Derangements

Lukas Bulwahn

December 17, 2016

Abstract

The Derangements Formula describes the number of fixpoint-free permutations as closed-form formula. This theorem is the 88th theorem of the Top 100 Theorems list.

Contents

1 Derangements

theory *Derangements*
imports
  ~~/src/HOL/Library/Permutations
  ~~/src/HOL/Decision-Procs/Approximation
begin

1.1 Preliminaries

1.1.1 Additions to *Finite-Set* Theory

lemma *card-product-dependent*:
  assumes *finite* S $\forall x \in S.$ *finite* $(T\ x)$
  shows *card* $\{(x, y).\ x \in S \wedge y \in T\ x\} = (\sum x \in S.$ *card* $(T\ x))$
using *card-SigmaI*[*OF assms*, *symmetric*] **by** (*auto intro*!: *arg-cong*[**where** *f*=*card*] *simp add*: *Sigma-def*)

1.1.2 Additions to *Permutations* Theory

lemma *permutes-imp-bij'*:
  assumes *p permutes S*
  shows *bij p*
using *assms* **by** (*meson bij-def permutes-inj permutes-surj*)

lemma *permutesE*:
  assumes *p permutes S*
  obtains *bij p* $\forall x.\ x \notin S \longrightarrow p\ x = x$
using *assms* **by** (*simp add*: *permutes-def permutes-imp-bij'*)

lemma *bij-imp-permutes'*:
  assumes *bij p* $\forall x.\ x \notin A \longrightarrow p\ x = x$
  shows *p permutes A*
using *assms bij-imp-permutes permutes-superset* **by** *force*

lemma *permutes-swap*:
  assumes *p permutes S*
  shows *Fun.swap x y p permutes* (*insert x* (*insert y S*))
proof −
  from *assms* **have** *p permutes* (*insert x* (*insert y S*)) **by** (*meson permutes-subset subset-insertI*)
  moreover have *Fun.swap x y id permutes* (*insert x* (*insert y S*)) **by** (*simp add*: *permutes-swap-id*)
  ultimately show *Fun.swap x y p permutes* (*insert x* (*insert y S*))
    by (*metis comp-id comp-swap permutes-compose*)
qed

lemma *bij-extends*:
  $bij\ p \Longrightarrow p\ x = x \Longrightarrow bij\ (p(x := y,\ inv\ p\ y := x))$
unfolding *bij-def*
proof (*rule conjI*; *erule conjE*)
  assume *a*: *inj p* $p\ x = x$
  show $inj\ (p(x := y,\ inv\ p\ y := x))$
  proof (*intro injI*)
    fix $z\ z'$
    assume $(p(x := y,\ inv\ p\ y := x))\ z = (p(x := y,\ inv\ p\ y := x))\ z'$
    from *this a* **show** $z = z'$
      by (*auto split*: *if-split-asm simp add*: *inv-f-eq inj-eq*)
  qed
next
  assume *a*: *inj p surj p* $p\ x = x$
  {
    fix x'
    from *a* **have** $(p(x := y,\ inv\ p\ y := x))\ (((inv\ p)(y := x,\ x := inv\ p\ y))\ x') = x'$
      by (*auto split*: *if-split-asm*) (*metis surj-f-inv-f*)+
  }
  from *this* **show** $surj\ (p(x := y,\ inv\ p\ y := x))$ **by** (*metis surjI*)

qed

lemma *permutes-add-one*:
assumes p *permutes* S $x \notin S$ $y \in S$
shows $p(x := y,\ inv\ p\ y := x)$ *permutes* $(insert\ x\ S)$
proof (*rule bij-imp-permutes′*)
from *assms* **show** $bij\ (p(x := y,\ inv\ p\ y := x))$
by (*meson bij-extends permutes-def permutes-imp-bij′*)
from *assms* **show** $\forall z.\ z \notin insert\ x\ S \longrightarrow (p(x := y,\ inv\ p\ y := x))\ z = z$
by (*metis fun-upd-apply insertCI permutes-def permutes-inverses*(*1*))
qed

lemma *permutations-skip-one*:
assumes p *permutes* S $x : S$
shows $p(x := x,\ inv\ p\ x := p\ x)$ *permutes* $(S - \{x\})$
proof (*rule bij-imp-permutes′*)
from *assms* **show** $\forall z.\ z \notin S - \{x\} \longrightarrow (p(x := x,\ inv\ p\ x := p\ x))\ z = z$
by (*auto elim*: *permutesE simp add*: *bij-inv-eq-iff*)
(*simp add*: *assms*(*1*) *permutes-in-image permutes-inv*)
from *assms* **have** $inj\ (p(x := x,\ inv\ p\ x := p\ x))$
by (*intro injI*) (*auto split*: *if-split-asm*; *metis permutes-inverses*(*2*))+
from *assms this* **show** $bij\ (p(x := x,\ inv\ p\ x := p\ x))$
by (*metis UNIV-I bij-betw-imageI bij-betw-swap-iff permutes-inj permutes-surj surj-f-inv-f swap-def*)
qed

lemma *permutes-drop-cycle-size-two*:
assumes p *permutes* S $p\ (p\ x) = x$
shows $Fun.swap\ x\ (p\ x)\ p$ *permutes* $(S - \{x,\ p\ x\})$
using *assms* **by** (*auto intro*!: *bij-imp-permutes′ elim*!: *permutesE*) (*metis swap-apply*(*1*,*3*))

1.2 Fixpoint-Free Permutations

definition $derangements :: nat\ set \Rightarrow (nat \Rightarrow nat)\ set$
where
$derangements\ S = \{p.\ p\ permutes\ S \wedge (\forall x \in S.\ p\ x \neq x)\}$

lemma *derangementsI*:
assumes p *permutes* S $\bigwedge x.\ x \in S \Longrightarrow p\ x \neq x$
shows $p \in derangements\ S$
using *assms* **unfolding** *derangements-def* **by** *auto*

lemma *derangementsE*:
assumes $d : derangements\ S$
obtains d *permutes* S $\forall x{\in}S.\ d\ x \neq x$
using *assms* **unfolding** *derangements-def* **by** *auto*

1.3 Properties of Derangements

lemma *derangements-inv*:

assumes d: $d \in$ *derangements* S
shows *inv* $d \in$ *derangements* S
using *assms*
by (*auto intro*!: *derangementsI elim*!: *derangementsE simp add*: *permutes-inv permutes-inv-eq*)

lemma *derangements-in-image*:
assumes $d \in$ *derangements* A $x \in A$
shows $d\ x \in A$
using *assms* **by** (*auto elim*: *derangementsE simp add*: *permutes-in-image*)

lemma *derangements-in-image-strong*:
assumes $d \in$ *derangements* A $x \in A$
shows $d\ x \in A - \{x\}$
using *assms* **by** (*auto elim*: *derangementsE simp add*: *permutes-in-image*)

lemma *derangements-inverse-in-image*:
assumes $d \in$ *derangements* A $x \in A$
shows *inv* $d\ x \in A$
using *assms* **by** (*auto intro*: *derangements-in-image derangements-inv*)

lemma *derangements-fixpoint*:
assumes $d \in$ *derangements* A $x \notin A$
shows $d\ x = x$
using *assms* **by** (*auto elim*!: *derangementsE simp add*: *permutes-def*)

lemma *derangements-no-fixpoint*:
assumes $d \in$ *derangements* A $x \in A$
shows $d\ x \neq x$
using *assms* **by** (*auto elim*: *derangementsE*)

lemma *finite-derangements*:
assumes *finite* A
shows *finite* (*derangements* A)
using *assms* **unfolding** *derangements-def*
by (*auto simp add*: *finite-permutations*)

1.4 Construction of Derangements

lemma *derangements-empty*[*simp*]:
derangements $\{\} = \{id\}$
unfolding *derangements-def* **by** *auto*

lemma *derangements-singleton*[*simp*]:
derangements $\{x\} = \{\}$
unfolding *derangements-def* **by** *auto*

lemma *derangements-swap*:
assumes $d \in$ *derangements* S $x \notin S$ $y \notin S$ $x \neq y$
shows *Fun.swap* $x\ y\ d \in$ *derangements* (*insert* x (*insert* y S))

proof (*rule derangementsI*)
from *assms* **show** *Fun.swap* x y d *permutes* (*insert* x (*insert* y S))
by (*auto intro*: *permutes-swap elim*: *derangementsE*)
from *assms* **have** *s*: $d\ x = x\ d\ y = y$
by (*auto intro*: *derangements-fixpoint*)
{
fix x'
assume x' : *insert* x (*insert* y S)
from *s assms* ‹$x \neq y$› *this* **show** *Fun.swap* $x\ y\ d\ x' \neq x'$
by (*cases* $x' = x$; *cases* $x' = y$) (*auto dest*: *derangements-no-fixpoint*)
}
qed

lemma *derangements-skip-one*:
assumes *d*: $d \in$ *derangements* S **and** $x \in S\ d\ (d\ x) \neq x$
shows $d(x := x,$ *inv* $d\ x := d\ x) \in$ *derangements* $(S - \{x\})$
proof −
from *d* **have** *bij*: *bij d*
by (*auto elim*: *derangementsE simp add*: *permutes-imp-bij'*)
from *d* ‹$x : S$› **have** *that*: $d\ x : S - \{x\}$
by (*auto dest*: *derangements-in-image derangements-no-fixpoint*)
from *d* ‹$d\ (d\ x) \neq x$› *bij* **have** $\forall\, x' {\in} S - \{x\}.\ (d(x := x,$ *inv* $d\ x := d\ x))\ x' \neq x'$
by (*auto elim!*: *derangementsE simp add*: *bij-inv-eq-iff*)
from *d* ‹$x : S$› *this* **show** *derangements*: $d(x{:=}x,$ *inv* $d\ x{:=}\ d\ x)$: *derangements* $(S - \{x\})$
by (*meson derangementsE derangementsI permutations-skip-one*)
qed

lemma *derangements-add-one*:
assumes $d \in$ *derangements* $S\ x \notin S\ y \in S$
shows $d(x := y,$ *inv* $d\ y := x) \in$ *derangements* (*insert* x S)
proof (*rule derangementsI*)
from *assms* **show** $d(x := y,$ *inv* $d\ y := x)$ *permutes* (*insert* x S)
by (*auto intro*: *permutes-add-one elim*: *derangementsE*)
next
fix z
assume z : *insert* x S
from *assms this derangements-inverse-in-image*[*OF assms*(*1*), *of y*]
show $(d(x := y,$ *inv* $d\ y := x))\ z \neq z$ **by** (*auto elim*: *derangementsE*)
qed

lemma *derangements-drop-minimal-cycle*:
assumes $d \in$ *derangements* $S\ d\ (d\ x) = x$
shows *Fun.swap* $x\ (d\ x)\ d \in$ *derangements* $(S - \{x, d\ x\})$
proof (*rule derangementsI*)
from *assms* **show** *Fun.swap* $x\ (d\ x)\ d$ *permutes* $(S - \{x, d\ x\})$
by (*meson derangementsE permutes-drop-cycle-size-two*)
next

```
  fix y
  assume y ∈ S − {x, d x}
  from assms this show Fun.swap x (d x) d y ≠ y
    by (auto elim: derangementsE)
qed
```

1.5 Cardinality of Derangements

1.5.1 Recursive Characterization

```
fun count-derangements :: nat ⇒ nat
where
  count-derangements 0 = 1
| count-derangements (Suc 0) = 0
| count-derangements (Suc (Suc n)) = (n + 1) ∗ (count-derangements (Suc n) +
count-derangements n)

lemma card-derangements:
  assumes finite S card S = n
  shows card (derangements S) = count-derangements n
using assms
proof (induct n arbitrary: S rule: count-derangements.induct)
  case 1
  from this show ?case by auto
next
  case 2
  from this derangements-singleton finite-derangements show ?case
   using Finite-Set.card-0-eq card-eq-SucD count-derangements.simps(2) by fastforce
next
  case (3 n)
  from 3(4) obtain x where x ∈ S using card-eq-SucD insertI1 by auto
 let ?D1 = (λ(y, d). Fun.swap x y d) ‘ {(y, d). y ∈ S & y ≠ x & d : derangements
(S − {x, y})}
  let ?D2 = (λ(y, f). f(x:=y, inv f y := x)) ‘ ((S − {x}) × derangements (S −
{x}))
  from ‹x ∈ S› have subset1: ?D1 ⊆ derangements S
  proof (auto)
    fix y d
    assume y ∈ S y ≠ x
    assume d: d ∈ derangements (S − {x, y})
    from ‹x : S› ‹y : S› have S: S = insert x (insert y (S − {x, y})) by auto
    from d ‹x : S› ‹y : S› ‹y ≠ x› show Fun.swap x y d ∈ derangements S
      by (subst S) (auto intro!: derangements-swap)
  qed
  have subset2: ?D2 ⊆ derangements S
  proof (rule subsetI, erule imageE, simp split: prod.split-asm, (erule conjE)+)
    fix d y
    assume d : derangements (S − {x}) y : S y ≠ x
    from this have d(x := y, inv d y := x) ∈ derangements (insert x (S − {x}))
      by (intro derangements-add-one) auto
```

from *this* ‹x : S› **show** $d(x := y,\ inv\ d\ y := x) \in$ *derangements S*
using *insert-Diff* **by** *fastforce*
qed
have *split*: *derangements S* = *?D1* ∪ *?D2*
proof
from *subset1 subset2* **show** *?D1* ∪ *?D2* ⊆ *derangements S* **by** *simp*
next
show *derangements S* ⊆ *?D1* ∪ *?D2*
proof
fix *d*
assume *d*: *d* : *derangements S*
show *d* : *?D1* ∪ *?D2*
proof (*cases d* (*d x*) = *x*)
case *True*
from ‹x : S› *d* **have** $d\ x \in S$ $d\ x \neq x$
by (*auto simp add*: *derangements-in-image derangements-no-fixpoint*)
from *d True* **have** *Fun.swap x* (*d x*) *d* ∈ *derangements* ($S - \{x,\ d\ x\}$)
by (*rule derangements-drop-minimal-cycle*)
from ‹$d\ x \in S$› ‹$d\ x \neq x$› *this* **have** *d* : *?D1*
by (*auto intro*: *image-eqI*[**where** x = (*d x*, *Fun.swap x* (*d x*) *d*)])
from *this* **show** *?thesis* **by** *auto*
next
case *False*
from *d* **have** *bij*: *bij d*
by (*auto elim*: *derangementsE simp add*: *permutes-imp-bij'*)
from *d* ‹x : S› **have** *that*: $d\ x : S - \{x\}$
by (*intro derangements-in-image-strong*)
from *d* ‹x : S› *False* **have** *derangements*: $d(x{:=}x,\ inv\ d\ x{:=}\ d\ x)$: *derangements* ($S - \{x\}$)
by (*auto intro*: *derangements-skip-one*)
from *this* **have** *bij* ($d(x := x,\ inv\ d\ x{:=}\ d\ x)$)
by (*metis derangementsE permutes-imp-bij'*)+
from *this* **have** *a*: $inv\ (d(x := x,\ inv\ d\ x := d\ x))\ (d\ x) = inv\ d\ x$
by (*metis bij-inv-eq-iff fun-upd-same*)
from *bij* **have** *x*: $d\ (inv\ d\ x) = x$ **by** (*meson bij-inv-eq-iff*)
from *d derangements-inv*[*of d*] ‹x : S› **have** $inv\ d\ x \neq x$ $d\ x \neq x$
by (*auto dest*: *derangements-no-fixpoint*)
from *this a x* **have** *d-eq*: $d = d(inv\ d\ x := d\ x,\ x := d\ x,\ inv\ (d(x := x,\ inv\ d\ x := d\ x))\ (d\ x) := x)$
by *auto*
from *derangements that* **have** ($d\ x$, $d(x{:=}x,\ inv\ d\ x{:=}d\ x)$) : (($S - \{x\}$) × *derangements* ($S - \{x\}$)) **by** *auto*
from *d-eq this* **have** *d* : *?D2*
by (*auto intro*: *image-eqI*[**where** x = ($d\ x$, $d(x{:=}x,\ inv\ d\ x{:=}d\ x)$)])
from *this* **show** *?thesis* **by** *auto*
qed
qed
qed
have *no-intersect*: *?D1* ∩ *?D2* = {}

proof –
have *that*: $\bigwedge d.\ d \in\ ?D1 \Longrightarrow d\ (d\ x) = x$
using *Diff-iff Diff-insert2 derangements-fixpoint insertI1 swap-apply*(*2*) **by** *fastforce*
have $\bigwedge d.\ d \in\ ?D2 \Longrightarrow d\ (d\ x) \neq x$
proof –
fix *d*
assume *a*: $d \in\ ?D2$
from *a* **obtain** $y\ d'$ **where** *d*: $d = d'(x := y,\ inv\ d'\ y := x)$
$d' \in derangements\ (S - \{x\})\ y \in S - \{x\}$
by *auto*
from *d*(*2*) **have** *inv*: $inv\ d' \in derangements\ (S - \{x\})$
by (*rule derangements-inv*)
from *d* **have** *inv-x*: $inv\ d'\ y \neq x$
by (*auto dest*: *derangements-inverse-in-image*)
from *inv* **have** *inv-y*: $inv\ d'\ y \neq y$
using *d*(*3*) *derangements-no-fixpoint* **by** *blast*
from *d inv-x* **have** *1*: $d\ x = y$ **by** *auto*
from *d inv-y* **have** *2*: $d\ y = d'\ y$ **by** *auto*
from *d*(*2*, *3*) **have** *3*: $d'\ y \in S - \{x\}$
by (*auto dest*: *derangements-in-image*)
from *1 2 3* **show** $d\ (d\ x) \neq x$ **by** *auto*
qed
from *this that* **show** *?thesis* **by** *blast*
qed
have *inj*: *inj-on* $(\lambda(y, f).\ Fun.swap\ x\ y\ f)\ \{(y, f).\ y \in S\ \&\ y \neq x\ \&\ f : derangements\ (S - \{x, y\})\}$
unfolding *inj-on-def*
by (*clarify*; *metis DiffD2 derangements-fixpoint insertI1 insert-commute swap-apply*(*1*) *swap-nilpotent*)
have *eq*: $\{(y, f).\ y \in S\ \&\ y \neq x\ \&\ f : derangements\ (S - \{x, y\})\} = \{(y, f).\ y \in S - \{x\}\ \&\ f : derangements\ (S - \{x, y\})\}$
by *simp*
have *eq'*: $\{(y, f).\ y \in S\ \&\ y \neq x\ \&\ f : derangements\ (S - \{x, y\})\} = Sigma\ (S - \{x\})\ (\%y.\ derangements\ (S - \{x, y\}))$
unfolding *Sigma-def* **by** *auto*
have *card*: $\bigwedge\ y.\ y \in S - \{x\} \Longrightarrow card\ (derangements\ (S - \{x, y\})) = count\text{-}derangements\ n$
proof –
fix *y*
assume $y \in S - \{x\}$
from *3*(*3*, *4*) ‹$x \in S$› *this* **have** *card* $(S - \{x, y\}) = n$
by (*metis Diff-insert2 card-Diff-singleton diff-Suc-1 finite-Diff*)
from *3*(*3*) *3*(*2*)[*OF* - *this*] **show** *card* (*derangements* $(S - \{x, y\})$) = *count-derangements n* **by** *auto*
qed
from *3*(*3*, *4*) ‹$x : S$› **have** *card2*: *card* $(S - \{x\}) = Suc\ n$ **by** (*simp add*: *card.insert-remove insert-absorb*)
from *inj* **have** *card ?D1* = *card* $\{(y, f).\ y \in S - \{x\} \wedge f \in derangements$ (*S*

$- \{x,\ y\})\}$
by (*simp add*: *card-image*)
also from *3(3)* have ... $= (\sum y{\in}S - \{x\}.\ card\ (derangements\ (S - \{x,\ y\})))$
by (*subst card-product-dependent*) (*simp add*: *finite-derangements*)+
finally have *card-1*: *card ?D1* $=(Suc\ n) * count\text{-}derangements\ n$
using *card card2* by *auto*
have *inj-2*: *inj-on* $(\lambda(y, f).\ f(x := y,\ inv\ f\ y := x))\ ((S - \{x\}) \times derangements\ (S - \{x\}))$
proof $-$
{
fix $d\ d'\ y\ y'$
assume *1*: $d \in derangements\ (S - \{x\})\ d' \in derangements\ (S - \{x\})$
assume *2*: $y \in S\ y \neq x\ y' \in S\ y' \neq x$
assume *eq*: $d(x := y,\ inv\ d\ y := x) = d'(x := y',\ inv\ d'\ y' := x)$
from *1 2 eq* ‹$x \in S$› have *y*: $y = y'$
by (*metis Diff-insert-absorb derangements-in-image derangements-inv fun-upd-same fun-upd-twist insert-iff mk-disjoint-insert*)
have $d = d'$
proof
fix *z*
from *1* have *d-x*: $d\ x = d'\ x$
by (*auto dest*!: *derangements-fixpoint*)
from *1* have *bij*: *bij d bij d'*
by (*metis derangementsE permutes-imp-bij'*)+
from *this* have *d-d*: $d\ (inv\ d\ y) = y\ d'\ (inv\ d'\ y') = y'$
by (*simp add*: *bij-is-surj surj-f-inv-f*)+
from *1 2 bij* have *neq*: $d\ (inv\ d'\ y') \neq x \wedge d'\ (inv\ d\ y) \neq x$
by (*metis Diff-iff bij-inv-eq-iff derangements-fixpoint singletonI*)
from *eq* have $(d(x := y,\ inv\ d\ y := x))\ z = (d'(x := y',\ inv\ d'\ y' := x))\ z$ by *auto*
from *y d-x d-d neq this* show $d\ z = d'\ z$ by (*auto split*: *if-split-asm*)
qed
from *y this* have $y = y'\ \&\ d = d'$ by *auto*
}
from *this* show *?thesis*
unfolding *inj-on-def* by *auto*
qed
from *3(3) 3(1)*[*OF - card2*] have *card3*: *card* $(derangements\ (S - \{x\})) = count\text{-}derangements\ (Suc\ n)$
by *auto*
from *3(3) inj-2* have *card-2*: *card ?D2* $= (Suc\ n) * count\text{-}derangements\ (Suc\ n)$
by (*simp only*: *card-image*) (*auto simp add*: *card-cartesian-product card3 card2*)
from ‹*finite S*›have *finite1*: *finite ?D1*
unfolding *eq'* by (*auto intro*: *finite-derangements*)
from ‹*finite S*› have *finite2*: *finite ?D2*
by (*auto intro*: *finite-cartesian-product finite-derangements*)
show *?case*
by (*simp add*: *split card-Un-disjoint finite1 finite2 no-intersect card-1 card-2*

field-simps)
qed

1.5.2 Closed-Form Characterization

lemma *count-derangements*:
  count-derangements n = *fact* $n * (\sum k \in \{0..n\}.\ (-1)\ \hat{}\ k\ /\ fact\ k)$
proof (*induct n rule*: *count-derangements.induct*)
  case (*3 n*)
  let *?f* = $\lambda n.$ *fact* $n * (\sum k = 0..n.\ (-\ 1)\ \hat{}\ k\ /\ fact\ k)$
  have *real* (*count-derangements* (*Suc* (*Suc n*))) = (*n* + *1*) ∗ (*count-derangements* (*n* + *1*) + *count-derangements n*)
    unfolding *count-derangements.simps* **by** *simp*
  also have ... = *real* (*n* + *1*) ∗ (*real* (*count-derangements* (*n* + *1*)) + *real* (*count-derangements n*))
    by (*simp only*: *of-nat-mult of-nat-add*)
  also have ... = (*n* + *1*) ∗ (*?f* (*n* + *1*) + *?f n*)
    unfolding *3*(*2*) *3*(*1*)[*unfolded Suc-eq-plus1*] **..**
  also have (*n* + *1*) ∗ (*?f* (*n* + *1*) + *?f n*) = *?f* (*n* + *2*)
  proof −
    define *f* **where** *f n* = ((*fact n*) :: *real*) $* (\sum k = 0..n.\ (-\ 1)\ \hat{}\ k\ /\ fact\ k)$ **for** *n*
    have *f-eq*: $\bigwedge n.$ *f* (*n* + *1*) = (*n* + *1*) ∗ *f n* + (− *1*) ˆ (*n* + *1*)
    proof −
      fix *n*
      have *?f* (*n* + *1*) = (*n* + *1*) ∗ *fact n* $* (\sum k = 0..n.\ (-\ 1)\ \hat{}\ k\ /\ fact\ k)$ + *fact* (*n* + *1*) ∗ ((− *1*) ˆ (*n* + *1*) / *fact* (*n* + *1*))
        by (*simp add*: *field-simps*)
      also have ... = (*n* + *1*) ∗ *?f n* + (− *1*) ˆ (*n* + *1*) **by** (*simp del*: *One-nat-def*)
      finally show *?thesis n* **unfolding** *f-def* **by** *simp*
    qed
    from *this* **have** *f-eq'*: $\bigwedge n.$ (*n* + *1*) ∗ *f n* = *f* (*n* + *1*) + (− *1*) ˆ *n* **by** *auto*
    from *f-eq'*[*of n*] **have** (*n* + *1*) ∗ (*f* (*n* + *1*) + *f n*) = ((*n* + *1*) ∗ *f* (*n* + *1*)) + *f* (*n* + *1*) + (− *1*) ˆ *n*
      by (*simp only*: *distrib-left of-nat-add of-nat-1*)
    also have ... = (*n* + *2*) ∗ *f* (*n* + *1*) + (− *1*) ˆ (*n* + *2*)
      by (*simp del*: *One-nat-def add-2-eq-Suc'* *add*: *algebra-simps*) *simp*
    also from *f-eq*[*of n* + *1*] **have** ... = *f* (*n* + *2*) **by** *simp*
    finally show *?thesis* **unfolding** *f-def* **by** *simp*
  qed
  finally show *?case* **by** *simp*
qed (*auto*)

1.5.3 Approximation of Cardinality

lemma *approximation*:
  assumes $n \neq 0$
  shows *abs*(*real* (*count-derangements n*) − *fact n* / *exp 1*) < *1* / *2*
proof (*cases* $n \geq 5$)
  case *False*

from *assms this* **have** *n*: $n = 1 \vee n = 2 \vee n = 3 \vee n = 4$ **by** *auto*
have *numeral-4-eq-4*: $4 = Suc\ (Suc\ (Suc\ (Suc\ 0)))$ **by** *auto*
have $exp\ 1 > (2 :: real)$ **by** (*approximation 4*)
have $exp\ 1 < (3 :: real)$ **by** (*approximation 6*)
have $(2 :: real) < 6\ /\ exp\ 1$ **by** (*approximation 6*)
have $(12 :: real)\ /\ exp\ 1 < 5$ **by** (*approximation 5*)
have $9 > 24\ /\ exp\ (1 :: real)$ **by** (*approximation 9*)
have $(17 :: real) < 48\ /\ exp\ 1$ **by** (*approximation 8*)
from ‹$exp\ 1 > 2$› **have** *1*: $abs(real\ (count\text{-}derangements\ 1) - fact\ 1\ /\ exp\ 1) < 1\ /\ 2$
by *simp*
from ‹$exp\ 1 > 2$› ‹$exp\ 1 < 3$› **have** *2*: $abs(real\ (count\text{-}derangements\ 2) - fact\ 2\ /\ exp\ 1) < 1\ /\ 2$
by (*auto simp add*: *numeral-2-eq-2 abs-real-def*)
from ‹$2 < 6\ /\ exp\ 1$› ‹$12\ /\ exp\ 1 < 5$› **have** *3*: $abs(real\ (count\text{-}derangements\ 3) - fact\ 3\ /\ exp\ 1) < 1\ /\ 2$
by (*simp add*: *numeral-3-eq-3*)
from ‹$9 > 24\ /\ exp\ 1$› ‹$17 < 48\ /\ exp\ 1$› **have** *4*: $abs(real\ (count\text{-}derangements\ 4) - fact\ 4\ /\ exp\ 1) < 1\ /\ 2$
by (*simp add*: *numeral-4-eq-4*)
from *1 2 3 4 n* **show** *?thesis* **by** *auto*
next
case *True*
have $exp\ 1 < (3 :: real)$ **by** (*approximation 6*)
from *Maclaurin-exp-le*[*of* $-\ 1\ n + 1$]
obtain *t* **where** *t*: $abs\ (t :: real) \le abs\ (-\ 1)$
and *exp*: $exp\ (-\ 1) = (\sum m{<}n + 1.\ (-\ 1)\ \hat{}\ m\ /\ fact\ m) + exp\ t\ /\ fact\ (n + 1) * (-\ 1)\ \hat{}\ (n + 1)$
by *blast*
from *exp* **have** *sum-eq-exp*: $(\sum k = 0..n.\ (-\ 1)\ \hat{}\ k\ /\ fact\ k) = exp\ (-\ 1) - exp\ t\ /\ fact\ (n + 1) * (-\ 1)\ \hat{}\ (n + 1)$
by (*simp add*: *atLeast0AtMost ivl-disj-un*(*2*)[*symmetric*])
have *fact-plus1*: $fact\ (n + 1) = (n + 1) * fact\ n$ **by** *simp*
have *eq*: $|real\ (count\text{-}derangements\ n) - fact\ n\ /\ exp\ 1| = exp\ t\ /\ (n + 1)$
by (*simp del*: *One-nat-def add*: *count-derangements sum-eq-exp exp-minus inverse-eq-divide algebra-simps abs-mult*)
(*simp add*: *fact-plus1*)
from *t* ‹$exp\ 1 < 3$› **have** $exp\ t < 3$ **by** *auto*
(*metis abs-le-iff exp-less-cancel-iff le-less less-le-trans*)
from *this* ‹$n \ge 5$› **show** *?thesis* **by** (*simp add*: *eq*)
qed

definition *round* :: $real \Rightarrow real$
where
$round\ x = floor\ (x + 1\ /\ 2)$

lemma *round-eqI*:
assumes $abs(real\ (n :: nat) - x) < 1\ /\ 2$
shows $round\ x = real\ (n :: nat)$

using *assms* **unfolding** *round-def*
by (*simp add*: *floor-eq*[*of n*] *abs-real-def split*: *if-split-asm*)

theorem *derangements-formula*:
  assumes $n \neq 0$ *finite S card S* $= n$
  shows *card* (*derangements S*) $=$ *round* (*fact n* / *exp 1*)
using *assms* **by** (*auto intro*: *round-eqI*[*symmetric*] *approximation simp add*: *card-derangements*)

end

References

[1] J. Harrison. Formula for the number of derangements. Available at https://github.com/jrh13/hol-light/blob/master/100/derangements.ml.

[2] Wikipedia. Rencontres numbers — Wikipedia, The Free Encyclopedia, 2014. https://en.wikipedia.org/w/index.php?title=Rencontres_numbers&oldid=616214357, [Online; accessed 18-November-2015].

[3] Wikipedia. Derangement — Wikipedia, The Free Encyclopedia, 2015. https://en.wikipedia.org/w/index.php?title=Derangement&oldid=686842426, [Online; accessed 18-November-2015].

[4] Wikipedia. Fixpunktfreie Permutation — Wikipedia, Die freie Enzyklopdie, 2015. https://de.wikipedia.org/w/index.php?title=Fixpunktfreie_Permutation&oldid=146073460, [Online; accessed 18-November-2015].