松坂和夫『集合・位相入門』の Isabelle/HOL による形式化

Cryolite

2020 年 5 月 6 日

目次

theory *Split-Pair*
  imports *Main*
    HOL−Eisbach.Eisbach
begin

lemma *split-paired-Coll*:
  shows $\{x.\ P\ x\} = \{(a,\ b).\ P\ (a,\ b)\}$
  by *simp*

lemma *split-paired-Collect*:
  shows $\{x \in X.\ P\ x\} = \{(a,\ b) \in X.\ P\ (a,\ b)\}$
  by *auto*

lemma *split-paired-Ball-Times*:
  shows $(\forall x \in A \times B.\ P\ x) \longleftrightarrow (\forall a \in A.\ \forall b \in B.\ P\ (a,\ b))$
  by *simp*

lemma *split-paired-Bex-Times*:
  shows $(\exists x \in A \times B.\ P\ x) \longleftrightarrow (\exists a \in A.\ \exists b \in B.\ P\ (a,\ b))$
  by *simp*

method *split-pair* = (

(*unfold split-paired-all*)?;
(*unfold split-paired-All*)?;
(*unfold split-paired-Ex*)?;
(*unfold split-paired-The*)?;
(*subst split-paired-Coll*)?;
(*subst split-paired-Collect*)?;
(*unfold split-paired-Ball-Times*)?;
(*unfold split-paired-Bex-Times*)?;
(*unfold case-prod-conv*)?)

experiment
fixes $A :: ('a \times 'b)$ *set*
begin

lemma
shows $A = A$
proof (*rule set-eqI*; *split-pair*)
fix a b
show $(a, b) \in A \longleftrightarrow (a, b) \in A$ **..**
qed

lemma *subsetI-atomize*:
assumes $\forall x. x \in X \longrightarrow x \in Y$
shows $X \subseteq Y$
using *assms* **by** *auto*

lemma
assumes $A = \{\}$
and $B = \{\}$
shows $A \subseteq B$
proof (*rule subsetI-atomize*; *split-pair*)
{
fix a b
assume $(a, b) \in A$
with *assms*(*1*) **have** $(a, b) \in B$ **by** *simp*
}
thus $\forall a\ b. (a, b) \in A \longrightarrow (a, b) \in B$ **by** *simp*
qed

lemma *ex-mem-imp-not-empty*:
assumes $\exists x. x \in X$
shows $X \neq \{\}$
using *assms* **by** *auto*

lemma

assumes $(a,\ b) \in A$
shows $A \neq \{\}$
proof (*rule ex-mem-imp-not-empty*; *split-pair*)
from *assms* **show** $\exists\, a\ b.\ (a,\ b) \in A$ **by** *auto*
qed

lemma *Coll-False-imp-empty*:
assumes $X = \{x.\ \mathit{False}\}$
shows $X = \{\}$
using *assms* **by** *simp*

lemma
assumes $A = \{(a,\ b).\ a \neq a\}$
shows $A = \{\}$
proof (*rule Coll-False-imp-empty*; *split-pair*)
from *assms* **show** $A = \{(a,\ b).\ \mathit{False}\}$ **by** *simp*
qed

lemma *Collect-False-imp-empty*:
assumes $X = \{x \in Y.\ \mathit{False}\}$
shows $X = \{\}$
using *assms* **by** *simp*

lemma
assumes $A = \{(a,\ b) \in Y.\ a \neq a\}$
shows $A = \{\}$
proof (*rule Collect-False-imp-empty*; *split-pair*)
from *assms* **show** $A = \{(a,\ b) \in Y.\ \mathit{False}\}$ **by** *simp*
qed

lemma *subsetI-Ball*:
assumes $\forall\, x \in X.\ x \in Y$
shows $X \subseteq Y$
using *assms* **by** *auto*

lemma
assumes $\bigwedge a\ b.\ (a,\ b) \in X \times Y \Longrightarrow (a,\ b) \in Z \times W$
shows $(X \times Y) \subseteq (Z \times W)$
proof (*rule subsetI-Ball*; *split-pair*)
{
fix $a\ b$
assume $a \in X$
and $b \in Y$
with *assms* **have** $(a,\ b) \in Z \times W$ **by** *simp*
}

thus $\forall\, a \in X.\ \forall\ \ b \in Y.\ (a,\ b) \in Z \times W$ **by** *simp*
qed

lemma *Bex-imp-not-empty*:
assumes $\exists\, x \in X.\ P\ x$
shows $X \neq \{\}$
using *assms* **by** *auto*

lemma
assumes $x \in X \times Y$
shows $X \times Y \neq \{\}$
proof (*rule Bex-imp-not-empty*[**where** $P = \lambda x.\ True$]; *split-pair*)
from *assms* **have** *fst* $x \in X$ **and** *snd* $x \in Y$ **by** *auto*
thus $\exists\, x \in X.\ \exists\, y \in Y.\ True$ **by** *auto*
qed

end

end
theory *Section-1-1*
imports *Main*
begin

第 1 章

Sets and Maps

1.1 Notion of Sets

1.1.1 A) Sets and Elements

1.1.2 B) Notation of Sets

1.1.3 C) Equality of Sets

1.1.4 D) Subsets

proposition *prop-1-1-3*:
  shows $A = B \longleftrightarrow A \subseteq B \wedge A \supseteq B$
  by (*fact set-eq-subset*)

proposition *prop-1-1-4*:
  assumes $A \subseteq B$
    and $B \subseteq C$
  shows $A \subseteq C$
  using *assms* **by** *simp*

proposition *prop-1-1-5*:
  fixes A :: *'a set*
  shows $\{\} \subseteq A$
  by *simp*

proposition *prop-1-1-6*:
  assumes $x \in \{\}$
  shows $x \in A$
  using *assms* **by** (*fact emptyE*)

1.1.5 Problems

proposition *prob-1-1-1*:
  shows $a \in A \longleftrightarrow \{a\} \subseteq A$
  by *simp*

proposition *prob-1-1-2*:
  shows $\{1, 2, 3\} = \{n :: nat.\ 1 \leq n \wedge n \leq 3\}$
  by *auto*

end
theory *Section-1-2*
  imports *Main*
    Section-1-1
begin

1.2 Operations among Sets

1.2.1 A) Union

proposition *prop-1-2-0-a*:
  defines $A \equiv \{1, 2, 3, 4, 5\}$
    and $B \equiv \{3, 5, 7, 9\}$
  shows $A \cup B = \{1, 2, 3, 4, 5, 7, 9\}$
  unfolding *A-def* **and** *B-def* **by** *auto*

proposition *prop-1-2-0-b*:
  defines $A \equiv \{n :: nat.\ even\ n\}$
    and $B \equiv \{n :: nat.\ odd\ n\}$
  shows $A \cup B = UNIV$
  unfolding *A-def* **and** *B-def* **by** *auto*

proposition *prop-1-2-1*:
  shows $A \cup B = \{x.\ x \in A \vee x \in B\}$
  by (*fact Un-def*)

proposition *prop-1-2-2-a*:
  shows $A \subseteq A \cup B$
  by (*fact Un-upper1*)

proposition *prop-1-2-2-b*:
  shows $B \subseteq A \cup B$
  by (*fact Un-upper2*)

proposition *prop-1-2-3*:
  assumes $A \subseteq C$
    and $B \subseteq C$
  shows $A \cup B \subseteq C$
  using *assms* **by** (*fact Un-least*)

proposition *prop-1-2-4*:
  shows $A \cup A = A$
  by (*fact Un-absorb*)

proposition *prop-1-2-5*:
  shows $A \cup B = B \cup A$
  by *auto*

proposition *prop-1-2-6*:
  shows $(A \cup B) \cup C = A \cup (B \cup C)$
  by *auto*

proposition *prop-1-2-6-b*:
  shows $((A \cup B) \cup C) \cup D = (A \cup B) \cup (C \cup D)$
    and $(A \cup B) \cup (C \cup D) = A \cup (B \cup (C \cup D))$
    and $A \cup (B \cup (C \cup D)) = A \cup ((B \cup C) \cup D)$
    and $A \cup ((B \cup C) \cup D) = (A \cup (B \cup C)) \cup D$
  by *auto*

proposition *prop-1-2-7*:
  assumes $A \subseteq B$
  shows $A \cup B = B$
  using *assms* **by** (*fact Un-absorb1*)

proposition *prop-1-2-8*:
  assumes $A \subseteq B$

shows $A \cup C \subseteq B \cup C$
using *assms* **by** *auto*

proposition *prop-1-2-9*:
shows $\{\} \cup A = A$
using *prop-1-1-5* **by** (*rule prop-1-2-7*)

1.2.2 B) Intersection

proposition *prop-1-2-0-c*:
defines $A :: nat\ set \equiv \{1, 2, 3, 4, 5\}$
and $B \equiv \{3, 5, 7, 9\}$
shows $A \cap B = \{3, 5\}$
unfolding *A-def* **and** *B-def* **by** *simp*

proposition *prop-1-2-0-d*:
defines $A \equiv \{n :: nat.\ even\ n\}$
and $B \equiv \{n :: nat.\ odd\ n\}$
shows $A \cap B = \{\}$
unfolding *A-def* **and** *B-def* **by** *auto*

proposition *prop-1-2-2′-a*:
shows $A \cap B \subseteq A$
by (*fact Int-lower1*)

proposition *prop-1-2-2′-b*:
shows $A \cap B \subseteq B$
by (*fact Int-lower2*)

proposition *prop-1-2-3′*:
assumes $C \subseteq A$
and $C \subseteq B$
shows $C \subseteq A \cap B$
using *assms* **by** (*fact Int-greatest*)

proposition *prop-1-2-4′*:
shows $A \cap A = A$
by (*fact Int-absorb*)

proposition *prop-1-2-5′*:
shows $A \cap B = B \cap A$
by *auto*

proposition *prop-1-2-6′*:
shows $(A \cap B) \cap C = A \cap (B \cap C)$

by *auto*

proposition *prop-1-2-7'*:
assumes $A \subseteq B$
shows $A \cap B = A$
using *assms* **by**(*fact Int-absorb2*)

proposition *prop-1-2-8'*:
assumes $A \subseteq B$
shows $A \cap C \subseteq B \cap C$
using *assms* **by** *auto*

proposition *prop-1-2-9'*:
shows $\{\} \cap A = \{\}$
using *prop-1-1-5* **by** (*rule prop-1-2-7'*)

proposition *prop-1-2-10*:
shows $(A \cup B) \cap C = A \cap C \cup B \cap C$
by (*fact Int-Un-distrib2*)

proposition *prop-1-2-10'*:
shows $(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$
by (*fact Un-Int-distrib2*)

proposition *prop-1-2-11*:
shows $(A \cup B) \cap A = A$
by *auto*

proposition *prop-1-2-11'*:
shows $(A \cap B) \cup A = A$
by *auto*

1.2.3 C) Difference

proposition *prop-1-2-0-e*:
defines $A :: nat\ set \equiv \{1, 2, 3, 4, 5\}$
and $B \equiv \{3, 5, 7, 9\}$
shows $A - B = \{1, 2, 4\}$
unfolding *A-def* **and** *B-def* **by** *auto*

1.2.4 D) Universal Set

proposition *prop-1-2-12-a*:
assumes $A \subseteq X$
shows $A \cup (X - A) = X$

using *assms* **by** *auto*

proposition *prop-1-2-12-b*:
— The assumption $A \subseteq X$ is not necessary.
shows $A \cap (X - A) = \{\}$
by (*fact Diff-disjoint*)

proposition *prop-1-2-13*:
assumes $A \subseteq X$
shows $X - (X - A) = A$
using *assms* **by** (*simp only*: *double-diff*)

proposition *prop-1-2-14-a*:
shows $X - \{\} = X$
by (*fact Diff-empty*)

proposition *prop-1-2-14-b*:
shows $X - X = \{\}$
by (*fact Diff-cancel*)

proposition *prop-1-2-15*:
assumes $A \subseteq X$
and $B \subseteq X$
shows $A \subseteq B \longleftrightarrow X - A \supseteq X - B$
using *assms* **by** *auto*

proposition *prop-1-2-16*:
— The assumption $A \subseteq X$ is not necessary.
— The assumption $B \subseteq X$ is not necessary.
shows $X - (A \cup B) = (X - A) \cap (X - B)$
by (*fact Diff-Un*)

proposition *prop-1-2-16'*:
— The assumption $A \subseteq X$ is not necessary.
— The assumption $B \subseteq X$ is not necessary.
shows $X - (A \cap B) = (X - A) \cup (X - B)$
by (*fact Diff-Int*)

1.2.5 E) Family of Sets, Power Set

proposition *prop-1-2-0-f*:
fixes a **and** b **and** c
defines $X \equiv \{a, b, c\}$
shows $Pow\ X = \{\{\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{c, a\}, \{a, b, c\}\}$
unfolding *X-def* **by** *blast*

```
proposition prop-1-2-0-g:
  shows Pow {} = {{}}
  by (fact Pow-empty)

proposition prop-1-2-0-h:
  assumes finite X
    and card X = n
  shows card (Pow X) = 2 ^ n
using assms proof (induct n arbitrary: X)
  case 0
  from 0 and assms(1) have X = {} by simp
  thus ?case by simp
next
  case (Suc n')
  from Suc.prems obtain x and X' where card X' = n' and insert x X' = X and x ∉ X'
    by (blast dest: card-eq-SucD)
  from Suc.prems(1) and this(2) have finite X' by auto
  with ‹card X' = n'› have card (Pow X') = 2 ^ n' by (intro Suc.hyps)
  note ‹finite X'›
  moreover have Pow (insert x X') = Pow X' ∪ insert x ` Pow X' by (fact Pow-insert)
  moreover from ‹x ∉ X'› have Pow X' ∩ insert x ` Pow X' = {} by auto
  ultimately have card (Pow (insert x X')) = card (Pow X') + card (insert x ` Pow X')
    by (simp add: card-Un-disjoint)
  moreover have card (insert x ` Pow X') = card (Pow X')
  proof (rule card-image, rule inj-onI)
    fix A B
    assume A ∈ Pow X'
      and B ∈ Pow X'
      and insert x A = insert x B
    from this(1,2) and ‹x ∉ X'› have x ∉ A and x ∉ B by auto
    with ‹insert x A = insert x B› show A = B by auto
  qed
  moreover note ‹insert x X' = X›
  ultimately have card (Pow X) = card (Pow X') + card (Pow X') by simp
  also from ‹card (Pow X') = 2 ^ n'› have ... = 2 ^ (Suc n') by simp
  finally show ?case .
qed
```

1.2.6 F) Union and Intersection of Family of Sets

```
proposition prop-1-2-17:
  shows ∀ A ∈ 𝔄. A ⊆ ⋃𝔄
  by auto
```

proposition *prop-1-2-18*:
  assumes $\forall A \in \mathfrak{A}.\ A \subseteq C$
  shows $\bigcup \mathfrak{A} \subseteq C$
  using *assms* **by** *auto*

proposition *prop-1-2-17'*:
  shows $\forall A \in \mathfrak{A}.\ \bigcap \mathfrak{A} \subseteq A$
  by *auto*

proposition *prop-1-2-18'*:
  assumes $\forall A \in \mathfrak{A}.\ C \subseteq A$
  shows $C \subseteq \bigcap \mathfrak{A}$
  using *assms* **by** *auto*

1.2.7 Problems

proposition *prob-1-2-1-a*:
  — The assumption $A \subseteq X$ is not necessary.
  — The assumption $B \subseteq X$ is not necessary.
  shows $(A \cup B) \cap (A \cup (X - B)) = A$
  by *auto*

proposition *prob-1-2-1-b*:
  — The assumption $A \subseteq X$ is not necessary.
  — The assumption $B \subseteq X$ is not necessary.
  shows $(A \cup B) \cap ((X - A) \cup B) \cap (A \cup (X - B)) = A \cap B$
  by *auto*

proposition *prob-1-2-2-a*:
  assumes — The assumption $A \subseteq X$ is not necessary.
    $B \subseteq X$
  shows $A \cap B = \{\} \longleftrightarrow B \subseteq X - A$
  using *assms* **by** *auto*

proposition *prob-1-2-2-b*:
  assumes $A \subseteq X$
    — The assumption $B \subseteq X$ is not necessary.
  shows $A \cap B = \{\} \longleftrightarrow A \subseteq X - B$
  using *assms* **by** *auto*

proposition *prob-1-2-3-a-a*:
  shows $A - B = (A \cup B) - B$
  by *auto*

proposition *prob-1-2-3-a-b*:

shows $A - B = A - (A \cap B)$
by *auto*

proposition *prob-1-2-3-a-c*:
assumes $A \subseteq X$
— The assumption $B \subseteq X$ is not necessary.
shows $A - B = A \cap (X - B)$
using *assms* **by** *auto*

proposition *prob-1-2-3-b*:
shows $A - B = A \longleftrightarrow A \cap B = \{\}$
by *auto*

proposition *prob-1-2-3-c*:
shows $A - B = \{\} \longleftrightarrow A \subseteq B$
by (*fact Diff-eq-empty-iff*)

proposition *prob-1-2-4-a*:
shows $A - (B \cup C) = (A - B) \cap (A - C)$
by (*fact prop-1-2-16*)

proposition *prob-1-2-4-b*:
shows $A - (B \cap C) = (A - B) \cup (A - C)$
by (*fact prop-1-2-16'*)

proposition *prob-1-2-4-c*:
shows $(A \cup B) - C = (A - C) \cup (B - C)$
by (*fact Un-Diff*)

proposition *prob-1-2-4-d*:
shows $(A \cap B) - C = (A - C) \cap (B - C)$
by *auto*

proposition *prob-1-2-4-e*:
shows $A \cap (B - C) = (A \cap B) - (A \cap C)$
by (*fact Diff-Int-distrib*)

proposition *prob-1-2-5-a*:
shows $(A - B) - C = A - (B \cup C)$
by *auto*

proposition *prob-1-2-5-b*:
shows $A - (B - C) = (A - B) \cup (A \cap C)$
by *auto*

proposition *prob-1-2-6*:
  assumes $A \subseteq C$
  shows $A \cup (B \cap C) = (A \cup B) \cap C$
  using *assms* **by** *auto*

definition *sym-diff* :: $'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set$ (**infixl** $\triangle$ *65*)
  where *sym-diff* $A\ B = (A - B) \cup (B - A)$

lemmas *sym-diff-eq* [*iff*] = *sym-diff-def*

proposition *prob-1-2-7-a*:
  shows $A \triangle B = B \triangle A$
  by *auto*

proposition *prob-1-2-7-b*:
  shows $A \triangle B = (A \cup B) - (A \cap B)$
  by *auto*

proposition *prob-1-2-7-c*:
  shows $(A \triangle B) \triangle C = A \triangle (B \triangle C)$
  by *auto*

proposition *prob-1-2-7-d*:
  shows $A \cap (B \triangle C) = (A \cap B) \triangle (A \cap C)$
  by *auto*

proposition *prob-1-2-8-a* [*simp*]:
  shows $A \triangle \{\} = A$
  by *simp*

proposition *prob-1-2-8-b*:
  assumes $A \subseteq X$
  shows $A \triangle X = X - A$
  using *assms* **by** *auto*

proposition *prob-1-2-8-c* [*simp*]:
  shows $A \triangle A = \{\}$
  by *simp*

proposition *prob-1-2-8-d*:
  assumes $A \subseteq X$
  shows $A \triangle (X - A) = X$
  using *assms* **by** *auto*

proposition *prob-1-2-9*:

assumes $A_1 \triangle A_2 = B_1 \triangle B_2$
shows $A_1 \triangle B_1 = A_2 \triangle B_2$
using *assms* **by** *auto*

end
theory *Section-1-3*
imports *Main*
Split-Pair
Section-1-2
begin

1.3 Correspondences, Functions

1.3.1 A) Direct Product of Two Sets

proposition *example-1-3-1*:
fixes p **and** q **and** r
defines $A \equiv \{1, 2\}$
and $B \equiv \{p, q, r\}$
shows $A \times B = \{(1, p), (1, q), (1, r), (2, p), (2, q), (2, r)\}$
and $A \times A = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$
unfolding *A-def* **and** *B-def* **by** *auto*

proposition *example-1-3-2*:
assumes *finite* A
and *finite* B
and *card* $A = m$
and *card* $B = n$
shows *card* $(A \times B) = m * n$
using *assms* **by** *simp*

1.3.2 B) Notion of Correspondence

In this section, a correspondence is defined by a function of the type $'a \Rightarrow 'b$ *set*. Neither the initial nor target set of a correspondence is not explicitly specified. Instead, for a correspondence Γ, its initial set is implicitly specified by a set $A::'a$ *set* such that $\forall a.\ a \notin A \longrightarrow \Gamma\ a = \{\}$, and its target set is implicitly specified by a set B such that Γ \`\` $A \subseteq B$.

1.3.3 C) Graph of Correspondence

definition *corr-graph* :: $('a \Rightarrow 'b\ set) \Rightarrow ('a \times 'b)$ *set*
where *corr-graph* $\Gamma = \{(a, b).\ b \in \Gamma\ a\}$

lemma *corr-graphI* [*intro*]:
  assumes $b \in \Gamma\ a$
  shows $(a,\ b) \in$ *corr-graph* Γ
  using *assms* **unfolding** *corr-graph-def* **by** *simp*

lemma *corr-graphD* [*dest*]:
  assumes $(a,\ b) \in$ *corr-graph* Γ
  shows $b \in \Gamma\ a$
  using *assms* **unfolding** *corr-graph-def* **by** *simp*

proposition *prop-1-3-1*:
  shows $\Gamma\ a = \{b.\ (a,\ b) \in$ *corr-graph* $\Gamma\}$
  by *auto*

theorem *thm-1-1*:
  shows $\exists!\Gamma.$ *corr-graph* $\Gamma = G$
proof −
  define Γ **where** $\Gamma\ a \equiv \{b.\ (a,\ b) \in G\}$ **for** a
  show *?thesis*
  proof (*rule ex1I*)
    from Γ-*def* **show** *corr-graph* $\Gamma = G$ **by** *auto*
  next
    fix Γ'
    assume *corr-graph* $\Gamma' = G$
    with Γ-*def* **show** $\Gamma' = \Gamma$ **by** *auto*
  qed
qed

lemma *corr-graph-inject*:
  assumes *corr-graph* $\Gamma =$ *corr-graph* Γ'
  shows $\Gamma = \Gamma'$
  using *assms thm-1-1* **by** *auto*

definition *corr-dom* :: $('a \Rightarrow 'b\ set) \Rightarrow 'a\ set$
  where *corr-dom* $\Gamma = \{a.\ \exists\, b.\ (a,\ b) \in$ *corr-graph* $\Gamma\}$

lemma *corr-domI* [*intro*]:
  assumes $b \in \Gamma\ a$
  shows $a \in$ *corr-dom* Γ
  using *assms* **unfolding** *corr-dom-def* **by** *auto*

lemma *corr-domE* [*elim*]:
  assumes $a \in$ *corr-dom* Γ
  obtains b **where** $b \in \Gamma\ a$
  using *assms* **unfolding** *corr-dom-def* **by** *auto*

definition *corr-range* :: $('a \Rightarrow 'b\ set) \Rightarrow 'b\ set$
where *corr-range* $\Gamma = \{b.\ \exists\, a.\ (a,\ b) \in$ *corr-graph* $\Gamma\}$

lemma *corr-rangeI* [*intro*]:
assumes $b \in \Gamma\ a$
shows $b \in$ *corr-range* Γ
using *assms* **unfolding** *corr-range-def* **by** *auto*

lemma *corr-rangeE* [*elim*]:
assumes $b \in$ *corr-range* Γ
obtains a **where** $b \in \Gamma\ a$
using *assms* **unfolding** *corr-range-def* **by** *auto*

1.3.4 D) Inverse of Correspondence

definition *corr-inv* :: $('a \Rightarrow 'b\ set) \Rightarrow 'b \Rightarrow 'a\ set$
where *corr-inv* $\Gamma\ b = \{a.\ b \in \Gamma\ a\}$

lemma *corr-invI* [*intro*]:
assumes $b \in \Gamma\ a$
shows $a \in$ *corr-inv* $\Gamma\ b$
using *assms* **unfolding** *corr-inv-def* **by** *simp*

lemma *corr-invD* [*dest*]:
assumes $a \in$ *corr-inv* $\Gamma\ b$
shows $b \in \Gamma\ a$
using *assms* **unfolding** *corr-inv-def* **by** *simp*

proposition *prop-1-3-2*:
shows $b \in \Gamma\ a \longleftrightarrow a \in$ *corr-inv* $\Gamma\ b$
by *auto*

proposition *prop-1-3-a*:
shows *corr-graph* (*corr-inv* Γ) $= \{(b,\ a).\ (a,\ b) \in$ *corr-graph* $\Gamma\}$
by *auto*

proposition *prop-1-3-3-a*:
shows *corr-dom* (*corr-inv* Γ) $=$ *corr-range* Γ
by *auto*

proposition *prop-1-3-3-b*:
shows *corr-range* (*corr-inv* Γ) $=$ *corr-dom* Γ
by *auto*

proposition *prop-1-3-4*:
  shows *corr-inv* (*corr-inv* Γ) = Γ
  by *auto*

proposition *prop-1-3-b*:
  shows *corr-inv* Γ $b \neq \{\} \longleftrightarrow b \in$ *corr-range* Γ
  by *auto*

1.3.5 E) Maps

definition *as-corr-on* :: $('a \Rightarrow 'b) \Rightarrow 'a\ set \Rightarrow 'a \Rightarrow 'b\ set$
  where *as-corr-on f A a* = (*if* $a \in A$ *then* $\{f\ a\}$ *else* $\{\}$)

as-corr-on transforms a function into the correspondence. Since a function f in Isabelle/HOL is total, which means that the function is always defined on the universal set *UNIV* of the type $'a$, *as-corr-on* additionally specifies a set of the type $'a\ set$ that should be considered as the domain of the function.

lemma *as-corr-onI* [*intro*]:
  assumes $a \in A$
    and $f\ a = b$
  shows $b \in$ *as-corr-on f A a*
  using *assms* **unfolding** *as-corr-on-def* **by** *simp*

lemma *as-corr-onE* [*elim*]:
  assumes $b \in$ *as-corr-on f A a*
  obtains $a \in A$ **and** $f\ a = b$
proof −
  from *assms* **have** $*$: $b \in$ (*if* $a \in A$ *then* $\{f\ a\}$ *else* $\{\}$) **by** (*simp only*: *as-corr-on-def*)
  {
    assume $a \notin A$
    with $*$ **have** *False* **by** *simp*
  }
  hence $a \in A$ **by** *auto*
  moreover from *this* **and** $*$ **have** $f\ a = b$ **by** *simp*
  ultimately show *thesis* **by** (*fact that*)
qed

definition *corr-functional-on* :: $('a \Rightarrow 'b\ set) \Rightarrow 'a\ set \Rightarrow bool$
  where *corr-functional-on* Γ $A \longleftrightarrow (\forall a.\ (a \in A \longrightarrow (\exists!b.\ b \in \Gamma\ a)) \wedge (a \notin A \longrightarrow \Gamma\ a = \{\}))$

definition *id-on* :: $('a \Rightarrow 'a) \Rightarrow 'a\ set \Rightarrow bool$
  where *id-on f A* $\longleftrightarrow (\forall a \in A.\ f\ a = a)$

id-on f A states that the function f behaves as an identity function on the set A. This proposition does not specify any other property of f on an element out of A.

lemmas *id-on-iff* = *id-on-def*

lemma *id-onI* [*intro*]:
  assumes $\bigwedge a.\ a \in A \Longrightarrow f\ a = a$
  shows *id-on* f A
  using *assms* **unfolding** *id-on-def* **by** *simp*

lemma *id-onD* [*dest*]:
  assumes *id-on* f A
    and $a \in A$
  shows $f\ a = a$
  using *assms* **unfolding** *id-on-def* **by** *fast*

lemma *id-on-empty* [*simp*]:
  shows *id-on* f $\{\}$
  by *auto*

lemma *set-eqI2*:
  assumes $\bigwedge x.\ x \in A \Longrightarrow x \in B$
    and $\bigwedge x.\ x \in B \Longrightarrow x \in A$
  shows $A = B$
  using *assms* **by** *blast*

lemma *id-on-imp-surj-on*:
  assumes *id-on* f A
  shows f ‘ $A = A$
  using *assms* **by** *force*

lemma *id-on-imp-inj-on*:
  assumes *id-on* f A
  shows *inj-on* f A
proof (*rule inj-onI*)
  fix a **and** a'
  assume $a \in A$ **and** $a' \in A$ **and** $f\ a = f\ a'$
  with *assms* **show** $a = a'$ **using** *id-onD* **by** *fastforce*
qed

lemma *id-on-imp-bij-betw*:
  assumes *id-on* f A
  shows *bij-betw* f A A
  using *assms* **by** (*auto intro*: *bij-betw-imageI dest*: *id-on-imp-surj-on id-on-imp-inj-on*)

lemma *thm-1-2-a*:
  assumes — The assumption $G \subseteq A \times B$ is not necessary.

```
    f ` A ⊆ B
    and corr-graph (as-corr-on f A) = G
  shows ∀ a ∈ A. ∃!b ∈ B. (a, b) ∈ G
  using assms by fast

lemma thm-1-2-b:
  assumes G ⊆ A × B
    and ∀ a ∈ A. ∃!b ∈ B. (a, b) ∈ G
  obtains f where
    f ` A ⊆ B
    and corr-graph (as-corr-on f A) = G
proof −
  define f where f a ≡ THE b. b ∈ B ∧ (a, b) ∈ G for a
  have f0: f a ∈ B and f1: (a, f a) ∈ G if a ∈ A for a
  proof −
    from assms(2) and that have ∃!b. b ∈ B ∧ (a, b) ∈ G by simp
    hence f a ∈ B ∧ (a, f a) ∈ G (is ?L ∧ ?R) unfolding f-def by (fact theI')
    thus ?L and ?R by simp+
  qed
  from f0 have f ` A ⊆ B by auto
  moreover have corr-graph (as-corr-on f A) = G
  proof (rule set-eqI2, split-pair; split-pair)
    fix a b
    assume (a, b) ∈ corr-graph (as-corr-on f A)
    hence b ∈ as-corr-on f A a by auto
    hence a ∈ A and f a = b by auto
    thus (a, b) ∈ G by (auto intro: f1)
  next
    fix a b
    assume (a, b) ∈ G
    with assms(1) have a ∈ A and b ∈ B by auto
    from this(1) and assms(2) have ∃!b ∈ B. (a, b) ∈ G by simp
    with ‹b ∈ B› and ‹(a, b) ∈ G› have f a = b unfolding f-def by auto
    with ‹a ∈ A› have b ∈ as-corr-on f A a by auto
    thus (a, b) ∈ corr-graph (as-corr-on f A) by auto
  qed
  ultimately show thesis by (fact that)
qed

theorem thm-1-2:
  assumes G ⊆ A × B
  shows (∃f. f ` A ⊆ B ∧ corr-graph (as-corr-on f A) = G) ⟷ (∀ a ∈ A. ∃!b ∈ B. (a, b) ∈ G)
    (is ?L ⟷ ?R)
proof (intro iffI)
  assume ?L
```

then obtain f **where** f ‘ $A \subseteq B$ **and** *corr-graph* (*as-corr-on* f A) $= G$ **by** *blast*
thus *?R* **by** (*elim thm-1-2-a*)
next
assume *?R*
with *assms* **obtain** f **where** f ‘ $A \subseteq B$ **and** *corr-graph* (*as-corr-on* f A) $= G$ **by** (*elim thm-1-2-b*)
thus *?L* **by** *auto*
qed

1.3.6 Problems

proposition *prob-1-3-3-a*:
assumes — The assumption *corr-graph* $\Gamma \subseteq A \times B$ is not necessary.
— Since *corr-dom* $\Gamma = A^*$ is an assumption, the assumption *corr-graph* $\Gamma \subseteq A \times B$ can be replaced by *corr-range* $\Gamma \subseteq B$.
corr-range $\Gamma \subseteq B$
and *corr-dom* $\Gamma = A$
and $\bigwedge b\ b'.\ b \in B \Longrightarrow b' \in B \Longrightarrow b \neq b' \Longrightarrow$ *corr-inv* Γ b $\cap$ *corr-inv* Γ $b' = \{\}$
obtains f **where** f ‘ $A \subseteq B$ **and** *as-corr-on* f $A = \Gamma$
proof −
{
fix a
assume $a \in A$
with *assms*(*2*) **obtain** b **where** $*$: $b \in \Gamma\ a$ **by** *auto*
with *assms*(*1*) **have** $b \in B$ **by** *auto*
moreover from $*$ **have** $(a, b) \in$ *corr-graph* Γ **by** *auto*
moreover {
fix b'
assume $b' \in B$ **and** $(a, b') \in$ *corr-graph* Γ
{
assume $b' \neq b$
moreover note ‹$b \in B$› **and** ‹$b' \in B$›
moreover have *corr-inv* Γ b $\cap$ *corr-inv* Γ $b' \neq \{\}$
proof −
from ‹$b \in \Gamma\ a$› **have** $a \in$ *corr-inv* Γ b **by** *auto*
moreover from ‹$(a, b') \in$ *corr-graph* Γ› **have** $a \in$ *corr-inv* Γ b' **by** *auto*
ultimately show *?thesis* **by** *auto*
qed
ultimately have *False* **using** *assms*(*3*) **by** *simp*
}
hence $b' = b$ **by** *auto*
}
ultimately have $\exists! b \in B.\ (a, b) \in$ *corr-graph* Γ **by** *blast*
}
hence $\forall a \in A.\ \exists! b \in B.\ (a, b) \in$ *corr-graph* Γ **..**
moreover from *assms*(*1*,*2*) **have** *corr-graph* $\Gamma \subseteq A \times B$ **by** *auto*

ultimately obtain f
where f ‘ $A \subseteq B$
and *corr-graph* (*as-corr-on* f A) = *corr-graph* Γ **by** (*elim thm-1-2-b*)
hence *as-corr-on* f $A = \Gamma$ **by** (*elim corr-graph-inject*)
with ‹f ‘ $A \subseteq B$› **show** *thesis* **by** (*rule that*)
qed

proposition *prob-1-3-3-b*:
assumes — The assumption *corr-graph* $\Gamma \subseteq A \times B$ is not necessary.
— Original assumptions would include *corr-graph* $\Gamma \subseteq A \times B$ but it can be weakened to the assumption *corr-dom* $\Gamma \subseteq A$.
corr-dom $\Gamma \subseteq A$
and f ‘ $A \subseteq B$
and *as-corr-on* f $A = \Gamma$
obtains *corr-dom* $\Gamma = A$
and $\bigwedge b\ b'.\ b \in B \Longrightarrow b' \in B \Longrightarrow b \neq b' \Longrightarrow$ *corr-inv* Γ b $\cap$ *corr-inv* Γ $b' = \{\}$
proof −
from *assms*(*3*) **have** *corr-graph* (*as-corr-on* f A) = *corr-graph* Γ **by** *simp*
with *assms*(*2*) **have** $*$: $\forall a \in A.\ \exists!b \in B.\ (a, b) \in$ *corr-graph* Γ **by** (*rule thm-1-2-a*)
note *assms*(*1*)
moreover have $A \subseteq$ *corr-dom* Γ **using** $*$ **by** *blast*
ultimately have *corr-dom* $\Gamma = A$ **..**
moreover {
fix b **and** b'
assume $b \in B$ **and** $b' \in B$ **and** $b \neq b'$
{
fix a
assume $a \in$ *corr-inv* Γ b **and** $a \in$ *corr-inv* Γ b'
hence $(a, b) \in$ *corr-graph* Γ **and** $(a, b') \in$ *corr-graph* Γ **by** *auto*
moreover from ‹$(a, b) \in$ *corr-graph* Γ› **and** *assms*(*1*) **have** $a \in A$ **by** *auto*
moreover note ‹$b \in B$› **and** ‹$b' \in B$›
ultimately have $b = b'$ **using** $*$ **by** *auto*
with ‹$b \neq b'$› **have** *False* **..**
}
hence *corr-inv* Γ b $\cap$ *corr-inv* Γ $b' = \{\}$ **by** *auto*
}
ultimately show *thesis* **by** (*fact that*)
qed

proposition *prob-1-3-3*:
assumes *corr-dom* $\Gamma \subseteq A$
and *corr-range* $\Gamma \subseteq B$
shows *corr-dom* $\Gamma = A \wedge (\forall b \in B.\ \forall b' \in B.\ b \neq b' \longrightarrow$ *corr-inv* Γ b $\cap$ *corr-inv* Γ $b' = \{\}) \longleftrightarrow$
$(\exists f.\ f$ ‘ $A \subseteq B \wedge$ *as-corr-on* f $A = \Gamma)$ (**is** *?L* $\longleftrightarrow$ *?R*)
proof (*intro iffI*)

```
  assume ?L
  hence corr-dom Γ = A
    and ⋀b b'. b ∈ B ⟹ b' ∈ B ⟹ b ≠ b' ⟹ corr-inv Γ b ∩ corr-inv Γ b' = {} by simp+
  with assms(2) obtain f where f ` A ⊆ B and as-corr-on f A = Γ by (rule prob-1-3-3-a)
  thus ?R by auto
next
  assume ?R
  then obtain f where f ` A ⊆ B and as-corr-on f A = Γ by auto
  with assms(1) show ?L by blast
qed

proposition prob-1-3-4-a:
  assumes id-on f A
  shows corr-graph (as-corr-on f A) = {(a, a) | a. a ∈ A} (is ?L = ?R)
proof -
  have ?L = {(a, b). a ∈ A ∧ b = f a} by auto
  also from assms have ... = ?R by auto
  finally show ?thesis .
qed

proposition prob-1-3-4-b:
  assumes ∀ a ∈ A. f a = b0
  shows corr-graph (as-corr-on f A) = {(a, b0) | a. a ∈ A} (is ?L = ?R)
proof -
  have ?L = {(a, b). a ∈ A ∧ b = f a} by auto
  also from assms have ... = ?R by fastforce
  finally show ?thesis .
qed

end
theory Section-1-4
  imports Main
    HOL−Library.Indicator-Function
    Section-1-3
begin
```

1.4 Various Concepts on Functions

In this and the following sections, functions in usual mathematics are formalized in terms of functions in Isabelle/HOL. Because careful handling is required in such formalization, let me explain the detail.

Functions in usual mathematics are associated with its domain and codomain. The careful handling is required in their formalization. A function in Isabelle/HOL (say, $f :: 'a \Rightarrow 'b$) is associated with the domain (the universal set of the type $'a$) and the codomain (the universal set of the type $'b$). However, because (the universal set of) types in Isabelle/HOL are not expressive enough to encode an arbitrary set (which is equivalent to an arbitrary proposition under the axiom schema of comprehension in Isabelle/HOL), a subset of the universal set of the type $'a$ is needed to be explicitly specified in order to formalize the domain of a function in usual mathematics.

As a result, a function f in Isabelle/HOL that formalizes a function in usual mathematics has "two domains"; one is the original one, i.e., the universal set of the type $'a$, and the other is the one that is explicitly specified in order to express the domain of the function in usual mathematics.

Let A be a set that is specified to express the domain of a function in usual mathematics. It should be always noted that the function in Isabelle/HOL is defined on $-$ A. This fact is essentially different from the function in usual mathematics, where it is simply undefined out of A. This difference demands careful handling. The statement of definitions and propositions concerning functions in usual mathematics should be properly rephrased to reflect the difference.

How each concept concerning functions in usual mathematics is rephrased in the following sections is named below;

- The codomain is explicitly specified as a set B such that f ` $A \subseteq B$ only when it is actually required in definitions and propositions.
- The inverse image of a set $Q \subseteq B$ under f is defined f $-$` $Q \cap A$ since $f$ $-$` Q could include an element out of A.
- Surjectivity of f is stated as the proposition f ` $A = B$.
- Injectivity of f is stated as the proposition *inj-on* f A.
- Equality of two function f and g on a set A is defined by the proposition $\forall\, a{\in}A.\ f\ a = g\ a$, which is shortly stated as the proposition *ext-eq-on* A f g. Note that f and g behaves differently out of A and thus $f = g$ does not hold even if *ext-eq-on* A f g holds.
- (TODO: inverse)

Extensional equality of two functions on a set.

definition *ext-eq-on* :: $'a\ set \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$
  where *ext-eq-on* $A\ f\ f' \longleftrightarrow (\forall\, a \in A.\ f\ a = f'\ a)$

lemma *ext-eq-onI* [*intro*]:
  assumes $\bigwedge a.\ a \in A \Longrightarrow f\ a = f'\ a$
  shows *ext-eq-on* $A\ f\ f'$
  using *assms* **unfolding** *ext-eq-on-def* **by** *simp*

lemma *ext-eq-onD* [*dest*]:
assumes *ext-eq-on* $A\ f\ f'$
and $a \in A$
shows $f\ a = f'\ a$
using *assms* **unfolding** *ext-eq-on-def* **by** *simp*

lemma *ext-eq-on-refl* [*simp*]:
shows *ext-eq-on* $A\ f\ f$
by *auto*

lemma *ext-eq-on-sym* [*sym*]:
assumes *ext-eq-on* $A\ f\ f'$
shows *ext-eq-on* $A\ f'\ f$
using *assms* **by** *fastforce*

lemma *ext-eq-on-trans* [*trans*]:
assumes *ext-eq-on* $A\ f\ g$
and *ext-eq-on* $A\ g\ h$
shows *ext-eq-on* $A\ f\ h$
using *assms* **by** *fastforce*

lemma *ext-eq-on-empty* [*simp*]:
shows *ext-eq-on* $\{\}\ f\ g$
by *auto*

1.4.1 A) Image and Inverse Image of Map

proposition *prop-1-4-1*:
assumes — The assumption $f\ `\ A \subseteq B$ is not necessary.
— The assumption $P_1 \subseteq A$ is not necessary.
— The assumption $P_2 \subseteq A$ is not necessary.
$P_1 \subseteq P_2$
shows $f\ `\ P_1 \subseteq f\ `\ P_2$
using *assms* **by** (*fact image-mono*)

proposition *prop-1-4-2*:
— The assumption $f\ `\ A \subseteq B$ is not necessary.
— The assumption $P_1 \subseteq A$ is not necessary.
— The assumption $P_2 \subseteq A$ is not necessary.
shows $f\ `\ (P_1 \cup P_2) = f\ `\ P_1 \cup f\ `\ P_2$
by (*fact image-Un*)

proposition *prop-1-4-3*:
— The assumption $f\ `\ A \subseteq B$ is not necessary.

— The assumption $P_1 \subseteq A$ is not necessary.
— The assumption $P_2 \subseteq A$ is not necessary.
shows $f \text{ ` } (P_1 \cap P_2) \subseteq f \text{ ` } P_1 \cap f \text{ ` } P_2$
by (*fact image-Int-subset*)

proposition *prop-1-4-4*:
— The assumption $f \text{ ` } A \subseteq B$ is not necessary.
— The assumption $P \subseteq A$ is not necessary.
shows $f \text{ ` } (A - P) \supseteq f \text{ ` } A - f \text{ ` } P$
by (*fact image-diff-subset*)

proposition *prop-1-4-1'*:
assumes — @The assumption prop "f ‘ A subseteq¿ B" is not necessary.
— The assumption $Q_1 \subseteq B$ is not necessary.
— The assumption $Q_2 \subseteq B$ is not necessary.
$Q_1 \subseteq Q_2$
shows $(f -\text{` } Q_1) \cap A \subseteq (f -\text{` } Q_2) \cap A$
using *assms* **by** *auto*

proposition *prop-1-4-2'*:
— The assumption $f \text{ ` } A \subseteq B$ is not necessary.
— The assumption $Q_1 \subseteq B$ is not necessary.
— The assumption $Q_2 \subseteq B$ is not necessary.
shows $f -\text{` } (Q_1 \cup Q_2) \cap A = (f -\text{` } Q_1 \cap A) \cup (f -\text{` } Q_2 \cap A)$
by *auto*

proposition *prop-1-4-3'*:
— The assumption $f \text{ ` } A \subseteq B$ is not necessary.
— The assumption $Q_1 \subseteq B$ is not necessary.
— The assumption $Q_2 \subseteq B$ is not necessary.
shows $f -\text{` } (Q_1 \cap Q_2) \cap A = (f -\text{` } Q_1 \cap A) \cap (f -\text{` } Q_2 \cap A)$
by *auto*

proposition *prop-1-4-4'*:
assumes $f \text{ ` } A \subseteq B$
— The assumption $Q \subseteq B$ is not necessary.
shows $f -\text{` } (B - Q) \cap A = A - (f -\text{` } Q \cap A)$
using *assms* **by** *auto*

proposition *prop-1-4-5*:
assumes
— The assumption $f \text{ ` } A \subseteq B$ is not necessary.
$P \subseteq A$
shows $P \subseteq f -\text{` } (f \text{ ` } P) \cap A$
using *assms* **by** *auto*

proposition *prop-1-4-5'*:
— The assumption f ‘ $A \subseteq B$ is not necessary.
— The assumption $Q \subseteq B$ is not necessary.
shows f ‘ $(f -‘ Q \cap A) \subseteq Q$
by *auto*

proposition *prob-1-4-a*:
assumes $P \subseteq A$
and $A - P = \{a\}$
and $P = \{p\}$
and $f(a) = f(p)$
shows f ‘ $A - f$ ‘ $P \subset f$ ‘ $(A - P)$
using *assms* **by** *blast*

proposition *prob-1-4-b*:
assumes $P \subseteq A$
and $\{a\} = A - P$
and $\{p\} = P$
and $f(a) = f(p)$
shows $P \subset f -‘ (f$ ‘ $(P))$
using *assms* **by** *blast*

1.4.2 B) Surjective, Injective, and Bijective Maps

lemma *surj-onI*:
assumes $\bigwedge a.\ a \in A \Longrightarrow f\ a \in B$
and $\bigwedge b.\ b \in B \Longrightarrow b \in f$ ‘ A
shows f ‘ $A = B$
proof (*intro equalityI*)
from *assms*(*1*) **show** f ‘ $A \subseteq B$ **by** *auto*
from *assms*(*2*) **show** $B \subseteq f$ ‘ A **by** *auto*
qed

lemma *bij-betw-imageI'*:
assumes $\bigwedge a\ a'.\ a \in A \Longrightarrow a' \in A \Longrightarrow f\ a = f\ a' \Longrightarrow a = a'$
and $\bigwedge a.\ a \in A \Longrightarrow f\ a \in B$
and $\bigwedge b.\ b \in B \Longrightarrow b \in f$ ‘ A
shows *bij-betw* $f\ A\ B$
proof (*rule bij-betw-imageI*)
from *assms*(*1*) **show** *inj-on* $f\ A$ **by** (*rule inj-onI*)
from *assms*(*2*,*3*) **show** f ‘ $A = B$ **by** *auto*
qed

lemma *corr-inv-fun-fun*:

```
  assumes a ∈ A
  shows corr-inv (as-corr-on f A) (f a) = {a' ∈ A. f a' = f a}
  using assms by auto

lemma corr-inv-fun-eq:
  shows corr-inv (as-corr-on f A) b = {a ∈ A. f a = b} (is ?L = ?R)
proof (rule set-eqI)
  fix a
  have a ∈ ?L ⟷ b ∈ as-corr-on f A a by auto
  also have ... ⟷ a ∈ A ∧ f a = b by auto
  also have ... ⟷ a ∈ ?R by simp
  finally show a ∈ ?L ⟷ a ∈ ?R .
qed

theorem thm-1-4-a:
  assumes f ` A ⊆ B
    and as-corr-on g B = corr-inv (as-corr-on f A)
  shows bij-betw f A B
proof (rule bij-betw-imageI')
  fix a and a'
  assume a ∈ A
    and a' ∈ A
    and f a = f a'
  from ‹a ∈ A› and assms(1) have f a ∈ B by auto
  moreover from assms(2) have corr-inv (as-corr-on f A) (f a) = as-corr-on g B (f a) by simp
  with ‹a ∈ A› and ‹f a ∈ B› have **: {a'' ∈ A. f a'' = f a} = {g (f a)}
    by (auto simp only: corr-inv-fun-fun)
  with ‹a ∈ A› have a = g (f a) by blast
  moreover from ** and ‹a' ∈ A› and ‹f a = f a'› have a' = g (f a) by auto
  ultimately show a = a' by simp
next
  fix a
  assume a ∈ A
  with assms(1) show f a ∈ B by auto
next
  fix b
  assume b ∈ B
  moreover from assms(2) have corr-inv (as-corr-on f A) b = as-corr-on g B b by simp
  ultimately have {a ∈ A. f a = b} = {g b} by (force simp only: corr-inv-fun-eq)
  hence g b ∈ A and f (g b) = b by auto
  thus b ∈ f ` A by force
qed

lemma bij-betwE2 [elim]:
  assumes bij-betw f A B
```

```
  obtains f ‘ A = B
    and inj-on f A
  using assms by (auto dest: bij-betw-imp-surj-on bij-betw-imp-inj-on)

theorem thm-1-4-b:
  assumes f ‘ A ⊆ B
    and bij-betw f A B
  obtains g where as-corr-on g B = corr-inv (as-corr-on f A)
proof −
  {
    fix b
    {
      assume b ∈ B
      from assms(2) have f ‘ A = B and inj-on f A by auto
      from this(1) and ‹b ∈ B› obtain a where a ∈ A and b = f a by auto
      {
        fix a′
        assume ‹a′ ∈ A›
        with ‹a ∈ A› and ‹inj-on f A› have a′ = a ⟷ f a′ = f a by (simp only: inj-on-eq-iff)
        with ‹a′ ∈ A› have a′ ∈ {a′. a′ = a} ⟷ a′ ∈ {a′. f a′ = f a} by simp
      }
      hence {a′ ∈ A. a′ = a} = {a′ ∈ A. f a′ = f a} by auto
      with ‹a ∈ A› have {a′. a′ = a} = {a′ ∈ A. f a′ = f a} by blast
      with ‹inj-on f A› and ‹a ∈ A›
      have {a′. a′ = the-inv-into A f (f a)} = {a′ ∈ A. f a′ = f a}
        by (simp only: the-inv-into-f-f)
      with ‹b = f a› have {a. a = (the-inv-into A f) b} = {a ∈ A. f a = b} by simp
      with ‹b ∈ B› have as-corr-on (the-inv-into A f) B b = corr-inv (as-corr-on f A) b
        by (force simp only: corr-inv-fun-eq)
    }
    moreover {
      assume b ∉ B
      with assms(1) have {a ∈ A. f a = b} = {} by blast
      with ‹b ∉ B› have as-corr-on (the-inv-into A f) B b = corr-inv (as-corr-on f A) b by blast
    }
    ultimately have as-corr-on (the-inv-into A f) B b = corr-inv (as-corr-on f A) b by auto
  }
  hence as-corr-on (the-inv-into A f) B = corr-inv (as-corr-on f A) by auto
  thus thesis by (fact that)
qed

theorem thm-1-4:
  assumes f ‘ A ⊆ B
  shows (∃ g. as-corr-on g B = corr-inv (as-corr-on f A)) ⟷ bij-betw f A B
proof (rule iffI)
```

assume $\exists\, g.$ *as-corr-on g B* = *corr-inv* (*as-corr-on f A*)
then obtain *g* **where** *as-corr-on g B* = *corr-inv* (*as-corr-on f A*) **by** *auto*
with *assms* **show** *bij-betw f A B* **by** (*auto intro*: *thm-1-4-a*)
next
assume *bij-betw f A B*
with *assms* **obtain** *g* **where** *as-corr-on g B* = *corr-inv* (*as-corr-on f A*) **by** (*elim thm-1-4-b*)
thus $\exists\, g.$ *as-corr-on g B* = *corr-inv* (*as-corr-on f A*) **by** *auto*
qed

definition *is-inv-into* :: *'a set* $\Rightarrow$ (*'a* $\Rightarrow$ *'b*) $\Rightarrow$ (*'b* $\Rightarrow$ *'a*) $\Rightarrow$ *bool*
where *is-inv-into A f g* $\longleftrightarrow$ *corr-inv* (*as-corr-on g* (*f* ‘ *A*)) = *as-corr-on f A*

lemma *is-inv-intoD1*:
assumes *is-inv-into A f g*
and $a \in A$
shows $g\ (f\ a) = a$
proof −
from *assms*(*1*) **have** *corr-inv* (*as-corr-on g* (*f* ‘ *A*)) = *as-corr-on f A* **by** (*unfold is-inv-into-def*)
hence *corr-inv* (*as-corr-on g* (*f* ‘ *A*)) *a* = *as-corr-on f A a* **by** *simp*
with *assms*(*2*) **have** $\{b \in f\ `\ A.\ g\ b = a\} = \{f\ a\}$ **by** (*auto simp only*: *corr-inv-fun-eq*)
thus $g\ (f\ a) = a$ **by** *auto*
qed

lemma *is-inv-intoD2*:
assumes *is-inv-into A f g*
and $b \in f\ `\ A$
shows $f\ (g\ b) = b$
using *assms* **by** (*auto dest*: *is-inv-intoD1*)

lemma *is-inv-into-imp-bij-betw1*:
assumes *is-inv-into A f g*
shows *bij-betw f A* (*f* ‘ *A*)
proof −
from *assms* **have** *corr-inv* (*as-corr-on g* (*f* ‘ *A*)) = *as-corr-on f A* **by** (*unfold is-inv-into-def*)
hence *as-corr-on g* (*f* ‘ *A*) = *corr-inv* (*as-corr-on f A*) **using** *prop-1-3-4* **by** *metis*
moreover have $f\ `\ A \subseteq f\ `\ A$ **by** *simp*
ultimately show *?thesis* **by** (*intro thm-1-4-a*)
qed

lemma *is-inv-into-imp-bij-betw2*:
assumes *is-inv-into A f g*
shows *bij-betw g* (*f* ‘ *A*) *A*
proof (*rule bij-betw-imageI*)
{
fix *a*

assume $a \in A$
with *assms* **have** $g\ (f\ a) = a$ **by** (*rule is-inv-intoD1*)
with $\langle a \in A \rangle$ **have**$g\ (f\ a) \in A$ **by** *simp*
}
moreover {
fix a
assume $a \in A$
with *assms* **have** $g\ (f\ a) = a$ **by** (*rule is-inv-intoD1*)
with $\langle a \in A \rangle$ **have** $\exists\, a' \in A.\ g\ (f\ a') = a$ **by** *auto*
}
ultimately show g ‘ f ‘ $A = A$ **by** (*blast intro*: *surj-onI*)
{
fix b **and** b'
assume $b \in f$ ‘ A
and $b' \in f$ ‘ A
and $g\ b = g\ b'$
then obtain a **and** a' **where** $a \in A$ **and** $b = f\ a$ **and** $a' \in A$ **and** $b' = f\ a'$ **by** *blast*
from $\langle g\ b = g\ b' \rangle$ **and** *this*(*2*,*4*) **have** $g\ (f\ a) = g\ (f\ a')$ **by** *simp*
with $\langle a \in A \rangle$ **and** $\langle a' \in A \rangle$ **and** *assms* **have** $a = a'$ **using** *is-inv-intoD1* **by** *fastforce*
with $\langle b = f\ a \rangle$ **and** $\langle b' = f\ a' \rangle$ **have** $b = b'$ **by** *simp*
}
thus *inj-on* g (f ‘ A) **by** (*blast intro*: *inj-onI*)
qed

1.4.3 C) Composition of Maps

theorem *thm-1-5-a*:
assumes f ‘ $A = B$
and g ‘ $B = C$
shows $(g \circ f)$ ‘ $A = C$
using *assms* **by** *auto*

theorem *thm-1-5-b*:
assumes f ‘ $A \subseteq B$
— The assumption g ‘ $B \subseteq C$ is not necessary.
and *inj-on* f A
and *inj-on* g B
shows *inj-on* $(g \circ f)$ A
proof (*rule inj-onI*)
fix $a\ a'$
assume $a \in A$ **and** $a' \in A$ **and** $(g \circ f)\ a = (g \circ f)\ a'$
from *this*(*3*) **have** $g\ (f\ a) = g\ (f\ a')$ **by** *simp*
moreover from $\langle a \in A \rangle$ **and** $\langle a' \in A \rangle$ **and** *assms*(*1*) **have** $f\ a \in B$ **and** $f\ a' \in B$ **by** *auto*
moreover note *assms*(*3*)
ultimately have $f\ a = f\ a'$ **by** (*elim inj-onD*)

with *assms*(*2*) **and** ⟨$a \in A$⟩ **and** ⟨$a' \in A$⟩ **show** $a = a'$**by** (*elim inj-onD*)
qed

theorem *thm-1-5-c*:
assumes *bij-betw f A B*
and *bij-betw g B C*
shows *bij-betw* $(g \circ f)$ *A C*
using *assms* **by** (*auto intro*: *bij-betw-trans*)

theorem *thm-1-6-1*:
shows $(h \circ g) \circ f = h \circ (g \circ f)$
by (*fact comp-assoc*)

theorem *thm-1-6-2-a*:
assumes — The assumption f ‘ $A \subseteq B$ is not necessary.
id-on g A
shows *ext-eq-on A* $(f \circ g)$ *f*
using *assms* **by** *fastforce*

theorem *thm-1-6-2-b*:
assumes f ‘ $A \subseteq B$
and *id-on g B*
shows *ext-eq-on A* $(g \circ f)$ *f*
using *assms* **by** *fastforce*

theorem *thm-1-6-3-a*:
assumes *bij-betw f A B*
and *is-inv-into A f g*
shows *id-on* $(g \circ f)$ *A*
using *assms* **by** (*fastforce dest*: *is-inv-intoD1*)

theorem *thm-1-6-3-b*:
assumes *bij-betw f A B*
and *is-inv-into A f g*
shows *id-on* $(f \circ g)$ *B*
using *assms* **by** (*fastforce dest*: *is-inv-intoD2*)

1.4.4 D) Restriction and Extension of Maps

1.4.5 E) Note on Codomain of Map

1.4.6 F) Sets of Maps

1.4.7 Problems

proposition *prob-1-4-2-b*:
  assumes — The assumption f ‘ $A \subseteq B$ is not necessary.
    $P \subseteq A$
    and $a \in P$
    and $a' \in A - P$
    and $f\ a = f\ a'$
  shows $P \subset (f\ -`\ f\ `\ P) \cap A$
proof (*rule psubsetI*)
  from *assms*(*1*) **show** $P \subseteq (f\ -`\ f\ `\ P) \cap A$ **by** (*fact prop-1-4-5*)
  from *assms*(*2*) **have** $f\ a \in f\ `\ P$ **by** *simp*
  with *assms*(*4*) **have** $f\ a' \in f\ `\ P$ **by** *simp*
  hence $a' \in f\ -`\ f\ `\ P$ **by** *simp*
  with ‹$a' \in A - P$› **have** $a' \in (f\ -`\ f\ `\ P) \cap A$ **by** *simp*
  moreover from *assms*(*3*) **have** $a' \notin P$ **by** *simp*
  ultimately show $P \neq (f\ -`\ f\ `\ P) \cap A$ **by** *auto*
qed

proposition *prob-1-4-3-a*:
  assumes — The assumption f ‘ $A \subseteq B$ is not necessary.
    inj-on $f\ A$
    and $P \subseteq A$
  shows $(f\ -`\ (f\ `\ P)) \cap A = P$
proof (*rule set-eqI2*)
  fix p
  assume $p \in (f\ -`\ (f\ `\ P)) \cap A$
  hence $f\ p \in f\ `\ P$ **and** $p \in A$ **by** *simp+*
  from *this*(*1*) **obtain** p' **where** $p' \in P$ **and** $f\ p = f\ p'$ **by** *auto*
  from *this*(*1*) **and** *assms*(*2*) **have** $p' \in A$ **by** *auto*
  with ‹$p \in A$› **and** ‹$f\ p = f\ p'$› **and** *assms*(*1*) **have** $p = p'$ **by** (*elim inj-onD*)
  with ‹$p' \in P$› **show** $p \in P$ **by** *simp*
next
  fix p
  assume $p \in P$
  hence $f\ p \in f\ `\ P$ **by** *simp*
  hence $p \in f\ -`\ (f\ `\ P)$ **by** *auto*
  moreover from ‹$p \in P$› **and** *assms*(*2*) **have** $p \in A$ **..**

ultimately show $p \in (f -` (f ` P)) \cap A$ **..**
qed

proposition *prob-1-4-3-b*:
assumes $Q \subseteq B$
and $f ` A = B$
shows $f ` ((f -` Q) \cap A) = Q$
using *assms* **by** *auto*

proposition *prob-1-4-4*:
assumes — The assumption $f ` A \subseteq B$ is not necessary.
inj-on f A
and $P_1 \subseteq A$
and $P_2 \subseteq A$
shows $f ` (P_1 \cap P_2) = f ` P_1 \cap f ` P_2$
using *assms* **by** (*auto dest*: *inj-on-image-Int*)

proposition *prob-1-4-5-b*:
assumes — The assumption $f ` A \subseteq B$ is not necessary.
— The assumption $P \subseteq A$ is not necessary.
$a \in P$
and $a' \in A - P$
and $f\ a = f\ a'$
shows $f ` A - f ` P \subset f ` (A - P)$
proof (*rule psubsetI*)
show $f ` A - f ` P \subseteq f ` (A - P)$ **by** (*fact prop-1-4-4*)
from *assms*(*2*) **have** $a' \in A$ **by** *simp*
hence $f\ a' \in f ` A$ **by** *simp*
from *assms*(*1*) **have** $f\ a \in f ` P$ **by** *simp*
with *assms*(*3*) **have** $f\ a' \in f ` P$ **by** *simp*
with ‹$f\ a' \in f ` A$› **have** $f\ a' \notin f ` A - f ` P$ **by** *simp*
moreover from *assms*(*2*) **have** $f\ a' \in f ` (A - P)$ **by** *simp*
ultimately show $f ` A - f ` P \neq f ` (A - P)$ **by** *blast*
qed

proposition *prob-1-4-5-c*:
assumes — The assumption $f ` A \subseteq B$ is not necessary.
$P \subseteq A$
and *inj-on f A*
shows $f ` (A - P) = f ` A - f ` P$
proof (*rule set-eqI2*)
fix b
assume $b \in f ` (A - P)$
then obtain a **where** $a \in A$ **and** $a \notin P$ **and** $b = f\ a$ **by** *auto*
{

assume $b \in f$ ‘ P
then obtain a' **where** $a' \in P$ **and** $b = f\ a'$ **by** *auto*
from *this*(*1*) **and** *assms*(*1*) **have** $a' \in A$ **by** *auto*
with ‹$a \in A$› **and** ‹$b = f\ a$› **and** ‹$b = f\ a'$› **and** *assms*(*2*) **have** $a = a'$ **by** (*auto dest*: *inj-onD*)
with ‹$a' \in P$› **and** ‹$a \notin P$› **have** *False* **by** *simp*
}
hence $b \notin f$ ‘ P **by** *auto*
with ‹$a \in A$› **and** ‹$b = f\ a$› **show** $b \in f$ ‘ $A - f$ ‘ P **by** *simp*
next
fix b
assume $b \in f$ ‘ $A - f$ ‘ P
thus $b \in f$ ‘ $(A - P)$ **by** *blast*
qed

proposition *prob-1-4-8*:
assumes — The assumption *bij-betw* $f\ A\ B$ can be weakened to f ‘ $A = B$ because *is-inv-into* $A\ f\ f'$ implies *bij-betw* $f\ A\ (f$ ‘ $A)$.
f ‘ $A = B$
— The assumption *bij-betw* $g\ B\ C$ can be weakened to g ‘ $B = C$ because *is-inv-into* $A\ f\ f'$ implies *bij-betw* $g\ B\ (g$ ‘ $B)$.
and g ‘ $B = C$
and *is-inv-into* $A\ f\ f'$
and *is-inv-into* $B\ g\ g'$
and *is-inv-into* $A\ (g \circ f)\ h$
shows *ext-eq-on* $C\ h\ (f' \circ g')$
proof (*rule ext-eq-onI*)
fix c
assume $c \in C$
from *assms*(*5*) **have** *bij-betw* $h\ ((g \circ f)$ ‘ $A)\ A$ **by** (*intro is-inv-into-imp-bij-betw2*)
moreover from *assms*(*1*,*2*) **have** $(g \circ f)$ ‘ $A = C$ **by** *auto*
ultimately have *bij-betw* $h\ C\ A$ **by** *simp*
with ‹$c \in C$› **have** $h\ c \in A$ **by** (*auto dest*: *bij-betw-imp-surj-on*)
from ‹$c \in C$› **and** ‹$(g \circ f)$ ‘ $A = C$› **have** $c \in (g \circ f)$ ‘ A **by** *simp*
with *assms*(*5*) **have** $g\ (f\ (h\ c)) = c$ **by** (*auto dest*: *is-inv-intoD2*)
hence $g'\ (g\ (f\ (h\ c))) = g'\ c$ **by** *simp*
moreover from ‹$h\ c \in A$› **and** *assms*(*1*) **have** $f\ (h\ c) \in B$ **by** *auto*
moreover note *assms*(*4*)
ultimately have $f\ (h\ c) = g'\ c$ **by** (*auto dest*: *is-inv-intoD1*)
hence $f'\ (f\ (h\ c)) = f'\ (g'\ c)$ **by** *simp*
moreover note ‹$h\ c \in A$›
moreover note *assms*(*3*)
ultimately have $h\ c = f'\ (g'\ c)$ **by** (*auto dest*: *is-inv-intoD1*)
thus $h\ c = (f' \circ g')\ c$ **by** *simp*
qed

proposition *prob-1-4-9-a*:
— The assumption f ‘ $A \subseteq B$ is not necessary.
— The assumption g ‘ $B \subseteq C$ is not necessary.
— The assumption $P \subseteq A$ is not necessary.
shows $(g \circ f)$ ‘ $P = g$ ‘ $(f$ ‘ $P)$ (**is** $?L = ?R$)
by *auto*

proposition *prob-1-4-9-b*:
— The assumption f ‘ $A \subseteq B$ is not necessary.
— The assumption g ‘ $B \subseteq C$ is not necessary.
— The assumption $R \subseteq C$ is not necessary.
shows $((g \circ f) -‘ R) \cap A = (f -‘ (g -‘ R)) \cap A$ (**is** $?L = ?R$)
by *fastforce*

proposition *prob-1-4-10-a*:
assumes f ‘ $A \subseteq B$
and g ‘ $B \subseteq C$
and $(g \circ f)$ ‘ $A = C$
shows g ‘ $B = C$
using *assms* **by** *fastforce*

proposition *prob-1-4-10-b*:
assumes — The assumption f ‘ $A \subseteq B$ is not necessary.
— The assumption g ‘ $B \subseteq C$ is not necessary.
inj-on $(g \circ f)$ A
shows *inj-on* f A
using *assms* **by** (*fact inj-on-imageI2*)

proposition *prob-1-4-11*:
assumes f ‘ $A = B$
— The assumption g ‘ $B \subseteq C$ is not necessary.
— The assumption g' ‘ $B \subseteq C$ is not necessary.
and *ext-eq-on* A $(g \circ f)$ $(g' \circ f)$
shows *ext-eq-on* B g g'
using *assms* **by** *fastforce*

proposition *prob-1-4-12*:
assumes f ‘ $A \subseteq B$
and f' ‘ $A \subseteq B$
— The assumption g ‘ $B \subseteq C$ is not necessary.
and *inj-on* g B
and *ext-eq-on* A $(g \circ f)$ $(g \circ f')$
shows *ext-eq-on* A f f'
proof (*rule ext-eq-onI*)
fix a

assume $a \in A$
with *assms*(*4*) **have** $g\ (f\ a) = g\ (f'\ a)$ **by** *auto*
moreover from ‹$a \in A$› **and** *assms*(*1*,*2*) **have** $f\ a \in B$ **and** $f'\ a \in B$ **by** *auto*
moreover note *assms*(*3*)
ultimately show $f\ a = f'\ a$ **by** (*elim inj-onD*)
qed

proposition *prob-1-4-13-a*:
assumes f ‘ $A \subseteq B$
and g ‘ $B \subseteq C$
and $(g \circ f)$ ‘ $A = C$
and *inj-on* $g\ B$
shows f ‘ $A = B$
proof (*rule surj-onI*)
fix a
assume $a \in A$
with *assms*(*1*) **show** $f\ a \in B$ **by** *auto*
next
fix b
assume $b \in B$
with *assms*(*2*) **have** $g\ b \in C$ **by** *auto*
with *assms*(*3*) **obtain** a **where** $a \in A$ **and** $g\ (f\ a) = g\ b$ **by** *force*
moreover from *this*(*1*) **and** *assms*(*1*) **have** $f\ a \in B$ **by** *auto*
moreover note ‹$b \in B$› **and** *assms*(*4*)
ultimately have $f\ a = b$ **by** (*elim inj-onD*)
with ‹$a \in A$› **show** $b \in f$ ‘ A **by** *auto*
qed

proposition *prob-1-4-13-b*:
assumes — The assumption g ‘ $B \subseteq C$ is not necessary.
inj-on $(g \circ f)\ A$
and f ‘ $A = B$
shows *inj-on* $g\ B$
proof (*rule inj-onI*)
fix $b\ b'$
assume $b \in B$ **and** $b' \in B$ **and** $g\ b = g\ b'$
from *this*(*1*,*2*) **and** *assms*(*2*) **obtain** a **and** a'
where $a \in A$ **and** $b = f\ a$ **and** $a' \in A$ **and** $b' = f\ a'$ **by** *auto*
from ‹$g\ b = g\ b'$› **and** *this*(*2*,*4*) **have** $g\ (f\ a) = g\ (f\ a')$ **by** *simp*
with ‹$a \in A$› **and** ‹$a' \in A$› **and** *assms*(*1*) **have** $a = a'$ **by** (*auto elim*: *inj-onD*)
with ‹$b = f\ a$› **and** ‹$b' = f\ a'$› **show** $b = b'$ **by** *simp*
qed

proposition *prob-1-4-14*:
assumes f ‘ $A \subseteq B$

and g ‘ $B \subseteq A$
and g' ‘ $B \subseteq A$
and *id-on* $(g \circ f)$ A
and *id-on* $(f \circ g')$ B
and *is-inv-into* A f f'
obtains *bij-betw* f A B
and *ext-eq-on* B g g'
and *ext-eq-on* B g' f'
proof $-$
have *bij-betw* f A B
proof (*rule bij-betw-imageI*)
{
fix a **and** a'
assume $a \in A$ **and** $a' \in A$ **and** $f\ a = f\ a'$
from *this*(*3*) **have** $g\ (f\ a) = g\ (f\ a')$ **by** *simp*
with $\langle a \in A \rangle$ **and** $\langle a' \in A \rangle$ **and** *assms*(*4*) **have** $a = a'$ **using** *id-onD* **by** *fastforce*
}
thus *inj-on* f A **by** (*intro inj-onI*)
{
fix b
assume $b \in B$
with *assms*(*5*) **have** $f\ (g'\ b) = b$ **by** *auto*
moreover from *assms*(*3*) **and** $\langle b \in B \rangle$ **have** $g'\ b \in A$ **by** *auto*
ultimately have $b \in f$ ‘ A **by** *force*
}
hence $B \subseteq f$ ‘ A **..**
with *assms*(*1*) **show** f ‘ $A = B$ **by** *simp*
qed
moreover have *ext-eq-on* B g g'
proof (*rule ext-eq-onI*)
fix b
assume $b \in B$
with *assms*(*5*) **have** $f\ (g'\ b) = b$ **by** *auto*
hence $g\ (f\ (g'\ b)) = g\ b$ **by** *simp*
moreover from $\langle b \in B \rangle$ **and** *assms*(*3*) **have** $g'\ b \in A$ **by** *auto*
moreover note *assms*(*4*)
ultimately show $g\ b = g'\ b$ **by** *auto*
qed
moreover have *ext-eq-on* B g' f'
proof (*rule ext-eq-onI*)
fix b
assume $b \in B$
with *assms*(*5*) **have** $f\ (g'\ b) = b$ **by** *auto*
hence $f'\ (f\ (g'\ b)) = f'\ b$ **by** *simp*
moreover from $\langle b \in B \rangle$ **and** *assms*(*3*) **have** $g'\ b \in A$ **by** *auto*

moreover note *assms*(*6*)
ultimately show $g'\ b = f'\ b$ **by** (*auto dest*: *is-inv-intoD1*)
qed
ultimately show *thesis* **by** (*fact that*)
qed

abbreviation χ :: $'a\ set \Rightarrow\ 'a \Rightarrow int$
where $\chi \equiv$ *indicator*

lemma *indicator-definition*:
obtains $x \in X \Longrightarrow \chi\ X\ x = 1$
| $x \notin X \Longrightarrow \chi\ X\ x = 0$
by *simp*

lemma *chi-0-1*:
shows $\chi\ A\ a \in \{0,\ 1\}$
proof −
{
assume $a \in A$
hence $\chi\ A\ a = 1$ **by** *simp*
}
moreover {
assume $a \notin A$
hence $\chi\ A\ a = 0$ **by** *simp*
}
ultimately show $\chi\ A\ a \in \{0,\ 1\}$ **by** *blast*
qed

lemma *one-leq-chiD*:
assumes $1 \leq \chi\ A\ a$
shows $a \in A$
proof −
from *assms* **and** *chi-0-1* **have** $\chi\ A\ a = 1$ **by** *force*
thus *?thesis* **by** (*simp only*: *indicator-eq-1-iff*)
qed

proposition *prob-1-4-15-0-a*:
assumes $A \subseteq X$
and $B \subseteq X$
and $\bigwedge x.\ x \in X \Longrightarrow \chi\ A\ x \leq \chi\ B\ x$
shows $A \subseteq B$
proof (*rule subsetI*)
fix *a*
assume $a \in A$
hence $1 = \chi\ A\ a$ **by** *simp*

```
    also have ... ≤ χ B a
    proof -
      from ‹a ∈ A› and assms(1) have a ∈ X by auto
      with assms(3) show ?thesis by simp
    qed
    finally have 1 ≤ χ B a by simp
    thus a ∈ B by (auto dest: one-leq-chiD)
  qed

  proposition prob-1-4-15-0-b:
    assumes A ⊆ X
      and B ⊆ X
      and A ⊆ B
      and x ∈ X
    shows χ A x ≤ χ B x
  proof -
    from assms consider (A) x ∈ A | (B) x ∈ B − A | (X) x ∈ X − B by auto
    thus ?thesis
    proof cases
      case A
      with assms(3) have x ∈ A and x ∈ B by auto
      thus ?thesis by simp
    next
      case B
      hence x ∉ A and x ∈ B by auto
      thus ?thesis by simp
    next
      case X
      with assms(3) have x ∉ A and x ∉ B by auto
      thus ?thesis by simp
    qed
  qed

  proposition prob-1-4-15:
    assumes A ⊆ X
      and B ⊆ X
    shows (∀ x ∈ X. χ A x ≤ χ B x) ⟷ A ⊆ B
  proof (rule iffI)
    assume ∀ x ∈ X. χ A x ≤ χ B x
    with assms show A ⊆ B using prob-1-4-15-0-a by metis
  next
    assume A ⊆ B
    with assms show ∀ x ∈ X. χ A x ≤ χ B x using prob-1-4-15-0-b by metis
  qed
```

proposition *prob-1-4-15-a*:
  shows $\chi\ (A \cap B)\ a = \chi\ A\ a * \chi\ B\ a$
  by (*fact indicator-inter-arith*)

proposition *prob-1-4-15-b*:
  shows $\chi\ (A \cup B)\ a = \chi\ A\ a + \chi\ B\ a - \chi\ (A \cap B)\ a$
proof –
  have $\chi\ (A \cup B)\ a = \chi\ A\ a + \chi\ B\ a - \chi\ A\ a * \chi\ B\ a$ **by** (*fact indicator-union-arith*)
  thus *?thesis* **by** (*simp only*: *prob-1-4-15-a*)
qed

proposition *prob-1-4-15-c*:
  assumes — The assumption $A \subseteq X$ is not necessary.
    $x \in X$
  shows $\chi\ (X - A)\ x = \mathit{1} - \chi\ A\ x$
proof –
  have $\chi\ (X - A)\ x = \chi\ X\ x * (\mathit{1} - \chi\ A\ x)$ **by** (*fact indicator-diff*)
  also from *assms* **have** $\ldots = \mathit{1} - \chi\ A\ x$ **by** *simp*
  finally show *?thesis* **.**
qed

proposition *prob-1-4-15-d*:
  shows $\chi\ (A - B)\ x = \chi\ A\ x * (\mathit{1} - (\chi\ B\ x))$
  by (*fact indicator-diff*)

proposition *prob-1-4-15-e*:
  shows $\chi\ (A \triangle B)\ x = |\chi\ A\ x - \chi\ B\ x|$
proof –
  consider (a) $x \in A$ **and** $x \in B$
    | (b) $x \in A$ **and** $x \notin B$
    | (c) $x \notin A$ **and** $x \in B$
    | (d) $x \notin A$ **and** $x \notin B$
    by *auto*
  thus *?thesis*
  proof *cases*
    case a
    hence $x \notin A \triangle B$ **by** *simp*
    hence $\chi\ (A \triangle B)\ x = \mathit{0}$ **by** *simp*
    moreover from a **have** $|\chi\ A\ x - \chi\ B\ x| = \mathit{0}$ **by** *simp*
    ultimately show *?thesis* **by** *simp*
  next
    case b
    hence $x \in A \triangle B$ **by** *simp*
    hence $\chi\ (A \triangle B)\ x = \mathit{1}$ **by** *simp*
    moreover from b **have** $|\chi\ A\ x - \chi\ B\ x| = \mathit{1}$ **by** *simp*

ultimately show *?thesis* **by** *simp*
next
case c
hence $x \in A \triangle B$ **by** *simp*
hence $\chi\ (A \triangle B)\ x = \mathit{1}$ **by** *simp*
moreover from c **have** $|\chi\ A\ x - \chi\ B\ x| = \mathit{1}$ **by** *simp*
ultimately show *?thesis* **by** *simp*
next
case d
hence $x \notin A \triangle B$ **by** *simp*
hence $\chi\ (A \triangle B)\ x = \mathit{0}$ **by** *simp*
moreover from d **have** $|\chi\ A\ x - \chi\ B\ x| = \mathit{0}$ **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
qed

end
theory *Section-1-5*
imports *Main*
HOL−Library.Disjoint-Sets
HOL−Library.FuncSet
Section-1-4
begin

1.5 Indexed Families, General Products

1.5.1 A) Infinite and Finite Sequence of Elements

1.5.2 B) Family of Elements

1.5.3 C) Families of Sets and Their Union and Intersection

proposition *prop-1-5-1*:
shows $(\bigcup l \in \Lambda.\ A\ l) \cap B = (\bigcup l \in \Lambda.\ (A\ l \cap B))$
by *simp*

proposition *prop-1-5-1'*:
shows $(\bigcap l \in \Lambda.\ A\ l) \cup B = (\bigcap l \in \Lambda.\ A\ l \cup B)$
by *simp*

proposition *prop-1-5-2*:
assumes $\Lambda \neq \{\}$ — This assumption is not specified in the book. However, there exits a counterexample without it.
— The assumption $\bigwedge l.\ l \in \Lambda \Longrightarrow A\ l \subseteq X$ is not necessary.
shows $X - (\bigcup l \in \Lambda.\ A\ l) = (\bigcap l \in \Lambda.\ (X - A\ l))$
using *assms* **by** *simp*

proposition *prop-1-5-2'*:
— The assumption $\bigwedge l.\ l \in \Lambda \Longrightarrow A\ l \subseteq X$ is not necessary.
shows $X - (\bigcap l \in \Lambda.\ A\ l) = (\bigcup l \in \Lambda.\ X - A\ l)$
by *simp*

proposition *prop-1-5-3*:
— The assumption f ‘ $A \subseteq B$ is not necessary.
— The assumption $\bigwedge l.\ l \in \Lambda \Longrightarrow P\ l \subseteq A$ is not necessary.
shows f ‘ $(\bigcup l \in \Lambda.\ P\ l) = (\bigcup l \in \Lambda.\ f$ ‘ $(P\ l))$
by (*fact image-UN*)

proposition *prop-1-5-4*:
— The assumption f ‘ $A \subseteq B$ is not necessary.
— The assumption $\bigwedge l.\ l \in \Lambda \Longrightarrow P\ l \subseteq A$ is not necessary.
shows f ‘ $(\bigcap l \in \Lambda.\ P\ l) \subseteq (\bigcap l \in \Lambda.\ f$ ‘ $(P\ l))$
by *auto*

proposition *prop-1-5-3'*:
— The assumption f ‘ $A \subseteq B$ is not necessary.
— The assumption $\bigwedge \mu.\ \mu \in M \Longrightarrow Q\ \mu \subseteq B$ is not necessary.
shows $(f$ −‘ $(\bigcup \mu \in M.\ Q\ \mu)) \cap A = (\bigcup \mu \in M.\ (f$ −‘ $(Q\ \mu)) \cap A)$
by *auto*

proposition *prop-1-5-4'*:
assumes $M \neq \{\}$ — This assumption is not specified in the book. However, there exists a counterexample without it.
— The assumption f ‘ $A \subseteq B$ is not necessary.
— The assumption $\bigwedge \mu.\ \mu \in M \Longrightarrow Q\ \mu \subseteq B$ is not necessary.
shows $(f$ −‘ $(\bigcap \mu \in M.\ Q\ \mu)) \cap A = (\bigcap \mu \in M.\ (f$ −‘ $(Q\ \mu)) \cap A)$
using *assms* **by** *auto*

1.5.4 Generalized Direct Product, Axiom of Choice

definition *dprod* :: $'a\ set \Rightarrow ('a \Rightarrow 'b\ set) \Rightarrow ('a \Rightarrow 'b)\ set$ **where**
dprod $\Lambda\ A \equiv \{a \in \Lambda \rightarrow_E \bigcup(A$ ‘ $\Lambda).\ \forall l \in \Lambda.\ a\ l \in A\ l\}$

syntax *-dprod* :: $pttrn \Rightarrow 'a\ set \Rightarrow ('a \Rightarrow 'b\ set) \Rightarrow ('a \Rightarrow 'b)\ set$
((3$\prod_d$ -∈-./ -) 10)

translations $\prod_d\ l \in \Lambda.\ A \rightleftharpoons CONST\ dprod\ \Lambda\ (\lambda l.\ A)$

lemma *dprodI* [*intro*]:
  assumes $\bigwedge l.\ l \in \Lambda \Longrightarrow a\ l \in A\ l$
    and $\bigwedge l.\ l \notin \Lambda \Longrightarrow a\ l = undefined$
  shows $a \in (\prod_d\ l \in \Lambda.\ A\ l)$
  using *assms* **unfolding** *dprod-def* **by** *auto*

lemma *dprodD1* [*dest*]:
  assumes $a \in (\prod_d\ l \in \Lambda.\ A\ l)$
    and $l \in \Lambda$
  shows $a\ l \in A\ l$
  using *assms* **unfolding** *dprod-def* **by** *blast*

lemma *dprodD2* [*dest*]:
  assumes $a \in (\prod_d\ l \in \Lambda.\ A\ l)$
    and $l \notin \Lambda$
  shows $a\ l = undefined$
  using *assms* **unfolding** *dprod-def* **by** *blast*

lemma *dprodE* [*elim*]:
  assumes $a \in (\prod_d\ l \in \Lambda.\ A\ l)$
  obtains $\bigwedge l.\ l \in \Lambda \Longrightarrow a\ l \in A\ l$
    and $\bigwedge l.\ l \notin \Lambda \Longrightarrow a\ l = undefined$
  using *assms* **unfolding** *dprod-def* **by** *blast*

theorem *AC*:
  assumes $\forall\, l \in \Lambda.\ A\ l \neq \{\}$
  shows $(\Pi\ l \in \Lambda.\ A\ l) \neq \{\}$
proof −
  let *?a* $= \lambda l.\ if\ l \in \Lambda\ then\ (SOME\ al.\ al \in A\ l)\ else\ undefined$
  {
    fix l
    assume $l \in \Lambda$
    with *assms* **have** $A\ l \neq \{\}$ **by** *auto*
    with $\langle l \in \Lambda \rangle$ **have** *?a* $l \in A\ l$ **by** (*auto intro*: *some-in-eq*[*THEN iffD2*])
  }
  hence *?a* $\in (\Pi\ l \in \Lambda.\ A\ l)$ **by** *simp*
  thus *?thesis* **by** *auto*
qed

lemma *AC-E*:
  assumes $\bigwedge l.\ l \in \Lambda \Longrightarrow A\ l \neq \{\}$
  obtains a **where** $a \in (\Pi\ l \in \Lambda.\ A\ l)$

proof –
  from *assms* **have** $(\Pi\ l \in \Lambda.\ A\ l) \neq \{\}$ **by** (*simp add*: *AC*)
  then obtain a **where** $a \in (\Pi\ l \in \Lambda.\ A\ l)$ **by** *blast*
  thus *thesis* **by** (*fact that*)
qed

lemma *AC-E-prop*:
  assumes $\bigwedge l.\ l \in \Lambda \Longrightarrow \exists\, al \in A\ l.\ P\ l\ al$
  obtains a **where** $a \in (\Pi\ l \in \Lambda.\ A\ l)$
    and $\bigwedge l.\ l \in \Lambda \Longrightarrow P\ l\ (a\ l)$
proof –
  let $?A' = \lambda l'.\ \{al \in A\ l'.\ P\ l'\ al\}$
  {
    fix l
    assume $l \in \Lambda$
    with *assms* **have** $?A'\ l \neq \{\}$ **by** *blast*
  }
  then obtain a **where** $*$: $a \in (\Pi\ l \in \Lambda.\ ?A'\ l)$ **by** (*elim AC-E*)
  {
    fix l
    assume $l \in \Lambda$
    with $*$ **have** $a\ l \in\ ?A'\ l$ **by** *auto*
    hence $a\ l \in A\ l$ **by** *simp*
  }
  hence $a \in (\Pi\ l \in \Lambda.\ A\ l)$ **by** *simp*
  moreover have $\bigwedge l.\ l \in \Lambda \Longrightarrow P\ l\ (a\ l)$
  proof –
    fix l
    assume $l \in \Lambda$
    with $*$ **have** $a\ l \in\ ?A'\ l$ **by** *auto*
    thus $P\ l\ (a\ l)$ **by** *simp*
  qed
  ultimately show *thesis* **by** (*fact that*)
qed

lemma *AC-E-ex*:
  assumes $\bigwedge l.\ l \in \Lambda \Longrightarrow \exists\, x.\ P\ l\ x$
  obtains a **where** $a \in (\Pi\ l \in \Lambda.\ \{x.\ P\ l\ x\})$
proof –
  {
    fix l
    assume $l \in \Lambda$
    with *assms* **have** $\{x.\ P\ l\ x\} \neq \{\}$ **by** *simp*
  }
  then obtain a **where** a: $a \in (\Pi\ l \in \Lambda.\ \{x.\ P\ l\ x\})$ **by** (*elim AC-E*)

```
  thus thesis by (fact that)
qed

lemma Pi-one-point:
  assumes (Π l ∈ Λ. A l) ≠ {}
    and l ∈ Λ
    and al ∈ A l
  obtains a where a ∈ (Π l ∈ Λ. A l)
    and a l = al
proof −
  from assms obtain a where a ∈ (Π l ∈ Λ. A l) by blast
  let ?a′ = λl′. if l′ = l then al else a l′
  {
    fix l′
    assume l′ ∈ Λ
    {
      assume l′ = l
      hence ?a′ l′ = al by simp
      also from assms(3) have . . . ∈ A l by simp
      also from ‹l′ = l› have . . . = A l′ by simp
      finally have ?a′ l′ ∈ A l′ by simp
    }
    moreover {
      assume l′ ≠ l
      hence ?a′ l′ = a l′ by simp
      also from ‹a ∈ (Π l ∈ Λ. A l)› and ‹l′ ∈ Λ› have . . . ∈ A l′ by auto
      finally have ?a′ l′ ∈ A l′ by simp
    }
    ultimately have ?a′ l′ ∈ A l′ by simp
  }
  hence ?a′ ∈ (Π l ∈ Λ. A l) by simp
  moreover have ?a′ l = al by simp
  ultimately show thesis by (fact that)
qed

definition pie :: ′a set ⇒ (′a ⇒ ′b) ⇒ ′a ⇒ ′b
  where pie Λ a = (λl. if l ∈ Λ then a l else undefined)

syntax
-pie :: pttrn ⇒ ′a set ⇒ (′a ⇒ ′b) ⇒ ′a ⇒ ′b ((1′(-∈-./ -′)))

translations
(l∈Λ. a) ⇌ CONST pie Λ (λl. a)

lemma pie-eq1 [iff]:
```

assumes $l \in \Lambda$
shows $(l \in \Lambda.\ a\ l)\ l = a\ l$
using *assms* **unfolding** *pie-def* **by** *simp*

lemma *pie-eq2* [*iff*]:
assumes $l \notin \Lambda$
shows $(l \in \Lambda.\ a\ l)\ l = \mathit{undefined}$
using *assms* **unfolding** *pie-def* **by** *simp*

definition *proj* :: $'a \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b$
where $\mathit{proj}\ l\ a = a\ l$

lemmas *proj-eq* = *proj-def*

lemma *Pi-imp-proj*:
assumes $a \in (\Pi\ l \in \Lambda.\ A\ l)$
and $l \in \Lambda$
shows $\mathit{proj}\ l\ a \in A\ l$
using *assms* **unfolding** *proj-def* **by** *auto*

1.5.5 E) A Theorem on Map

theorem *thm-1-7-a-a*:
assumes $f\ \text{`}\ A = B$
obtains s **where** $s\ \text{`}\ B \subseteq A$
and $\mathit{id\text{-}on}\ (f \circ s)\ B$
proof $-$
{
fix b
assume $b \in B$
with *assms*(*1*) **obtain** a **where** $a \in A$ **and** $b = f\ a$ **by** *auto*
hence $a \in f\ -\text{`}\ \{b\}$ **by** *auto*
with $\langle a \in A \rangle$ **have** $(f\ -\text{`}\ \{b\}) \cap A \neq \{\}$ **by** *auto*
}
then obtain s **where** $*$: $s \in (\Pi\ b \in B.\ (f\ -\text{`}\ \{b\}) \cap A)$ **by** (*rule AC-E*)
have $s\ \text{`}\ B \subseteq A$
proof (*rule subsetI*)
fix a
assume $a \in s\ \text{`}\ B$
then obtain b **where** $b \in B$ **and** $a = s\ b$ **by** *auto*
from *this*(*1*) **and** $*$ **have** $s\ b \in A$ **by** *auto*
with $\langle a = s\ b \rangle$ **show** $a \in A$ **by** *simp*
qed
moreover have $\mathit{id\text{-}on}\ (f \circ s)\ B$
proof (*rule id-onI*)

```
    fix b
    assume b ∈ B
    with * have s b ∈ f -` {b} by auto
    thus (f ∘ s) b = b by simp
  qed
  ultimately show thesis by (fact that)
qed

theorem thm-1-7-a-b:
  assumes f ` A ⊆ B
    and s ` B ⊆ A
    and id-on (f ∘ s) B
  shows f ` A = B
proof -
  from assms(3) have (f ∘ s) ` B = B by (fact id-on-imp-surj-on)
  with assms(1,2) show ?thesis by (intro prob-1-4-10-a[where A = B and f = s and g = f])
qed

theorem thm-1-7-a:
  assumes f ` A ⊆ B
  shows f ` A = B ⟷ (∃ s. s ` B ⊆ A ∧ id-on (f ∘ s) B)
proof (rule iffI)
  assume f ` A = B
  then obtain s where s ` B ⊆ A and id-on (f ∘ s) B by (rule thm-1-7-a-a)
  thus ∃ s. s ` B ⊆ A ∧ id-on (f ∘ s) B by auto
next
  assume ∃ s. s ` B ⊆ A ∧ id-on (f ∘ s) B
  then obtain s where s ` B ⊆ A and id-on (f ∘ s) B by auto
  with assms show f ` A = B by (rule thm-1-7-a-b)
qed

theorem thm-1-7-b-a:
  assumes A = {} ⟹ B = {} — This assumption is not specified in the book. However, there
exists a counterexample without it.
    — The assumption f ` A ⊆ B is not necessary.
    and inj-on f A
  obtains r where r ` B ⊆ A
    and id-on (r ∘ f) A
proof -
  {
    assume A = {}
    with assms(1) have B = {} by simp
    let ?r = λb. undefined
    from ‹A = {}› and ‹B = {}› have ?r ` B ⊆ A by simp
    moreover from ‹A = {}› have id-on (?r ∘ f) A by simp
```

ultimately have *thesis* **by** (*fact that*)

}

moreover {

assume $A \neq \{\}$

then obtain a **where** $a \in A$ **by** *auto*

let $?r = \lambda b.$ *if* $b \in f$ ‘ A *then the-inv-into* $A\ f\ b$ *else* a

have $?r$ ‘ $B \subseteq A$

proof (*rule subsetI*)

fix a'

assume $a' \in ?r$ ‘ B

then obtain b **where** $b \in B$ **and** $a' = ?r\ b$ **by** *auto*

{

assume $b \in f$ ‘ A

then obtain a'' **where** $a'' \in A$ **and** $b = f\ a''$ **by** *auto*

from ⟨$b \in f$ ‘ A⟩ **have** $?r\ b =$ *the-inv-into* $A\ f\ b$ **by** *simp*

with ⟨$a' = ?r\ b$⟩ **have** *the-inv-into* $A\ f\ b = a'$ **by** *simp*

with ⟨$b = f\ a''$⟩ **have** *the-inv-into* $A\ f\ (f\ a'') = a'$ **by** *simp*

moreover from *this* **and** *assms*(*2*) **and** ⟨$a'' \in A$⟩ **have** *the-inv-into* $A\ f\ (f\ a'') = a''$ **by** (*intro the-inv-into-f-f*)

ultimately have $a' = a''$ **by** *simp*

with ⟨$a'' \in A$⟩ **have** $a' \in A$ **by** *simp*

}

moreover {

assume $b \notin f$ ‘ A

hence $?r\ b = a$ **by** *simp*

with ⟨$a' = ?r\ b$⟩ **have** $a' = a$ **by** *simp*

with ⟨$a \in A$⟩ **have** $a' \in A$ **by** *simp*

}

ultimately show $a' \in A$ **by** *auto*

qed

moreover have *id-on* $(?r \circ f)\ A$

proof (*rule id-onI*)

fix a'

assume $a' \in A$

hence $f\ a' \in f$ ‘ A **by** *simp*

hence $?r\ (f\ a') =$ *the-inv-into* $A\ f\ (f\ a')$ **by** *simp*

also from *assms*(*2*) **and** ⟨$a' \in A$⟩ **have** $\ldots = a'$ **by** (*intro the-inv-into-f-f*)

finally have $?r\ (f\ a') = a'$ **.**

thus $(?r \circ f)\ a' = a'$ **by** *simp*

qed

ultimately have *thesis* **by** (*fact that*)

}

ultimately show *thesis* **by** *auto*

qed

theorem *thm-1-7-b-b*:
  assumes — The assumption f ‘ $A \subseteq B$ is not necessary.
    — The assumption r ‘ $B \subseteq A$ is not necessary.
    id-on $(r \circ f)$ A
  shows *inj-on* f A
proof –
  from *assms* **have** *inj-on* $(r \circ f)$ A **by** (*fact id-on-imp-inj-on*)
  thus *?thesis* **by** (*fact prob-1-4-10-b*)
qed

theorem *thm-1-7-b*:
  assumes $A = \{\} \Longrightarrow B = \{\}$— This assumption is not specified in the book. However, there exists a counterexample without it.
  shows *inj-on* f $A \longleftrightarrow (\exists r.\ r$ ‘ $B \subseteq A \wedge$ *id-on* $(r \circ f)$ $A)$
  using *assms thm-1-7-b-a thm-1-7-b-b* **by** *metis*

definition *right-inv-into* :: $'a$ *set* $\Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow$ *bool*
  where *right-inv-into* A f $s \longleftrightarrow s$ ‘ $(f$ ‘ $A) \subseteq A \wedge$ *id-on* $(f \circ s)$ $(f$ ‘ $A)$

lemma *right-inv-into*:
  obtains s **where** *right-inv-into* A f s
proof –
  obtain s **where** s ‘ f ‘ $A \subseteq A$ **and** *id-on* $(f \circ s)$ $(f$ ‘ $A)$ **using** *thm-1-7-a-a* **by** *blast*
  thus *thesis* **using** *that* **unfolding** *right-inv-into-def* **by** *simp*
qed

lemma *right-inv-intoI*:
  assumes $\bigwedge b.\ b \in f$ ‘ $A \Longrightarrow s\ b \in A$
    and $\bigwedge b.\ b \in f$ ‘ $A \Longrightarrow f\ (s\ b) = b$
  shows *right-inv-into* A f s
  using *assms* **unfolding** *right-inv-into-def* **by** *fastforce*

lemma *right-inv-intoD1*:
  assumes *right-inv-into* A f s
  shows s ‘ $(f$ ‘ $A) \subseteq A$
  using *assms* **unfolding** *right-inv-into-def* **by** *simp*

lemma *right-inv-intoD2-pf*:
  assumes *right-inv-into* A f s
  shows *id-on* $(f \circ s)$ $(f$ ‘ $A)$
  using *assms* **unfolding** *right-inv-into-def* **by** *simp*

lemma *right-inv-intoD2*:
  assumes *right-inv-into* A f s
    and $x \in f$ ‘ A

```
  shows f (s x) = x
proof -
  from assms(1) have id-on (f ∘ s) (f ` A) by (fact right-inv-intoD2-pf)
  with assms(2) have (f ∘ s) x = x by blast
  thus ?thesis by simp
qed

definition left-inv-into :: 'a set ⇒ ('a ⇒ 'b) ⇒ ('b ⇒ 'a) ⇒ bool
  where left-inv-into A f r ⟷ id-on (r ∘ f) A

lemma left-inv-into:
  assumes inj-on f A
  obtains r where left-inv-into A f r
proof -
  {
    assume A = {}
    then obtain r where id-on (r ∘ f) A by simp
    hence left-inv-into A f r unfolding left-inv-into-def by simp
    hence thesis by (fact that)
  }
  moreover {
    assume A ≠ {}
    with assms obtain r where id-on (r ∘ f) A by (blast intro: thm-1-7-b-a)
    hence left-inv-into A f r unfolding left-inv-into-def by simp
    hence thesis by (fact that)
  }
  ultimately show thesis by auto
qed

lemma left-inv-intoI:
  assumes ⋀a. a ∈ A ⟹ r (f a) = a
  shows left-inv-into A f r
  using assms unfolding left-inv-into-def by auto

lemma left-inv-intoD1:
  assumes left-inv-into A f r
  shows r ` f ` A ⊆ A
  using assms unfolding left-inv-into-def by auto

lemma left-inv-intoD2-pf:
  assumes left-inv-into A f r
  shows id-on (r ∘ f) A
  using assms unfolding left-inv-into-def by simp

lemma left-inv-intoD2:
```

assumes *left-inv-into* A f r
and $a \in A$
shows $r\ (f\ a) = a$
using *assms* **unfolding** *left-inv-into-def* **by** *fastforce*

corollary *cor-inj-on-iff-surj-on-a*:
assumes $A = \{\} \Longrightarrow B = \{\}$
and $f\ `\ A \subseteq B$
and *inj-on* f A
obtains g **where** $g\ `\ B = A$
proof −
from *assms*(*1*,*3*) **obtain** g **where** $g\ `\ B \subseteq A$ **and** *id-on* $(g \circ f)$ A **by** (*elim thm-1-7-b-a*)
with ‹$f\ `\ A \subseteq B$› **have** $g\ `\ B = A$ **by** (*intro thm-1-7-a-b*)
thus *thesis* **by** (*fact that*)
qed

corollary *cor-inj-on-iff-surj-on-b*:
assumes $f\ `\ A = B$
obtains g **where** $g\ `\ B \subseteq A$ **and** *inj-on* g B
proof −
from *assms* **obtain** g **where** $g\ `\ B \subseteq A$ **and** *id-on* $(f \circ g)$ B **by** (*elim thm-1-7-a-a*)
from *this*(*2*) **have** *inj-on* g B **by** (*intro thm-1-7-b-b*)
with ‹$g\ `\ B \subseteq A$› **show** *thesis* **by** (*fact that*)
qed

corollary *cor-inj-on-iff-surj-on*:
assumes $A = \{\} \Longrightarrow B = \{\}$ — This assumption is not specified in the book. However, there exists a counterexample without it.
shows $(\exists f.\ f\ `\ A \subseteq B \wedge$ *inj-on* $f\ A) \longleftrightarrow (\exists g.\ g\ `\ B = A)$
proof (*rule iffI*)
assume $\exists f.\ f\ `\ A \subseteq B \wedge$ *inj-on* f A
then obtain f **where** $f\ `\ A \subseteq B$ **and** *inj-on* f A **by** *auto*
with *assms* **obtain** g **where** $g\ `\ B = A$ **by** (*elim cor-inj-on-iff-surj-on-a*)
thus $\exists f.\ f\ `\ B = A$ **by** *auto*
next
assume $\exists g.\ g\ `\ B = A$
then obtain g **where** $g\ `\ B = A$ **..**
then obtain f **where** $f\ `\ A \subseteq B$ **and** *inj-on* f A **by** (*elim cor-inj-on-iff-surj-on-b*)
thus $\exists f.\ f\ `\ A \subseteq B \wedge$ *inj-on* f A **by** *auto*
qed

1.5.6 F) Multivariate Maps

1.5.7 Problems

proposition *prob-1-5-5-a*:
shows $(\bigcup l \in \Lambda.\ A\ l) \cap (\bigcup \mu \in M.\ B\ \mu) = (\bigcup (l, \mu) \in \Lambda \times M.\ A\ l \cap B\ \mu)$
by *auto*

proposition *prob-1-5-5-b*:
shows $(\bigcap l \in \Lambda.\ A\ l) \cup (\bigcap \mu \in M.\ B\ \mu) = (\bigcap (l, \mu) \in \Lambda \times M.\ A\ l \cup B\ \mu)$
by *blast*

proposition *prob-1-5-5-c*:
shows $(\bigcup l \in \Lambda.\ A\ l) \times (\bigcup \mu \in M.\ B\ \mu) = (\bigcup (l, \mu) \in \Lambda \times M.\ A\ l \times B\ \mu)$
by *auto*

proposition *prob-1-5-5-d*:
assumes $\Lambda \neq \{\}$
and $M \neq \{\}$ — These assumptions are not specified in the book. However, there exists a counterexample without them.
shows $(\bigcap l \in \Lambda.\ A\ l) \times (\bigcap \mu \in M.\ B\ \mu) = (\bigcap (l, \mu) \in \Lambda \times M.\ A\ l \times B\ \mu)$
using *assms* **by** *blast*

lemma *disjoint-family-on-imp-eq*:
assumes *disjoint-family-on* $A\ \Lambda$
and $l \in \Lambda$
and $l' \in \Lambda$
and $a \in A\ l$
and $a \in A\ l'$
shows $l = l'$
proof −
from *assms(4,5)* **have** $a \in A\ l \cap A\ l'$ **by** *simp*
{
assume $l \neq l'$
with *assms(1−3)* **have** $A\ l \cap A\ l' = \{\}$ **by** (*elim disjoint-family-onD*)
with $\langle a \in A\ l \cap A\ l' \rangle$ **have** *False* **by** *simp*
}
thus *?thesis* **by** *auto*
qed

lemma *disjoint-family-on-imp-uniq-idx*:
assumes *disjoint-family-on* $A\ \Lambda$
and $a \in (\bigcup l \in \Lambda.\ A\ l)$
shows $\exists! l \in \Lambda.\ a \in A\ l$

```
proof (rule ex-ex1I)
  from assms(2) obtain l where l ∈ Λ and a ∈ A l by auto
  thus ∃ l. l ∈ Λ ∧ a ∈ A l by auto
next
  thm disjoint-family-onD
  fix l and l′
  assume l ∈ Λ ∧ a ∈ A l and l′ ∈ Λ ∧ a ∈ A l′
  with assms(1) show l = l′ by (blast dest: disjoint-family-on-imp-eq)
qed

proposition prob-1-5-6-existence:
  assumes disjoint-family-on A Λ
    and ⋀l. l ∈ Λ ⟹ (f l) ` (A l) ⊆ B
  obtains F where F ` (⋃ l ∈ Λ. A l) ⊆ B
    and ⋀l. l ∈ Λ ⟹ ext-eq-on (A l) F (f l)
proof −
  let ?F = λa. f (THE l. l ∈ Λ ∧ a ∈ A l) a
  have ?F ` (⋃ l ∈ Λ. A l) ⊆ B
  proof (rule image-subsetI)
    fix a
    assume a ∈ (⋃ l ∈ Λ. A l)
    with assms(1) have ∃!l ∈ Λ. a ∈ A l by (elim disjoint-family-on-imp-uniq-idx)
    then obtain l where l ∈ Λ ∧ a ∈ A l by auto
    hence l ∈ Λ and ‹a ∈ A l› by auto
    from ‹∃!l ∈ Λ. a ∈ A l› and ‹l ∈ Λ ∧ a ∈ A l› have (THE l. l ∈ Λ ∧ a ∈ A l) = l by auto
    hence ?F a = f l a by simp
    from assms(2) and ‹l ∈ Λ› have (f l) ` (A l) ⊆ B by simp
    with ‹a ∈ A l› have f l a ∈ B by auto
    with ‹?F a = f l a› show ?F a ∈ B by simp
  qed
  moreover have ⋀l. l ∈ Λ ⟹ ext-eq-on (A l) ?F (f l)
  proof −
    fix l
    assume l ∈ Λ
    {
      fix a
      assume a ∈ A l
      with ‹l ∈ Λ› have l ∈ Λ ∧ a ∈ A l ..
      moreover {
        fix l′
        assume *: l′ ∈ Λ ∧ a ∈ A l′
        hence l′ ∈ Λ and a ∈ A l′ by simp+
        with ‹l ∈ Λ› and ‹a ∈ A l› and assms(1) have l′ = l by (elim disjoint-family-on-imp-eq)
      }
      ultimately have (THE l. l ∈ Λ ∧ a ∈ A l) = l by (intro the-equality)
```

hence *?F a* $=$ *f l a* **by** *simp*
}
thus *ext-eq-on* $(A\ l)$ *?F* $(f\ l)$ **by** (*intro ext-eq-onI*)
qed
ultimately show *thesis* **by** (*fact that*)
qed

proposition *prob-1-5-6-uniqueness*:
assumes — The assumption *disjoint-family-on* $A\ \Lambda$ is not necessary.
— The assumption F ‘ $\bigcup\ (A$ ‘ $\Lambda) \subseteq B$ is not necessary.
$\bigwedge l.\ l \in \Lambda \Longrightarrow$ *ext-eq-on* $(A\ l)\ F\ (f\ l)$
— The assumption F' ‘ $\bigcup\ (A$ ‘ $\Lambda) \subseteq B$ is not necessary.
and $\bigwedge l.\ l \in \Lambda \Longrightarrow$ *ext-eq-on* $(A\ l)\ F'\ (f\ l)$
shows *ext-eq-on* $(\bigcup l \in \Lambda.\ A\ l)\ F\ F'$
proof (*rule ext-eq-onI*)
fix *a*
assume $a \in (\bigcup l \in \Lambda.\ A\ l)$
then obtain *l* **where** $l \in \Lambda$ **and** $a \in A\ l$ **by** *blast*
from *this*(*1*) **and** *assms*(*1*) **have** *ext-eq-on* $(A\ l)\ F\ (f\ l)$ **by** *simp*
also have *ext-eq-on* $(A\ l)\ \ldots\ F'$
proof −
from ‹$l \in \Lambda$› **and** *assms*(*2*) **have** *ext-eq-on* $(A\ l)\ F'\ (f\ l)$ **by** *simp*
thus *?thesis* **by** (*fact ext-eq-on-sym*)
qed
finally have *ext-eq-on* $(A\ l)\ F\ F'$ **.**
with ‹$a \in A\ l$› **show** $F\ a = F'\ a$ **by** *auto*
qed

proposition *prob-1-5-7*:
assumes $\bigwedge l.\ l \in \Lambda \Longrightarrow A\ l \neq \{\}$
and $l \in \Lambda$
shows (*proj l*) ‘ $(\Pi\ l \in \Lambda.\ A\ l) = A\ l$
proof (*rule surj-onI*)
fix *a*
assume $a \in (\Pi\ l \in \Lambda.\ A\ l)$
with *assms*(*2*) **show** *proj l a* $\in A\ l$ **by** (*intro Pi-imp-proj*)
next
fix *al*
assume *al* $\in A\ l$
moreover from *assms*(*1*) **have** $(\Pi\ l \in \Lambda.\ A\ l) \neq \{\}$ **by** (*auto intro*: *AC*)
moreover note *assms*(*2*)
ultimately obtain *a* **where** $a \in (\Pi\ l \in \Lambda.\ A\ l)$ **and** $a\ l =$ *al* **by** (*elim Pi-one-point*)
from *this*(*2*) **have** *proj l a* $=$ *al* **unfolding** *proj-eq* **by** *simp*
with ‹$a \in (\Pi\ l \in \Lambda.\ A\ l)$› **show** *al* $\in$ *proj l* ‘ *Pi* $\Lambda\ A$ **by** *auto*
qed

proposition *prob-1-5-8-a*:
  assumes $\bigwedge l.\ l \in \Lambda \Longrightarrow A\ l \neq \{\}$
    and $(\Pi\ l \in \Lambda.\ A\ l) \subseteq (\Pi\ l \in \Lambda.\ B\ l)$
    and $l \in \Lambda$
  shows $A\ l \subseteq B\ l$
proof −
  {
    assume $\exists\, l \in \Lambda.\ B\ l = \{\}$
    then obtain l' **where** $l' \in \Lambda$ **and** $B\ l' = \{\}$ **by** *auto*
    hence $(\Pi\ l \in \Lambda.\ B\ l) = \{\}$ **by** *auto*
    with *assms(2)* **have** $(\Pi\ l \in \Lambda.\ A\ l) = \{\}$ **by** *simp*
    moreover from *assms(1)* **have** $(\Pi\ l \in \Lambda.\ A\ l) \neq \{\}$ **by** (*auto intro*: *AC*)
    ultimately have *False* **by** *simp*
  }
  hence $*$: $\forall\, l \in \Lambda.\ B\ l \neq \{\}$ **by** *auto*
  from *assms(1,3)* **have** $A\ l = (proj\ l)$ ‘ $(\Pi\ l \in \Lambda.\ A\ l)$ **by** (*fact prob-1-5-7[THEN sym]*)
  also from *assms(2)* **have** $\ldots \subseteq (proj\ l)$ ‘ $(\Pi\ l \in \Lambda.\ B\ l)$ **by** *auto*
  also from $*$ **and** *assms(3)* **have** $\ldots = B\ l$ **by** (*simp add*: *prob-1-5-7*)
  finally show *?thesis* **.**
qed

proposition *prob-1-5-8-b*:
  assumes — The assumption $\bigwedge l.\ l \in \Lambda \Longrightarrow A\ l \neq \{\}$ is not necessary.
   $\bigwedge l.\ l \in \Lambda \Longrightarrow A\ l \subseteq B\ l$
  shows $(\Pi\ l \in \Lambda.\ A\ l) \subseteq (\Pi\ l \in \Lambda.\ B\ l)$
  using *assms* **by** (*fact Pi-mono*)

proposition *prob-1-5-8*:
  assumes $\bigwedge l.\ l \in \Lambda \Longrightarrow A\ l \neq \{\}$
  shows $(\Pi\ l \in \Lambda.\ A\ l) \subseteq (\Pi\ l \in \Lambda.\ B\ l) \longleftrightarrow (\forall\, l \in \Lambda.\ A\ l \subseteq B\ l)$
proof (*rule iffI*)
  assume $(\Pi\ l \in \Lambda.\ A\ l) \subseteq (\Pi\ l \in \Lambda.\ B\ l)$
  {
    fix l
    assume $l \in \Lambda$
    with *assms* **and** ‹$(\Pi\ l \in \Lambda.\ A\ l) \subseteq (\Pi\ l \in \Lambda.\ B\ l)$› **have** $A\ l \subseteq B\ l$ **by** (*rule prob-1-5-8-a*)
  }
  thus $\forall\, l \in \Lambda.\ A\ l \subseteq B\ l$ **..**
next
  assume $\forall\, l \in \Lambda.\ A\ l \subseteq B\ l$
  thus $(\Pi\ l \in \Lambda.\ A\ l) \subseteq (\Pi\ l \in \Lambda.\ B\ l)$ **by** (*auto intro*: *prob-1-5-8-b*)
qed

proposition *prob-1-5-9*:

shows $(\Pi\ l \in \Lambda.\ A\ l) \cap (\Pi\ l \in \Lambda.\ B\ l) = (\Pi\ l \in \Lambda.\ A\ l \cap B\ l)$
by (*fact Pi-Int*)

proposition *prob-1-5-10*:
assumes $(\Pi\ l \in \Lambda.\ A\ l) \neq \{\}$ — This assumption is not specified in the book. However, there exists a counterexample without it.
and $\bigwedge l.\ l \in \Lambda \Longrightarrow (f\ l)$ ‘ $(A\ l) \subseteq B\ l$
defines F: $F\ a \equiv (l \in \Lambda.\ f\ l\ (a\ l))$
obtains $(\forall\, b \in (\Pi\ l \in \Lambda.\ B\ l).\ \exists\, a \in (\Pi\ l \in \Lambda.\ A\ l).\ \textit{ext-eq-on}\ \Lambda\ (F\ a)\ b)$
$\longleftrightarrow (\forall\, l \in \Lambda.\ f\ l$ ‘ $A\ l = B\ l)$
and $(\forall\, a \in (\Pi\ l \in \Lambda.\ A\ l).\ \forall\, a' \in (\Pi\ l \in \Lambda.\ A\ l).\ F\ a = F\ a' \longrightarrow \textit{ext-eq-on}\ \Lambda\ a\ a')$
$\longleftrightarrow (\forall\, l \in \Lambda.\ \textit{inj-on}\ (f\ l)\ (A\ l))$
proof −
let $?\mathfrak{A} = \Pi\ l \in \Lambda.\ A\ l$
and $?\mathfrak{B} = \Pi\ l \in \Lambda.\ B\ l$
let $?L0 = \forall\, b \in ?\mathfrak{B}.\ \exists\, a \in ?\mathfrak{A}.\ \textit{ext-eq-on}\ \Lambda\ (F\ a)\ b$
and $?R0 = \forall\, l \in \Lambda.\ f\ l$ ‘ $A\ l = B\ l$
and $?L1 = \forall\, a \in ?\mathfrak{A}.\ \forall\, a' \in ?\mathfrak{A}.\ F\ a = F\ a' \longrightarrow \textit{ext-eq-on}\ \Lambda\ a\ a'$
and $?R1 = \forall\, l \in \Lambda.\ \textit{inj-on}\ (f\ l)\ (A\ l)$
{
fix a
assume $a \in ?\mathfrak{A}$
{
fix l
assume $l \in \Lambda$
with ‹$a \in ?\mathfrak{A}$› **have** $a\ l \in A\ l$ **by** *auto*
with ‹$l \in \Lambda$› **and** *assms*(*2*) **have** $f\ l\ (a\ l) \in B\ l$ **by** *auto*
with ‹$l \in \Lambda$› **have** $F\ a\ l \in B\ l$ **unfolding** F **by** *simp*
}
hence $F\ a \in ?\mathfrak{B}$ **by** *simp*
}
hence F ‘ $?\mathfrak{A} \subseteq ?\mathfrak{B}$ **by** *auto*
with *assms*(*1*) **have** $?\mathfrak{B} \neq \{\}$ **by** *blast*
have $*$: $F\ a\ l = f\ l\ (a\ l)$ **if** $l \in \Lambda$ **for** a **and** l **using** *that* **unfolding** F **by** *simp*
have $?L0 \longleftrightarrow ?R0$
proof (*rule iffI*)
assume $?L0$
{
fix l
assume $l \in \Lambda$
{
fix al
assume $al \in A\ l$
with ‹$l \in \Lambda$› **have** $f\ l\ al \in B\ l$ **by** (*auto dest*: *assms*(*2*))
}

moreover {
  fix *bl*
  assume $bl \in B\ l$
  with ‹$?\mathfrak{B} \neq \{\}$› **and** ‹$l \in \Lambda$› **obtain** b **where** $b \in ?\mathfrak{B}$ **and** $b\ l = bl$ **by** (*elim Pi-one-point*)
  from *this*(1) **and** ‹*?L0*› **obtain** a **where** $a \in ?\mathfrak{A}$ **and** *ext-eq-on* $\Lambda\ (F\ a)\ b$ **by** *blast*
  from *this*(2) **and** ‹$l \in \Lambda$› **have** $F\ a\ l = b\ l$ **by** *auto*
  with ‹$l \in \Lambda$› **and** $*$ **have** $f\ l\ (a\ l) = b\ l$ **by** *simp*
  also from ‹$b\ l = bl$› **have** $\ldots = bl$ **by** *simp*
  moreover from ‹$a \in ?\mathfrak{A}$› **and** ‹$l \in \Lambda$› **have** $a\ l \in A\ l$ **by** *auto*
  ultimately have $\exists\, al \in A\ l.\ f\ l\ al = bl$ **by** *auto*
}
ultimately have $f\ l\ `\ A\ l = B\ l$ **by** *blast*
}
thus *?R0* **by** *simp*
next
assume *?R0*
{
  fix b
  assume $b \in ?\mathfrak{B}$
  {
    fix l
    assume $l \in \Lambda$
    with ‹$b \in ?\mathfrak{B}$› **have** $b\ l \in B\ l$ **by** *auto*
    with ‹$l \in \Lambda$› **and** ‹*?R0*› **obtain** *al* **where** $al \in A\ l$ **and** $b\ l = f\ l\ al$ **by** *auto*
    hence $\exists\, al \in A\ l.\ b\ l = f\ l\ al$ **by** *auto*
  }
  then obtain a **where** $a \in ?\mathfrak{A}$ **and** $**$: $\bigwedge l.\ l \in \Lambda \Longrightarrow b\ l = f\ l\ (a\ l)$
    using *AC-E-prop*[**where** $P = \lambda l.\ \lambda al.\ b\ l = f\ l\ al$] **by** *blast*
  {
    fix l
    assume $l \in \Lambda$
    with $**$ **have** $b\ l = f\ l\ (a\ l)$ **by** *simp*
    also from ‹$l \in \Lambda$› **and** $*$ **have** $\ldots = F\ a\ l$ **by** *simp*
    finally have $F\ a\ l = b\ l$ **by** *simp*
  }
  with ‹$a \in ?\mathfrak{A}$› **have** $\exists\, a \in ?\mathfrak{A}.$ *ext-eq-on* $\Lambda\ (F\ a)\ b$ **by** *auto*
}
thus *?L0* **by** *blast*
qed
moreover have $?L1 \longleftrightarrow ?R1$
proof (*rule iffI*)
  assume *?L1*
  {
    fix l
    assume $l \in \Lambda$

{
fix al and al'
assume $al \in A\ l$ and $al' \in A\ l$ and $f\ l\ al = f\ l\ al'$
from ⟨$?\mathfrak{A} \neq \{\}$⟩ and $this(1)$ and ⟨$l \in \Lambda$⟩ obtain a where $a \in ?\mathfrak{A}$ and $a\ l = al$
using $Pi\text{-}one\text{-}point$ by $metis$
let $?a' = \lambda l'.\ if\ l' = l\ then\ al'\ else\ a\ l'$
{
fix l'
assume $l' \in \Lambda$
{
assume $l' = l$
with ⟨$al' \in A\ l$⟩ have $?a'\ l' \in A\ l'$ by $simp$
}
moreover {
assume $l' \neq l$
with ⟨$a \in ?\mathfrak{A}$⟩ and ⟨$l' \in \Lambda$⟩ have $?a'\ l' \in A\ l'$ by $auto$
}
ultimately have $?a'\ l' \in A\ l'$ by $blast$
}
hence $?a' \in ?\mathfrak{A}$ by $simp$
{
fix l'
{
assume $l' \in \Lambda$
with ⟨$a\ l = al$⟩ and ⟨$f\ l\ al = f\ l\ al'$⟩ and $*$ have $F\ a\ l' = F\ ?a'\ l'$ by $simp$
}
moreover {
assume $l' \notin \Lambda$
hence $F\ a\ l' = F\ ?a'\ l'$ unfolding F by $simp$
}
ultimately have $F\ a\ l' = F\ ?a'\ l'$ by $auto$
}
hence $F\ a = F\ ?a'$ by $auto$
with ⟨$a \in ?\mathfrak{A}$⟩ and ⟨$?a' \in ?\mathfrak{A}$⟩ and ⟨$?L1$⟩ have $ext\text{-}eq\text{-}on\ \Lambda\ a\ ?a'$ by $blast$
with ⟨$l \in \Lambda$⟩ have $a\ l = ?a'\ l$ by $auto$
with ⟨$a\ l = al$⟩ have $al = al'$ by $simp$
}
hence $inj\text{-}on\ (f\ l)\ (A\ l)$ by ($fact\ inj\text{-}onI$)
}
thus $?R1$ by $simp$
next
assume $?R1$
{
fix a and a'
assume $a \in ?\mathfrak{A}$ and $a' \in ?\mathfrak{A}$ and $F\ a = F\ a'$

```
      {
        fix l
        assume l ∈ Λ
        from ‹F a = F a'› have F a l = F a' l by simp
        with ‹l ∈ Λ› have f l (a l) = f l (a' l) by (simp only: *)
        moreover from ‹a ∈ ?𝔄› and ‹l ∈ Λ› have a l ∈ A l by auto
        moreover from ‹a' ∈ ?𝔄› and ‹l ∈ Λ› have a' l ∈ A l by auto
        moreover from ‹l ∈ Λ› and ‹?R1› have inj-on (f l) (A l) by simp
        ultimately have a l = a' l using ‹?R1› by (elim inj-onD)
      }
      hence ext-eq-on Λ a a' by auto
    }
    thus ?L1 by blast
  qed
  ultimately show thesis by (fact that)
qed

proposition prob-1-5-11-a:
  assumes f ‘ A = B
    and right-inv-into A f s
    and right-inv-into A f s'
    and s ‘ B ⊆ s' ‘ B
  shows ext-eq-on B s s'
proof (rule ext-eq-onI)
  fix b
  assume b ∈ B
  with assms(4) have s b ∈ s' ‘ B by auto
  then obtain b' where b' ∈ B and s' b' = s b by auto
  from this(2) have f (s' b') = f (s b) by simp
  moreover from assms(1) and ‹b ∈ B› and ‹b' ∈ B› have b ∈ f ‘ A and b' ∈ f ‘ A by simp+
  moreover note assms(2,3)
  ultimately have b' = b by (fastforce dest: right-inv-intoD2)
  with ‹s' b' = s b› show s b = s' b by simp
qed

proposition prob-1-5-11:
  assumes f ‘ A = B
    and right-inv-into A f s
    and right-inv-into A f s'
    and s ‘ B ⊆ s' ‘ B ∨ s' ‘ B ⊆ s ‘ B
  shows ext-eq-on B s s'
proof −
  {
    assume s ‘ B ⊆ s' ‘ B
    with assms(1−3) have ?thesis by (elim prob-1-5-11-a)
```

}
moreover {
assume $s' ` B \subseteq s ` B$
with *assms*(*1*−*3*) **have** *ext-eq-on* B s' s **by** (*elim prob-1-5-11-a*)
hence *?thesis* **by** (*fact ext-eq-on-sym*)
}
moreover note *assms*(*4*)
ultimately show *?thesis* **by** *auto*
qed

proposition *prob-1-5-12-a*:
assumes $f ` A = B$
— The assumption $f' ` B = C$ is not necessary.
and *right-inv-into* A f s
and *right-inv-into* B f' s'
shows *right-inv-into* A $(f' \circ f)$ $(s \circ s')$
proof (*rule right-inv-intoI*)
fix c
assume $c \in (f' \circ f) ` A$
with *assms*(*1*) **have** $c \in f' ` B$ **by** *auto*
moreover from *assms*(*3*) **have** $s' ` f' ` B \subseteq B$ **by** (*elim right-inv-intoD1*)
ultimately have $s'\ c \in B$ **by** *auto*
with *assms*(*1*) **have** $s'\ c \in f ` A$ **by** *simp*
moreover from *assms*(*2*) **have** $s ` f ` A \subseteq A$ **by** (*elim right-inv-intoD1*)
ultimately have $s\ (s'\ c) \in A$ **by** *auto*
thus $(s \circ s')\ c \in A$ **by** *simp*
from ‹$s'\ c \in B$› **and** *assms*(*1*) **have** $s'\ c \in f ` A$ **by** *simp*
with *assms*(*2*) **have** $f\ (s\ (s'\ c)) = s'\ c$ **by** (*elim right-inv-intoD2*)
moreover from ‹$c \in f' ` B$› **and** *assms*(*3*) **have** $f'\ (s'\ c) = c$ **by** (*elim right-inv-intoD2*)
ultimately show $(f' \circ f)\ ((s \circ s')\ c) = c$ **by** *simp*
qed

proposition *prob-1-5-12-b*:
assumes $f ` A \subseteq B$
and $f' ` B \subseteq C$
and *inj-on* f A
and *inj-on* f' B
and *left-inv-into* A f r
and *left-inv-into* B f' r'
shows *left-inv-into* A $(f' \circ f)$ $(r \circ r')$
proof (*rule left-inv-intoI*)
fix a
assume $a \in A$
with *assms*(*1*) **have** $f\ a \in B$ **by** *auto*
with *assms*(*6*) **have** $r'\ (f'\ (f\ a)) = f\ a$ **by** (*auto dest*: *left-inv-intoD2*)

hence $r\ (r'\ (f'\ (f\ a))) = r\ (f\ a)$ **by** *simp*
with $\langle a \in A \rangle$ **and** *assms*(*5*) **have** $r\ (r'\ (f'\ (f\ a))) = a$ **by** (*auto dest*: *left-inv-intoD2*)
thus $(r \circ r')\ ((f' \circ f)\ a) = a$ **by** *simp*
qed

proposition *prob-1-5-13-a*:
assumes — The assumption g ‘ $B \subseteq C$ is not necessary.
— The assumption h ‘ $A \subseteq C$ is not necessary.
f ‘ $A \subseteq B$
and *ext-eq-on* $A\ h\ (g \circ f)$
shows h ‘ $A \subseteq g$ ‘ B
proof (*rule subsetI*)
fix c
assume $c \in h$ ‘ A
then obtain a **where** $a \in A$ **and** $c = h\ a$ **by** *auto*
from $\langle a \in A \rangle$ **and** *assms*(*1*) **have** $f\ a \in B$ **by** *auto*
moreover from $\langle a \in A \rangle$ **and** *assms*(*2*) **and** $\langle c = h\ a \rangle$ **have** $g\ (f\ a) = c$ **by** *auto*
ultimately show $c \in g$ ‘ B **by** *auto*
qed

proposition *prob-1-5-13-b*:
assumes — The assumption g ‘ $B \subseteq C$ is not necessary.
— The assumption h ‘ $A \subseteq C$ is not necessary.
h ‘ $A \subseteq g$ ‘ B
obtains f **where** f ‘ $A \subseteq B$
and *ext-eq-on* $A\ h\ (g \circ f)$
proof –
obtain s **where** $*$: *right-inv-into* $B\ g\ s$ **by** (*elim right-inv-into*)
hence *id-on* $(g \circ s)\ (g$ ‘ $B)$ **by** (*elim right-inv-intoD2-pf*)
let *?f* $= \lambda a.\ s\ (h\ a)$
have *?f* ‘ $A \subseteq B$
proof (*rule image-subsetI*)
fix a
assume $a \in A$
with *assms* **have** $h\ a \in g$ ‘ B **by** *auto*
moreover from $*$ **have** s ‘ g ‘ $B \subseteq B$ **by** (*elim right-inv-intoD1*)
ultimately show *?f* $a \in B$ **by** *auto*
qed
moreover have *ext-eq-on* $A\ h\ (g \circ$ *?f*$)$
proof (*rule ext-eq-onI*)
fix a
assume $a \in A$
with *assms* **have** $h\ a \in g$ ‘ B **by** *auto*
with $*$ **have** $g\ (s\ (h\ a)) = h\ a$ **by** (*elim right-inv-intoD2*)
thus $h\ a = (g \circ$ *?f*$)\ a$ **by** *simp*

qed
ultimately show *thesis* **by** (*rule that*)
qed

proposition *prob-1-5-13*:
— The assumption g ‘ $B \subseteq C$ is not necessary.
— The assumption h ‘ $A \subseteq C$ is not necessary.
shows $(\exists f.\ f$ ‘ $A \subseteq B \wedge$ *ext-eq-on* $A\ h\ (g \circ f)) \longleftrightarrow h$ ‘ $A \subseteq g$ ‘ B
using *prob-1-5-13-a prob-1-5-13-b* **by** *metis*

proposition *prob-1-5-14-a*:
assumes — The assumption f ‘ $A \subseteq B$ is not necessary.
— The assumption h ‘ $A \subseteq C$ is not necessary.
— The assumption g ‘ $B \subseteq C$ is not necessary.
ext-eq-on $A\ h\ (g \circ f)$
and $a \in A$
and $a' \in A$
and $f\ a = f\ a'$
shows $h\ a = h\ a'$
proof –
from *assms(4)* **have** $g\ (f\ a) = g\ (f\ a')$ **by** *simp*
with *assms(1–3)* **show** $h\ a = h\ a'$ **using** *ext-eq-onD* **by** *fastforce*
qed

lemma *the-singleton-equality*:
assumes $a \in A$
and $\bigwedge x.\ x \in A \Longrightarrow x = a$
shows $(THE\ a.\ A = \{a\}) = a$
using *assms* **by** *blast*

proposition *prob-1-5-14-b*:
fixes A :: *'a set*
and B :: *'b set*
and C :: *'c set*
assumes $A = \{\} \Longrightarrow B = \{\}$ — This assumption is not specified in the book. However, there exists a counterexample without it.
— The assumption f ‘ $A \subseteq B$ is not necessary.
and h ‘ $A \subseteq C$
and $\bigwedge a\ a'.\ a \in A \Longrightarrow a' \in A \Longrightarrow f\ a = f\ a' \Longrightarrow h\ a = h\ a'$
obtains g **where** g ‘ $B \subseteq C$ **and** *ext-eq-on* $A\ h\ (g \circ f)$
proof –
{
assume $A \neq \{\}$
then obtain a **where** $a \in A$ **by** *auto*
let $?g = \lambda b.$ *if* $b \in f$ ‘ A *then* $(THE\ c.\ h$ ‘ $(f$ –‘ $\{b\} \cap A) = \{c\})$ *else* $h\ a$

have $*$: $?g\ (f\ a) = h\ a$ **if** $a \in A$ **for** a
proof $-$
  from *that* **have** $f\ a \in f\ `\ A$ **by** *simp*
  hence $?g\ (f\ a) = (THE\ c.\ h\ `\ (f\ -`\ \{f\ a\} \cap A) = \{c\})$ **by** *simp*
  also have $\ldots = h\ a$
  proof (*rule the-singleton-equality*)
    have $a \in f\ -`\ \{f\ a\}$ **by** *simp*
    with *that* **have** $a \in f\ -`\ \{f\ a\} \cap A$ **by** *simp*
    thus $h\ a \in h\ `\ (f\ -`\ \{f\ a\} \cap A)$ **by** *simp*
    fix x
    assume $x \in h\ `\ (f\ -`\ \{f\ a\} \cap A)$
    then obtain a' **where** $a' \in f\ -`\ \{f\ a\} \cap A$ **and** $x = h\ a'$ **by** *auto*
    from $\langle a' \in f\ -`\ \{f\ a\} \cap A\rangle$ **have** $f\ a' = f\ a$ **by** *simp*
    moreover note $\langle a \in A\rangle$
    moreover from $\langle a' \in f\ -`\ \{f\ a\} \cap A\rangle$ **have** $a' \in A$ **by** *simp*
    moreover note *assms*(*3*)
    ultimately have $h\ a' = h\ a$ **by** *blast*
    with $\langle x = h\ a'\rangle$ **show** $x = h\ a$ **by** *simp*
  qed
  finally show *?thesis* **.**
qed
have $?g\ `\ B \subseteq C$
proof (*rule image-subsetI*)
  fix b
  assume $b \in B$
  {
    assume $b \in f\ `\ A$
    then obtain a **where** $a \in A$ **and** $b = f\ a$ **by** *auto*
    from $\langle a \in A\rangle$ **and** $*$ **have** $?g\ (f\ a) = h\ a$ **by** *simp*
    with $\langle b = f\ a\rangle$ **have** $?g\ b = h\ a$ **by** *blast*
    moreover from $\langle a \in A\rangle$ **and** *assms*(*2*) **have** $h\ a \in C$ **by** *auto*
    ultimately have $?g\ b \in C$ **by** *simp*
  }
  moreover {
    assume $b \notin f\ `\ A$
    hence $?g\ b = h\ a$ **by** *simp*
    also from $\langle a \in A\rangle$ **and** *assms*(*2*) **have** $h\ a \in C$ **by** *auto*
    finally have $?g\ b \in C$ **.**
  }
  ultimately show $?g\ b \in C$ **by** *blast*
qed
moreover have *ext-eq-on* $A\ h\ (?g \circ f)$
proof (*rule ext-eq-onI*)
  fix a'
  assume $a' \in A$

hence $?g\ (f\ a') = h\ a'$ **by** $(intro\ *)$
thus $h\ a' = (?g \circ f)\ a'$ **by** *simp*
qed
ultimately have *thesis* **using** *that* **by** *blast*
}
moreover {
let $?g = \lambda b.\ undefined :: 'c$
assume $A = \{\}$
with $assms(1)$ **have** $B = \{\}$ **.**
hence $?g\ `\ B \subseteq C$ **by** *simp*
moreover from $\langle A = \{\}\rangle$ **have** $ext\text{-}eq\text{-}on\ A\ h\ (?g \circ f)$ **by** *simp*
ultimately have *thesis* **by** (*fact that*)
}
ultimately show *thesis* **by** *auto*
qed

proposition *prob-1-5-14*:
assumes $A = \{\} \Longrightarrow B = \{\}$ — This assumption is not specified in the book. However, there exists a counterexample without it.
— The assumption $f\ `\ A \subseteq B$ is not necessary.
and $h\ `\ A \subseteq C$
shows $(\exists\, g.\ g\ `\ B \subseteq C \wedge ext\text{-}eq\text{-}on\ A\ h\ (g \circ f)) \longleftrightarrow (\forall\, a \in A.\ \forall\, a' \in A.\ f\ a = f\ a' \longrightarrow h\ a = h\ a')$
proof (*rule iffI*)
assume $\exists\, g.\ g\ `\ B \subseteq C \wedge ext\text{-}eq\text{-}on\ A\ h\ (g \circ f)$
then obtain g **where** $g\ `\ B \subseteq C$ **and** $*$: $ext\text{-}eq\text{-}on\ A\ h\ (g \circ f)$ **by** *auto*
{
fix a **and** a'
assume $a \in A$ **and** $a' \in A$ **and** $f\ a = f\ a'$
with $*$ **have** $h\ a = h\ a'$ **by** (*rule prob-1-5-14-a*)
}
thus $\forall\, a \in A.\ \forall\, a' \in A.\ f\ a = f\ a' \longrightarrow h\ a = h\ a'$ **by** *blast*
next
assume $*$: $\forall\, a \in A.\ \forall\, a' \in A.\ f\ a = f\ a' \longrightarrow h\ a = h\ a'$
{
fix a **and** a'
assume $a \in A$ **and** $a' \in A$ **and** $f\ a = f\ a'$
with $*$ **have** $h\ a = h\ a'$ **by** *blast*
}
with *assms* **obtain** g **where** $g\ `\ B \subseteq C$ **and** $ext\text{-}eq\text{-}on\ A\ h\ (g \circ f)$ **by** (*elim prob-1-5-14-b*)
thus $\exists\, g.\ g\ `\ B \subseteq C \wedge ext\text{-}eq\text{-}on\ A\ h\ (g \circ f)$ **by** *auto*
qed

proposition *prob-1-5-15*:
assumes $A' = \{\} \Longrightarrow A = \{\}$ — This assumption is not specified in the book. However, there exists a counterexample without it.

    and u ‘ $A' \subseteq A$
    and v ‘ $B \subseteq B'$
  defines *Phi*: $\Phi\ f \equiv v \circ f \circ u$
  obtains u ‘ $A' = A \longrightarrow$ *inj-on* $v\ B$
            $\longrightarrow (\forall f \in A \to B.\ \forall f' \in A \to B.$ *ext-eq-on* $A'\ (\Phi\ f)\ (\Phi\ f') \longrightarrow$ *ext-eq-on* $A\ f\ f')$
    and *inj-on* $u\ A' \longrightarrow v$ ‘ $B = B' \longrightarrow (\forall f' \in A' \to B'.\ \exists f \in A \to B.$ *ext-eq-on* $A'\ (\Phi\ f)\ f')$
proof −
  have u ‘ $A' = A \longrightarrow$ *inj-on* $v\ B$
        $\longrightarrow (\forall f \in A \to B.\ \forall f' \in A \to B.$ *ext-eq-on* $A'\ (\Phi\ f)\ (\Phi\ f') \longrightarrow$ *ext-eq-on* $A\ f\ f')$
  proof (*intro impI*)
    assume u ‘ $A' = A$ **and** *inj-on* $v\ B$
    {
      fix f **and** f'
      assume $f \in A \to B$ **and** $f' \in A \to B$ **and** *ext-eq-on* $A'\ (\Phi\ f)\ (\Phi\ f')$
      obtain u' **where** *right-inv-into* $A'\ u\ u'$ **by** (*fact right-inv-into*)
      hence *id-on* $(u \circ u')\ (u$ ‘ $A')$ **by** (*fact right-inv-intoD2-pf*)
      with ⟨u ‘ $A' = A$⟩ **have** *id-on* $(u \circ u')\ A$ **by** *simp*
      from ⟨*inj-on* $v\ B$⟩ **obtain** v' **where** *left-inv-into* $B\ v\ v'$ **by** (*elim left-inv-into*)
      hence $*$: *id-on* $(v' \circ v)\ B$ **by** (*fact left-inv-intoD2-pf*)
      from ⟨*id-on* $(u \circ u')\ A$⟩ **have** *ext-eq-on* $A\ (f \circ (u \circ u'))\ f$ **by** (*fact thm-1-6-2-a*)
      hence *ext-eq-on* $A\ (f \circ u \circ u')\ f$ **by** (*simp only*: *comp-assoc*)
      moreover have $(f \circ u \circ u')$ ‘ $A \subseteq B$
      proof −
        from ⟨*id-on* $(u \circ u')\ A$⟩ **have** $(u \circ u')$ ‘ $A = A$ **by** (*fact id-on-imp-surj-on*)
        with ⟨$f \in A \to B$⟩ **show** *?thesis* **by** *fastforce*
      qed
      moreover note $*$
      ultimately have *ext-eq-on* $A\ ((v' \circ v) \circ f \circ u \circ u')\ f$ **by** *fastforce*
      hence *ext-eq-on* $A\ (v' \circ (\Phi\ f) \circ u')\ f$ **unfolding** *Phi* **by** (*simp only*: *comp-assoc*)
      {
        fix a
        assume $a \in A$
        moreover from ⟨u ‘ $A' = A$⟩ **and** ⟨*right-inv-into* $A'\ u\ u'$⟩ **have** u' ‘ $A \subseteq A'$
          by (*auto dest*: *right-inv-intoD1*)
        ultimately have $u'\ a \in A'$ **by** *auto*
        with ⟨*ext-eq-on* $A'\ (\Phi\ f)\ (\Phi\ f')$⟩ **have** $(\Phi\ f')\ (u'\ a) = (\Phi\ f)\ (u'\ a)$ **by** *auto*
        hence $v'\ ((\Phi\ f')\ (u'\ a)) = v'\ ((\Phi\ f)\ (u'\ a))$ **by** *simp*
        also have $\ldots = v'\ (v\ (f\ (u\ (u'\ a))))$ **by** (*simp add*: *Phi*)
        also from ⟨u ‘ $A' = A$⟩ **and** ⟨*right-inv-into* $A'\ u\ u'$⟩ **and** ⟨$a \in A$⟩ **have** $\ldots = v'\ (v\ (f\ a))$
          by (*simp only*: *right-inv-intoD2*)
        also have $\ldots = f\ a$
        proof −
          from ⟨$f \in A \to B$⟩ **and** ⟨$a \in A$⟩ **have** $f\ a \in B$ **by** *auto*
          with ⟨*left-inv-into* $B\ v\ v'$⟩ **show** *?thesis* **by** (*fact left-inv-intoD2*)
        qed

finally have v' $((\Phi\ f')\ (u'\ a)) = f\ a$ **.**
}
hence *ext-eq-on* A $(v' \circ (\Phi\ f') \circ u')$ f **by** *auto*
hence *ext-eq-on* A $((v' \circ v) \circ f' \circ u \circ u')$ f **unfolding** *Phi* **by** (*simp only*: *comp-assoc*)
moreover have $(f' \circ u \circ u')$ ‘ $A \subseteq B$
proof −
from ⟨*id-on* $(u \circ u')$ A⟩ **have** $(u \circ u')$ ‘ $A = A$ **by** (*fact id-on-imp-surj-on*)
with ⟨$f' \in A \to B$⟩ **show** *?thesis* **by** *fastforce*
qed
moreover note ∗
ultimately have *ext-eq-on* A $(f' \circ u \circ u')$ f **by** *fastforce*
with ⟨*id-on* $(u \circ u')$ A⟩ **have** *ext-eq-on* A f' f **by** *fastforce*
hence *ext-eq-on* A f f' **by** (*fact ext-eq-on-sym*)
}
thus $\forall f \in A \to B.\ \forall f' \in A \to B.$ *ext-eq-on* A' $(\Phi\ f)$ $(\Phi\ f') \longrightarrow$ *ext-eq-on* A f f' **by** *simp*
qed
moreover have *inj-on* u $A' \longrightarrow v$ ‘ $B = B'$
$\longrightarrow (\forall f' \in A' \to B'.\ \exists f \in A \to B.$ *ext-eq-on* A' $(\Phi\ f)$ $f')$
proof (*intro impI*)
assume *inj-on* u A' **and** v ‘ $B = B'$
{
fix f'
assume $f' \in A' \to B'$
from ⟨*inj-on* u A'⟩ **and** *assms*(*1*) **obtain** u'
where u' ‘ $A \subseteq A'$ **and** *id-on* $(u' \circ u)$ A' **by** (*elim thm-1-7-b-a*)
obtain v' **where** *right-inv-into* B v v' **by** (*fact right-inv-into*)
hence v' ‘ v ‘ $B \subseteq B$ **and** *id-on* $(v \circ v')$ $(v$ ‘ $B)$
by (*fact right-inv-intoD1*, *fact right-inv-intoD2-pf*)
with ⟨v ‘ $B = B'$⟩ **have** v' ‘ $B' \subseteq B$ **and** *id-on* $(v \circ v')$ B' **by** *simp*+
let *?f* $= v' \circ f' \circ u'$
from ⟨u' ‘ $A \subseteq A'$⟩ **and** ⟨$f' \in A' \to B'$⟩ **and** ⟨v' ‘ $B' \subseteq B$⟩ **have** *?f* ‘ $A \subseteq B$ **by** *fastforce*
moreover have *ext-eq-on* A' $(\Phi$ *?f*$)$ f'
proof −
have *ext-eq-on* A' $(\Phi$ *?f*$)$ $(v \circ v' \circ f' \circ u' \circ u)$ **unfolding** *Phi* **by** *auto*
also have *ext-eq-on* A' $\ldots$ $(f' \circ u' \circ u)$
proof −
from ⟨*id-on* $(u' \circ u)$ A'⟩ **and** ⟨$f' \in A' \to B'$⟩ **have** $(f' \circ u' \circ u)$ ‘ $A' \subseteq B'$ **by** *fastforce*
with ⟨*id-on* $(v \circ v')$ B'⟩ **show** *?thesis* **by** *fastforce*
qed
also from ⟨*id-on* $(u' \circ u)$ A'⟩ **have** *ext-eq-on* A' $\ldots$ f' **by** *fastforce*
finally show *ext-eq-on* A' $(\Phi$ *?f*$)$ f' **.**
qed
ultimately have $\exists f \in A \to B.$ *ext-eq-on* A' $(\Phi\ f)$ f' **by** *blast*
}
thus $\forall f' \in A' \to B'.\ \exists f \in A \to B.$ *ext-eq-on* A' $(\Phi\ f)$ f' **..**

```
    qed
    ultimately show thesis by (fact that)
  qed

  proposition prob-1-5-15-ext-a:
    assumes u ` A' = A
      and inj-on v B
    defines Φ f ≡ λa'. if a' ∈ A' then v (f (u a')) else undefined
    shows inj-on Φ (A →E B)
  proof (rule inj-onI)
    fix f f'
    assume f: f ∈ A →E B
      and f': f' ∈ A →E B
      and Φ f = Φ f'
    {
      fix a
      consider a ∈ A | a ∉ A by auto
      moreover {
        assume a: a ∈ A
        with assms(1) obtain a' where a' ∈ A' and u a' = a by auto
        with ‹Φ f = Φ f'› and Φ-def have v (f a) = v (f' a) by metis
        moreover from f and a have f a ∈ B by auto
        moreover from f' and a have f' a ∈ B by auto
        moreover note assms(2)
        ultimately have f a = f' a by (simp only: inj-onD)
      }
      moreover {
        assume a ∉ A
        with f and f' have f a = f' a by fastforce
      }
      ultimately have f a = f' a by auto
    }
    thus f = f' by auto
  qed

  proposition prob-1-5-15-ext-b:
    fixes A :: 'a set
      and B :: 'b set
    assumes A' = {} ⟹ A = {}
      and u ` A' ⊆ A
      and inj-on u A'
      and v ` B = B'
    defines Φ f ≡ λa'. if a' ∈ A' then v (f (u a')) else undefined
    shows Φ ` (A →E B) = (A' →E B')
  proof (rule surj-onI)
```

fix f
assume f: $f \in A \rightarrow_E B$
{
fix a'
assume a': $a' \in A'$
with Φ-*def* **have** $\Phi\ f\ a' = v\ (f\ (u\ a'))$ **by** *simp*
also from a' **and** *assms*(*2*) **and** f **and** *assms*(*4*) **have** $\ldots \in B'$ **by** *auto*
finally have $\Phi\ f\ a' \in B'$ **.**
}
moreover {
fix a'
assume $a' \notin A'$
with Φ-*def* **have** $\Phi\ f\ a' = \textit{undefined}$ **by** *simp*
}
ultimately show $\Phi\ f \in A' \rightarrow_E B'$ **by** *auto*
next
fix g
assume g: $g \in A' \rightarrow_E B'$
consider $A' = \{\} \mid A' \neq \{\}$ **by** *auto*
moreover {
assume $A' = \{\}$
with *assms*(*1*) **have** $A = \{\}$ **by** *simp*
moreover define f :: $'a \Rightarrow 'b$ **where** $f\ a \equiv \textit{undefined}$ **for** a
ultimately have $f \in A \rightarrow_E B$ **by** *auto*
moreover have $\Phi\ f = g$
proof (*rule ext*)
fix a'
from Φ-*def* **and** $\langle A' = \{\} \rangle$ **and** g **show** $\Phi\ f\ a' = g\ a'$ **by** *simp*
qed
ultimately have $\exists f \in A \rightarrow_E B.\ \Phi\ f = g$ **by** *auto*
}
moreover {
assume $A' \neq \{\}$
with g **and** *assms*(*4*) **have** $B \neq \{\}$ **by** *auto*
then obtain b **where** b: $b \in B$ **by** *auto*
from *assms*(*3*) **obtain** u' **where** *left-inv-into* $A'\ u\ u'$ **by** (*rule left-inv-into*)
hence $u'1$: u' ` $u$ ` $A' \subseteq A'$ **and** $u'2$: *id-on* $(u' \circ u)\ A'$
by (*fact left-inv-intoD1*, *fact left-inv-intoD2-pf*)
obtain v' **where** *right-inv-into* $B\ v\ v'$ **by** (*fact right-inv-into*)
hence v' ` $v$ ` $B \subseteq B$ **and** *id-on* $(v \circ v')\ (v$ ` $B)$
by (*fact right-inv-intoD1*, *fact right-inv-intoD2-pf*)
with *assms*(*4*) **have** $v'1$: v' ` $B' \subseteq B$ **and** $v'2$: *id-on* $(v \circ v')\ B'$ **by** *simp+*
define f **where** $f\ a \equiv$ *if* $a \in u$ ` A' *then* $v'\ (g\ (u'\ a))$ *else* (*if* $a \in A$ *then* b *else undefined*) **for** a
{
fix a

```
      assume a ∈ A
      consider a ∈ u ‘ A' | a ∉ u ‘ A' by auto
      moreover {
        assume a ∈ u ‘ A'
        with f-def have f a = v' (g (u' a)) by simp
        also from ‹a ∈ u ‘ A'› and u'1 and g and v'1 have ... ∈ B by auto
        finally have f a ∈ B by simp
      }
      moreover {
        assume a ∉ u ‘ A'
        with ‹a ∈ A› and f-def and b have f a ∈ B by simp
      }
      ultimately have f a ∈ B by auto
    }
    moreover {
      fix a
      assume a ∉ A
      moreover from this and assms(2) have a ∉ u ‘ A' by auto
      moreover note f-def
      ultimately have f a = undefined by simp
    }
    ultimately have f ∈ A →E B by auto
    moreover have Φ f = g
    proof (rule ext)
      fix a'
      consider a' ∈ A' | a' ∉ A' by auto
      moreover {
        assume a': a' ∈ A'
        with Φ-def have Φ f a' = v (f (u a')) by simp
        also from a' and f-def have ... = (v ∘ v') (g ((u' ∘ u) a')) by auto
        also from a' and u'2 and g and v'2 have ... = g a' by fastforce
        finally have Φ f a' = g a' .
      }
      moreover {
        assume a': a' ∉ A'
        with Φ-def and g have Φ f a' = g a' by auto
      }
      ultimately show Φ f a' = g a' by auto
    qed
    ultimately have ∃f ∈ A →E B. Φ f = g by auto
  }
  ultimately show g ∈ Φ ‘ (A →E B) by auto
qed

end
```

theory *Section-1-6*
  imports *Main*
    HOL−Library.Disjoint-Sets
    Split-Pair
    Section-1-5
begin

1.6 Equivalence Relation

1.6.1 A) Notion of Relation

1.6.2 B) Equivalence Relation

lemma *refl-onE* [*elim*]:
  assumes *refl-on* A r
  obtains $a \in A \Longrightarrow (a,\ a) \in r$
    and $(x,\ y) \in r \Longrightarrow x \in A$
    and $(x,\ y) \in r \Longrightarrow y \in A$
  using *assms* **by** (*blast dest*: *refl-onD refl-on-domain*)

lemmas [*elim*] = *symE transE equivE*

definition *equiv-kernel-on* :: $('a \Rightarrow 'b) \Rightarrow 'a\ set \Rightarrow ('a \times 'a)\ set$
  where *equiv-kernel-on* f $A = \{(a,\ a') \in A \times A.\ f\ a = f\ a'\}$

lemma *equiv-kernel-onI* [*intro*]:
  assumes $a \in A$
    and $a' \in A$
    and $f\ a = f\ a'$
  shows $(a,\ a') \in$ *equiv-kernel-on* f A
  using *assms* **unfolding** *equiv-kernel-on-def* **by** *simp*

lemma *equiv-kernel-onE* [*elim*]:
  assumes $(a,\ a') \in$ *equiv-kernel-on* f A
  obtains $a \in A$
    and $a' \in A$
    and $f\ a = f\ a'$
  using *assms* **unfolding** *equiv-kernel-on-def* **by** *simp*

lemma *equiv-equiv-kernel-on*:
  shows *equiv* A (*equiv-kernel-on* f A)
proof (*rule equivI*)
  show *refl-on* A (*equiv-kernel-on* f A)
  proof (*rule refl-onI*)
    show *equiv-kernel-on* f $A \subseteq A \times A$ **by** *auto*

fix a
assume $a \in A$
moreover have $f\ a = f\ a$ **by** *simp*
ultimately show $(a,\ a) \in$ *equiv-kernel-on* $f\ A$ **by** *auto*
qed
show *sym* (*equiv-kernel-on* $f\ A$)
proof (*rule symI*)
fix a **and** a'
assume $(a,\ a') \in$ *equiv-kernel-on* $f\ A$
thus $(a',\ a) \in$ *equiv-kernel-on* $f\ A$ **by** *fastforce*
qed
show *trans* (*equiv-kernel-on* $f\ A$)
proof (*rule transI*)
fix a **and** a' **and** a''
assume $(a,\ a') \in$ *equiv-kernel-on* $f\ A$ **and** $(a',\ a'') \in$ *equiv-kernel-on* $f\ A$
thus $(a,\ a'') \in$ *equiv-kernel-on* $f\ A$ **by** *fastforce*
qed
qed

lemma *partition-onI2*:
assumes $\{\} \notin \mathfrak{M}$
and $\bigcup \mathfrak{M} = A$
and $\bigwedge M\ M'.\ M \in \mathfrak{M} \Longrightarrow M' \in \mathfrak{M} \Longrightarrow M \neq M' \Longrightarrow M \cap M' = \{\}$
shows *partition-on* $A\ \mathfrak{M}$
using *assms* **by** (*blast intro*: *partition-onI disjnt-def*[*THEN iffD2*])

lemma *partition-onE* [*elim*]:
assumes *partition-on* $A\ \mathfrak{M}$
obtains $\{\} \notin \mathfrak{M}$
and $\bigcup \mathfrak{M} = A$
and $M \in \mathfrak{M} \Longrightarrow M' \in \mathfrak{M} \Longrightarrow M \neq M' \Longrightarrow M \cap M' = \{\}$
using *assms* **by** (*auto dest*: *partition-onD1 partition-onD2 partition-onD3 disjointD*)

lemma *partition-on-definition*:
shows *partition-on* $A\ \mathfrak{M}$
$\longleftrightarrow (\forall M \in \mathfrak{M}.\ M \neq \{\}) \wedge \bigcup \mathfrak{M} = A \wedge (\forall M \in \mathfrak{M}.\ \forall M' \in \mathfrak{M}.\ M \neq M' \longrightarrow M \cap M' =$
$\{\})$
(**is** $?L \longleftrightarrow ?R0 \wedge ?R1 \wedge ?R2$)
proof (*rule iffI*)
assume $?L$
thus $?R0 \wedge ?R1 \wedge ?R2$ **by** *fast*
next
assume $*$: $?R0 \wedge ?R1 \wedge ?R2$
from $*$ **have** $?R1$ **by** *simp*
moreover {

fix M **and** M'
assume $M \in \mathfrak{M}$ **and** $M' \in \mathfrak{M}$ **and** $M \neq M'$
with $*$ **have** *disjnt M M'* **unfolding** *disjnt-def* **by** *simp*
}
moreover from $*$ **have** $\{\} \notin \mathfrak{M}$ **by** *auto*
ultimately show *partition-on* A $\mathfrak{M}$ **by** (*fact partition-onI*)
qed

definition *equiv-by-partition* :: $'a\ set\ set \Rightarrow {'a}\ rel$
where *equiv-by-partition* $\mathfrak{M} = \{(a,\ a').\ \exists\, C \in \mathfrak{M}.\ a \in C \wedge a' \in C\}$

lemma *equiv-by-partition-iff* [*iff*]:
shows $(a,\ a') \in$ *equiv-by-partition* $\mathfrak{M} \longleftrightarrow (\exists\, C \in \mathfrak{M}.\ a \in C \wedge a' \in C)$
unfolding *equiv-by-partition-def* **by** *simp*

1.6.3 C) Equivalence Classes, Quotient Set

proposition *prop-1-6-1*:
assumes *equiv* A R
and $a \in A$
shows $a \in R$ `` $\{a\}$
using *assms* **by** (*fact equiv-class-self*)

proposition *prop-1-6-2-a*:
assumes *equiv* A R
and $(a,\ b) \in R$
shows R `` $\{a\} = R$ `` $\{b\}$
using *assms* **by** (*fact equiv-class-eq*)

proposition *prop-1-6-2-b*:
assumes *equiv* A R
and $a \in A$
and R `` $\{a\} = R$ `` $\{b\}$
shows $(a,\ b) \in R$
proof −
from *assms*(*1*,*2*) **have** $a \in R$ `` $\{a\}$ **by** (*rule prop-1-6-1*)
with *assms*(*3*) **have** $a \in R$ `` $\{b\}$ **by** *simp*
with *assms*(*1*) **show** $(a,\ b) \in R$ **by** *auto*
qed

proposition *prop-1-6-2*:
assumes *equiv* A R
and $a \in A$
shows $(a,\ b) \in R \longleftrightarrow R$ `` $\{a\} = R$ `` $\{b\}$
using *assms prop-1-6-2-a prop-1-6-2-b* **by** *metis*

proposition *prop-1-6-3*:
assumes *equiv* A R
and R `` $\{a\} \neq R$ `` $\{b\}$
shows R `` $\{a\} \cap R$ `` $\{b\} = \{\}$
proof (*rule ccontr*)
assume R `` $\{a\} \cap R$ `` $\{b\} \neq \{\}$
then obtain c **where** $c \in R$ `` $\{a\}$ **and** $c \in R$ `` $\{b\}$ **by** *auto*
with *assms*(*1*) **have** $(a, b) \in R$ **by** *auto*
with *assms*(*1*) **have** R `` $\{a\} = R$ `` $\{b\}$ **by** (*rule prop-1-6-2-a*)
with *assms*(*2*) **show** *False* **by** *simp*
qed

theorem *thm-1-8-a*:
assumes *equiv* A R
shows *partition-on* A $(A\ //\ R)$
using *assms* **by** (*fact partition-on-quotient*)

theorem *thm-1-8-b*:
assumes *equiv* A R
shows *equiv-by-partition* $(A\ //\ R) = R$
proof (*rule set-eqI*, *split-pair*)
fix a **and** a'
have $(a, a') \in$ *equiv-by-partition* $(A\ //\ R) \longleftrightarrow (\exists\, C \in A\ //\ R.\ a \in C \wedge a' \in C)$ **by** *simp*
also have $\ldots \longleftrightarrow (\exists\, C \in (\bigcup a \in A.\ \{R$ `` $\{a\}\}).\ a \in C \wedge a' \in C)$
unfolding *quotient-def* **by** *blast*
also have $\ldots \longleftrightarrow (\exists\, C.\ \exists\, a'' \in A.\ C = R$ `` $\{a''\} \wedge a \in C \wedge a' \in C)$ **by** *auto*
also have $\ldots \longleftrightarrow (\exists\, a'' \in A.\ a \in R$ `` $\{a''\} \wedge a' \in R$ `` $\{a''\})$ **by** *blast*
also have $\ldots \longleftrightarrow (\exists\, a'' \in A.\ (a'', a) \in R \wedge (a'', a') \in R)$ **by** *simp*
also from *assms* **have** $\ldots \longleftrightarrow (\exists\, a'' \in A.\ (a, a'') \in R \wedge (a'', a') \in R)$ **by** *auto*
also from *assms* **have** $\ldots \longleftrightarrow (a, a') \in R$ **by** *auto*
finally show $(a, a') \in$ *equiv-by-partition* $(A\ //\ R) \longleftrightarrow (a, a') \in R$ **.**
qed

lemmas [*intro*] = *quotientI*

lemmas [*elim*] = *quotientE*

proposition *ex-1-1-a*:
shows *equiv* A $\{(a, a') \in A \times A.\ a = a'\}$
proof (*rule equivI*)
let *?R* $= \{(a, a') \in A \times A.\ a = a'\}$
show *refl-on* A *?R*
proof (*rule refl-onI*)
show *?R* $\subseteq A \times A$ **by** *auto*

fix a
assume $a \in A$
hence $(a, a) \in A \times A$ **by** *simp*
moreover have $a = a$ **by** *simp*
ultimately show $(a, a) \in ?R$ **by** *simp*
qed
show *sym ?R*
proof (*rule symI*)
fix a **and** a'
assume $(a, a') \in \{(a, a') \in A \times A.\ a = a'\}$
hence $(a', a) \in A \times A$ **and** $a' = a$ **by** *simp+*
thus $(a', a) \in \{(a, a') \in A \times A.\ a = a'\}$ **by** *simp*
qed
show *trans ?R*
proof (*rule transI*)
fix a **and** a' **and** a''
assume $(a, a') \in ?R$ **and** $(a', a'') \in ?R$
hence $(a, a'') \in A \times A$ **and** $a = a''$ **by** *simp+*
thus $(a, a'') \in ?R$ **by** *simp*
qed
qed

proposition *ex-1-1-b*:
shows *equiv* A $\{(a, a') \in A \times A.\ True\}$
proof (*rule equivI*)
let $?R = \{(a, a') \in A \times A.\ True\}$
show *refl-on A ?R*
proof (*rule refl-onI*)
show $?R \subseteq A \times A$ **by** *simp*
fix a
assume $a \in A$
thus $(a, a) \in ?R$ **by** *simp*
qed
show *sym ?R*
proof (*rule symI*)
fix a **and** a'
assume $(a, a') \in ?R$
hence $(a, a') \in A \times A$ **by** *simp*
thus $(a', a) \in ?R$ **by** *simp*
qed
show *trans ?R*
proof (*rule transI*)
fix a **and** a' **and** a''
assume $(a, a') \in ?R$ **and** $(a', a'') \in ?R$
hence $(a, a'') \in A \times A$ **by** *simp*

thus $(a,\ a'') \in\ ?R$ **by** *simp*
qed
qed

1.6.4 D) Decomposition of Map

proposition *prop-1-6-4*:
assumes f ‘ $A \subseteq B$
obtains φ **and** g **and** j **where**
φ ‘ $A = A$ // (*equiv-kernel-on* f A)
and *bij-betw* g (A // (*equiv-kernel-on* f A)) (f ‘ A)
and *id-on* j (f ‘ A)
and *ext-eq-on* A ($j \circ g \circ \varphi$) f
proof $-$
have $*$: *equiv* A (*equiv-kernel-on* f A) **by** (*fact equiv-equiv-kernel-on*)
let $?\varphi = \lambda a.$ *equiv-kernel-on* f A ‘‘ $\{a\}$
let $?g = \lambda C.$ (*THE* $b.\ \exists\, a \in A.\ ?\varphi\ a = C \land f\ a = b$)
let $?j = \lambda b.$ *if* $b \in f$ ‘ A *then* b *else undefined*
have $**$: $?g\ (?\varphi\ a) = f\ a$ **if** $a \in A$ **for** a
proof (*rule the-equality*)
from *that* **show** $\exists\, a' \in A.\ ?\varphi\ a' =$ *equiv-kernel-on* f A ‘‘ $\{a\} \land f\ a' = f\ a$ **by** *blast*
fix b
assume $\exists\, a' \in A.\ ?\varphi\ a' =$ *equiv-kernel-on* f A ‘‘ $\{a\} \land f\ a' = b$
then obtain a' **where** $a' \in A$ **and** $?\varphi\ a' =$ *equiv-kernel-on* f A ‘‘ $\{a\}$ **and** $f\ a' = b$ **by** *auto*
from $*$ **and** *this*(*1*,*2*) **have** $(a',\ a) \in$ *equiv-kernel-on* f A **by** (*rule prop-1-6-2-b*)
hence $f\ a' = f\ a$ **by** *auto*
with $\langle f\ a' = b\rangle$ **show** $b = f\ a$ **by** *simp*
qed
have $?\varphi$ ‘ $A = A$ // (*equiv-kernel-on* f A)
proof (*rule surj-onI*)
fix a
assume $a \in A$
thus $?\varphi\ a \in A$ // (*equiv-kernel-on* f A) **by** *auto*
next
fix C
assume $C \in A$ // (*equiv-kernel-on* f A)
then obtain a **where** $a \in A$ **and** $C =$ *equiv-kernel-on* f A ‘‘ $\{a\}$ **by** *fast*
thus $C \in\ ?\varphi$ ‘ A **by** *simp*
qed
moreover have *bij-betw* $?g$ (A // (*equiv-kernel-on* f A)) (f ‘ A)
proof (*rule bij-betw-imageI*)
show $?g$ ‘ (A // (*equiv-kernel-on* f A)) $= f$ ‘ A
proof (*rule surj-onI*)
fix C
assume $C \in A$ // (*equiv-kernel-on* f A)

then obtain a **where** $a \in A$ **and** $C = ?\varphi\ a$ **by** *fast*
from *this(1)* **have** $?g\ (?\varphi\ a) = f\ a$ **by** (*fact* $**$)
with $\langle C = ?\varphi\ a\rangle$ **have** $?g\ C = f\ a$ **by** *simp*
with $\langle a \in A\rangle$ **show** $?g\ C \in f\ `\ A$ **by** *simp*
next
fix b
assume $b \in f\ `\ A$
then obtain a **where** $a \in A$ **and** $b = f\ a$ **by** *auto*
from *this(1)* **have** $?g\ (?\varphi\ a) = f\ a$ **by** (*fact* $**$)
with $\langle b = f\ a\rangle$ **have** $?g\ (?\varphi\ a) = b$ **by** *simp*
moreover from $\langle a \in A\rangle$ **have** $?\varphi\ a \in A\ //\ \textit{equiv-kernel-on}\ f\ A$ **by** *auto*
ultimately show $b \in ?g\ `\ (A\ //\ (\textit{equiv-kernel-on}\ f\ A))$ **by** *auto*
qed
show $\textit{inj-on}\ ?g\ (A\ //\ (\textit{equiv-kernel-on}\ f\ A))$
proof (*rule inj-onI*)
fix C **and** C'
assume $C \in A\ //\ (\textit{equiv-kernel-on}\ f\ A)$
and $C' \in A\ //\ (\textit{equiv-kernel-on}\ f\ A)$
and $?g\ C = ?g\ C'$
from *this(1,2)* **obtain** a **and** a' **where** $a \in A$ **and** $C = ?\varphi\ a\ a' \in A$ **and** $C' = ?\varphi\ a'$
by *fastforce*
from *this* **have** $?g\ (?\varphi\ a) = f\ a$ **and** $?g\ (?\varphi\ a') = f\ a'$ **by** (*auto intro*: $**$)
with $\langle ?g\ C = ?g\ C'\rangle$ **and** $\langle C = ?\varphi\ a\rangle$ **and** $\langle C' = ?\varphi\ a'\rangle$ **have** $f\ a = f\ a'$ **by** *force*
with $\langle a \in A\rangle$ **and** $\langle a' \in A\rangle$ **have** $?\varphi\ a = ?\varphi\ a'$ **by** *fastforce*
with $\langle C = ?\varphi\ a\rangle$ **and** $\langle C' = ?\varphi\ a'\rangle$ **show** $C = C'$ **by** *simp*
qed
qed
moreover have $\textit{id-on}\ ?j\ (f\ `\ A)$
proof (*rule id-onI*)
fix b
assume $b \in f\ `\ A$
thus $?j\ b = b$ **by** *simp*
qed
moreover have $\textit{ext-eq-on}\ A\ (?j \circ ?g \circ ?\varphi)\ f$
proof (*rule ext-eq-onI*)
fix a
assume $a \in A$
hence $?g\ (?\varphi\ a) = f\ a$ **by** (*fact* $**$)
with $\langle a \in A\rangle$ **have** $?g\ (?\varphi\ a) \in f\ `\ A$ **by** *simp*
hence $?j\ (?g\ (?\varphi\ a)) = ?g\ (?\varphi\ a)$ **by** *simp*
with $\langle ?g\ (?\varphi\ a) = f\ a\rangle$ **show** $(?j \circ ?g \circ ?\varphi)\ a = f\ a$ **by** *force*
qed
ultimately show *thesis* **by** (*fact that*)
qed

1.6.5 Problems

```
proposition prob-1-6-1-a:
  assumes A − B ≠ {}
    and B − A ≠ {}
    and A ∩ B ≠ {}
  defines R: R ≡ A × A ∪ B × B
  shows refl-on (A ∪ B) R
    and sym R
    and ¬trans R
proof −
  have R ⊆ (A ∪ B) × (A ∪ B) unfolding R by auto
  moreover {
    fix a
    assume a ∈ A ∪ B
    hence (a, a) ∈ R unfolding R by simp
  }
  ultimately show refl-on (A ∪ B) R by (rule refl-onI)
  {
    fix a and a′
    assume (a, a′) ∈ R
    hence (a′, a) ∈ R unfolding R by auto
  }
  thus sym R by (rule symI)
  {
    assume trans R
    from assms obtain a and b and c where a ∈ A − B and b ∈ B − A and c ∈ A ∩ B by auto
    hence (a, c) ∈ R and (c, b) ∈ R and (a, b) ∉ R unfolding R by auto
    from this(1,2) and ‹trans R› have (a, b) ∈ R by auto
    with ‹(a, b) ∉ R› have False ..
  }
  thus ¬trans R ..
qed

proposition prob-1-6-1-b:
  defines R: R :: nat rel ≡ {(a, b). a ≤ b}
  shows refl-on UNIV R
    and ¬sym R
    and trans R
proof −
  have R ⊆ UNIV × UNIV unfolding R by simp
  moreover {
    fix a
    have (a, a) ∈ R unfolding R by simp
```

```
  }
  ultimately show refl-on UNIV R by (rule refl-onI)
  {
    assume sym R
    moreover have (0, 1) ∈ R unfolding R by simp
    ultimately have (1, 0) ∈ R by auto
    moreover have (1, 0) ∉ R unfolding R by simp
    ultimately have False by simp
  }
  thus ¬sym R by auto
  {
    fix a and a′ and a′′
    assume (a, a′) ∈ R and (a′, a′′) ∈ R
    hence (a, a′′) ∈ R unfolding R by fastforce
  }
  thus trans R by (rule transI)
qed

proposition prob-1-6-2:
  assumes R ⊆ A × A
    and sym R
    and trans R
    and ∀ a ∈ A. ∃ x. (a, x) ∈ R
  shows equiv A R
proof (rule equivI[OF - assms(2) assms(3)])
  {
    fix a
    assume a ∈ A
    with assms(4) obtain a′ where (a, a′) ∈ R by auto
    moreover from this and assms(2) have (a′, a) ∈ R by auto
    moreover note assms(3)
    ultimately have (a, a) ∈ R by auto
  }
  with assms(1) show refl-on A R by (rule refl-onI)
qed

proposition prob-1-6-3:
  assumes — The assumption R ⊆ A × A is not necessary since it is implied by refl-on A R.
    refl-on A R
    and ∀ a b c. (a, b) ∈ R ∧ (b, c) ∈ R ⟶ (c, a) ∈ R
  shows equiv A R
proof (rule equivI[OF assms(1)])
  {
    fix a and a′
    assume (a, a′) ∈ R
```

moreover from *this* **and** *assms*(*1*) **have** $a \in A$ **and** $a' \in A$ **by** *auto*
moreover from *this*(*2*) **and** *assms*(*1*) **have** $(a', a') \in R$ **by** *auto*
moreover note *assms*(*2*)
ultimately have $(a', a) \in R$ **by** *blast*
}
thus *sym R* **by** (*rule symI*)
{
fix a **and** a' **and** a''
assume $(a, a') \in R$ **and** $(a', a'') \in R$
with *assms*(*2*) **have** $(a'', a) \in R$ **by** *blast*
with ‹*sym R*› **have** $(a, a'') \in R$ **by** *auto*
}
thus *trans R* **by** (*rule transI*)
qed

proposition *prob-1-6-4*:
defines A: $A :: int\ rel \equiv UNIV \times (UNIV - \{0\})$
defines R: $R \equiv \{((m, n), (m', n')) \in A \times A.\ m * n' = m' * n\}$
shows *equiv A R*
proof (*rule equivI*)
have $R \subseteq A \times A$ **unfolding** R **by** *auto*
moreover {
fix m **and** n
assume $(m, n) \in A$
hence $((m, n), (m, n)) \in R$ **unfolding** R **by** *simp*
}
ultimately show *refl-on A R* **by** (*intro refl-onI*; *split-pair*)
{
fix m **and** n **and** m' **and** n'
assume $((m, n), (m', n')) \in R$
hence $(m, n) \in A$ **and** $(m', n') \in A$ **and** $m * n' = m' * n$ **unfolding** R **by** *auto*
from *this*(*3*) **have** $m' * n = m * n'$ **by** *simp*
with ‹$(m, n) \in A$› **and** ‹$(m', n') \in A$› **have** $((m', n'), (m, n)) \in R$ **unfolding** R **by** *fast*
}
thus *sym R* **by** (*intro symI*, *split-pair*)
{
fix m **and** n **and** m' **and** n' **and** m'' **and** n''
assume $((m, n), (m', n')) \in R$ **and** $((m', n'), (m'', n'')) \in R$
hence $(m, n) \in A$
and $(m', n') \in A$
and $(m'', n'') \in A$
and $m * n' = m' * n$
and $m' * n'' = m'' * n'$ **unfolding** R **by** *auto*
from *this*(*4*,*5*) **have** $m * n' * n'' = m' * n * n''$
and $m' * n * n'' = m'' * n * n'$ **by** *simp+*

```
    hence m * n'' * n' = m'' * n * n' by algebra
    moreover from ‹(m', n') ∈ A› have n' ≠ 0 unfolding A by simp
    ultimately have m * n'' = m'' * n by simp
    with ‹(m, n) ∈ A› and ‹(m'', n'') ∈ A› have ((m, n), (m'', n'')) ∈ R unfolding R by simp
  }
  thus trans R by (intro transI, split-pair)
qed

proposition prob-1-6-5:
  shows equiv-kernel-on fst (A × B) = {((a, b), (a', b')) ∈ (A × B) × (A × B). a = a'}
    (is ?L = ?R)
proof (rule set-eqI; split-pair)
  fix a and b and a' and b'
  have ((a, b), (a', b')) ∈ ?L
        ⟷ (a, b) ∈ A × B ∧ (a', b') ∈ A × B ∧ fst (a, b) = fst (a', b') by blast
  also have ... ⟷ (a, b) ∈ A × B ∧ (a', b') ∈ A × B ∧ a = a' by simp
  also have ... ⟷ ((a, b), (a', b')) ∈ ?R by simp
  finally show ((a, b), (a', b')) ∈ ?L ⟷ ((a, b), (a', b')) ∈ ?R .
qed

proposition prob-1-6-6-a:
  assumes equiv A R
  defines phi: φ a ≡ R `` {a}
  assumes — The assumption f ` A ⊆ B is not necessary.
    — The assumption g ` A // R ⊆ B is not necessary.
    ext-eq-on A f (g ∘ φ)
  shows ∀ a ∈ A. ∀ a' ∈ A. (a, a') ∈ R ⟶ f a = f a'
proof (intro ballI, rule impI)
  fix a and a'
  assume (a, a') ∈ R
  with assms(1) have R `` {a} = R `` {a'} by (rule prop-1-6-2-a)
  hence φ a = φ a' by (unfold phi)
  moreover from assms(1) and ‹(a, a') ∈ R› have a ∈ A and a' ∈ A by auto
  moreover note assms(3)
  ultimately show f a = f a' by (elim prob-1-5-14-a)
qed

proposition prob-1-6-6-b:
  assumes equiv A R
  defines phi: φ ≡ λa. R `` {a}
  assumes f ` A ⊆ B
    and ∀ a ∈ A. ∀ a' ∈ A. (a, a') ∈ R ⟶ f a = f a'
  obtains g where g ` (A // R) ⊆ B and ext-eq-on A f (g ∘ φ)
proof –
  {
```

fix a **and** a'
assume $a \in A$ **and** $a' \in A$ **and** $\varphi\ a = \varphi\ a'$
from *this*(*3*) **have** R ‘‘ $\{a\} = R$ ‘‘ $\{a'\}$ **unfolding** *phi* **by** *simp*
with *assms*(*1*) **and** ‹$a \in A$› **have** $(a,\ a') \in R$ **by** (*rule prop-1-6-2-b*)
with ‹$a \in A$› **and** ‹$a' \in A$› **and** *assms*(*4*) **have** $f\ a = f\ a'$ **by** *simp*
}
moreover from *assms*(*1*) **have** $A = \{\} \Longrightarrow A\ //\ R = \{\}$ **by** *simp*
moreover note *assms*(*3*)
ultimately obtain g **where** g ‘ $(A\ //\ R) \subseteq B$ **and** *ext-eq-on* $A\ f\ (g \circ \varphi)$ **by** (*elim prob-1-5-14-b*)
thus *thesis* **by** (*fact that*)
qed

proposition *prob-1-6-6*:
assumes *equiv* $A\ R$
defines *phi*: $\varphi \equiv \lambda a.\ R$ ‘‘ $\{a\}$
assumes f ‘ $A \subseteq B$
shows $(\exists\, g.\ g$ ‘ $(A\ //\ R) \subseteq B \wedge$ *ext-eq-on* $A\ f\ (g \circ \varphi))$
$\longleftrightarrow (\forall\, a \in A.\ \forall\, a' \in A.\ (a,\ a') \in R \longrightarrow f\ a = f\ a')$
proof (*rule iffI*)
assume $\exists\, g.\ g$ ‘ $(A\ //\ R) \subseteq B \wedge$ *ext-eq-on* $A\ f\ (g \circ \varphi)$
then obtain g **where** *ext-eq-on* $A\ f\ (g \circ \varphi)$ **by** *auto*
with *assms*(*1*) **and** *phi* **show** $\forall\, a \in A.\ \forall\, a' \in A.\ (a,\ a') \in R \longrightarrow f\ a = f\ a'$
by (*auto dest*: *prob-1-6-6-a*)
next
assume $\forall\, a \in A.\ \forall\, a' \in A.\ (a,\ a') \in R \longrightarrow f\ a = f\ a'$
with *assms*(*1*,*3*) **obtain** g **where** g ‘ $(A\ //\ R) \subseteq B$ **and** *ext-eq-on* $A\ f\ (g \circ \varphi)$
unfolding *phi* **by** (*elim prob-1-6-6-b*)
thus $\exists\, g.\ g$ ‘ $(A\ //\ R) \subseteq B \wedge$ *ext-eq-on* $A\ f\ (g \circ \varphi)$ **by** *auto*
qed

end
theory *Section-2-1*
imports *Complex-Main*
HOL−Library.Disjoint-Sets
HOL−Library.Quadratic-Discriminant
HOL−Computational-Algebra.Primes
Split-Pair
Section-1-6
begin

context includes *cardinal-syntax* **begin**

第 2 章

Cardinality of Sets

2.1 Equipotence and Cardinality of Sets

2.1.1 A) Equipotence of Sets

definition *equipotent* :: $'a\ set \Rightarrow 'b\ set \Rightarrow bool$
  where *equipotent* $A\ B \longleftrightarrow (\exists f.\ bij\text{-}betw\ f\ A\ B)$

lemma *equipotentI* [*intro*]:
  assumes *bij-betw* $f\ A\ B$
  shows *equipotent* $A\ B$
  using *assms* **unfolding** *equipotent-def* **by** *auto*

lemma *equipotentE* [*elim*]:
  assumes *equipotent* $A\ B$
  obtains f **where** *bij-betw* $f\ A\ B$
  using *assms* **unfolding** *equipotent-def* **by** *auto*

lemma *equipotent-empty1* [*simp*]:
  assumes *equipotent* $\{\}\ B$
  shows $B = \{\}$
  using *assms* **by** *auto*

lemma *equipotent-empty2* [*simp*]:
  assumes *equipotent* $A\ \{\}$
  shows $A = \{\}$
  using *assms* **by** *auto*

proposition *prop-2-1-1* [*simp*]:
  shows *equipotent* $A\ A$
  by *auto*

proposition *prop-2-1-2*:

assumes *equipotent A B*
shows *equipotent B A*
proof –
from *assms* **obtain** *f* **where** *bij-betw f A B* **by** *auto*
hence *bij-betw* (*the-inv-into A f*) *B A* **by** (*fact bij-betw-the-inv-into*)
thus *?thesis* **by** *auto*
qed

proposition *prop-2-1-3* [*trans*]:
assumes *equipotent A B*
and *equipotent B C*
shows *equipotent A C*
proof –
from *assms* **obtain** *f* **and** *g* **where** *bij-betw f A B* **and** *bij-betw g B C* **by** *blast*
hence *bij-betw* ($g \circ f$) *A C* **by** (*rule thm-1-5-c*)
thus *?thesis* **by** *auto*
qed

proposition *ex-2-3-factorization-existence*:
assumes $0 < n$
obtains i :: *nat* **and** j :: *nat* **where** $n = 2\hat{}i * (2 * j + 1)$
proof –
have *prime* (2 :: *nat*) **by** (*fact two-is-prime-nat*)
with *assms* **obtain** i **and** m
where $\neg 2$ *dvd* m
and $n = m * 2\hat{}i$ **by** (*blast dest*: *prime-power-canonical*)
from *this*(*1*) **obtain** j **where** $m = 2 * j + 1$ **by** (*elim oddE*)
with ‹$n = m * 2\hat{}i$› **have** $n = 2\hat{}i * (2 * j + 1)$ **by** *simp*
thus *thesis* **by** (*fact that*)
qed

proposition *ex-2-3-factorization-uniqueness*:
fixes $i\ j\ i'\ j'$ :: *nat*
assumes $2\hat{}i * (2 * j + 1) = 2\hat{}i' * (2 * j' + 1)$
obtains $i = i'$ **and** $j = j'$
proof –
from *assms* **have** $(2 * j' + 1) * 2\hat{}i' = (2 * j + 1) * 2\hat{}i$ **by** (*simp only*: *mult.commute*)
moreover have *prime* (2 :: *nat*) **by** *simp*
moreover have $\neg 2$ *dvd* $(2 * j' + 1)$ **by** *simp*
moreover have $\neg 2$ *dvd* $(2 * j + 1)$ **by** *simp*
ultimately have $2 * j' + 1 = 2 * j + 1$ **and** $i' = i$ **using** *prime-power-cancel2* **by** *blast+*
hence $j = j'$ **and** $i = i'$ **by** *simp+*
thus *thesis* **by** (*intro that*)
qed

```
proposition ex-2-3:
  shows equipotent ((UNIV :: nat set) × (UNIV :: nat set)) (UNIV :: nat set)
proof −
  let ?f = λ(i :: nat, j :: nat). 2^i ∗ (2 ∗ j + 1) − 1
  have ?f ‘ (UNIV × UNIV) = UNIV
  proof (rule surj-onI; split-pair)
    fix i j :: nat
    show 2^i ∗ (2 ∗ j + 1) − 1 ∈ UNIV by simp
  next
    fix n :: nat
    obtain i and j where n + 1 = 2^i ∗ (2 ∗ j + 1) using ex-2-3-factorization-existence by auto
    hence 2^i ∗ (2 ∗ j + 1) − 1 = n by presburger
    thus n ∈ ?f ‘ (UNIV × UNIV) by auto
  qed
  moreover have inj-on ?f (UNIV × UNIV)
  proof (rule inj-onI, split-pair)
    thm inj-onI
    fix i j i′ j′ :: nat
    assume 2^i ∗ (2 ∗ j + 1) − 1 = 2^i′ ∗ (2 ∗ j′ + 1) − 1
    moreover have 0 < 2^i ∗ (2 ∗ j + 1) and 0 < 2^i′ ∗ (2 ∗ j′ + 1) by simp+
    ultimately have 2^i ∗ (2 ∗ j + 1) = 2^i′ ∗ (2 ∗ j′ + 1) by linarith
    hence i = i′ and j = j′ using ex-2-3-factorization-uniqueness by blast+
    thus (i, j) = (i′, j′) by simp
  qed
  ultimately have bij-betw ?f (UNIV × UNIV) UNIV by (intro bij-betw-imageI)
  thus equipotent ((UNIV :: nat set) × (UNIV :: nat set)) (UNIV :: nat set) by auto
qed

proposition ex-2-4:
  assumes (a :: real) < (b :: real)
    and (c :: real) < (d :: real)
  shows equipotent {a .. b} {c .. d}
proof −
  define f where f x ≡ (d − c) ∗ (x − a) / (b − a) + c for x
  have bij-betw f {a .. b} {c .. d}
  proof (rule bij-betw-imageI′)
    fix x and x′
    assume x ∈ {a .. b}
      and x′ ∈ {a .. b}
      and f x = f x′
    with f-def have (d − c) ∗ (x − a) / (b − a) = (d − c) ∗ (x′ − a) / (b − a) by simp
    with assms(1) have (d − c) ∗ (x − a) = (d − c) ∗ (x′ − a) by simp
    with assms(2) show x = x′ by simp
  next
    fix x
```

```
  assume x: x ∈ {a .. b}
  have c ≤ f x
  proof −
    from assms(2) have 0 ≤ d − c by simp
    moreover from x have 0 ≤ x − a by simp
    moreover from assms(1) have 0 ≤ b − a by simp
    ultimately have 0 ≤ (d − c) ∗ (x − a) / (b − a) by simp
    with f-def show ?thesis by simp
  qed
  moreover have f x ≤ d
  proof −
    from x and assms(1) have (x − a) / (b − a) ≤ 1 by simp
    moreover from assms(2) have 0 ≤ d − c by simp
    ultimately have (d − c) ∗ ((x − a) / (b − a)) ≤ d − c by (fact mult-left-le)
    thus ?thesis unfolding f-def by simp
  qed
  ultimately show f x ∈ {c .. d} by simp
next
  fix y
  assume y: y ∈ {c .. d}
  define x where x ≡ (y − c) ∗ (b − a) / (d − c) + a
  with f-def have f x = y
  proof −
    have f x = (d − c) ∗ (((y − c) ∗ (b − a) / (d − c) + a) − a) / (b − a) + c
      unfolding f-def and x-def ..
    also from assms have . . . = y by simp
    finally show ?thesis .
  qed
  moreover have x ∈ {a .. b}
  proof −
    have a ≤ x
    proof −
      from y have 0 ≤ y − c by simp
      moreover from assms(1) have 0 ≤ b − a by simp
      moreover from assms(2) have 0 ≤ d − c by simp
      ultimately have 0 ≤ (y − c) ∗ (b − a) / (d − c) by simp
      thus ?thesis unfolding x-def by simp
    qed
    moreover have x ≤ b
    proof −
      from y and assms(2) have (y − c) / (d − c) ≤ 1 by simp
      moreover from assms(1) have 0 ≤ (b − a) by simp
      ultimately have (b − a) ∗ ((y − c) / (d − c)) ≤ b − a by (fact mult-left-le)
      hence (y − c) ∗ (b − a) / (d − c) + a ≤ b by argo
      thus ?thesis unfolding x-def by simp
```

```
      qed
      ultimately show ?thesis by simp
    qed
    ultimately show y ∈ f ‘ {a .. b} by auto
  qed
  thus ?thesis by (fact equipotentI)
qed

lemma non-negative-quadratic-discriminant-implies-real-root:
  assumes a ≠ 0
    and discrim a b c ≥ 0
  obtains x where a ∗ x^2 + b ∗ x + c = 0
    and x = (−b + sqrt(discrim a b c)) / (2 ∗ a)
  using assms and discriminant-iff by blast

proposition ex-2-5:
  shows equipotent {−(1 :: real) <..< (1 :: real)} (UNIV :: real set)
proof −
  define f where f x ≡ x / (1 − x^2) for x :: real
  have bij-betw f {−(1 ::real) <..< (1 :: real)} (UNIV :: real set)
  proof (rule bij-betw-imageI')
    fix x and x'
    assume x: x ∈ {−1 <..< 1}
      and x': x' ∈ {−1 <..< 1}
      and f x = f x'
    from this(3) have x / (1 − x^2) = x' / (1 − x'^2) unfolding f-def .
    hence (1 − x^2) ∗ (1 − x'^2) ∗ (x / (1 − x^2)) = (1 − x^2) ∗ (1 − x'^2) ∗ (x' / (1 − x'^2)) by simp
    moreover have 1 − x^2 ≠ 0
    proof (rule notI)
      assume 1 − x^2 = 0
      hence x = 1 ∨ x = −1 by algebra
      with x show False by simp
    qed
    moreover have 1 − x'^2 ≠ 0
    proof (rule notI)
      assume 1 − x'^2 = 0
      hence x' = 1 ∨ x' = −1 by algebra
      with x' show False by simp
    qed
    ultimately have (1 − x'^2) ∗ x = (1 − x^2) ∗ x' by auto
    hence x − x ∗ x'^2 − x' + x' ∗ x^2 = 0 by argo
    hence ∗: x − x' + x ∗ x' ∗ (x − x') = 0 by algebra
    {
      assume x ≠ x'
      with ∗ have 1 + x ∗ x' = 0 by algebra
```

```
    moreover from x and x' have -1 < x * x'
    proof -
      consider (A) x * x' ≥ 0
          | (B) x > 0
          | (C) x' > 0
        by (metis less-eq-real-def linorder-neqE-linordered-idom zero-le-mult-iff)
      thus ?thesis
      proof cases
        case A
        thus ?thesis by simp
      next
        case B
        moreover from x' have -1 < x' by simp
        ultimately have x * (-1) < x * x' by (blast dest: mult-strict-left-mono)
        hence -x < x * x' by simp
        moreover from x have -1 < -x by simp
        ultimately show ?thesis by simp
      next
        case C
        moreover from x have -1 < x by simp
        ultimately have (-1) * x' < x * x' by (blast dest: mult-strict-right-mono)
        hence -x' < x * x' by simp
        moreover from x' have -1 < -x' by simp
        ultimately show ?thesis by simp
      qed
    qed
    ultimately have False by simp
  }
  thus x = x' by (fact ccontr)
next
  fix x :: real
  show f x ∈ UNIV by simp
next
  fix y :: real
  {
    assume y: y = 0
    with f-def have  f 0 = y by simp
    hence y ∈ f ` {-1 <..< 1} by auto
  }
  moreover {
    assume y ≠ 0
    have discrim: discrim y 1 (-y) ≥ 0 unfolding discrim-def by simp
    from ‹y ≠ 0› and discrim obtain x where x-root: y * x^2 + 1 * x + (-y) = 0
      and x: x = (-1 + sqrt(discrim y 1 (-y))) / (2 * y)
      by (elim non-negative-quadratic-discriminant-implies-real-root)
```

```
have f x = y
proof -
  from x-root have (1 - x^2) * y = x by argo
  moreover have 1 - x^2 ≠ 0
  proof (rule notI)
    assume 1 - x^2 = 0
    moreover from this and x-root have x = 0 by simp
    ultimately show False by simp
  qed
  ultimately have y = x / (1 - x^2) by (metis nonzero-mult-div-cancel-left)
  thus ?thesis unfolding f-def by simp
qed
moreover have x ∈ {-1 <..< 1}
proof -
  {
    assume y > 0
    have 0 < x
    proof -
      from ‹y > 0› have 0 < -1 + sqrt(1 + 4 * y * y) by simp
      moreover from ‹y > 0› have 0 < 2 * y by simp
      ultimately have 0 < (-1 + sqrt(1 + 4 * y * y)) / (2 * y) by simp
      with x show 0 < x unfolding discrim-def by simp
    qed
    moreover have x < 1
    proof -
      from ‹y > 0› have 1 + 4 * y * y < (1 + 2 * y) * (1 + 2 * y) by argo
      hence sqrt(1 + 4 * y * y) < sqrt((1 + 2 * y) * (1 + 2 * y))
        by (fact real-sqrt-less-mono)
      moreover from ‹y > 0› have 0 < 1 + 2 * y by simp
      ultimately have sqrt(1 + 4 * y * y) < 1 + 2 * y by simp
      hence -1 + sqrt(1 + 4 * y * y) < 2 * y by simp
      moreover from ‹y > 0› have 0 < 2 * y by simp
      ultimately have (-1 + sqrt(1 + 4 * y * y)) / (2 * y) < 1 by simp
      with x show x < 1 unfolding discrim-def by simp
    qed
    ultimately have x ∈ {-1 <..< 1} by simp
  }
  moreover {
    assume y < 0
    have -1 < x
    proof -
      from ‹y < 0› have 0 < -4 * y by simp
      hence 1 + 4 * y * y < (1 - 2 * y) * (1 - 2 * y) by argo
      hence sqrt(1 + 4 * y * y) < sqrt((1 - 2 * y) * (1 - 2 * y))
        by (fact real-sqrt-less-mono)
```

```
        moreover from ‹y < 0› have 0 < 1 − 2 ∗ y by simp
        ultimately have sqrt(1 + 4 ∗ y ∗ y) < 1 − 2 ∗ y by simp
        hence 2 ∗ y < 1 − sqrt(1 + 4 ∗ y ∗ y) by simp
        moreover from ‹y < 0› have 2 ∗ y < 0 by simp
        ultimately have (1 − sqrt(1 + 4 ∗ y ∗ y)) / (2 ∗ y) < 1 by simp
        hence −1 < (−1 + sqrt(1 + 4 ∗ y ∗ y)) / (2 ∗ y) by argo
        with x show ?thesis unfolding discrim-def by simp
      qed
      moreover have x < 0
      proof −
        from ‹y < 0› have y ≠ 0 by simp
        hence 0 < y ∗ y using not-real-square-gt-zero by blast
        hence 0 < −1 + sqrt(1 + 4 ∗ y ∗ y) by simp
        moreover from ‹y < 0› have 2 ∗ y < 0 by simp
        ultimately have (−1 + sqrt(1 + 4 ∗ y ∗ y)) / (2 ∗ y) < 0 by (fact divide-pos-neg)
        with x show ?thesis unfolding discrim-def by simp
      qed
      ultimately have x ∈ {−1 <..< 1} by simp
    }
    moreover note ‹y ≠ 0›
    ultimately show x ∈ {−1 <..< 1} by argo
  qed
  ultimately have y ∈ f ‘ {−1 <..< 1} by auto
 }
 ultimately show y ∈ f ‘ {−1 <..< 1} by auto
qed
thus ?thesis by auto
qed
```

— Proof for the remaining propositions of example 2.5 is postponed after theorem 2.2

2.1.2 B) Bernstein Theorem

```
fun bernstein-seq :: 'a set ⇒ 'b set ⇒ ('a ⇒ 'b) ⇒ ('b ⇒ 'a) ⇒ nat ⇒ 'a set
  and bernstein-seq' :: 'a set ⇒ 'b set ⇒ ('a ⇒ 'b) ⇒ ('b ⇒ 'a) ⇒ nat ⇒ 'b set where
    bernstein-seq' A B f g 0 = B − f ‘ A
  | bernstein-seq A B f g n = g ‘ bernstein-seq' A B f g n
  | bernstein-seq' A B f g (Suc n) = f ‘ bernstein-seq A B f g n
```

```
lemma bernstein-seq-subset:
  assumes f ‘ A ⊆ B
    and g ‘ B ⊆ A
  shows bernstein-seq A B f g n ⊆ A
proof (induct n)
  case 0
```

```
  have bernstein-seq A B f g 0 = g ‘ (B − f ‘ A) by simp
  also from assms have ... ⊆ A by auto
  finally show ?case .
next
  case (Suc n)
  have bernstein-seq A B f g (Suc n) = g ‘ f ‘ bernstein-seq A B f g n by simp
  with assms and Suc.hyps show ?case by fastforce
qed

lemma bernstein-seq′-subset:
  assumes f ‘ A ⊆ B
    and g ‘ B ⊆ A
  shows bernstein-seq′ A B f g n ⊆ B
proof (induct n)
  case 0
  have bernstein-seq′ A B f g 0 = B − f ‘ A by simp
  also have ... ⊆ B by auto
  finally show ?case .
next
  case (Suc n)
  have bernstein-seq′ A B f g (Suc n) = f ‘ g ‘ bernstein-seq′ A B f g n by simp
  with assms and Suc.hyps show ?case by fastforce
qed

lemma zero-Un-UNION-Suc-eq:
  shows A 0 ∪ (⋃n. A (Suc n)) = (⋃n. A n)
proof −
  have A 0 ∪ (⋃n. A (Suc n)) = (⋃n ∈ {0}. A n) ∪(⋃n ∈ {Suc n | n. True}. A n) by auto
  also have ... = (⋃n ∈ {0} ∪ {Suc n | n. True}. A n) by simp
  also have ... = (⋃n. A n)
  proof −
    {
      fix n
      have n ∈ {0} ∪ {Suc n | n. True}
      proof (induct n)
        case 0
        show ?case by simp
      next
        case (Suc n)
        show ?case by simp
      qed
    }
    hence {0} ∪ {Suc n | n. True} = UNIV by blast
    thus ?thesis by simp
  qed
```

finally show *?thesis* **.**
qed

theorem *thm-2-2*:
assumes f ‘ $A \subseteq B$
and *inj-on* f A
and g ‘ $B \subseteq A$
and *inj-on* g B
shows *equipotent* A B
proof $-$
let $?A = \bigcup n.$ *bernstein-seq* A B f g n
let $?A' = A - ?A$
let $?B = \bigcup n.$ *bernstein-seq'* A B f g n
let $?B' = B - ?B$
{
fix n
from *assms*(*1*,*3*) **have** *bernstein-seq* A B f g $n \subseteq A$ **by** (*rule bernstein-seq-subset*)
}
hence $?A \subseteq A$ **by** *auto*
hence $?A' \subseteq A$ **by** *auto*
{
fix n
from *assms*(*1*,*3*) **have** *bernstein-seq'* A B f g $n \subseteq B$ **by** (*rule bernstein-seq'-subset*)
}
hence $?B \subseteq B$ **by** *auto*
hence $?B' \subseteq B$ **by** *auto*
have f ‘ $?A' = ?B'$
proof $-$
from ‹$?A \subseteq A$› **and** *assms*(*2*) **have** f ‘ $?A' = f$ ‘ $A - f$ ‘ $?A$ **by** (*intro prob-1-4-5-c*)
also have $\ldots = f$ ‘ $A - (\bigcup n.\ f$ ‘ *bernstein-seq* A B f g $n)$ **by** *blast*
also have $\ldots = f$ ‘ $A - (\bigcup n.$ *bernstein-seq'* A B f g $(Suc\ n))$ **by** *simp*
also from *assms*(*1*) **have** $\ldots = B - (B - f$ ‘ $A) - (\bigcup n.$ *bernstein-seq'* A B f g $(Suc\ n))$ **by** *auto*
also have $\ldots = B -$ *bernstein-seq'* A B f g $0 - (\bigcup n.$ *bernstein-seq'* A B f g $(Suc\ n))$ **by** *simp*
also have $\ldots = B - ($*bernstein-seq'* A B f g $0 \cup (\bigcup n.$ *bernstein-seq'* A B f g $(Suc\ n)))$ **by** *auto*
also have $\ldots = ?B'$
by (*simp only*: *zero-Un-UNION-Suc-eq*[**where** $A = \lambda n.$ *bernstein-seq'* A B f g n])
finally show *?thesis* **.**
qed
moreover from *assms*(*2*) **have** *inj-on* f $?A'$ **by** (*fact inj-on-diff*)
ultimately have *bij-betw* f $?A'$ $?B'$ **by** (*intro bij-betw-imageI*)
have g ‘ $?B = ?A$
proof (*rule surj-onI*)
fix b
assume $b \in ?B$

  then obtain n **where** $b \in \textit{bernstein-seq}'\ A\ B\ f\ g\ n$ **by** *fast*
  hence $g\ b \in \textit{bernstein-seq}\ A\ B\ f\ g\ n$ **by** *simp*
  thus $g\ b \in ?A$ **by** *auto*
next
  fix a
  assume $a \in ?A$
  then obtain n **where** $a \in \textit{bernstein-seq}\ A\ B\ f\ g\ n$ **by** *blast*
  hence $a \in g\ `\ \textit{bernstein-seq}'\ A\ B\ f\ g\ n$ **by** *simp*
  then obtain b **where** $b \in \textit{bernstein-seq}'\ A\ B\ f\ g\ n$ **and** $a = g\ b$ **by** *auto*
  from $\textit{this}(1)$ **have** $b \in ?B$ **by** *auto*
  with $\langle a = g\ b\rangle$ **show** $a \in g\ `\ ?B$ **by** *simp*
qed
moreover from $\langle ?B \subseteq B\rangle$ **and** $\textit{assms}(4)$ **have** $\textit{inj-on}\ g\ ?B$ **by** $(\textit{elim inj-on-subset})$
ultimately have $\textit{bij-betw}\ g\ ?B\ ?A$ **by** $(\textit{intro bij-betw-imageI})$
then obtain f' **where** $\textit{bij-betw}\ f'\ ?A\ ?B$ **using** *bij-betw-inv* **by** *blast*
hence $f'\ `\ ?A = ?B$ **by** $(\textit{fact bij-betw-imp-surj-on})$
let $?F = \lambda a.\ \textit{if}\ a \in ?A'\ \textit{then}\ f\ a\ \textit{else}\ f'\ a$
have $?F\ `\ A = B$
proof $(\textit{rule surj-onI})$
  fix a
  assume $a \in A$
  {
    assume $a \in ?A'$
    hence $?F\ a = f\ a$ **by** *simp*
    with $\langle a \in A\rangle$ **and** $\textit{assms}(1)$ **have** $?F\ a \in B$ **by** *fast*
  }
  moreover {
    assume $a \notin ?A'$
    hence $?F\ a = f'\ a$ **by** *argo*
    also have $\ldots \in ?B$
    proof $-$
      from $\langle a \in A\rangle$ **and** $\langle a \notin ?A'\rangle$ **have** $a \in ?A$ **by** *simp*
      moreover note $\langle f'\ `\ ?A = ?B\rangle$
      ultimately show $f'\ a \in ?B$ **by** *blast*
    qed
    also note $\langle \ldots \subseteq B\rangle$
    finally have $?F\ a \in B$ **.**
  }
  ultimately show $?F\ a \in B$ **by** *blast*
next
  fix b
  assume $b \in B$
  {
    assume $b \in ?B$
    with $\langle f'\ `\ ?A = ?B\rangle$ **obtain** a **where** $a \in ?A$ **and** $b = f'\ a$ **by** *blast*

hence *?F* $a = b$ **by** *simp*
moreover from $\langle a \in$ *?A*$\rangle$ **and** $\langle$*?A* $\subseteq A\rangle$ **have** $a \in A$ **..**
ultimately have $b \in$ *?F* ‘ A **by** *blast*
}
moreover {
assume $b \notin$ *?B*
with $\langle b \in B\rangle$ **have** $b \in$ *?B'* **by** *simp*
with $\langle f$ ‘ *?A'* $=$ *?B'*$\rangle$ **have** $b \in f$ ‘ *?A'* **by** *simp*
then obtain a **where** $a \in$ *?A'* **and** $b = f\ a$ **by** *blast*
hence *?F* $a = b$ **by** *argo*
moreover from $\langle a \in$ *?A'*$\rangle$ **and** $\langle$*?A'* $\subseteq A\rangle$ **have** $a \in A$ **..**
ultimately have $b \in$ *?F* ‘ A **by** *blast*
}
ultimately show $b \in$ *?F* ‘ A **by** *blast*
qed
moreover have *inj-on ?F A*
proof (*rule inj-onI*)
fix a **and** a'
assume $a \in A$ **and**
$a' \in A$ **and**
?F $a =$ *?F* a'
consider (A) $a \in$ *?A'* **and** $a' \in$ *?A'* |
(B) $a \in$ *?A'* **and** $a' \notin$ *?A'* |
(C) $a \notin$ *?A'* **and** $a' \in$ *?A'* |
(D) $a \notin$ *?A'* **and** $a' \notin$ *?A'* **by** *blast*
thus $a = a'$
proof *cases*
case A
with $\langle$*?F* $a =$ *?F* $a'\rangle$ **have** $f\ a = f\ a'$ **by** *argo*
with A **and** $\langle$*inj-on f ?A'*$\rangle$ **show** *?thesis* **by** (*elim inj-onD*)
next
case B
from $B(2)$ **and** $\langle a' \in A\rangle$ **have** $a' \in$ *?A* **by** *blast*
from B **and** $\langle$*?F* $a =$ *?F* $a'\rangle$ **have** $f\ a = f'\ a'$ **by** *argo*
moreover from $\langle a \in$ *?A'*$\rangle$ **and** $\langle f$ ‘ *?A'* $=$ *?B'*$\rangle$ **have** $f\ a \in$ *?B'* **by** *blast*
moreover from $\langle a' \in$ *?A*$\rangle$ **and** $\langle f'$ ‘ *?A* $=$ *?B*$\rangle$ **have** $f'\ a' \in$ *?B* **by** *blast*
ultimately have *False* **by** *simp*
thus *?thesis* **..**
next
case C
from $C(1)$ **and** $\langle a \in A\rangle$ **have** $a \in$ *?A* **by** *simp*
from C **and** $\langle$*?F* $a =$ *?F* $a'\rangle$ **have** $f'\ a = f\ a'$ **by** *argo*
moreover from $\langle a \in$ *?A*$\rangle$ **and** $\langle f'$ ‘ *?A* $=$ *?B*$\rangle$ **have** $f'\ a \in$ *?B* **by** *blast*
moreover from $\langle a' \in$ *?A'*$\rangle$ **and** $\langle f$ ‘ *?A'* $=$ *?B'*$\rangle$ **have** $f\ a' \in$ *?B'* **by** *blast*
ultimately have *False* **by** *simp*

```
      thus ?thesis ..
    next
      case D
      from D and ‹?F a = ?F a'› have f' a = f' a' by argo
      moreover from D and ‹a ∈ A› and ‹a' ∈ A› have a ∈ ?A and a' ∈ ?A by simp+
      moreover from ‹bij-betw f' ?A ?B› have inj-on f' ?A by auto
      ultimately show ?thesis by (elim inj-onD)
    qed
  qed
  ultimately have bij-betw ?F A B by (intro bij-betw-imageI)
  thus ?thesis by auto
qed

lemma ex-2-5':
  assumes (a :: real) < (b :: real)
    and (c :: real) < (d :: real)
  obtains f where f ` {a <..< b} ⊆ {c <..< d}
    and inj-on f {a <..< b}
proof −
  define f where f x ≡ (d − c) ∗ (x − a) / (b − a) + c for x
  have f ` {a <..< b} ⊆ {c <..< d}
  proof (rule image-subsetI)
    fix x
    assume x: x ∈ {a <..< b}
    have c < f x
    proof −
      from assms(1) have 0 < b − a by simp
      moreover from assms(2) have 0 < d − c by simp
      moreover from x have 0 < x − a by simp
      ultimately show ?thesis unfolding f-def by simp
    qed
    moreover have f x < d
    proof −
      from assms(1) have 0 < b − a by simp
      moreover from x have 0 < x − a by simp
      moreover from x have x − a < b − a by simp
      ultimately have (x − a) / (b − a) < 1 by simp
      moreover from assms(2) have 0 < d − c by simp
      ultimately have (d − c) ∗ ((x − a) / (b − a)) < (d − c) ∗ 1 by (fact mult-strict-left-mono)
      thus ?thesis unfolding f-def by simp
    qed
    ultimately show f x ∈ {c <..< d} by simp
  qed
  moreover have inj-on f {a <..< b}
  proof (rule inj-onI)
```

fix x **and** x'
assume $f\, x = f\, x'$
hence $(d - c) * (x - a) \;/\; (b - a) = (d - c) * (x' - a) \;/\; (b - a)$ **unfolding** *f-def* **by** *simp*
moreover from *assms*(*1*) **have** $b - a \neq 0$ **by** *simp*
ultimately have $(d - c) * (x - a) = (d - c) * (x' - a)$ **by** *simp*
moreover from *assms*(*2*) **have** $d - c \neq 0$ **by** *simp*
ultimately show $x = x'$ **by** *simp*
qed
ultimately show *thesis* **by** (*fact that*)
qed

lemma *ex-2-5''*:
assumes $(a :: real) < (b :: real)$
and $(c :: real) < (d :: real)$
shows *equipotent* $\{a <..< b\}$ $\{c <..< d\}$
proof –
from *assms* **obtain** f **where** *f1*: f ‘ $\{a <..< b\} \subseteq \{c <..< d\}$
and *f2*: *inj-on* f $\{a <..< b\}$ **by** (*auto elim*: *ex-2-5'*)
from *assms* **obtain** g **where** *g1*: g ‘ $\{c <..< d\} \subseteq \{a <..< b\}$
and *g2*: *inj-on* g $\{c <..< d\}$ **by** (*auto elim*: *ex-2-5'*)
from *f1* **and** *f2* **and** *g1* **and** *g2* **show** *?thesis* **by** (*fact thm-2-2*)
qed

lemma *ex-2-5'''*:
assumes $(a :: real) < (b :: real)$
shows *equipotent* $\{a <..< b\}$ $(UNIV :: real\ set)$
proof –
from *assms* **have** *equipotent* $\{a <..< b\}$ $\{-(1 :: real) <..< (1 :: real)\}$ **by** (*simp add*: *ex-2-5''*)
moreover have *equipotent* $\{-(1 :: real) <..< (1 :: real)\}$ $(UNIV :: real\ set)$ **by** (*fact ex-2-5*)
ultimately show *?thesis* **by** (*fact prop-2-1-3*)
qed

theorem *thm-2-2'*:
assumes f ‘ $A \subseteq B$
and *inj-on* f A
and f' ‘ $A = B$
shows *equipotent* A B
proof –
from *assms*(*3*) **obtain** g **where** g ‘ $B \subseteq A$ **and** *inj-on* g B **by** (*elim cor-inj-on-iff-surj-on-b*)
moreover from *assms*(*3*) **have** f' ‘ $A \subseteq B$ **by** *simp*
moreover note *assms*(*1*,*2*)
ultimately show *equipotent* A B **by** (*intro thm-2-2*)
qed

theorem *thm-2-2''*:

```
  assumes f ‘ A = B
    and g ‘ B = A
  shows equipotent A B
proof –
  from assms(2) obtain f′ where f′ ‘ A ⊆ B and inj-on f′ A by (elim cor-inj-on-iff-surj-on-b)
  with assms(1) show ?thesis by (intro thm-2-2′)
qed

lemma surj-on-from-subset-imp-surj-on:
  assumes f ‘ A′ = B
    and A′ ⊆ A
    and A′ = {} ⟹ A = {}
  obtains f′ where f′ ‘ A = B
proof –
  {
    assume A′ = {}
    with assms(1,3) have A = {} and B = {} by simp+
    hence ∃f′. f′ ‘ A = B by simp
  }
  moreover {
    assume A′ ≠ {}
    with assms(1) have B ≠ {} by auto
    then obtain b where b ∈ B by auto
    let ?f′ = λa. if a ∈ A′ then f a else b
    have ?f′ ‘ A = B
    proof (rule surj-onI)
      fix a
      assume a ∈ A
      {
        assume a ∈ A′
        with assms(1) have ?f′ a ∈ B by auto
      }
      moreover {
        assume a ∉ A′
        with ‹b ∈ B› have ?f′ a ∈ B by simp
      }
      ultimately show ?f′ a ∈ B by simp
    next
      fix b′
      assume b′ ∈ B
      with assms(1) obtain a where a ∈ A′ and f a = b′ by blast
      hence ?f′ a = b′ by simp
      moreover from ‹a ∈ A′› and assms(2) have a ∈ A ..
      ultimately show b′ ∈ ?f′ ‘ A by auto
    qed
```

  hence $\exists f'. f' \text{ ‘ } A = B$ **by** *blast*
 }
 ultimately show *thesis* **using** *that* **by** *auto*
qed

theorem *thm-2-2‴*:
 assumes $B_1 \subseteq B$
  and *equipotent* A B_1
  and $A_1 \subseteq A$
  and *equipotent* B A_1
 shows *equipotent* A B
proof –
 from *assms*(2) **obtain** f **where** *bij-betw* f A B_1 **by** *auto*
 then obtain g **where** *bij-betw* g B_1 A **by** (*auto dest*: *bij-betw-inv*)
 hence $g \text{ ‘ } B_1 = A$ **by** *auto*
 moreover have $B_1 = \{\} \Longrightarrow B = \{\}$
 proof –
  assume $B_1 = \{\}$
  with *assms*(2) **have** $A = \{\}$ **by** *simp*
  with *assms*(3) **have** $A_1 = \{\}$ **by** *simp*
  with *assms*(4) **show** $B = \{\}$ **by** *simp*
 qed
 moreover note *assms*(1)
 ultimately obtain g' **where** $g' \text{ ‘ } B = A$ **using** *surj-on-from-subset-imp-surj-on* **by** *blast*
 from *assms*(4) **obtain** g'' **where** *bij-betw* g'' B A_1 **by** *auto*
 then obtain f' **where** *bij-betw* f' A_1 B **by** (*auto dest*: *bij-betw-inv*)
 hence $f' \text{ ‘ } A_1 = B$ **by** *auto*
 moreover have $A_1 = \{\} \Longrightarrow A = \{\}$
 proof –
  assume $A_1 = \{\}$
  with *assms*(4) **have** $B = \{\}$ **by** *simp*
  with *assms*(1) **have** $B_1 = \{\}$ **by** *simp*
  with *assms*(2) **show** $A = \{\}$ **by** *simp*
 qed
 moreover note *assms*(3)
 ultimately obtain f'' **where** $f'' \text{ ‘ } A = B$ **by** (*auto intro*: *surj-on-from-subset-imp-surj-on*)
 with $\langle g' \text{ ‘ } B = A \rangle$ **show** *equipotent* A B **by** (*intro thm-2-2″*)
qed

2.1.3 C) Notion of Cardinality

abbreviation *aleph-zero* :: *nat rel* ($\aleph_0$)
 where *aleph-zero* $\equiv$ |*UNIV* :: *nat set*|

abbreviation *aleph* :: *real rel* ($\aleph$)

where *aleph* $\equiv$ $|UNIV :: real\ set|$

lemmas *card-eqI* [*intro*] = *card-of-ordIsoI*

lemma *card-eqI'* [*intro*]:
assumes *equipotent A B*
shows $|A| =o |B|$
proof –
from *assms* **obtain** *f* **where** *bij-betw f A B* **by** *auto*
thus *?thesis* **by** *auto*
qed

lemma *card-eqE* [*elim*]:
assumes $|A| =o |B|$
obtains *f* **where** *bij-betw f A B*
proof –
from *assms* **obtain** *f* **where** *bij-betw f A B* **using** *card-of-ordIso* **by** *blast*
thus *thesis* **by** (*fact that*)
qed

lemma *card-eqE'* [*elim*]:
assumes $|A| =o |B|$
obtains *f* **where** *bij-betw f B A*
proof –
from *assms* **obtain** *f* **where** *bij-betw f A B* **by** *auto*
hence *bij-betw* (*inv-into A f*) *B A* **by** (*fact bij-betw-inv-into*)
with *that* **show** *thesis* **by** *simp*
qed

proposition *card-eq-definition*:
shows $|A| =o |B| \longleftrightarrow$ *equipotent A B*
by *auto*

lemma *card-eq-refl* [*simp*]:
shows $|A| =o |A|$
proof –
have *equipotent A A* **by** *simp*
thus *?thesis* **by** *auto*
qed

lemma *card-eq-sym*:
fixes $A :: 'a\ set$
and $B :: 'b\ set$
assumes $|A| =o |B|$
shows $|B| =o |A|$

proof –
from *assms* **have** *equipotent A B* **by** *auto*
hence *equipotent B A* **by** (*fact prop-2-1-2*)
thus *?thesis* **by** *auto*
qed

lemma *card-eq-trans* [*trans*]:
fixes A :: $'a$ *set*
and B :: $'b$ *set*
and C :: $'c$ *set*
assumes $|A| =o |B|$
and $|B| =o |C|$
shows $|A| =o |C|$
proof –
from *assms*(*1*) **have** *equipotent A B* **by** *auto*
moreover from *assms*(*2*) **have** *equipotent B C* **by** *auto*
ultimately have *equipotent A C* **by** (*fact prop-2-1-3*)
thus *?thesis* **by** *auto*
qed

proposition *czero-definition*:
shows (*czero* :: $'a$ *rel*) = $|\{\}$ :: $'a$ *set*$|$
by (*fact czero-def*)

lemma *empty-card-eq-czero*:
shows $|\{\}$ :: $'a$ *set*$|$ $=o$ (*czero* :: $'b$ *rel*)
proof –
define f :: $'a \Rightarrow 'b$ **where** f $a \equiv$ *undefined* **for** a
from *f-def* **have** *inj-on* f $\{\}$ **and** f ‘ $\{\} = \{\}$ **by** *simp-all*
hence *bij-betw* f ($\{\}$ :: $'a$ *set*) ($\{\}$ :: $'b$ *set*) **by** (*intro bij-betw-imageI*)
hence $|\{\}$ :: $'a$ *set*$|$ $=o$ $|\{\}$ :: $'b$ *set*$|$ **by** *auto*
thus *?thesis* **unfolding** *czero-definition* **by** *simp*
qed

lemma *empty-card-eq-empty*:
shows $|\{\}$ :: $'a$ *set*$|$ $=o$ $|\{\}$ :: $'b$ *set*$|$
proof –
define f :: $'a \Rightarrow 'b$ **where** f $a \equiv$ *undefined* **for** a
from *f-def* **have** *inj-on* f $\{\}$ **and** f ‘ $\{\} = \{\}$ **by** *simp-all*
hence *bij-betw* f $\{\}$ $\{\}$ **by** (*intro bij-betw-imageI*)
thus *?thesis* **by** *auto*
qed

lemma *czero-card-eq-empty*:
shows (*czero* :: $'a$ *rel*) $=o$ $|\{\}$ :: $'b$ *set*$|$

proof −
  have $(czero :: 'a\ rel) = |\{\} :: 'a\ set|$ **unfolding** *czero-definition* **..**
  moreover have $|\{\} :: 'a\ set| =o\ |\{\} :: 'b\ set|$ **by** (*fact empty-card-eq-empty*)
  ultimately show *?thesis* **by** *simp*
qed

lemma *czero-refl*:
  shows $(czero :: 'a\ rel) =o\ (czero :: 'b\ rel)$
proof −
  define $f :: 'a \Rightarrow 'b$ **where** $f\ a \equiv undefined$ **for** a
  have *bij-betw* $f\ (\{\} :: 'a\ set)\ (\{\} :: 'b\ set)$ **by** (*auto intro*: *bij-betwI'*)
  hence $|\{\} :: 'a\ set| =o\ |\{\} :: 'b\ set|$ **by** *auto*
  thus *?thesis* **unfolding** *czero-definition* **by** *simp*
qed

lemma *eq-empty-imp-card-eq-czero*:
  fixes $A :: 'a\ set$
  assumes $A = \{\}$
  shows $|A| =o\ (czero :: 'b\ rel)$
proof −
  from *assms* **have** $|A| = |\{\} :: 'a\ set|$ **by** *auto*
  also have $\ldots\ =o\ (czero :: 'b\ rel)$ **by** (*fact empty-card-eq-czero*)
  finally show *?thesis* **by** *auto*
qed

proposition *cone-definition* [*simp*]:
  shows $cone =o\ |\{a\}|$
proof −
  have $cone = |\{()\}|$ **by** (*simp only*: *card-of-refl cone-def*)
  also have $\ldots\ =o\ |\{a\}|$
  proof −
    let $?f = \lambda x.\ a$
    have $?f\ `\ \{()\} = \{a\}$ **by** *simp*
    moreover have *inj-on* $?f\ \{()\}$ **by** *simp*
    ultimately have *bij-betw* $?f\ \{()\}\ \{a\}$ **by** (*intro bij-betw-imageI*)
    thus *?thesis* **by** (*fact card-of-ordIsoI*)
  qed
  finally show $cone =o\ |\{a\}|$ **.**
qed

lemma *singleton-card-eq-cone*:
  shows $|\{a\}| =o\ cone$
proof −
  define f **where** $f\ x \equiv if\ x = a\ then\ ()\ else\ undefined$ **for** x
  from *f-def* **have** *inj-on* $f\ \{a\}$ **by** *simp*

moreover from *f-def* **have** f ‘ $\{a\} = \{()\}$ **by** *simp*
ultimately have *bij-betw* f $\{a\}$ $\{()\}$ **by** (*intro bij-betw-imageI*)
hence $|\{a\}|$ $=o$ $|\{()\}|$ **by** *auto*
thus *?thesis* **unfolding** *cone-def* **by** *simp*
qed

lemma *ctwo-definition*:
shows *ctwo* = |*UNIV* :: *bool set*|
unfolding *ctwo-def* **..**

lemma *doubleton-card-eq-ctwo*:
assumes $a \neq b$
shows $|\{a,\ b\}|$ $=o$ *ctwo*
using *assms* **unfolding** *ctwo-definition* **by** (*simp add*: *card-of-bool ordIso-symmetric*)

lemma *card-leqI* [*intro*]:
assumes f ‘ $A \subseteq B$
and *inj-on* f A
shows $|A|$ $\leq o$ $|B|$
proof −
from *assms* **show** $|A|$ $\leq o$ $|B|$ **by** (*blast intro*: *card-of-ordLeq*[*THEN iffD1*])
qed

lemma *card-leqE* [*elim*]:
assumes $|A|$ $\leq o$ $|B|$
obtains f **where** f ‘ $A \subseteq B$ **and** *inj-on* f A
proof −
from *assms* **obtain** f **where** f ‘ $A \subseteq B$ **and** *inj-on* f A
by (*fast dest*: *card-of-ordLeq*[*THEN iffD2*])
thus *thesis* **by** (*fact that*)
qed

proposition *card-leq-definition*:
shows $|A|$ $\leq o$ $|B| \longleftrightarrow (\exists f.\ f$ ‘ $A \subseteq B \wedge$ *inj-on* f $A)$
by *fast*

lemma *card-lessI* [*intro*]:
assumes $|A|$ $\leq o$ $|B|$
and $\neg|A|$ $=o$ $|B|$
shows $|A|$ $<o$ $|B|$
using *assms* **by** (*simp add*: *ordLeq-iff-ordLess-or-ordIso*)

lemma *card-less-imp-card-leq* [*dest*]:
assumes $|A|$ $<o$ $|B|$
shows $|A|$ $\leq o$ $|B|$

by (*fact ordLess-imp-ordLeq*[*OF assms*])

lemma *card-less-imp-not-card-eq* [*dest*]:
assumes $|A| <o |B|$
shows $\neg|A| =o |B|$
by (*fact not-ordLess-ordIso*[*OF assms*])

lemma *card-lessE* [*elim*]:
assumes $|A| <o |B|$
obtains $|A| \leq o |B|$
and $\neg|A| =o |B|$
using *assms* **by** *auto*

lemma *card-less-definition*:
shows $|A| <o |B| \longleftrightarrow |A| \leq o |B| \wedge \neg |A| =o |B|$
by *auto*

lemma *card-eq-imp-card-leq*:
assumes $|A| =o |B|$
shows $|A| \leq o |B|$
proof −
from *assms* **obtain** f **where** *bij-betw* $f\ A\ B$ **by** *auto*
hence f ‘ $A \subseteq B$ **and** *inj-on* $f\ A$ **by** *auto*
thus *?thesis* **by** *auto*
qed

theorem *thm-2-3-1* [*simp*]:
shows $|A| \leq o |A|$
proof −
have $|A| =o |A|$ **by** *simp*
thus *?thesis* **by** (*fact card-eq-imp-card-leq*)
qed

theorem *thm-2-3-2*:
assumes $|A| \leq o |B|$
and $|B| \leq o |A|$
shows $|A| =o |B|$
proof −
from *assms*(*1*) **obtain** f **where** f ‘ $A \subseteq B$ **and** *inj-on* $f\ A$ **by** *auto*
moreover from *assms*(*2*) **obtain** g **where** g ‘ $B \subseteq A$ **and** *inj-on* $g\ B$ **by** *auto*
ultimately have *equipotent* $A\ B$ **by** (*rule thm-2-2*)
thus *?thesis* **by** (*fact card-eqI′*)
qed

theorem *thm-2-3-3* [*trans*]:

assumes $|A| \leq o \ |B|$
  and $|B| \leq o \ |C|$
shows $|A| \leq o \ |C|$
proof −
  from *assms(1)* **obtain** f **where** $f \text{ ‘ } A \subseteq B$ **and** *inj-on f A* **by** *auto*
  from *assms(2)* **obtain** g **where** $g \text{ ‘ } B \subseteq C$ **and** *inj-on g B* **by** *auto*
  from $\langle f \text{ ‘ } A \subseteq B \rangle$ **and** $\langle g \text{ ‘ } B \subseteq C \rangle$ **have** $(g \circ f) \text{ ‘ } A \subseteq C$ **by** *fastforce*
  moreover from $\langle f \text{ ‘ } A \subseteq B \rangle$ **and** ⟨*inj-on f A*⟩ **and** ⟨*inj-on g B*⟩ **have** *inj-on* $(g \circ f)$ A
    by (*fact thm-1-5-b*)
  ultimately show *?thesis* **by** *blast*
qed

lemma *card-eq-card-leq-trans* [*trans*]:
  fixes A :: *'a set*
    and B :: *'b set*
    and C :: *'c set*
  assumes $|A| =o \ |B|$
    and $|B| \leq o \ |C|$
  shows $|A| \leq o \ |C|$
proof −
  from *assms(1)* **have** $|A| \leq o \ |B|$ **by** *auto*
  also note *assms(2)*
  finally show *?thesis* **.**
qed

lemma *card-leq-card-eq-trans* [*trans*]:
  fixes A :: *'a set*
    and B :: *'b set*
    and C :: *'c set*
  assumes $|A| \leq o \ |B|$
    and $|B| =o \ |C|$
  shows $|A| \leq o \ |C|$
proof −
  note *assms(1)*
  also from *assms(2)* **have** $|B| \leq o \ |C|$ **by** (*fact card-eq-imp-card-leq*)
  finally show *?thesis* **.**
qed

2.1.4 Problems

proposition *prob-2-1-2*:
  fixes X :: *'a set*
  assumes $X \subseteq Y$
    and $Y \subseteq Z$
    and *equipotent X Z*

```
  obtains equipotent X Y and equipotent Y Z
proof -
  have equipotent X Y
  proof -
    from assms(3) have equipotent Z X by (fact prop-2-1-2)
    then obtain h where bij-betw h Z X by auto
    have equipotent X X by simp
    moreover note assms(1)
    moreover have equipotent Y (h ` Y)
    proof (rule equipotentI)
      from ‹bij-betw h Z X› and assms(2) have bij-betw h Y (h ` Y)
        by (auto elim: bij-betw-subset)
      moreover have bij-betw id Y Y by simp
      ultimately show bij-betw (h ∘ id) Y (h ` Y) by simp
    qed
    moreover from ‹bij-betw h Z X› and assms(2) have h ` Y ⊆ X by auto
    ultimately show ?thesis by (blast intro: thm-2-2''')
  qed
  moreover have equipotent Y Z
  proof -
    from ‹equipotent X Y› have equipotent Y X by (fact prop-2-1-2)
    also note assms(3)
    finally show ?thesis .
  qed
  ultimately show thesis by (fact that)
qed
```

```
proposition prob-2-1-4:
  fixes A :: 'a set
  assumes B ≠ {}
  shows |A| ≤o |A × B|
proof -
  from assms obtain b where b ∈ B by auto
  let ?f = λa :: 'a. (a, b)
  {
    fix a
    assume a ∈ A
    with ‹b ∈ B› have ?f a ∈ A × B by simp
  }
  hence ?f ` A ⊆ A × B by auto
  moreover have inj-on ?f A
  proof (rule inj-onI)
    fix a and a'
```

assume *?f* a = *?f* a'
thus $a = a'$ **by** *simp*
qed
ultimately show *?thesis* **by** *auto*
qed

proposition *prob-2-1-5*:
assumes $\bigwedge l.\ l \in \Lambda \Longrightarrow A\ l \neq \{\}$
and *disjoint-family-on* $A\ \Lambda$
shows $|\Lambda| \leq o\ |\bigcup l \in \Lambda.\ A\ l|$
proof −
let *?f* = $\lambda a.$ (*THE* $l.\ l \in \Lambda \wedge a \in A\ l$)
have ∗: *?f* $a = l$ **if** $l \in \Lambda$ **and** $a \in A\ l$ **for** a **and** l
proof −
from *that* **have** $a \in (\bigcup l \in \Lambda.\ A\ l)$ **by** *auto*
with *assms* **have** $\exists!l \in \Lambda.\ a \in A\ l$ **by** (*intro disjoint-family-on-imp-uniq-idx*)
with *that* **show** *?thesis* **by** *auto*
qed
have *?f* ‘ $(\bigcup l \in \Lambda.\ A\ l) = \Lambda$
proof (*rule surj-onI*)
fix a
assume $a \in (\bigcup l \in \Lambda.\ A\ l)$
then obtain l **where** $l \in \Lambda$ **and** $a \in A\ l$ **by** *auto*
hence *?f* $a = l$ **by** (*fact* ∗)
with ‹$l \in \Lambda$› **show** *?f* $a \in \Lambda$ **by** *simp*
next
fix l
assume $l \in \Lambda$
moreover from *this* **and** *assms*(*1*) **obtain** a **where** $a \in A\ l$ **by** *auto*
ultimately have $a \in (\bigcup l \in \Lambda.\ A\ l)$ **and** *?f* $a = l$ **by** (*auto dest*: ∗)
thus $l \in$ *?f* ‘ $(\bigcup l \in \Lambda.\ A\ l)$ **by** *auto*
qed
then obtain g **where** g ‘ $\Lambda \subseteq (\bigcup l \in \Lambda.\ A\ l)$ **and** *inj-on* $g\ \Lambda$
by (*elim cor-inj-on-iff-surj-on-b*)
thus *?thesis* **by** *auto*
qed

lemma *AC-E-ext*:
assumes $\bigwedge l.\ l \in \Lambda \Longrightarrow A\ l \neq \{\}$
obtains a **where** $a \in (\Pi_E\ l \in \Lambda.\ A\ l)$
proof −
from *assms* **obtain** a **where** a: $a \in (\Pi\ l \in \Lambda.\ A\ l)$ **by** (*elim AC-E*)
let *?a'* = $\lambda l.$ *if* $l \in \Lambda$ *then* $a\ l$ *else undefined*
{
fix l

```
    assume l ∈ Λ
    with a have ?a' l ∈ A l by auto
  }
  moreover {
    fix l
    assume l ∉ Λ
    hence ?a' l = undefined by simp
  }
  ultimately have ?a' ∈ (Π_E l ∈ Λ. A l) by blast
  thus thesis by (fact that)
qed

proposition prob-2-1-6:
  assumes ⋀l. l ∈ Λ ⟹ ∃ a ∈ A l. ∃ b ∈ A l. a ≠ b
  shows |Λ| ≤o |Π_E l ∈ Λ. A l|
proof −
  {
    fix l
    assume l ∈ Λ
    with assms have A l ≠ {} by auto
  }
  then obtain a where a ∈ (Π_E l ∈ Λ. A l)  by (elim AC-E-ext)
  {
    fix l
    assume l ∈ Λ
    with assms have A l − {a l} ≠ {} by auto
  }
  then obtain b where b ∈ (Π_E l ∈ Λ. A l − {a l}) by (elim AC-E-ext)
  let ?f = λl. λl'. if l' = l then a l' else b l'
  have ?f ` Λ ⊆ (Π_E l ∈ Λ. A l)
  proof (rule subsetI)
    fix c
    assume c ∈ ?f ` Λ
    then obtain l where l ∈ Λ and c = ?f l by auto
    {
      fix l'
      assume l' ∈ Λ
      from ‹c = ?f l› have c l' = ?f l l' by simp
      {
        assume l' = l
        hence ?f l l' = a l' by simp
        also from this and ‹a ∈ (Π_E l ∈ Λ. A l)› and ‹l' ∈ Λ› have a l' ∈ A l' by auto
        finally have ?f l l' ∈ A l' .
      }
      moreover {
```

assume $l' \neq l$
hence *?f* $l\ l' = b\ l'$ **by** *simp*
also from *this* **and** ‹$b \in (\Pi_E\ l \in \Lambda.\ A\ l - \{a\ l\})$› **and** ‹$l' \in \Lambda$› **have** $b\ l' \in A\ l'$ **by** *auto*
finally have *?f* $l\ l' \in A\ l'$ **.**
}
ultimately have *?f* $l\ l' \in A\ l'$ **by** *simp*
with ‹$c\ l' =$ *?f* $l\ l'$› **have** $c\ l' \in A\ l'$ **by** *simp*
}
moreover {
fix l'
assume $l' \notin \Lambda$
with ‹$l \in \Lambda$› **have** $l' \neq l$ **by** *auto*
from ‹$c =$ *?f* l› **have** $c\ l' =$ *?f* $l\ l'$ **by** *simp*
also from ‹$l' \neq l$› **have** *?f* $l\ l' = b\ l'$ **by** *simp*
also from ‹$l' \notin \Lambda$› **and** ‹$b \in (\Pi_E\ l \in \Lambda.\ A\ l - \{a\ l\})$› **have** $b\ l' =$ *undefined* **by** *auto*
finally have $c\ l' =$ *undefined* **.**
}
ultimately show $c \in (\Pi_E\ l \in \Lambda.\ A\ l)$ **by** (*intro PiE-I*)
qed
moreover have *inj-on ?f* Λ
proof (*rule inj-onI*)
fix l **and** l'
assume $l \in \Lambda$ **and**
$l' \in \Lambda$ **and**
?f $l =$ *?f* l'
{
assume $l \neq l'$
hence *?f* $l'\ l = b\ l$ **by** *simp*
moreover have *?f* $l\ l = a\ l$ **by** *simp*
moreover note ‹*?f* $l =$ *?f* l'›
ultimately have $a\ l = b\ l$ **by** *metis*
also from ‹$l \in \Lambda$› **and** ‹$b \in (\Pi_E\ l \in \Lambda.\ A\ l - \{a\ l\})$› **have** $b\ l \in A\ l - \{a\ l\}$ **by** *blast*
finally have $a\ l \in A\ l - \{a\ l\}$ **.**
hence *False* **by** *simp*
}
thus $l = l'$ **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed

proposition *prob-2-1-7*:
assumes f ‘ $A = B$
obtains R **where** *equiv* $A\ R$ **and** *equipotent* $B\ (A\ //\ R)$
proof $-$
from *assms* **have** f ‘ $A \subseteq B$ **by** *simp*

```
  then obtain g where bij-betw g (A // (equiv-kernel-on f A)) (f ` A)
    by (fastforce elim: prop-1-6-4)
  with assms have bij-betw g (A // (equiv-kernel-on f A)) B by blast
  hence equipotent (A // equiv-kernel-on f A) B by auto
  hence equipotent B (A // equiv-kernel-on f A) by (fact prop-2-1-2)
  moreover have equiv A (equiv-kernel-on f A) by (fact equiv-equiv-kernel-on)
  ultimately show thesis by (intro that)
qed

end

end
theory Section-2-2
  imports Complex-Main
    Split-Pair
    Section-2-1
begin

context includes cardinal-syntax begin

fun dc-seq :: 'a set ⇒ ('a set ⇒ 'a) ⇒ nat ⇒ 'a set where
dc-seq M a 0 = {} |
dc-seq M a (Suc n) = dc-seq M a n ∪ {a (M − dc-seq M a n)}

lemma finite-dc-seq:
  shows finite (dc-seq M a n)
proof (induct n)
  case 0
  show ?case by simp
next
  case (Suc n)
  from Suc.hyps show ?case by simp
qed

lemma strict-dec-induct [consumes 1, case-names base step]:
  assumes i < j
    and P (Suc i)
    and ⋀j. i < j ⟹ P j ⟹ P (Suc j)
  shows P j
proof −
  let ?P = λn. P (n + i + 1)
  have ?P n for n
  proof (induct n)
```

```
    case 0
    from assms(2) show ?P 0 by simp
  next
    case (Suc n)
    note ‹?P n›
    from Suc.hyps have P (n + i + 1) by simp
    moreover have i < n + i + 1 by simp
    ultimately have P (Suc (n + i + 1)) by (intro assms(3))
    thus ?P (Suc n) by simp
  qed
  from assms(1) obtain k where j = Suc (i + k) by (auto dest: less-imp-Suc-add)
  hence j = k + i + 1 by simp
  with ‹?P k› show P j by simp
qed

theorem thm-2-4:
  assumes infinite M
  obtains A where A ⊆ M and |A| =o ℵ0
proof −
  let ?𝔐 = Pow M − {{}}
  {
    fix A
    assume A ∈ ?𝔐
    hence A ≠ {} by simp
  }
  then obtain a where a ∈ (Π A ∈ ?𝔐. A) by (elim AC-E)
  have ∗: dc-seq M a n ⊂ M for n
  proof (induct n)
    case 0
    from assms have M ≠ {} by auto
    thus ?case by auto
  next
    case (Suc n)
    from Suc.hyps have M − dc-seq M a n ≠ {} by auto
    moreover have M − dc-seq M a n ⊆ M by auto
    ultimately have M − dc-seq M a n ∈ ?𝔐 by simp
    with ‹a ∈ (Π A ∈ ?𝔐. A)› have a (M − dc-seq M a n) ∈ M − dc-seq M a n by (fact Pi-mem)
    also have M − dc-seq M a n ⊆ M by auto
    finally have a (M − dc-seq M a n) ∈ M .
    moreover note Suc.hyps
    ultimately have dc-seq M a (Suc n) ⊆ M by simp
    moreover from assms and finite-dc-seq have dc-seq M a (Suc n) ≠ M by metis
    ultimately show ?case by auto
  qed
  hence M − dc-seq M a n ∈ ?𝔐 for n by auto
```

```
let ?A = ⋃n. dc-seq M a n
let ?f = λn. a (M − dc-seq M a n)
have **: ?f n ∈ M − dc-seq M a n for n
  using ‹a ∈ (Π A ∈ ?𝔐. A)› and ‹M − dc-seq M a n ∈ ?𝔐› by (fact Pi-mem)
have ***: n < n' ⟹ ?f n ∈ dc-seq M a n' for n and n'
proof (induct n' rule: strict-dec-induct)
  case base
  show a (M − dc-seq M a n) ∈ dc-seq M a (Suc n) by simp
next
  case (step n')
  from step.hyps(2) show a (M − dc-seq M a n) ∈ dc-seq M a (Suc n') by simp
qed
have bij-betw ?f (UNIV :: nat set) ?A
proof (rule bij-betw-imageI)
  show inj-on ?f (UNIV :: nat set)
  proof (rule inj-onI)
    fix n and n'
    assume ?f n = ?f n'
    {
      assume n < n'
      with *** have ?f n ∈ dc-seq M a n' by simp
      moreover from ** have ?f n' ∉ dc-seq M a n' by simp
      moreover note ‹?f n = ?f n'›
      ultimately have False by simp
    }
    moreover {
      assume n' < n
      with *** have ?f n' ∈ dc-seq M a n by simp
      moreover from ** have ?f n ∉ dc-seq M a n by simp
      moreover note ‹?f n = ?f n'›
      ultimately have False by simp
    }
    ultimately show n = n' by fastforce
  qed
  show ?f ` UNIV = ?A
  proof (rule surj-onI)
    fix n
    have ?f n ∈ dc-seq M a n ∪ {?f n} by simp
    hence ?f n ∈ dc-seq M a (Suc n) by simp
    thus ?f n ∈ ?A by blast
  next
    fix x
    assume x ∈ ?A
    then obtain n where x ∈ dc-seq M a n by auto
    {
```

assume $\forall n.\ ?f\ n \neq x$
have $x \notin$ *dc-seq M a n′* **for** n'
proof (*induct n′*)
case *0*
show *?case* **by** *simp*
next
case (*Suc n*)
from *Suc.hyps* **and** ⟨$\forall n.\ ?f\ n \neq x$⟩ **show** *?case* **by** *auto*
qed
with ⟨$x \in$ *dc-seq M a n*⟩ **have** *False* **by** *simp*
}
thus $x \in ?f$ ‘ *UNIV* **by** *auto*
qed
qed
hence $\aleph_0 =o\ |?A|$ **by** *auto*
hence $|?A| =o\ \aleph_0$ **by** (*fact card-eq-sym*)
moreover from $*$ **have** $?A \subseteq M$ **by** *auto*
ultimately show *thesis* **by** (*intro that*)
qed

corollary *cor-infinite-imp-card-leq-aleph-zero*:
assumes *infinite M*
shows $\aleph_0 \leq o\ |M|$
proof −
from *assms* **obtain** A **where** $A \subseteq M$ **and** $|A| =o\ \aleph_0$ **by** (*elim thm-2-4*)
from *this*(*2*) **have** $\aleph_0 =o\ |A|$ **by** (*fact card-eq-sym*)
moreover from ⟨$A \subseteq M$⟩ **have** $|A| \leq o\ |M|$ **by** *auto*
ultimately show $\aleph_0 \leq o\ |M|$ **by** (*fact card-eq-card-leq-trans*)
qed

lemma *nat-Times-nat-card-eq-aleph-zero*:
shows $|(UNIV :: nat\ set) \times (UNIV :: nat\ set)| =o\ \aleph_0$
using *ex-2-3* **by** *auto*

theorem *thm-2-5-1-a*:
assumes $|A| \leq o\ \aleph_0$
and $|B| \leq o\ \aleph_0$
shows $|A \times B| \leq o\ \aleph_0$
proof −
from *assms*(*1*) **obtain** f **where** f ‘ $A \subseteq (UNIV :: nat\ set)$ **and** *inj-on f A* **by** *auto*
from *assms*(*2*) **obtain** g **where** g ‘ $B \subseteq (UNIV :: nat\ set)$ **and** *inj-on g B* **by** *auto*
from ⟨f ‘ $A \subseteq UNIV$⟩ **and** ⟨g ‘ $B \subseteq UNIV$⟩ **have** *map-prod f g* ‘ $(A \times B) \subseteq UNIV \times UNIV$ **by** *simp*
moreover from ⟨*inj-on f A*⟩ **and** ⟨*inj-on g B*⟩ **have** *inj-on* (*map-prod f g*) $(A \times B)$
by (*rule map-prod-inj-on*)

ultimately have $|A \times B| \leq o\ |(UNIV :: nat\ set) \times (UNIV :: nat\ set)|$ **by** *auto*
moreover note *nat-Times-nat-card-eq-aleph-zero*
ultimately show *?thesis* **by** (*fact card-leq-card-eq-trans*)
qed

lemma *thm-2-5-1-b-a*:
assumes $|A| =o\ \aleph_0$
and $|B| \leq o\ \aleph_0$
and $B \neq \{\}$
shows $|A \times B| =o\ \aleph_0$
proof −
from *assms*(*1*) **have** $\aleph_0 =o\ |A|$ **by** (*fact card-eq-sym*)
then obtain f **where** *bij-betw* f (*UNIV* :: *nat set*) A **by** *auto*
from *assms*(*3*) **obtain** b **where** $b \in B$ **by** *auto*
let $?g = \lambda n.\ (f\ n,\ b)$
have $?g$ ‘ $UNIV \subseteq A \times B$
proof (*rule image-subsetI*)
fix n
from ‹*bij-betw f UNIV A*› **have** $f\ n \in A$ **by** *auto*
with ‹$b \in B$› **show** $?g\ n \in A \times B$ **by** *simp*
qed
moreover have *inj-on ?g UNIV*
proof (*rule inj-onI*)
fix $m\ n$
assume $(f\ m,\ b) = (f\ n,\ b)$
hence $f\ m = f\ n$ **by** *simp*
moreover from ‹*bij-betw f UNIV A*› **have** *inj-on f UNIV* **by** *auto*
ultimately show $m = n$ **by** (*auto dest*: *injD*)
qed
ultimately have $\aleph_0 \leq o\ |A \times B|$ **by** *auto*
from *assms*(*1*) **have** $|A| \leq o\ \aleph_0$ **by** *blast*
with *assms*(*2*) **have** $|A \times B| \leq o\ \aleph_0$ **by** (*intro thm-2-5-1-a*)
with ‹$\aleph_0 \leq o\ |A \times B|$› **show** *?thesis* **by** (*intro thm-2-3-2*)
qed

lemma *Times-card-commute*:
fixes A :: *'a set*
shows $|A \times B| =o\ |B \times A|$
proof −
have *prod.swap* $\in A \times B \to B \times A$ **by** *auto*
moreover have *prod.swap* $\in B \times A \to A \times B$ **by** *auto*
moreover {
fix x
assume $x \in A \times B$
have *prod.swap* (*prod.swap* x) $= x$ **by** *simp*

}
moreover {
fix x
assume $x \in B \times A$
have *prod.swap* (*prod.swap* x) $= x$ **by** *simp*
}
ultimately have *bij-betw prod.swap* $(A \times B)$ $(B \times A)$ **by** (*fact bij-betwI*)
thus *?thesis* **by** *auto*
qed

lemma *thm-2-5-1-b-b*:
assumes $|A| \leq o\ \aleph_0$
and $A \neq \{\}$
and $|B| =o\ \aleph_0$
shows $|A \times B| =o\ \aleph_0$
proof –
have $|A \times B| =o\ |B \times A|$ **by** (*fact Times-card-commute*)
moreover from *assms* **have** $|B \times A| =o\ \aleph_0$ **by** (*intro thm-2-5-1-b-a*)
ultimately show *?thesis* **by** (*fact card-eq-trans*)
qed

theorem *thm-2-5-1-b*:
assumes $|A| \leq o\ \aleph_0$
and $A \neq \{\}$
and $|B| \leq o\ \aleph_0$
and $B \neq \{\}$
and $|A| =o\ \aleph_0 \vee |B| =o\ \aleph_0$
shows $|A \times B| =o\ \aleph_0$
using *assms thm-2-5-1-b-a thm-2-5-1-b-b* **by** *metis*

lemma *aleph-zero-Times-aleph-zero*:
assumes $|A| =o\ \aleph_0$
and $|B| =o\ \aleph_0$
shows $|A \times B| =o\ \aleph_0$
proof –
from *assms(2)* **have** $|B| \leq o\ \aleph_0$ **and** $B \neq \{\}$ **by** *blast+*
with *assms* **show** *?thesis* **by** (*blast intro*: *thm-2-5-1-b-a*)
qed

lemma *card-leq-imp-surj-on*:
assumes $|A| \leq o\ |B|$
and $A = \{\} \Longrightarrow B = \{\}$
obtains g **where** g ‘ $B = A$
proof –
from *assms* **obtain** f **where** f ‘ $A \subseteq B$ **and** *inj-on* f A **by** *auto*

```
  with assms(2) obtain g where g ` B = A by(elim cor-inj-on-iff-surj-on-a)
  thus thesis by (fact that)
qed

lemma surj-on-imp-card-leq:
  assumes f ` A = B
  shows |B| ≤o |A|
proof −
  from assms obtain g where g ` B ⊆ A and inj-on g B by (elim cor-inj-on-iff-surj-on-b)
  thus ?thesis by auto
qed

theorem thm-2-5-2-a:
  fixes Λ :: 'b set
    and A :: 'b ⇒ 'a set
  assumes Λ ≠ {}
    and ⋀l. l ∈ Λ ⟹ |A l| ≤o ℵ0
    and |Λ| ≤o ℵ0
  shows |⋃l ∈ Λ. A l| ≤o ℵ0
proof −
  let ?A = ⋃l ∈ Λ. A l
  {
    assume ?A = {}
    hence ?thesis by auto
  }
  moreover {
    assume ?A ≠ {}
    then obtain a0 where a0 ∈ ?A by blast
    {
      fix l
      assume l ∈ Λ
      {
        let ?g = λn :: nat. a0
        assume A l = {}
        hence A l ⊆ ?g ` UNIV by simp
        moreover from ‹a0 ∈ ?A› have ?g ` UNIV ⊆ ?A by auto
        ultimately have A l ⊆ ?g ` UNIV ∧ ?g ` UNIV ⊆ ?A ..
        hence {g :: nat ⇒ 'a. A l ⊆ g ` UNIV ∧ g ` UNIV ⊆ ?A} ≠ {} by fast
      }
      moreover {
        assume A l ≠ {}
        with ‹l ∈ Λ› assms(2) obtain g :: nat ⇒ 'a where g ` UNIV = A l
          by (metis card-leq-imp-surj-on)
        moreover from this and ‹l ∈ Λ› have g ` UNIV ⊆ ?A by auto
        ultimately have A l ⊆ g ` UNIV ∧ g ` UNIV ⊆ ?A by simp
```

        hence $\{g :: nat \Rightarrow 'a.\ A\ l \subseteq g\ `\ UNIV \wedge g\ `\ UNIV \subseteq\ ?A\} \neq \{\}$ **by** *auto*
      }
      ultimately have $\{g :: nat \Rightarrow 'a.\ A\ l \subseteq g\ `\ UNIV \wedge g\ `\ UNIV \subseteq\ ?A\} \neq \{\}$ **by** *blast*
    }
    then obtain $f :: 'b \Rightarrow nat \Rightarrow 'a$
      where f: $f \in (\Pi\ l \in \Lambda.\ \{g.\ A\ l \subseteq g\ `\ UNIV \wedge g\ `\ UNIV \subseteq\ ?A\})$ **by** (*elim AC-E*)
    let $?\varphi = \lambda(l,\ n).\ f\ l\ n$
    have $?\varphi\ `\ (\Lambda \times UNIV) =\ ?A$
    proof (*rule surj-onI*; *split-pair*)
      fix l **and** $n :: nat$
      assume $(l,\ n) \in \Lambda \times UNIV$
      hence $l \in \Lambda$ **by** *simp*
      with f **have** $f\ l\ `\ UNIV \subseteq\ ?A$ **by** *fast*
      thus $f\ l\ n \in\ ?A$ **by** *auto*
    next
      fix a
      assume $a \in\ ?A$
      then obtain l **where** $l \in \Lambda$ **and** $a \in A\ l$ **by** *auto*
      with f **obtain** n **where** $f\ l\ n = a$ **by** *auto*
      with $\langle l \in \Lambda \rangle$ **show** $a \in\ ?\varphi\ `\ (\Lambda \times UNIV)$ **by** *auto*
    qed
    hence $|?A| \leq o\ |\Lambda \times (UNIV :: nat\ set)|$ **by** (*fact surj-on-imp-card-leq*)
    also from *assms*(*3*) **have** $|\Lambda \times (UNIV :: nat\ set)| \leq o\ \aleph_0$ **by** (*auto elim*: *thm-2-5-1-a*)
    finally have *?thesis* **.**
  }
  ultimately show *?thesis* **by** *blast*
qed

theorem *thm-2-5-2-b*:
  fixes $\Lambda :: 'b\ set$
    and $A :: 'b \Rightarrow 'a\ set$
  assumes $\Lambda \neq \{\}$
    and $\bigwedge l.\ l \in \Lambda \Longrightarrow |A\ l| \leq o\ \aleph_0$
    and $|\Lambda| \leq o\ \aleph_0$
    and $l \in \Lambda$
    and $|A\ l| =o\ \aleph_0$
  shows $|\bigcup l \in \Lambda.\ A\ l| =o\ \aleph_0$
proof $-$
  let $?A = \bigcup l \in \Lambda.\ A\ l$
  from *assms*(*5*) **have** $\aleph_0 =o\ |A\ l|$ **by** (*fact ordIso-symmetric*)
  then obtain $f :: nat \Rightarrow 'a$ **where** $*$: *bij-betw* f *UNIV* $(A\ l)$ **by** *blast*
  with *assms*(*4*) **have** $f\ `\ UNIV \subseteq\ ?A$ **by** *auto*
  moreover from $*$ **have** *inj-on* f *UNIV* **by** *auto*
  ultimately have $\aleph_0 \leq o\ |?A|$ **by** *auto*
  moreover from *assms*(*1*$-$*3*) **have** $|?A| \leq o\ \aleph_0$ **by** (*rule thm-2-5-2-a*)

ultimately show *?thesis* **by** (*intro thm-2-3-2*)
qed

lemma *countable-Un-aleph-zero*:
  assumes $|A| \le o\ \aleph_0$
    and $|B| =o\ \aleph_0$
  shows $|A \cup B| =o\ \aleph_0$
proof –
  let $?\Lambda = \{0 :: nat,\ 1\}$
  let $?A = (\lambda l :: nat.\ undefined)(0 := A,\ 1 := B)$
  have $?\Lambda \neq \{\}$ **by** *simp*
  moreover have $\bigwedge l.\ l \in\ ?\Lambda \Longrightarrow |?A\ l| \le o\ \aleph_0$
  proof –
    fix l
    assume $l \in\ ?\Lambda$
    moreover {
      assume $l = 0$
      with *assms*(*1*) **have** $|?A\ l| \le o\ \aleph_0$ **by** *simp*
    }
    moreover {
      assume $l = 1$
      from *assms*(*2*) **have** $|?A\ 1| \le o\ \aleph_0$ **by** *fastforce*
      with ‹$l = 1$› **have** $|?A\ l| \le o\ \aleph_0$ **by** *simp*
    }
    ultimately show $|?A\ l| \le o\ \aleph_0$ **by** *blast*
  qed
  moreover have $|?\Lambda| \le o\ \aleph_0$
  proof –
    have *inj-on id ?Λ* **by** *simp*
    thus *?thesis* **by** *blast*
  qed
  moreover have $1 \in\ ?\Lambda$ **by** *simp*
  moreover from *assms*(*2*) **have** $|?A\ 1| =o\ \aleph_0$ **by** *simp*
  ultimately have $|\bigcup l \in\ ?\Lambda.\ ?A\ l| =o\ \aleph_0$ **by** (*rule thm-2-5-2-b*)
  moreover have $(\bigcup l \in\ ?\Lambda.\ ?A\ l) = A \cup B$ **by** *simp*
  ultimately show *?thesis* **by** *argo*
qed

lemma *aleph-zero-Un-aleph-zero*:
  assumes $|A| =o\ \aleph_0$
    and $|B| =o\ \aleph_0$
  shows $|A \cup B| =o\ \aleph_0$
proof –
  from *assms*(*1*) **have** $|A| \le o\ \aleph_0$ **by** *fast*
  with *assms*(*2*) **show** *?thesis* **by** (*intro countable-Un-aleph-zero*)

qed

corollary *cor-card-int-eq-aleph-zero*:
  shows $|UNIV :: int\ set| =o\ \aleph_0$
proof −
  let $?A = \{x :: int.\ 0 \leq x\}$
  let $?B = \{x :: int.\ x \leq 0\}$
  have $|?A| =o\ \aleph_0$
  proof −
    let $?f = \lambda n :: nat.\ (of\text{-}nat\ n) :: int$
    have $?f$ ‘ $UNIV = ?A$
    proof (*rule surj-onI*)
      fix n
      show $int\ n \in ?A$ **by** *simp*
    next
      fix n
      assume $n \in ?A$
      hence $0 \leq n$ **by** *simp*
      then obtain m **where** $n = ?f\ m$ **by** (*elim nonneg-int-cases*)
      thus $n \in ?f$ ‘ $UNIV$ **by** *simp*
    qed
    moreover have *inj-on ?f UNIV* **by** (*fact inj-of-nat*)
    ultimately have *bij-betw ?f UNIV ?A* **by** (*intro bij-betw-imageI*)
    hence $\aleph_0 =o\ |?A|$ **by** *blast*
    thus *?thesis* **by** (*fact ordIso-symmetric*)
  qed
  moreover have $|?B| =o\ \aleph_0$
  proof −
    let $?f = \lambda n :: nat.\ -((of\text{-}nat\ n) :: int)$
    have $?f$ ‘ $UNIV = ?B$
    proof (*rule surj-onI*)
      fix n
      show $?f\ n \in ?B$ **by** *simp*
    next
      fix n
      assume $n \in ?B$
      hence $0 \leq -n$ **by** *simp*
      then obtain $m :: nat$ **where** $-n = m$ **by** (*elim nonneg-int-cases*)
      hence $-int\ m = n$ **by** *simp*
      thus $n \in ?f$ ‘ $UNIV$ **by** *auto*
    qed
    moreover have *inj-on ?f UNIV* **by** (*simp add*: *inj-on-def*)
    ultimately have *bij-betw ?f UNIV ?B* **by** (*intro bij-betw-imageI*)
    hence $\aleph_0 =o\ |?B|$ **by** *auto*
    thus *?thesis* **by** (*fact ordIso-symmetric*)

```
  qed
  ultimately have |?A ∪ ?B| =o ℵ0 by (rule aleph-zero-Un-aleph-zero)
  moreover have ?A ∪ ?B = UNIV by auto
  ultimately show ?thesis by simp
qed

corollary cor-card-rat-eq-aleph-zero:
  shows |UNIV :: rat set| =o ℵ0
proof −
  let ?f = λ(a, b). Fract a b
  have ?f ‘ (UNIV × UNIV) = UNIV
  proof (rule surj-onI; split-pair)
    fix a b
    show Fract a b ∈ UNIV by simp
  next
    fix q
    obtain a b where q = Fract a b by (auto intro: Rat-cases)
    thus q ∈ ?f ‘ (UNIV × UNIV) by auto
  qed
  hence |UNIV :: rat set| ≤o |(UNIV :: int set) × (UNIV :: int set)|
    by (fact surj-on-imp-card-leq)
  also have |(UNIV :: int set) × (UNIV :: int set)| =o ℵ0
  proof −
    have |UNIV :: int set| =o ℵ0 by (fact cor-card-int-eq-aleph-zero)
    moreover from this have |UNIV :: int set| ≤o ℵ0 by fast
    ultimately show |(UNIV :: int set) × (UNIV :: int set)| =o ℵ0 by (blast intro: thm-2-5-1-b)
  qed
  finally have |UNIV :: rat set| ≤o ℵ0 .
  moreover have ℵ0 ≤o |UNIV :: rat set|
  proof −
    let ?g = λn :: nat. (of-nat n) :: rat
    have inj-on ?g UNIV by (auto intro: inj-of-nat)
    thus ?thesis by blast
  qed
  ultimately show ?thesis by (intro thm-2-3-2)
qed

theorem thm-2-6:
  assumes infinite A
    and B ⊆ A
    and |B| ≤o ℵ0
    and infinite (A − B)
  shows equipotent (A − B) A
proof −
  let ?A1 = A − B
```

from *assms*(*4*) **obtain** C **where** $C \subseteq ?A_1$ **and** $|C| =o \aleph_0$ **by** (*elim thm-2-4*)
let $?A_2 = ?A_1 - C$
from *assms*(*3*) **and** ‹$|C| =o \aleph_0$› **have** $|B \cup C| =o \aleph_0$ **by** (*rule countable-Un-aleph-zero*)
also from ‹$|C| =o \aleph_0$› **have** $\aleph_0 =o |C|$ **by** (*fact ordIso-symmetric*)
finally have $|B \cup C| =o |C|$ **.**
then obtain f_1 **where** *bij-betw* f_1 $(B \cup C)$ C **by** *auto*
let $?f = \lambda a.$ *if* $a \in ?A_2$ *then* a *else* f_1 a
have $?f$ ‘ $A = ?A_1$
proof (*rule surj-onI*)
  fix a
  assume $a \in A$
  {
    assume $a \in ?A_2$
    hence $?f\ a \in ?A_1$ **by** *simp*
  }
  moreover {
    assume $a \notin ?A_2$
    with ‹$a \in A$› **have** $a \in B \cup C$ **by** *simp*
    from ‹$a \notin ?A_2$› **have** $?f\ a = f_1$ a**by** *argo*
    also from ‹$a \in B \cup C$› **and** ‹*bij-betw* f_1 $(B \cup C)$ C› **have** $\ldots \in C$ **by** *auto*
    also from ‹$C \subseteq ?A_1$› **have** $\ldots \subseteq ?A_1$ **by** *simp*
    finally have $?f\ a \in ?A_1$ **.**
  }
  ultimately show $?f\ a \in ?A_1$ **by** *blast*
next
  fix b
  assume $b \in ?A_1$
  {
    assume $b \in C$
    with ‹*bij-betw* f_1 $(B \cup C)$ C› **obtain** a **where** $a \in B \cup C$ **and** $b = f_1$ a **by** *auto*
    from *this*(*1*) **have** $a \notin ?A_2$ **by** *simp*
    with ‹$b = f_1$ a› **have** $?f\ a = b$ **by** *argo*
    moreover from ‹$a \in B \cup C$› **and** *assms*(*2*) **and** ‹$C \subseteq ?A_1$› **have** $a \in A$ **by** *auto*
    ultimately have $\exists\, a \in A.\ ?f\ a = b$ **by** *blast*
  }
  moreover {
    assume $b \notin C$
    with ‹$b \in ?A_1$› **have** $b \in ?A_2$ **by** *simp*
    hence $?f\ b = b$ **by** *simp*
    moreover from ‹$b \in ?A_1$› **have** $b \in A$ **by** *simp*
    ultimately have $\exists\, a \in A.\ ?f\ a = b$ **by** *blast*
  }
  ultimately show $b \in ?f$ ‘ A **by** *blast*
qed
moreover have *inj-on* $?f$ A

```
  proof (rule inj-onI)
    fix a a′
    assume a ∈ A and a′ ∈ A and ?f a = ?f a′
    consider (A) a ∈ ?A₂ and a′ ∈ ?A₂
      | (B) a ∈ ?A₂ and a′ ∉ ?A₂
      | (C) a ∉ ?A₂ and a′ ∈ ?A₂
      | (D) a ∉ ?A₂ and a′ ∉ ?A₂ by argo
    thus a = a′
    proof cases
      case A
      with ‹?f a = ?f a′› show ?thesis by simp
    next
      case B
      from B(2) and ‹a′ ∈ A› have a′ ∈ B ∪ C by simp
      with ‹?f a = ?f a′› B have a = f₁ a′ by meson
      with ‹bij-betw f₁ (B ∪ C) C› and ‹a′ ∈ B ∪ C› and B(1) have False by fast
      thus ?thesis ..
    next
      case C
      from C(1) and ‹a ∈ A› have a ∈ B ∪ C by simp
      with ‹?f a = ?f a′› C have f₁ a = a′ by metis
      with ‹bij-betw f₁ (B ∪ C) C› and ‹a ∈ B ∪ C› and C(2) have False by auto
      thus ?thesis ..
    next
      case D
      with ‹?f a = ?f a′› have f₁ a = f₁ a′ by argo
      moreover from D and ‹a ∈ A› and ‹a′ ∈ A› have a ∈ B ∪ C and a′ ∈ B ∪ C by blast+
      moreover from ‹bij-betw f₁ (B ∪ C) C› have inj-on f₁ (B ∪ C) by auto
      ultimately show ?thesis by (elim inj-onD)
    qed
  qed
  ultimately have bij-betw ?f A ?A₁ by (intro bij-betw-imageI)
  hence equipotent A ?A₁ by auto
  thus ?thesis by (fact prop-2-1-2)
qed

corollary cor-2-1:
  assumes infinite A
    and |B| ≤o ℵ₀
  shows equipotent (A ∪ B) A
proof −
  have A: (A ∪ B) − (B − A) = A by auto
  from assms(1) have infinite (A ∪ B) by simp
  moreover have B − A ⊆ A ∪ B by auto
  moreover have |B − A| ≤o ℵ₀
```

proof −
from *assms(2)* **obtain** f **where** f ‘ $B \subseteq$ (*UNIV* :: *nat set*) **and** *inj-on* f B **by** *auto*
hence f ‘ $(B - A) \subseteq$ *UNIV* **and** *inj-on* f $(B - A)$ **by** (*auto dest*: *inj-on-diff*)
thus *?thesis* **by** *blast*
qed
moreover from *assms(1)* **and** A **have** *infinite* $((A \cup B) - (B - A))$ **by** *simp*
ultimately have *equipotent* $((A \cup B) - (B - A))$ $(A \cup B)$ **by** (*rule thm-2-6*)
with A **have** *equipotent* A $(A \cup B)$ **by** *simp*
thus *?thesis* **by** (*fact prop-2-1-2*)
qed

corollary *cor-dedekind-infinity*:
assumes *infinite* A
obtains B **where** $B \subset A$ **and** *equipotent* A B
proof −
from *assms* **obtain** a **where** $a \in A$ **by** *fastforce*
from *assms* **have** *infinite* $(A - \{a\})$ **by** *simp*
moreover have $|\{a\}| \leq o$ $\aleph_0$ **by** *fast*
ultimately have *equipotent* $((A - \{a\}) \cup \{a\})$ $(A - \{a\})$ **by** (*rule cor-2-1*)
moreover from $\langle a \in A \rangle$ **have** $(A - \{a\}) \cup \{a\} = A$ **by** *blast*
ultimately have *equipotent* A $(A - \{a\})$ **by** *simp*
moreover from $\langle a \in A \rangle$ **have** $A - \{a\} \subset A$ **by** *auto*
ultimately show *thesis* **by** (*intro that*)
qed

theorem *thm-2-8*:
shows $|M| <o$ $|$*Pow* $M|$
proof (*rule card-lessI*)
show $|M| \leq o$ $|$*Pow* $M|$
proof (*rule card-leqI*)
let *?f* $= \lambda a.\ \{a\}$
show *?f* ‘ $M \subseteq$ *Pow* M **by** *auto*
show *inj-on* *?f* M **by** *simp*
qed
show $\neg|M| =o$ $|$*Pow* $M|$
proof (*rule notI*)
assume $|M| =o$ $|$*Pow* $M|$
then obtain f **where** f: *bij-betw* f M (*Pow* M) **by** *auto*
let *?X* $= \{x \in M.\ x \notin f\ x\}$
have *?X* $\in$ *Pow* M **by** *auto*
moreover from f **have** f ‘ $M =$ *Pow* M **by** *auto*
ultimately have *?X* $\in f$ ‘ M **by** *simp*
then obtain x **where** $x \in M$ **and** $f\ x =$ *?X* **by** *auto*

{
  assume $x \in ?X$
  hence $x \notin f\, x$ **by** *simp*
  with ‹$f\, x = ?X$› **have** $x \notin ?X$ **by** *blast*
  with ‹$x \in ?X$› **have** *False* **by** *simp*
}
moreover {
  assume $x \notin ?X$
  with ‹$x \in M$› **have** $x \in f\, x$ **by** *simp*
  with ‹$f\, x = ?X$› **have** $x \in ?X$ **by** *blast*
  with ‹$x \notin ?X$› **have** *False* **by** *simp*
}
ultimately show *False* **by** *auto*
qed
qed

2.1.5 Problems

proposition *prob-2-2-1*:
  assumes $|A| =o\ \aleph_0$
    and $B \subseteq A$
    and *infinite* B
  shows $|B| =o\ \aleph_0$
proof –
  from *assms*(*3*) **have** $\aleph_0 \le o\ |B|$ **by** (*fact cor-infinite-imp-card-leq-aleph-zero*)
  moreover have $|B| \le o\ \aleph_0$
  proof –
    from *assms*(*2*) **obtain** f **where** *f0*: f ‘ $B \subseteq A$ **and** *f1*: *inj-on* f B **by** *fastforce*
    from *assms*(*1*) **obtain** g **where** *g0*: g ‘ $A \subseteq$ (*UNIV* :: *nat set*) **and** *g1*: *inj-on* g A **by** *fast*
    from *f0* **and** *g0* **have** $(g \circ f)$ ‘ $B \subseteq$ *UNIV* **by** *simp*
    moreover from *f0* **and** *f1* **and** *g1* **have** *inj-on* $(g \circ f)$ B **by** (*blast dest*: *thm-1-5-b*)
    ultimately show *?thesis* **by** *auto*
  qed
  ultimately show *?thesis* **by** (*intro thm-2-3-2*)
qed

lemma *disjoint-family-onI*:
  assumes $\bigwedge i\ j.\ i \in I \Longrightarrow j \in I \Longrightarrow i \neq j \Longrightarrow A\ i \cap A\ j = \{\}$
  shows *disjoint-family-on* A I
  using *assms* **unfolding** *disjoint-family-on-def* **by** *simp*

proposition *prob-2-2-2*:
  fixes A :: *'a set*
  assumes $|A| =o\ \aleph_0$
  obtains A' :: *nat* ⇒ *'a set* **where** $\bigwedge n.\ |A'\ n| =o\ \aleph_0$

and $(\bigcup n.\ A'\ n) = A$
and *disjoint-family-on* A' *UNIV*
proof $-$
have $|(UNIV :: nat\ set) \times (UNIV :: nat\ set)| =o \aleph_0$ **by** (*fact nat-Times-nat-card-eq-aleph-zero*)
also from *assms* **have** $\aleph_0 =o |A|$ **by** (*fact ordIso-symmetric*)
finally have $|(UNIV :: nat\ set) \times (UNIV :: nat\ set)| =o |A|$ **.**
then obtain $f :: nat \times nat \Rightarrow 'a$ **where** f: *bij-betw* f $(UNIV \times UNIV)$ A **by** *auto*
hence *inj-f*: *inj* f **by** *auto*
let $?A' = \lambda n :: nat.\ f\ \text{‘}\ (\{n\} \times UNIV)$
have $\bigwedge n.\ |?A'\ n| =o \aleph_0$
proof $-$
fix m
let $?g = \lambda a.\ THE\ n.\ f\ (m,\ n) = a$
have g: $?g\ a = n$ **if** $f\ (m,\ n) = a$ **for** n **and** a
proof $-$
{
fix n'
assume $f\ (m,\ n') = a$
with *that* **have** $f\ (m,\ n') = f\ (m,\ n)$ **by** *simp*
with *inj-f* **have** $n' = n$ **by** (*auto dest*: *injD*)
}
with *that* **show** *?thesis* **by** *blast*
qed
have $?g\ \text{‘}\ (?A'\ m) = UNIV$
proof (*rule surj-onI*)
fix a
assume $a \in\ ?A'\ m$
show $?g\ a \in UNIV$ **by** *simp*
next
fix n
have $f\ (m,\ n) \in\ ?A'\ m$ **by** *simp*
moreover have $?g\ (f\ (m,\ n)) = n$
proof (*rule the-equality*, *rule refl*)
fix n'
assume $f\ (m,\ n') = f\ (m,\ n)$
with *inj-f* **have** $(m,\ n') = (m,\ n)$ **by** (*auto dest*: *injD*)
thus $n' = n$ **by** *simp*
qed
ultimately show $n \in\ ?g\ \text{‘}\ ?A'\ m$ **by** *fastforce*
qed
moreover have *inj-on* $?g$ $(?A'\ m)$
proof (*rule inj-onI*)
fix a **and** a'
assume $a \in\ ?A'\ m$ **and** $a' \in\ ?A'\ m$ **and** $?g\ a =\ ?g\ a'$
from *this*(*1*,*2*) **obtain** n **and** n' **where** $f\ (m,\ n) = a$ **and** $f\ (m,\ n') = a'$ **by** *auto*

moreover from *this* **and** ‹$?g\ a = ?g\ a'$› **have** $n = n'$ **using** g **by** *simp*+
ultimately show $a = a'$ **by** *simp*
qed
ultimately show $|?A'\ m| =o\ \aleph_0$ **by** (*blast intro*: *bij-betw-imageI*)
qed
moreover have $(\bigcup n.\ ?A'\ n) = A$
proof (*rule set-eqI2*)
fix a
assume $a \in (\bigcup n.\ ?A'\ n)$
then obtain m **where** $a \in\ ?A'\ m$ **by** *auto*
then obtain n **where** $a = f\ (m, n)$ **by** *auto*
also from f **have** $f\ (m, n) \in A$ **by** *blast*
finally show $a \in A$ **.**
next
fix a
assume $a \in A$
with f **obtain** m **and** n **where** $a = f\ (m, n)$ **by** *blast*
hence $a \in\ ?A'\ m$ **by** *simp*
thus $a \in (\bigcup n.\ ?A'\ n)$ **by** *blast*
qed
moreover have *disjoint-family-on ?A′ UNIV*
proof (*rule disjoint-family-onI*)
fix $m\ m' :: nat$
assume $m \neq m'$
{
fix a
assume $a \in\ ?A'\ m$ **and** $a \in\ ?A'\ m'$
then obtain $n\ n' :: nat$ **where** $a = f\ (m, n)$ **and** $a = f\ (m', n')$ **by** *blast*
hence $f\ (m, n) = f\ (m', n')$ **by** *simp*
with *inj-f* **have** $(m, n) = (m', n')$ **by** (*auto dest*: *injD*)
with ‹$m \neq m'$› **have** *False* **by** *simp*
}
thus $?A'\ m \cap\ ?A'\ m' = \{\}$ **by** *fast*
qed
ultimately show *thesis* **by** (*fact that*)
qed

proposition *prob-2-2-3*:
defines QQ: $\mathfrak{Q} \equiv \{\{a :: rat <..< b\} \mid a\ b.\ a < b\}$
shows $|\mathfrak{Q}| =o\ \aleph_0$
proof −
have $|\mathfrak{Q}| \leq o\ \aleph_0$
proof −
let $?f = \lambda(a :: rat, b).\ \{a <..< b\}$
have $?f$ ‘ $(UNIV \times UNIV) = \mathfrak{Q} \cup \{\{\}\}$

proof (*rule surj-onI*; *split-pair*)
fix $a\ b :: rat$
{
assume $a < b$
hence $\{a <..< b\} \in \mathfrak{Q} \cup \{\{\}\}$ **unfolding** *QQ* **by** *auto*
}
moreover {
assume $b \leq a$
hence $\{a <..< b\} = \{\}$ **by** *simp*
hence $\{a <..< b\} \in \mathfrak{Q} \cup \{\{\}\}$ **by** *simp*
}
ultimately show $\{a <..< b\} \in \mathfrak{Q} \cup \{\{\}\}$ **by** *linarith*
next
fix Q
assume $Q \in \mathfrak{Q} \cup \{\{\}\}$
moreover {
assume $Q \in \mathfrak{Q}$
hence $\exists\, a\ b.\ \{a <..< b\} = Q$ **unfolding** *QQ* **by** *auto*
}
moreover {
assume $Q \in \{\{\}\}$
moreover have $\{0 :: rat <..< 0\} = \{\}$ **by** *simp*
ultimately have $\exists\, a\ b.\ \{a <..< b\} = Q$ **by** *auto*
}
ultimately show $Q \in ?f\ `\ (UNIV \times UNIV)$ **by** *fast*
qed
hence $|\mathfrak{Q} \cup \{\{\}\}| \leq o\ |(UNIV :: rat\ set) \times (UNIV :: rat\ set)|$ **by** (*fact surj-on-imp-card-leq*)
also have $|(UNIV :: rat\ set) \times (UNIV :: rat\ set)| =o\ \aleph_0$
proof −
have $|UNIV :: rat\ set| =o\ \aleph_0$ **by** (*fact cor-card-rat-eq-aleph-zero*)
thus *?thesis* **by** (*blast intro*: *aleph-zero-Times-aleph-zero*)
qed
finally have $|\mathfrak{Q} \cup \{\{\}\}| \leq o\ \aleph_0$ **.**
thus $|\mathfrak{Q}| \leq o\ \aleph_0$ **by** *fastforce*
qed
moreover have $\aleph_0 \leq o\ |\mathfrak{Q}|$
proof (*rule card-leqI*)
let $?f = \lambda n :: nat.\ \{(rat\text{-}of\text{-}nat\ n) - 1 <..< (of\text{-}nat\ n) + 1\}$
{
fix n
have $\{(rat\text{-}of\text{-}nat\ n) - 1 <..< (of\text{-}nat\ n) + 1\} \in \mathfrak{Q}$ **unfolding** *QQ* **by** *fastforce*
}
thus *range* $?f \subseteq \mathfrak{Q}$ **by** *auto*
{
fix $n\ n' :: nat$

```
      assume ?f n = ?f n′
      {
        assume n < n′
        hence of-nat n ∈ ?f n and of-nat n ∉ ?f n′ by simp+
        hence ?f n ≠ ?f n′ by blast
        with ‹?f n = ?f n′› have False by simp
      }
      moreover {
        assume n′ < n
        hence of-nat n ∈ ?f n and of-nat n ∉ ?f n′ by simp+
        hence ?f n ≠ ?f n′ by blast
        with ‹?f n = ?f n′› have False by simp
      }
      ultimately have n = n′ by fastforce
    }
    thus inj ?f by (fact injI)
  qed
  ultimately show ?thesis by (fact thm-2-3-2)
qed

lemmas [dest] = disjointD

proposition prob-2-2-4:
  assumes ⋀I. I ∈ ℑ ⟹ ∃ a b :: real. a < b ∧ I = {a <..< b}
    and disjoint ℑ
  shows |ℑ| ≤o ℵ₀
proof −
  let ?f = λI. {q :: rat. (of-rat q) ∈ I}
  {
    fix I
    assume I ∈ ℑ
    with assms(1) obtain a and b where a < b and I: I = {a <..< b} by blast
    from this(1) obtain q :: rat where a < of-rat q and of-rat q < b
      by (blast dest: of-rat-dense)
    with I have of-rat q ∈ I by simp
    hence ?f I ≠ {} by auto
  }
  moreover have disjoint-family-on ?f ℑ
  proof (rule disjoint-family-onI)
    fix I and J
    assume I ∈ ℑ and J ∈ ℑ and I ≠ J
    with assms(2) have I ∩ J = {} by blast
    from ‹I ∈ ℑ› and ‹J ∈ ℑ› obtain a b a′ b′
      where a < b and I: I = {a <..< b}
        and a′ < b′ and J: J = {a′ <..< b′} by (fast dest: assms(1))
```

```
    {
      assume a' < b ∧ a < b'
      with ‹a < b› and ‹a' < b'› have max a a' < min b b' by simp
      then obtain c where max a a' < c and c < min b b' by (blast dest: dense)
      with I and J have c ∈ I and c ∈ J by simp+
      with ‹I ∩ J = {}› have False by auto
    }
    hence b ≤ a' ∨ b' ≤ a by argo
    moreover {
      assume b ≤ a'
      with I and J have ?f I ∩ ?f J = {} by fastforce
    }
    moreover {
      assume b' ≤ a
      with I and J have ?f I ∩ ?f J = {} by fastforce
    }
    ultimately show ?f I ∩ ?f J = {} by linarith
  qed
  ultimately have |ℑ| ≤o |⋃I ∈ ℑ. {q :: rat. (of-rat q) ∈ I}| by (fact prob-2-1-5)
  also have |⋃I ∈ ℑ. {q :: rat. (of-rat q) ∈ I}| ≤o |UNIV :: rat set| by auto
  also have |UNIV :: rat set| =o ℵ0 by (fact cor-card-rat-eq-aleph-zero)
  finally show ?thesis .
qed

lemma fixed-length-lists-of-aleph-zero:
  fixes A :: 'a set
  assumes |A| =o ℵ0
    and 1 ≤ n
  defines XS: XS l ≡ {xs ∈ lists A. length xs = l}
  shows |XS n| =o ℵ0
using assms(2) proof (induct n rule: dec-induct)
  case base
  let ?f = λa. [a]
  have ?f ` A = XS 1
  proof (rule surj-onI)
    fix a
    assume a ∈ A
    thus ?f a ∈ XS 1 unfolding XS by simp
  next
    fix xs
    assume xs ∈ XS 1
    hence xs ∈ lists A and length xs = 1 unfolding XS by simp+
    from this(2) have length xs = Suc 0 by simp
    then obtain y and ys where xs = y # ys and length ys = 0 by (auto simp: length-Suc-conv)
    from this(2) have ys = [] by simp
```

with ‹$xs = y \# ys$› **have** $xs = [y]$ **by** *simp*
moreover from ‹$xs = y \# ys$› **and** ‹$xs \in lists\ A$› **have** $y \in A$ **by** *simp*
ultimately show $xs \in ?f\ `\ A$ **by** *simp*
qed
moreover have *inj-on ?f A*
proof (*rule inj-onI*)
fix $a\ a' :: 'a$
assume $?f\ a = ?f\ a'$
thus $a = a'$ **by** *simp*
qed
ultimately have *bij-betw ?f A* (*XS 1*) **by** (*intro bij-betw-imageI*)
hence $|A| =o\ |XS\ 1|$ **by** *auto*
hence $|XS\ 1| =o\ |A|$ **by** (*fact ordIso-symmetric*)
also note *assms*(*1*)
finally show *?case* **.**
next
case (*step n*)
let $?f = \lambda(xs, x).\ x \# xs$
have $?f\ `\ ((XS\ n) \times A) = XS\ (Suc\ n)$
proof (*rule surj-onI*; *split-pair*)
fix *xs* **and** *a*
assume $(xs, a) \in (XS\ n) \times A$
hence $xs \in XS\ n$ **and** $a \in A$ **by** *simp+*
from *this*(*1*) **have** $xs \in lists\ A$ **and** $length\ xs = n$ **unfolding** *XS* **by** *simp+*
from *this*(*1*) **and** ‹$a \in A$› **have** $a \# xs \in lists\ A$ **by** *simp*
moreover from ‹$length\ xs = n$› **have** $length\ (a \# xs) = Suc\ n$ **by** *simp*
ultimately show $a \# xs \in XS\ (Suc\ n)$ **unfolding** *XS* **by** *simp*
next
fix *xs*
assume $xs \in XS\ (Suc\ n)$
hence $xs \in lists\ A$ **and** $length\ xs = Suc\ n$ **unfolding** *XS* **by** *simp+*
from *this*(*2*) **obtain** *a* **and** *ys* **where** $xs = a \# ys$ **and** $length\ ys = n$
by (*auto simp*: *length-Suc-conv*)
from ‹$xs \in lists\ A$› **and** *this*(*1*) **have** $ys \in lists\ A$ **by** *simp*
with ‹$length\ ys = n$› **have** $ys \in XS\ n$ **unfolding** *XS* **by** *simp*
moreover from ‹$xs \in lists\ A$› **and** ‹$xs = a \# ys$› **have** $a \in A$ **by** *simp*
moreover note ‹$xs = a \# ys$›
ultimately show $xs \in ?f\ `\ (XS\ n \times A)$ **by** *auto*
qed
hence $|XS\ (Suc\ n)| \leq o\ |(XS\ n) \times A|$ **by** (*fact surj-on-imp-card-leq*)
also from *assms*(*1*) **and** *step.hyps* **have** $|(XS\ n) \times A| =o\ \aleph_0$
by (*intro aleph-zero-Times-aleph-zero*)
finally have $|XS\ (Suc\ n)| \leq o\ \aleph_0$ **.**
moreover have $\aleph_0 \leq o\ |XS\ (Suc\ n)|$
proof −

```
    from step.hyps(3) have ℵ0 =o |XS n| by (fact ordIso-symmetric)
    also have |XS n| ≤o |XS (Suc n)|
    proof −
      from assms(1) obtain a where a ∈ A by blast
      let ?f = λxs. a # xs
      have ?f ‘ (XS n) ⊆ XS (Suc n)
      proof (rule image-subsetI)
        fix xs
        assume xs ∈ XS n
        hence xs ∈ lists A and length xs = n unfolding XS by simp+
        from this(1) and ‹a ∈ A› have ?f xs ∈ lists A by simp
        moreover from ‹length xs = n› have length (?f xs) = Suc n by simp
        ultimately show ?f xs ∈ XS (Suc n) unfolding XS by simp
      qed
      moreover have inj-on ?f (XS n) by simp
      ultimately show ?thesis by blast
    qed
    finally show ?thesis .
  qed
  ultimately show ?case by (fact thm-2-3-2)
qed

lemma lists-aleph-zero-eq-aleph-zero:
  assumes |A| =o ℵ0
  shows |lists A| =o ℵ0
proof −
  let ?A′ = λn. {xs ∈ lists A. length xs = n}
  let ?B = ⋃i. ?A′ i
  have lists A ⊆ ?B
  proof (rule subsetI)
    fix xs
    assume xs ∈ lists A
    hence xs ∈ ?A′ (length xs) by simp
    thus xs ∈ ?B by blast
  qed
  hence |lists A| ≤o |?B| by fastforce
  also have |?B| =o ℵ0
  proof (rule thm-2-5-2-b; simp?)
    fix n :: nat
    {
      assume n = 0
      hence ?A′ n = {[]} by auto
      hence |?A′ n| ≤o ℵ0 by auto
    }
    moreover {
```

      assume $1 \leq n$
      with *assms* **have** $|?A'\ n| =o\ \aleph_0$ **by** (*fact fixed-length-lists-of-aleph-zero*)
      hence $|?A'\ n| \leq o\ \aleph_0$ **by** (*fact card-eq-imp-card-leq*)
    }
    ultimately show $|?A'\ n| \leq o\ \aleph_0$ **by** *linarith*
  next
    from *assms* **show** $|?A'\ 1| =o\ \aleph_0$ **by** (*blast intro*: *fixed-length-lists-of-aleph-zero*)
  qed
  finally have $|lists\ A| \leq o\ \aleph_0$ **by** *simp*
  moreover have $\aleph_0 \leq o\ |lists\ A|$
  proof −
    from *assms* **have** $|?A'\ 1| =o\ \aleph_0$ **by** (*blast intro*: *fixed-length-lists-of-aleph-zero*)
    hence $\aleph_0 =o\ |?A'\ 1|$ **by** (*fact ordIso-symmetric*)
    also have $|?A'\ 1| \leq o\ |lists\ A|$
    proof −
      have $?A'\ 1 \subseteq lists\ A$ **by** *auto*
      thus *?thesis* **by** *auto*
    qed
    finally show *?thesis* **.**
  qed
  ultimately show *?thesis* **by** (*fact thm-2-3-2*)
qed

proposition *prob-2-2-5*:
  assumes $|A| =o\ \aleph_0$
  defines AA: $\mathfrak{A} \equiv \{X.\ X \subseteq A \wedge finite\ X\}$
  shows $|\mathfrak{A}| =o\ \aleph_0$
proof −
  let $?A' = \lambda n.\ \{xs \in lists\ A.\ length\ xs = n\}$
  let $?B = \bigcup n.\ ?A'\ n$
  let $?f = \lambda xs.\ set\ xs$
  have $?f\ `\ ?B = \mathfrak{A}$
  proof (*rule surj-onI*)
    fix *xs*
    assume $xs \in ?B$
    then obtain n **where** $xs \in ?A'\ n$ **by** *simp*
    hence $xs \in lists\ A$ **by** *simp*
    hence $?f\ xs \subseteq A$ **by** *auto*
    moreover have *finite* $(?f\ xs)$ **by** *simp*
    ultimately show $?f\ xs \in \mathfrak{A}$ **unfolding** AA **by** *simp*
  next
    fix X
    assume $X \in \mathfrak{A}$
    hence $X \subseteq A$ **and** *finite* X **unfolding** AA **by** *simp+*
    then obtain *xs* **where** $xs \in lists\ A$ **and** $set\ xs = X$ **by** (*fast dest*: *finite-list*)

```
    from this(1) have xs ∈ ?A' (length xs) by simp
    hence xs ∈ ?B by simp
    with ‹set xs = X› show X ∈ ?f ` ?B by auto
  qed
  hence |𝔄| ≤o |?B| by (fact surj-on-imp-card-leq)
  also have |?B| ≤o ℵ0
  proof (rule thm-2-5-2-a; simp)
    fix n
    have ?A' n ⊆ lists A by auto
    hence |?A' n| ≤o |lists A| by auto
    also from assms(1) have |lists A| =o ℵ0 by (fact lists-aleph-zero-eq-aleph-zero)
    finally show |?A' n| ≤o ℵ0 .
  qed
  finally have |𝔄| ≤o ℵ0 .
  moreover have ℵ0 ≤o |𝔄|
  proof -
    from assms(1) have ℵ0 =o |A| by (fact ordIso-symmetric)
    also have |A| ≤o |𝔄|
    proof (rule card-leqI)
      let ?f = λa :: 'a. {a}
      {
        fix a
        assume a ∈ A
        hence ?f a ∈ 𝔄 unfolding AA by simp
      }
      thus ?f ` A ⊆ 𝔄 by auto
      show inj-on ?f A by simp
    qed
    finally show ?thesis .
  qed
  ultimately show ?thesis by (fact thm-2-3-2)
qed


end

end
theory Section-2-3
  imports Main
    HOL−Library.Disjoint-Sets
    HOL−Library.FuncSet
    Section-2-2
```

```
begin

context includes cardinal-syntax begin
```

2.2 3. Operations on Cardinalities

2.2.1 A) Sum and Product of Cardinalities

```
proposition csum-definition:
  fixes A :: 'a set
    and B :: 'b set
  shows |A| +c |B| = |A <+> B|
  unfolding csum-def by (simp only: Field-card-of)

proposition csum-welldefinedness:
  assumes |A| =o |A'|
    and |B| =o |B'|
  shows |A| +c |B| =o |A'| +c |B'|
proof −
  from assms(1) obtain f where f: bij-betw f A A' by auto
  hence inj-on f A and f ` A = A' by auto
  from assms(2) obtain g where g: bij-betw g B B' by auto
  hence inj-on g B and g ` B = B' by auto
  define h where h ≡ map-sum f g
  have bij-betw h (A <+> B) (A' <+> B')
  proof (rule bij-betw-imageI)
    show inj-on h (A <+> B)
    proof (rule inj-onI)
      fix x and x'
      assume x: x ∈ A <+> B
        and x': x' ∈ A <+> B
        and h: h x = h x'
      from x and x' consider (A) x ∈ Inl ` A and x' ∈ Inl ` A
        | (B) x ∈ Inl ` A and x' ∈ Inr ` B
        | (C) x ∈ Inr ` B and x' ∈ Inl ` A
        | (D) x ∈ Inr ` B and x' ∈ Inr ` B
        by blast
      thus x = x'
      proof cases
        case A
        then obtain a and a' where
          a ∈ A
          and Inl a = x
          and a' ∈ A
```

and *Inl* $a' = x'$ **by** *auto*
with *h-def* **have** $h\ x =$ *Inl* $(f\ a)$ **and** $h\ x' =$ *Inl* $(f\ a')$ **by** *auto*
with h **have** $f\ a = f\ a'$ **by** *simp*
with ‹$a \in A$› **and** ‹$a' \in A$› **and** ‹*inj-on* $f\ A$› **have** $a = a'$ **by** (*elim inj-onD*)
with ‹*Inl* $a = x$› **and** ‹*Inl* $a' = x'$› **show** *?thesis* **by** *simp*
next
case B
with *h-def* **and** h **have** *False* **by** *auto*
thus *?thesis* **..**
next
case C
with *h-def* **and** h **have** *False* **by** *auto*
thus *?thesis* **..**
next
case D
then obtain b **and** b' **where**
$b \in B$
and *Inr* $b = x$
and $b' \in B$
and *Inr* $b' = x'$ **by** *auto*
with *h-def* **have** $h\ x =$ *Inr* $(g\ b)$ **and** $h\ x' =$ *Inr* $(g\ b')$ **by** *auto*
with h **have** $g\ b = g\ b'$ **by** *simp*
with ‹$b \in B$› **and** ‹$b' \in B$› **and** ‹*inj-on* $g\ B$› **have** $b = b'$ **by** (*elim inj-onD*)
with ‹*Inr* $b = x$› **and** ‹*Inr* $b' = x'$› **show** *?thesis* **by** *simp*
qed
qed
next
show h ‘ $(A <+> B) = A' <+> B'$
proof (*rule surj-onI*)
fix x
assume $x \in A <+> B$
with *h-def* **and** ‹f ‘ $A = A'$› **and** ‹g ‘ $B = B'$› **show** $h\ x \in A' <+> B'$ **by** *auto*
next
fix x'
assume $x' \in A' <+> B'$
then consider (A) $x' \in$ *Inl* ‘ A'
| (B) $x' \in$ *Inr* ‘ B' **by** *auto*
thus $x' \in h$ ‘ $(A <+> B)$
proof *cases*
case A
then obtain a' **where** $a' \in A'$ **and** *Inl* $a' = x'$ **by** *blast*
from ‹$a' \in A'$› **and** ‹f ‘ $A = A'$› **obtain** a **where** $a \in A$ **and** $f\ a = a'$ **by** *auto*
with ‹*Inl* $a' = x'$› **and** *h-def* **have** $x' = h$ (*Inl* a) **by** *simp*
with ‹$a \in A$› **show** *?thesis* **by** *auto*
next

case B
then obtain b' **where** $b' \in B'$ **and** *Inr* $b' = x'$ **by** *blast*
from ‹$b' \in B'$› **and** ‹g ‘ $B = B'$› **obtain** b **where** $b \in B$ **and** $g\ b = b'$ **by** *auto*
with ‹*Inr* $b' = x'$› **and** *h-def* **have** $x' = h$ (*Inr* b) **by** *simp*
with ‹$b \in B$› **show** *?thesis* **by** *auto*
qed
qed
qed
hence $|A <+> B| =o |A' <+> B'|$ **by** *auto*
thus *?thesis* **unfolding** *csum-definition* **by** *simp*
qed

lemmas *csum-cong'* = *csum-welldefinedness*

lemma *csum-cong1'*:
assumes $|A| =o |A'|$
shows $|A| +c |B| =o |A'| +c |B|$
proof –
have $|B| =o |B|$ **by** *simp*
with *assms* **show** *?thesis* **by** (*intro csum-cong'*)
qed

lemma *csum-cong2'*:
assumes $|B| =o |B'|$
shows $|A| +c |B| =o |A| +c |B'|$
proof –
have $|A| =o |A|$ **by** *simp*
with *assms* **show** *?thesis* **by** (*intro csum-cong'*)
qed

primrec *sum-swap* **where**
sum-swap (*Inl a*) = *Inr a*
| *sum-swap* (*Inr b*) = *Inl b*

lemma *inj-on-sum-swap*:
shows *inj-on sum-swap* $(A <+> B)$
unfolding *inj-on-def* **by** *auto*

lemma *sum-swap-surj-on*:
shows *sum-swap* ‘ $(A <+> B) = B <+> A$
unfolding *sum-swap-def* **by** *force*

lemma *bij-betw-sum-swap*:
shows *bij-betw sum-swap* $(A <+> B)$ $(B <+> A)$
using *inj-on-sum-swap sum-swap-surj-on* **by** (*intro bij-betw-imageI*)

```
proposition prop-2-3-1:
  shows |A| +c |B| =o |B| +c |A|
proof −
  from bij-betw-sum-swap have |A <+> B| =o |B <+> A| by auto
  thus ?thesis unfolding csum-definition by simp
qed

fun sum-rotate :: ('a + 'b) + 'c ⇒ 'a + ('b + 'c) where
  sum-rotate (Inl (Inl a)) = Inl a
| sum-rotate (Inl (Inr b)) = Inr (Inl b)
| sum-rotate (Inr c) = Inr (Inr c)

lemma inj-on-sum-rotate:
  shows inj-on sum-rotate ((A <+> B) <+> C)
  unfolding inj-on-def by auto

lemma sum-rotate-surj-on:
  shows sum-rotate ‘ ((A <+> B) <+> C) = A <+> (B <+> C)
  by force

lemma bij-betw-sum-rotate:
  shows bij-betw sum-rotate ((A <+> B) <+> C) (A <+> (B <+> C))
  using inj-on-sum-rotate and sum-rotate-surj-on by (intro bij-betw-imageI)

proposition prop-2-3-2:
  shows ( |A| +c |B| ) +c |C| =o |A| +c ( |B| +c |C| )
proof −
  from bij-betw-sum-rotate have |(A <+> B) <+> C| =o |A <+> (B <+> C)| by auto
  thus ?thesis unfolding csum-definition by simp
qed

proposition prop-2-3-3:
  fixes A :: 'a set
  shows |A| +c czero =o |A|
proof −
  have |A| +c czero = |A| +c |{}| unfolding czero-def ..
  moreover have |A| +c |{}| = |A <+> {}| unfolding csum-definition ..
  moreover have |A <+> {}| =o |A|
  proof (rule card-eqI)
    have inj-on Inl A by simp
    moreover have Inl ‘ A = A <+> {} by auto
    ultimately have bij-betw Inl A (A <+> {}) by (intro bij-betw-imageI)
    thus bij-betw (the-inv-into A Inl) (A <+> {}) A by (intro bij-betw-the-inv-into)
  qed
```

```
  ultimately show ?thesis by metis
qed

proposition prop-2-3-4:
  assumes |A| ≤o |A′|
    and |B| ≤o |B′|
  shows |A| +c |B| ≤o |A′| +c |B′|
proof −
  from assms(1) obtain f where f0: f ‘ A ⊆ A′ and f1: inj-on f A ..
  from assms(2) obtain g where g0: g ‘ B ⊆ B′ and g1: inj-on g B ..
  define h where h ≡ map-sum f g
  have |A| +c |B| = |A <+> B| by (fact csum-definition)
  also have |A <+> B| ≤o |A′ <+> B′|
  proof (rule card-leqI)
    show h ‘ (A <+> B) ⊆ A′ <+> B′
    proof (rule image-subsetI)
      fix x
      assume x ∈ A <+> B
      then consider a where a ∈ A and x = Inl a
        | b where b ∈ B and x = Inr b by auto
      note this[where thesis = h x ∈ A′ <+> B′]
      thus h x ∈ A′ <+> B′ using f0 and g0 and h-def by auto
    qed
    moreover show inj-on h (A <+> B)
    proof (intro inj-onI)
      fix x y
      assume x ∈ A <+> B
        and y ∈ A <+> B
        and h x = h y
      from this(1,2) consider
        (A)a and s where a ∈ A and x = Inl a and s ∈ A and y = Inl s
        | (B) a and b where a ∈ A and x = Inl a and b ∈ B and y = Inr b
        | (C) b and a where b ∈ B and x = Inr b and a ∈ A and y = Inl a
        | (D) b and t where b ∈ B and x = Inr b and t ∈ B and y = Inr t by auto
      thus x = y
      proof cases
        case A
        from ‹x = Inl a› and ‹y = Inl s› have h x = Inl (f a) and h y = Inl (f s)
          unfolding h-def by simp+
        with ‹h x = h y› have f a = f s by simp
        with ‹a ∈ A› and ‹s ∈ A› and f1 have a = s by (elim inj-onD)
        with ‹x = Inl a› and ‹y = Inl s› show ?thesis by simp
      next
        case B
        from ‹x = Inl a› and ‹y = Inr b› have h x = Inl (f a) and h y = Inr (g b)
```

unfolding *h-def* **by** *simp+*
with ‹$h\ x = h\ y$› **have** *False* **by** *simp*
thus *?thesis* **..**
next
case C
from ‹$x = Inr\ b$› **and** ‹$y = Inl\ a$› **have** $h\ x = Inr\ (g\ b)$ **and** $h\ y = Inl\ (f\ a)$
unfolding *h-def* **by** *simp+*
with ‹$h\ x = h\ y$› **have** *False* **unfolding** *h-def* **by** *simp*
thus *?thesis* **..**
next
case D
from ‹$x = Inr\ b$› **and** ‹$y = Inr\ t$› **have** $h\ x = Inr\ (g\ b)$ **and** $h\ y = Inr\ (g\ t)$
unfolding *h-def* **by** *simp+*
with ‹$h\ x = h\ y$› **have** $g\ b = g\ t$ **by** *simp*
with ‹$b \in B$› **and** ‹$t \in B$› **and** *g1* **have** $b = t$ **by** (*elim inj-onD*)
with ‹$x = Inr\ b$› **and** ‹$y = Inr\ t$› **show** *?thesis* **by** *simp*
qed
qed
qed
also have $|A' <\!+\!> B'| = |A'| +c\ |B'|$ **unfolding** *csum-definition* **by** *simp*
finally show *?thesis* **.**
qed

lemma *union-card-leq-csum*:
shows $|A \cup B| \leq o\ |A| +c\ |B|$
proof $-$
define f **where** $f\ x \equiv$ *if* $x \in A$ *then* $Inl\ x$ *else* $Inr\ x$ **for** x
have f ‘ $(A \cup B) \subseteq A <\!+\!> B$
proof (*rule image-subsetI*)
fix x
assume $x \in A \cup B$
moreover {
assume $x \in A$
hence $f\ x \in A <\!+\!> B$ **unfolding** *f-def* **by** *auto*
}
moreover {
assume $x \in B$
hence $f\ x \in A <\!+\!> B$ **unfolding** *f-def* **by** *auto*
}
ultimately show $f\ x \in A <\!+\!> B$ **by** *auto*
qed
moreover have *inj-on* $f\ (A \cup B)$
proof (*rule inj-onI*)
fix x **and** y
assume $x \in A \cup B$

```
      and y ∈ A ∪ B
      and f: f x = f y
    from this(1,2) consider
      (A) x ∈ A and y ∈ A
      | (B) x ∈ A and y ∈ B − A
      | (C) x ∈ B − A and y ∈ A
      | (D) x ∈ B − A and y ∈ B − A
      by auto
    thus x = y
    proof cases
      case A
      with f show ?thesis unfolding f-def by simp
    next
      case B
      with f show ?thesis unfolding f-def by simp
    next
      case C
      with f show ?thesis unfolding f-def by simp
    next
      case D
      with f show ?thesis unfolding f-def by simp
    qed
  qed
  ultimately show ?thesis unfolding csum-definition by auto
qed

fun fun-disjoint-union-card-eq-csum where
  fun-disjoint-union-card-eq-csum (Inl x) = x
| fun-disjoint-union-card-eq-csum (Inr x) = x

lemma disjoint-union-card-eq-csum:
  assumes A ∩ B = {}
  shows |A ∪ B| =o |A| +c |B|
proof −
  have fun-disjoint-union-card-eq-csum ‘ (A <+> B) ⊆ A ∪ B
  proof (rule image-subsetI)
    fix x
    assume x ∈ A <+> B
    moreover {
      assume x ∈ Inl ‘ A
      hence fun-disjoint-union-card-eq-csum x ∈ A ∪ B by auto
    }
    moreover {
      assume x ∈ Inr ‘ B
      hence fun-disjoint-union-card-eq-csum x ∈ A ∪ B by auto
```

}
ultimately show *fun-disjoint-union-card-eq-csum* $x \in A \cup B$ **by** *auto*
qed
moreover have *inj-on fun-disjoint-union-card-eq-csum* $(A <+> B)$
proof (*rule inj-onI*)
fix x **and** y
assume $x \in A <+> B$
and $y \in A <+> B$
and $*$: *fun-disjoint-union-card-eq-csum* x = *fun-disjoint-union-card-eq-csum* y
from *this*(*1*,*2*) **consider**
(A) $x \in Inl$ ‘ A **and** $y \in Inl$ ‘ A
| (B) $x \in Inl$ ‘ A **and** $y \in Inr$ ‘ B
| (C) $x \in Inr$ ‘ B **and** $y \in Inl$ ‘ A
| (D) $x \in Inr$ ‘ B **and** $y \in Inr$ ‘ B
by *blast*
thus $x = y$
proof *cases*
case A
with $*$ **show** *?thesis* **by** *auto*
next
case B
with $*$ **and** *assms* **show** *?thesis* **by** *fastforce*
next
case C
with $*$ **and** *assms* **show** *?thesis* **by** *fastforce*
next
case D
with $*$ **show** *?thesis* **by** *auto*
qed
qed
ultimately have $|A| +c |B| \leq o |A \cup B|$ **unfolding** *csum-definition* **by** *auto*
moreover have $|A \cup B| \leq o |A <+> B|$
proof $-$
have $|A \cup B| \leq o |A| +c |B|$ **by** (*fact union-card-leq-csum*)
thus *?thesis* **unfolding** *csum-definition* **.**
qed
ultimately show $|A \cup B| =o |A| +c |B|$ **unfolding** *csum-definition* **by** (*intro thm-2-3-2*)
qed

proposition *cprod-definition*:
shows $|A| *c |B| = |A \times B|$
unfolding *cprod-def* **by** (*simp only*: *Field-card-of*)

lemma *cprod-welldefinedness*:
assumes $|A| =o |A'|$

and $|B| =o |B'|$
shows $|A| *c |B| =o |A'| *c |B'|$
proof –
from *assms(1)* **obtain** *f* **where** *f*: *bij-betw f A A'* **by** *auto*
from *f* **have** *f-inj*: *inj-on f A* **and** *f-surj*: *f ‘ A = A'* **by** *auto*
from *assms(2)* **obtain** *g* **where** *g*: *bij-betw g B B'* **by** *auto*
from *g* **have** *g-inj*: *inj-on g B* **and** *g-surj*: *g ‘ B = B'* **by** *auto*
define *h* **where** $h \equiv$ *map-prod f g*
from *f-inj* **and** *g-inj* **and** *h-def* **have** *inj-on h* $(A \times B)$ **by** (*auto intro*: *map-prod-inj-on*)
moreover from *f-surj* **and** *g-surj* **and** *h-def* **have** *h ‘* $(A \times B) = A' \times B'$ **by** *auto*
ultimately have *bij-betw h* $(A \times B)$ $(A' \times B')$ **by** (*rule bij-betw-imageI*)
hence $|A \times B| =o |A' \times B'|$ **by** *auto*
thus *?thesis* **unfolding** *cprod-definition* **by** *simp*
qed

lemmas *cprod-cong'* = *cprod-welldefinedness*

lemma *cprod-cong1'*:
assumes $|A| =o |A'|$
shows $|A| *c |B| =o |A'| *c |B|$
proof –
have $|B| =o |B|$ **by** *simp*
with *assms* **show** *?thesis* **by** (*intro cprod-cong*)
qed

lemma *cprod-cong2'*:
assumes $|B| =o |B'|$
shows $|A| *c |B| =o |A| *c |B'|$
proof –
have $|A| =o |A|$ **by** *simp*
with *assms* **show** *?thesis* **by** (*intro cprod-cong*)
qed

proposition *prop-2-3-5*:
shows $|A| *c |B| =o |B| *c |A|$
proof –
have $|A \times B| =o |B \times A|$ **by** (*fact Times-card-commute*)
thus *?thesis* **unfolding** *cprod-definition* **by** *simp*
qed

fun *prod-rotate* **where**
prod-rotate $((a, b), c) = (a, (b, c))$

lemma *inj-on-prod-rotate*:
shows *inj-on prod-rotate* $((A \times B) \times C)$

unfolding *inj-on-def* **by** *force*

lemma *prod-rotate-surj-on*:
shows *prod-rotate* ‘ $((A \times B) \times C) = (A \times (B \times C))$
by *force*

lemma *bij-betw-prod-rotate*:
shows *bij-betw prod-rotate* $((A \times B) \times C)$ $(A \times (B \times C))$
using *inj-on-prod-rotate* **and** *prod-rotate-surj-on* **by** (*intro bij-betw-imageI*)

proposition *prop-2-3-6*:
shows ($|A|$ $*c$ $|B|$) $*c$ $|C|$ $=o$ $|A|$ $*c$ ($|B|$ $*c$ $|C|$)
proof −
from *bij-betw-prod-rotate* **have** $|(A \times B) \times C|$ $=o$ $|A \times (B \times C)|$ **by** *auto*
thus *?thesis* **unfolding** *cprod-definition* **by** *simp*
qed

proposition *prop-2-3-7-a*:
fixes A :: *'a set*
shows $|A|$ $*c$ (*czero* :: *'b rel*) $=o$ (*czero* :: *'c rel*)
proof −
have *czero* = $|\{\}$ :: *'b set*$|$ **by** (*fact czero-def*)
hence $|A|$ $*c$ (*czero* :: *'b rel*) = $|A|$ $*c$ $|\{\}$ :: *'b set*$|$ **by** *simp*
moreover have $|A|$ $*c$ $|\{\}$ :: *'b set*$|$ = $|A \times \{\}|$ **unfolding** *cprod-definition* **..**
moreover have $|A \times (\{\}$ :: *'b set*$)|$ = $|\{\}$ :: (*'a* × *'b*) *set*$|$ **by** *simp*
moreover have $|\{\}$ :: (*'a* × *'b*) *set*$|$ = (*czero* :: (*'a* × *'b*) *rel*)
  unfolding *czero-definition* **by** *simp*
moreover have (*czero* :: (*'a* × *'b*) *rel*) $=o$ (*czero* :: *'c rel*) **by** (*fact czero-refl*)
ultimately show *?thesis* **by** *simp*
qed

lemma *empty-cprod-card-eq-czero*:
fixes A :: *'a set*
shows (*czero* :: *'b rel*) $*c$ $|A|$ $=o$ (*czero* :: *'c rel*)
proof −
have (*czero* :: *'b rel*) $*c$ $|A|$ = $|\{\}$ :: *'b set*$|$ $*c$ $|A|$ **unfolding** *czero-definition* **..**
moreover have $|\{\}$ :: *'b set*$|$ $*c$ $|A|$ = $|\{\} \times A|$ **by** (*fact cprod-definition*)
moreover have $|\{\} \times A|$ = $|\{\}|$ **by** *simp*
moreover have $|\{\}|$ $=o$ (*czero* :: *'c rel*) **by** (*fact empty-card-eq-czero*)
ultimately show *?thesis* **by** *simp*
qed

lemma *cprod-empty-card-eq-czero*:
fixes A :: *'a set*
shows $|A|$ $*c$ (*czero* :: *'b rel*) $=o$ (*czero* :: *'c rel*)

proof −
  have $|A| *c$ (*czero* :: $'b$ *rel*) $= |A| *c$ $|\{\}$ :: $'b$ *set*$|$ **unfolding** *czero-definition* **..**
  moreover have $|A| *c$ $|\{\}$ :: $'b$ *set*$| =o$ $|\{\}$ :: $'b$ *set*$| *c$ $|A|$ **by** (*fact prop-2-3-5*)
  moreover have $|\{\}$ :: $'b$ *set*$| *c$ $|A| =$ (*czero* :: $'b$ *rel*) $*c$ $|A|$ **unfolding** *czero-definition* **..**
  moreover have (*czero* :: $'b$ *rel*) $*c$ $|A| =o$ (*czero* :: $'c$ *rel*) **by** (*fact empty-cprod-card-eq-czero*)
  ultimately show *?thesis* **by** (*intro prop-2-3-7-a*)
qed

lemma *empty-imp-cprod-card-eq-czero1*:
  fixes A :: $'a$ *set*
    and B:: $'b$ *set*
  assumes $A = \{\}$
  shows $|A| *c$ $|B| =o$ (*czero* :: $'c$ *rel*)
proof −
  from *assms* **have** $|A| =o$ *czero* **by** (*fact eq-empty-imp-card-eq-czero*)
  have $|A| *c$ $|B| = |A \times B|$ **unfolding** *cprod-definition* **..**
  moreover from *assms*(*1*) **have** $|A \times B| = |\{\}|$ **by** *simp*
  moreover have $|\{\}$ :: ($'a \times 'b$) *set*$| =o$ (*czero* :: $'c$ *rel*) **by** (*fact empty-card-eq-czero*)
  ultimately show *?thesis* **by** *simp*
qed

lemma *empty-imp-cprod-card-eq-czero2*:
  fixes A :: $'a$ *set*
    and B :: $'b$ *set*
  assumes $B = \{\}$
  shows $|A| *c$ $|B| =o$ (*czero* :: $'c$ *rel*)
proof −
  from *assms* **have** $|A| *c$ $|B| = |A| *c$ $|\{\}|$ **by** *simp*
  moreover have $|A| *c$ $|\{\}| = |A| *c$ *czero* **unfolding** *czero-definition* **..**
  moreover have $|A| *c$ *czero* $=o$ (*czero* :: $'c$ *rel*) **by** (*fact prop-2-3-7-a*)
  ultimately show *?thesis* **by** *metis*
qed

lemma *inj-on-fst-Times-unit*:
  shows *inj-on fst* ($A \times \{()\}$)
  unfolding *inj-on-def* **by** *simp*

lemma *fst-surj-on-Times-unit*:
  shows *fst* ‘ ($A \times \{()\}$) $= A$
  by *simp*

lemma *bij-betw-fst-Times-unit*:
  shows *bij-betw fst* ($A \times \{()\}$) A
  using *inj-on-fst-Times-unit* **and** *fst-surj-on-Times-unit* **by** (*intro bij-betw-imageI*)

proposition *prop-2-3-7-b*:
shows $|A| *c\ cone =o\ |A|$
proof –
have $|A| *c\ cone = |A| *c\ |\{()\}|$ **unfolding** *cone-def* ..
moreover have $|A| *c\ |\{()\}| = |A \times \{()\}|$ **unfolding** *cprod-definition* ..
moreover from *bij-betw-fst-Times-unit* **have** $|A \times \{()\}| =o\ |A|$ **by** *auto*
ultimately show *?thesis* **by** *simp*
qed

proposition *prop-2-3-8*:
assumes $|A| \leq o\ |B|$
and $|A'| \leq o\ |B'|$
shows $|A| *c\ |A'| \leq o\ |B| *c\ |B'|$
proof –
from *assms(1)* **obtain** f **where** f ‘ $A \subseteq B$ **and** *inj-on f A* **by** *auto*
from *assms(2)* **obtain** g **where** g ‘ $A' \subseteq B'$ **and** *inj-on g A'* **by** *auto*
define h **where** $h \equiv$ *map-prod f g*
from ⟨*inj-on f A*⟩ **and** ⟨*inj-on g A'*⟩ **and** *h-def* **have** *inj-on h* $(A \times A')$
by (*auto intro*: *map-prod-inj-on*)
moreover from ⟨f ‘ $A \subseteq B$⟩ **and** ⟨g ‘ $A' \subseteq B'$⟩ **and** *h-def* **have** h ‘ $(A \times A') \subseteq B \times B'$ **by** *auto*
ultimately have $|A \times A'| \leq o\ |B \times B'|$ **by** *auto*
thus *?thesis* **unfolding** *cprod-definition* **by** *simp*
qed

fun *prod-sum-distribute* **where**
prod-sum-distribute $(a,\ Inl\ b) = Inl\ (a,\ b)$
| *prod-sum-distribute* $(a,\ Inr\ c) = Inr\ (a,\ c)$

lemma *inj-on-prod-sum-distribute*:
shows *inj-on prod-sum-distribute* $(A \times (B <+> C))$
unfolding *inj-on-def* **by** *fastforce*

lemma *prod-sum-distribute-surj-on*:
shows *prod-sum-distribute* ‘ $(A \times (B <+> C)) = A \times B <+> A \times C$
by *force*

lemma *bij-betw-prod-sum-distribute*:
shows *bij-betw prod-sum-distribute* $(A \times (B <+> C))\ (A \times B <+> A \times C)$
using *inj-on-prod-sum-distribute* **and** *prod-sum-distribute-surj-on* **by** (*intro bij-betw-imageI*)

proposition *prop-2-3-9'*:
shows $|A| *c\ (\ |B| +c\ |C|\ ) =o\ |A| *c\ |B| +c\ |A| *c\ |C|$
proof –
from *bij-betw-prod-sum-distribute* **have** $|A \times (B <+> C)| =o\ |A \times B <+> A \times C|$ **by** *auto*
thus *?thesis* **unfolding** *csum-definition* **and** *cprod-definition* **by** *simp*

qed

proposition *prop-2-3-9*:
  shows ($|A|$ $+c$ $|B|$) $*c$ $|C|$ $=o$ $|A|$ $*c$ $|C|$ $+c$ $|B|$ $*c$ $|C|$
proof −
  have ($|A|$ $+c$ $|B|$) $*c$ $|C|$ = $|A$ <+> $B|$ $*c$ $|C|$ **unfolding** *csum-definition* **..**
  also have ... $=o$ $|C|$ $*c$ $|A$ <+> $B|$ **by** (*fact prop-2-3-5*)
  also have $|C|$ $*c$ $|A$ <+> $B|$ = $|C|$ $*c$ ($|A|$ $+c$ $|B|$) **unfolding** *csum-definition* **..**
  also have ... $=o$ $|C|$ $*c$ $|A|$ $+c$ $|C|$ $*c$ $|B|$ **by** (*fact prop-2-3-9'*)
  also have $|C|$ $*c$ $|A|$ $+c$ $|C|$ $*c$ $|B|$ = $|C \times A|$ $+c$ $|C \times B|$ **unfolding** *cprod-definition* **..**
  also have ... $=o$ $|A \times C|$ $+c$ $|B \times C|$
  proof −
    have $|C \times A|$ $=o$ $|A \times C|$ **unfolding** *cprod-definition* **by** (*fact Times-card-commute*)
    moreover have $|C \times B|$ $=o$ $|B \times C|$ **unfolding** *cprod-definition* **by** (*fact Times-card-commute*)
    ultimately show *?thesis* **by** (*fact csum-cong'*)
  qed
  also have $|A \times C|$ $+c$ $|B \times C|$ = $|A|$ $*c$ $|C|$ $+c$ $|B|$ $*c$ $|C|$ **unfolding** *cprod-definition* **..**
  finally show *?thesis* **.**
qed

theorem *thm-2-9*:
  fixes A :: $'b \Rightarrow {'a}\ set$
  assumes $|\Lambda| = \mathfrak{n}$
    and $\bigwedge l.\ l \in \Lambda \Longrightarrow |A\ l|$ $=o$ $\mathfrak{m}$
    and $\Lambda = \{\} \Longrightarrow \exists X.\ |X| = \mathfrak{m}$ — This assumption guarantees that $\mathfrak{m}$ is a cardinal number even if $\Lambda = \{\}$.
    and *disjoint-family-on* A Λ
  shows $|\bigcup l \in \Lambda.\ A\ l|$ $=o$ $\mathfrak{m}$ $*c$ $\mathfrak{n}$
proof −
  let $?B = \bigcup l \in \Lambda.\ A\ l$
  {
    assume $\Lambda = \{\}$
    have $|?B|$ $=o$ $|\{\} :: {'c}\ set|$
    proof −
      from ⟨$\Lambda = \{\}$⟩ **have** $?B = \{\}$ **by** *simp*
      hence $|?B| = |\{\}|$ **by** *simp*
      also have $|\{\} :: {'a}\ set|$ $=o$ $|\{\} :: {'c}\ set|$ **by** (*fact empty-card-eq-empty*)
      finally show *?thesis* **.**
    qed
    also have $|\{\} :: {'c}\ set|$ $=o$ $\mathfrak{m}$ $*c$ $\mathfrak{n}$
    proof −
      from ⟨$\Lambda = \{\}$⟩ **obtain** X **where** $|X| = \mathfrak{m}$ **by** (*auto dest*: *assms*(*3*))
      from ⟨$\Lambda = \{\}$⟩ **have** $|X \times \Lambda| = |\{\}|$ **by** *simp*
      also have ... $=o$ $|\{\} :: {'c}\ set|$ **by** (*fact empty-card-eq-empty*)
      finally have $|X \times \Lambda|$ $=o$ $|\{\} :: {'c}\ set|$ **.**

hence $|\{\} :: {}'c\ set| =o\ |X \times \Lambda|$ **by** *auto*

also have $|X \times \Lambda| = |X| *c\ |\Lambda|$ **unfolding** *cprod-definition* **..**

also from $\langle |X| = \mathfrak{m} \rangle$ **and** $\langle |\Lambda| = \mathfrak{n} \rangle$ **have** $\ldots = \mathfrak{m} *c\ \mathfrak{n}$ **by** *simp*

finally show *?thesis* **.**

qed

finally have $|?B| =o\ \mathfrak{m} *c\ \mathfrak{n}$ **.**

}

moreover {

assume $\Lambda \neq \{\}$

then obtain l_0 **where** $l_0 \in \Lambda$ **by** *auto*

hence $|A\ l_0| =o\ \mathfrak{m}$ **by** $(auto\ dest: assms(2))$

{

fix l

assume $l \in \Lambda$

hence $|A\ l| =o\ \mathfrak{m}$ **by** $(auto\ dest: assms(2))$

also from $\langle |A\ l_0| =o\ \mathfrak{m} \rangle$ **have** $\mathfrak{m} =o\ |A\ l_0|$ **by** (*fact ordIso-symmetric*)

finally have $|A\ l| =o\ |A\ l_0|$ **.**

hence $\exists f.\ bij\text{-}betw\ f\ (A\ l)\ (A\ l_0)$ **by** *auto*

hence $\exists f.\ bij\text{-}betw\ f\ (A\ l_0)\ (A\ l)$ **by** (*auto dest*: *bij-betw-inv*)

}

then obtain f **where** f: $f \in (\Pi\ l \in \Lambda.\ \{f.\ bij\text{-}betw\ f\ (A\ l_0)\ (A\ l)\})$ **by** (*elim AC-E-ex*)

hence *fl*: $bij\text{-}betw\ (f\ l)\ (A\ l_0)\ (A\ l)$ **if** $l \in \Lambda$ **for** l **using** *that* **by** *auto*

let $?f' = \lambda(a, l).\ f\ l\ a$

have $?f'\ `\ (A\ l_0 \times \Lambda) = ?B$

proof (*rule surj-onI*; *split-pair*)

fix a **and** l

assume $(a, l) \in A\ l_0 \times \Lambda$

hence $a \in A\ l_0$ **and** $l \in \Lambda$ **by** *simp+*

from $this(2)$ **and** *fl* **have** $bij\text{-}betw\ (f\ l)\ (A\ l_0)\ (A\ l)$ **by** *simp*

with $\langle a \in A\ l_0 \rangle$ **have** $f\ l\ a \in A\ l$ **by** *auto*

with $\langle l \in \Lambda \rangle$ **show** $f\ l\ a \in ?B$ **by** *auto*

next

fix b

assume $b \in ?B$

then obtain l **where** $l \in \Lambda$ **and** $b \in A\ l$ **by** *auto*

with *fl* **obtain** a **where** $a \in A\ l_0$ **and** $b = f\ l\ a$ **by** *blast*

with $\langle l \in \Lambda \rangle$ **show** $b \in ?f'\ `\ (A\ l_0 \times \Lambda)$ **by** *auto*

qed

moreover have $inj\text{-}on\ ?f'\ (A\ l_0 \times \Lambda)$

proof (*rule inj-onI*; *split-pair*)

fix $a\ l\ a'\ l'$

assume $(a, l) \in A\ l_0 \times \Lambda$

and $(a', l') \in A\ l_0 \times \Lambda$

and $f\ l\ a = f\ l'\ a'$

from $this(1,2)$ **have** $a \in A\ l_0$ **and** $l \in \Lambda$ **and** $a' \in A\ l_0$ **and** $l' \in \Lambda$ **by** *auto*

with *fl* **have** $f\ l\ a \in A\ l$ **and** $f\ l'\ a' \in A\ l'$ **by** *auto*
with $\langle f\ l\ a = f\ l'\ a' \rangle$ **have** $A\ l \cap A\ l' \neq \{\}$ **by** *auto*
{
assume $l \neq l'$
with $\langle l \in \Lambda \rangle$ **and** $\langle l' \in \Lambda \rangle$ **and** *assms(4)* **have** $A\ l \cap A\ l' = \{\}$
by (*elim disjoint-family-onD*)
with $\langle A\ l \cap A\ l' \neq \{\} \rangle$ **have** *False* **..**
}
hence $l = l'$ **by** *auto*
with $\langle f\ l\ a = f\ l'\ a' \rangle$ **have** $f\ l\ a = f\ l\ a'$ **by** *simp*
moreover from $\langle l \in \Lambda \rangle$ **and** *fl* **have** *inj-on* $(f\ l)\ (A\ l_0)$ **by** *auto*
moreover note $\langle a \in A\ l_0 \rangle$ **and** $\langle a' \in A\ l_0 \rangle$
ultimately have $a = a'$ **by** (*elim inj-onD*)
with $\langle l = l' \rangle$ **show** $(a,\ l) = (a',\ l')$ **by** *simp*
qed
ultimately have *bij-betw* $?f'\ (A\ l_0 \times \Lambda)\ ?B$ **by** (*intro bij-betw-imageI*)
hence $|A\ l_0 \times \Lambda| =o\ |?B|$ **by** *auto*
hence $|?B| =o\ |A\ l_0 \times \Lambda|$ **by** (*fact ordIso-symmetric*)
also have $|A\ l_0 \times \Lambda| = |A\ l_0|\ *c\ |\Lambda|$ **unfolding** *cprod-definition* **..**
also have $|A\ l_0|\ *c\ |\Lambda| =o\ \mathfrak{m}\ *c\ |\Lambda|$
proof −
from $\langle l_0 \in \Lambda \rangle$ **have** $|A\ l_0| =o\ \mathfrak{m}$ **by** (*fact assms(2)*)
thus *?thesis* **by** (*fact cprod-cong1*)
qed
finally have $|?B| =o\ \mathfrak{m}\ *c\ |\Lambda|$ **.**
with *assms(1)* **have** $|?B| =o\ \mathfrak{m}\ *c\ \mathfrak{n}$ **by** *simp*
}
ultimately show *?thesis* **by** *blast*
qed

2.2.2 B) Power of Cardinalities

proposition *cexp-definition*:
shows $|A|\ \hat{}c\ |B| = |B \rightarrow_E A|$
proof −
have $|A|\ \hat{}c\ |B| = |Func\ B\ A|$ **unfolding** *cexp-def* **by** (*simp only*: *Field-card-of*)
also have $\ldots = |B \rightarrow_E A|$
proof −
have $Func\ B\ A = B \rightarrow_E A$ **unfolding** *Func-def* **by** *auto*
thus *?thesis* **by** *simp*
qed
finally show *?thesis* **.**
qed

lemma *bij-betw-imp-id-on-comp*:

assumes *bij-betw* f A B
obtains g **where** *id-on* $(g \circ f)$ A
and *id-on* $(f \circ g)$ B
proof $-$
have *id-on* $(($*the-inv-into* A $f) \circ f)$ A
proof $($*rule id-onI*$)$
fix a
assume $a \in A$
moreover from *assms* **have** *inj-on* f A **..**
ultimately have $($*the-inv-into* A $f)$ $(f\ a) = a$ **by** $($*intro the-inv-into-f-f*$)$
thus $(($*the-inv-into* A $f) \circ f)$ $a = a$ **by** *simp*
qed
moreover have *id-on* $(f \circ ($*the-inv-into* A $f))$ B
proof $($*rule id-onI*$)$
fix b
assume $b \in B$
with *assms* **have** f $(($*the-inv-into* A $f)$ $b) = b$ **by** $($*intro f-the-inv-into-f-bij-betw*$)$
thus $(f \circ ($*the-inv-into* A $f))$ $b = b$ **by** *simp*
qed
moreover note *that*
ultimately show *thesis* **by** *simp*
qed

lemma *cexp-cong*$'$:
assumes $|A|$ $=o$ $|A'|$
and $|B|$ $=o$ $|B'|$
shows $|B|$ ^c $|A|$ $=o$ $|B'|$ ^c $|A'|$ (**is** *?LHS* $=o$ *?RHS*)
proof $-$
from *assms*(1) **obtain** u **where** *bij-betw* u A' A **by** *auto*
hence *u1*: u ‘ $A' = A$ **and** *u2*: *inj-on* u A' **by** *auto*
from *assms*(2) **obtain** v **where** v: *bij-betw* v B B' **by** *auto*
hence *v1*: v ‘ $B = B'$ **and** *v2*: *inj-on* v B **by** *auto*
define Φ **where** Φ f $a' \equiv$ *if* $a' \in A'$ *then* v $(f$ $(u$ $a'))$ *else undefined* **for** f a'
from *u1* **and** *v2* **have** Φ*1*: *inj-on* Φ $(A \rightarrow_E B)$ **unfolding** Φ-*def* **by** $($*rule prob-1-5-15-ext-a*$)$
from *assms*(1) **have** $A' = \{\} \Longrightarrow A = \{\}$ **by** *auto*
moreover from *u1* **have** u ‘ $A' \subseteq A$ **by** *simp*
moreover note *u2* **and** *v1*
ultimately have Φ ‘ $(A \rightarrow_E B) = (A' \rightarrow_E B')$ **unfolding** Φ-*def* **by** $($*rule prob-1-5-15-ext-b*$)$
with Φ*1* **have** Φ*3*: *bij-betw* Φ $(A \rightarrow_E B)$ $(A' \rightarrow_E B')$ **by** $($*rule bij-betw-imageI*$)$
have *?LHS* $=o$ $|A \rightarrow_E B|$ **unfolding** *cexp-definition* **by** *simp*
also from Φ*3* **have** $|A \rightarrow_E B|$ $=o$ $|A' \rightarrow_E B'|$ **by** *auto*
also have $|A' \rightarrow_E B'|$ $=o$ *?RHS* **unfolding** *cexp-definition* **by** *simp*
finally show *?LHS* $=o$ *?RHS* **.**
qed

lemma *cexp-cong1'*:
  assumes $|M| =o\ |N|$
  shows $|M|\ \hat{}c\ |P| =o\ |N|\ \hat{}c\ |P|$
  by (*rule cexp-cong'*, *simp*, *fact assms*)

lemma *cexp-cong2'*:
  assumes $|P| =o\ |Q|$
  shows $|M|\ \hat{}c\ |P| =o\ |M|\ \hat{}c\ |Q|$
  by (*rule cexp-cong'*, *fact assms*, *simp*)

proposition *prop-2-3-10-a*:
  fixes $A :: 'a\ set$
    and $x :: 'x$
  shows $|A|\ \hat{}c\ |\{x\}| =o\ |A|$
proof −
  define $\mathfrak{F} :: ('x \Rightarrow 'a) \Rightarrow 'a$ **where** $\mathfrak{F}\ f \equiv f\ x$ **for** f
  have $\mathfrak{F}$ ‘ $(\{x\} \rightarrow_E A) = A$
  proof (*rule surj-onI*)
    fix f
    assume $f \in \{x\} \rightarrow_E A$
    with $\mathfrak{F}$-*def* **show** $\mathfrak{F}\ f \in A$ **by** *auto*
  next
    fix a
    assume a: $a \in A$
    define f **where** $f\ t \equiv$ *if* $t = x$ *then* a *else undefined* **for** t
    have $f \in \{x\} \rightarrow_E A$
    proof (*rule PiE-I*)
      fix t
      assume $t \in \{x\}$
      with *f-def* **and** a **show** $f\ t \in A$ **by** *simp*
    next
      fix t
      assume $t \notin \{x\}$
      with *f-def* **show** $f\ t =$ *undefined* **by** *simp*
    qed
    moreover from *f-def* **and** $\mathfrak{F}$-*def* **have** $\mathfrak{F}\ f = a$ **by** *simp*
    ultimately show $a \in \mathfrak{F}$ ‘ $(\{x\} \rightarrow_E A)$ **by** *auto*
  qed
  moreover have *inj-on* $\mathfrak{F}\ (\{x\} \rightarrow_E A)$
  proof (*rule inj-onI*)
    fix $f\ g$
    assume *f0*: $f \in \{x\} \rightarrow_E A$
      and *g0*: $g \in \{x\} \rightarrow_E A$
      and $\mathfrak{F}\ f = \mathfrak{F}\ g$
    from *this*(*3*) **and** $\mathfrak{F}$-*def* **have** $f\ x = g\ x$ **by** *simp*

show $f = g$
proof (*rule ext*)
fix t
consider $t = x \mid t \neq x$ **by** *auto*
moreover {
assume $t = x$
with ⟨$f\ x = g\ x$⟩ **have** $f\ t = g\ t$ **by** *simp*
}
moreover {
assume $t \neq x$
with *f0* **and** *g0* **have** $f\ t = g\ t$ **by** *fastforce*
}
ultimately show $f\ t = g\ t$ **by** *auto*
qed
qed
ultimately have $\mathfrak{F}$: *bij-betw* $\mathfrak{F}$ $(\{x\} \to_E A)$ A **by** (*intro bij-betw-imageI*)
have $|A|$ $\hat{}c$ $|\{x\}| = |\{x\} \to_E A|$ **by** (*fact cexp-definition*)
also from $\mathfrak{F}$ **have** $\ldots =o\ |A|$ **by** *auto*
finally show *?thesis* **.**
qed

proposition *prop-2-3-10-b*:
shows *cone* $\hat{}c$ $|A| =o$ *cone*
proof –
define f **where** $f\ a \equiv$ *if* $a \in A$ *then* () *else undefined* **for** a
have *cone* $\hat{}c$ $|A| = |\{()\}|$ $\hat{}c$ $|A|$ **unfolding** *cone-def* **..**
also have $\ldots = |A \to_E \{()\}|$ **by** (*fact cexp-definition*)
also have $|A \to_E \{()\}| = |\{f\}|$
proof –
have $A \to_E \{()\} = \{f\}$ **by** *auto*
thus *?thesis* **by** *simp*
qed
also have $\ldots =o$ *cone* **by** (*fact singleton-card-eq-cone*)
finally show *?thesis* **.**
qed

proposition *prop-2-3-11*:
assumes $|A| \leq o\ |A'|$
and $|B| \leq o\ |B'|$
and $B = \{\} \longleftrightarrow B' = \{\}$ — Is this assumption necessary?
shows $|A|$ $\hat{}c$ $|B| \leq o\ |A'|$ $\hat{}c$ $|B'|$
proof –
from *assms(2)* **and** *assms(3)[THEN iffD1]* **obtain** u **where** u: u ‘ $B' = B$
by (*elim card-leq-imp-surj-on*)
from *assms(1)* **obtain** v **where** *v1*: v ‘ $A \subseteq A'$ **and** *v2*: *inj-on* v A **by** *auto*

```
let ?Φ = λf. v ∘ f ∘ u
let ?Φ′ = λf. λb′. if b′ ∈ B′ then ?Φ f b′ else undefined
from assms(3)[THEN iffD2] u v1 v2
have Phi-inj: ∀f ∈ B → A. ∀f′ ∈ B → A. ext-eq-on B′ (?Φ f) (?Φ f′) ⟶ ext-eq-on B f f′
  by (auto intro: prob-1-5-15[where A = B and A′ = B′ and B = A])
have |A| ^c |B| = |B →E A| by (fact cexp-definition)
also have |B →E A| ≤o |B′ →E A′|
proof −
  have ?Φ′ ‘ (B →E A) ⊆ B′ →E A′
  proof (rule image-subsetI)
    fix f
    assume f: f ∈ B →E A
    {
      fix b′
      assume b′ ∈ B′
      with u and f and v1 have ?Φ′ f b′ ∈ A′ by auto
    }
    moreover {
      fix b′
      assume b′ ∉ B′
      hence ?Φ′ f b′ = undefined by simp
    }
    ultimately show ?Φ′ f ∈ B′ →E A′ by blast
  qed
  moreover have inj-on ?Φ′ (B →E A)
  proof (rule inj-onI)
    fix f and f′
    assume f: f ∈ B →E A and f′: f′ ∈ B →E A and ?Φ′ f = ?Φ′ f′
    {
      fix b′
      assume b′ ∈ B′
      with ‹?Φ′ f = ?Φ′ f′› have ?Φ′ f b′ = ?Φ′ f′ b′ by meson
      with ‹b′ ∈ B′› have ?Φ f b′ = ?Φ f′ b′ by simp
    }
    hence ext-eq-on B′ (?Φ f) (?Φ f′) by blast
    with f and f′ and Phi-inj have ext-eq-on B f f′ by blast
    moreover {
      fix b
      assume b ∉ B
      with f and f′ have f b = f′ b by fastforce
    }
    ultimately show f = f′ by auto
  qed
  ultimately show ?thesis by auto
qed
```

also have $|B' \to_E A'| = |A'| \ \hat{} c \ |B'|$ **unfolding** *cexp-definition* **by** *simp*
finally show *?thesis* **.**
qed

primrec *map-dom-sum* $:: ('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'c) \Rightarrow ('a + 'b) \Rightarrow 'c$ **where**
map-dom-sum f g $($*Inl a*$) = f\ a$
$|$ *map-dom-sum f g* $($*Inr a*$) = g\ a$

lemma *pair-neqE*:
assumes $p \neq q$
obtains *fst* $p \neq$ *fst* q
$|$ *snd* $p \neq$ *snd* q
proof $-$
{
assume *fst* $p =$ *fst* q
and *snd* $p =$ *snd* q
hence $p = q$ **by** (*fact prod-eqI*)
with *assms* **have** *False* **..**
}
hence *fst* $p \neq$ *fst* $q \vee$ *snd* $p \neq$ *snd* q **by** *auto*
thus *thesis* **by** (*auto intro*: *that*)
qed

theorem *thm-2-10-a*:
fixes $A :: 'a\ set$
and $B :: 'b\ set$
and $C :: 'c\ set$
shows $|C| \ \hat{} c \ |A| *c \ |C| \ \hat{} c \ |B| =o \ |C| \ \hat{} c \ (\ |A| +c \ |B| \ )$
proof $-$
define $\Phi :: ('a + 'b \Rightarrow 'c) \Rightarrow ('a \Rightarrow 'c) \times ('b \Rightarrow 'c)$
where $\Phi\ h \equiv (\lambda a.\ h\ (Inl\ a),\ \lambda b.\ h\ (Inr\ b))$ **for** h
let $?f = \lambda\varphi :: 'a + 'b \Rightarrow 'c.\ (\lambda a.\ \varphi\ (Inl\ a),\ \lambda b.\ \varphi\ (Inr\ b))$
have $|C| \ \hat{} c \ |A| *c \ |C| \ \hat{} c \ |B| = |A \to_E C| *c \ |B \to_E C|$
proof $-$
have $|C| \ \hat{} c \ |A| = |A \to_E C|$ **and** $|C| \ \hat{} c \ |B| = |B \to_E C|$ **by** (*fact cexp-definition*)+
thus *?thesis* **by** *simp*
qed
also have $\ldots = |(A \to_E C) \times (B \to_E C)|$ **by** (*fact cprod-definition*)
also have $\ldots =o \ |A <+> B \to_E C|$
proof $-$
let $?S = (A \to_E C) \times (B \to_E C)$
let $?T = A <+> B \to_E C$
let $?f = \lambda x :: ('a \Rightarrow 'c) \times ('b \Rightarrow 'c).$ *map-dom-sum* (*fst* x) (*snd* x)
have *inj-on ?f ?S*
proof (*intro inj-onI*)

```
  fix x x′
  assume x ∈ ?S
    and x′ ∈ ?S
    and map-dom-sum (fst x) (snd x) = map-dom-sum (fst x′) (snd x′)
  {
    assume x ≠ x′
    moreover {
      assume fst x ≠ fst x′
      then obtain a where fst x a ≠ fst x′ a by auto
      hence map-dom-sum (fst x) (snd x) (Inl a) ≠ map-dom-sum (fst x′) (snd x′) (Inl a)
        by simp
      with ‹map-dom-sum (fst x) (snd x) = map-dom-sum (fst x′) (snd x′)› have False by simp
    }
    moreover {
      assume snd x ≠ snd x′
      then obtain b where snd x b ≠ snd x′ b by auto
      hence map-dom-sum (fst x) (snd x) (Inr b) ≠ map-dom-sum (fst x′) (snd x′) (Inr b)
        by simp
      with ‹map-dom-sum (fst x) (snd x) = map-dom-sum (fst x′) (snd x′)› have False by simp
    }
    ultimately have False by (fact pair-neqE)
  }
  thus x = x′ by auto
qed
moreover have ?f ‘ ?S = ?T
proof (intro surj-onI)
  fix s
  assume s ∈ ?S
  {
    fix x
    assume x ∈ A <+> B
    moreover {
      fix a
      assume a ∈ A
        and x = Inl a
      from this(2) have ?f s x = ?f s (Inl a) by simp
      also have ... = (fst s) a by simp
      also have ... ∈ C
      proof −
        from ‹s ∈ ?S› have fst s ∈ A →E C by auto
        with ‹a ∈ A› show ?thesis by auto
      qed
      finally have ?f s x ∈ C .
    }
    moreover {
```

```
    fix b
    assume b ∈ B
      and x = Inr b
    from this(2) have ?f s x = ?f s (Inr b) by simp
    also have ... = (snd s) b by simp
    also have ... ∈ C
    proof –
      from ‹s ∈ ?S› have snd s ∈ B →E C by auto
      with ‹b ∈ B› show ?thesis by auto
    qed
    finally have ?f s x ∈ C .
  }
  ultimately have ?f s x ∈ C by (fact PlusE)
}
moreover {
  fix x
  assume x ∉ A <+> B
  {
    fix a
    assume x = Inl a
    with ‹x ∉ A <+> B› have a ∉ A by auto
    from ‹x = Inl a› have ?f s x = ?f s (Inl a) by simp
    also have ... = (fst s) a by simp
    also have ... = undefined
    proof –
      from ‹s ∈ ?S› have fst s ∈ A →E C by auto
      with ‹a ∉ A› show ?thesis by auto
    qed
    finally have ?f s x = undefined .
  }
  moreover {
    fix b
    assume x = Inr b
    with ‹x ∉ A <+> B› have b ∉ B by auto
    from ‹x = Inr b› have ?f s x = ?f s (Inr b) by simp
    also have ... = (snd s) b by simp
    also have ... = undefined
    proof –
      from ‹s ∈ ?S› have snd s ∈ B →E C by auto
      with ‹b ∉ B› show ?thesis by auto
    qed
    finally have ?f s x = undefined .
  }
  ultimately have ?f s x = undefined by (fact sumE)
}
```

```
  ultimately show ?f s ∈ ?T by auto
next
  fix t
  assume t ∈ ?T
  let ?g = λa. t (Inl a)
  let ?h = λb. t (Inr b)
  {
    fix x :: 'a + 'b
    {
      fix a
      assume x = Inl a
      hence ?f (?g, ?h) x = t x by simp
    }
    moreover {
      fix b
      assume x = Inr b
      hence ?f (?g, ?h) x = t x by simp
    }
    ultimately have ?f (?g, ?h) x = t x by (fact sumE)
  }
  hence ?f (?g, ?h) = t by auto
  moreover have (?g, ?h) ∈ ?S
  proof −
    have ?g ∈ A →E C
    proof (intro PiE-I)
      fix a
      assume a ∈ A
      hence Inl a ∈ A <+> B by auto
      with ‹t ∈ ?T› show ?g a ∈ C by auto
    next
      fix a
      assume a ∉ A
      hence Inl a ∉ A <+> B by auto
      with ‹t ∈ ?T› show ?g a = undefined by auto
    qed
    moreover have ?h ∈ B →E C
    proof (intro PiE-I)
      fix b
      assume b ∈ B
      hence Inr b ∈ A <+> B by auto
      with ‹t ∈ ?T› show ?h b ∈ C by auto
    next
      fix b
      assume b ∉ B
      hence Inr b ∉ A <+> B by auto
```

with ‹$t \in$ *?T*› **show** *?h* b = *undefined* **by** *auto*
qed
ultimately show *?thesis* **..**
qed
ultimately show $t \in$ *?f* ‘ *?S* **by** *force*
qed
ultimately have *bij-betw* *?f* $((A \rightarrow_E C) \times (B \rightarrow_E C))$ (A<+> $B \rightarrow_E C$) **by** (*fact* *bij-betw-imageI*)
thus *?thesis* **by** *auto*
qed
also have $|A$ <+> $B \rightarrow_E C| =o |C|$ ^c $|A$ <+> $B|$ **unfolding** *cexp-definition* **by** *simp*
also have $|C|$ ^c $|A$ <+> $B| = |C|$ ^c ($|A|$ +c $|B|$) **unfolding** *csum-definition* **..**
finally show *?thesis* **by** *simp*
qed

primrec *map-prod′* **where** *map-prod′* $(f, g)\ x = (f\ x, g\ x)$

lemma *map-prod′-eq1*:
assumes *map-prod′* (f, g) = *map-prod′* (f', g')
shows $f = f'$
proof (*rule ext*)
fix x
from *assms* **have** *map-prod′* $(f, g)\ x$ = *map-prod′* $(f', g')\ x$ **by** *simp*
thus $f\ x = f'\ x$ **by** *simp*
qed

lemma *map-prod′-eq2*:
assumes *map-prod′* (f, g) = *map-prod′* (f', g')
shows $g = g'$
proof (*rule ext*)
fix x
from *assms* **have** *map-prod′* $(f, g)\ x$ = *map-prod′* $(f', g')\ x$ **by** *simp*
thus $g\ x = g'\ x$ **by** *simp*
qed

theorem *thm-2-10-b*:
fixes P :: *′p set*
and M :: *′m set*
and N :: *′n set*
shows ($|M|$ *c $|N|$) ^c $|P| =o |M|$ ^c $|P|$ *c $|N|$ ^c $|P|$
proof −
define $\mathfrak{F}$ **where** $\mathfrak{F}$ $h\ p \equiv$ *if* $p \in P$ *then map-prod′* $h\ p$ *else undefined*
for h :: $('p \Rightarrow 'm) \times ('p \Rightarrow 'n)$ **and** p
have *bij-betw* $\mathfrak{F}$ $((P \rightarrow_E M) \times (P \rightarrow_E N))\ (P \rightarrow_E M \times N)$
proof (*rule bij-betw-imageI*)

```
  show inj-on 𝔉 ((P →E M) × (P →E N))
  proof (rule inj-onI, split-pair)
    fix f and g and f' and g'
    assume (f, g) ∈ (P →E M) × (P →E N)
      and (f', g') ∈ (P →E M) × (P →E N)
      and 𝔉: 𝔉 (f, g) = 𝔉 (f', g')
    hence f ∈ P →E M
      and g ∈ P →E N
      and f' ∈ P →E M
      and g' ∈ P →E N by simp-all
    have f = f'
    proof (rule ext)
      fix p
      {
        assume p ∈ P
        from 𝔉 have 𝔉 (f, g) p = 𝔉 (f', g') p by simp
        with 𝔉-def and ‹p ∈ P› have f p = f' p by simp
      }
      moreover {
        assume p ∉ P
        with ‹f ∈ P →E M› and ‹f' ∈ P →E M› have f p = f' p by fastforce
      }
      ultimately show f p = f' p by auto
    qed
    moreover have g = g'
    proof (rule ext)
      fix p
      {
        assume p ∈ P
        from 𝔉 have 𝔉 (f, g) p = 𝔉 (f', g') p by simp
        with 𝔉-def and ‹p ∈ P› have g p = g' p by simp
      }
      moreover {
        assume p ∉ P
        with ‹g ∈ P →E N› and ‹g' ∈ P →E N› have g p = g' p by fastforce
      }
      ultimately show g p = g' p by auto
    qed
    ultimately show (f, g) = (f', g') by simp
  qed
next
  show 𝔉 ` ((P →E M) × (P →E N)) = P →E M × N
  proof (rule surj-onI, split-pair)
    fix f and g
    assume (f, g) ∈ (P →E M) × (P →E N)
```

hence $f \in P \rightarrow_E M$ **and** $g \in P \rightarrow_E N$ **by** *simp-all*
show $\mathfrak{F}\ (f,\ g) \in P \rightarrow_E M \times N$
proof (*rule PiE-I*)
fix p
assume $p \in P$
hence $\mathfrak{F}\ (f,\ g)\ p = (f\ p,\ g\ p)$ **unfolding** $\mathfrak{F}$-*def* **by** *simp*
also from $\langle p \in P \rangle$ **and** $\langle f \in P \rightarrow_E M \rangle$ **and** $\langle g \in P \rightarrow_E N \rangle$ **have** $\ldots \in M \times N$ **by** *auto*
finally show $\mathfrak{F}\ (f,\ g)\ p \in M \times N$ **.**
next
fix p
assume $p \notin P$
with $\mathfrak{F}$-*def* **show** $\mathfrak{F}\ (f,\ g)\ p = \mathit{undefined}$ **by** *simp*
qed
next
fix h
assume h: $h \in P \rightarrow_E M \times N$
define f **where** $f\ p \equiv \mathit{if}\ p \in P\ \mathit{then}\ \mathit{fst}\ (h\ p)\ \mathit{else}\ \mathit{undefined}$ **for** p
from *f-def* **and** h **have** $f \in P \rightarrow_E M$ **by** *force*
define g **where** $g\ p \equiv \mathit{if}\ p \in P\ \mathit{then}\ \mathit{snd}\ (h\ p)\ \mathit{else}\ \mathit{undefined}$ **for** p
from *g-def* **and** h **have** $g \in P \rightarrow_E N$ **by** *force*
have $\mathfrak{F}\ (f,\ g) = h$
proof (*rule ext*)
fix p
{
assume $p \in P$
hence $\mathfrak{F}\ (f,\ g)\ p = (f\ p,\ g\ p)$ **unfolding** $\mathfrak{F}$-*def* **and** *map-prod′-def* **by** *simp*
also from $\langle p \in P \rangle$ **have** $\ldots = h\ p$ **unfolding** *f-def* **and** *g-def* **by** *simp*
finally have $\mathfrak{F}\ (f,\ g)\ p = h\ p$ **.**
}
moreover {
assume $p \notin P$
hence $\mathfrak{F}\ (f,\ g)\ p = \mathit{undefined}$ **unfolding** $\mathfrak{F}$-*def* **by** *simp*
also from h **and** $\langle p \notin P \rangle$ **have** $\ldots = h\ p$ **by** *fastforce*
finally have $\mathfrak{F}\ (f,\ g)\ p = h\ p$ **.**
}
ultimately show $\mathfrak{F}\ (f,\ g)\ p = h\ p$ **by** *auto*
qed
with $\langle f \in P \rightarrow_E M \rangle$ **and** $\langle g \in P \rightarrow_E N \rangle$ **show** $h \in \mathfrak{F}\ `\ ((P \rightarrow_E M) \times (P \rightarrow_E N))$ **by** *auto*
qed
qed
hence *bij-betw* (*the-inv-into* $((P \rightarrow_E M) \times (P \rightarrow_E N))\ \mathfrak{F})\ (P \rightarrow_E M \times N)\ ((P \rightarrow_E M) \times (P \rightarrow_E N))$
by (*fact bij-betw-the-inv-into*)
hence $|P \rightarrow_E M \times N| =o\ |(P \rightarrow_E M) \times (P \rightarrow_E N)|$ **by** *auto*
thus *?thesis* **unfolding** *cprod-definition* **and** *cexp-definition* **by** *simp*

```
qed

theorem thm-2-10-c:
  fixes A :: 'a set
    and B :: 'b set
    and C :: 'c set
  shows ( |C| ^c |A| ) ^c |B| =o |C| ^c ( |A| *c |B| )
proof -
  define 𝔉 where 𝔉 f b ≡ if b ∈ B then λa. f (b, a) else undefined
    for f :: 'b × 'a ⇒ 'c and b
  have bij-betw 𝔉 (B × A →E C) (B →E (A →E C))
  proof (rule bij-betw-imageI)
    show inj-on 𝔉 (B × A →E C)
    proof (rule inj-onI)
      fix f and g
      assume f: f ∈ B × A →E C
        and g: g ∈ B × A →E C
        and 𝔉: 𝔉 f = 𝔉 g
      show f = g
      proof (rule ext)
        fix x :: 'b × 'a
        from 𝔉 have *: 𝔉 f (fst x) (snd x) = 𝔉 g (fst x) (snd x) by simp
        {
          assume fst x ∈ B
          with * have f x = g x unfolding 𝔉-def by simp
        }
        moreover {
          assume fst x ∉ B
          hence x ∉ B × A by auto
          with f and g have f x = undefined and g x = undefined by auto
          hence f x = g x by simp
        }
        ultimately show f x = g x by auto
      qed
    qed
  next
    show 𝔉 ‘ (B × A →E C) = B →E (A →E C)
    proof (rule surj-onI)
      fix f
      assume f: f ∈ B × A →E C
      show 𝔉 f ∈ B →E (A →E C)
      proof (rule PiE-I)
        fix b
        assume b ∈ B
        show 𝔉 f b ∈ A →E C
```

proof (*rule PiE-I*)

fix a

assume $a \in A$

with $\langle b \in B \rangle$ **and** f **show** $\mathfrak{F}\ f\ b\ a \in C$ **unfolding** $\mathfrak{F}$-*def* **by** *auto*

next

fix a

assume $a \notin A$

with $\langle b \in B \rangle$ **and** f **show** $\mathfrak{F}\ f\ b\ a =$ *undefined* **unfolding** $\mathfrak{F}$-*def* **by** *auto*

qed

next

fix b

assume $b \notin B$

with $\mathfrak{F}$-*def* **show** $\mathfrak{F}\ f\ b =$ *undefined* **by** *simp*

qed

next

fix f

assume f: $f \in B \rightarrow_E (A \rightarrow_E C)$

define f' **where** f': $f'\ x \equiv$ *if* $x \in B \times A$ *then* $f\ (fst\ x)\ (snd\ x)$ *else undefined* **for** x

have $f' \in B \times A \rightarrow_E C$

proof (*rule PiE-I*)

fix x

assume $x \in B \times A$

with f' **and** f **show** $f'\ x \in C$ **by** *fastforce*

next

fix x

assume $x \notin B \times A$

with f' **show** $f'\ x =$ *undefined* **by** *simp*

qed

moreover have $\mathfrak{F}\ f' = f$

proof (*rule ext*)

fix b

consider (A) $b \in B$

| (B) $b \notin B$ **by** *auto*

thus $\mathfrak{F}\ f'\ b = f\ b$

proof *cases*

case A

show *?thesis*

proof (*rule ext*)

fix a

consider (C) $a \in A$

| (D) $a \notin A$ **by** *auto*

thus $\mathfrak{F}\ f'\ b\ a = f\ b\ a$

proof *cases*

case C

with A **and** f' **show** *?thesis* **unfolding** $\mathfrak{F}$-*def* **by** *simp*

next
case D
with A **and** f' **and** f **show** *?thesis* **unfolding** $\mathfrak{F}$-*def* **by** *force*
qed
qed
next
case B
with f **show** *?thesis* **unfolding** $\mathfrak{F}$-*def* **by** *auto*
qed
qed
ultimately show $f \in \mathfrak{F}$ ‘ $(B \times A \rightarrow_E C)$ **by** *auto*
qed
qed
hence $|B \times A \rightarrow_E C| =o |B \rightarrow_E (A \rightarrow_E C)|$ **by** *auto*
moreover have $|A \times B \rightarrow_E C| =o |B \times A \rightarrow_E C|$
proof $-$
have $|A \times B| =o |B \times A|$ **by** (*fact Times-card-commute*)
hence $|C|$ ^*c* $|A \times B| =o |C|$ ^*c* $|B \times A|$ **by** (*fact cexp-cong2'*)
thus *?thesis* **unfolding** *cexp-definition* **by** *simp*
qed
ultimately have $|A \times B \rightarrow_E C| =o |B \rightarrow_E (A \rightarrow_E C)|$ **by** (*auto intro*: *card-eq-trans*)
hence $|B \rightarrow_E (A \rightarrow_E C)| =o |A \times B \rightarrow_E C|$ **by** *auto*
moreover have $|B \rightarrow_E (A \rightarrow_E C)| = (|C|$ ^*c* $|A|)$ ^*c* $|B|$ **unfolding** *cexp-definition* **by** *simp*
moreover have $|A \times B \rightarrow_E C| = |C|$ ^*c* $(|A| *c |B|)$
unfolding *cprod-definition* **and** *cexp-definition* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

proposition *thm-2-3-b*:
fixes $\Lambda :: 'l\ set$
and $B :: 'l \Rightarrow 'b\ set$
and $B_0 :: 'c\ set$
assumes $\bigwedge l.\ l \in \Lambda \Longrightarrow |B\ l| =o |B_0|$
shows $|\prod_d\ l \in \Lambda.\ B\ l| =o |B_0|$ ^*c* $|\Lambda|$
proof $-$
{
fix l
assume $l \in \Lambda$
with *assms* **have** $\exists \sigma.$ *bij-betw* $(\sigma\ l)\ (B\ l)\ B_0$ **by** *fast*
}
then obtain σ' **where** σ': $\sigma' \in (\Pi\ l \in \Lambda.\ \{\sigma.$ *bij-betw* $(\sigma\ l)\ (B\ l)\ B_0\})$ **by** (*elim AC-E-ex*)
define σ **where** $\sigma\ l \equiv (\sigma'\ l)\ l$ **for** l
{
fix l
assume $l \in \Lambda$

with σ' **have** *bij-betw* $((\sigma'\ l)\ l)\ (B\ l)\ B_0$ **by** *auto*
}
hence σ: *bij-betw* $(\sigma\ l)\ (B\ l)\ B_0$ **if** $l \in \Lambda$ **for** l **unfolding** σ-*def* **by** (*simp add*: *that*)
hence σ-*inj*: *inj-on* $(\sigma\ l)\ (B\ l)$ **if** $l \in \Lambda$ **for** l **by** (*auto intro*: *that*)
from σ **have** σ-*surj*: $(\sigma\ l)$ ` $(B\ l) = B_0$ **if** $l \in \Lambda$ **for** l
by (*simp add*: *bij-betw-imp-surj-on that*)
define $\mathfrak{F}$ **where** $\mathfrak{F}\ \mathfrak{B}\ l \equiv$ *if* $l \in \Lambda$ *then* $(\sigma\ l)\ (\mathfrak{B}\ l)$ *else undefined* **for** $\mathfrak{B} :: 'l \Rightarrow 'b$ **and** l
have *bij-betw* $\mathfrak{F}\ (\prod_d\ l \in \Lambda.\ B\ l)\ (\Lambda \rightarrow_E B_0)$
proof (*rule bij-betw-imageI*)
show *inj-on* $\mathfrak{F}\ (\prod_d\ l \in \Lambda.\ B\ l)$
proof (*rule inj-onI*)
fix $\mathfrak{B}$ **and** $\mathfrak{B}'$
assume $\mathfrak{B}$: $\mathfrak{B} \in (\prod_d\ l \in \Lambda.\ B\ l)$
and $\mathfrak{B}'$: $\mathfrak{B}' \in (\prod_d\ l \in \Lambda.\ B\ l)$
and $*$: $\mathfrak{F}\ \mathfrak{B} = \mathfrak{F}\ \mathfrak{B}'$
show $\mathfrak{B} = \mathfrak{B}'$
proof (*rule ext*)
fix l
{
assume l: $l \in \Lambda$
with $\mathfrak{B}$ **and** $\mathfrak{B}'$ **have** $\mathfrak{B}\ l \in B\ l$ **and** $\mathfrak{B}'\ l \in B\ l$ **by** *auto*
moreover from $*$ **and** l **have** $(\sigma\ l)\ (\mathfrak{B}\ l) = (\sigma\ l)\ (\mathfrak{B}'\ l)$ **unfolding** $\mathfrak{F}$-*def* **by** *meson*
moreover from σ-*inj* **and** l **have** *inj-on* $(\sigma\ l)\ (B\ l)$ **by** *simp*
ultimately have $\mathfrak{B}\ l = \mathfrak{B}'\ l$ **by** (*auto dest*: *inj-onD*)
}
moreover {
assume l: $l \notin \Lambda$
with $\mathfrak{B}$ **and** $\mathfrak{B}'$ **have** $\mathfrak{B}\ l = \mathfrak{B}'\ l$ **by** *fastforce*
}
ultimately show $\mathfrak{B}\ l = \mathfrak{B}'\ l$ **by** *auto*
qed
qed
next
show $\mathfrak{F}$ ` $(\prod_d\ l \in \Lambda.\ B\ l) = (\Lambda \rightarrow_E B_0)$
proof (*rule surj-onI*)
fix $\mathfrak{B}$
assume $\mathfrak{B}$: $\mathfrak{B} \in (\prod_d\ l \in \Lambda.\ B\ l)$
show $\mathfrak{F}\ \mathfrak{B} \in \Lambda \rightarrow_E B_0$
proof (*rule PiE-I*)
fix l
assume l: $l \in \Lambda$
hence $\mathfrak{F}\ \mathfrak{B}\ l = (\sigma\ l)\ (\mathfrak{B}\ l)$ **unfolding** $\mathfrak{F}$-*def* **by** *simp*
also from $\mathfrak{B}$ **and** l **and** σ-*surj* **have** $\ldots \in B_0$ **by** *fast*
finally show $\mathfrak{F}\ \mathfrak{B}\ l \in B_0$ **.**
next

fix l
assume $l \notin \Lambda$
with $\mathfrak{F}$-*def* **show** $\mathfrak{F}\ \mathfrak{B}\ l = \mathit{undefined}$ **by** *simp*
qed
next
fix f
assume f: $f \in \Lambda \to_E B_0$
define τ **where** $\tau\ l \equiv$ *the-inv-into* $(B\ l)\ (\sigma\ l)$ **for** l
define $\mathfrak{B}$ **where** $\mathfrak{B}\ l \equiv$ *if* $l \in \Lambda$ *then* $(\tau\ l)\ (f\ l)$ *else undefined* **for** l
have $\mathfrak{B} \in (\prod_d\ l \in \Lambda.\ B\ l)$
proof (*rule dprodI*)
fix l
assume l: $l \in \Lambda$
with f **have** $f\ l \in B_0$ **by** *auto*
moreover have $(\tau\ l)$ ‘ $B_0 = B\ l$
proof $-$
from σ **and** l **have** *bij-betw* $(\sigma\ l)\ (B\ l)\ B_0$ **by** *simp*
hence *bij-betw* $(\tau\ l)\ B_0\ (B\ l)$ **unfolding** τ-*def* **by** (*fact bij-betw-the-inv-into*)
thus *?thesis* **by** *auto*
qed
moreover note l
ultimately show $\mathfrak{B}\ l \in B\ l$ **unfolding** $\mathfrak{B}$-*def* **by** *auto*
next
fix l
assume $l \notin \Lambda$
thus $\mathfrak{B}\ l = \mathit{undefined}$ **unfolding** $\mathfrak{B}$-*def* **by** *simp*
qed
moreover have $\mathfrak{F}\ \mathfrak{B} = f$
proof (*rule ext*)
fix l
{
assume l: $l \in \Lambda$
hence $(\mathfrak{F}\ \mathfrak{B})\ l = (\sigma\ l)\ ((\tau\ l)\ (f\ l))$ **unfolding** $\mathfrak{F}$-*def* $\mathfrak{B}$-*def* **by** *simp*
moreover have *id-on* $((\sigma\ l) \circ (\tau\ l))\ B_0$
proof (*rule id-onI*)
fix b
assume b: $b \in B_0$
have $((\sigma\ l) \circ (\tau\ l))\ b = (\sigma\ l)\ ((\tau\ l)\ b)$ **by** *simp*
also have $\ldots = (\sigma\ l)$ ((*the-inv-into* $(B\ l)\ (\sigma\ l))\ b)$ **unfolding** τ-*def* **..**
also have $\ldots = b$
proof (*rule f-the-inv-into-f*)
from σ-*inj* **and** l **show** *inj-on* $(\sigma\ l)\ (B\ l)$ **.**
next
from σ-*surj* **and** l **and** b **show** $b \in (\sigma\ l)$ ‘ $(B\ l)$ **by** *simp*
qed

finally show $((\sigma\ l) \circ (\tau\ l))\ b = b$ **.**
qed
moreover from f **and** l **have** $f\ l \in B_0$ **by** *auto*
ultimately have $(\mathfrak{F}\ \mathfrak{B})\ l = f\ l$ **by** *auto*
}
moreover {
assume $l \notin \Lambda$
with f **have** $(\mathfrak{F}\ \mathfrak{B})\ l = f\ l$ **unfolding** $\mathfrak{F}$*-def* **by** *fastforce*
}
ultimately show $(\mathfrak{F}\ \mathfrak{B})\ l = f\ l$ **by** *auto*
qed
ultimately show $f \in \mathfrak{F}$ ‘ $(\prod_d\ l \in \Lambda.\ B\ l)$ **by** *auto*
qed
qed
thus *?thesis* **unfolding** *cexp-definition* **by** *auto*
qed

proposition *prop-2-3-15-a*:
fixes M :: *'m set*
shows $|Pow\ M|$ *=o ctwo ^c* $|M|$
proof −
define $\mathfrak{F}$ **where** $\mathfrak{F}\ X\ m \equiv$ *if* $m \in M$ *then* $m \in X$ *else undefined* **for** X :: *'m set* **and** m
have *bij-betw* $\mathfrak{F}$ $(Pow\ M)$ $(M \to_E (UNIV :: bool\ set))$
proof (*rule bij-betw-imageI'*)
fix X **and** Y
assume $X \in Pow\ M$
and $Y \in Pow\ M$
and $\mathfrak{F}$: $\mathfrak{F}\ X = \mathfrak{F}\ Y$
show $X = Y$
proof (*rule equalityI*)
{
fix x
assume x: $x \in X$
with $\langle X \in Pow\ M\rangle$ **have** x': $x \in M$ **by** *auto*
{
assume $x \notin Y$
with x **and** x' **and** $\mathfrak{F}$ **have** *False* **unfolding** $\mathfrak{F}$*-def* **by** *meson*
}
hence $x \in Y$ **by** *auto*
}
thus $X \subseteq Y$ **..**
next
{
fix y
assume y: $y \in Y$

```
      with ‹Y ∈ Pow M› have y': y ∈ M by auto
      {
        assume y ∉ X
        with y and y' and 𝔉 have False unfolding 𝔉-def by meson
      }
      hence y ∈ X by auto
    }
    thus Y ⊆ X ..
  qed
next
  fix X
  assume X ∈ Pow M
  show 𝔉 X ∈ M →E UNIV
  proof (rule PiE-I)
    fix x
    show 𝔉 X x ∈ UNIV ..
  next
    fix x
    assume x ∉ M
    thus 𝔉 X x = undefined unfolding 𝔉-def by simp
  qed
next
  fix f :: 'm ⇒ bool
  assume f: f ∈ M →E UNIV
  define X where X ≡ {x ∈ M. f x}
  have 𝔉 X = f
  proof (rule ext)
    fix x
    {
      assume x: x ∈ M
      hence 𝔉 X x = f x unfolding 𝔉-def and X-def by simp
    }
    moreover {
      assume x ∉ M
      with f have 𝔉 X x = f x unfolding 𝔉-def by auto
    }
    ultimately show 𝔉 X x = f x by auto
  qed
  moreover have X ∈ Pow M
  proof (rule PowI; rule subsetI)
    fix x
    assume x ∈ X
    hence x ∈ M and f x unfolding X-def by simp-all
    with f show x ∈ M by simp
  qed
```

ultimately show $f \in \mathfrak{F}$ ‘ $(Pow\ M)$ **by** *auto*
qed
hence $|Pow\ M| =o\ |M \rightarrow_E (UNIV :: bool\ set)|$ **by** *auto*
thus *?thesis* **unfolding** *cexp-definition* **and** *ctwo-def* **.**
qed

proposition *prop-2-3-15-b*:
shows $|M| <o\ ctwo\ \hat{}c\ |M|$
proof −
have $|M| <o\ |Pow\ M|$ **by** (*fact thm-2-8*)
also have $|Pow\ M| =o\ ctwo\ \hat{}c\ |M|$ **by** (*fact prop-2-3-15-a*)
finally show *?thesis* **.**
qed

2.2.3 C) Operations on Cardinalities $\aleph_0$ and $\aleph$

fun *fun-2-11-a* **where**
fun-2-11-a f $(Inl\ x) = 2 * (f\ x)$
| *fun-2-11-a f* $(Inr\ x) = 2 * x + 1$

theorem *thm-2-11-a*:
fixes $M :: 'm\ set$
assumes $|M| \leq o\ \aleph_0$
shows $|M| +c\ \aleph_0 =o\ \aleph_0$
proof −
have $|M$ <+> $(UNIV :: nat\ set)| \leq o\ |UNIV :: nat\ set|$
proof −
from *assms* **obtain** f **where** f ‘ $M \subseteq (UNIV :: nat\ set)$ **and** f: *inj-on f M* **by** *auto*
have *inj-on* (*fun-2-11-a f*) (M <+> $(UNIV :: nat\ set)$)
proof (*rule inj-onI*)
fix x **and** y
assume $x \in M$ <+> $(UNIV :: nat\ set)$
and $y \in M$ <+> $(UNIV :: nat\ set)$
and *fun-2-11-a*: *fun-2-11-a f x* = *fun-2-11-a f y*
from *this*(*1*,*2*) **consider** (*A*) $x \in Inl$ ‘ M **and** $y \in Inl$ ‘ M
| (*B*) $x \in Inl$ ‘ M **and** $y \in Inr$ ‘ $UNIV$
| (*C*) $x \in Inr$ ‘ $UNIV$ **and** $y \in Inl$ ‘ M
| (*D*) $x \in Inr$ ‘ $UNIV$ **and** $y \in Inr$ ‘ $UNIV$ **by** *blast*
thus $x = y$
proof *cases*
case *A*
then obtain m **and** m' **where** $m \in M$ **and** $Inl\ m = x$ **and** $m' \in M$ **and** $Inl\ m' = y$ **by** *auto*
with *fun-2-11-a* **have** $2 * (f\ m) = 2 * (f\ m')$ **by** *auto*
hence $f\ m = f\ m'$ **by** *simp*
with f **and** $\langle m \in M \rangle$ **and** $\langle m' \in M \rangle$ **have** $m = m'$ **by** (*elim inj-onD*)

```
      with ‹Inl m = x› and ‹Inl m' = y› show ?thesis by simp
    next
      case B
      then obtain m and n where Inl m = x and Inr n = y by auto
      with fun-2-11-a have 2 * (f m) = 2 * n + 1 by auto
      hence False by presburger
      thus ?thesis by simp
    next
      case C
      then obtain n and m where Inr n = x and Inl m = y by auto
      with fun-2-11-a have 2 * n + 1 = 2 * (f m) by auto
      hence False by presburger
      thus ?thesis by simp
    next
      case D
      then obtain n and n' where n: Inr n = x and n': Inr n' = y by auto
      with fun-2-11-a have 2 * n + 1 = 2 * n' + 1 by auto
      hence n = n' by simp
      with n and n' show ?thesis by simp
    qed
  qed
  thus ?thesis by auto
 qed
 moreover have |UNIV :: nat set| ≤o |M <+> (UNIV :: nat set)|
 proof -
  have inj-on Inr (UNIV :: nat set) by simp
  thus ?thesis by auto
 qed
 ultimately have |M <+> (UNIV :: nat set)| =o |UNIV :: nat set| by (fact thm-2-3-2)
 thus |M| +c ℵ0 =o ℵ0 unfolding csum-definition by simp
qed

theorem thm-2-11-a':
 shows ℵ0 +c ℵ0 =o ℵ0
 by (rule thm-2-11-a; simp)

lemma inj-on-map-sum:
 assumes inj-on f A
  and inj-on g B
 shows inj-on (map-sum f g) (A <+> B)
proof (rule inj-onI)
 fix x and x'
 assume x ∈ A <+> B
  and x' ∈ A <+> B
  and *: (map-sum f g) x = (map-sum f g) x'
```

from *this*(*1*,*2*) **consider**
(A) $x \in \mathit{Inl}$ ‘ A **and** $x' \in \mathit{Inl}$ ‘ A
| (B) $x \in \mathit{Inl}$ ‘ A **and** $x' \in \mathit{Inr}$ ‘ B
| (C) $x \in \mathit{Inr}$ ‘ B **and** $x' \in \mathit{Inl}$ ‘ A
| (D) $x \in \mathit{Inr}$ ‘ B **and** $x' \in \mathit{Inr}$ ‘ B
by *blast*
thus $x = x'$
proof *cases*
case A
then obtain a **and** a' **where** $a \in A$ **and** $\mathit{Inl}\ a = x$ **and** $a' \in A$ **and** $\mathit{Inl}\ a' = x'$ **by** *auto*
with $*$ **have** $\mathit{Inl}\ (f\ a) = \mathit{Inl}\ (f\ a')$ **by** *auto*
hence $f\ a = f\ a'$ **by** *simp*
with *assms*(*1*) **and** $\langle a \in A \rangle$ **and** $\langle a' \in A \rangle$ **have** $a = a'$ **by** (*elim inj-onD*)
with $\langle \mathit{Inl}\ a = x \rangle$ **and** $\langle \mathit{Inl}\ a' = x' \rangle$ **show** *?thesis* **by** *simp*
next
case B
with $*$ **have** *False* **by** *auto*
thus *?thesis* **..**
next
case C
with $*$ **have** *False* **by** *auto*
thus *?thesis* **..**
next
case D
then obtain b **and** b' **where** $b \in B$ **and** $\mathit{Inr}\ b = x$ **and** $b' \in B$ **and** $\mathit{Inr}\ b' = x'$ **by** *auto*
with $*$ **have** $\mathit{Inr}\ (g\ b) = \mathit{Inr}\ (g\ b')$ **by** *auto*
hence $g\ b = g\ b'$ **by** *simp*
with *assms*(*2*) **and** $\langle b \in B \rangle$ **and** $\langle b' \in B \rangle$ **have** $b = b'$ **by** (*elim inj-onD*)
with $\langle \mathit{Inr}\ b = x \rangle$ **and** $\langle \mathit{Inr}\ b' = x' \rangle$ **show** *?thesis* **by** *simp*
qed
qed

theorem *thm-2-11-b*:
assumes $|M| \leq o\ \aleph$
shows $|M| +c\ \aleph =o\ \aleph$
proof $-$
let $?A = \{-(1 :: \mathit{real}) <..< (0 :: \mathit{real})\}$
let $?B = \{(0 :: \mathit{real}) <..< (1 :: \mathit{real})\}$
have $|\mathit{UNIV} :: \mathit{real\ set}| \leq o\ |M <+> (\mathit{UNIV} :: \mathit{real\ set})|$ **by** *auto*
moreover have $|M <+> (\mathit{UNIV} :: \mathit{real\ set})| \leq o\ |\mathit{UNIV} :: \mathit{real\ set}|$
proof $-$
have $|M <+> (\mathit{UNIV} :: \mathit{real\ set})| = |M| +c\ |\mathit{UNIV} :: \mathit{real\ set}|$ **unfolding** *csum-definition* **..**
also have $\ldots =o\ |M| +c\ |?B|$
proof $-$
have *equipotent* $?B$ $(\mathit{UNIV} :: \mathit{real\ set})$ **by** (*simp add*: *ex-2-5'''*)

hence $|?B| =o |UNIV :: real\ set|$ **by** *auto*
hence $|UNIV :: real\ set| =o |?B|$ **by** *auto*
thus *?thesis* **by** (*fact csum-cong2'*)
qed
also have $|M| +c |?B| = |M <+> ?B|$ **unfolding** *csum-definition* **..**
also have $\ldots \leq o |?A <+> ?B|$
proof –
have *equipotent ?A* (*UNIV* :: *real set*) **by** (*simp add*: *ex-2-5'''*)
hence $|?A| =o |UNIV :: real\ set|$ **by** *auto*
hence $|UNIV :: real\ set| =o |?A|$ **by** *auto*
with *assms* **have** $|M| \leq o |?A|$ **by** (*fact card-leq-card-eq-trans*)
then obtain f **where** f: f ‘ $M \subseteq ?A$ **and** f': *inj-on f M* **by** *auto*
have *id*: *id* ‘ $?B \subseteq ?B$ **and** *id'*: *inj-on id ?B* **by** *simp-all*
from f **and** *id* **have** (*map-sum f id*) ‘ $(M <+> ?B) \subseteq ?A <+> ?B$ **by** *fastforce*
moreover from f' **and** *id'* **have** *inj-on* (*map-sum f id*) $(M <+> ?B)$ **by** (*fact inj-on-map-sum*)
ultimately show *?thesis* **by** *auto*
qed
also have $|?A <+> ?B| \leq o |?A \cup ?B|$
proof –
have $?A \cap ?B = \{\}$ **by** *simp*
hence $|?A \cup ?B| =o |?A| +c |?B|$ **by** (*fact disjoint-union-card-eq-csum*)
thus *?thesis* **unfolding** *csum-definition* **by** *fast*
qed
also have $|?A \cup ?B| \leq o |UNIV :: real\ set|$ **by** *auto*
finally show *?thesis* **by** *simp*
qed
ultimately have $|M <+> (UNIV :: real\ set)| =o |UNIV :: real\ set|$ **by** (*elim thm-2-3-2*)
thus *?thesis* **unfolding** *csum-definition* **by** *simp*
qed

theorem *thm-2-11-b''*:
shows $\aleph +c\ \aleph =o\ \aleph$
by (*simp add*: *thm-2-11-b*)

theorem *thm-2-11-c*:
fixes M :: *'m set*
assumes *cone* $\leq o |M|$
and $|M| \leq o\ \aleph_0$
shows $|M| *c\ \aleph_0 =o\ \aleph_0$
proof –
have $|M \times (UNIV :: nat\ set)| \leq o |UNIV :: nat\ set|$
proof –
from *assms*(*2*) **obtain** f **where** f: f ‘ $M \subseteq (UNIV :: nat\ set)$ **and** f': *inj-on f M* **by** *auto*

define $\mathfrak{F}$ :: $'m \times nat \Rightarrow nat \times nat$ **where** $\mathfrak{F} \equiv$ *map-prod f id*
have *id*: *id* ‘ (*UNIV* :: *nat set*) ⊆ (*UNIV* :: *nat set*) **by** *simp*
have *id'*: *inj-on id* (*UNIV* :: *nat set*) **by** *simp*
from *f* **and** *id* **have** $\mathfrak{F}$ ‘ (*M* × (*UNIV* :: *nat set*)) ⊆ (*UNIV* :: *nat set*) × (*UNIV* :: *nat set*)
unfolding $\mathfrak{F}$-*def* **by** *simp*
moreover from *f'* **and** *id'* **have** *inj-on* $\mathfrak{F}$ (*M* × (*UNIV* :: *nat set*))
unfolding $\mathfrak{F}$-*def* **by** (*fact map-prod-inj-on*)
ultimately have |*M* × (*UNIV* :: *nat set*)| ≤*o* |(*UNIV* :: *nat set*) × (*UNIV* :: *nat set*)| **by** *auto*
also have |(*UNIV* :: *nat set*) × (*UNIV* :: *nat set*)| =*o* |*UNIV* :: *nat set*|
by (*fact nat-Times-nat-card-eq-aleph-zero*)
finally show *?thesis* **unfolding** *cprod-definition* **.**
qed
moreover have |*UNIV* :: *nat set*| ≤*o* |*M* × (*UNIV* :: *nat set*)|
proof −
from *assms*(*1*) **have** |{()}| ≤*o* |*M*| **unfolding** *cone-def* **by** *simp*
then obtain *f* **where** *f* ‘ {()} ⊆ *M* **by** *auto*
hence *f* () ∈ *M* **by** *simp*
define $\mathfrak{F}$ **where** $\mathfrak{F}$ *n* ≡ (*f* (), *n*) **for** *n* :: *nat*
from ‹*f* () ∈ *M*› **have** $\mathfrak{F}$ ‘ (*UNIV* :: *nat set*) ⊆ *M* × (*UNIV* :: *nat set*) **unfolding** $\mathfrak{F}$-*def* **by** *auto*
moreover have *inj-on* $\mathfrak{F}$ (*UNIV* :: *nat set*) **unfolding** $\mathfrak{F}$-*def* **by** (*meson injI prod.inject*)
ultimately show *?thesis* **by** *auto*
qed
ultimately show *?thesis* **unfolding** *cprod-definition* **by** (*fact thm-2-3-2*)
qed

lemma *surj-Real*:
shows *surj Real.Real*
proof −
obtain *rr* :: (*real* ⇒ *bool*) ⇒ *nat* ⇒ *rat* **where**
∀ *p r*. *p r* ∨ ¬ *p* (*Real.Real* (*rr p*)) ∧ *realrel* (*rr p*) (*rr p*)
by (*metis real.abs-induct*)
hence *surj* (*quot-type.abs realrel Abs-real*) **using** *Real-def* **by** *auto*
thus *?thesis* **unfolding** *Real.Real-def* **.**
qed

lemma *aleph-card-leq-rational-sequence*:
shows |*UNIV* :: *real set*| ≤*o* |*UNIV* :: (*nat* ⇒ *rat*) *set*|
using *surj-Real* **by** (*fact surj-on-imp-card-leq*)

2.2.4 Problems

proposition *prob-2-3-1-a*:
shows |*M*| +*c* |*N*| =*o* |*N*| +*c* |*M*|
by (*fact prop-2-3-1*)

proposition *prob-2-3-1-b*:
  shows ($|M|$ $+c$ $|N|$) $+c$ $|P|$ $=o$ $|M|$ $+c$ ($|N|$ $+c$ $|P|$)
  by (*fact prop-2-3-2*)

proposition *prob-2-3-1-c*:
  shows $|M|$ $+c$ *czero* $=o$ $|M|$
  by (*fact prop-2-3-3*)

proposition *prob-2-3-1-d*:
  assumes $|M|$ $\leq o$ $|M'|$
    and $|N|$ $\leq o$ $|N'|$
  shows $|M|$ $+c$ $|N|$ $\leq o$ $|M'|$ $+c$ $|N'|$
  using *assms* **by** (*fact prop-2-3-4*)

proposition *prob-2-3-1-e*:
  shows $|M|$ $*c$ $|N|$ $=o$ $|N|$ $*c$ $|M|$
  by (*fact prop-2-3-5*)

proposition *prob-2-3-1-f*:
  shows ($|M|$ $*c$ $|N|$) $*c$ $|P|$ $=o$ $|M|$ $*c$ ($|N|$ $*c$ $|P|$)
  by (*fact prop-2-3-6*)

proposition *prob-2-3-1-g-a*:
  shows $|M|$ $*c$ *czero* $=o$ *czero*
  by (*fact prop-2-3-7-a*)

proposition *prob-2-3-1-g-b*:
  shows $|M|$ $*c$ *cone* $=o$ $|M|$
  by (*fact prop-2-3-7-b*)

proposition *prob-2-3-1-h*:
  assumes $|M|$ $\leq o$ $|M'|$
    and $|N|$ $\leq o$ $|N'|$
  shows $|M|$ $*c$ $|N|$ $\leq o$ $|M'|$ $*c$ $|N'|$
  using *assms* **by** (*fact prop-2-3-8*)

proposition *prob-2-3-1-i*:
  shows ($|M|$ $+c$ $|N|$) $*c$ $|P|$ $=o$ $|M|$ $*c$ $|P|$ $+c$ $|N|$ $*c$ $|P|$
  by (*fact prop-2-3-9*)

proposition *prob-2-3-2*:
  assumes *equipotent* A A'
    and *equipotent* B B'
  shows *equipotent* $(A \times B)$ $(A' \times B')$

proof −
  from *assms* **have** $|A| =o\ |A'|$ **and** $|B| =o\ |B'|$ **by** *auto*
  hence $|A| *c\ |B| =o\ |A'| *c\ |B'|$ **by** (*fact cprod-cong'*)
  thus *?thesis* **unfolding** *cprod-definition* **by** *auto*
qed

proposition *prob-2-3-3*:
  assumes $|N| \leq o\ |N'|$
    and $|M| \leq o\ |M'|$
    and $M = \{\} \longleftrightarrow M' = \{\}$ — Is this assumption really necessary?
  shows $|N|\ \hat{}c\ |M| \leq o\ |N'|\ \hat{}c\ |M'|$
  using *assms* **by** (*fact prop-2-3-11*)

proposition *prob-2-3-5*:
  fixes $M :: 'm\ set$
  assumes *infinite M*
  shows $|M| +c\ \aleph_0 =o\ |M|$
proof −
  define $N :: ('m + nat)\ set$ **where** $N \equiv Inr$ ‘ *UNIV*
  have *infinite* (*M* <+> (*UNIV* :: *nat set*)) **by** *simp*
  moreover have $N \subseteq M$ <+> *UNIV* **unfolding** *N-def* **by** *auto*
  moreover have $|N| \leq o\ \aleph_0$ **unfolding** *N-def* **by** (*fact card-of-image*)
  moreover have *infinite* ((*M* <+> *UNIV*) − *N*)
  proof −
    have (*M* <+> *UNIV*) − *N* = *Inl* ‘ *M* **unfolding** *N-def* **by** *auto*
    moreover have *infinite* (*Inl* ‘ *M*)
    proof −
      have *inj-on Inl M* **by** *simp*
      with *assms* **show** *?thesis* **by** (*simp add*: *finite-image-iff*)
    qed
    ultimately show *?thesis* **by** *simp*
  qed
  ultimately have *equipotent* ((*M* <+> *UNIV*) − *N*) (*M* <+> (*UNIV* :: *nat set*)) **by** (*fact thm-2-6*)
  moreover have *equipotent M* ((*M* <+> *UNIV*) − *N*)
  proof −
    have *bij-betw Inl M* (*Inl* ‘ *M*) **by** (*metis bij-betw-imageI' image-eqI sum.inject*(*1*))
    hence *equipotent M* (*Inl* ‘ *M*) **by** *auto*
    moreover have *Inl* ‘ *M* = (*M* <+> *UNIV*) − *N* **unfolding** *N-def* **by** *auto*
    ultimately show *?thesis* **by** *metis*
  qed
  ultimately have *equipotent M* (*M* <+> (*UNIV* :: *nat set*)) **by** (*auto dest*: *prop-2-1-3*)
  hence $|M| =o$ |*M* <+> (*UNIV* :: *nat set*)| **by** *auto*

thus *?thesis* **unfolding** *csum-definition* **by** *auto*
qed

end

end