POLYNOM_1

---

*C polynom_1_begin

********** POLYNOM_1 **********

This theory contains the definitions of classes for constructing polynomial algebras.

The classes are:

1. Free abelian monoids:
   The monomials on a set of indeterminates can be characterized as a free abelian monoid.
2. Monoid copower:
   This class is regarded as a core class for both monomials and polynomials. Any monoid copower construction can be specialized to give a free abelian monoid, and generalized to give a free monoid algebra.
3. Free monoid algebras:
   The polynomials over a monoid of monomials and a commutative ring of coefficients can be characterized as a free monoid algebra.
4. Polynomial algebras:
   Elements of this class can be regarded as full implementations of polynomials.

These standard constructions are described in most algebra books (e.g. Lang).

*C polynom_1_summary

Class definitions for free abelian monoids, free abelian groups, monoid copowers, free monoid algebras and polynomial algebras.

*C polynom_1_intro

Class definitions given for free abelian monoids, free abelian groups, monoid copowers, and free monoid algebras. These classes are used as ADT's for functions of finite support with various domains and ranges.

A class definition for polynomial algebras serves as an ADT for implementations of multivariate polynomials.

*C free_abmonoid_com

==========

FREE ABELIAN MONOID CLASS

==========

*D free_abmonoid_df

FAbMon{<i:level>}(<S:S:*>)== free_abmonoid{<i>:l}(<S>)

FAbMon(<S:S:*>)== free_abmonoid{i:l}(<S>)

*A free_abmonoid

FAbMon(S) ==

  mon:AbMon

  × inj:(|S| ⟶ |mon|)

  × (mon':AbMon ⟶ f':(|S| ⟶ |mon'|) ⟶ {!g:MonHom(mon,mon') | g o inj = f'})

*T free_abmonoid_wf 0 2 ∀S:DSet. FAbMon(S) ∈ 𝕌'

*D free_abmon_mon_df <f:free_abmon:E>.mon== free_abmon_mon{}(<f>)

*A free_abmon_mon f.mon == f.1

*T free_abmon_mon_wf 0 0 ∀S:DSet. ∀f:FAbMon(S). f.mon ∈ AbMon

*D free_abmon_inj_df <f:free_abmon:E>.inj== free_abmon_inj{}(<f>)

*A free_abmon_inj f.inj == f.2.1

*T free_abmon_inj_wf 0 0 ∀S:DSet. ∀f:FAbMon(S). f.inj ∈ |S| ⟶ |f.mon|

*D free_abmon_umap_df <f:free_abmon:E>.umap== free_abmon_umap{}(<f>)

*A free_abmon_umap f.umap == f.2.2

```
*T free_abmon_umap_wf     0 0
               ∀S:DSet. ∀f:FAbMon(S).
                 f.umap ∈ mon':AbMon
                          → f':(|S| → |mon'|)
                          → {!g:MonHom(f.mon,mon') | g o f.inj = f'}
*M create_free_abmonoid
               Class Declaration for: FAbMon(S)
                 Long Name: free_abmonoid
                 Short Name: free_abmon
                 Parameters:
                    S : DSet
                 Fields:
                    mon : AbMon
                    inj : |S| → |mon|
                    umap : mon':AbMon
                           → f':(|S| → |mon'|)
                           → {!g:MonHom(mon,mon') | g o inj = f'}
                 Universe: 𝕌'
*T free_abmon_umap_properties 3 4
               ∀S:DSet. ∀M:FAbMon(S). ∀N:AbMon. ∀p:|S| → |N|.
                 M.umap N p o M.inj = p ∧ (∀f:MonHom(M.mon,N). f o M.inj = p ⇒ f = M.umap N p)
*T free_abmon_umap_properties_1 2 3
               ∀S:DSet. ∀M:FAbMon(S). ∀N:AbMon. ∀p:|S| → |N|.  M.umap N p o M.inj = p
*T free_abmon_endomorph_is_id 3 2
               ∀S:DSet. ∀M:FAbMon(S). ∀f:MonHom(M.mon,M.mon).
                 f o M.inj = M.inj ⇒ f = Id{|M.mon|}
*T free_abmon_unique      3 5
               ∀S:DSet. ∀M,N:FAbMon(S).
                 ∃f:MonHom(M.mon,N.mon). ∃g:MonHom(N.mon,M.mon). InvFuns(|M.mon|;|N.mon|;f;g)
*T mk_fabmon              3 5
               ∀s:DSet. ∀g:AbMon. ∀i:|s| → |g|. ∀U:g':AbMon → (|s| → |g'|) → |g| → |g'|.
                 (∀g':AbMon. ∀f:|s| → |g'|.
                    IsMonHom{g,g'}(U g' f)
                    ∧ U g' f o i = f
                    ∧ (∀u:|g| → |g'|. IsMonHom{g,g'}(u) ⇒ u o i = f ⇒ u = U g' f))
                 ⇒ <g, i, U> ∈ FAbMon(s)
*C mcopower_sig_com
               ==================
               MONOID COPOWER ADT
               ==================
*D mcopower_sig_df
               MCopowerSig{<i:level>}(<s:s:*>;<g:g:*>)== mcopower_sig{<i>:l}(<s>; <g>)
               MCopowerSig(<s:s:*>;<g:g:*>)== mcopower_sig{i:l}(<s>; <g>)
*A mcopower_sig
               MCopowerSig(s;g) ==
                 mon:AbMon
                 × inj:(|s| → |g| → |mon|)
                 × (h:AbMon → (|s| → |g| → |h|) → |mon| → |h|)
*T mcopower_sig_wf        0 2 ∀s:DSet. ∀g:AbMon.  MCopowerSig(s;g) ∈ 𝕌'
*D mcopower_mon_df            <m:mcopower:E>.mon== mcopower_mon{}(<m>)
*A mcopower_mon               m.mon ==  m.1
*T mcopower_mon_wf        0 0 ∀s:DSet. ∀g:AbMon. ∀m:MCopowerSig(s;g).  m.mon ∈ AbMon
*D mcopower_inj_df            <m:mcopower:E>.inj== mcopower_inj{}(<m>)
*A mcopower_inj               m.inj ==  m.2.1
*T mcopower_inj_wf        0 0
               ∀s:DSet. ∀g:AbMon. ∀m:MCopowerSig(s;g).  m.inj ∈ |s| → |g| → |m.mon|
*D mcopower_umap_df           <m:mcopower:E>.umap== mcopower_umap{}(<m>)
```

```
*A mcopower_umap              m.umap ==  m.2.2
*T mcopower_umap_wf       0 0
              ∀s:DSet. ∀g:AbMon. ∀m:MCopowerSig(s;g).
                m.umap ∈ h:AbMon ⟶ (|s| ⟶ |g| ⟶ |h|) ⟶ |m.mon| ⟶ |h|
*M create_mcopower_sig
              Class Declaration for: MCopowerSig(s;g)
                Long Name: mcopower_sig
                Short Name: mcopower
                Parameters:
                   s : DSet
                   g : AbMon
                Fields:
                   mon : AbMon
                   inj : |s| ⟶ |g| ⟶ |mon|
                   umap : h:AbMon ⟶ (|s| ⟶ |g| ⟶ |h|) ⟶ |mon| ⟶ |h|
                Universe: 𝕌'
*D mcopower_df MCopower{<i:level>}(<s:s:*>;<g:g:*>)== mcopower{<i>:l}(<s>; <g>)
              MCopower(<s:s:*>;<g:g:*>)== mcopower{i:l}(<s>; <g>)
*A mcopower    MCopower(s;g) ==
                {c:MCopowerSig(s;g)|
                 (∀j:|s|. IsMonHom{g,c.mon}(c.inj j))
                 ∧ (∀h:AbMon. ∀f:|s| ⟶ MonHom(g,h).
                      c.umap h f = !v:|c.mon| ⟶ |h|
                                   IsMonHom{c.mon,h}(v) ∧ (∀j:|s|. f j = v o c.inj j))}
*T mcopower_wf            0 3 ∀s:DSet. ∀g:AbMon.  MCopower(s;g) ∈ 𝕌'
*M mcopower_ml_inc
              add_set_inclusion_info
                `mcopower` `mcopower_sig`
                AbSetDForInc Auto ;;
*T mcopower_properties    0 4
              ∀s:DSet. ∀g:AbMon. ∀c:MCopower(s;g).
                (∀j:|s|. IsMonHom{g,c.mon}(c.inj j))
                ∧ (∀h:AbMon. ∀f:|s| ⟶ MonHom(g,h).
                     c.umap h f = !v:|c.mon| ⟶ |h|
                                  IsMonHom{c.mon,h}(v) ∧ (∀j:|s|. f j = v o c.inj j))
*T mcopower_inj_is_hom    2 0
              ∀s:DSet. ∀g:AbMon. ∀c:MCopower(s;g). ∀j:|s|.  IsMonHom{g,c.mon}(c.inj j)
*T mcopower_umap_is_hom   2 1
              ∀s:DSet. ∀g:AbMon. ∀c:MCopower(s;g). ∀h:AbMon. ∀f:|s| ⟶ MonHom(g,h).
                IsMonHom{c.mon,h}(c.umap h f)
*T mcopower_umap_comm_tri 2 1
              ∀s:DSet. ∀g:AbMon. ∀c:MCopower(s;g). ∀h:AbMon. ∀f:|s| ⟶ MonHom(g,h). ∀j:|s|.
                f j = c.umap h f o c.inj j
*T mcopower_umap_comm_tri_a 1 2
              ∀s:DSet. ∀g:AbMon. ∀c:MCopower(s;g). ∀h:AbMon. ∀f:|s| ⟶ MonHom(g,h). ∀a:|g|.
              ∀j:|s|.
                f j a = c.umap h f (c.inj j a)
*T mcopower_umap_unique   2 3
              ∀s:DSet. ∀g:AbMon. ∀c:MCopower(s;g). ∀h:AbMon. ∀f:|s| ⟶ MonHom(g,h).
              ∀a':|c.mon| ⟶ |h|.
                IsMonHom{c.mon,h}(a') ⇒ (∀j:|s|. f j = a' o c.inj j) ⇒ a' = c.umap h f
*C fabmon_of_nat_mcp_com
              ======================================
              CREATION OF FREE ABMON FROM MON COPOWER
              ======================================
*D fabmon_of_nat_mcp_df       fabmon_of_nat_mcp(<F:F:*>)== fabmon_of_nat_mcp{}(<F>)
*A fabmon_of_nat_mcp
```

```
              fabmon_of_nat_mcp(m) ==
                <m.mon, λu.m.inj u zhgrp(1), λm',f.m.umap m' (λz,n.nat(n) · f z)>
*T fabmon_of_nat_mcp_wf   4 5
              ∀s:DSet. ∀m:MCopower(s;<ℤ+>↓hgrp).  fabmon_of_nat_mcp(m) ∈ FAbMon(s)
*C fabgrp_com  ============================
              FREE ABELIAN GROUP OVER A SET
              ============================
*D fabgrp_sig_df
              FAbGrpSig{<i:level>}(<s:s:*>)== fabgrp_sig{<i>:l}(<s>)
              FAbGrpSig(<s:s:*>)== fabgrp_sig{i:l}(<s>)
*A fabgrp_sig  FAbGrpSig(s) ==
                grp:AbGrp
                × inj:(|s| → |grp|)
                × (grp':AbGrp → (|s| → |grp'|) → |grp| → |grp'|)
*T fabgrp_sig_wf          0 1 ∀s:DSet. FAbGrpSig(s) ∈ 𝕌'
*D fabgrp_grp_df              <f:fabgrp:E>.grp== fabgrp_grp{}(<f>)
*A fabgrp_grp                 f.grp ==  f.1
*T fabgrp_grp_wf          0 0 ∀s:DSet. ∀f:FAbGrpSig(s).  f.grp ∈ AbGrp
*D fabgrp_inj_df              <f:fabgrp:E>.inj== fabgrp_inj{}(<f>)
*A fabgrp_inj                 f.inj ==  f.2.1
*T fabgrp_inj_wf          0 0 ∀s:DSet. ∀f:FAbGrpSig(s).  f.inj ∈ |s| → |f.grp|
*D fabgrp_umap_df             <f:fabgrp:E>.umap== fabgrp_umap{}(<f>)
*A fabgrp_umap                f.umap ==  f.2.2
*T fabgrp_umap_wf         0 0
              ∀s:DSet. ∀f:FAbGrpSig(s).
                f.umap ∈ grp':AbGrp → (|s| → |grp'|) → |f.grp| → |grp'|
*M create_fabgrp_sig
              Class Declaration for: FAbGrpSig(s)
                Long Name: fabgrp_sig
                Short Name: fabgrp
                Parameters:
                   s : DSet
                Fields:
                   grp : AbGrp
                   inj : |s| → |grp|
                   umap : grp':AbGrp → (|s| → |grp'|) → |grp| → |grp'|
                Universe: 𝕌'
*D fabgrp_df   FAbGrp{<i:level>}(<s:s:*>)== fabgrp{<i>:l}(<s>)
              FAbGrp(<s:s:*>)== fabgrp{i:l}(<s>)
*A fabgrp      FAbGrp(s) ==
                {fg:FAbGrpSig(s)|
                 ∀g:AbGrp. ∀h:|s| → |g|.
                   fg.umap g h = !h':|fg.grp| → |g|
                                   IsMonHom{fg.grp,g}(h') ∧ h' o fg.inj = h}
*T fabgrp_wf              0 2 ∀s:DSet. FAbGrp(s) ∈ 𝕌'
*C gcopower_com
              =============
              GROUP COPOWER
              =============
              We need to create a new signature here, because the desired
              group copower type is incomparable with the monoid
              copower sig.  Maybe it is is a poor design decision to include
              any domain restrictions in signatures.
              I assume that Any total function with the class of monoids or groups
              as domain, will in practice be also total over the more
              general signatures.
*D gcopower_sig_df
```

```
              GCopowerSig{<i:level>}(<s:s:*>;<g:g:*>)== gcopower_sig{<i>:l}(<s>; <g>)
              GCopowerSig(<s:s:*>;<g:g:*>)== gcopower_sig{i:l}(<s>; <g>)
*A gcopower_sig
              GCopowerSig(s;g) ==
                grp:GrpSig
                × inj:(|s| → |g| → |grp|)
                × (h:AbGrp → (|s| → |g| → |h|) → |grp| → |h|)
*T gcopower_sig_wf        0 1 ∀s:PosetSig. ∀g:GrpSig.  GCopowerSig(s;g) ∈ 𝕌'
*D gcopower_grp_df            <g1:gcopower:E>.grp== gcopower_grp{}(<g1>)
*A gcopower_grp               g1.grp ==  g1.1
*T gcopower_grp_wf        0 0 ∀s:PosetSig. ∀g:GrpSig. ∀g1:GCopowerSig(s;g).  g1.grp ∈ GrpSig
*D gcopower_inj_df            <g1:gcopower:E>.inj== gcopower_inj{}(<g1>)
*A gcopower_inj               g1.inj ==  g1.2.1
*T gcopower_inj_wf        0 0
              ∀s:PosetSig. ∀g:GrpSig. ∀g1:GCopowerSig(s;g).  g1.inj ∈ |s| → |g| → |g1.grp|
*D gcopower_umap_df           <g1:gcopower:E>.umap== gcopower_umap{}(<g1>)
*A gcopower_umap              g1.umap ==  g1.2.2
*T gcopower_umap_wf       0 0
              ∀s:PosetSig. ∀g:GrpSig. ∀g1:GCopowerSig(s;g).
                g1.umap ∈ h:AbGrp → (|s| → |g| → |h|) → |g1.grp| → |h|
*M create_gcopower_sig
              Class Declaration for: GCopowerSig(s;g)
                Long Name: gcopower_sig
                Short Name: gcopower
                Parameters:
                   s : PosetSig
                   g : GrpSig
                Fields:
                   grp : GrpSig
                   inj : |s| → |g| → |grp|
                   umap : h:AbGrp → (|s| → |g| → |h|) → |grp| → |h|
                Universe: 𝕌'
*D mon_p_df                   mon_p(<g:g:*>)== mon_p{}(<g>)
*A mon_p                      mon_p(g) ==  IsMonoid(|g|;*;e)
*T mon_p_wf               0 0 ∀g:GrpSig. mon_p(g) ∈ ℙ
*D grp_p_df                   grp_p(<g:g:*>)== grp_p{}(<g>)
*A grp_p                      grp_p(g) ==  IsGroup(|g|;*;e;∼)
*T grp_p_wf               0 0 ∀g:GrpSig. grp_p(g) ∈ ℙ
*D rng_p_df                   rng_p(<r:r:*>)== rng_p{}(<r>)
*A rng_p                      rng_p(r) ==  grp_p(r↓+gp) ∧ mon_p(r↓xmn) ∧ BiLinear(|r|;+r;*)
*T rng_p_wf               0 0 ∀r:RngSig. rng_p(r) ∈ ℙ
*D alg_p_df                   AlgP(<r:rng:*>;<a:alg:*>)== alg_p{}(<r>; <a>)
*A alg_p       AlgP(r;a) ==
                rng_p(a↓rg)
                ∧ IsAction(|r|;*;1;|a|;·a)
                ∧ IsBilinear(|r|;|a|;|a|;+r;+a;+a;·a)
                ∧ (∀p:|r|. Dist1op2opLR(|a|;·a p;xa))
*T alg_p_wf               0 2 ∀r:RngSig. ∀a:AlgebraSig(|r|).  AlgP(r;a) ∈ ℙ
*D gcopower_df                gcopower{<i:level>}(<s:s:*>;<g:g:*>)== gcopower{<i>:l}(<s>; <g>)
*A gcopower    gcopower{i}(s;g) ==
                {c:GCopowerSig(s;g)|
                 IsEqFun(|c.grp|;=_b)
                 ∧ grp_p(c.grp)
                 ∧ Comm(|c.grp|;*)
                 ∧ (∀j:|s|. IsMonHom{g,c.grp}(c.inj j))
                 ∧ (∀h:AbGrp. ∀f:|s| → MonHom(g,h).
                      c.umap h f = !v:|c.grp| → |h|
```

```
                                    IsMonHom{c.grp,h}(v) ∧ (∀j:|s|. f j = v o c.inj j))}
*T gcopower_wf            0 4 ∀s:DSet. ∀g:AbGrp.  gcopower{i}(s;g) ∈ 𝕌'
*M gcopower_ml_inc
               add_set_inclusion_info
                 'gcopower' 'gcopower_sig'
                 AbSetDForInc Auto ;;
*T gcopower_properties    1 5
               ∀s:DSet. ∀g:AbGrp. ∀c:gcopower{i}(s;g).
                 IsEqFun(|c.grp|;=_b)
                 ∧ grp_p(c.grp)
                 ∧ Comm(|c.grp|;*)
                 ∧ (∀j:|s|. IsMonHom{g,c.grp}(c.inj j))
                 ∧ (∀h:AbGrp. ∀f:|s| → MonHom(g,h).
                      c.umap h f = !v:|c.grp| → |h|
                                    IsMonHom{c.grp,h}(v) ∧ (∀j:|s|. f j = v o c.inj j))
*C fmonalg_com ====================
               FREE MONOID ALGEBRA
               ====================
*D fma_sig_df  FMASig{<i:level>}(<G:grp:*>;<A:rng:*>)== fma_sig{<i>:l}(<G>; <A>)
               FMASig(<G:grp:*>;<A:rng:*>)== fma_sig{i:l}(<G>; <A>)
*A fma_sig     FMASig(G;A) ==
                 alg:A-Algebra
                 × inj:(|G| → |alg|)
                 × (N:A-Algebra → (|G| → |N|) → |alg| → |N|)
*T fma_sig_wf             0 1 ∀G:GrpSig. ∀A:RngSig.  FMASig(G;A) ∈ 𝕌'
*D fma_alg_df                 <f:fma:E>.alg== fma_alg{}(<f>)
*A fma_alg                    f.alg ==  f.1
*T fma_alg_wf             0 0 ∀G:GrpSig. ∀A:RngSig. ∀f:FMASig(G;A).  f.alg ∈ A-Algebra
*D fma_inj_df                 <f:fma:E>.inj== fma_inj{}(<f>)
*A fma_inj                    f.inj ==  f.2.1
*T fma_inj_wf             0 0 ∀G:GrpSig. ∀A:RngSig. ∀f:FMASig(G;A).  f.inj ∈ |G| → |f.alg|
*D fma_umap_df                <f:fma:E>.umap== fma_umap{}(<f>)
*A fma_umap                   f.umap ==  f.2.2
*T fma_umap_wf            0 0
               ∀G:GrpSig. ∀A:RngSig. ∀f:FMASig(G;A).
                 f.umap ∈ N:A-Algebra → (|G| → |N|) → |f.alg| → |N|
*M create_fma_sig
               % Create type of free (G,A) Monoid Algebras %
               % Do in two stages. Allows verification of existence
               of instances also to be broken up into stages
               Note that this type is more general than one with
               AlgebraSig(|A|) as the first domain of umap.
               This generality is necessary. It turns out that a umap
               considered in the polynom_3 theory isn't even well-formed
               for arbitrary elements of AlgebraSig(|A|). (Problem reduces
               to that mset_for requires for functionality reasons the addition
               of the algebra to be an abelian monoid.    %
               Class Declaration for: FMASig(G;A)
                 Long Name: fma_sig
                 Short Name: fma
                 Parameters:
                    G : GrpSig
                    A : RngSig
                 Fields:
                    alg : AlgebraSig(|A|)
                    inj : |G| → |alg|
                    umap : N:A-Algebra → (|G| → |N|) → |alg| → |N|
```

```
                Universe: 𝕌'
*D fmonalg_df  FMonAlg{<i:level>}(<g:g:*>;<a:a:*>)== fmonalg{<i>:l}(<g>; <a>)
               FMonAlg(<g:g:*>;<a:a:*>)== fmonalg{i:l}(<g>; <a>)
*A fmonalg     FMonAlg(g;a) ==
                 {m:FMASig(g;a)|
                  IsMonHom{g,m.alg↓rg↓xmn}(m.inj)
                  ∧ (∀n:a-Algebra. ∀f:MonHom(g,n↓rg↓xmn).
                       m.umap n f = !f':|m.alg| → |n|
                                      IsAlgHom{a,m.alg,n}(f') ∧ f' o m.inj = f)}
*T fmonalg_wf             0 3 ∀g:AbMon. ∀a:CRng.  FMonAlg(g;a) ∈ 𝕌'
*M fmonalg_ml_inc
               add_set_inclusion_info
                 'fmonalg' 'fma_sig'
                 AbSetDForInc Auto ;;
*T fmonalg_properties     0 4
               ∀g:AbMon. ∀a:CRng. ∀m:FMonAlg(g;a).
                 IsMonHom{g,m.alg↓rg↓xmn}(m.inj)
                 ∧ (∀n:a-Algebra. ∀f:MonHom(g,n↓rg↓xmn).
                      m.umap n f = !f':|m.alg| → |n|
                                     IsAlgHom{a,m.alg,n}(f') ∧ f' o m.inj = f)
*D polynom_alg_df
               PolynomAlg{<i:level>}(<S:S:*>;<A:A:*>)== polynom_alg{<i>:l}(<S>; <A>)
               PolynomAlg(<S:S:*>;<A:A:*>)== polynom_alg{i:l}(<S>; <A>)
*A polynom_alg               PolynomAlg(S;A) ==  mo:FAbMon(S) × FMonAlg(mo.mon;A)
*T polynom_alg_wf         0 0 ∀S:DSet. ∀A:CRng.  PolynomAlg(S;A) ∈ 𝕌'
*D polyalg_mo_df              <p:polyalg:E>.mo== polyalg_mo{}(<p>)
*A polyalg_mo                 p.mo ==  p.1
*T polyalg_mo_wf          0 0 ∀S:DSet. ∀A:CRng. ∀p:PolynomAlg(S;A).  p.mo ∈ FAbMon(S)
*D polyalg_poly_df            <p:polyalg:E>.poly== polyalg_poly{}(<p>)
*A polyalg_poly               p.poly ==  p.2
*T polyalg_poly_wf        0 0
               ∀S:DSet. ∀A:CRng. ∀p:PolynomAlg(S;A).  p.poly ∈ FMonAlg(p.mo.mon;A)
*M create_polynom_alg
               Class Declaration for: PolynomAlg(S;A)
                 Long Name: polynom_alg
                 Short Name: polyalg
                 Parameters:
                    S : DSet
                    A : CRng
                 Fields:
                    mo : FAbMon(S)            % the monomials %
                    poly : FMonAlg(mo.mon;A)  % the polynomials %
                 Universe: 𝕌'
*D rng_from_grp_df
               rng_from_grp(<g:g:*>;<times:times:*>;<one:one:*>;<div:div:*>)
               == rng_from_grp{}(<g>; <times>; <one>; <div>)
*A rng_from_grp
               rng_from_grp(g;times;one;div) ==  <|g|, =_b, ≤_b, *, e, ∼, times, one, div>
*T rng_from_grp_wf        0 3
               ∀g:GrpSig. ∀times:|g| → |g| → |g|. ∀one:|g|. ∀div:|g| → |g| → ?|g|.
                 rng_from_grp(g;times;one;div) ∈ RngSig
*D alg_from_rng_df
               alg_from_rng(<A:A:*>;<r:r:*>;<act:act:*>)== alg_from_rng{}(<A>; <r>; <act>)
*A alg_from_rng               alg_from_rng(A;r;act) ==  <|r|, =_b, ≤_b, +r, 0, -r, *, 1, ÷r, act>
*T alg_from_rng_wf        0 3
               ∀A:𝕌. ∀r:RngSig. ∀act:A → |r| → |r|.  alg_from_rng(A;r;act) ∈ AlgebraSig(A)
*D fma_from_gcp_df
```

```
              fma_from_gcp(<g:g:*>;<r:r:*>;<c:c:*>;<a:a:*>)
              == fma_from_gcp{}(<g>; <r>; <c>; <a>)
*A fma_from_gcp
              fma_from_gcp(g;r;c;a) ==
                <a
                , λp:|g|. c.inj p 1
                , λa.(λh:|g| → |a|. c.umap a↓grp (λp:|g|. λv:|r|. ·a v (h p)))>
*T fma_from_gcp_wf        3 4
              ∀g:GrpSig. ∀r:RngSig. ∀c:GCopowerSig(g↓set;r↓+gp). ∀a:r-Algebra.
                a↓grp = c.grp ⇒ fma_from_gcp(g;r;c;a) ∈ FMASig(g;r)
#T fma_from_gcp_wf2       2 4
              ∀g:Mon. ∀r:CRng. ∀c:gcopower{i}(g↓set;r↓+gp). ∀a:r-Algebra.
                a↓grp = c.grp ⇒ fma_from_gcp(g;r;c;a) ∈ FMonAlg(g;r)
*C polynom_1_end              ************************************
Thm stats: <log2 (# pscript lines)> <log2 (expansion time in sec)>
```