The TECTON Concept Library

David R. Musser, Sibylle Schupp,
Christoph Schwarzweller and Rüdiger Loos

WSI 99-2

30. September 1999

Wilhelm-Schickard-Institut für Informatik
Arbeitsbereich Computeralgebra
Prof. Dr. Rüdiger Loos
Sand 13
D-72076 Tübingen

e-mail: {musser,schupp}@cs.rpi.edu, {schwarzw,loos}@informatik.uni-tuebingen.de

© WSI 1999
ISSN 0946-3852

Abstract

TECTON is an algebraic specification language. This report contains a considerable body of TECTON concepts which evolved over a long time. The concepts serve as a test bed for a TECTON translator and are a formal base for declarations occurring in algorithms from all areas of programming but in particular from computer algebra.

Contents

Chapter 1

Introduction

We have defined well over 100 algebraic concepts in the concept description language TECTON [5], [4]. The main goal of these definitions is to provide a torture test of TECTON; but at the same time we are interested to have a conceptual framework for algorithm specification for a generic library for computer algebra. If the TECTON translator and the generic library evolve in the future the concepts of this report should always remain a fixed base for regression testing.
A second goal of this report are combined definitions of concepts important both for computer science and mathematics with all rigour and details which are needed for formal reasoning and software construction with provable properties.We claim, for example, that our definition of the reals inspired by Tarski [6], is the first single concept definition of the reals on a machine without any loss of mathematical contents and precision. Of course the complete concept of the reals is at the basis of many algorithms over the reals, as for example in quantifier elimination algorithms for real closed fields; but the concept is not explicitly represented on the machine in any formal sense. Also the real numbers have been constucted using theorem provers like for example MIZAR [9] or HOL [8];but the emphasize of these constructions is "only" to prove properties of the reals and not to serve in addition as the basis for a data structure over which algorithms are to be executed.

The concepts are organized in a single acyclic tree of concept families with the `set`-family as its root. The family `algebra1` provides concepts for a single, the family `algebra2` for two connectives. They are first independent of any `relation` and `order` concepts, but then merged with these families into the family of ordered algebra (`ordalgebra`). Another offspring of the algebra families is the family of `morphisms`.
Ordered algebra is the base family for arithmetical concepts (with the samll exception of the naturals, which are needed already in the root family in order to define sequences and similar concepts). The `arithmetic`al family comprises integers, rationals, reals, complex and quaternions. This artihmetical hierachy was inspired from a similar effort in OBJ3, but stays within the framework provided by mathematics. `Polynomial`s are finally inherited from arithmetical concepts such that a firm conceptual base for this important computer algebra domain with its algorithms can be established. `Ample set`, the last family in this report, play an important role in gcd-algorithms and any algorithm working on canonical forms of its input.

The reader is invited to report errors to `loos@informatik.uni-tuebingen.de`.

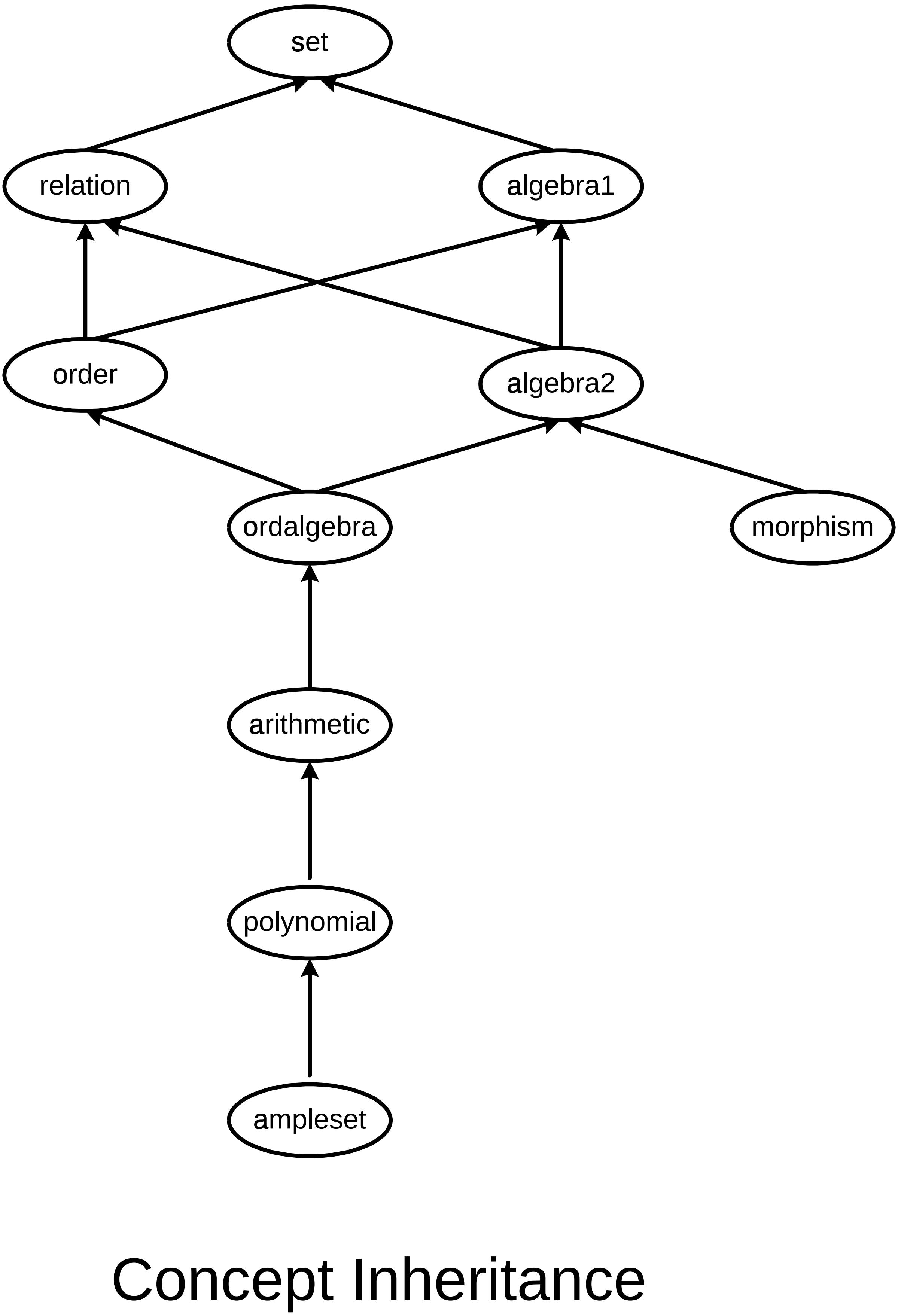

Figure 1.1: Structure of the library

Chapter 2

Sets, Maps and Sequences

2.1 Boolean, Domain, Set, Finite-set, Range

"src/set.tec" 4a ≡

```
  Library: std
    Boolean, Domain, Set, Finite-set, Range, Map, Finite-map, Natural, Segment,
    Natural-set, Sequence, Finite-sequence, Cartesian-product-of-set.

  Definition: Boolean
    introduces bool,
      true -> bool,
      false -> bool;
    generates bool freely using true, false.

  Precedence: nonassociative{=, !=}.
  Precedence: {implies} < {or, xor} < {and}
                < prefix{not} < nonassociative{=} < {:}.
  Precedence: confix{(, ,, )}.

  Extension: Boolean
    introduces
      not     : bool -> bool,
      and     : bool x bool -> bool,
      or      : bool x bool -> bool,
      xor     : bool x bool -> bool,
      implies : bool x bool -> bool;
    requires (for x, y: bool)
      (not true)    = false,
      (not false)   = true,
      (true and x)  = x,
      (false and x) = false,
      (x or y)      = (not (not x and not y)),
      (x xor y)     = (not x = y),
      (x implies y) = (not x or y).
```

◇

File defined by parts 4ab, 5ab, 6ab, 7abc.

2.2 Domain, Range and Set

"src/set.tec" 4b ≡

```
  Definition: Domain
    uses Boolean;
```

```
  introduces domain.

Precedence: nonassociative{=} < nonassociative{in}.
Definition: Range
  uses Domain[with range as domain].

Definition: Set
  uses Domain;
  introduces sets,
    empty : -> sets,
    member: domain x sets -> bool;
  requires
    (for a: domain) member(a, empty) = false.
```

◇
File defined by parts 4ab, 5ab, 6ab, 7abc.

"src/set.tec" 5a ≡

```
Precedence: nonassociative{in, into}.
Precedence: nonassociative{=} = {union} < {intersection} < {subset}.

Extension: Set
introduces
  nonempty-sets < sets,
  subset       : sets x sets -> bool,
  is_empty     : sets -> bool,
  complement   : sets -> sets,
  singleton    : domain -> sets,
  into         : domain x sets -> sets,
  union        : sets x sets -> sets,
  intersection : sets xsets -> sets;
requires (for d, e: domain; s, s1, s2: sets)
  (s1 subset s2) = (member(d, s1) implies member(d,s2)),
  is_empty(s) = (s = empty),
  member(d, (e into s1)) = ((d = e) or member(d, s1)),
  member(d, complement(s)) = not member(d, s),
  singleton(d) = (d into empty),
  (s1 union empty) = s1,
  (s1 union (d into s2)) =
     if member(d, s1) then s1 union s2
                 else d into (s1 union s2),
  (s1 intersection empty) = empty,
  (s1 intersection (d into s2)) =
     if member(d, s1) then d into (s1 intersection s2)
                 else s1 intersection s2,
  s in nonempty-sets = (s != empty).
```

◇
File defined by parts 4ab, 5ab, 6ab, 7abc.

"src/set.tec" 5b ≡

```
Lemma: Set
obeys (for d, e: domain; s, s1, s2: sets)
  is_empty(empty),
  (s subset empty) implies (s = empty),
  member(d, singleton(e)) = (d = e),
  member(d, s1 union s2) = (member(d, s1) or member(d, s2)),
  member(d, s1 intersection s2) = (member(d, s1) and member(d, s2)).
```

```
Definition: Finite-set
  refines Set,
  introduces
    finite-sets < sets,
    nonempty-finite-sets < nonempty-sets,
    into : domain x finite-sets -> nonempty-finite-sets;
  generates finite-sets using empty, into;
  requires (for s: sets; s1: nonempty-sets)
    s in finite-sets
      = (s = empty or s != empty
        and (for some d: domain; s': finite-sets) s = d into s'),
    s1 in nonempty-finite-sets = (s1 != empty).
```

◇
File defined by parts 4ab, 5ab, 6ab, 7abc.

2.3 Map, Finite-map, Natural, Segment, Natural-set

"src/set.tec" 6a ≡

```
Definition: Map
  refines Set[with maps as sets, nonempty-maps as nonempty-sets];
  uses Range;
  introduces
     apply : maps x domain -> range.

Definition: Finite-map
  refines Map;
  introduces
    finite-maps < maps,
    nonempty-finite-maps < nonempty-maps,
    into : domain x finite-maps -> nonempty-finite-maps;
  generates finite-maps using empty, into;
  requires (for s: maps; s1: nonempty-maps)
    s in finite-maps
      = (s = empty or s != empty
        and (for some d: domain; s': finite-maps) s = d into s'),
    s1 in nonempty-finite-maps = (s1 != empty).
```
◇
File defined by parts 4ab, 5ab, 6ab, 7abc.

"src/set.tec" 6b ≡

```
Precedence:
  nonassociative{<, <=, >=, >, =} < {+, -} < {*}.

Definition: Natural
  refines Domain [with naturals as domain];
  introduces
    0 -> naturals,
    1 -> naturals,
    succ : naturals  -> naturals,
    +    : naturals x naturals -> naturals,
    *    : naturals x naturals -> naturals;
  generates naturals freely using 0, succ;
  requires (for n, m: naturals)
    n + 0 = n,
    n + succ(m) = succ(n + m),
```

```
      1 = succ(0),
      n * 0 = 0,
      n * succ(m) = n * m + n.
```

◇
File defined by parts 4ab, 5ab, 6ab, 7abc.

"src/set.tec" 7a ≡

```
  Extension:  Natural
    introduces
      nonzero-naturals < naturals,
      naturals < naturals?,
      2   : -> naturals,
      natural-underflow -> naturals?,
      -    : naturals x naturals -> naturals?,
      <=  : naturals x naturals -> bool,
      <  (x: naturals, y: naturals) = (x <= y and not(x = y)),
      >= (x: naturals, y: naturals) = not(x < y),
      >  (x: naturals, y: naturals) = not(x <= y);
    requires (for n, m: naturals; k: naturals?)
      2 = 1 + 1,
      n - 0 = n,
      0 - n = if n=0 then 0 else natural-underflow,
      succ(n) - succ(m) = n - m,
      0 <= n,
      not(succ(n) <= 0),
      (succ(m) <= succ(n)) = (m <= n),
      n in nonzero-naturals = (n != 0),
      k in naturals = (k != natural-underflow).
```

◇
File defined by parts 4ab, 5ab, 6ab, 7abc.

"src/set.tec" 7b ≡

```
  Definition: Segment
    uses Natural;
    introduces
      segments < naturals,
      max: -> naturals;
    requires (for n: naturals)
      n in segments = (n < max).

  Abbreviation: Natural-set is
    Set [with Natural as Domain,
              naturals as domain,
              natural-sets as sets].
```

◇
File defined by parts 4ab, 5ab, 6ab, 7abc.

2.4 Sequence, Finite-sequence, Cartesian-product-of-set

"src/set.tec" 7c ≡

```
  Definition: Sequence
    refines Map [with Natural as Domain,
                    naturals as domain,
                        n_th as apply,
                   sequences as maps,
         nonempty-sequences as nonempty-maps].
```

```
Definition: Finite-sequence
  refines Sequence;
  introduces
    finite-sequences < sequences,
    nonempty-finite-sequences < nonempty-sequences,
    into : domain x finite-sequences -> nonempty-finite-sequences;
  generates finite-sequences freely using empty, into;
  requires (for s: sequences; s1: nonempty-sequences)
    s in finite-sequences
      = (s = empty or s != empty
         and (for some d: domain; s': finite-sequences) s = d into s'),
    s1 in nonempty-finite-sequences = (s1 != empty).

Definition: Cartesian-product-of-set
  refines Finite-sequence [with Set as Range, sets as range].
```
◇
File defined by parts 4ab, 5ab, 6ab, 7abc.

Chapter 3

Relations

3.1 Unary-relation, General-binary-relation, Function, Binary-relation

"src/relation.tec" 9a ≡

```
Pragma: include="set.xgf".
Pragma: concepts.
Library: std
  Unary-relation, General-binary-relation, Function, Binary-relation, Surjection,
  Injection, Transitive, Symmetric, Reflexive, Irreflexive, Antisymmetric,
  Bijection, Finite, Equivalence-relation, Equivalence-class,
  Set-of-representatives.

Precedence: nonassociative{=, R} < prefix{P}.

Definition: Unary-relation
  refines Domain;
  introduces P : domain -> bool.
```

◇

File defined by parts 9abc, 10abcdef, 11abc.

"src/relation.tec" 9b ≡

```
Precedence: nonassociative{R, <=}.

Definition: General-binary-relation
  uses Domain, Range;
  introduces R : domain x range -> bool.
```

◇

File defined by parts 9abc, 10abcdef, 11abc.

"src/relation.tec" 9c ≡

```
Definition: Function
  refines General-binary-relation;
  introduces f: domain -> range;
  requires (for x: domain; y, y': range)
    (f(x) = y) = (x R y),
    f(x) = y and f(x) = y' implies y = y'.
```

◇

File defined by parts 9abc, 10abcdef, 11abc.

"src/relation.tec" 10a ≡

```
Definition: Binary-relation
  refines Domain;
  introduces R : domain x domain -> bool.

Lemma: Binary-relation is General-binary-relation.
```

◇
File defined by parts 9abc, 10abcdef, 11abc.

3.2 Surjection, Injection

"src/relation.tec" 10b ≡

```
Definition: Surjection
  refines Function;
  requires (for y: range) (for at least 1 x: domain)
    f(x) = y.
```

◇
File defined by parts 9abc, 10abcdef, 11abc.

"src/relation.tec" 10c ≡

```
Definition: Injection
  refines Function;
  requires (for y: range) (for at most 1 x: domain)
    f(x) = y.
```

◇
File defined by parts 9abc, 10abcdef, 11abc.

3.3 Transitive, Symmetric, Reflexive, Irreflexive, Antisymmetric

"src/relation.tec" 10d ≡

```
Definition: Transitive
  refines Binary-relation;
  requires
    (for x, y, z: domain) x R y and y R z implies x R z.
```

◇
File defined by parts 9abc, 10abcdef, 11abc.

"src/relation.tec" 10e ≡

```
Definition: Symmetric
  refines Binary-relation;
  requires
    (for x, y: domain) x R y implies y R x.
```

◇
File defined by parts 9abc, 10abcdef, 11abc.

"src/relation.tec" 10f ≡

```
    Definition: Reflexive
      refines Binary-relation;
      requires
        (for x: domain) x R x.

    Definition: Irreflexive
      refines Binary-relation;
      requires
        (for x: domain) not x R x.

    Definition: Antisymmetric
      refines Binary-relation;
      requires
        (for x, y: domain) x R y and y R x implies x = y.
```

◇
File defined by parts 9abc, 10abcdef, 11abc.

3.4 Bijection, Finite

"src/relation.tec" 11a ≡

```
    Definition: Bijection
      refines Surjection, Injection.

    Definition: Finite
      refines
        Domain,
        Bijection [with segments as domain, domain as range];
      uses Segment.
```

◇
File defined by parts 9abc, 10abcdef, 11abc.

3.5 Equivalence-relation, Equivalence-class, Set-of-representatives

"src/relation.tec" 11b ≡

```
    Definition: Equivalence-relation
      refines Reflexive, Symmetric, Transitive.

    Precedence: nonassociative{=, equiv}.

    Definition: Equivalence-class
      uses Domain, Equivalence-relation [with equiv as R];
      introduces equivalence-classes,
        member : domain x equivalence-classes -> bool,
        equivalence-class : domain -> equivalence-classes;
      requires (for x, y: domain; e: equivalence-classes)
        (equivalence-class(x) = equivalence-class(y)) = (x equiv y),
        member(x, e) = (equivalence-class(x) = e).
```

◇
File defined by parts 9abc, 10abcdef, 11abc.

"src/relation.tec" 11c ≡

```
Definition: Set-of-representatives
  uses Equivalence-class;
  introduces
    set-of-representatives < domain,
    representative : equivalence-classes  -> domain,
    representative : domain -> domain;
  requires (for x: domain; e: equivalence-classes)
    x in set-of-representatives = (representative(x) = x),
    equivalence-class(representative(e)) = e,
    representative(x) = representative(equivalence-class(x)).
```

◇

File defined by parts 9abc, 10abcdef, 11abc.

Chapter 4

Order Concepts

4.1 Strict-partial-order, Partial-order, Total-order, Trichotomy

"src/order.tec" 13a ≡

```
Pragma: include="relation.xgf", include="algebra1.xgf".

Precedence: nonassociative{R, <}.
Library: std
  Strict-partial-order, Partial-order, Total-order, Trichotomy, Nondense-order,
  Dense-order, Archimedean-order, Continuous-order.

Definition: Strict-partial-order
  refines Irreflexive [with < as R],
          Transitive [with < as R].

Precedence: nonassociative{R, <=, =}.

Definition: Partial-order
  refines Reflexive [with <= as R],
          Antisymmetric [with <= as R],
          Transitive [with <= as R].

Precedence: nonassociative{<, <=, >=, >, =}.

Extension: Partial-order
  introduces
    <  : domain x domain -> bool,
    >  : domain x domain -> bool,
    >= : domain x domain -> bool;
  requires (for x, y: domain)
    (x < y)  = (x <= y and x != y),
    (x > y)  = (not x <= y),
    (x >= y) = (x > y or x = y).

Lemma: Partial-order implies Strict-partial-order.
```
◇

File defined by parts 13ab, 14ab.

"src/order.tec" 13b ≡

```
Definition: Total-order
  refines Partial-order;
  requires (for x, y: domain)
    x <= y or y <= x.

Definition: Trichotomy
  refines Strict-partial-order;
  requires (for x, y : domain)
    x < y or x = y or y < x.

Lemma: Trichotomy is Total-order .
```

◇
File defined by parts 13ab, 14ab.

4.2 Nondense-order, Dense-order, Archimedean-order, Continuous-order

"src/order.tec" 14a ≡

```
Definition: Nondense-order
  refines Total-order;
  requires
    not ((for x, y: domain)
           x < y implies (for some z: domain) x < z and z < y).

Definition: Dense-order
  refines Total-order;
  requires (for x, y: domain)
           x < y implies (for some z: domain) x < z and z < y.

Definition: Archimedean-order
  refines Total-order, Abelian-group;
  requires (for x, y, z: domain)
    x <= y implies x + z <= y + z.
```

◇
File defined by parts 13ab, 14ab.

"src/order.tec" 14b ≡

```
Definition: Continuous-order
  refines Total-order;
  uses Set;
  introduces
    precedes     : sets x sets -> bool,
    separates    : sets x domain x sets -> bool;
  requires (for K, L: sets; z: domain)
    precedes(K, L) =
      (for x, y: domain) member(x, K) and member(y, L) implies x < y,
    separates(K, z, L) =
      (for x, y: domain)
        member(x, K) and member(y, L) and x != z and y != z
          implies  x < z and z < y,
    // Dedekind's axiom
    precedes(K, L) implies (for at least 1 z: domain) separates(K, z, L).

Lemma: Continuous-order implies Dense-order.
```

◇

File defined by parts 13ab, 14ab.

Chapter 5

Algebras with 1 Connective

5.1 Binary-op, Right-regular, Right-identity, Left-regular, Left-identity

"src/algebra1.tec" 16a ≡

```
Pragma: include="set.xgf".
Precedence: nonassociative{=} < {*}.
Library: std
  Binary-op, Right-regular, Right-identity, Left-regular, Left-identity,
  Commutative, Associative, Right-inverses, Regular, Left-inverses, Identity,
  Semigroup, Inverses, Regular-semigroup, Monoid, Commutative-semigroup, Group,
  Abelian-monoid, Trivial-group, Group-of-order-2, Commutative-group,
  Abelian-group, Additive-trivial-group.

Definition: Binary-op
  uses Domain;
  introduces * : domain x domain -> domain.

Precedence: nonassociative{=} < {|} < {+, -}.

Definition: Right-regular
  refines Binary-op;
  introduces | : domain x domain -> bool;
  requires (for x, y: domain)
    x | y = (for some d: domain) x * d = y.

Definition: Right-identity
  refines Binary-op;
  introduces 1 -> domain;
  requires (for x: domain)
    x * 1 = x.

Definition: Left-regular
  refines Binary-op;
  introduces | : domain x domain -> bool;
  requires (for x, y: domain)
    x | y = (for some d: domain) d * x = y.
```

◇
File defined by parts 16ab, 17ab, 18abcde, 19ab.

"src/algebra1.tec" 16b ≡

```
Definition: Left-identity
  refines Binary-op;
  introduces 1 -> domain;
  requires (for x: domain)
    1 * x = x.
```

◇
File defined by parts 16ab, 17ab, 18abcde, 19ab.

5.2 Commutative, Associative, Right-inverses, Regular, Left-inverses, Identity

"src/algebra1.tec" 17a ≡

```
Definition: Commutative
  refines Binary-op;
  requires (for x, y: domain)
    x * y = y * x.

Definition: Associative
  refines Binary-op;
  requires (for x, y, z: domain)
    x * (y * z) = (x * y) * z.

Precedence: prefix{-} < {*} < postfix{^(-1)}.

Definition: Right-inverses
  refines Right-identity, Right-regular;
  introduces ^(-1) : domain -> domain;
  requires (for x: domain)
    x * x^(-1) = 1.
```

◇
File defined by parts 16ab, 17ab, 18abcde, 19ab.

"src/algebra1.tec" 17b ≡

```
Lemma: Right-inverses implies Right-regular.

Definition: Regular
  refines Left-regular, Right-regular.

Precedence: prefix{-} < {*} < postfix{^(-1)}.

Definition: Left-inverses
  refines Left-identity, Left-regular;
  introduces ^(-1) : domain -> domain;
  requires (for x: domain)
    x^(-1) * x = 1.

Lemma: Left-inverses implies Left-regular.

Definition: Identity
  refines Left-identity, Right-identity.
```

◇
File defined by parts 16ab, 17ab, 18abcde, 19ab.

5.3 Semigroup, Inverses, Regular-semigroup, Monoid, Commutative-semigroup

"src/algebra1.tec" 18a ≡

```
Abbreviation: Semigroup is Associative.
```

◇
File defined by parts 16ab, 17ab, 18abcde, 19ab.

"src/algebra1.tec" 18b ≡

```
Definition: Inverses
  refines Left-inverses, Right-inverses.

Lemma: Inverses implies Regular.

Precedence: {/, *}.

Extension: Inverses
  introduces  / : domain x domain -> domain;
  requires (for x, y:domain)
    x/y = x * y^(-1).

Definition: Regular-semigroup
  refines Regular, Semigroup.

Definition: Monoid
  refines Semigroup, Identity.
```

◇
File defined by parts 16ab, 17ab, 18abcde, 19ab.

"src/algebra1.tec" 18c ≡

```
Precedence: nonassociative{=} < {+, -}.

Definition: Commutative-semigroup
  refines Regular-semigroup[with + as *],
          Commutative[with + as *].
```

◇
File defined by parts 16ab, 17ab, 18abcde, 19ab.

5.4 Group, Abelian-monoid, Trivial-group, Group-of-order-2

"src/algebra1.tec" 18d ≡

```
Definition: Group
  refines Monoid, Inverses.

Definition: Abelian-monoid
  refines Monoid, Commutative.
```

◇
File defined by parts 16ab, 17ab, 18abcde, 19ab.

"src/algebra1.tec" 18e ≡

```
Definition: Trivial-group
  refines Group;
  requires (for x: domain) x = 1.

Definition: Group-of-order-2
  refines Group;
  requires (for x: domain) x * x = 1.

Lemma: Group-of-order-2 is Commutative.
```

◇
File defined by parts 16ab, 17ab, 18abcde, 19ab.

5.5 Commutative-group, Abelian-group, Additive-trivial-group

"src/algebra1.tec" 19a ≡

```
Definition: Commutative-group
  refines Commutative, Group.
```

◇
File defined by parts 16ab, 17ab, 18abcde, 19ab.

"src/algebra1.tec" 19b ≡

```
Precedence: nonassociative{in} < {+, -} < prefix{-, +}
            < {*} < postfix{^(-1)}.

Definition: Abelian-group
  refines Commutative-group[with + as *, - as ^(-1), 0 as 1, - as /];
  introduces nonzeros < domain,
    - : domain x domain -> domain,
    + : domain -> domain;
  requires (for x, y: domain)
    x in nonzeros = (x != 0),
    x - y = x + (-y),
      + x = x.

Definition: Additive-trivial-group
  refines Abelian-group[with Trivial-group as Commutative-group].
```

◇
File defined by parts 16ab, 17ab, 18abcde, 19ab.

Chapter 6

Algebras with 2 Connectives

6.1 Right-distributive, Left-distributive, Distributive, Semiring, Ring

"src/algebra2.tec" 20a ≡

```
Pragma: include="relation.xgf", include="algebra1.xgf".
Precedence: nonassociative{=} < {+, -} < prefix{-, +} < {*}.
Library: std
  Right-distributive, Left-distributive, Distributive, Semiring, Ring,
  Commutative-ring, Ring-with-identity, Commutative-ring-with-identity, Unit,
  Right-module, No-zero-divisors, Left-module, Division-ring, Module, Skewfield,
  Right-ideal, Left-ideal, Integral-domain, Gcd-domain, Euclidean-domain,
  Coefficient-ring, Unique-right-ideal, Unique-left-ideal, Ideal, Unique-ideal,
  Set-of-pairwise-spanning-ideals, Trivial-ideal, Proper-ideal, Ideal-equivalence,
  Ideal-equivalence-class, Field, Quotient-ring.

Definition: Right-distributive
  refines Binary-op, Binary-op [with + as *];
  requires (for x, y, z: domain)
    (x + y) * z = x * z + y * z.

Definition: Left-distributive
  refines Binary-op, Binary-op [with + as *];
  requires (for x, y, z: domain)
    x * (y + z) = x * y + x * z.

Definition: Distributive
  refines Left-distributive, Right-distributive.

Definition: Semiring
  refines Commutative-semigroup, Semigroup, Distributive.

Definition: Ring
  refines Abelian-group, Semigroup, Distributive.
```
◇

File defined by parts 20ab, 21, 22abc, 23ab, 24abcd, 25.

6.2 Commutative-ring, Ring-with-identity, Commutative-ring-with-identity, Unit

"src/algebra2.tec" 20b ≡

```
Definition: Commutative-ring
  refines Ring, Commutative.

Definition: Ring-with-identity
  refines Ring, Identity.

Definition: Commutative-ring-with-identity
  refines Commutative-ring, Identity.

Definition: Unit
  refines Ring-with-identity;
  uses Regular;
  introduces units < domain, nonunits < domain;
  requires
    (for u: domain)
    u in units = u | 1,
    u in nonunits = (not u | 1).

Lemma: Unit implies Group.
```

◇
File defined by parts 20ab, 21, 22abc, 23ab, 24abcd, 25.

6.3 Right-module, No-zero-divisors, Left-module, Division-ring, Module, Skewfield

"src/algebra2.tec" 21 ≡

```
Precedence: {+, -} < prefix{-} < {*}.

Definition: Right-module
  refines Abelian-group[with right-module-elements as domain];
  uses Ring, Right-identity;
  introduces
    * : right-module-elements x domain -> right-module-elements;
  requires (for a, b: domain; x, y: right-module-elements)
    x * (a * b)  = (x * a) * b,
    x * (a + b)  = x * a + x * b,
    (x + y) * a = x * a + y * a,
    x * 1 = x.

Definition: No-zero-divisors
  refines Ring;
  introduces
    * : nonzeros x nonzeros -> nonzeros,
    1 : -> nonzeros;
  requires (for x, y: domain)
    x * y = 0 implies x = 0 or y = 0.

Precedence: {+, -} < prefix{-} < {*}.

Definition: Left-module
  refines Abelian-group[with left-module-elements as domain];
  uses Ring, Left-identity;
  introduces
    * : domain x left-module-elements -> left-module-elements;
  requires (for a, b: domain; x, y: left-module-elements)
    (a * b) * x = a * (b * x),
```

```
        (a + b) * x = a * x + b * x,
        a * (x + y) = a * x + a * y,
        1 * x = x.
```

◇
File defined by parts 20ab, 21, 22abc, 23ab, 24abcd, 25.

"src/algebra2.tec" 22a ≡

```
   Precedence: prefix{-} < {*} < postfix{^(-1)}.

   Definition: Division-ring
     refines Ring, Inverses;
     introduces ^(-1) : nonzeros -> nonzeros;
     requires
       0 != 1,
       (for y: nonzeros) y * y^(-1) = 1.
```

◇
File defined by parts 20ab, 21, 22abc, 23ab, 24abcd, 25.

"src/algebra2.tec" 22b ≡

```
   Definition: Module
     refines
       Left-module  [with module-elements as left-module-elements],
       Right-module [with module-elements as right-module-elements].

   Lemma: Module[with Additive-trivial-group as Abelian-group] implies
           Module.

   Lemma: Ring implies Module[with domain as module-elements].

   Abbreviation: Skewfield is Division-ring.

   Extension: Commutative-ring-with-identity
     uses Unit;
     introduces prime-elements < nonzeros;
     requires (for d: nonzeros)
       d in prime-elements = not((for some q, r: nonunits) d = q * r).

   Lemma: Map[with nonempty-sets as domain, Module as Range] implies
           Module.
```

◇
File defined by parts 20ab, 21, 22abc, 23ab, 24abcd, 25.

6.4 Right-ideal, Left-ideal, Integral-domain, Gcd-domain, Euclidean-domain

"src/algebra2.tec" 22c ≡

```
   Definition: Right-ideal
     refines Set [with ideals as sets];
     uses Ring;
     requires (for I: ideals; a, b: domain)
       member(0, I),
       member(a, I) and member(b, I) implies member(a + b, I),
```

```
      member(a, I) implies member(a * b, I).

  Definition: Left-ideal
    refines Set [with ideals as sets];
    uses Ring;
    requires (for I: ideals; a, b: domain)
      member(0, I),
      member(a, I) and member(b, I) implies member(a + b, I),
      member(a, I) implies member(b * a, I).
```

◇
File defined by parts 20ab, 21, 22abc, 23ab, 24abcd, 25.

"src/algebra2.tec" 23a ≡

```
  Definition: Integral-domain
    refines Commutative-ring-with-identity, No-zero-divisors.

  Definition: Gcd-domain
    refines Integral-domain;
    uses Set-of-representatives;
    introduces gcd : domain x domain -> set-of-representatives;
    requires (for x, y: domain)
      gcd(x, y) | x and gcd(x, y) | y and
        ((for z: domain) (z | x and z | y) implies z | gcd(x, y)),
      (for some z: domain) gcd(x, y) = z.

  Definition: Euclidean-domain
    refines Gcd-domain;
    uses Natural;
    introduces
      Euclidean_function : nonzeros -> naturals,
      div : domain x nonzeros -> domain,
      rem : domain x nonzeros -> domain;
    requires (for a: domain; b, c: nonzeros)
      Euclidean_function(b * c) >= Euclidean_function(b),
      (for some q, r: domain)
        a = q * b + r
          where
            q = div(a, b),
            r = rem(a, b),
            r=0 or Euclidean_function(r:nonzeros) < Euclidean_function(b).
```

◇
File defined by parts 20ab, 21, 22abc, 23ab, 24abcd, 25.

6.5 Coefficient-ring, Unique-right-ideal, Unique-left-ideal, Ideal, Unique-ideal

"src/algebra2.tec" 23b ≡

```
  Abbreviation: Coefficient-ring is
    Commutative-ring-with-identity [with coefficient-domain as domain].

  Definition: Unique-right-ideal
    refines Right-ideal;
    requires (for I1, I2: ideals) I1 = I2.

  Definition: Unique-left-ideal
```

```
    refines Left-ideal;
    requires (for I1, I2: ideals) I1 = I2.

  Definition: Ideal
    refines Left-ideal, Right-ideal.

  Definition: Unique-ideal
    refines Ideal;
    requires (for I1, I2: ideals) I1 = I2.
```

◇
File defined by parts 20ab, 21, 22abc, 23ab, 24abcd, 25.

6.6 Set-of-pairwise-spanning-ideals, Trivial-ideal, Proper-ideal, Ideal-equivalence

"src/algebra2.tec" 24a ≡

```
  Definition: Set-of-pairwise-spanning-ideals
    refines Ideal;
    requires (for I1, I2: ideals)
      I1 != I2 implies (for a: domain) member(a, I1) or member(a, I2).
```

◇
File defined by parts 20ab, 21, 22abc, 23ab, 24abcd, 25.

"src/algebra2.tec" 24b ≡

```
  Definition: Trivial-ideal
    refines Unique-ideal;
    requires (for I: ideals; a: domain)
      member(a, I) implies a = 0.

  Definition: Proper-ideal
    refines Unique-ideal;
    requires (for I: ideals)
      (for some a: domain) not member(a, I).
```

◇
File defined by parts 20ab, 21, 22abc, 23ab, 24abcd, 25.

6.7 Ideal-equivalence-class, Field, Quotient-ring

"src/algebra2.tec" 24c ≡

```
  Definition: Ideal-equivalence
    refines Equivalence-class;
    uses Unique-ideal;
    requires (for I: ideals; x, y: domain)
      (x equiv y) = member(x - y, I).
```

◇
File defined by parts 20ab, 21, 22abc, 23ab, 24abcd, 25.

"src/algebra2.tec" 24d ≡

```
  Lemma: Euclidean-domain implies Gcd-domain.

  Definition: Ideal-equivalence-class
    refines Equivalence-class [with Ideal-equivalence as Equivalence-relation].
```

```
Definition: Field
  refines Commutative, Division-ring.

Lemma: Field implies Euclidean-domain.
```
◇
File defined by parts 20ab, 21, 22abc, 23ab, 24abcd, 25.

"src/algebra2.tec" 25 ≡

```
Definition: Quotient-ring
  refines
    Commutative-ring-with-identity [with equivalence-classes as domain];
  uses
    Commutative-ring-with-identity [with base-domain as domain],
    Ideal-equivalence-class[with base-domain as domain];
  requires (for e1, e2: equivalence-classes;
               x, x1, x2: base-domain)
    member(x, e1 + e2) = (for some x1, x2: base-domain)
                      member(x1, e1) and member(x2, e2) and x1 + x2 equiv x,
    member(x, (e1 * e2)) = (for some x1, x2: base-domain)
                       member(x1, e1) and member(x2, e2) and x1 * x2 equiv x,
    0 = equivalence-class(0),
    1 = equivalence-class(1).

Extension: Euclidean-domain
  introduces gcdc: domain x domain ->
                   set-of-representatives x domain x domain;
  requires (for x, y, u, v: domain;
               z: set-of-representatives)
    (gcdc(x,y) = (z, u, v)) =
    (z = gcd(x, y) and u * x + v * y = z).
```
◇
File defined by parts 20ab, 21, 22abc, 23ab, 24abcd, 25.

Chapter 7

Ordered Algebras

7.1 Ordered-ring, Ordered-field

"src/ordalgebra.tec" 26a ≡

```
Pragma: include="order.xgf", include="algebra2.xgf".
Library: std
  Ordered-ring, Ordered-field.

Definition: Ordered-ring
  refines Ring-with-identity[with Archimedean-order as Abelian-group];
  requires (for x, y, z: domain)
    x <= y and 0 <= z implies x*z <= y*z.
```

◇
File defined by parts 26abc.

"src/ordalgebra.tec" 26b ≡

```
Precedence: confix{|, |}.

Extension: Ordered-ring
  introduces
    sign: domain -> domain,
    | | : domain -> domain,
    positive : domain -> bool,
    negative : domain -> bool,
    non_positive : domain -> bool,
    non_negative : domain -> bool;
  requires
    (for x: domain)
      sign(x) = if x>0 then +1 else if x<0 then -1 else 0,
      |x| = if x<0 then -x else x,
      positive(x) = (x>0),
      non_positive(x) = (not x>0),
      negative(x) = (x<0),
      non_negative(x) = (not x<0).
```

◇
File defined by parts 26abc.

"src/ordalgebra.tec" 26c ≡

```
Definition: Ordered-field
  refines Field [with Ordered-ring as Ring].
```

◇
File defined by parts 26abc.

Chapter 8

Arithmetic Hierarchy

8.1 Exponentiation, Integer, Rational

"src/arithmetic.tec" 27a ≡

```
Pragma: include="ordalgebra.xgf".
Precedence: {*}<{^}.

Library: std
  Exponentiation, Integer, Rational, Formal-real-field, Real, Complex, Quaternion,
  Integer-congruence-mod-p, Integers-mod-p, Integer-ample-set-mod-p.

Definition: Exponentiation
  uses Monoid, Natural;
  introduces ^ : domain x naturals -> domain;
  requires (for x: domain; n: naturals)
    x ^ 0 = 1,
    x ^ (n + 1) = (x ^ n) * x.
```

◇
File defined by parts 27ab, 28ab, 29ab, 30.

"src/arithmetic.tec" 27b ≡

```
Definition: Integer
  refines
    Nondense-order [with integers as domain],
    Euclidean-domain [with Ordered-ring as Ring,
                      integers as domain,
                      nonzero-integers as nonzeros];
  uses Natural;
  introduces
    naturals < integers,
    nonzero-integers < integers,
    succ : integers -> integers,
    pred : integers -> integers,
    d : naturals x naturals -> integers (private);
  generates integers using d;

  requires
    (for m, n, p, q: naturals; z: domain)
    (d(m, n) = d(p, q)) = (m + q = p + n),
    0 = d(0, 0),
    1 = d(succ(0), 0),
```

```
      succ(d(m, n)) = d(succ(m), n),
      pred(d(m, n)) = d(m, succ(n)),
      d(m, n) + d(p, q) = d(m + p, n + q),
      -(d(m, n)) = d(n, m),
      d(m, n) * d(p, q) = d(m * p + n * q, n * p + m * q),
      (d(m, n) <= d(p, q)) = (m + q <= p + n),
      ((for x : integers) x in naturals = (x >= 0)),
      ((for x : integers) x in nonzero-integers = (x != 0)).
```

◇
File defined by parts 27ab, 28ab, 29ab, 30.

"src/arithmetic.tec" 28a ≡

```
  Definition: Rational
    refines
      Ordered-field [with rationals as domain,
                     nonzero-rationals as nonzeros];
    uses Integer;
    introduces
      integers < rationals,
      fraction : integers x nonzero-integers -> rationals,
      numerator : rationals -> integers,
      denominator : rationals -> nonzero-integers;
    generates rationals using fraction;
    requires (for i, j: integers; k, l: nonzero-integers)
      (fraction(i, k) = fraction(j, l)) = (i * l = j * k),
      (fraction(i, k) <= fraction(j, l)) = (i * l <= j * k),
      0 = fraction(0, 1),
      1 = fraction(1, 1),
      fraction(i, k) + fraction(j, l) = fraction(i * l + j * k, k * l),
      fraction(i, k) * fraction(j, l) = fraction(i * j, k * l),
      numerator(fraction(i, k)) = i, denominator(fraction(i, k)) = k,
      ((for r: rationals) r in integers = (denominator(r)=1)),
      ((for r: rationals) r in nonzero-rationals = (r!=0)).

  Extension: Rational
    introduces
      numerator : nonzero-rationals -> nonzero-integers;
    requires (for r,s: rationals)
      (r <= s) =
        (numerator(r)*denominator(s) <= numerator(s)*denominator(r)),
      -r = fraction(-numerator(r), denominator(r)),
      (for s: nonzero-rationals)
        s^(-1) = fraction(denominator(s), numerator(s)).
```

◇
File defined by parts 27ab, 28ab, 29ab, 30.

8.2 Formal-real-field, Real, Complex, Quaternion

"src/arithmetic.tec" 28b ≡

```
  Definition: Formal-real-field
    refines Ordered-field;
    uses Continuous-order;
    requires
      not ((for some x, y: domain) x*x + y*y = -1).

  Definition: Real
```

```
  refines
    Continuous-order [with reals as domain],
    Ordered-field [with reals as domain, nonzero-reals as nonzeros];
  uses Rational;
  introduces rationals < reals;
  requires (for x: reals)
    x in rationals =
      ((for some i: integers; k: nonzero-integers) x = fraction(i, k)).
```

◇

File defined by parts 27ab, 28ab, 29ab, 30.

"src/arithmetic.tec" 29a ≡

```
Precedence: confix{⌈,⌉}.
Precedence: confix{⌊,⌋}.
Precedence: confix{[, ]}.

// Pragma: requires.

Extension: Real
  introduces
    ⌈⌉ : reals -> integers,
    ⌊⌋ : reals -> integers,
    [] : reals -> integers;
  requires (for r: reals; n: integers)
    ⌈r⌉ = n where n - 1 < r and r <= n,
    ⌊r⌋ = n where n  <= r and r < n + 1,
    [r] = if r>=0 then ⌊r⌋ else ⌈r⌉.
```

◇

File defined by parts 27ab, 28ab, 29ab, 30.

"src/arithmetic.tec" 29b ≡

```
Definition: Complex
  refines Field [with complexes as domain, nonzero-complexes as nonzeros];
  uses Real;
  introduces reals < complexes,
    cp: reals x reals -> complexes,  // a + i*b = z
     i: -> complexes;
  generates complexes freely using cp;
  requires (for m, n, p, q: reals)
    0 = cp(0, 0),
    1 = cp(1, 0),
    i = cp(0, 1),
    cp(m, n) + cp(p, q) = cp(m + p, n + q),
    -(cp(m, n)) = cp(-n, -m),
    cp(m, n) * cp(p, q) = cp(m * p - n * q, m * q + n * p),
    ((for x : complexes) x in reals = ((for some r: reals) x = cp(r,0))),
    ((for x : complexes) x in nonzero-complexes = (x != 0)).

Extension: Complex
  introduces
    real-part: complexes -> reals,
    imag-part: complexes -> reals,
    conjugate: complexes -> complexes,
         sqrt: complexes -> complexes,
         norm: complexes -> reals;
  requires (for c: complexes; a, b: reals)
    real-part(cp(a, b)) = a,
```

```
      imag-part(cp(a, b)) = b,
      conjugate(cp(a, b)) = cp(a, -b),
      (sqrt(c) = a) = (a * a = c),
      norm(cp(a, b)) = sqrt(a * a + b * b).

  Definition: Quaternion
    refines Skewfield [with quaternions as domain,
                            nonzero-quaternions as nonzeros];
    uses Real;
    introduces reals < quaternions, complexes < quaternions,
      qn: reals x reals x reals x reals -> quaternions,
      i: -> quaternions,
      j: -> quaternions,
      k: -> quaternions;
    generates quaternions freely using qn;
    requires (for a, b, c, d, a', b', c', d': reals)
      0 = qn(0, 0, 0, 0),
      1 = qn(1, 0, 0, 0),
      i = qn(0, 1, 0, 0),
      j = qn(0, 0, 1, 0),
      k = qn(0, 0, 0, 1),
      qn(a, b, c, d) + qn(a', b', c', d') =
        qn(a + a', b + b', c + c', d + d'),
      qn(a, b, c, d) * qn(a', b', c', d') =
         qn(a * a' - b * b' - c * c' - d * d',
           a * b' + b * a' + c * d' + d * c',
           a * c' + c * a' + d * b' - b * d',
           a * d' + d * a' + b * c' - c * b'),
      ((for x : quaternions) x in reals
          = ((for some r: reals) x = qn(r,0,0,0))),
      ((for x : quaternions) x in complexes
          = ((for some z: complexes) x = qn(real-part(z),imag-part(z),0,0))),
      ((for x : quaternions) x in nonzero-quaternions = (x != 0)).

  Extension: Quaternion
    introduces conjugate: quaternions -> quaternions,
               norm: quaternions -> reals;
    requires (for q: quaternions; a, b, c, d: reals)
      conjugate(qn(a, b, c, d)) = qn(a, -b, -c, -d),
      norm(qn(a, b, c, d)) = a * a + b * b + c * c + d * d.
```
◇
File defined by parts 27ab, 28ab, 29ab, 30.

8.3 Integer-congruence-mod-p, Integers-mod-p, Integer-ample-set-mod-p

"src/arithmetic.tec" 30 ≡

```
  Pragma: operator.
  Definition: Integer-congruence-mod-p
    refines Equivalence-relation [with equiv as R, integers as domain];
    uses Integer;
    introduces p: -> prime-elements;
    requires (for x, y: integers; d: domain)
      (x equiv y) = p | x - y.

  Lemma:
    Integer-congruence-mod-p implies Equivalence-relation.
```

```
Lemma: Integer-congruence-mod-p
  obeys (for x, x', y, y': integers)
    ((x equiv y) and (x' equiv y')) implies
    (((x + y) equiv (x' + y')) and ((x * y) equiv (x' * y'))).

Definition: Integers-mod-p
  refines Field [with equivalence-classes as domain];
  uses Set-of-representatives
           [with integers as domain,
                 Integer-congruence-mod-p as Equivalence-relation];
  requires (for x, y: equivalence-classes;
                a, b : domain)
    x + y = equivalence-class(representative(x) + representative(y)),
    x * y = equivalence-class(representative(x) * representative(y)),
    0 = equivalence-class(0),
    1 = equivalence-class(1).

Abbreviation: Integer-ample-set-mod-p
  is Set-of-representatives
        [with integers as domain,
              Integer-congruence-mod-p as Equivalence-relation].
```

◇
File defined by parts 27ab, 28ab, 29ab, 30.

Chapter 9

Polynomials

9.1 Polynomials

"src/polynomial.tec" 32a ≡

```
Pragma: include="arithmetic.xgf".
Library: std
  Polynomial, Poly, Polynomial-over-integers, Bivariate-polynomial-over-integers.

Definition: Polynomial
  refines Map [with polynomials as maps,
                    naturals as domain,
                    Coefficient-ring as Range,
                    coefficient-domain as range,
                    c as apply],
          Commutative-ring-with-identity [with polynomials as domain];
  uses Natural;
  introduces nonzero-polynomials < polynomials,
    nonzero     : polynomials -> bool,
    ldcf        : nonzero-polynomials  -> coefficient-domain,
    degree      : nonzero-polynomials -> naturals,
    convolution : polynomials x polynomials x naturals x naturals
                  -> coefficient-domain;
  requires
    (for p, q: polynomials; r: nonzero-polynomials; m, n: naturals)
      p in nonzero-polynomials = nonzero(p),
      nonzero(p) = ((for some n: naturals) c(p, n) != 0),
    (for some n: naturals) (for all m: naturals)
      m > n implies c(p, m) = 0,
      degree(r) = n where (c(r, n) != 0 and
        ((for all m: naturals) m > n implies c(p, m) = 0)),
      ldcf(r) = c(r, degree(r)),
      convolution(p, q, m, 0) = c(p, m) * c(q, 0),
      convolution(p, q, m, n + 1) =
        c(p, m) * c(q, n + 1) + convolution(p, q, m + 1, n),
      c(-(p), n) = -(c(p, n)),
      c(p + q, n) = c(p, n) + c(q, n),
      c(p * q, n) = convolution(p, q, 0, n),
      (p = q) = ((for all n: naturals) c(p, n) = c(q, n)).
```

◇
File defined by parts 32ab, 33ab, 34abc.

"src/polynomial.tec" 32b ≡

```
Lemma: Polynomial
  obeys (for p, q: polynomials)  // for example
    c(p * q, 1) = c(p, 0) * c(q, 1) + c(p, 1) * c(q, 0),
    c(p * q, 2) =
       c(p, 0) * c(q, 2) + c(p, 1) * c(q, 1) + c(p, 2) * c(q, 0).

Lemma: Polynomial
  obeys (for p, q: polynomials; n: naturals)
    c(p - q, n) = c(p, n) - c(q, n).

Precedence: {-|, *}.
```

◇
File defined by parts 32ab, 33ab, 34abc.

9.2 Polynomial Extensions

"src/polynomial.tec" 33a ≡

```
Extension: Polynomial
introduces
   -| : coefficient-domain x polynomials -> polynomials,
   +  : coefficient-domain x polynomials -> polynomials,
   *  : coefficient-domain x polynomials -> polynomials;
requires (for p: polynomials; a: coefficient-domain; n: naturals)
   c(a -| p, n) = if n = 0 then a else c(p, n - 1),
   c(a + p, n) = if n = 0 then a + c(p, 0) else c(p, n),
   c(a * p, n) = a * c(p, n).

Extension: Polynomial
  introduces
     monic-monomial : naturals -> polynomials,
     red            : nonzero-polynomials -> polynomials;
  requires (for r: nonzero-polynomials; m, n: naturals)
     c(monic-monomial(m), n) = if m = n then 1 else 0,
     c(red(r), n) = if n = degree(r) then 0 else c(r, n).
```

◇
File defined by parts 32ab, 33ab, 34abc.

"src/polynomial.tec" 33b ≡

```
Lemma: Polynomial
  obeys (for r: nonzero-polynomials)
    red(r) = r - ldcf(r) * monic-monomial(degree(r)).

Lemma: Polynomial
  obeys (for r: nonzero-polynomials; n: naturals)
    c(red(r), n) =
      c(r - ldcf(r) * monic-monomial(degree(r)), n),
    c(red(r), n) =
      c(r, n)  - c(ldcf(r) * monic-monomial(degree(r)), n),
    c(red(r), n) =
      c(r, n)  - ldcf(r) * c(monic-monomial(degree(r)), n),
    c(red(r), n) =
      c(r, n)  - ldcf(r) * (if degree(r) = n then 1 else 0),
    c(red(r), n) =
      if degree(r) = n then c(r, n) - ldcf(r) * 1
      else c(r, n) - ldcf(r) * 0,
    c(red(r), n) =
```

```
          if degree(r) = n then c(r, n) - ldcf(r) else c(r, n),
        c(red(r), n) =
          if degree(r) = n then c(r, n) - c(r, degree(r))
          else c(r, n),
        c(red(r), n) = if degree(r) = n then 0 else c(r, n).
```

◇
File defined by parts 32ab, 33ab, 34abc.

"src/polynomial.tec" 34a ≡

```
    Extension: Polynomial
      introduces monic-polynomials < nonzero-polynomials;
      requires (for p: nonzero-polynomials)
        p in monic-polynomials = (ldcf(p) = 1).

    Extension: Polynomial
      introduces unit-polynomials < polynomials,
              nonunit-polynomials < polynomials;
      requires (for p: polynomials)
        p in unit-polynomials = p | 1,
        p in nonunit-polynomials = (not p | 1).

    Extension: Polynomial
      introduces prime-polynomials < polynomials;
      requires (for p: polynomials)
        p in prime-polynomials =
          not((for some q, r: nonunit-polynomials) p = q * r).
```

◇
File defined by parts 32ab, 33ab, 34abc.

"src/polynomial.tec" 34b ≡

```
    Abbreviation: Poly is Polynomial.

    Extension: Polynomial
      uses Exponentiation [with Poly as Monoid].
```

◇
File defined by parts 32ab, 33ab, 34abc.

9.3 Polynomial-over-integers, Bivariate-polynomial-over-integers

"src/polynomial.tec" 34c ≡

```
    Abbreviation: Polynomial-over-integers is
      Polynomial [with Integer as Coefficient-ring,
                       integers as coefficient-domain,
                       univariate-polynomials as polynomials].

    Abbreviation: Bivariate-polynomial-over-integers is
      Polynomial [with Polynomial-over-integers as Coefficient-ring,
                       univariate-polynomials as coefficient-domain,
                       bivariate-polynomials as polynomials].
```

◇
File defined by parts 32ab, 33ab, 34abc.

Chapter 10

Ample Sets

10.1 Unit-equivalence, Ample-set, Normal-ample-set, Multiplicative-ample-set

"src/ampleset.tec" 35a ≡

```
Pragma: include="polynomial.xgf".

Precedence: nonassociative{in, into}.
Library: std
  Unit-equivalence, Ample-set, Normal-ample-set, Multiplicative-ample-set,
  Integer-ample-set, Ample-coefficient, Multiplicative-gcd-domain,
  Rational-ample-set, Absolut-value-integer-ample-set, Ample-polynomial,
  Integer-ample-polynomial, Standard-integer-ample-set-mod-p,
  Symmetric-integer-ample-set-mod-p, Normal-integer-ample-set-mod-p.

Definition: Unit-equivalence
  refines Equivalence-class;
  uses Commutative-ring-with-identity, Unit;
  requires (for x, y: domain)
    (x equiv y) = ((for some z: units) x = z * y).

Lemma: Unit-equivalence implies Equivalence-relation.

Abbreviation: Ample-set is
  Set-of-representatives
    [with Unit-equivalence as Equivalence-relation].

Lemma: Ample-set obeys representative(0) = 0.
```

◇
File defined by parts 35ab, 36abc, 37abcd, 38.

"src/ampleset.tec" 35b ≡

```
Definition: Normal-ample-set
  refines Ample-set;
  requires representative(1) = 1.

Definition: Multiplicative-ample-set
  refines Normal-ample-set;
  requires (for x, y: set-of-representatives)
    (for exactly 1 z: set-of-representatives) x*y = z.
```

◇

File defined by parts 35ab, 36abc, 37abcd, 38.

10.2 Integer-ample-set, Ample-coefficient, Multiplicative-gcd-domain

"src/ampleset.tec" 36a ≡

```
Definition: Integer-ample-set
  refines
    Multiplicative-ample-set[with integers as domain];
  uses Integer.

Definition: Ample-coefficient
  refines
    Multiplicative-ample-set
    [with coefficient-domain as domain,
          ample-coefficient-domain as set-of-representatives].

Definition: Multiplicative-gcd-domain
  refines Gcd-domain, Multiplicative-ample-set.
```

◇
File defined by parts 35ab, 36abc, 37abcd, 38.

10.3 Rational-ample-set, Absolut-value-integer-ample-set, Ample-polynomial

"src/ampleset.tec" 36b ≡

```
Definition: Rational-ample-set
  refines Multiplicative-ample-set [with rationals as domain];
  uses Rational, Integer-ample-set;
  requires (for i, j: integers; k, l: nonzero-integers)
    (fraction(i, k) = representative(fraction(j, l))) =
        (fraction(i, k) = fraction(j, l) and
         gcd(i, k) = 1 and representative(k) = k).
```

◇
File defined by parts 35ab, 36abc, 37abcd, 38.

"src/ampleset.tec" 36c ≡

```
Definition: Absolut-value-integer-ample-set
  refines Integer-ample-set, Ordered-ring[with integers as domain];
  requires (for i: integers)
    representative(i) = |i|.

Definition: Ample-polynomial
  refines Polynomial, Ample-coefficient;
  requires (for p: polynomials)
    ldcf(p) : ample-coefficient-domain.
```

◇
File defined by parts 35ab, 36abc, 37abcd, 38.

10.4 Integer-ample-polynomial, Standard-integer-ample-set-mod-p

"src/ampleset.tec" 37a ≡

```
Definition: Integer-ample-polynomial
  refines Ample-polynomial
    [with Polynomial-over-integers as Polynomial,
          integers as coefficient-domain,
          univariate-polynomials as polynomials,
          Integer-ample-set as Ample-coefficient].
```

◇
File defined by parts 35ab, 36abc, 37abcd, 38.

"src/ampleset.tec" 37b ≡

```
Extension: Integer-ample-set-mod-p
  introduces
    +: set-of-representatives x set-of-representatives
                                   -> set-of-representatives,
    *: set-of-representatives x set-of-representatives
                                   -> set-of-representatives;
  requires (for a, b: set-of-representatives)
     a + b = representative(a + b),
     a * b = representative(a * b).

Remark: Lemma: Integer-ample-set-mod-p implies
  Field [with integers as domain,
                      representative(0) as 0,
                      representative(1) as 1].

Realization: Integers-mod-p by Integer-ample-set-mod-p
  introduces rep: set-of-representatives -> equivalence-classes (private);
  requires (for x: set-of-representatives; e: equivalence-classes)
        (rep(x) = e) = (equivalence-class(representative(x)) = e).
```

◇
File defined by parts 35ab, 36abc, 37abcd, 38.

"src/ampleset.tec" 37c ≡

```
Definition: Standard-integer-ample-set-mod-p
  refines Integer-ample-set-mod-p;
  requires (for x: integers)
    representative(x) = rem(x,p).
```

◇
File defined by parts 35ab, 36abc, 37abcd, 38.

10.5 Symmetric-integer-ample-set-mod-p, Normal-integer-ample-set-mod-p

"src/ampleset.tec" 37d ≡

```
Definition: Symmetric-integer-ample-set-mod-p
  refines Integer-ample-set-mod-p;
  uses Real;
  requires (for x: integers)
```

```
      representative(x) =
        if (rem(x,p) = 0)
        then 0
        else if ((x | 2) = true)
             then - ⌊rem(x,p)/2⌋
             else ⌊(rem(x,p))/2⌋ + 1.

  Definition: Normal-integer-ample-set-mod-p
    refines Integer-ample-set-mod-p,
            Normal-ample-set.
```

◇

File defined by parts 35ab, 36abc, 37abcd, 38.

"src/ampleset.tec" 38 ≡

```
  Lemma: Standard-integer-ample-set-mod-p implies
         Normal-integer-ample-set-mod-p.

  Lemma: Symmetric-integer-ample-set-mod-p implies
         Normal-integer-ample-set-mod-p.
```

◇

File defined by parts 35ab, 36abc, 37abcd, 38.

Chapter 11

Morphisms

11.1 Morphisms for Semi-groups

"src/morphism.tec" 39 ≡

```
Pragma: include="algebra2.xgf".
Library: std
  Semigroup-homomorphism, Semigroup-monomorphism, Semigroup-epimorphism,
  Semigroup-embedding, Semigroup-isomorphism, Semigroup-endomorphism,
  Semigroup-automorphism, Ring-homomorphism, Ring-monomorphism, Ring-epimorphism,
  Kernel, Ring-isomorphism.

Definition: Semigroup-homomorphism
  refines Semigroup, Semigroup [with image as domain];
  introduces
    h : domain -> image;
  requires (for x, y: domain)
    h(x*y) = h(x)*h(y).

Definition: Semigroup-monomorphism
  refines
    Semigroup-homomorphism,
    Injection [with h as f, image as range].

Definition: Semigroup-epimorphism
  refines
    Semigroup-homomorphism,
    Surjection [with h as f, image as range].

Abbreviation: Semigroup-embedding is
  Semigroup-monomorphism.

Definition: Semigroup-isomorphism
  refines Semigroup-epimorphism, Semigroup-monomorphism.

Definition: Semigroup-endomorphism
  refines
    Semigroup-epimorphism,
    Semigroup-monomorphism.

Definition: Semigroup-automorphism
  refines
    Semigroup-endomorphism,
    Semigroup-isomorphism.
```

◇

File defined by parts 39, 40.

11.2 Morphisms for Rings

"src/morphism.tec" 40 ≡

```
Definition: Ring-homomorphism
  refines
    Ring-with-identity,
    Ring-with-identity [with image as domain];
  introduces h: domain -> image;
  requires (for x, y: domain)
    h(x+y) = h(x) + h(y),
    h(x*y) = h(x) * h(y),
    h(1) = 1.

Definition: Ring-monomorphism
  refines Ring-homomorphism,
          Injection [with h as f, image as range].

Definition: Ring-epimorphism
  refines Ring-homomorphism,
          Surjection [with h as f, image as range].

Definition: Kernel
  uses Ring-homomorphism;
  introduces ker < domain;
  requires (for x: domain)
    x in ker = (h(x) = 0).

Definition: Ring-isomorphism
  refines Ring-epimorphism, Ring-monomorphism.
```

◇
File defined by parts 39, 40.

Appendix A

Indices and References

Bibliography

[1] Rüdiger G. K. Loos and George E. Collins:*Revised Report on the Algorithm Description Language ALDES*, Technischer Report WSI 92-14, Fakultät für Informatik der Eberhard-Karls-Universität Tübingen, 1992.

[2] Rüdiger Loos and George Collins,*Revised Report on the Algorithm Description Language* Aldes, Technical Report WSI-92-14, Fakultät für Informatik, Universität Tübingen,1992.

[3] Sibylle Schupp, *Generic Programming*—SuchThat *One Can Build an Algebraic Library*, PHD-Thesis, Fakultät für Informatik, Universität Tübingen,1996.

[4] David R. Musser, *The* Tecton *Concept Description Language*, Fakultät für Informatik, Universität Tübingen,July 1998. 1

[5] Deepak Kapur and David R. Musser, Tecton*: A Framework for Specifying and Verifying Generic System Components*, Report 92-20, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, 1992. 1

[6] Alfred Tarski, Introduction to Logic and to the Methodology of Deductive Sciences, Oxford University Press, 1965. 1

[7] Rüdiger G. K. Loos and George E. Collins:*Specifications and Index of SAC-2 Algorithms*, Technischer Report WSI 90-4, Fakultät für Informatik der Eberhard-Karls-Universität Tübingen, 1990.

[8] *The* HOL *Sytem Home Page*, http://www.cl.cam.ac.uk/Research/HVG/HOL/HOL.html. 1

[9] *The* Mizar *Home Page*, http://www.mizar.org. 1

[10] Brian L. Meek: *A taxonomy of datatypes*, ACM SIGPLAN Notices, p. 159-167, September 1994.

[11] Sibylle Schupp: *Generic programming* — SuchThat *one can build an algebraic library*, Dissertation am Wilhelm-Schickard-Institut für Informatik derEberhard-Karls-Universität Tübingen, 1996.

[12] Silicon Graphics Developer Team: *Standard Template Library Programmer's Guide*, 1996. available at: `http://www.sgi.com/Technology/STL`

[13] Alexander Stepanow, Meng Lee: *The Standard Template Library*, Hewlett-Packard Company, Palo Alto, 1995.

[14] D. Kapur and D. R. Musser, *Tecton: a framework for specifying and verifying generic system components*, RPI Computer Science Department Technical Report 92-20, Troy, NY, July 1992.

[15] D. Kapur and D. R. Musser, *Examples of Tecton concept descriptions*, working paper, May, 1992.

[16] D. Kapur, D. R. Musser, and A. A. Stepanov, "Tecton: a language for manipulating generic objects," *Proc. of Program Specification Workshop, University of Aarhus, Denmark*, August 1981, *Lecture Notes in Computer Science*, Springer-Verlag, Vol. 134, 1982.

[17] D. Kapur, D. R. Musser, and X. Nie, "An overview of the Tecton Proof System," *Proc. of a Workshop on Formal Methods in Databases and Software Engineering*, Concordia University, Montreal, May 15-16, 1992.

[18] J. V. Guttag and J. J. Horning, "Report on the Larch Shared Language," *Sci. Comput. Program.*, vol. 6, no. 2, pp. 103-134, Mar. 1986.

[19] J. A. Goguen, T. Winker, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud, "Introducing OBJ," in J.A. Goguen, D. Coleman, and R. Gallimore (editors). *Applications of Algebraic Specification using OBJ*, Cambridge University Press, 1992.