The Sunflower Lemma of Erdős and Rado

René Thiemann

March 17, 2025

Abstract

We formally define sunflowers and provide a formalization of the sunflower lemma of Erdős and Rado: whenever a set of size-k-sets has a larger cardinality than $(r-1)^k \cdot k!$, then it contains a sunflower of cardinality r.

1 Sunflowers

Sunflowers are sets of sets, such that whenever an element is contained in at least two of the sets, then it is contained in all of the sets.

```
theory Sunflower
  imports Main
    HOL−Library.FuncSet
begin

definition sunflower :: 'a set set ⇒ bool where
  sunflower S = (∀ x. (∃ A B. A ∈ S ∧ B ∈ S ∧ A ≠ B ∧
     x ∈ A ∧ x ∈ B)
    ⟶ (∀ A. A ∈ S ⟶ x ∈ A))

lemma sunflower-subset: F ⊆ G ⟹ sunflower G ⟹ sunflower F
  unfolding sunflower-def by blast

lemma pairwise-disjnt-imp-sunflower:
  pairwise disjnt F ⟹ sunflower F
  unfolding sunflower-def
  by (metis disjnt-insert1 mk-disjoint-insert pairwiseD)

lemma card2-sunflower: assumes finite S and card S ≤ 2
  shows sunflower S
proof −
  from assms have card S = 0 ∨ card S = Suc 0 ∨ card S = 2 by linarith
  with ‹finite S› obtain A B where S = {} ∨ S = {A} ∨ S = {A,B}
    using card-2-iff[of S] card-1-singleton-iff[of S] by auto
  thus ?thesis unfolding sunflower-def by auto
qed
```

```
lemma empty-sunflower: sunflower {}
  by (rule card2-sunflower, auto)

lemma singleton-sunflower: sunflower {A}
  by (rule card2-sunflower, auto)

lemma doubleton-sunflower: sunflower {A,B}
  by (rule card2-sunflower, auto, cases A = B, auto)

lemma sunflower-imp-union-intersect-unique:
  assumes sunflower S
    and x ∈ (⋃ S) − (⋂ S)
  shows ∃! A. A ∈ S ∧ x ∈ A
proof −
  from assms obtain A where A: A ∈ S x ∈ A by auto
  show ?thesis
  proof
    show A ∈ S ∧ x ∈ A using A by auto
    fix B
    assume B: B ∈ S ∧ x ∈ B
    show B = A
    proof (rule ccontr)
      assume B ≠ A
      with A B have ∃ A B. A ∈ S ∧ B ∈ S ∧ A ≠ B ∧ x ∈ A ∧ x ∈ B by auto
      from ‹sunflower S›[unfolded sunflower-def, rule-format, OF this]
      have x ∈ ⋂ S by auto
      with assms show False by auto
    qed
  qed
qed

lemma union-intersect-unique-imp-sunflower:
  assumes ⋀ x. x ∈ (⋃ S) − (⋂ S) ⟹ ∃≤1 A. A ∈ S ∧ x ∈ A
  shows sunflower S
  unfolding sunflower-def
proof (intro allI impI, elim exE conjE, goal-cases)
  case (1 x C A B)
  hence x: x ∈ ⋃ S by auto
  show ?case
  proof (cases x ∈ ⋂ S)
    case False
    with assms[of x] x have ∃≤1 A. A ∈ S ∧ x ∈ A by blast
    with 1 have False unfolding Uniq-def by blast
    thus ?thesis ..
  next
    case True
    with 1 show ?thesis by blast
  qed
```

```
qed

lemma sunflower-iff-union-intersect-unique:
  sunflower S ⟷ (∀ x ∈ ⋃ S − ⋂ S. ∃! A. A ∈ S ∧ x ∈ A)
  (is ?l = ?r)
proof
  assume ?l
  from sunflower-imp-union-intersect-unique[OF this]
  show ?r by auto
next
  assume ?r
  hence *: ∀x∈⋃ S − ⋂ S. ∃≤1 A. A ∈ S ∧ x ∈ A
    unfolding ex1-iff-ex-Uniq by auto
  show ?l
    by (rule union-intersect-unique-imp-sunflower, insert *, auto)
qed

lemma sunflower-iff-intersect-Uniq:
  sunflower S ⟷ (∀ x. x ∈ ⋂ S ∨ (∃≤1 A. A ∈ S ∧ x ∈ A))
  (is ?l = ?r)
proof
  assume ?l
  from sunflower-imp-union-intersect-unique[OF this]
  show ?r unfolding ex1-iff-ex-Uniq
    by (metis (no-types, lifting) DiffI UnionI Uniq-I)
next
  assume ?r
  show ?l
    by (rule union-intersect-unique-imp-sunflower, insert ‹?r›, auto)
qed
```

If there exists sunflowers whenever all elements are sets of the same cardinality r, then there also exists sunflowers whenever all elements are sets with cardinality at most r.

```
lemma sunflower-card-subset-lift: fixes F :: 'a set set
  assumes sunflower: ⋀ G :: ('a + nat) set set.
      (∀ A ∈ G. finite A ∧ card A = k) ⟹ card G > c
        ⟹ ∃ S. S ⊆ G ∧ sunflower S ∧ card S = r
    and kF: ∀ A ∈ F. finite A ∧ card A ≤ k
    and cardF: card F > c
  shows ∃ S. S ⊆ F ∧ sunflower S ∧ card S = r
proof −
  let ?n = Suc c
  from cardF have card F ≥ ?n by auto
  then obtain FF where sub: FF ⊆ F and cardF: card FF = ?n
    by (rule obtain-subset-with-card-n)
  let ?N = {0 ..< ?n}
  from cardF have finite FF
    by (simp add: card-ge-0-finite)
```

```
  from ex-bij-betw-nat-finite[OF this, unfolded cardF]
  obtain f where f: bij-betw f ?N FF by auto
  hence injf: inj-on f ?N by (rule bij-betw-imp-inj-on)
  have Ff: FF = f ‘ ?N
    by (metis bij-betw-imp-surj-on f)
  define g where g = (λ i. (Inl ‘ f i) ∪ (Inr ‘ {0 ..< (k − card (f i))}))
  have injg: inj-on g ?N unfolding g-def using f
  proof (intro inj-onI, goal-cases)
    case (1 x y)
    hence f x = f y by auto
    with injf 1 show x = y
      by (meson inj-onD)
  qed
  hence cardgN: card (g ‘ ?N) > c
    by (simp add: card-image)
  {
    fix i
    assume i ∈ ?N
    hence f i ∈ FF unfolding Ff by auto
    with sub have f i ∈ F by auto
    hence card (f i) ≤ k finite (f i) using kF by auto
    hence card (g i) = k ∧ finite (g i) unfolding g-def
      by (subst card-Un-disjoint, auto, subst (1 2) card-image, auto intro: inj-onI)
  }
  hence ∀ A ∈ g ‘ ?N. finite A ∧ card A = k by auto
  from sunflower[OF this cardgN]
  obtain S where SgN: S ⊆ g ‘ ?N and sf: sunflower S and card: card S = r
by auto
  from SgN obtain N where NN: N ⊆ ?N and SgN: S = g ‘ N
    by (meson subset-image-iff)
  from injg NN have inj-g: inj-on g N
    by (rule inj-on-subset)
  from injf NN have inj-f: inj-on f N
    by (rule inj-on-subset)
  from card-image[OF inj-g] SgN card
  have cardN: card N = r by auto
  let ?S = f ‘ N
  show ?thesis
  proof (intro exI[of - ?S] conjI)
    from NN show ?S ⊆ F using Ff sub by auto
    from card-image[OF inj-f] cardN show card ?S = r by auto
    show sunflower ?S unfolding sunflower-def
    proof (intro allI impI, elim exE conjE, goal-cases)
      case (1 x C A B)
      from ‹A ∈ f ‘ N› obtain i where i: i ∈ N and A: A = f i by auto
      from ‹B ∈ f ‘ N› obtain j where j: j ∈ N and B: B = f j by auto
      from ‹C ∈ f ‘ N› obtain k where k: k ∈ N and C: C = f k by auto
      hence gk: g k ∈ g ‘ N by auto
      from ‹A ≠ B› A B have ij: i ≠ j by auto
```

```
      from inj-g ij i j have gij: g i ≠ g j by (metis inj-on-contraD)
      from ‹x ∈ A› have memi: Inl x ∈ g i unfolding A g-def by auto
      from ‹x ∈ B› have memj: Inl x ∈ g j unfolding B g-def by auto
      have ∃ A B. A ∈ g ‘ N ∧ B ∈ g ‘ N ∧ A ≠ B ∧ Inl x ∈ A ∧ Inl x ∈ B
        using memi memj gij i j by auto
      from sf[unfolded sunflower-def SgN, rule-format, OF this gk] have Inl x ∈ g
k .
      thus x ∈ C unfolding C g-def by auto
    qed
  qed
qed
```

We provide another sunflower lifting lemma that ensures non-empty cores. Here, all elements must be taken from a finite set, and the bound is multiplied the cardinality.

```
lemma sunflower-card-core-lift:
  assumes finE: finite (E :: 'a set)
    and sunflower: ⋀ G :: 'a set set.
     (∀ A ∈ G. finite A ∧ card A ≤ k) ⟹ card G > c
       ⟹ ∃ S. S ⊆ G ∧ sunflower S ∧ card S = r
    and F: ∀ A ∈ F. A ⊆ E ∧ s ≤ card A ∧ card A ≤ k
    and cardF: card F > (card E choose s) ∗ c
    and s: s ≠ 0
    and r: r ≠ 0
  shows ∃ S. S ⊆ F ∧ sunflower S ∧ card S = r ∧ card (⋂ S) ≥ s
proof −
  let ?g = λ (A :: 'a set) x. card x = s ∧ x ⊆ A
  let ?E = {X. X ⊆ E ∧ card X = s}
  from cardF have finF: finite F
    by (metis card.infinite le-0-eq less-le)
  from cardF have FnE: F ≠ {} by force
  {
    from FnE obtain B where B: B ∈ F by auto
    with F[rule-format, OF B] obtain A where A ⊆ E card A = s
      by (meson obtain-subset-with-card-n order-trans)
    hence ?E ≠ {} using B by auto
  } note EnE = this
  define f where f = (λ A. SOME x. ?g A x)
  from finE have finiteE: finite ?E by simp

  have f ∈ F → ?E
  proof
    fix B
    assume B: B ∈ F
   with F[rule-format, OF B] have ∃ x. ?g B x by (meson obtain-subset-with-card-n)
    from someI-ex[OF this] B F show f B ∈ ?E unfolding f-def by auto
  qed
  from pigeonhole-card[OF this finF finiteE EnE]
  obtain a where a: a ∈ ?E
```

```
    and le: card F ≤ card (f −‘ {a} ∩ F) ∗ card ?E by auto
  have precond: ∀ A∈f −‘ {a} ∩ F. finite A ∧ card A ≤ k
    using F finite-subset[OF - finE] by auto
  have c ∗ (card E choose s) = (card E choose s) ∗ c by simp
  also have . . . < card F by fact
  also have . . . ≤ (card (f −‘ {a} ∩ F)) ∗ card ?E by fact
  also have card ?E = card E choose s by (rule n-subsets[OF finE])
  finally have c < card (f −‘ {a} ∩ F) by auto
  from sunflower[OF precond this]
  obtain S where ∗: S ⊆ f −‘ {a} ∩ F sunflower S card S = r
    by auto
  from finite-subset[OF - finF, of S]
  have finS: finite S using ∗ by auto
  from ∗ r have SnE: S ≠ {} by auto
  have finIS: finite (⋂ S)
  proof (rule finite-Inter)
    from SnE obtain A where A: A ∈ S by auto
    with F s have finite A
      using ∗ precond by blast
    thus ∃ A∈S. finite A using A by auto
  qed
  show ?thesis
  proof (intro exI[of - S] conjI ∗)
    show S ⊆ F using ∗ by auto
    {
      fix A
      assume A ∈ S
      with ∗(1) have A ∈ f −‘ {a} and A: A ∈ F using ∗ by auto
      from this have ∗∗: f A = a A ∈ F by auto
      from F[rule-format, OF A] have ∃ x. card x = s ∧ x ⊆ A
        by (meson obtain-subset-with-card-n order-trans)
      from someI-ex[of ?g A, OF this] ∗∗
      have a ⊆ A unfolding f-def by auto
    }
    hence a ⊆ ⋂ S by auto
    from card-mono[OF finIS this]
    have card a ≤ card (⋂ S) .
    with a show s ≤ card (⋂ S) by auto
  qed
qed

lemma sunflower-nonempty-core-lift:
  assumes finE: finite (E :: 'a set)
    and sunflower: ⋀ G :: 'a set set.
     (∀ A ∈ G. finite A ∧ card A ≤ k) ⟹ card G > c
        ⟹ ∃ S. S ⊆ G ∧ sunflower S ∧ card S = r
    and F: ∀ A ∈ F. A ⊆ E ∧ card A ≤ k
    and empty: {} ∉ F
    and cardF: card F > card E ∗ c
```

```
  shows ∃ S. S ⊆ F ∧ sunflower S ∧ card S = r ∧ (⋂ S) ≠ {}
proof (cases r = 0)
  case False
  from F empty have F': ∀ A∈F. A ⊆ E ∧ 1 ≤ card A ∧ card A ≤ k  using finE
    by (metis One-nat-def Suc-leI card-gt-0-iff finite-subset)
  from cardF have cardF': (card E choose 1) * c < card F by auto
  from sunflower-card-core-lift[OF finE sunflower, of k c F 1, OF - - F' cardF' -
False]
  obtain S where S ⊆ F and main: sunflower S card S = r 1 ≤ card (⋂ S) by
auto
  thus ?thesis by (intro exI[of - S], auto)
next
  case True
  thus ?thesis by (intro exI[of - {}], auto simp: empty-sunflower)
qed


end
```

2 The Sunflower Lemma

We formalize the proof of the sunflower lemma of Erdős and Rado [2], as it is presented in the textbook [3, Chapter 6]. We further integrate Exercise 6.2 from the textbook, which provides a lower bound on the existence of sunflowers.

```
theory Erdos-Rado-Sunflower
  imports
    Sunflower
begin
```

When removing an element from all subsets, then one can afterwards add these elements to a sunflower and get a new sunflower.

```
lemma sunflower-remove-element-lift:
  assumes S: S ⊆ { A − {a} | A . A ∈ F ∧ a ∈ A}
    and sf: sunflower S
  shows ∃ Sa. sunflower Sa ∧ Sa ⊆ F ∧ card Sa = card S ∧ Sa = insert a ‘ S
proof (intro exI[of - insert a ‘ S] conjI refl)
  let ?Sa = insert a ‘ S
  {
    fix B
    assume B ∈ ?Sa
    then obtain C where C: C ∈ S and B: B = insert a C
      by auto
    from C S obtain T where T ∈ F a ∈ T C = T − {a}
      by auto
    with B have B = T by auto
    with ‹T ∈ F› have B ∈ F by auto
  }
```

```
  thus SaF: ?Sa ⊆ F by auto
  have inj: inj-on (insert a) S using S
    by (intro inj-on-inverseI[of - λ B. B − {a}], auto)
  thus card ?Sa = card S by (rule card-image)
  show sunflower ?Sa unfolding sunflower-def
  proof (intro allI, intro impI)
    fix x
    assume ∃ C D. C ∈ ?Sa ∧ D ∈ ?Sa ∧ C ≠ D ∧ x ∈ C ∧ x ∈ D
    then obtain C D where ∗: C ∈ ?Sa D ∈ ?Sa C ≠ D x ∈ C x ∈ D
      by auto
    from ∗(1−2) obtain C′ D′ where
      ∗∗: C′ ∈ S D′ ∈ S C = insert a C′ D = insert a D′
      by auto
    with ‹C ≠ D› inj have CD′: C′ ≠ D′ by auto
    show ∀ E. E ∈ ?Sa ⟶ x ∈ E
    proof (cases x = a)
      case False
      with ∗ ∗∗ have x ∈ C′ x ∈ D′ by auto
      with ∗∗ CD′ have ∃ C D. C ∈ S ∧ D ∈ S ∧ C ≠ D ∧ x ∈ C ∧ x ∈ D by
auto
      from sf[unfolded sunflower-def, rule-format, OF this]
      show ?thesis by auto
    qed auto
  qed
qed
```

The sunflower-lemma of Erdős and Rado: if a set has a certain size and all elements have the same cardinality, then a sunflower exists.

```
lemma Erdos-Rado-sunflower-same-card:
  assumes ∀ A ∈ F. finite A ∧ card A = k
    and card F > (r − 1)^k ∗ fact k
  shows ∃ S. S ⊆ F ∧ sunflower S ∧ card S = r ∧ {} ∉ S
  using assms
proof (induct k arbitrary: F)
  case 0
  hence F = {{}} ∨ F = {} card F ≥ 2 by auto
  hence False by auto
  thus ?case by simp
next
  case (Suc k F)
  define pd-sub :: ′a set set ⇒ nat ⇒ bool where
    pd-sub = (λ G t. G ⊆ F ∧ card G = t ∧ pairwise disjnt G ∧ {} ∉ G)
  show ?case
  proof (cases ∃ t G. pd-sub G t ∧ t ≥ r)
    case True
    then obtain t G where pd-sub: pd-sub G t and t: t ≥ r by auto
    from pd-sub[unfolded pd-sub-def] pairwise-disjnt-imp-sunflower
    have ∗: G ⊆ F card G = t sunflower G {} ∉ G by auto
    from t ‹card G = t› obtain H where H ⊆ G card H = r
```

```
    by (metis obtain-subset-with-card-n)
  with sunflower-subset[OF ‹H ⊆ G›] ∗ show ?thesis by blast
next
  case False
  define P where P = (λ t. ∃ G. pd-sub G t)
  have ex: ∃ t. P t unfolding P-def
    by (intro exI[of - 0] exI[of - {}], auto simp: pd-sub-def)
  have large': ⋀ t. P t ⟹ t < r using False unfolding P-def by auto
  hence large: ⋀ t. P t ⟹ t ≤ r by fastforce
  define t where t = (GREATEST t. P t)
  from GreatestI-ex-nat[OF ex large, folded t-def] have Pt: P t .
  from Greatest-le-nat[of P, OF - large]
  have greatest: ⋀ s. P s ⟹ s ≤ t unfolding t-def by auto
  from large'[OF Pt] have tr: t ≤ r − 1 by simp
  from Pt[unfolded P-def pd-sub-def] obtain G where
    cardG: card G = t and
    disj: pairwise disjnt G and
    GF: G ⊆ F
    by blast
  define A where A = (⋃ G)
  from Suc(3) have card F > 0 by simp
  hence finite F by (rule card-ge-0-finite)
  from GF ‹finite F› have finG: finite G by (rule finite-subset)
  have card (⋃ G) ≤ sum card G
    using card-Union-le-sum-card by blast
  also have ... ≤ of-nat (card G) ∗ Suc k
    by (metis GF Suc.prems(1) le-Suc-eq subsetD sum-bounded-above)
  also have ... ≤ (r − 1) ∗ Suc k
    using tr[folded cardG] by (metis id-apply mult-le-mono1 of-nat-eq-id)
  finally have cardA: card A ≤ (r − 1) ∗ Suc k unfolding A-def .
  {
    fix B
    assume ∗: B ∈ F
    with Suc(2) have nE: B ≠ {} by auto
    from Suc(2) have eF: {} ∉ F by auto
    have B ∩ A ≠ {}
    proof
      assume dis: B ∩ A = {}
      hence disj: pairwise disjnt ({B} ∪ G)  using disj unfolding A-def
        by (smt (verit, ccfv-SIG) Int-commute Un-iff
            Union-disjoint disjnt-def pairwise-def singleton-iff)
      from nE dis have B ∉ G unfolding A-def by auto
      with finG have c: card ({B} ∪ G) = Suc t by (simp add: cardG)
      have P (Suc t) unfolding P-def pd-sub-def
        by (intro exI[of - {B} ∪ G], insert eF disj c ∗ GF, auto)
      with greatest show False by force
    qed
  } note overlap = this
  have F ≠ {} using Suc(2−) by auto
```

with *overlap* **have** *Ane*: *A* ≠ {} **unfolding** *A-def* **by** *auto*
have *finite A* **unfolding** *A-def* **using** *finG Suc*(*2*−) *GF* **by** *auto*
let *?g* = λ *B x*. *x* ∈ *B* ∩ *A*
define *f* **where** *f* = (λ *B*. *SOME x*. *?g B x*)
have *f* ∈ *F* → *A*
proof
  fix *B*
  assume *B* ∈ *F*
  from *overlap*[*OF this*] **have** ∃ *x*. *?g B x* **unfolding** *A-def* **by** *auto*
  from *someI-ex*[*OF this*] **show** *f B* ∈ *A* **unfolding** *f-def* **by** *auto*
qed
from *pigeonhole-card*[*OF this* ‹*finite F*› ‹*finite A*› *Ane*]
obtain *a* **where** *a*: *a* ∈ *A*
  and *le*: *card F* ≤ *card* (*f* −‘ {*a*} ∩ *F*) ∗ *card A* **by** *auto*
{
  fix *S*
  assume *S* ∈ *F f S* ∈ {*a*}
  with *someI-ex*[*of ?g S*] *a overlap*[*OF this*(*1*)]
  have *a* ∈ *S* **unfolding** *f-def* **by** *auto*
} **note** *FaS* = *this*
let *?F* = {*S* − {*a*} | *S* . *S* ∈ *F* ∧ *f S* ∈ {*a*}}
from *cardA* **have** ((*r* − *1*) ^ *k* ∗ *fact k*) ∗ *card A* ≤ ((*r* − *1*) ^ *k* ∗ *fact k*) ∗ ((*r* − *1*) ∗ *Suc k*)
  by *simp*
also have ... = (*r* − *1*) ^(*Suc k*) ∗ *fact* (*Suc k*)
  by (*metis* (*no-types*, *lifting*) *fact-Suc mult.assoc mult.commute of-nat-id power-Suc2*)
also have ... < *card* (*f* −‘ {*a*} ∩ *F*) ∗ *card A*
  using *Suc*(*3*) *le* **by** *auto*
also have *f* −‘ {*a*} ∩ *F* = {*S* ∈ *F*. *f S* ∈ {*a*}} **by** *auto*
also have *card* ... = *card* ((λ *S*. *S* − {*a*}) ‘ {*S* ∈ *F*. *f S* ∈ {*a*}})
  by (*subst card-image*; *intro inj-onI refl*, *insert FaS*) *auto*
also have (λ *S*. *S* − {*a*}) ‘ {*S* ∈ *F*. *f S* ∈ {*a*}} = *?F* **by** *auto*
finally have *lt*: (*r* − *1*) ^ *k* ∗ *fact k* < *card ?F* **by** *simp*
have ∀ *A* ∈ *?F*. *finite A* ∧ *card A* = *k* **using** *Suc*(*2*) *FaS* **by** *auto*
from *Suc*(*1*)[*OF this lt*] **obtain** *S*
  where *sunflower S card S* = *r S* ⊆ *?F* **by** *auto*
from ‹*S* ⊆ *?F*› *FaS* **have** *S* ⊆ {*A* − {*a*} |*A*. *A* ∈ *F* ∧ *a* ∈ *A*} **by** *auto*
from *sunflower-remove-element-lift*[*OF this* ‹*sunflower S*›] ‹*card S* = *r*›
show *?thesis* **by** *auto*
qed
qed

Using *sunflower-card-subset-lift* we can easily replace the condition that the cardinality is exactly k by the requirement that the cardinality is at most k. However, then {} ∉ *S* cannot be ensured. Consider $r = 1 \wedge 0 < k \wedge F = \{\{\}\}$.

lemma *Erdos-Rado-sunflower*:
assumes ∀ *A* ∈ *F*. *finite A* ∧ *card A* ≤ *k*

and *card F* > (*r* − *1*)^*k* ∗ *fact k*
shows ∃ *S*. *S* ⊆ *F* ∧ *sunflower S* ∧ *card S* = *r*
by (*rule sunflower-card-subset-lift*[*OF* - *assms*],
metis Erdos-Rado-sunflower-same-card)

We further provide a lower bound on the existence of sunflowers, i.e., Exercise 6.2 of the textbook [3]. To be more precise, we prove that there is a set of sets of cardinality $(r - 1)^k$, where each element is a set of cardinality k, such that there is no subset which is a sunflower with cardinality of at least r.

lemma *sunflower-lower-bound*:
assumes *inf*: *infinite* (*UNIV* :: *'a set*)
and *r*: *r* ≠ *0*
and *rk*: *r* = *1* ⟹ *k* ≠ *0*
shows ∃ *F*.
card F = (*r* − *1*)^*k* ∧ *finite F* ∧
(∀ *A* ∈ *F*. *finite* (*A* :: *'a set*) ∧ *card A* = *k*) ∧
(∄ *S*. *S* ⊆ *F* ∧ *sunflower S* ∧ *card S* ≥ *r*)
proof (*cases r* = *1*)
case *False*
with *r* **have** *r*: *r* > *1* **by** *auto*
show *?thesis*
proof (*induct k*)
case *0*
have *id*: *S* ⊆ {{}} ⟷ (*S* = {} ∨ *S* = {{}}) **for** *S* :: *'a set set* **by** *auto*
show *?case* **using** *r*
by (*intro exI*[*of* - {{}}], *auto simp*: *id*)
next
case (*Suc k*)
then obtain *F* **where**
cardF: *card F* = (*r* − *1*) ^ *k* **and**
fin: *finite F* **and**
AF: ⋀ *A*. (*A* :: *'a set*) ∈ *F* ⟹ *finite A* ∧ *card A* = *k* **and**
sf: ¬ (∃ *S*⊆*F*. *sunflower S* ∧ *r* ≤ *card S*)
by *metis*

main idea: get $k - 1$ fresh elements and add one of these to all elements of F

have *finite* (⋃ *F*) **using** *fin AF* **by** *simp*
hence *infinite* (*UNIV* − ⋃ *F*) **using** *inf* **by** *simp*
from *infinite-arbitrarily-large*[*OF this*, *of r* − *1*]
obtain *New* **where** *New*: *finite New card New* = *r* − *1*
New ∩ ⋃ *F* = {} **by** *auto*
define *G* **where** *G* = (λ (*A*, *a*). *insert a A*) ‘ (*F* × *New*)
show *?case*
proof (*intro exI*[*of* - *G*] *conjI*)
show *finite G* **using** *New fin* **unfolding** *G-def* **by** *simp*
have *card G* = *card* (*F* × *New*) **unfolding** *G-def*
proof ((*subst card-image*; (*intro refl*)?), *intro inj-onI*, *clarsimp*, *goal-cases*)

```
    case (1 A a B b)
    hence ab: a = b using New by auto
    from 1(1) have insert a A − {a} = insert b B − {a} by simp
    also have insert a A − {a} = A using New 1 by auto
    also have insert b B − {a} = B using New 1 ab[symmetric] by auto
    finally show ?case using ab by auto
  qed
  also have ... = card F ∗ card New using New fin by auto
  finally show card G = (r − 1) ^ Suc k
    unfolding cardF New by simp
  {
    fix B
    assume B ∈ G
    then obtain a A where G: a ∈ New A ∈ F B = insert a A
      unfolding G-def by auto
    with AF[of A] New have finite B card B = Suc k
      by (auto simp: card-insert-if)
  }
  thus ∀ A∈G. finite A ∧ card A = Suc k by auto
  show ¬ (∃ S⊆G. sunflower S ∧ r ≤ card S)
  proof (intro notI, elim exE conjE)
    fix S
    assume ∗: S ⊆ G sunflower S r ≤ card S
    define g where g B = (SOME a. a ∈ New ∧ a ∈ B) for B
    {
      fix B
      assume B ∈ S
      with ‹S ⊆ G› have B ∈ G by auto
      hence ∃ a. a ∈ New ∧ a ∈ B unfolding G-def by auto
      from someI-ex[OF this, folded g-def]
      have g B ∈ New g B ∈ B by auto
    } note gB = this
    have card (g ‘ S) ≤ card New
      by (rule card-mono, insert New gB, auto)
    also have ... < r unfolding New using r by simp
    also have ... ≤ card S by fact
    finally have card (g ‘ S) < card S .
    from pigeonhole[OF this] have ¬ inj-on g S .
    then obtain B1 B2 where B12: B1 ∈ S B2 ∈ S B1 ≠ B2 g B1 = g B2
      unfolding inj-on-def by auto
    define a where a = g B2
    from B12 gB[of B1] gB[of B2] have a: a ∈ New a ∈ B1 a ∈ B2
      unfolding a-def by auto
    with B12 have ∃ B1 B2. B1 ∈ S ∧ B2 ∈ S ∧ B1 ≠ B2 ∧ a ∈ B1 ∧ a ∈
B2
      unfolding a-def by blast
    from ‹sunflower S›[unfolded sunflower-def, rule-format, OF this]
    have aS: B ∈ S ⟹ a ∈ B for B by auto
    define h where h B = B − {a} for B
```

```
        define T where T = h ` S
        have ∃S⊆F. sunflower S ∧ r ≤ card S
        proof (intro exI[of - T] conjI)
          {
            fix B
            assume B ∈ S
            have hB: h B = B − {a}
              unfolding h-def T-def by auto
            from aS ‹B ∈ S› have aB: a ∈ B by auto
            from ‹B ∈ S› ‹S ⊆ G› obtain a' A where AF: A ∈ F
              and B: B = insert a' A
              and a': a' ∈ New unfolding G-def by force
            from aB B a' New AF a(1) hB AF have insert a (h B) = B h B = A
by auto
            hence insert a (h B) = B h B ∈ F insert a (h B) ∈ S using AF ‹B ∈
S› by auto
          } note main = this
          have CTS: C ∈ T ⟹ insert a C ∈ S for C using main unfolding
T-def by force
          show T ⊆ F unfolding T-def using main by auto
          have r ≤ card S by fact
          also have ... = card T unfolding T-def
            by (subst card-image, intro inj-on-inverseI[of - insert a], insert main,
auto)
          finally show r ≤ card T .
          show sunflower T unfolding sunflower-def
          proof (intro allI impI, elim exE conjE, goal-cases)
            case (1 x C C1 C2)
            from CTS[OF ‹C1 ∈ T›] CTS[OF ‹C2 ∈ T›] CTS[OF ‹C ∈ T›]
            have ∗: insert a C1 ∈ S insert a C2 ∈ S insert a C ∈ S by auto
            from 1 have insert a C1 ≠ insert a C2 using main
              unfolding T-def by auto
            hence ∃A B. A ∈ S ∧ B ∈ S ∧ A ≠ B ∧ x ∈ A ∧ x ∈ B
              using ∗ 1 by auto
            from ‹sunflower S›[unfolded sunflower-def, rule-format, OF this ∗(3)]
            have x ∈ insert a C .
            with 1 show x ∈ C unfolding T-def h-def by auto
          qed
        qed
        with sf
        show False ..
      qed
    qed
  qed
next
  case r: True
  with rk have k ≠ 0 by auto
  then obtain l where k: k = Suc l by (cases k, auto)
  show ?thesis unfolding r k
```

by (*intro exI*[*of* - {}], *auto*)
qed

The difference between the lower and the upper bound on the existence of sunflowers as they have been formalized is *fact k*. There is more recent work with tighter bounds [1], but we only integrate the initial result of Erdős and Rado in this theory.

We further provide the Erdős Rado lemma lifted to obtain non-empty cores or cores of arbitrary cardinality.

lemma *Erdos-Rado-sunflower-card-core*:
assumes *finite E*
and $\forall\ A \in F.\ A \subseteq E \wedge s \leq card\ A \wedge card\ A \leq k$
and $card\ F > (card\ E\ choose\ s) * (r - 1)\hat{}k * fact\ k$
and $s \neq 0$
and $r \neq 0$
shows $\exists\ S.\ S \subseteq F \wedge sunflower\ S \wedge card\ S = r \wedge card\ (\bigcap\ S) \geq s$
by (*rule sunflower-card-core-lift*[*OF assms*(*1*) - *assms*(*2*) - *assms*(*4*−*5*),
of (*r* − *1*)^*k* ∗ *fact k*],
rule Erdos-Rado-sunflower, *insert assms*(*3*), *auto simp*: *ac-simps*)

lemma *Erdos-Rado-sunflower-nonempty-core*:
assumes *finite E*
and $\forall\ A \in F.\ A \subseteq E \wedge card\ A \leq k$
and $\{\} \notin F$
and $card\ F > card\ E * (r - 1)\hat{}k * fact\ k$
shows $\exists\ S.\ S \subseteq F \wedge sunflower\ S \wedge card\ S = r \wedge \bigcap\ S \neq \{\}$
by (*rule sunflower-nonempty-core-lift*[*OF assms*(*1*)
- *assms*(*2*−*3*), *of* (*r* − *1*)^*k* ∗ *fact k*],
rule Erdos-Rado-sunflower, *insert assms*(*4*), *auto simp*: *ac-simps*)

end

References

[1] Ryan Alweiss, Shachar Lovett, Kewen Wu, and Jiapeng Zhang. Improved bounds for the sunflower lemma. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 624–630. ACM, 2020. doi:10.1145/3357713.3384234.

[2] Paul Erdős and Richard Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 35:85–90, 1960. doi:10.1112/jlms/s1-35.1.85.

[3] Stasys Jukna. *Extremal Combinatorics.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2011. doi:10.1007/978-3-642-17364-6_6.