Lovasz Local Lemma

Chelsea Edmonds and Lawrence C. Paulson

March 17, 2025

Abstract

This entry aims to formalise several useful general techniques for using the *probabilistic method* for combinatorial structures (or discrete spaces more generally). In particular, it focuses on bounding tools, such as the union and complete independence bounds, and the first formalisation of the pivotal Lovász local lemma. The formalisation focuses on the general lemma, however also proves several useful variations, including the more well known symmetric version. Both the original formalisation and several of the variations used dependency graphs, which were formalised using Noschinski's general directed graph library [2]. Additionally, the entry provides several useful existence lemmas, required at the end of most probabilistic proofs on combinatorial structures. Finally, the entry includes several significant extensions to the existing probability libraries, particularly for conditional probability (such as Bayes theorem) and independent events. The formalisation is primarily based on Alon and Spencer's textbook [1], as well as Zhao's course notes [3].

Contents

1 Extensional function extras

Counting lemmas (i.e. reasoning on cardinality) of sets on the extensional function relation

theory *PiE-Rel-Extras* **imports** *Card-Partitions.Card-Partitions*
begin

1.1 Relations and Extensional Function sets

A number of lemmas to convert between relations and functions for counting purposes. Note, ultimately not needed in this formalisation, but may be of use in the future

lemma *Range-unfold*: $Range\ r = \{y.\ \exists x.\ (x,\ y) \in r\}$
by *blast*

definition *fun-to-rel*:: $'a\ set \Rightarrow 'b\ set \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('a \times 'b)\ set$ **where**
$fun\text{-}to\text{-}rel\ A\ B\ f \equiv \{(a,\ b) \mid a\ b\ .\ a \in A \wedge b \in B \wedge f\ a = b\}$

definition *rel-to-fun*:: $('a \times 'b)\ set \Rightarrow ('a \Rightarrow 'b)$ **where**
$rel\text{-}to\text{-}fun\ R \equiv \lambda\ a\ .\ (if\ a \in Domain\ R\ then\ (THE\ b\ .\ (a,\ b) \in R)\ else\ undefined)$

lemma *fun-to-relI*: $a \in A \Longrightarrow b \in B \Longrightarrow f\ a = b \Longrightarrow (a,\ b) \in fun\text{-}to\text{-}rel\ A\ B\ f$
unfolding *fun-to-rel-def* **by** *auto*

lemma *fun-to-rel-alt*: $fun\text{-}to\text{-}rel\ A\ B\ f \equiv \{(a,\ f\ a) \mid a\ b\ .\ a \in A \wedge f\ a \in B\}$
unfolding *fun-to-rel-def* **by** *simp*

lemma *fun-to-relI2*: $a \in A \Longrightarrow f\ a \in B \Longrightarrow (a,\ f\ a) \in fun\text{-}to\text{-}rel\ A\ B\ f$
using *fun-to-rel-alt* **by** *fast*

lemma *rel-to-fun-in*[*simp*]: $a \in \mathit{Domain}\ R \Longrightarrow (\mathit{rel\text{-}to\text{-}fun}\ R)\ a = (\mathit{THE}\ b\ .\ (a,\ b) \in R)$
unfolding *rel-to-fun-def* **by** *simp*

lemma *rel-to-fun-undefined*[*simp*]: $a \notin \mathit{Domain}\ R \Longrightarrow (\mathit{rel\text{-}to\text{-}fun}\ R)\ a = \mathit{undefined}$
unfolding *rel-to-fun-def* **by** *simp*

lemma *single-valued-unique-Dom-iff*: *single-valued* $R \longleftrightarrow (\forall\ x \in \mathit{Domain}\ R.\ \exists!\ y\ .\ (x,\ y) \in R)$
using *single-valued-def* **by** *fastforce*

lemma *rel-to-fun-range*:
assumes *single-valued* R
assumes $a \in \mathit{Domain}\ R$
shows $(\mathit{THE}\ b\ .\ (a,\ b) \in R) \in \mathit{Range}\ R$
using *single-valued-unique-Dom-iff*
by (*metis Range-iff assms*(*1*) *assms*(*2*) *theI'*)

lemma *rel-to-fun-extensional*: *single-valued* $R \Longrightarrow \mathit{rel\text{-}to\text{-}fun}\ R \in (\mathit{Domain}\ R \to_E \mathit{Range}\ R)$
by (*intro PiE-I*) (*simp-all add*: *rel-to-fun-range*)

lemma *single-value-fun-to-rel*: *single-valued* (*fun-to-rel A B f*)
unfolding *single-valued-def fun-to-rel-def*
by *simp*

lemma *fun-to-rel-domain*:
assumes $f \in A \to_E B$
shows $\mathit{Domain}\ (\mathit{fun\text{-}to\text{-}rel}\ A\ B\ f) = A$
unfolding *fun-to-rel-def* **using** *assms* **by** (*auto simp add*: *subset-antisym subsetI Domain-unfold*)

lemma *fun-to-rel-range*:
assumes $f \in A \to_E B$
shows $\mathit{Range}\ (\mathit{fun\text{-}to\text{-}rel}\ A\ B\ f) \subseteq B$
unfolding *fun-to-rel-def* **using** *assms* **by** (*auto simp add*: *subsetI Range-unfold*)

lemma *rel-to-fun-to-rel*:
assumes $f \in A \to_E B$
shows $\mathit{rel\text{-}to\text{-}fun}\ (\mathit{fun\text{-}to\text{-}rel}\ A\ B\ f) = f$
proof (*intro ext allI*)
fix x
show $\mathit{rel\text{-}to\text{-}fun}\ (\mathit{fun\text{-}to\text{-}rel}\ A\ B\ f)\ x = f\ x$
proof (*cases* $x \in A$)
case *True*
then have *ind*: $x \in \mathit{Domain}\ (\mathit{fun\text{-}to\text{-}rel}\ A\ B\ f)$ **using** *fun-to-rel-domain assms*

by *blast*
have $(x,\ f\ x) \in \mathit{fun\text{-}to\text{-}rel}\ A\ B\ f$ **using** *fun-to-rel-alt True single-value-fun-to-rel*

using *assms* **by** *fastforce*
moreover have *rel-to-fun* (*fun-to-rel A B f*) *x* = (*THE b*. (*x*, *b*) ∈ (*fun-to-rel A B f*)) **by** (*simp add*: *ind*)
ultimately show *?thesis* **using** *single-value-fun-to-rel single-valuedD the-equality*
by (*metis* (*no-types*, *lifting*))
next
case *False*
then have *x* ∉ *Domain* (*fun-to-rel A B f*) **unfolding** *fun-to-rel-def*
by *blast*
then show *?thesis*
using *False assms* **by** *auto*
qed
qed

lemma *fun-to-rel-to-fun*:
assumes *single-valued R*
shows *fun-to-rel* (*Domain R*) (*Range R*) (*rel-to-fun R*) = *R*
proof (*intro subset-antisym subsetI*)
fix *x* **assume** *x* ∈ *fun-to-rel* (*Domain R*) (*Range R*) (*rel-to-fun R*)
then obtain *a b* **where** *x* = (*a*, *b*) **and** *a* ∈ *Domain R* **and** *b* ∈ *Range R* **and** (*rel-to-fun R a*) = *b*
using *fun-to-rel-def* **by** (*smt* (*verit*) *mem-Collect-eq*)
then have *b* = (*THE b*′. (*a*, *b*′) ∈ *R*) **using** *rel-to-fun-in*
by *simp*
then show *x* ∈ *R*
by (*metis* (*no-types*, *lifting*) ‹*a* ∈ *Domain R*› ‹*x* = (*a*, *b*)› *assms single-valued-unique-Dom-iff the1-equality*)
next
fix *x* **assume** *x* ∈ *R*
then obtain *a b* **where** *x* = (*a*, *b*) **and** (*a*, *b*) ∈ *R* **and** ∀ *c* . (*a*, *c*) ∈ *R* ⟶ *b* = *c*
using *assms*
by (*metis prod.collapse single-valued-def*)
then have *a* ∈ *Domain R b* ∈ *Range R* **by** *blast*+
then have *b* = (*THE b*′ . (*a*, *b*′) ∈ *R*)
by (*metis* ‹∀ *c*. (*a*, *c*) ∈ *R* ⟶ *b* = *c*› ‹*x* = (*a*, *b*)› ‹*x* ∈ *R*› *the-equality*)
then have (*a*, *b*) ∈ *fun-to-rel* (*Domain R*) (*Range R*) (*rel-to-fun R*)
using ‹*a* ∈ *Domain R*› ‹*b* ∈ *Range R*› **by** (*intro fun-to-relI*) (*simp-all*)
then show *x* ∈ *fun-to-rel* (*Domain R*) (*Range R*) (*rel-to-fun R*) **using** ‹*x* = (*a*, *b*)› **by** *simp*
qed

lemma *bij-betw-fun-to-rel*:
assumes *f* ∈ *A* →$_E$ *B*
shows *bij-betw* (λ *a* . (*a*, *f a*)) *A* (*fun-to-rel A B f*)
proof (*intro bij-betw-imageI inj-onI*)
show ⋀*x y*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ (*x*, *f x*) = (*y*, *f y*) ⟹ *x* = *y* **by** *simp*
next
show (λ*a*. (*a*, *f a*)) ‘ *A* = *fun-to-rel A B f*

proof (*intro subset-antisym subsetI*)
fix x **assume** $x \in (\lambda a.\ (a, f\ a))\ `\ A$
then obtain a **where** $a \in A$ **and** $x = (a, f\ a)$ **by** *blast*
then show $x \in$ *fun-to-rel A B f* **using** *fun-to-rel-alt assms*
by *fastforce*
next
fix x **assume** $x \in$ *fun-to-rel A B f*
then show $x \in (\lambda a.\ (a, f\ a))\ `\ A$ **using** *fun-to-rel-alt*
using *image-iff* **by** *fastforce*
qed
qed

lemma *fun-to-rel-indiv-card*:
assumes $f \in A \rightarrow_E B$
shows *card* (*fun-to-rel A B f*) = *card A*
using *bij-betw-fun-to-rel assms bij-betw-same-card*[*of* (λ *a* . (*a*, *f a*)) *A* (*fun-to-rel A B f*)]
by (*metis*)

lemma *fun-to-rel-inj*:
assumes $C \subseteq A \rightarrow_E B$
shows *inj-on* (*fun-to-rel A B*) *C*
proof (*intro inj-onI ext allI*)
fix *f g x* **assume** *fin*: $f \in C$ **and** *gin*: $g \in C$ **and** *eq*: *fun-to-rel A B f* = *fun-to-rel A B g*
then show *f x* = *g x*
proof (*cases* $x \in A$)
case *True*
then have (*x*, *f x*) ∈ *fun-to-rel A B f* **using** *fun-to-rel-alt*
by (*smt* (*verit*) *PiE-mem assms fin fun-to-rel-def mem-Collect-eq subset-eq*)
moreover have (*x*, *g x*) ∈ *fun-to-rel A B g* **using** *fun-to-rel-alt True*
by (*smt* (*verit*) *PiE-mem assms fun-to-rel-def gin mem-Collect-eq subset-eq*)
ultimately show *?thesis* **using** *eq single-value-fun-to-rel single-valued-def*
by *metis*
next
case *False*
then have *f x* = *undefined g x* = *undefined* **using** *fin gin assms* **by** *auto*
then show *?thesis* **by** *simp*
qed
qed

lemma *fun-to-rel-ss*: *fun-to-rel A B f* ⊆ *A* × *B*
unfolding *fun-to-rel-def* **by** *auto*

lemma *card-fun-to-rel*: $C \subseteq A \rightarrow_E B \Longrightarrow$ *card C* = *card* ((λ *f* . *fun-to-rel A B f*) ` *C*)
using *card-image fun-to-rel-inj* **by** *metis*

1.2 Cardinality Lemmas

Lemmas to count variations of filtered sets over the extensional function set relation

lemma *card-PiE-filter-range-set*:
assumes $\bigwedge a.\ a \in A' \Longrightarrow X\ a \in C$
assumes $A' \subseteq A$
assumes *finite A*
shows *card* $\{f \in A \rightarrow_E C\ .\ \forall\ a \in A'\ .\ f\ a = X\ a\} = (card\ C) \hat{}(card\ A - card\ A')$
proof −
have *finA*: *finite* A' **using** *assms*(3) *finite-subset assms*(2) **by** *auto*
have *c1*: *card* $(A - A') =$ *card* $A -$ *card* A' **using** *assms*(2)
using *card-Diff-subset finA* **by** *blast*
define $g :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b)$ **where** $g \equiv \lambda\ f.\ (\lambda\ a'\ .\ if\ a' \in A'\ then\ undefined\ else\ f\ a')$
have *bij-betw g* $\{f \in A \rightarrow_E C\ .\ \forall\ a \in A'\ .\ f\ a = X\ a\}\ ((A - A') \rightarrow_E C)$
proof (*intro bij-betw-imageI inj-onI*)
fix *h h'* **assume** *h1in*: $h \in \{f \in A \rightarrow_E C.\ \forall\ a \in A'\ .\ f\ a = X\ a\}$ **and** *h2in*: $h' \in \{f \in A \rightarrow_E C.\ \forall\ a \in A'\ .\ f\ a = X\ a\}\ g\ h = g\ h'$
then have *eq*: $(\lambda\ a'\ .\ if\ a' \in A'\ then\ undefined\ \ else\ h\ a') = (\lambda\ a'\ .\ \ if\ a' \in A'\ then\ undefined\ else\ h'\ a')$
using *g-def* **by** *simp*
show $h = h'$
proof (*intro ext allI*)
fix *x*
show $h\ x = h'\ x$ **using** *h1in h2in eq* **by** (*cases* $x \in A'$, *simp*, *meson*)
qed
next
show $g\ `\ \{f \in A \rightarrow_E C.\ \forall\ a \in A'\ .\ f\ a = X\ a\} = A - A' \rightarrow_E C$
proof (*intro subset-antisym subsetI*)
fix g' **assume** $g' \in g\ `\ \{f \in A \rightarrow_E C.\ \forall\ a \in A'\ .\ f\ a = X\ a\}$
then obtain f' **where** *geq*: $g' = g\ f'$ **and** *fin*: $f' \in A \rightarrow_E C$ **and** $\forall\ a \in A'\ .\ f'\ a = X\ a$
by *blast*
show $g' \in A - A' \rightarrow_E C$
using *g-def fin geq* **by** (*intro PiE-I*)(*auto*)
next
fix g' **assume** *gin*: $g' \in A - A' \rightarrow_E C$
define f' **where** $f' = (\lambda\ a'\ .\ (if\ a' \in A'\ then\ X\ a'\ else\ g'\ a'))$
then have *eqc*: $\forall\ a' \in A'\ .\ f'\ a' = X\ a'$ **by** *auto*
have *fin*: $f' \in A \rightarrow_E C$
proof (*intro PiE-I*)
fix *x* **assume** $x \in A$
have $x \notin A' \Longrightarrow f'\ x = g'\ x$ **using** *f'-def* **by** *auto*
moreover have $x \in A' \Longrightarrow f'\ x = X\ x$ **using** *f'-def* **by** (*simp add*: ‹$x \in A$›)
ultimately show $f'\ x \in C$
using *gin PiE-E* ‹$x \in A$› *assms*(1)[*of x*] **by** (*metis Diff-iff*)

next
fix x **assume** $x \notin A$
then show $f'\ x = \mathit{undefined}$
using f'-*def gin assms*(*2*) **by** *auto*
qed
have $g' = g\ f'$ **unfolding** f'-*def g-def*
by (*auto simp add*: *fun-eq-iff*) (*metis DiffE PiE-arb gin*)
then show $g' \in g\ `\ \{f \in A \to_E C.\ \forall\ a \in A'\ .\ f\ a = X\ a\}$ **using** *fin eqc* **by** *blast*
qed
qed
then have *card* $\{f \in A \to_E C\ .\ \forall\ a \in A'\ .\ f\ a = X\ a\} = \mathit{card}\ ((A - A') \to_E C)$
using *bij-betw-same-card* **by** *blast*
also have ... $= (\mathit{card}\ C)\hat{}\mathit{card}\ (A - A')$
using *card-funcsetE assms*(*3*) **by** (*metis finite-Diff*)
finally show *?thesis* **using** *c1* **by** *auto*
qed

lemma *card-PiE-filter-range-indiv*: $X\ a' \in C \Longrightarrow a' \in A \Longrightarrow \mathit{finite}\ A \Longrightarrow$
$\mathit{card}\ \{f \in A \to_E C\ .\ f\ a' = X\ a'\} = (\mathit{card}\ C)\hat{}(\mathit{card}\ A - 1)$
using *card-PiE-filter-range-set*[*of* $\{a'\}$ X C A] **by** *auto*

lemma *card-PiE-filter-range-set-const*: $c \in C \Longrightarrow A' \subseteq A \Longrightarrow \mathit{finite}\ A \Longrightarrow$
$\mathit{card}\ \{f \in A \to_E C\ .\ \forall\ a \in A'\ .\ f\ a = c\} = (\mathit{card}\ C)\hat{}(\mathit{card}\ A - \mathit{card}\ A')$
using *card-PiE-filter-range-set*[*of* $A'\ \lambda\ a\ .\ c$] **by** *auto*

lemma *card-PiE-filter-range-set-nat*: $c \in \{0..<n\} \Longrightarrow A' \subseteq A \Longrightarrow \mathit{finite}\ A \Longrightarrow$
$\mathit{card}\ \{f \in A \to_E \{0..<n\}\ .\ \forall\ a \in A'\ .\ f\ a = c\} = n\hat{}(\mathit{card}\ A - \mathit{card}\ A')$
using *card-PiE-filter-range-set-const*[*of* c $\{0..<n\}$ A' A] **by** *auto*

end

2 Digraph extensions

Extensions to the existing library for directed graphs, basically neighborhood

theory *Digraph-Extensions*
imports
Graph-Theory.Digraph
Graph-Theory.Pair-Digraph
begin

definition (**in** *pre-digraph*) *neighborhood* :: $'a \Rightarrow 'a\ set$ **where**
neighborhood $u \equiv \{v \in verts\ G\ .\ dominates\ G\ u\ v\}$

lemma (**in** *wf-digraph*) *neighborhood-wf*: *neighborhood* $v \subseteq verts\ G$
unfolding *neighborhood-def* **by** *auto*

lemma (**in** *pair-pre-digraph*) *neighborhood-alt*:
neighborhood u = {*v* ∈ *pverts G* . (*u*, *v*) ∈ *parcs G*}
unfolding *neighborhood-def* **by** *simp*

lemma (**in** *fin-digraph*) *neighborhood-finite*: *finite* (*neighborhood v*)
using *neighborhood-wf finite-subset finite-verts* **by** *fast*

lemma (**in** *wf-digraph*) *neighborhood-edge-iff*: *y* ∈ *neighborhood x* ⟷ (*x*, *y*) ∈ *arcs-ends G*
unfolding *neighborhood-def* **using** *in-arcs-imp-in-arcs-ends* **by** *auto*

lemma (**in** *loopfree-digraph*) *neighborhood-self-not*: *v* ∉ (*neighborhood v*)
unfolding *neighborhood-def* **using** *adj-not-same* **by** *auto*

lemma (**in** *nomulti-digraph*) *inj-on-head-out-arcs*: *inj-on* (*head G*) (*out-arcs G u*)
proof (*intro inj-onI*)
fix *x y* **assume** *xin*: *x* ∈ *out-arcs G u* **and** *yin*: *y* ∈ *out-arcs G u* **and** *heq*: *head G x* = *head G y*
then have *tail G x* = *u tail G y* = *u*
using *out-arcs-def* **by** *auto*
then have *arc-to-ends G x* = *arc-to-ends G y*
unfolding *arc-to-ends-def heq* **by** *auto*
then show *x* = *y* **using** *no-multi-arcs xin yin* **by** *simp*
qed

lemma (**in** *nomulti-digraph*) *out-degree-neighborhood*: *out-degree G u* = *card* (*neighborhood u*)
proof −
let *?f* = λ *e*. *head G e*
have *bij-betw ?f* (*out-arcs G u*) (*neighborhood u*)
proof (*intro bij-betw-imageI*)
show *inj-on* (*head G*) (*out-arcs G u*) **using** *inj-on-head-out-arcs* **by** *simp*
show *head G* ‘ *out-arcs G u* = *neighborhood u*
unfolding *neighborhood-def* **using** *in-arcs-imp-in-arcs-ends* **by** *auto*
qed
then show *?thesis* **unfolding** *out-degree-def*
by (*simp add*: *bij-betw-same-card*)
qed

lemma (**in** *digraph*) *neighborhood-empty-iff*: *out-degree G u* = *0* ⟷ *neighborhood u* = {}
using *out-degree-neighborhood neighborhood-finite* **by** *auto*

end

3 General Event Lemmas

General lemmas for reasoning on events in probability spaces after different operations

theory *Prob-Events-Extras*
imports
HOL−Probability.Probability
PiE-Rel-Extras
begin

context *prob-space*
begin

lemma *prob-sum-Union*:
assumes *measurable*: *finite A* $A \subseteq$ *events disjoint A*
shows *prob* $(\bigcup A) = (\sum e{\in}A.\ prob\ (e))$
proof −
obtain *f* **where** *bb*: *bij-betw f* $\{0..<card\ A\}$ *A*
using *assms*(*1*) *ex-bij-betw-nat-finite* **by** *auto*
then have *eq*: *f* ‘ $\{0..<card\ A\} = A$
by (*simp add*: *bij-betw-imp-surj-on*)
moreover have *inj-on f* $\{0..<card\ A\}$
using *bb bij-betw-def* **by** *blast*
ultimately have *disjoint-family-on f* $\{0..<card\ A\}$
using *disjoint-image-disjoint-family-on*[*of f* $\{0..<card\ A\}$] *assms* **by** *auto*
moreover have $(\sum e{\in}A.\ prob\ (e)) = (\sum i{\in}\{0..<card\ A\}.\ prob\ (f\ i))$ **using** *sum.reindex bb*
by (*simp add*: *sum.reindex-bij-betw*)
ultimately show *?thesis* **using** *finite-measure-finite-Union eq assms*(*1*) *assms*(*2*)
by (*metis bb bij-betw-finite*)
qed

lemma *events-inter*:
assumes *finite S*
assumes $S \neq \{\}$
shows $(\bigwedge A.\ A \in S \Longrightarrow A \in events) \Longrightarrow \bigcap S \in events$
using *assms* **proof** (*induct S rule*: *finite-ne-induct*)
case (*singleton x*)
then show *?case* **by** *auto*
next
case (*insert x F*)
then show *?case* **using** *sets.Int*
by (*metis complete-lattice-class.Inf-insert insertCI*)
qed

lemma *events-union*:
assumes *finite S*
shows $(\bigwedge A.\ A \in S \Longrightarrow A \in events) \Longrightarrow \bigcup S \in events$
using *assms*(*1*) **proof** (*induct S rule*: *finite-induct*)

case *empty*
then show *?case* **by** *auto*
next
case (*insert x F*)
then show *?case* **using** *sets.Un*
by (*simp add*: *insertI1*)
qed

lemma *prob-inter-set-lt-elem*: $A \in events \Longrightarrow prob\ (A \cap (\bigcap AS)) \leq prob\ A$
by (*simp add*: *finite-measure-mono*)

lemma *Inter-event-ss*: *finite* $A \Longrightarrow A \subseteq events \Longrightarrow A \neq \{\} \Longrightarrow \bigcap A \in events$
by (*simp add*: *events-inter subset-iff*)

lemma *prob-inter-ss-lt*:
assumes *finite A*
assumes $A \subseteq events$
assumes $B \neq \{\}$
assumes $B \subseteq A$
shows $prob\ (\bigcap A) \leq prob\ (\bigcap B)$
proof (*cases* $B = A$)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
then obtain C **where** $C = A - B$ **and** $C \neq \{\}$
using *assms*(*4*) **by** *auto*
then have $\bigcap A = \bigcap C \cap \bigcap B$
by (*metis Inter-Un-distrib Un-Diff-cancel2 assms*(*4*) *sup.orderE*)
moreover have $\bigcap B \in events$ **using** *assms*(*1*) *assms*(*3*) *assms*(*2*) *Inter-event-ss*
by (*meson assms*(*2*) *assms*(*4*) *dual-order.trans finite-subset*)
ultimately show *?thesis* **using** *prob-inter-set-lt-elem*
by (*simp add*: *inf-commute*)
qed

lemma *prob-inter-ss-lt-index*:
assumes *finite A*
assumes $F\ `\ A \subseteq events$
assumes $B \neq \{\}$
assumes $B \subseteq A$
shows $prob\ (\bigcap (F\ `\ A)) \leq prob\ (\bigcap (F\ `B))$
using *prob-inter-ss-lt*[*of* $F\ `\ A\ F\ `\ B$] *assms* **by** *auto*

lemma *space-compl-double*:
assumes $S \subseteq events$
shows $((-)\ (space\ M))\ `\ (((-)\ (space\ M))\ `\ S) = S$
proof (*intro subset-antisym subsetI*)
fix x **assume** $x \in (-)\ (space\ M)\ `\ (-)\ (space\ M)\ `\ S$
then obtain x' **where** *xeq*: $x = space\ M - x'$ **and** $x' \in (-)\ (space\ M)\ `\ S$ **by**

blast
then obtain x'' **where** $x' = space\ M - x''$ **and** *xin*: $x'' \in S$ **by** *blast*
then have $x'' = x$ **using** *xeq assms*
by (*simp add*: *Diff-Diff-Int Set.basic-monos*(*7*))
then show $x \in S$ **using** *xin* **by** *simp*
next
fix x **assume** $x \in S$
then obtain x' **where** *xeq*: $x' = space\ M - x$ **and** $x' \in (-)\ (space\ M)\ `\ S$ **by** *simp*
then have $space\ M - x' \in (-)\ (space\ M)\ `\ (-)\ (space\ M)\ `\ S$ **by** *auto*
moreover have $space\ M - x' = x$ **using** *xeq assms*
by (*simp add*: *Diff-Diff-Int* ‹$x \in S$› *subset-iff*)
ultimately show $x \in (-)\ (space\ M)\ `\ (-)\ (space\ M)\ `\ S$ **by** *simp*
qed

lemma *bij-betw-compl-sets*:
assumes $S \subseteq events$
assumes $S' = ((-)\ (space\ M))\ `\ S$
shows *bij-betw* $((-)\ (space\ M))\ S'\ S$
proof (*intro bij-betwI'*)
show $\bigwedge x\ y.\ x \in S' \Longrightarrow y \in S' \Longrightarrow (space\ M - x = space\ M - y) = (x = y)$
using *assms*(*2*) **by** *blast*
next
show $\bigwedge x.\ x \in S' \Longrightarrow space\ M - x \in S$ **using** *space-compl-double assms* **by** *auto*
next
show $\bigwedge y.\ y \in S \Longrightarrow \exists x{\in}S'.\ y = space\ M - x$ **using** *space-compl-double assms* **by** *auto*
qed

lemma *bij-betw-compl-sets-rev*:
assumes $S \subseteq events$
assumes $S' = ((-)\ (space\ M))\ `\ S$
shows *bij-betw* $((-)\ (space\ M))\ S\ S'$
proof (*intro bij-betwI'*)
show $\bigwedge x\ y.\ x \in S \Longrightarrow y \in S \Longrightarrow (space\ M - x = space\ M - y) = (x = y)$
using *assms* **by** (*metis Diff-Diff-Int sets.Int-space-eq1 subset-eq*)
next
show $\bigwedge x.\ x \in S \Longrightarrow space\ M - x \in S'$ **using** *space-compl-double assms* **by** *auto*
next
show $\bigwedge y.\ y \in S' \Longrightarrow \exists x{\in}S.\ y = space\ M - x$ **using** *space-compl-double assms* **by** *auto*
qed

lemma *prob0-basic-inter*: $A \in events \Longrightarrow B \in events \Longrightarrow prob\ A = 0 \Longrightarrow prob\ (A \cap B) = 0$
by (*metis Int-lower1 finite-measure-mono measure-le-0-iff*)

lemma *prob0-basic-Inter*: $A \in events \Longrightarrow B \subseteq events \Longrightarrow prob\ A = 0 \Longrightarrow prob\ (A \cap (\bigcap B)) = 0$

by (*metis Int-lower1 finite-measure-mono measure-le-0-iff*)

lemma *prob1-basic-inter*: $A \in events \Longrightarrow B \in events \Longrightarrow prob\ A = 1 \Longrightarrow prob\ (A \cap B) = prob\ B$
by (*metis inf-commute measure-space-inter prob-space*)

lemma *prob1-basic-Inter*:
assumes $A \in events\ B \subseteq events$
assumes $prob\ A = 1$
assumes $B \neq \{\}$
assumes *finite B*
shows $prob\ (A \cap (\bigcap B)) = prob\ (\bigcap B)$
proof −
have $\bigcap B \in events$ **using** *Inter-event-ss assms* **by** *auto*
then show *?thesis* **using** *assms prob1-basic-inter* **by** *auto*
qed

lemma *compl-identity*: $A \in events \Longrightarrow space\ M - (space\ M - A) = A$
by (*simp add*: *double-diff sets.sets-into-space*)

lemma *prob-addition-rule*: $A \in events \Longrightarrow B \in events \Longrightarrow$
$prob\ (A \cup B) = prob\ A + prob\ B - prob\ (A \cap B)$
by (*simp add*: *finite-measure-Diff' finite-measure-Union' inf-commute*)

lemma *compl-subset-in-events*: $S \subseteq events \Longrightarrow (-)\ (space\ M)\ `\ S \subseteq events$
by *auto*

lemma *prob-compl-diff-inter*: $A \in events \Longrightarrow B \in events \Longrightarrow$
$prob\ (A \cap (space\ M - B)) = prob\ A - prob\ (A \cap B)$
by (*simp add*: *Diff-Int-distrib finite-measure-Diff sets.Int*)

lemma *bij-betw-prod-prob*: $bij\text{-}betw\ f\ A\ B \Longrightarrow (\prod b{\in}B.\ prob\ b) = (\prod a{\in}A.\ prob\ (f\ a))$
by (*simp add*: *prod.reindex-bij-betw*)

definition *event-compl* :: *'a set* ⇒ *'a set* **where**
event-compl A ≡ *space M* − *A*

lemma *compl-Union*: $A \neq \{\} \Longrightarrow space\ M - (\bigcup A) = (\bigcap a \in A\ .\ (space\ M - a))$
by (*simp*)

lemma *compl-Union-fn*: $A \neq \{\} \Longrightarrow space\ M - (\bigcup (F\ `\ A)) = (\bigcap a \in A\ .\ (space\ M - F\ a))$
by (*simp*)

end

Reasoning on the probability of function sets

lemma *card-PiE-val-ss-eq*:
assumes *finite A*
assumes $b \in B$
assumes $d \subseteq A$
assumes $B \neq \{\}$
assumes *finite B*
shows *card* $\{f \in (A \to_E B)$. $(\forall\ v \in d$.*f v* $= b)\}$/*card* $(A \to_E B)$ $= 1/((card\ B)\ powi\ (card\ d))$
(**is** *card* $\{f \in ?C$. $(\forall\ v \in d$.*f v* $= b)\}$/*card ?C* $= 1/((card\ B)\ powi\ (card\ d)))$
proof −
have *lt*: *card d* $\leq$ *card A*
by (*simp add*: *card-mono assms*(*1*) *assms*(*3*))
then have *scard*: *card* $\{f \in ?C$. $\forall\ v \in d$. *f v* $= b\}$ $= (card\ B)\ powi\ ((card\ A) - card\ d)$
using *assms card-PiE-filter-range-set-const*[*of b B d A*] *assms*
by (*simp flip*: *of-nat-diff*)
have *Ccard*: *card ?C* $= (card\ B)\ powi\ (card\ A)$ **using** *card-funcsetE assms*(*2*) *assms*(*1*) **by** *auto*
have *bgt*: *card B* $\neq$ *0* **using** *assms*(*5*) *assms*(*4*) **by** *auto*
have *card* $\{f \in ?C$. $\forall\ v \in d$. *f v* $= b\}$/ $(card\ ?C) =$
$((card\ B)\ powi\ ((card\ A) - card\ d))/((card\ B)\ powi\ (card\ A))$
using *Ccard scard* **by** *simp*
also have ... $= (card\ B)\ powi\ (int\ (card\ A - card\ d) - int\ (card\ A))$
using *bgt* **by** (*simp add*: *power-int-diff*)
also have ... $=$ *inverse* $((card\ B)\ powi\ (card\ d))$
using *power-int-minus*[*of card B* (*int* (*card d*))] **by** (*simp add*: *lt*)
finally show *?thesis* **by** (*simp add*: *inverse-eq-divide*)
qed

lemma *card-PiE-val-indiv-eq*:
assumes *finite A*
assumes $b \in B$
assumes $d \in A$
assumes $B \neq \{\}$
assumes *finite B*
shows *card* $\{f \in (A \to_E B)$. *f d* $= b\}$/*card* $(A \to_E B)$ $= 1/(card\ B)$
(**is** *card* $\{f \in ?C$.*f d* $= b\}$/*card ?C* $= 1/(card\ B))$
proof −
have $\{d\} \subseteq A$ **using** *assms*(*3*) **by** *simp*
moreover have $\bigwedge f$. $f \in ?C \Longrightarrow f\ d = b \longleftrightarrow (\forall\ d' \in \{d\}.\ f\ d' = b)$ **by** *auto*
ultimately have *card* $\{f \in ?C$.*f d* $= b\}$/*card ?C* $= 1/((card\ B)\ powi\ (card\ \{d\}))$
using *card-PiE-val-ss-eq*[*of A b B* $\{d\}$] *assms* **by** *auto*
also have ... $= 1/((card\ B)\ powi\ 1)$ **by** *auto*
finally show *?thesis* **by** *simp*
qed

lemma *prob-uniform-ex-fun-space*:
assumes *finite A*

assumes $b \in B$
assumes $d \subseteq A$
assumes $B \neq \{\}$
assumes $A \neq \{\}$
assumes *finite B*
shows *prob-space.prob* (*uniform-count-measure* ($A \rightarrow_E B$)) $\{f \in (A \rightarrow_E B)$. $(\forall v \in d$. $f\ v = b)\} =$
$1/((card\ B)\ powi\ (card\ d))$
proof −
let *?C* = ($A \rightarrow_E B$)
let *?M* = *uniform-count-measure ?C*
have *finC*: *finite ?C* **using** *assms*(*2*) *assms*(*6*) *assms*(*1*)
by (*simp add*: *finite-PiE*)
moreover have *?C* $\neq \{\}$ **using** *assms*(*4*) *assms*(*1*)
by (*simp add*: *PiE-eq-empty-iff*)
ultimately interpret *P*: *prob-space ?M*
using *assms*(*3*) **by** (*simp add*: *prob-space-uniform-count-measure*)
have *P.prob* $\{f \in$ *?C* . $\forall\ v \in d$. $f\ v = b\}$ = *card* $\{f \in$ *?C* . $\forall\ v \in d$. $f\ v = b\}$/(*card ?C*)
using *measure-uniform-count-measure*[*of ?C* $\{f \in$ *?C* . $\forall\ v \in d$. $f\ v = b\}$] *finC assms*(*3*)
by *fastforce*
then show *?thesis* **using** *card-PiE-val-ss-eq assms* **by** (*simp*)
qed

proposition *integrable-uniform-count-measure-finite*:
fixes *g* :: *'a* ⇒ *'b*::{*banach*, *second-countable-topology*}
shows *finite A* ⟹ *integrable* (*uniform-count-measure A*) *g*
unfolding *uniform-count-measure-def*
using *integrable-point-measure-finite* **by** *fastforce*

end

4 Conditional Probability Library Extensions

theory *Cond-Prob-Extensions*
imports
Prob-Events-Extras
Design-Theory.Multisets-Extras
begin

4.1 Miscellaneous Set and List Lemmas

lemma *nth-image-tl*:
assumes *xs* ≠ []
shows *nth xs* ‘ {*1*..<*length xs*} = *set*(*tl xs*)
proof −
have *set* (*tl xs*) = {(*tl xs*)!*i* | *i*. *i* < *length* (*tl xs*)}
using *set-conv-nth* **by** *metis*

```
  then have set (tl xs) = {xs! (Suc i) | i. i  < length xs − 1}
    using nth-tl by fastforce
  then have set (tl xs) = {xs ! j | j. j > 0 ∧ j < length xs}
   by (smt (verit, best) Collect-cong Suc-diff-1 Suc-less-eq assms length-greater-0-conv
zero-less-Suc)
  thus ?thesis by auto
qed

lemma exists-list-card:
  assumes finite S
  obtains xs where set xs = S and length xs = card S
  by (metis assms distinct-card finite-distinct-list)

lemma bij-betw-inter-empty:
  assumes bij-betw f A B
  assumes A' ⊆ A
  assumes A'' ⊆ A
  assumes A' ∩ A'' = {}
  shows f ‘ A' ∩ f ‘ A'' = {}
 by (metis assms(1) assms(2) assms(3) assms(4) bij-betw-inter-subsets image-empty)


lemma bij-betw-image-comp-eq:
  assumes bij-betw g T S
  shows (F ∘ g) ‘ T = F ‘ S
  using assms bij-betw-imp-surj-on by (metis image-comp)

lemma prod-card-image-set-eq:
  assumes bij-betw f {0..<card S} S
  assumes finite S
  shows (∏ i ∈ {n..<(card S)} . g (f i)) = (∏ i ∈ f ‘ {n..<card S} . g i)
proof (cases n ≥ card S)
  case True
  then show ?thesis by simp
next
  case False
  then show ?thesis using assms
  proof (induct card S arbitrary: S)
    case 0
    then show ?case by auto
  next
    case (Suc x)
    then have nlt: n < Suc x by simp
    then have split: {n..<Suc x} = {n..<x} ∪ {x} by auto
    then have f ‘ {n..<Suc x} = f ‘ ({n..< x} ∪ {x}) by simp
    then have fsplit: f ‘ {n..<Suc x} = f ‘ {n..< x} ∪ {f x}
      by simp
    have {n..<x} ⊆ {0..<card S}
      using Suc(2)  by auto
```

moreover have $\{x\} \subseteq \{0..<card\ S\}$ **using** *Suc(2)* **by** *auto*
moreover have $\{n..< x\} \cap \{x\} = \{\}$ **by** *auto*
ultimately have *finter*: $f\ `\ \{n..< x\} \cap \{f\ x\} = \{\}$ **using** *Suc.prems(2) Suc.prems(1)*
bij-betw-inter-empty[*of f* $\{0..<card\ S\}$ *S* $\{n..< x\}$ $\{x\}$] **by** *auto*
have $(\prod i = n..<Suc\ x.\ g\ (f\ i)) = (\prod i = n..<x.\ g\ (f\ i)) * g\ (f(\ x))$ **using** *nlt* **by** *simp*
moreover have $(\prod x \in f\ `\ \{n..<Suc\ x\}.\ g\ x) = (\prod i \in f\ `\ \{n..< x\}.\ g\ i) * g\ (f\ x)$ **using** *finter fsplit*
by (*simp add*: *Groups.mult-ac(2)*)
moreover have $(\prod i \in f\ `\ \{n..< x\}.\ g\ i) = (\prod i = n..<x.\ g\ (f\ i))$
proof −
let *?S'* = $f\ `\ \{0..<x\}$
have $\{0..<x\} \subseteq \{0..<card\ S\}$ **using** *Suc(2)* **by** *auto*
then have *bij*: *bij-betw f* $\{0..<x\}$ *?S'* **using** *Suc.prems(2)*
using *bij-betw-subset* **by** *blast*
moreover have *card ?S'* = *x* **using** *bij-betw-same-card*[*of f* $\{0..<x\}$ *?S'*] *bij* **by** *auto*
moreover have *finite ?S'* **using** *finite-subset* **by** *auto*
ultimately show *?thesis*
by (*metis bij-betw-subset ivl-subset less-eq-nat.simps(1) order-refl prod.reindex-bij-betw*)

qed
ultimately show *?case* **using** *Suc(2)* **by** *auto*
qed
qed

lemma *set-take-distinct-elem-not*:
assumes *distinct xs*
assumes *i* < *length xs*
shows *xs* ! *i* ∉ *set* (*take i xs*)
by (*metis assms(1) assms(2) distinct-take id-take-nth-drop not-distinct-conv-prefix*)

4.2 Conditional Probability Basics

context *prob-space*
begin

Abbreviation to mirror mathematical notations

abbreviation *cond-prob-ev* :: *'a set* ⇒ *'a set* ⇒ *real* (‹$\mathcal{P}$'(- | -')›) **where**
$\mathcal{P}(B \mid A) \equiv \mathcal{P}(x\ in\ M.\ (x \in B) \mid (x \in A))$

lemma *cond-prob-inter*: $\mathcal{P}(B \mid A) = \mathcal{P}(\omega\ in\ M.\ (\omega \in B \cap A)) / \mathcal{P}(\omega\ in\ M.\ (\omega \in A))$
using *cond-prob-def* **by** *auto*

lemma *cond-prob-ev-def*:
assumes *A* ∈ *events B* ∈ *events*
shows $\mathcal{P}(B \mid A) = prob\ (A \cap B) / prob\ A$
proof −

have *a*: $\mathcal{P}(B \mid A) = \mathcal{P}(\omega \text{ in } M.\ (\omega \in B \cap A)) \ / \ \mathcal{P}(\omega \text{ in } M.\ (\omega \in A))$
using *cond-prob-inter* **by** *auto*
also have ... = *prob* $\{w \in$ *space* M . $w \in B \cap A\}$ / *prob* $\{w \in$ *space* M . $w \in A\}$
by *auto*
finally show *?thesis* **using** *assms*
by (*simp add*: *Collect-conj-eq a inf-commute*)
qed

lemma *measurable-in-ev*:
assumes $A \in$ *events*
shows *Measurable.pred* M $(\lambda\ x\ .\ x \in A)$
using *assms* **by** *auto*

lemma *measure-uniform-measure-eq-cond-prob-ev*:
assumes $A \in$ *events* $B \in$ *events*
shows $\mathcal{P}(A \mid B) =$$\mathcal{P}(x$ *in uniform-measure* M $\{x \in$ *space* $M.\ x \in B\}.\ x \in A)$
using *assms measurable-in-ev measure-uniform-measure-eq-cond-prob* **by** *auto*

lemma *measure-uniform-measure-eq-cond-prob-ev2*:
assumes $A \in$ *events* $B \in$ *events*
shows $\mathcal{P}(A \mid B) =$ *measure* (*uniform-measure* M $\{x \in$ *space* $M.\ x \in B\}$) A
using *measure-uniform-measure-eq-cond-prob-ev assms*
by (*metis Int-def sets.Int-space-eq1 space-uniform-measure*)

lemma *measure-uniform-measure-eq-cond-prob-ev3*:
assumes $A \in$ *events* $B \in$ *events*
shows $\mathcal{P}(A \mid B) =$ *measure* (*uniform-measure* M B) A
using *measure-uniform-measure-eq-cond-prob-ev assms Int-def sets.Int-space-eq1 space-uniform-measure*
by *metis*

lemma *prob-space-cond-prob-uniform*:
assumes *prob* $(\{x \in$ *space* $M.\ Q\ x\}) > 0$
shows *prob-space* (*uniform-measure* M $\{x \in$ *space* $M.\ Q\ x\}$)
using *assms* **by** (*intro prob-space-uniform-measure*) (*simp-all add*: *emeasure-eq-measure*)

lemma *prob-space-cond-prob-event*:
assumes *prob* $B > 0$
shows *prob-space* (*uniform-measure* M B)
using *assms* **by** (*intro prob-space-uniform-measure*) (*simp-all add*: *emeasure-eq-measure*)

Note this case shouldn't be used. Conditional probability should have $>$ 0 assumption

lemma *cond-prob-empty*: $\mathcal{P}(B \mid \{\}) = 0$
using *cond-prob-inter*[*of* B $\{\}$] **by** *auto*

lemma *cond-prob-space*: $\mathcal{P}(A \mid$ *space* $M) = \mathcal{P}(w$ *in* M . $w \in A)$
proof −
have *p1*: *prob* $\{\omega \in$ *space* $M.\ \omega \in$ *space* $M\} = 1$

by (*simp add*: *prob-space*)
have $\bigwedge$ *w*. *w* ∈ *space M* ⟹ *w* ∈ *A* ∩ (*space M*) ⟷ *w* ∈ *A* **by** *auto*
then have *prob* {ω ∈ *space M*. ω ∈ *A* ∩ *space M*} = $\mathcal{P}$(*w in M* . *w* ∈ *A*)
by *meson*
then show *?thesis* **using** *cond-prob-inter*[*of A space M*] *p1* **by** *auto*
qed

lemma *cond-prob-space-ev*: **assumes** *A* ∈ *events* **shows** $\mathcal{P}$(*A* | *space M*) = *prob A*
using *cond-prob-space assms*
by (*metis Int-commute Int-def measure-space-inter sets.top*)

lemma *cond-prob-UNIV*: $\mathcal{P}$(*A* | *UNIV*) = $\mathcal{P}$(*w in M* . *w* ∈ *A*)
proof −
have *p1*: *prob* {ω ∈ *space M*. ω ∈ *UNIV*} = *1*
by (*simp add*: *prob-space*)
have $\bigwedge$ *w*. *w* ∈ *space M* ⟹ *w* ∈ *A* ∩ *UNIV* ⟷ *w* ∈ *A* **by** *auto*
then have *prob* {ω ∈ *space M*. ω ∈ *A* ∩ *UNIV*} = $\mathcal{P}$(*w in M* . *w* ∈ *A*)
by *meson*
then show *?thesis* **using** *cond-prob-inter*[*of A UNIV*] *p1* **by** *auto*
qed

lemma *cond-prob-UNIV-ev*: *A* ∈ *events* ⟹ $\mathcal{P}$(*A* | *UNIV*) = *prob A*
using *cond-prob-UNIV*
by (*metis Int-commute Int-def measure-space-inter sets.top*)

lemma *cond-prob-neg*:
assumes *A* ∈ *events B* ∈ *events*
assumes *prob A >0*
shows $\mathcal{P}$((*space M* − *B*) | *A*) = *1* − $\mathcal{P}$(*B* | *A*)
proof −
have *negB*: *space M* − *B* ∈ *events* **using** *assms* **by** *auto*
have *prob* ((*space M* − *B*) ∩ *A*) = *prob A* − *prob* (*B* ∩ *A*)
by (*simp add*: *Diff-Int-distrib2 assms*(*1*) *assms*(*2*) *finite-measure-Diff sets.Int*)
then have $\mathcal{P}$((*space M* − *B*) | *A*) = (*prob A* − *prob* (*B* ∩ *A*))/*prob A*
using *cond-prob-ev-def*[*of A space M* − *B*] *assms negB* **by** (*simp add*: *Int-commute*)

also have ... = ((*prob A*)/*prob A*) − ((*prob* (*B* ∩ *A*))/*prob A*) **by** (*simp add*: *field-simps*)
also have ... = *1* − ((*prob* (*B* ∩ *A*))/*prob A*) **using** *assms*(*3*) **by** (*simp add*: *field-simps*)
finally show $\mathcal{P}$((*space M* − *B*) | *A*) = *1* − $\mathcal{P}$(*B* | *A*) **using** *cond-prob-ev-def*[*of A B*] *assms*
by (*simp add*: *inf-commute*)
qed

4.3 Bayes Theorem

lemma *prob-intersect-A*:

assumes $A \in events$ $B \in events$
shows $prob\ (A \cap B) = prob\ A * \mathcal{P}(B \mid A)$
using *cond-prob-ev-def assms* **apply** *simp*
by (*metis Int-lower1 finite-measure-mono measure-le-0-iff*)

lemma *prob-intersect-B*:
assumes $A \in events$ $B \in events$
shows $prob\ (A \cap B) = prob\ B * \mathcal{P}(A \mid B)$
using *cond-prob-ev-def assms*
by (*simp-all add*: *inf-commute*)(*metis Int-lower2 finite-measure-mono measure-le-0-iff*)

theorem *Bayes-theorem*:
assumes $A \in events$ $B \in events$
shows $prob\ B * \mathcal{P}(A \mid B) = prob\ A * \mathcal{P}(B \mid A)$
using *prob-intersect-A prob-intersect-B assms* **by** *simp*

corollary *Bayes-theorem-div*:
assumes $A \in events$ $B \in events$
shows $\mathcal{P}(A \mid B) = (prob\ A * \mathcal{P}(B \mid A))/(prob\ B)$
using *assms Bayes-theorem*
by (*metis cond-prob-ev-def prob-intersect-A*)

lemma *cond-prob-dual-intersect*:
assumes $A \in events$ $B \in events$ $C \in events$
assumes $prob\ C \neq 0$
shows $\mathcal{P}(A \mid (B \cap C)) = \mathcal{P}(A \cap B \mid C)/\ \mathcal{P}(B \mid C)$ (**is** *?LHS = ?RHS*)
proof −
have $B \cap C \in events$ **using** *assms* **by** *auto*
then have *lhs*: *?LHS* = $prob\ (A \cap B \cap C)/prob\ (B \cap C)$
using *assms cond-prob-ev-def*[*of* $B \cap C$ A] *inf-commute inf-left-commute* **by** (*metis*)
have $A \cap B \in events$ **using** *assms* **by** *auto*
then have $\mathcal{P}(A \cap B \mid C) = prob\ (A \cap B \cap C)\ /\ prob\ C$
using *assms cond-prob-ev-def*[*of* C $A \cap B$] *inf-commute* **by** (*metis*)
moreover have $\mathcal{P}(B \mid C) = prob\ (B \cap C)/prob\ C$ **using** *cond-prob-ev-def*[*of* C B] *assms inf-commute* **by** *metis*
ultimately have *?RHS* = $(prob\ (A \cap B \cap C)\ /\ prob\ C)/(\ prob\ (B \cap C)/prob\ C)$
by *simp*
also have ... = $(prob\ (A \cap B \cap C)\ /\ prob\ C)*(\ prob\ (C)/prob\ (B \cap C))$ **by** *simp*
also have ... = $prob\ (A \cap B \cap C)/prob\ (B \cap C)$ **using** *assms*(*4*) **by** *simp*
finally show *?thesis* **using** *lhs* **by** *simp*
qed

lemma *cond-prob-ev-double*:
assumes $A \in events$ $B \in events$ $C \in events$
assumes $prob\ C > 0$

```
  shows P(x in (uniform-measure M C). (x ∈ A) | (x ∈ B)) =  P(A | (B ∩ C))
proof −
  let ?M = uniform-measure M C
   interpret cps: prob-space ?M using assms(4) prob-space-cond-prob-event by
auto
  have probne: prob C ≠ 0 using assms(4) by auto
  have ev: cps.events = events using sets-uniform-measure by auto
  have iev: A ∩ B ∈ events using assms(1) assms(2) by simp
  have 0: P(x in (uniform-measure M C). (x ∈ A) | (x ∈ B)) = cps.cond-prob-ev
A B by simp
 also have 1: ... = (measure ?M (A ∩ B))/(measure ?M B) using cond-prob-ev-def
assms(1) assms(2) ev
    by (metis Int-commute cps.cond-prob-ev-def)
  also have 2: ... = P((A ∩ B) | C)/(measure ?M B)
   using measure-uniform-measure-eq-cond-prob-ev3[of A ∩ B C] assms(3) iev by
auto
 also have 3: ... = P((A ∩ B) | C)/ P(B | C) using measure-uniform-measure-eq-cond-prob-ev3[of
B C] assms(3) assms(2) by auto
  also have 4: ... = P(A | (B ∩ C))
    using cond-prob-dual-intersect[of A B C] assms(1) assms(2) assms(3) probne
by presburger
  finally show ?thesis using 1 2 3 4 by presburger
qed

lemma cond-prob-inter-set-lt:
  assumes A ∈ events B ∈ events AS ⊆ events
  assumes finite AS
  shows P((A ∩ (⋂ AS)) | B) ≤ P(A|B) (is ?LHS ≤ ?RHS)
using measure-uniform-measure-eq-cond-prob-ev finite-measure-mono
proof (cases AS = {})
  case True
  then have (A ∩ (⋂ AS)) = A by simp
  then show ?thesis by simp
next
  case False
  then have (⋂ AS) ∈ events using assms(3) assms(4) Inter-event-ss by simp
  then have (A ∩ (⋂ AS)) ∈ events using assms by simp
  then have ?LHS = prob (A ∩ (⋂ AS) ∩ B)/prob B
    using assms cond-prob-ev-def[of B (A ∩ (⋂ AS))] inf-commute by metis
 moreover have prob (A ∩ (⋂ AS) ∩ B) ≤ prob (A ∩ B) using finite-measure-mono
     assms(1) inf-commute inf-left-commute by (metis assms(2) inf-sup-ord(1)
sets.Int)
  ultimately show ?thesis using cond-prob-ev-def[of B A]
    by (simp add: assms(1) assms(2) divide-right-mono inf-commute)
qed
```

4.4 Conditional Probability Multiplication Rule

Many list and indexed variations of this lemma

lemma *prob-cond-Inter-List*:
assumes $xs \neq []$
assumes $\bigwedge A.\ A \in set\ xs \Longrightarrow A \in events$
shows $prob\ (\bigcap (set\ xs)) = prob\ (hd\ xs) * (\prod i = 1..<(length\ xs)\ .$
$\mathcal{P}((xs\ !\ i)\ |\ (\bigcap (set\ (take\ i\ xs)))))$
using *assms(1) assms(2)*
proof (*induct xs rule*: *rev-nonempty-induct*)
case (*single x*)
then show *?case* **by** *auto*
next
case (*snoc x xs*)
have $xs \neq []$
by (*simp add*: *snoc.hyps(1)*)
then have *inev*: $(\bigcap (set\ xs)) \in events$ **using** *events-inter*
by (*simp add*: *snoc.prems*)
have *len*: $(length\ (xs\ @\ [x])) = length\ xs + 1$ **by** *auto*
have *last-p*: $\mathcal{P}(x\ |\ (\bigcap (set\ xs))) =$
$\mathcal{P}((xs\ @\ [x])\ !\ length\ xs\ |\ \bigcap\ (set\ (take\ (length\ xs)\ (xs\ @\ [x]))))$
by *auto*
have $prob\ (\bigcap\ (set\ (xs\ @\ [x]))) = prob\ (x \cap (\bigcap (set\ xs)))$
by *auto*
also have $... = prob\ (\bigcap (set\ xs)) * \mathcal{P}(x\ |\ (\bigcap (set\ xs)))$
using *prob-intersect-B snoc.prems inev* **by** *simp*
also have $... = prob\ (hd\ xs) * (\prod i = 1..<length\ xs.\ \mathcal{P}(xs\ !\ i\ |\ \bigcap\ (set\ (take\ i\ xs)))) *$
$\mathcal{P}(x\ |\ (\bigcap (set\ xs)))$
using *snoc.hyps snoc.prems* **by** *auto*
finally have $prob\ (\bigcap\ (set\ (xs\ @\ [x]))) = prob\ (hd\ (xs\ @[x])) *$
$(\prod i = 1..<length\ xs.\ \mathcal{P}((xs\ @\ [x])\ !\ i\ |\ \bigcap\ (set\ (take\ i\ (xs\ @\ [x]))))) *\ \mathcal{P}(x\ |\ (\bigcap (set\ xs)))$
using *nth-append*[*of xs* [*x*]] *nth-take* **by** (*simp add*: *snoc.hyps(1)*)
then show *?case* **using** *last-p* **by** *auto*
qed

lemma *prob-cond-Inter-index*:
fixes $n :: nat$
assumes $n > 0$
assumes $F\ `\ \{0..<n\} \subseteq events$
shows $prob\ (\bigcap\ (F\ `\{0..<n\}\ )) = prob\ (F\ 0) * (\prod i \in \{1..<n\}\ .$
$\mathcal{P}(F\ i\ |\ (\bigcap\ (F`\{0..<i\}))))$
proof −
define *xs* **where** $xs \equiv map\ F\ [0..<n]$
have $prob\ (\bigcap (set\ xs)) = prob\ (hd\ xs) * (\prod i = 1..<(length\ xs)\ .$
$\mathcal{P}((xs\ !\ i)\ |\ (\bigcap (set\ (take\ i\ xs)))))$ **using** *xs-def assms prob-cond-Inter-List*[*of xs*] **by** *auto*
then have $prob\ (\bigcap (set\ xs)) = prob\ (hd\ xs) * (\prod i \in \{1..<n\}\ .\ \mathcal{P}((xs\ !\ i)\ |\ (\bigcap (set\ (take\ i\ xs)))))$
using *xs-def* **by** *auto*
moreover have $hd\ xs = F\ 0$

unfolding *xs-def* **by** (*simp add*: *assms*(*1*) *hd-map*)
moreover have $\bigwedge i.\ i \in \{1..<n\} \Longrightarrow F\ `\ \{0..<i\} = set\ (take\ i\ xs\ )$
by (*metis atLeastLessThan-iff atLeastLessThan-upt image-set less-or-eq-imp-le plus-nat.add-0*
take-map take-upt xs-def)
ultimately show *?thesis* **using** *xs-def* **by** *auto*
qed

lemma *prob-cond-Inter-index-compl*:
fixes $n :: nat$
assumes $n > 0$
assumes $F\ `\ \{0..<n\} \subseteq events$
shows $prob\ (\bigcap\ x \in \{0..<n\}\ .\ space\ M - F\ x) = prob\ (space\ M - F\ 0) * (\prod i \in \{1..<n\}\ .$
$\mathcal{P}(space\ M - F\ i \mid (\bigcap\ j \in \{0..<i\}.\ space\ M - F\ j)))$
proof −
define G **where** $G \equiv \lambda\ i.\ space\ M - F\ i$
then have $G\ `\ \{0..<n\} \subseteq events$ **using** *assms*(*2*) **by** *auto*
then show *?thesis* **using** *prob-cond-Inter-index*[*of n G*] *G-def*
using *assms*(*1*) **by** *blast*
qed

lemma *prob-cond-Inter-take-cond*:
assumes $xs \neq []$
assumes $set\ xs \subseteq events$
assumes $S \subseteq events$
assumes $S \neq \{\}$
assumes *finite S*
assumes $prob\ (\bigcap S) > 0$
shows $\mathcal{P}((\bigcap (set\ xs)) \mid (\bigcap\ S)) = (\prod i = 0..<(length\ xs)\ .\ \mathcal{P}((xs\ !\ i) \mid (\bigcap (set\ (take\ i\ xs\ ) \cup S))))$
proof −
define M' **where** $M' = uniform\text{-}measure\ M\ (\bigcap S)$
interpret *cps*: *prob-space* M' **using** *prob-space-cond-prob-event M'-def assms*(*6*) **by** *auto*
have *len*: $length\ xs > 0$ **using** *assms*(*1*) **by** *simp*
have *cps-ev*: $cps.events = events$ **using** *sets-uniform-measure M'-def* **by** *auto*
have *sevents*: $\bigcap S \in events$ **using** *assms*(*3*) *assms*(*4*) *Inter-event-ss assms*(*5*) **by** *auto*
have *fin*: *finite* (*set xs*) **by** *auto*
then have *xevents*: $\bigcap (set\ xs) \in events$ **using** *assms*(*1*) *assms*(*2*) *Inter-event-ss* **by** *blast*
then have *peq*: $\mathcal{P}((\bigcap (set\ xs)) \mid (\bigcap\ S)) = cps.prob\ (\bigcap\ (set\ xs))$
using *measure-uniform-measure-eq-cond-prob-ev3*[*of* $\bigcap (set\ xs)\ \bigcap S$] *sevents M'-def*
by *blast*
then have $cps.prob\ (\bigcap\ (set\ xs)) = cps.prob\ (hd\ xs) * (\prod i = 1..<(length\ xs)\ .$
$cps.cond\text{-}prob\text{-}ev\ (xs\ !\ i)\ (\bigcap (set\ (take\ i\ xs\ ))))$ **using** *assms cps.prob-cond-Inter-List*

cps-ev
by *blast*
moreover have *cps.prob* (*hd xs*) = $\mathcal{P}$((*xs* ! *0*) | ($\bigcap$(*set* (*take 0 xs*) ∪ *S*)))
proof −
have *ev*: *hd xs* ∈ *events* **using** *assms*(*2*) *len* **by** *auto*
then have *cps.prob* (*hd xs*) = $\mathcal{P}$(*hd xs* | $\bigcap$ *S*)
using *ev sevents measure-uniform-measure-eq-cond-prob-ev3*[*of hd xs* $\bigcap$ *S*] *M′-def* **by** *presburger*
then show *?thesis* **using** *len* **by** (*simp add*: *hd-conv-nth*)
qed
moreover have $\bigwedge$*i*. *i* > *0* ⟹ *i* < *length xs* ⟹
cps.cond-prob-ev (*xs* ! *i*) ($\bigcap$(*set* (*take i xs*))) = $\mathcal{P}$((*xs* ! *i*) | ($\bigcap$(*set* (*take i xs*) ∪ *S*)))
proof −
fix *i* **assume** *igt*: *i* > *0* **and** *ilt*: *i* <*length xs*
then have *set* (*take i xs*) ⊆ *events* **using** *assms*(*2*)
by (*meson set-take-subset subset-trans*)
moreover have *set* (*take i xs*) ≠ {} **using** *len igt ilt* **by** *auto*
ultimately have ($\bigcap$(*set* (*take i xs*))) ∈ *events*
using *Inter-event-ss fin* **by** *auto*
moreover have *xs* ! *i* ∈ *events* **using** *assms*(*2*)
using *nth-mem subset-iff igt ilt* **by** *blast*
moreover have ($\bigcap$(*set* (*take i xs*) ∪ *S*)) = ($\bigcap$(*set* (*take i xs*))) ∩ ($\bigcap$ *S*)
by (*simp add*: *Inf-union-distrib*)
ultimately show *cps.cond-prob-ev* (*xs* ! *i*) ($\bigcap$(*set* (*take i xs*))) = $\mathcal{P}$((*xs* ! *i*) | ($\bigcap$(*set* (*take i xs*) ∪ *S*)))
using *sevents cond-prob-ev-double*[*of xs* ! *i* ($\bigcap$(*set* (*take i xs*))) $\bigcap$ *S*] *assms*(*6*) *M′-def* **by** *presburger*
qed
ultimately have *eq*: *cps.prob* ($\bigcap$ (*set xs*)) = $\mathcal{P}$((*xs* ! *0*) | ($\bigcap$(*set* (*take 0 xs*) ∪ *S*))) ∗ ($\prod$ *i* ∈ {*1*..<(*length xs*)} .
$\mathcal{P}$((*xs* ! *i*) | ($\bigcap$(*set* (*take i xs*) ∪ *S*)))) **by** *simp*
moreover have {*1*..<*length xs*} = {*0*..<*length xs*} − {*0*}
by (*simp add*: *atLeast1-lessThan-eq-remove0 lessThan-atLeast0*)
moreover have *finite* {*0*..<*length xs*} **by** *auto*
moreover have *0* ∈ {*0*..<*length xs*}**by** (*simp add*: *assms*(*1*))
ultimately have $\mathcal{P}$((*xs* ! *0*) | ($\bigcap$(*set* (*take 0 xs*) ∪ *S*))) ∗ ($\prod$ *i* ∈ {*1*..<(*length xs*)} .
$\mathcal{P}$((*xs* ! *i*) | ($\bigcap$(*set* (*take i xs*) ∪ *S*)))) = ($\prod$ *i* ∈ {*0*..<(*length xs*)} .
$\mathcal{P}$((*xs* ! *i*) | ($\bigcap$(*set* (*take i xs*) ∪ *S*)))) **using** *prod.remove*[*of* {*0*..<*length xs*} *0* λ *i*. $\mathcal{P}$((*xs* ! *i*) | ($\bigcap$(*set* (*take i xs*) ∪ *S*)))]
by *presburger*
then have *cps.prob* ($\bigcap$ (*set xs*)) = ($\prod$ *i* ∈ {*0*..<(*length xs*)} .
$\mathcal{P}$((*xs* ! *i*) | ($\bigcap$(*set* (*take i xs*) ∪ *S*)))) **using** *eq* **by** *simp*
then show *?thesis* **using** *peq* **by** *auto*
qed

lemma *prob-cond-Inter-index-cond-set*:
fixes *n* :: *nat*

assumes $n > 0$
assumes *finite* E
assumes $E \neq \{\}$
assumes $E \subseteq$ *events*
assumes F ' $\{0..<n\} \subseteq$ *events*
assumes *prob* $(\bigcap E) > 0$
shows $\mathcal{P}((\bigcap (F \text{ ' } \{0..<n\})) \mid (\bigcap E)) = (\prod i \in \{0..<n\}.\ \mathcal{P}(F\ i \mid (\bigcap ((F \text{ ' } \{0..<i\}) \cup E))))$
proof −
define M' **where** $M' =$ *uniform-measure* M $(\bigcap E)$
interpret *cps*: *prob-space* M' **using** *prob-space-cond-prob-event* M'*-def assms*(6) **by** *auto*
have *cps-ev*: *cps.events* = *events* **using** *sets-uniform-measure* M'*-def* **by** *auto*
have *sevents*: $(\bigcap (E)) \in$ *events* **using** *assms*(6) *assms*(2) *assms*(3) *assms*(4) *Inter-event-ss* **by** *auto*
have *fin*: *finite* $(F \text{ ' } \{0..<n\})$ **by** *auto*
then have *xevents*: $\bigcap (F \text{ ' } \{0..<n\}) \in$ *events* **using** *assms Inter-event-ss* **by** *auto*
then have *peq*: $\mathcal{P}((\bigcap (F \text{ ' } \{0..<n\})) \mid (\bigcap\ E)) =$ *cps.prob* $(\bigcap\ (F \text{ ' } \{0..<n\}))$
using *measure-uniform-measure-eq-cond-prob-ev3*[*of* $\bigcap (F \text{ ' } \{0..<n\})\ \bigcap E$] *sevents* M'*-def*
by *blast*
moreover have F '$\{0..<n\} \subseteq$ *cps.events* **using** *cps-ev assms*(5) **by** *force*
ultimately have *cps.prob* $(\bigcap\ (F \text{ ' } \{0..<n\}))$ = *cps.prob* $(F\ 0) * (\prod i = 1..<n$
.
cps.cond-prob-ev $(F\ i)$ $(\bigcap (F \text{ ' } \{0..<i\})))$
using *assms*(1) *cps.prob-cond-Inter-index*[*of n F*] **by** *blast*
moreover have *cps.prob* $(F\ 0) = \mathcal{P}((F\ 0) \mid (\bigcap E))$
proof −
have *ev*: $F\ 0 \in$ *events* **using** *assms*(1) *assms*(5) **by** *auto*
then show *?thesis*
using *ev sevents measure-uniform-measure-eq-cond-prob-ev3*[*of* $F\ 0\ \bigcap E$] M'*-def* **by** *presburger*
qed
moreover have $\bigwedge i.\ i > 0 \Longrightarrow i < n \Longrightarrow$
cps.cond-prob-ev $(F\ i)$ $(\bigcap (F \text{ ' } \{0..<i\})) = \mathcal{P}((F\ i) \mid (\bigcap ((F \text{ ' } \{0..<i\}) \cup E)))$
proof −
fix i **assume** *igt*: $i > 0$ **and** *ilt*: $i < n$
then have $(\bigcap (F \text{ ' } \{0..<i\})) \in$ *events*
using *assms subset-trans igt Inter-event-ss fin* **by** *auto*
moreover have $F\ i \in$ *events* **using** *assms*
using *subset-iff igt ilt* **by** *simp*
moreover have $(\bigcap ((F \text{ ' } \{0..<i\}) \cup (E))) = (\bigcap ((F \text{ ' } \{0..<i\}))) \cap (\bigcap (E))$
by (*simp add*: *Inf-union-distrib*)
ultimately show *cps.cond-prob-ev* $(F\ i)$ $(\bigcap (F \text{ ' } \{0..<i\})) = \mathcal{P}((F\ i) \mid (\bigcap ((F \text{ ' } \{0..<i\}) \cup E)))$
using *sevents cond-prob-ev-double*[*of* $F\ i$ $(\bigcap ((F \text{ ' } \{0..<i\})))\ \bigcap E$] *assms* M'*-def* **by** *presburger*
qed

ultimately have *eq*: *cps.prob* $(\bigcap\ (F\ `\ \{0..<n\})) = \mathcal{P}((F\ 0) \mid (\bigcap E)) * (\prod i \in \{1..<n\}$.
$\mathcal{P}((F\ i) \mid (\bigcap((F\ `\ \{0..<i\}) \cup E))))$ **by** *simp*
moreover have $\{1..<n\} = \{0..<n\} - \{0\}$
by (*simp add*: *atLeast1-lessThan-eq-remove0 lessThan-atLeast0*)
ultimately have $\mathcal{P}((F\ 0) \mid (\bigcap E)) * (\prod i \in \{1..<n\}\ .\ \ \mathcal{P}((F\ i) \mid (\bigcap((F\ `\ \{0..<i\}) \cup E)))) =$
$(\prod i \in \{0..<n\}\ .\ \mathcal{P}((F\ i) \mid (\bigcap((F\ `\ \{0..<i\}) \cup E))))$ **using** *assms(1)*
prod.remove[*of* $\{0..<n\}\ 0\ \lambda\ i.\ \mathcal{P}((F\ i) \mid (\bigcap((F\ `\{0..<i\}) \cup E)))$] **by** *fastforce*
then show *?thesis* **using** *peq eq* **by** *auto*
qed

lemma *prob-cond-Inter-index-cond-compl-set*:
fixes n :: *nat*
assumes $n > 0$
assumes *finite E*
assumes $E \neq \{\}$
assumes $E \subseteq$ *events*
assumes $F\ `\ \{0..<n\} \subseteq$ *events*
assumes *prob* $(\bigcap E) > 0$
shows $\mathcal{P}((\bigcap((-)\ (space\ M)\ `\ F\ `\ \{0..<n\})) \mid (\bigcap E)) =$
$(\prod i = 0..<n\ .\ \ \mathcal{P}((space\ M - F\ i) \mid (\bigcap((-)\ (space\ M)\ `\ F\ `\ \{0..<i\} \cup E))))$
proof −
define G **where** $G \equiv \lambda\ i.\ (space\ M - F\ i)$
then have $G\ `\ \{0..<n\} \subseteq$ *events* **using** *assms(5)* **by** *auto*
then have $\mathcal{P}((\bigcap(G\ `\ \{0..<n\})) \mid (\bigcap E)) = (\prod i \in \{0..<n\}.\ \ \mathcal{P}(G\ i \mid (\bigcap((G\ `\ \{0..<i\}) \cup E))))$
using *prob-cond-Inter-index-cond-set*[*of n E G*] *assms* **by** *blast*
moreover have $((-)\ (space\ M)\ `\ F\ `\ \{0..<n\}) = (G\ `\ \{0..<n\})$ **unfolding** *G-def* **by** *auto*
moreover have $\bigwedge\ i.\ i \in \{0..<n\} \Longrightarrow \mathcal{P}((space\ M - F\ i) \mid (\bigcap((-)\ (space\ M)\ `\ F\ `\ \{0..<i\} \cup E))) =$
$\mathcal{P}(G\ i \mid (\bigcap((G\ `\ \{0..<i\}) \cup E)))$
proof −
fix i **assume** *iin*: $i \in \{0..<n\}$
have $(-)\ (space\ M)\ `\ F\ `\ \{0..<i\} =\ \ G\ `\ \{0..<i\}$ **unfolding** *G-def* **using** *iin* **by** *auto*
then show $\mathcal{P}((space\ M - F\ i) \mid (\bigcap((-)\ (space\ M)\ `\ F\ `\ \{0..<i\} \cup E))) =$
$\mathcal{P}(G\ i \mid (\bigcap((G\ `\ \{0..<i\}) \cup E)))$ **unfolding** *G-def* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *prob-cond-Inter-index-cond*:
fixes n :: *nat*
assumes $n > 0$
assumes $n < m$
assumes $F\ `\ \{0..<m\} \subseteq$ *events*
assumes *prob* $(\bigcap j \in \{n..<m\}.\ F\ j) > 0$

shows $\mathcal{P}((\bigcap (F \mathbin{`} \{0..<n\})) \mid (\bigcap j \in \{n..<m\} \,.\, F\, j)) = (\prod i \in \{0..<n\}.\ \mathcal{P}(F\, i \mid (\bigcap ((F \mathbin{`} \{0..<i\}) \cup (F \mathbin{`} \{n..<m\})))))$
proof −
let *?E* = $F \mathbin{`} \{n..<m\}$
have $F \mathbin{`} \{0..<n\} \subseteq$ *events* **using** *assms(2) assms(3)* **by** *auto*
moreover have *?E* $\subseteq$ *events* **using** *assms(2) assms(3)* **by** *auto*
moreover have *prob*$(\bigcap$ *?E*$) > 0$ **using** *assms(4)* **by** *simp*
moreover have *?E* $\neq \{\}$ **using** *assms(2)* **by** *simp*
ultimately show *?thesis* **using** *prob-cond-Inter-index-cond-set*[*of n ?E F*] *assms(1)* **by** *blast*
qed

lemma *prob-cond-Inter-index-cond-compl*:
fixes n :: *nat*
assumes $n > 0$
assumes $n < m$
assumes $F \mathbin{`} \{0..<m\} \subseteq$ *events*
assumes *prob* $(\bigcap j \in \{n..<m\}.\ F\, j) > 0$
shows $\mathcal{P}((\bigcap ((-)\ (\textit{space}\ M) \mathbin{`} F \mathbin{`} \{0..<n\})) \mid (\bigcap (\ F \mathbin{`} \{n..<m\}))) =$
$(\prod i = 0..<n\ .\ \mathcal{P}((\textit{space}\ M - F\, i) \mid (\bigcap ((-)\ (\textit{space}\ M) \mathbin{`} F \mathbin{`} \{0..<i\} \cup (F \mathbin{`} \{n..<m\}))))))$
proof −
define G **where** $G \equiv \lambda\, i.\ \textit{if}\ (i < n)\ \textit{then}\ (\textit{space}\ M - F\, i)\ \textit{else}\ F\, i$
then have $G \mathbin{`} \{0..<m\} \subseteq$ *events* **using** *assms(3)* **by** *auto*
moreover have *prob* $(\bigcap j \in \{n..<m\}.\ G\, j) > 0$ **using** *G-def assms(4)* **by** *simp*
ultimately have $\mathcal{P}((\bigcap (G \mathbin{`} \{0..<n\})) \mid (\bigcap (\ G \mathbin{`} \{n..<m\}))) = (\prod i \in \{0..<n\}.\ \mathcal{P}(G\, i \mid (\bigcap ((G \mathbin{`} \{0..<i\}) \cup (G \mathbin{`} \{n..<m\})))))$
using *prob-cond-Inter-index-cond*[*of n m G*] *assms(1) assms(2)* **by** *blast*
moreover have $((-)\ (\textit{space}\ M) \mathbin{`} F \mathbin{`} \{0..<n\}) = (G \mathbin{`} \{0..<n\})$ **unfolding** *G-def* **by** *auto*
moreover have *meq*: $(\ F \mathbin{`} \{n..<m\}) = (\ G \mathbin{`} \{n..<m\})$ **unfolding** *G-def* **by** *auto*
moreover have $\bigwedge\, i.\ i \in \{0..<n\} \Longrightarrow \mathcal{P}((\textit{space}\ M - F\, i) \mid (\bigcap ((-)\ (\textit{space}\ M) \mathbin{`} F \mathbin{`} \{0..<i\} \cup (F \mathbin{`} \{n..<m\})))) =$
$\mathcal{P}(G\, i \mid (\bigcap ((G \mathbin{`} \{0..<i\}) \cup (G \mathbin{`} \{n..<m\}))))$
proof −
fix i **assume** *iin*: $i \in \{0..<n\}$
then have $(\textit{space}\ M - F\, i) = G\, i$ **unfolding** *G-def* **by** *auto*
moreover have $(-)\ (\textit{space}\ M) \mathbin{`} F \mathbin{`} \{0..<i\} = \ G \mathbin{`} \{0..<i\}$ **unfolding** *G-def* **using** *iin* **by** *auto*
ultimately show $\mathcal{P}((\textit{space}\ M - F\, i) \mid (\bigcap ((-)\ (\textit{space}\ M) \mathbin{`} F \mathbin{`} \{0..<i\} \cup (F \mathbin{`} \{n..<m\})))) =$
$\mathcal{P}(G\, i \mid (\bigcap ((G \mathbin{`} \{0..<i\}) \cup (G \mathbin{`} \{n..<m\}))))$ **using** *meq* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *prob-cond-Inter-take-cond-neg*:

assumes $xs \neq []$
assumes $set\ xs \subseteq events$
assumes $S \subseteq events$
assumes $S \neq \{\}$
assumes *finite* S
assumes $prob\ (\bigcap S) > 0$
shows $\mathcal{P}((\bigcap((-)\ (space\ M)\ `(set\ xs))) \mid (\bigcap\ S)) =$
$(\prod i = 0..<(length\ xs)\ .\ \mathcal{P}((space\ M - xs\ !\ i) \mid (\bigcap((-)\ (space\ M)\ `(set\ (take\ i\ xs\ )) \cup S))))$
proof −
define ys **where** $ys = map\ ((-)\ (space\ M))\ xs$
have *set*: $((-)\ (space\ M)\ `(set\ xs)) = set\ (ys)$
using *ys-def* **by** *simp*
then have $set\ ys \subseteq events$
by (*metis assms*(2) *image-subset-iff sets.compl-sets subsetD*)
moreover have $ys \neq []$ **using** *ys-def assms*(1) **by** *simp*
ultimately have $\mathcal{P}(\bigcap(set\ ys) \mid (\bigcap S)) =$
$(\prod i = 0..<(length\ ys)\ .\ \mathcal{P}((ys\ !\ i) \mid (\bigcap(set\ (take\ i\ ys\ ) \cup S))))$
using *prob-cond-Inter-take-cond assms* **by** *auto*
moreover have *len*: $length\ ys = length\ xs$ **using** *ys-def* **by** *auto*
moreover have $\bigwedge i.\ i < length\ xs \Longrightarrow ys\ !\ i = space\ M - xs\ !\ i$ **using** *ys-def nth-map len* **by** *auto*
moreover have $\bigwedge i.\ i < length\ xs \Longrightarrow set\ (take\ i\ ys) = (-)\ (space\ M)\ `set\ (take\ i\ xs)$
using *ys-def take-map len* **by** (*metis set-map*)
ultimately show *?thesis* **using** *set* **by** *auto*
qed

lemma *prob-cond-Inter-List-Index*:
assumes $xs \neq []$
assumes $set\ xs \subseteq events$
shows $prob\ (\bigcap(set\ xs)) = prob\ (hd\ xs) * (\prod i = 1..<(length\ xs)\ .$
$\mathcal{P}((xs\ !\ i) \mid (\bigcap\ j \in \{0..<i\}\ .\ xs\ !\ j)))$
proof −
have $\bigwedge\ i.\ i < length\ xs \Longrightarrow set\ (take\ i\ xs) = ((!)\ xs\ `\{0..<i\})$
by (*metis nat-less-le nth-image*)
thus *?thesis* **using** *prob-cond-Inter-List*[*of xs*] *assms* **by** *auto*
qed

lemma *obtains-prob-cond-Inter-index*:
assumes $S \neq \{\}$
assumes $S \subseteq events$
assumes *finite* S
obtains xs **where** $set\ xs = S$ **and** $length\ xs = card\ S$ **and**
$prob\ (\bigcap S) = prob\ (hd\ xs) * (\prod i = 1..<(length\ xs)\ .\ \mathcal{P}((xs\ !\ i) \mid (\bigcap\ j \in \{0..<i\}\ .\ xs\ !\ j)))$
using *assms prob-cond-Inter-List-Index exists-list-card*
by (*metis* (*no-types*, *lifting*) *set-empty2*)

lemma *obtain-list-index*:
assumes *bij-betw g {0..<card S} S*
assumes *finite S*
obtains *xs* **where** *set xs = S* **and** $\bigwedge i . i \in \{0..<card\ S\} \Longrightarrow g\ i = xs\ !\ i$ **and** *distinct xs*
proof −
let *?xs = map g [0..<card S]*
have *seq*: *g ‘ {0..<card S} = S* **using** *assms(1)*
by (*simp add*: *bij-betw-imp-surj-on*)
then have *set-eq*: *set ?xs = S*
by *simp*
moreover have $\bigwedge i . i \in \{0..<card\ S\} \Longrightarrow g\ i = ?xs\ !\ i$
by *auto*
moreover have *leneq*: *length ?xs = card S* **using** *seq* **by** *auto*
moreover have *distinct ?xs* **using** *set-eq leneq*
by (*simp add*: *card-distinct*)
ultimately show *?thesis*
using *that* **by** *blast*
qed

lemma *prob-cond-inter-fn*:
assumes *bij-betw g {0..<card S} S*
assumes *finite S*
assumes $S \neq \{\}$
assumes $S \subseteq events$
shows $prob\ (\bigcap S) = prob\ (g\ 0) * (\prod i \in \{1..<(card\ S)\} . \mathcal{P}(g\ i \mid (\bigcap (g\ ‘ \{0..<i\}))))$
proof −
obtain *xs* **where** *seq*: *set xs = S* **and** *geq*: $\bigwedge i . i \in \{0..<card\ S\} \Longrightarrow g\ i = xs\ !\ i$ **and** *distinct xs*
using *obtain-list-index assms* **by** *auto*
then have *len*: *length xs = card S* **by** (*metis distinct-card*)
then have $prob\ (\bigcap S) = prob\ (hd\ xs) * (\prod i \in \{1..<(length\ xs)\} . \mathcal{P}((xs\ !\ i) \mid (\bigcap\ j \in \{0..<i\} . xs\ !\ j)))$
using *prob-cond-Inter-List-Index*[*of xs*] *assms(3) assms(4) seq* **by** *auto*
then have $prob\ (\bigcap S) = prob\ (hd\ xs) * (\prod i \in \{1..<card\ S\} . \mathcal{P}(g\ i \mid (\bigcap\ j \in \{0..<i\} . g\ j)))$
using *geq len* **by** *auto*
moreover have *hd xs = g 0*
proof −
have *length xs > 0* **using** *seq assms(3)* **by** *auto*
then have *hd xs = xs ! 0*
by (*simp add*: *hd-conv-nth*)
then show *?thesis* **using** *geq len*
using ‹*0 < length xs*› **by** *auto*
qed
ultimately show *?thesis* **by** *simp*
qed

lemma *prob-cond-inter-obtain-fn*:
assumes $S \neq \{\}$
assumes $S \subseteq events$
assumes *finite S*
obtains *f* **where** *bij-betw f* $\{0..<card\ S\}\ S$ **and**
$prob\ (\bigcap S) = prob\ (f\ 0) * (\prod i \in \{1..<(card\ S)\}\ .\ \mathcal{P}(f\ i \mid (\bigcap (f\ `\ \{0..<i\}))))$
proof −
obtain *f* **where** *bij-betw f* $\{0..<card\ S\}\ S$
using *assms*(3) *ex-bij-betw-nat-finite* **by** *blast*
then show *?thesis* **using** *that prob-cond-inter-fn assms* **by** *auto*
qed

lemma *prob-cond-inter-obtain-fn-compl*:
assumes $S \neq \{\}$
assumes $S \subseteq events$
assumes *finite S*
obtains *f* **where** *bij-betw f* $\{0..<card\ S\}\ S$ **and** $prob\ (\bigcap((-)\ (space\ M)\ `\ S))$
=
$prob\ (space\ M - f\ 0) * (\prod i \in \{1..<(card\ S)\}\ .\ \mathcal{P}(space\ M - f\ i \mid (\bigcap((-)$
$(space\ M)\ `\ f\ `\ \{0..<i\}))))$
proof −
let *?c* = $(-)\ (space\ M)$
obtain *f* **where** *bb*: *bij-betw f* $\{0..<card\ S\}\ S$
using *assms*(3) *ex-bij-betw-nat-finite* **by** *blast*
moreover have *bij*: *bij-betw ?c S* $((-)\ (space\ M)\ `\ S)$
using *bij-betw-compl-sets-rev assms*(2) **by** *auto*
ultimately have *bij-betw* $(?c \circ f)\ \{0..<card\ S\}\ (?c\ `\ S)$
using *bij-betw-comp-iff* **by** *blast*
moreover have $?c\ `\ S \neq \{\}$ **using** *assms*(1) **by** *auto*
moreover have *finite* $(?c\ `\ S\ )$ **using** *assms*(3) **by** *auto*
moreover have $?c\ `\ S \subseteq events$ **using** *assms*(2) **by** *auto*
moreover have $card\ S = card\ (?c\ `\ S)$ **using** *bij*
by (*simp add*: *bij-betw-same-card*)
ultimately have $prob\ (\bigcap(?c\ `\ S)) = prob\ ((?c \circ f)\ 0) *$
$(\prod i \in \{1..<(card\ S)\}\ .\ \mathcal{P}((?c \circ f)\ i \mid (\bigcap((?c \circ f)\ `\ \{0..<i\}))))$
using *prob-cond-inter-fn*[*of* $(?c \circ f)\ (?c\ `\ S)$] **by** *auto*
then have $prob\ (\bigcap(?c\ `\ S)) = prob\ (space\ M - (f\ 0)) *$
$(\prod i \in \{1..<(card\ S)\}\ .\ \mathcal{P}(space\ M - (f\ i) \mid (\bigcap((?c \circ f)\ `\ \{0..<i\}))))$ **by** *simp*
then show *?thesis* **using** *that bb* **by** *simp*
qed

lemma *prob-cond-Inter-index-cond-fn*:
assumes $I \neq \{\}$
assumes *finite I*
assumes *finite E*
assumes $E \neq \{\}$
assumes $E \subseteq events$
assumes $F\ `\ I \subseteq events$

assumes *prob* $(\bigcap E) > 0$
assumes *bb*: *bij-betw g* $\{0..<card\ I\}$ *I*
shows $\mathcal{P}((\bigcap (F\ `\ g\ `\ \{0..<card\ I\})) \mid (\bigcap E)) =$
$(\prod i \in \{0..<card\ I\}.\ \mathcal{P}(F\ (g\ i) \mid (\bigcap ((F\ `\ g`\ \{0..<i\}) \cup E))))$
proof −
let *?n* = *card I*
have *eq*: $F\ `\ I = (F \circ g)\ `\ \{0..<card\ I\}$ **using** *bij-betw-image-comp-eq bb* **by** *metis*
moreover have $0 <$ *?n* **using** *assms*(*1*) *assms*(*2*) **by** *auto*
ultimately have $\mathcal{P}(\bigcap\ ((F \circ g)\ `\ \{0..<card\ I\}) \mid \bigcap\ E) =$
$(\prod i = 0..<?n.\ \mathcal{P}(F\ (g\ i) \mid \bigcap\ ((F \circ g)\ `\ \{0..<i\} \cup E)))$
using *prob-cond-Inter-index-cond-set*[*of ?n E* $(F \circ g)$] *assms*(*3*) *assms*(*4*) *assms*(*5*) *assms*(*6*)
assms(*7*) **by** *auto*
moreover have $\bigwedge i.\ i \in \{0..<?n\} \Longrightarrow (F \circ g)\ `\ \{0..<i\} = F\ `\ g\ `\ \{0..<i\}$ **using** *image-comp* **by** *auto*
ultimately have $\mathcal{P}(\bigcap\ (F\ `\ g\ `\ \{0..<card\ I\}) \mid \bigcap\ E) = (\prod i = 0..<?n.\ \mathcal{P}(F\ (g\ i) \mid \bigcap\ (F\ `\ g\ `\ \{0..<i\} \cup E)))$
using *image-comp*[*of F g* $\{0..<card\ I\}$] **by** *auto*
then show *?thesis* **using** *eq bb assms* **by** *blast*
qed

lemma *prob-cond-Inter-index-cond-obtains*:
assumes $I \neq \{\}$
assumes *finite I*
assumes *finite E*
assumes $E \neq \{\}$
assumes $E \subseteq$ *events*
assumes $F\ `\ I \subseteq$ *events*
assumes *prob* $(\bigcap E) > 0$
obtains *g* **where** *bij-betw g* $\{0..<card\ I\}$ *I* **and** $\mathcal{P}((\bigcap (F\ `\ g\ `\ \{0..<card\ I\})) \mid (\bigcap E)) =$
$(\prod i \in \{0..<card\ I\}.\ \mathcal{P}(F\ (g\ i) \mid (\bigcap ((F\ `\ g`\ \{0..<i\}) \cup E))))$
proof −
obtain *g* **where** *bb*: *bij-betw g* $\{0..<card\ I\}$ *I* **using** *assms*(*2*) *ex-bij-betw-nat-finite* **by** *auto*
then show *thesis* **using** *assms prob-cond-Inter-index-cond-fn*[*of I E F g*] *that* **by** *blast*
qed

lemma *prob-cond-Inter-index-cond-compl-fn*:
assumes $I \neq \{\}$
assumes *finite I*
assumes *finite E*
assumes $E \neq \{\}$
assumes $E \subseteq$ *events*
assumes $F\ `\ I \subseteq$ *events*
assumes *prob* $(\bigcap E) > 0$
assumes *bb*: *bij-betw g* $\{0..<card\ I\}$ *I*

shows $\mathcal{P}((\bigcap Aj \in I \;.\; space\ M - F\ Aj) \mid (\bigcap E)) =$
$(\prod i \in \{0..<card\ I\}.\ \mathcal{P}(space\ M - F\ (g\ i) \mid (\bigcap (((\lambda Aj.\ space\ M - F\ Aj)\ `\ g\ `\ \{0..<i\}) \cup E))))$
proof −
let *?n* = *card I*
let *?G* = λ *i. space M − F i*
have *eq*: *?G ` I* = (*?G* ∘ *g*) *` {0..<card I}* **using** *bij-betw-image-comp-eq bb* **by** *metis*
then have (*?G* ∘ *g*) *` {0..<card I}* ⊆ *events* **using** *assms(5)*
by (*metis assms(6) compl-subset-in-events image-image*)
moreover have *0 < ?n* **using** *assms(1) assms(2)* **by** *auto*
ultimately have $\mathcal{P}(\bigcap ((?G \circ g)\ `\ \{0..<card\ I\}) \mid \bigcap E) = (\prod i = 0..<?n.\ \mathcal{P}(?G\ (g\ i) \mid \bigcap ((?G \circ g)\ `\ \{0..<i\} \cup E)))$
using *prob-cond-Inter-index-cond-set*[*of ?n E* (*?G* ∘ *g*)] *assms(3) assms(4) assms(5) assms(6) assms(7)* **by** *auto*
moreover have $\bigwedge i.\ i \in \{0..<?n\} \Longrightarrow (?G \circ g)\ `\ \{0..<i\} = ?G\ `\ g\ `\ \{0..<i\}$ **using** *image-comp* **by** *auto*
ultimately have $\mathcal{P}(\bigcap (?G\ `\ I) \mid \bigcap E) = (\prod i = 0..<?n.\ \mathcal{P}(?G\ (g\ i) \mid \bigcap (?G\ `\ g\ `\ \{0..<i\} \cup E)))$
using *image-comp*[*of ?G g {0..<card I}*] *eq* **by** *auto*
then show *?thesis* **using** *bb* **by** *blast*
qed

lemma *prob-cond-Inter-index-cond-compl-obtains*:
assumes *I* ≠ {}
assumes *finite I*
assumes *finite E*
assumes *E* ≠ {}
assumes *E* ⊆ *events*
assumes *F ` I* ⊆ *events*
assumes *prob* $(\bigcap E) > 0$
obtains *g* **where** *bij-betw g {0..<card I} I* **and** $\mathcal{P}((\bigcap Aj \in I\ .\ space\ M - F\ Aj) \mid (\bigcap E)) =$
$(\prod i \in \{0..<card\ I\}.\ \mathcal{P}(space\ M - F\ (g\ i) \mid (\bigcap (((\lambda Aj.\ space\ M - F\ Aj)\ `\ g\ `\ \{0..<i\}) \cup E))))$
proof −
let *?n* = *card I*
let *?G* = λ *i. space M − F i*
obtain *g* **where** *bb*: *bij-betw g {0..<?n} I* **using** *assms(2) ex-bij-betw-nat-finite* **by** *auto*
then show *?thesis* **using** *assms prob-cond-Inter-index-cond-compl-fn*[*of I E F g*] *that* **by** *blast*
qed

lemma *prob-cond-inter-index-fn2*:
assumes *F ` S* ⊆ *events*
assumes *finite S*
assumes *card S* > *0*

assumes *bij-betw g* {*0..<card S*} *S*
shows *prob* (⋂(*F* ‘*S*)) = *prob* (*F* (*g* 0)) ∗ (∏ *i* ∈ {*1..<*(*card S*)} . *P*(*F* (*g i*) | (⋂(*F* ‘ *g* ‘ {*0..<i*})))))
proof −
have *1*: *F* ‘ *S* = (*F* ∘ *g*) ‘ {*0..<card S*} **using** *assms*(*4*) *bij-betw-image-comp-eq* **by** *metis*
moreover have *prob* (⋂((*F* ∘ *g*) ‘ {*0..<card S*})) =
prob (*F* (*g* 0)) ∗ (∏ *i* ∈ {*1..<*(*card S*)} . *P*(*F* (*g i*) | (⋂(*F* ‘ *g* ‘ {*0..<i*})))))
using *1 prob-cond-Inter-index*[*of card S F* ∘ *g*] *assms*(*3*) *assms*(*1*) **by** *auto*
ultimately show *?thesis* **using** *assms*(*4*)
by *metis*
qed

lemma *prob-cond-inter-index-fn*:
assumes *F* ‘ *S* ⊆ *events*
assumes *finite S*
assumes *S* ≠ {}
assumes *bij-betw g* {*0..<card S*} *S*
shows *prob* (⋂(*F* ‘*S*)) = *prob* (*F* (*g* 0)) ∗ (∏ *i* ∈ {*1..<*(*card S*)} . *P*(*F* (*g i*) | (⋂(*F* ‘ *g* ‘ {*0..<i*})))))
proof −
have *card S* > *0* **using** *assms*(*3*) *assms*(*2*)
by (*simp add*: *card-gt-0-iff*)
moreover have (*F* ∘ *g*) ‘ {*0..<card S*} ⊆ *events* **using** *assms*(*1*) *assms*(*4*)
using *bij-betw-imp-surj-on* **by** (*metis image-comp*)
ultimately have *prob* (⋂((*F* ∘ *g*) ‘ {*0..<card S*})) =
prob (*F* (*g* 0)) ∗ (∏ *i* ∈ {*1..<*(*card S*)} . *P*(*F* (*g i*) | (⋂(*F* ‘ *g* ‘ {*0..<i*})))))
using *prob-cond-Inter-index*[*of card S F* ∘ *g*] **by** *auto*
moreover have *F* ‘ *S* = (*F* ∘ *g*) ‘ {*0..<card S*} **using** *assms*(*4*)
using *bij-betw-imp-surj-on image-comp* **by** (*metis*)
ultimately show *?thesis* **using** *assms*(*4*) **by** *presburger*
qed

lemma *prob-cond-inter-index-obtain-fn*:
assumes *F* ‘ *S* ⊆ *events*
assumes *finite S*
assumes *S* ≠ {}
obtains *g* **where** *bij-betw g* {*0..<card S*} *S* **and**
prob (⋂(*F* ‘*S*)) = *prob* (*F* (*g* 0)) ∗ (∏ *i* ∈ {*1..<*(*card S*)} . *P*(*F* (*g i*) | (⋂(*F* ‘ *g* ‘ {*0..<i*})))))
proof −
obtain *f* **where** *bb*: *bij-betw f* {*0..<card S*} *S*
using *assms*(*2*) *ex-bij-betw-nat-finite* **by** *blast*
then show *?thesis* **using** *prob-cond-inter-index-fn that assms* **by** *blast*
qed

lemma *prob-cond-inter-index-fn-compl*:
assumes *S* ≠ {}
assumes *F* ‘ *S* ⊆ *events*

assumes *finite S*
assumes *bij-betw f {0..<card S} S*
shows *prob* $(\bigcap((-)\ (space\ M)\ `\ F\ `\ S)) = prob\ (space\ M - F\ (f\ 0)) *$
$(\prod i \in \{1..<(card\ S)\}\ .\ \mathcal{P}(space\ M - F\ (f\ i) \mid (\bigcap((-)\ (space\ M)\ `\ F\ `\ f\ `\ \{0..<i\}))))$
proof −
define *G* **where** $G \equiv \lambda\ i.\ space\ M - F\ i$
then have $G\ `\ S \subseteq events$ **using** *G-def assms*(2) **by** *auto*
then have $prob\ (\bigcap\ (G\ `\ S)) = prob\ (G\ (f\ 0)) * (\prod i = 1..<card\ S.\ \mathcal{P}(G\ (f\ i) \mid \bigcap\ (G\ `\ f\ `\ \{0..<i\})))$
using *prob-cond-inter-index-fn*[*of G S*] *assms* **by** *auto*
moreover have $(\bigcap((-)\ (space\ M)\ `\ F\ `\ S)) = (\bigcap i \in S.\ space\ M - F\ i)$ **by** *auto*
ultimately show *?thesis* **unfolding** *G-def* **by** *auto*
qed

lemma *prob-cond-inter-index-obtain-fn-compl*:
assumes $S \neq \{\}$
assumes $F\ `\ S \subseteq events$
assumes *finite S*
obtains *f* **where** *bij-betw f {0..<card S} S* **and**
$prob\ (\bigcap((-)\ (space\ M)\ `\ F\ `\ S)) = prob\ (space\ M - F\ (f\ 0)) *$
$(\prod i \in \{1..<(card\ S)\}\ .\ \mathcal{P}(space\ M - F\ (f\ i) \mid (\bigcap((-)\ (space\ M)\ `\ F\ `\ f\ `\ \{0..<i\}))))$
proof −
obtain *f* **where** *bb*: *bij-betw f {0..<card S} S*
using *assms*(3) *ex-bij-betw-nat-finite* **by** *blast*
then show *?thesis* **using** *prob-cond-inter-index-fn-compl*[*of S F f*] *assms that* **by** *blast*
qed

lemma *prob-cond-Inter-take*:
assumes $S \neq \{\}$
assumes $S \subseteq events$
assumes *finite S*
obtains *xs* **where** *set xs* $= S$ **and** *length xs* $=$ *card S* **and**
$prob\ (\bigcap S) = prob\ (hd\ xs) * (\prod i = 1..<(length\ xs)\ .\ \mathcal{P}((xs\ !\ i) \mid (\bigcap(set\ (take\ i\ xs)))))$
using *assms prob-cond-Inter-List exists-list-card*
by (*metis* (*no-types*, *lifting*) *set-empty2 subset-code*(1))

lemma *prob-cond-Inter-set-bound*:
assumes $A \neq \{\}$
assumes $A \subseteq events$
assumes *finite A*
assumes $\bigwedge Ai\ .\ f\ Ai \geq 0 \wedge f\ Ai \leq 1$
assumes $\bigwedge Ai\ S.\ Ai \in A \Longrightarrow S \subseteq A - \{Ai\} \Longrightarrow S \neq \{\} \Longrightarrow \mathcal{P}(Ai \mid (\bigcap S)) \geq f\ Ai$

assumes $\bigwedge Ai.\ Ai \in A \Longrightarrow prob\ Ai \geq f\ Ai$
shows $prob\ (\bigcap A) \geq (\prod a' \in A\ .\ f\ a')$
proof −
obtain *xs* **where** *eq*: $set\ xs = A$ **and** *seq*: $length\ xs = card\ A$ **and**
pA: $prob\ (\bigcap A) = prob\ (hd\ xs) * (\prod i = 1..<(length\ xs)\ .\ \mathcal{P}((xs\ !\ i) \mid (\bigcap\ j \in \{0..<i\}\ .\ xs\ !\ j)))$
using *assms obtains-prob-cond-Inter-index*[*of A*] **by** *blast*
then have *dis*: *distinct xs* **using** *card-distinct*
by *metis*
then have $hd\ xs \in A$ **using** *eq hd-in-set assms*(*1*) **by** *auto*
then have $prob\ (hd\ xs) \geq (f\ (hd\ xs))$ **using** *assms*(*6*) **by** *blast*
have $\bigwedge i.\ i \in \{1..<(length\ xs)\} \Longrightarrow \mathcal{P}((xs\ !\ i) \mid (\bigcap\ j \in \{0..<i\}\ .\ xs\ !\ j)) \geq f\ (xs\ !\ i)$
proof −
fix *i* **assume** $i \in \{1..<length\ xs\}$
then have *ilb*: $i \geq 1$ **and** *iub*: $i < length\ xs$ **by** *auto*
then have *xsin*: $xs\ !\ i \in A$ **using** *eq* **by** *auto*
define *S* **where** $S = (\lambda\ j.\ xs\ !\ j)\ `\ \{0..<i\}$
then have $S = set\ (take\ i\ xs)$
by (*simp add*: *iub less-or-eq-imp-le nth-image*)
then have $xs\ !\ i \notin S$ **using** *dis set-take-distinct-elem-not iub* **by** *simp*
then have $S \subseteq A - \{(xs\ !\ i)\}$
using ‹$S = set\ (take\ i\ xs)$› *eq set-take-subset* **by** *fastforce*
moreover have $S \neq \{\}$ **using** *S-def ilb* **by** (*simp*)
moreover have $\mathcal{P}((xs\ !\ i) \mid (\bigcap\ j \in \{0..<i\}\ .\ xs\ !\ j)) = \mathcal{P}((xs\ !\ i) \mid (\bigcap\ Aj \in S\ .\ Aj))$
using *S-def* **by** *auto*
ultimately show $\mathcal{P}((xs\ !\ i) \mid (\bigcap\ j \in \{0..<i\}\ .\ xs\ !\ j)) \geq f\ (xs\ !\ i)$
using *assms*(*5*) *xsin* **by** *auto*
qed
then have $(\prod i = 1..<(length\ xs)\ .\ \mathcal{P}((xs\ !\ i) \mid (\bigcap\ j \in \{0..<i\}\ .\ xs\ !\ j))) \geq (\prod i = 1..<(length\ xs)\ .\ f\ (xs\ !\ i))$
by (*meson assms*(*4*) *prod-mono*)
moreover have $(\prod i = 1..<(length\ xs)\ .\ f\ (xs\ !\ i)) = (\prod a \in A - \{hd\ xs\}\ .\ f\ a)$
proof −
have *ne*: $xs \neq []$ **using** *assms*(*1*) *eq* **by** *auto*
have $A = (\lambda\ j.\ xs\ !\ j)\ `\ \{0..<length\ xs\}$ **using** *eq*
by (*simp add*: *nth-image*)
have $A - \{hd\ xs\} = set\ (tl\ xs)$ **using** *dis*
by (*metis Diff-insert-absorb distinct.simps*(*2*) *eq list.exhaust-sel list.set*(*2*) *ne*)
also have ... $= (\lambda\ j.\ xs\ !\ j)\ `\ \{1..<length\ xs\}$ **using** *nth-image-tl ne* **by** *auto*
finally have *Ahdeq*: $A - \{hd\ xs\} = (\lambda\ j.\ xs\ !\ j)\ `\ \{1..<length\ xs\}$ **by** *simp*
have *io*: *inj-on* $(nth\ xs)\ \{1..<length\ xs\}$ **using** *inj-on-nth dis*
by (*metis atLeastLessThan-iff*)
have $(\prod i = 1..<(length\ xs)\ .\ f\ (xs\ !\ i)) = (\prod i \in \{1..<(length\ xs)\}\ .\ f\ (xs\ !\ i))$ **by** *simp*
also have ... $= (\prod i \in (\lambda\ j.\ xs\ !\ j)\ `\ \{1..<length\ xs\}\ .\ f\ i)$
using *io* **by** (*simp add*: *prod.reindex-cong*)
finally show *?thesis* **using** *Ahdeq*

using ‹$(\prod i = 1..<length\ xs.\ f\ (xs\ !\ i)) = prod\ f\ ((!)\ xs\ `\ \{1..<length\ xs\})$›
by *presburger*
qed
ultimately have $prob\ (\bigcap A) \geq f\ (hd\ xs) * (\prod a \in A - \{hd\ xs\}\ .\ f\ a)$
using pA ‹$f\ (hd\ xs) \leq prob\ (hd\ xs)$› *assms*(4) *ordered-comm-semiring-class.comm-mult-left-mono*
by (*simp add*: *mult-mono'* *prod-nonneg*)
then show *?thesis*
by (*metis* ‹$hd\ xs \in A$› *assms*(3) *prod.remove*)
qed
end

end

5 Independent Events

theory *Indep-Events* **imports** *Cond-Prob-Extensions*
begin

5.1 More bijection helpers

lemma *bij-betw-obtain-subsetr*:
assumes *bij-betw f A B*
assumes $A' \subseteq A$
obtains B' **where** $B' \subseteq B$ **and** $B' = f\ `\ A'$
using *assms* **by** (*metis bij-betw-def image-mono*)

lemma *bij-betw-obtain-subsetl*:
assumes *bij-betw f A B*
assumes $B' \subseteq B$
obtains A' **where** $A' \subseteq A$ **and** $B' = f\ `\ A'$
using *assms*
by (*metis bij-betw-imp-surj-on subset-imageE*)

lemma *bij-betw-remove*: *bij-betw f A B* $\Longrightarrow$ $a \in A$ $\Longrightarrow$ *bij-betw f* $(A - \{a\})$ $(B - \{f\ a\})$
using *bij-betwE notIn-Un-bij-betw3*
by (*metis Un-insert-right insert-Diff member-remove remove-def sup-bot.right-neutral*)

5.2 Independent Event Extensions

Extensions on both the indep_event definition and the indep_events definition

context *prob-space*
begin

lemma *indep-eventsD*: *indep-events A I* $\Longrightarrow$ $(A`I \subseteq events) \Longrightarrow J \subseteq I \Longrightarrow J \neq \{\} \Longrightarrow$ *finite J* $\Longrightarrow$
$prob\ (\bigcap j \in J.\ A\ j) = (\prod j \in J.\ prob\ (A\ j))$

using *indep-events-def*[*of A I*] **by** *auto*

lemma
assumes *indep*: *indep-event A B*
shows *indep-eventD-ev1*: *A* ∈ *events*
and *indep-eventD-ev2*: *B* ∈ *events*
using *indep* **unfolding** *indep-event-def indep-events-def UNIV-bool* **by** *auto*

lemma *indep-eventD*:
assumes *ie*: *indep-event A B*
shows *prob* (*A* ∩ *B*) = *prob* (*A*) ∗ *prob* (*B*)
using *assms indep-eventD-ev1 indep-eventD-ev2 ie*[*unfolded indep-event-def*, *THEN indep-eventsD*,*of UNIV*]
by (*simp add*: *ac-simps UNIV-bool*)

lemma *indep-eventI*[*intro*]:
assumes *ev*: *A* ∈ *events B* ∈ *events*
and *indep*: *prob* (*A* ∩ *B*) = *prob A* ∗ *prob B*
shows *indep-event A B*
unfolding *indep-event-def*
proof (*intro indep-eventsI*)
show ⋀*i*. *i* ∈ *UNIV* ⟹ (*case i of True* ⇒ *A* | *False* ⇒ *B*) ∈ *events*
using *assms* **by** (*auto split*: *bool.split*)
next
fix *J* :: *bool set* **assume** *jss*: *J* ⊆ *UNIV* **and** *jne*: *J* ≠ {} **and** *finJ*: *finite J*
have *J* ∈ *Pow UNIV* **by** *auto*
then have *c*: *J* = *UNIV* ∨ *J* = {*True*} ∨ *J* = {*False*} **using** *jne jss UNIV-bool*
by (*metis* (*full-types*) *UNIV-eq-I insert-commute subset-insert subset-singletonD*)

then show *prob* (⋂*i*∈*J*. *case i of True* ⇒ *A* | *False* ⇒ *B*) =
(∏ *i*∈*J*. *prob* (*case i of True* ⇒ *A* | *False* ⇒ *B*))
unfolding *UNIV-bool* **using** *indep* **by** (*auto simp*: *ac-simps*)
qed

Alternate set definition - when no possibility of duplicate objects

definition *indep-events-set* :: *'a set set* ⇒ *bool* **where**
indep-events-set E ≡ (*E* ⊆ *events* ∧ (∀ *J*. *J* ⊆ *E* ⟶ *finite J* ⟶ *J* ≠ {} ⟶ *prob* (⋂ *J*) = (∏ *i*∈*J*. *prob i*)))

lemma *indep-events-setI*[*intro*]: *E* ⊆ *events* ⟹ (⋀*J*. *J* ⊆ *E* ⟹ *finite J* ⟹ *J* ≠ {} ⟹
prob (⋂ *J*) = (∏ *i*∈*J*. *prob i*)) ⟹ *indep-events-set E*
using *indep-events-set-def* **by** *simp*

lemma *indep-events-subset*:
indep-events-set E ⟷ (∀ *J*⊆*E*. *indep-events-set J*)
by (*auto simp*: *indep-events-set-def*)

lemma *indep-events-subset2*:

indep-events-set E $\Longrightarrow$ *J* $\subseteq$ *E* $\Longrightarrow$ *indep-events-set J*
by (*auto simp*: *indep-events-set-def*)

lemma *indep-events-set-events*: *indep-events-set E* $\Longrightarrow$ ($\bigwedge$*e*. *e* $\in$ *E* $\Longrightarrow$ *e* $\in$ *events*)

using *indep-events-set-def* **by** *auto*

lemma *indep-events-set-events-ss*: *indep-events-set E* $\Longrightarrow$ *E* $\subseteq$ *events*
using *indep-events-set-events* **by** *auto*

lemma *indep-events-set-probs*: *indep-events-set E* $\Longrightarrow$ *J* $\subseteq$ *E* $\Longrightarrow$ *finite J* $\Longrightarrow$ *J* $\neq$ {} $\Longrightarrow$
prob ($\bigcap$ *J*) = ($\prod$ *i*$\in$*J*. *prob i*)
by (*simp add*: *indep-events-set-def*)

lemma *indep-events-set-prod-all*: *indep-events-set E* $\Longrightarrow$ *finite E* $\Longrightarrow$ *E* $\neq$ {} $\Longrightarrow$
prob ($\bigcap$ *E*) = *prod prob E*
using *indep-events-set-probs* **by** *simp*

lemma *indep-events-not-contain-compl*:
assumes *indep-events-set E*
assumes *A* $\in$ *E*
assumes *prob A* > *0 prob A* < *1*
shows (*space M* − *A*) $\notin$ *E* (**is** *?A′* $\notin$ *E*)
proof (*rule ccontr*)
assume ¬ (*?A′*) $\notin$ *E*
then have *?A′* $\in$ *E* **by** *auto*
then have {*A*, *?A′*} $\subseteq$ *E* **using** *assms*(*2*) **by** *auto*
moreover have *finite* {*A*, *?A′*} **by** *simp*
moreover have {*A*, *?A′*} $\neq$ {}
by *simp*
ultimately have *prob* ($\bigcap$ *i*$\in${*A*, *?A′*}. *i*) = ($\prod$ *i*$\in${*A*, *?A′*}. *prob i*)
using *indep-events-set-probs*[*of E* {*A*, *?A′*}] *assms*(*1*) **by** *auto*
then have *prob* (*A* ∩ *?A′*) = *prob A* ∗ *prob ?A′* **by** *simp*
moreover have *prob* (*A* ∩ *?A′*) = *0* **by** *simp*
moreover have *prob A* ∗ *prob ?A′* = *prob A* ∗ (*1* − *prob A*)
using *assms*(*1*) *assms*(*2*) *indep-events-set-events prob-compl* **by** *auto*
moreover have *prob A* ∗ (*1* − *prob A*) > *0* **using** *assms*(*3*) *assms*(*4*) **by** (*simp add*: *algebra-simps*)
ultimately show *False* **by** *auto*
qed

lemma *indep-events-contain-compl-prob01*:
assumes *indep-events-set E*
assumes *A* $\in$ *E*
assumes *space M* − *A* $\in$ *E*
shows *prob A* = *0* ∨ *prob A* = *1*
proof (*rule ccontr*)
let *?A′* = *space M* − *A*

assume *a*: ¬ (*prob A = 0* ∨ *prob A = 1*)
then have *prob A > 0*
by (*simp add*: *zero-less-measure-iff*)
moreover have *prob A < 1*
using *a measure-ge-1-iff* **by** *fastforce*
ultimately have *?A′ ∉ E* **using** *assms(1) assms(2) indep-events-not-contain-compl*
by *auto*
then show *False* **using** *assms(3)* **by** *auto*
qed

lemma *indep-events-set-singleton*:
assumes *A ∈ events*
shows *indep-events-set* {*A*}
proof (*intro indep-events-setI*)
show {*A*} ⊆ *events* **using** *assms* **by** *simp*
next
fix *J* **assume** *J* ⊆ {*A*} *finite J J* ≠ {}
then have *J* = {*A*} **by** *auto*
then show *prob* (⋂ *J*) = *prod prob J* **by** *simp*
qed

lemma *indep-events-pairs*:
assumes *indep-events-set S*
assumes *A ∈ S B ∈ S A* ≠ *B*
shows *indep-event A B*
using *assms indep-events-set-probs*[*of S* {*A*, *B*}]
by (*intro indep-eventI*) (*simp-all add*: *indep-events-set-events*)

lemma *indep-events-inter-pairs*:
assumes *indep-events-set S*
assumes *finite A finite B*
assumes *A* ≠ {} *B* ≠ {}
assumes *A* ⊆ *S B* ⊆ *S A* ∩ *B* = {}
shows *indep-event* (⋂ *A*) (⋂ *B*)
proof (*intro indep-eventI*)
have *A* ⊆ *events B* ⊆ *events* **using** *indep-events-set-events assms* **by** *auto*
then show ⋂ *A ∈ events* ⋂ *B ∈ events* **using** *Inter-event-ss assms* **by** *auto*
next
have *A* ∪ *B* ⊆ *S* **using** *assms* **by** *auto*
then have *prob* (⋂(*A* ∪ *B*)) = *prod prob* (*A* ∪ *B*) **using** *assms*
by (*metis Un-empty indep-events-subset infinite-Un prob-space.indep-events-set-prod-all prob-space-axioms*)
also have *... = prod prob A* ∗ *prod prob B* **using** *assms(8)*
by (*simp add*: *assms(2) assms(3) prod.union-disjoint*)
finally have *prob* (⋂(*A* ∪ *B*)) = *prob* (⋂ *A*) ∗ *prob* (⋂ *B*)
using *assms indep-events-subset indep-events-set-prod-all* **by** *metis*
moreover have ⋂ (*A* ∪ *B*) = (⋂ *A* ∩ ⋂ *B*) **by** *auto*
ultimately show *prob* (⋂ *A* ∩ ⋂ *B*) = *prob* (⋂ *A*) ∗ *prob* (⋂ *B*)

by *simp*
qed

lemma *indep-events-inter-single*:
assumes *indep-events-set S*
assumes *finite B*
assumes $B \neq \{\}$
assumes $A \in S$ $B \subseteq S$ $A \notin B$
shows *indep-event A* $(\bigcap B)$
proof −
have $\{A\} \neq \{\}$ *finite* $\{A\}$ $\{A\} \subseteq S$ **using** *assms* **by** *simp-all*
moreover have $\{A\} \cap B = \{\}$ **using** *assms*(6) **by** *auto*
ultimately show *?thesis* **using** *indep-events-inter-pairs*[*of S* $\{A\}$ *B*] *assms* **by** *auto*
qed

lemma *indep-events-set-prob1*:
assumes $A \in$ *events*
assumes *prob A* = *1*
assumes $A \notin S$
assumes *indep-events-set S*
shows *indep-events-set* $(S \cup \{A\})$
proof (*intro indep-events-setI*)
show $S \cup \{A\} \subseteq$ *events* **using** *assms*(1) *assms*(4) *indep-events-set-events* **by** *auto*
next
fix *J* **assume** *jss*: $J \subseteq S \cup \{A\}$ **and** *finJ*: *finite J* **and** *jne*: $J \neq \{\}$
show *prob* $(\bigcap J)$ = *prod prob J*
proof (*cases* $A \in J$)
case *t1*: *True*
then show *?thesis*
proof (*cases* $J = \{A\}$)
case *True*
then show *?thesis* **using** *indep-events-set-singleton assms*(1) **by** *auto*
next
case *False*
then have *jun*: $(J - \{A\}) \cup \{A\} = J$ **using** *t1* **by** *auto*
have $J - \{A\} \subseteq S$ **using** *jss* **by** *auto*
then have *iej*: *indep-events-set* $(J - \{A\})$ **using** *indep-events-subset2*[*of S* $J - \{A\}$] *assms*(4)
by *auto*
have *jsse*: $J - \{A\} \subseteq$ *events* **using** *indep-events-set-events jss*
using *assms*(4) **by** *blast*
have *jne2*: $J - \{A\} \neq \{\}$ **using** *False jss jne* **by** *auto*
have *split*: $(J - \{A\}) \cap \{A\} = \{\}$ **by** *auto*
then have *prob* $(\bigcap i \in J.\ i)$ = *prob* $((\bigcap i \in (J - \{A\}).\ i) \cap A)$ **using** *jun*
by (*metis Int-commute Inter-insert Un-ac*(3) *image-ident insert-is-Un*)
also have ... = *prob* $((\bigcap i \in (J - \{A\}).\ i))$
using *prob1-basic-Inter*[*of A* $J - \{A\}$] *jsse assms*(2) *jne2 assms*(1) *finJ*

```
      by (simp add: Int-commute)
    also have ... = prob (⋂(J − {A})) ∗ prob A using assms(2) by simp
    also have ... = (prod prob (J − {A})) ∗ prob A
      using iej indep-events-set-prod-all[of J − {A}] jne2 finJ finite-subset by
auto
    also have ... = prod prob ((J − {A}) ∪ {A}) using split
      by (metis finJ jun mult.commute prod.remove t1)
    finally show ?thesis using jun by auto
   qed
  next
   case False
   then have jss2: J ⊆ S using jss by auto
   then have indep-events-set J using assms(4) indep-events-subset2[of S J] by
auto
   then show ?thesis using indep-events-set-probs finJ jne jss2 by auto
  qed
qed

lemma indep-events-set-prob0:
  assumes A ∈ events
  assumes prob A = 0
  assumes A ∉ S
  assumes indep-events-set S
  shows indep-events-set (S ∪ {A})
proof (intro indep-events-setI)
  show S ∪ {A}⊆ events using assms(1) assms(4) indep-events-set-events by
auto
next
  fix J assume jss: J ⊆ S ∪ {A} and finJ: finite J and jne: J ≠ {}
  show prob (⋂ J) = prod prob J
  proof (cases A ∈ J)
   case t1: True
   then show ?thesis
   proof (cases J = {A})
    case True
    then show ?thesis using indep-events-set-singleton assms(1) by auto
   next
    case False
    then have jun: (J − {A}) ∪ {A} = J using t1 by auto
    have J − {A} ⊆ S using jss by auto
     then have iej: indep-events-set (J − {A}) using indep-events-subset2[of S
J − {A}] assms(4) by auto
    have jsse: J − {A} ⊆ events using indep-events-set-events jss
     using assms(4) by blast
    have jne2: J − {A} ≠ {} using False jss jne by auto
    have split: (J − {A}) ∩ {A} = {} by auto
    then have prob (⋂ i∈J. i) = prob ((⋂ i∈(J − {A}). i) ∩ A) using jun
     by (metis Int-commute Inter-insert Un-ac(3) image-ident insert-is-Un)
    also have ... = 0
```

```
      using prob0-basic-Inter[of A J − {A}] jsse assms(2) jne2 assms(1) finJ
      by (simp add: Int-commute)
    also have ... = prob (⋂(J − {A})) ∗ prob A using assms(2) by simp
   also have ... = (prod prob (J − {A})) ∗ prob A using iej indep-events-set-prod-all[of
J − {A}] jne2 finJ finite-subset by auto
    also have ... = prod prob ((J − {A}) ∪ {A}) using split
      by (metis finJ jun mult.commute prod.remove t1)
    finally show ?thesis using jun by auto
   qed
 next
   case False
   then have jss2: J ⊆ S using jss by auto
   then have indep-events-set J using assms(4) indep-events-subset2[of S J] by
auto
   then show ?thesis using indep-events-set-probs finJ jne jss2 by auto
 qed
qed


lemma indep-event-commute:
 assumes indep-event A B
 shows indep-event B A
 using indep-eventI[of B A] indep-eventD[unfolded assms(1), of A B]
by (metis Groups.mult-ac(2) Int-commute assms indep-eventD-ev1 indep-eventD-ev2)
```

Showing complement operation maintains independence

```
lemma indep-event-one-compl:
 assumes indep-event A B
 shows indep-event A (space M − B)
proof −
 let ?B′ = space M − B
 have A = (A ∩ B) ∪ (A ∩ ?B′)
  by (metis Int-Diff Int-Diff-Un assms prob-space.indep-eventD-ev1 prob-space-axioms
sets.Int-space-eq2)
 then have prob A = prob (A ∩ B) + prob (A ∩ ?B′)
  by (metis Diff-Int-distrib Diff-disjoint assms finite-measure-Union indep-eventD-ev1

      indep-eventD-ev2 sets.Int sets.compl-sets)
 then have prob (A ∩ ?B′) = prob A − prob (A ∩ B) by simp
 also have ... = prob A − prob A ∗ prob B using indep-eventD assms(1) by auto
 also have ... = prob A ∗ (1 − prob B)
   by (simp add: vector-space-over-itself.scale-right-diff-distrib)
 finally have prob (A ∩ ?B′) = prob A ∗ prob ?B′
   using prob-compl indep-eventD-ev1 assms(1) indep-eventD-ev2 by presburger
then show indep-event A ?B′ using indep-eventI indep-eventD-ev2 indep-eventD-ev1
assms(1)
   by (meson sets.compl-sets)
qed
```

lemma *indep-event-one-compl-rev*:
assumes $B \in events$
assumes *indep-event* A (*space* $M - B$)
shows *indep-event* A B
proof −
have *space* $M - B \in events$ **using** *indep-eventD-ev2 assms* **by** *auto*
have *space* $M -$ (*space* $M - B$) $= B$ **using** *compl-identity assms* **by** *simp*
then show *?thesis* **using** *indep-event-one-compl*[*of* A *space* $M - B$] *assms*(*2*)
by *auto*
qed

lemma *indep-event-double-compl*: *indep-event* A $B \Longrightarrow$ *indep-event* (*space* $M -$ A) (*space* $M - B$)
using *indep-event-one-compl indep-event-commute* **by** *auto*

lemma *indep-event-double-compl-rev*: $A \in events \Longrightarrow B \in events \Longrightarrow$
indep-event (*space* $M - A$) (*space* $M - B$) $\Longrightarrow$ *indep-event* A B
using *indep-event-double-compl*[*of space* $M - A$ *space* $M - B$] *compl-identity*
by *auto*

lemma *indep-events-set-one-compl*:
assumes *indep-events-set* S
assumes $A \in S$
shows *indep-events-set* ($\{$*space* $M - A\} \cup (S - \{A\})$)
proof (*intro indep-events-setI*)
show $\{$*space* $M - A\} \cup (S - \{A\}) \subseteq events$
using *indep-events-set-events assms*(*1*) *assms*(*2*) **by** *auto*
next
fix J **assume** *jss*: $J \subseteq \{$*space* $M - A\} \cup (S - \{A\})$
assume *finJ*: *finite* J
assume *jne*: $J \neq \{\}$
show *prob* ($\bigcap J$) $=$ *prod prob* J
proof (*cases* $J - \{$*space* $M - A\} = \{\}$)
case *True*
then have $J = \{$*space* $M - A\}$ **using** *jne* **by** *blast*
then show *?thesis* **by** *simp*
next
case *jne2*: *False*
have *jss2*: $J - \{$*space* $M - A\} \subseteq S$ **using** *jss assms*(*2*) **by** *auto*
moreover have $A \notin (J - \{$*space* $M - A\})$ **using** *jss* **by** *auto*
moreover have *finite* ($J - \{$*space* $M - A\}$) **using** *finJ* **by** *simp*
ultimately have *indep-event* A ($\bigcap$ ($J - \{$*space* $M - A\}$))
using *indep-events-inter-single*[*of* S ($J - \{$*space* $M - A\}$) A] *assms jne2* **by**
auto
then have *ie*: *indep-event* (*space* $M - A$) ($\bigcap$ ($J - \{$*space* $M - A\}$))
using *indep-event-one-compl indep-event-commute* **by** *auto*
have *iess*: *indep-events-set* ($J - \{$*space* $M - A\}$)
using *jss2 indep-events-subset2*[*of* S $J - \{$*space* $M - A\}$] *assms*(*1*) **by** *auto*
show *?thesis*

```
    proof (cases space M − A ∈ J)
      case True
      then have split: J = (J − {space M − A}) ∪ {space M − A} by auto
       then have prob (⋂ J) = prob (⋂ ((J − {space M − A}) ∪ {space M −
A})) by simp
      also have ... = prob ((⋂ (J − {space M − A})) ∩ (space M − A))
        by (metis Inter-insert True ‹J = J − {space M − A} ∪ {space M − A}›
inf.commute insert-Diff)
      also have ... = prob (⋂ (J − {space M − A})) * prob (space M − A)
         using ie indep-eventD[of ⋂ (J − {space M − A}) space M − A] in-
dep-event-commute by auto
      also have ... = (prod prob ((J − {space M − A}))) * prob (space M − A)
        using indep-events-set-prod-all[of J − {space M − A}] iess jne2 finJ by
auto
      finally have prob (⋂ J) = prod prob J using split
        by (metis Groups.mult-ac(2) True finJ prod.remove)
      then show ?thesis by simp
    next
      case False
      then show ?thesis using iess
        by (simp add: assms(1) finJ indep-events-set-prod-all jne)
    qed
  qed
qed

lemma indep-events-set-update-compl:
  assumes indep-events-set E
  assumes E = A ∪ B
  assumes A ∩ B = {}
  assumes finite E
  shows indep-events-set (((−) (space M) ‘ A) ∪ B)
using assms(2) assms(3)  proof (induct card A arbitrary: A B)
  case 0
  then show ?case using assms(1)
    using assms(4) by auto
next
  case (Suc x)
  then obtain a A′ where aeq: A = insert a A′ and anotin: a ∉ A′
    by (metis card-Suc-eq-finite)
  then have xcard: card A′ = x
    using Suc(2) Suc(3) assms(4) by auto
  let ?B′ = B ∪ {a}
  have E = A′ ∪ ?B′ using aeq Suc.prems by auto
  moreover have A′ ∩ ?B′ = {} using anotin Suc.prems(2) aeq by auto
  moreover have ?B′ ≠ {} by simp
  ultimately have ies: indep-events-set ((−) (space M) ‘ A′ ∪ ?B′)
    using Suc.hyps(1)[of A′ ?B′] xcard by auto
  then have a ∈ A ∪ B using aeq by auto
  then show ?case
```

proof (*cases* $(A \cup B) - \{a\} = \{\}$)
case *True*
then have $A = \{a\}$ $B = \{\}$ **using** *Suc.prems aeq* **by** *auto*
then have $((-)\ (space\ M)\ `\ A \cup B) = \{space\ M - a\}$ **by** *auto*
moreover have *space M − a ∈ events* **using** *aeq assms(1) Suc.prems indep-events-set-events* **by** *auto*
ultimately show *?thesis* **using** *indep-events-set-singleton* **by** *simp*
next
case *False*
have $a \in (-)\ (space\ M)\ `\ A' \cup ?B'$ **using** *aeq* **by** *auto*
then have *ie*: *indep-events-set* $(\{space\ M - a\} \cup ((-)\ (space\ M)\ `\ A' \cup ?B' - \{a\}))$
using *indep-events-set-one-compl*[*of* $(-)\ (space\ M)\ `\ A' \cup ?B'\ a$] *ies* **by** *auto*
show *?thesis*
proof (*cases a ∈* $(-)\ (space\ M)\ `\ A'$)
case *True*
then have *space M − a ∈ A′*
by (*smt* (*verit*) ‹$E = A' \cup (B \cup \{a\})$› *assms(1) compl-identity image-iff indep-events-set-events*
indep-events-subset2 inf-sup-ord(3))
then have *space M − a ∈ A* **using** *aeq* **by** *auto*
moreover have *indep-events-set A* **using** *Suc.prems(1) indep-events-subset2 assms(1)*
using *aeq* **by** *blast*
moreover have *a ∈ A* **using** *aeq* **by** *auto*
ultimately have *probs*: *prob a = 0 ∨ prob a = 1* **using** *indep-events-contain-compl-prob01*[*of A a*] **by** *auto*
have $((-)\ (space\ M)\ `\ A \cup B) = (-)\ (space\ M)\ `\ A' \cup \{space\ M - a\} \cup B$ **using** *aeq* **by** *auto*
moreover have $((-)\ (space\ M)\ `\ A' \cup ?B' - \{a\}) = ((-)\ (space\ M)\ `\ A' - \{a\}) \cup B$
using *Suc.prems(2) aeq* **by** *auto*
moreover have $(-)\ (space\ M)\ `\ A' = ((-)\ (space\ M)\ `\ A' - \{a\}) \cup \{a\}$ **using** *True* **by** *auto*
ultimately have $((-)\ (space\ M)\ `\ A \cup B) = \{space\ M - a\} \cup ((-)\ (space\ M)\ `\ A' \cup ?B' - \{a\}) \cup \{a\}$
by (*smt* (*verit*) *Un-empty-right Un-insert-right Un-left-commute*)
moreover have $a \notin \{space\ M - a\} \cup ((-)\ (space\ M)\ `\ A' \cup ?B' - \{a\})$
using *Diff-disjoint* ‹*space M − a ∈ A′*› *anotin empty-iff insert-iff* **by** *fastforce*
moreover have *a ∈ events* **using** *Suc.prems(1) assms(1) indep-events-set-events aeq* **by** *auto*
ultimately show *?thesis*
using *ie indep-events-set-prob0 indep-events-set-prob1 probs* **by** *presburger*
next
case *False*
then have $(((-)\ (space\ M)\ `A' \cup ?B') - \{a\}) = (-)\ (space\ M)\ `A' \cup B$
using *Suc.prems(2) aeq* **by** *auto*
moreover have $(-)\ (space\ M)\ `\ A = (-)\ (space\ M)\ `\ A' \cup \{space\ M - a\}$

using *aeq*
by *simp*
ultimately have ((−) (*space M*) ‘ *A* ∪ *B*) = {*space M* − *a*} ∪ ((−) (*space M*) ‘ *A′* ∪ *?B′* − {*a*})
by *auto*
then show *?thesis* **using** *ie* **by** *simp*
qed
qed
qed

lemma *indep-events-set-compl*:
assumes *indep-events-set E*
assumes *finite E*
shows *indep-events-set* ((λ *e*. *space M* − *e*) ‘ *E*)
using *indep-events-set-update-compl*[*of E E* {}] *assms* **by** *auto*

lemma *indep-event-empty*:
assumes *A* ∈ *events*
shows *indep-event A* {}
using *assms indep-eventI* **by** *auto*

lemma *indep-event-compl-inter*:
assumes *indep-event A C*
assumes *B* ∈ *events*
assumes *indep-event A* (*B* ∩ *C*)
shows *indep-event A* ((*space M* − *B*) ∩ *C*)
proof (*intro indep-eventI*)
show *A* ∈ *events* **using** *assms*(*1*) *indep-eventD-ev1* **by** *auto*
show (*space M* − *B*) ∩ *C* ∈ *events* **using** *assms*(*3*) *indep-eventD-ev2*
by (*metis Diff-Int-distrib2 assms*(*1*) *sets.Diff sets.Int-space-eq1*)
next
have *ac*: *A* ∩ *C* ∈ *events* **using** *assms*(*1*) *indep-eventD-ev1 indep-eventD-ev2 sets.Int-space-eq1*
by *auto*
have *prob* (*A* ∩ ((*space M* − *B*) ∩ *C*)) = *prob* (*A* ∩ (*space M* − *B*) ∩ *C*)
by (*simp add*: *inf-sup-aci*(*2*))
also have ... = *prob* (*A* ∩ *C* ∩ (*space M* − *B*))
by (*simp add*: *ac-simps*)
also have ... = *prob* (*A* ∩ *C*) − *prob* (*A* ∩ *C* ∩ *B*)
using *prob-compl-diff-inter*[*of A* ∩ *C B*] *ac assms*(*2*) **by** *auto*
also have ... = *prob* (*A*) ∗ *prob C* − (*prob A* ∗ *prob* (*C* ∩ *B*))
using *assms*(*1*) *assms*(*3*) *indep-eventD*
by (*simp add*: *inf-commute inf-left-commute*)
also have ... = *prob A* ∗ (*prob C* − *prob* (*C* ∩ *B*)) **by** (*simp add*: *algebra-simps*)
finally have *prob* (*A* ∩ ((*space M* − *B*) ∩ *C*)) = *prob A* ∗ (*prob* (*C* ∩ (*space M* − *B*)))
using *prob-compl-diff-inter*[*of C B*] **using** *assms*(*1*) *assms*(*2*)
by (*simp add*: *indep-eventD-ev2*)

then show *prob* $(A \cap ((space\ M - B) \cap C)) = prob\ A * prob\ ((space\ M - B) \cap C)$ **by** (*simp add*: *ac-simps*)
qed

lemma *indep-events-index-subset*:
indep-events F E $\longleftrightarrow$ $(\forall J \subseteq E.$ *indep-events F J*)
unfolding *indep-events-def*
by (*meson image-mono set-eq-subset subset-trans*)

lemma *indep-events-index-subset2*:
indep-events F E $\Longrightarrow$ $J \subseteq E \Longrightarrow$ *indep-events F J*
using *indep-events-index-subset* **by** *auto*

lemma *indep-events-events-ss*: *indep-events F E* $\Longrightarrow$ F ‘ $E \subseteq$ *events*
unfolding *indep-events-def* **by** (*auto*)

lemma *indep-events-events*: *indep-events F E* $\Longrightarrow$ $(\bigwedge e.\ e \in E \Longrightarrow F\ e \in events)$
using *indep-events-events-ss* **by** *auto*

lemma *indep-events-probs*: *indep-events F E* $\Longrightarrow$ $J \subseteq E \Longrightarrow$ *finite J* $\Longrightarrow$ $J \neq \{\}$ $\Longrightarrow$ *prob* $(\bigcap (F$ ‘ $J)) = (\prod i \in J.$ *prob* $(F\ i))$
unfolding *indep-events-def* **by** *auto*

lemma *indep-events-prod-all*: *indep-events F E* $\Longrightarrow$ *finite E* $\Longrightarrow$ $E \neq \{\} \Longrightarrow$ *prob* $(\bigcap (F$ ‘ $E)) = (\prod i \in E.$ *prob* $(F\ i))$
using *indep-events-probs* **by** *auto*

lemma *indep-events-ev-not-contain-compl*:
assumes *indep-events F E*
assumes $A \in E$
assumes *prob* $(F\ A) > 0$ *prob* $(F\ A) < 1$
shows $(space\ M - F\ A) \notin F$ ‘ E (**is** $?A' \notin F$ ‘ E)
proof (*rule ccontr*)
assume $\neg\ ?A' \notin F$ ‘ E
then have $?A' \in F$ ‘ E **by** *auto*
then obtain *Ae* **where** *aeq*: $?A' = F\ Ae$ **and** $Ae \in E$ **by** *blast*
then have $\{A, Ae\} \subseteq E$ **using** *assms(2)* **by** *auto*
moreover have *finite* $\{A, Ae\}$ **by** *simp*
moreover have $\{A, Ae\} \neq \{\}$
by *simp*
ultimately have *prob* $(\bigcap i \in \{A, Ae\}.\ F\ i) = (\prod i \in \{A, Ae\}.$ *prob* $(F\ i))$ **using** *indep-events-probs*[*of F E* $\{A, Ae\}$] *assms(1)* **by** *auto*
moreover have $A \neq Ae$
using *subprob-not-empty* **using** *aeq* **by** *auto*
ultimately have *prob* $(F\ A \cap ?A') = prob\ (F\ A) * prob\ (?A')$ **using** *aeq* **by** *simp*
moreover have *prob* $(F\ A \cap ?A') = 0$ **by** *simp*

moreover have *prob* (*F A*) ∗ *prob ?A′* = *prob* (*F A*) ∗ (*1* − *prob* (*F A*))
using *assms*(*1*) *assms*(*2*) *indep-events-events prob-compl* **by** *metis*
moreover have *prob* (*F A*) ∗ (*1* − *prob* (*F A*)) > *0* **using** *assms*(*3*) *assms*(*4*)
by (*simp add*: *algebra-simps*)
ultimately show *False* **by** *auto*
qed

lemma *indep-events-singleton*:
assumes *F A* ∈ *events*
shows *indep-events F* {*A*}
proof (*intro indep-eventsI*)
show ⋀*i*. *i* ∈ {*A*} ⟹ *F i* ∈ *events* **using** *assms* **by** *simp*
next
fix *J* **assume** *J* ⊆ {*A*} *finite J J* ≠ {}
then have *J* = {*A*} **by** *auto*
then show *prob* (⋂ (*F* ‘ *J*)) = (∏ *i*∈*J*. *prob* (*F i*)) **by** *simp*
qed

lemma *indep-events-ev-pairs*:
assumes *indep-events F S*
assumes *A* ∈ *S B* ∈ *S A* ≠ *B*
shows *indep-event* (*F A*) (*F B*)
using *assms indep-events-probs*[*of F S* {*A*, *B*}]
by (*intro indep-eventI*) (*simp-all add*: *indep-events-events*)

lemma *indep-events-ev-inter-pairs*:
assumes *indep-events F S*
assumes *finite A finite B*
assumes *A* ≠ {} *B* ≠ {}
assumes *A* ⊆ *S B* ⊆ *S A* ∩ *B* = {}
shows *indep-event* (⋂(*F* ‘ *A*)) (⋂(*F* ‘ *B*))
proof (*intro indep-eventI*)
have (*F* ‘ *A*) ⊆ *events* (*F* ‘ *B*) ⊆ *events* **using** *indep-events-events assms*(*1*)
assms(*6*) *assms*(*7*) **by** *fast+*
then show ⋂ (*F* ‘ *A*) ∈ *events* ⋂ (*F* ‘*B*) ∈ *events* **using** *Inter-event-ss assms*
by *auto*
next
have *A* ∪ *B* ⊆ *S* **using** *assms* **by** *auto*
moreover have *finite* (*A* ∪ *B*) **using** *assms*(*2*) *assms*(*3*) **by** *simp*
moreover have *A* ∪ *B* ≠ {} **using** *assms* **by** *simp*
ultimately have *prob* (⋂(*F* ‘(*A* ∪ *B*))) = (∏ *i*∈*A* ∪ *B*. *prob* (*F i*)) **using** *assms*
using *indep-events-probs*[*of F S A* ∪ *B*] **by** *simp*
also have ... = (∏ *i*∈*A*. *prob* (*F i*)) ∗ (∏ *i*∈*B*. *prob* (*F i*))
using *assms*(*8*) *prod.union-disjoint*[*of A B* λ *i*. *prob* (*F i*)] *assms*(*2*) *assms*(*3*)
by *simp*
finally have *prob* (⋂(*F* ‘(*A* ∪ *B*))) = *prob* (⋂ (*F* ‘ *A*)) ∗ *prob* (⋂ (*F* ‘ *B*))
using *assms indep-events-index-subset indep-events-prod-all* **by** *metis*
moreover have ⋂ (*F* ‘ (*A* ∪ *B*)) = (⋂ (*F* ‘ *A*)) ∩ ⋂ (*F* ‘ *B*) **by** *auto*

ultimately show *prob* ($\bigcap$ (*F* ‘ *A*) ∩ $\bigcap$ (*F* ‘ *B*)) = *prob* ($\bigcap$ (*F* ‘ *A*)) ∗ *prob* ($\bigcap$ (*F* ‘ *B*))
by *simp*
qed

lemma *indep-events-ev-inter-single*:
assumes *indep-events F S*
assumes *finite B*
assumes *B* ≠ {}
assumes *A* ∈ *S B* ⊆ *S A* ∉ *B*
shows *indep-event* (*F A*) ($\bigcap$(*F* ‘ *B*))
proof −
have {*A*} ≠ {} *finite* {*A*} {*A*} ⊆ *S* **using** *assms* **by** *simp-all*
moreover have {*A*} ∩ *B* = {} **using** *assms*(*6*) **by** *auto*
ultimately show *?thesis* **using** *indep-events-ev-inter-pairs*[*of F S* {*A*} *B*] *assms*
by *auto*
qed

lemma *indep-events-fn-eq*:
assumes ⋀ *Ai*. *Ai* ∈ *E* ⟹ *F Ai* = *G Ai*
assumes *indep-events F E*
shows *indep-events G E*
proof (*intro indep-eventsI*)
show ⋀*i*. *i* ∈ *E* ⟹ *G i* ∈ *events* **using** *assms*(*2*) *indep-events-events assms*(*1*)
by *metis*
next
fix *J* **assume** *jss*: *J* ⊆ *E finite J J* ≠ {}
moreover have *G* ‘ *J* = *F* ‘ *J* **using** *assms*(*1*) *calculation*(*1*) **by** *auto*
moreover have ⋀ *i* . *i* ∈ *J* ⟹ *prob* (*G i*) = *prob* (*F i*) **using** *jss assms*(*1*)
by *auto*
moreover have (∏ *i*∈*J*. *prob* (*F i*)) = (∏ *i*∈*J*. *prob* (*G i*)) **using** *calculation*(*5*)
by *auto*
ultimately show *prob* ($\bigcap$ (*G* ‘ *J*)) = (∏ *i*∈*J*. *prob* (*G i*))
using *assms*(*2*) *indep-events-probs*[*of F E J*] **by** *simp*
qed

lemma *indep-events-fn-eq-iff*:
assumes ⋀ *Ai*. *Ai* ∈ *E* ⟹ *F Ai* = *G Ai*
shows *indep-events F E* ⟷ *indep-events G E*
using *indep-events-fn-eq assms* **by** *auto*

lemma *indep-events-one-compl*:
assumes *indep-events F S*
assumes *A* ∈ *S*
shows *indep-events* (λ *i*. *if* (*i* = *A*) *then* (*space M* − *F i*) *else F i*) *S* (**is** *indep-events ?G S*)
proof (*intro indep-eventsI*)
show ⋀*i*. *i* ∈ *S* ⟹ (*if i* = *A then space M* − *F i else F i*) ∈ *events*

```
    using indep-events-events assms(1) assms(2)
    by (metis sets.compl-sets)
next
  define G where G ≡?G
  fix J assume jss: J ⊆ S
  assume finJ: finite J
  assume jne: J ≠ {}
  show prob (⋂i∈J. ?G i) = (∏i∈J. prob (?G i))
  proof (cases J = {A})
    case True
    then show ?thesis by simp
  next
    case jne2: False
    have jss2: J − {A} ⊆ S using jss assms(2) by auto
    moreover have A ∉ (J − {A}) using jss by auto
    moreover have finite (J − {A}) using finJ by simp
    moreover have J − {A} ≠ {} using jne2 jne by auto
    ultimately have indep-event (F A) (⋂ (F ‘ (J − {A})))
      using indep-events-ev-inter-single[of F S (J − {A}) A] assms by auto
    then have ie: indep-event (G A) (⋂ (G ‘ (J − {A})))
      using indep-event-one-compl indep-event-commute G-def by auto
    have iess: indep-events G (J − {A})
      using jss2 G-def indep-events-index-subset2[of F S J − {A}] assms(1)
        indep-events-fn-eq[of J − {A}] by auto
    show ?thesis
    proof (cases A ∈ J)
      case True
      then have split: G ‘ J = insert (G A) (G ‘ (J − {A})) by auto
     then have prob (⋂ (G ‘ J)) = prob (⋂ (insert (G A) (G ‘ (J − {A})))) by
auto
      also have ... = prob ((G A)∩ ⋂ (G ‘ (J − {A})) )
        using Inter-insert by simp
      also have ... =  prob (G A) ∗ prob (⋂ (G ‘ (J − {A})))
        using ie indep-eventD[of G A ⋂ (G ‘ (J − {A})) ] by auto
      also have ... = prob (G A) ∗ (∏i ∈ (J − {A}). prob (G i))
        using indep-events-prod-all[of G J − {A}] iess jne2 jne finJ by auto
      finally have prob (⋂ (G ‘ J)) = (∏i ∈ J. prob (G i)) using split
        by (metis True finJ prod.remove)
      then show ?thesis using G-def by simp
    next
      case False
      then have prob (⋂i∈J. G i) = (∏i∈J. prob (G i)) using iess
        by (simp add: assms(1) finJ indep-events-prod-all jne)
      then show ?thesis using G-def by simp
    qed
  qed
qed

lemma indep-events-update-compl:
```

assumes *indep-events F E*
assumes $E = A \cup B$
assumes $A \cap B = \{\}$
assumes *finite E*
shows *indep-events* (λ *Ai. if* ($Ai \in A$) *then* (*space M* − (*F Ai*)) *else* (*F Ai*)) *E*
using *assms*(*2*) *assms*(*3*) **proof** (*induct card A arbitrary*: *A B*)
case *0*
let *?G* = (λ*Ai. if Ai* $\in$ *A then space M* − *F Ai else F Ai*)
have $E = B$ **using** *assms*(*4*) ‹$E = A \cup B$› ‹*0* = *card A*›
by *simp*
then have $\bigwedge$ *i. i* $\in$ *E* $\Longrightarrow$ *F i* = *?G i* **using** ‹$A \cap B = \{\}$› **by** *auto*
then show *?case* **using** *assms*(*1*) *indep-events-fn-eq*[*of E F ?G*] **by** *simp*
next
case (*Suc x*)
define *G* **where** *G* ≡ (λ*Ai. if Ai* $\in$ *A then space M* − *F Ai else F Ai*)
obtain *a A'* **where** *aeq*: *A* = *insert a A'* **and** *anotin*: $a \notin A'$
using *Suc.hyps* **by** (*metis card-Suc-eq-finite*)
then have *xcard*: *card A'* = *x*
using *Suc*(*2*) *Suc*(*3*) *assms*(*4*) **by** *auto*
define *G1* **where** *G1* ≡ (λ*Ai. if Ai* $\in$ *A' then space M* − *F Ai else F Ai*)
let *?B'* = *B* ∪ {*a*}
have *eeq*: *E* = *A'* ∪ *?B'* **using** *aeq Suc.prems* **by** *auto*
moreover have *A'* ∩ *?B'* = {} **using** *anotin Suc.prems*(*2*) *aeq* **by** *auto*
moreover have *?B'* ≠ {} **by** *simp*
ultimately have *ies*: *indep-events G1* (*A'* ∪ *?B'*)
using *Suc.hyps*(*1*)[*of A' ?B'*] *xcard G1-def* **by** *auto*
then have *a* $\in$ *A* ∪ *B* **using** *aeq* **by** *auto*
define *G2* **where** *G2* ≡ λ *Ai. if Ai* = *a then* (*space M* − (*G1 Ai*)) *else* (*G1 Ai*)
have *a* $\in$ *A'* ∪ *?B'* **by** *auto*
then have *ie*: *indep-events G2 E*
using *indep-events-one-compl*[*of G1* (*A'* ∪ *?B'*) *a*] *ies G2-def eeq* **by** *auto*
moreover have $\bigwedge$ *i. i* $\in$ *E* $\Longrightarrow$ *G2 i* = *G i*
unfolding *G2-def G1-def G-def*
by (*simp add*: *aeq anotin*)
ultimately have *indep-events G E* **using** *indep-events-fn-eq*[*of E G2 G*] **by** *auto*
then show *?case* **using** *G-def* **by** *simp*
qed

lemma *indep-events-compl*:
assumes *indep-events F E*
assumes *finite E*
shows *indep-events* (λ *Ai. space M* − *F Ai*) *E*
proof −
have *indep-events* (λ*Ai. if Ai* $\in$ *E then space M* − *F Ai else F Ai*) *E*
using *indep-events-update-compl*[*of F E E* {}] *assms* **by** *auto*
moreover have $\bigwedge$ *i. i* $\in$ *E* $\Longrightarrow$ (λ*Ai. if Ai* $\in$ *E then space M* − *F Ai else F Ai*) *i* = (λ *Ai. space M* − *F Ai*) *i*
by *simp*

ultimately show *?thesis*
using *indep-events-fn-eq*[*of E* (λ*Ai*. *if Ai* ∈ *E then space M* − *F Ai else F Ai*)]
by *auto*
qed

lemma *indep-events-impl-inj-on*:
assumes *finite A*
assumes *indep-events F A*
assumes ⋀ *A′* . *A′* ∈ *A* ⟹ *prob* (*F A′*) > *0* ∧ *prob* (*F A′*) < *1*
shows *inj-on F A*
proof (*intro inj-onI*, *rule ccontr*)
fix *x y* **assume** *xin*: *x* ∈ *A* **and** *yin*: *y* ∈ *A* **and** *feq*: *F x* = *F y*
assume *contr*: *x* ≠ *y*
then have {*x*, *y*} ⊆ *A* {*x*, *y*} ≠ {} *finite* {*x*, *y*} **using** *xin yin* **by** *auto*
then have *prob* (⋂*j*∈{*x*, *y*}. *F j*) = (∏*j*∈{*x*, *y*}. *prob* (*F j*))
using *assms*(*2*) *indep-events-probs*[*of F A* {*x*, *y*}] **by** *auto*
moreover have (∏*j*∈{*x*, *y*}. *prob* (*F j*)) = *prob* (*F x*) ∗ *prob* (*F y*) **using** *contr*
by *auto*
moreover have *prob* (⋂*j*∈{*x*, *y*}. *F j*) = *prob* (*F x*) **using** *feq* **by** *simp*
ultimately have *prob* (*F x*) = *prob* (*F x*) ∗ *prob* (*F x*) **using** *feq* **by** *simp*
then show *False* **using** *assms*(*3*) **using** *xin* **by** *fastforce*
qed

lemma *indep-events-imp-set*:
assumes *finite A*
assumes *indep-events F A*
assumes ⋀ *A′* . *A′* ∈ *A* ⟹ *prob* (*F A′*) > *0* ∧ *prob* (*F A′*) < *1*
shows *indep-events-set* (*F* ‘ *A*)
proof (*intro indep-events-setI*)
show *F* ‘ *A* ⊆ *events* **using** *assms*(*2*) *indep-events-events* **by** *auto*
next
fix *J* **assume** *jss*: *J* ⊆ *F* ‘ *A* **and** *finj*: *finite J* **and** *jne*:*J* ≠ {}
have *bb*: *bij-betw F A* (*F* ‘*A*) **using** *bij-betw-imageI indep-events-impl-inj-on*
assms **by** *meson*
then obtain *I* **where** *iss*: *I* ⊆ *A* **and** *jeq*: *J* = *F* ‘ *I*
using *bij-betw-obtain-subsetl*[*OF bb*] *jss* **by** *metis*
moreover have *I* ≠ {} *finite I* **using** *finj jeq jne assms*(*1*) *finite-subset iss* **by**
blast+
ultimately have *prob* (⋂ (*F* ‘ *I*)) = (∏ *i*∈*I*. *prob* (*F i*))
using *jne finj jss indep-events-probs*[*of F A I*] *assms*(*2*) **by** (*simp*)
moreover have *bij-betw F I J* **using** *jeq iss jss bb* **by** (*meson bij-betw-subset*)
ultimately show *prob* (⋂ *J*) = *prod prob J* **using** *bij-betw-prod-prob jeq* **by**
(*metis*)
qed

lemma *indep-event-set-equiv-bij*:
assumes *bij-betw F A E*
assumes *finite E*
shows *indep-events-set E* ⟷ *indep-events F A*

proof −
have *im*: F ‘ $A = E$
using *assms*(*1*) **by** (*simp add*: *bij-betw-def*)
then have *ss*: $(\forall\, e.\ e \in E \longrightarrow e \in events) \longleftrightarrow (F$ ‘ $A \subseteq events)$
using *image-iff* **by** (*simp add*: *subset-iff*)
have *prob*: $(\forall\ J.\ J \subseteq E \longrightarrow finite\ J \longrightarrow J \neq \{\} \longrightarrow prob\ (\bigcap i{\in}J.\ i) = (\prod i{\in}J.\ prob\ i)) \longleftrightarrow$
$(\forall\ I.\ I \subseteq A \longrightarrow finite\ I \longrightarrow I \neq \{\} \longrightarrow prob\ (\bigcap i{\in}I.\ F\ i) = (\prod i{\in}I.\ prob\ (F\ i)))$
proof (*intro allI impI iffI*)
fix I **assume** *p1*: $\forall J{\subseteq}E.\ finite\ J \longrightarrow J \neq \{\} \longrightarrow prob\ (\bigcap i{\in}J.\ i) = prod\ prob\ J$
and *iss*: $I \subseteq A$ **and** *f1*: *finite* I **and** *i1*: $I \neq \{\}$
then obtain J **where** *jeq*: $J = F$ ‘ I **and** *jss*: $J \subseteq E$
using *bij-betw-obtain-subsetr*[*OF assms*(*1*) *iss*]**by** *metis*
then have *prob* $(\bigcap J) = prod\ prob\ J$ **using** *i1 f1 p1 jss* **by** *auto*
moreover have *bij-betw F I J* **using** *jeq jss assms*(*1*) *iss*
by (*meson bij-betw-subset*)
ultimately show *prob* $(\bigcap\ (F$ ‘ $I)) = (\prod i{\in}I.\ prob\ (F\ i))$ **using** *bij-betw-prod-prob*
by (*metis jeq*)
next
fix J **assume** *p2*: $\forall I{\subseteq}A.\ finite\ I \longrightarrow I \neq \{\} \longrightarrow prob\ (\bigcap\ (F$ ‘ $I)) = (\prod i{\in}I.\ prob\ (F\ i))$
and *jss*: $J \subseteq E$ **and** *f2*: *finite* J **and** *j1*: $J \neq \{\}$
then obtain I **where** *iss*: $I \subseteq A$ **and** *jeq*: $J = F$ ‘ I
using *bij-betw-obtain-subsetl*[*OF assms*(*1*)] **by** *metis*
moreover have *finite A* **using** *assms*(*1*) *assms*(*2*)
by (*simp add*: *bij-betw-finite*)
ultimately have *prob* $(\bigcap\ (F$ ‘ $I)) = (\prod i{\in}I.\ prob\ (F\ i))$ **using** *j1 f2 p2 jss*
by (*simp add*: *finite-subset*)
moreover have *bij-betw F I J* **using** *jeq iss assms*(*1*) *jss* **by** (*meson bij-betw-subset*)
ultimately show *prob* $(\bigcap i{\in}J.\ i) = prod\ prob\ J$ **using** *bij-betw-prod-prob jeq*
by (*metis image-ident*)
qed
have *indep-events-set E* $\Longrightarrow$ *indep-events F A*
proof (*intro indep-eventsI*)
show $\bigwedge i.$ *indep-events-set E* $\Longrightarrow i \in A \Longrightarrow F\ i \in events$
using *indep-events-set-events ss* **by** *auto*
show $\bigwedge J.$ *indep-events-set E* $\Longrightarrow J \subseteq A \Longrightarrow$ *finite J* $\Longrightarrow J \neq \{\} \Longrightarrow prob\ (\bigcap (F$ ‘ $J)) = (\prod i{\in}J.\ prob\ (F\ i))$
using *indep-events-set-probs prob* **by** *auto*
qed
moreover have *indep-events F A* $\Longrightarrow$ *indep-events-set E*
proof (*intro indep-events-setI*)
have $\bigwedge e.$ *indep-events F A* $\Longrightarrow e \in E \Longrightarrow e \in events$ **using** *ss indep-events-def* **by** *metis*
then show *indep-events F A* $\Longrightarrow E \subseteq events$ **by** *auto*
show $\bigwedge J.$ *indep-events F A* $\Longrightarrow J \subseteq E \Longrightarrow$ *finite J* $\Longrightarrow J \neq \{\} \Longrightarrow prob\ (\bigcap J) = prod\ prob\ J$

using *prob indep-events-def* **by** (*metis image-ident*)
qed
ultimately show *?thesis* **by** *auto*
qed

5.3 Mutual Independent Events

Note, set based version only if no duplicates in usage case. The mutual_indep_events definition is more general and recommended

definition *mutual-indep-set*:: *'a set ⇒ 'a set set ⇒ bool*
where *mutual-indep-set A S ⟷ A ∈ events ∧ S ⊆ events ∧ (∀ T ⊆ S . T ≠ {} ⟶ prob (A ∩ (⋂ T)) = prob A * prob (⋂ T))*

lemma *mutual-indep-setI*[*intro*]: *A ∈ events ⟹ S ⊆ events ⟹ (⋀ T. T ⊆ S ⟹ T ≠ {} ⟹*
*prob (A ∩ (⋂ T)) = prob A * prob (⋂ T)) ⟹ mutual-indep-set A S*
using *mutual-indep-set-def* **by** *simp*

lemma *mutual-indep-setD*[*dest*]: *mutual-indep-set A S ⟹ T ⊆ S ⟹ T ≠ {} ⟹ prob (A ∩ (⋂ T)) = prob A * prob (⋂ T)*
using *mutual-indep-set-def* **by** *simp*

lemma *mutual-indep-setD2*[*dest*]: *mutual-indep-set A S ⟹ A ∈ events*
using *mutual-indep-set-def* **by** *simp*

lemma *mutual-indep-setD3*[*dest*]: *mutual-indep-set A S ⟹ S ⊆ events*
using *mutual-indep-set-def* **by** *simp*

lemma *mutual-indep-subset*: *mutual-indep-set A S ⟹ T ⊆ S ⟹ mutual-indep-set A T*
using *mutual-indep-set-def* **by** *auto*

lemma *mutual-indep-event-set-defD*:
assumes *mutual-indep-set A S*
assumes *finite T*
assumes *T ⊆ S*
assumes *T ≠ {}*
shows *indep-event A (⋂ T)*
proof (*intro indep-eventI*)
show *A ∈ events* **using** *mutual-indep-setD2 assms(1)* **by** *auto*
show *⋂ T ∈ events* **using** *Inter-event-ss assms mutual-indep-setD3 finite-subset* **by** *blast*
show *prob (A ∩ ⋂ T) = prob A * prob (⋂ T)*
using *assms(1) mutual-indep-setD assms(3) assms(4)* **by** *simp*
qed

lemma *mutual-indep-event-defI*: *A ∈ events ⟹ S ⊆ events ⟹ (⋀ T. T ⊆ S*

$\Longrightarrow T \neq \{\} \Longrightarrow$
indep-event A $(\bigcap T)) \Longrightarrow$ *mutual-indep-set A S*
using *indep-eventD mutual-indep-set-def* **by** *simp*

lemma *mutual-indep-singleton-event*: *mutual-indep-set A S* $\Longrightarrow B \in S \Longrightarrow$ *indep-event A B*
using *mutual-indep-event-set-defD empty-subsetI*
by (*metis Set.insert-mono cInf-singleton finite.emptyI finite-insert insert-absorb insert-not-empty*)

lemma *mutual-indep-cond*:
assumes $A \in$ *events* **and** $T \subseteq$ *events* **and** *finite T*
and *mutual-indep-set A S* **and** $T \subseteq S$ **and** $T \neq \{\}$ **and** *prob* $(\bigcap T) \neq 0$
shows $\mathcal{P}(A \mid (\bigcap T)) =$ *prob A*
proof −
have $\bigcap T \in$ *events* **using** *assms*
by (*simp add*: *Inter-event-ss*)
then have $\mathcal{P}(A \mid (\bigcap T)) =$ *prob* $((\bigcap T) \cap A)/$*prob*$(\bigcap T)$ **using** *cond-prob-ev-def assms*(*1*)
by *blast*
also have ... = *prob* $(A \cap (\bigcap T))/$*prob*$(\bigcap T)$
by (*simp add*: *inf-commute*)
also have ... = *prob A* ∗ *prob* $(\bigcap T)/$*prob*$(\bigcap T)$ **using** *assms mutual-indep-setD* **by** *auto*
finally show *?thesis* **using** *assms*(*7*) **by** *simp*
qed

lemma *mutual-indep-cond-full*:
assumes $A \in$ *events* **and** $S \subseteq$ *events* **and** *finite S*
and *mutual-indep-set A S* **and** $S \neq \{\}$ **and** *prob* $(\bigcap S) \neq 0$
shows $\mathcal{P}(A \mid (\bigcap S)) =$ *prob A*
using *mutual-indep-cond*[*of A S S*] *assms* **by** *auto*

lemma *mutual-indep-cond-single*:
assumes $A \in$ *events* **and** $B \in$ *events*
and *mutual-indep-set A S* **and** $B \in S$ **and** *prob* $B \neq 0$
shows $\mathcal{P}(A \mid B) =$ *prob A*
using *mutual-indep-cond*[*of A* $\{B\}$ *S*] *assms* **by** *auto*

lemma *mutual-indep-set-empty*: $A \in$ *events* $\Longrightarrow$ *mutual-indep-set A* $\{\}$
using *mutual-indep-setI* **by** *auto*

lemma *not-mutual-indep-set-itself*:
assumes *prob A* > *0* **and** *prob A* < *1*
shows ¬ *mutual-indep-set A* $\{A\}$
proof (*rule ccontr*)
assume ¬ ¬ *mutual-indep-set A* $\{A\}$
then have *mutual-indep-set A* $\{A\}$
by *simp*

then have $\bigwedge T .\ T \subseteq \{A\} \Longrightarrow T \neq \{\} \Longrightarrow prob\ (A \cap (\bigcap\ T)) = prob\ A * prob\ (\bigcap T)$
using *mutual-indep-setD* **by** *simp*
then have *eq*: $prob\ (A \cap (\bigcap\ \{A\})) = prob\ A * prob\ (\bigcap \{A\})$
by *blast*
have $prob\ (A \cap (\bigcap \{A\})) = prob\ A$ **by** *simp*
moreover have $prob\ A * (prob\ (\bigcap\ \{A\})) = (prob\ A)\hat{}2$
by (*simp add*: *power2-eq-square*)
ultimately show *False* **using** *eq assms* **by** *auto*
qed

lemma *is-mutual-indep-set-itself*:
assumes $A \in events$
assumes $prob\ A = 0 \vee prob\ A = 1$
shows *mutual-indep-set A* $\{A\}$
proof (*intro mutual-indep-setI*)
show $A \in events$ $\{A\} \subseteq events$ **using** *assms*(*1*) **by** *auto*
fix T **assume** $T \subseteq \{A\}$ **and** $T \neq \{\}$
then have *teq*: $T = \{A\}$ **by** *auto*
have $prob\ (A \cap (\bigcap \{A\})) = prob\ A$ **by** *simp*
moreover have $prob\ A * (prob\ (\bigcap\ \{A\})) = (prob\ A)\hat{}2$
by (*simp add*: *power2-eq-square*)
ultimately show $prob\ (A \cap (\bigcap\ T)) = prob\ A * prob\ (\bigcap T)$ **using** *teq assms*
by *auto*
qed

lemma *mutual-indep-set-singleton*:
assumes *indep-event A B*
shows *mutual-indep-set A* $\{B\}$
using *indep-eventD-ev1 indep-eventD-ev2 assms*
by (*intro mutual-indep-event-defI*) (*simp-all add*: *subset-singleton-iff*)

lemma *mutual-indep-set-one-compl*:
assumes *mutual-indep-set A S*
assumes *finite S*
assumes $B \in S$
shows *mutual-indep-set A* $(\{space\ M - B\} \cup S)$
proof (*intro mutual-indep-event-defI*)
show $A \in events$ **using** *assms*(*1*) *mutual-indep-setD2* **by** *auto*
next
show $\{space\ M - B\} \cup (S) \subseteq events$
using *assms*(*1*) *assms*(*2*) *mutual-indep-setD3 assms*(*3*) **by** *blast*
next
fix T **assume** *jss*: $T \subseteq \{space\ M - B\} \cup (S)$
assume *tne*: $T \neq \{\}$
let $?T' = T - \{space\ M - B\}$
show *indep-event A* $(\bigcap\ T)$
proof (*cases* $?T' = \{\}$)
case *True*

then have $T = \{space\ M - B\}$ **using** *tne* **by** *blast*
moreover have *indep-event A B* **using** *assms(1) assms(3) assms(3) mutual-indep-singleton-event* **by** *auto*
ultimately show *?thesis* **using** *indep-event-one-compl* **by** *auto*
next
case *tne2*: *False*
have *finT*: *finite T* **using** *jss assms(2) finite-subset* **by** *fast*
have *tss2*: $?T' \subseteq S$ **using** *jss assms(2)* **by** *auto*
show *?thesis* **proof** (*cases space M − B ∈ T*)
case *True*
have $?T' \cup \{B\} \subseteq S$ **using** *assms(3) tss2* **by** *auto*
then have *indep-event A* $(\bigcap (?T' \cup \{B\}))$ **using** *assms(1) mutual-indep-event-set-defD tne2 finT*
by (*meson Un-empty assms(2) finite-subset*)
moreover have *indep-event A* $(\bigcap ?T')$
using *assms(1) mutual-indep-event-set-defD finT finite-subset tss2 tne2* **by** *auto*
moreover have $\bigcap (?T' \cup \{B\}) = B \cap (\bigcap ?T')$ **by** *auto*
moreover have $B \in events$ **using** *assms(3) assms(1) mutual-indep-setD3* **by** *auto*
ultimately have *indep-event A* $((space\ M - B) \cap (\bigcap ?T'))$ **using** *indep-event-compl-inter* **by** *auto*
then show *?thesis*
by (*metis Inter-insert True insert-Diff*)
next
case *False*
then have $T \subseteq S$ **using** *jss* **by** *auto*
then show *?thesis* **using** *assms(1) mutual-indep-event-set-defD finT tne* **by** *auto*
qed
qed
qed

lemma *mutual-indep-events-set-update-compl*:
assumes *mutual-indep-set X E*
assumes $E = A \cup B$
assumes $A \cap B = \{\}$
assumes *finite E*
shows *mutual-indep-set X* $(((-)\ (space\ M)\ `\ A) \cup B)$
using *assms(2) assms(3)* **proof** (*induct card A arbitrary*: *A B*)
case *0*
then show *?case* **using** *assms(1)*
using *assms(4)* **by** *auto*
next
case (*Suc x*)
then obtain *a A'* **where** *aeq*: $A = insert\ a\ A'$ **and** *anotin*: $a \notin A'$
by (*metis card-Suc-eq-finite*)
then have *xcard*: $card\ A' = x$
using *Suc(2) Suc(3) assms(4)* **by** *auto*

let *?B′ = B ∪ {a}*
have *E = A′ ∪ ?B′* **using** *aeq Suc.prems* **by** *auto*
moreover have *A′ ∩ ?B′ = {}* **using** *anotin Suc.prems(2) aeq* **by** *auto*
ultimately have *ies*: *mutual-indep-set X ((−) (space M) ‘ A′ ∪ ?B′)*
using *Suc.hyps(1)[of A′ ?B′] xcard* **by** *auto*
then have *a ∈ A ∪ B* **using** *aeq* **by** *auto*
then show *?case*
proof (*cases (A ∪ B) − {a} = {}*)
case *True*
then have *A = {a} B = {}* **using** *Suc.prems aeq* **by** *auto*
moreover have *indep-event X a* **using** *mutual-indep-singleton-event ies* **by** *auto*
ultimately show *?thesis* **using** *mutual-indep-set-singleton indep-event-one-compl* **by** *simp*
next
case *False*
let *?c = (−) (space M)*
have *un*: *?c ‘ A ∪ B = ?c ‘ A′ ∪ ({?c a}) ∪ (?B′ − {a})*
using *Suc(4) aeq* **by** *force*
moreover have *?B′ − {a} ⊆ ?B′* **by** *auto*
moreover have *?B′ − {a} ⊆ ?c ‘ A′ ∪ {?c a} ∪ (?B′)* **by** *auto*
moreover have *?c ‘ A′ ∪ {?c a} ⊆ ?c ‘ A′ ∪ {?c a} ∪ (?B′)* **by** *auto*
ultimately have *ss*: *?c ‘ A ∪ B ⊆ {?c a} ∪ (?c ‘ A′ ∪ ?B′)*
using *Un-least* **by** *auto*
have *a ∈ (−) (space M) ‘ A′ ∪ ?B′* **using** *aeq* **by** *auto*
then have *ie*: *mutual-indep-set X ({?c a} ∪ (?c ‘ A′ ∪ ?B′))*
using *mutual-indep-set-one-compl[of X ?c ‘ A′ ∪ ?B′ a] ies ‹E = A′ ∪ (B ∪ {a})› assms(4)* **by** *blast*
then show *?thesis* **using** *mutual-indep-subset ss* **by** *auto*
qed
qed

lemma *mutual-indep-events-compl*:
assumes *finite S*
assumes *mutual-indep-set A S*
shows *mutual-indep-set A ((λ s . space M − s) ‘ S)*
using *mutual-indep-events-set-update-compl[of A S S {}] assms* **by** *auto*

lemma *mutual-indep-set-all*:
assumes *A ⊆ events*
assumes *⋀ Ai. Ai ∈ A ⟹ (mutual-indep-set Ai (A − {Ai}))*
shows *indep-events-set A*
proof (*intro indep-events-setI*)
show *A ⊆ events*
using *assms(1)* **by** *auto*
next
fix *J* **assume** *ss*: *J ⊆ A* **and** *fin*: *finite J* **and** *ne*: *J ≠ {}*
from *fin ne ss* **show** *prob (⋂ J) = prod prob J*
proof (*induct J rule*: *finite-ne-induct*)

case (*singleton x*)
then show *?case* **by** *simp*
next
case (*insert x F*)
then have *mutual-indep-set x* (*A* − {*x*}) **using** *assms*(*2*) **by** *simp*
moreover have *F* ⊆ (*A* − {*x*}) **using** *insert.prems insert.hyps* **by** *auto*
ultimately have *prob* (*x* ∩ (⋂ *F*)) = *prob x* ∗ *prob* (⋂ *F*)
by (*simp add*: *local.insert*(*2*) *mutual-indep-setD*)
then show *?case* **using** *insert.hyps insert.prems* **by** *simp*
qed
qed

Prefered version using indexed notation

definition *mutual-indep-events*:: *'a set* ⇒ (*nat* ⇒ *'a set*) ⇒ *nat set* ⇒ *bool*
where *mutual-indep-events A F I* ⟷ *A* ∈ *events* ∧ (*F* ‘ *I* ⊆ *events*) ∧ (∀ *J* ⊆ *I* . *J* ≠ {} ⟶ *prob* (*A* ∩ (⋂*j* ∈ *J* . *F j*)) = *prob A* ∗ *prob* (⋂*j* ∈ *J* . *F j*))

lemma *mutual-indep-eventsI*[*intro*]: *A* ∈ *events* ⟹ (*F* ‘ *I* ⊆ *events*) ⟹ (⋀ *J*. *J* ⊆ *I* ⟹ *J* ≠ {} ⟹
prob (*A* ∩ (⋂*j* ∈ *J* . *F j*)) = *prob A* ∗ *prob* (⋂*j* ∈ *J* . *F j*)) ⟹ *mutual-indep-events A F I*
using *mutual-indep-events-def* **by** *simp*

lemma *mutual-indep-eventsD*[*dest*]: *mutual-indep-events A F I* ⟹ *J* ⊆ *I* ⟹ *J* ≠ {} ⟹ *prob* (*A* ∩ (⋂*j* ∈ *J* . *F j*)) = *prob A* ∗ *prob* (⋂*j* ∈ *J* . *F j*)
using *mutual-indep-events-def* **by** *simp*

lemma *mutual-indep-eventsD2*[*dest*]: *mutual-indep-events A F I* ⟹ *A* ∈ *events*
using *mutual-indep-events-def* **by** *simp*

lemma *mutual-indep-eventsD3*[*dest*]: *mutual-indep-events A F I* ⟹ *F* ‘ *I* ⊆ *events*
using *mutual-indep-events-def* **by** *simp*

lemma *mutual-indep-ev-subset*: *mutual-indep-events A F I* ⟹ *J* ⊆ *I* ⟹ *mutual-indep-events A F J*
using *mutual-indep-events-def* **by** (*meson image-mono subset-trans*)

lemma *mutual-indep-event-defD*:
assumes *mutual-indep-events A F I*
assumes *finite J*
assumes *J* ⊆ *I*
assumes *J* ≠ {}
shows *indep-event A* (⋂*j* ∈ *J* . *F j*)
proof (*intro indep-eventI*)
show *A* ∈ *events* **using** *mutual-indep-setD2 assms*(*1*) **by** *auto*
show *prob* (*A* ∩ ⋂ (*F* ‘ *J*)) = *prob A* ∗ *prob* (⋂ (*F* ‘ *J*))

using *assms(1) mutual-indep-eventsD assms(3) assms(4)* **by** *simp*
have *finite* $(F \mathbin{`} J)$ **using** *finite-subset assms(2)* **by** *simp*
then show $(\bigcap j \in J \,.\, F\ j) \in events$
using *Inter-event-ss[of F ‘ J] assms mutual-indep-eventsD3* **by** *blast*
qed

lemma *mutual-ev-indep-event-defI*: $A \in events \Longrightarrow F \mathbin{`} I \subseteq events \Longrightarrow (\bigwedge J.\ J \subseteq I \Longrightarrow J \neq \{\} \Longrightarrow$
indep-event A $(\bigcap (F \mathbin{`} J))) \Longrightarrow$ *mutual-indep-events A F I*
using *indep-eventD mutual-indep-events-def[of A F I]* **by** *auto*

lemma *mutual-indep-ev-singleton-event*:
assumes *mutual-indep-events A F I*
assumes $B \in F \mathbin{`} I$
shows*indep-event A B*
proof −
obtain J **where** *beq*: $B = F\ J$ **and** $J \in I$ **using** *assms(2)* **by** *blast*
then have $\{J\} \subseteq I$ **and** *finite* $\{J\}$ **and** $\{J\} \neq \{\}$ **by** *auto*
moreover have $B = \bigcap\ (F \mathbin{`} \{J\})$ **using** *beq* **by** *simp*
ultimately show *?thesis* **using** *mutual-indep-event-defD assms(1)*
by *meson*
qed

lemma *mutual-indep-ev-singleton-event2*:
assumes *mutual-indep-events A F I*
assumes $i \in I$
shows*indep-event A* $(F\ i)$
using *mutual-indep-event-defD[of A F I* $\{i\}$*] assms* **by** *auto*

lemma *mutual-indep-iff*:
shows *mutual-indep-events A F I* $\longleftrightarrow$ *mutual-indep-set A* $(F \mathbin{`} I)$
proof (*intro iffI mutual-indep-setI mutual-indep-eventsI*)
show *mutual-indep-events A F I* $\Longrightarrow A \in events$ **using** *mutual-indep-eventsD2* **by** *simp*
show *mutual-indep-set A* $(F \mathbin{`} I) \Longrightarrow A \in events$ **using** *mutual-indep-setD2* **by** *simp*
show *mutual-indep-events A F I* $\Longrightarrow F \mathbin{`} I \subseteq events$ **using** *mutual-indep-eventsD3* **by** *simp*
show *mutual-indep-set A* $(F \mathbin{`} I) \Longrightarrow F \mathbin{`} I \subseteq events$ **using** *mutual-indep-setD3* **by** *simp*
show $\bigwedge T.$ *mutual-indep-events A F I* $\Longrightarrow T \subseteq F \mathbin{`} I \Longrightarrow T \neq \{\} \Longrightarrow$ *prob* $(A \cap \bigcap\ T) =$ *prob A* $*$ *prob* $(\bigcap\ T)$
using *mutual-indep-eventsD* **by** (*metis empty-is-image subset-imageE*)
show $\bigwedge J.$ *mutual-indep-set A* $(F \mathbin{`} I) \Longrightarrow J \subseteq I \Longrightarrow J \neq \{\} \Longrightarrow$ *prob* $(A \cap \bigcap (F \mathbin{`} J)) =$ *prob A* $*$ *prob* $(\bigcap\ (F \mathbin{`} J))$
using *mutual-indep-setD* **by** (*simp add*: *image-mono*)
qed

lemma *mutual-indep-ev-cond*:

assumes $A \in$ *events* **and** F ‘ $J \subseteq$ *events* **and** *finite* J
and *mutual-indep-events* A F I **and** $J \subseteq I$ **and** $J \neq \{\}$ **and** *prob* $(\bigcap (F$ ‘$J)) \neq 0$
shows $\mathcal{P}(A\ |(\bigcap (F$ ‘ $J))) =$ *prob* A
proof −
have $\bigcap (F$ ‘ $J) \in$ *events* **using** *assms*
by (*simp add*: *Inter-event-ss*)
then have $\mathcal{P}(A \mid (\bigcap (F$ ‘ $J))) =$ *prob* $((\bigcap (F$ ‘ $J)) \cap A)/$*prob*$(\bigcap (F$ ‘ $J))$
using *cond-prob-ev-def assms*(*1*) **by** *blast*
also have ... = *prob* $(A \cap (\bigcap (F$ ‘ $J)))/$*prob*$(\bigcap (F$ ‘ $J))$
by (*simp add*: *inf-commute*)
also have ... = *prob* $A *$ *prob* $(\bigcap (F$ ‘ $J))/$*prob*$(\bigcap (F$ ‘ $J))$
using *assms mutual-indep-eventsD* **by** *auto*
finally show *?thesis* **using** *assms*(*7*) **by** *simp*
qed

lemma *mutual-indep-ev-cond-full*:
assumes $A \in$ *events* **and** F ‘ $I \subseteq$ *events* **and** *finite* I
and *mutual-indep-events* A F I **and** $I \neq \{\}$ **and** *prob* $(\bigcap (F$ ‘ $I)) \neq 0$
shows $\mathcal{P}(A\ |(\bigcap (F$ ‘$I))) =$ *prob* A
using *mutual-indep-ev-cond*[*of* A F I I] *assms* **by** *auto*

lemma *mutual-indep-ev-cond-single*:
assumes $A \in$ *events* **and** $B \in$ *events*
and *mutual-indep-events* A F I **and** $B \in F$ ‘ I **and** *prob* $B \neq 0$
shows $\mathcal{P}(A\ |B) =$ *prob* A
proof −
obtain i **where** $B = F\ i$ **and** $i \in I$ **using** *assms* **by** *blast*
then show *?thesis* **using** *mutual-indep-ev-cond*[*of* A F $\{i\}$ I] *assms* **by** *auto*
qed

lemma *mutual-indep-ev-empty*: $A \in$ *events* $\Longrightarrow$ *mutual-indep-events* A F $\{\}$
using *mutual-indep-eventsI* **by** *auto*

lemma *not-mutual-indep-ev-itself*:
assumes *prob* $A > 0$ **and** *prob* $A < 1$ **and** $A = F\ i$
shows ¬ *mutual-indep-events* A F $\{i\}$
proof (*rule ccontr*)
assume ¬ ¬ *mutual-indep-events* A F $\{i\}$
then have *mutual-indep-events* A F $\{i\}$
by *simp*
then have $\bigwedge J$. $J \subseteq \{i\} \Longrightarrow J \neq \{\} \Longrightarrow$ *prob* $(A \cap (\bigcap\ (F$ ‘ $J))) =$ *prob* $A *$ *prob* $(\bigcap (F$ ‘ $J))$
using *mutual-indep-eventsD* **by** *simp*
then have *eq*: *prob* $(A \cap (\bigcap\ (F$ ‘$\{i\}))) =$ *prob* $A *$ *prob* $(\bigcap (F$ ‘ $\{i\}))$
by *blast*
have *prob* $(A \cap (\bigcap (F$ ‘$\{i\}))) =$ *prob* A **using** *assms*(*3*) **by** *simp*
moreover have *prob* $A *$ (*prob* $(\bigcap\ \{A\})) = ($*prob* $A)\hat{}2$
by (*simp add*: *power2-eq-square*)

ultimately show *False* **using** *eq assms* **by** *auto*
qed

lemma *is-mutual-indep-ev-itself*:
assumes $A \in$ *events* **and** $A = F\ i$
assumes *prob* $A = 0 \vee$ *prob* $A = 1$
shows *mutual-indep-events* $A\ F\ \{i\}$
proof (*intro mutual-indep-eventsI*)
show $A \in$ *events* F ' $\{i\} \subseteq$ *events* **using** *assms*(*1*) *assms*(*2*) **by** *auto*
fix J **assume** $J \subseteq \{i\}$ **and** $J \neq \{\}$
then have *teq*: $J = \{i\}$ **by** *auto*
have *prob* $(A \cap (\bigcap (F\ `\{i\}))) =$ *prob* A **using** *assms*(*2*) **by** *simp*
moreover have *prob* $A *$ (*prob* $(\bigcap\ (F\ `\{i\}))) =$ (*prob* $A)\hat{}2$
using *assms*(*2*) **by** (*simp add*: *power2-eq-square*)
ultimately show *prob* $(A \cap \bigcap\ (F\ `\ J)) =$ *prob* $A *$ *prob* $(\bigcap\ (F\ `\ J))$ **using** *teq assms* **by** *auto*
qed

lemma *mutual-indep-ev-singleton*:
assumes *indep-event* $A\ (F\ i)$
shows *mutual-indep-events* $A\ F\ \{i\}$
using *indep-eventD-ev1 indep-eventD-ev2 assms*
by (*intro mutual-ev-indep-event-defI*) (*simp-all add*: *subset-singleton-iff*)

lemma *mutual-indep-ev-one-compl*:
assumes *mutual-indep-events* $A\ F\ I$
assumes *finite* I
assumes $i \in I$
assumes *space* $M - F\ i = F\ j$
shows *mutual-indep-events* $A\ F\ (\{j\} \cup\ I)$
proof (*intro mutual-ev-indep-event-defI*)
show $A \in$ *events* **using** *assms*(*1*) *mutual-indep-setD2* **by** *auto*
next
show F ' $(\{j\} \cup I) \subseteq$ *events*
using *assms*(*1*) *assms*(*2*) *mutual-indep-eventsD3 assms*(*3*) *assms*(*4*)
by (*metis image-insert image-subset-iff insert-is-Un insert-subset sets.compl-sets*)

next
fix J **assume** *jss*: $J \subseteq \{j\} \cup I$
assume *tne*: $J \neq \{\}$
let *?J'* $= J - \{j\}$
show *indep-event* $A\ (\bigcap\ (F\ `\ J))$
proof (*cases ?J'* $= \{\}$)
case *True*
then have $J = \{j\}$ **using** *tne* **by** *blast*
moreover have *indep-event* $A\ (F\ i)$
using *assms*(*1*) *assms mutual-indep-ev-singleton-event2* **by** *simp*
ultimately show *?thesis* **using** *indep-event-one-compl assms*(*4*) **by** *fastforce*
next

```
    case tne2: False
    have finT: finite J using jss assms(2) finite-subset by fast
    have tss2: ?J' ⊆ I using jss assms(2) by auto
    show ?thesis proof (cases j ∈ J)
      case True
      have ?J' ∪ {i} ⊆ I using assms(3) tss2 by auto
      then have indep-event A (⋂(F ` ?J' ∪ { F i}))
        using assms(1) mutual-indep-event-defD tne2 finT  assms(2) finite-subset
       by (metis Diff-cancel Un-Diff-cancel Un-absorb Un-insert-right image-insert)

      moreover have indep-event A (⋂(F ` ?J'))
       using assms(1) mutual-indep-event-defD finT finite-subset tss2 tne2 by auto
      moreover have (⋂(F ` ?J' ∪ { F i})) = F i ∩ (⋂(F ` ?J')) by auto
     moreover have F i ∈ events using assms(3) assms(1) mutual-indep-eventsD3
by simp
      ultimately have indep-event A (F j ∩ (⋂(F ` ?J')))
        using indep-event-compl-inter[of A ⋂(F ` ?J') F i] assms(4) by auto
     then show ?thesis using Inter-insert True insert-Diff by (metis image-insert)

    next
      case False
      then have J ⊆ I using jss by auto
     then show ?thesis using assms(1) mutual-indep-event-defD finT tne by auto
    qed
  qed
qed

lemma mutual-indep-events-update-compl:
  assumes mutual-indep-events X F S
  assumes S = A ∪ B
  assumes A ∩ B = {}
  assumes finite S
  assumes bij-betw G A A'
  assumes ⋀ i. i ∈ A ⟹ F (G i) = space M − F i
  shows mutual-indep-events X F (A' ∪ B)
using assms(2) assms(3) assms(6) assms(5) proof (induct card A arbitrary: A
B A')
  case 0
  then have aempty: A = {} using finite-subset assms(4) by simp
 then have A' = {} using 0.prems(4) by (metis all-not-in-conv bij-betwE bij-betw-inv)
  then show ?case using assms(1) using 0.prems(1) aempty by simp
next
  case (Suc x)
  then obtain a C where aeq: C = A − {a} and ain: a ∈ A
    by fastforce
  then have xcard: card C = x
    using Suc(2) Suc(3) assms(4) by auto
  let ?C' = A' − {G a}
  have compl:  (⋀i. i ∈ C ⟹ F (G i) = space M − F i) using Suc.prems aeq
```

by *simp*
have *bb*: *bij-betw G C ?C′* **using** *Suc.prems*(*4*) *aeq bij-betw-remove*[*of G A A′ a*] *ain* **by** *simp*
let *?B′* = *B* ∪ {*a*}
have *S* = *C* ∪ *?B′* **using** *aeq Suc.prems ain* **by** *auto*
moreover have *C* ∩ *?B′* = {} **using** *ain Suc.prems*(*2*) *aeq* **by** *auto*
ultimately have *ies*: *mutual-indep-events X F* (*?C′* ∪ *?B′*)
using *Suc.hyps*(*1*)[*of C ?B′*] *xcard compl bb* **by** *auto*
then have *a* ∈ *A* ∪ *B* **using** *ain* **by** *auto*
then show *?case*
proof (*cases* (*A* ∪ *B*) − {*a*} = {})
case *True*
then have *aeq*: *A* = {*a*} **and** *beq*: *B* = {} **using** *Suc.prems ain* **by** *auto*
then have *A′* = {*G a*} **using** *aeq Suc.prems ain aeq bb bij-betwE bij-betw-empty1 insert-Diff*
by (*metis Un-Int-eq*(*4*) *Un-commute* ‹*C* ∩ (*B* ∪ {*a*}) = {}› ‹*S* = *C* ∪ (*B* ∪ {*a*})›)
moreover have *F* (*G a*) = *space M* − (*F a*) **using** *Suc.prems ain* **by** *auto*
moreover have *indep-event X* (*F a*) **using** *mutual-indep-ev-singleton-event ies* **by** *auto*
ultimately show *?thesis* **using** *mutual-indep-ev-singleton indep-event-one-compl beq* **by** *auto*
next
case *False*
have *un*: *A′* ∪ *B* = *?C′* ∪ {*G a*} ∪ (*?B′* − {*a*}) **using** *Suc.prems aeq*
by (*metis Diff-insert-absorb Un-empty-right Un-insert-right ain bij-betwE disjoint-iff-not-equal insert-Diff*)
moreover have *?B′* − {*a*} ⊆ *?B′* **by** *auto*
moreover have *?B′* − {*a*} ⊆ *?C′* ∪ {*G a*} ∪ (*?B′*) **by** *auto*
moreover have *?C′* ∪ {*G a*} ⊆ *?C′* ∪ {*G a*} ∪ (*?B′*) **by** *auto*
ultimately have *ss*: *A′* ∪ *B* ⊆ {*G a*} ∪ (*?C′* ∪ *?B′*)
using *Un-least* **by** *auto*
have *a* ∈ *?C′* ∪ *?B′* **using** *aeq* **by** *auto*
then have *ie*: *mutual-indep-events X F* ({*G a*} ∪ (*?C′* ∪ *?B′*))
using *mutual-indep-ev-one-compl*[*of X F* (*?C′* ∪ *?B′*) *a G a*] **using** *Suc.prems*(*3*)
by (*metis* ‹*S* = *C* ∪ (*B* ∪ {*a*})› *ain assms*(*4*) *bb bij-betw-finite ies infinite-Un*)

then show *?thesis* **using** *mutual-indep-ev-subset ss* **by** *auto*
qed
qed

lemma *mutual-indep-ev-events-compl*:
assumes *finite S*
assumes *mutual-indep-events A F S*
assumes *bij-betw G S S′*
assumes ⋀ *i*. *i* ∈ *S* ⟹ *F* (*G i*) = *space M* − *F i*
shows *mutual-indep-events A F S′*
using *mutual-indep-events-update-compl*[*of A F S S* {}] *assms* **by** *auto*

Important lemma on relation between independence and mutual inde-

pendence of a set

lemma *mutual-indep-ev-set-all*:
assumes F ' $I \subseteq$ *events*
assumes $\bigwedge i.\ i \in I \Longrightarrow$ (*mutual-indep-events* ($F\ i$) F ($I - \{i\}$))
shows *indep-events* F I
proof (*intro indep-eventsI*)
show $\bigwedge i.\ i \in I \Longrightarrow F\ i \in$ *events*
using *assms*(*1*) **by** *auto*
next
fix J **assume** *ss*: $J \subseteq I$ **and** *fin*: *finite* J **and** *ne*: $J \neq \{\}$
from *fin ne ss* **show** *prob* ($\bigcap$ (F ' J)) = ($\prod i \in J.$ *prob* ($F\ i$))
proof (*induct* J *rule*: *finite-ne-induct*)
case (*singleton* x)
then show *?case* **by** *simp*
next
case (*insert* x X)
then have *mutual-indep-events* ($F\ x$) F ($I - \{x\}$) **using** *assms*(*2*) **by** *simp*
moreover have $X \subseteq (I - \{x\})$ **using** *insert.prems insert.hyps* **by** *auto*
ultimately have *prob* ($F\ x \cap (\bigcap (F$ '$X))$) = *prob* ($F\ x$) $*$ *prob* ($\bigcap (F$ ' $X)$)
by (*simp add*: *local.insert*(*2*) *mutual-indep-eventsD*)
then show *?case* **using** *insert.hyps insert.prems* **by** *simp*
qed
qed

end
end

6 The Basic Probabilistic Method Framework

This theory includes all aspects of step (3) and (4) of the basic method framework, which are purely probabilistic

theory *Basic-Method* **imports** *Indep-Events*
begin

6.1 More Set and Multiset lemmas

lemma *card-size-set-mset*: *card* (*set-mset* A) $\leq$ *size* A
using *size-multiset-overloaded-eq*
by (*metis card-eq-sum count-greater-eq-one-iff sum-mono*)

lemma *Union-exists*: $\{a \in A\ .\ \exists\ b \in B\ .\ P\ a\ b\} = (\bigcup b \in B\ .\ \{a \in A\ .\ P\ a\ b\})$
by *blast*

lemma *Inter-forall*: $B \neq \{\} \Longrightarrow \{a \in A\ .\ \forall\ b \in B\ .\ P\ a\ b\} = (\bigcap b \in B\ .\ \{a \in A\ .\ P\ a\ b\})$
by *auto*

lemma *function-map-multi-filter-size*:

assumes *image-mset F* (*mset-set A*) = *B* **and** *finite A*
shows *card* {*a* ∈ *A* . *P* (*F a*)} = *size* {# *b* ∈# *B* . *P b* #}
using *assms*(*2*) *assms*(*1*) **proof** (*induct A arbitrary*: *B rule*: *finite-induct*)
case *empty*
then show *?case* **by** *simp*
next
case (*insert x C*)
then have *beq*: *B*= *image-mset F* (*mset-set C*) + {#*F x*#} **by** *auto*
then show *?case* **proof** (*cases P* (*F x*))
case *True*
then have *filter-mset P B* = *filter-mset P* (*image-mset F* (*mset-set C*)) + {#*F x*#}
by (*simp add*: *True beq*)
then have *s*: *size* (*filter-mset P B*) = *size* (*filter-mset P* (*image-mset F* (*mset-set C*))) + *1*
using *size-single size-union* **by** *auto*
have {*a* ∈ *insert x C*. *P* (*F a*)} = *insert x* {*a* ∈ *C*. *P* (*F a*)} **using** *True* **by** *auto*
moreover have *x* ∉ {*a* ∈ *C*. *P* (*F a*)} **using** *insert.hyps*(*2*) **by** *simp*
ultimately have *card* {*a* ∈ *insert x C*. *P* (*F a*)} = *card* {*a* ∈ *C*. *P* (*F a*)} + *1*
using *card-insert-disjoint insert.hyps*(*1*) **by** *auto*
then show *?thesis* **using** *s insert.hyps*(*3*) **by** *simp*
next
case *False*
then have *filter-mset P B* = *filter-mset P* (*image-mset F* (*mset-set C*)) **using** *beq* **by** *simp*
moreover have {*a* ∈ *insert x C*. *P* (*F a*)} = {*a* ∈ *C*. *P* (*F a*)} **using** *False* **by** *auto*
ultimately show *?thesis* **using** *insert.hyps*(*3*) **by** *simp*
qed
qed

lemma *bij-mset-obtain-set-elem*:
assumes *image-mset F* (*mset-set A*) = *B*
assumes *b* ∈# *B*
obtains *a* **where** *a* ∈ *A* **and** *F a* = *b*
using *assms set-image-mset*
by (*metis finite-set-mset-mset-set image-iff mem-simps*(*2*) *mset-set.infinite set-mset-empty*)

lemma *bij-mset-obtain-mset-elem*:
assumes *finite A*
assumes *image-mset F* (*mset-set A*) = *B*
assumes *a* ∈ *A*
obtains *b* **where** *b* ∈# *B* **and** *F a* = *b*
using *assms* **by** *fastforce*

lemma *prod-fn-le1*:

fixes $f :: 'c \Rightarrow ('d :: \{comm\text{-}monoid\text{-}mult, linordered\text{-}semidom\})$
assumes *finite A*
assumes $A \neq \{\}$
assumes $\bigwedge y.\ y \in A \Longrightarrow f\ y \geq 0 \wedge f\ y < 1$
shows $(\prod x \in A.\ f\ x) < 1$
using *assms(1) assms(2) assms(3)* **proof** (*induct A rule*: *finite-ne-induct*)
case (*singleton x*)
then show *?case* **by** *auto*
next
case (*insert x F*)
then show *?case*
proof (*cases* $x \in F$)
case *True*
then show *?thesis* **using** *insert.hyps* **by** *auto*
next
case *False*
then have *prod f* (*insert x F*) = $f\ x * prod\ f\ F$ **by** (*simp add*: *local.insert(1)*)
moreover have *prod f F* < *1* **using** *insert.hyps insert.prems* **by** *auto*
moreover have $f\ x < 1\ f\ x \geq 0$ **using** *insert.prems* **by** *auto*
ultimately show *?thesis*
by (*metis basic-trans-rules(20) basic-trans-rules(23) more-arith-simps(6)*
mult-left-less-imp-less verit-comp-simplify1(3))
qed
qed

context *prob-space*
begin

6.2 Existence Lemmas

lemma *prob-lt-one-obtain*:
assumes $\{e \in space\ M\ .\ Q\ e\} \in events$
assumes *prob* $\{e \in space\ M\ .\ Q\ e\} < 1$
obtains *e* **where** $e \in space\ M$ **and** $\neg\ Q\ e$
proof −
have *sin*: $\{e \in space\ M\ .\ \neg\ Q\ e\} \in events$ **using** *assms(1)*
using *sets.sets-Collect-neg* **by** *blast*
have *prob* $\{e \in space\ M\ .\ \neg\ Q\ e\} = 1 - prob\ \{e \in space\ M\ .\ Q\ e\}$ **using** *prob-neg assms* **by** *auto*
then have *prob* $\{e \in space\ M\ .\ \neg\ Q\ e\} > 0$ **using** *assms(2)* **by** *auto*
then show *?thesis* **using** *that*
by (*smt* (*verit*, *best*) *empty-Collect-eq measure-empty*)
qed

lemma *prob-gt-zero-obtain*:
assumes $\{e \in space\ M\ .\ Q\ e\} \in events$
assumes *prob* $\{e \in space\ M\ .\ Q\ e\} > 0$
obtains *e* **where** $e \in space\ M$ **and** $Q\ e$
using *assms* **by** (*smt* (*verit*) *empty-Collect-eq inf.strict-order-iff measure-empty*)

lemma *inter-gt0-event*:
assumes F ‘ $I \subseteq$ *events*
assumes *prob* $(\bigcap\ i \in I\ .\ (space\ M - (F\ i))) > 0$
shows $(\bigcap\ i \in I\ .\ (space\ M - (F\ i))) \in$ *events* **and** $(\bigcap\ i \in I\ .\ (space\ M - (F\ i))) \neq \{\}$
using *assms* **using** *measure-notin-sets* **by** (*smt* (*verit*), *fastforce*)

lemma *obtain-intersection*:
assumes F ‘ $I \subseteq$ *events*
assumes *prob* $(\bigcap\ i \in I\ .\ (space\ M - (F\ i)))> 0$
obtains e **where** $e \in space\ M$ **and** $\bigwedge\ i.\ i \in I \Longrightarrow e \notin F\ i$
proof −
have *ine*: $(\bigcap\ i \in I\ .\ (space\ M - (F\ i))) \neq \{\}$ **using** *inter-gt0-event*[*of F I*] *assms* **by** *fast*
then obtain e **where** $\bigwedge\ i.\ i \in I \Longrightarrow\ e \in space\ M - F\ i$ **by** *blast*
then show *?thesis*
by (*metis Diff-iff ex-in-conv subprob-not-empty that*)
qed

lemma *obtain-intersection-prop*:
assumes F ‘ $I \subseteq$ *events*
assumes $\bigwedge\ i.\ i \in I \Longrightarrow F\ i = \{e \in space\ M\ .\ P\ e\ i\}$
assumes *prob* $(\bigcap\ i \in I\ .\ (space\ M - (F\ i)))> 0$
obtains e **where** $e \in space\ M$ **and** $\bigwedge\ i.\ i \in I \Longrightarrow \neg\ P\ e\ i$
proof −
obtain e **where** *ein*: $e \in space\ M$ **and** $\bigwedge\ i.\ i \in I \Longrightarrow e \notin F\ i$
using *obtain-intersection assms*(*1*) *assms*(*3*) **by** *auto*
then have $\bigwedge\ i.\ i \in I \Longrightarrow e \in \{e \in space\ M\ .\ \neg\ P\ e\ i\}$ **using** *assms*(*2*) **by** *simp*
then show *?thesis* **using** *ein that* **by** *simp*
qed

lemma *not-in-big-union*:
assumes $\bigwedge\ i\ .\ i \in A \Longrightarrow e \notin i$
shows $e \notin (\bigcup A)$
using *assms* **by** (*induct A rule*: *infinite-finite-induct*) *auto*

lemma *not-in-big-union-fn*:
assumes $\bigwedge\ i\ .\ i \in A \Longrightarrow e \notin F\ i$
shows $e \notin (\bigcup i \in A\ .\ F\ i)$
using *assms* **by** (*induct A rule*: *infinite-finite-induct*) *auto*

lemma *obtain-intersection-union*:
assumes F ‘ $I \subseteq$ *events*
assumes *prob* $(\bigcap\ i \in I\ .\ (space\ M - (F\ i)))> 0$
obtains e **where** $e \in space\ M$ **and** $e \notin (\bigcup i \in I.\ F\ i)$
proof −
obtain e **where** $e \in space\ M$ **and** *cond*: $\bigwedge\ i.\ i \in I \Longrightarrow\ e\ \notin\ F\ i$
using *obtain-intersection*[*of F I*] *assms* **by** *blast*

then show *?thesis* **using** *not-in-big-union-fn*[*of I e F*] *that* **by** *blast*
qed

6.3 Basic Bounds

Lemmas on the Complete Independence and Union bound

lemma *complete-indep-bound1*:
assumes *finite A*
assumes $A \neq \{\}$
assumes $A \subseteq$ *events*
assumes *indep-events-set A*
assumes $\bigwedge a . a \in A \Longrightarrow$ *prob a* < 1
shows *prob* (*space M* $-$ ($\bigcap A$)) > 0
proof $-$
have $\bigcap A \in$ *events* **using** *assms*(*1*) *assms*(*2*) *assms*(*3*) *Inter-event-ss* **by** *simp*
then have *prob* (*space M* $-$ ($\bigcap A$)) $= 1 -$ *prob* ($\bigcap A$)
by (*simp add*: *prob-compl*)
then have *1*: *prob* (*space M* $-$ ($\bigcap A$)) $= 1 -$ *prod prob A*
using *indep-events-set-prod-all assms* **by** *simp*
moreover have *prod prob A* < 1 **using** *assms*(*5*) *assms*(*1*) *assms*(*2*) *assms*(*4*) *indep-events-set-events*
by (*metis Inf-lower* ‹*prob* (*space M* $-$ $\bigcap$ *A*) $= 1 -$ *prob* ($\bigcap$ *A*)›
basic-trans-rules(*21*) *1 diff-gt-0-iff-gt finite-has-maximal finite-measure-mono*
)
ultimately show *?thesis* **by** *simp*
qed

lemma *complete-indep-bound1-index*:
assumes *finite A*
assumes $A \neq \{\}$
assumes *F* ‘ *A* $\subseteq$ *events*
assumes *indep-events F A*
assumes $\bigwedge a . a \in A \Longrightarrow$ *prob* (*F a*) < 1
shows *prob* (*space M* $-$ ($\bigcap$(*F*‘ *A*))) > 0
proof $-$
have *pos*: $\bigwedge a. a \in A \Longrightarrow$ *prob* (*F a*) ≥ 0 **using** *assms*(*3*) **by** *auto*
have $\bigcap$(*F* ‘ *A*) $\in$ *events* **using** *assms*(*1*) *assms*(*2*) *assms*(*3*) *Inter-event-ss* **by** *simp*
then have *eq*: *prob* (*space M* $-$ ($\bigcap$(*F* ‘ *A*))) $= 1 -$ *prob* ($\bigcap$(*F* ‘ *A*))
by (*simp add*: *prob-compl*)
then have *prob* (*space M* $-$ ($\bigcap$(*F* ‘ *A*))) $= 1 -$ ($\prod i \in A.$ *prob* (*F i*))
using *indep-events-prod-all assms* **by** *simp*
moreover have ($\prod i \in A.$ *prob* (*F i*)) < 1
using *assms*(*5*) *eq assms*(*2*) *assms*(*1*) *prod-fn-le1*[*of A* λ *i. prob* (*F i*)] **by** *auto*

ultimately show *?thesis* **by** *simp*
qed

lemma *complete-indep-bound2*:

assumes *finite A*
assumes $A \subseteq$ *events*
assumes *indep-events-set A*
assumes $\bigwedge a . a \in A \Longrightarrow$ *prob a* < 1
shows *prob* (*space M* $-$ ($\bigcup A$)) > 0
proof (*cases A* $= \{\}$)
case *True*
then show *?thesis* **by** (*simp add*: *True prob-space*)
next
case *False*
then have *prob* (*space M* $-$ $\bigcup A$) $=$ *prob* ($\bigcap a \in A . (space\ M - a)$) **by** *simp*
moreover have *indep-events-set* ((λ *a*. *space M* $-$ *a*) ‘ *A*)
using *assms*(*1*) *assms*(*3*) *indep-events-set-compl* **by** *auto*
moreover have *finite* ((λ *a*. *space M* $-$ *a*) ‘ *A*) **using** *assms*(*1*) **by** *auto*
moreover have ((λ *a*. *space M* $-$ *a*) ‘ *A*) $\neq \{\}$ **using** *False* **by** *auto*
ultimately have *eq*: *prob* (*space M* $-$ $\bigcup A$) $=$ *prod prob* ((λ *a*. *space M* $-$ *a*) ‘ *A*)
using *indep-events-set-prod-all*[*of* ((λ *a*. *space M* $-$ *a*) ‘ *A*)] **by** *linarith*
have $\bigwedge a.\ a \in$ ((λ *a*. *space M* $-$ *a*) ‘ *A*) $\Longrightarrow$ *prob a* > 0
proof $-$
fix *a* **assume** *a* $\in$ ((λ *a*. *space M* $-$ *a*) ‘ *A*)
then obtain *a*′ **where** *a* $=$ *space M* $-$ *a*′ **and** *ain*: *a*′ $\in$ *A* **by** *blast*
then have *prob a* $= 1 -$ *prob a*′ **using** *prob-compl assms*(*2*) **by** *auto*
moreover have *prob a*′ < 1 **using** *assms*(*4*) *ain* **by** *simp*
ultimately show *prob a* > 0 **by** *simp*
qed
then have *prod prob* ((λ *a*. *space M* $-$ *a*) ‘ *A*) > 0 **by** (*meson prod-pos*)
then show *?thesis* **using** *eq* **by** *simp*
qed

lemma *complete-indep-bound2-index*:
assumes *finite A*
assumes *F* ‘ *A* $\subseteq$ *events*
assumes *indep-events F A*
assumes $\bigwedge a . a \in A \Longrightarrow$ *prob* (*F a*) < 1
shows *prob* (*space M* $-$ ($\bigcup$ (*F* ‘ *A*))) > 0
proof (*cases A* $= \{\}$)
case *True*
then show *?thesis* **by** (*simp add*: *True prob-space*)
next
case *False*
then have *prob* (*space M* $-$ $\bigcup$ (*F* ‘*A*)) $=$ *prob* ($\bigcap a \in A . (space\ M - F\ a)$) **by** *simp*
moreover have *indep-events* (λ *a*. *space M* $-$ *F a*) *A*
using *assms*(*1*) *assms*(*3*) *indep-events-compl* **by** *auto*
ultimately have *eq*: *prob* (*space M* $-$ $\bigcup$ (*F* ‘ *A*)) $=$ ($\prod i \in A$. *prob* ((λ *a*. *space M* $-$ *F a*) *i*))
using *indep-events-prod-all*[*of* (λ *a*. *space M* $-$ *F a*) *A*] *assms*(*1*) *False* **by** *linarith*

have $\bigwedge a.\ a \in A \Longrightarrow prob\ (space\ M - F\ a) > 0$
using *prob-compl assms(2) assms(4)* **by** *auto*
then have $(\prod i \in A.\ prob\ ((\lambda\ a.\ space\ M - F\ a)\ i)) > 0$ **by** (*meson prod-pos*)
then show *?thesis* **using** *eq* **by** *simp*
qed

lemma *complete-indep-bound3*:
assumes *finite A*
assumes $A \neq \{\}$
assumes $F\ `\ A \subseteq events$
assumes *indep-events F A*
assumes $\bigwedge a\ .\ a \in A \Longrightarrow prob\ (F\ a) < 1$
shows $prob\ (\bigcap a \in A.\ space\ M - F\ a) > 0$
using *complete-indep-bound2-index compl-Union-fn assms* **by** *auto*

Combining complete independence with existence step

lemma *complete-indep-bound-obtain*:
assumes *finite A*
assumes $A \subseteq events$
assumes *indep-events-set A*
assumes $\bigwedge a\ .\ a \in A \Longrightarrow prob\ a < 1$
obtains *e* **where** $e \in space\ M$ **and** $e \notin \bigcup A$
proof −
have $prob\ (space\ M - (\bigcup A)) > 0$ **using** *complete-indep-bound2 assms* **by** *auto*
then show *?thesis*
by (*metis Diff-eq-empty-iff less-numeral-extra(3) measure-empty subsetI that*)
qed

lemma *Union-bound-events*:
assumes *finite A*
assumes $A \subseteq events$
shows $prob\ (\bigcup A) \leq (\sum a \in A.\ prob\ a)$
using *finite-measure-subadditive-finite*[*of A* $\lambda\ x.\ x$] *assms* **by** *auto*

lemma *Union-bound-events-fun*:
assumes *finite A*
assumes $f\ `\ A \subseteq events$
shows $prob\ (\bigcup (f\ `\ A)) \leq (\sum\ a \in A.\ prob\ (f\ a))$
by (*simp add*: *assms(1) assms(2) finite-measure-subadditive-finite*)

lemma *Union-bound-avoid*:
assumes *finite A*
assumes $(\sum a \in A.\ prob\ a) < 1$
assumes $A \subseteq events$
shows $prob\ (space\ M - \bigcup A) > 0$
proof −
have $\bigcup A \in events$
by (*simp add*: *assms(1) assms(3) sets.finite-Union*)

then have *prob* (*space M* − ⋃ *A*) = *1* − *prob* (⋃ *A*)
using *prob-compl* **by** *simp*
moreover have *prob* (⋃ *A*) < *1* **using** *assms Union-bound-events*
by *fastforce*
ultimately show *?thesis* **by** *simp*
qed

lemma *Union-bound-avoid-fun*:
assumes *finite A*
assumes (∑ *a* ∈ *A*. *prob* (*f a*)) < *1*
assumes *f*‘*A* ⊆ *events*
shows *prob* (*space M* − ⋃ (*f* ‘ *A*)) > *0*
proof −
have ⋃ (*f* ‘ *A*) ∈ *events*
by (*simp add*: *assms*(*1*) *assms*(*3*) *sets.finite-Union*)
then have *prob* (*space M* − ⋃ (*f* ‘ *A*)) = *1* − *prob* (⋃ (*f* ‘ *A*))
using *prob-compl* **by** *simp*
moreover have *prob* (⋃ (*f* ‘ *A*)) < *1* **using** *assms Union-bound-events-fun*
by (*smt* (*verit*, *ccfv-SIG*) *sum.cong*)
ultimately show *?thesis* **by** *simp*
qed

Combining union bound with existance step

lemma *Union-bound-obtain*:
assumes *finite A*
assumes (∑ *a* ∈ *A*. *prob a*) < *1*
assumes *A* ⊆ *events*
obtains *e* **where** *e* ∈ *space M* **and** *e* ∉ ⋃ *A*
proof −
have *prob* (*space M* − ⋃ *A*) > *0* **using** *Union-bound-avoid assms* **by** *simp*
then show *?thesis* **using** *that prob-gt-zero-obtain*
by (*metis Diff-eq-empty-iff less-numeral-extra*(*3*) *measure-empty subsetI*)
qed

lemma *Union-bound-obtain-fun*:
assumes *finite A*
assumes (∑ *a* ∈ *A*. *prob* (*f a*)) < *1*
assumes *f* ‘ *A* ⊆ *events*
obtains *e* **where** *e* ∈ *space M* **and** *e* ∉ ⋃ (*f*‘ *A*)
proof −
have *prob* (*space M* − ⋃ (*f*‘ *A*)) > *0* **using** *Union-bound-avoid-fun assms* **by** *simp*
then show *?thesis* **using** *that prob-gt-zero-obtain*
by (*metis Diff-eq-empty-iff less-numeral-extra*(*3*) *measure-empty subsetI*)
qed

lemma *Union-bound-obtain-compl*:
assumes *finite A*
assumes (∑ *a* ∈ *A*. *prob a*) < *1*

assumes $A \subseteq$ *events*
obtains *e* **where** $e \in (space\ M - \bigcup A)$
proof −
have *prob* $(space\ M - \bigcup A) > 0$ **using** *Union-bound-avoid assms* **by** *simp*
then show *?thesis* **using** *that prob-gt-zero-obtain*
by (*metis all-not-in-conv measure-empty verit-comp-simplify*(*2*) *verit-comp-simplify1*(*3*))
qed

lemma *Union-bound-obtain-compl-fun*:
assumes *finite A*
assumes $(\sum a \in A.\ prob\ (f\ a)) < 1$
assumes $f\ `\ A \subseteq$ *events*
obtains *e* **where** $e \in (space\ M - \bigcup(\ f`\ A))$
proof −
obtain *e* **where** $e \in space\ M$ **and** $e \notin \bigcup(f`\ A)$
using *assms Union-bound-obtain-fun* **by** *blast*
then have $e \in space\ M - \bigcup\ (f\ `\ A)$ **by** *simp*
then show *?thesis* **by** *fact*
qed

end

end

7 Lovasz Local Lemma

theory *Lovasz-Local-Lemma*
imports
Basic-Method
HOL−Real-Asymp.Real-Asymp
Indep-Events
Digraph-Extensions
begin

7.1 Random Lemmas on Product Operator

lemma *prod-constant-ge*:
fixes $y :: 'b :: \{$*comm-monoid-mult*, *linordered-semidom*$\}$
assumes *card* $A \leq k$
assumes $y \geq 0$ **and** $y < 1$
shows $(\prod x{\in}A.\ y) \geq y\ \hat{}\ k$
using *assms power-decreasing* **by** *fastforce*

lemma (**in** *linordered-idom*) *prod-mono3*:
assumes *finite J I* $\subseteq J$ $\bigwedge i.\ i \in J \Longrightarrow 0 \leq f\ i\ \ (\bigwedge i.\ i \in J \Longrightarrow f\ i \leq 1)$
shows *prod f J* $\leq$ *prod f I*
proof −
have *prod f J* $\leq (\prod i{\in}J.$ *if i* $\in$ *I then f i else 1*$)$
using *assms* **by** (*intro prod-mono*) *auto*

also have ... = *prod f I*
using ‹*finite J*› ‹*I* ⊆ *J*› **by** (*simp add*: *prod.If-cases Int-absorb1*)
finally show *?thesis* **.**
qed

lemma *bij-on-ss-image*:
assumes *A* ⊆ *B*
assumes *bij-betw g B B′*
shows *g ‘ A* ⊆ *B′*
using *assms* **by** (*auto simp add*: *bij-betw-apply subsetD*)

lemma *bij-on-ss-proper-image*:
assumes *A* ⊂ *B*
assumes *bij-betw g B B′*
shows *g ‘ A* ⊂ *B′*
by (*smt* (*verit*, *ccfv-SIG*) *assms bij-betw-iff-bijections bij-betw-subset leD psubsetD psubsetI subsetI*)

7.2 Dependency Graph Concept

Uses directed graphs. The pair_digraph locale was sufficient as multi-edges are irrelevant

locale *dependency-digraph* = *pair-digraph G* :: *nat pair-pre-digraph* + *prob-space M* :: *′a measure*
for *G M* + **fixes** *F* :: *nat* ⇒ *′a set*
assumes *vss*: *F ‘* (*pverts G*) ⊆ *events*
assumes *mis*: ⋀ *i*. *i* ∈ (*pverts G*) ⟹ *mutual-indep-events* (*F i*) *F* ((*pverts G*) − ({*i*} ∪ *neighborhood i*))
begin

lemma *dep-graph-indiv-nh-indep*:
assumes *A* ∈ *pverts G B* ∈ *pverts G*
assumes *B* ∉ *neighborhood A*
assumes *A* ≠ *B*
assumes *prob* (*F B*) ≠ *0*
shows *P*((*F A*) | (*F B*)) = *prob* (*F A*)
proof−
have *B* ∉ {*A*} ∪ *neighborhood A* **using** *assms*(*3*) *assms*(*4*) **by** *auto*
then have *B* ∈ (*pverts G* − ({*A*} ∪ *neighborhood A*)) **using** *assms*(*2*) **by** *auto*
moreover have *mutual-indep-events* (*F A*) *F* (*pverts G* − ({*A*} ∪ *neighborhood A*)) **using** *mis assms* **by** *auto*
ultimately show *?thesis* **using**
assms(*5*) *assms*(*1*) *assms*(*2*) *vss mutual-indep-ev-cond-single* **by** *auto*
qed

lemma *mis-subset*:
assumes *i* ∈ *pverts G*
assumes *A* ⊆ *pverts G*
shows *mutual-indep-events* (*F i*) *F* (*A* − ({*i*} ∪ *neighborhood i*))

proof (*cases* $A \subseteq (\{i\} \cup \textit{neighborhood}\ i)$)
case *True*
then have $A - (\{i\} \cup \textit{neighborhood}\ i) = \{\}$ **by** *auto*
then show *?thesis* **using** *mutual-indep-ev-empty vss assms(1)* **by** *blast*
next
case *False*
then have $A - (\{i\} \cup \textit{neighborhood}\ i) \subseteq \textit{pverts}\ G - (\{i\} \cup \textit{neighborhood}\ i)$
using *assms(2)* **by** *auto*
then show *?thesis* **using** *mutual-indep-ev-subset mis assms(1)* **by** *blast*
qed

lemma *dep-graph-indep-events*:
assumes $A \subseteq \textit{pverts}\ G$
assumes $\bigwedge Ai.\ Ai \in A \Longrightarrow \textit{out-degree}\ G\ Ai = 0$
shows *indep-events F A*
proof −
have $\bigwedge Ai.\ Ai \in A \Longrightarrow (\textit{mutual-indep-events}\ (F\ Ai)\ F\ (A - \{Ai\}))$
proof −
fix *Ai* **assume** *ain*: $Ai \in A$
then have $(\textit{neighborhood}\ Ai) = \{\}$ **using** *assms(2) neighborhood-empty-iff* **by** *simp*
moreover have *mutual-indep-events* $(F\ Ai)\ F\ (A - (\{Ai\} \cup \textit{neighborhood}\ Ai))$
using *mis-subset*[*of Ai A*] *ain assms(1)* **by** *auto*
ultimately show *mutual-indep-events* $(F\ Ai)\ F\ (A - \{Ai\})$ **by** *simp*
qed
then show *?thesis* **using** *mutual-indep-ev-set-all*[*of F A*] *vss* **by** *auto*
qed

end

7.3 Lovasz Local General Lemma

context *prob-space*
begin

lemma *compl-sets-index*:
assumes $F\ `\ A \subseteq \textit{events}$
shows $(\lambda\ i.\ \textit{space}\ M - F\ i)\ `\ A \subseteq \textit{events}$
proof (*intro subsetI*)
fix x **assume** $x \in (\lambda i.\ \textit{space}\ M - F\ i)\ `\ A$
then obtain i **where** *xeq*: $x = \textit{space}\ M - F\ i$ **and** $i \in A$ **by** *blast*
then have $F\ i \in \textit{events}$ **using** *assms* **by** *auto*
thus $x \in \textit{events}$ **using** *sets.compl-sets xeq* **by** *simp*
qed

lemma *lovasz-inductive-base*:
assumes *dependency-digraph G M F*
assumes $\bigwedge Ai\ .\ Ai \in A \Longrightarrow g\ Ai \geq 0 \wedge g\ Ai < 1$

assumes $\bigwedge Ai.\ Ai \in A \Longrightarrow (prob\ (F\ Ai) \leq (g\ Ai) * (\prod\ Aj \in pre\text{-}digraph.neighborhood\ G\ Ai.\ (1 - (g\ Aj))))$
assumes $Ai \in A$
assumes $pverts\ G = A$
shows $prob\ (F\ Ai) \leq g\ Ai$
proof −
have *genprod*: $\bigwedge S.\ S \subseteq A \Longrightarrow (\prod Aj \in S\ .\ (1 - (g\ Aj))) \leq 1$ **using** *assms*(2)
by (*smt* (*verit*) *prod-le-1 subsetD*)
interpret *dg*: *dependency-digraph G M F* **using** *assms*(1) **by** *simp*
have *dg.neighborhood Ai* $\subseteq A$ **using** *assms*(3) *dg.neighborhood-wf assms*(5) **by** *simp*
then show *?thesis*
using *genprod assms mult-left-le* **by** (*smt* (*verit*))
qed

lemma *lovasz-inductive-base-set*:
assumes $N \subseteq A$
assumes $\bigwedge Ai\ .\ Ai \in A \Longrightarrow g\ Ai \geq 0 \wedge g\ Ai < 1$
assumes $\bigwedge Ai.\ Ai \in A \Longrightarrow (prob\ (F\ Ai) \leq (g\ Ai) * (\prod\ Aj \in N.\ (1 - (g\ Aj))))$
assumes $Ai \in A$
shows $prob\ (F\ Ai) \leq g\ Ai$
proof −
have *genprod*: $\bigwedge S.\ S \subseteq A \Longrightarrow (\prod Aj \in S\ .\ (1 - (g\ Aj))) \leq 1$ **using** *assms*(2)
by (*smt* (*verit*) *prod-le-1 subsetD*)
then show *?thesis*
using *genprod assms mult-left-le* **by** (*smt* (*verit*))
qed

lemma *split-prob-lt-helper*:
assumes *dep-graph*: *dependency-digraph G M F*
assumes *dep-graph-verts*: $pverts\ G = A$
assumes *fbounds*: $\bigwedge i\ .\ i \in A \Longrightarrow f\ i \geq 0 \wedge f\ i < 1$
assumes *prob-Ai*: $\bigwedge Ai.\ Ai \in A \Longrightarrow prob\ (F\ Ai) \leq$
$(f\ Ai) * (\prod\ Aj \in pre\text{-}digraph.neighborhood\ G\ Ai\ .\ (1 - (f\ Aj)))$
assumes *aiin*: $Ai \in A$
assumes $N \subseteq pre\text{-}digraph.neighborhood\ G\ Ai$
assumes $\exists\ P1\ P2.\ \mathcal{P}(F\ Ai \mid \bigcap Aj \in S.\ space\ M - F\ Aj) = P1/P2 \wedge$
$P1 \leq prob\ (F\ Ai) \wedge\ P2 \geq (\prod\ Aj \in N\ .\ (1 - (f\ Aj)))$
shows $\mathcal{P}(F\ Ai \mid \bigcap Aj \in S.\ space\ M - F\ Aj) \leq f\ Ai$
proof −
interpret *dg*: *dependency-digraph G M F* **using** *assms*(1) **by** *simp*
have *lt1*: $\bigwedge Aj.\ Aj \in A \Longrightarrow\ (1 - (f\ Aj)) \leq 1$
using *assms*(3) **by** *auto*
have *gt0*: $\bigwedge Aj.\ Aj \in A \Longrightarrow (1 - (f\ Aj)) > 0$ **using** *assms*(3) **by** *auto*
then have *prodgt0*: $\bigwedge S'.\ S' \subseteq A \Longrightarrow\ (\prod\ Aj \in S'\ .\ (1 - f\ Aj)) > 0$
using *prod-pos* **by** (*metis subsetD*)
obtain *P1 P2* **where** *peq*: $\mathcal{P}(F\ Ai \mid \bigcap Aj \in S.\ space\ M - F\ Aj) = P1/P2$ **and**
$P1 \leq prob\ (F\ Ai)$
and *p2gt*: $P2 \geq (\prod\ Aj \in N\ .\ (1 - (f\ Aj)))$ **using** *assms*(7) **by** *auto*

then have $P1 \leq (f\ Ai) * (\prod\ Aj \in pre\text{-}digraph.neighborhood\ G\ Ai\ .\ (1 - (f\ Aj)))$

using *prob-Ai aiin* **by** *fastforce*
moreover have $P2 \geq (\prod\ Aj \in dg.neighborhood\ Ai\ .\ (1 - (f\ Aj)))$ **using** *assms*(*6*)
gt0 dg.neighborhood-wf dep-graph-verts subset-iff lt1 dg.neighborhood-finite p2gt
by (*smt* (*verit*, *ccfv-threshold*) *prod-mono3*)
ultimately have $P1/P2 \leq ((f\ Ai) * (\prod\ Aj \in dg.neighborhood\ Ai\ .\ (1 - (f\ Aj))))/(\prod\ Aj \in dg.neighborhood\ Ai\ .\ (1 - (f\ Aj))))$
using *frac-le*[*of* $(f\ Ai) * (\prod\ Aj \in dg.neighborhood\ Ai\ .\ (1 - (f\ Aj)))$ *P1* $(\prod\ Aj \in dg.neighborhood\ Ai\ .\ (1 - (f\ Aj)))$]
prodgt0[*of dg.neighborhood Ai*] *assms*(*3*) *dg.neighborhood-wf*[*of Ai*]
by (*simp add*: *assms*(*2*) *bounded-measure finite-measure-compl assms*(*5*))
then show *?thesis* **using** *prodgt0*[*of dg.neighborhood Ai*] *dg.neighborhood-wf*[*of Ai*] *assms*(*2*) *peq*
by (*metis divide-eq-imp rel-simps*(*70*))
qed

lemma *lovasz-inequality*:
assumes *finS*: *finite S*
assumes *sevents*: $F\ `\ S \subseteq events$
assumes *S-subset*: $S \subseteq A - \{Ai\}$
assumes *prob2*: $prob\ (\bigcap\ Aj \in S\ .\ (space\ M - (F\ Aj))) > 0$
assumes *irange*: $i \in \{0..<card\ S1\}$
assumes *bb*: *bij-betw g* $\{0..<card\ S1\}$ *S1*
assumes *s1-def*: $S1 = (S \cap N)$
assumes *s2-def*: $S2 = S - S1$
assumes *ne-cond*: $i > 0 \vee S2 \neq \{\}$
assumes *hyps*: $\bigwedge B.\ B \subset S \Longrightarrow g\ i \in A \Longrightarrow B \subseteq A - \{g\ i\} \Longrightarrow B \neq \{\} \Longrightarrow$
$0 < prob\ (\bigcap Aj{\in}B.\ space\ M - F\ Aj) \Longrightarrow \mathcal{P}(F\ (g\ i) \mid \bigcap Aj{\in}B.\ space\ M - F\ Aj) \leq f\ (g\ i)$
shows $\mathcal{P}((space\ M - F\ (g\ i)) \mid (\bigcap\ ((\lambda\ i.\ space\ M - F\ i)\ `\ g\ `\ \{0..<i\} \cup ((\lambda\ i.\ space\ M - F\ i)\ `\ S2))))$
$\geq (1 - f\ (g\ i))$
proof −
let *?c* $= (\lambda\ i.\ space\ M - F\ i)$
define *S1ss* **where** $S1ss = g\ `\ \{0..<i\}$
have $i \notin \{0..<i\}$ **by** *simp*
moreover have $\{0..<i\} \subseteq \{0..<card\ S1\}$ **using** *irange* **by** *simp*
ultimately have *ginotin1*: $g\ i \notin S1ss$ **using** *bb S1ss-def irange*
by (*smt* (*verit*, *best*) *bij-betw-iff-bijections image-iff subset-eq*)
have *ginotin2*: $g\ i \notin S2$ **unfolding** *s2-def* **using** *irange bb* **by** (*simp add*: *bij-betwE*)
have *giS*: $g\ i \in S$ **using** *irange bij-betw-imp-surj-on imageI Int-iff s1-def bb*
by *blast*
have $\{0..<i\} \subset \{0..<card\ S1\}$ **using** *irange* **by** *auto*
then have $S1ss \subset S1$ **unfolding** *S1ss-def* **using** *irange bb bij-on-ss-proper-image* **by** *meson*
then have *sss*: $S1ss \cup S2 \subset S$ **using** *s1-def s2-def* **by** *blast*

moreover have *xsiin*: $g\ i \in A$**using** *irange*
using *giS S-subset* **by** (*metis DiffE in-mono*)
moreover have *ne*: $S1ss \cup S2 \neq \{\}$ **using** *ne-cond S1ss-def* **by** *auto*
moreover have $S1ss \cup S2 \subseteq A - \{g\ i\}$ **using** *S-subset sss ginotin1 ginotin2* **by** *auto*
moreover have *gt02*: $0 < prob\ (\bigcap\ (?c\ `\ (S1ss \cup S2)))$ **using** *finS prob2 sevents prob-inter-ss-lt-index*[*of S ?c S1ss* $\cup$ *S2*] *ne sss compl-sets-index*[*of F S*] **by** *fastforce*
ultimately have *ltfAi*: $\mathcal{P}(F\ (g\ i) \mid \bigcap\ (?c\ `\ (S1ss \cup S2))) \leq f\ (g\ i)$
using *hyps*[*of S1ss* $\cup$ *S2*] **by** *blast*
have $?c\ `\ (S1ss \cup S2) \subseteq events$ **using** *sss* ‹*S1ss* $\subset$ *S1*› *compl-subset-in-events sevents s1-def s2-def*
by *fastforce*
then have $\bigcap\ (?c\ `\ (S1ss \cup S2)) \in events$ **using** *Inter-event-ss sss*
by (*meson* ‹*S1ss* $\cup$ *S2* $\neq \{\}$› *finite-imageI finite-subset image-is-empty finS subset-iff-psubset-eq*)
moreover have $F\ (g\ i) \in events$ **using** *xsiin giS sevents* **by** *auto*
ultimately have $\mathcal{P}(?c\ (g\ i) \mid \bigcap\ (?c\ `\ (S1ss \cup S2))) \geq 1 - f\ (g\ i)$
using *cond-prob-neg*[*of* $\bigcap\ (?c\ `\ (S1ss \cup S2))\ F\ (g\ i)$] *gt02 xsiin ltfAi* **by** *simp*
then show $\mathcal{P}(?c\ (g\ i) \mid (\bigcap\ (?c\ `\ g\ `\ \{0..<i\} \cup (?c\ `\ S2)))) \geq (1 - f\ (g\ i))$
by (*simp add*: *S1ss-def image-Un*)
qed

The main helper lemma

lemma *lovasz-inductive*:
assumes *finA*: *finite A*
assumes *Aevents*: $F\ `\ A \subseteq events$
assumes *fbounds*: $\bigwedge\ i\ .\ i \in A \Longrightarrow f\ i \geq 0 \wedge f\ i < 1$
assumes *dep-graph*: *dependency-digraph G M F*
assumes *dep-graph-verts*: *pverts G* = *A*
assumes *prob-Ai*: $\bigwedge\ Ai.\ \ Ai \in A \Longrightarrow prob\ (F\ Ai) \leq$
$(f\ Ai) * (\prod\ Aj \in$ *pre-digraph.neighborhood G Ai* $.\ (1 - (f\ Aj)))$
assumes *Ai-in*: $Ai \in A$
assumes *S-subset*: $S \subseteq A - \{Ai\}$
assumes *S-nempty*: $S \neq \{\}$
assumes *prob2*: $prob\ (\bigcap\ Aj \in S\ .\ (space\ M - (F\ Aj))) > 0$
shows $\mathcal{P}((F\ Ai) \mid (\bigcap\ Aj \in S\ .\ (space\ M - (F\ Aj)))) \leq f\ Ai$
proof −
let $?c = \lambda\ i.\ space\ M - F\ i$
have *ceq*: $\bigwedge\ A.\ ?c\ `\ A = ((-)\ (space\ M))\ `\ (F\ `\ A)$ **by** *auto*
interpret *dg*: *dependency-digraph G M F* **using** *assms*(*4*) **by** *simp*
have *finS*: *finite S* **using** *assms finite-subset* **by** (*metis finite-Diff*)
show $\mathcal{P}((\ F\ Ai) \mid (\bigcap\ Aj \in S\ .\ (space\ M - (F\ Aj)))) \leq f\ Ai$
using *finS Ai-in S-subset S-nempty prob2*
proof (*induct S arbitrary*: *Ai rule*: *finite-psubset-induct*)
case (*psubset S*)
define *S1* **where** $S1 = (S \cap dg.neighborhood\ Ai)$
define *S2* **where** $S2 = S - S1$
have $\bigwedge\ s\ .\ s \in S2 \Longrightarrow s \in A - (\{Ai\} \cup dg.neighborhood\ Ai)$

using *S1-def S2-def psubset.prems(2)* **by** *blast*
then have *s2ssmis*: $S2 \subseteq A - (\{Ai\} \cup dg.neighborhood\ Ai)$ **by** *auto*
have *sevents*: $F\ `\ S \subseteq events$ **using** *assms(2) psubset.prems(2)* **by** *auto*
then have *s1events*: $F\ `\ S1 \subseteq events$ **using** *S1-def* **by** *auto*
have *finS2*: *finite S2* **and** *finS1*: *finite S1* **using** *S2-def S1-def* **by** (*simp-all add*: *psubset(1)*)
have *mutual-indep-set* $(F\ Ai)$ $(F\ `\ S2)$ **using** *dg.mis*[*of Ai*] *mutual-indep-ev-subset s2ssmis*
psubset.prems(1) dep-graph-verts mutual-indep-iff **by** *auto*
then have *mis2*: *mutual-indep-set* $(F\ Ai)$ $(?c\ `\ S2)$
using *mutual-indep-events-compl*[*of* $F\ `\ S2\ F\ Ai$] *finS2 ceq*[*of S2*] **by** *simp*
have *scompl-ev*: $?c\ `\ S \subseteq events$
using *compl-sets-index sevents* **by** *simp*
then have *s2cev*: $?c\ `\ S2 \subseteq events$ **using** *S2-def scompl-ev* **by** *blast*
have $(\bigcap\ Aj \in S\ .\ space\ M - (F\ Aj)) \subseteq (\bigcap\ Aj \in S2\ .\ space\ M - (F\ Aj))$
unfolding *S2-def* **using** *Diff-subset image-mono Inter-anti-mono* **by** *blast*
then have $S2 \neq \{\} \Longrightarrow prob\ (\bigcap\ Aj \in S2\ .\ space\ M - (F\ Aj)) \neq 0$ **using** *psubset.prems(4) s2cev*
finS2 Inter-event-ss[*of* $?c\ `\ S2$] *finite-measure-mono*[*of* $\bigcap\ (?c\ `\ S) \bigcap (?c\ `\ S2)$] **by** *simp*
then have *s2prob-eq*: $S2 \neq \{\} \Longrightarrow \mathcal{P}((F\ Ai) \mid (\bigcap\ (?c\ `\ S2))) = prob\ (F\ Ai)$ **using** *assms(2)*
mutual-indep-cond-full[*of* $F\ Ai\ ?c\ `\ S2$] *psubset.prems(1) s2cev finS2 mis2* **by** *simp*
show *?case*
proof (*cases S1* $= \{\}$)
case *True*
then show *?thesis* **using** *lovasz-inductive-base*[*of G F A f Ai*] *psubset.prems(3) S2-def*
assms(3) assms(4) psubset.prems(1) prob-Ai s2prob-eq dep-graph-verts **by** (*simp*)
next
case *s1F*: *False*
then have *csgt0*: *card S1* > 0 **using** *s1F finS1 card-gt-0-iff* **by** *blast*
obtain *g* **where** *bb*: *bij-betw g* $\{0..<card\ S1\}$ *S1* **using** *finS1 ex-bij-betw-nat-finite* **by** *auto*
have *igt0*: $\bigwedge i.\ i \in \{0..<card\ S1\} \Longrightarrow 1 - f\ (g\ i) \geq 0$
using *S1-def psubset.prems(2) bb bij-betw-apply assms(3)* **by** *fastforce*
have *s1ss*: $S1 \subseteq dg.neighborhood\ Ai$ **using** *S1-def* **by** *auto*
moreover have $\exists\ P1\ P2.\ \mathcal{P}(F\ Ai \mid \bigcap Aj \in S.\ space\ M - F\ Aj) = P1/P2 \wedge P1 \leq prob\ (F\ Ai)$
$\wedge\ P2 \geq (\prod\ Aj \in S1\ .\ (1 - (f\ Aj)))$
proof (*cases S2* $= \{\}$)
case *True*
then have *Seq*: $S1 = S$ **using** *S1-def S2-def* **by** *auto*
have *inter-eventsS*: $(\bigcap\ Aj \in S\ .\ (space\ M - (F\ Aj))) \in events$ **using** *psubset.prems assms*
by (*meson measure-notin-sets zero-less-measure-iff*)
then have *peq*: $\mathcal{P}((F\ Ai) \mid (\bigcap\ Aj \in S1\ .\ ?c\ Aj)) =$

prob (($\bigcap$ *Aj* ∈ *S1* . *?c Aj*) ∩ (*F Ai*))/*prob* (($\bigcap$ (*?c* ‘ *S1*)))
(**is** $\mathcal{P}$((*F Ai*) | ($\bigcap$ *Aj* ∈ *S1* . *?c Aj*)) = *?Num*/*?Den*)
using *cond-prob-ev-def*[*of* ($\bigcap$ *Aj* ∈ *S1* . (*space M* − (*F Aj*))) *F Ai*]
using *Seq psubset.prems*(*1*) *assms*(*2*) **by** *blast*
have *?Num* ≤ *prob* (*F Ai*) **using** *finite-measure-mono assms*(*2*) *psubset.prems*(*1*) **by** *simp*
moreover have *?Den* ≥ ($\prod$ *Aj* ∈ *S1* . (*1* − (*f Aj*)))
proof −
have *pcond*: *prob* ($\bigcap$(*?c* ‘ *S1*)) =
prob (*?c* (*g 0*)) ∗ ($\prod$ *i* ∈ {*1*..<*card S1*} . $\mathcal{P}$(*?c* (*g i*) | ($\bigcap$(*?c* ‘ *g* ‘ {*0*..<*i*}))))
using *prob-cond-inter-index-fn-compl*[*of S1 F*] *Seq s1events psubset*(*1*) *s1F bb* **by** *auto*
have *ineq*: $\bigwedge$ *i*. *i* ∈ {*1*..<*card S1*} ⟹ $\mathcal{P}$(*?c* (*g i*) | ($\bigcap$(*?c* ‘ *g* ‘ {*0*..<*i*}))) ≥ (*1* − (*f* (*g i*)))
using *lovasz-inequality*[*of S1 F A Ai* - *S1 g S1* {} *f*] *sevents finS psubset.prems*(*2*)
psubset.prems(*4*) *bb psubset.hyps*(*2*)[*of* - *g* -] *Seq* **by** *fastforce*
have ($\bigwedge$*i*. *i* ∈ {*1*..<*card S1*} ⟹ *1* − *f* (*g i*) ≥ *0*) **using** *igt0* **by** *simp*
then have ($\prod$ *i* ∈ {*1*..<(*card S1*)} . $\mathcal{P}$(*?c* (*g i*) | ($\bigcap$(*?c* ‘ *g* ‘ {*0*..<*i*}))))
≥ ($\prod$ *i* ∈ {*1*..<(*card S1*)} . (*1* − (*f* (*g i*))))
using *ineq prod-mono* **by** (*smt*(*verit*, *ccfv-threshold*))
moreover have *prob* (*?c* (*g 0*)) ≥ (*1* − *f* (*g 0*))
proof −
have *g0in*: *g 0* ∈ *A* **using** *bb csgt0* **using** *psubset.prems*(*2*) *bij-betwE Seq* **by** *fastforce*
then have *prob* (*?c* (*g 0*)) = *1* − *prob* (*F* (*g 0*)) **using** *Aevents* **by** (*simp add*: *prob-compl*)
then show *?thesis* **using** *lovasz-inductive-base*[*of G F A f g 0*]
prob-Ai assms(*4*) *dep-graph-verts fbounds g0in* **by** *auto*
qed
moreover have *0* ≤ ($\prod$ *i* = *1*..<*card S1*. *1* − *f* (*g i*)) **using** *igt0* **by** (*force intro*: *prod-nonneg*)
ultimately have *prob* ($\bigcap$(*?c* ‘ *S1*)) ≥ (*1* − (*f* (*g 0*))) ∗ ($\prod$ *i* ∈ {*1*..<(*card S1*)} . (*1* − (*f* (*g i*))))
using *pcond igt0 mult-mono*′[*of* (*1* − (*f* (*g 0*)))] **by** *fastforce*
moreover have {*0*..<*card S1*} = {*0*} ∪ {*1*..<*card S1*} **using** *csgt0* **by** *auto*
ultimately have *prob* ($\bigcap$(*?c* ‘ *S1*)) ≥ ($\prod$ *i* ∈ {*0*..<(*card S1*)} . (*1* − (*f* (*g i*)))) **by** *auto*
moreover have ($\prod$ *i* ∈ {*0*..<(*card S1*)} . (*1* − (*f* (*g i*)))) = ($\prod$ *i* ∈ *S1* . (*1* − (*f* (*i*))))
using *prod.reindex-bij-betw bb* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
ultimately show *?thesis* **using** *peq Seq* **by** *blast*
next
case *s2F*: *False*
have *s2inter*: $\bigcap$ (*?c* ‘ *S2*) ∈ *events*

```
      using s2F finS2 s2cev Inter-event-ss[of ?c ‘ S2] by auto
    have split: (⋂ Aj ∈ S . (?c Aj)) = (⋂ (?c ‘S1)) ∩ (⋂ (?c ‘ S2))
      using S1-def S2-def by auto
    then have 𝒫(F Ai | (⋂ Aj ∈ S . (?c Aj))) = 𝒫(F Ai | (⋂ (?c ‘S1)) ∩ (⋂ (?c ‘ S2))) by simp
      moreover have s2n0: prob (⋂ (?c ‘ S2)) ≠ 0 using psubset.prems(4) S2-def
        by (metis Int-lower2 split finite-measure-mono measure-le-0-iff s2inter semiring-norm(137))
    moreover have ⋂ (?c ‘ S1) ∈ events
      using finS1 S1-def scompl-ev s1F Inter-event-ss[of (?c ‘ S1)] by auto
    ultimately have peq: 𝒫(F Ai | (⋂ Aj ∈ S . (?c Aj))) = 𝒫(F Ai ∩ (⋂ (?c ‘S1)) | ⋂ (?c ‘ S2))/
        𝒫(⋂ (?c ‘S1) | ⋂ (?c ‘S2)) (is 𝒫(F Ai | (⋂ Aj ∈ S . (?c Aj))) = ?Num/?Den)
      using cond-prob-dual-intersect[of F Ai ⋂ (?c ‘S1) ⋂ (?c ‘S2)] assms(2)
        psubset.prems(1) s2inter by fastforce
      have ?Num ≤ 𝒫(F Ai | ⋂ (?c ‘S2)) using cond-prob-inter-set-lt[of F Ai ⋂ (?c ‘S2) ?c ‘ S1]
       using s1events finS1 psubset.prems(1) assms(2) s2inter finite-imageI[of S1 F] by blast
      then have ?Num ≤ prob (F Ai) using s2F s2prob-eq by auto
      moreover have ?Den ≥ (∏ Aj ∈ S1 . (1 − (f Aj))) using psubset.hyps
      proof −
       have prob (⋂(?c ‘ S2)) > 0 using s2n0 by (meson zero-less-measure-iff)

        then have pcond: 𝒫(⋂ (?c ‘S1) | ⋂ (?c ‘S2)) =
          (∏ i = 0..<card S1 . 𝒫(?c (g i) | (⋂ (?c ‘ g ‘ {0..<i} ∪ (?c ‘ S2)))))
          using prob-cond-Inter-index-cond-compl-fn[of S1 ?c ‘ S2 F] s1F finS1 s2cev finS2 s2F
            s1events bb by auto
        have ⋀ i. i ∈ {0..<card S1} ⟹ 𝒫(?c (g i) | (⋂ (?c ‘ g ‘ {0..<i} ∪ (?c ‘ S2)))) ≥ (1 − f (g i))
         using lovasz-inequality[of S F A Ai - S1 g dg.neighborhood Ai S2 f] S1-def S2-def sevents
           finS psubset.prems(2) psubset.prems(4) bb psubset.hyps(2)[of - g -] psubset(1) s2F by meson
        then have c1: 𝒫(⋂ (?c ‘S1) | ⋂ (?c ‘S2)) ≥ (∏ i = 0..<card S1 . (1 − f (g i)))
          using prod-mono igt0 pcond bb by (smt(verit, ccfv-threshold))
        then have 𝒫(⋂ (?c ‘S1) | ⋂ (?c ‘S2)) ≥ (∏ i ∈ {0..<card S1} . (1 − f (g i))) by blast
        moreover have (∏ i ∈ {0..<card S1} . (1 − f (g i))) = (∏ x ∈ S1 . (1 − f x)) using bb
          using prod.reindex-bij-betw by fastforce
        ultimately show ?thesis by simp
      qed
      ultimately show ?thesis using peq by blast
    qed
```

ultimately show *?thesis* **by** (*intro split-prob-lt-helper*[*of G F A*])
(*simp-all add*: *dep-graph dep-graph-verts fbounds psubset.prems*(*1*) *prob-Ai*)
qed
qed
qed

The main lemma

theorem *lovasz-local-general*:
assumes $A \neq \{\}$
assumes F ‘ $A \subseteq$ *events*
assumes *finite A*
assumes $\bigwedge$ *Ai* . $Ai \in A \Longrightarrow$ $f\ Ai \geq 0 \wedge f\ Ai < 1$
assumes *dependency-digraph G M F*
assumes $\bigwedge$ *Ai*. $Ai \in A \Longrightarrow$ (*prob* (*F Ai*) $\leq$ (*f Ai*) $*$ ($\prod$ *Aj* $\in$ *pre-digraph.neighborhood G Ai*. (*1* $-$ (*f Aj*))))
assumes *pverts G* $=$ *A*
shows *prob* ($\bigcap$ *Ai* $\in$ *A* . (*space M* $-$ (*F Ai*))) $\geq$ ($\prod$ *Ai* $\in$ *A* . (*1* $-$ *f Ai*)) ($\prod$ *Ai* $\in$ *A* . (*1* $-$ *f Ai*)) $>$ *0*
proof $-$
show *gt0*: ($\prod$ *Ai* $\in$ *A* . (*1* $-$ *f Ai*)) $>$ *0* **using** *assms*(*4*) **by** (*simp add*: *prod-pos*)
let *?c* $=$ λ *i*. *space M* $-$ *F i*
interpret *dg*: *dependency-digraph G M F* **using** *assms*(*5*) **by** *simp*
have *general*: $\bigwedge$*Ai S*. $Ai \in A \Longrightarrow S \subseteq A - \{Ai\} \Longrightarrow S \neq \{\} \Longrightarrow$ *prob* ($\bigcap$ *Aj* $\in$ *S* . (*?c Aj*)) $>$ *0*
$\Longrightarrow$ $\mathcal{P}$(*F Ai* | ($\bigcap$ *Aj* $\in$ *S* . (*?c Aj*))) $\leq$ *f Ai*
using *assms lovasz-inductive*[*of A F f G*] **by** *simp*
have *base*: $\bigwedge$ *Ai*. $Ai \in A \Longrightarrow$ *prob* (*F Ai*) $\leq$ *f Ai*
using *lovasz-inductive-base assms*(*4*) *assms*(*6*) *assms*(*5*) *assms*(*7*) **by** *blast*
show *prob* ($\bigcap$ *Ai* $\in$ *A* . (*?c Ai*)) $\geq$ ($\prod$ *Ai* $\in$ *A* . (*1* $-$ *f Ai*))
using *assms*(*3*) *assms*(*1*) *assms*(*2*) *assms*(*4*) *general base*
proof (*induct A rule*: *finite-ne-induct*)
case (*singleton x*)
then show *?case* **using** *singleton.prems singleton prob-compl* **by** *auto*
next
case (*insert x X*)
define *Ax* **where** *Ax* $=$ *?c* ‘ (*insert x X*)
have *xie*: *F x* $\in$ *events* **using** *insert.prems* **by** *simp*
have *A'ie*: $\bigcap$(*?c* ‘ *X*) $\in$ *events* **using** *insert.prems insert.hyps* **by** *auto*
have ($\bigwedge$*Ai S*. *Ai* $\in$ *insert x X* $\Longrightarrow$ *S* $\subseteq$ *insert x X* $-$ {*Ai*} $\Longrightarrow$ *S* $\neq$ {} $\Longrightarrow$ *prob* ($\bigcap$ *Aj* $\in$ *S* . (*?c Aj*)) $>$ *0*
$\Longrightarrow$ $\mathcal{P}$(*F Ai* | $\bigcap$ (*?c* ‘ *S*)) $\leq$ *f Ai*) **using** *insert.prems* **by** *simp*
then have ($\bigwedge$*Ai S*. *Ai* $\in$ *X* $\Longrightarrow$ *S* $\subseteq$ *X* $-$ {*Ai*} $\Longrightarrow$ *S* $\neq$ {} $\Longrightarrow$ *prob* ($\bigcap$ *Aj* $\in$ *S* . (*?c Aj*)) $>$ *0*
$\Longrightarrow$ $\mathcal{P}$(*F Ai* | $\bigcap$ (*?c* ‘ *S*)) $\leq$ *f Ai*) **by** *auto*
then have *A'gt*: ($\prod$ *Ai*$\in$*X*. *1* $-$ *f Ai*) $\leq$ *prob* ($\bigcap$ (*?c* ‘ *X*))
using *insert.hyps*(*4*) *insert.prems*(*2*) *insert.prems*(*1*) *insert.prems*(*4*) **by** *auto*
then have *prob* ($\bigcap$(*?c* ‘ *X*)) $>$ *0* **using** *insert.hyps insert.prems prod-pos basic-trans-rules*(*22*)
diff-gt-0-iff-gt **by** (*metis* (*no-types*, *lifting*) *insert-Diff insert-subset sub-*

set-insertI)
then have $\mathcal{P}((?c\ x) \mid (\bigcap (?c\ `\ X))) = 1 - \mathcal{P}(F\ x \mid (\bigcap (?c\ `\ X)))$
using *cond-prob-neg*[*of* $\bigcap (?c\ `\ X)$ $F\ x$] *xie A'ie* **by** *simp*
moreover have $\mathcal{P}(F\ x \mid (\bigcap (?c\ `\ X))) \leq f\ x$ **using** *insert.prems*(*3*)[*of* x X] *insert.hyps*(*2*) *insert*(*3*)
$A'gt$ ‹*0* < *prob* $(\bigcap\ (?c\ `\ X))$› **by** *fastforce*
ultimately have *pnxgt*: $\mathcal{P}((?c\ x) \mid (\bigcap (?c\ `\ X))) \geq 1 - f\ x$ **by** *simp*
have *xgt0*: $1 - f\ x \geq 0$ **using** *insert.prems*(*2*)[*of x*] **by** *auto*
have *prob* $(\bigcap\ Ax) =$ *prob* $((?c\ x) \cap \bigcap (?c\ `\ X))$ **using** *Ax-def* **by** *simp*
also have ... = *prob* $(\bigcap (?c\ `\ X)) * \mathcal{P}((?c\ x) \mid (\bigcap (?c\ `\ X)))$
using *prob-intersect-B xie A'ie* **by** *simp*
also have ... $\geq (\prod Ai \in X.\ 1 - f\ Ai) * (1 - f\ x)$ **using** *A'gt pnxgt mult-left-le*
‹*0* < *prob* $(\bigcap (?c\ `\ X))$› *xgt0 mult-mono* **by** (*smt*(*verit*))
finally have *prob* $(\bigcap\ Ax) \geq (\prod Ai \in insert\ x\ X.\ 1 - f\ Ai)$
by (*simp add*: *local.insert*(*1*) *local.insert*(*3*) *mult.commute*)
then show *?case* **using** *Ax-def* **by** *auto*
qed
qed

7.4 Lovasz Corollaries and Variations

corollary *lovasz-local-general-positive*:
assumes $A \neq \{\}$
assumes $F\ `\ A \subseteq events$
assumes *finite A*
assumes $\bigwedge Ai\ .\ Ai \in A \Longrightarrow\ f\ Ai \geq 0 \wedge f\ Ai < 1$
assumes *dependency-digraph G M F*
assumes $\bigwedge Ai.\ \ Ai \in A \Longrightarrow$ (*prob* $(F\ Ai) \leq$
$(f\ Ai) * (\prod\ Aj \in$ *pre-digraph.neighborhood G Ai*. $(1 - (f\ Aj))))$
assumes *pverts* $G = A$
shows *prob* $(\bigcap\ Ai \in A\ .\ (space\ M - (F\ Ai))) > 0$
using *assms lovasz-local-general*(*1*)[*of A F f G*] *lovasz-local-general*(*2*)[*of A F f G*] **by** *simp*

theorem *lovasz-local-symmetric-dep-graph*:
fixes e :: *real*
fixes d :: *nat*
assumes $A \neq \{\}$
assumes $F\ `\ A \subseteq events$
assumes *finite A*
assumes *dependency-digraph G M F*
assumes $\bigwedge Ai.\ \ Ai \in A \Longrightarrow$ *out-degree G Ai* $\leq d$
assumes $\bigwedge Ai.\ Ai \in A \Longrightarrow$ *prob* $(F\ Ai) \leq p$
assumes $exp(1) * p * (d + 1) \leq 1$
assumes *pverts* $G = A$
shows *prob* $(\bigcap\ Ai \in A\ .\ (space\ M - (F\ Ai))) > 0$
proof (*cases* $d = 0$)
case *True*
interpret *g*: *dependency-digraph G M F* **using** *assms*(*4*) **by** *simp*

have *indep-events F A* **using** *g.dep-graph-indep-events*[*of A*] *assms*(*8*) *assms*(*5*) *True* **by** *simp*
moreover have $p < 1$
proof −
have *exp* (*1*) ∗ $p \leq 1$ **using** *assms*(*7*) *True* **by** *simp*
then show *?thesis* **using** *exp-gt-one less-1-mult linorder-neqE-linordered-idom rel-simps*(*68*)
verit-prod-simplify(*2*) **by** (*smt* (*verit*) *mult-le-cancel-left1*)
qed
ultimately show *?thesis*
using *complete-indep-bound3*[*of A F*] *assms*(*2*) *assms*(*1*) *assms*(*3*) *assms*(*6*) **by** *force*
next
case *False*
define $f :: nat \Rightarrow real$ **where** $f \equiv (\lambda\ Ai\ .\ 1\ /(d + 1))$
then have *fbounds*: $\bigwedge Ai.\ f\ Ai \geq 0 \wedge f\ Ai < 1$ **using** *f-def False* **by** *simp*
interpret *dg*: *dependency-digraph G M F* **using** *assms*(*4*) **by** *auto*

have $\bigwedge Ai.\ Ai \in A \Longrightarrow prob\ (F\ Ai) \leq (f\ Ai) * (\prod\ Aj \in dg.neighborhood\ Ai\ .\ (1 - (f\ Aj)))$
proof −
fix *Ai* **assume** *ain*: $Ai \in A$
have *d-boundslt1*: $(1/(d + 1)) < 1$ **and** *d-boundsgt0*: $(1/(d + 1)) > 0$ **using** *False* **by** *fastforce+*
have *d-bounds2*: $(1 - (1\ /(d + 1)))\hat{}d < 1$ **using** *False*
by(*simp add*: *field-simps*) (*smt* (*verit*) *of-nat-0-le-iff power-mono-iff*)
have *d-bounds0*: $(1 - (1\ /(d + 1)))\hat{}d > 0$ **using** *False* **by** (*simp*)
have *exp*(*1*) > (*1* + *1*/*d*) *powr d* **using** *exp-1-gt-powr False* **by** *simp*
then have *exp*(*1*) > (*1* + *1*/*d*)^*d* **using** *False* **by** (*simp add*: *powr-realpow zero-compare-simps*(*2*))
moreover have $1/(1+ 1/d)\hat{}d = (1 - (1/(d+1)))\hat{}d$
proof −
have $1/(1+ 1/d)\hat{}d = 1/((d/d) + 1/d)\hat{}d$ **by** (*simp add*: *field-simps*)
then show *?thesis* **by** (*simp add*: *field-simps*)
qed
ultimately have *exp-lt*: $1/exp(1) < (1 - (1\ /(d + 1)))\hat{}d$
by (*metis d-bounds0 frac-less2 less-eq-real-def of-nat-zero-less-power-iff power-eq-if zero-less-divide-1-iff*)
then have $(1\ /(d + 1)) * (1 - (1\ /(d + 1)))\hat{}d > (1\ /(d + 1)) * (1/exp(1))$
using *exp-lt mult-strict-left-mono*[*of* $1/exp(1)$ $(1 - (1\ /(d + 1)))\hat{}d$ $(1/(d+1))$] *d-boundslt1*
by *simp*
then have $(1\ /(d + 1)) * (1 - (1\ /(d + 1)))\hat{}d > (1/((d+1) * exp(1)))$ **by** *auto*
then have *gtp*: $(1\ /(d + 1)) * (1 - (1\ /(d + 1)))\hat{}d > p$
by (*smt* (*verit*, *ccfv-SIG*) *d-boundslt1 d-boundsgt0 assms*(*7*) *divide-divide-eq-left divide-less-cancel*
divide-less-eq divide-nonneg-nonpos nonzero-mult-div-cancel-left not-exp-le-zero)

have *card* (*dg.neighborhood Ai*) ≤ *d* **using** *assms*(*5*) *dg.out-degree-neighborhood ain* **by** *auto*
then have (∏ *Aj* ∈ *dg.neighborhood Ai* . (*1* − (*1* /(*d* + *1*)))) ≥ (*1* − (*1* /(*d* + *1*)))^*d*
using *prod-constant-ge*[*of dg.neighborhood Ai d 1* − (*1*/*d*+*1*)] **using** *d-boundslt1* **by** *auto*
then have (*1* /(*d* + *1*)) ∗ (∏ *Aj* ∈ *dg.neighborhood Ai* . (*1* − (*1* /(*d* + *1*)))) ≥ (*1* /(*d* + *1*))∗ (*1* − (*1* /(*d* + *1*)))^*d*
by (*simp add*: *divide-right-mono*)
then have (*1* /(*d* + *1*)) ∗ (∏ *Aj* ∈ *dg.neighborhood Ai* . (*1* − (*1* /(*d* + *1*)))) > *p*
using *gtp* **by** *simp*
then show *prob* (*F Ai*) ≤ *f Ai* ∗ (∏ *Aj* ∈ *dg.neighborhood Ai* . (*1* − *f Aj*))
using *assms*(*6*) ‹*Ai* ∈ *A*› *f-def* **by** *force*
qed
then show *?thesis* **using** *lovasz-local-general-positive*[*of A F f G*]
assms(*4*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*8*) *fbounds* **by** *auto*
qed

corollary *lovasz-local-symmetric4gt*:
fixes *e* :: *real*
fixes *d* :: *nat*
assumes *A* ≠ {}
assumes *F* ‘ *A* ⊆ *events*
assumes *finite A*
assumes *dependency-digraph G M F*
assumes ⋀ *Ai*. *Ai* ∈ *A* ⟹ *out-degree G Ai* ≤ *d*
assumes ⋀ *Ai*. *Ai* ∈ *A* ⟹ *prob* (*F Ai*) ≤ *p*
assumes *4* ∗ *p* ∗ *d* ≤ *1*
assumes *d* ≥ *3*
assumes *pverts G* = *A*
shows *prob* (⋂ *Ai* ∈ *A* . (*space M* − *F Ai*)) > *0*
proof −
have *exp*(*1*)∗ *p* ∗ (*d* + *1*) ≤ *1*
proof (*cases p* = *0*)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
then have *pgt*: *p* > *0* **using** *assms*(*1*) *assms*(*6*) *assms*(*3*) *ex-min-if-finite less-eq-real-def*
by (*meson basic-trans-rules*(*23*) *basic-trans-rules*(*24*) *linorder-neqE-linordered-idom measure-nonneg*)
have *3* ∗ (*d* + *1*) ≤ *4* ∗ *d* **by** (*simp add*: *field-simps assms*(*8*))
then have *exp*(*1*) ∗ (*d* + *1*) ≤ *4* ∗*d*
using *exp-le exp-gt-one*[*of 1*] *assms*(*8*)
by (*smt* (*verit*, *del-insts*) *Num.of-nat-simps*(*2*) *Num.of-nat-simps*(*5*) *le-add2 le-eq-less-or-eq*

mult-right-mono nat-less-real-le numeral.simps(3) numerals(1) of-nat-numeral)

then have *exp(1) * (d + 1) * p ≤ 4 *d *p* **using** *pgt* **by** *simp*
then show *?thesis* **using** *assms(7)* **by** (*simp add*: *field-simps*)
qed
then show *?thesis* **using** *assms lovasz-local-symmetric-dep-graph*[*of A F G d p*]
by *auto*
qed

lemma *lovasz-local-symmetric4*:
fixes *e* :: *real*
fixes *d* :: *nat*
assumes *A* ≠ {}
assumes *F* ' *A* ⊆ *events*
assumes *finite A*
assumes *dependency-digraph G M F*
assumes ⋀ *Ai*. *Ai* ∈ *A* ⟹ *out-degree G Ai* ≤ *d*
assumes ⋀ *Ai*. *Ai* ∈ *A* ⟹ *prob* (*F Ai*) ≤ *p*
assumes *4 * p * d* ≤ *1*
assumes *d* ≥ *1*
assumes *pverts G* = *A*
shows *prob* (⋂ *Ai* ∈ *A* . (*space M* − *F Ai*)) > *0*
proof (*cases d* ≥ *3*)
case *True*
then show *?thesis* **using** *lovasz-local-symmetric4gt assms*
by *presburger*
next
case *d3*: *False*
define *f* :: *nat* ⇒ *real* **where** *f* ≡ (λ *Ai* . *1* /(*d* + *1*))
then have *fbounds*: ⋀ *Ai*. *f Ai* ≥ *0* ∧ *f Ai* < *1* **using** *f-def assms(8)* **by** *simp*
interpret *dg*: *dependency-digraph G M F* **using** *assms(4)* **by** *auto*
have ⋀ *Ai*. *Ai* ∈ *A* ⟹ *prob* (*F Ai*) ≤ (*f Ai*) * (∏ *Aj* ∈ *dg.neighborhood Ai* .
(*1* − (*f Aj*)))
proof −
fix *Ai* **assume** *ain*: *Ai* ∈ *A*
have *d-boundslt1*: (*1*/(*d* + *1*)) < *1* **and** *d-boundsgt0*: (*1*/(*d* + *1*))> *0* **using**
assms **by** *fastforce+*
have *plt*: *1*/(*4*∗*d*) ≥ *p* **using** *assms(7) assms(8)*
by (*metis* (*mono-tags*, *opaque-lifting*) *Num.of-nat-simps(5) bot-nat-0.not-eq-extremum*
le-numeral-extra(2)
more-arith-simps(11) mult-of-nat-commute nat-0-less-mult-iff of-nat-0-less-iff
of-nat-numeral
pos-divide-less-eq rel-simps(51) verit-comp-simplify(3))
then have *gtp*: (*1* /(*d* + *1*))∗ (*1* − (*1* /(*d* + *1*)))^*d* ≥ *p*
proof (*cases d* = *1*)
case *False*
then have *d* = *2* **using** *d3 assms(8)* **by** *auto*
then show *?thesis* **using** *plt* **by** (*simp add*: *field-simps*)

qed (*simp*)
have *card* (*dg.neighborhood Ai*) ≤ *d* **using** *assms*(*5*) *dg.out-degree-neighborhood ain* **by** *auto*
then have (∏ *Aj* ∈ *dg.neighborhood Ai* . (*1* − (*1* /(*d* + *1*)))) ≥ (*1* − (*1* /(*d* + *1*)))^*d*
using *prod-constant-ge*[*of dg.neighborhood Ai d 1* − (*1*/*d*+*1*)] **using** *d-boundslt1* **by** *auto*
then have (*1* /(*d* + *1*)) ∗ (∏ *Aj* ∈ *dg.neighborhood Ai* . (*1* − (*1* /(*d* + *1*)))) ≥ (*1* /(*d* + *1*))∗ (*1* − (*1* /(*d* + *1*)))^*d*
by (*simp add*: *divide-right-mono*)
then have (*1* /(*d* + *1*)) ∗ (∏ *Aj* ∈ *dg.neighborhood Ai* . (*1* − (*1* /(*d* + *1*)))) ≥ *p*
using *gtp* **by** *simp*
then show *prob* (*F Ai*) ≤ *f Ai* ∗ (∏ *Aj* ∈ *dg.neighborhood Ai* . (*1* − *f Aj*))
using *assms*(*6*) ‹*Ai* ∈ *A*› *f-def* **by** *force*
qed
then show *?thesis*
using *lovasz-local-general-positive*[*of A F f G*] *assms*(*4*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*9*) *fbounds* **by** *auto*
qed

Converting between dependency graph and indexed set representation of mutual independence

lemma (**in** *pair-digraph*) *g-Ai-simplification*:
assumes *Ai* ∈ *A*
assumes *g Ai* ⊆ *A* − {*Ai*}
assumes *pverts G* = *A*
assumes *parcs G* = {*e* ∈ *A* × *A* . *snd e* ∈ (*A* − ({*fst e*} ∪ (*g* (*fst e*))))}
shows *g Ai* = *A* − ({*Ai*} ∪ *neighborhood Ai*)
proof −
have *g Ai* = *A* − ({*Ai*} ∪ {*v* ∈ *A* . *v* ∈ (*A* − ({*Ai*} ∪ (*g* (*Ai*))))}) **using** *assms*(*2*) **by** *auto*
then have *g Ai* = *A* − ({*Ai*} ∪ {*v* ∈ *A* . (*Ai*, *v*) ∈ *parcs G*})
using *Collect-cong assms*(*1*) *mem-Collect-eq assms*(*3*) *assms*(*4*) **by** *auto*
then show *g Ai* = *A* − ({*Ai*} ∪ *neighborhood Ai*) **unfolding** *neighborhood-def* **using** *assms*(*3*) **by** *simp*
qed

lemma *define-dep-graph-set*:
assumes *A* ≠ {}
assumes *F* ‘ *A* ⊆ *events*
assumes *finite A*
assumes ⋀ *Ai*. *Ai* ∈ *A* ⟹ *g Ai* ⊆ *A* − {*Ai*} ∧ *mutual-indep-events* (*F Ai*) *F* (*g Ai*)
shows *dependency-digraph* (| *pverts* = *A*, *parcs* = {*e* ∈ *A* × *A* . *snd e* ∈ (*A* − ({*fst e*} ∪ (*g* (*fst e*))))} |) *M F*
(**is** *dependency-digraph ?G M F*)
proof −
interpret *pd*: *pair-digraph ?G*

using *assms(3)***by** (*unfold-locales*) *auto*
have $\bigwedge$ *Ai*. *Ai* ∈ *A* ⟹ *g Ai* ⊆ *A* − {*Ai*} **using** *assms(4)* **by** *simp*
then have $\bigwedge$ *i*. *i* ∈ *A* ⟹ *g i* = *A* − ({*i*} ∪ *pd.neighborhood i*)
using *pd.g-Ai-simplification*[*of - A g*] *pd.pair-digraph* **by** *auto*
then have *dependency-digraph ?G M F* **using** *assms(2) assms(4)* **by** (*unfold-locales*) *auto*
then show *?thesis* **by** *simp*
qed

lemma *define-dep-graph-deg-bound*:
assumes *A* ≠ {}
assumes *F* ‘ *A* ⊆ *events*
assumes *finite A*
assumes $\bigwedge$ *Ai*. *Ai* ∈ *A* ⟹ *g Ai* ⊆ *A* − {*Ai*} ∧ *card* (*g Ai*) ≥ *card A* − *d* − *1* ∧
mutual-indep-events (*F Ai*) *F* (*g Ai*)
shows $\bigwedge$ *Ai*. *Ai* ∈ *A* ⟹
out-degree (| *pverts* = *A*, *parcs* = {*e* ∈ *A* × *A* . *snd e* ∈ (*A* − ({*fst e*} ∪ (*g* (*fst e*))))} |) *Ai* ≤ *d*
(**is** $\bigwedge$ *Ai*. *Ai* ∈ *A* ⟹ *out-degree* (*with-proj ?G*) *Ai* ≤ *d*)
proof −
interpret *pd*: *dependency-digraph ?G M F* **using** *assms define-dep-graph-set* **by** *simp*
show $\bigwedge$ *Ai*. *Ai* ∈ *A* ⟹ *out-degree ?G Ai* ≤ *d*
proof −
fix *Ai* **assume** *a*: *Ai* ∈ *A*
then have *geq*: *g Ai* = *A* − ({*Ai*} ∪ *pd.neighborhood Ai*)
using *assms(4)*[*of Ai*] *pd.pair-digraph pd.g-Ai-simplification*[*of Ai A g*] **by** *simp*
then have *pss*: *g Ai* ⊂ *A* **using** *a* **by** *auto*
then have *card* (*g Ai*) = *card* (*A* − ({*Ai*} ∪ *pd.neighborhood Ai*)) **using** *assms(4) geq* **by** *argo*
moreover have *ss*: ({*Ai*} ∪ *pd.neighborhood Ai*) ⊆ *A* **using** *pd.neighborhood-wf a* **by** *simp*
moreover have *finite* ({*Ai*} ∪ *pd.neighborhood Ai*)
using *calculation(2) assms(3) finite-subset* **by** *auto*
moreover have *Ai* ∉ *pd.neighborhood Ai* **using** *pd.neighborhood-self-not* **by** *simp*
moreover have *card* {*Ai*} = *1* **using** *is-singleton-altdef* **by** *auto*
moreover have *cardss*: *card* ({*Ai*} ∪ *pd.neighborhood Ai*) = *1* + *card* (*pd.neighborhood Ai*)
using *calculation(5) calculation(4) card-Un-disjoint pd.neighborhood-finite* **by** *auto*
ultimately have *eq*: *card* (*g Ai*) = *card A* − *1* − *card* (*pd.neighborhood Ai*)
using *card-Diff-subset*[*of* ({*Ai*} ∪ *pd.neighborhood Ai*) *A*] *assms(3)* **by** *presburger*
have *ggt*: $\bigwedge$ *Ai*. *Ai* ∈ *A* ⟹ *card* (*g Ai*) ≥ *int* (*card A*) − *int d* − *1*
using *assms(4)* **by** *fastforce*
have *card* (*pd.neighborhood Ai*) = *card A* − *1* − *card* (*g Ai*)

using *cardss assms(3) card-mono diff-add-inverse diff-diff-cancel diff-le-mono ss eq*
by (*metis* (*no-types*, *lifting*))
moreover have *card A* ≥ (*1* + *card* (*g Ai*)) **using** *pss assms(3) card-seteq not-less-eq-eq* **by** *auto*
ultimately have *card* (*pd.neighborhood Ai*) = *int* (*card A*) − *1* − *int* (*card* (*g Ai*)) **by** *auto*
moreover have *int* (*card* (*g Ai*)) ≥ (*card A*) − (*int d*) − *1* **using** *ggt a* **by** *simp*
ultimately show *out-degree ?G Ai* ≤ *d* **using** *pd.out-degree-neighborhood* **by** *simp*
qed
qed

lemma *obtain-dependency-graph*:
assumes *A* ≠ {}
assumes *F* ‘ *A* ⊆ *events*
assumes *finite A*
assumes ⋀ *Ai*. *Ai* ∈ *A* ⟹
(∃ *S* . *S* ⊆ *A* − {*Ai*} ∧ *card S* ≥ *card A* − *d* − *1* ∧ *mutual-indep-events* (*F Ai*) *F S*)
obtains *G* **where** *dependency-digraph G M F pverts G* = *A* ⋀ *Ai*. *Ai* ∈ *A* ⟹ *out-degree G Ai* ≤ *d*
proof −
obtain *g* **where** *gdef*: ⋀ *Ai*. *Ai* ∈ *A* ⟹ *g Ai* ⊆ *A* − {*Ai*} ∧ *card* (*g Ai*) ≥ *card A* − *d* − *1* ∧
mutual-indep-events (*F Ai*) *F* (*g Ai*) **using** *assms(4)* **by** *metis*
then show *?thesis*
using *define-dep-graph-set*[*of A F g*] *define-dep-graph-deg-bound*[*of A F g d*]*that assms* **by** *auto*
qed

This is the variation of the symmetric version most commonly in use

theorem *lovasz-local-symmetric*:
fixes *d* :: *nat*
assumes *A* ≠ {}
assumes *F* ‘ *A* ⊆ *events*
assumes *finite A*
assumes ⋀ *Ai*. *Ai* ∈ *A* ⟹ (∃ *S* . *S* ⊆ *A* − {*Ai*} ∧ *card S* ≥ *card A* − *d* − *1* ∧ *mutual-indep-events* (*F Ai*) *F S*)
assumes ⋀ *Ai*. *Ai* ∈ *A* ⟹ *prob* (*F Ai*) ≤ *p*
assumes *exp*(*1*)∗ *p* ∗ (*d* + *1*) ≤ *1*
shows *prob* (⋂ *Ai* ∈ *A* . (*space M* − (*F Ai*))) > *0*
proof −
obtain *G* **where** *odg*: *dependency-digraph G M F pverts G* = *A* ⋀ *Ai*. *Ai* ∈ *A* ⟹ *out-degree G Ai* ≤ *d*
using *assms obtain-dependency-graph* **by** *metis*
then show *?thesis* **using** *odg assms lovasz-local-symmetric-dep-graph*[*of A F G d p*] **by** *auto*

qed

lemma *lovasz-local-symmetric4-set*:
fixes $d :: nat$
assumes $A \neq \{\}$
assumes F ' $A \subseteq events$
assumes *finite* A
assumes $\bigwedge Ai.\ Ai \in A \Longrightarrow (\exists\ S\ .\ S \subseteq A - \{Ai\} \wedge card\ S \geq card\ A - d - 1 \wedge mutual\text{-}indep\text{-}events\ (F\ Ai)\ F\ S)$
assumes $\bigwedge Ai.\ Ai \in A \Longrightarrow prob\ (F\ Ai) \leq p$
assumes $4 * p * d \leq 1$
assumes $d \geq 1$
shows $prob\ (\bigcap\ Ai \in A\ .\ (space\ M - F\ Ai)) > 0$
proof −
obtain G **where** *odg*: *dependency-digraph* $G\ M\ F\ pverts\ G = A \bigwedge Ai.\ \ Ai \in A \Longrightarrow out\text{-}degree\ G\ Ai \leq d$
using *assms obtain-dependency-graph* **by** *metis*
then show *?thesis* **using** *odg assms lovasz-local-symmetric4* [*of A F G d p*] **by** *auto*
qed
end

end
theory *Lovasz-Local-Root*
imports
PiE-Rel-Extras
Digraph-Extensions

Prob-Events-Extras
Cond-Prob-Extensions
Indep-Events

Basic-Method
Lovasz-Local-Lemma
begin
end

References

[1] N. Alon and J. H. Spencer. *The Probabilistic Method*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, Hoboken, N.J, 4th edition, 2016.

[2] L. Noschinski. A Graph Library for Isabelle. *Mathematics in Computer Science*, 9(1):23–39, Mar. 2015.

[3] Y. Zhao. Probabilistic methods in combinatorics, 2020. Lecture notes MIT 18.226, Fall 2020, https://ocw.mit.edu/courses/

18-226-probabilistic-method-in-combinatorics-fall-2020/resources/mit18_226f20_full_notes/.