Formalizing Results on Directed Sets

Akihisa Yamada and Jérémy Dubut

May 2, 2024

Abstract

Directed sets are of fundamental interest in domain theory and topology. In this paper, we formalize some results on directed sets in Isabelle/HOL, most notably: under the axiom of choice, a poset has a supremum for every directed set if and only if it does so for every chain; and a function between such posets preserves suprema of directed sets if and only if it preserves suprema of chains. The known pen-and-paper proofs of these results crucially use uncountable transfinite sequences, which are not directly implementable in Isabelle/HOL. We show how to emulate such proofs by utilizing Isabelle/HOL's ordinal and cardinal library. Thanks to the formalization, we relax some conditions for the above results.

Contents

1 Introduction

A *directed set* is a set D equipped with a binary relation $\sqsubseteq$ such that any finite subset $X \subseteq D$ has an upper bound in D with respect to $\sqsubseteq$. The property is often equivalently stated that D is non-empty and any two elements $x, y \in D$ have a bound in D, assuming that $\sqsubseteq$ is transitive (as in posets).

Directed sets find uses in various fields of mathematics and computer science. In topology (see for example the textbook [7]), directed sets are used to generalize the set of natural numbers: sequences $\mathbb{N} \to A$ are generalized to *nets* $D \to A$, where D is an arbitrary directed set. For example, the usual result on metric spaces that continuous functions are precisely functions that preserve limits of sequences can be generalized in general topological spaces as: the continuous functions are precisely functions that preserve limits of nets. In domain theory [1], key ingredients are *directed-complete posets*, where every directed subset has a supremum in the poset, and *Scott-continuous functions* between posets, that is, functions that preserve suprema of directed sets. Thanks to their fixed-point properties (which we have formalized in Isabelle/HOL in a previous work [5]), directed-complete posets naturally appear in denotational semantics of languages with loops or fixed-point operators (see for example Scott domains [11, 13]). Directed sets also appear in reachability and coverability analyses of transition systems through the notion of ideals, that is, downward-closed directed sets. They allow effective representations of objects, making forward and backward analysis of well-structured transition systems – such as Petri nets – possible (see e.g., [6]).

Apparently milder generalizations of natural numbers are chains (totally ordered sets) or even well-ordered sets. In the mathematics literature, the following results are known (assuming the axiom of choice):

Theorem 1 **([4])** *A poset is directed-complete if (and only if) it has a supremum for every non-empty well-ordered subset.*

Theorem 2 **([9])** *Let f be a function between posets, each of which has a supremum for every non-empty chain. If f preserves suprema of non-empty chains, then it is Scott-continuous.*

The pen-and-paper proofs of these results use induction on cardinality, where the finite case is merely the base case. The core of the proof is a technical result called Iwamura's Lemma [8], where the countable case is merely an easy case, and the main part heavily uses transfinite sequences indexed by uncountable ordinals.

To formalize these results in Isabelle/HOL we extensively use the existing library for ordinals and cardinals [3], but we needed some delicate work in emulating the pen-and-paper proofs. In Isabelle/HOL, or any proof assistant based on higher-order logic (HOL), it is not possible to have a datatype for arbitrarily large ordinals; hence, it is not possible to directly formalize transfinite sequences. We show how to emulate transfinite sequences using the ordinal and cardinal library [3]. As far as the authors know, our work is the first to mechanize the proof of Theorems 1 and 2, as well as Iwamura's Lemma. We prove the two theorems for quasi-ordered sets, relaxing antisymmetry, and strengthen Theorem 2 so that chains are replaced by well-ordered sets and conditions on the codomain are completely dropped.

Related Work Systems based on Zermelo-Fraenkel set theory, such as Mizar [2] and Isabelle/ZF [10], have more direct support for ordinals and cardinals and should pose less challenge in mechanizing the above results. Nevertheless, a part of our contribution is in demonstrating that the power of (Isabelle/)HOL is strong enough to deal with uncountable transfinite sequences.

Except for the extra care for transfinite sequences, our proof of Iwamura's Lemma is largely based on the original proof from [8]. Markowsky presented a proof of Theorem 1 using Iwamura's Lemma [9, Corollary 1]. While he took a minimal-counterexample approach, we take a more constructive approach to build a well-ordered set of suprema. This construction was crucial to be reused in the proof of Theorem 2, which Markowsky claimed without a proof [9]. Another proof of Theorem 1 can be found in [4], without using Iwamura's Lemma, but still crucially using transfinite sequences.

This work has been published in the conference paper [14].

2 Preliminaries

2.1 Connecting Predicate-Based and Set-Based Relations

```
theory Well-Order-Connection
  imports
    Main
    Complete-Non-Orders.Well-Relations
begin

lemma refl-on-relation-of: refl-on A (relation-of r A) ⟷ reflexive A r
  by (auto simp: refl-on-def reflexive-def relation-of-def)

lemma trans-relation-of: trans (relation-of r A) ⟷ transitive A r
  by (auto simp: trans-def relation-of-def transitive-def)

lemma preorder-on-relation-of: preorder-on A (relation-of r A) ⟷ quasi-ordered-set
A r
 by (simp add: preorder-on-def refl-on-relation-of trans-relation-of quasi-ordered-set-def)

lemma antisym-relation-of: antisym (relation-of r A) ⟷ antisymmetric A r
  by (auto simp: antisym-def relation-of-def antisymmetric-def)

lemma partial-order-on-relation-of:
  partial-order-on A (relation-of r A) ⟷ partially-ordered-set A r
  by (auto simp: partial-order-on-def preorder-on-relation-of antisym-relation-of
      quasi-ordered-set-def partially-ordered-set-def)
```

```
lemma total-on-relation-of: total-on A (relation-of r A) ⟷ semiconnex A r
  by (auto simp: total-on-def relation-of-def semiconnex-def)

lemma linear-order-on-relation-of:
  shows linear-order-on A (relation-of r A) ⟷ total-ordered-set A r
  by (auto simp: linear-order-on-def partial-order-on-relation-of total-on-relation-of
      total-ordered-set-def total-quasi-ordered-set-def partially-ordered-set-def
      connex-iff-semiconnex-reflexive)

lemma relation-of-sub-Id: (relation-of r A − Id) = relation-of (λx y. r x y ∧ x ≠
y) A
  by (auto simp: relation-of-def)

lemma (in antisymmetric) asympartp-iff-weak-neq:
  shows x ∈ A ⟹ y ∈ A ⟹ asympartp (⊑) x y ⟷ x ⊑ y ∧ x ≠ y
  by (auto intro!: asympartpI antisym)

lemma wf-relation-of: wf (relation-of r A) = well-founded A r
 apply (simp add: wf-eq-minimal relation-of-def well-founded-iff-ex-extremal Ball-def)
  by (metis (no-types, opaque-lifting) equals0I insert-Diff insert-not-empty  subsetI
subset-iff)

lemma well-order-on-relation-of:
  shows well-order-on A (relation-of r A) ⟷ well-ordered-set A r
  by (auto simp: well-order-on-def linear-order-on-relation-of relation-of-sub-Id
      wf-relation-of well-ordered-iff-well-founded-total-ordered
      antisymmetric.asympartp-iff-weak-neq total-ordered-set-def
      cong: well-founded-cong)

lemma (in connex) Field-relation-of: Field (relation-of (⊑) A) = A
  by (auto simp: Field-def relation-of-def)

lemma (in well-ordered-set) Well-order-relation-of:
  shows Well-order (relation-of (⊑) A)
 by (auto simp: Field-relation-of well-order-on-relation-of well-ordered-set-axioms)

lemma in-relation-of: (x,y) ∈ relation-of r A ⟷ x ∈ A ∧ y ∈ A ∧ r x y
  by (simp add: relation-of-def)

lemma relation-of-triv: relation-of (λx y. (x,y) ∈ r) UNIV = r
  by (auto simp: relation-of-def)

lemma Restr-eq-relation-of: Restr R A = relation-of (λx y. (x,y) ∈ R) A
  by (auto simp: relation-of-def)

theorem ex-well-order: ∃ r. well-ordered-set A r
proof−
  from well-order-on obtain R where R: well-order-on A R by auto
  then have well-order-on A (Restr R A)
```

```
    by (simp add: well-order-on-Field[OF R] Restr-Field)
  then show ?thesis by (auto simp: Restr-eq-relation-of well-order-on-relation-of)
qed


end
theory Directed-Completeness
  imports
    Complete-Non-Orders.Continuity
    Well-Order-Connection
    HOL−Cardinals.Cardinals
    HOL−Library.FuncSet
begin
```

2.2 Missing Lemmas

```
no-notation disj  (infixr | 30)

lemma Sup-funpow-mono:
  fixes f :: ′a :: complete-lattice ⇒ ′a
  assumes mono: mono f
  shows mono (⨆ i. f ^^ i)
  by (intro monoI, auto intro!: Sup-mono dest: funpow-mono[OF mono])

lemma iso-imp-compat:
  assumes iso: iso r r′ f shows compat r r′ f
  by (simp add: compat-def iso iso-forward)

lemma iso-inv-into:
  assumes ISO: iso r r′ f
  shows iso r′ r (inv-into (Field r) f)
  using assms unfolding iso-def
  using bij-betw-inv-into inv-into-Field-embed-bij-betw by blast

lemmas iso-imp-compat-inv-into = iso-imp-compat[OF iso-inv-into]

lemma infinite-iff-natLeq: infinite A ⟷ natLeq ≤o |A|
  using infinite-iff-natLeq-ordLeq by blast
```

As we cannot formalize transfinite sequences directly, we take the following approach: We just use A as the index set, and instead of the ordering on ordinals, we take the well-order that is chosen by the cardinality library to denote $|A|$.

```
definition well-order-of ((′(≼-′)) [0]1000) where (≼A) x y ≡ (x,y) ∈ |A|

abbreviation well-order-le (- ≼- - [51,0,51]50) where x ≼A y ≡ (≼A) x y

abbreviation well-order-less (- ≺- - [51,0,51]50) where x ≺A y ≡ asympartp
(≼A) x y
```

```
lemmas well-order-ofI = well-order-of-def[unfolded atomize-eq, THEN iffD2]
lemmas well-order-ofD = well-order-of-def[unfolded atomize-eq, THEN iffD1]

lemma carrier: assumes x ≼_A y shows x ∈ A and y ∈ A
  using assms by (auto dest!: well-order-ofD dest: FieldI1 FieldI2)

lemma relation-of[simp]: relation-of (≼_A) A = |A|
  by (auto simp: relation-of-def well-order-of-def dest: FieldI1 FieldI2)

interpretation well-order-of: well-ordered-set A (≼_A)
  apply (fold well-order-on-relation-of)
  by auto
```

Thanks to the well-order theorem, one can have a sequence $\{A_\alpha\}_{\alpha<|A|}$ of subsets of A that satisfies the following three conditions:

- cardinality: $|A_\alpha| < |A|$ for every $\alpha < |A|$,
- monotonicity: $A_\alpha \subseteq A_\beta$ whenever $\alpha \leq \beta < |A|$, and
- range: if A is infinite, $A = \bigcup_{\alpha<|A|} A_\alpha$.

The following serves the purpose.

```
definition Pre (-≺ [1000]1000) where A≺ a ≡ {b ∈ A. b ≺_A a}

lemma Pre-eq-underS: A≺ a = underS |A| a
  by (auto simp: Pre-def underS-def well-order-ofD carrier well-order-of.antisym
dest!: well-order-ofI)

lemma Pre-card: assumes aA: a ∈ A shows |A≺ a| <o |A|
  by (auto simp: Pre-eq-underS aA intro!: card-of-underS[OF card-of-Card-order])

lemma Pre-carrier: A≺ a ⊆ A by (auto simp: Pre-def)

lemma Pre-mono: monotone-on A (≼_A) (⊆) (A≺)
 by (auto intro!: monotone-onI simp: Pre-def dest: well-order-of.asym-trans well-order-of.asym.irrefl)

lemma extreme-imp-finite:
  assumes e: extreme A (≼_A) e shows finite A
proof (rule ccontr)
  assume inf: infinite A
  from e have eA: e ∈ A by auto
  from e have A = {a ∈ A. a ≼_A e} by auto
  also have ... − {e} = A≺ e
    using eA by (auto simp: Pre-def dest: well-order-of.asympartp-iff-weak-neq)
  finally have AeP: A − {e} = ....
  have infinite (A − {e}) using infinite-remove[OF inf].
  with AeP have infP: infinite (A≺ e) by simp
  have A = insert e (A≺ e) using eA by (fold AeP, auto)
  also have |. . . | =o |A≺ e| using infinite-card-of-insert[OF infP].
  finally have |A≺ e| =o |A| using ordIso-symmetric by auto
  with Pre-card[OF eA] not-ordLess-ordIso
  show False by auto
qed

lemma infinite-imp-ex-Pre:
  assumes inf: infinite A and xA: x ∈ A shows ∃ y ∈ A. x ∈ A≺ y
proof-
  from inf
  have ¬ extreme A (⪯_A) x by (auto dest!: extreme-imp-finite)
  with xA obtain y where yA: y ∈ A and ¬ y ⪯_A x by auto
  with xA have x ≺_A y by (auto simp: well-order-of.not-weak-iff asympartpI)
  with yA show ?thesis by (auto simp: Pre-def xA)
qed

lemma infinite-imp-Un-Pre: assumes inf: infinite A shows ⋃(A≺ ‘ A) = A
proof (safe)
  fix x assume xA: x ∈ A
  show y ∈ A≺ x ⟹ y ∈ A for y using Pre-carrier[of A x] by auto
  from infinite-imp-ex-Pre[OF inf xA]
  show x ∈ ⋃ (A≺ ‘ A) by (auto simp: Pre-def)
qed
```

3 Iwamura’s lemma

As the proof involves a number of (inductive) definitions, we build a locale for collecting those definitions and lemmas.

```
locale Iwamura-proof = related-set +
  assumes dir: directed-set A (⊑)
begin
```

Inside this locale, a related set $(A, \sqsubseteq)$ is fixed and assumed to be directed. The proof starts with declaring, using the axiom of choice, a function f that chooses a bound $f\ X \in A$ for every finite subset $X \subseteq A$. This function can be formalized using the SOME construction:

```
definition f where f X ≡ SOME z. z ∈ A ∧ bound X (⊑) z

lemma assumes XA: X ⊆ A and Xfin: finite X
  shows f-carrier: f X ∈ A and f-bound: bound X (⊑) (f X)
  using directed-setD[OF dir XA Xfin, unfolded Bex-def, THEN someI-ex]
  by (auto simp: f-def)
```

3.1 Uncountable Case

Actually, the main part of the proof of Iwamura’s Lemma is about monotonically expanding an infinite subset (in particular A_α) of A into a directed one, without changing the cardinality. To this end, Iwamura's original proof introduces a function $F\colon \mathit{PowA} \to \mathit{PowA}$ that expands a set with upper bounds of *all finite subsets*. This approach is different from Markowsky's reproof (based on [12]) which uses nested transfinite induction to extend a set one element after another.

```
definition F where F X ≡ X ∪ f ‘ Fpow X

lemma F-carrier: X ⊆ A ⟹ F X ⊆ A
  and F-infl: X ⊆ F X
  and F-fin: finite X ⟹ finite (F X)
  by (auto simp: F-def Fpow-def f-carrier)

lemma F-card: assumes inf: infinite X shows |F X| =o |X|
proof–
  have |f ‘ Fpow X| ≤o |Fpow X| using card-of-image.
  thm card-of-Fpow-infinite
  also have |Fpow X| =o |X| using card-of-Fpow-infinite[OF inf].
  finally have |f ‘ Fpow X| ≤o |X|.
  with inf show ?thesis by (auto simp: F-def)
qed

lemma F-mono: mono F
proof(intro monoI)
  show X ⊆ Y ⟹ F X ⊆ F Y for X Y
    using Fpow-mono[of X Y] by (auto simp: F-def)
qed

lemma Fn-carrier: X ⊆ A ⟹ (F ^^ n) X ⊆ A
  and Fn-infl: X ⊆ (F ^^ n) X
  and Fn-fin: finite X ⟹ finite ((F ^^ n) X)
  and Fn-card: infinite X ⟹ |(F ^^ n) X| =o |X|
proof (atomize(full), induct n)
  case (Suc n)
  define Y where Y ≡ (F^^n) X
  then have ∗: (F ^^ Suc n) X = F Y by auto
  from Suc[folded Y-def]
  have infinite X ⟹ infinite Y ∧ |Y| =o |X|
    and finite X ⟹ finite Y
    and X ⊆ Y
    and X ⊆ A ⟹ Y ⊆ A by (auto simp: Y-def)
  with F-carrier[of Y] F-infl[of Y] F-card[of Y] F-fin[of Y]
  show ?case by (unfold ∗, auto del:subsetI dest:ordIso-transitive)
qed auto

lemma Fn-mono1: i ≤ j ⟹ (F ^^ i) X ⊆ (F ^^ j) X for i j
  using Fn-infl[of (F^^i) X j−i] funpow-add[of j−i i F]
  by auto
```

We take the ω-iteration of the monotone function F, namely:

```
definition Flim (F^ω) where F^ω X ≡ ⋃ i. (F ^^ i) X

lemma Flim-mono: mono F^ω
proof–
  have F^ω = (⨆ range ((^^) F)) by (auto simp: Flim-def)
  with Sup-funpow-mono[OF F-mono]
  show ?thesis by auto
qed

lemma Flim-infl: X ⊆ F^ω X
  using Fn-infl by (auto simp: Flim-def)

lemma Flim-carrier: assumes X ⊆ A shows F^ω X ⊆ A
  using Fn-carrier[OF assms] by (auto simp: Flim-def)

lemma Flim-directed: assumes X ⊆ A shows directed-set (F^ω X) (⊑)
proof (safe intro!: directed-setI)
  fix Y assume YC: Y ⊆ F^ω X and finY: finite Y
  from finY YC have ∃ i. Y ⊆ (F ^^ i) X
  proof (induct)
    case empty
    then show ?case by auto
  next
    case (insert y Y)
     then obtain i j where Yi: Y ⊆ (F ^^ i) X and y ∈ (F ^^ j) X by (auto
simp: Flim-def)
   with Fn-mono1[OF max.cobounded1[of i j], of X] Fn-mono1[OF max.cobounded2[of
j i], of X]
    show ?case by (auto intro!: exI[of - max i j])
  qed
  then obtain i where Yi: Y ⊆ (F ^^ i) X by auto
  with Fn-carrier[OF assms] have YA: Y ⊆ A by auto
  from Yi finY have f Y ∈ (F ^^ Suc i) X by (auto simp: F-def Fpow-def)
  then have f Y ∈ F^ω X by (auto simp: Flim-def simp del: funpow.simps)
  with f-bound[OF YA finY]
  show ∃ z∈F^ω X. bound Y (⊑) z by auto
qed

lemma Flim-card: assumes infinite X shows |F^ω X| =o |X|
proof–
 from assms have natX: |UNIV :: nat set| ≤o |X| by (simp add: infinite-iff-card-of-nat)
  have |F^ω X| ≤o |X|
    apply (unfold Flim-def, rule card-of-UNION-ordLeq-infinite[OF assms natX])
    using Fn-card[OF assms] ordIso-imp-ordLeq
    by auto
  with Flim-infl show |F^ω X| =o |X| by (simp add: ordIso-iff-ordLeq)
qed

lemma Flim-fin: assumes finite X shows |F^ω X| ≤o natLeq
```

```
proof–
  have |F^ω X| ≤o |UNIV :: nat set|
    apply (unfold Flim-def)
    apply (rule card-of-UNION-ordLeq-infinite)
    by (auto simp: Fn-fin[OF assms] intro!: ordLess-imp-ordLeq)
  then show ?thesis using card-of-nat ordLeq-ordIso-trans by auto
qed

lemma mono-uncountable: monotone-on A (≼_A) (⊆) (F^ω ∘ A_≺)
  using monotone-on-o[OF Flim-mono Pre-mono]
  by (auto simp: o-def)

lemma card-uncountable:
  assumes aA: a ∈ A and unc: natLeq <o |A|
  shows |F^ω (A_≺ a)| <o |A|
proof (cases finite (A_≺ a))
  case True
  note Flim-fin[OF this]
  also note unc
  finally show ?thesis
    using unc not-ordLess-ordIso by auto
next
  case False
  note Flim-card[OF this]
  also note Pre-card[OF aA]
  finally show ?thesis using unc not-ordLess-ordIso by auto
qed

lemma in-I-uncountable:
  assumes aA: a ∈ A and inf: infinite A
  shows ∃ a′ ∈ A. a ∈ F^ω (A_≺ a′)
  using infinite-imp-ex-Pre[OF inf aA] Flim-infl
  by auto

lemma carrier-uncountable:
  shows F^ω (A_≺ a) ⊆ A
  using Flim-carrier[OF Pre-carrier]
  by auto

lemma range-uncountable: assumes inf: infinite A shows ⋃((F^ω ∘ A_≺) ‘ A) =
A
proof (safe intro!: subset-antisym)
  fix a assume aA: a ∈ A
  from infinite-imp-ex-Pre[OF inf aA] Flim-infl
  show a ∈ ⋃ ((F^ω ∘ A_≺) ‘ A) by auto
  show x ∈ (F^ω ∘ A_≺) a ⟹ x ∈ A for x
    using carrier-uncountable by auto
qed
```

```
lemma infl-uncountable:
  assumes aA: a ∈ A and bA: b ∈ A and ab: a ≺_A b
  shows a ∈ F^ω (A_≺ b)
  using assms Flim-infl[of A_≺ b]
  by (auto simp: Pre-def)
```

3.2 Countable Case

```
context
  assumes countable: |A| =o natLeq
begin
```

The assumption above means that there exists an order-isomorphism between $(\mathbb{N}, \leq)$ and $(A, \preceq_A)$.

```
definition seq :: nat ⇒ 'a where seq ≡ SOME f. iso natLeq |A| f

lemma seq-iso: iso natLeq |A| seq
  apply (unfold seq-def)
  apply (rule someI-ex[of iso natLeq |A|])
  using countable[THEN ordIso-symmetric]
  apply (unfold ordIso-def) by auto

lemma seq-bij-betw: bij-betw seq UNIV A
  using seq-iso by (auto simp: iso-def Field-natLeq)
```

This means that A has been indexed by $\mathbb{N}$.

```
lemma range-seq: range seq = A
  using seq-bij-betw bij-betw-imp-surj-on by force

lemma seq-mono: monotone (≤) (⪯_A) seq
  using iso-imp-compat[OF seq-iso]
  by (auto intro!: monotoneI well-order-ofI simp: compat-def natLeq-def)

lemma inv-seq-mono: monotone-on A (⪯_A) (≤) (inv seq)
  using iso-imp-compat-inv-into[OF seq-iso]
  unfolding Field-natLeq
  by (auto intro!: monotone-onI simp: natLeq-def compat-def well-order-of-def)
```

We turn the sequence into a sequence of directed subsets of A:

```
fun Seq :: nat ⇒ 'a set where
  Seq 0 = {f {}}
| Seq (Suc n) = Seq n ∪ {seq n, f (Seq n ∪ {seq n})}

lemma seq-n-in-Seq-n: seq n ∈ Seq (Suc n) by auto

lemma Seq-finite: finite (Seq n)
  by (induction n) auto

lemma Seq-card: |Seq n| <o |A|
  using countable Seq-finite by (simp add: ordIso-natLeq-infinite1)

lemma Seq-carrier: Seq n ⊆ A
proof(induction n)
  case 0
  show ?case by (auto intro!: f-carrier)
next
  case (Suc n)
  with range-seq have sgA: Seq n ∪ {seq n} ⊆ A by auto
  from Seq-finite f-carrier[OF sgA]
  have f (Seq n ∪ {seq n}) ∈ A by auto
  with sgA show ?case by auto
qed

lemma Seq-range: ⋃(range Seq) = A
proof (intro equalityI)
  from Seq-carrier show ⋃(range Seq) ⊆ A by auto
  show A ⊆ ⋃(range Seq)
  proof
    fix a assume aA: a ∈ A
    with seq-bij-betw obtain n where a = seq n
      by (metis bij-betw-inv-into-right)
    with seq-n-in-Seq-n show a ∈ ⋃(range Seq) by (auto intro!: exI[of - Suc n])
  qed
qed

lemma Seq-extremed:
  assumes refl: reflexive A (⊑) shows extremed (Seq n) (⊑)
proof –
  interpret reflexive using refl.
  show ?thesis
  proof(induction n)
    case 0
    show ?case by (auto intro!: extremedI extremeI f-carrier)
  next
    case (Suc n)
    show ?case
    proof (intro extremedI extremeI)
      show f (Seq n ∪ {seq n}) ∈ Seq (Suc n) by auto
      fix x assume xssn: x ∈ Seq (Suc n)
      show x ⊑ f (Seq n ∪ {seq n})
      proof(cases x ∈ Seq n ∪ {seq n})
        case True
        with f-bound[of Seq n ∪ {seq n}] range-seq Seq-finite[of n]
          Seq-carrier[of n]
        show ?thesis by (auto simp: bound-def)
      next
        case False
        with xssn have x: x = f (Seq n ∪ {seq n}) by auto
    from range-seq Seq-finite[of n] Seq-carrier[of n]
    show ?thesis by (auto simp: x intro!: f-carrier)
   qed
  qed
 qed
qed

lemma Seq-directed: assumes refl: reflexive A (⊑) shows directed-set (Seq n) (⊑)
 using Seq-extremed[OF refl] by (simp add: directed-set-iff-extremed[OF Seq-finite])

lemma range-countable: ⋃((Seq ∘ inv seq) ‘ A) = A
 apply (fold image-comp)
 apply (unfold bij-betw-imp-surj-on[OF bij-betw-inv-into[OF seq-bij-betw]])
 using Seq-range.

lemma Seq-mono: mono Seq
proof (intro monoI)
 show n ≤ m ⟹ Seq n ⊆ Seq m for n m by (induct rule:inc-induct, auto)
qed

lemma mono-countable: monotone-on A (⪯_A) (⊆) (Seq ∘ inv seq)
 by (rule monotone-on-o[OF Seq-mono inv-seq-mono]) auto

lemma infl-countable:
 assumes aA: a ∈ A and bA: b ∈ A and ab: a ≺_A b
 shows a ∈ Seq (inv seq b)
proof–
 from aA seq-bij-betw seq-n-in-Seq-n
 have a: a ∈ Seq (Suc (inv seq a)) by (simp add: bij-betw-inv-into-right)
 from ab have inv seq a < inv seq b
   by (metis (mono-tags, lifting) aA well-order-of.asympartp-iff-weak-neq bA
range-seq inv-seq-mono inv-into-injective not-le-imp-less ord.mono-onD verit-la-disequality)
 then have Suc (inv seq a) ≤ inv seq b by auto
 from a monoD[OF Seq-mono this] have a ∈ Seq (inv seq b) by auto
 then show ?thesis by auto
qed

end
```

To match the types, we use the inverse *inv seq* of the isomorphism isaseq. We define the final I as follows:

```
definition I where I ≡ if |A| =o natLeq then Seq ∘ inv seq else F^ω ∘ A_≺

lemma I-carrier: I a ⊆ A
 using Seq-carrier carrier-uncountable by (auto simp: I-def)

lemma I-directed: assumes reflexive A (⊑) shows directed-set (I a) (⊑)
 using Seq-directed[OF - assms] Flim-directed[OF Pre-carrier]
 by (auto simp: I-def)
```

lemma *I-mono*: *monotone-on A* ($\preceq_A$) ($\subseteq$) *I*
  by (*auto simp*: *mono-uncountable mono-countable I-def*)

lemma *I-card*:
  assumes *inf*: *infinite A* **and** *aA*: $a \in A$
  shows $|I\ a| <o\ |A|$
proof (*cases* $|A| =o$ *natLeq*)
  case *True*
  with *Seq-finite*[*OF this*] **show** *?thesis* **by** (*simp add*: *I-def inf*)
next
  case *F*: *False*
  with *inf* **have** *natLeq* $<o\ |A|$
   by (*auto simp*: *infinite-iff-natLeq ordLeq-iff-ordLess-or-ordIso ordIso-symmetric*)
  from *card-uncountable*[*OF aA this*] **show** *?thesis* **by** (*auto simp*: *I-def F*)
qed

lemma *I-range*: **assumes** *inf*: *infinite A* **shows** $\bigcup(I`A) = A$
  using *range-uncountable*[*OF inf*] *range-countable* **by** (*auto simp*: *I-def*)

lemma *I-infl*: **assumes** $a \in A$ $b \in A$ $a \prec_A b$ **shows** $a \in I\ b$
  using *infl-countable infl-uncountable assms* **by** (*auto simp*: *I-def*)

end

Now we close the locale *Iwamura-proof* and state the final result in the global scope.

theorem (**in** *reflexive*) *Iwamura*:
  assumes *dir*: *directed-set A* ($\sqsubseteq$) **and** *inf*: *infinite A*
  shows $\exists I.\ (\forall a \in A.$ *directed-set* $(I\ a)$ $(\sqsubseteq) \wedge |I\ a| <o\ |A|\ ) \wedge$
    monotone-on A ($\preceq_A$) ($\subseteq$) $I \wedge \bigcup(I`A) = A$
proof−
  interpret *Iwamura-proof* **using** *dir* **by** *unfold-locales*
  show *?thesis* **using** *I-mono I-card*[*OF inf*] *I-directed I-range*[*OF inf*]
    by (*auto intro*!: *exI*[*of - I*])
qed

4 Directed Completeness and Scott-Continuity

abbreviation *nonempty A* ≡ *if* $A = \{\}$ *then* ⊥ *else* ⊤

lemma (**in** *quasi-ordered-set*) *directed-completeness-lemma*:
  fixes *leB* (**infix** $\trianglelefteq$ *50*)
  assumes *comp*: (*nonempty* ⊓ *well-related-set*)−*complete A* ($\sqsubseteq$) **and** *dir*: *directed-set D* ($\sqsubseteq$) **and** *DA*: $D \subseteq A$
  shows $\exists s.$ *extreme-bound A* ($\sqsubseteq$) *D s*
   and *well-related-set*−*continuous A* ($\sqsubseteq$) *B* ($\trianglelefteq$) $f \Longrightarrow$
     $D \neq \{\} \Longrightarrow$ *extreme-bound A* ($\sqsubseteq$) *D t* $\Longrightarrow$ *extreme-bound B* ($\trianglelefteq$) (*f* ‘ *D*) (*f t*)

```
proof (atomize(full), insert wf-ordLess dir DA, induct |D| arbitrary: D t rule:
wf-induct-rule)
  interpret less-eq-symmetrize.
  case less
  note this(1)
  note IH = this[THEN conjunct1]
    and IH2 = this[THEN conjunct2, rule-format]
  note DA = ‹D ⊆ A›
  interpret D: quasi-ordered-set D (⊑) using quasi-ordered-subset[OF DA].
  note dir = ‹directed-set D (⊑)›
  show ?case
  proof(cases finite D)
    case True
    from directed-set-iff-extremed[OF True] dir
      obtain d where dD: d ∈ D and exd: extreme D (⊑) d by (auto simp:
extremed-def)
    then have dd: d ⊑ d by (auto simp: extreme-def)
    show ?thesis
    proof(intro conjI allI impI exI[of - d])
      from extreme-imp-extreme-bound[OF exd DA]
      show exbd: extreme-bound A (⊑) D d by auto
      assume f: well-related-set−continuous A (⊑) B (⊴) f
        and Dt: extreme-bound A (⊑) D t and D0: D ≠ {}
      from f[THEN continuous-carrierD] have fAB: f ‘ A ⊆ B by auto
      from Dt have tA: t ∈ A by auto
      show extreme-bound B (⊴) (f ‘ D) (f t)
      proof (safe intro!: extreme-boundI)
        from fAB tA show f t ∈ B by auto
        fix x assume xD: x ∈ D
        from xD Dt have xt: x ⊑ t by auto
        have monotone-on A (⊑) (⊴) f
          by (auto intro!: continuous-imp-monotone-on[OF f] pair-well-related)
        from monotone-onD[OF this] xD DA tA xt
        show f x ⊴ f t by (auto simp: bound-empty extreme-def)
      next
        fix b assume bound (f ‘ D) (⊴) b and bB: b ∈ B
        with dD have fdb: f d ⊴ b by auto
        from Dt exbd have dt: d ∼ t by (auto simp: extreme-bound-iff)
        from dD DA have dA: d ∈ A by auto
        with extreme-bound-sym-trans[OF - extreme-bound-singleton[OF dA] dt tA]
        have extreme-bound A (⊑) {d} t by auto
        from dD DA f[THEN continuousD, OF well-related-singleton-refl - - this]
        have exfdt: extreme-bound B (⊴) {f d} (f t) by auto
        from fdb bB exfdt show f t ⊴ b by auto
      qed
    qed
  next
    case inf: False
    from D.Iwamura[OF dir inf]
    obtain I where Imono: monotone-on D (⪯_D) (⊆) I
      and Icard: ∀ a ∈ D. |I a| <o |D|
      and Idir: ∀ a ∈ D. directed-set (I a) (⊑)
      and Irange: ⋃ (I ‘ D) = D
      by auto
    have ∀ d ∈ D. ∃ s. extreme-bound A (⊑) (I d) s
    proof safe
      fix d assume dD: d ∈ D
      with Irange DA have IdA: I d ⊆ A by auto
      with IH Icard Idir dD range DA
      show ∃ s. extreme-bound A (⊑) (I d) s by auto
    qed
    from bchoice[OF this]
    obtain s where s: ⋀d. d ∈ D ⟹ extreme-bound A (⊑) (I d) (s d) by auto
    then have sDA: s ‘ D ⊆ A by auto
    have smono: monotone-on D (⪯_D) (⊑) s
    proof (intro monotone-onI)
      fix x y assume xD: x ∈ D and yD: y ∈ D and xy: x ⪯_D y
      show s x ⊑ s y
        apply (rule extreme-bound-subset[OF monotone-onD[OF Imono xD yD xy],
of A])
        using s xD yD by auto
    qed
    from well-order-of.monotone-image-well-related[OF this]
    have wsD: well-related-set (s ‘ D) (⊑).
    from inf have sD0: nonempty (s ‘ D) (⊑) by auto
    from completeD[OF comp sDA] wsD sD0
    obtain x where x: extreme-bound A (⊑) (s ‘ D) x by auto
    show ?thesis
    proof (intro conjI allI impI exI[of - x])
      show Dx: extreme-bound A (⊑) D x
      proof (intro smono exI[of - x] extreme-boundI)
        from x show xA: x ∈ A by auto
        fix d assume dD: d ∈ D
        with Irange obtain d′ where d′D: d′ ∈ D and d ∈ I d′ by auto
        with s have 1: d ⊑ s d′ by auto
        from x d′D have 2: . . . ⊑ x by auto
        from trans[OF 1 2] show d ⊑ x using dD sDA d′D DA xA by auto
      next
        fix b assume bA: b ∈ A and Db: bound D (⊑) b
        have bound (s ‘ D) (⊑) b
        proof safe
          fix d assume dD: d ∈ D
          from dD Db Irange have bound (I d) (⊑) b by auto
          with s dD bA show s d ⊑ b by auto
        qed
        with x bA show x ⊑ b by auto
      qed
      assume f: well-related-set−continuous A (⊑) B (⊴) f
```

```
        and Dt: extreme-bound A (⊑) D t and D0: D ≠ {}
      from Dt have tA: t ∈ A by auto
      have fmono: monotone-on A (⊑) (⊴) f
        by (auto intro!:continuous-imp-monotone-on[OF f] pair-well-related)
      show extreme-bound B (⊴) (f ‘ D) (f t)
      proof (safe intro!: extreme-boundI)
        from f tA show f t ∈ B by auto
        fix d assume dD: d ∈ D
        from dD Dt have dt: d ⊑ t by auto
       from dD Dt DA show f d ⊴ f t by (auto intro!: monotone-onD[OF fmono])
      next
        fix b assume fDb: bound (f ‘ D) (⊴) b and bB: b ∈ B
        from Dx Dt have x ∼ t by (auto intro!: sympartpI elim!: extreme-boundE)
        with extreme-bound-sym-trans[OF sDA x this tA]
        have extreme-bound A (⊑) (s ‘ D) t by auto
        from f[THEN continuousD, OF wsD - sDA this] D0
        have ft: extreme-bound B (⊴) (f ‘ s ‘ D) (f t) by auto
        have bound (f ‘ s ‘ D) (⊴) b
        proof (safe)
          fix d assume dD: d ∈ D
          from Irange dD have IdD: I d ⊆ D by auto
          with DA have IdA: I d ⊆ A by auto
          from directed-setD[OF Idir[rule-format, OF dD], of {}]
          have Idne: I d ≠ {} by auto
          have fsd: extreme-bound B (⊴) (f ‘ I d) (f (s d))
            apply (rule IH2[OF - - IdA f Idne s[OF dD]])
            using Icard Idir dD by auto
          from IdD have f ‘ I d ⊆ f ‘ D by auto
          from bound-subset[OF this fDb] fsd bB
          show f (s d) ⊴ b by auto
        qed
        with ft bB show f t ⊴ b by auto
      qed
    qed
  qed
qed
```

The next Theorem corresponds to Proposition 5.9 of [4], without antisymmetry on *A*.

```
theorem (in quasi-ordered-set) well-complete-iff-directed-complete:
  (nonempty ⊓ well-related-set)−complete A (⊑) ⟷ directed-set−complete A (⊑)
(is ?l ⟷ ?r)
proof (intro iffI)
  show ?l ⟹ ?r
    by (auto intro!: completeI dest!: directed-completeness-lemma(1))
  assume r: ?r
  show ?l
    apply (rule complete-subclass[OF r])
    using well-related-set.directed-set
  by auto
qed
```

The next Theorem corresponds to Corollary 3 of [9] without any assumptions on the codomain B and without antisymmetry on the domain A.

```
theorem (in quasi-ordered-set)
  fixes leB (infix ⊴ 50)
  assumes comp: (nonempty ⊓ well-related-set)−complete A (⊑)
  shows well-related-set−continuous A (⊑) B (⊴) f ⟷ directed-set−continuous
A (⊑) B (⊴) f
    (is ?l ⟷ ?r)
proof (intro iffI)
  assume l: ?l
  show ?r
    using continuous-carrierD[OF l]
    using directed-completeness-lemma(2)[OF comp - - l]
    by (auto intro!: continuousI)
next
  assume r: ?r
  show ?l
    apply (rule continuous-subclass[OF - r])
    using well-related-set.directed-set by auto
qed

end
```

References

[1] S. Abramsky and A. Jung. *Domain Theory.* Number III in Handbook of Logic in Computer Science. Oxford University Press, 1994.

[2] G. Bancerek. The ordinal numbers. *Journal of Formalized Mathematics*, 1, 1989.

[3] J. C. Blanchette, A. Popescu, and D. Traytel. Cardinals in Isabelle/HOL. In G. Klein and R. Gamboa, editors, *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8558 of *Lecture Notes in Computer Science*, pages 111–127. Springer, 2014.

[4] P. M. Cohn. *Universal Algebra.* Harper & Row, 1965.

[5] J. Dubut and A. Yamada. Fixed point theorems for non-transitive relations. *Log. Methods Comput. Sci.*, 18(1), 2022.

[6] A. Finkel and J. Goubault-Larrecq. Forward Analysis for WSTS, Part I: Completions. In S. Albers and J.-Y. Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science*, volume 3 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 433–444, Dagstuhl, Germany, 2009. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[7] J. Goubault-Larrecq. *Non-Hausdorff Topology and Domain Theory: Selected Topics in Point-Set Topology*, volume 22 of *New Mathematical Monographs*. Cambridge University Press, 2013.

[8] T. Iwamura. A lemma on directed sets. In *Zenkoku Shijo Sugaku Danwakai*, number 262, pages 107–111, 1944. in Japanese.

[9] G. Markowsky. Chain-complete posets and directed sets with applications. *Algebra Universalis*, 6:53–68, 1976.

[10] L. C. Paulson and K. Grabczewski. Mechanizing set theory. *J. Autom. Reason.*, 17(3):291–323, 1996.

[11] D. Scott. Outline of a Mathematical Theory of Computation. Technical Report PRG02, OUCL, 1970.

[12] L. A. Skorni akov. *Complemented modular lattices and regular rings.* Oliver & Boyd, 1964.

[13] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction.* Foundations of Computing. The MIT Press, 1993.

[14] A. Yamada and J. Dubut. Formalizing Results on Directed Sets in Isabelle/HOL. In *Proceedings of the fourteenth conference on Interactive Theorem Proving (ITP'23)*, 2023.