

FCFamilies

Filip Marić, Bojan Vučković, Miodrag Živković

April 8, 2018

Contents

1	Combinatorics	4
1.1	Generating all permutations	4
1.2	Generating all combinations	5
1.3	Generating all nonempty subsets	7
1.4	Generating all m-elements subsets of n-element set	7
2	Representing lists of natural numbers by natural numbers	7
2.1	sumpows	8
2.2	nat2list	8
2.3	list2nat	10
3	Union closed families	11
3.1	Union closed families	11
3.1.1	Closure – minimal union closed family containing F	11
3.2	Union closed families closed to unions with an additional family	13
3.2.1	Union closed extensions	13
4	Families of sets	14
5	Frankl’s condition	14
5.1	Frankl’s condition	14
5.1.1	Frankl’s function	14
5.2	FC Families	15
6	Executable implementations of set families	16
6.1	Abstract representation of sets and families	16
6.1.1	Implementation of sets by sorted and distinct lists	17
6.1.2	Implementation of sets of nats by nats	18
6.2	Abstract representation of union closed families	18
6.2.1	Implementation by sorted and distinct lists	22
6.2.2	Implementation by sets represented as natural numbers	24

7	Weights and shares	25
7.1	Weight functions	25
7.2	Shares	25
7.3	Hypercube construction	26
7.4	Frankl's families characterization using weights	27
7.5	Frankl's families characterization using shares	27
8	FC families characterization using shares (Poonen's theorem)	28
8.1	HyperCube projection	28
8.2	All shares non-negative	29
9	nonFC families characterization using shares (Poonen's theorem)	29
10	SomeShareNegative — combinatorial search for union-closed extension with a negative share	30
10.1	Abstract representation of sets and families with weights and shares	31
10.1.1	Implementation by sorted and distinct lists	32
10.1.2	Implementation by natural numbers	32
10.2	Weight functions implemented by mappings	32
10.2.1	Representation of sets by lists	33
10.2.2	Representation of sets by natural numbers	33
11	SomeShareNegative – executable implementation	33
11.1	Refinement of SomeShareNegativeFunction	34
11.1.1	Initial version	34
11.1.2	Passing current family share as a parameter	34
11.1.3	Caching set shares	35
11.1.4	Passing the sum of shares of elements in the current list as a parameter	36
11.1.5	Incremental insert and close operation and calculation of share of the extended family	37
11.1.6	Final implementation	38
11.2	Interpretations	39
11.2.1	Representation of sets by lists	39
11.2.2	Representation of sets by natural numbers	39
12	Isomorphisms of set families	40
12.1	Properties preserved by injective functions	40
12.2	Injective embedding	41
12.3	Isomorphism definition	42
12.4	Iso-representing collections of families	43

12.5	Non-isomorphic families of sets	43
12.5.1	Implementation by sets represented by (sorted and distinct) lists	45
13	Expressible sets and irreducible families	46
13.1	Irreducible families	46
13.2	Removing all expressible sets	47
13.3	Uniqueness of irreducible subfamily for the given closure . . .	47
13.3.1	Implementation by sorted and distinct lists	48
14	Covering	48
14.1	Definition of Covering	48
14.2	Covering and empty set	49
14.3	Covering and isomorphic families	50
14.4	Covering, closure and irreducible families	50
14.4.1	FC and nonFC covering implementation	51
15	L-partitioned families	52
15.1	Ordering of L-partitions	53
16	Generating all L-partitioned families with some given prop- erties	54
16.1	Implementation by sorted and distinct lists	55
16.2	Recursive enumeration procedure	57
16.3	Dynamic programming enumeration procedure	57
16.4	Generating all irreducible families	59
16.5	Generating all families that are not FCs covered by the given collection	60
16.5.1	Implementation	61
16.6	Generating all irreducible families that are not FCs covered by the given collection	63
16.6.1	Implementation	64
17	Minimal FC(6) families and maximal nonFC(6) families	65
18	Tactics	72
18.1	Tactics for verifying FC families	73
18.2	Tactics for verifying nonFC families	73
19	FC status proofs	73
20	Proof that all families are covered	83

1 Combinatorics

```

theory Combinatorics
imports Main
        HOL-Library.Permutation
        HOL-Library.List-lexord
        More.MoreList
begin

```

1.1 Generating all permutations

```

primrec interleave :: 'a  $\Rightarrow$  'a list  $\Rightarrow$  'a list list where
  interleave x [] = [[x]]
| interleave x (h # t) = (x # (h # t)) # (map ( $\lambda$  l. h # l) (interleave x t))

```

For example, *interleave* 1 [2, 3, 4] = [[1, 2, 3, 4], [2, 1, 3, 4], [2, 3, 1, 4], [2, 3, 4, 1]].

```

primrec permute :: 'a list  $\Rightarrow$  'a list list where
  permute [] = [[]]
| permute (h # t) = concat (map ( $\lambda$  l. interleave h l) (permute t))

```

For example, *permute* [1, 2, 3] = [[1, 2, 3], [2, 1, 3], [2, 3, 1], [1, 3, 2], [3, 1, 2], [3, 2, 1]].

```

lemma multiset-interleave:
  shows  $p \in \text{set } (\text{interleave } h \ a) \implies \text{mset } p = \text{mset } a + \{\#h\# \}$ 
  <proof>

```

```

lemma interleave-hd [simp]:  $h \# t \in \text{set } (\text{interleave } h \ t)$ 
  <proof>

```

```

lemma interleave-append [simp]:  $t1 \ @ \ [h] \ @ \ t2 \in \text{set } (\text{interleave } h \ (t1 \ @ \ t2))$ 
  <proof>

```

```

lemma isPermutation-permute:
  shows  $p \in \text{set } (\text{permute } l) \implies p <\sim\sim> l$ 
  <proof>

```

```

lemma mset-append:
  assumes  $\text{mset } p = \text{mset } t + \{\#h\# \}$ 
  shows  $\exists \ t1 \ t2. \ p = t1 \ @ \ [h] \ @ \ t2 \wedge \text{mset } t = \text{mset } t1 + \text{mset } t2$ 
  <proof>

```

```

lemma permute-isPermutation:
   $p <\sim\sim> l \implies p \in \text{set } (\text{permute } l)$ 
  <proof>

```

lemma *permute-bij*:
assumes $p \in \text{set } (\text{permute } l)$
shows $\exists f. \text{bij-betw } f \{..
 $(\forall i < \text{length } p. p ! i = l ! (f i))$
 $\langle \text{proof} \rangle$$

1.2 Generating all combinations

fun *combine-aux* :: 'a list \Rightarrow nat \Rightarrow nat \Rightarrow 'a list list **where**
combine-aux l n k =
 (if k = 0 then [[]] else
 if k = n then [l] else
 (case l of
 [] \Rightarrow []
 | (h # t) \Rightarrow
 (map ($\lambda l'. h \# l'$) (combine-aux t (n-(1::nat)) (k-(1::nat)))) @
 combine-aux t (n-(1::nat)) k))
declare *combine-aux.simps*[simp del]

lemma *combine-aux-induct*:
assumes
 $\bigwedge l n. P \text{ [] } l n 0$
 $\bigwedge l n. 0 < n \implies P [l] l n n$
 $\bigwedge k n. [0 < k; k \neq n] \implies P \text{ [] } n k$
 $\bigwedge h t n k. [$
 $P (\text{combine-aux } t (n-(1::nat)) (k-(1::nat))) t (n-(1::nat)) (k-(1::nat));$
 $P (\text{combine-aux } t (n-(1::nat)) k) t (n-(1::nat)) k] \implies$
 $P (\text{map } (op \# h) (\text{combine-aux } t (n-(1::nat)) (k-(1::nat)))) @$
 $\text{combine-aux } t (n-(1::nat)) k) (h \# t) n k$
shows $P (\text{combine-aux } l n k) l n k$
 $\langle \text{proof} \rangle$

definition *combine* :: 'a list \Rightarrow nat \Rightarrow 'a list list **where**
combine l k = *combine-aux* l (length l) k

For example, *combine* [1, 2, 3] 2 = [[1, 2], [1, 3], [2, 3]].

lemma *combine-aux-subset*:
shows $\forall A. A \in \text{set } (\text{combine-aux } l n k) \longrightarrow \text{set } A \subseteq \text{set } l$
 $\langle \text{proof} \rangle$

lemma *combine-subset*:
assumes $A \in \text{set } (\text{combine } l k)$
shows $\text{set } A \subseteq \text{set } l$
 $\langle \text{proof} \rangle$

lemma *combine-aux-sublist*:
shows $\forall A. A \subseteq \text{set } L \wedge n = \text{length } L \wedge m = \text{card } A \longrightarrow (\exists x \in \text{set } (\text{combine-aux } L n m). A = \text{set } x)$
 $\langle \text{proof} \rangle$

lemma *combine-sublist*:

assumes $A \subseteq \text{set } L$

shows $\exists x \in \text{set } (combine\ L\ (card\ A)).\ A = \text{set } x$

$\langle proof \rangle$

lemma *combine-aux-combines*:

assumes *sorted l and distinct l and $n = \text{length } l$*

shows $A \in \text{set } (combine_aux\ l\ n\ k) \longleftrightarrow \text{sorted } A \wedge \text{distinct } A \wedge \text{length } A = k \wedge \text{set } A \subseteq \text{set } l$

$\langle proof \rangle$

lemma *combine-combines*:

assumes

sorted l and distinct l

shows

$A \in \text{set } (combine\ l\ k) \longleftrightarrow (\text{sorted } A \wedge \text{distinct } A \wedge \text{length } A = k \wedge \text{set } A \subseteq \text{set } l)$

$\langle proof \rangle$

lemma *combine-aux-length*:

assumes $A \in \text{set } (combine_aux\ l\ n\ k)$ **and** $\text{length } l = n$

shows $\text{length } A = k$

$\langle proof \rangle$

lemma *combine-length*:

assumes $A \in \text{set } (combine\ l\ k)$

shows $\text{length } A = k$

$\langle proof \rangle$

lemma *distinct-combine-aux*:

assumes $n = \text{length } l$ **and** *distinct l*

shows *distinct (combine-aux l n k)*

$\langle proof \rangle$

lemma *distinct-combine*:

assumes *distinct l*

shows *distinct (combine l k)*

$\langle proof \rangle$

lemma *sorted-prepend*: $\text{sorted } l = \text{sorted } (\text{map } (op \ \# \ h) \ l)$

$\langle proof \rangle$

lemma *sorted-combine-aux-lemma*:

fixes $h :: 'a::linorder$

assumes $\forall x \in \text{set } t.\ h \leq x$ **and** $h \notin \text{set } t$ **and** $\text{set } l' \subseteq \text{set } t$ **and** $l' \neq []$

shows $h \ \# \ l \leq l'$

$\langle proof \rangle$

lemma *sorted-combine-aux*:
 assumes $n = \text{length } l$ and *sorted* l and *distinct* l
 shows *sorted* (*combine-aux* l n k)
 $\langle \text{proof} \rangle$

lemma *sorted-combine*:
 assumes *sorted* l and *distinct* l
 shows *sorted* (*combine* l k)
 $\langle \text{proof} \rangle$

1.3 Generating all nonempty subsets

definition *all-nonempty-subsets* **where**
 $\text{all-nonempty-subsets } F = \text{concat } (\text{map } (\lambda k. \text{combine } F k) [1..<\text{length } F + 1])$

lemma *all-sub-set*: $\text{set } (\text{all-nonempty-subsets } l) \subseteq \{A. \text{set } A \subseteq \text{set } l \wedge \text{length } A \geq 1\}$
 $\langle \text{proof} \rangle$

1.4 Generating all m-elements subsets of n-element set

definition *all-mn-subsets* **where**
 $\text{all-mn-subsets } n m = \text{combine } [0..<n] m$

lemma *all-mn-subsets*:
 $\text{set } (\text{map set } (\text{all-mn-subsets } n m)) = \{A. \text{card } A = m \wedge A \subseteq \{0..<n\}\}$ (**is** ?lhs
 = ?rhs)
 $\langle \text{proof} \rangle$

lemma *all-mn-subsets-completeness*:
 assumes $\text{card } A = m$ and $A \subseteq \{0..<n\}$
 shows $\exists Al \in \text{set } (\text{all-mn-subsets } n m). \text{set } Al = A$
 $\langle \text{proof} \rangle$

lemma *all-mn-subsets-sorted-distinct*:
 assumes $A \in \text{set } (\text{all-mn-subsets } n m)$
 shows *sorted* A and *distinct* A
 $\langle \text{proof} \rangle$

lemma [*simp*]: $A \in \text{set } (\text{all-mn-subsets } n m) \implies \text{set } A \subseteq \{0..<n\}$
 $\langle \text{proof} \rangle$

end

2 Representing lists of natural numbers by natural numbers

theory *ListNat*
imports *Main* *More.MoreNat* *More.MoreBigOperators* *More.MoreList*

begin

In this section, we define of representation of sets of natural numbers by single natural numbers. The set $\{a_0, a_1, \dots, a_n\}$ is represented by $2^{a_0} + 2^{a_1} + \dots + 2^{a_n}$.

2.1 sumpows

If the set is stored in a distinct list, the function *sumpows2* returns the natural number that represents it.

abbreviation *sumpows2* **where**

sumpows2 *l* \equiv *sum-list* (*map* (*op* \wedge (*2::nat*)) *l*)

lemma *sumpows2-mod*:

assumes *finite X*

shows (*sum* (*op* \wedge 2) *X*) *mod* 2 \neq (*0::nat*) \longleftrightarrow $0 \in X$

<proof>

lemma *unique-sumpows2*:

fixes *n::nat*

shows $\exists! X. \text{finite } X \wedge \text{sum } (\text{op } \wedge 2) X = n$

<proof>

lemma *sumpows2-inj*:

assumes *sorted l and distinct l and sorted l' and distinct l'*

sumpows2 l = sumpows2 l'

shows *l = l'*

<proof>

lemma *sumpows2-shift*:

shows $2 * \text{sumpows2 } l = \text{sumpows2 } (\text{map } (\text{op } + 1) l)$

<proof>

2.2 nat2list

The function *nat2list* deconstructs the natural number back to the set it represents (for executability, given by a sorted, distinct list).

fun *nat2list-aux* *:: nat \Rightarrow nat \Rightarrow nat list **where***

nat2list-aux n k =

(if n = 0 then

[]

else if n mod 2 = 0 then

nat2list-aux (n div 2) (k + 1)

else

k # nat2list-aux (n div 2) (k + 1))

definition *nat2list* **where**

$\text{nat2list } n \equiv \text{nat2list-aux } n \ 0$

lemma *sumpows2-nat2list-aux*:
 $\text{sumpows2 } (\text{nat2list-aux } n \ k) = n * 2^k$
 <proof>

lemma *sumpows2-nat2list*:
 $\text{sumpows2 } (\text{nat2list } n) = n$
 <proof>

lemma *nat2list-aux-empty*:
 $(\text{nat2list-aux } x \ k = []) = (x = 0)$
 <proof>

lemma *nat2list-aux-shift*:
 $\text{nat2list-aux } n \ (k + 1) = \text{map } (op + 1) (\text{nat2list-aux } n \ k)$
 <proof>

lemma *sorted-distinct-nat2list-aux*:
 $\text{sorted } (\text{nat2list-aux } n \ k) \wedge$
 $\text{distinct } (\text{nat2list-aux } n \ k) \wedge$
 $(\forall a \in \text{List.set } (\text{nat2list-aux } n \ k). a \geq k)$
 <proof>

lemma
sorted-nat2list: $\text{sorted } (\text{nat2list } n)$ **and**
distinct-nat2list: $\text{distinct } (\text{nat2list } n)$
 <proof>

lemma *sumpows2-nat2list-unique*:
assumes $\text{sumpows2 } l = n$ **and** $\text{sorted } l$ **and** $\text{distinct } l$
shows $l = \text{nat2list } n$
 <proof>

lemma *inj-nat2list-aux'*:
assumes $\text{nat2list-aux } x \ k = \text{nat2list-aux } y \ k$
shows $x = y$
 <proof>

lemma *inj-nat2list*:
 $\text{inj } \text{nat2list}$
 <proof>

lemma *nat2list-even*:
 $\text{nat2list } (2*n) = \text{map } (op + 1) (\text{nat2list } n)$
 <proof>

lemma *nat2list-odd*:
 $\text{nat2list } (2*n + 1) = 0 \ \# \ \text{map } (op + 1) (\text{nat2list } n)$

$\langle proof \rangle$

2.3 list2nat

Although the function *sumpows2* converts a set (given by a distinct list) to its representing natural number, we define the function *list2nat* that does the same, but is somewhat more efficient.

fun *list2nat-aux-measure* **where**

list2nat-aux-measure (*l*, *i*, *s*, *r*) =
 (case *l* of
 [] \Rightarrow 0
 | (*h* # *t*) \Rightarrow if *i* > *h* then 0 else Max (*set l*) - *i* + 1)

function *list2nat-aux* :: nat list \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow nat **where**

list2nat-aux [] *i* *s* *r* = *r*
 | *list2nat-aux* (*h* # *t*) *i* *s* *r* =
 (if *i* = *h* then
 list2nat-aux *t* (*i*+1) (2*s) (*s* + *r*)
 else if *i* < *h* then
 list2nat-aux (*h* # *t*) (*i*+1) (2*s) *r*
 else *r*)

$\langle proof \rangle$

termination

$\langle proof \rangle$

definition *list2nat* **where**

list2nat *l* = *list2nat-aux* *l* 0 1 0

lemma *list2nat-aux-sumpows2*:

assumes *s* = 2 ^ *i* **and** sorted *l* **and** distinct *l* **and**

l \neq [] \longrightarrow *i* \leq hd *l*

shows *list2nat-aux* *l* *i* *s* *r* = *r* + *sumpows2* *l*

$\langle proof \rangle$

lemma *list2nat-sumpows2*:

assumes sorted *l* **and** distinct *l*

shows *list2nat* *l* = *sumpows2* *l*

$\langle proof \rangle$

lemma *nat2list-list2nat*:

assumes distinct *l* **and** sorted *l*

shows nat2list (*list2nat* *l*) = *l*

$\langle proof \rangle$

lemma *inj-list2nat*:

assumes sorted *l1* **and** distinct *l1* **and** sorted *l2* **and** distinct *l2*

assumes *list2nat* *l1* = *list2nat* *l2*

shows *l1* = *l2*

$\langle proof \rangle$

end

3 Union closed families

theory *UnionClosed*
imports *Main*
begin

3.1 Union closed families

definition *sum-family* (**infixl** \uplus 100) **where**
 $A \uplus B = (\lambda (a, b). a \cup b) \text{ ' } \{(a, b) . a \in A \wedge b \in B\}$

definition *union-closed* :: '*a set set* \Rightarrow *bool* **where**
 $\text{union-closed } F \equiv \forall A B. A \in F \wedge B \in F \longrightarrow A \cup B \in F$

lemma *union-closed* $F \longleftrightarrow F \uplus F = F$
 $\langle \text{proof} \rangle$

abbreviation *finite-union-closed* **where**
 $\text{finite-union-closed } F \equiv \text{union-closed } F \wedge \text{finite } (\bigcup F)$

lemma *union-closed-n*:
assumes *finite* F' **and** $F' \neq \{\}$ **and** $F' \subseteq F$
assumes *union-closed* F
shows $\bigcup F' \in F$
 $\langle \text{proof} \rangle$

lemma *union-closed-insert*:
assumes *union-closed* F **and** $\forall B \in F. A \cup B \in F \cup \{A\}$
shows *union-closed* $(F \cup \{A\})$
 $\langle \text{proof} \rangle$

3.1.1 Closure – minimal union closed family containing F

definition *closure* **where**
 $\text{closure } F = \text{Union ' } (\text{Pow } F - \{\{\}\})$

lemma *closure-closed*:
union-closed $(\text{closure } F)$
 $\langle \text{proof} \rangle$

lemma *closure-min-closed*:
assumes *finite* F **and** $F \subseteq F'$ **and** *union-closed* F'
shows $\text{closure } F \subseteq F'$
 $\langle \text{proof} \rangle$

lemma *closure-union-closed-id*:

assumes *finite F and union-closed F*
shows $\text{closure } F = F$
 $\langle \text{proof} \rangle$

lemma [*simp*]:
shows $\bigcup (\text{closure } F) = \bigcup F$
 $\langle \text{proof} \rangle$

lemma *finite-closure*:
assumes *finite ($\bigcup F$)*
shows *finite ($\bigcup (\text{closure } F)$)*
 $\langle \text{proof} \rangle$

lemma *closure-mono*:
assumes $F \subseteq F'$
shows $\text{closure } F \subseteq \text{closure } F'$
 $\langle \text{proof} \rangle$

Iterative closure: insert set to closed family and close.

abbreviation *insert-and-close* :: *'a set \Rightarrow 'a set set \Rightarrow 'a set set* **where**
insert-and-close A F $\equiv F \cup \{A\} \cup (\text{op} \cup A) \text{ ' } F$

lemma *union-closed-insert-and-close*:
assumes *union-closed F*
shows *union-closed (insert-and-close A F)*
 $\langle \text{proof} \rangle$

lemma *closure-insert*:
assumes *finite F*
shows $\text{closure } (F \cup \{A\}) = \text{insert-and-close } A (\text{closure } F)$ (**is** ?lhs = ?rhs)
 $\langle \text{proof} \rangle$

lemma *closure-subset*:
shows $F \subseteq \text{closure } F$
 $\langle \text{proof} \rangle$

lemma *closure-empty*:
shows $\{\} \in \text{closure } F \longleftrightarrow \{\} \in F$
 $\langle \text{proof} \rangle$

lemma *insert-and-close-empty*:
shows $\text{insert-and-close } \{\} F = F \cup \{\{\}\}$
 $\langle \text{proof} \rangle$

lemma *closure-remove-empty*:
assumes *finite F*
shows $\text{closure } (F - \{\{\}\}) = \text{closure } F - \{\{\}\}$
 $\langle \text{proof} \rangle$

lemma *insert-and-close-closure*:

assumes *finite F and union-closed F*

shows $\text{closure } (F \cup \{A\}) = \text{insert-and-close } A \ F$ (**is** ?lhs = ?rhs)

$\langle \text{proof} \rangle$

3.2 Union closed families closed to unions with an additional family

definition *union-closed-additional* **where**

[simp]: $\text{union-closed-additional } F \ Fc \equiv$

$\text{union-closed } F \wedge (\forall A \in F. (op \cup A) ' Fc \subseteq F)$

lemma

shows $\text{union-closed-additional } F \ Fc \longleftrightarrow F \uplus F = F \wedge F \uplus Fc \subseteq F$

$\langle \text{proof} \rangle$

lemma *union-closed-additional*:

shows $(\forall A \in F. \forall Ai \in Fc. A \cup Ai \in F) \longleftrightarrow$

$(\forall A \in F. (op \cup A) ' Fc \subseteq F)$

$\langle \text{proof} \rangle$

abbreviation *insert-and-close-additional* **where**

$\text{insert-and-close-additional } A \ F \ Fc \equiv F \cup \{A\} \cup ((op \cup A) ' F) \cup ((op \cup A) ' Fc)$

lemma *closure-additional-set*:

assumes *finite F and A ∈ closure F and*

A' ∈ F' and union-closed F' and $\forall A' \in F'. op \cup A' ' F \subseteq F'$

shows $A \cup A' \in F'$

$\langle \text{proof} \rangle$

lemma *union-closed-additional-closure*:

assumes *finite Fc and union-closed-additional F Fc*

shows $\text{union-closed-additional } F \ (\text{closure } Fc)$

$\langle \text{proof} \rangle$

3.2.1 Union closed extensions

definition *union-closed-extensions* **where**

[simp]: $\text{union-closed-extensions } Fc \equiv$

$\{F. F \subseteq \text{Pow } (\bigcup Fc) \wedge \text{union-closed-additional } F \ Fc\}$

syntax

-union-closed-extensions :: 'a set set \Rightarrow 'a set set set ($\{\{-\}\}$)

translations

$\{\{Fc\}\} == \text{CONST union-closed-extensions } Fc$

end

4 Families of sets

```
theory Family
imports Main
begin
```

```
definition count :: 'a  $\Rightarrow$  'a set set  $\Rightarrow$  nat where
  count a F  $\equiv$  card {S  $\in$  F. a  $\in$  S}
```

```
lemma count-Un-disjoint:
  assumes A  $\cap$  B = {} and finite A and finite B
  shows count a (A  $\cup$  B) = count a A + count a B
  <proof>
```

```
end
```

5 Frankl's condition

```
theory Frankl
imports Family UnionClosed More.MoreSet
begin
```

5.1 Frankl's condition

Note that, in order to avoid using rational numbers, the following condition is not formulated as $\text{count } x \ F \geq (\text{card } F) / 2$, what would be more expected.

```
abbreviation frankl-element :: 'a  $\Rightarrow$  'a set set  $\Rightarrow$  bool where
  frankl-element x F  $\equiv$   $x \in \bigcup F \wedge 2 * \text{count } x \ F \geq \text{card } F$ 
```

```
definition frankl :: 'a set set  $\Rightarrow$  bool where
  frankl F  $\equiv$  ( $\exists x. \text{frankl-element } x \ F$ )
```

```
lemma obtain-frankl-element:
  assumes frankl F
  obtains x where  $x \in \bigcup F$  and  $2 * \text{count } x \ F \geq \text{card } F$ 
  <proof>
```

5.1.1 Frankl's function

```
abbreviation frankl-fun :: 'a  $\Rightarrow$  'a set set  $\Rightarrow$  int where
  frankl-fun x F  $\equiv$   $2 * \text{int } (\text{count } x \ F) - \text{int } (\text{card } F)$ 
```

```
lemma
  shows frankl-element x F  $\longleftrightarrow$  ( $x \in \bigcup F \wedge \text{frankl-fun } x \ F \geq 0$ )
  <proof>
```

```
lemma frankl-fun-Un-disjoint:
  assumes A  $\cap$  B = {} and finite A and finite B
```

shows $\text{frankl-fun } a \ (A \cup B) = \text{frankl-fun } a \ A + \text{frankl-fun } a \ B$
 $\langle \text{proof} \rangle$

lemma *frankl-fun-Un-disjoint-3*:
assumes *finite A and finite B and finite C*
 $A \cap B = \{\}$ **and** $A \cap C = \{\}$ **and** $B \cap C = \{\}$
shows $\text{frankl-fun } i \ (A \cup B \cup C) =$
 $\text{frankl-fun } i \ A + \text{frankl-fun } i \ B + \text{frankl-fun } i \ C$
 $\langle \text{proof} \rangle$

lemma *frankl-fun-UN-disjoint*:
assumes *finite $(\bigcup \text{ set } l)$ and*
 $\forall i \ j. \ i < \text{length } l \wedge j < \text{length } l \wedge i \neq j \longrightarrow l ! i \cap l ! j = \{\}$
shows $\text{frankl-fun } i \ (\bigcup \text{ set } l) = \text{sum-list } (\text{map } (\text{frankl-fun } i) \ l)$
 $\langle \text{proof} \rangle$

lemma *frankl-fun-Pow*:
assumes *finite A and $a \in A$*
shows $\text{frankl-fun } a \ (\text{Pow } A) = 0$
 $\langle \text{proof} \rangle$

5.2 FC Families

definition *FC-family where*
 $\text{FC-family } Fc \equiv$
 $\forall F. \ F \supseteq Fc \wedge \text{finite-union-closed } F \longrightarrow$
 $(\exists a \in \bigcup Fc. \ 2 * \text{count } a \ F \geq \text{card } F)$

lemma *FC-family-frankl*:
assumes *FC-family Fc and $Fc \subseteq F$ and finite-union-closed F*
shows $\text{frankl } F$
 $\langle \text{proof} \rangle$

lemma *FC-family-mono*:
assumes $Fc \subseteq Fc'$ **and** *FC-family Fc*
shows *FC-family Fc'*
 $\langle \text{proof} \rangle$

lemma *FC-family-empty-set-remove*:
shows $\text{FC-family } Fc \longleftrightarrow \text{FC-family } (Fc - \{\{\}\})$
 $\langle \text{proof} \rangle$

lemma *FC-family-empty-set-insert*:
shows $\text{FC-family } (Fc \cup \{\{\}\}) \longleftrightarrow \text{FC-family } Fc$
 $\langle \text{proof} \rangle$

lemma *FC-family-closure*:
assumes *finite Fc*
shows $\text{FC-family } Fc \longleftrightarrow \text{FC-family } (\text{closure } Fc)$

$\langle proof \rangle$

end

6 Executable implementations of set families

6.1 Abstract representation of sets and families

```
theory FamilyImpl
imports Family
        Frankl
        More.MoreSet ListNat
        HOL-Library.List-lexord
        HOL-Library.Code-Target-Nat
begin
```

abbreviation *map* **where** *map* \equiv *List.map*

Set operations are specified in the following locale. Families are always represented as (distinct) lists of sets.

```
locale SetImpl =
  fixes to-set :: 's  $\Rightarrow$  'a set
  fixes inv :: 's  $\Rightarrow$  bool
  assumes to-set-inj:  $\llbracket \text{inv } s1; \text{inv } s2; \text{to-set } s1 = \text{to-set } s2 \rrbracket \Longrightarrow s1 = s2$ 
  assumes to-set-ex:  $\text{finite } a \Longrightarrow \exists s. a = \text{to-set } s \wedge \text{inv } s$ 
  assumes to-set-finite: finite (to-set s)
begin
```

definition *f-to-set* :: 's list \Rightarrow 'a set set **where**
[*simp*]: *f-to-set* *F* = *set* (*map to-set F*)

abbreviation *fs-to-set* :: 's list list \Rightarrow 'a set set set (\circ) **where**
 $\circ FF \equiv \text{set } (\text{map } f\text{-to-set } FF)$

lemma *to-set-inj-on*:
shows $\forall a \in \text{set } A. \text{inv } a \Longrightarrow \text{inj-on } \text{to-set } (\text{set } A)$
 $\langle proof \rangle$

lemma *set-set*:
 $\llbracket \forall a \in \text{set } A. \text{inv } a; \text{inv } h \rrbracket \Longrightarrow$
 $h \in \text{set } A \longleftrightarrow \text{to-set } h \in f\text{-to-set } A$
 $\langle proof \rangle$

lemma *f-to-set-ex*:
assumes *finite* ($\bigcup F$)
shows $\exists Fl. F = f\text{-to-set } Fl \wedge (\forall A \in \text{set } Fl. \text{inv } A)$
 $\langle proof \rangle$

abbreviation *contains* :: 'a \Rightarrow 's \Rightarrow bool **where**

contains a S $\equiv a \in \text{to-set } S$

definition *count* :: 'a \Rightarrow 's list \Rightarrow nat **where**
count a F = length (filter (contains a) F)

lemma *count-set*:
assumes distinct F $\forall x \in \text{set } F. \text{inv } x$
shows count a F = Family.count a (f-to-set F)
 <proof>

definition *frankl-fun* :: 'a \Rightarrow 's list \Rightarrow int **where**
frankl-fun x F $\equiv 2 * \text{int } (\text{count } x F) - \text{int } (\text{length } F)$

lemma *frankl-fun-set*:
assumes distinct Fs $\forall A \in \text{set } Fs. \text{inv } A$
shows frankl-fun a Fs = Frankl.frankl-fun a (f-to-set Fs)
 <proof>

end

6.1.1 Implementation of sets by sorted and distinct lists

abbreviation *sd* **where**
sd A $\equiv \text{sorted } A \wedge \text{distinct } A$

abbreviation *sdf* **where**
sdf F $\equiv \forall A \in \text{set } F. \text{sd } A$

abbreviation *sdff* **where**
sdff F $\equiv \forall F \in \text{set } \mathcal{F}. \text{sdf } F$

For example, the family $\{\{1, 2, 3\}, \{2, 3, 4\}\}$ is represented by $[[1, 2, 3], [2, 3, 4]]$.

global-interpretation *SetImpl-lists*: SetImpl set :: nat list \Rightarrow nat set *sd*
defines f-to-set-l = SetImpl-lists.f-to-set **and**
 count-l = SetImpl-lists.count **and**
 frankl-fun-l = SetImpl-lists.frankl-fun
 <proof>

abbreviation *dm* **where**
dm F n $\equiv \bigcup \text{f-to-set-l } F \subseteq \{0..<n\}$

abbreviation *dmf* **where**
dmf FF n $\equiv \forall F \in \text{set } FF. \text{dm } F n$

abbreviation *fs-to-set-l* **where**
fs-to-set-l F $\equiv \text{set } (\text{map f-to-set-l } F)$

6.1.2 Implementation of sets of nats by nats

For example, the family $\{\{1, 2, 3\}, \{2, 3, 4\}\}$ is represented by [14, 28]. Encoding and decoding functions are defined in the theory ListNat.

global-interpretation *SetImpl-nats*: *SetImpl set* \circ *nat2list* λ *n*. *True*

defines *f-to-set-n* = *SetImpl-nats.f-to-set* **and**

count-n = *SetImpl-nats.count* **and**

frankl-fun-n = *SetImpl-nats.frankl-fun*

\langle *proof* \rangle

\langle *ML* \rangle

end

theory *UnionClosedImpl*

imports *UnionClosed* *FamilyImpl*

begin

6.2 Abstract representation of union closed families

Sets that support empty set, unions and powerset.

locale *SetUnionImpl* = *SetImpl to-set* **for** *to-set* $:: 's \Rightarrow 'a \text{ set} +$

fixes *empty* $:: 's$

assumes *empty-set*: *to-set empty* = $\{\}$

assumes *empty-inv*: *inv empty*

fixes *union* $:: 's \Rightarrow 's \Rightarrow 's$ (**infixl** \sqcup 100)

assumes *union-set*: *to-set* (*s1* \sqcup *s2*) = *to-set s1* \cup *to-set s2*

assumes *union-inv*: $\llbracket \text{inv } s1; \text{inv } s2 \rrbracket \Longrightarrow \text{inv } (s1 \sqcup s2)$

fixes *pow* $:: 's \Rightarrow 's \text{ list}$

assumes *pow-set*: *f-to-set* (*pow s*) = *Pow* (*to-set s*)

assumes *pow-inv*: *inv s* $\Longrightarrow \forall A \in \text{set } (\text{pow } s). \text{inv } A$

assumes *pow-distinct*: *inv s* $\Longrightarrow \text{distinct } (\text{pow } s)$

begin

lemma *subset-in-pow*:

assumes *inv A* **and** *inv S* **and** *to-set A* \subseteq *to-set S*

shows *A* $\in \text{set } (\text{pow } S)$

\langle *proof* \rangle

definition *Union* $:: 's \text{ list} \Rightarrow 's$ **where**

Union F = *foldl union empty F*

lemma *Union-set*:

shows $to\text{-}set\ (Union\ F) = \bigcup\ (f\text{-}to\text{-}set\ F)$
 $\langle proof \rangle$

lemma *Union-inv*:
shows $\forall\ A \in set\ F.\ inv\ A \implies inv\ (Union\ F)$
 $\langle proof \rangle$

abbreviation *union-with-all* $:: 's \Rightarrow 's\ list \Rightarrow 's\ list$ **where**
 $union\text{-}with\text{-}all\ A\ F \equiv (map\ (op\ \sqcup\ A)\ F)$

lemma *union-with-all-set*:
shows $map\ to\text{-}set\ (union\text{-}with\text{-}all\ A\ F) = map\ (op\ \cup\ (to\text{-}set\ A) \circ to\text{-}set)\ F$
 $\langle proof \rangle$

lemma *union-with-all-set'*:
shows $(to\text{-}set\ ' (op\ \sqcup\ A\ ' set\ F)) = (op\ \cup\ (to\text{-}set\ A)\ ' to\text{-}set\ ' set\ F)$
 $\langle proof \rangle$

definition *insert-set* $:: 's \Rightarrow 's\ list \Rightarrow 's\ list$ **where**
 $insert\text{-}set\ A\ F \equiv (if\ A \in set\ F\ then\ F\ else\ A\ \#\ F)$

lemma *insert-set-set [simp]*:
shows $set\ (insert\text{-}set\ A\ F) = \{A\} \cup set\ F$
 $\langle proof \rangle$

definition *insert-sets* $:: 's\ list \Rightarrow 's\ list \Rightarrow 's\ list$ **where**
 $insert\text{-}sets\ l\ F = foldl\ (\lambda\ F'\ a.\ insert\text{-}set\ a\ F')\ F\ l$

lemma *insert-sets-set [simp]*:
shows $set\ (insert\text{-}sets\ l\ F) = set\ l \cup set\ F$
 $\langle proof \rangle$

lemma *insert-sets-remdups*:
shows $distinct\ F \implies insert\text{-}sets\ l\ F = remdups\ (rev\ l\ @\ F)$
 $\langle proof \rangle$

lemma *insert-sets-distinct*:
shows $distinct\ F \implies distinct\ (insert\text{-}sets\ l\ F)$
 $\langle proof \rangle$

definition *insert-and-close* $:: 's \Rightarrow 's\ list \Rightarrow 's\ list$ **where**
 $[simp]: insert\text{-}and\text{-}close\ A\ F \equiv insert\text{-}sets\ ([A]\ @\ union\text{-}with\text{-}all\ A\ F)\ F$

lemma *insert-and-close-set*:

assumes *distinct F*

shows $f\text{-to-set } (\text{insert-and-close } h \ F) = \text{UnionClosed.insert-and-close } (\text{to-set } h)$
 $(f\text{-to-set } F)$
 $\langle \text{proof} \rangle$

lemma *insert-and-close-inv*:

assumes $\forall l \in \text{set } F. \text{inv } l$ **and** $\text{inv } h$ **and** *distinct F*

shows $\forall l \in \text{set } (\text{insert-and-close } h \ F). \text{inv } l$
 $\langle \text{proof} \rangle$

lemma *insert-and-close-subset*:

assumes $\forall l \in \text{set } F. \text{to-set } l \subseteq S$ **and** $\text{to-set } h \subseteq S$ **and** *distinct F*

shows $\forall l \in \text{set } (\text{insert-and-close } h \ F). \text{to-set } l \subseteq S$
 $\langle \text{proof} \rangle$

definition *insert-and-close-additional* :: $'s \Rightarrow 's \text{ list} \Rightarrow 's \text{ list} \Rightarrow 's \text{ list}$ **where**

$[\text{simp}]$: $\text{insert-and-close-additional } A \ F \ Fc \equiv \text{insert-sets } ([A] \ @ \ \text{union-with-all } A \ F \ @ \ \text{union-with-all } A \ Fc)$

lemma *insert-and-close-additional-set*:

assumes *distinct F*

shows $f\text{-to-set } (\text{insert-and-close-additional } A \ F \ Fc) = \text{UnionClosed.insert-and-close-additional}$
 $(\text{to-set } A) \ (f\text{-to-set } F) \ (f\text{-to-set } Fc)$
 $\langle \text{proof} \rangle$

lemma *insert-and-close-additional-subset*:

assumes $\text{to-set } A \subseteq S$ **and**

$\forall l \in \text{set } F. \text{to-set } l \subseteq S$ **and**

$\forall l \in \text{set } Fc. \text{to-set } l \subseteq S$

assumes *distinct F*

shows $\forall l \in \text{set } (\text{insert-and-close-additional } A \ F \ Fc). \text{to-set } l \subseteq S$
 $\langle \text{proof} \rangle$

lemma *insert-and-close-additional-inv*:

assumes $\text{inv } A$ **and** $\forall l \in \text{set } F. \text{inv } l$ **and** $\forall l \in \text{set } Fc. \text{inv } l$

assumes *distinct F*

shows $\forall l \in \text{set } (\text{insert-and-close-additional } A \ F \ Fc). \text{inv } l$
 $\langle \text{proof} \rangle$

lemma *insert-and-close-additional-cong*:

assumes $\text{set } F = \text{set } F'$ **and** *distinct F* **and** *distinct F'*

shows $\text{set } (\text{insert-and-close-additional } A \ F \ Fc) =$

$\text{set } (\text{insert-and-close-additional } A \ F' \ Fc)$

$\langle \text{proof} \rangle$

definition $close :: 's\ list \Rightarrow 's\ list$ **where**
 $close\ F = foldl\ (\lambda\ F\ A.\ insert\ and\ close\ A\ F)\ []\ F$

lemma $close\ snoc$:
shows $close\ (F\ @\ [A]) = insert\ and\ close\ A\ (close\ F)$
 $\langle proof \rangle$

lemma $close\ distinct$:
shows $distinct\ (close\ A)$
 $\langle proof \rangle$

lemma $close\ set$:
shows $f\text{-}to\text{-}set\ (close\ A) = UnionClosed.closure\ (f\text{-}to\text{-}set\ A)$
 $\langle proof \rangle$

lemma $close\ completeness$:
assumes $inv\ A$ **and** $\forall\ a \in set\ F.\ inv\ a$
assumes $to\text{-}set\ A \in UnionClosed.closure\ (f\text{-}to\text{-}set\ F)$
shows $A \in set\ (close\ F)$
 $\langle proof \rangle$

lemma $close\ inv$:
assumes $\forall\ l \in set\ F.\ inv\ l$
shows $\forall\ l \in set\ (close\ F).\ inv\ l$
 $\langle proof \rangle$

lemma $close\ subset$:
assumes $\forall\ l \in set\ F.\ to\text{-}set\ l \subseteq S$
shows $\forall\ l \in set\ (close\ F).\ to\text{-}set\ l \subseteq S$
 $\langle proof \rangle$

definition $close\ and\ insert\ empty :: 's\ list \Rightarrow 's\ list$ **where**
 $close\ and\ insert\ empty\ F \equiv$
 $let\ X = close\ F$
 $in\ (if\ empty \in set\ X\ then\ X\ else\ empty\ \# X)$

lemma $close\ and\ insert\ empty\ set'$:
shows $set\ (close\ and\ insert\ empty\ F) = set\ (close\ F) \cup \{empty\}$
 $\langle proof \rangle$

lemma $close\ and\ insert\ empty\ set$:
shows $f\text{-}to\text{-}set\ (close\ and\ insert\ empty\ F) =$
 $(UnionClosed.closure\ (f\text{-}to\text{-}set\ F)) \cup \{\{\}\}$
 $\langle proof \rangle$

primrec $union\ closed :: 's\ list \Rightarrow bool$ **where**
 $union\ closed\ [] = True$
 $| union\ closed\ (h\ \# t) = (list\ all\ (\lambda\ x.\ h\sqcup x \in set\ (h\ \# t))\ t) \wedge union\ closed\ t$

lemma *union-closed-set*:

shows $\text{union-closed } F \implies \text{UnionClosed.union-closed } (f\text{-to-set } F)$
 $\langle \text{proof} \rangle$

primrec *union-closed-additional'* :: 's list \Rightarrow 's list \Rightarrow bool **where**

$\text{union-closed-additional}' F [] = \text{True}$
 $| \text{union-closed-additional}' F (h \# t) = (\text{list-all } (\lambda x. h \sqcup x \in \text{set } F) F) \wedge$
 $\text{union-closed-additional}' F t)$

lemma *union-closed-additional'-set*:

assumes $\text{union-closed-additional}' F Fc$
shows $\forall A \in f\text{-to-set } Fc. \forall B \in (f\text{-to-set } F). A \cup B \in (f\text{-to-set } F)$
 $\langle \text{proof} \rangle$

definition *union-closed-additional* :: 's list \Rightarrow 's list \Rightarrow bool **where**

$\text{union-closed-additional } F Fc \longleftrightarrow \text{union-closed } F \wedge \text{union-closed-additional}' F Fc$

lemma *union-closed-additional-set*:

assumes $\text{union-closed-additional}' F Fc$
shows $\text{UnionClosed.union-closed-additional } (f\text{-to-set } F) (f\text{-to-set } Fc)$
 $\langle \text{proof} \rangle$

end

6.2.1 Implementation by sorted and distinct lists

Power set of a list

primrec *Pow-l* :: 'a list \Rightarrow 'a list list **where**

$\text{Pow-l } [] = [[]]$
 $| \text{Pow-l } (h \# t) =$
 $(\text{let } X = \text{Pow-l } t$
 $\text{in } X @ \text{map } (op \# h) X)$

lemma *Pow-l-set*:

shows $\text{set } (\text{map set } (\text{Pow-l } X)) = \text{Pow } (\text{set } X)$
 $\langle \text{proof} \rangle$

lemma *Pow-l-distinct*:

shows $\text{distinct } A \implies \text{distinct } (\text{Pow-l } A)$
 $\langle \text{proof} \rangle$

lemma *Pow-l-elems-distinct*:

assumes $\text{distinct } A$ **and** $x \in \text{set } (\text{Pow-l } A)$
shows $\text{distinct } x$
 $\langle \text{proof} \rangle$

lemma *Pow-l-elems-sorted*:

```

assumes sorted A and  $x \in \text{set } (\text{Pow-}l \ A)$ 
shows sorted x
<proof>

```

```

lemma Pow-l-no-perms:
assumes distinct A and sorted A
assumes  $x \in \text{set } (\text{Pow-}l \ A)$  and  $y \in \text{set } (\text{Pow-}l \ A)$ 
assumes  $\text{set } x = \text{set } y$ 
shows  $x = y$ 
<proof>

```

Merging two sorted lists

```

fun merge :: 'a::linorder list  $\Rightarrow$  'a::linorder list  $\Rightarrow$  'a::linorder list (infixl  $\sqcup$  100)
where
  merge [] l = l
| merge l [] = l
| merge (h1 # t1) (h2 # t2) =
  (if h1 < h2 then
    h1 # merge t1 (h2 # t2)
  else if h1 > h2 then
    h2 # merge (h1 # t1) t2
  else
    h1 # merge t1 t2)

```

```

lemma merge-set:
shows  $\text{set } (l1 \sqcup l2) = \text{set } l1 \cup \text{set } l2$ 
<proof>

```

```

lemma merge-sorted:
assumes sorted l1 and sorted l2
shows sorted ( $l1 \sqcup l2$ )
<proof>

```

```

lemma merge-distinct:
assumes distinct l1 and distinct l2 and sorted l1 and sorted l2
shows distinct ( $l1 \sqcup l2$ )
<proof>

```

```

global-interpretation SetUnionImpl-lists: SetUnionImpl  $\lambda$  (l::nat list). sorted l
 $\wedge$  distinct l set [] merge Pow-l

```

```

defines
close-l = SetUnionImpl-lists.close and
insert-set-l = SetUnionImpl-lists.insert-set and
insert-sets-l = SetUnionImpl-lists.insert-sets and
insert-and-close-l = SetUnionImpl-lists.insert-and-close and
Union-l = SetUnionImpl-lists.Union and
close-insert-empty-l = SetUnionImpl-lists.close-and-insert-empty and
union-closed-l = SetUnionImpl-lists.union-closed and
union-closed-additional'-l = SetUnionImpl-lists.union-closed-additional' and

```

union-closed-additional-l = *SetUnionImpl-lists.union-closed-additional*
 <proof>

6.2.2 Implementation by sets represented as natural numbers

For example, the family $\{\{0, 1, 2\}, \{1, 2, 3\}\}$ is represented as $[7, 14]$.

Powerset

definition *pow-n* :: *nat* \Rightarrow *nat list* **where**
pow-n *n* = *map list2nat (Pow-l (nat2list n))*

Union is obtained by bitwise disjunction – this is a naive implementation

function *bitor* :: *nat* \Rightarrow *nat* \Rightarrow *nat* **where**

bitor *x* *y* =
 (if *x* = 0 then *y*
 else if *y* = 0 then *x* else
 let (*xd*, *xm*) = *Divides.divmod-nat* *x* 2;
 (*yd*, *ym*) = *Divides.divmod-nat* *y* 2 in
 if (*xm* = 1 | *ym* = 1)
 then 1 + 2 * *bitor* *xd* *yd*
 else 2 * *bitor* *xd* *yd*)

<proof>

termination

<proof>

declare *bitor.simps*[*simp del*]

lemma *bitor-set*:

shows *set* (*nat2list* (*bitor* *x* *y*)) = *set* (*nat2list* *x*) \cup *set* (*nat2list* *y*)

<proof>

global-interpretation *SetUnionImpl-nats*: *SetUnionImpl* λ *n*. *True set* \circ *nat2list*
 0 *bitor* *pow-n*

defines

close-n = *SetUnionImpl-nats.close* **and**
insert-set-n = *SetUnionImpl-nats.insert-set* **and**
insert-sets-n = *SetUnionImpl-nats.insert-sets* **and**
insert-and-close-n = *SetUnionImpl-nats.insert-and-close* **and**
Union-n = *SetUnionImpl-nats.Union* **and**
close-and-insert-empty-n = *SetUnionImpl-nats.close-and-insert-empty* **and**
union-closed-n = *SetUnionImpl-nats.union-closed* **and**
union-closed-additional'-n = *SetUnionImpl-nats.union-closed-additional'* **and**
union-closed-additional-n = *SetUnionImpl-nats.union-closed-additional*

<proof>

end

7 Weights and shares

```
theory WeightsShares
imports Main Frankl
begin
```

7.1 Weight functions

A technique, introduced by Poonen, used for analyzing Frankl's conjecture is based on the concept of weights and shares.

definition *weight-fun* :: $('a \Rightarrow 'b :: \{\text{zero}, \text{ord}\}) \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ **where**
 $\text{weight-fun } w \ X \equiv (\forall a \in X. w \ a \geq 0) \wedge (\exists a \in X. w \ a > 0)$

definition *set-weight* :: $('a \Rightarrow 'b :: \{\text{comm-monoid-add}\}) \Rightarrow 'a \text{ set} \Rightarrow 'b$ (**infixl** \triangleright_s 100) **where**
 $w \triangleright_s S = (\sum_{x \in S} w \ x)$

definition *family-weight* :: $('a \Rightarrow 'b :: \{\text{comm-monoid-add}\}) \Rightarrow 'a \text{ set set} \Rightarrow 'b$ (**infixl** \triangleright_f 100) **where**
 $w \triangleright_f F = (\sum_{S \in F} w \triangleright_s S)$

lemma *set-weight-cong*:
assumes $\forall v \in S. w \ v = w' \ v$
shows $w \triangleright_s S = w' \triangleright_s S$
 $\langle \text{proof} \rangle$

lemma *family-weight-lemma*:
assumes *finite* $(\bigcup F)$
shows $w \triangleright_f F = (\sum_{a \in \bigcup F} (w \ a) * \text{count } a \ F)$
 $\langle \text{proof} \rangle$

lemma *sum-card-reorder*:
assumes *finite* $(\bigcup F)$
shows $(\sum_{A \in F} \text{card } A) = (\sum_{a \in \bigcup F} \text{count } a \ F)$
 $\langle \text{proof} \rangle$

7.2 Shares

definition *set-share* :: $'a \text{ set} \Rightarrow ('a \Rightarrow \text{nat}) \Rightarrow 'a \text{ set} \Rightarrow \text{int}$ **where**
 $\text{set-share } S \ w \ X \equiv 2 * \text{int } (w \triangleright_s S) - \text{int } (w \triangleright_s X)$

syntax
 $\text{-set-share} :: 'a \text{ set} \Rightarrow ('a \Rightarrow \text{nat}) \Rightarrow 'a \text{ set} \Rightarrow \text{int} \quad (- \bowtie_s -)$

translations
 $w \bowtie_s S \ X == \text{CONST } \text{set-share } S \ w \ X$

definition *family-share* :: $'a \text{ set set} \Rightarrow ('a \Rightarrow \text{nat}) \Rightarrow 'a \text{ set} \Rightarrow \text{int}$ **where**
 $\text{family-share } F \ w \ X \equiv (\sum S \in F. (w \bowtie_s S \ X))$

syntax
 $\text{-family-share} :: 'a \text{ set} \Rightarrow ('a \Rightarrow \text{nat}) \Rightarrow 'a \text{ set} \Rightarrow \text{int} \quad (- \bowtie_f -)$

translations

$w \bowtie_f F X == \text{CONST family-share } F w X$

lemma family-share-cong:

assumes $\forall v \in X. w v = w' v \cup F \subseteq X$

shows $(w \bowtie_f F X) = (w' \bowtie_f F X)$

$\langle \text{proof} \rangle$

lemma family-share-Pow:

assumes *finite* $(\bigcup F)$

shows $(w \bowtie_f (\text{Pow } (\bigcup F)) (\bigcup F)) = 0$

$\langle \text{proof} \rangle$

lemma family-share-lemma:

shows $(w \bowtie_f F X) = \text{int } (2 * w \triangleright_f F) - \text{int } (w \triangleright_s X * (\text{card } F))$

$\langle \text{proof} \rangle$

7.3 Hypercube construction

definition hypercube $:: 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set set}$ **where**

$\text{hypercube } K S \equiv \{L. (K \subseteq L) \wedge L \subseteq (K \cup S)\}$

syntax

$\text{-hypercube} :: 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set set} \quad (\langle -, - \rangle)$

translations

$\langle K, S \rangle == \text{CONST hypercube } K S$

lemma hypercube-Pow: $\langle K, S \rangle = (\text{op } \cup K) \text{ ' Pow } S$ **(is ?H = ?KP)**

$\langle \text{proof} \rangle$

lemma hypercube-inter:

assumes $K1 \neq K2$ **and** $K1 \cap S = \{\}$ **and** $K2 \cap S = \{\}$

shows $\langle K1, S \rangle \cap \langle K2, S \rangle = \{\}$

$\langle \text{proof} \rangle$

lemma hypercube-UN-Pow:

shows $\bigcup ((\lambda K. \langle K, S \rangle) \text{ ' (Pow } K)) = \text{Pow } (K \cup S)$ **(is ?lhs = ?rhs)**

$\langle \text{proof} \rangle$

definition hyper-share **where**

$\text{hyper-share } K S F w X \equiv (\sum L \in \langle K, S \rangle \cap F. (w \bowtie_s L X))$

syntax

$\text{-hyper-share} :: ('a \Rightarrow \text{nat}) \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set set} \Rightarrow 'a \text{ set} \Rightarrow \text{int} \quad (- \odot_f - - -)$

translations

$w \odot_f K S F X == \text{CONST hyper-share } K S F w X$

lemma family-share-hyper-share:

assumes *finite* $(\bigcup F)$

assumes $K' \cup S = (\bigcup F)$ **and** $K' \cap S = \{\}$

shows $(w \bowtie_f F (\bigcup F)) = (\sum K \in Pow K'. (w \odot_f K S F (\bigcup F)))$
 $\langle proof \rangle$

end

7.4 Frankl's families characterization using weights

theory *WeightsShares-Frankl*
imports *Main WeightsShares*
More.MoreBigOperators
begin

lemma *frankl-weight'*:

fixes $w :: 'a \Rightarrow nat$
assumes $F \neq \{\}$ **and** *finite* $(\bigcup F)$ **and** *weight-fun* $w (\bigcup F)$ **and**
 $2 * (w \triangleright_f F) \geq (w \triangleright_s (\bigcup F)) * card F$
shows $\exists x. frankl\text{-}element\ x\ F \wedge w\ x \neq 0$
 $\langle proof \rangle$

lemma *frankl-weight''*:

assumes *frankl* F **and** $F \neq \{\}$ **and** *finite* $(\bigcup F)$
shows $\exists w :: ('a \Rightarrow nat). weight\text{-}fun\ w (\bigcup F) \wedge$
 $2 * (w \triangleright_f F) \geq (w \triangleright_s (\bigcup F)) * card F$
 $\langle proof \rangle$

theorem *frankl-weight*:

assumes $F \neq \{\}$ **and** *finite* $(\bigcup F)$
shows *frankl* $F \longleftrightarrow$
 $(\exists w :: 'a \Rightarrow nat. weight\text{-}fun\ w (\bigcup F) \wedge$
 $2 * (w \triangleright_f F) \geq (w \triangleright_s (\bigcup F)) * card F) \text{ (is ?lhs } \longleftrightarrow \text{ ?rhs)}$
 $\langle proof \rangle$

7.5 Frankl's families characterization using shares

lemma *frankl-share'*:

assumes $F \neq \{\}$ **and** *finite* $(\bigcup F)$ *weight-fun* $w (\bigcup F) \wedge (w \bowtie_f F (\bigcup F)) \geq 0$
shows $\exists a. frankl\text{-}element\ a\ F \wedge w\ a \neq (0 :: nat)$
 $\langle proof \rangle$

lemma *frankl-share''*:

assumes $F \neq \{\}$ **and** *finite* $(\bigcup F)$ **and** *frankl* F
shows $\exists w. weight\text{-}fun\ w (\bigcup F) \wedge (w \bowtie_f F (\bigcup F)) \geq 0$
 $\langle proof \rangle$

Sufficient condition for Frankl using hypershares.

theorem *frankl-share*:

assumes $F \neq \{\}$ **and** *finite* $(\bigcup F)$
shows *frankl* $F \longleftrightarrow (\exists w. weight\text{-}fun\ w (\bigcup F) \wedge (w \bowtie_f F (\bigcup F)) \geq 0)$

<proof>

theorem *frankl-hyper-share:*

assumes $F \neq \{\}$ **and** *finite* $(\bigcup F)$

assumes *weight-fun* w $(\bigcup F)$

assumes $K' \cup S = \bigcup F$ **and** $K' \cap S = \{\}$

assumes $\forall K \in \text{Pow } K'. (w \odot_f K S F (\bigcup F)) \geq 0$

shows $\exists a. \text{frankl-element } a F \wedge w a \neq 0$

<proof>

end

8 FC families characterization using shares (Poonen's theorem)

theory *WeightsShares-FCFamily*

imports *WeightsShares-Frankl*

begin

8.1 HyperCube projection

abbreviation *hypercube-prj* **where**

hypercube-prj $K S F \equiv (\lambda S. S - K) \text{ ` } (F \cap \text{op} \cup K \text{ ` } \text{Pow } S)$

syntax

-hypercube-prj $:: 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set set} \Rightarrow 'a \text{ set set} \quad (\langle -, -, - \rangle)$

translations

$\langle K, S, F \rangle == \text{CONST } \text{hypercube-prj } K S F$

lemma *hypercube-prj-union-closed:*

assumes *union-closed* F

shows *union-closed* $\langle K, S, F \rangle$

<proof>

lemma *hypercube-prj-union-closed-additional:*

assumes *union-closed* F **and** $F_c \subseteq F$ **and** $S = \bigcup F_c$ **and** $K \cap S = \{\}$

shows *union-closed-additional* $\langle K, S, F \rangle F_c$

<proof>

lemma *set-weight-w0:*

assumes *finite* K **and** *finite* S

assumes $K \cap S = \{\}$ **and** $\forall x \in K. w x = 0$ **and** $S' \subseteq S$

shows $w \triangleright_s (K \cup S') = w \triangleright_s S'$

<proof>

lemma *hypercube-prj-hyper-share-w0:*

assumes *finite* S **and** *finite* K **and** $\forall x \in K. w x = 0$

shows $(w \odot_f K S F X) = (w \bowtie_f \langle K, S, F \rangle X)$
 $\langle proof \rangle$

8.2 All shares non-negative

abbreviation *uce-shares-nonneg* **where**

uce-shares-nonneg $Fc w \equiv \forall F' \in \mathbb{F}Fc. (w \bowtie_f F' (\bigcup Fc)) \geq 0$

theorem *frankl-uce-shares-nonneg*:

assumes $F \neq \{\}$ **and** *finite-union-closed* F **and** *weight-fun* $w (\bigcup F)$

assumes $Fc \subseteq F$ **and** $\forall x \in (\bigcup F) - (\bigcup Fc). w x = 0$

assumes *uce-shares-nonneg* $Fc w$

shows $\exists a \in \bigcup Fc. \text{frankl-element } a F$

$\langle proof \rangle$

theorem *FC-family-uce-shares-nonneg*:

assumes *weight-fun* $w (\bigcup Fc)$ **and** *uce-shares-nonneg* $Fc w$

shows *FC-family* Fc

$\langle proof \rangle$

end

9 nonFC families characterization using shares (Poonen's theorem)

theory *WeightsShares-NotFCFamily*

imports *Main WeightsShares-FCFamily*

begin

lemma *frankl-fun-hypercube*:

assumes $S = \bigcup F$ **and** $K \cap S = \{\}$ **and** $a \notin K$

shows *frankl-fun* $a (op \cup K \text{ ' } F) = \text{frankl-fun } a F$

$\langle proof \rangle$

lemma *sum-over-spread*:

fixes $d c :: nat$ **and** f

assumes $d \neq 0$

shows $(\sum s \leftarrow [0..<d * c]. f (s \text{ div } d)) = \text{int } d * (\sum s \leftarrow [0..<c]. f s)$

$\langle proof \rangle$

lemma *archimedean-negative*:

fixes $x y :: int$

assumes $y < 0$

shows $\exists d. x + \text{int } d * y < 0$

$\langle proof \rangle$

lemma *nonFC*:

fixes $Fc :: nat \text{ set set}$

```

assumes length  $c = \text{length } Fs$ 
assumes  $\exists \ cj \in \text{set } c. \ cj > 0$ 
assumes finite  $(\bigcup Fc)$ 
assumes union-closed  $Fc$ 
assumes  $\forall \ F \in \text{set } Fs. \ F \in \llbracket Fc \rrbracket$ 
assumes let  $Fs' = Fs$ 
           in  $(\forall \ a \in \bigcup Fc. \ \text{sum-list } (\text{map } (\lambda \ (x, y). \ \text{int } x * y) \ (\text{zip } c \ (\text{map}$ 
(frankl-fun  $a) \ Fs')))) < 0)$ 
shows  $\neg \text{FC-family } Fc$ 
 $\langle \text{proof} \rangle$ 

end

```

10 SomeShareNegative — combinatorial search for union-closed extension with a negative share

```

theory SomeShareNegative
imports Main WeightsShares-FCFamily
begin

```

According to the theorem

$$\llbracket \text{weight-fun } w \ (\bigcup Fc); \forall F' \in \llbracket Fc \rrbracket. \ 0 \leq w \bowtie_f F' \bigcup Fc \rrbracket \implies \text{FC-family } Fc$$

to show that a family is an FC family, it suffices to show that there is no member of union closed extension of a given family with a negative share. The function *some-share-negative* performs a combinatorial enumeration of union closed extensions of a given family and checks whether it contains a family with a negative share with respect to the given weight function. We give a rather abstract (nonexecutable) definition of this function and prove its main properties. This function will be later refined and a more efficient, executable implementation will be given.

```

primrec some-share-negative-aux :: 'a set list  $\Rightarrow$  'a set set  $\Rightarrow$  'a set set  $\Rightarrow$  ('a  $\Rightarrow$ 
nat)  $\Rightarrow$  'a set  $\Rightarrow$  bool where
  some-share-negative-aux [] Ft Fc w X =  $((w \bowtie_f Ft X) < 0)$ 
| some-share-negative-aux (h # t) Ft Fc w X =
   $(\text{if } (w \bowtie_f Ft X) + \text{sum-list } (\text{map } (\lambda \ A. \ (w \bowtie_s A X)) \ (h \# t)) \geq 0 \text{ then}$ 
    False
   $\text{else if } \text{some-share-negative-aux } t \ Ft \ Fc \ w \ X \text{ then}$ 
    True
   $\text{else if } h \in Ft \text{ then}$ 
    False
   $\text{else}$ 
    some-share-negative-aux } t \ (\text{insert-and-close-additional } h \ Ft \ Fc) \ Fc \ w \ X
   $)$ 

```

lemma *some-share-negative-aux-soundness*:
assumes *some-share-negative-aux* L Ft Fc w $X = False$ **and**
 $\forall A \in \text{set } L. (w \bowtie_s A X) < 0$ **and**
distinct L **and**
finite F **and** $F \supseteq Ft$ **and**
union-closed-additional F Fc **and**
 $\forall A \in F - Ft. (w \bowtie_s A X) < 0 \longrightarrow A \in \text{List.set } L$
shows $(w \bowtie_f F X) \geq 0$
 $\langle \text{proof} \rangle$

definition *some-share-negative* :: $'a \text{ set set} \Rightarrow ('a \Rightarrow \text{nat}) \Rightarrow \text{bool}$ **where**
some-share-negative Fc $w \equiv$
 $\text{let } X = (\bigcup Fc);$
 $P = \text{Pow } X;$
 $Fc = \text{closure } Fc;$
 $N = \{x. x \in P \wedge (w \bowtie_s x X) < 0\};$
 $L = (\text{SOME } L. \text{distinct } L \wedge \text{set } L = N) \text{ in}$
some-share-negative-aux $L \{\}$ Fc w X

lemma *some-share-negative-soundness'*:
assumes *some-share-negative* Fc $w = False$ **and**
finite $(\bigcup Fc)$ **and** $F \in \{\ Fc \}$
shows $(w \bowtie_f F (\bigcup Fc)) \geq 0$
 $\langle \text{proof} \rangle$

theorem *some-share-negative-soundness*:
assumes *finite* $(\bigcup Fc)$
shows *some-share-negative* Fc $w = False \implies \text{uce-shares-nonneg } Fc$ w
 $\langle \text{proof} \rangle$

end

10.1 Abstract representation of sets and families with weights and shares

theory *WeightsSharesImpl*
imports *WeightsShares* *FamilyImpl*
 $HOL-Library.Mapping$
 $HOL-Library.RBT-Mapping$ $HOL-Library.RBT-Impl$
begin

locale *SetWeightsSharesImpl* = *SetImpl* *to-set* *inv* **for**
 $\text{inv} :: 's \Rightarrow \text{bool}$ **and** $\text{to-set} :: 's \Rightarrow 'a \text{ set} +$
fixes *set-weight* :: $('a \Rightarrow \text{nat}) \Rightarrow 's \Rightarrow \text{nat}$
assumes *set-weight-set*: $\text{inv } A \implies \text{set-weight } w A = w \triangleright_s \text{to-set } A$
begin

definition *set-share* :: $'s \Rightarrow ('a \Rightarrow \text{nat}) \Rightarrow 's \Rightarrow \text{int}$ **where**
 $\text{set-share } A w X = 2 * \text{int } (\text{set-weight } w A) - \text{int } (\text{set-weight } w X)$

definition *family-share* :: 's list \Rightarrow ('a \Rightarrow nat) \Rightarrow 's \Rightarrow int **where**
family-share F w X = sum-list (map (λ A. set-share A w X) F)

lemma *set-share-set*:
assumes inv x **and** inv X
shows set-share x w X = (w \bowtie_s (to-set x) (to-set X))
 <proof>

lemma *family-share-set*:
assumes distinct F **and** $\forall l \in \text{set } F. \text{inv } l$ **and** inv X
shows family-share F w X = (w \bowtie_f (f-to-set F) (to-set X))
 <proof>

end

10.1.1 Implementation by sorted and distinct lists

definition *set-weight-l* :: ('a \Rightarrow nat) \Rightarrow 'a list \Rightarrow nat **where**
set-weight-l w S = sum-list (map w S)

lemma *set-weight-l*:
assumes distinct A
shows set-weight-l w A = w \triangleright_s set A
 <proof>

global-interpretation *SetWeightsSharesImpl-lists*: SetWeightsSharesImpl λ (l::nat list). sorted l \wedge distinct l set set-weight-l
 <proof>

10.1.2 Implementation by natural numbers

definition *set-weight-n* :: (nat \Rightarrow nat) \Rightarrow nat \Rightarrow nat **where**
set-weight-n w n = sum-list (map w (nat2list n))

lemma *set-weight-n*:
shows set-weight-n w A = w \triangleright_s set (nat2list A)
 <proof>

global-interpretation *SetWeightsSharesImpl-nats*: SetWeightsSharesImpl λ -. True
 set \circ nat2list set-weight-n
 <proof>

10.2 Weight functions implemented by mappings

locale *SetWeightsSharesMapImpl* = SetWeightsSharesImpl inv to-set set-weight
for
 inv :: 's \Rightarrow bool **and** to-set :: 's \Rightarrow 'a set **and** set-weight :: ('a \Rightarrow nat) \Rightarrow 's \Rightarrow nat +


```

fixes set-weight-map :: ('a, nat) mapping  $\Rightarrow$  's  $\Rightarrow$  nat
assumes set-weight-map:
   $\llbracket \text{inv } A; \forall x \in \text{to-set } A. \text{Mapping.lookup } \text{wm } x = \text{Some } (w \ x) \rrbracket \implies$ 
    set-weight-map wm A = set-weight w A
begin

abbreviation set-share-map :: 's  $\Rightarrow$  ('a, nat) mapping  $\Rightarrow$  's  $\Rightarrow$  int where
  set-share-map A w X  $\equiv$  2 * int(set-weight-map w A) - int (set-weight-map w X)

end

```

10.2.1 Representation of sets by lists

```

definition set-weight-map-l :: ('a, nat) mapping  $\Rightarrow$  'a list  $\Rightarrow$  nat where
  set-weight-map-l w S = sum-list (map (the  $\circ$  (Mapping.lookup w)) S)

```

```

lemma set-weight-map-l:
   $\llbracket \text{distinct } A; \forall x \in \text{set } A. \text{Mapping.lookup } \text{wm } x = \text{Some } (w \ x) \rrbracket \implies$ 
    set-weight-map-l wm A = set-weight-l w A
  <proof>

```

```

global-interpretation SetWeightsSharesMapImpl-lists:
  SetWeightsSharesMapImpl  $\lambda$  (l::nat list). sorted l  $\wedge$  distinct l List.set set-weight-l
  set-weight-map-l
  <proof>

```

10.2.2 Representation of sets by natural numbers

```

definition set-weight-map-n :: (nat, nat) Mapping.mapping  $\Rightarrow$  nat  $\Rightarrow$  nat where
  set-weight-map-n w n = sum-list (map (the  $\circ$  (Mapping.lookup w)) (nat2list n))

```

```

global-interpretation SetWeightsSharesCachedImpl-nats:
  SetWeightsSharesMapImpl  $\lambda$  -. True List.set  $\circ$  nat2list set-weight-n set-weight-map-n
  <proof>

```

end

11 SomeShareNegative – executable implementation

```

theory SomeShareNegativeImpl
imports Main
  HOL-Library.List-lexord
  More.MoreMap
  SomeShareNegative UnionClosedImpl WeightsSharesImpl
begin

```

In this section we define a refinement of the *some-share-negative* function in several steps in order to obtain a more efficient, executable implementation.

Optimizations are introduced gradually, through separate refinement steps.

11.1 Refinement of SomeShareNegativeFunction

```
locale SomeShareNegativeImpl =
  SetWeightsSharesMapImpl inv to-set set-weight set-weight-map +
  SetUnionImpl inv to-set empty union pow
  for inv :: 's  $\Rightarrow$  bool and to-set :: 's  $\Rightarrow$  'a set and set-weight :: ('a  $\Rightarrow$  nat)  $\Rightarrow$  's
 $\Rightarrow$  nat and set-weight-map :: ('a, nat) mapping  $\Rightarrow$  's  $\Rightarrow$  nat and
  empty :: 's and union :: 's  $\Rightarrow$  's  $\Rightarrow$  's and pow :: 's  $\Rightarrow$  's list
begin
```

11.1.1 Initial version

```
fun some-share-negative-aux-1 :: 's list  $\Rightarrow$  's list  $\Rightarrow$  's list  $\Rightarrow$  ('a  $\Rightarrow$  nat)  $\Rightarrow$  's  $\Rightarrow$ 
bool where
  some-share-negative-aux-1 [] F Init w X = (family-share F w X < 0)
| some-share-negative-aux-1 (h # t) F Init w X =
  (if family-share F w X + sum-list (map ( $\lambda$  A. set-share A w X) (h # t))  $\geq$  0
  then
    False
  else if some-share-negative-aux-1 t F Init w X then
    True
  else if h  $\in$  set F then
    False
  else
    some-share-negative-aux-1 t (insert-and-close-additional h F Init) Init w X
  )
```

lemma some-share-negative-aux-1-some-share-negative-aux:

```
assumes distinct F
   $\forall l \in \text{set } F. \text{inv } l$ 
   $\forall l \in \text{set } L. \text{inv } l$ 
   $\forall l \in \text{set } \text{Init}. \text{inv } l$ 
  inv X
shows some-share-negative-aux-1 L F Init w X = some-share-negative-aux (map
to-set L) (f-to-set F) (f-to-set Init) w (to-set X)
<proof>
```

11.1.2 Passing current family share as a parameter

```
fun some-share-negative-aux-2 :: ('s  $\times$  int) list  $\Rightarrow$  ('s list  $\times$  int)  $\Rightarrow$  's list  $\Rightarrow$  ('a
 $\Rightarrow$  nat)  $\Rightarrow$  's  $\Rightarrow$  bool where
  some-share-negative-aux-2 [] (F, s) Init w X = (s < 0)
| some-share-negative-aux-2 (h # t) (F, s) Init w X =
  (if s + sum-list (map snd (h # t))  $\geq$  0 then
    False
  else if some-share-negative-aux-2 t (F, s) Init w X then
```

```

    True
  else if fst h ∈ set F then
    False
  else
    let F' = insert-and-close-additional (fst h) F Init;
    s' = family-share F' w X in
    some-share-negative-aux-2 t (F', s') Init w X
)

```

lemma *some-share-negative-aux-2-some-share-negative-aux-1*:

assumes $L = \text{zip } l \ (\text{map } (\lambda F. \text{set-share } F \ w \ X) \ l) \ c = \text{family-share } F \ w \ X$
shows *some-share-negative-aux-2* $L \ (F, c) \text{ Init } w \ X = \text{some-share-negative-aux-1}$
 $l \ F \text{ Init } w \ X$
 ⟨proof⟩

11.1.3 Caching set shares

Instead of calculating set share every time from scratch when calculating, family shares, we shall maintain a mapping from sets to their shares.

abbreviation *family-share-cached* $:: 's \text{ list} \Rightarrow ('s, \text{int}) \text{ mapping} \Rightarrow \text{int}$ **where**
family-share-cached $F \text{ shares} \equiv \text{sum-list } (\text{map } (\text{the} \circ (\lambda A. \text{Mapping.lookup shares } A))) \ F)$

lemma *family-share-cached-family-share*:

assumes $\forall A. \text{inv } A \wedge \text{to-set } A \subseteq S \longrightarrow \text{Mapping.lookup shares } A = \text{Some}$
 $(\text{set-share } A \ w \ X)$
 $\forall A \in \text{set } F. \text{inv } A \wedge \text{to-set } A \subseteq S$
shows *family-share-cached* $F \text{ shares} = \text{family-share } F \ w \ X$
 ⟨proof⟩

lemma *family-share-cached-cong*:

assumes $\text{set } F = \text{set } F' \text{ distinct } F \text{ distinct } F'$
shows *family-share-cached* $F \text{ shares} = \text{family-share-cached } F' \text{ shares}$
 ⟨proof⟩

fun *some-share-negative-aux-3* $:: ('s \times \text{int}) \text{ list} \Rightarrow ('s \text{ list} \times \text{int}) \Rightarrow 's \text{ list} \Rightarrow ('s,$
 $\text{int}) \text{ mapping} \Rightarrow \text{bool}$ **where**

some-share-negative-aux-3 $\square (F, s) \text{ Init shares} = (s < 0)$
 $| \text{some-share-negative-aux-3 } (h \# t) (F, s) \text{ Init shares} =$
 $(\text{if } s + \text{sum-list } (\text{map snd } (h \# t)) \geq 0 \text{ then}$
 False
 $\text{else if some-share-negative-aux-3 } t (F, s) \text{ Init shares then}$
 True
 $\text{else if fst } h \in \text{set } F \text{ then}$
 False
 else
 $\text{let } F' = \text{insert-and-close-additional } (\text{fst } h) \ F \text{ Init};$
 $s' = \text{family-share-cached } F' \text{ shares in}$

)
some-share-negative-aux-3 $t (F', s') \text{ Init shares}$

lemma *some-share-negative-aux-3-some-share-negative-aux-2*:

assumes $\forall A. \text{inv } A \wedge \text{to-set } A \subseteq S \longrightarrow \text{Mapping.lookup shares } A = \text{Some } (\text{set-share } A \text{ w } X)$

$\forall A \in \text{set } F. \text{inv } A \wedge \text{to-set } A \subseteq S$
 $\forall A \in \text{set } \text{Init}. \text{inv } A \wedge \text{to-set } A \subseteq S$
 $\forall A \in \text{set } (\text{map fst } l). \text{inv } A \wedge \text{to-set } A \subseteq S$
 $\text{distinct } F$

shows *some-share-negative-aux-3* $l (F, s) \text{ Init shares} = \text{some-share-negative-aux-2}$
 $l (F, s) \text{ Init w } X$
 $\langle \text{proof} \rangle$

lemma *some-share-negative-aux-3-correct*:

assumes *some-share-negative-aux-3* $L (F, c) \text{ Init shares} = \text{False}$

$\forall (x, y) \in \text{set } L. y = \text{set-share } x \text{ w } X$

$c = \text{family-share } F \text{ w } X$

$\forall A. \text{inv } A \wedge \text{to-set } A \subseteq S \longrightarrow \text{Mapping.lookup shares } A = \text{Some } (\text{set-share } A \text{ w } X)$

$\forall A \in \text{set } F. \text{inv } A \wedge \text{to-set } A \subseteq S$
 $\forall A \in \text{set } \text{Init}. \text{inv } A \wedge \text{to-set } A \subseteq S$
 $\forall A \in \text{set } (\text{map fst } L). \text{inv } A \wedge \text{to-set } A \subseteq S$
 $\text{distinct } F$
 $\text{inv } X$

$\forall A \in \text{set } (\text{map fst } L). (w \bowtie_s (\text{to-set } A) (\text{to-set } X)) < 0$

$\text{finite } F'$

$\text{UnionClosed.union-closed-additional } F' (\text{f-to-set } \text{Init})$

$\text{f-to-set } F \subseteq F'$

$\forall A \in F' - \text{f-to-set } F. ((w \bowtie_s A (\text{to-set } X)) < 0 \longrightarrow A \in \text{f-to-set } (\text{map fst } L))$

$\text{distinct } L$

shows $(w \bowtie_f F' (\text{to-set } X)) \geq 0$

$\langle \text{proof} \rangle$

11.1.4 Passing the sum of shares of elements in the current list as a parameter

fun *some-share-negative-aux-4* :: $((s \times \text{int}) \text{ list} \times \text{int}) \Rightarrow (s \text{ list} \times \text{int}) \Rightarrow s \text{ list}$
 $\Rightarrow (s, \text{int}) \text{ mapping} \Rightarrow \text{bool}$ **where**

some-share-negative-aux-4 $([], -) (F, s) \text{ Init shares} = (s < 0)$

| *some-share-negative-aux-4* $((h \# t), ls) (F, s) \text{ Init shares} =$

$(\text{if } s + ls \geq 0 \text{ then}$

False

$\text{else if some-share-negative-aux-4 } (t, ls - \text{snd } h) (F, s) \text{ Init shares then}$

True

$\text{else if fst } h \in \text{set } F \text{ then}$

False

else

$\text{let } F' = \text{insert-and-close-additional } (\text{fst } h) F \text{ Init};$

$s' = \text{family-share-cached } F' \text{ shares in}$
 $\text{some-share-negative-aux-4 } (t, ls - \text{snd } h) (F', s') \text{ Init shares}$
 $)$

lemma *some-share-negative-aux-4-some-share-negative-aux-3:*
shows $\text{some-share-negative-aux-4 } (l, \text{sum-list } (\text{map } \text{snd } l)) (F, s) \text{ Init shares} =$
 $\text{some-share-negative-aux-3 } l (F, s) \text{ Init shares}$
 $\langle \text{proof} \rangle$

lemma *some-share-negative-aux-4-equal-set:*
assumes $\text{set } F = \text{set } F' \text{ distinct } F \text{ distinct } F'$
shows $\text{some-share-negative-aux-4 } (l, ls) (F, s) \text{ Init shares} =$
 $\text{some-share-negative-aux-4 } (l, ls) (F', s) \text{ Init shares}$
 $\langle \text{proof} \rangle$

11.1.5 Incremental insert and close operation and calculation of share of the extended family

fun *insert-and-close-additional-cached* **where**
 $\text{insert-and-close-additional-cached } A (Ft, s) Fc \text{ shares} =$
 $(\text{let } \text{add} = [A] @ \text{union-with-all } A Ft @ \text{union-with-all } A Fc;$
 $\text{add} = \text{filter } (\lambda x. x \notin \text{set } Ft) (\text{remdups } \text{add})$
 $\text{in } (\text{add} @ Ft, s + \text{family-share-cached } \text{add shares}))$

lemma *insert-and-close-additional-cached-family-share-cached:*
shows $\text{let } (Ft', s') = \text{insert-and-close-additional-cached } A (Ft, \text{family-share-cached}$
 $Ft \text{ shares}) Fc \text{ shares}$
 $\text{in } s' = \text{family-share-cached } Ft' \text{ shares}$
 $\langle \text{proof} \rangle$

lemma *insert-and-close-additional-cached:*
assumes
 $\text{distinct } Ft$
 $\text{insert-and-close-additional-cached } A (Ft, \text{family-share-cached } Ft \text{ shares}) Fc \text{ shares}$
 $=$
 (Ft', s')
shows
 $\text{set } Ft' = \text{set } (\text{insert-and-close-additional } A Ft Fc) \wedge$
 $s' = \text{family-share-cached } Ft' \text{ shares}$
 $\langle \text{proof} \rangle$

function *some-share-negative-aux-5* $:: ((s \times \text{int}) \text{ list} \times \text{int}) \Rightarrow (s \text{ list} \times \text{int}) \Rightarrow$
 $s \text{ list} \Rightarrow (s, \text{int}) \text{ mapping} \Rightarrow \text{bool}$ **where**
 $\text{some-share-negative-aux-5 } ([], -) (F, s) \text{ Init shares} =$
 $(\text{if } s < 0 \text{ then True else False})$
 $| \text{some-share-negative-aux-5 } ((h \# t), ls) (F, s) \text{ Init shares} =$
 $(\text{if } s + ls \geq 0 \text{ then}$
 False

else if some-share-negative-aux-5 (t, ls - snd h) (F, s) Init shares then
 True
 else if fst h ∈ set F then
 False
 else
 let (F', s') = insert-and-close-additional-cached (fst h) (F, s) Init shares in
 some-share-negative-aux-5 (t, ls - snd h) (F', s') Init shares
)
 ⟨proof⟩
termination
 ⟨proof⟩

lemma some-share-negative-aux-5-some-share-negative-aux-4:
assumes s = family-share-cached F shares distinct F
shows some-share-negative-aux-5 (L, ls) (F, s) Init shares =
 some-share-negative-aux-4 (L, ls) (F, s) Init shares
 ⟨proof⟩

11.1.6 Final implementation

definition some-share-negative :: 'a list ⇒ ('a, nat) mapping ⇒ bool **where**
 some-share-negative A w ≡
 let S = Union A;
 P = pow S;
 A = close A;
 U' = map (λ A. set-share-map A w S) P;
 U = sort-key snd (zip P U');
 shares = tabulate U;
 L = takeWhile (λ (a, b). b < 0) U;
 F = [] in
 some-share-negative-aux-5 (L, sum-list (map snd L)) (F, 0) A shares

lemma some-share-negativeFalse-FamilyShare:
assumes
 some-share-negative A w = False
 A' = f-to-set A
 ∀ l ∈ set A. inv l
 finite F
 UnionClosed.union-closed-additional F (closure A')
 ∀ x ∈ F. x ⊆ ⋃ A'
 ∀ x ∈ ⋃ A'. Mapping.lookup w x = Some (w' x)
shows (w' ⋈_f F (⋃ A')) ≥ 0
 ⟨proof⟩

lemma some-share-negativeSound:
assumes
 ∀ l ∈ set A. inv l

$A' = f\text{-to-set } A$
 $\forall x \in A'. \text{ finite } x$
 $\forall x \in \bigcup A'. \text{ Mapping.lookup } w \ x = \text{Some } (w' \ x)$
shows $\text{some-share-negative } A \ w = \text{False} \implies \text{uce-shares-nonneg } A' \ w'$
 $\langle \text{proof} \rangle$
end

11.2 Interpretations

11.2.1 Representation of sets by lists

global-interpretation *SomeShareNegativeImpl-lists:*

$\text{SomeShareNegativeImpl } \lambda \ (l :: \text{nat list}). \text{ sorted } l \wedge \text{ distinct } l \text{ List.set set-weight-}l$
 $\text{set-weight-map-}l \sqsubseteq \text{ merge Pow-}l$
 $\langle \text{proof} \rangle$

11.2.2 Representation of sets by natural numbers

global-interpretation *SomeShareNegativeImpl-nats:*

$\text{SomeShareNegativeImpl } \lambda \ n. \text{ True set } \circ \text{ nat2list set-weight-}n \text{ set-weight-map-}n \ 0$
 $\text{bitor pow-}n$

defines

$\text{insert-and-close-additional-cached-}n = \text{SomeShareNegativeImpl-nats.insert-and-close-additional-cached}$

and

$\text{some-share-negative-aux-5-}n = \text{SomeShareNegativeImpl-nats.some-share-negative-aux-5}$

and

$\text{some-share-negative-}n = \text{SomeShareNegativeImpl-nats.some-share-negative}$

$\langle \text{proof} \rangle$

definition *some-share-negative where*

$\text{some-share-negative } A \ w \equiv \text{some-share-negative-}n \ (\text{map list2nat } A) \ w$

lemma *some-share-negative-soundness-nats:*

assumes $\text{finite } A \ \forall \ X \in A. \text{ finite } X$

$\text{set } (\text{map set } A') = A$

$\forall \ X \in \text{set } A'. \text{ distinct } X \wedge \text{ sorted } X$

$w' = \text{Mapping.tabulate } (\text{sorted-list-of-set } (\bigcup A)) \ w$

shows $\text{some-share-negative } A' \ w' = \text{False} \implies \text{uce-shares-nonneg } A \ w$

$\langle \text{proof} \rangle$

definition *weights2map where*

$\text{weights2map } l = \text{foldl } (\lambda \ w \ (k, v). \ w \ (k := v)) \ (\lambda \ -. \ 0) \ l$

lemma *weights2map-snoc[simp]:* $\text{weights2map } (xs \ @ \ [x]) = (\text{weights2map } xs) \ (\text{fst}$

$x := \text{snd } x)$

$\langle \text{proof} \rangle$

definition *ssn* **where**
ssn f $w = \text{some-share-negative } f \text{ (tabulate } w)$

end

12 Isomorphisms of set families

theory *IsomorphicFamilies*
imports *Main*
More/MoreSet More/MoreFun
UnionClosed Frankl
begin

Each injective function f on the domain $\bigcup F$ maps the set family F to its isomorphic image $op \text{ ' } f \text{ ' } F$. Elements of $\bigcup F$ are mapped using f , while elements of F are mapped using $op \text{ ' } f$.

12.1 Properites preserved by injective functions

lemma *empty-iso*:
shows $F = \{\}$ $\longleftrightarrow (op \text{ ' } f \text{ ' } F) = \{\}$
 $\langle proof \rangle$

lemma *inj-on-iso*:
assumes *inj-on* f $(\bigcup F)$
shows *inj-on* $(op \text{ ' } f) F$
 $\langle proof \rangle$

lemma *inj-on-iso-strong*:
assumes *inj-on* f $(\bigcup F)$
shows *inj-on* $(op \text{ ' } f) (Pow (\bigcup F))$
 $\langle proof \rangle$

lemma *finite-iso*:
assumes *inj-on* f $(\bigcup F)$
shows *finite* $F \longleftrightarrow \text{finite } (op \text{ ' } f \text{ ' } F)$
 $\langle proof \rangle$

lemma *card-iso*:
assumes *inj-on* f $(\bigcup F)$
shows *card* $(op \text{ ' } f \text{ ' } F) = \text{card } F$
 $\langle proof \rangle$

lemma *finite-elems-iso*:
assumes *inj-on* f $(\bigcup F)$
shows $(\forall S \in F. \text{finite } S) \longleftrightarrow (\forall S \in (op \text{ ' } f \text{ ' } F). \text{finite } S)$

$\langle proof \rangle$

lemma *card-elem-iso*:

assumes *inj-on* f $(\bigcup F)$ $A \in F$ *card* $A > 0$

shows *card* $(f \text{ ` } A) = \text{card } A$

$\langle proof \rangle$

lemma *finite-Union-iso*:

assumes *inj-on* f $(\bigcup F)$

shows *finite* $(\bigcup F) = \text{finite } (\bigcup (op \text{ ` } f \text{ ` } F))$

$\langle proof \rangle$

lemma *union-closed-iso*:

assumes *inj-on* f $(\bigcup F)$

shows *union-closed* $F \longleftrightarrow \text{union-closed } (op \text{ ` } f \text{ ` } F)$

$\langle proof \rangle$

lemma *finite-union-closed-iso*:

assumes *inj-on* f $(\bigcup F)$

shows *finite-union-closed* $F \longleftrightarrow \text{finite-union-closed } (op \text{ ` } f \text{ ` } F)$

$\langle proof \rangle$

lemma *count-iso*:

assumes *inj-on* f $(\bigcup F)$ **and** $a \in \bigcup F$

shows *count* $a F = \text{count } (f a) (op \text{ ` } f \text{ ` } F)$

$\langle proof \rangle$

lemma *Frankl-iso*:

assumes *inj-on* f $(\bigcup F)$

shows *frankl* $F \longleftrightarrow \text{frankl } (op \text{ ` } f \text{ ` } F)$

$\langle proof \rangle$

lemma *closure-iso*:

shows *closure* $(op \text{ ` } g \text{ ` } F) = op \text{ ` } g \text{ ` } (\text{closure } F)$

$\langle proof \rangle$

lemma *FC-family-iso*:

fixes $f :: 'a \Rightarrow nat$

assumes *inj-on* f $(\bigcup Fc)$ *FC-family* $(op \text{ ` } f \text{ ` } Fc)$

shows *FC-family* Fc

$\langle proof \rangle$

12.2 Injective embedding

definition *inj-embed* **where**

$[simp]: \text{inj-embed } F F' \longleftrightarrow (\exists f. \text{inj-on } f (\bigcup F) \wedge F' = (op \text{ ` } f) \text{ ` } F)$

lemma *inj-embed-refl* $[simp]:$

shows *inj-embed* $F F$
 $\langle proof \rangle$

lemma *inj-embed-sym*:
assumes *inj-embed* $F F'$
shows *inj-embed* $F' F$
 $\langle proof \rangle$

lemma *inj-embed-trans*:
assumes *inj-embed* $F F'$ **and** *inj-embed* $F' F''$
shows *inj-embed* $F F''$
 $\langle proof \rangle$

12.3 Isomorphism definition

There is a bijective mapping between equivalent families.

definition *iso* **where**
 $[simp]: iso\ F\ F' \longleftrightarrow (\exists\ h.\ bij\ betw\ h\ (\bigcup\ F)\ (\bigcup\ F') \wedge F' = (op\ 'h)\ 'F)$

lemma *inj-embed-iso*:
shows *inj-embed* $F F' \longleftrightarrow iso\ F F'$
 $\langle proof \rangle$

lemma *iso-refl*:
shows *iso* $F F$
 $\langle proof \rangle$

lemma *iso-sym*:
assumes *iso* $F F'$
shows *iso* $F' F$
 $\langle proof \rangle$

lemma *iso-trans*:
assumes *iso* $F F'$ **and** *iso* $F' F''$
shows *iso* $F F''$
 $\langle proof \rangle$

lemma *iso-finite*:
assumes *iso* $F F'$
shows *finite* $F \longleftrightarrow finite\ F'$
 $\langle proof \rangle$

lemma *iso-insert*:
assumes *inj-on* $f\ (\bigcup\ F \cup A)$
shows *iso* $(F \cup \{A\})\ ((op\ 'f\ 'F) \cup \{f\ 'A\})$
 $card\ (f\ 'A) = card\ A$
 $f\ 'A \subseteq f\ '(\bigcup\ F \cup A)$
 $A \notin F \implies f\ 'A \notin op\ 'f\ 'F$
 $\langle proof \rangle$

12.4 Iso-representing collections of families

FF' iso-represents FF if an isomorphic image of each element of FF is in FF'

definition *iso-represents* **where**

iso-represents $FF' FF \longleftrightarrow (\forall F \in FF. \exists F' \in FF'. \text{iso } F F')$

abbreviation *iso-representing-subset* **where**

iso-representing-subset $FF' FF \equiv \text{iso-represents } FF' FF \wedge FF' \subseteq FF$

end

12.5 Non-isomorphic families of sets

theory *NonIsomorphicFamilies*

imports *Main*

IsomorphicFamilies

FamilyImpl

begin

In this section we define a combinatorial algorithm (implemented by a function *non-isomorphic-families*) that computes non isomorphic families of sets. Two uniform families (over the same domain) are called isomorphic if one can be obtained from the other by permuting elements of the domain. This function is used to remove symmetric cases in verification of FC families.

locale *Mapping* =

fixes *to-fun* :: $'m \Rightarrow ('a \Rightarrow 'a)$

locale *SetPermutations* = *SetImpl to-set inv* + *Mapping to-fun* **for**

inv :: $'s::\text{linorder} \Rightarrow \text{bool}$ **and** *to-set* :: $'s::\text{linorder} \Rightarrow 'a \text{ set}$ **and** *to-fun* :: $'m \Rightarrow ('a \Rightarrow 'a)$ +

fixes *permute-set* :: $'m \Rightarrow 's \Rightarrow 's$

assumes *permute-set-set*: $\text{to-set } (\text{permute-set } p \ s) = (\text{to-fun } p) \text{ ` } (\text{to-set } s)$

assumes *permute-set-inv*: $\llbracket \text{inj-on } (\text{to-fun } p) (\text{to-set } s); \text{inv } s \rrbracket \implies \text{inv } (\text{permute-set } p \ s)$

begin

definition *permute-family* :: $'m \Rightarrow 's \text{ list} \Rightarrow 's \text{ list}$ **where**

permute-family $p \ F = \text{map } (\text{permute-set } p) \ F$

lemma *permute-family-set*:

shows $f\text{-to-set } (\text{permute-family } p \ F) = \text{op ` } (\text{to-fun } p) \text{ ` } f\text{-to-set } F$
 $\langle \text{proof} \rangle$

lemma *permute-family-inj-embed*:

assumes *inj-on* $(\text{to-fun } p) (\bigcup f\text{-to-set } F)$

shows *inj-embed* $(f\text{-to-set } F) (f\text{-to-set } (\text{permute-family } p \ F))$
 $\langle \text{proof} \rangle$

function *non-isomorphic-families-aux* **where**
non-isomorphic-families-aux perms fams res =
 (case fams of
 [] \Rightarrow res
 | h # t \Rightarrow
 let hp = remdups (h # map (λp . permute-family p h) perms) in
 non-isomorphic-families-aux perms (filter (λl . sort l \notin set (map sort hp)) fams)
 (h # res))
 <proof>
termination
 <proof>
declare *non-isomorphic-families-aux.simps* [simp del]

definition *non-isomorphic-families* **where**
 [simp]: *non-isomorphic-families* perms fams = *non-isomorphic-families-aux* perms
 fams []

lemma *non-isomorphic-families-aux-res-mono*:
 shows set res \subseteq set (*non-isomorphic-families-aux* perms fams res)
 <proof>

lemma *non-isomorphic-families-aux-subset*:
 assumes $F \supseteq$ set res $F \supseteq$ set fams
 shows $F \supseteq$ set (*non-isomorphic-families-aux* perms fams res)
 <proof>

lemma *non-isomorphic-families-subset*:
 shows set (*non-isomorphic-families* perms F) \subseteq set F
 <proof>

lemma *non-isomorphic-families-aux*:
 assumes $\forall p \in \text{set perms. } \forall F \in \text{set fams. inj-on (to-fun p) } (\bigcup f\text{-to-set } F)$
 shows $\forall F \in \text{set fams. } \exists F' \in \text{set } (\text{non-isomorphic-families-aux perms fams res}). \text{inj-embed } (f\text{-to-set } F) (f\text{-to-set } F')$
 <proof>

lemma *non-isomorphic-families'*:
 assumes $\forall p \in \text{set perms. } \forall F \in \text{set fams. inj-on (to-fun p) } (\bigcup f\text{-to-set } F)$
 shows $\forall F \in \text{set fams. } \exists F' \in \text{set } (\text{non-isomorphic-families perms fams}).$
 inj-embed (f-to-set F) (f-to-set F')
 <proof>

lemma *generating-subset-non-isomorphic-families*:
 assumes $\forall p \in \text{set perms. } \forall F \in \text{set FF. inj-on (to-fun p) } (\bigcup f\text{-to-set } F)$
 iso-representing-subset ($\bigcirc FF$) FF'
 shows iso-representing-subset ($\bigcirc(\text{non-isomorphic-families perms FF})$) FF'
 <proof>

end

end

12.5.1 Implementation by sets represented by (sorted and distinct) lists

theory *NonIsomorphicFamiliesImpl*

imports *NonIsomorphicFamilies*

Combinatorics

HOL-Library.List-lexord

begin

definition *list-to-fun* :: *nat list* \Rightarrow (*nat* \Rightarrow *nat*) **where**

list-to-fun l = (λ *n. l ! n*)

definition *permute-set-l* :: *nat list* \Rightarrow *nat list* \Rightarrow *nat list* **where**

permute-set-l p A = *sort (map (list-to-fun p) A)*

global-interpretation *SetPermutations-lists*: *SetPermutations* *sd set list-to-fun*

permute-set-l

defines

non-isomorphic-families-aux-l = *SetPermutations-lists.non-isomorphic-families-aux*

and

permute-family-l = *SetPermutations-lists.permute-family* **and**

non-isomorphic-families-l = *SetPermutations-lists.non-isomorphic-families*

<proof>

lemma *nat-list-to-fun-inj-on'*:

assumes *distinct p p* $<\sim\sim> [0..<n]$ $X \subseteq \{0..<\text{length } p\}$

shows *inj-on (list-to-fun p) X*

<proof>

lemma *nat-list-to-fun-inj-on*:

assumes

$\forall p \in \text{set perms. } p <\sim\sim> [0..<n]$ **and**

$\forall F \in \text{set fams. } \bigcup f\text{-to-set-l } F \subseteq \{0..<n\}$

shows $\forall p \in \text{set perms. } \forall F \in \text{set fams. } \text{inj-on } (list\text{-to-fun } p) (\bigcup f\text{-to-set-l } F)$

<proof>

lemma *iso-permute-family-l*:

assumes *iso (f-to-set-l F) (f-to-set-l F')* *sdf F' sdf F*

assumes *dm F n dm F' n perms = permute [0..<n]*

shows $\exists p \in \text{set perms. set } F' = \text{set } (permute\text{-family-l } p F)$

<proof>

```

lemmas non-isomorphic-families-soundness = SetPermutations-lists.non-isomorphic-families '[OF
nat-list-to-fun-inj-on]'
lemmas permute-set-inv-l = SetPermutations-lists.permute-set-inv '[OF nat-list-to-fun-inj-on]'
lemmas iso-representing-subset-non-isomorphic-families-l = SetPermutations-lists.generating-subset-non-isom
nat-list-to-fun-inj-on]

end

```

13 Expressible sets and irreducible families

```

theory IrreducibleFamilies
imports UnionClosed
begin

```

```

definition expressible where
expressible A F  $\longleftrightarrow (\exists F'. F' \subseteq F \wedge F' \neq \{\} \wedge A = \bigcup F')$ 

```

```

lemma expressible A F  $\longleftrightarrow (\exists F' \subseteq F. F' \neq \{\} \wedge (\forall A' \in F'. A' \subseteq A) \wedge \bigcup F' = A)$ 
 $\langle proof \rangle$ 

```

```

lemma expressible-closure1:
  assumes expressible A F
  shows A  $\in$  closure F
 $\langle proof \rangle$ 

```

```

lemma expressible-closure2:
  assumes expressible A F
  shows  $(op \cup A) ' (closure\ F) \subseteq closure\ F$ 
 $\langle proof \rangle$ 

```

```

lemma expressible-closure:
  assumes expressible A F finite F
  shows closure  $(F \cup \{A\}) = closure\ F$ 
 $\langle proof \rangle$ 

```

13.1 Irreducible families

```

definition reducible :: 'a set  $\Rightarrow$  bool where
reducible F  $\longleftrightarrow$ 
 $(\exists A F'. A \in F \wedge F' \subseteq F \wedge F' \neq \{\} \wedge A \notin F' \wedge A = \bigcup F')$ 

```

```

abbreviation irreducible :: 'a set  $\Rightarrow$  bool where
irreducible F  $\equiv \neg reducible\ F$ 

```

```

lemma reducible F  $\longleftrightarrow (\exists A \in F. expressible\ A\ (F - \{A\}))$ 
 $\langle proof \rangle$ 

```

13.2 Removing all expressible sets

function (*domintros*) *remove-expressible* **where**
remove-expressible $F =$
 (*if* ($\exists A \in F. \text{expressible } A (F - \{A\})$) *then*
 let $A = \text{SOME } A. A \in F \wedge \text{expressible } A (F - \{A\})$
 in *remove-expressible* $(F - \{A\})$
 else
 F)
<proof>

lemma *remove-expressible-dom*:
 assumes *finite* F
 shows *remove-expressible-dom* F
<proof>

lemma *remove-expressible-subset*:
 assumes *finite* F
 shows *remove-expressible* $F \subseteq F$
<proof>

lemma *remove-expressible-closure*:
 assumes *finite* F
 shows *closure* $F = \text{closure } (\text{remove-expressible } F)$
<proof>

lemma *remove-expressible-irreducible*:
 assumes *finite* F
 shows *irreducible* $(\text{remove-expressible } F)$
<proof>

lemma *ex-irreducible-closure*:
 fixes F
 assumes *finite* F
 shows $\exists F' \subseteq F. \text{irreducible } F' \wedge \text{closure } F' = \text{closure } F$
<proof>

13.3 Uniqueness of irreducible subfamily for the given closure

lemma *irreducible-closure'*:
 assumes *irreducible* F' **and** *closure* $F = \text{closure } F'$
 shows $F' \subseteq F$
<proof>

lemma *irreducible-closure*:
 assumes *irreducible* F **and** *irreducible* F' **and** *closure* $F = \text{closure } F'$
 shows $F = F'$
<proof>

end

13.3.1 Implementation by sorted and distinct lists

theory *IrreducibleFamiliesImpl*
imports *IrreducibleFamilies UnionClosedImpl Combinatorics*
begin

definition *expressible-l* **where**
expressible-l $A\ F \equiv A \in \text{set } (\text{map } \text{Union-l } (\text{all-nonempty-subsets } F))$

lemma *expressible-l-soundness*:
assumes *expressible-l* $A\ F$
shows *expressible* $(\text{set } A) (f\text{-to-set-l } F)$
 $\langle \text{proof} \rangle$

lemma *f-to-set-l-subset-ex*:
fixes $F'::\text{nat set set}$
assumes $F' \subseteq f\text{-to-set-l } Fl$
shows $\exists\ F'l. \text{set } F'l \subseteq \text{set } Fl \wedge f\text{-to-set-l } F'l = F' \wedge \text{distinct } F'l \wedge \text{length } F'l \leq \text{length } Fl$
 $\langle \text{proof} \rangle$

lemma *expressible-l-completeness-lemma*:
assumes $F' \subseteq f\text{-to-set-l } Fl\ F' \neq \{\}$
shows $\bigcup F' \in \text{set } (\text{map set } (\text{map } \text{Union-l } (\text{all-nonempty-subsets } Fl)))$
 $\langle \text{proof} \rangle$

lemma *expressible-l-completeness*:
assumes $sd\ A\ sdf\ F$
assumes
expressible $(\text{set } A) (f\text{-to-set-l } F)$
shows *expressible-l* $A\ F$
 $\langle \text{proof} \rangle$

end

14 Covering

theory *Covering*
imports *UnionClosed Frankl IsomorphicFamilies IrreducibleFamilies*
begin

14.1 Definition of Covering

definition *FC-covered* **where**

$FC\text{-covered } Fc F \equiv \exists Fc'. \text{ iso } Fc' Fc \wedge Fc' \subseteq \text{closure } F$

definition *nonFC-covered* **where**

$\text{nonFC-covered } Nc F \equiv \exists Nc'. \text{ iso } Nc Nc' \wedge \text{closure } F \subseteq \text{closure } Nc' \cup \{\{\}\}$

abbreviation *FCs-covered* **where**

$FCs\text{-covered } \mathcal{F} F \equiv \exists Fc \in \mathcal{F}. FC\text{-covered } Fc F$

abbreviation *nonFCs-covered* **where**

$\text{nonFCs-covered } \mathcal{N} F \equiv \exists Nc \in \mathcal{N}. \text{nonFC-covered } Nc F$

definition *covered* **where**

$\text{covered } \mathcal{F} \mathcal{N} F \equiv FCs\text{-covered } \mathcal{F} F \vee \text{nonFCs-covered } \mathcal{N} F$

lemma *FC-covered-sound*:

fixes $Fc :: \text{nat set set}$

assumes *finite* F **and** $FC\text{-covered } Fc F$ **and** *FC-family* Fc

shows *FC-family* F

$\langle \text{proof} \rangle$

lemma *nonFC-covered-sound*:

fixes $F :: \text{nat set set}$

assumes *finite* F' **and** $\text{nonFC-covered } F' F$ **and** $\neg \text{FC-family } F'$

shows $\neg \text{FC-family } F$

$\langle \text{proof} \rangle$

lemma *FC-covered-mono*:

assumes $F \subseteq F'$ $FC\text{-covered } \mathcal{F} F$

shows $FC\text{-covered } \mathcal{F} F'$

$\langle \text{proof} \rangle$

lemma *nonFC-covered-mono*:

assumes $F \supseteq F'$ $\text{nonFC-covered } \mathcal{F} F$

shows $\text{nonFC-covered } \mathcal{F} F'$

$\langle \text{proof} \rangle$

14.2 Covering and empty set

lemma *FC-covered-remove-empty*:

assumes *finite* F

shows $FC\text{-covered } Fc (F - \{\{\}\}) \longrightarrow FC\text{-covered } Fc F$

$\langle \text{proof} \rangle$

lemma *nonFC-covered-remove-empty*:

assumes *finite* F

shows $\text{nonFC-covered } Nc (F - \{\{\}\}) \longrightarrow \text{nonFC-covered } Nc F$

$\langle \text{proof} \rangle$

lemma *covered-remove-empty*:

assumes $\text{finite } F$
shows $\text{covered } \mathcal{F} \mathcal{N} (F - \{\{\}\}) \longrightarrow \text{covered } \mathcal{F} \mathcal{N} F$
 $\langle \text{proof} \rangle$

14.3 Covering and isomorphic families

lemma FC-covered-iso :
assumes $\text{iso } F F'$ **and** $\text{FC-covered } Fc F$
shows $\text{FC-covered } Fc F'$
 $\langle \text{proof} \rangle$

lemma nonFC-covered-iso :
fixes $F F' :: \text{nat set set}$
assumes $\text{iso } F F'$ **and** $\text{nonFC-covered } Nc F$ **and**
 $\text{finite } (\bigcup F')$ **and** $\text{finite } (\bigcup Nc)$
shows $\text{nonFC-covered } Nc F'$
 $\langle \text{proof} \rangle$

lemma covered-iso :
fixes $F F' :: \text{nat set set}$
assumes $\text{finite } (\bigcup F')$ **and**
 $\forall Fc \in \mathcal{F}. \text{finite } (\bigcup Fc)$ **and** $\forall Nc \in \mathcal{N}. \text{finite } (\bigcup Nc)$
assumes $\text{iso } F F'$
shows $\text{covered } \mathcal{F} \mathcal{N} F \implies \text{covered } \mathcal{F} \mathcal{N} F'$
 $\langle \text{proof} \rangle$

lemma $\text{iso-represents-FCs-covered}$:
fixes $FF FFb :: \text{nat set set set}$
assumes $\text{iso-represents } FFb FF$ **and** $\forall F \in FFb. \text{FCs-covered } \mathcal{F} F$
shows $\forall F \in FF. \text{FCs-covered } \mathcal{F} F$
 $\langle \text{proof} \rangle$

lemma $\text{iso-represents-nonFCs-covered}$:
fixes $FF FFb :: \text{nat set set set}$
assumes $\forall F \in FF. \text{finite } (\bigcup F) \forall F \in \mathcal{N}. \text{finite } (\bigcup F)$
assumes $\text{iso-represents } FFb FF \forall F \in FFb. \text{nonFCs-covered } \mathcal{N} F$
shows $\forall F \in FF. \text{nonFCs-covered } \mathcal{N} F$
 $\langle \text{proof} \rangle$

14.4 Covering, closure and irreducible families

lemma closure-covered :
assumes $\text{closure } F = \text{closure } F'$
shows $\text{covered } \mathcal{F} \mathcal{N} F \longleftrightarrow \text{covered } \mathcal{F} \mathcal{N} F'$
 $\langle \text{proof} \rangle$

lemma $\text{ex-irreducible-covered}$:
fixes F
assumes $\text{finite } (\bigcup F)$
shows $\exists F' \subseteq F. \text{irreducible } F' \wedge (\text{covered } \mathcal{F} \mathcal{N} F \longleftrightarrow \text{covered } \mathcal{F} \mathcal{N} F')$

$\langle \text{proof} \rangle$

lemma *all-irreducible-covered-all-covered*:

assumes $\forall F. \text{irreducible } F \wedge \bigcup F \subseteq \{0..<n::\text{nat}\} \longrightarrow \text{covered } \mathcal{F} \mathcal{N} F$

shows $\forall F. \bigcup F \subseteq \{0..<n\} \longrightarrow \text{covered } \mathcal{F} \mathcal{N} F$

$\langle \text{proof} \rangle$

end

14.4.1 FC and nonFC covering implementation

theory *CoveringImpl*

imports

Covering

FamilyImpl UnionClosedImpl NonIsomorphicFamiliesImpl

More.MoreBinomial

begin

definition *FC-covered-l* :: $\text{nat list list} \Rightarrow \text{nat list list} \Rightarrow \text{nat list list} \Rightarrow \text{bool}$ **where**

$\text{FC-covered-l } Fc F \text{ perms} \longleftrightarrow \text{list-ex } (\lambda Fc'. \text{set } Fc' \subseteq \text{set } (\text{close-l } F)) (\text{map } (\lambda p. \text{permute-family-l } p Fc) \text{ perms})$

lemma *FC-covered-l-soundness*:

assumes $\forall p \in \text{set perms}. p <\sim\sim> [0..<n] \text{ dm } Fc n$

assumes $\text{FC-covered-l } Fc F \text{ perms}$

shows $\text{FC-covered } (f\text{-to-set-l } Fc) (f\text{-to-set-l } F)$

$\langle \text{proof} \rangle$

lemma *FC-covered-l-completeness*:

assumes $\text{sdf } F \text{ sdf } Fc \text{ dm } Fc n \text{ dm } F n \text{ perms} = \text{permute } [0..<n]$

assumes $\text{FC-covered } (f\text{-to-set-l } Fc) (f\text{-to-set-l } F)$

shows $\text{FC-covered-l } Fc F \text{ perms}$

$\langle \text{proof} \rangle$

definition *FCs-covered-l* **where**

$\text{FCs-covered-l } \mathcal{F} F \text{ perms} \longleftrightarrow \text{list-ex } (\lambda Fc. \text{FC-covered-l } Fc F \text{ perms}) \mathcal{F}$

definition *nonFC-covered-l* **where**

$\text{nonFC-covered-l } Nc F \text{ perms} \longleftrightarrow$

$\text{list-ex } (\lambda Nc'. \text{set } (\text{close-l } F) \subseteq \text{set } Nc')$

$(\text{map } (\lambda p. (\text{close-insert-empty-l } (\text{permute-family-l } p Nc))) \text{ perms})$

lemma *nonFC-covered-l-soundness*:

assumes $\forall p \in \text{set perms}. p <\sim\sim> [0..<n] \text{ dm } Nc n$

assumes $\text{nonFC-covered-l } Nc F \text{ perms}$

shows *nonFC-covered* (*f-to-set-l* *Nc*) (*f-to-set-l* *F*)
 <proof>

definition *nonFCs-covered-l* **where**
nonFCs-covered-l \mathcal{N} *F* *perms* \longleftrightarrow
list-ex (λ *Nc*. *nonFC-covered-l* *Nc* *F* *perms*) \mathcal{N}

lemma *nonFCs-covered-l-soundness*:
assumes $\forall p \in \text{set } \text{perms}. p < \sim \sim > [0..<n]$ *dmf* \mathcal{N} *n*
assumes *nonFCs-covered-l* \mathcal{N} *F* *perms*
shows *nonFCs-covered* (*fs-to-set-l* \mathcal{N}) (*f-to-set-l* *F*)
 <proof>

definition *nonFC-covered-l-opt* :: *nat list list list* \Rightarrow *nat list list* \Rightarrow *bool* **where**
nonFC-covered-l-opt *Nc-perms* *F* \longleftrightarrow
list-ex (λ *Nc'*. *set* (*close-l* *F*) \subseteq *set* *Nc'*) *Nc-perms*

definition *nonFCs-covered-l-opt* **where**
nonFCs-covered-l-opt \mathcal{N} -*perms* *F* \longleftrightarrow
list-ex (λ *Nc*. *nonFC-covered-l-opt* *Nc* *F*) \mathcal{N} -*perms*

lemma *nonFCs-covered-l-opt*:
assumes \mathcal{N} -*perms* = (*map* (λ *Nc*. (*map* (λ *p*. *close-insert-empty-l* (*permute-family-l* *p* *Nc*)) *perms*)) \mathcal{N})
shows *nonFCs-covered-l-opt* \mathcal{N} -*perms* *F* \longleftrightarrow *nonFCs-covered-l* \mathcal{N} *F* *perms*
 <proof>

end

15 L-partitioned families

theory *LPartitioning*
imports *Main More.MoreSet IsomorphicFamilies*
begin

definition *is-L-part* **where**
is-L-part *n* *L* *F* \longleftrightarrow
 $\bigcup F \subseteq \{0..<n::\text{nat}\} \wedge$
 $(\forall A \in F. \text{card } A < \text{length } L) \wedge$
 $(\forall n < \text{length } L. \text{card } \{A \in F. \text{card } A = n\} = L ! n)$

abbreviation *L-part* **where**
L-part *n* *L* $\equiv \{F. \text{is-L-part } n \text{ } L \text{ } F\}$

lemma *is-L-part-finite*:
assumes *is-L-part* *n* *L* *F*
shows *finite* *F* $\forall A \in F. \text{finite } A$

$\langle \text{proof} \rangle$

lemma *is-L-part-zero*:

shows $\text{is-L-part } n \ L \ F \longleftrightarrow \text{is-L-part } n \ (L @ [0]) \ F$

$\langle \text{proof} \rangle$

lemma *is-L-part-zeros*:

assumes $\forall \ k'. \ k' > k \wedge k' < \text{length } L \longrightarrow L ! k' = 0$

shows $\text{is-L-part } n \ L \ F \longleftrightarrow \text{is-L-part } n \ (\text{take } (k+1) \ L) \ F$

$\langle \text{proof} \rangle$

lemma *is-L-part-zeros-replicate*:

$\text{is-L-part } n \ (\text{replicate } k \ 0) \ F \longleftrightarrow \text{is-L-part } n \ [] \ F$

$\langle \text{proof} \rangle$

lemma *is-L-part-mem*:

assumes $\text{is-L-part } n \ L \ F \ A \in F$

shows $\text{card } A < \text{length } L \wedge L ! \text{card } A > 0$

$\langle \text{proof} \rangle$

lemma *is-L-part-empty-mem*:

assumes

$\text{is-L-part } n \ L \ F \ \text{hd } L > 0 \ \text{length } L > 0$

shows $\{\} \in F$

$\langle \text{proof} \rangle$

lemma *is-L-part-remove*:

assumes $A \in F \ \text{is-L-part } n \ L \ F$

shows $\text{is-L-part } n \ (L[\text{card } A := (L ! \text{card } A) - 1]) \ (F - \{A\})$

$\langle \text{proof} \rangle$

lemma *is-L-part-remove-last*:

assumes $\text{is-L-part } n \ (L @ [k + 1]) \ F \ \text{card } A = \text{length } L \ A \in F$

shows $\text{is-L-part } n \ (L @ [k]) \ (F - \{A\})$

$\langle \text{proof} \rangle$

lemma *is-L-part-insert-last*:

assumes $\text{is-L-part } n \ (L @ [k]) \ F \ A \notin F \ A \subseteq \{0..<n\} \ \text{card } A = \text{length } L$

shows $\text{is-L-part } n \ (L @ [k + 1]) \ (F \cup \{A\})$

$\langle \text{proof} \rangle$

15.1 Ordering of L-partitions

definition *pwge* (**infixl** \succeq 100) **where**

$L' \succeq L \longleftrightarrow (\text{length } L' = \text{length } L) \wedge (\forall \ i. \ i < \text{length } L \longrightarrow L' ! i \geq L ! i)$

lemma *pwge-Cons*:

$(a \# L) \succeq (b \# L') \longleftrightarrow (a \geq b \wedge L \succeq L')$

$\langle proof \rangle$

lemma *pwge-Nil*:

$\Box \succeq \Box$

$\langle proof \rangle$

lemma *pwge-refl*:

fixes $L::'a::linorder\ list$

shows $[simp]: L \succeq L$

$\langle proof \rangle$

lemma *pwge-trans* $[trans]$:

fixes $L::'a::linorder\ list$

assumes $L \succeq L' \ L' \succeq L''$

shows $L \succeq L''$

$\langle proof \rangle$

lemma *pwge-replicate-0*:

assumes $length\ X = n$

shows $X \succeq replicate\ n\ (0::nat)$

$\langle proof \rangle$

lemma *pwge-list-update*:

fixes $L::nat\ list$

assumes $n < length\ L \ nk \leq L\ !\ n$

shows $L \succeq L\ [n := nk]$

$\langle proof \rangle$

lemma *is-L-part-subset*:

assumes $L' \succeq L \ is-L-part\ n\ L'\ F'$

shows $\exists\ F. F \subseteq F' \wedge is-L-part\ n\ L\ F$

$\langle proof \rangle$

lemma *is-L-part-subset'*:

assumes $F' \subseteq F \ is-L-part\ n\ L\ F$

shows $\exists\ L'. L \succeq L' \wedge is-L-part\ n\ L'\ F'$

$\langle proof \rangle$

16 Generating all L-partitioned families with some given properties

abbreviation *L-part-P* **where**

$L-part-P\ P\ n\ L \equiv \{F \in L-part\ n\ L. P\ F\}$

abbreviation *mult-P* **where**

$mult-P\ P\ F\ A \equiv \{f \cup \{A\} \mid f. f \in F \wedge A \notin f \wedge P\ A\ f\}$

abbreviation

$mult-all-P\ P\ F\ n\ m \equiv \bigcup \{mult-P\ P\ F\ A \mid A. card\ A = m \wedge A \subseteq \{0..<n\}\}$

abbreviation *incrementally-checks* **where**

incrementally-checks $Pinc\ P \equiv$

$$(\forall\ A\ F. (finite\ (\bigcup\ F\ \cup\ A) \wedge (\forall\ A' \in F. card\ A \geq card\ A') \wedge A \notin F) \longrightarrow \\ (P\ (F \cup \{A\}) \longleftrightarrow P\ F \wedge Pinc\ A\ F))$$

lemma *L-part-mult-P*:

assumes *incrementally-checks* $Pinc\ P$

shows $L\text{-part-}P\ P\ n\ (L\ @\ [k + 1]) = mult\text{-all-}P\ Pinc\ (L\text{-part-}P\ P\ n\ (L\ @\ [k]))$
 $n\ (length\ L)$

(**is** *?lhs = ?rhs*)

$\langle proof \rangle$

abbreviation *inj-preserved* **where**

inj-preserved $P \equiv \forall\ A\ F\ f. (inj\text{-on}\ f\ (\bigcup\ F\ \cup\ A) \wedge P\ A\ F) \longrightarrow P\ (f\ 'A)\ (op\ 'f\ 'F)$

lemma *L-part-mult-iso-representing-subset*:

assumes

incrementally-checks $Pinc\ P$ *inj-preserved* $Pinc$

iso-representing-subset $FFb\ (L\text{-part-}P\ P\ n\ (L\ @\ [k]))$ (**is** *iso-representing-subset*
 $- ?FFk$)

$length\ L \leq n$

$FFb' = mult\text{-all-}P\ Pinc\ FFb\ n\ (length\ L)$

shows *iso-representing-subset* $FFb'\ (L\text{-part-}P\ P\ n\ (L\ @\ [k + 1]))$ (**is** *iso-representing-subset*
 $?FFb'\ ?FFk1$)

$\langle proof \rangle$

end

16.1 Implementation by sorted and distinct lists

theory *LPartitioningImpl*

imports *LPartitioning* *FamilyImpl* *NonIsomorphicFamiliesImpl*

begin

definition *mult-P-l* **where**

mult-P-l $P\ F\ A =$

$$(let\ F' = [f \leftarrow F. A \notin set\ f \wedge P\ A\ f]\ in \\ map\ (\lambda\ x. A\ \# x)\ F')$$

lemma *mult-P-l-sorted-distinct*:

assumes *sdff* $F\ sd\ A$

shows *sdff* $(mult\text{-P-l}\ P\ F\ A)$

$\langle proof \rangle$

lemma *mult-P-l-domain*:

assumes *dmf* $FFb\ n\ set\ A \subseteq \{0..<n::nat\}$

shows *dmf* $(mult\text{-P-l}\ P\ FFb\ A)\ n$

$\langle proof \rangle$

lemma *mult-P-l-correctness*:
fixes $FFb :: \text{nat list list list}$
assumes $\text{sdff } FFb \text{ sd } A \text{ dmf } FFb \text{ n set } A \subseteq \{0..<n\}$
assumes $\bigwedge A F. \llbracket \text{sd } A; \text{sdf } F; \text{dm } (A \# F) \text{ n} \rrbracket \implies P \ A \ F \longleftrightarrow P' \ (\text{set } A)$
(f-to-set-l F)
shows $\text{fs-to-set-l } (\text{mult-P-l } P \ FFb \ A) = \text{mult-P } P' \ (\text{fs-to-set-l } FFb) \ (\text{set } A)$ **(is**
 $?lhs = ?rhs)$
 $\langle \text{proof} \rangle$

definition *mult-all-P-l where*
 $\text{mult-all-P-l } P \ F \ n \ m = \text{concat } (\text{map } (\lambda A. \text{mult-P-l } P \ F \ A) \ (\text{all-mn-subsets } n \ m))$

lemma *mult-all-P-l-sorted-distinct*:
assumes $\text{sdff } F$
shows $\text{sdff } (\text{mult-all-P-l } P \ F \ n \ m)$
 $\langle \text{proof} \rangle$

lemma *mult-all-P-l-domain*:
assumes $\text{dmf } FFb \ n$
shows $\text{dmf } (\text{mult-all-P-l } P \ FFb \ n \ m) \ n$
 $\langle \text{proof} \rangle$

lemma *mult-all-P-correctness*:
fixes $FFb :: \text{nat list list list}$
assumes $\text{sdff } FFb \text{ dmf } FFb \ n$
 $\bigwedge A F. \llbracket \text{sd } A; \text{sdf } F; \text{dm } (A \# F) \text{ n} \rrbracket \implies P \ A \ F \longleftrightarrow P' \ (\text{set } A) \text{ (f-to-set-l } F)$
shows $\text{fs-to-set-l } (\text{mult-all-P-l } P \ FFb \ n \ m) = \text{mult-all-P } P' \ (\text{fs-to-set-l } FFb) \ n \ m$
(is $?lhs = ?rhs)$
 $\langle \text{proof} \rangle$

definition *mult-all-base-P-l where*
 $\text{mult-all-base-P-l } P \ F \ n \ m \ \text{perms} = \text{non-isomorphic-families-l perms } (\text{mult-all-P-l } P \ F \ n \ m)$

lemma *mult-all-base-P-l-sorted-distinct*:
assumes $\text{sdff } F$
shows $\text{sdff } (\text{mult-all-base-P-l } P \ F \ n \ m \ \text{perms})$
 $\langle \text{proof} \rangle$

lemma *mult-all-base-P-l-domain*:
assumes $\text{dmf } F \ n$
shows $\text{dmf } (\text{mult-all-base-P-l } P \ F \ n \ m \ \text{perms}) \ n$
 $\langle \text{proof} \rangle$

lemma *mult-all-base-P-l-correctness*:

assumes $\bigwedge A F. \llbracket sd\ A; sdf\ F; dm\ (A \# F)\ n \rrbracket \implies Pinc'\ A\ F \longleftrightarrow Pinc\ (set\ A)\ (f\text{-to-set-l}\ F)$
incrementally-checks $Pinc\ P$ *inj-preserved* $Pinc$
assumes $sdf\ FFb\ dm\ FFb\ n\ \forall p \in set\ perms. p <\sim\sim> [0..<n]$
assumes *iso-representing-subset* $(fs\text{-to-set-l}\ FFb)\ (L\text{-part-}P\ P\ n\ (L\ @\ [k]))$
 $length\ L \leq n$
 $FFb' = fs\text{-to-set-l}\ (mult\text{-all-base-}P\text{-l}\ Pinc'\ FFb\ n\ (length\ L)\ perms)$
shows *iso-representing-subset* $FFb'\ (L\text{-part-}P\ P\ n\ (L\ @\ [k+1]))$
 $\langle proof \rangle$

16.2 Recursive enumeration procedure

For a given list L , find an iso-representing collection of all L -partitioned families that satisfy some predicate P .

abbreviation *dec-last* **where**
 $dec\text{-last}\ L \equiv butlast\ L\ @\ [last\ L - (1::nat)]$

function *enum-rec* **where**
 $enum\text{-rec}\ L\ v0\ f =$
 $(if\ L = []\ then\ v0$
 $\quad else\ if\ last\ L = 0\ then\ enum\text{-rec}\ (butlast\ L)\ v0\ f$
 $\quad else\ f\ (enum\text{-rec}\ (dec\text{-last}\ L)\ v0\ f)\ L)$
 $\langle proof \rangle$

termination

$\langle proof \rangle$

declare $enum\text{-rec}.simps[simp\ del]$

lemma *enum-rec*:
assumes $P\ v0\ []$
assumes $\bigwedge v\ L. P\ v\ L \implies P\ v\ (L\ @\ [0])$
 $\bigwedge v\ L'::(nat\ list). \llbracket L' \neq []; last\ L' > 0; length\ L' \leq length\ L; P\ v\ (dec\text{-last}\ L') \rrbracket \implies P\ (f\ v\ L')\ L'$
shows $P\ (enum\text{-rec}\ L\ v0\ f)\ L$
 $\langle proof \rangle$

lemma *enum-rec-iso-representing-subset*:
assumes $\bigwedge A F. \llbracket sd\ A; sdf\ F; dm\ (A \# F)\ n \rrbracket \implies Pinc'\ A\ F \longleftrightarrow Pinc\ (set\ A)\ (f\text{-to-set-l}\ F)$ **and**
incrementally-checks $Pinc\ P$ **and** *inj-preserved* $Pinc$ **and**
 $\forall p \in set\ perms. p <\sim\sim> [0..<n]$ **and** $length\ L - 1 \leq n$
assumes $P\ \{\} Mult = (\lambda F\ L. mult\text{-all-base-}P\text{-l}\ Pinc'\ F\ n\ (length\ L - 1)\ perms)$
shows *iso-representing-subset* $(fs\text{-to-set-l}\ (enum\text{-rec}\ L\ [])\ Mult)\ (L\text{-part-}P\ P\ n\ L)$
 $\langle proof \rangle$

16.3 Dynamic programming enumeration procedure

definition *pwge-impl* **where**

$pwge\text{-impl}\ L\ L' \longleftrightarrow list\text{-all}\ (\lambda (x, y). x \geq y)\ (zip\ L\ L')$

lemma *pwge-impl*:

assumes $\text{length } L = \text{length } L'$

shows $\text{pwge-impl } L \ L' \longleftrightarrow L \succeq L'$

$\langle \text{proof} \rangle$

function *enum-dp* **where**

enum-dp $\text{val res } k \ L \ g \ \text{stop } \text{maxL} = \text{foldl}$

$(\lambda \ r \ n. \ \text{let } l = \text{inc-nth } L \ n$

$\text{in if } \text{length } l \neq \text{length } \text{maxL} \vee \text{stop } l \vee \neg \text{pwge-impl } \text{maxL } l \text{ then}$

r

else

$\text{enum-dp } (g \ \text{val } n) \ r \ n \ l \ g \ \text{stop } \text{maxL}$

$)$

$(\text{val } \# \ \text{res})$

$[k..<\text{length } L]$

$\langle \text{proof} \rangle$

abbreviation *enum-dp-step* **where**

enum-dp-step $\text{val res } k \ g \ \text{stop } \text{maxL} \ L \equiv \text{let } l = \text{inc-nth } L \ k$

$\text{in if } \text{length } l \neq \text{length } \text{maxL} \vee$

$\text{stop } l \vee$

$\neg \text{pwge-impl } \text{maxL } l$

$\text{then res else enum-dp } (g \ \text{val } k) \ \text{res } k \ l \ g \ \text{stop } \text{maxL}$

definition *listdiff* **where**

listdiff $xs \ ys = \text{sum-list } (\text{map } (\lambda \ (a, \ b). \ a - b) \ (\text{zip } xs \ ys))$

termination

$\langle \text{proof} \rangle$

declare *enum-dp.simps* $[\text{simp del}]$

lemma *enum-dp-mono*:

assumes $\text{length } L = \text{length } \text{maxL}$

shows $\text{set } (\text{val } \# \ \text{res}) \subseteq \text{set } (\text{enum-dp } \text{val res } k \ L \ g \ \text{stop } \text{maxL})$

$\langle \text{proof} \rangle$

lemma *enum-dp-mono'*:

assumes $\text{length } L = \text{length } \text{maxL}$

shows $\text{set res} \subseteq \text{set } (\text{foldl } (\lambda \ r \ n. \ \text{enum-dp-step } \text{val } r \ n \ g \ \text{stop } \text{maxL} \ L) \ \text{res} \ [k..<\text{length } L])$

$\langle \text{proof} \rangle$

lemma *enum-dp-lemma*:

assumes

$\text{length } L = \text{length } \text{maxL}$ **and**

$\forall \ k''. \ k'' > k \wedge k'' < \text{length } L \longrightarrow L ! \ k'' = 0$ **and**

$X = \{L'. \ L' \succeq L \wedge \text{maxL} \succeq L' \wedge \text{take } k \ L' = \text{take } k \ L \wedge \neg (\exists \ Ls. \ Ls \succeq L \wedge \text{take } k \ L = \text{take } k \ Ls \wedge L' \succeq Ls \wedge \text{stop } Ls)\}$ **and**

$P \text{ val } L \text{ and}$
 $k < \text{length } L \text{ and}$
 $\bigwedge \text{ val } L \ k'. \llbracket P \text{ val } L; \ k' < \text{length } L; \ k' < \text{length } \text{max}L; \forall \ k''. \ k'' > k' \wedge k'' < \text{length } L \longrightarrow L \ ! \ k'' = 0 \rrbracket \Longrightarrow$
 $P \ (g \text{ val } k') \ (\text{inc-nth } L \ k')$
shows $\forall \ L' \in X. \exists \ B \in \text{set} \ (\text{enum-dp val res } k \ L \ g \ \text{stop } \text{max}L). \ P \ B \ L'$
 $\langle \text{proof} \rangle$

Incrementally build all generating subsets for L-partitioned lists upto the given bounds

definition *enum-dp-mult-P* **where**

$\text{enum-dp-mult-P } P \ n \ \text{perms} \ \text{stops} \ \text{max}L =$
 $\text{enum-dp } [\Box] \ \Box \ 0 \ (\text{replicate } (n+1) \ (0::\text{nat}))$
 $(\lambda \text{ FFb } m. \text{mult-all-base-P-l } P \ \text{FFb } n \ m \ \text{perms})$
 $(\lambda \ L. \text{list-ex } (\lambda \ L'. \text{pwge-impl } L \ L') \ \text{stops})$
 $\text{max}L$

lemma *enum-dp-mult-P-correct*:

assumes $\bigwedge \ A \ F. \llbracket \text{sd } A; \text{sdf } F; \text{dm } (A \ \# \ F) \ n \rrbracket \Longrightarrow \text{Pinc}' \ A \ F \longleftrightarrow \text{Pinc} \ (\text{set } A)$
 $(f\text{-to-set-l } F)$

incrementally-checks Pinc P inj-preserved Pinc

assumes

$X = \{L'. \text{max}L \succeq L' \wedge \neg (\exists \ S \in \text{set stops}. L' \succeq S)\} \ (\text{is } - = ?\text{lhs}) \text{ and}$

$P \ \{\}$ **and**

$\text{length } \text{max}L = n+1 \text{ and}$

$\forall \ S \in \text{set stops}. \text{length } S = n+1 \text{ and}$

$\forall \ p \in \text{set perms}. \ p <\sim\sim> [0..<n]$

shows $\forall \ L' \in X. \exists \ B \in \text{set} \ (\text{enum-dp-mult-P } \text{Pinc}' \ n \ \text{perms} \ \text{stops} \ \text{max}L).$
 $\text{iso-representing-subset } (f\text{-to-set-l } B) \ (L\text{-part-P } P \ n \ L')$

$\langle \text{proof} \rangle$

end

16.4 Generating all irreducible families

theory *LPartitioningIrreducible*

imports *LPartitioning IrreducibleFamilies*

begin

lemma *irreducible-expressible*:

incrementally-checks $(\lambda \ A \ F. \neg \text{expressible } A \ F) \text{ irreducible}$

$\langle \text{proof} \rangle$

lemma *not-expressible-iso*:

assumes $\neg \text{expressible } A \ F \text{ and } \text{inj-on } f \ (\bigcup \ F \cup A)$

shows $\neg \text{expressible } (f \text{ ' } A) \ (\text{op ' } f \text{ ' } F)$

$\langle \text{proof} \rangle$

lemma *inj-preserved-not-expressible*:

shows *inj-preserved* $(\lambda A F. \neg \text{expressible } A F)$
 $\langle \text{proof} \rangle$

abbreviation *L-part-irreducible* **where**

$L\text{-part-irreducible} \equiv L\text{-part-}P \text{ irreducible}$

abbreviation *mult-irreducible* **where**

$\text{mult-irreducible} \equiv \text{mult-}P (\lambda A F. \neg \text{expressible } A F)$

abbreviation *mult-all-irreducible* **where**

$\text{mult-all-irreducible} \equiv \text{mult-all-}P (\lambda A F. \neg \text{expressible } A F)$

lemma *L-part-irreducible-mult-irreducible*:

$L\text{-part-irreducible } n (L @ [k+1]) = \text{mult-all-irreducible } (L\text{-part-irreducible } n (L @ [k])) n (\text{length } L)$
 $\langle \text{proof} \rangle$

lemmas *L-part-irreducible-mult-irreducible-iso-representing-subset* =

$L\text{-part-mult-iso-representing-subset}[\text{of irreducible } \lambda A F. \neg \text{expressible } A F, \\ \text{OF irreducible-expressible inj-preserved-not-expressible}]$

end

16.5 Generating all families that are not FCs covered by the given collection

theory *LPartitioningNonFCsCovered*

imports *LPartitioning Covering*

begin

lemma *pwge-FCs-covered*:

assumes $\forall F \in L\text{-part } n L. \text{FCs-covered } \mathcal{F} F L' \succeq L$

shows $\forall F \in L\text{-part } n L'. \text{FCs-covered } \mathcal{F} F$

$\langle \text{proof} \rangle$

lemma *notFCs-covered*:

$\text{incrementally-checks } (\lambda A F. \neg \text{FCs-covered } \mathcal{F} (F \cup \{A\})) (\lambda F. \neg \text{FCs-covered } \mathcal{F} F)$

$\langle \text{proof} \rangle$

lemma *notFCs-covered-iso*:

assumes $\neg \text{FCs-covered } \mathcal{F} (F \cup \{A\})$ **and** *inj-on* $f (\bigcup F \cup A)$

shows $\neg \text{FCs-covered } \mathcal{F} ((\text{op } 'f' F) \cup \{f' A\})$

$\langle \text{proof} \rangle$

lemma *inj-preserved-not-FCs-covered*:

$\text{inj-preserved } (\lambda A F. \neg \text{FCs-covered } \mathcal{F} (F \cup \{A\}))$

$\langle \text{proof} \rangle$

abbreviation *L-part-notFCs-covered* **where**

$L\text{-part-notFCs-covered } \mathcal{F} \equiv L\text{-part-}P (\lambda F. \neg \text{FCs-covered } \mathcal{F} F)$

abbreviation *mult-all-notFCs-covered* **where**

mult-all-notFCs-covered $\mathcal{F} \equiv \text{mult-all-}P (\lambda A f. \neg \text{FCs-covered } \mathcal{F} (f \cup \{A\}))$

lemma *L-part-mult-iso-representing-subset-notFC-covered*:

assumes

iso-representing-subset FFb (*L-part-notFCs-covered* \mathcal{F} n ($L @ [k]$))

$FFb' = \text{mult-all-notFCs-covered } \mathcal{F} FFb$ n ($\text{length } L$)

$\text{length } L \leq n$

shows *iso-representing-subset* FFb' (*L-part-notFCs-covered* \mathcal{F} n ($L @ [k + 1]$))

$\langle \text{proof} \rangle$

end

16.5.1 Implementation

theory *LPartitioningNonFCsCoveredImpl*

imports *LPartitioningNonFCsCovered CoveringImpl LPartitioningImpl*

begin

lemma *notFCs-covered-l*:

assumes $\text{perms} = \text{permute } [0..<n]$ $\text{dmf } \mathcal{F}$ n $\text{sdff } \mathcal{F}$ $\text{dm } (A \# F)$ n
 $\text{sd } A$ $\text{sd } F$

shows

$\neg \text{FCs-covered-l } \mathcal{F} (A \# F) \text{ perms} \longleftrightarrow$

$\neg \text{FCs-covered } (\text{fs-to-set-l } \mathcal{F}) (\text{f-to-set-l } F \cup \{\text{set } A\})$

$\langle \text{proof} \rangle$

definition *mult-notFCs-covered-l* **where**

mult-notFCs-covered-l $\mathcal{F} F A \text{ perms} =$

($\text{let } F' = \text{filter } (\lambda f. A \notin \text{set } f) F;$

$F'' = \text{map } (\lambda x. A \# x) F'$

$\text{in filter } (\lambda X. \neg \text{FCs-covered-l } \mathcal{F} X \text{ perms}) F''$)

definition *mult-all-notFCs-covered-l* **where**

mult-all-notFCs-covered-l $\mathcal{F} F n m \text{ perms} = \text{concat } (\text{map } (\lambda A. \text{mult-notFCs-covered-l } \mathcal{F} F A \text{ perms}) (\text{all-mn-subsets } n m))$

definition *mult-all-base-notFCs-covered-l* **where**

mult-all-base-notFCs-covered-l $\mathcal{F} F n m \text{ perms} = \text{non-isomorphic-families-l } \text{perms}$
 $(\text{mult-all-notFCs-covered-l } \mathcal{F} F n m \text{ perms})$

lemma *mult-all-base-notFCs-covered-l*:

mult-all-base-notFCs-covered-l $\mathcal{F} F n m \text{ perms} = \text{mult-all-base-}P\text{-l } (\lambda A F. \neg \text{FCs-covered-l } \mathcal{F} (A \# F) \text{ perms}) F n m \text{ perms}$

$\langle \text{proof} \rangle$

definition *FC-covered-l-opt* $:: \text{nat list list list} \Rightarrow \text{nat list list} \Rightarrow \text{bool}$ **where**

FC-covered-l-opt $Fc\text{-perms } F \longleftrightarrow \text{list-ex } (\lambda Fc'. \text{set } Fc' \subseteq \text{set } (\text{close-l } F))$

Fc-perms

definition *FCs-covered-l-opt* **where**

FCs-covered-l-opt \mathcal{F} -perms $F \longleftrightarrow \text{list-ex } (\lambda Fc. \text{FC-covered-l-opt } Fc \ F) \ \mathcal{F}\text{-perms}$

definition *FCs-covered-l-opt'* **where**

FCs-covered-l-opt' \mathcal{F} -perms $F \longleftrightarrow$
 (let $clF = \text{set } (\text{close-l } F)$
 in $\text{list-ex } (\text{list-ex } (\lambda Fc'. \text{set } Fc' \subseteq clF)) \ \mathcal{F}\text{-perms}$)

lemma [*simp*]: *FCs-covered-l-opt'* = *FCs-covered-l-opt*

$\langle \text{proof} \rangle$

definition *mult-notFCs-covered-l-opt* **where**

mult-notFCs-covered-l-opt \mathcal{F} -perms $F \ A =$
 (let $F' = \text{filter } (\lambda f. A \notin \text{set } f) \ F;$
 $F'' = \text{map } (\lambda x. A \# x) \ F'$
 in $\text{filter } (\lambda X. \neg \text{FCs-covered-l-opt}' \ \mathcal{F}\text{-perms } X) \ F''$)

definition *mult-all-notFCs-covered-l-opt* **where**

mult-all-notFCs-covered-l-opt \mathcal{F} -perms $F \ n \ m = \text{concat } (\text{map } (\lambda A. \text{mult-notFCs-covered-l-opt} \ \mathcal{F}\text{-perms } F \ A) \ (\text{all-mn-subsets } n \ m))$

definition *mult-all-base-notFCs-covered-l-opt* **where**

mult-all-base-notFCs-covered-l-opt \mathcal{F} -perms $F \ n \ m \ \text{perms} = \text{non-isomorphic-families-l} \ \text{perms} \ (\text{mult-all-notFCs-covered-l-opt } \mathcal{F}\text{-perms } F \ n \ m)$

lemma *mult-all-base-notFCs-covered-opt*:

assumes $\mathcal{F}\text{-perms} = \text{map } (\lambda F. \text{map } (\lambda p. \text{permute-family-l } p \ F) \ \text{perms}) \ \mathcal{F}$

shows *mult-all-base-notFCs-covered-l-opt* \mathcal{F} -perms $F \ n \ m \ \text{perms} =$
mult-all-base-notFCs-covered-l $\mathcal{F} \ F \ n \ m \ \text{perms}$

$\langle \text{proof} \rangle$

abbreviation *enum-rec-notFCs-covered-l* **where**

enum-rec-notFCs-covered-l \mathcal{F} -perms $L \ n \ \text{perms} \equiv$
 $\text{enum-rec } L \ [\square] \ (\lambda F \ L. \text{mult-all-base-notFCs-covered-l-opt } \mathcal{F}\text{-perms } F \ n \ (\text{length } L - 1) \ \text{perms})$

lemma *enum-rec-notFCs-covered-l-iso-representing-subset*:

assumes $\mathcal{F}\text{-perms} = \text{map } (\lambda F. \text{map } (\lambda p. \text{permute-family-l } p \ F) \ (\text{permute } [0..<n])) \ \mathcal{F} \ \square \notin \text{set } \mathcal{F}$

$\text{dmf } \mathcal{F} \ n \ \text{sdff } \mathcal{F} \ \text{length } L - 1 \leq n$

shows *iso-representing-subset* (*fs-to-set-l* (*enum-rec-notFCs-covered-l* \mathcal{F} -perms $L \ n \ (\text{permute } [0..<n]))$) (*L-part-notFCs-covered* (*fs-to-set-l* \mathcal{F}) $n \ L$)

$\langle \text{proof} \rangle$

end

16.6 Generating all irreducible families that are not FCs covered by the given collection

theory *LPartitioningIrreducibleNonFCsCovered*
imports *LPartitioningIrreducible LPartitioningNonFCsCovered IrreducibleFamilies*
begin

lemma *irreducible-expressible-notFCs-covered:*

incrementally-checks

$(\lambda A F. \neg \text{expressible } A F \wedge \neg \text{FCs-covered } \mathcal{F} (F \cup \{A\}))$
 $(\lambda F. \text{irreducible } F \wedge \neg \text{FCs-covered } \mathcal{F} F)$

$\langle \text{proof} \rangle$

lemma *irreducible-expressible-notFCs-covered-iso:*

assumes $\neg \text{expressible } A F \wedge \neg \text{FCs-covered } \mathcal{F} (F \cup \{A\})$ *inj-on* $f (\bigcup F \cup A)$

shows $\neg \text{expressible } (f \text{ ` } A) (\text{op ` } f \text{ ` } F) \wedge \neg \text{FCs-covered } \mathcal{F} (\text{op ` } f \text{ ` } F \cup \{f \text{ ` } A\})$

$\langle \text{proof} \rangle$

lemma *inj-preserved-not-expressible-notFCs-covered:*

inj-preserved $(\lambda A F. \neg \text{expressible } A F \wedge \neg \text{FCs-covered } \mathcal{F} (F \cup \{A\}))$

$\langle \text{proof} \rangle$

abbreviation *L-part-irreducible-notFCs-covered where*

L-part-irreducible-notFCs-covered $\mathcal{F} \equiv \text{L-part-P } (\lambda F. \text{irreducible } F \wedge \neg \text{FCs-covered } \mathcal{F} F)$

abbreviation *mult-all-irreducible-notFCs-covered where*

mult-all-irreducible-notFCs-covered $\mathcal{F} \equiv \text{mult-all-P } (\lambda A f. \neg \text{expressible } A f \wedge \neg \text{FCs-covered } \mathcal{F} (f \cup \{A\}))$

lemma *L-part-mult-iso-representing-subset-irreducible-notFC-covered:*

assumes

iso-representing-subset $\text{FFb } (L\text{-part-irreducible-notFCs-covered } \mathcal{F} \text{ } n \text{ } (L @ [k]))$

length $L \leq n$

$\text{FFb}' = \text{mult-all-irreducible-notFCs-covered } \mathcal{F} \text{ } \text{FFb } n \text{ } (\text{length } L)$

shows *iso-representing-subset* $\text{FFb}' (L\text{-part-irreducible-notFCs-covered } \mathcal{F} \text{ } n \text{ } (L @ [k + 1]))$

$\langle \text{proof} \rangle$

lemma *iso-represents-L-part-irreducible-notFCs-covered:*

fixes $\mathcal{F} \mathcal{N} \text{FFb} :: \text{nat set set set}$

assumes *iso-representing-subset* $\text{FFb } (L\text{-part-irreducible-notFCs-covered } \mathcal{F} \text{ } n \text{ } L)$

$\forall F \in \text{FFb}. \text{nonFCs-covered } \mathcal{N} F$

$\forall N \in \mathcal{N}. \text{finite } (\bigcup N)$

shows $\forall F \in L\text{-part-irreducible } n \text{ } L. \text{covered } \mathcal{F} \mathcal{N} F$

$\langle \text{proof} \rangle$

end

16.6.1 Implementation

theory *LPartitioningIrreducibleNonFCsCoveredImpl*
imports *LPartitioningIrreducibleNonFCsCovered IrreducibleFamiliesImpl CoveringImpl*
LPartitioningNonFCsCoveredImpl
begin

lemma *irreducible-expressible-notFCs-covered-l*:

assumes $\text{perms} = \text{permute } [0..<n] \text{ dmf } \mathcal{F} \text{ n sdff } \mathcal{F}$
 $\text{sd } A \text{ sdf } F \text{ dm } (A \# F) \text{ n}$
shows $\neg \text{expressible-l } A \text{ } F \wedge \neg \text{FCs-covered-l } \mathcal{F} (A \# F) \text{ perms} \longleftrightarrow \neg \text{expressible}$
 $(\text{set } A) (f\text{-to-set-l } F) \wedge \neg \text{FCs-covered } (fs\text{-to-set-l } \mathcal{F}) (f\text{-to-set-l } F \cup \{\text{set } A\})$
 $\langle \text{proof} \rangle$

definition *mult-irreducible-notFCs-covered-l* **where**

$\text{mult-irreducible-notFCs-covered-l } \mathcal{F} \text{ } F \text{ } A \text{ perms} =$
 $(\text{let } F' = \text{filter } (\lambda f. A \notin \text{set } f \wedge \neg \text{expressible-l } A \text{ } f) \text{ } F;$
 $F'' = \text{map } (\lambda x. A \# x) \text{ } F'$
 $\text{in filter } (\lambda X. \neg \text{FCs-covered-l } \mathcal{F} \text{ } X \text{ perms}) \text{ } F'')$

definition *mult-all-irreducible-notFCs-covered-l* **where**

$\text{mult-all-irreducible-notFCs-covered-l } \mathcal{F} \text{ } F \text{ } n \text{ } m \text{ perms} =$
 $\text{concat } (\text{map } (\lambda A. \text{mult-irreducible-notFCs-covered-l } \mathcal{F} \text{ } F \text{ } A \text{ perms}) (\text{all-mn-subsets}$
 $n \text{ } m))$

definition *mult-all-base-irreducible-notFCs-covered-l* **where**

$\text{mult-all-base-irreducible-notFCs-covered-l } \mathcal{F} \text{ } F \text{ } n \text{ } m \text{ perms} =$
 $\text{non-isomorphic-families-l perms } (\text{mult-all-irreducible-notFCs-covered-l } \mathcal{F} \text{ } F \text{ } n$
 $m \text{ perms})$

lemma *mult-all-base-irreducible-notFCs-covered*:

$\text{mult-all-base-irreducible-notFCs-covered-l } \mathcal{F} \text{ } F \text{ } n \text{ } m \text{ perms} =$
 $\text{mult-all-base-P-l } (\lambda A \text{ } F. \neg \text{expressible-l } A \text{ } F \wedge \neg \text{FCs-covered-l } \mathcal{F} (A \# F)$
 $\text{perms}) \text{ } F \text{ } n \text{ } m \text{ perms}$
 $\langle \text{proof} \rangle$

definition *mult-irreducible-notFCs-covered-l-opt* **where**

$\text{mult-irreducible-notFCs-covered-l-opt } \mathcal{F}\text{-perms } F \text{ } A =$
 $(\text{let } F' = \text{filter } (\lambda f. A \notin \text{set } f \wedge \neg \text{expressible-l } A \text{ } f) \text{ } F;$
 $F'' = \text{map } (\lambda x. A \# x) \text{ } F'$
 $\text{in filter } (\lambda X. \neg \text{FCs-covered-l-opt'} \mathcal{F}\text{-perms } X) \text{ } F'')$

definition *mult-all-irreducible-notFCs-covered-l-opt* **where**

$\text{mult-all-irreducible-notFCs-covered-l-opt } \mathcal{F}\text{-perms } F \text{ } n \text{ } m =$
 $\text{concat } (\text{map } (\lambda A. \text{mult-irreducible-notFCs-covered-l-opt } \mathcal{F}\text{-perms } F \text{ } A) (\text{all-mn-subsets}$
 $n \text{ } m))$

definition *mult-all-base-irreducible-notFCs-covered-l-opt* **where**

$\text{mult-all-base-irreducible-notFCs-covered-l-opt } \mathcal{F}\text{-perms } F \text{ } n \text{ } m \text{ perms} =$
 $\text{non-isomorphic-families-l perms } (\text{mult-all-irreducible-notFCs-covered-l-opt } \mathcal{F}\text{-perms}$
 $F \text{ } n \text{ } m)$

lemma *mult-all-base-irreducible-notFCs-covered-opt:*
assumes $\mathcal{F}\text{-perms} = \text{map } (\lambda F. \text{map } (\lambda p. \text{permute-family-l } p \ F) \ \text{perms}) \ \mathcal{F}$
shows $\text{mult-all-base-irreducible-notFCs-covered-l-opt } \mathcal{F}\text{-perms } F \ n \ m \ \text{perms} =$
 $\text{mult-all-base-irreducible-notFCs-covered-l } \mathcal{F} \ F \ n \ m \ \text{perms}$
 $\langle \text{proof} \rangle$

abbreviation *enum-rec-irreducible-notFCs-covered-l* **where**
 $\text{enum-rec-irreducible-notFCs-covered-l } \mathcal{F}\text{-perms } L \ n \ \text{perms} \equiv$
 $\text{enum-rec } L \ [\Box] \ (\lambda F \ L. \text{mult-all-base-irreducible-notFCs-covered-l-opt } \mathcal{F}\text{-perms } F$
 $n \ (\text{length } L - 1) \ \text{perms})$

abbreviation *enum-dp-irreducible-notFCs-covered-l* **where**
 $\text{enum-dp-irreducible-notFCs-covered-l } \mathcal{F}\text{-perms } n \ \text{perms} \ \text{stops } \text{maxL} \equiv$
 $\text{enum-dp } [\Box] \ [\Box] \ 0 \ (\text{replicate } (n+1) \ (0::\text{nat}))$
 $(\lambda \text{FFb } m. \text{mult-all-base-irreducible-notFCs-covered-l-opt } \mathcal{F}\text{-perms } \text{FFb } n \ m$
 $\text{perms})$
 $(\lambda L. \text{list-ex } (\lambda L'. \text{pwge-impl } L \ L') \ \text{stops})$
 maxL

lemma *enum-dp-irreducible-notFCs-covered-l:*
assumes $\mathcal{F}\text{-perms} = \text{map } (\lambda F. \text{map } (\lambda p. \text{permute-family-l } p \ F) \ \text{perms}) \ \mathcal{F}$
shows
 $\text{enum-dp-irreducible-notFCs-covered-l } \mathcal{F}\text{-perms } n \ \text{perms} \ \text{stops } \text{maxL} =$
 $\text{enum-dp-mult-P } (\lambda A \ F. \neg \text{expressible-l } A \ F \wedge \neg \text{FCs-covered-l } \mathcal{F} \ (A \ \# \ F)$
 $\text{perms}) \ n \ \text{perms} \ \text{stops } \text{maxL}$
 $\langle \text{proof} \rangle$

lemma *enum-dp-irreducible-notFCs-covered-l-iso-representing-subsets:*
assumes
 $\mathcal{F}\text{-perms} = \text{map } (\lambda F. \text{map } (\lambda p. \text{permute-family-l } p \ F) \ \text{perms}) \ \mathcal{F}$
 $X = \{L'. \text{maxL} \succeq L' \wedge \neg (\exists S \in \text{set stops. } L' \succeq S)\} \ (\text{is -} = \text{?lhs})$
 $\text{length } \text{maxL} = n + 1 \ \forall S \in \text{set stops. } \text{length } S = n + 1$
 $\Box \notin \text{set } \mathcal{F} \ \text{perms} = \text{permute } [0..<n] \ \text{dmf } \mathcal{F} \ n \ \text{sdff } \mathcal{F}$
shows $\forall L' \in X. \exists B \in \text{set } (\text{enum-dp-irreducible-notFCs-covered-l } \mathcal{F}\text{-perms } n$
 $\text{perms stops maxL}).$
 $\text{iso-representing-subset } (\text{fs-to-set-l } B) \ (L\text{-part-irreducible-notFCs-covered}$
 $(\text{fs-to-set-l } \mathcal{F}) \ n \ L')$
 $\langle \text{proof} \rangle$

end

17 Minimal FC(6) families and maximal nonFC(6) families

theory *FC6-Data*
imports *Main*
begin

definition *FC6* :: *nat list list list* **where**

FC6 = [
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,2,3,4],[0,1,2,3,5],[1,2,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,1,2,5],[0,1,3,4,5],[0,2,3,4,5],[1,2,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,2,3,4],[1,2,3,4]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,2,3,4],[1,2,3,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,1,2,5],[0,2,3,5],[1,2,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,1,2,5],[0,3,4,5],[1,2,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,2,3,5],[1,2,3,5],[0,1,2,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,2,3,5],[0,2,4,5],[1,2,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,2,3,5],[1,2,4,5],[0,2,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,2,3,5],[2,3,4,5],[0,1,2,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,2,3,5],[2,3,4,5],[1,2,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,2,5],[0,3,4,5],[1,3,4,5],[0,2,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,5],[0,1,4,5],[2,3,4,5],[0,2,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,5],[0,2,4,5],[0,3,4,5],[1,2,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,5],[0,2,4,5],[1,3,4,5],[0,2,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,1,2,5],[2,3,4,5],[0,1,3,4,5],[0,2,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,1,2,5],[2,3,4,5],[0,2,3,4,5],[1,2,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,2,3,4],[0,1,2,5],[0,1,3,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,2,3,4],[0,1,2,5],[0,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,1,2,5],[0,1,3,5],[2,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,1,2,5],[0,2,3,5],[1,2,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,1,2,5],[0,1,3,5],[0,1,4,5],[0,2,3,4,5]],
 [[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,1,2,5],[0,1,3,5],[0,2,4,5],[0,2,3,4,5]],
 [[0,1,2],[0,1,3,4],[0,2,3,4],[0,1,2,3,5],[1,2,3,4,5]],
 [[0,1,2],[0,1,3,4],[0,1,3,5],[0,2,3,4,5],[1,2,3,4,5]],
 [[0,1,2],[0,1,3,4],[0,2,3,5],[0,1,2,4,5],[1,2,3,4,5]],
 [[0,1,2],[0,1,3,4],[0,2,3,5],[0,1,3,4,5],[1,2,3,4,5]],
 [[0,1,2],[0,1,3,4],[0,3,4,5],[0,1,2,3,5],[0,1,2,4,5],[1,2,3,4,5]],
 [[0,1,2],[0,1,3,4],[2,3,4,5],[0,1,2,3,5],[0,1,2,4,5],[0,2,3,4,5],[1,2,3,4,5]],
 [[0,1,2],[0,1,2,3],[0,1,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,1,3,4,5],[0,2,3,4,5],[1,2,3,4,5]],
 [[0,1,2],[0,3,4,5],[1,3,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,2,3,4,5],[1,2,3,4,5]],
 [[0,1,2],[0,1,3,4],[0,2,3,4],[1,2,3,4]],
 [[0,1,2],[0,1,3,4],[0,2,3,4],[1,2,3,5]],
 [[0,1,2],[0,1,3,4],[0,2,3,5],[1,2,4,5]],
 [[0,1,2],[0,1,3,4],[0,2,3,5],[1,3,4,5]],
 [[0,1,2],[0,1,2,3],[0,1,3,4],[0,2,3,4],[1,2,3,4,5]],
 [[0,1,2],[0,1,2,3],[0,1,3,4],[0,2,3,5],[1,2,3,4,5]],
 [[0,1,2],[0,1,2,3],[0,1,3,4],[0,2,4,5],[1,2,3,4,5]],
 [[0,1,2],[0,1,2,3],[0,1,4,5],[0,2,4,5],[1,2,3,4,5]],
 [[0,1,2],[0,1,3,4],[0,2,3,4],[0,1,3,5],[0,1,2,4,5]],
 [[0,1,2],[0,1,3,4],[0,2,3,4],[0,3,4,5],[1,2,3,4,5]],
 [[0,1,2],[0,1,3,4],[0,2,3,4],[1,3,4,5],[0,1,2,3,5]],
 [[0,1,2],[0,1,3,4],[0,2,3,4],[1,3,4,5],[1,2,3,4,5]],
 [[0,1,2],[0,1,3,4],[0,1,3,5],[0,1,4,5],[0,2,3,4,5]],
 [[0,1,2],[0,1,3,4],[0,1,3,5],[0,2,4,5],[0,2,3,4,5]],
 [[0,1,2],[0,1,3,4],[0,1,3,5],[0,3,4,5],[1,2,3,4,5]],
 [[0,1,2],[0,1,3,4],[0,1,3,5],[0,3,4,5],[1,2,3,4,5]]]

[illegible]

$[[0,1,2],[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,3,4,5],[2,3,4,5]],$
 $[[0,1,2],[0,1,2,3],[0,1,2,4],[0,1,3,5],[0,2,4,5],[0,3,4,5]],$
 $[[0,1,2],[0,1,2,3],[0,1,2,4],[0,3,4,5],[1,3,4,5],[2,3,4,5]],$
 $[[0,1,2],[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,1,2,5],[2,3,4,5],[0,2,3,4,5]],$
 $[[0,1,2],[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,1,3,5],[0,3,4,5],[0,2,3,4,5]],$
 $[[0,1,2],[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,2,3,5],[0,3,4,5],[0,1,2,4,5]],$
 $[[0,1,2],[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,1,2,5],[0,1,3,5],[0,3,4,5]],$
 $[[0,1,2],[0,1,3],[0,2,3,4,5],[1,2,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,1,3,4,5],[1,2,3,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,1,3,4,5],[0,2,3,4,5],[1,2,3,4,5]],$
 $[[0,1,2],[0,1,3],[0,2,3,4]],$
 $[[0,1,2],[0,3,4],[1,2,3,4]],$
 $[[0,1,2],[0,1,3],[0,1,4,5],[0,2,3,4,5]],$
 $[[0,1,2],[0,1,3],[0,2,4,5],[1,2,3,4,5]],$
 $[[0,1,2],[0,1,3],[2,3,4,5],[0,2,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,3,5],[0,1,2,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,3,5],[1,2,3,4,5]],$
 $[[0,1,2],[0,3,4],[1,2,3,5],[0,1,3,4,5]],$
 $[[0,1,2],[0,1,3],[0,2,4,5],[0,1,3,4,5],[0,2,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,2,4,5],[1,2,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,3,4,5],[1,2,3,4,5]],$
 $[[0,1,2],[0,3,4],[1,2,3,5],[0,1,2,4,5],[1,2,3,4,5]],$
 $[[0,1,2],[0,1,3],[0,2,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,3,4,5]],$
 $[[0,1,2],[0,1,3],[0,2,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,2,3,4,5]],$
 $[[0,1,2],[0,1,3],[2,3,4,5],[0,1,2,3,4],[0,1,2,4,5],[0,1,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,5],[0,1,2,3,5],[0,1,3,4,5],[1,2,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,5],[0,1,3,4,5],[0,2,3,4,5],[1,2,3,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,2,3,4,5]],$
 $[[0,1,2],[0,1,3],[0,1,4,5],[0,2,4,5]],$
 $[[0,1,2],[0,1,3],[0,1,4,5],[2,3,4,5]],$
 $[[0,1,2],[0,1,3],[0,2,4,5],[1,2,4,5]],$
 $[[0,1,2],[0,1,3],[0,2,4,5],[1,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[1,3,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,3,4],[0,2,3,5]],$
 $[[0,1,2],[0,1,3],[0,1,2,4],[0,2,4,5],[0,1,3,4,5]],$
 $[[0,1,2],[0,1,3],[0,1,2,4],[0,2,4,5],[0,2,3,4,5]],$
 $[[0,1,2],[0,1,3],[0,1,2,4],[0,3,4,5],[0,1,2,3,5]],$
 $[[0,1,2],[0,1,3],[0,1,2,4],[0,3,4,5],[0,1,2,4,5]],$
 $[[0,1,2],[0,1,3],[0,1,2,4],[0,3,4,5],[0,2,3,4,5]],$
 $[[0,1,2],[0,1,3],[0,1,2,4],[2,3,4,5],[0,1,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,2,4],[1,2,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,3,4],[1,2,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[1,2,3,5],[0,1,2,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[1,2,4,5],[0,1,2,3,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,5],[1,2,3,5],[0,1,2,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,5],[1,2,3,5],[1,2,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,5],[1,3,4,5],[0,2,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,5],[1,3,4,5],[1,2,3,4,5]],$

$[[0,1,2],[3,4,5],[0,1,3,4],[0,2,3,4],[0,1,2,3,5]],$
 $[[0,1,2],[3,4,5],[0,1,3,4],[0,2,3,4],[1,2,3,4,5]],$
 $[[0,1,2],[0,1,3],[0,1,2,4],[2,3,4,5],[0,1,2,3,5],[0,1,2,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,3,4],[0,1,2,4,5],[0,2,3,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,3,4],[0,2,3,4,5],[1,2,3,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,4,5],[0,2,3,4,5],[1,2,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,1,3,4,5],[0,2,3,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,2,4],[0,1,2,3,5],[0,1,3,4,5],[0,2,3,4,5],[1,2,3,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,3,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,1,3,4,5],[1,2,3,4,5]],$
 $[[0,1,2],[0,1,3],[0,1,2,4],[0,1,3,4],[0,2,4,5]],$
 $[[0,1,2],[0,1,3],[0,1,2,4],[0,1,3,4],[2,3,4,5]],$
 $[[0,1,2],[0,1,3],[0,1,2,4],[0,1,2,5],[0,2,4,5]],$
 $[[0,1,2],[0,1,3],[0,1,2,4],[0,1,3,5],[2,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,2,4],[1,2,3,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,3,4],[0,1,3,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,2,5],[0,1,3,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,2,5],[1,2,3,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,3,5],[0,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[1,2,3,5],[0,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[1,2,4,5],[0,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,5],[1,2,3,5],[0,3,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,3,4],[0,2,3,4]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,3,4],[0,1,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,4,5],[0,2,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,2,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,2,5],[0,3,4,5],[1,2,3,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,2,4],[0,1,3,5],[0,2,3,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,3,4],[0,3,4,5],[0,1,2,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,3,4],[2,3,4,5],[0,1,2,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,3,4],[0,1,2,5],[0,1,2,4,5],[0,2,3,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,3,4],[2,3,4,5],[0,1,2,3,5],[0,2,3,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,4,5],[2,3,4,5],[0,1,2,3,4],[0,2,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,2,3,4]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,2,4],[0,1,3,4],[2,3,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,2,4],[0,1,3,5],[0,3,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,2,4],[0,1,3,5],[2,3,4,5]],$
 $[[0,1,2],[0,3,4],[0,1,2,3],[0,1,2,4],[0,1,2,5],[0,3,4,5],[0,1,3,4,5],[0,2,3,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,2,4],[0,3,4,5],[1,3,4,5],[0,1,2,3,5],[0,1,2,4,5]],$
 $[[0,1,2],[3,4,5],[0,1,2,3],[0,1,2,4],[0,3,4,5],[1,3,4,5],[0,1,2,3,5],[0,2,3,4,5]],$
 $[[0,1,2],[0,1,3],[0,2,3]],$
 $[[0,1,2],[0,1,3],[0,1,4]],$
 $[[0,1,2],[0,1,3],[0,2,4]],$
 $[[0,1,2],[0,1,3],[2,3,4]],$
 $[[0,1,2],[0,1,3],[0,4,5]],$
 $[[0,1,2],[0,3,4],[1,3,5]],$
 $[[0,1,2],[0,1,3],[2,4,5],[0,2,3,4,5]],$
 $[[0,1,2],[0,1,3],[2,4,5],[0,1,2,3,4],[0,1,3,4,5]],$
 $[[0,1,2],[0,1,3],[2,4,5],[0,2,4,5]],$
 $[[0,1,2],[0,1,3],[2,4,5],[0,1,2,4],[0,1,2,3,5]],$

]

 $nonFC6 = [$
$$\begin{array}{l} \{\emptyset, [0,1,2], [0,1,2,3], [0,1,3,4], [2,3,4,5], [0,1,2,3,4], [0,1,2,3,5], [0,1,2,4,5], [0,1,3,4,5], [0,2,3,4,5], [0,1,2,3,4, \\ \emptyset, [0,1,2], [0,1,2,3], [0,1,4,5], [2,3,4,5], [0,1,2,3,4], [0,1,2,3,5], [0,1,2,4,5], [0,1,3,4,5], [0,2,3,4,5], [0,1,2,3,4, \end{array}$$

[illegible]

```

[[[0,1,2],[3,4,5],[0,1,2,3],[0,1,4,5],[0,1,2,3,4],[0,1,2,4,5],[0,1,3,4,5],[0,2,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[0,3,4],[0,1,2,3],[0,1,3,4],[0,1,2,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,2,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[0,3,4],[0,1,2,5],[0,1,3,5],[0,3,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[3,4,5],[0,1,2,3],[0,1,3,4],[0,1,3,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[3,4,5],[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,1,2,3,4],[0,1,3,4,5],[0,2,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[3,4,5],[0,1,2,3],[0,1,3,4],[2,3,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[3,4,5],[0,1,2,3],[0,1,3,4],[2,3,4,5],[0,1,2,3,4],[0,1,3,4,5],[0,2,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[3,4,5],[0,1,2,3],[0,1,4,5],[2,3,4,5],[0,1,2,3,4],[0,1,2,4,5],[0,1,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[0,3,4],[0,1,2,3],[0,1,2,4],[0,1,2,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,1,3,4,5],[0,2,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[3,4,5],[0,1,2,3],[0,1,3,4],[0,3,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,3,4,5],[0,2,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[0,3,4],[0,1,2,3],[0,1,2,5],[0,3,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,1,3,4,5],[0,2,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[3,4,5],[0,1,2,3],[0,1,2,4],[0,3,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,3,4,5],[1,2,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[3,4,5],[0,1,2,3],[0,1,2,4],[0,3,4,5],[0,1,2,3,4],[0,1,3,4,5],[0,2,3,4,5],[1,2,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[0,3,4],[0,1,2,3],[0,1,2,4],[0,3,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,1,3,4,5],[0,2,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[3,4,5],[0,1,2,3],[0,1,2,4],[0,1,2,5],[0,1,3,4],[0,1,2,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,1,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[3,4,5],[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,3,4,5],[0,1,2,3,4],[0,1,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[0,1,3],[0,1,2,3],[0,1,2,4],[0,1,2,5],[0,1,3,4],[0,1,3,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,1,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[3,4,5],[0,1,2,3],[0,1,2,4],[0,1,2,5],[0,3,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,2,4,5],[1,2,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[3,4,5],[0,1,2,3],[0,1,2,4],[0,1,2,5],[0,3,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[3,4,5],[0,1,2,3],[0,1,2,4],[0,1,2,5],[0,3,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,1,3,4,5],[0,2,3,4,5]],
[[0,1,2],[0,1,3],[0,1,2,3],[0,1,2,4],[0,1,2,5],[0,1,3,4],[0,1,3,5],[0,1,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,1,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[0,3,4],[0,1,2,3],[0,1,2,4],[0,1,2,5],[0,1,3,4],[0,3,4,5],[0,1,2,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,1,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[0,1,3],[2,4,5],[0,1,2,3],[0,1,2,4,5],[0,1,3,4,5],[0,1,2,3,4,5]],
[[0,1,2],[0,1,3],[2,4,5],[0,1,2,3],[0,1,2,3,4],[0,1,2,3,5],[0,1,2,4,5],[0,1,2,3,4,5]],
[[0,1,2],[0,1,3],[2,4,5],[0,1,2,3],[0,1,2,4],[0,1,2,3,4],[0,1,2,4,5],[0,1,2,3,4,5]],
[[0,1,2],[0,1,3],[2,4,5],[0,1,2,3],[0,1,3,4],[0,1,2,3,4],[0,1,2,4,5],[0,1,2,3,4,5]],
[[0,1,2],[0,1,3],[2,4,5],[0,1,2,3],[2,3,4,5],[0,1,2,4,5],[0,1,2,3,4,5]]
]

```

```

end
theory FranklImpl
imports Frankl FamilyImpl
begin

```

```

context SetImpl
begin

```

```

end

```

```

end

```

18 Tactics

```

theory FCTactics
imports FranklImpl SomeShareNegativeImpl
begin

```


$\langle ML \rangle$

18.1 Tactics for verifying FC families

Tactic relies on applying the *SomeShareNegativeImpl.some-share-negative* function.

$\langle ML \rangle$

18.2 Tactics for verifying nonFC families

Tactic for checking if a family belongs to the union-closed extension of a given family

thm *SetUnionImpl-nats.union-closed-additional-set*

$\langle ML \rangle$

Tactic for checking if the given coefficients *c* satisfy the given system of inequalities determined by the given families *Fs*

Lemma that allows to reformulate the statement as a problem over lists and gain faster executability.

lemma *nonFC-is-system-solution-lists*:

assumes $Fs = \text{map } f\text{-to-set-l } Fs\text{-l set } Fc\text{-l} = \bigcup Fc \quad \forall F \in \text{set } Fs\text{-l. distinct } F$
 $\forall F \in \text{set } Fs\text{-l. } \forall A \in \text{set } F. \text{sorted } A \wedge \text{distinct } A$

shows $(\text{let } Fs' = Fs \text{ in } \forall a \in \bigcup Fc. \text{sum-list } (\text{map } (\lambda (x, y). \text{int } x * y) (\text{zip } c$
 $(\text{map } (\text{Frankl.frankl-fun } a) Fs')))) < 0) \longleftrightarrow$

$(\text{list-all } (\lambda a. \text{sum-list } (\text{map } (\lambda (x, y). \text{int } x * y) (\text{zip } c (\text{map } (\text{frankl-fun-l}$
 $a) Fs\text{-l})))) < 0) Fc\text{-l})$

$\langle \text{proof} \rangle$

$\langle ML \rangle$

end

19 FC status proofs

theory *FC6-Status*

imports *FC6-Data SomeShareNegativeImpl WeightsShares-NotFCFamily FCTactics*

begin

lemma *[simp]*: $\neg \text{FC-family } (\{\{\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,4\}, \{0,2,3,4\}, \{0,1,2,3,4\}, \{1,2,3,4,5\}, \{0,1,2,$
 $\text{set set}\} (\text{is } \neg \text{FC-family } ?Fc)$

$\langle \text{proof} \rangle$

lemma *[simp]*: $\neg FC\text{-family} (\{\{\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,4\}, \{0,1,3,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,3,4,5\}\})$
set set) (**is** $\neg FC\text{-family} ?Fc$)
<proof>

lemma *[simp]*: $\neg FC\text{-family } (\{\{\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,5\}, \{0,1,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,2,$
set set) **(is** $\neg FC\text{-family } ?Fc$)
<proof>

lemma *[simp]*: $\neg FC\text{-family} (\{\{\}, \{0, 1, 2, 3\}, \{0, 1, 2, 4\}, \{0, 1, 3, 5\}, \{0, 2, 4, 5\}, \{0, 1, 2, 3, 4\}, \{0, 1, 2, 3, 5\}, \{0, 1, 2, 4, 5\}\})$ (**is** $\neg FC\text{-family} ?Fc$)
<proof>

lemma *[simp]*: $\neg FC\text{-family} (\{\{\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,4\}, \{0,2,3,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,2,$
set set) (**is** $\neg FC\text{-family ?Fc}$)
<proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,5\}, \{2,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,2,$
 $set\ set)\ (\text{is } \neg FC\text{-family } ?Fc)$
 $\langle proof \rangle$

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,3,4,5\}, \{1,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,2,$
 $set\ set\})$ (is $\neg FC\text{-family}\ ?Fc$)
 ⟨proof⟩

lemma *[simp]:* $\neg FC\text{-family} (\{\{\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,4\}, \{2,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,2,$
set set) **(is** $\neg FC\text{-family} ?Fc$)
<proof>

lemma *[simp]:* $\neg FC\text{-family} (\{\{\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,4\}, \{0,2,3,5\}, \{1,2,3,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}\})$
set set) (**is** $\neg FC\text{-family} ?Fc$)
<proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,4\}, \{0,2,3,5\}, \{2,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}\})$
set set) (**is** $\neg FC\text{-family} ?Fc$)
 ⟨proof⟩

lemma *[simp]:* $\neg FC\text{-family}(\{\{\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,5\}, \{0,1,4,5\}, \{2,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}\})$
set set) (**is** $\neg FC\text{-family} ?Fc$)
<proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,5\}, \{0,2,4,5\}, \{1,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}\})$ (is $\neg FC\text{-family} ?Fc$)
 ⟨proof⟩

lemma *[simp]*: $\neg FC\text{-family}(\{\{\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,4\}, \{0,1,3,5\}, \{2,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}\})$
set set) (**is** $\neg FC\text{-family} ?Fc$)
<proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0, 1, 2, 3\}, \{0, 1, 2, 4\}, \{0, 1, 3, 4\}, \{0, 2, 3, 5\}, \{1, 2, 4, 5\}, \{0, 1, 2, 3, 4\}, \{0, 1, 2, 3, 5\}\})$

set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

lemma [*simp*]: \neg *FC-family* ($\{\{\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,4\}, \{0,1,3,5\}, \{0,2,3,5\}, \{0,3,4,5\}, \{0,1,2,3,4\}\}$)
set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

lemma [*simp*]: \neg *FC-family* ($\{\{\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,4\}, \{0,1,3,5\}, \{0,2,3,5\}, \{0,2,4,5\}, \{0,1,2,3,4\}\}$)
set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

lemma [*simp*]: \neg *FC-family* ($\{\{\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,4\}, \{0,2,3,5\}, \{0,2,4,5\}, \{0,3,4,5\}, \{0,1,2,3,4\}\}$)
set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

lemma [*simp*]: \neg *FC-family* ($\{\{\}, \{0,1,2\}, \{0,1,3,4\}, \{0,2,3,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{1,2,3,4,5\}, \{0,1,2,3,4,5\}\}$)
set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

lemma [*simp*]: \neg *FC-family* ($\{\{\}, \{0,1,2\}, \{0,1,3,4\}, \{0,2,3,4\}, \{0,1,2,3,4\}, \{0,1,3,4,5\}, \{0,2,3,4,5\}, \{1,2,3,4,5\}\}$)
set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

lemma [*simp*]: \neg *FC-family* ($\{\{\}, \{0,1,2\}, \{0,1,3,4\}, \{0,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,3,4,5\}, \{0,2,3,4,5\}\}$)
set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

lemma [*simp*]: \neg *FC-family* ($\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,3,4\}, \{0,1,2,3,4\}, \{0,1,2,4,5\}, \{0,1,3,4,5\}, \{0,2,3,4,5\}\}$)
set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

lemma [*simp*]: \neg *FC-family* ($\{\{\}, \{0,1,2\}, \{0,1,3,4\}, \{2,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,3,4,5\}, \{0,2,3,4,5\}\}$)
set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

lemma [*simp*]: \neg *FC-family* ($\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,2,4,5\}, \{0,1,3,4,5\}\}$)
set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

lemma [*simp*]: \neg *FC-family* ($\{\{\}, \{0,1,2\}, \{0,1,3,4\}, \{0,2,3,4\}, \{0,2,3,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,3,4,5\}\}$)
set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

lemma [*simp*]: \neg *FC-family* ($\{\{\}, \{0,1,2\}, \{0,1,3,4\}, \{0,2,3,4\}, \{1,3,4,5\}, \{0,1,2,3,4\}, \{0,1,3,4,5\}, \{0,2,3,4,5\}\}$)
set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

lemma [*simp*]: \neg *FC-family* ($\{\{\}, \{0,1,2\}, \{0,1,3,4\}, \{0,3,4,5\}, \{2,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,3,4,5\}\}$)
set set) (**is** \neg *FC-family* ?*Fc*)

⟨proof⟩

lemma *[simp]*: $\neg FC\text{-family } (\{\{\}, \{0, 1, 2\}, \{0, 1, 2, 3\}, \{0, 1, 3, 4\}, \{0, 3, 4, 5\}, \{0, 1, 2, 3, 4\}, \{0, 1, 3, 4, 5\}, \{0, 2, 3, 4, 5\}\})$
set set) (**is** $\neg FC\text{-family } ?Fc$)
<proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,4,5\}, \{0,3,4,5\}, \{0,1,2,4,5\}, \{0,1,3,4,5\}, \{0,2,3,4,5\}\})$
 $\text{set set} \text{ (is } \neg FC\text{-family } ?Fc)$
 $\langle \text{proof} \rangle$

lemma *[simp]:* $\neg FC\text{-family}(\{\{\},\{0,1,2\},\{0,1,3,4\},\{0,1,3,5\},\{0,3,4,5\},\{0,1,2,3,4\},\{0,1,2,3,5\},\{0,1,2,4,5\}\})$
set set) (**is** $\neg FC\text{-family} ?Fc$)
<proof>

lemma *[simp]:* $\neg FC\text{-family}(\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,4\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,2,3,4\},$
set set) **(is** $\neg FC\text{-family ?Fc}$)
<proof>

lemma *[simp]*: $\neg FC\text{-family}(\{\{\}, \{0, 1, 2\}, \{0, 1, 2, 3\}, \{0, 1, 2, 4\}, \{0, 1, 3, 4\}, \{0, 1, 2, 3, 4\}, \{0, 1, 3, 4, 5\}, \{0, 2, 3, 4, 5\}\})$
(is $\neg FC\text{-family} ?Fc$)
<proof>

lemma *[simp]*: $\neg FC\text{-family}(\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,2,4,5\}\})$
(is $\neg FC\text{-family} ?Fc$)
<proof>

lemma [simp]: $\neg FC\text{-family}(\{\{\}, \{0, 1, 2\}, \{0, 1, 2, 3\}, \{0, 1, 2, 4\}, \{0, 1, 3, 5\}, \{0, 1, 2, 3, 4\}, \{0, 1, 2, 3, 5\}, \{0, 1, 3, 4, 5\}\})$ (is $\neg FC\text{-family} ?Fc$)
 $\langle proof \rangle$

lemma *[simp]*: $\neg FC\text{-family } (\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,3,4\}, \{2,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,2,3,4,5\}\})$
set set) (**is** $\neg FC\text{-family } ?Fc$)
<proof>

lemma *[simp]*: $\neg FC\text{-family}(\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,3,4\}, \{2,3,4,5\}, \{0,1,2,3,4\}, \{0,1,3,4,5\}, \{0,2,3,4,5\}\})$
(is $\neg FC\text{-family} ?Fc$)
<proof>

lemma [simp]: $\neg FC\text{-family}(\{\{f\}, \{0, 1, 2\}, \{0, 1, 2, 3\}, \{0, 1, 4, 5\}, \{2, 3, 4, 5\}, \{0, 1, 2, 3, 4\}, \{0, 1, 2, 4, 5\}, \{0, 2, 3, 4, 5\}\})$ (is $\neg FC\text{-family} ?Fc$)
 ⟨proof⟩

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0, 1, 2\}, \{0, 1, 2, 3\}, \{0, 1, 4, 5\}, \{2, 3, 4, 5\}, \{0, 1, 2, 4, 5\}, \{0, 1, 3, 4, 5\}, \{0, 2, 3, 4, 5\}\})$ (is $\neg FC\text{-family} ?Fc$)
 $\langle proof \rangle$

lemma [simp]: $\neg FC\text{-family}(\{\{\}, \{0, 1, 2\}, \{0, 1, 2, 3\}, \{0, 1, 3, 4\}, \{2, 3, 4, 5\}, \{0, 1, 2, 3, 4\}, \{0, 1, 2, 3, 5\}, \{0, 1, 2, 4, 5\}\})$ (is $\neg FC\text{-family} ?Fc$)
 $\langle proof \rangle$

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,4,5\}, \{2,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,2,4,5\}\})$
 set set) (is $\neg FC\text{-family} ?Fc$)
 <proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,3,4,5\}, \{1,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,4,5\}, \{0,1,3,4,5\}\})$
 set set) (is $\neg FC\text{-family} ?Fc$)
 <proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,3,4\}, \{0,1,3,5\}, \{0,2,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}\})$
 set set) (is $\neg FC\text{-family} ?Fc$)
 <proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,3,4\}, \{0,1,3,5\}, \{2,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}\})$
 set set) (is $\neg FC\text{-family} ?Fc$)
 <proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,3,4\}, \{0,2,3,5\}, \{0,2,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}\})$
 set set) (is $\neg FC\text{-family} ?Fc$)
 <proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,3,4\}, \{0,1,4,5\}, \{0,2,4,5\}, \{0,1,2,3,4\}, \{0,1,2,4,5\}\})$
 set set) (is $\neg FC\text{-family} ?Fc$)
 <proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,3,4\}, \{0,1,4,5\}, \{2,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,4,5\}\})$
 set set) (is $\neg FC\text{-family} ?Fc$)
 <proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,3,4\}, \{0,3,4,5\}, \{2,3,4,5\}, \{0,1,2,3,4\}, \{0,1,3,4,5\}\})$
 set set) (is $\neg FC\text{-family} ?Fc$)
 <proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,4,5\}, \{0,3,4,5\}, \{1,3,4,5\}, \{0,1,2,4,5\}, \{0,1,3,4,5\}\})$
 set set) (is $\neg FC\text{-family} ?Fc$)
 <proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,4,5\}, \{0,3,4,5\}, \{2,3,4,5\}, \{0,1,2,4,5\}, \{0,1,3,4,5\}\})$
 set set) (is $\neg FC\text{-family} ?Fc$)
 <proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,3,5\}, \{0,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}\})$
 set set) (is $\neg FC\text{-family} ?Fc$)
 <proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2\}, \{0,1,2,3\}, \{0,1,3,4\}, \{0,1,3,5\}, \{0,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}\})$
 set set) (is $\neg FC\text{-family} ?Fc$)
 <proof>

$\langle \text{proof} \rangle$

lemma $[simp]: \neg FC\text{-family } (\{\{\}, \{0,1,2\}, \{0,3,4\}, \{0,1,2,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,2,4,5\}, \{1,2,3,4\},$
 $set\ set) \ (\text{is} \neg FC\text{-family } ?Fc)$
 $\langle \text{proof} \rangle$

lemma $[simp]: \neg FC\text{-family } (\{\{\}, \{0,1,2\}, \{3,4,5\}, \{0,1,2,3\}, \{0,1,2,3,4\}, \{0,1,2,4,5\}, \{0,1,3,4,5\}, \{0,2,3,4\},$
 $set\ set) \ (\text{is} \neg FC\text{-family } ?Fc)$
 $\langle \text{proof} \rangle$

lemma $[simp]: \neg FC\text{-family } (\{\{\}, \{0,1,2\}, \{0,1,3\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,3,4,5\}, \{0,1,2,3,4\}, \{0,1,3,4,5\}, \{$
 $set\ set) \ (\text{is} \neg FC\text{-family } ?Fc)$
 $\langle \text{proof} \rangle$

lemma $[simp]: \neg FC\text{-family } (\{\{\}, \{0,1,2\}, \{0,3,4\}, \{0,1,2,3\}, \{0,1,3,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,3,4,5\},$
 $set\ set) \ (\text{is} \neg FC\text{-family } ?Fc)$
 $\langle \text{proof} \rangle$

lemma $[simp]: \neg FC\text{-family } (\{\{\}, \{0,1,2\}, \{0,3,4\}, \{0,1,2,3\}, \{1,2,4,5\}, \{0,1,2,3,4\}, \{0,1,2,4,5\}, \{0,1,2,3,4\},$
 $set\ set) \ (\text{is} \neg FC\text{-family } ?Fc)$
 $\langle \text{proof} \rangle$

lemma $[simp]: \neg FC\text{-family } (\{\{\}, \{0,1,2\}, \{0,3,4\}, \{0,1,2,5\}, \{1,3,4,5\}, \{0,1,2,3,4\}, \{0,1,3,4,5\}, \{0,1,2,3,4\},$
 $set\ set) \ (\text{is} \neg FC\text{-family } ?Fc)$
 $\langle \text{proof} \rangle$

lemma $[simp]: \neg FC\text{-family } (\{\{\}, \{0,1,2\}, \{0,1,3\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,2,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{$
 $set\ set) \ (\text{is} \neg FC\text{-family } ?Fc)$
 $\langle \text{proof} \rangle$

lemma $[simp]: \neg FC\text{-family } (\{\{\}, \{0,1,2\}, \{0,1,3\}, \{0,1,2,3\}, \{0,1,2,4\}, \{2,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{$
 $set\ set) \ (\text{is} \neg FC\text{-family } ?Fc)$
 $\langle \text{proof} \rangle$

lemma $[simp]: \neg FC\text{-family } (\{\{\}, \{0,1,2\}, \{0,1,3\}, \{0,1,2,3\}, \{0,1,2,4\}, \{2,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,4,5\}, \{$
 $set\ set) \ (\text{is} \neg FC\text{-family } ?Fc)$
 $\langle \text{proof} \rangle$

lemma $[simp]: \neg FC\text{-family } (\{\{\}, \{0,1,2\}, \{0,3,4\}, \{0,1,2,3\}, \{0,1,2,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{1,2,3,4,5\},$
 $set\ set) \ (\text{is} \neg FC\text{-family } ?Fc)$
 $\langle \text{proof} \rangle$

lemma $[simp]: \neg FC\text{-family } (\{\{\}, \{0,1,2\}, \{0,3,4\}, \{0,1,2,3\}, \{1,2,3,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{1,2,3,4,5\},$
 $set\ set) \ (\text{is} \neg FC\text{-family } ?Fc)$
 $\langle \text{proof} \rangle$

lemma $[simp]: \neg FC\text{-family } (\{\{\}, \{0,1,2\}, \{0,3,4\}, \{0,1,2,3\}, \{0,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{1,2,3,4,5\},$
 $set\ set) \ (\text{is} \neg FC\text{-family } ?Fc)$
 $\langle \text{proof} \rangle$

lemma *[simp]*: $\neg FC\text{-family } (\{\{\}, \{0,1,2\}, \{3,4,5\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,2,3,4,5\}\})$
set set) **(is** $\neg FC\text{-family } ?Fc$)
<proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2\}, \{3,4,5\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,3,4,5\}, \{0,1,2,3,4\}, \{0,1,3,4,5\}, \{0,2,3,4,5\}\})$
 $(\text{is } \neg FC\text{-family } ?Fc)$
 $\langle proof \rangle$

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0, 1, 2\}, \{0, 3, 4\}, \{0, 1, 2, 3\}, \{0, 1, 2, 4\}, \{0, 3, 4, 5\}, \{0, 1, 2, 3, 4\}, \{0, 1, 2, 3, 5\}, \{0, 1, 2, 3, 4, 5\}\})$
set set) (is $\neg FC\text{-family} ?Fc$)
 ⟨proof⟩

lemma *[simp]*: $\neg FC\text{-family}(\{\{\}, \{0,1,2\}, \{3,4,5\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,2,5\}, \{0,1,3,4\}, \{0,1,2,3,4\}, \{0,$
set set) (**is** $\neg FC\text{-family} \text{ ?Fc}$)
<proof>

lemma [simp]: $\neg FC\text{-family}(\{\{\}, \{0, 1, 2\}, \{3, 4, 5\}, \{0, 1, 2, 3\}, \{0, 1, 2, 4\}, \{0, 1, 3, 4\}, \{0, 3, 4, 5\}, \{0, 1, 2, 3, 4\}, \{0, 1, 2, 3, 5\}, \{0, 1, 2, 4, 5\}, \{0, 1, 3, 4, 5\}\})$
 (is $\neg FC\text{-family } ?Fc$)
 $\langle proof \rangle$

lemma *[simp]*: $\neg FC\text{-family}(\{\{\}, \{0,1,2\}, \{0,1,3\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,2,5\}, \{0,1,3,4\}, \{0,1,3,5\}, \{0,1,4,5\}, \{0,2,3,4\}, \{0,2,3,5\}, \{0,2,4,5\}, \{0,3,4,5\}\})$
set set) (**is** $\neg FC\text{-family} ?Fc$)
<proof>

lemma *[simp]*: $\neg FC\text{-family}(\{\{\}, \{0,1,2\}, \{3,4,5\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,2,5\}, \{0,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,2,4,5\}, \{0,3,4,5\}\})$
(is $\neg FC\text{-family } ?Fc$)
<proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0,1,2\}, \{3,4,5\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,3,4,5\}, \{1,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,2,4,5\}, \{0,3,4,5\}\})$
set set) (is $\neg FC\text{-family } ?Fc$)
 ⟨proof⟩

lemma [simp]: $\neg FC\text{-family}(\{\{\}, \{0,1,2\}, \{3,4,5\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,2,5\}, \{0,3,4,5\}, \{0,1,2,3,4\}, \{0,1,2,3,5\}, \{0,1,2,4,5\}, \{0,3,4,5\}\})$
set set) (is $\neg FC\text{-family } ?Fc$)
 ⟨proof⟩

lemma *[simp]*: $\neg FC\text{-family}(\{\{\}, \{0,1,2\}, \{0,1,3\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,2,5\}, \{0,1,3,4\}, \{0,1,3,5\}, \{0,1,$
set set) (**is** $\neg FC\text{-family } ?Fc$)
<proof>

lemma [simp]: $\neg FC\text{-family} (\{\{\}, \{0, 1, 2\}, \{0, 3, 4\}, \{0, 1, 2, 3\}, \{0, 1, 2, 4\}, \{0, 1, 2, 5\}, \{0, 1, 3, 4\}, \{0, 3, 4, 5\}, \{0, 1, 3, 4, 5\}\})$ (is $\neg FC\text{-family} ?Fc$)
 <proof>

lemma [simp]: $\neg FC\text{-family}(\{\{\}, \{0,1,2\}, \{0,1,3\}, \{2,4,5\}, \{0,1,2,3\}, \{0,1,2,4,5\}, \{0,1,3,4,5\}, \{0,1,2,3,4,5\}\})$
set set) (is $\neg FC\text{-family} ?Fc$)
<proof>

lemma *[simp]*: $\neg FC\text{-family}(\{\{\}, \{0, 1, 2\}, \{0, 1, 3\}, \{2, 4, 5\}, \{0, 1, 2, 3\}, \{0, 1, 2, 3, 4\}, \{0, 1, 2, 3, 5\}, \{0, 1, 2, 4, 5\}, \{$

set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

lemma [*simp*]: \neg *FC-family* ($\{\{\}, \{0,1,2\}, \{0,1,3\}, \{2,4,5\}, \{0,1,2,3\}, \{0,1,2,4\}, \{0,1,2,3,4\}, \{0,1,2,4,5\}, \{0,1,2,3,4,5\}\}$)
set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

lemma [*simp*]: \neg *FC-family* ($\{\{\}, \{0,1,2\}, \{0,1,3\}, \{2,4,5\}, \{0,1,2,3\}, \{0,1,3,4\}, \{0,1,2,3,4\}, \{0,1,2,4,5\}, \{0,1,2,3,4,5\}\}$)
set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

lemma [*simp*]: \neg *FC-family* ($\{\{\}, \{0,1,2\}, \{0,1,3\}, \{2,4,5\}, \{0,1,2,3\}, \{2,3,4,5\}, \{0,1,2,4,5\}, \{0,1,2,3,4,5\}\}$)
set set) (**is** \neg *FC-family* ?*Fc*)
 \langle *proof* \rangle

theorem $\forall F \in \text{set nonFC6}. \neg$ *FC-family* (*f-to-set-l* *F*)
 \langle *proof* \rangle

end

20 Proof that all families are covered

theory *FC6*
imports *FC6-Data LPartitioningIrreducibleNonFCsCoveredImpl*
begin

definition *FC6-perms* **where**
 $\text{FC6-perms} = \text{map } (\lambda F. \text{map } (\lambda p. \text{permute-family-l } p F) (\text{permute } [0..<6]))$
FC6

definition *nonFC6-perms* **where**
 $\text{nonFC6-perms} = \text{map } (\lambda Nc. \text{map } (\lambda p. \text{close-insert-empty-l } (\text{permute-family-l } p Nc)) (\text{permute } [0..<6]))$ *nonFC6*

lemma *L-part-6*:
assumes $\bigcup F \subseteq \{0..<6::\text{nat}\}$
shows $\exists n0\ n1\ n2\ n3\ n4\ n5\ n6.$
 $\text{is-L-part } 6\ [n0, n1, n2, n3, n4, n5, n6]\ F \wedge$
 $n0 \leq 1 \wedge n1 \leq 6 \wedge n2 \leq 15 \wedge n3 \leq 20 \wedge n4 \leq 15 \wedge n5 \leq 6 \wedge n6 \leq 1$
 \langle *proof* \rangle

theorem *enum-rec-notFCs-covered-l*:

assumes *enum-rec-notFCs-covered-l FC6-perms* $[0, n1, n2, n3, n4, n5, n6]$ 6
(*permute* $[0..<6]$) = []
shows $\forall F \in L\text{-part } 6 [0, n1, n2, n3, n4, n5, n6]. FCs\text{-covered } (fs\text{-to-set-}l FC6) F$

<proof>

lemma *FC-0000560*:

shows $\forall F \in L\text{-part } 6 [0, 0, 0, 0, 5, 6, 0]. FCs\text{-covered } (fs\text{-to-set-}l FC6) F$
<proof>

lemma *FC-0000700*:

shows $\forall F \in L\text{-part } 6 [0, 0, 0, 0, 7, 0, 0]. FCs\text{-covered } (fs\text{-to-set-}l FC6) F$
<proof>

lemma *FC-0001650*:

shows $\forall F \in L\text{-part } 6 [0, 0, 0, 1, 6, 5, 0]. FCs\text{-covered } (fs\text{-to-set-}l FC6) F$
<proof>

lemma *FC-0002060*:

shows $\forall F \in L\text{-part } 6 [0, 0, 0, 2, 0, 6, 0]. FCs\text{-covered } (fs\text{-to-set-}l FC6) F$
<proof>

lemma *FC-0003040*:

shows $\forall F \in L\text{-part } 6 [0, 0, 0, 3, 0, 4, 0]. FCs\text{-covered } (fs\text{-to-set-}l FC6) F$
<proof>

lemma *FC-0003230*:

shows $\forall F \in L\text{-part } 6 [0, 0, 0, 3, 2, 3, 0]. FCs\text{-covered } (fs\text{-to-set-}l FC6) F$
<proof>

lemma *FC-0003300*:

shows $\forall F \in L\text{-part } 6 [0, 0, 0, 3, 3, 0, 0]. FCs\text{-covered } (fs\text{-to-set-}l FC6) F$
<proof>

lemma *FC-0004000*:

shows $\forall F \in L\text{-part } 6 [0, 0, 0, 4, 0, 0, 0]. FCs\text{-covered } (fs\text{-to-set-}l FC6) F$
<proof>

lemma *FC-0010000*:

shows $\forall F \in L\text{-part } 6 [0, 0, 1, 0, 0, 0, 0]. FCs\text{-covered } (fs\text{-to-set-}l FC6) F$
<proof>

lemma *FC-0100000*:

shows $\forall F \in L\text{-part } 6 [0, 1, 0, 0, 0, 0, 0]. FCs\text{-covered } (fs\text{-to-set-}l FC6) F$
<proof>

definition *all-FC-partitions* :: nat list list **where**

all-FC-partitions =
 [
 [0, 0, 0, 0, 5, 6, 0],
 [0, 0, 0, 0, 7, 0, 0],
 [0, 0, 0, 1, 6, 5, 0],
 [0, 0, 0, 2, 0, 6, 0],
 [0, 0, 0, 3, 0, 4, 0],
 [0, 0, 0, 3, 2, 3, 0],
 [0, 0, 0, 3, 3, 0, 0],
 [0, 0, 0, 4, 0, 0, 0],
 [0, 1, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0]
]

definition *notAllFC-partitions* **where**

notAllFC-partitions = { $L'. [1, 6, 15, 20, 15, 6, 1] \succeq L' \wedge \neg (\exists S \in \text{set } \text{all-FC-partitions}. L' \succeq S)$ }

lemma *notAllFC-partitions-remove-empty*:

assumes $[1, n1, n2, n3, n4, n5, n6] \in \text{notAllFC-partitions}$ (**is** ? $L \in -$)
shows $[0, n1, n2, n3, n4, n5, n6] \in \text{notAllFC-partitions}$ (**is** ? $L' \in -$)
 <proof>

theorem *notAllFC-partitions*:

assumes $\forall L F. L \in \text{notAllFC-partitions} \wedge \text{hd } L = 0 \wedge F \in L\text{-part-irreducible}$
 $6 \text{ } L \longrightarrow \text{covered } \mathcal{F} \mathcal{N} F$
 $\mathcal{F} = \text{fs-to-set-l FC6}$
shows $\forall F. \bigcup F \subseteq \{0..<6::\text{nat}\} \longrightarrow \text{covered } \mathcal{F} \mathcal{N} F$

<proof>

theorem

shows $\forall F. \bigcup F \subseteq \{0..<6::\text{nat}\} \longrightarrow \text{covered } (\text{fs-to-set-l FC6}) (\text{fs-to-set-l nonFC6}) F$

<proof>

definition *all-nonFC-partitions* **where**

all-nonFC-partitions =
 [
 [0, 0, 0, 0, 3, 6, 1],
]

```
[0, 0, 0, 0, 4, 1, 1],  
[0, 0, 0, 1, 1, 6, 1],  
[0, 0, 0, 1, 2, 1, 1],  
[0, 0, 0, 2, 0, 1, 1]  
]  
  
end
```