Nagata Factoriality

Arthur Freitas Ramos* David Barros Hulak
Ruy J. G. B. de Queiroz

April 30, 2026

Abstract

This entry formalizes a prime-generated version of Nagata's factoriality theorem in Isabelle/HOL. It develops the basic theory of prime-generated multiplicative sets, packages a wrapper interface around the AFP entry `Localization_Ring`, and proves record-based descent theorems showing that factoriality descends from a localization to the base ring under prime-generated and prime-or-unit hypotheses on the multiplicative set. The theorem package also includes closure-based corollaries for arbitrary and finite families of prime generators. The application layer specializes this framework to polynomial rings, both for localization away the polynomial variable X and for localizations generated by constant prime polynomials.

Contents

*Maintainer: `arfreita@microsoft.com`

1 Overview

This entry formalizes a prime-generated version of Nagata's factoriality theorem for noetherian domains, following the classical commutative-algebra framework developed by Nagata, Samuel, and Matsumura.[2, 3, 1] It packages the descent theorem itself, closure-based corollaries for prime-generated multiplicative sets, and abstract polynomial applications for localization away the polynomial variable X and for localizations generated by constant prime polynomials.

theory *Prime-Generated*
  imports *HOL−Computational-Algebra.Factorial-Ring*
begin

2 Prime-generated multiplicative sets

This theory isolates the reusable combinatorial layer behind Nagata's factoriality theorem. The full localization argument is developed in later theories; here we focus on the multiplicative sets generated by prime elements and on the closure lemmas that do not depend on any localization API.

definition *avoids* :: *'a* :: *comm-semiring-1 set* ⇒ *'a* ⇒ *bool* **where**
  avoids S p ⟷ (∀ *s*∈*S*. ¬ *p dvd s*)

definition *prime-generated* :: *'a* :: *comm-semiring-1 set* ⇒ *bool* **where**
  prime-generated S ⟷
    (∀ *s*∈*S*. ∃ *M*. (∀ *q*. *q* ∈# *M* ⟶ *q* ∈ *S* ∧ *prime-elem q*) ∧ *prod-mset M* = *s*)

inductive-set *mult-submonoid-closure* :: *'a* :: *comm-monoid-mult set* ⇒ *'a set* **for** *A* **where**
  one-closed: *1* ∈ *mult-submonoid-closure A*
| *generator*: *a* ∈ *A* ⟹ *a* ∈ *mult-submonoid-closure A*
| *mult-closed*:
    a ∈ *mult-submonoid-closure A* ⟹ *b* ∈ *mult-submonoid-closure A* ⟹
      a ∗ *b* ∈ *mult-submonoid-closure A*

definition *powers-set* :: *'a* :: *monoid-mult* ⇒ *'a set* **where**
  powers-set p = {*x*. ∃ *n*. *x* = *p* ^ *n*}

lemma *prime-generatedI*:
  assumes ⋀*s*. *s* ∈ *S* ⟹ ∃ *M*. (∀ *q*. *q* ∈# *M* ⟶ *q* ∈ *S* ∧ *prime-elem q*) ∧ *prod-mset M* = *s*
  shows *prime-generated S*
  using *assms* **unfolding** *prime-generated-def* **by** *blast*

lemma *prime-generatedE*:
  assumes *prime-generated S s* ∈ *S*
  obtains *M* **where** (∀ *q*. *q* ∈# *M* ⟶ *q* ∈ *S* ∧ *prime-elem q*) *prod-mset M* = *s*
  using *assms* **unfolding** *prime-generated-def* **by** *blast*

lemma *prime-generated-powers-set*:
assumes *prime-elem p*
shows *prime-generated* (*powers-set p*)
proof (*rule prime-generatedI*)
fix *s*
assume *s* ∈ *powers-set p*
then obtain *n* **where** *hs*: *s* = *p* ^ *n*
unfolding *powers-set-def* **by** *blast*
have *factors*:
∀ *q*. *q* ∈# *replicate-mset n p* ⟶ *q* ∈ *powers-set p* ∧ *prime-elem q*
proof
fix *q*
show *q* ∈# *replicate-mset n p* ⟶ *q* ∈ *powers-set p* ∧ *prime-elem q*
proof (*cases n*)
case *0*
then show *?thesis*
by *simp*
next
case (*Suc m*)
show *?thesis*
proof
assume *q* ∈# *replicate-mset n p*
from ‹*q* ∈# *replicate-mset n p*› **have** *n* > *0* ∧ *q* = *p*
by (*simp only*: *in-replicate-mset*)
with *Suc* **have** *q-eq*: *q* = *p*
by *simp*
have *q-in-powers*: *q* ∈ *powers-set p*
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq power-one-right powers-set-def q-eq*)
from *q-eq assms* **have** *q-prime*: *prime-elem q*
by *simp*
from *q-in-powers q-prime* **show** *q* ∈ *powers-set p* ∧ *prime-elem q*
by *blast*
qed
qed
qed
show ∃ *M*. (∀ *q*. *q* ∈# *M* ⟶ *q* ∈ *powers-set p* ∧ *prime-elem q*) ∧ *prod-mset M* = *s*
by (*rule exI*[*of* - *replicate-mset n p*]) (*use factors hs* **in** *auto*)
qed

lemma *prime-generated-mult-submonoid-closure*:
assumes ⋀*q*. *q* ∈ *A* ⟹ *prime-elem q*
shows *prime-generated* (*mult-submonoid-closure A*)
proof (*rule prime-generatedI*)
fix *s*
assume *s* ∈ *mult-submonoid-closure A*
then show

```
    ∃M. (∀q. q ∈# M ⟶ q ∈ mult-submonoid-closure A ∧ prime-elem q) ∧
prod-mset M = s
  proof induction
    case one-closed
    show ?case
      by (rule exI[of - {#}]) (auto intro: mult-submonoid-closure.one-closed)
  next
    case (generator a)
    show ?case
    proof (rule exI)
      from generator assms show
        (∀q. q ∈# {#a#} ⟶ q ∈ mult-submonoid-closure A ∧ prime-elem q) ∧
          prod-mset {#a#} = a
        by (auto intro: mult-submonoid-closure.generator)
    qed
  next
    case (mult-closed a b)
    show ?case
      by (metis mult-closed.IH Un-iff prod-mset-Un set-mset-union)
  qed
qed

lemma zero-notin-prime-generated:
  assumes prime-generated S
  shows (0 :: 'a :: semidom) ∉ S
  using assms prime-generated-def by force

end
theory Localization-Interface
  imports
    HOL−Algebra.Ring-Divisibility
    HOL−Algebra.QuotRing
    Localization-Ring.Localization
begin
```

3 Localization helper lemmas

The AFP entry *Localization-Ring.Localization* develops localizations as quotient rings in the HOL-Algebra hierarchy. For the present development we package a small wrapper layer at the level of equality of representatives, denominator rescaling, units coming from the multiplicative set, and injectivity of the canonical map.

```
context eq-obj-rng-of-frac
begin

lemma fraction-eq-iff-rel:
  assumes (r, s) ∈ carrier rel
    and (r', s') ∈ carrier rel
```

shows $(r \mid_{rel} s) = (r' \mid_{rel} s') \longleftrightarrow (r,\ s) .=_{rel} (r',\ s')$
proof
from *assms* **have** *rs*: $(r,\ s) \in$ *carrier* $R \times S$ **and** *rs'*: $(r',\ s') \in$ *carrier* $R \times S$
by (*simp-all add*: *rel-def*)
assume $(r \mid_{rel} s) = (r' \mid_{rel} s')$
then show $(r,\ s) .=_{rel} (r',\ s')$
using *eq-class-to-rel*[*of r s r' s'*] *rs rs'* **by** *blast*
next
assume *hrel*: $(r,\ s) .=_{rel} (r',\ s')$
have *class-of*$_{rel}$ $(r,\ s) =$ *class-of*$_{rel}$ $(r',\ s')$
using *equiv-obj-rng-of-frac assms hrel* **by** (*rule elem-eq-class*)
then show $(r \mid_{rel} s) = (r' \mid_{rel} s')$
by (*simp add*: *class-of-to-rel*)
qed

lemma *fraction-zero-rep* [*simp*]:
assumes $s \in S$
shows $(\mathbf{0} \mid_{rel} s) = \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$
using *assms* **by** (*rule class-of-zero-rng-of-frac*)

lemma *fraction-surj*:
assumes $x \in$ *carrier rec-rng-of-frac*
shows $\exists\, r \in$ *carrier* $R.\ \exists\, s \in S.\ x = (r \mid_{rel} s)$
using *assms*
unfolding *rec-rng-of-frac-def set-eq-class-of-rng-of-frac-def rel-def*
by *auto*

lemma *fraction-rescale*:
assumes $(r,\ s) \in$ *carrier rel*
and $s' \in S$
shows $(r \mid_{rel} s) = (s' \otimes r \mid_{rel} s' \otimes s)$
using *assms* **by** (*rule simp-in-frac*)

lemma *fraction-mult-rep*:
assumes *rs*: $(r,\ s) \in$ *carrier rel*
and *r's'*: $(r',\ s') \in$ *carrier rel*
shows $(r \mid_{rel} s) \otimes_{rec\text{-}rng\text{-}of\text{-}frac} (r' \mid_{rel} s') =$
$(r \otimes_R r' \mid_{rel} s \otimes_R s')$
proof −
have *hfund*:
$(r \mid_{rel} s) \otimes_{rec\text{-}monoid\text{-}rng\text{-}of\text{-}frac} (r' \mid_{rel} s') =$
$(r \otimes_R r' \mid_{rel} s \otimes_R s')$
using *mult-rng-of-frac-fundamental-lemma*[*OF rs r's'*]
by *simp*
have *hmonoid-mult*:
$(r \mid_{rel} s) \otimes_{rec\text{-}monoid\text{-}rng\text{-}of\text{-}frac} (r' \mid_{rel} s') =$
mult-rng-of-frac $(r \mid_{rel} s)$ $(r' \mid_{rel} s')$
by (*simp add*: *rec-monoid-rng-of-frac-def*)
have $(r \mid_{rel} s) \otimes_{rec\text{-}rng\text{-}of\text{-}frac} (r' \mid_{rel} s') =$

mult-rng-of-frac $(r \mid_{rel} s)$ $(r' \mid_{rel} s')$
by (*simp only*: *rec-rng-of-frac-def ring-record-simps*)
also have $\ldots = (r \mid_{rel} s) \otimes_{\textit{rec-monoid-rng-of-frac}} (r' \mid_{rel} s')$
using *hmonoid-mult* **by** *simp*
also have $\ldots = (r \otimes_R r' \mid_{rel} s \otimes_R s')$
by (*rule hfund*)
finally show *?thesis* **.**
qed

lemma *map-mul-fraction*:
assumes *a-in*: $a \in$ *carrier* R
and *rs*: $(r, s) \in$ *carrier rel*
shows *rng-to-rng-of-frac* $a \otimes_{\textit{rec-rng-of-frac}} (r \mid_{rel} s) = (a \otimes_R r \mid_{rel} s)$
proof −
have *one-in*: $\mathbf{1} \in S$
by (*rule one-closed*)
have *a-rel*: $(a, \mathbf{1}) \in$ *carrier rel*
using *a-in one-in* **by** (*simp add*: *rel-def*)
have *s-in*: $s \in S$
using *rs* **by** (*simp add*: *rel-def*)
have *s-carrier*: $s \in$ *carrier* R
using *subset s-in* **by** *blast*
have *one-s*: $\mathbf{1} \otimes_R s = s$
using *s-carrier* **by** *simp*
have *rng-to-rng-of-frac* $a \otimes_{\textit{rec-rng-of-frac}} (r \mid_{rel} s) = (a \mid_{rel} \mathbf{1}) \otimes_{\textit{rec-rng-of-frac}} (r \mid_{rel} s)$
by (*simp add*: *rng-to-rng-of-frac-def*)
also have $\ldots = (a \otimes_R r \mid_{rel} \mathbf{1} \otimes_R s)$
by (*rule fraction-mult-rep*[*OF a-rel rs*])
also have $\ldots = (a \otimes_R r \mid_{rel} s)$
using *one-s* **by** *simp*
finally show *?thesis* **.**
qed

lemma *fraction-mul-map*:
assumes *rs*: $(r, s) \in$ *carrier rel*
and *a-in*: $a \in$ *carrier* R
shows $(r \mid_{rel} s) \otimes_{\textit{rec-rng-of-frac}} \textit{rng-to-rng-of-frac}\ a = (r \otimes_R a \mid_{rel} s)$
proof −
have *one-in*: $\mathbf{1} \in S$
by (*rule one-closed*)
have *a-rel*: $(a, \mathbf{1}) \in$ *carrier rel*
using *a-in one-in* **by** (*simp add*: *rel-def*)
have *s-in*: $s \in S$
using *rs* **by** (*simp add*: *rel-def*)
have *s-carrier*: $s \in$ *carrier* R
using *subset s-in* **by** *blast*
have *s-one*: $s \otimes_R \mathbf{1} = s$
using *s-carrier* **by** *simp*

have $(r \mid_{rel} s) \otimes_{rec\text{-}rng\text{-}of\text{-}frac} rng\text{-}to\text{-}rng\text{-}of\text{-}frac\ a = (r \mid_{rel} s) \otimes_{rec\text{-}rng\text{-}of\text{-}frac} (a \mid_{rel} \mathbf{1})$
by (*simp add*: *rng-to-rng-of-frac-def*)
also have $\ldots = (r \otimes_R a \mid_{rel} s \otimes_R \mathbf{1})$
by (*rule fraction-mult-rep*[*OF rs a-rel*])
also have $\ldots = (r \otimes_R a \mid_{rel} s)$
using *s-one* **by** *simp*
finally show *?thesis* **.**
qed

lemma *fraction-eq-iff-cross-multiply*:
assumes *rs*: $(r, s) \in carrier\ rel$
and *rs'*: $(r', s') \in carrier\ rel$
and *zero-notin*: $\mathbf{0} \notin S$
and *no-zero-div*: $\forall a \in carrier\ R.\ \forall b \in carrier\ R.\ a \otimes b = \mathbf{0} \longrightarrow a = \mathbf{0} \vee b = \mathbf{0}$
shows $(r \mid_{rel} s) = (r' \mid_{rel} s') \longleftrightarrow s' \otimes_R r = s \otimes_R r'$
proof (*intro iffI*)
from *rs rs'* **have** *r-in*: $r \in carrier\ R$ **and** *s-in*: $s \in S$
and *r'-in*: $r' \in carrier\ R$ **and** *s'-in*: $s' \in S$
by (*simp-all add*: *rel-def*)
have *s-carrier*: $s \in carrier\ R$ **and** *s'-carrier*: $s' \in carrier\ R$
using *s-in s'-in subset rev-subsetD* **by** *auto*
have *lhs-in*: $s' \otimes_R r \in carrier\ R$
using *s'-carrier r-in* **by** *auto*
have *rhs-in*: $s \otimes_R r' \in carrier\ R$
using *s-carrier r'-in* **by** *auto*
assume *eq-frac*: $(r \mid_{rel} s) = (r' \mid_{rel} s')$
have *hrel*: $(r, s) .=_{rel} (r', s')$
using *fraction-eq-iff-rel*[*OF rs rs'*] *eq-frac* **by** *blast*
then obtain *t* **where** *t-in*: $t \in S$ **and** *t-zero*: $t \otimes ((s' \otimes_R r) \ominus_R (s \otimes_R r')) = \mathbf{0}$
unfolding *rel-def* **by** *auto*
have *t-carrier*: $t \in carrier\ R$
using *t-in subset rev-subsetD* **by** *auto*
have *diff-in*: $(s' \otimes_R r) \ominus_R (s \otimes_R r') \in carrier\ R$
using *lhs-in rhs-in* **by** *auto*
have $t = \mathbf{0} \vee (s' \otimes_R r) \ominus_R (s \otimes_R r') = \mathbf{0}$
using *no-zero-div t-carrier diff-in t-zero* **by** *blast*
moreover have $t \neq \mathbf{0}$
using *t-in zero-notin* **by** *auto*
ultimately have $(s' \otimes_R r) \ominus_R (s \otimes_R r') = \mathbf{0}$
by *auto*
then have $((s' \otimes_R r) \ominus_R (s \otimes_R r')) \oplus_R (s \otimes_R r') = \mathbf{0} \oplus_R (s \otimes_R r')$
by *simp*
then have $(s' \otimes_R r) \oplus_R ((\ominus_R (s \otimes_R r')) \oplus_R (s \otimes_R r')) = s \otimes_R r'$
using *lhs-in rhs-in* **by** (*simp add*: *a-minus-def a-assoc*)

```
    have inv-cancel: (⊖_R (s ⊗_R r')) ⊕_R (s ⊗_R r') = 𝟬
      using rhs-in by (rule l-neg)
    then have (s' ⊗_R r) ⊕_R ((⊖_R (s ⊗_R r')) ⊕_R (s ⊗_R r')) =
        (s' ⊗_R r) ⊕_R 𝟬
      by simp
    then have (s' ⊗_R r) ⊕_R 𝟬 = s ⊗_R r'
      using ‹(s' ⊗_R r) ⊕_R ((⊖_R (s ⊗_R r')) ⊕_R (s ⊗_R r')) =
        s ⊗_R r'›
      by simp
  then show s' ⊗_R r = s ⊗_R r'
    using lhs-in by (simp add: r-zero)
next
  assume cross-mul: s' ⊗_R r = s ⊗_R r'
  from rs rs' have r-in': r ∈ carrier R and s'-in': s' ∈ S
    by (simp-all add: rel-def)
  have lhs-in': s' ⊗_R r ∈ carrier R
  proof −
    have s' ∈ carrier R
      using s'-in' subset rev-subsetD by auto
    then show ?thesis
      using r-in' by auto
  qed
  have diff-zero: (s' ⊗_R r) ⊖_R (s ⊗_R r') = 𝟬
  proof −
    have (s' ⊗_R r) ⊖_R (s ⊗_R r') = (s' ⊗_R r) ⊖_R (s' ⊗_R r)
      using cross-mul by simp
    also have ... = (s' ⊗_R r) ⊕_R (⊖_R (s' ⊗_R r))
      by (simp add: a-minus-def)
    also have ... = 𝟬
      using lhs-in' by (rule r-neg)
    finally show ?thesis .
  qed
  have (r, s) .=_rel (r', s')
  proof −
    have 𝟭 ∈ S
      by (rule one-closed)
    moreover have 𝟭 ⊗ ((s' ⊗_R r) ⊖_R (s ⊗_R r')) = 𝟬
      using diff-zero by simp
    ultimately show ?thesis
      unfolding rel-def using rs rs' by auto
  qed
  then show (r |_rel s) = (r' |_rel s')
    using fraction-eq-iff-rel[OF rs rs'] by blast
qed

lemma fraction-eq-zero-iff:
  assumes rs: (r, s) ∈ carrier rel
    and zero-notin: 𝟬 ∉ S
    and no-zero-div: ∀ a ∈ carrier R. ∀ b ∈ carrier R. a ⊗ b = 𝟬 ⟶ a = 𝟬 ∨ b =
```

0
shows $(r \mid_{rel} s) = \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac} \longleftrightarrow r = \mathbf{0}$
proof
from *rs* **have** *s-in*: $s \in S$
by (*simp add*: *rel-def*)
have *s-carrier*: $s \in carrier\ R$
using *subset s-in* **by** *blast*
from *rs* **have** *r-in*: $r \in carrier\ R$
by (*simp add*: *rel-def*)
have *zero-rel*: $(\mathbf{0}, \mathbf{1}) \in carrier\ rel$
by (*simp add*: *rel-def one-closed*)
assume *hzero*: $(r \mid_{rel} s) = \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$
have $(r \mid_{rel} s) = (\mathbf{0} \mid_{rel} \mathbf{1})$
using *hzero* **by** (*simp add*: *class-of-zero-rng-of-frac*[*OF one-closed*])
then have $\mathbf{1} \otimes_R r = s \otimes_R \mathbf{0}$
using *fraction-eq-iff-cross-multiply*[*OF rs zero-rel zero-notin no-zero-div*]
by *simp*
then have $r = s \otimes_R \mathbf{0}$
using *r-in* **by** *simp*
also have $\ldots = \mathbf{0}$
using *s-carrier* **by** *simp*
finally show $r = \mathbf{0}$ **.**
next
assume $r = \mathbf{0}$
then have *r-zero*: $r = \mathbf{0}$ **.**
have *s-in*: $s \in S$
using *rs* **by** (*simp add*: *rel-def*)
from *r-zero s-in* **show** $(r \mid_{rel} s) = \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$
by *simp*
qed

lemma *map-eq-zero-iff*:
assumes *a-in*: $a \in carrier\ R$
and *zero-notin*: $\mathbf{0} \notin S$
and *no-zero-div*: $\forall\, a' \in carrier\ R.\ \forall\, b' \in carrier\ R.\ a' \otimes b' = \mathbf{0} \longrightarrow a' = \mathbf{0} \vee b' = \mathbf{0}$
shows *rng-to-rng-of-frac* $a = \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac} \longleftrightarrow a = \mathbf{0}$
proof −
have *a-rel*: $(a, \mathbf{1}) \in carrier\ rel$
using *a-in one-closed* **by** (*simp add*: *rel-def*)
show *?thesis*
using *fraction-eq-zero-iff*[*OF a-rel zero-notin no-zero-div*]
by (*simp add*: *rng-to-rng-of-frac-def*)
qed

lemma *dvd-map-iff*:
assumes *a-in*: $a \in carrier\ R$
and *b-in*: $b \in carrier\ R$
and *zero-notin*: $\mathbf{0} \notin S$

and *no-zero-div*: $\forall\, a' \in$ *carrier R*. $\forall\, b' \in$ *carrier R*. $a' \otimes b' = \mathbf{0} \longrightarrow a' = \mathbf{0} \vee b' = \mathbf{0}$
shows *rng-to-rng-of-frac a divides*$_{\textit{rec-rng-of-frac}}$ *rng-to-rng-of-frac b* $\longleftrightarrow$ $(\exists\, s \in S.$ *a divides*$_R$ $(s \otimes_R b))$
proof
assume *hdiv*: *rng-to-rng-of-frac a divides*$_{\textit{rec-rng-of-frac}}$ *rng-to-rng-of-frac b*
then obtain *x* **where** *x-mem*: $x \in$ *carrier rec-rng-of-frac*
and *hx*: *rng-to-rng-of-frac b* = *rng-to-rng-of-frac a* $\otimes_{\textit{rec-rng-of-frac}}$ *x*
unfolding *factor-def* **by** *blast*
from *fraction-surj*[*OF x-mem*] **obtain** *c* **where** *c-in*: $c \in$ *carrier R*
and *hs*: $\exists\, s \in S.\ x = (c \mid_{rel} s)$
by *blast*
from *hs* **obtain** *s* **where** *s-in*: $s \in S$ **and** *x-def*: $x = (c \mid_{rel} s)$
by *blast*
have *b-rel*: $(b, \mathbf{1}) \in$ *carrier rel*
using *b-in one-closed* **by** (*simp add*: *rel-def*)
have *cs-rel*: $(c, s) \in$ *carrier rel*
using *c-in s-in* **by** (*simp add*: *rel-def*)
have *ac-rel*: $(a \otimes_R c, s) \in$ *carrier rel*
using *a-in c-in s-in* **by** (*simp add*: *rel-def*)
have $(b \mid_{rel} \mathbf{1}) = (a \otimes_R c \mid_{rel} s)$
proof −
have *h1*: *rng-to-rng-of-frac b* = *rng-to-rng-of-frac a* $\otimes_{\textit{rec-rng-of-frac}}$ $(c \mid_{rel} s)$
using *hx x-def* **by** *simp*
have *h2*: *rng-to-rng-of-frac a* $\otimes_{\textit{rec-rng-of-frac}}$ $(c \mid_{rel} s) = (a \otimes_R c \mid_{rel} s)$
using *map-mul-fraction*[*OF a-in cs-rel*] **by** *simp*
from *h1 h2* **show** *?thesis*
by (*simp add*: *rng-to-rng-of-frac-def*)
qed
then have $s \otimes_R b = \mathbf{1} \otimes_R (a \otimes_R c)$
using *fraction-eq-iff-cross-multiply*[*OF b-rel ac-rel zero-notin no-zero-div*]
by *simp*
then have $s \otimes_R b = a \otimes_R c$
using *a-in c-in* **by** *simp*
then have *a divides*$_R$ $(s \otimes_R b)$
unfolding *factor-def* **using** *c-in* **by** *blast*
then show $\exists\, s \in S.$ *a divides*$_R$ $(s \otimes_R b)$
using *s-in* **by** *blast*
next
assume *hdiv*: $\exists\, s \in S.$ *a divides*$_R$ $(s \otimes_R b)$
then obtain *s* **where** *s-in*: $s \in S$ **and** *hsab*: *a divides*$_R$ $(s \otimes_R b)$
by *blast*
then obtain *c* **where** *c-in*: $c \in$ *carrier R* **and** *hc*: $s \otimes_R b = a \otimes_R c$
unfolding *factor-def* **by** *blast*
have *cs-rel*: $(c, s) \in$ *carrier rel*
using *c-in s-in* **by** (*simp add*: *rel-def*)
have *ac-rel*: $(a \otimes_R c, s) \in$ *carrier rel*
using *a-in c-in s-in* **by** (*simp add*: *rel-def*)
have *b-rel*: $(b, \mathbf{1}) \in$ *carrier rel*

```
    using b-in one-closed by (simp add: rel-def)
  have eq-frac: (b |_rel 1) = (a ⊗_R c |_rel s)
  proof −
    have s ⊗_R b = 1 ⊗_R (a ⊗_R c)
      using hc a-in c-in by simp
    then show ?thesis
      using fraction-eq-iff-cross-multiply[OF b-rel ac-rel zero-notin no-zero-div]
      by blast
  qed
  have frac-mem: (c |_rel s) ∈ carrier rec-rng-of-frac
    using cs-rel unfolding rec-rng-of-frac-def set-eq-class-of-rng-of-frac-def
    by auto
  have rng-to-rng-of-frac b = rng-to-rng-of-frac a ⊗_rec-rng-of-frac (c |_rel s)
  proof −
    have h1: rng-to-rng-of-frac b = (b |_rel 1)
      by (simp add: rng-to-rng-of-frac-def)
    have h2: (b |_rel 1) = (a ⊗_R c |_rel s)
      using eq-frac by simp
    have h3: rng-to-rng-of-frac a ⊗_rec-rng-of-frac (c |_rel s) = (a ⊗_R c |_rel s)
      using map-mul-fraction[OF a-in cs-rel] by simp
    from h1 h2 h3 show ?thesis
      by simp
  qed
  then show rng-to-rng-of-frac a divides_rec-rng-of-frac rng-to-rng-of-frac b
    unfolding factor-def using frac-mem by blast
qed

lemma image-submonoid-is-unit:
  assumes x ∈ rng-to-rng-of-frac ‘ S
  shows x ∈ Units rec-rng-of-frac
  using assms by (rule Im-rng-to-rng-of-frac-unit)

lemma map-submonoid-elem-is-unit:
  assumes s ∈ S
  shows rng-to-rng-of-frac s ∈ Units rec-rng-of-frac
  using assms image-submonoid-is-unit by blast

lemma map-unit-is-unit:
  assumes u-unit: u ∈ Units R
  shows rng-to-rng-of-frac u ∈ Units rec-rng-of-frac
proof −
  have u-in: u ∈ carrier R
    using u-unit by blast
  have inv-u-in: inv u ∈ carrier R
    using u-unit by blast
  have map-u-in: rng-to-rng-of-frac u ∈ carrier rec-rng-of-frac
    using ring-hom-closed[OF rng-to-rng-of-frac-is-ring-hom u-in] .
  have map-inv-u-in: rng-to-rng-of-frac (inv u) ∈ carrier rec-rng-of-frac
    using ring-hom-closed[OF rng-to-rng-of-frac-is-ring-hom inv-u-in] .
```

```
  have left-inv:
     rng-to-rng-of-frac u ⊗_rec-rng-of-frac rng-to-rng-of-frac (inv u) =
       1_rec-rng-of-frac
  proof −
    have rng-to-rng-of-frac u ⊗_rec-rng-of-frac rng-to-rng-of-frac (inv u) =
        rng-to-rng-of-frac (u ⊗_R inv u)
      using ring-hom-mult[OF rng-to-rng-of-frac-is-ring-hom u-in inv-u-in]
      by simp
    also have ... = rng-to-rng-of-frac 1
      using u-unit by simp
    also have ... = 1_rec-rng-of-frac
      using ring-hom-one[OF rng-to-rng-of-frac-is-ring-hom] .
    finally show ?thesis .
  qed
  have right-inv:
     rng-to-rng-of-frac (inv u) ⊗_rec-rng-of-frac rng-to-rng-of-frac u =
       1_rec-rng-of-frac
  proof −
    have rng-to-rng-of-frac (inv u) ⊗_rec-rng-of-frac rng-to-rng-of-frac u =
        rng-to-rng-of-frac (inv u ⊗_R u)
      using ring-hom-mult[OF rng-to-rng-of-frac-is-ring-hom inv-u-in u-in]
      by simp
    also have ... = rng-to-rng-of-frac 1
      using u-unit by simp
    also have ... = 1_rec-rng-of-frac
      using ring-hom-one[OF rng-to-rng-of-frac-is-ring-hom] .
    finally show ?thesis .
  qed
  show ?thesis
    unfolding Units-def
    using map-u-in map-inv-u-in left-inv right-inv by blast
qed

lemma fraction-unit-numerator-is-unit:
  assumes u-unit: u ∈ Units R
    and s-in: s ∈ S
  shows (u |_rel s) ∈ Units rec-rng-of-frac
proof −
  have u-in: u ∈ carrier R
    using Units-closed[OF u-unit] .
  have s-carrier: s ∈ carrier R
    using subset s-in by blast
  have u-rel: (u, 1) ∈ carrier rel
    using u-in one-closed by (simp add: rel-def)
  have one-rel: (1, s) ∈ carrier rel
    using s-in by (simp add: rel-def)
  have frac-eq′:
     rng-to-rng-of-frac u ⊗_rec-rng-of-frac (1 |_rel s) =
       (u |_rel s)
```

proof −
have *rng-to-rng-of-frac u* $\otimes_{\textit{rec-rng-of-frac}}$ $(\mathbf{1} \mid_{\textit{rel}} s) =$
$(u \mid_{\textit{rel}} \mathbf{1}) \otimes_{\textit{rec-rng-of-frac}} (\mathbf{1} \mid_{\textit{rel}} s)$
by (*simp add*: *rng-to-rng-of-frac-def*)
also have $\ldots = (u \otimes_R \mathbf{1} \mid_{\textit{rel}} \mathbf{1} \otimes_R s)$
by (*rule fraction-mult-rep*[*OF u-rel one-rel*])
also have $\ldots = (u \mid_{\textit{rel}} s)$
using *u-in s-carrier* **by** *simp*
finally show *?thesis* **.**
qed
have *frac-eq*:
$(u \mid_{\textit{rel}} s) =$
rng-to-rng-of-frac u $\otimes_{\textit{rec-rng-of-frac}}$ $(\mathbf{1} \mid_{\textit{rel}} s)$
using *frac-eq′* **by** *simp*
have *map-unit*: *rng-to-rng-of-frac u* $\in$ *Units rec-rng-of-frac*
using *u-unit* **by** (*rule map-unit-is-unit*)
have *denom-unit*: $(\mathbf{1} \mid_{\textit{rel}} s) \in$ *Units rec-rng-of-frac*
proof −
have *s-carrier*: $s \in$ *carrier R*
using *subset s-in* **by** *blast*
have *s-rel*: $(s, \mathbf{1}) \in$ *carrier rel*
using *s-carrier one-closed* **by** (*simp add*: *rel-def*)
have *left-inv*:
$(\mathbf{1} \mid_{\textit{rel}} s) \otimes_{\textit{rec-rng-of-frac}}$ *rng-to-rng-of-frac s* =
$\mathbf{1}_{\textit{rec-rng-of-frac}}$
proof −
have $(\mathbf{1} \mid_{\textit{rel}} s) \otimes_{\textit{rec-rng-of-frac}}$ *rng-to-rng-of-frac s* =
$(\mathbf{1} \mid_{\textit{rel}} s) \otimes_{\textit{rec-rng-of-frac}} (s \mid_{\textit{rel}} \mathbf{1})$
by (*simp add*: *rng-to-rng-of-frac-def*)
also have $\ldots = (\mathbf{1} \otimes_R s \mid_{\textit{rel}} s \otimes_R \mathbf{1})$
by (*rule fraction-mult-rep*[*OF one-rel s-rel*])
also have $\ldots = (\mathbf{1} \mid_{\textit{rel}} \mathbf{1})$
proof −
have *one-rel′*: $(\mathbf{1}, \mathbf{1}) \in$ *carrier rel*
using *one-closed* **by** (*simp add*: *rel-def*)
have $(\mathbf{1} \mid_{\textit{rel}} \mathbf{1}) = (s \otimes_R \mathbf{1} \mid_{\textit{rel}} s \otimes_R \mathbf{1})$
by (*rule fraction-rescale*[*OF one-rel′ s-in*])
then show *?thesis*
using *s-carrier* **by** *simp*
qed
also have $\ldots = \mathbf{1}_{\textit{rec-rng-of-frac}}$
by (*simp only*: *rec-rng-of-frac-def ring-record-simps*)
finally show *?thesis* **.**
qed
have *right-inv*:
rng-to-rng-of-frac s $\otimes_{\textit{rec-rng-of-frac}}$ $(\mathbf{1} \mid_{\textit{rel}} s) =$
$\mathbf{1}_{\textit{rec-rng-of-frac}}$
proof −
have *rng-to-rng-of-frac s* $\otimes_{\textit{rec-rng-of-frac}}$ $(\mathbf{1} \mid_{\textit{rel}} s) =$

```
        (s |_rel 1) ⊗_rec-rng-of-frac (1 |_rel s)
      by (simp add: rng-to-rng-of-frac-def)
    also have ... = (s ⊗_R 1 |_rel 1 ⊗_R s)
      by (rule fraction-mult-rep[OF s-rel one-rel])
    also have ... = (1 |_rel 1)
    proof −
      have one-rel': (1, 1) ∈ carrier rel
        using one-closed by (simp add: rel-def)
      have (1 |_rel 1) = (s ⊗_R 1 |_rel s ⊗_R 1)
        by (rule fraction-rescale[OF one-rel' s-in])
      then show ?thesis
        using s-carrier by simp
    qed
    also have ... = 1_rec-rng-of-frac
      by (simp only: rec-rng-of-frac-def ring-record-simps)
    finally show ?thesis .
  qed
  have frac-in: (1 |_rel s) ∈ carrier rec-rng-of-frac
  proof −
    have one-s-rel: (1, s) ∈ carrier rel
      using s-in by (simp add: rel-def)
    show ?thesis
      using one-s-rel
      unfolding rec-rng-of-frac-def set-eq-class-of-rng-of-frac-def
      by auto
  qed
  have s-carrier: s ∈ carrier R
    using s-in subset rev-subsetD by blast
  have map-s-in: rng-to-rng-of-frac s ∈ carrier rec-rng-of-frac
    using ring-hom-closed[OF rng-to-rng-of-frac-is-ring-hom s-carrier] .
  show ?thesis
    unfolding Units-def using frac-in map-s-in left-inv right-inv by blast
 qed
 show ?thesis
 proof −
  have rng-to-rng-of-frac u ⊗_rec-rng-of-frac (1 |_rel s) ∈ Units rec-rng-of-frac
   by (rule monoid.Units-m-closed[OF ring.is-monoid[OF rng-rng-of-frac] map-unit
denom-unit])
  then show ?thesis
    using frac-eq by simp
 qed
qed

lemma map-inj-on:
 assumes 0 ∉ S
  and ∀ a ∈ carrier R. ∀ b ∈ carrier R. a ⊗ b = 0 ⟶ a = 0 ∨ b = 0
 shows inj-on rng-to-rng-of-frac (carrier R)
proof −
 have a-kernel R rec-rng-of-frac rng-to-rng-of-frac = {0}
```

using *assms* **by** (*rule rng-to-rng-of-frac-without-zero-div-is-inj*)
then show *?thesis*
using *ring-hom-ring.trivial-ker-imp-inj*[
OF ring-hom-ringI2[*OF ring-axioms rng-rng-of-frac rng-to-rng-of-frac-is-ring-hom*]
]
by *blast*
qed

end

end
theory *Nagata-Lemmas*
imports *Localization-Interface*
begin

4 Record-based Nagata descent lemmas

definition *ring-avoids* ::
(*'a*, *'b*) *ring-scheme* ⇒ *'a set* ⇒ *'a* ⇒ *bool*
where
ring-avoids R S p ⟷ (∀ *s*∈*S*. ¬ *p divides*$_R$ *s*)

definition *ring-prime-generated* ::
(*'a*, *'b*) *ring-scheme* ⇒ *'a set* ⇒ *bool*
where
ring-prime-generated R S ⟷
(∀ *s*∈*S*. ∃ *fs*.
set fs ⊆ *S* ∧
(∀ *q*∈*set fs*. *ring-prime*$_R$ *q*) ∧
foldr (⊗$_R$) *fs* **1**$_R$ = *s*)

lemma *ring-prime-generatedI*:
assumes ⋀*s*. *s* ∈ *S* ⟹ ∃ *fs*.
set fs ⊆ *S* ∧
(∀ *q*∈*set fs*. *ring-prime*$_R$ *q*) ∧
foldr (⊗$_R$) *fs* **1**$_R$ = *s*
shows *ring-prime-generated R S*
using *assms* **unfolding** *ring-prime-generated-def* **by** *blast*

lemma *ring-prime-generatedE*:
assumes *ring-prime-generated R S s* ∈ *S*
obtains *fs* **where**
set fs ⊆ *S*
∀ *q*∈*set fs*. *ring-prime*$_R$ *q*
foldr (⊗$_R$) *fs* **1**$_R$ = *s*
using *assms* **unfolding** *ring-prime-generated-def* **by** *blast*

definition *ring-powers-set* ::
(*'a*, *'b*) *ring-scheme* ⇒ *'a* ⇒ *'a set*

where
ring-powers-set R p = {*x*. ∃ *n*::*nat*. *x* = *p* [^]$_R$ *n*}

inductive-set *ring-mult-submonoid-closure* ::
(′*a*, ′*b*) *ring-scheme* ⇒ ′*a set* ⇒ ′*a set*
for *R* **and** *A*
where
one-closed: **1**$_R$ ∈ *ring-mult-submonoid-closure R A*
| *generator*: *a* ∈ *A* ⟹ *a* ∈ *ring-mult-submonoid-closure R A*
| *mult-closed*:
a ∈ *ring-mult-submonoid-closure R A* ⟹
b ∈ *ring-mult-submonoid-closure R A* ⟹
a ⊗$_R$ *b* ∈ *ring-mult-submonoid-closure R A*

lemma *ring-mult-submonoid-closure-subset*:
assumes *ring-R*: *ring R*
and *A-sub*: *A* ⊆ *carrier R*
shows *ring-mult-submonoid-closure R A* ⊆ *carrier R*
proof −
interpret *R*: *ring R*
by (*rule ring-R*)
show *?thesis*
proof
fix *x*
assume *x-in*: *x* ∈ *ring-mult-submonoid-closure R A*
then show *x* ∈ *carrier R*
by *induction* (*use A-sub* **in** *auto*)
qed
qed

lemma *ring-mult-submonoid-closure-submonoid*:
assumes *ring-R*: *ring R*
and *A-sub*: *A* ⊆ *carrier R*
shows *submonoid R* (*ring-mult-submonoid-closure R A*)
proof −
interpret *R*: *ring R*
by (*rule ring-R*)
show *?thesis*
proof
show *ring-mult-submonoid-closure R A* ⊆ *carrier R*
by (*rule ring-mult-submonoid-closure-subset*[*OF ring-R A-sub*])
qed (*auto simp*: *ring-mult-submonoid-closure.one-closed ring-mult-submonoid-closure.mult-closed*)
qed

lemma *foldr-mult-right*:
assumes *ring-R*: *ring R*
and *xs-sub*: *set xs* ⊆ *carrier R*
and *y-in*: *y* ∈ *carrier R*
shows *foldr* (⊗$_R$) *xs y* =

```
    foldr (⊗_R) xs 1_R ⊗_R y
proof −
  interpret R: ring R
    by (rule ring-R)
  show ?thesis
    using xs-sub y-in
  proof (induction xs)
    case Nil
    then show ?case
      by simp
  next
    case (Cons x xs)
    have x-in: x ∈ carrier R
      using Cons.prems(1) by simp
    have xs-sub': set xs ⊆ carrier R
      using Cons.prems(1) by simp
    have prod-in: foldr (⊗_R) xs 1_R ∈ carrier R
      using xs-sub'
    proof (induction xs)
      case Nil
      then show ?case by simp
    next
      case (Cons z zs)
      then show ?case by simp
    qed
    show ?case
      using Cons.IH[OF xs-sub' Cons.prems(2)] x-in prod-in Cons.prems(2)
      by (simp add: R.m-assoc)
  qed
qed

lemma ring-powers-submonoid:
  assumes ring-R: ring R
    and p-in: p ∈ carrier R
  shows submonoid R (ring-powers-set R p)
proof −
  interpret R: ring R
    by (rule ring-R)
  show ?thesis
  proof (unfold-locales)
    show ring-powers-set R p ⊆ carrier R
      using p-in unfolding ring-powers-set-def by auto
    show ⋀x y. x ∈ ring-powers-set R p ⟹ y ∈ ring-powers-set R p ⟹ x ⊗_R y
∈ ring-powers-set R p
    proof −
      fix x y
      assume x-in: x ∈ ring-powers-set R p and y-in: y ∈ ring-powers-set R p
      then obtain m n :: nat where x-def: x = p [^]_R m and y-def: y = p [^]_R n
        unfolding ring-powers-set-def by blast
```

```
    show x ⊗_R y ∈ ring-powers-set R p
      using R.nat-pow-mult p-in ring-powers-set-def x-def y-def by fastforce
   qed
   show 1_R ∈ ring-powers-set R p
   proof −
    have 1_R = p [^]_R (0::nat)
      using p-in by simp
    then show ?thesis
      unfolding ring-powers-set-def by blast
   qed
  qed
qed

lemma ring-prime-generated-powers-set:
  assumes ring-R: ring R
    and p-in: p ∈ carrier R
    and hp: ring-prime_R p
  shows ring-prime-generated R (ring-powers-set R p)
proof (rule ring-prime-generatedI)
  interpret R: ring R
    by (rule ring-R)
  fix s
  assume s-in: s ∈ ring-powers-set R p
  then obtain n :: nat where s-def: s = p [^]_R n
    unfolding ring-powers-set-def by blast
  show ∃fs.
      set fs ⊆ ring-powers-set R p ∧
      (∀q∈set fs. ring-prime_R q) ∧
      foldr (⊗_R) fs 1_R = s
  proof (intro exI conjI)
    show set (replicate n p) ⊆ ring-powers-set R p
    proof
      fix q
      assume q-in: q ∈ set (replicate n p)
      then have q-eq: q = p
        by simp
      show q ∈ ring-powers-set R p
        by (metis (mono-tags, lifting) R.nat-pow-eone mem-Collect-eq p-in q-eq
            ring-powers-set-def)
    qed
    show (∀q∈set (replicate n p). ring-prime_R q)
      using hp by simp
    show foldr (⊗_R) (replicate n p) 1_R = s
    proof −
      have foldr (⊗_R) (replicate n p) 1_R = p [^]_R n
        using p-in
      proof (induction n)
        case 0
        then show ?case by simp
```

```
    next
      case (Suc n)
      have foldr (⊗_R) (replicate (Suc n) p) 1_R =
          p ⊗_R p [^]_R n
        using Suc.IH p-in by simp
      also have ... = p [^]_R (Suc n)
        by (rule sym[OF R.nat-pow-Suc2[OF p-in]])
      finally show ?case .
    qed
    then show ?thesis
      using s-def by simp
  qed
 qed
qed

lemma ring-prime-generated-mult-submonoid-closure:
  assumes ring-R: ring R
    and A-sub: A ⊆ carrier R
    and hprime: ⋀q. q ∈ A ⟹ ring-prime_R q
  shows ring-prime-generated R (ring-mult-submonoid-closure R A)
proof (rule ring-prime-generatedI)
  interpret R: ring R
    by (rule ring-R)
  have closure-sub: ring-mult-submonoid-closure R A ⊆ carrier R
    by (rule ring-mult-submonoid-closure-subset[OF ring-R A-sub])
  fix s
  assume s-in: s ∈ ring-mult-submonoid-closure R A
  show ∃fs.
      set fs ⊆ ring-mult-submonoid-closure R A ∧
      (∀q∈set fs. ring-prime_R q) ∧
      foldr (⊗_R) fs 1_R = s
    using s-in
  proof induction
    case one-closed
    show ?case
      by (intro exI[of - []]) auto
  next
    case (generator a)
    show ?case
    proof (intro exI[of - [a]] conjI)
      have a-in-closure: a ∈ ring-mult-submonoid-closure R A
        by (rule ring-mult-submonoid-closure.generator[OF generator.hyps])
      show set [a] ⊆ ring-mult-submonoid-closure R A
        using a-in-closure by simp
      show ∀q∈set [a]. ring-prime_R q
        using hprime generator.hyps by simp
      have a-in: a ∈ carrier R
        using A-sub generator.hyps by blast
      show foldr (⊗_R) [a] 1_R = a
```

```
      using a-in by simp
    qed
  next
    case (mult-closed a b)
    from mult-closed.IH(1) obtain fs where
      fs-sub: set fs ⊆ ring-mult-submonoid-closure R A
      and fs-prime: ∀ q∈set fs. ring-prime_R q
      and fs-prod: foldr (⊗_R) fs 1_R = a
      by blast
    from mult-closed.IH(2) obtain gs where
      gs-sub: set gs ⊆ ring-mult-submonoid-closure R A
      and gs-prime: ∀ q∈set gs. ring-prime_R q
      and gs-prod: foldr (⊗_R) gs 1_R = b
      by blast
    have fs-carr: set fs ⊆ carrier R
      using fs-sub closure-sub by blast
    have b-in: b ∈ carrier R
      using mult-closed.hyps(2) closure-sub by blast
    have prod-append:
      foldr (⊗_R) (fs @ gs) 1_R = a ⊗_R b
    proof −
      have foldr (⊗_R) (fs @ gs) 1_R =
        foldr (⊗_R) fs (foldr (⊗_R) gs 1_R)
        by simp
      also have ... = foldr (⊗_R) fs b
        by (simp add: gs-prod)
      also have ... = foldr (⊗_R) fs 1_R ⊗_R b
        by (rule foldr-mult-right[OF ring-R fs-carr b-in])
      also have ... = a ⊗_R b
        by (simp add: fs-prod)
      finally show ?thesis .
    qed
    show ?case
      by (intro exI[of - fs @ gs])
        (use fs-sub fs-prime gs-sub gs-prime prod-append in auto)
  qed
qed

locale nagata-localization = eq-obj-rng-of-frac R S + domain R for R (structure)
and S
begin

lemma no-zero-divisors:
  ∀ a ∈ carrier R. ∀ b ∈ carrier R. a ⊗_R b = 0 ⟶ a = 0 ∨ b = 0
  using integral by blast

lemma multlist-closed:
  assumes xs-sub: set xs ⊆ carrier R
  shows foldr (⊗_R) xs 1_R ∈ carrier R
```

using *xs-sub*
by (*induction xs*) *auto*

lemma *multlist-mem-submonoid*:
assumes *fs-sub*: *set fs* $\subseteq$ *S*
shows *foldr* ($\otimes_R$) *fs* $\mathbf{1}_R \in S$
using *fs-sub*
proof (*induction fs*)
case *Nil*
show *?case*
by *simp*
next
case (*Cons q qs*)
have *q-in*: *q* $\in$ *S*
using *Cons.prems* **by** *simp*
have *qs-in*: *set qs* $\subseteq$ *S*
using *Cons.prems* **by** *simp*
show *?case*
using *q-in Cons.IH*[*OF qs-in*] **by** *simp*
qed

lemma *multlist-nonzero-of-prime-factors*:
assumes *fs-sub*: *set fs* $\subseteq$ *S*
and *hf*: $\forall$ *q*$\in$*set fs*. *ring-prime*$_R$ *q*
shows *foldr* ($\otimes_R$) *fs* $\mathbf{1}_R \neq \mathbf{0}$
using *fs-sub hf*
proof (*induction fs*)
case *Nil*
show *?case*
by *simp*
next
case (*Cons q qs*)
have *q-in*: *q* $\in$ *S*
using *Cons.prems*(*1*) **by** *simp*
have *q-carr*: *q* $\in$ *carrier R*
using *q-in subset rev-subsetD* **by** *blast*
have *q-prime*: *ring-prime*$_R$ *q*
using *Cons.prems*(*2*) **by** *simp*
have *q-nz*: *q* $\neq \mathbf{0}$
using *ring-primeE*(*1*)[*OF q-carr q-prime*] **.**
have *qs-sub*: *set qs* $\subseteq$ *S*
using *Cons.prems*(*1*) **by** *simp*
have *qs-prime*: $\forall$ *r*$\in$*set qs*. *ring-prime*$_R$ *r*
using *Cons.prems*(*2*) **by** *simp*
have *qs-nz*: *foldr* ($\otimes_R$) *qs* $\mathbf{1}_R \neq \mathbf{0}$
using *Cons.IH*[*OF qs-sub qs-prime*] **.**
have *qs-carr*: *set qs* $\subseteq$ *carrier R*
using *qs-sub subset* **by** *blast*
have *prod-carr*: *foldr* ($\otimes_R$) *qs* $\mathbf{1}_R \in$ *carrier R*

```
    by (rule multlist-closed[OF qs-carr])
  show ?case
    by (simp add: integral-iff prod-carr q-carr q-nz qs-nz)
qed

lemma zero-notin-submonoid-of-prime-generated:
  assumes hS: ring-prime-generated R S
  shows 0 ∉ S
proof
  assume zero-in: 0 ∈ S
  obtain fs where
      fs-sub: set fs ⊆ S
      and hf: ∀ q∈set fs. ring-prime_R q
      and hprod: foldr (⊗_R) fs 1_R = 0
    using ring-prime-generatedE[OF hS zero-in] by blast
  have foldr (⊗_R) fs 1_R ≠ 0
    using multlist-nonzero-of-prime-factors[OF fs-sub hf] .
  with hprod show False
    by contradiction
qed

lemma zero-notin-submonoid-of-prime-or-unit:
  assumes hS: ⋀s. s ∈ S ⟹ ring-prime_R s ∨ s ∈ Units R
  shows 0 ∉ S
proof
  assume zero-in: 0 ∈ S
  from hS[OF zero-in] show False
  proof
    assume ring-prime_R 0
    then show False
      by (simp add: ring-prime-def)
  next
    assume zero-unit: 0 ∈ Units R
    have inv-zero-in: inv 0 ∈ carrier R
      using zero-unit by simp
    have (0 ⊗_R inv 0) = 0
      using inv-zero-in by simp
    moreover have (0 ⊗_R inv 0) = 1
      using zero-unit by simp
    ultimately show False
      using one-not-zero by simp
  qed
qed

lemma ring-prime-imp-ring-irreducible:
  assumes p-in: p ∈ carrier R
    and hp: ring-prime_R p
  shows ring-irreducible_R p
proof −
```

```
  from ring-primeE[OF p-in hp] have p-nz: p ≠ 0
    and p-prime-mult: prime (mult-of R) p
    by auto
  have irreducible (mult-of R) p
    using p-prime-mult by (rule mult-of.prime-irreducible)
  have p-nz-in: p ∈ carrier R − {0}
    using p-in p-nz by blast
  from p-nz-in and ‹irreducible (mult-of R) p› show ?thesis
    by (rule ring-irreducibleI')
qed

lemma prime-of-irreducible-of-dvd-mem:
  assumes hS: ⋀s. s ∈ S ⟹ ring-prime_R s ∨ s ∈ Units R
    and p-in: p ∈ carrier R
    and hp: ring-irreducible_R p
    and s-in: s ∈ S
    and p-dvd-s: p divides_R s
  shows ring-prime_R p
proof −
  have p-not-unit: p ∉ Units R
    using ring-irreducibleE(4)[OF p-in hp] .
  from hS[OF s-in] show ?thesis
  proof
    assume hs-prime: ring-prime_R s
    have s-in-carrier: s ∈ carrier R
      using s-in subset rev-subsetD by blast
    have s-irreducible: ring-irreducible_R s
      using s-in-carrier hs-prime by (rule ring-prime-imp-ring-irreducible)
    have p-assoc-s: p ∼_R s
    by (meson divides-irreducible-condition p-dvd-s p-in p-not-unit ring-irreducible-def
        s-irreducible)
    have s-prime: prime R s
      using ring-primeE(3)[OF s-in-carrier hs-prime] .
    have s-nz: s ≠ 0
      using ring-primeE(1)[OF s-in-carrier hs-prime] .
    have p-nz: p ≠ 0
      using ring-irreducibleE(1)[OF p-in hp] .
    have s-assoc-p-mult: s ∼_(mult-of R) p
      using assoc-iff-assoc-mult[OF s-in-carrier p-in] associated-sym[OF p-assoc-s]
by blast
    have s-prime-mult: prime (mult-of R) s
      using prime-eq-prime-mult[OF s-in-carrier] s-prime by blast
    have prime R p
    proof −
      have prime (mult-of R) p
      proof (rule mult-of.prime-cong[OF s-prime-mult s-assoc-p-mult])
        show s ∈ carrier (mult-of R)
          using s-in-carrier s-nz by simp
        show p ∈ carrier (mult-of R)
```

```
        using p-in p-nz by simp
    qed
    then show ?thesis
      using prime-eq-prime-mult[OF p-in] by blast
  qed
  then have p-prime: prime R p .
  from p-nz p-prime show ?thesis
    by (rule ring-primeI)
 next
  assume hs-unit: s ∈ Units R
  have p ∈ Units R
    using divides-unit[OF p-dvd-s p-in hs-unit] .
  with p-not-unit show ?thesis
    by contradiction
 qed
qed

lemma prime-of-irreducible-of-dvd-prime-factors:
 assumes fs-sub: set fs ⊆ S
  and hf: ∀ q∈set fs. ring-prime_R q
  and p-in: p ∈ carrier R
  and hp: ring-irreducible_R p
  and hdiv: p divides_R foldr (⊗_R) fs 1_R
 shows ring-prime_R p
proof −
 have hmain:
    ⋀gs. set gs ⊆ S ⟹
     (∀ q∈set gs. ring-prime_R q) ⟹
     p divides_R foldr (⊗_R) gs 1_R ⟹
     ring-prime_R p
 proof −
  fix gs
  show set gs ⊆ S ⟹
     (∀ q∈set gs. ring-prime_R q) ⟹
     p divides_R foldr (⊗_R) gs 1_R ⟹
     ring-prime_R p
  proof (induction gs)
   case Nil
   have p-dvd-one: p divides_R 1_R
     using Nil.prems(3) by simp
   have p ∈ Units R
     by (rule divides-unit[OF p-dvd-one p-in Units-one-closed])
   with ring-irreducibleE(4)[OF p-in hp] show ?case
     by contradiction
  next
   case (Cons q qs)
   have q-in: q ∈ S
     using Cons.prems(1) by simp
   have q-carr: q ∈ carrier R
```

```
    using q-in subset rev-subsetD by blast
  have q-ring-prime: ring-prime_R q
    using Cons.prems(2) by simp
  have q-prime: prime R q
    using ring-primeE(3)[OF q-carr q-ring-prime] .
  have q-nz: q ≠ 0
    using ring-primeE(1)[OF q-carr q-ring-prime] .
  have q-ring-irred: ring-irreducible_R q
    using ring-prime-imp-ring-irreducible[OF q-carr q-ring-prime] .
  have q-not-unit: q ∉ Units R
    using ring-irreducibleE(4)[OF q-carr q-ring-irred] .
  have qs-sub: set qs ⊆ S
    using Cons.prems(1) by simp
  have qs-prime: ∀ r∈set qs. ring-prime_R r
    using Cons.prems(2) by simp
  have qs-carr: set qs ⊆ carrier R
    using qs-sub subset by blast
  have rest-carr: foldr (⊗_R) qs 1_R ∈ carrier R
    by (rule multlist-closed[OF qs-carr])
  obtain d where
      d-in: d ∈ carrier R
    and hd: foldr (⊗_R) (q # qs) 1_R = p ⊗_R d
    using Cons.prems(3) unfolding factor-def by blast
  have q-dvd-pd: q divides_R (p ⊗_R d)
  proof −
    have q ⊗_R foldr (⊗_R) qs 1_R = p ⊗_R d
      using hd by simp
    then show ?thesis
      using rest-carr by force
  qed
  have q-dvd-p-or-d: q divides_R p ∨ q divides_R d
    using primeE[OF q-prime] p-in d-in q-dvd-pd by blast
  from q-dvd-p-or-d show ?case
  proof
    assume q-dvd-p: q divides_R p
    have q-assoc-p: q ∼_R p
          by (meson divides-irreducible-condition hp q-carr q-dvd-p q-not-unit
ring-irreducible-def)
    have q-assoc-p-mult: q ∼_(mult-of R) p
      using assoc-iff-assoc-mult[OF q-carr p-in] q-assoc-p by blast
    have q-prime-mult: prime (mult-of R) q
      using prime-eq-prime-mult[OF q-carr] q-prime by blast
    have prime R p
    proof −
      have prime (mult-of R) p
      proof (rule mult-of.prime-cong[OF q-prime-mult q-assoc-p-mult])
        show q ∈ carrier (mult-of R)
          using q-carr q-nz by simp
        show p ∈ carrier (mult-of R)
```

```
            using p-in ring-irreducibleE(1)[OF p-in hp] by simp
          qed
          then show ?thesis
            using prime-eq-prime-mult[OF p-in] by blast
        qed
        then have p-prime: prime R p .
        from ring-irreducibleE(1)[OF p-in hp] p-prime show ?thesis
          by (rule ring-primeI)
      next
        assume q-dvd-d: q dividesR d
        obtain e where e-in: e ∈ carrier R and he: d = q ⊗R e
          using q-dvd-d unfolding factor-def by blast
        have pe-carr: p ⊗R e ∈ carrier R
          using p-in e-in by auto
        have rest-eq: foldr (⊗R) qs 1R = p ⊗R e
        proof −
          have q ⊗R foldr (⊗R) qs 1R = q ⊗R (p ⊗R e)
            using hd he q-carr rest-carr p-in e-in by (simp add: m-assoc m-comm
m-lcomm)
          then show ?thesis
            using q-nz q-carr rest-carr pe-carr by (simp add: m-lcancel)
        qed
        have p-dvd-rest: p dividesR foldr (⊗R) qs 1R
          unfolding factor-def using e-in rest-eq by blast
        show ?thesis
          by (rule Cons.IH[OF qs-sub qs-prime p-dvd-rest])
      qed
    qed
  qed
  show ?thesis
    by (rule hmain[OF fs-sub hf hdiv])
qed

lemma prime-of-irreducible-of-dvd-mem-prime-generated:
  assumes hS: ring-prime-generated R S
    and p-in: p ∈ carrier R
    and hp: ring-irreducibleR p
    and s-in: s ∈ S
    and p-dvd-s: p dividesR s
  shows ring-primeR p
proof −
  obtain fs where
      fs-sub: set fs ⊆ S
      and hf: ∀ q∈set fs. ring-primeR q
      and hprod: foldr (⊗R) fs 1R = s
    using ring-prime-generatedE[OF hS s-in] by blast
  have p dividesR foldr (⊗R) fs 1R
    using p-dvd-s by (simp add: hprod)
  then show ?thesis
```

by (*rule prime-of-irreducible-of-dvd-prime-factors*[*OF fs-sub hf p-in hp*])
qed

lemma *dvd-of-localization-dvd*:
assumes *hS*: $\bigwedge s.\ s \in S \Longrightarrow \mathit{ring\text{-}prime}_R\ s \vee s \in \mathit{Units}\ R$
and *p-in*: $p \in \mathit{carrier}\ R$
and *a-in*: $a \in \mathit{carrier}\ R$
and *hp*: $\mathit{ring\text{-}irreducible}_R\ p$
and *havoid*: *ring-avoids R S p*
and *hdiv*: $\mathit{rng\text{-}to\text{-}rng\text{-}of\text{-}frac}\ p\ \mathit{divides}_{\mathit{rec\text{-}rng\text{-}of\text{-}frac}\ \mathit{rng\text{-}to\text{-}rng\text{-}of\text{-}frac}}\ a$
shows $p\ \mathit{divides}_R\ a$
proof −
have *zero-notin*: $\mathbf{0} \notin S$
using *hS* **by** (*rule zero-notin-submonoid-of-prime-or-unit*)
have *hdiv'*:
$\exists s \in S.\ p\ \mathit{divides}_R\ (s \otimes_R a)$
using *dvd-map-iff*[*OF p-in a-in zero-notin no-zero-divisors*] *hdiv*
by *blast*
then obtain *s* **where** *s-in*: $s \in S$ **and** *p-dvd-sa*: $p\ \mathit{divides}_R\ (s \otimes_R a)$
by *blast*
have *s-in-carrier*: $s \in \mathit{carrier}\ R$
using *s-in subset rev-subsetD* **by** *blast*
then obtain *c* **where** *c-in*: $c \in \mathit{carrier}\ R$ **and** *hc*: $s \otimes_R a = p \otimes_R c$
using *p-dvd-sa* **unfolding** *factor-def* **by** *blast*
from *hS*[*OF s-in*] **show** *?thesis*
proof
assume *hs-prime*: $\mathit{ring\text{-}prime}_R\ s$
have *s-nz*: $s \neq \mathbf{0}$ **and** *s-prime*: *prime R s*
using *ring-primeE*[*OF s-in-carrier hs-prime*] **by** *auto*
have *s-not-unit*: $s \notin \mathit{Units}\ R$
using *primeE*[*OF s-prime*] **by** *blast*
have *s-dvd-pc*: $s\ \mathit{divides}_R\ (p \otimes_R c)$
using *a-in hc* **by** *force*
have *s-dvd-p-or-c*: $s\ \mathit{divides}_R\ p \vee s\ \mathit{divides}_R\ c$
using *primeE*[*OF s-prime*] *p-in c-in s-dvd-pc* **by** *blast*
from *s-dvd-p-or-c* **show** *?thesis*
proof
assume *s-dvd-p*: $s\ \mathit{divides}_R\ p$
have *s-assoc-p*: $s \sim_R p$
by (*meson divides-irreducible-condition hp ring-irreducible-def s-dvd-p s-in-carrier s-not-unit*)
have $p\ \mathit{divides}_R\ s$
using *associatedD*[*OF associated-sym*[*OF s-assoc-p*]] .
moreover have $\neg\ p\ \mathit{divides}_R\ s$
using *havoid s-in* **unfolding** *ring-avoids-def* **by** *blast*
ultimately show *?thesis*
by *contradiction*
next

```
    assume s-dvd-c: s divides_R c
    then obtain d where d-in: d ∈ carrier R and hd: c = s ⊗_R d
      unfolding factor-def by blast
    have pd-in: p ⊗_R d ∈ carrier R
      using p-in d-in by auto
    have s ⊗_R a = s ⊗_R (p ⊗_R d)
      using hc hd s-in-carrier a-in p-in d-in by (simp add: m-assoc m-comm
m-lcomm)
    then have a = p ⊗_R d
      using s-nz s-in-carrier a-in pd-in by (simp add: m-lcancel)
    then show ?thesis
      unfolding factor-def using d-in by blast
  qed
 next
  assume hs-unit: s ∈ Units R
  have sa-assoc-a: (s ⊗_R a) ∼_R a
    using hs-unit a-in s-in-carrier by (intro associatedI2') (simp add: m-comm)
  have p divides_R (s ⊗_R a)
    using p-dvd-sa .
  then show ?thesis
    using divides-cong-r[OF - sa-assoc-a p-in] by blast
 qed
qed

lemma prime-of-localization-prime:
 assumes hS: ⋀s. s ∈ S ⟹ ring-prime_R s ∨ s ∈ Units R
  and p-in: p ∈ carrier R
  and hp: ring-irreducible_R p
  and havoid: ring-avoids R S p
  and hploc: ring-prime_rec-rng-of-frac (rng-to-rng-of-frac p)
 shows ring-prime_R p
proof (rule ring-primeI)
 show p ≠ 0
  using ring-irreducibleE(1)[OF p-in hp] .
next
 show prime R p
 proof (rule primeI)
  show p ∉ Units R
    using ring-irreducibleE(4)[OF p-in hp] .
 next
  fix a b
  assume a-in: a ∈ carrier R
    and b-in: b ∈ carrier R
    and p-dvd-ab: p divides_R (a ⊗_R b)
  obtain c where c-in: c ∈ carrier R and hc: a ⊗_R b = p ⊗_R c
    using p-dvd-ab unfolding factor-def by blast
  have a-frac-in: rng-to-rng-of-frac a ∈ carrier rec-rng-of-frac
    by (rule ring-hom-closed[OF rng-to-rng-of-frac-is-ring-hom a-in])
  have b-frac-in: rng-to-rng-of-frac b ∈ carrier rec-rng-of-frac
```

```
    by (rule ring-hom-closed[OF rng-to-rng-of-frac-is-ring-hom b-in])
  have c-frac-in: rng-to-rng-of-frac c ∈ carrier rec-rng-of-frac
    by (rule ring-hom-closed[OF rng-to-rng-of-frac-is-ring-hom c-in])
  have hloc-div:
    rng-to-rng-of-frac p divides_rec-rng-of-frac
      (rng-to-rng-of-frac a ⊗_rec-rng-of-frac rng-to-rng-of-frac b)
  proof −
    have map-ab:
      rng-to-rng-of-frac (a ⊗_R b) =
        rng-to-rng-of-frac a ⊗_rec-rng-of-frac rng-to-rng-of-frac b
      using ring-hom-mult[OF rng-to-rng-of-frac-is-ring-hom a-in b-in] .
    have map-pc:
      rng-to-rng-of-frac (p ⊗_R c) =
        rng-to-rng-of-frac p ⊗_rec-rng-of-frac rng-to-rng-of-frac c
      using ring-hom-mult[OF rng-to-rng-of-frac-is-ring-hom p-in c-in] .
    have rng-to-rng-of-frac p divides_rec-rng-of-frac rng-to-rng-of-frac (a ⊗_R b)
      unfolding factor-def using c-frac-in hc map-pc by auto
    then show ?thesis
      using map-ab by simp
  qed
  have hloc-prime: prime rec-rng-of-frac (rng-to-rng-of-frac p)
    using hploc unfolding ring-prime-def by simp
  have hloc-cases:
    rng-to-rng-of-frac p divides_rec-rng-of-frac rng-to-rng-of-frac a ∨
      rng-to-rng-of-frac p divides_rec-rng-of-frac rng-to-rng-of-frac b
    by (metis a-frac-in b-frac-in hloc-div hloc-prime primeE)
  then show p divides_R a ∨ p divides_R b
    using dvd-of-localization-dvd[OF hS p-in a-in hp havoid]
      dvd-of-localization-dvd[OF hS p-in b-in hp havoid]
    by blast
 qed
qed

lemma dvd-of-mul-eq-prime-factors:
 assumes fs-sub: set fs ⊆ S
   and hf: ∀ q∈set fs. ring-prime_R q
   and p-in: p ∈ carrier R
   and a-in: a ∈ carrier R
   and hp: ring-irreducible_R p
   and hnot: ∀ q∈set fs. ¬ p divides_R q
   and c-in: c ∈ carrier R
   and hEq: foldr (⊗_R) fs 1_R ⊗_R a = p ⊗_R c
 shows p divides_R a
proof −
 have hmain:
   ⋀gs d. set gs ⊆ S ⟹
     (∀ q∈set gs. ring-prime_R q) ⟹
     (∀ q∈set gs. ¬ p divides_R q) ⟹
     d ∈ carrier R ⟹
```

```
    foldr (⊗_R) gs 1_R ⊗_R a = p ⊗_R d ⟹
    p divides_R a
proof −
  fix gs d
  show set gs ⊆ S ⟹
    (∀ q∈set gs. ring-prime_R q) ⟹
    (∀ q∈set gs. ¬ p divides_R q) ⟹
    d ∈ carrier R ⟹
    foldr (⊗_R) gs 1_R ⊗_R a = p ⊗_R d ⟹
    p divides_R a
  proof (induction gs arbitrary: d)
    case Nil
    from Nil.prems have d-in: d ∈ carrier R and eq: a = p ⊗_R d
      using a-in by simp-all
    show ?case
      using d-in eq unfolding factor-def by blast
  next
    case (Cons q qs)
  have q-in: q ∈ S
    using Cons.prems(1) by simp
  have q-carr: q ∈ carrier R
    using q-in subset rev-subsetD by blast
  have q-ring-prime: ring-prime_R q
    using Cons.prems(2) by simp
  have q-prime: prime R q
    using ring-primeE(3)[OF q-carr q-ring-prime] .
  have q-nz: q ≠ 0
    using ring-primeE(1)[OF q-carr q-ring-prime] .
  have q-ring-irred: ring-irreducible_R q
    using ring-prime-imp-ring-irreducible[OF q-carr q-ring-prime] .
  have q-not-unit: q ∉ Units R
    using ring-irreducibleE(4)[OF q-carr q-ring-irred] .
  have qs-sub: set qs ⊆ S
    using Cons.prems(1) by simp
  have qs-prime: ∀ r∈set qs. ring-prime_R r
    using Cons.prems(2) by simp
  have qs-not: ∀ r∈set qs. ¬ p divides_R r
    using Cons.prems(3) by simp
  have qs-carr: set qs ⊆ carrier R
    using qs-sub subset by blast
  have rest-carr: foldr (⊗_R) qs 1_R ∈ carrier R
    by (rule multlist-closed[OF qs-carr])
  have resta-carr: foldr (⊗_R) qs 1_R ⊗_R a ∈ carrier R
    using rest-carr a-in by auto
  have q-dvd-pd: q divides_R (p ⊗_R d)
  proof −
    have hqd: q ⊗_R (foldr (⊗_R) qs 1_R ⊗_R a) = p ⊗_R d
      using Cons.prems(5) a-in m-comm m-lcomm q-carr rest-carr by force
    then show ?thesis
```

```
      by (metis dividesI resta-carr)
    qed
    have q-dvd-p-or-d: q divides_R p ∨ q divides_R d
      using primeE[OF q-prime] p-in Cons.prems(4) q-dvd-pd by blast
    from q-dvd-p-or-d show ?case
    proof
      assume q-dvd-p: q divides_R p
      have q-assoc-p: q ∼_R p
      by (metis divides-irreducible-condition hp q-carr q-dvd-p q-not-unit ring-irreducible-def)
      have p divides_R q
        using associatedD[OF associated-sym[OF q-assoc-p]] .
      with Cons.prems(3) show ?thesis
        by simp
    next
      assume q-dvd-d: q divides_R d
      obtain e where e-in: e ∈ carrier R and he: d = q ⊗_R e
        using q-dvd-d unfolding factor-def by blast
      have pe-carr: p ⊗_R e ∈ carrier R
        using p-in e-in by auto
      have rest-eq: foldr (⊗_R) qs 1_R ⊗_R a = p ⊗_R e
      proof −
        have q ⊗_R (foldr (⊗_R) qs 1_R ⊗_R a) =
            q ⊗_R (p ⊗_R e)
          using Cons.prems(5) he q-carr rest-carr a-in p-in e-in
          by (simp add: m-assoc m-comm m-lcomm)
        then show ?thesis
          using q-nz q-carr resta-carr pe-carr by (simp add: m-lcancel)
      qed
      show ?thesis
        by (rule Cons.IH[OF qs-sub qs-prime qs-not e-in rest-eq])
    qed
  qed
  qed
  show ?thesis
    by (rule hmain[OF fs-sub hf hnot c-in hEq])
qed

lemma dvd-of-localization-dvd-prime-generated:
  assumes hS: ring-prime-generated R S
    and p-in: p ∈ carrier R
    and a-in: a ∈ carrier R
    and hp: ring-irreducible_R p
    and havoid: ring-avoids R S p
    and hdiv: rng-to-rng-of-frac p divides_{rec-rng-of-frac} rng-to-rng-of-frac a
  shows p divides_R a
proof −
  have zero-notin: 0 ∉ S
    using zero-notin-submonoid-of-prime-generated[OF hS] .
  have hdiv':
```

```
    ∃ s ∈ S. p divides_R (s ⊗_R a)
   using dvd-map-iff[OF p-in a-in zero-notin no-zero-divisors] hdiv
   by blast
  then obtain s where s-in: s ∈ S and p-dvd-sa: p divides_R (s ⊗_R a)
   by blast
  obtain c where c-in: c ∈ carrier R and hc: s ⊗_R a = p ⊗_R c
   using p-dvd-sa unfolding factor-def by blast
  obtain fs where
    fs-sub: set fs ⊆ S
    and hf: ∀ q∈set fs. ring-prime_R q
    and hprod: foldr (⊗_R) fs 1_R = s
   using ring-prime-generatedE[OF hS s-in] by blast
  have hnot: ∀ q∈set fs. ¬ p divides_R q
   using havoid fs-sub unfolding ring-avoids-def by blast
  have hEq: foldr (⊗_R) fs 1_R ⊗_R a = p ⊗_R c
   using hc by (simp add: hprod)
  show ?thesis
   using dvd-of-mul-eq-prime-factors[OF fs-sub hf p-in a-in hp hnot c-in hEq] .
qed

lemma map-irreducible-not-unit-of-zero-notin:
  assumes zero-notin: 0 ∉ S
   and loc-dom: domain rec-rng-of-frac
   and p-in: p ∈ carrier R
   and hp: ring-irreducible_R p
   and havoid: ring-avoids R S p
  shows rng-to-rng-of-frac p ∉ Units rec-rng-of-frac
proof
  interpret L: domain rec-rng-of-frac
   by (rule loc-dom)
  assume map-p-unit: rng-to-rng-of-frac p ∈ Units rec-rng-of-frac
  have map-p-dvd-one:
    rng-to-rng-of-frac p divides_rec-rng-of-frac 1_rec-rng-of-frac
   using L.unit-divides[OF map-p-unit L.one-closed] .
  have map-p-dvd-map-one:
    rng-to-rng-of-frac p divides_rec-rng-of-frac rng-to-rng-of-frac 1
   using map-p-dvd-one ring-hom-one[OF rng-to-rng-of-frac-is-ring-hom]
   by simp
  have one-in: 1 ∈ carrier R
   by simp
  have hdiv:
    ∃ s ∈ S. p divides_R (s ⊗_R 1)
   using dvd-map-iff[OF p-in one-in zero-notin no-zero-divisors] map-p-dvd-map-one
   by blast
  then obtain s where s-in: s ∈ S and p-dvd-s: p divides_R (s ⊗_R 1)
   by blast
  have s-carr: s ∈ carrier R
   using s-in subset rev-subsetD by blast
  have p divides_R s
```

```
    using p-dvd-s s-carr by simp
  moreover have ¬ p dividesR s
    using havoid s-in unfolding ring-avoids-def by blast
  ultimately show False
    by contradiction
qed

lemma map-irreducible-not-unit:
  assumes hS: ⋀s. s ∈ S ⟹ ring-primeR s ∨ s ∈ Units R
    and loc-dom: domain rec-rng-of-frac
    and p-in: p ∈ carrier R
    and hp: ring-irreducibleR p
    and havoid: ring-avoids R S p
  shows rng-to-rng-of-frac p ∉ Units rec-rng-of-frac
proof −
  have zero-notin: 0 ∉ S
    using hS by (rule zero-notin-submonoid-of-prime-or-unit)
  show ?thesis
     using map-irreducible-not-unit-of-zero-notin[OF zero-notin loc-dom p-in hp
havoid] .
qed

lemma localization-irreducible-of-irreducible:
  assumes hS: ⋀s. s ∈ S ⟹ ring-primeR s ∨ s ∈ Units R
    and loc-dom: domain rec-rng-of-frac
    and p-in: p ∈ carrier R
    and hp: ring-irreducibleR p
    and havoid: ring-avoids R S p
  shows ring-irreducible_{rec-rng-of-frac} (rng-to-rng-of-frac p)
proof −
  interpret L: domain rec-rng-of-frac
    by (rule loc-dom)
  have zero-notin: 0 ∉ S
    using hS by (rule zero-notin-submonoid-of-prime-or-unit)
  have map-p-in: rng-to-rng-of-frac p ∈ carrier rec-rng-of-frac
    by (rule ring-hom-closed[OF rng-to-rng-of-frac-is-ring-hom p-in])
  have map-p-nz: rng-to-rng-of-frac p ≠ 0_{rec-rng-of-frac}
    using map-eq-zero-iff[OF p-in zero-notin no-zero-divisors]
      ring-irreducibleE(1)[OF p-in hp]
    by blast
  have map-p-not-unit: rng-to-rng-of-frac p ∉ Units rec-rng-of-frac
    using map-irreducible-not-unit[OF hS loc-dom p-in hp havoid] .
  show ?thesis
  proof (rule L.ring-irreducibleI)
    show rng-to-rng-of-frac p ∈ carrier rec-rng-of-frac − {0_{rec-rng-of-frac}}
      using map-p-in map-p-nz by blast
  next
    show rng-to-rng-of-frac p ∉ Units rec-rng-of-frac
      using map-p-not-unit .
```

next
fix *x y*
assume *x-in*: $x \in$ *carrier rec-rng-of-frac*
and *y-in*: $y \in$ *carrier rec-rng-of-frac*
and *xy*: *rng-to-rng-of-frac* $p = x \otimes_{\textit{rec-rng-of-frac}} y$
from *fraction-surj*[*OF x-in*] **obtain** a **where** *a-in*: $a \in$ *carrier* R
and *hs*: $\exists\, s \in S.\ x = (a \mid_{rel} s)$
by *blast*
from *hs* **obtain** s **where** *s-in*: $s \in S$ **and** *x-def*: $x = (a \mid_{rel} s)$
by *blast*
from *fraction-surj*[*OF y-in*] **obtain** b **where** *b-in*: $b \in$ *carrier* R
and *ht*: $\exists\, t \in S.\ y = (b \mid_{rel} t)$
by *blast*
from *ht* **obtain** t **where** *t-in*: $t \in S$ **and** *y-def*: $y = (b \mid_{rel} t)$
by *blast*
have *as-rel*: $(a, s) \in$ *carrier rel*
using *a-in s-in* **by** (*simp add*: *rel-def*)
have *bt-rel*: $(b, t) \in$ *carrier rel*
using *b-in t-in* **by** (*simp add*: *rel-def*)
have *st-in*: $s \otimes_R t \in S$
using *s-in t-in* **by** *simp*
have *st-carrier*: $s \otimes_R t \in$ *carrier* R
using *st-in subset rev-subsetD* **by** *blast*
have *p-rel*: $(p, \mathbf{1}) \in$ *carrier rel*
using *p-in one-closed* **by** (*simp add*: *rel-def*)
have *ab-in*: $a \otimes_R b \in$ *carrier* R
using *a-in b-in* **by** *auto*
have *ab-rel*: $(a \otimes_R b,\ s \otimes_R t) \in$ *carrier rel*
using *a-in b-in st-in* **by** (*simp add*: *rel-def*)
have *frac-prod*:
$x \otimes_{\textit{rec-rng-of-frac}} y = (a \otimes_R b \mid_{rel} s \otimes_R t)$
using *fraction-mult-rep*[*OF as-rel bt-rel*] *x-def y-def* **by** *simp*
have *eq-frac*:
rng-to-rng-of-frac $p = (a \otimes_R b \mid_{rel} s \otimes_R t)$
using *xy frac-prod* **by** (*simp add*: *rng-to-rng-of-frac-def*)
have *p-eq-raw*: $(s \otimes_R t) \otimes_R p = \mathbf{1}_R \otimes_R (a \otimes_R b)$
proof −
have *eq-cross*:
rng-to-rng-of-frac $p = (a \otimes_R b \mid_{rel} s \otimes_R t) \longleftrightarrow$
$(s \otimes_R t) \otimes_R p = \mathbf{1}_R \otimes_R (a \otimes_R b)$
using *fraction-eq-iff-cross-multiply*[*OF p-rel ab-rel zero-notin no-zero-divisors*]
unfolding *rng-to-rng-of-frac-def* **by** *simp*
from *eq-cross eq-frac* **show** *?thesis*
by *blast*
qed
have *p-eq*: $(s \otimes_R t) \otimes_R p = a \otimes_R b$
using *p-eq-raw ab-in* **by** *simp*
from *hS*[*OF st-in*] **show** $x \in$ *Units rec-rng-of-frac* $\vee$ $y \in$ *Units rec-rng-of-frac*
proof

assume *hst-prime*: $ring\text{-}prime_R\ (s \otimes_R t)$
have *st-prime*: $prime\ R\ (s \otimes_R t)$
and *st-nz*: $s \otimes_R t \neq \mathbf{0}$
using *ring-primeE*[*OF st-carrier hst-prime*] **by** *auto*
have *st-dvd-ab*: $(s \otimes_R t)\ divides_R\ (a \otimes_R b)$
proof −
have *ab-eq*: $a \otimes_R b = (s \otimes_R t) \otimes_R p$
using *p-eq* **by** *simp*
show *?thesis*
by (*rule dividesI*[*OF p-in ab-eq*])
qed
have *st-dvd-a-or-b*:
$(s \otimes_R t)\ divides_R\ a \vee (s \otimes_R t)\ divides_R\ b$
using *primeE*[*OF st-prime*] *a-in b-in st-dvd-ab* **by** *blast*
from *st-dvd-a-or-b* **show** *?thesis*
proof
assume *st-dvd-a*: $(s \otimes_R t)\ divides_R\ a$
then obtain *d* **where** *d-in*: $d \in carrier\ R$ **and** *hd*: $a = (s \otimes_R t) \otimes_R d$
unfolding *factor-def* **by** *blast*
have *db-in*: $d \otimes_R b \in carrier\ R$
using *d-in b-in* **by** *auto*
have $p = d \otimes_R b$
proof −
have $(s \otimes_R t) \otimes_R p = a \otimes_R b$
by (*rule p-eq*)
also have $\ldots = ((s \otimes_R t) \otimes_R d) \otimes_R b$
using *hd* **by** *simp*
also have $\ldots = (s \otimes_R t) \otimes_R (d \otimes_R b)$
using *st-carrier d-in b-in* **by** (*simp add*: *m-assoc*)
finally have $(s \otimes_R t) \otimes_R p =$
$(s \otimes_R t) \otimes_R (d \otimes_R b)$ **.**
then show *?thesis*
using *st-nz st-carrier p-in db-in* **by** (*simp add*: *m-lcancel*)
qed
then have *d-or-b-unit*: $d \in Units\ R \vee b \in Units\ R$
using *ring-irreducibleE*(5)[*OF p-in hp d-in b-in*] **by** *blast*
then show *?thesis*
proof
assume *d-unit*: $d \in Units\ R$
have *t-in-carrier*: $t \in carrier\ R$
using *t-in subset rev-subsetD* **by** *blast*
have *td-in*: $t \otimes_R d \in carrier\ R$
using *t-in-carrier d-in* **by** *auto*
have *td-rel*: $(t \otimes_R d, \mathbf{1}) \in carrier\ rel$
using *td-in one-closed* **by** (*simp add*: *rel-def*)
have *x-eq-map-td*: $x = rng\text{-}to\text{-}rng\text{-}of\text{-}frac\ (t \otimes_R d)$
proof −
have *s-in-carrier*: $s \in carrier\ R$
using *s-in subset rev-subsetD* **by** *blast*

```
      have x = (a |_rel s)
        using x-def by simp
      also have ... = ((s ⊗_R t) ⊗_R d |_rel s)
        using hd by (simp add: m-assoc)
      also have ... = (s ⊗_R (t ⊗_R d) |_rel s)
        using s-in-carrier t-in-carrier d-in by (simp add: m-assoc)
      also have ... = (s ⊗_R (t ⊗_R d) |_rel s ⊗_R 1)
        using s-in-carrier by simp
      also have ... = (t ⊗_R d |_rel 1)
        using fraction-rescale[OF td-rel s-in] by simp
      also have ... = rng-to-rng-of-frac (t ⊗_R d)
        by (simp add: rng-to-rng-of-frac-def)
      finally show ?thesis .
    qed
    have t-unit-loc: rng-to-rng-of-frac t ∈ Units rec-rng-of-frac
      using map-submonoid-elem-is-unit[OF t-in] .
    have d-unit-loc: rng-to-rng-of-frac d ∈ Units rec-rng-of-frac
      using map-unit-is-unit[OF d-unit] .
    have rng-to-rng-of-frac (t ⊗_R d) =
        rng-to-rng-of-frac t ⊗_rec-rng-of-frac rng-to-rng-of-frac d
     using ring-hom-mult[OF rng-to-rng-of-frac-is-ring-hom t-in-carrier d-in]
.
    then show ?thesis
      using x-eq-map-td t-unit-loc d-unit-loc by simp
  next
    assume b-unit: b ∈ Units R
    have (b |_rel t) ∈ Units rec-rng-of-frac
      by (rule fraction-unit-numerator-is-unit[OF b-unit t-in])
    then show ?thesis
      using y-def by blast
  qed
next
  assume st-dvd-b: (s ⊗_R t) divides_R b
  then obtain d where d-in: d ∈ carrier R and hd: b = (s ⊗_R t) ⊗_R d
    unfolding factor-def by blast
  have ad-in: a ⊗_R d ∈ carrier R
    using a-in d-in by auto
  have p = a ⊗_R d
  proof −
    have (s ⊗_R t) ⊗_R p = a ⊗_R b
      by (rule p-eq)
    also have ... = a ⊗_R ((s ⊗_R t) ⊗_R d)
      using hd by simp
    also have ... = (s ⊗_R t) ⊗_R (a ⊗_R d)
      using st-carrier a-in d-in by (simp add: m-assoc m-comm m-lcomm)
    finally have (s ⊗_R t) ⊗_R p =
        (s ⊗_R t) ⊗_R (a ⊗_R d) .
    then show ?thesis
      using st-nz st-carrier p-in ad-in by (simp add: m-lcancel)
```

```
    qed
    then have a-or-d-unit: a ∈ Units R ∨ d ∈ Units R
      using ring-irreducibleE(5)[OF p-in hp a-in d-in] by blast
    then show ?thesis
    proof
      assume a-unit: a ∈ Units R
      have (a |_rel s) ∈ Units rec-rng-of-frac
        by (rule fraction-unit-numerator-is-unit[OF a-unit s-in])
      then show ?thesis
        using x-def by blast
    next
      assume d-unit: d ∈ Units R
      have s-in-carrier: s ∈ carrier R
        using s-in subset rev-subsetD by blast
      have sd-in: s ⊗_R d ∈ carrier R
        using s-in-carrier d-in by auto
      have sd-rel: (s ⊗_R d, 1) ∈ carrier rel
        using sd-in one-closed by (simp add: rel-def)
      have y-eq-map-sd: y = rng-to-rng-of-frac (s ⊗_R d)
      proof −
        have t-in-carrier: t ∈ carrier R
          using t-in subset rev-subsetD by blast
        have y = (b |_rel t)
          using y-def by simp
        also have ... = ((s ⊗_R t) ⊗_R d |_rel t)
          using hd by (simp add: m-assoc)
        also have ... = (t ⊗_R (s ⊗_R d) |_rel t)
          using t-in-carrier s-in-carrier d-in by (simp add: m-assoc m-comm
m-lcomm)
        also have ... = (t ⊗_R (s ⊗_R d) |_rel t ⊗_R 1)
          using t-in-carrier by simp
        also have ... = (s ⊗_R d |_rel 1)
          using fraction-rescale[OF sd-rel t-in] by simp
        also have ... = rng-to-rng-of-frac (s ⊗_R d)
          by (simp add: rng-to-rng-of-frac-def)
        finally show ?thesis .
      qed
      have s-unit-loc: rng-to-rng-of-frac s ∈ Units rec-rng-of-frac
        using map-submonoid-elem-is-unit[OF s-in] .
      have d-unit-loc: rng-to-rng-of-frac d ∈ Units rec-rng-of-frac
        using map-unit-is-unit[OF d-unit] .
      have rng-to-rng-of-frac (s ⊗_R d) =
          rng-to-rng-of-frac s ⊗_rec-rng-of-frac rng-to-rng-of-frac d
       using ring-hom-mult[OF rng-to-rng-of-frac-is-ring-hom s-in-carrier d-in]
.
      then show ?thesis
        using y-eq-map-sd s-unit-loc d-unit-loc by simp
    qed
  qed
```

next
assume *hst-unit*: $s \otimes_R t \in Units\ R$
have *ab-assoc-p*: $(a \otimes_R b) \sim_R p$
proof −
have $a \otimes_R b = (s \otimes_R t) \otimes_R p$
using *p-eq* **by** *simp*
also have $\ldots = p \otimes_R (s \otimes_R t)$
using *p-in Units-closed*[*OF hst-unit*] **by** (*simp add*: *m-assoc m-comm m-lcomm*)
finally have $a \otimes_R b = p \otimes_R (s \otimes_R t)$ **.**
then show *?thesis*
using *hst-unit p-in* **by** (*rule associatedI2'*)
qed
have *p-mult-irreducible*: *irreducible* (*mult-of R*) *p*
using *ring-irreducibleE*(*3*)[*OF p-in hp*] **.**
have *p-assoc-ab-mult*: $p \sim_{mult\text{-}of\ R} (a \otimes_R b)$
using *assoc-iff-assoc-mult*[*OF p-in ab-in*] *associated-sym*[*OF ab-assoc-p*] **by** *blast*
have *st-carr*: $s \otimes_R t \in carrier\ R$
using *Units-closed*[*OF hst-unit*] **.**
have *st-nz*: $s \otimes_R t \neq \mathbf{0}$
using *st-in zero-notin* **by** *auto*
have *p-nz*: $p \neq \mathbf{0}$
using *ring-irreducibleE*(*1*)[*OF p-in hp*] **.**
have *ab-nz*: $a \otimes_R b \neq \mathbf{0}$
using *ab-assoc-p divides-cong-r p-nz zero-divides* **by** *blast*
have *a-nz*: $a \neq \mathbf{0}$
using *ab-nz b-in* **by** *fastforce*
have *b-nz*: $b \neq \mathbf{0}$
using *a-in ab-nz* **by** *force*
have *a-in-mult*: $a \in carrier\ (mult\text{-}of\ R)$
using *a-in a-nz* **by** *simp*
have *b-in-mult*: $b \in carrier\ (mult\text{-}of\ R)$
using *b-in b-nz* **by** *simp*
have *ab-mult-irreducible*: *irreducible* (*mult-of R*) $(a \otimes_R b)$
proof (*rule mult-of.irreducible-cong*[*OF p-mult-irreducible p-assoc-ab-mult*])
show $p \in carrier\ (mult\text{-}of\ R)$
using *p-in p-nz* **by** *simp*
show $(a \otimes_R b) \in carrier\ (mult\text{-}of\ R)$
using *ab-in ab-nz* **by** *simp*
qed
have *xy-unit*: $x \in Units\ rec\text{-}rng\text{-}of\text{-}frac \vee y \in Units\ rec\text{-}rng\text{-}of\text{-}frac$
proof (*rule mult-of.irreducible-prodE*[*OF ab-mult-irreducible a-in-mult b-in-mult*])
assume *a-irreducible*: *irreducible* (*mult-of R*) *a* **and** *b-unit-mult*: $b \in Units\ (mult\text{-}of\ R)$
have *b-unit*: $b \in Units\ R$
using *b-unit-mult* **by** *simp*
have *y-unit*: $y \in Units\ rec\text{-}rng\text{-}of\text{-}frac$
unfolding *y-def* **using** *fraction-unit-numerator-is-unit*[*OF b-unit t-in*] **by**

blast
      show *x* ∈ *Units rec-rng-of-frac* ∨ *y* ∈ *Units rec-rng-of-frac*
        using *y-unit* **by** *auto*
      next
        assume *a-unit-mult*: *a* ∈ *Units* (*mult-of R*) **and** *b-irreducible*: *irreducible* (*mult-of R*) *b*
      have *a-unit*: *a* ∈ *Units R*
        using *a-unit-mult* **by** *simp*
      have *x-unit*: *x* ∈ *Units rec-rng-of-frac*
        unfolding *x-def* **using** *fraction-unit-numerator-is-unit*[*OF a-unit s-in*] **by** *blast*
      show *x* ∈ *Units rec-rng-of-frac* ∨ *y* ∈ *Units rec-rng-of-frac*
        by (*simp add*: *x-unit*)
      qed
      then show *?thesis* **.**
    qed
  qed
qed

lemma *nagata-key-lemma*:
  assumes *hS*: ⋀*s*. *s* ∈ *S* ⟹ *ring-prime*$_R$ *s* ∨ *s* ∈ *Units R*
    and *loc-fd*: *factorial-domain rec-rng-of-frac*
    and *p-in*: *p* ∈ *carrier R*
    and *hp*: *ring-irreducible*$_R$ *p*
  shows *ring-prime*$_R$ *p*
proof (*cases* ∃ *s* ∈ *S*. *p divides*$_R$ *s*)
  case *True*
  then obtain *s* **where** *s-in*: *s* ∈ *S* **and** *p-dvd-s*: *p divides*$_R$ *s*
    by *blast*
  show *?thesis*
    using *prime-of-irreducible-of-dvd-mem*[*OF hS p-in hp s-in p-dvd-s*] **.**
next
  case *False*
  interpret *Lfd*: *factorial-domain rec-rng-of-frac*
    by (*rule loc-fd*)
  interpret *L*: *domain rec-rng-of-frac*
    by (*rule Lfd.domain-axioms*)
  have *havoid*: *ring-avoids R S p*
    using *False* **unfolding** *ring-avoids-def* **by** *blast*
  have *loc-irreducible*: *ring-irreducible*$_{rec\text{-}rng\text{-}of\text{-}frac}$ (*rng-to-rng-of-frac p*)
    using *localization-irreducible-of-irreducible*[*OF hS Lfd.domain-axioms p-in hp havoid*] **.**
  have *loc-irred-mult*: *irreducible* (*mult-of rec-rng-of-frac*) (*rng-to-rng-of-frac p*)
    using *Lfd.ring-irreducibleE*(*3*)[*OF ring-hom-closed*[*OF rng-to-rng-of-frac-is-ring-hom p-in*] *loc-irreducible*] **.**
  have *loc-in-nz*: *rng-to-rng-of-frac p* ∈ *carrier rec-rng-of-frac* − {**0**$_{rec\text{-}rng\text{-}of\text{-}frac}$}
    using *ring-hom-closed*[*OF rng-to-rng-of-frac-is-ring-hom p-in*]
    Lfd.ring-irreducibleE(*1*)[*OF ring-hom-closed*[*OF rng-to-rng-of-frac-is-ring-hom p-in*] *loc-irreducible*]

by *blast*
have *loc-in-mult*: *rng-to-rng-of-frac p* ∈ *carrier* (*mult-of rec-rng-of-frac*)
using *loc-in-nz* **by** *simp*
have *loc-prime-mult*: *prime* (*mult-of rec-rng-of-frac*) (*rng-to-rng-of-frac p*)
by (*rule factorial-monoid.irreducible-prime*[*OF Lfd.factorial-monoid-axioms loc-irred-mult loc-in-mult*])
have *loc-ring-prime*: *ring-prime*$_{\textit{rec-rng-of-frac}}$ (*rng-to-rng-of-frac p*)
using *Lfd.ring-primeI'*[*OF loc-in-nz loc-prime-mult*] **.**
show *?thesis*
using *prime-of-localization-prime*[*OF hS p-in hp havoid loc-ring-prime*] **.**
qed

lemma *split-prime-factors-of-mul-eq*:
assumes *fs-sub*: *set fs* ⊆ *S*
and *hf*: ∀ *q*∈*set fs*. *ring-prime*$_R$ *q*
and *p-in*: *p* ∈ *carrier R*
and *a-in*: *a* ∈ *carrier R*
and *b-in*: *b* ∈ *carrier R*
and *hEq*: *p* ⊗$_R$ *foldr* (⊗$_R$) *fs* **1**$_R$ = *a* ⊗$_R$ *b*
shows ∃ *fs1 fs2 a' b'*.
fs <~~> *fs1* @ *fs2* ∧
set fs1 ⊆ *S* ∧ (∀ *q*∈*set fs1*. *ring-prime*$_R$ *q*) ∧
set fs2 ⊆ *S* ∧ (∀ *q*∈*set fs2*. *ring-prime*$_R$ *q*) ∧
a' ∈ *carrier R* ∧ *b'* ∈ *carrier R* ∧
a = *foldr* (⊗$_R$) *fs1* **1**$_R$ ⊗$_R$ *a'* ∧
b = *foldr* (⊗$_R$) *fs2* **1**$_R$ ⊗$_R$ *b'* ∧
p = *a'* ⊗$_R$ *b'*
using *fs-sub hf p-in a-in b-in hEq*
proof (*induction fs arbitrary*: *a b*)
case *Nil*
have *hpab-eq*: *p* ⊗$_R$ **1**$_R$ = *a* ⊗$_R$ *b*
using *Nil.prems*(*6*) **by** *simp*
have *hpab*: *p* = *a* ⊗$_R$ *b*
using *Nil.prems*(*3*) *hpab-eq* **by** *simp*
show *?case*
proof (*intro exI conjI*)
show [] <~~> [] @ []
by *simp*
show *set* [] ⊆ *S*
by *simp*
show ∀ *q*∈*set* []. *ring-prime*$_R$ *q*
by *simp*
show *set* [] ⊆ *S*
by *simp*
show ∀ *q*∈*set* []. *ring-prime*$_R$ *q*
by *simp*
show *a* ∈ *carrier R*
by (*rule Nil.prems*(*4*))

```
    show b ∈ carrier R
      by (rule Nil.prems(5))
    show a = foldr (⊗_R) [] 1_R ⊗_R a
      using Nil.prems(4) by simp
    show b = foldr (⊗_R) [] 1_R ⊗_R b
      using Nil.prems(5) by simp
    show p = a ⊗_R b
      by (rule hpab)
  qed
next
  case (Cons q qs)
  have q-in: q ∈ S
    using Cons.prems(1) by simp
  have q-carr: q ∈ carrier R
    using q-in subset rev-subsetD by blast
  have q-ring-prime: ring-prime_R q
    using Cons.prems(2) by simp
  have q-prime: prime R q
    using ring-primeE(3)[OF q-carr q-ring-prime] .
  have q-nz: q ≠ 0
    using ring-primeE(1)[OF q-carr q-ring-prime] .
  have qs-sub: set qs ⊆ S
    using Cons.prems(1) by simp
  have qs-prime: ∀ r∈set qs. ring-prime_R r
    using Cons.prems(2) by simp
  have qs-carr: set qs ⊆ carrier R
    using qs-sub subset by blast
  have rest-carr: foldr (⊗_R) qs 1_R ∈ carrier R
    by (rule multlist-closed[OF qs-carr])
  have prest-carr: p ⊗_R foldr (⊗_R) qs 1_R ∈ carrier R
    using Cons.prems(3) rest-carr by auto
  have q-dvd-ab: q divides_R (a ⊗_R b)
  proof −
    have hEq-cons: p ⊗_R (q ⊗_R foldr (⊗_R) qs 1_R) = a ⊗_R b
      using Cons.prems(6) by simp
    have q ⊗_R (p ⊗_R foldr (⊗_R) qs 1_R) =
        p ⊗_R (q ⊗_R foldr (⊗_R) qs 1_R)
     using Cons.prems(3) q-carr rest-carr by (simp add: m-assoc m-comm m-lcomm)
    also have ... = a ⊗_R b
      using hEq-cons .
    finally have q ⊗_R (p ⊗_R foldr (⊗_R) qs 1_R) = a ⊗_R b .
    then have ab-eq: a ⊗_R b =
        q ⊗_R (p ⊗_R foldr (⊗_R) qs 1_R)
      by simp
    show ?thesis
      by (rule dividesI[OF prest-carr ab-eq])
  qed
  have q-dvd-a-or-b: q divides_R a ∨ q divides_R b
    using primeE[OF q-prime] Cons.prems(4) Cons.prems(5) q-dvd-ab by blast
```

```
  from q-dvd-a-or-b show ?case
  proof
    assume q-dvd-a: q divides_R a
    obtain a1 where a1-in: a1 ∈ carrier R and ha1: a = q ⊗_R a1
      using q-dvd-a unfolding factor-def by blast
    have a1b-carr: a1 ⊗_R b ∈ carrier R
      using a1-in Cons.prems(5) by auto
    have hEq': p ⊗_R foldr (⊗_R) qs 1_R = a1 ⊗_R b
    proof −
      have q ⊗_R (p ⊗_R foldr (⊗_R) qs 1_R) =
          p ⊗_R (q ⊗_R foldr (⊗_R) qs 1_R)
           using Cons.prems(3) q-carr rest-carr by (simp add: m-assoc m-comm
m-lcomm)
      also have ... = a ⊗_R b
        using Cons.prems(6) by simp
      also have ... = (q ⊗_R a1) ⊗_R b
        using ha1 by simp
      also have ... = q ⊗_R (a1 ⊗_R b)
        using q-carr a1-in Cons.prems(5) by (simp add: m-assoc)
      finally have q ⊗_R (p ⊗_R foldr (⊗_R) qs 1_R) =
          q ⊗_R (a1 ⊗_R b) .
      then show ?thesis
        using q-nz q-carr prest-carr a1b-carr by (simp add: m-lcancel)
    qed
    obtain fs1 fs2 a' b' where
        part: qs <~~> fs1 @ fs2
        and fs1-sub: set fs1 ⊆ S
        and fs1-prime: ∀ r∈set fs1. ring-prime_R r
        and fs2-sub: set fs2 ⊆ S
        and fs2-prime: ∀ r∈set fs2. ring-prime_R r
        and a'-in: a' ∈ carrier R
        and b'-in: b' ∈ carrier R
        and ha: a1 = foldr (⊗_R) fs1 1_R ⊗_R a'
        and hb: b = foldr (⊗_R) fs2 1_R ⊗_R b'
        and hp': p = a' ⊗_R b'
     using Cons.IH[OF qs-sub qs-prime Cons.prems(3) a1-in Cons.prems(5) hEq']
by blast
    show ?thesis
    proof (intro exI conjI)
      show q # qs <~~> (q # fs1) @ fs2
        using part by simp
      show set (q # fs1) ⊆ S
        using q-in fs1-sub by simp
      show ∀ r∈set (q # fs1). ring-prime_R r
        using q-ring-prime fs1-prime by simp
      show set fs2 ⊆ S
        by (rule fs2-sub)
      show ∀ r∈set fs2. ring-prime_R r
        by (rule fs2-prime)
```

```
    show a' ∈ carrier R
      by (rule a'-in)
    show b' ∈ carrier R
      by (rule b'-in)
    show a = foldr (⊗_R) (q # fs1) 1_R ⊗_R a'
    proof −
      have fs1-carr: set fs1 ⊆ carrier R
        using fs1-sub subset by blast
      have fs1-prod-carr: foldr (⊗_R) fs1 1_R ∈ carrier R
        by (rule multlist-closed[OF fs1-carr])
      have a = q ⊗_R a1
        by (rule ha1)
      also have ... = q ⊗_R (foldr (⊗_R) fs1 1_R ⊗_R a')
        using ha by simp
      also have ... = (q ⊗_R foldr (⊗_R) fs1 1_R) ⊗_R a'
        using q-carr fs1-prod-carr a'-in by (simp add: m-assoc)
      also have ... = foldr (⊗_R) (q # fs1) 1_R ⊗_R a'
        by simp
      finally show ?thesis .
    qed
    show b = foldr (⊗_R) fs2 1_R ⊗_R b'
      by (rule hb)
    show p = a' ⊗_R b'
      by (rule hp')
  qed
 next
  assume q-dvd-b: q divides_R b
  obtain b1 where b1-in: b1 ∈ carrier R and hb1: b = q ⊗_R b1
    using q-dvd-b unfolding factor-def by blast
  have ab1-carr: a ⊗_R b1 ∈ carrier R
    using Cons.prems(4) b1-in by auto
  have hEq': p ⊗_R foldr (⊗_R) qs 1_R = a ⊗_R b1
  proof −
    have q ⊗_R (p ⊗_R foldr (⊗_R) qs 1_R) =
        p ⊗_R (q ⊗_R foldr (⊗_R) qs 1_R)
         using Cons.prems(3) q-carr rest-carr by (simp add: m-assoc m-comm
m-lcomm)
    also have ... = a ⊗_R b
      using Cons.prems(6) by simp
    also have ... = a ⊗_R (q ⊗_R b1)
      using hb1 by simp
    also have ... = q ⊗_R (a ⊗_R b1)
     using q-carr Cons.prems(4) b1-in by (simp add: m-assoc m-comm m-lcomm)
    finally have q ⊗_R (p ⊗_R foldr (⊗_R) qs 1_R) =
        q ⊗_R (a ⊗_R b1) .
    then show ?thesis
      using q-nz q-carr prest-carr ab1-carr by (simp add: m-lcancel)
  qed
  obtain fs1 fs2 a' b' where
```

```
      part: qs <~~> fs1 @ fs2
      and fs1-sub: set fs1 ⊆ S
      and fs1-prime: ∀ r∈set fs1. ring-prime_R r
      and fs2-sub: set fs2 ⊆ S
      and fs2-prime: ∀ r∈set fs2. ring-prime_R r
      and a'-in: a' ∈ carrier R
      and b'-in: b' ∈ carrier R
      and ha: a = foldr (⊗_R) fs1 1_R ⊗_R a'
      and hb: b1 = foldr (⊗_R) fs2 1_R ⊗_R b'
      and hp': p = a' ⊗_R b'
    using Cons.IH[OF qs-sub qs-prime Cons.prems(3) Cons.prems(4) b1-in hEq']
by blast
  show ?thesis
  proof (intro exI conjI)
    show q # qs <~~> fs1 @ (q # fs2)
      using part by simp
    show set fs1 ⊆ S
      by (rule fs1-sub)
    show ∀ r∈set fs1. ring-prime_R r
      by (rule fs1-prime)
    show set (q # fs2) ⊆ S
      using q-in fs2-sub by simp
    show ∀ r∈set (q # fs2). ring-prime_R r
      using q-ring-prime fs2-prime by simp
    show a' ∈ carrier R
      by (rule a'-in)
    show b' ∈ carrier R
      by (rule b'-in)
    show a = foldr (⊗_R) fs1 1_R ⊗_R a'
      by (rule ha)
    show b = foldr (⊗_R) (q # fs2) 1_R ⊗_R b'
    proof −
      have fs2-carr: set fs2 ⊆ carrier R
        using fs2-sub subset by blast
      have fs2-prod-carr: foldr (⊗_R) fs2 1_R ∈ carrier R
        by (rule multlist-closed[OF fs2-carr])
      have b = q ⊗_R b1
        by (rule hb1)
      also have ... = q ⊗_R (foldr (⊗_R) fs2 1_R ⊗_R b')
        using hb by simp
      also have ... = (q ⊗_R foldr (⊗_R) fs2 1_R) ⊗_R b'
        using q-carr fs2-prod-carr b'-in by (simp add: m-assoc)
      also have ... = foldr (⊗_R) (q # fs2) 1_R ⊗_R b'
        by simp
      finally show ?thesis .
    qed
    show p = a' ⊗_R b'
      by (rule hp')
  qed
```

qed
qed

lemma *localization-irreducible-of-irreducible-prime-generated*:
assumes *hS*: *ring-prime-generated R S*
and *loc-dom*: *domain rec-rng-of-frac*
and *p-in*: $p \in$ *carrier R*
and *hp*: *ring-irreducible*$_R$ *p*
and *havoid*: *ring-avoids R S p*
shows *ring-irreducible*$_{\textit{rec-rng-of-frac}}$ (*rng-to-rng-of-frac p*)
proof −
interpret *L*: *domain rec-rng-of-frac*
by (*rule loc-dom*)
have *zero-notin*: $\mathbf{0} \notin S$
using *zero-notin-submonoid-of-prime-generated*[*OF hS*] **.**
have *map-p-in*: *rng-to-rng-of-frac p* $\in$ *carrier rec-rng-of-frac*
by (*rule ring-hom-closed*[*OF rng-to-rng-of-frac-is-ring-hom p-in*])
have *map-p-nz*: *rng-to-rng-of-frac p* $\neq \mathbf{0}_{\textit{rec-rng-of-frac}}$
using *map-eq-zero-iff*[*OF p-in zero-notin no-zero-divisors*]
ring-irreducibleE(*1*)[*OF p-in hp*]
by *blast*
have *map-p-not-unit*: *rng-to-rng-of-frac p* $\notin$ *Units rec-rng-of-frac*
using *map-irreducible-not-unit-of-zero-notin*[*OF zero-notin loc-dom p-in hp havoid*] **by** *blast*
show *?thesis*
proof (*rule L.ring-irreducibleI*)
show *rng-to-rng-of-frac p* $\in$ *carrier rec-rng-of-frac* − $\{\mathbf{0}_{\textit{rec-rng-of-frac}}\}$
using *map-p-in map-p-nz* **by** *blast*
show *rng-to-rng-of-frac p* $\notin$ *Units rec-rng-of-frac*
by (*rule map-p-not-unit*)
fix *x y*
assume *x-in*: $x \in$ *carrier rec-rng-of-frac*
and *y-in*: $y \in$ *carrier rec-rng-of-frac*
and *xy*: *rng-to-rng-of-frac p* $= x \otimes_{\textit{rec-rng-of-frac}} y$
from *fraction-surj*[*OF x-in*] **obtain** *a* **where**
a-in: $a \in$ *carrier R*
and *hs*: $\exists\, s \in S.\ x = (a \mid_{rel} s)$
by *blast*
from *hs* **obtain** *s* **where**
s-in: $s \in S$
and *x-def*: $x = (a \mid_{rel} s)$
by *blast*
from *fraction-surj*[*OF y-in*] **obtain** *b* **where**
b-in: $b \in$ *carrier R*
and *ht*: $\exists\, t \in S.\ y = (b \mid_{rel} t)$
by *blast*
from *ht* **obtain** *t* **where**
t-in: $t \in S$
and *y-def*: $y = (b \mid_{rel} t)$

by *blast*
have *s-carr*: $s \in$ *carrier* R
using *s-in subset rev-subsetD* **by** *blast*
have *t-carr*: $t \in$ *carrier* R
using *t-in subset rev-subsetD* **by** *blast*
have *st-in*: $s \otimes_R t \in S$
using *s-in t-in* **by** (*simp add*: *m-closed*)
obtain *fs* **where**
fs-sub: *set fs* $\subseteq S$
and *hf*: $\forall\, q \in$ *set fs*. *ring-prime*$_R$ *q*
and *hprod*: *foldr* $(\otimes_R)$ *fs* $\mathbf{1}_R = s \otimes_R t$
using *ring-prime-generatedE*[*OF hS st-in*] **by** *blast*
have *p-rel*: $(p, \mathbf{1}) \in$ *carrier rel*
using *p-in one-closed* **by** (*simp add*: *rel-def*)
have *as-rel*: $(a, s) \in$ *carrier rel*
using *a-in s-in* **by** (*simp add*: *rel-def*)
have *bt-rel*: $(b, t) \in$ *carrier rel*
using *b-in t-in* **by** (*simp add*: *rel-def*)
have *ab-rel*: $(a \otimes_R b, s \otimes_R t) \in$ *carrier rel*
using *a-in b-in s-in t-in* **by** (*simp add*: *rel-def*)
have *frac-prod*: $x \otimes_{\textit{rec-rng-of-frac}} y = (a \otimes_R b \mid_{\textit{rel}} s \otimes_R t)$
using *fraction-mult-rep*[*OF as-rel bt-rel*] *x-def y-def* **by** *simp*
have *eq-frac*: *rng-to-rng-of-frac* $p = (a \otimes_R b \mid_{\textit{rel}} s \otimes_R t)$
using *xy frac-prod* **by** (*simp add*: *rng-to-rng-of-frac-def*)
have *hEq-cross-raw*: $(s \otimes_R t) \otimes_R p =$
$\mathbf{1}_R \otimes_R (a \otimes_R b)$
using *fraction-eq-iff-cross-multiply*[*OF p-rel ab-rel zero-notin no-zero-divisors*]
eq-frac
unfolding *rng-to-rng-of-frac-def* **by** *simp*
have *hEq-cross*: $(s \otimes_R t) \otimes_R p = a \otimes_R b$
using *hEq-cross-raw a-in b-in* **by** *simp*
have *hEq*: $p \otimes_R$ *foldr* $(\otimes_R)$ *fs* $\mathbf{1}_R = a \otimes_R b$
using *hEq-cross hprod m-comm p-in s-carr t-carr* **by** *auto*
obtain *fs1 fs2* a' b' **where**
part: *fs* $<\sim\sim>$ *fs1* @ *fs2*
and *fs1-sub*: *set fs1* $\subseteq S$
and *fs1-prime*: $\forall\, q \in$ *set fs1*. *ring-prime*$_R$ *q*
and *fs2-sub*: *set fs2* $\subseteq S$
and *fs2-prime*: $\forall\, q \in$ *set fs2*. *ring-prime*$_R$ *q*
and a'*-in*: $a' \in$ *carrier* R
and b'*-in*: $b' \in$ *carrier* R
and *ha*: $a =$ *foldr* $(\otimes_R)$ *fs1* $\mathbf{1}_R \otimes_R a'$
and *hb*: $b =$ *foldr* $(\otimes_R)$ *fs2* $\mathbf{1}_R \otimes_R b'$
and *hpab*: $p = a' \otimes_R b'$
using *split-prime-factors-of-mul-eq*[*OF fs-sub hf p-in a-in b-in hEq*] **by** *blast*
have *fs1-carr*: *set fs1* $\subseteq$ *carrier* R
using *fs1-sub subset* **by** *blast*
have *fs2-carr*: *set fs2* $\subseteq$ *carrier* R
using *fs2-sub subset* **by** *blast*

```
    have prod1-in: foldr (⊗_R) fs1 1_R ∈ carrier R
      by (rule multlist-closed[OF fs1-carr])
    have prod2-in: foldr (⊗_R) fs2 1_R ∈ carrier R
      by (rule multlist-closed[OF fs2-carr])
    have prod1-mem: foldr (⊗_R) fs1 1_R ∈ S
      using multlist-mem-submonoid[OF fs1-sub] .
    have prod2-mem: foldr (⊗_R) fs2 1_R ∈ S
      using multlist-mem-submonoid[OF fs2-sub] .
    have a's-rel: (a', s) ∈ carrier rel
      using a'-in s-in by (simp add: rel-def)
    have b't-rel: (b', t) ∈ carrier rel
      using b'-in t-in by (simp add: rel-def)
    have x-eq:
       x = rng-to-rng-of-frac (foldr (⊗_R) fs1 1_R) ⊗_rec-rng-of-frac (a' |_rel s)
      by (simp add: a's-rel ha map-mul-fraction prod1-in x-def)
    have y-eq:
       y = rng-to-rng-of-frac (foldr (⊗_R) fs2 1_R) ⊗_rec-rng-of-frac (b' |_rel t)
      using b't-rel hb map-mul-fraction prod2-in y-def by presburger
    have a'-unit-or-b'-unit: a' ∈ Units R ∨ b' ∈ Units R
      using ring-irreducibleE(5)[OF p-in hp a'-in b'-in] hpab by blast
    then show x ∈ Units rec-rng-of-frac ∨ y ∈ Units rec-rng-of-frac
    proof
      assume a'-unit: a' ∈ Units R
     have prod1-unit: rng-to-rng-of-frac (foldr (⊗_R) fs1 1_R) ∈ Units rec-rng-of-frac
        using map-submonoid-elem-is-unit[OF prod1-mem] .
      have frac1-unit: (a' |_rel s) ∈ Units rec-rng-of-frac
        by (rule fraction-unit-numerator-is-unit[OF a'-unit s-in])
      show ?thesis
        using x-eq prod1-unit frac1-unit by simp
    next
      assume b'-unit: b' ∈ Units R
     have prod2-unit: rng-to-rng-of-frac (foldr (⊗_R) fs2 1_R) ∈ Units rec-rng-of-frac
        using map-submonoid-elem-is-unit[OF prod2-mem] .
      have frac2-unit: (b' |_rel t) ∈ Units rec-rng-of-frac
        by (rule fraction-unit-numerator-is-unit[OF b'-unit t-in])
      show ?thesis
        using y-eq prod2-unit frac2-unit by simp
    qed
  qed
qed

lemma prime-of-localization-prime-prime-generated:
  assumes hS: ring-prime-generated R S
    and p-in: p ∈ carrier R
    and hp: ring-irreducible_R p
    and havoid: ring-avoids R S p
    and hploc: ring-prime_rec-rng-of-frac (rng-to-rng-of-frac p)
  shows ring-prime_R p
proof −
```

```
  have p-nz: p ≠ 0
    using ring-irreducibleE(1)[OF p-in hp] .
  have hloc-prime: prime rec-rng-of-frac (rng-to-rng-of-frac p)
    using hploc unfolding ring-prime-def by simp
  have p-prime: prime R p
  proof (rule primeI)
    show p ∉ Units R
      using ring-irreducibleE(4)[OF p-in hp] .
    fix a b
    assume a-in: a ∈ carrier R
      and b-in: b ∈ carrier R
      and hdiv: p dividesR (a ⊗R b)
    then obtain d where d-in: d ∈ carrier R and hfactor: a ⊗R b = p ⊗R d
      unfolding factor-def by blast
    then have map-p-dvd-ab:
        rng-to-rng-of-frac p dividesrec-rng-of-frac
          (rng-to-rng-of-frac a ⊗rec-rng-of-frac rng-to-rng-of-frac b)
     by (smt (verit, del-insts) a-in b-in dividesI p-in ring-hom-closed ring-hom-mult

           rng-to-rng-of-frac-is-ring-hom)
    have map-a-in: rng-to-rng-of-frac a ∈ carrier rec-rng-of-frac
      using ring-hom-closed[OF rng-to-rng-of-frac-is-ring-hom a-in] .
    have map-b-in: rng-to-rng-of-frac b ∈ carrier rec-rng-of-frac
      using ring-hom-closed[OF rng-to-rng-of-frac-is-ring-hom b-in] .
    have map-p-dvd-a-or-b:
        rng-to-rng-of-frac p dividesrec-rng-of-frac rng-to-rng-of-frac a ∨
         rng-to-rng-of-frac p dividesrec-rng-of-frac rng-to-rng-of-frac b
      by (meson hloc-prime map-a-in map-b-in map-p-dvd-ab primeE)
    then show p dividesR a ∨ p dividesR b
      using dvd-of-localization-dvd-prime-generated[OF hS p-in a-in hp havoid]
        dvd-of-localization-dvd-prime-generated[OF hS p-in b-in hp havoid]
      by blast
  qed
  from p-nz p-prime show ?thesis
    by (rule ring-primeI)
qed

lemma nagata-key-lemma-prime-generated:
  assumes hS: ring-prime-generated R S
    and loc-fd: factorial-domain rec-rng-of-frac
    and p-in: p ∈ carrier R
    and hp: ring-irreducibleR p
  shows ring-primeR p
proof (cases ∃ s ∈ S. p dividesR s)
  case True
  then obtain s where s-in: s ∈ S and p-dvd-s: p dividesR s
    by blast
  show ?thesis
      using prime-of-irreducible-of-dvd-mem-prime-generated[OF hS p-in hp s-in
```

p-dvd-s] .
next
case *False*
interpret *Lfd*: *factorial-domain rec-rng-of-frac*
by (*rule loc-fd*)
interpret *L*: *domain rec-rng-of-frac*
by (*rule Lfd.domain-axioms*)
have *havoid*: *ring-avoids R S p*
using *False* **unfolding** *ring-avoids-def* **by** *blast*
have *loc-irreducible*: $\mathit{ring\text{-}irreducible}_{\mathit{rec\text{-}rng\text{-}of\text{-}frac}}$ (*rng-to-rng-of-frac p*)
using *localization-irreducible-of-irreducible-prime-generated*[*OF hS Lfd.domain-axioms p-in hp havoid*] .
have *loc-irred-mult*: *irreducible* (*mult-of rec-rng-of-frac*) (*rng-to-rng-of-frac p*)
using *Lfd.ring-irreducibleE*(*3*)[*OF ring-hom-closed*[*OF rng-to-rng-of-frac-is-ring-hom p-in*] *loc-irreducible*] .
have *loc-in-nz*: *rng-to-rng-of-frac p* ∈ *carrier rec-rng-of-frac* − $\{\mathbf{0}_{\mathit{rec\text{-}rng\text{-}of\text{-}frac}}\}$
using *ring-hom-closed*[*OF rng-to-rng-of-frac-is-ring-hom p-in*]
Lfd.ring-irreducibleE(*1*)[*OF ring-hom-closed*[*OF rng-to-rng-of-frac-is-ring-hom p-in*] *loc-irreducible*]
by *blast*
have *loc-in-mult*: *rng-to-rng-of-frac p* ∈ *carrier* (*mult-of rec-rng-of-frac*)
using *loc-in-nz* **by** *simp*
have *loc-prime-mult*: *prime* (*mult-of rec-rng-of-frac*) (*rng-to-rng-of-frac p*)
by (*rule factorial-monoid.irreducible-prime*[*OF Lfd.factorial-monoid-axioms loc-irred-mult loc-in-mult*])
have *loc-ring-prime*: $\mathit{ring\text{-}prime}_{\mathit{rec\text{-}rng\text{-}of\text{-}frac}}$ (*rng-to-rng-of-frac p*)
using *Lfd.ring-primeI'*[*OF loc-in-nz loc-prime-mult*] .
show *?thesis*
using *prime-of-localization-prime-prime-generated*[*OF hS p-in hp havoid loc-ring-prime*]
.
qed

lemma *nagata-theorem*:
assumes *noeth*: *noetherian-domain R*
and *hS*: *ring-prime-generated R S*
and *loc-fd*: *factorial-domain rec-rng-of-frac*
shows *factorial-domain R*
proof −
interpret *N*: *noetherian-domain R*
by (*rule noeth*)
interpret *PC*: *primeness-condition-monoid mult-of R*
proof
fix *a*
assume *a-in*: *a* ∈ *carrier* (*mult-of R*)
and *a-irred*: *irreducible* (*mult-of R*) *a*
have *a-in-R*: *a* ∈ *carrier R* − {**0**}
using *a-in* **by** *simp*
have *a-ring-irred*: $\mathit{ring\text{-}irreducible}_R$ *a*
using *ring-irreducibleI'*[*OF a-in-R a-irred*] .

```
    have a-prime: ring-prime_R a
      using nagata-key-lemma-prime-generated[OF hS loc-fd DiffD1[OF a-in-R]
a-ring-irred] .
    have a-prime-mult: prime (mult-of R) a
      using ring-primeE(2)[OF DiffD1[OF a-in-R] a-prime] .
    show prime (mult-of R) a
      by (rule a-prime-mult)
  qed
  have wfactors-exist-mult:
      ⋀a. ⟦a ∈ carrier (mult-of R); a ∉ Units (mult-of R)⟧ ⟹
        ∃fs. set fs ⊆ carrier (mult-of R) ∧ wfactors (mult-of R) fs a
  proof −
    fix a
    assume a-in: a ∈ carrier (mult-of R)
      and a-not-unit: a ∉ Units (mult-of R)
    have a-in-R: a ∈ carrier R − {0}
      using a-in by simp
    have a-not-unit-R: a ∉ Units R
      using a-not-unit by simp
    show ∃fs. set fs ⊆ carrier (mult-of R) ∧ wfactors (mult-of R) fs a
      using N.factorization-property[OF a-in-R a-not-unit-R] .
  qed
  have factorial-mult: factorial-monoid (mult-of R)
   by (rule mult-of.factorial-monoidI[OF wfactors-exist-mult PC.wfactors-unique])
  show ?thesis
    unfolding factorial-domain-def using domain-axioms factorial-mult by simp
qed

lemma nagata-theorem-of-prime-or-unit:
  assumes noeth: noetherian-domain R
    and hS: ⋀s. s ∈ S ⟹ ring-prime_R s ∨ s ∈ Units R
    and loc-fd: factorial-domain rec-rng-of-frac
  shows factorial-domain R
proof −
  interpret N: noetherian-domain R
    by (rule noeth)
  interpret PC: primeness-condition-monoid mult-of R
  proof
    fix a
    assume a-in: a ∈ carrier (mult-of R)
      and a-irred: irreducible (mult-of R) a
    have a-in-R: a ∈ carrier R − {0}
      using a-in by simp
    have a-ring-irred: ring-irreducible_R a
      using ring-irreducibleI'[OF a-in-R a-irred] .
    have a-ring-prime: ring-prime_R a
      using nagata-key-lemma[OF hS loc-fd DiffD1[OF a-in-R] a-ring-irred] .
    show prime (mult-of R) a
      using ring-primeE(2)[OF DiffD1[OF a-in-R] a-ring-prime] .
```

qed
have *wfactors-exist-mult*:
$\bigwedge a.\ [\![a \in carrier\ (mult\text{-}of\ R);\ a \notin Units\ (mult\text{-}of\ R)]\!] \Longrightarrow$
$\exists fs.\ set\ fs \subseteq carrier\ (mult\text{-}of\ R) \wedge wfactors\ (mult\text{-}of\ R)\ fs\ a$
proof −
fix *a*
assume *a-in*: $a \in carrier\ (mult\text{-}of\ R)$
and *a-not-unit*: $a \notin Units\ (mult\text{-}of\ R)$
have *a-in-R*: $a \in carrier\ R - \{\mathbf{0}\}$
using *a-in* **by** *simp*
have *a-not-unit-R*: $a \notin Units\ R$
using *a-not-unit* **by** *simp*
show $\exists fs.\ set\ fs \subseteq carrier\ (mult\text{-}of\ R) \wedge wfactors\ (mult\text{-}of\ R)\ fs\ a$
using *N.factorization-property*[*OF a-in-R a-not-unit-R*] **.**
qed
have *factorial-mult*: *factorial-monoid* (*mult-of R*)
by (*rule mult-of.factorial-monoidI*[*OF wfactors-exist-mult PC.wfactors-unique*])
show *?thesis*
unfolding *factorial-domain-def* **using** *domain-axioms factorial-mult* **by** *simp*
qed

lemma *nagata-theorem-of-prime-generators*:
assumes *noeth*: *noetherian-domain R*
and *S-eq*: $S = ring\text{-}mult\text{-}submonoid\text{-}closure\ R\ A$
and *A-sub*: $A \subseteq carrier\ R$
and *hprime*: $\bigwedge q.\ q \in A \Longrightarrow ring\text{-}prime_R\ q$
and *loc-fd*: *factorial-domain rec-rng-of-frac*
shows *factorial-domain R*
proof −
have *hS*: *ring-prime-generated R S*
unfolding *S-eq*
by (*rule ring-prime-generated-mult-submonoid-closure*[*OF ring-axioms A-sub hprime*])
show *?thesis*
by (*rule nagata-theorem*[*OF noeth hS loc-fd*])
qed

lemma *nagata-theorem-of-finite-prime-generators*:
assumes *noeth*: *noetherian-domain R*
and *finA*: *finite A*
and *S-eq*: $S = ring\text{-}mult\text{-}submonoid\text{-}closure\ R\ A$
and *A-sub*: $A \subseteq carrier\ R$
and *hprime*: $\bigwedge q.\ q \in A \Longrightarrow ring\text{-}prime_R\ q$
and *loc-fd*: *factorial-domain rec-rng-of-frac*
shows *factorial-domain R*
using *finA nagata-theorem-of-prime-generators*[*OF noeth S-eq A-sub hprime loc-fd*]
by *blast*

end

end
theory *Polynomial-Applications*
imports
Nagata-Lemmas
HOL−Algebra.Polynomial-Divisibility
begin

5 Polynomial applications

This theory packages the first concrete application layer on top of the record-based Nagata descent theorem. The present results isolate the abstract "localize away X" step for HOL-Algebra polynomial rings, together with the standard field-coefficient specialization in which X is prime by the degree-one irreducibility criterion.

context *domain*
begin

lemma *polynomial-prime-X*:
assumes *K*: *subfield K R*
shows *ring-prime*$_{K[X]}$ *X*
proof −
have *X-in*: *X* ∈ *carrier* (*K*[*X*])
using *var-closed*(*1*)[*OF subfieldE*(*1*)[*OF K*]] **.**
have *X-irred*: *pirreducible K X*
using *degree-one-imp-pirreducible*[*OF K X-in*]
by (*simp add*: *var-def*)
show *?thesis*
using *pprime-iff-pirreducible*[*OF K X-in*] *X-irred* **by** *simp*
qed

lemma *polynomial-prime-generated-powers-X*:
assumes *K*: *subring K R*
and *hX*: *ring-prime*$_{K[X]}$ *X*
shows *ring-prime-generated* (*K*[*X*]) (*ring-powers-set* (*K*[*X*]) *X*)
using *ring-prime-generated-powers-set*[*OF univ-poly-is-ring*[*OF K*] *var-closed*(*1*)[*OF K*] *hX*] **.**

end

locale *polynomial-away-X-localization* =
fixes *R* (**structure**) **and** *P* (**structure**) **and** *S* **and** *K*
assumes *poly-axioms*: *nagata-localization P S*
and *base-axioms*: *domain R*
and *P-eq*: *P* = *K*[*X*]
and *S-eq*: *S* = *ring-powers-set* (*K*[*X*]) *X*
begin

abbreviation *loc-ring* **where** *loc-ring* ≡ *eq-obj-rng-of-frac.rec-rng-of-frac P S*

Once a localization of K[X] at the powers of X has been fixed, Nagata's theorem reduces factoriality of K[X] to factoriality of that localization, provided X is prime.

lemma *polynomial-factorial-of-localized-X-factorial*:
assumes *K*: *subring K R*
and *noeth*: *noetherian-domain* (*K*[*X*])
and *hX*: *ring-prime*$_{K[X]}$ *X*
and *loc-fd*: *factorial-domain loc-ring*
shows *factorial-domain* (*K*[*X*])
proof −
have *noethP*: *noetherian-domain P*
using *noeth* **unfolding** *P-eq* **.**
have *hXP*: *ring-prime*$_P$ *X*
using *hX* **unfolding** *P-eq* **.**
have *hS*: *ring-prime-generated P S*
unfolding *P-eq S-eq*
by (*rule domain.polynomial-prime-generated-powers-X*[*OF base-axioms K hX*])
have *fdP*: *factorial-domain P*
by (*rule nagata-localization.nagata-theorem*[*OF poly-axioms noethP hS loc-fd*])
show *?thesis*
using *fdP* **unfolding** *P-eq* **.**
qed

lemma *polynomial-factorial-of-localized-X-factorial-field*:
assumes *K*: *subfield K R*
and *noeth*: *noetherian-domain* (*K*[*X*])
and *loc-fd*: *factorial-domain loc-ring*
shows *factorial-domain* (*K*[*X*])
proof −
have *hX*: *ring-prime*$_{K[X]}$ *X*
by (*rule domain.polynomial-prime-X*[*OF base-axioms K*])
show *?thesis*
by (*rule polynomial-factorial-of-localized-X-factorial*[
OF subfieldE(*1*)[*OF K*] *noeth hX loc-fd*])
qed

end

end
theory *Fraction-Field-Applications*
imports
Nagata-Lemmas
Polynomial-Applications
HOL−Algebra.Polynomial-Divisibility
begin

6 Constant-prime localization applications

This theory packages the constant-prime specialization of the polynomial application layer at the same level of abstraction as *Polynomial-Applications*: it specializes Nagata's theorem to multiplicative sets generated by constant prime polynomials and isolates the corresponding descent step for polynomial rings.

context *domain*
begin

lemma *polynomial-prime-generated-constant-closure*:
  assumes *Ksub*: *subring K R*
    and *A-sub*: $A \subseteq$ *carrier* $(R \langle\!|\mathit{carrier} := K|\!\rangle)$
    and *hprime*: $\bigwedge q.\ q \in A \Longrightarrow \mathit{ring\text{-}prime}_{K[X]}$ (*poly-of-const q*)
  shows
    ring-prime-generated $(K[X])$
      (*ring-mult-submonoid-closure* $(K[X])$ (*poly-of-const* ‘ *A*))
proof (*rule ring-prime-generated-mult-submonoid-closure*[*OF univ-poly-is-ring*[*OF Ksub*]])
  show *poly-of-const* ‘ *A* $\subseteq$ *carrier* $(K[X])$
  proof
    fix *p*
    assume *p-in*: $p \in$ *poly-of-const* ‘ *A*
    then obtain *q* **where** *q-in*: $q \in A$ **and** *p-def*: $p =$ *poly-of-const q*
      by *blast*
    have *qK*: $q \in$ *carrier* $(R \langle\!|\mathit{carrier} := K|\!\rangle)$
      using *A-sub q-in* **by** *blast*
    show $p \in$ *carrier* $(K[X])$
     unfolding *p-def* **by** (*rule ring-hom-closed*[*OF canonical-embedding-is-hom*[*OF Ksub*] *qK*])
  qed
  fix *p*
  assume $p \in$ *poly-of-const* ‘ *A*
  then obtain *q* **where** *q-in*: $q \in A$ **and** *p-def*: $p =$ *poly-of-const q*
    by *blast*
  show $\mathit{ring\text{-}prime}_{K[X]}$ *p*
    unfolding *p-def* **by** (*rule hprime*[*OF q-in*])
qed

end

locale *polynomial-constant-prime-localization* =
  fixes *R* (**structure**) **and** *P* (**structure**) **and** *S* **and** *K* :: *'a set* **and** *A* :: *'a set*
  assumes *poly-axioms*: *nagata-localization P S*
    and *base-axioms*: *domain R*
    and *P-eq*: $P = K[X]$
   and *S-eq*: $S =$ *ring-mult-submonoid-closure* $(K[X])$ (*ring.poly-of-const* $(R \langle\!|\mathit{carrier} := K|\!\rangle)$ ‘ *A*)

begin

abbreviation *const-poly* **where**
const-poly ≡ *ring.poly-of-const* (*R* (|*carrier* := *K*|))

abbreviation *loc-ring* **where** *loc-ring* ≡ *eq-obj-rng-of-frac.rec-rng-of-frac P S*

Once a localization of $K[X]$ at a constant-prime closure has been fixed, Nagata's theorem immediately reduces factoriality of $K[X]$ to factoriality of that localization.

lemma *polynomial-factorial-of-localized-constant-primes-factorial*:
assumes *Ksub*: *subring K R*
and *A-sub*: *A* ⊆ *carrier* (*R* (|*carrier* := *K*|))
and *noeth*: *noetherian-domain* (*K*[*X*])
and *hprime*: ⋀*q*. *q* ∈ *A* ⟹ *ring-prime*$_{K[X]}$ (*const-poly q*)
and *loc-fd*: *factorial-domain loc-ring*
shows *factorial-domain* (*K*[*X*])
proof −
interpret *base*: *domain R*
by (*rule base-axioms*)
interpret *K-ring*: *ring R* (|*carrier* := *K*|)
by (*rule base.subring-is-ring*[*OF Ksub*])
have *noethP*: *noetherian-domain P*
using *noeth* **unfolding** *P-eq* **.**
have *hprimeP*: ⋀*q*. *q* ∈ *A* ⟹ *ring-prime*$_P$ (*const-poly q*)
using *P-eq hprime* **by** *blast*
have *S-closure-eq*: *S* = *ring-mult-submonoid-closure P* (*const-poly* ‘ *A*)
using *P-eq S-eq* **by** *blast*
have *Aimg-sub*: *const-poly* ‘ *A* ⊆ *carrier P*
proof
fix *p*
assume *p-in*: *p* ∈ *const-poly* ‘ *A*
then obtain *q* **where** *q-in*: *q* ∈ *A* **and** *p-def*: *p* = *const-poly q*
by *blast*
have *qK*: *q* ∈ *carrier* (*R* (|*carrier* := *K*|))
using *A-sub q-in* **by** *blast*
have *q-poly-in-base*: *base.poly-of-const q* ∈ *carrier* (*K*[*X*])
using *ring-hom-memE*(*1*)[*OF base.canonical-embedding-is-hom*[*OF Ksub*] *qK*]
by *simp*
have *p-def-base*: *p* = *base.poly-of-const q*
using *p-def fun-cong*[*OF base.poly-of-const-consistent*[*OF Ksub*], *of q*] **by** *simp*
show *p* ∈ *carrier P*
unfolding *P-eq*
using *p-def-base q-poly-in-base* **by** *simp*
qed
have *hprime-img*: ⋀*p*. *p* ∈ *const-poly* ‘ *A* ⟹ *ring-prime*$_P$ *p*
using *hprimeP* **by** *blast*
have *fdP*: *factorial-domain P*
by (*rule nagata-localization.nagata-theorem-of-prime-generators*[

OF poly-axioms noethP S-closure-eq Aimg-sub hprime-img loc-fd])
show *?thesis*
using *fdP* **unfolding** *P-eq* **.**
qed

end

end

theory *Nagata-Factoriality*
imports
Prime-Generated
Nagata-Lemmas
Polynomial-Applications
Fraction-Field-Applications
HOL−Computational-Algebra.Polynomial
begin

7 Nagata-factoriality scaffolding

Nagata's factoriality theorem descends unique factorization from a localization to the base ring under a prime-generated hypothesis on the multiplicative set. The present entry now packages the prime-generated core, a wrapper layer over the AFP localization entry, and a record-based HOL-Algebra proof of both the prime-generated and prime-or-unit descent variants in *Nagata-Lemmas*. That theory now also packages theorem-level entry points for submonoid closures generated by prime families, including a finite-generator wrapper. The additional theories *Polynomial-Applications* and *Fraction-Field-Applications* package abstract polynomial applications for localization away X and for localizations generated by constant prime polynomials, cf. Nagata and Samuel.[2] [3]

lemma *prime-generated-constant-prime-polynomials*:
fixes $A :: 'a :: semidom\ set$
assumes $\bigwedge c.\ c \in A \Longrightarrow prime\text{-}elem\ c$
shows *prime-generated* (*mult-submonoid-closure* ((λ*c*. [:*c*:]) ‘ *A*))
proof (*rule prime-generated-mult-submonoid-closure*)
fix *q*
assume $q \in (\lambda c.\ [:c:])\ `\ A$
then obtain *c* **where** $c \in A$ **and** *q-def*: $q = [:c:]$
by *blast*
from *assms*[*OF* ‹$c \in A$›] **show** *prime-elem q*
unfolding *q-def* **by** (*rule lift-prime-elem-poly*)
qed

corollary *zero-notin-constant-prime-polynomial-closure*:
fixes $A :: 'a :: idom\ set$

assumes $\bigwedge c.\ c \in A \Longrightarrow$ *prime-elem* c
shows $0 \notin$ *mult-submonoid-closure* $((\lambda c.\ [:c:])\ `\ A)$
proof −
have *prime-generated* (*mult-submonoid-closure* $((\lambda c.\ [:c:])\ `\ A)$)
using *assms* **by** (*rule prime-generated-constant-prime-polynomials*)
then show *?thesis*
by (*rule zero-notin-prime-generated*[**where** $S =$ *mult-submonoid-closure* $((\lambda c.\ [:c:])\ `\ A)$])
qed

The last corollary isolates one of the key configurations in the constant-prime polynomial application: the multiplicative set generated by constant prime polynomials is prime-generated and therefore avoids zero.

Separately, the theory *Localization-Interface* exposes an Isabelle/HOL wrapper around the AFP localization construction with lemmas for representative equality, numerator-denominator surjectivity, denominator rescaling, cross-multiplication in the domain case, units coming from both the multiplicative set and base-ring units, and injectivity of the canonical localization map under the usual domain hypotheses. On top of that, *Nagata-Lemmas* proves the descent lemmas needed for Nagata's theorem, together with a record-based multiplicative-closure API

$[\![$*ring ?R*; *?A* $\subseteq$ *carrier ?R*; $\bigwedge q.\ q \in$ *?A* $\Longrightarrow$ *ring-prime*$_{?R}$ $q]\!]$
$\Longrightarrow$ *ring-prime-generated ?R* (*ring-mult-submonoid-closure ?R ?A*)

for the constant-prime submonoids used in the fraction-field route, culminating in theorem statements

$[\![$*nagata-localization ?R ?S*; *noetherian-domain ?R*;
ring-prime-generated ?R ?S;
factorial-domain (*eq-obj-rng-of-frac.rec-rng-of-frac ?R ?S*)$]\!]$
$\Longrightarrow$ *factorial-domain ?R*

and

$[\![$*nagata-localization ?R ?S*; *noetherian-domain ?R*;
$\bigwedge s.\ s \in$ *?S* $\Longrightarrow$ *ring-prime*$_{?R}$ $s \vee s \in$ *Units ?R*;
factorial-domain (*eq-obj-rng-of-frac.rec-rng-of-frac ?R ?S*)$]\!]$
$\Longrightarrow$ *factorial-domain ?R*

together with the prime-generator closure wrappers

$[\![$*nagata-localization ?R ?S*; *noetherian-domain ?R*;
?S = *ring-mult-submonoid-closure ?R ?A*; *?A* $\subseteq$ *carrier ?R*;
$\bigwedge q.\ q \in$ *?A* $\Longrightarrow$ *ring-prime*$_{?R}$ q;
factorial-domain (*eq-obj-rng-of-frac.rec-rng-of-frac ?R ?S*)$]\!]$
$\Longrightarrow$ *factorial-domain ?R*

and

⟦*nagata-localization ?R ?S*; *noetherian-domain ?R*; *finite ?A*;
?S = *ring-mult-submonoid-closure ?R ?A*; *?A* ⊆ *carrier ?R*;
⋀*q*. *q* ∈ *?A* ⟹ *ring-prime*$_{?R}$ *q*;
factorial-domain (*eq-obj-rng-of-frac.rec-rng-of-frac ?R ?S*)⟧
⟹ *factorial-domain ?R*

. The theory *Polynomial-Applications* then specializes this framework to the polynomial ring case by proving

⟦*domain ?R*; *subfield ?K ?R*⟧ ⟹ *pprime*$_{?R}$ *?K X*$_{?R}$

over fields and the abstract away-X descent theorem

⟦*polynomial-away-X-localization ?R ?P ?S ?K*; *subring ?K ?R*;
noetherian-domain (*?K* [*X*]$_{?R}$); *pprime*$_{?R}$ *?K X*$_{?R}$;
factorial-domain (*eq-obj-rng-of-frac.rec-rng-of-frac ?P ?S*)⟧
⟹ *factorial-domain* (*?K* [*X*]$_{?R}$)

, while *Fraction-Field-Applications* packages the companion constant-prime closure theorem

⟦*polynomial-constant-prime-localization ?R ?P ?S ?K ?A*; *subring ?K ?R*;
?A ⊆ *carrier* (*?R*(|*carrier* := *?K*|)); *noetherian-domain* (*?K* [*X*]$_{?R}$);
⋀*q*. *q* ∈ *?A* ⟹ *pprime*$_{?R}$ *?K* (*ring.poly-of-const* (*?R*(|*carrier* := *?K*|)) *q*);
factorial-domain (*eq-obj-rng-of-frac.rec-rng-of-frac ?P ?S*)⟧
⟹ *factorial-domain* (*?K* [*X*]$_{?R}$)

.

end

References

[1] H. Matsumura. *Commutative Ring Theory*, volume 8 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1986.

[2] M. Nagata. *Local Rings*, volume 13 of *Interscience Tracts in Pure and Applied Mathematics*. Interscience Publishers, New York, 1962.

[3] P. Samuel. *Lectures on Unique Factorization Domains*, volume 30 of *Tata Institute of Fundamental Research Lectures on Mathematics and Physics*. Tata Institute of Fundamental Research, Bombay, 1964.