DESIGN DOCUMENT FOR FUNCTION FIELD PROJECT

JENS BAUCH AND AVI KULKARNI

1. Theoretical Background

In this section we describe the theoretical framwork for our package (what will be the package name?).

Let A be a Dedekind domain and K the fraction field of A. Denote by θ a root of a monic irreducible separable polynomial $f \in A[x]$ of degree n and let $L = K(\theta)$ be the algebraic extension defined by f. We denote by $\mathcal{O}_L$ the integral closure of A in L. If A is a PID, then $\mathcal{O}_L$ is a free A-module of rank n. Any ideal I of $\mathcal{O}_L$ can alse be consider as a free A-module of rank n. For any prime ideal $\mathfrak{P}$ of $\mathcal{O}_L$ we denote by $v_{\mathfrak{P}}$ the discrete valuation induced by $\mathfrak{P}$. For $I = \prod_{\mathfrak{P} \in \mathrm{Spec}(\mathcal{O}_L)} \mathfrak{P}^{a_{\mathfrak{P}}}$ we define $v_{\mathfrak{P}}(I) = a_{\mathfrak{P}}$ and $\mathrm{supp}(I) = \{\mathfrak{P} \in \mathrm{Spec}(\mathcal{O}_L) \mid a_{\mathfrak{P}} \neq 0\}$.

1.1. **Short Representation of Ideals.** In this section we summarize the results from [?] in the context of the algebraic extension L of a fraction field K of the Dedekind domain A. Let I be a fractional ideal of $\mathcal{O}_L$ and S be a set of prime ideals of A such that the numerator of $\mathrm{Norm}(I)$ factors over S. We call the pair (g, γ) an S-normal presentation for I iff

(i) g belongs to $I \cap A$ and $\gamma \in A$,
(ii) for all prime ideals $\mathfrak{P}$ of $\mathcal{O}_L$ over $p \notin S$ we have $v_{\mathfrak{P}}(g) = v_P(I)$=0,
(iii) for all prime ideals $\mathfrak{P}$ of $\mathcal{O}_L$ over $p \in S$ we have $v_{\mathfrak{P}}(\gamma) = v_P(I)$.

1.1.1. *Multiplication.* For the multiplication of two ideals I and I' we consider their S-normalized presentations (g, γ) and (g', γ'), respectively. Then $(g \cdot g', \gamma \cdot \gamma')$ is an S-normal presentations of $I \cdot I'$. We define the $\mathrm{supp}(I)$ to be the set of all prime ideals of $\mathcal{O}_L$ which divide I and set $S(I) := \mathrm{supp}_A(I) = \{\mathfrak{P} \cap A \mid \mathfrak{P} \in \mathrm{supp}(I)\}$. In practice we present and ideal by an $S(I)$-normal presentation.

In practice L will be a number field or an algebraic function field. Thus $A \in \{\mathbb{Z}, k[x]\}$, where k denotes a field. If $A = \mathbb{Z}$ we define $\min(I)$ the minimum of an ideal I to be the absolute value of the minimal generator of $I \cap A$. In the case of $A = k[x]$ we define $\min(I)$ to be the monic polynomial of minimal degree generating $I \cap A$. We call an element in A maximal if it is maximal with respect to the absolute value or the degree depending if $A = \mathbb{Z}$ or $A = k[x]$, respectively.

Then a $S(I)$-normal presentation of I is given by $(\min(I), \gamma)$ with γ as in item 3. Let $\mathfrak{p}$ be a prime ideal of A and $\mathfrak{P}$ be prime ideal of $\mathcal{O}_L$ lying over $\mathfrak{p} = \pi A$. We take $\pi_{\mathfrak{P}} \in \mathfrak{P}$ satisfying $v_{\mathfrak{Q}}(\pi_{\mathfrak{P}}) = \delta_{\mathfrak{Q},\mathfrak{P}}$ for all prime ideals $\mathfrak{Q}$ lying of $\mathfrak{p}$. Then $\{\mathfrak{p}\}$-normal presentation for $\mathfrak{P}$ is given by $(\pi, \pi_{\mathfrak{P}})$.

Given an $S(I)$-normal presentation of I and an $S(I')$-normal presentation of I' we have to compute for both ideals an $S = S(I) \cup S(I')$-normal presentation and multiply their components in order to get an S-normal presentation for $I \cdot I'$.

Date: October 16, 2017.

Let $s_I = \prod_{p \in S(I)} p$ and $s_{I'} = \prod_{p \in S(I') \setminus S(I)} p$. If $\gcd(gs_I, s_{I'}) = 1 = ugs_I + vs_{I'}$, then (g, δ) is an S-normal presentation according Lemma 6.3 in [**?**], where $\beta = vs_{I'}\gamma + ugs_I$.

The bottleneck is the computation of a S-normal presentation for I and I'.

In practice we do not store the sets $S(I)$ or $S(I')$. We just update the second generator I with respect to the first generator of I' and vice versa. Herefore, we consider the algorithm PPIO from Dan Bernsteil [][I have the reference]:

Algorithm 1 : PPIO

Input: $a, b \in A$.
Output: $a = c \cdot n$ with $c, n \in A$ maximal such that $\gcd(c, b) = 1$.

```
$c \longleftarrow \gcd(a, b)$
$n \longleftarrow a$ div $c$
$g \longleftarrow \gcd(c, n)$
while $g \neq 1$ do
    $c \longleftarrow c \cdot g$
    $n \longleftarrow n$ div $g$
    $g \longleftarrow \gcd(c, n)$
end while
return $c, n$
```

Denote by (g_1, γ_1) the representation for I and by (g_2, γ_2) the representation for I', respectively. We expand (g_1, γ_1) w.r.t. g_2 as follows:

Algorithm 2 : Support Expansion

```
$g \longleftarrow g_1$
$t, r \longleftarrow$ PPIO$(g_2, g)$
$1, u, v \longleftarrow$ xgcd$(g^2, t)$
$\gamma \longleftarrow v \cdot t \cdot \gamma_1 + u \cdot g^2$
return (g,$\gamma$)
```

Then (g, γ) is an S-normal preentation for I.

1.1.2. *Inversion.* Let (g, γ) be a $S(I)$-normal presentation for an ideal I. For the inversion of I, we compute the denominator d of $\beta = 1/\gamma$ in the maximal order $\mathcal{O}_L$. Suppose d factors into $d_1 \cdot d_2$, where d_2 is maximal being coprime to g. Then $(1, d_2\beta)$ is an $S(I^{-1})$-normal presentation for I^{-1}.

For a principal ideal $I = \langle \gamma \rangle$ an $S(I)$-normal presentation is given by (g, γ), where generates $I \cap A$.

We represent

2. General Architecture

As per the manual page regarding the depreciated "Dedekind domain" class, we should implement Dedekind domains as a Sage Category and implement our methods as generic operations whenever possible. Of course, we can always override these methods for specific classes of Dedekind domains, such as with NumberFields.

Why use the category framework:

(i) It really does not seem like it is that difficult to use.
(ii) If our implementations are sufficiently generic, then we do not need to rewrite the class methods every time we want to implement a new type of Dedekind domain.
(iii) We minimize code duplication.
(iv) We can augment pre-existing classes in SAGE by refining the category. What sage does is attaches the ”ParentMethods” defined in the category to the parent class, ”ElementMethods” to the elements, etc. This means, for instance, that the preexisting ”AbsoluteOrder” class for number fields can inherit the methods we write without restructuring the entire class hierarchy. Of course, whether these methods compute the correct thing when called can be tricky!

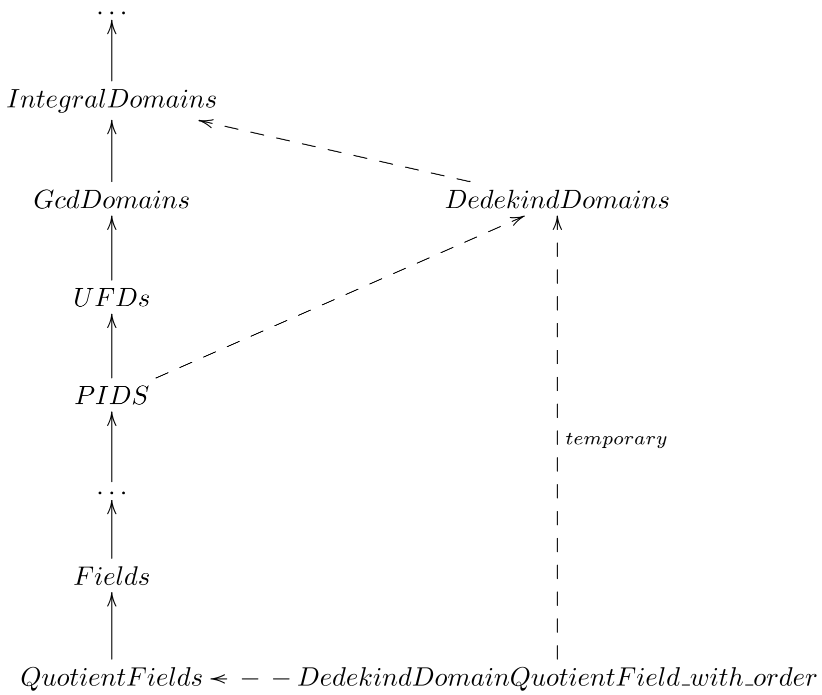
Remark. We need to be careful regarding the order of inheritance. For instance, the method ”_mul_” defined for ”NumberFieldIdeal” should override the generic category method we write and not vice-versa. To me it is unclear how exactly to do this. For now, we can throw an error if someone tries to turn a ”NumberFieldIdeal” into a ”DedekindDomainIdeal”.

Remark. After implementing the ”DedekindDomains” category, in principle we should find all of the Sage Categories for which ”DedekindDomains” is an immediate supercategory (basically, PIDS).

Remark. The Sage Category ”NumberFields” implements none of the important routines. Instead, the design decision there was to implement all routines as class methods in ”NumberFieldIdeal”. This is likely because the original code was written before 2009. That is, before the category framework was in Sage.

2.1. **Category hierarchies.** An arrow $A \longrightarrow B$ indicates that the category A is a subcategory of B.

The following is a piece of Sage's category graph. The dotted arrows shows where our future category resides



Note that the reason we might need "DedekindDomainQuotientField_with_order" is so that our valuations have somewhere to live. Temporarily, we should declare the immediate super category to be Dedekind domains so we don't have to update the super category list for PIDs. Otherwise, doing things properly, requires us to keep a full Sage source in the git repository.

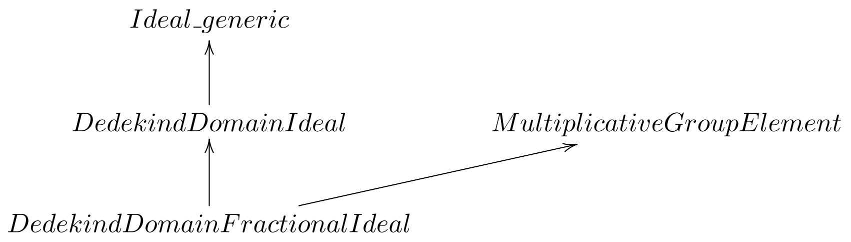
TODO: It is presently unclear what to do regarding the category structure of ideals of Dedekind domains. One approach is to simply have a class "DedekindDomainIdeal", but modify the category of "DedekindDomains" to have the ideal() method automatically generate a "DedekindDomainIdeal". The other is to make a category of "DedekindDomainIdeals" and implement functionality there. Both methods are likely equivalent for our purposes. We should draw inspiration from the treatment of ideals in the category "PIDs". Ultimately, this decision comes down to what works best with interfacing with the existing operations for "NumberFieldIdeals".

3. Class hierarchy

An arrow $A \longrightarrow B$ indicates that the class A inherits from the class B.

Remark. The DedekindDomain class is depreciated, but already exists in sage. However, the implementation there seems to be unsubstantial.

The documentation also recommends the creation of a DedekindDomain *category*. We should probably ask about this. Regardless, it seems like we need to code the class anyways.



4. Classes

4.1. Class/Category: DedekindDomain. :

Attributes:

- Maximal Order

Init:

- NumberFields → Invoke sage's routines
- FunctionFields → do our things
- Else → NotImplementedError
- Category: Demand required parent/element methods (valuation, gcd, etc)

Methods:

- FractionalIdealGroup
- krull_dimension
- valuation

4.2. Class/Category: DedekindDomainElt. :

Attributes:

- denominator

Init:

- NumberFields → Invoke sage's routines
- FunctionFields → do our things
- Else → NotImplementedError

Methods:

- denominator
- numerator

4.3. Class: DedekindDomainIdeal. :

Do we want to make the 2-element representation/ OM representation different classes, or do we want the same object to record both representations?

Attributes:

- Basis

Methods:

- is_prime

- denominator
- comparison method
- mult
- norm
- index
- inertia_degree
- ramification_degree
- maximal_order_basis
- convert_to_OM
  - if self is prime, do conversion
  - else raise NotImplementedError
- convert_to_hnf-representation
  - Only for number fields
- $\mathbb{Z}$-basis to 2-elt representation

4.4. **Class: DedekindDomainFractionalIdeal.** :

Attributes:

- Basis

Methods:

- is_integral
- denominator
- comparison method
- mult
- inver
- norm
- index
- inertia_degree
- ramification_degree
- convert_to_OM
  - if self is prime, do conversion
  - else raise NotImplementedError
- convert_to_hnf-representation
  - Only for number fields

5. Accessory functions

6. TODO

(i) Decide if 2-elt/OM representations should be distinct classes or unified in a single class

(ii) ~~Decide on Category versus Object approach for Dedekind Domains~~ We should use the Category approach.

(iii) Decide on Category or Object approach for DedekindDomainIdeals

(iv) Determine useful information to store as attributes