Defining Gröbner bases in the Concept Description Language Tecton

Christoph Schwarzweller
Wilhelm-Schickard-Institute for Computer Science
Sand 13
D-72076 Tübingen
schwarzw@informatik.uni-tuebingen.de

8th November 2001

Abstract

Tecton is a language for describing and using abstract concepts in formal software development and hardware design. Its syntax and semantics only uses first order and inductive proof methods rather than higher order techniques.

In this paper we describe the definition of ideals and Gröbner bases in the concept description language Tecton, so that the specification of Buchberger's algorithm for constructing Gröbner bases can be given. In addition we give Tecton versions of two well-known characterizations for Gröbner bases, namely the one based on division in polynomial rings and the one based on confluent rewriting.

Contents

1 Introduction

Over the last several years generic programming has received more and more attention. Many programming languages nowadays include generic concepts like polymorphism in functional programming languages, overloading or templates in C++; or they are even completely designed as a generic language like SUCH THAT. Also generic libraries have been developed like the ADA Generic Library or the STL. Common to all these approaches is that algorithms are not longer developed for special domains but rather for classes of domains that fulfill the requirements necessary to make the algorithm work.

TECTON [9] is a specification language that takes this view of programming into account: Based on order-sorted algebra it offers language constructs to describe abstract concepts representing the classes of domains with common requirements just mentioned. Algorithms formulated with respect to these concepts then work for all special domains fulfilling the corresponding requirements. This leads to the definition of generic progamming as "programming with concepts" or as "requirement oriented programming", first mentioned by David Musser.

In this paper we present a case study extending the concepts given in the TECTON concept library [10]. The goal is to provide TECTON concepts that enable the specification of Buchberger's well-known algorithm for constructing Gröbner bases in polynomials rings over (arbitrary) fields. This case study does not only give more insight into the use of TECTON and its translator, but also is the first example of an involved algebraic algorithm completely specified using the TECTON language.

The plan of the paper is as follows. After defining ideals in the first section, we introduce linear combinations to formalize the usual characterization of (finitely) generated ideals. Next, we formalize multivariate polynomials and term orders, necessary to lift the natural ordering of univariate to multivariate polynomials.[1] We connect the concept of univariate polynomials, already included in the TECTON concept library, with our approach by giving a realization for it in terms of multivariate polynomials. Then we are ready to define Gröbner bases and to extend the multivariate polynomial concept by s-polynomials and division of polynomials. This allows us to state a first characterization of Gröbner bases in the TECTON language, namely that all s-polynomials with respect to the base divided by the polynomials of the base leave no remainder.

As has been pointed out in the literature, Gröbner bases can also be characterized using the methodology of rewriting systems (see for example [3]). We introduce the basic notion of rewriting systems in the TECTON language and formalize this characterization as a TECTON lemma.

We close with the description of Buchberger's algorithm in the context of the generic programming language SUCHTHAT [12] and a TECTON formalization of the Hilbert basis theorem from which easily follows that every ideal in a polynomial ring over a field has a finite base, and hence a Gröbner base.

[1]In the TECTON concept library (univariate) polynomials already has been defined, but we consider our approach more suitable in order to introduce Gröbner bases (for arbitrary term orders).

2 Ideals

Ideals in (commutative) rings are except for polynomials themselves the basic objects in Gröbner base theory. Here, we introduce ideals for arbitrary rings as it is no problem in TECTON to later use already defined concepts in a more specific context[2]. The TECTON concept library [10] already provides the following definition of ideals.

"Library-Version-of-Ideals.tec" 4a ≡

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

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

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

◇

Here ideals are introduced by importing the concept `Set` with its `sets` being renamed into `ideals`. The two sorts `domain` of the concepts `Set` and `Ring` respectively are indentified, so that the elements of a `sets` — now called `ideals` — can be added and multiplied. However, this definition does not allow to compare `ideals` with arbitrary subsets of the domain just because the type `ideals` does not wide to the type `sets` (over domain).[3] Consequently we use in the following a slightly different definition of ideals in which the concept `Set` is also imported, but the types `lideals`, `rideal` and `ideals` are introduced as a subsort of `sets` (over domain). In addition this gives a much better distinction between left ideals and right ideals on one side and (two-sided) ideals on the other side than the definition from above.

⟨Ideals 4b⟩ ≡

```
Definition: Left-ideal
  refines Set, Ring;
  introduces
    lideals < sets;
  requires (for s: sets)
    s : lideals =
     0 in s,
     (for a,b: domain)
     ((a in s and b in s) implies (a+b in s)) and
     ((a in s) implies (b * a in s)).
```

[2]In fact, this is one of the most important design goals of the TECTON language.

[3]This problem did not occur when defining the first version of the TECTON concept library as the major goal was to provide the basic algebraic structures.

```
    Definition: Right-ideal
      refines Set, Ring;
      introduces
        rideals < sets;
      requires (for s: sets)
        s : rideals =
         0 in s,
         (for a,b: domain)
         ((a in s and b in s) implies (a+b in s)) and
         ((a in s) implies (b * a in s)).

    Definition: Ideal
      refines Left-ideal, Right-ideal;
      introduces
        ideals < lideals,
        ideals < rideals,
        ideals < sets,
        empty-ideal: -> ideals;
      requires
        (for i : lideals) i : ideals = i : rideals,
        (for i : rideals) i : ideals = i : lideals,
        empty-ideal = empty.
◇
```

Definition defined by parts 4b, 5, 6ab.
Definition referenced in part 47.

Note that the axioms of an ideal though of course still present in the new definition play now a different role. Before they were used as additional requirements on the `sets` — or better `ideals` due to the renaming, of the concept — but now they are in fact the requirements a `sets` has to fulfill to be of the (sub)type `lideals` or `rideals`.[4] Note also that the combination of the concepts `Left-ideal` and `Right-ideal` to get the concept `Ideal` has become a bit more involved. `ideals` are introduced as both a subtype of `lideals` and `rideals`, so we have to give the corresponding requirements for both. This was not necessary before just because the types `lideals` and `rideals` were not included.

In commutative rings left and right ideals of course coinside with (two-sided) ideals just because we have $a * b = b * a$ for all elements of the domain. This fact is formalized in Tecton as a lemma:

$\langle$`Ideals 5`$\rangle \equiv$

```
    Abbreviation: Ideal-over-commutative-ring is
      Ideal [with Commutative-ring as Ring].

    Lemma: Ideal-over-commutative-ring
      obeys (for a: lideals) a: ideals,
            (for a: rideals) a: ideals.
◇
```

Definition defined by parts 4b, 5, 6ab.
Definition referenced in part 47.

We also need (finitely) generated ideals, because this is the kind of ideals Buchberger's algorithm deals with. The ideal generated by a given set is the smallest ideal containing

[4]This again points out that there may be further `sets` being not of the type `lideals` or `rideals`, which was the reason for changing the defnition.

this set. This is modelled by introducing a new operator `gen` that transforms a set into the ideal generated by this set. The notion of a finitely generated ideal is similar, and we define it in the TECTON language as a refinement of the concept `Generated-ideal`.

⟨Ideals 6a⟩ ≡

```
Definition: Generated-ideal
  uses Ideal;
  introduces
    gen : sets -> ideals;
  requires (for s: sets; i: ideals)
    (s subset gen(s)),
    (s subset i) implies (gen(s) subset i).

Definition: Finitely-generated-ideal
  refines Generated-ideal [with Finite-set as Set];
  requires (for i: ideals)
    (for some s: finite-sets) i = gen(s).
```

◇
Definition defined by parts 4b, 5, 6ab.
Definition referenced in part 47.

Note that the concept `Finitely-generated-ideal` imports `Generated-ideal` with its subconcept `Set` being replaced by the concept `Finite-set`[5] in order to make the type `finite-sets` available in the concept.

Based on the notion of generated ideals is the concept of an ideal base. A (finite) base for an ideal is a (finite) subset of this ideal that generates it. Note that for each ideal an infinite base indeed exists, namely the ideal itself is a base. In the TECTON language this can be easily expressed by introducing an operator `base` mapping an ideal onto its (finite) base.

⟨Ideals 6b⟩ ≡

```
Definition: Ideal-base
  uses Generated-ideal;
  introduces
    base : ideals -> sets;
  requires (for i: ideals)
    (base(i) subset i) and gen(base(i)) = i.

Definition: Finite-ideal-base
  refines Finitely-generated-ideal, Ideal-base;
  introduces
    base : ideals -> nonempty-finite-sets;
  requires (for i: ideals)
    (base(i) subset i) and gen(base(i)) = i.

Lemma: Finite-ideal-base obeys base(empty-ideal) = empty.
```

◇
Definition defined by parts 4b, 5, 6ab.
Definition referenced in part 47.

Note that the concept `Finite-ideal-base` is not only a refinement of the concepts `Finitely-generated-ideal` and `Ideal-base`,[6] but also introduces the operator `base`

[5]The concept `Finite-set` is a refinement of `set` which means that both types `sets` and `finite-sets` are now available.

[6]In this case it would only follow that for each ideal there is a finite basis, but not that the operator `base` in fact gives such a finite base.

again now returning a finite set. As TECTON allows overloading of operators also by different return types, it is not implicitely clear that this new operator `base` behaves like the former one. Consequently we have to state its requirements again.[7]

This completes the basic definitions concerning ideals in (arbitrary) rings. To define Gröbner bases we need to introduce some more properties of polynomials than already is included in the TECTON concept library. Before this will be done in section 4 we turn to finite sums and linear combinations which are not necessary to define Gröbner bases, but to formalize the already mentioned characterization of Gröbner bases.

3 Linear Combinations

Our next goal is a TECTON formalization of the well-known fact, that the ideal generated by a subset A of a given ring R equals the set of "linear combinatons" over A (see for example [3]):

$$\mathrm{Gen}(A) \;=\; \left\{ \sum_{i=1}^{n} a_i \cdot r_i \;\middle|\; 0 < n \in I\!N,\ r_i \in R \text{ and } a_i \in A \text{ for } 1 \le i \le n \right\}.$$

In the TECTON language, however, a sum of (finitely) many elements is not directly expressible, so we will use the notion of (finite) sequences already realized in the TECTON concept library by maps from the natural numbers into a domain.[8] Because we want to add and multiply the elements given by a (finite) sequence, we have to specialize their ranges: Instead of an arbitrary domain, now the domain has to provide an addition and a multiplication, hence has to be a semiring.

According to the principle of genericity we first define the notion for arbitrary sequences, before we introduce finite sequences over semirings as a refinement of these sequences and finite sequences.

```
⟨LinComb 7⟩ ≡
    Definition: Sequence-over-semiring
      refines Sequence [with Semiring as Range, domain as range];
      uses Semiring;
      introduces
        + : sequences x sequences -> sequences,
        * : domain x sequences -> sequences,
        * : sequences x sequences -> sequences;
      requires (for s,t: sequences; d: domain; m: naturals)
        n_th(s+t,m) = n_th(s,m) + n_th(t,m),
        n_th(d*t,m) = d * n_th(t,m),
        n_th(s*t,m) = n_th(s,m) * n_th(t,m).
◇
```

Definition defined by parts 7, 8ab, 9abc.
Definition referenced in part 47.

When defining finite sequences over a semiring, we have the same problem as just described for finite ideal bases: we have to introduce the operators `+` and `*` again,

[7]Alternatively, instead of introducing the operator `base` again, we could just add the requirement `base(i) : finite-sets`. This also would imply that bases are finite sets in the concept, but the type of `base(i)` would still be `sets` rather than `finite-sets`. We prefer the first solution as it makes things easier when later defining Groebner bases.

[8]see the concepts `Sequence` and `Finite-sequence` in appendix B.1.

because we want to change their types into `finite-sequences`.[9] To ensure that these restricted addition and multiplication behave as expected we add requirements saying that if the arguments of the operators are considered as ordinary sequences — which they can be as `finite-sequences` is a subtype of `sequences` — the result has to be the same as if they were considered as `finite-sequences`.

⟨`LinComb 8a`⟩ ≡

```
    Definition: Finite-sequence-over-semiring
      refines Sequence-over-semiring, Finite-sequence;
      introduces
        + : finite-sequences x finite-sequences -> finite-sequences,
        * : domain x finite-sequences -> finite-sequences,
        * : sequences x finite-sequences -> finite-sequences;
      requires (for s,t: sequences; u,v: finite-sequences; a: domain)
        (s = u and t = v) implies (u + v = s + t and u * v = s * t),
        (s = u) implies (a * u = a * s).
```

◇
Definition defined by parts 7, 8ab, 9abc.
Definition referenced in part 47.

Next we introduce finite sums and products over `Ring` and `Ring-with-identity` respectively.[10] Note that to define `sum` and `prod` for the empty sequence we must use a varible `s` of type `finite-sequence` saying that `s = empty`. This is because the type of `empty` is only `sequence`[11] whereas `sum` and `prod` take `finite-sequences` as their arguments. In fact, avoiding this defect was the reason for repeating the definitions of `finite-base` in the last section and of the operators `+` and `*` for `finite-sequences` from above.[12]

⟨`LinComb 8b`⟩ ≡

```
    Definition: Finite-sum-over-ring
      uses Finite-sequence-over-semiring [with Ring as Semiring];
      introduces
        sum : finite-sequences -> domain;
      requires (for s: finite-sequences; d: domain)
        (s = empty) implies sum(s) = 0,
        sum(d into s) = d + sum(s).

    Definition: Finite-product-over-ring-with-identity
      uses Finite-sequence-over-semiring [with Ring-with-identity as Semiring];
      introduces
        prod : finite-sequences -> domain;
      requires (for s: finite-sequences; d: domain)
        (s = empty) implies prod(s) = 1,
        prod(d into s) = d * prod(s).
```

◇
Definition defined by parts 7, 8ab, 9abc.
Definition referenced in part 47.

[9]Note that here not only the return but also the input types are changed.
[10]These more involved algebraic structures are necessary because `0` and `1` are not available in concept `Semiring`; compare appendix B.3.
[11]compare appendix B.1; maybe this will be changed in the next version of the TECTON concept library.
[12]see also again footnote 7.

If we want to use both sum and prod in one concept this is of course only possible over the more specialized concept Ring-with-identity. The combination is done by refining Finite-product-over-ring-with-identity and Finite-sum-over-ring where the subconcept Ring is replaced by the concept Ring-with-identity.

⟨LinComb 9a⟩ ≡

```
Definition: Finite-sum-and-product-over-ring-with-identity
  refines Finite-sum-over-ring [with Ring-with-identity as Ring],
          Finite-product-over-ring-with-identity.
```

◇
Definition defined by parts 7, 8ab, 9abc.
Definition referenced in part 47.

Now we are ready to define the concept of linear combinations using the just described concept Finite-sum-over-ring. We introduce an operator LinComb mapping a set onto the set of linear combinations over this set.[13] This is done by taking the sums of the finite-sequences having only elements of the form b * c in their ranges, where c is in the given set and b is an arbirary ring element.

⟨LinComb 9b⟩ ≡

```
Definition: Linear-combination
  uses Finite-sum-over-ring, Ideal;
  introduces
    LinComb : sets -> ideals;
  requires (for a: domain; s1: sets)
    (a in LinComb(s1)) =
       (for some s: finite-sequences) a = sum(s) and
          (for n: naturals) (for some b,c: domain)
                      (c in s1) and (n_th(s,n) = b * c).
```

◇
Definition defined by parts 7, 8ab, 9abc.
Definition referenced in part 47.

Before we can state the desired property of linear combinations, namely the the set (or better the ideal) of linear combinations over a given set s equals the ideal generated by s, there is one more thing to do: we have to make the notion of generated ideals, that is the operator gen, available.[14] This is done by an abbreviation in which the concept Ideal imported by the concept Linear-combination is replaced by the concept Genrated-ideal.[15]

⟨LinComb 9c⟩ ≡

```
Abbreviation: Linear-combination-with-generated-ideal is
  Linear-combination [with Generated-ideal as Ideal].

Lemma: Linear-combination-with-generated-ideal obeys
  (for s: sets) gen(s) = LinComb(s).
```

◇
Definition defined by parts 7, 8ab, 9abc.
Definition referenced in part 47.

[13]Note that the return type of LinComb is ideals implicitely saying that the set of all linear combinations over a set fulfills the axioms of an ideal. This also is the reason for importing the concept Ideal; with LinComb having simply sets as its retrun type this would not be necessary.

[14]We also could do this in the definition of the concept Linear-combination by importing the concept Genrated-ideal instead of Ideal, but we prefer to be in a definition as general as possible.

[15]Note that this implies additional semantic requirements, see [9]. Here, these requirements are trivially fulfilled just because the concept Generated-ideal imports the concept Ideal.

4 Term Orders

In section we give a Tecton formalization of term orders[16] necessary to lift the natural ordering on univariate to multivariate polynomials that are defined in the next section. We start with the definition of well orders that is not included in the Tecton concept library so far. A well order is an order in which no infinite chains are possible. Consequently the concept `Well-order` refines the concept `Partial-order` by introducing a predicate `allows_infinite_chains` that checks whether a particular element of the domain is part of an infinite chain. Well-foundness is then achieved by requiring that no element fulfills this predicate.

⟨Order 10a⟩ ≡

```
Definition: Well-order
  refines Partial-order;
  introduces
    allows_infinite_chains : domain -> bool;
  requires (for a: domain)
    allows_infinite_chains(a) =
      ((for some b: domain) b < a) and
      ((for b: domain) b < a implies (for some c: domain) c < b),
    not(allows_infinite_chains(a)).
```

◇
Definition defined by parts 10abc, 11a.
Definition referenced in part 47.

Now we can define a term order as a total well-founded order over a monoid[17] fulfilling two additional properties, namely that `1` is the smallest element and that the order is compatible with the multiplication, which is easily translated in the Tecton language as folllows.

⟨Order 10b⟩ ≡

```
Definition: Term-order
  refines Total-order, Well-order;
  uses Monoid;
  requires (for a,b,c: domain)
    1 <= a,
    a <= b implies a * c <= b * c.
```

◇
Definition defined by parts 10abc, 11a.
Definition referenced in part 47.

An easy consequence of the last definition is the following lemma, which in fact is not necessary for our development of Gröbner bases, but we include it as another example for the Tecton lemma construct.

⟨Order 10c⟩ ≡

```
Lemma: Term-order obeys
  (for a,b,c,d: domain) (a <= b and c <= d) implies a * c <= b * d.
```

◇
Definition defined by parts 10abc, 11a.
Definition referenced in part 47.

[16]also called monomial orders; see [6].

[17]Note that the set $T_n := \{X_1^{i_1} \cdots X_n^{i_n}\}$ of monomials over the set $\{X_1, \ldots X_n\}$ forms a monoid with respect to concatenation; see [3].

Later we need to talk about the maximal monomial of a polynomial. As the TECTON concept library only provides a maximum operator for orders on numbers[18] we introduce it here for general orders. Note, that according to the principle of genericity we introduce the `max` operator not in an extenstion of the concept `Term-order`, but of the concept `Total-order`. Nevertheless, because `Term-order` imports `Total-order` the operator `max` will be available there also.

⟨Order 11a⟩ ≡
```
Extension: Total-order
  uses Set;
  introduces
    max : domain x domain -> domain,
    max : sets -> domain;
  requires (for a,b: domain; s: sets)
    (max(a,b) = a) = (b <= a),
    (max(s) = a) = ((for b: domain) b in s implies b <= a).
```
◇
Definition defined by parts 10abc, 11a.
Definition referenced in part 47.

5 Multivariate Polynomials

As already mentioned (univariate) polynomials are already included in the TECTON concept library [10]. There they have been modelled as functions from the natural numbers into a coefficient domain, namely a commutative ring with identity.[19] Polynomial rings with more then one variable then can be inductively defined by taking a polynomial ring as the coefficient domain. Nevertheless we decided to use a different approach following [11]: we first introduce bags as functions from a (possibly infinite) set of variables into the natural numbers giving us the notion of power products.[20] Consequently a multivariate polynomial is a function from this set of bags into a coefficient domain with only finitely many values of the function being not zero. This has two advantages: first, we found out that using our apporach it is easier to formalize special orderings for monomials (with an arbitrary number of variables); second, the approach is more general. Not only that the case of infinitely many variables is included, we also easily get the notion of power series (with infinitely many variables) by first dropping the requirement that only finitely many values of the bags are not zero. Multivariate polynomials then can be defined as a refinement of power series.

We start with introducing the concept of variables, that are in fact nothing more than an arbitrary set with its `domain` being renamed in `variables`. Please note again that the number of variables may be infinite in our concept.

⟨Poly 11b⟩ ≡
```
Abbreviation: Variable is Set [with variables as domain].
```
◇
Definition defined by parts 11b, 12abc, 13ab, 14ab, 15ab, 16abc, 17a.
Definition referenced in part 47.

[18]compare appendix B.1 and B.2.

[19]compare appendix B.4.

[20]In fact a bag is almost the same as a yet to be introduced monomial, but we want to have a type `monomials` being a subtype of the type `polynomials`, so we cannot use the definition of bags as the definition for monomials.

Bags are modelled as functions from `variables` into the natural numbers using the concept `Map` already included in the TECTON concept library.[21] All that has to be done is to rename `Map`'s parameters appropriately. Note that we also import the concept `Monoid`[22] in order to make available the concatenation of bags, that is the concatenation of power products.

⟨Poly 12a⟩ ≡

```
  Definition: Bag
    refines Map [with bags as maps, variables as domain,
                      Natural as Range, naturals as range],
            Monoid [with bags as domain];
```

◇
Definition defined by parts 11b, 12abc, 13ab, 14ab, 15ab, 16abc, 17a.
Definition referenced in part 47.

Power products are finite that is there are only finitely many variables having an exponent unequal to zero. So we have to formulate further requirements for the concept `Bag`. We introduce an operator `support` from bags into finite sets (of variables) giving exactly the variables for which a bag has a non zero result.[23] Note that by the operator's retrun type `finite-sets` we implicitely require that there are only finitely many variables with this property. In addition we have to identify the operators `*` and `1` given by the monoid structure with the corresponding operations on bags.

⟨Poly 12b⟩ ≡

```
    uses Finite-set [with variables as domain];
    introduces
      support : bags -> finite-sets;
    requires (for b,b': bags; v: variables)
      v in support(b) = not(apply(b,v) = 0),
      apply(b*b',v) = apply(b,v) + apply(b',v),
      (b = 1) = (for v: variables) apply(b,v) = 0.
```

◇
Definition defined by parts 11b, 12abc, 13ab, 14ab, 15ab, 16abc, 17a.
Definition referenced in part 47.

Now we are ready to introduce power series as functions from bags into a coefficient domain. Again this is done by instantiating the parameters of the concept `Map`. In an extension we redefine the `support` function giving here the bags of a series with non zero values. This set may be infinite, hence its return type is `sets` rather than `finite-sets`.

⟨Poly 12c⟩ ≡

```
  Abbreviation: Power-series is
    Map [with series as maps, bags as domain,
              Coefficient-ring as Range, coefficient-domain as range].

  Extension: Power-series
    uses Finite-set [with bags as domain];
    introduces
      support : series -> sets;
```

[21]compare appendix B.1.
[22]compare appendix B.2.
[23]compare [11].

```
      requires (for s: series; b: bags)
        (b in support(s)) = (not(apply(s,b) = 0)).
```
◇
Definition defined by parts 11b, 12abc, 13ab, 14ab, 15ab, 16abc, 17a.
Definition referenced in part 47.

Polynomials are the power series having only finitely many bags with a value not equal to zero. Consequently the concept `Multivariate-polynomials` is a refinement of the concept `Power-series` in which the retrun type of the operator `support` is restricted to `finite-sets`.[24] Again the operator's return type `finite-sets` implicitely serves as a requirement.[25] We also import the concept `Gcd-domain` making available in particular the usual ring operators for polynomials. The concept `Finite-sum-over-ring` is necessary here only to define multiplication of polynomials.

⟨Poly 13a⟩ ≡

```
    Definition: Multivariate-polynomial
      refines Power-series [with polynomials as series],
              Gcd-domain [with polynomials as domain];
      uses Bag, Finite-sum-over-ring [with coefficient-domain as domain];
      introduces
        support : polynomials -> finite-sets,
```
◇
Definition defined by parts 11b, 12abc, 13ab, 14ab, 15ab, 16abc, 17a.
Definition referenced in part 47.

Now we have to connect the ring operators imported from the concept `Gcd-domain` with their corresponding operations on polynomials. The most work causes of course the definition of multiplication: to get the value of a bag `b` under the product of two polynomials `p` and `q` one has to collect all bags `b1` and `b2` with `b1 * b2 = b` and to multiply the value of `b1` under `p` with the value of `b2` under `q`, that is one has to build `apply(p,b1) * apply(q,b2)`. The sum of all these products then gives the desried value for the bag `b`. We model this using a helper function called `product-sequence` taking two polynomials and a bag as its input returning a finite sequence of the just described products.[26] Please note that in this definition we use the TECTON construct `(for exactly 1 n: naturals)` avoiding the duplication of products in the finite sequence. The sum of the elements occurring in this finite sequence, that is the sum of all collected products — giving the value of the bag `b` under the product of the two given polynomials — is then built with the `sum` operator of the concept `Finite-sum-over-ring`.[27]

⟨Poly 13b⟩ ≡

```
        product-sequence :
           polynomials x polynomials x bags -> finite-sequences (private);
      requires (for p,q: polynomials; b: bags; f: finite-sequences)
        (product-sequence(p,q,b) = f) =
           ((for b1,b2: bags)
            (b1 * b2 = b) =
            (for exactly 1 n: naturals) n_th(f,n) = apply(p,b1) * apply(q,b2)),
```

[24] compare the discussion on changing return types of and reintroducing operators in footnote 7.

[25] compare the definition of the concept `Bag`.

[26] The function `product-sequence` is declared `(private)` which means that it is only available in the concept `Multivariate-polynomial` and not in other concepts that import `Multivariate-polynomial`. This is the usual TECTON mechanism to hide internal helper functions.

[27] The necessity to build this sum was the only reason to (define and) import the concept `Finite-sum-over-ring` into the definition of `Multivariate-polynomial`.

```
        apply(p*q,b) = sum(product-sequence(p,q,b)),
```
◇
Definition defined by parts 11b, 12abc, 13ab, 14ab, 15ab, 16abc, 17a.
Definition referenced in part 47.

The other algebraic operations of polynomials, namely addition, equality, the zero and the unit polynomial, and the definition of the `support` of a polynomial `p` can be straightforward translated in the TECTON language as follows.

⟨Poly 14a⟩ ≡
```
        apply(p+q,b) = apply(p,b) + apply(q,b),
        (p = q) = (for b: bags) apply(p,b) = apply(q,b),
        (p = 0) = (for b: bags) apply(p,b) = 0,
        (p = 1) = (apply(p,1) = 1 and
                   (for b: bags) not(b = 1) implies apply(p,b) = 0),
        (b in support(p)) = (not(apply(p,b) = 0)).
```
◇
Definition defined by parts 11b, 12abc, 13ab, 14ab, 15ab, 16abc, 17a.
Definition referenced in part 47.

The following definition extends the concept `Multivariate-polynomial` with some additional types needed later: `nonzero-polynomials`, `monomials` which are polynomials with only one power product, that is only one bag gives a value not equal to zero, `nonzero-monomials` and `monic-monomials` which are monomials having `1` as their (only) coefficient, that is the value of the bag not giving zero is `1`. Note that all are subtypes of type `polynomials` and that we have to state requirements characterizing the subtypes.[28]

⟨Poly 14b⟩ ≡
```
    Extension: Multivariate-polynomial
      introduces
        nonzero-polynomials < polynomials,
        monomials < polynomials,
        nonzero-monomials < monomials,
        nonzero-monomials < nonzero-polynomials,
        monic-monomials < nonzero-monomials,
        monic-monomials < monomials;
      requires (for p: polynomials; q: nonzero-polynomials; m: monomials)
        p : nonzero-polynomials = not(support(p) = empty),
        p : monomials = (for some b: bags) support(p) = singleton(b),
        q : nonzero-monomials = q : monomials,
        m : nonzero-monomials = m : nonzero-polynomials,
        m : monic-monomials = (for some b: bags) apply(m,b) = 1.
```
◇
Definition defined by parts 11b, 12abc, 13ab, 14ab, 15ab, 16abc, 17a.
Definition referenced in part 47.

The next natural step is to introduce the notion of headterms, headmonomials and headcoefficients for polynomials. But this is not possible in the case of multivariate polynomials without introducing an order on the (multivariate) monomials.[29] Therefore we employ the concept `Term-order` defined in section 4: we refine the concept

[28]The type expression `monic-monomials < monomials` should not be necessary as the TECTON subtype relation is transivtive, but without the TECTON type checker does not accept `monic-monomials` as a subtype of `monomials`.

[29]Remember that univariate monomials are naturally ordered by their exponents, that is we have $X^n < X^m \iff n < m$.

`Multivariate-polynomial` by importing `Term-order` with its `domain` being replaced by `monomials`.

⟨`Poly 15a`⟩ ≡

```
    Definition: Multivariate-polynomial-with-monomial-order
      refines Multivariate-polynomial,
              Term-order [with monomials as domain].
```

◇

Definition defined by parts 11b, 12abc, 13ab, 14ab, 15ab, 16abc, 17a.
Definition referenced in part 47.

Note that the above definition makes available a term order on monomials without saying anything about the realization of the order.[30] But this does not matter: to introduce headterms, headmonomials and headcoefficients for polynomials (and later Gröbner bases) all we need to know is that there is such an order. The details can be filled in later.[31]

It turned out that in order to define headterms, headmonomials and headcoefficients for polynomials we need two additional operators: First, the set of monomials appearing in a given polynomials and, second, another multiplication, namely the multiplication of an element of the coefficient domain with a polynomial. Both are introduced in the following extension of the concept `Multivariate-polynomial`.[32]

⟨`Poly 15b`⟩ ≡

```
    Extension: Multivariate-polynomial
      uses Finite-set [with monomials as domain];
      introduces
        monoms : polynomials -> finite-sets,
        * : coefficient-domain x polynomials -> polynomials;
      requires (for p: polynomials; m: monomials;
                    b: bags; a: coefficient-domain)
        m in monoms(p) =
          (for some b: bags) apply(m,b) = 1 and not(apply(p,b) = 0),
        apply(a*p,b) = a * apply(p,b).
```

◇

Definition defined by parts 11b, 12abc, 13ab, 14ab, 15ab, 16abc, 17a.
Definition referenced in part 47.

Now it is straightforward to introduce operators `HT`, `HM` and `HC` giving the headterm, the headmonomial and the headcoefficient of a polynomial respectively.[33] Note that the result type of `HT` is `monomials` and that of `HM` is `monic-monimials` which means that both can be widened to the type `polynomials`.[34]

[30]In fact the definition describes the concept of multivariate polynomials with an arbitrary but fixed term order on the monomials; compare the semantics of Tecton in [9].

[31]In appendix A we give an example of how to do this by introducing a total order on the variables and lifting this order to monomials.

[32]Note that this is not an extension of the concept `Multivariate-polynomial-with-monomial-order` as the properties do not depend on a monomial order. Nevertheless, due to Tecton's inheritance mechanism the introduced operators `monoms` and `*` will be available there, too.

[33]We like to mention that the result of these three operators may vary depending on the monomial order being used, that is instantiating different orders in our concept may result in different operators `HT`, `HM` and `HC`; compare e.g. [6].

[34]In fact also the return type of `HC` could be widened to the type `polynomials` by identifying an element of the coefficient domain with a constant polynomial, but this is not necessary in our context.

⟨Poly 16a⟩ ≡

```
Extension: Multivariate-polynomial-with-monomial-order
  introduces
    HT : polynomials -> monomials,
    HM : polynomials -> monic-monomials,
    HC : polynomials -> coefficient-domain;
  requires (for p: polynomials; b: bags; a: coefficient-domain)
    HT(p) = HC(p) * HM(p),
    HM(p) = max(monoms(p)),
    (HC(p) = a) =
      (for some b: bags) apply(HM(p),b) = 1 and apply(p,b) = a.
```

◇

Definition defined by parts 11b, 12abc, 13ab, 14ab, 15ab, 16abc, 17a.
Definition referenced in part 47.

The just defined order on monomials can be lifted to a partial order on polynomials by recursively comparing the monomials occuring in a polynomial. To do so we define the reductum `Red` of a polynomial `p` which is nothing more than `p` without its headterm. For completion[35] we introduce in addition the type `monic-polynomials`.[36]

⟨Poly 16b⟩ ≡

```
Extension: Multivariate-polynomial-with-monomial-order
  uses Partial-order [with polynomials as domain];
  introduces
    monic-polynomials < nonzero-polynomials,
    Red : nonzero-polynomials -> polynomials;
  requires (for p: polynomials; q,q': nonzero-polynomials)
    q : monic-polynomials = (HT(q) = 1),
    Red(q) = q - HT(q),
    0 < q,
    (q < q') = (HT(q) < HT(q') or (HT(q) = HT(q') and Red(q) < Red(q'))).
```

◇

Definition defined by parts 11b, 12abc, 13ab, 14ab, 15ab, 16abc, 17a.
Definition referenced in part 47.

We close this section with another look at univariate polynomials. As already mentioned the concept `Polynomial` of univariate polynomials is already included in the Tecton concept library.[37] In the following we give a realization of univariate polynomials using our approach of multivariate polynomials. We start with renaming the concept of univariate polynomials. This is necessary because both concepts `Polynomial` and `Multivariate-polynomial` use the type `polynomials`.

⟨Poly 16c⟩ ≡

```
Abbreviation: Univariate-polynomial is
  Polynomial [with u-polynomials as polynomials].
```

◇

Definition defined by parts 11b, 12abc, 13ab, 14ab, 15ab, 16abc, 17a.
Definition referenced in part 47.

A Tecton realization connects two already defined concepts. It introduces a representation function `rep` mapping more general objects onto the objects being realized.

[35]This type is not necessary for our development

[36]Note that this could not have been done before as we need the notion of headterm to do so.

[37]see appendix B.4.

Requirements have to be given so that the resulting concept is indeed part of the semantics of the realized concept (see [9]). In our case multivariate `polynomials` are mapped onto univariate `u-polynomials`. The first additional requirement is of course that there is only one variable `v`.[38] The second clause desribes when a multivariate polynomial `p` is a prepresentation of a univariate polynomial `up`; to be more precise, this holds if and only if all coefficients of `p` giving by a bag `b` that evaluates to the natural number `n` coincide with the coefficients of `up` at the natural number `n`.[39]

⟨Poly 17a⟩ ≡

```
Realization: Univariate-polynomial by Multivariate-polynomial
  introduces
    rep : polynomials -> u-polynomials (private);
  requires (for p: polynomials; up: u-polynomials)
    (for some v: variables) (for v': variables) v' = v,
    (rep(p) = up) =
      ((for b: bags; a: coefficient-domain; v: variables; n: naturals)
       (apply(p,b) = a and apply(b,v) = n) = (c(up,n) = a)).
```

◇

Definition defined by parts 11b, 12abc, 13ab, 14ab, 15ab, 16abc, 17a.
Definition referenced in part 47.

6 Gröbner bases

Gröbner bases are finite ideal bases with the additional porperty that polynomial division describes the ideal membership, that is if $G = \{g_1, \ldots g_n\}$ is a Gröbner base for the ideal I [40] we have for all polynomials f

$$f \in I \iff \mathrm{rem}(f; g_1, \ldots g_n) = 0$$

where $\mathrm{rem}(f; g_1, \ldots g_n)$ stands for the remainder of the division of the polynomials f by the polynomials $g_1, \ldots g_n$. The definition of Gröbner bases, however, is usually stated somewhat different using headterms giving the above equivalence as a theorem: G is a Gröbner base for the ideal I if and only if (see [6])

$$< HT(g_1), \ldots HT(g_n) > = < HT(I) >,$$

that is the headterms of the base $\{g_1, \ldots g_n\}$ for the ideal I generate the same ideal as all headterms occuring in I. We follow this approach, so we first have to introduce the headterm set of a given set of polynomials. This is done in an extension of the concept `Multivariate-polynomial-with-monomial-order` as the concept `Multivariate-polynomial` does not provide the notion of headterms.[41]

⟨Groebner 17b⟩ ≡

```
Extension: Multivariate-polynomial-with-monomial-order
  uses Set [with polynomials as domain];
```

[38]Note that this implies that the realization is in some sense a refinement of the concept `Multivariate-polynomial`.

[39]Remember that univariate polynomials are modelled as functions from the natural numbers into a coefficient domain.

[40]Usually I is taken as $< g_1, \ldots g_n >$.

[41]compare the last section.

```
      introduces HTs : sets -> sets;
      requires (for p: monomials; s: sets)
        (p in HTs(s)) =
          ((for some r: nonzero-polynomials) (r in s and p = HT(r))).
```
◇
Definition defined by parts 17b, 18, 19ab, 20abc, 21ab, 22abc, 23a.
Definition referenced in part 47.

Before we can define Göbner bases we first have to specialize our concept of polynomials a bit: Gröbnerbases are defined for polynomial rings with finitely many variables over fields[42], but our concepts so far provide infinitely many variables and use commutative rings with identity as coefficient domain[43] So we define two new concepts — one for finitely many variables[44] and one for fields as coefficient domains — and glue them in a third concept together with our multivariate polynomial concept.[45] Of course we then have to replace the subconcept `Coefficient-ring` by concept `Field` in the two other concepts, too.

⟨`Groebner 18`⟩ ≡
```
    Definition: Multivariate-polynomial-with-finitely-many-variables
      refines Multivariate-polynomial;
      requires (for s: sets)
        (for some t: finite-sets) (for v: variables) v in t.

    Abbreviation: Multivariate-polynomial-over-field is
      Multivariate-polynomial [with Field as Coefficient-ring].

    Definition: Multivariate-polynomial-with-finitely-many-variables-over-field
      refines Multivariate-polynomial-with-finitely-many-variables
                                  [with Field as Coefficient-ring],
              Multivariate-polynomial-over-field,
              Multivariate-polynomial-with-monomial-order
                          [with Field as Coefficient-ring].
```
◇
Definition defined by parts 17b, 18, 19ab, 20abc, 21ab, 22abc, 23a.
Definition referenced in part 47.

Now we are ready to formalize the concept of Gröbner bases in the TECTON language. It refines the concept `Finite-ideal-base` by replacing the subconcept `Ring` with the appropriate just defined ring structure `Multivariate-polynomial-with-finitely-many-variables-over-field`. In addition it is required that the bases given by the operator `base` fulfill the property of Gröbner bases from the beginning of this section, that is the headterms of the base for the ideal `i` generate the same ideal as all headterms occuring in `i`. A predicate `Groebnerbase` indicating whether a finite set of polynomials is a Gröbner base is also defined.

[42]It may be interesting to investigate whether Gröbner bases always exist with weaker restrictions.

[43]In fact this can also be weakened, but we decided to use the concept `Coefficient-domain` included in the TECTON concept library.

[44]Please note the dummy `(for s: sets)` here, without which the TECTON type checker does not accept the definition of `Multivariate-polynomial-with-finitely-many-variables`.

[45]Note that we use the concept `Multivariate-polynomial-with-monomial-order` in the last definition. We could also first use `Multivariate-polynomial` only introducing the necessary monomial order in a second step, but compared with the above two first definitions — which for sure exist on their own merit — this seems a bit overgeneralizing to us.

⟨Groebner 19a⟩ ≡

```
Definition: Groebner-base
  refines Finite-ideal-base
    [with Multivariate-polynomial-with-finitely-many-variables-over-field
           as Ring,
          polynomials as domain];
  introduces
    Groebnerbase : sets x ideals -> bool;
  requires (for i: ideals; s: finite-sets)
    gen(HTs(base(i))) = gen(HTs(i)),
    Groebnerbase(s,i) = (gen(s) = i and gen(HTs(s)) = gen(HTs(i))).
```

◇

Definition defined by parts 17b, 18, 19ab, 20abc, 21ab, 22abc, 23a.
Definition referenced in part 47.

We end this section with a well-known lemma, extending most definitions of Gröbner bases in which the empty set is excluded (see e.g. [6]).

⟨Groebner 19b⟩ ≡

```
Lemma: Groebner-base obeys Groebnerbase(empty,empty-ideal).
```

◇

Definition defined by parts 17b, 18, 19ab, 20abc, 21ab, 22abc, 23a.
Definition referenced in part 47.

7 Polynomial Division and S-Polynomials

Now that we have defined Gröbner bases in TECTON, we want to formalize the already mentioned characteriztion, namely that a polynomial p is in the ideal generated by a Gröbner base $G = \{g_1, \ldots g_n\}$ if and only if the division of p by $g_1, \ldots g_n$ leaves no remainder. In fact, we go even further: we introduce s-polynomials (see [5]) and formalize the famous Gröbner base characterization based on them: a set $G = \{g_1, \ldots g_n\}$ is a Gröbner base for $< g_1, \ldots g_n >$ if and only if all s-polynomials that can be built with polynomials out of G leave no remainder if divided by $g_1, \ldots g_n$.

This required some prelimanary work, namely to define the division with rest for polynomials. The goal is to divide a (multivariate) polynomial f by polynomials $g_1, \ldots g_n$ resulting in n quotients q_n and a remainder r such that

$$f \;=\; q_1 \cdot g_1 + \cdots + q_n \cdot g_n + r.$$

Note that $r = 0$ obviouly implies $f \in < g_1, \ldots g_n >$. The problem is that in general division of polynomials is not unique: it depends on the order of the g_i. In particular, the remainder r may be zero or not according to g_i chosen first to divide f.[46] In addition we have to provide the polynomials $\{g_1, \ldots g_n\}$ with an order to keep track of which q_j belongs to which g_i. This will be done using the concept of finite sequences. So we start with extending the concept `Finite-sequence` with operators `len` giving the length of a finite sequence and `rng` giving the (finite) set of the domain elements occuring in a finite sequence later needed.[47]

[46] Exactly these defects are repaired when taking a Gröbner base as the set $\{g_1, \ldots g_n\}$; see [6].

[47] Note again that we cannot write `len(empty) = 0` nor `rng(empty) = empty` as a requirement because `empty` is not of type `finite-sequences`, but only of `sequences`.

⟨Groebner 20a⟩ ≡

```
    Extension: Finite-sequence
      uses Finite-set;
      introduces
        len : finite-sequences -> naturals,
        rng : finite-sequences -> finite-sets;
      requires (for d: domain; s: finite-sequences)
        (s = empty) = (len(s) = 0),
        len(d into s) = len(s) + 1,
        (s = empty) = (rng(s) = empty),
        rng(d into s) = d into rng(s).
```

◇

Definition defined by parts 17b, 18, 19ab, 20abc, 21ab, 22abc, 23a.
Definition referenced in part 47.

Now we can introduce operators `div` and `mod` giving the quotients and the remainder of a division, respectively. Note that the return type of `div` is `finite-sequences`. These are indeed finite sequences over `polynomials` because the concept `Finite-sum- over-ring` — necessary to make the `sum` operator available — is imported with its `domain` being replaced by `polynomials`. This implies that in concept `Finite-sequence` imported by `Finite-sum-over-ring`[48] the `domain` also is replaced by `polynomials`. The concept `Regular` is imported because it provides the notion of divisibility we need to state the requirements for polynomial division.[49]

⟨Groebner 20b⟩ ≡

```
    Precedence: {div, mod} < {+}.

    Extension: Multivariate-polynomial-with-monomial-order
      uses Finite-sum-over-ring [with polynomials as domain],
           Regular [with monomials as domain];
      introduces
        div : polynomials x finite-sequences -> finite-sequences,
        mod : polynomials x finite-sequences -> polynomials;
```

◇

Definition defined by parts 17b, 18, 19ab, 20abc, 21ab, 22abc, 23a.
Definition referenced in part 47.

Here are the already mentioned requirements: the first one states that the number of quotients equals the number of divisors, the second describes the equation of polynomial division from above, namely that the sum of the products $q_i \cdot g_i$ — here realized by using the operators `*` and `sum` for finite sequences imported by `Finite-sum-over-ring` — plus the remainder of the division equals the original polynomial p. The last requirement states that the division is indeed "completed", that is the remainder is not divisible by any of the polynomials g_i occuring in the finite sequence `s`. This is established by checking whether no monomial in the remainder may be divided by a headmonomial of a polynomial `q` occurring in `s`.

⟨Groebner 20c⟩ ≡

```
      requires (for p: polynomials; m: monomials; s: finite-sequences)
```

[48]This also is the reason that we need not import the concept `Finite-sequence` explicitely in order to make `len` and `rng` available.

[49]Precedences as here given for the operators `div` and `mod` are not part of the entire TECTON language (see [9]). They are additional information for the TECTON type checker allowing to save brackets when formulating the requirements.

```
        len(p div s) = len(s),
        sum((p div s) * s) + (p mod s) = p,
        m in monoms(p mod s) implies
         (not (for some q: polynomials) (q in rng(s) and (HM(q)|m))).
```
◇
Definition defined by parts 17b, 18, 19ab, 20abc, 21ab, 22abc, 23a.
Definition referenced in part 47.

The last problem yet to solve in order to formalize the characteriztion of Gröbner bases using polynomial division is concerned with types: a Gröbner base is a finite subset of a polynomial ring whereas the polynomials used for division are stored in a finite sequence. Hence, if we want to divide a polynomial using the polynomials of a (Gröbner) base as divisors, we have to transform them into a finite sequence first. This is done by introducing two operators `set_to_seq` — one for empty and the other for non empty fiinite sets. In addition we have to extend the concept `Groebner-base` by the concept `Finite-sequence` to make the just defined opertor `set_to_seq` available.[50]

⟨Groebner 21a⟩ ≡
```
    Extension: Finite-sequence
      introduces
        set_to_seq : finite-sets -> finite-sequences,
        set_to_seq : nonempty-finite-sets -> nonempty-finite-sequences;
      requires (for d: domain; t: finite-sets; t': nonempty-finite-sets)
        (t = empty) = (set_to_seq(t) = empty),
        set_to_seq(d into t) = d into set_to_seq(t),
        t = t' implies set_to_seq(t') = set_to_seq(t).

    Extension: Groebner-base
      uses Finite-sequence [with polynomials as domain].
```
◇
Definition defined by parts 17b, 18, 19ab, 20abc, 21ab, 22abc, 23a.
Definition referenced in part 47.

Finally, we can state the desired property of Gröbner bases, namely that a polynomial p is in the ideal generated by a Gröbner base $G = \{g_1, \dots g_n\}$ if and only if the division of p by $g_1, \dots g_n$ leaves no remainder, as a TECTON lemma.

⟨Groebner 21b⟩ ≡
```
    Lemma: Groebner-base obeys
      (for p: polynomials; i: ideals)
      (p in i) = ((p mod set_to_seq(base(i))) = 0).
```
◇
Definition defined by parts 17b, 18, 19ab, 20abc, 21ab, 22abc, 23a.
Definition referenced in part 47.

Our next goal is to introduce s-polynomials (see [3]) and to formalize the Gröbner base characterization based on them in the TECTON language. To do so, we need the notion of the least common multiple (of two monomials). We introduce this in a broader context, that is we extend the concept `Gcd-domain`[51] by an operator `lcm` giving the least common multiple of two arbitray elements `x` and `y`.[52]

[50]Note that the division of polynomials is automatically available in concept `Groebner-base` as it was introduced as an extension of concept `Multivariate-polynomial-with-monomial-order` which is a part of concept `Multivariate-polynomial-with-finitely-many-variables-over-field` imported by `Groebner-base`.

[51]This concept is already included in the TECTON concept library; see appendix B.3.

[52]The notion `set-of-representatives` is used to get a uniquely result from the operator `lcm`; compare appendix B.2 and [12].

⟨Groebner 22a⟩ ≡

```
Extension: Gcd-domain
  introduces lcm : domain x domain -> set-of-representatives;
  requires (for x, y: domain)
    x | lcm(x,y) and y | lcm(x,y) and
    ((for z: domain) (x | z and y | z) implies lcm(x,y) | z).
```

◇
Definition defined by parts 17b, 18, 19ab, 20abc, 21ab, 22abc, 23a.
Definition referenced in part 47.

As the concept `Multivariate-polynomial-with-monomial-order` is a refinement of the concept `Multivariate-polynomial` which imports the concept `Gcd-domain` with the `domain` being replaced by `polynomials` the operator `lcm` just defined is available there for arbitrary polynomials, and hence for monomials, too. In addition we need the division of two monomials without remainder. Again we introduce the operator `/` for arbitrary `polynomials` and not for its subtype `monomials`. Note that there is a result of `p/q` only if `p|q`, that is if polynomial `p` divides polynomial `q`.[53]

⟨Groebner 22b⟩ ≡

```
Extension: Multivariate-polynomial-with-monomial-order
  introduces
    / : polynomials x polynomials -> polynomials;
  requires (for p,q,r: polynomials; u,v,w: monomials)
    p | q implies q/p = r where p * r = q.
```

◇
Definition defined by parts 17b, 18, 19ab, 20abc, 21ab, 22abc, 23a.
Definition referenced in part 47.

Now everything is prepared to introduce s-polynomials as a Tecton operator called `s-polynomial` taking polynomials `p` and `q` as input. The returned polynomial, that is the s-polynomial of `p` and `q`, can be given as a straightforward translation of the definition of s-polynomials found in textbooks (see for example [3]). Note that we use the operator `/` only with `HM(p)` respectively `HM(q)` and `lcm(HM(p),HM(q))` as arguments, that is the condition `p|q` from the specification above is always fulfilled here.[54]

⟨Groebner 22c⟩ ≡

```
Extension: Multivariate-polynomial-with-monomial-order
  introduces monic-monomials < polynomials,
    s-polynomial : polynomials x polynomials -> polynomials;
  requires (for p,q: polynomials)
    s-polynomial(p,q) =
      HC(q) * (HM(p)/lcm(HM(p),HM(q))) * p -
      HC(p) * (HM(q)/lcm(HM(p),HM(q))) * q,
    p : monic-monomials = (p : monomials and HC(p) = 1).
```

◇
Definition defined by parts 17b, 18, 19ab, 20abc, 21ab, 22abc, 23a.
Definition referenced in part 47.

[53] To be more precise, we only specify the value in this case. However, in the Tecton language only total function are allowed, which means that `/` always have a value. If `p/q` does not hold, we now nothing about the resulting polynomial; and as we will see in the follwing we are not interested in.

[54] The type expression `monic-monomials < polynomials` in this concept definition should not be necessary as the Tecton subtype relation is transitive, but without the Tecton type checker does not accept `monic-monomials` as the type of the arguments of operator `lcm`; compare footnote 28.

The characterization of Gröbner bases using s-polynomials is similar to the one from above using division of polynomials: for each pair of polynomials p and q being part of a Gröbner base we have that the s-polynomial of p and q divided by the polynomials of the base leaves no remainder. We also use this characteriztion to give an equivalent definition of the predicate Groebnerbase testing whether a set s is a Gröbner base for a given ideal i.

⟨Groebner 23a⟩ ≡
```
Lemma: Groebner-base obeys
  (for p,q: polynomials; i: ideals; s: finite-sets)
  (p in base(i) and q in base(i)) implies
    (s-polynomial(p,q) mod set_to_seq(base(i))) = 0,
  Groebnerbase(s,i) =
    ((p in s and q in s) implies (s-polynomial(p,q) mod set_to_seq(s)) = 0).
```
◇
Definition defined by parts 17b, 18, 19ab, 20abc, 21ab, 22abc, 23a.
Definition referenced in part 47.

8 Rewrite Relations

In this section we introduce the basic notions of rewriting systems (see for example [1]) necessary to charactarize Gröbner bases in this context. A rewrite relation $\longrightarrow$ is a binary relation over an arbitrary domain. We start with the introduction of some standard notations such as $\longleftrightarrow$, $\xrightarrow{*}$, $\xleftrightarrow{*}$ and others.[55]

⟨Rewrite 23b⟩ ≡
```
Precedence: {-->, <--, <-->, -*->, <-*->, -+->, <-+->}
                                                   < nonassociative{=}.
Precedence: {or, and} < {-->, <--, <-->, -*->, <-*->, -+->, <-+->}.

Abbreviation: Rewrite-relation is Binary-relation [with --> as R].

Extension: Rewrite-relation
  uses Natural;
  introduces
    <--   : domain x domain -> bool,
    f     : naturals x domain x domain -> bool,
    <-->  : domain x domain -> bool,
    -*->  : domain x domain -> bool,
    g     : naturals x domain x domain -> bool,
    <-*-> : domain x domain -> bool,
    -+->  : domain x domain -> bool,
    <-+-> : domain x domain -> bool;
```
◇
Definition defined by parts 23b, 24abc, 25abc, 26abc.
Definition referenced in part 47.

The following requirements for these operators are straightforward translations from [1]. We also included the well-known fact that $\xleftrightarrow{*}$ is an equivalence relation on its domain as a Tecton lemma.

[55]We use f and g as a notation for $\xrightarrow{n}$ and $\xleftrightarrow{n}$ respectively. We would prefer to use -n-> and <-n->, but it seems that the Tecton language (or the Tecton type checker) does not allow to use operators in this way; note that n actually is an argument of the operator.

⟨Rewrite 24a⟩ ≡
```
      requires (for a,b: domain; n: naturals)
        (a <-- b)  = (b --> a),
        (a <--> b) = (a --> b or b --> a),
        f(0,a,b) = (a = b),
        f(n+1,a,b) = (for some c: domain) a --> c and f(n,c,b),
        (a -*-> b) = (for some n: naturals) f(n,a,b),
        g(0,a,b) = (a = b),
        g(n+1,a,b) = (for some c: domain) a <--> c and g(n,c,b),
        (a <-*-> b) = (for some n: naturals) g(n,a,b),
        (a -+-> b)  = (a -*-> b and not(f(0,a,b))),
        (a <-+-> b) = (a <-*-> b and not(g(0,a,b))).

    Lemma: Rewrite-relation implies Equivalence-relation [with <-*-> as R].
```
◇
Definition defined by parts 23b, 24abc, 25abc, 26abc.
Definition referenced in part 47.

Two main properties of rewrite relations are termination and confluence. Termination deals with the problem, that there may be infinite chains in the rewrite relation; terminating rewrite relations do not allow this, hence can be seen as comuputation sequenes. Here we start with cycle-free rewrite relations as having no cycles is obviously a necessary condition for termination.

⟨Rewrite 24b⟩ ≡
```
    Definition: Cycle-free-rewrite-relation
      refines Rewrite-relation;
      requires (for a: domain) not(a -+-> a).
```
◇
Definition defined by parts 23b, 24abc, 25abc, 26abc.
Definition referenced in part 47.

An element a is in normalform if it not further reducible, that is there exists no element c with $a \longrightarrow c$. A rewrite relation in which each element a has a normalform is called a weakly-terminating rewrite relation. We model this by introducing a predicate `in_normalform` testing whether an element `a` is in normalform and an operator `normalform` mapping elements `a` onto a normalform of them.[56]

⟨Rewrite 24c⟩ ≡
```
    Definition: Weakly-terminating-rewrite-relation
      refines Rewrite-relation;
      introduces
        in_normalform : domain -> bool,
        normalform : domain -> domain;
      requires (for a: domain)
        in_normalform(a) = not((for some c: domain) a --> c),
        normalform(a) <-*-> a and in_normalform(normalform(a)).
```
◇
Definition defined by parts 23b, 24abc, 25abc, 26abc.
Definition referenced in part 47.

[56]Note, that this indeed implies the existence of a normalform for every element `a` because `normalform` is as a TECTON function total.

It may be worth mentioning that the existence of a normalform for every element a does not imply that no infinite chains are possible: for example the rewrite system consisting of the rules $\{a \longrightarrow b,\ a \longrightarrow c,\ b \longrightarrow a,\ b \longrightarrow c\}$ is (finite and) weakly terminating, because c is a normalform for all elements, but allows infinite chains as it is not cycle free. To prohibit infinite chains in rewrite relations we use the concept `Well-order` introduced in section 4: We define terminating rewrite relations as a refinement of concept `Weakly-terminating-rewrite-relation` — making the operator `normalform` available — and concept `Well-order` with the relation `>` being replaced by `-+->`, thus requiring that the order `-+->` induced by the rewrite relation `-->` is well-founded, hence that `-->` does not allow infinite chains.

⟨`Rewrite 25a`⟩ ≡

```
Definition: Terminating-rewrite-relation
  refines Weakly-terminating-rewrite-relation,
          Well-order [with -+-> as >].
```

◇
Definition defined by parts 23b, 24abc, 25abc, 26abc.
Definition referenced in part 47.

As should be clear from the above mentioned the following lemma trivially holds.

⟨`Rewrite 25b`⟩ ≡

```
Lemma: Terminating-rewrite-relation implies Cycle-free-rewrite-relation.
```

◇
Definition defined by parts 23b, 24abc, 25abc, 26abc.
Definition referenced in part 47.

We have seen that in terminating rewrite realtions every element a has (at least) one normalform, but there may be more than one.[57] The corresponding property stating that every element a has at most one normalform is called confluence. It requires that if an element c can be reduced to two different elements a and b, then there is another element d to which both a and b can be reduced. We also introduce the weaker property of local concluence in which this test is restricted to c being reducible to a and b in one step.

⟨`Rewrite 25c`⟩ ≡

```
Definition: Local-confluent-rewrite-relation
  refines Rewrite-relation;
  requires (for a,b,c: domain)
    (c --> a and c --> b) implies
    (for some d: domain) a -*-> d and b -*-> d.

Definition: Confluent-rewrite-relation
  refines Local-confluent-rewrite-relation;
  requires (for a,b,c: domain)
    (c -*-> a and c -*-> b) implies
    (for some d: domain) a -*-> d and b -*-> d.
```

◇
Definition defined by parts 23b, 24abc, 25abc, 26abc.
Definition referenced in part 47.

[57]Note that this implies that there are usually several possibilities for a realization of the operator `normalform`.

There is a third property in this context: the Church-Russer property. It deals with elements a and b for which $a \overset{*}{\longleftrightarrow} b$ holds[58] and requires that in this case there is another element d to which both a and b can be reduced. Newman's lemma (see [1]) shows that this is in fact equivalent to the property of being confluent. We stated this as a TECTON lemma.

⟨Rewrite 26a⟩ ≡

```
Definition: Church-Rosser-rewrite-relation
  refines Rewrite-relation;
  requires (for a,b: domain)
    (a <-*-> b) implies (for some d: domain) a -*-> d and b -*-> d.

Lemma: Church-Rosser-rewrite-relation is Confluent-rewrite-relation.
```

◇
Definition defined by parts 23b, 24abc, 25abc, 26abc.
Definition referenced in part 47.

The important role of local confluent rewrite sytems is further outlined by the following lemma stating that for terminating rewrite systems the properties of being local confluent and confluent coincide. We will use this lemma in the next section where we deal with rewriting in the special case of polynomials.

⟨Rewrite 26b⟩ ≡

```
Lemma: Terminating-rewrite-relation [with Local-confluent-rewrite-relation
                                          as Rewrite-relation]
       implies Confluent-rewrite-relation.
```

◇
Definition defined by parts 23b, 24abc, 25abc, 26abc.
Definition referenced in part 47.

For completion we also introduce the combination of terminating and confluent rewrite relations, called complete rewrite system. It easily follows that in complete rewrite relations every element a has exactly one normalform; hence complete rewriting systems are an appropriate modell for (total) computations.[59] As should be clear from the last lemma, a terminating and local confluent rewrite system is in fact a complete rewrite system as the closing lemma states.

⟨Rewrite 26c⟩ ≡

```
Definition: Complete-rewrite-relation
  refines Confluent-rewrite-relation, Terminating-rewrite-relation.

Lemma: Terminating-rewrite-relation [with Local-confluent-rewrite-relation
                                          as Rewrite-relation]
       is Complete-rewrite-relation.
```

◇
Definition defined by parts 23b, 24abc, 25abc, 26abc.
Definition referenced in part 47.

[58]For the reader familiar with the topic, this means that a and b lie in the same equivalence class with respect to the equivalence realtion $\overset{*}{\longleftrightarrow}$.

[59]Note that these computations are indeterministic in nature and that the strategy of selecting the next reduction step may be of great importance for efficiency.

9 Polynomial Rewriting

In the following we give a Tecton definition of polynomial rewriting with the goal of formalizing the well-known fact (see for example [3]), that $G = \{g_1, \ldots g_n\}$ is a Gröbner base for $<g_1, \ldots g_n>$ if and only if the polynomial rewrite system induced by G is complete. To define rewriting with polynomials we need to introduce the notion of a coefficient. So we extend the concept `Multivariate-polynomial-with-monomial-order` with an operator `coeff` that takes a monomial `m` and a polynomial `p` as input and returns the coefficient of `m` in `p`.

⟨Poly-Rewrite 27a⟩ ≡

```
Extension: Multivariate-polynomial-with-monomial-order
  introduces
    coeff : monomials x polynomials -> coefficient-domain;
  requires (for p: nonzero-polynomials; m: monomials)
    coeff(m,0) = 0,
    coeff(m,p) = if HM(p) = m then HC(p) else coeff(m,Red(p)).
```

◇

Definition defined by parts 27ab, 28abc, 29abc, 30a, 32ab.
Definition referenced in part 47.

For polynomial rewriting one selects a (finite) set of polynomials with which reductions can be done. Consequently we define polynomial rewriting as a refinement of the concept `Multivariate-polynomial-with-finitely-many-variables-over-field` making the notion of polynomials availble and introduce an operator `rewrite-polynomials` giving the finite set of polynomials for rewriting.[60] In addition our concept `Polynomial-rewriting` refines `Terminating-rewrite-relation` with the `domain` being replaced by `polynomials`, hence making the operator `normalform` available for `polynomials`.[61] The import of the concept `Generated-ideal` is not necessary for the definition of polynomial rewriting, but we import it, because we need it for the next lemma.[62]

⟨Poly-Rewrite 27b⟩ ≡

```
Definition: Polynomial-rewriting
  refines Terminating-rewrite-relation [with polynomials as domain],
          Multivariate-polynomial-with-finitely-many-variables-over-field;
  uses Generated-ideal;
  introduces
    nonzero-monomials < polynomials,
    rewrite-polynomials : -> finite-sets;
```

◇

Definition defined by parts 27ab, 28abc, 29abc, 30a, 32ab.
Definition referenced in part 47.

Via the concept `Terminating-rewrite-relation` we imported a rewrite relation `-->`. Now, in the requirements we have to say, how this relation looks like in the special case of polynomials: a polynomial `p` is reducible with respect to a set F of polynomials if there is a polynomial $r \in F$ such that the headmonomial of r divides a monimal in p.

[60] In fact the definition describes the concept of polynomial rewriting with an arbitrary but fixed set of polynomials used for reduction; compare the semantics of Tecton in [9].

[61] Note that polynomial rewriting over polynomial rings with finitely many variables and with fields as coefficient domain is indeed terminating (see for example [3]).

[62] Again the type expression `nonzero-monomials < polynomials` in this concept definition should not be necessary as the Tecton subtype relation is transitive, but without the Tecton type checker does not accept `nonzero-monomials` as the type of `m` in `m in monoms(p)`; compare footnote 28.

The result of this reduction is the polynomial $q = p - a * (m/\mathrm{HT}(r)) * \mathrm{coeff}(m, p)$.[63] In addition we require that the zero polynomial is not included in the set F rewriting polynomials.

⟨Poly-Rewrite 28a⟩ ≡

```
      requires (for p,q: polynomials; a: coefficient-domain)
        (p in rewrite-polynomials) implies (p: nonzero-polynomials),
        (p --> q) =
           (for some r: nonzero-polynomials; m: nonzero-monomials)
             r in rewrite-polynomials and
             m in monoms(p) and HM(r) | m and
             q = p - coeff(m,p) * (m/HT(r)) * r ,
        p : nonzero-monomials = (p : monomials and p : nonzero-polynomials).
```

◇
Definition defined by parts 27ab, 28abc, 29abc, 30a, 32ab.
Definition referenced in part 47.

One major property of the just defined rewrite relation `-->` making polynomial reduction adequate for dealing with questions about polynomial ideals, is the following: the reflexive, symmetric and transitive closure `<-*->` of `-->` exactly describes the ideal congruence induced by the ideal that is generated by the set F of rewriting polynomials, that is

⟨Poly-Rewrite 28b⟩ ≡

```
    Lemma: Polynomial-rewriting obeys
      (for p,q: polynomials)
      (p <-*-> q) = (p - q in gen(rewrite-polynomials)).
```

◇
Definition defined by parts 27ab, 28abc, 29abc, 30a, 32ab.
Definition referenced in part 47.

Now we introduce rewriting for sets F of polynomials that in fact are Gröbner bases. This is easily done as a refinement of the concept `Polynomial-rewriting`, in which we have the additional requirement that our rewriting polynomials contained in `rewrite-polynomials` are a Gröbner base. We formalize this using the predicate `Groebnerbase` imported from the concept `Groebner-base` — which of course has to be imported to do so.

⟨Poly-Rewrite 28c⟩ ≡

```
    Definition: Polynomial-rewriting-with-Groebner-base
      refines Polynomial-rewriting,
              Groebner-base;
      requires Groebnerbase(rewrite-polynomials,gen(rewrite-polynomials)).
```

◇
Definition defined by parts 27ab, 28abc, 29abc, 30a, 32ab.
Definition referenced in part 47.

Now we are ready to state the first implication of the equivalence mentioned at the beginning of this section as a TECTON lemma: if we use a Gröbner base for polynomial reduction, then the resulting rewrite relation is (terminating and) confluent, hence a complete rewrite relation.

[63] It might be worth mentioning that a reduction step corresponds to making one step in the division of the polynomial `p` by the polynomial `q`; see [6].

⟨Poly-Rewrite 29a⟩ ≡

```
    Lemma: Polynomial-rewriting-with-Groebner-base implies
           Complete-rewrite-relation.
```

◇
Definition defined by parts 27ab, 28abc, 29abc, 30a, 32ab.
Definition referenced in part 47.

Based on the notion of polynomial rewriting we can also give a new solution of the ideal membership problem and a new formulation of our predicate `Groebnerbase` testing whether a set is a Gröbner base: a polynomial `p` is a member of the ideal generated by the rewriting polynomials if and only if the normalform of `p` with respect to the rewriting polynomials is the zero polynomial; a set `s` is a Gröbner base for the ideal generated by the rewriting polynomials if and only if all s-polynomials that can be built out of polynomials in `s` reduce to the zero polynomial.

⟨Poly-Rewrite 29b⟩ ≡

```
    Lemma: Polynomial-rewriting-with-Groebner-base obeys
      (for p: polynomials; s: finite-sets)
      (p in gen(rewrite-polynomials)) = (normalform(p) = 0),
      Groebnerbase(s,gen(rewrite-polynomials)) =
        (for p,q: polynomials)
        (p in s and q in s) implies s-polynomial(p,q) -*-> 0.
```

◇
Definition defined by parts 27ab, 28abc, 29abc, 30a, 32ab.
Definition referenced in part 47.

Now we turn to the other implication mentioned above, namely that if the (finite) set F of polynomials used for reduction induces a complete rewriting system, then F in fact is a Gröbner base for the ideal generated by F. To do so, we introduce a new concept `Polynomial-complete-rewriting` that refines the concept `Complete-rewrite-relation` with its subconcept `Rewrite-relation` being replaced by the above concept `Polynomial- rewriting` (without Gröbner bases), hence picking out polynomial rewriting for sets F of polynomials — that is possible realizations of `rewrite-polynomials` — that induces a complete rewrite relation.

⟨Poly-Rewrite 29c⟩ ≡

```
    Definition: Polynomial-complete-rewriting
      refines Complete-rewrite-relation [with Polynomial-rewriting
                                              as Rewrite-relation];
      uses Groebner-base.
```

◇
Definition defined by parts 27ab, 28abc, 29abc, 30a, 32ab.
Definition referenced in part 47.

Note that we also imported the concept `Groebner-base`. This import is necessary for the follwing lemma, because the condition for a concept C_1 to imply another concept C_2 is that $\mathrm{sem}(C_1) \subseteq \mathrm{sem}(C_2)$ holds. And for that it is necessary that all sorts and operators available in C_2 also have to be available in C_1; compare the semantic of the TECTON language in [9]. So before we can state the desired lemma, we first have to make available in particular the operator `gen` and the predicate `Groebnerbase`, which is done by importing the concept `Groebner-base`. Now it is easy to state in TECTON that a complete (finite) set of polynomial rewrite rules F is in fact a Gröbner base for the ideal generated by F:

```
⟨Poly-Rewrite 30a⟩ ≡
    Lemma: Polynomial-complete-rewriting implies
           Polynomial-rewriting-with-Groebner-base.
◇
```

Definition defined by parts 27ab, 28abc, 29abc, 30a, 32ab.
Definition referenced in part 47.

10 Buchberger's Algorithm

The main goal of the case study presented in this paper was to provide TECTON concepts necessary to specify Buchberger's algorithm for constructing Gröbner bases. This have been finished with the last section. Now we want to give such a specification explicitely using the programming language SUCHTHAT [12]. SUCHTHAT is equipped with a declaration mechanism to describe the concepts over which its generic algorithms are formulated. The declarations for Buchberger's algorithm for example could be

```
"Buchberger.st" 30b ≡
    let K be Field;
    let P[$X_1$,... $X_n$] be Polynomialring over K;
    let R be Term Order over {$X_1$,... $X_n$}.
◇
```

File defined by parts 30bcd.

SUCHTHAT algorithms consist of two parts: the so-called prototype, in which the algorithm's name is given and both its input and its output are specified. It may be worth mentioning that the specification of the input and the output is no comment, but indeed part of the programming language here. In our case we get the following prototype.[64]

```
"Buchberger.st" 30c ≡
    Algorithm: $G$ := GB($F$)
    Input:     $F \subseteq$ P[$X_1$,... $X_n$] such that $F$ is finite;
    Output:    $G \subseteq$ P[$X_1$,... $X_n$] such that $G$ is a Groebner base for $<F>$
◇
```

File defined by parts 30bcd.

The second part of a SUCHTHAT algorithm contains the body, that is the program code written in ALDES [8]. Buchberger's algorithm based on polynomial reduction is well-known[65], so we present it here without any further comments.

```
"Buchberger.st" 30d ≡
    (1) [Initialization]
        $G := F$;
        $B := \{(g_1, g_2) \mid g_1, g_2 \in G, g_1 \neq g_2\}$.
    (2) [Loop]
        while $B \neq \emptyset$ do
           { select $(g_1, g_2)$ from $B$   /* with a fair strategy */;
             $B := B \backslash \{(g_1, g_2)\}$;
             $h :=$ s-polynomial($g_1, g_2$);
```

[64]Note that the field K and the term order R do not explicitely occur in the algorithm's prototype; nevertheless they are implicit parameters of the algorithm as they heavily influence the resulting Gröbner base G.

[65]see for example [3].

```
        $h_0$ := normalform($h, G$);
        if $h_0 \neq 0$ then
            { $B := B \cup \{(g, h_0) \mid g \in G\}$;
              $G := G \cup \{h_0\}$ };
      }.  $\Box$
```
◇
File defined by parts 30bcd.

Note that so far SuchThat declarations are only symbolic. They get their meaning by connecting them to Tecton concepts. In particular, all subalgorithms appearing in an algorithm — in our case these are algorithms for `s-polynomial` and `normalform`[66] — must have a Tecton operator as a formal counterpart.[67] To realize this connection an intermediate language called elementary SuchThat has been introduced (see [12]) in which Tecton concepts can explicitely occur in the algorithm's prototype. In the case of Buchberger's algorithm we need to import the concepts `Groebner-base` and `Polynomial-rewriting`. Note that we need not state that the coefficients of the polynomial ring form a field `K`, because the concept imported here is `Multivariate-polynomial-with-finitely-many-variables-over-field`.[68]

"Buchberger.est" 31 ≡
```
    Algorithm: $G$ := GB($F$)
      uses Groebner-base, Polynomial-rewriting;
    Input:     $F \in$ finite-sets;
    Output:    $G \in$ finite-sets such that Groebnerbase($G$,gen($F$))
```
◇
File defined by parts 31, 32c.

Using this declaration for Buchberger's algorithm the subalgorithm `s-polynomial` is identified with the operator `s-polynomial` imported by concept `Groebner-base`.[69] However, a problem occured with the subalgorithm `normalform`: In Buchberger's algorithm it has two arguments, a polynomial h and a set of polynomials G for reduction.[70] But we defined in section 9 the Tecton operator `normalform` for one argument only, namely for a polynomial `p` the set of polynomials being fixed through the set `rewrite-polynomials`.[71] Fortunately this defect can be easily repaired by extending the concept `Polynomial-rewriting` with a second Tecton operator `normalform` taking two arguments as desired. The definition is almost analogous: a polynomial `q` is the normalform of another polynomial `p` with respect to a set `s` of polynomials if the difference `p - q` lies in the ideal generated by `s` and `q` in addition is not reducible by any polynomial contained in `s`. The predicate `is-reducible-wrt` which also has to be defined again for arbitrary sets of polynomials `s` checks whether a monomial `m` of a given polynomial `p` is divided by the headmonomial of a polynomial `r` being included in a given set of polynomials `s`.[72]

[66]We do not consider set operations or the realization of a fair strategy in this paper.

[67]Of course also algorithms for these have to provide in order to make the whole algorithm running, but this is beyond the scope of this paper.

[68]In fact, we even cannot state this, because the subconcept `Coefficient-ring` is not available as it was already replaced in both `Groebner-base` and `Polynomial-rewriting` by the concept `Field`.

[69]Note that `Groebner-base` itself imports this operator from `Multivariate-polynomial-with-finitely-many-variables-over-field`.

[70]Please note that G is no Gröbner base yet.

[71]There the decision of fixing a set of polynomials for reduction made the definition of rewriting with Gröbner bases easier as we only needed to add requirements on `rewrite-polynomials`.

[72]compare the definition of reduction with respect to the fixed set of polynomials given by `rewrite-polynomials` in section 9.

⟨Poly-Rewrite 32a⟩ ≡

```
Precedence: {is-reducible-wrt} < nonassociative{=}.

Extension: Polynomial-rewriting
  introduces
    is-reducible-wrt : polynomials x finite-sets -> bool,
    normalform : polynomials x finite-sets -> polynomials;
  requires (for p,q: polynomials; s: finite-sets)
    (p is-reducible-wrt s) =
      ((for some r: nonzero-polynomials; m: nonzero-monomials)
        r in s and
        m in monoms(p) and HM(r) | m),
    normalform(p,s) = q where (p - q in gen(s) and
                               not(q is-reducible-wrt s)).
```

◇
Definition defined by parts 27ab, 28abc, 29abc, 30a, 32ab.
Definition referenced in part 47.

To connect this new operator `normalform` with the one already defined in section 9, we state the following lemma. It says that if the set `s` of polynomials is chosen to be the set given by `rewrite-polynomials` then both `normalform`-operators coincide.

⟨Poly-Rewrite 32b⟩ ≡

```
Lemma: Polynomial-rewriting obeys
  (for p: polynomials) normalform(p,rewrite-polynomials) = normalform(p).
```

◇
Definition defined by parts 27ab, 28abc, 29abc, 30a, 32ab.
Definition referenced in part 47.

With this extension all subconcepts of Buchberger's algorithm have a corresponding TECTON operator (that is included in one of the concepts `Groebner-base` and `Polynomial- rewriting`), hence the ELEMENTARY SUCHTHAT specification of the prototype is complete. The body of the algorithm that is the ALDES code is of course the same as above for the SUCHTHAT algorithm. The reason for repeating it here is that we wanted the file `Buchberger.est` generated by NUWEB to be a complete runable file.[73]

"Buchberger.est" 32c ≡

```
(1) [Initialization]
    $G := F$;
    $B := \{(g_1, g_2) \mid g_1, g_2 \in G, g_1 \neq g_2\}$.
(2) [Loop]
    while $B \neq \emptyset$ do
       { select $(g_1, g_2)$ from $B$   /* with a fair strategy */;
         $B := B \backslash \{(g_1, g_2)\}$;
         $h :=$ s-polynomial($g_1, g_2$);
         $h_0 :=$ normalform($h, G$);
         if $h_0 \neq 0$ then
            { $B := B \cup \{(g, h_0) \mid g \in G\}$;
              $G := G \cup \{h_0\}$ };
       }.  □
```

◇
File defined by parts 31, 32c.

[73]Throughout this paper we presented TECTON and SUCHTHAT code using NUWEB [4], a tool supporting the literate programming style of D. Knuth, which allows to extract the code files directly from a papers' source and to construct extensive indexes automatically.

11 The Hilbert Basis Theorem

This last section is devoted to Hilbert's basis theorem. It states that under certain requirements on the coefficient domain, namely that it constitute a so-called Noetherian ring, every ideal in a polynomial ring has a finite basis, hence is finitely generated. From this one can conclude using for example Buchberger's algorithm that every ideal in such a polynomial ring has a Gröbner base.[74] We introduce the necessary notation of a Noetherian ring and state this theorem as a TECTON lemma. A Noetherian ring is an integral domain in which each ideal is finitely generated; this is easily modelled using the operator `gen` from the concept `Finitely-generated-ideal`.

⟨`Hilbert 33a`⟩ ≡

```
Definition: Noetherian-ring
  refines Integral-domain;
  uses Finitely-generated-ideal;
  requires (for i: ideals)
    (for some s: finite-sets) i = gen(s).
```

◇
Definition defined by parts 33abc, 34.
Definition referenced in part 47.

An easy concequence of this definition is the following: in a field F the only ideals are the zero ideal $\{0\}$ and F itself. Because both ideals are finitely generated, every field is a Noethrian ring.

⟨`Hilbert 33b`⟩ ≡

```
Lemma: Field implies Noetherian-ring.
```

◇
Definition defined by parts 33abc, 34.
Definition referenced in part 47.

Now we turn over to Hilbert's basis theorem. The above mentioned condition on the coefficient domain of a polynomial ring is that this domain itself is Noetherian (see for example [3]). In order to formalize this in the TECTON language we first introduce the notion of a `Notherian-coefficient-ring` by refining the concept `Coefficient-ring` with its subconcept `Commutative-ring-with-identity` being replaced by the just defined concept `Notherian-ring`

⟨`Hilbert 33c`⟩ ≡

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

◇
Definition defined by parts 33abc, 34.
Definition referenced in part 47.

Using the concept `Notherian-coefficient-ring` it is easy to state Hilbert's basis theorem as a TECTON lemma. The only thing we have to take into account is that the polynomial ring must have a finite number of variables[75], that is we must use the concept `Multivariate-polynomial-with-finitely-many-variables` instead of the more general `Multivariate-polynomial`.

[74]Note that polynomial rings with finitely many variables over fields are Noetherian, as we will see in the following.
[75]Note that polynomial rings with infinitely many variables are no longer Noetherian; see [2].

⟨Hilbert 34⟩ ≡

```
    Lemma: Multivariate-polynomial-with-finitely-many-variables [with
                     Notherian-coefficient-ring as Coefficient-ring]
           implies Noetherian-ring.
```

◇

Definition defined by parts 33abc, 34.
Definition referenced in part 47.

12 Conclusion

We have presented a case study on the use of the concept description language Tecton, in which we provided Tecton concepts necessary for the specification of Buchberger's algorithm to compute Gröbner bases for polynomial ideals over fields in the programming language SuchThat. SuchThat is a generic language providing a declaration mechanism to describe abstract domains over which algorithms can be formulated. These declarations are connected with Tecton concepts, so that all subalgorithms appearing in an algorithm msut have a corresponding Tecton operator. Consequently, a Tecton specification of a generic algorithm consists of defining concepts providing all the Tecton operators necessary for the algorithm.

To specify Buchberger's algorithm the definition of 29 concepts was necessary. We defined some more concepts (44 alltogether) in order to formalize well-known theorems as Tecton lemmas, for example the formalization of Hilbert's basis theorem. In addition we tried to be as general as possible this leading to more concepts than in fact are necessary, for example we defined multivariate polynomials as a special case of power series, not to mention that we introduced these concepts for infinitely many variables first, then restricting the number of variables in a second definition. It should be clear, that being more general in the beginning may increase the total number of concepts for a particular specification due to necessary specializations later. From the Tecton concept library we used 56 concepts, 44 of them being necessary for the entire specification of Buchberger's algoritms. These concepts can be found in appendix B.

During our work we also found some shortcomings of the Tecton type checker. The Tecton type relation is based on the inclusion of sets and hence transitive. However, to get our concept definitions accepted by the checker it was sometimes necessary to explicitely state a subtype relation although it follows from transitivity. For example though we included the subtype relations `monic-monomials < nonzero-monomials` and `nonzero-monomials < monomials` the type checker did not accept an argument of type `monic-monomials` where one of type `monomials` is required. By introducing the extra subtype relation `monic-monomials < monomials` one could overcome this problem. In addition the type checker does not accept requirements starting with an existencial quantifier, that is for example `(for some t: sets)`, although the Tecton semantics allow for this. We hope that in the next version of the Tecton type checker these things will be fixed.

The reader familiar with Tecton's semantics will have observed that our specification in some sense is not complete yet. First, the lemmas have not been proven so far. Second, more important, also a number of Tecton definitions come with further proof obligations. For example, if a concept is imported by another concept using the keyword `uses` one has to show that the appearing other requirements do not change the semantics of the imported concept. So far Tecton does not provide any support to prove these obligations. Our aim is to look for theorem provers or proof checkers in which such obligations can be formalized and shown. The concepts of our case study should give a good starting point to investigate how non-trivial proof obligations appearing in TECTON specifications can be formally proven with machine assistance.

References

[1] J. Avenhaus, *Reduktionssysteme*; Springer Verlag, 1995.

[2] J. Backer, P. Rudnicki, *Hilbert's Basis Theorem*; to appear in Formalized Mathematics, 2000.

[3] T. Becker and V. Weispfenning, *Gröbner bases*; Springer Verlag, 1993.

[4] P. Briggs, NUWEB — *A Simple Literate Programming Tool*; available by anonymous ftp from `http://softlib.rice.edu/MSCP/nuweb.html`, 1989.

[5] B. Buchberger, *Gröbner bases: An Algorithmic Method in Polynomial Ideal Theory*; in: N. Bose (ed.), Multidimensional Systems Theory, Reidel Publishing Company, 1985.

[6] D. Cox, J. Little, D. O'Shea, *Ideals Varieties and Algorithms*; Springer Verlag, 1992.

[7] J. von zur Gathen, J. Gerhard, *Modern Computer Algebra*; Camebridge University Press, 1999.

[8] R. Loos, G. Collins, *Revised Report on the Algorithm Description Language ALDES*; Technical Report WSI-92-14, Wilhelm-Schickard-Institute for Computer Science, 1992.

[9] D. Musser, *The* TECTON *Concept Description Language*; available by anonymous ftp from `http://www.cs.rpi.edu/~musser/gp/tecton`, 1998.

[10] D. Musser, S. Schupp, C. Schwarzweller, R. Loos *The* TECTON *Concept Library*; Technical Report WSI-99-02, Wilhelm-Schickard-Institut für Informatik, Univertät Tübingen, 1999.

[11] P. Rudnicki, A. Trybulec, *Multivariate Polynomials with Arbitrary Number of Variables*; Formalized Mathematics(8), 317-332, 1999.

[12] S. Schupp, R. Loos, SUCHTHAT — *Generic Programming Works*; in: M. Jazayeri, R. Loos, D. Musser (eds.), Generic Programming, International Seminar on Generic Programming, Dagstuhl Castle, Germany, LNCS 1766, 1998.

A An Example of a Monomial Order

Monomial orders for multivariate polynomials have been introduced in section 5. There we only required the existence of such orders by importing the concept `Term-order` of section 4. In the following we give a TECTON formalization of a concrete monomial order[76] that may be used to specialize the TECTON concepts given so far.

[76]For the reader familiar with this topic, we introduce the lexicographic order on monomials.

We start with introducing a total order on the variables, which then will be lifted (to bags and) to monomials. This is done by a refinement of the concepts `Variable` and `Total-order`.

```
⟨Extension 36a⟩ ≡
    Definition: Ordered-variable
      refines Variable [with ordered-variables as variables],
              Total-order [with ordered-variables as domain].
◇
```

Definition defined by parts 36abc, 37a.
Definition referenced in part 47.

Monomials are in fact pairs consisting of a bag `b` and an element `a` of the coefficient domain. A monimial is connected with the variables via its bag `b`, which is a function from the variables into the natural numbers. Therefore, to compare monomials we first have to consider the order on bags induced by the order on the variables.[77] We combine the concept `Bag` with the concept `Total-order` using a refinement, in which both `bags` and `domain` are replaced by `ordered-bags`. It remains to give the description of the relation `<` imported from the concept `Total-order`:[78] A bag `b` is smaller than a bag `b'` if there is a variable `v` for which `b` gives a smaller natural number than `b'` and for all variables that are smaller than `v` both bags `b` and `b'` give the same value.

```
⟨Extension 36b⟩ ≡
    Definition: Ordered-bag
      refines Bag [with ordered-bags as bags, ordered-variables as variables],
              Total-order [with ordered-bags as domain];
      uses Ordered-variable;
      requires (for b,b': ordered-bags)
        (b < b') =
          ((for some v: ordered-variables) apply(b,v) < apply(b',v) and
           (for v': ordered-variables)
           v' < v implies apply(b,v') = apply(b',v')).
◇
```

Definition defined by parts 36abc, 37a.
Definition referenced in part 47.

Now using the order on bags we can easily compare monomials:[79] a monomial `m` has exactly one bag `b` and is smaller than another monomial `m'` if `b` is smaller than the bag `b'` of `m'`. Note that we define this order for polynomials with finitely many variables only.[80]

```
⟨Extension 36c⟩ ≡
    Definition: Multivariate-polynomial-with-finitely-many-ordered-variables
      refines Multivariate-polynomial-with-finitely-many-variables
                      [with Ordered-bag as Bag, ordered-bags as bags];
      uses Term-order [with monomials as domain];
      requires (for m,m': monomials)
```

[77]The reason for this additional step is that we wanted the type `monomials` to be a subtype of the type `polynomials`. Consequentley a monomial must be a function from bags into the coefficient domain, and cannot be a function from the variables into the natural numbers; compare section 5.

[78]Without this description we would specify bags with an arbitratry but fixed order.

[79]From the above it should be clear that every order on bags gives an order on monomials.

[80]It would be interesting to investigate, whether this order is also a monomial order for infinitely many variables.

```
    (m < m') = (for some b,b': ordered-bags; v,v': variables)
                 not(apply(m,b) = 0) and not(apply(m',b') = 0) and b < b'.
```
◇
Definition defined by parts 36abc, 37a.
Definition referenced in part 47.

We end this appendix with the following lemma stating that the just defined concept `Multivariate-polynomial-with-finitely-many-ordered-variables` having a very special order on the monomials indeed is a specializtion of the general concept `Multivariate-polynomial-with-monomial-order` from section 5.

⟨Extension 37a⟩ ≡

```
  Lemma: Multivariate-polynomial-with-finitely-many-ordered-variables implies
    Multivariate-polynomial-with-monomial-order.
```
◇
Definition defined by parts 36abc, 37a.
Definition referenced in part 47.

B Additional Tecton Concepts

For completion we present in the follwing all Tecton concepts our definition of Gröbner bases uses. In this appendix one finds 56 concepts, so that it took alltogether 100 concepts to formulate Buchberger's algorithm.[81] We devided the concepts in this appendix in several subclasses: 10 basic concepts such as sets and natural numbers, 11 order concepts, 33 algebraic concepts and 2 concepts about polynomials. This roughly correcponds to the classification of concepts introduced in the Tecton concept library [10]. Extensions of and lemmas about already existing concepts have not been counted again.

B.1 Basic Concepts

"Groeb.tec" 37b ≡

```
  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;
```

[81]To be more precise, 100 concepts were used in this paper. Not all of them (in fact only 83) are necessary for the sole specification of Buchberger's algorithm; compare section 12.

```
  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 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 38a ≡

```
Definition: Domain
  uses Boolean;
  introduces domain.
```
◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 38b ≡

```
Precedence: nonassociative{=} < nonassociative{in}.

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

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) = (d in s1 implies d in s2),
  is_empty(s) = (s = empty),
  (d in (e into s1)) = ((d = e) or d in s1),
  (d in complement(s)) = (not d in s),
  singleton(d) = (d into empty),
  (s1 union empty) = s1,
  (s1 union (d into s2)) =
     if d in s1 then s1 union s2
                else d into (s1 union s2),
  (s1 intersection empty) = empty,
  (s1 intersection (d into s2)) =
     if d in s1 then d into (s1 intersection s2)
```

```
                   else s1 intersection s2,
      s : nonempty-sets = (s != empty).

    Definition: Finite-set
      uses Set;
      introduces finite-sets < sets,
        nonempty-finite-sets < nonempty-sets,
        nonempty-finite-sets < finite-sets,
        into: domain x finite-sets -> finite-sets;
      generates finite-sets freely using empty, into;
      requires (for s: sets; s1: nonempty-sets; s2: finite-sets)
        s : finite-sets =
              (s = empty or
               s != empty and
                (for some d: domain; s': finite-sets) s = d into s'),
        s1 : nonempty-finite-sets = (s1 : finite-sets),
        s2 : nonempty-finite-sets = (s1 != empty).
```
◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 39a ≡
```
    Definition: Pair
      refines Domain [with domain1 as domain],
              Domain [with domain2 as domain];
      introduces pairs,
        make-pair: domain1 x domain2 -> pairs,
        first:  pairs -> domain1,
        second: pairs -> domain2;
      generates pairs freely using make-pair;
      requires (for d1: domain1; d2: domain2; p: pairs)
        first(make-pair(d1,d2)) = d1,
        second(make-pair(d1,d2)) = d2,
        make-pair(first(p),second(p)) = p.
```
◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 39b ≡
```
    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,
        * : naturals x naturals -> naturals;
      generates naturals freely using 0, succ;
      requires (for n, m: naturals)
         n + 0 = n,
         n + succ(m) = succ(n + m),
         n - 0 = n,
         0 - n = 0,
          succ(n) - succ(m) = n - m,
          1 = succ(0),
          n * 0 = 0,
          n * succ(m) = (n * m) + n.

    Extension:  Natural
      introduces
        nonzero-naturals < naturals,
        2   : -> naturals,
        <=  : naturals x naturals -> bool,
        <   : naturals x naturals -> bool,
        >=  : naturals x naturals -> bool,
        >   : naturals x naturals -> bool;
      requires (for n, m: naturals)
        2 = 1 + 1,
        0 <= n,
        not(succ(n) <= 0),
        (succ(m) <= succ(n)) = (m <= n),
        (n < m) = (n <= m and n != m),
        (n >= m) = not(n < m),
        (n > m) = not(n <= m),
        n : nonzero-naturals = (n != 0).
```
◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 40a ≡

```
    Definition: Range
      uses Domain[with range as domain].

    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 -> finite-maps;
      generates finite-maps freely using empty, into;
      requires (for s: maps; s1: nonempty-maps)
        s : finite-maps =
              (s = empty or
               s != empty and
               (for some d: domain; s': finite-maps) s = d into s'),
        s1 : nonempty-finite-maps = (s1 != empty).
```
◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 40b ≡

```
    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
  uses Sequence [with Domain as Range, domain as range];
  introduces finite-sequences < sequences,
    nonempty-finite-sequences < finite-sequences,
    nonempty-finite-sequences < nonempty-sequences,
    into: domain x finite-sequences -> finite-sequences;
  generates finite-sequences using empty, into;
  requires (for s: sequences; s1: finite-sequences; s2: nonempty-sequences)
    s : finite-sequences =
          (s = empty or
           s != empty and
            (for some d: domain; s': finite-sequences) s = d into s'),
    s1 : nonempty-finite-sequences = (s1 != empty),
    s2 : nonempty-finite-sequences = (s2 : finite-sequences).
```
◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

B.2 Order Concepts

"Groeb.tec" 41a ≡

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

Precedence: nonassociative{R, <=}.

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

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

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 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 41b ≡

```
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).

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

◇

File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 42 ≡

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

Precedence: nonassociative{=, equiv}.

Definition: Equivalence-class
  uses Domain, Equivalence-relation [with equiv as R];
  introduces equivalence-classes,
    in : 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),
    x in e = (equivalence-class(x) = e).

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: set-of-representatives = (representative(x) = x),
    equivalence-class(representative(e)) = e,
    representative(x) = representative(equivalence-class(x)).
```

◇

File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

B.3 Algebraic Concepts

"Groeb.tec" 43a ≡

```
Precedence: nonassociative{=} < {*}.

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

◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 43b ≡

```
Precedence: nonassociative{=} < {|} < {+, -}.
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: Left-regular
  refines Binary-op;
  introduces | : domain x domain -> bool;
  requires (for x, y: domain)
    x | y = (for some d: domain) d * x = y.

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

◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 43c ≡

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

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

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

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

◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 43d ≡

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

Lemma: Right-inverses implies 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: 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).
```
◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 44a ≡

```
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.
```
◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 44b ≡

```
Abbreviation: Semigroup is Associative.

Definition: Regular-semigroup
  refines Regular, Semigroup.

Definition: Monoid
  refines Semigroup, Identity.

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

Definition: Commutative-semigroup
```

```
      refines Regular-semigroup[with + as *],
              Commutative[with + as *].
```
◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 45a ≡
```
    Definition: Group
      refines Monoid, Inverses.

    Definition: Abelian-monoid
      refines Monoid, Commutative.

    Definition: Commutative-group
      refines Commutative, Group.

    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 : nonzeros = (x != 0),
        x - y = x + (-y),
          + x = x.
```
◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 45b ≡
```
    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.
```
◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 45c ≡
```
    Definition: Ring
      refines Abelian-group, Semigroup, Distributive.

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

```
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.

Definition: Ring-with-identity
  refines Ring, Identity.
```
◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

"Groeb.tec" 46a ≡
```
Definition: Commutative-ring-with-identity
  refines Commutative-ring, Identity.

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

Definition: Field
  refines Commutative, Division-ring.
```
◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

B.4 Polynomial Concepts

"Groeb.tec" 46b ≡
```
Abbreviation: Coefficient-ring is
  Commutative-ring-with-identity [with coefficient-domain as domain].

Definition: Polynomial
  refines Map [with polynomials as maps,
                    naturals as domain,
                    Coefficient-ring as Range,
                    coefficient-domain as range,
                    c as apply],
          Gcd-domain [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: nonzero-polynomials = nonzero(p),
      nonzero(p) = ((for some n: naturals) c(p, n) != 0),
    (for some n: naturals) (for 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 n: naturals) c(p, n) = c(q, 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 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

`"Groeb.tec"` 47 ≡

⟨Ideals 4b, ... ⟩
⟨LinComb 7, ... ⟩
⟨Order 10a, ... ⟩
⟨Poly 11b, ... ⟩
⟨Groebner 17b, ... ⟩
⟨Rewrite 23b, ... ⟩
⟨Poly-Rewrite 27a, ... ⟩
⟨Hilbert 33a, ... ⟩
⟨Extension 36a, ... ⟩

◇
File defined by parts 37b, 38ab, 39ab, 40ab, 41ab, 42, 43abcd, 44ab, 45abc, 46ab, 47.

C Indices

C.1 Files

C.2 Macros

C.3 Concept Names