] of models of the specification A-SPEC to the signature $\Sigma = (\{s\}, \{<\})$ are exactly the algebras $\mathcal{A} = (A, <_{\mathcal{A}})$ with A a non-empty, countable set and $<_{\mathcal{A}}$ a total ordering such that A is dense and without endpoints.

5 The Formal Proof

This section explains the formal proof of Cantor's theorem which we have carried out in the KIV system. As argued above it works in three steps: (1) providing an appropriate algebra $\mathcal{A}_0$, (2) constructing a function $I_{\mathcal{A}} : A \rightarrow A_0$ for each $\mathcal{A}$, and (3) proving that $I_{\mathcal{A}}$ is an isomorphism.

5.1 Step 1: providing an appropriate algebra $\mathcal{A}_0$

We provide the algebra $\mathcal{A}_0 := \left((\texttt{lft}, \texttt{rgt})^*, \prec\right)$ in terms of an implementation. Without further mentioning we assume an implementation of paths, i.e. lists over $\{\texttt{lft}, \texttt{rgt}\}$, with the constructors `<>` for the empty path, and cons for adding either `lft` or `rgt` to a path, and usual selectors head and tail. Thus, it remains to define the ordering $\prec$ on these paths, which is given in figure 6. Notice that termination of the procedure less is obvious (and easy to prove by structural induction over any of the inputs). In what follows we

[5] For a definition of *restriction* see ([6], p. 683).

```
proc less(p1, p2 : path) : bool is
begin
    if p1 = nil then return p2 ≠ nil and then head(p2) = rgt
    else if head(p1) = lft then return p2 = nil
                                    or else head(p2) = rgt
                                    or else less(tail(p1), tail(p2))
    else if head(p1) = rgt then return p2 ≠ nil
                                    and then head(p2) = rgt
                                    and then less(tail(p1), tail(p2))
end;
```

Figure 6: Implementation of $\prec$ on $(\texttt{lft}, \texttt{rgt})^*$.

will use the procedure less like a predicate and write $p_1 \prec p_2$ if $less(p_1, p_2)$ yields **true**.

5.2 Step 2: constructing a function $I_{\mathcal{A}} : A \to A_0$

In this step of the proof an isomorphism $I_{\mathcal{A}} : A \to A_0$ is constructed, i.e. a procedure is given which computes an isomorphism. As already indicated above our procedure works as follows:

An enumeration of A is performed via the enumeration function a. At the same time the procedure follows a path in the imagined binary tree. In particular, at every step in this computation, a kind of interval is maintained, such that all elements of A positioned in the subtree rooted at the current node lie in this interval. For instance, when considering $a(8)$ in figure 3, this would be the open interval $(a(0), a(3))$. Now, for nodes on the leftmost and rightmost branch, we can give only half-intervals as restrictions, and nodes below and including the root, $a(0)$, obviously include all of A.

The implementation of the isomorphism in pseudo-code notation is shown in figure 7. Here we have used a relaxed but convenient notation. The parameters l, r of iso-rec which represent the (left and right border of the) maintained interval are allowed to take values from $A \cup \{-\infty, +\infty\}$.[6] Fur-

[6] In the actual implementation within the KIV system, we did not add infinite elements to A. Instead, we maintained a boolean variable together with each interval border, which indicated whether the border was to be interpreted as such, or taken as infinite.

```
proc iso(a : A) : path is
    proc iso-rec(a : A; l, r : A; k : nat) : path is
    begin
        if not a(k) ∈ (l, r) then
          return iso-rec(a, l, r, k + 1)                    (α)
        else if a = a(k) then
          return <>                                         (β)
        else if a < a(k) then
          return cons(lft, iso-rec(a, l, a(k), k + 1))      (γ)
        else if a(k) < a then
          return cons(rgt, iso-rec(a, a(k), r, k + 1))      (δ)
    end;
begin
    return iso-rec(a, −∞, +∞, 0)
end;
```

Figure 7: Implementation of the isomorphism.

thermore, we define for all $l, r \in A$:

$$\begin{aligned} a \in (l, r) &\text{ iff } l < a \text{ and } a < r \\ a \in (-\infty, r) &\text{ iff } a < r \\ a \in (l, +\infty) &\text{ iff } l < a \\ a \in (-\infty, +\infty) &\text{ holds for all } a \in A. \end{aligned}$$

The procedure basically does a recursion in which A is enumerated via the parameter k. For each value of k, one of four cases can occur. The corresponding code fragments are marked (α) through (δ) in figure 7. Applied to the notion of stepping through a path in a binary tree, (α) means, that $a(k)$ is not in the subtree of the current node (remember, that (l, r) is the interval of A containing all elements of A associated with the subtree rooted at the current node). (β) is the termination case, that is we have reached the node we were looking for. Cases (γ) and (δ) correspond to advancing to the left and right descendant of the current node, respectively.

5.3 Step 3: Proving the Properties

It remains to show that the function computed by the procedure iso is in fact an isomorphism: we have to prove totality, homomorphism property and bijectivity. All the proofs sketched here have been carried out within the KIV system.

5.3.1 Totality

For totality we have to prove the following theorem.

Theorem (termination of iso**)**
The procedure iso *terminates for all possible inputs.*

Proof. It is sufficient to notice, that $a \in (l, r)$ for any of the recursive calls (induction hypothesis). Since all elements of A are enumerated, there is a $k \in \mathbb{N}$ such that $a = a(k)$. In the corresponding call to iso-rec, case (β) will be reached and the recursion terminates. ■

5.3.2 Homomorphism Property

For the homomorphism property we have to prove the following theorem.[7]

Theorem (homomorphism property of iso**)**
$\text{iso}(a_1) \prec \text{iso}(a_2)$ *iff* $a_1 < a_2$ *for all* $a_1, a_2 \in A$.

Proof. The proof is done by reduction to a statement about iso-rec: if

- $a_1, a_2 \in (l, r)$
- $a(k') \notin (l, r) \quad \text{for all} \quad k' < k$
- $a_1 < a_2$

then

$$\text{iso-rec}(a_1, l, r, k) \prec \text{iso-rec}(a_2, l, r, k)$$

The other direction of the equivalence can be shown using this lemma with a_1 and a_2 exchanged.

The proof for iso-rec works by induction over the depth of recursion in one of the two calls, say $\text{iso-rec}(a_1, l, r, k)$. As the parameters l,r and k are equal in the two calls, they can only reach case (α) together, where the induction hypothesis can be applied to the recursive calls. If $\text{iso-rec}(a_1, l, r, k)$ runs into (β), $a_1 = a(k)$ and `<>` is returned. Since $a(k) = a_1 < a_2$, $\text{iso-rec}(a_2, l, r, k)$ will come to case (δ) and return a path with head `rgt`, which is thus greater than the empty path. For case (γ) in $\text{iso-rec}(a_1, l, r, k)$, we must have $a_1 < a(k)$ and a path with head `lft` is returned. Calling $\text{iso-rec}(a_2, l, r, k)$ will yield the empty path resp. a path with head `rgt` in cases (β) resp. (δ). These paths are greater than any path with head `lft`. For case (γ), the induction hypothesis can be applied, as the values of l,r and k passed to the recursive calls are equal. The case, that $\text{iso-rec}(a_1, l, r, k)$ runs into (δ) is symmetric to the (γ) case. ■

[7] We denote the *total* function computed by iso again by iso.

5.3.3 Bijectivity

It remains to prove bijectivity of the function computed by iso.

Theorem (injectivity of iso**)**
For all $a_1, a_2 \in A$*, if* $\mathrm{iso}(a_1) = \mathrm{iso}(a_2)$ *then* $a_1 = a_2$*.*

Proof. The theorem is a consequence of homomorphism property theorem above, since $\mathrm{iso}(a_1) = \mathrm{iso}(a_2)$ iff $\mathrm{iso}(a_1) \not\preceq \mathrm{iso}(a_2)$ and $\mathrm{iso}(a_2) \not\preceq \mathrm{iso}(a_1)$ iff (homomorphism property) $a_1 \not\preceq a_2$ and $a_2 \not\preceq a_1$ iff $a_1 = a_2$. ■

Theorem (surjectivity of iso**)**
For all $p \in (\mathtt{lft}, \mathtt{rgt})^*$ *there is some* $a \in A$ *such that* $\mathrm{iso}(a) = p$*.*

A particularly hard part of Cantor's proof concerned the surjectivity. In our approach, we decided to solve this by implementing a further procedure inv and showing, that it computes the right-inverse[8] of the function computed by iso, that is $\mathrm{iso} \circ \mathrm{inv} = id$. Figure 8 shows the implementation. The procedure inv computes an element of A given a path in $(\mathtt{lft}, \mathtt{rgt})^*$: The parameter p always contains the rest of the path to be considered. The node is thus reached, when p is empty. The above theorem (surjectivity of iso) is a consequence of the following two lemmas.

Lemma 5.1 (termination of inv**)**
The procedure inv *terminates for all possible inputs.*

Proof. For the termination of inv, a different argument as for the termination of iso is needed. Note, that for each triple (l, r, k) of parameters in a recursive call to inv-rec, all previously enumerated $a(j)$ with $j < k$ are not in the interval (l, r) (induction hypothesis). Density, together with the enumeration function, guarantees the existence of a k_0 with $a(k_0) \in (l, r)$, which must obviously be greater than k. This means, that there cannot be an infinite chain of recursions all leading to case (α'). Therefore, p gradually becomes shorter, until it is empty, and finally case (β') is reached. The formal proof works by an induction over the length of p, and then over $k_0 - k$. ■

Lemma 5.2 ($\mathrm{iso} \circ \mathrm{inv} = id$)
If $\mathrm{inv}(p)$ *returns* a*, then a call of* $\mathrm{iso}(a)$ *yields* p*.*

Proof. For the proof it is important to see, that for corresponding $a \in A$ and $l \in L$, calls to $\mathrm{iso}(a)$ resp. $\mathrm{inv}(p)$ lead to the same sequence of triples (l, r, k) in the sequence of recursive calls. Again, this is reduced to a statement about iso-rec and inv-rec. We show, that under the conditions, that

[8] This is also the inverse, because of the injectivivty already shown. We do not need this for the surjectivity, though

```
proc inv(p : path) : A is
    proc inv-rec(p : path; l, r : A; k : nat) : A is
    begin
        if not a(k) ∈ (l, r) then
          return inv-rec(p, l, r, k + 1)                    (α′)
        else if p = <> then
          return a(k)                                       (β′)
        else if head(p) = lft then
          return inv-rec(tail(p), l, a(k), k + 1)           (γ′)
        else if head(p) = rgt then
          return inv-rec(tail(p), a(k), r, k + 1)           (δ′)
    end;
begin
    return inv-rec(p, −∞, +∞, 0)
end;
```

Figure 8: Implementation of the inverse of the isomorphism.

- $a \in (l, r)$
- $a(k') \notin (l, r)$ for all $k' < k$

iso-rec(a, l, r, k) returns p, if inv-rec(p, l, r, k) returns a.[9] For the induction, it is important, that the above conditions are propagated to recursive calls.

The proof makes extensive use of the fact, that a call to inv-rec(p, l, r, k) always returns a value $a \in (l, r)$. The proof of this fact works by induction over recursion depth and is rather straightforward.

The parameter / return-value a has an index, $a = a(k_0)$ with $k_0 < k$, and induction is done over $k_0 - k$. Thus, let inv-rec(p, l, r, k) return $a(k_0)$. We show, that iso-rec$(a(k_0), l, r, k)$ returns p.

The proof splits into four main branches, one for each case that might be reached in inv-rec. We show that in each case possible in inv-rec, the call of iso-rec on the return value leads to the corresponding case in iso-rec, so the induction hypothesis can be applied. As reaching case (α') is determined only by l,r and k, the call to iso-rec must lead to (α). In the remaining cases, (α) can thus not be reached in iso-rec. If (β') is executed in inv-rec, $a(k)$ is returned, so iso-rec reaches (β). In cases (γ') and (δ'), we use the fact mentioned above, that $a(k_0) \in (l, r)$ to establish that $a(k_0) < a(k)$,

[9] Termination has already been proved, so it is no longer an issue here. All procedures are guaranteed to actually return some value.

resp. $a(k) < a(k_0)$. From this, we conclude, that iso-rec must reach the corresponding case and apply the induction hypothesis for the recursive call. ■

5.4 Statistics

Most of the time used to complete this work was spent on the search for an appropriate way to do the proof. When the idea of using paths as standard models was there, specification, implementation and proving took about 13 hours of work done within one week.

Of course, to do the proof, a number of lemmas about lists and natural numbers was required. The proofs concerning the 46 lines of code of the isomorphism, its inverse and one further help procedure (which computes k_0 in the proof of lemma 5.1) totalled 391 proof steps, 108 of which were interactions. Two auxiliary lemmas were needed, namely one for the proof of lemma 5.1[10] and one for the fact that inv-rec$(p, l, r, k) \in (l, r)$, used in the proof of lemma 5.2.

6 Conclusion

We have presented a formal proof of the fact that *any two countable, densely ordered sets without endpoints are isomorphic*. Instead of (directly) formalizing the informal proofs given in [1] or [4], we took another approach for doing this proof. The central technique is to explicitly provide as much as possible by procedures, following the paradigm *"programming is easier than proving"*.[11] This technique comes to fruition at three spots in the proof:

- a representative of the class of countable, densely ordered sets without endpoints is explicitly provided in terms of an implementation,
- the existence of an isomorphism is proved constructively, i.e. by programming it,
- the surjectivity of the isomorphism is proved by implementing its inverse

Thus, the original problem is reduced to a more concrete and more tractable one. Furthermore, one is enabled to employ well-established tools and techniques, known from software-verification.

[10] This proof took 26 interactions and the actual termination proof another 24, which seems to reflect the difficulty of the surjectivity argument.

[11] "Proofs as programs" (c.f. e.g. [3]) is a fairly nice concept from a theoretical point of view. However, we believe that for practical reasons "programs as proofs" is the better choice.

References

[1] G. Cantor. *Gesammelte Abhandlungen*, chapter 9, page 303 ff. Springer, 1932.

[2] C.C. Chang and H.J. Keisler. *Model Theory*, volume 73 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science Publishers B. V., 1990.

[3] R.L. Constable, S.F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W. Harper, D.J. Howe, T.B. Knoblock, N.P. Mendler, P. Panangaden, J.T. Sasaki, and S.F. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice Hall, 1986.

[4] E. Kamke. *Mengenlehre*, volume 999/999a of *Sammlung Göschen*, page 100 ff. de Gruyter, 1962.

[5] W. Reif. The KIV-system: Systematic construction of verified software. In *Proceedings of the 11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Computer Science*. Springer Verlag, 1992.

[6] M. Wirsing. *Algebraic Specification*, volume B of *Handbook of Theoretical Computer Science*, chapter 13, pages 675–788. Elsevier Science Publishers B. V., 1990.