Программная реализация декодера одного класса помехоустойчивых кодов на алгебраических кривых: проектирование на основе шаблонов обобщённого и мета-программирования

А.М. Пеленицын
http://mmcs.sfedu.ru/~ulysses/, apel@sfedu.ru

Факультет математики, механики и компьютерных наук
Южный федеральный университет

27–28 апреля / FOSS Lviv 2012

Содержание

Outline

«The C++ template facility goes far beyond simple “containers of T”»

- 1988: средство безопасного параметрического полиморфизма для C++,
- 1993: STL и обобщённое программирование А. Степанова,
- 1994: метапрограммирование (пример Эрвина Унруха),
- 2001: “Modern C++ Design” Андрея Александреску (policy-based design, typelists, применение...).
- 2000s: Abrahams-Gurtovoy и Boost MPL, Extended C++ Мэтью Уилсона.

«The C++ template facility goes far beyond simple “containers of T”»

- 1988: средство безопасного параметрического полиморфизма для C++,
- 1993: STL и обобщённое программирование А. Степанова,
- 1994: метапрограммирование (пример Эрвина Унруха),
- 2001: “Modern C++ Design” Андрея Александреску (policy-based design, typelists, применение...).
- 2000s: Abrahams-Gurtovoy и Boost MPL, Extended C++ Мэтью Уилсона.

«The C++ template facility goes far beyond simple “containers of T”»

- 1988: средство безопасного параметрического полиморфизма для C++,
- 1993: STL и обобщённое программирование А. Степанова,
- 1994: метапрограммирование (пример Эрвина Унруха),
- 2001: “Modern C++ Design” Андрея Александреску (policy-based design, typelists, применение...).
- 2000s: Abrahams-Gurtovoy и Boost MPL, Extended C++ Мэтью Уилсона.

«The C++ template facility goes far beyond simple “containers of T”»

- 1988: средство безопасного параметрического полиморфизма для C++,
- 1993: STL и обобщённое программирование А. Степанова,
- 1994: метапрограммирование (пример Эрвина Унруха),
- 2001: “Modern C++ Design” Андрея Александреску (policy-based design, typelists, применение...).
- 2000s: Abrahams-Gurtovoy и Boost MPL, Extended C++ Мэтью Уилсона.

«The C++ template facility goes far beyond simple “containers of T”»

- 1988: средство безопасного параметрического полиморфизма для C++,
- 1993: STL и обобщённое программирование А. Степанова,
- 1994: метапрограммирование (пример Эрвина Унруха),
- 2001: “Modern C++ Design” Андрея Александреску (policy-based design, typelists, применение...).
- 2000s: Abrahams-Gurtovoy и Boost MPL, Extended C++ Мэтью Уилсона.

Outline

Основные программные модули

- Point<N, OrderPolicy> — элементы $\mathbb{Z}^N$,
- Polynomial<T> — многочлены,
- BMSAlgorithm< SeqT, PolynomialT, OrderPolicy > — BMS-алгоритм, основной конструктивный элемент декодера,
- BMSDecoding<N, ECCodeParams> — декодер.

Основные программные модули

- Point<N, OrderPolicy> — элементы $\mathbb{Z}^N$,
- Polynomial<T> — многочлены,
- BMSAlgorithm< SeqT, PolynomialT, OrderPolicy > — BMS-алгоритм, основной конструктивный элемент декодера,
- BMSDecoding<N, ECCodeParams> — декодер.

Основные программные модули

- Point<N, OrderPolicy> — элементы $\mathbb{Z}^N$,
- Polynomial<T> — многочлены,
- BMSAlgorithm< SeqT, PolynomialT, OrderPolicy > — BMS-алгоритм, основной конструктивный элемент декодера,
- BMSDecoding<N, ECCodeParams> — декодер.

Основные программные модули

- Point<N, OrderPolicy> — элементы $\mathbb{Z}^N$,
- Polynomial<T> — многочлены,
- BMSAlgorithm< SeqT, PolynomialT, OrderPolicy > — BMS-алгоритм, основной конструктивный элемент декодера,
- BMSDecoding<N, ECCodeParams> — декодер.

Outline

Outline

Идея и реализация

- Polynomial<...Polynomial<T>...> $\longrightarrow$ многочлены многих переменных;
- метаобёртка:

Код

```
template<int VarCnt, typename Coef>
struct MVPolyType {
  typedef Polynomial< // "recursive call"
      typename MVPolyType<VarCnt-1, Coef>::type > type;
};

template<typename Coef>
struct MVPolyType<1, Coef> {
  typedef Polynomial<Coef> type;
};
```

- MVPolyType<2, int> $\sim$ Ploynomial<Ploynomial<int>>

Идея и реализация

- Polynomial<...Polynomial<T>...> $\longrightarrow$ многочлены многих переменных;
- метаобёртка:

Код

```
template<int VarCnt, typename Coef>
struct MVPolyType {
  typedef Polynomial< // "recursive call"
      typename MVPolyType<VarCnt-1, Coef>::type > type;
};

template<typename Coef>
struct MVPolyType<1, Coef> {
  typedef Polynomial<Coef> type;
};
```

- MVPolyType<2, int> $\sim$ Ploynomial<Ploynomial<int>>

Идея и реализация

- Polynomial<...Polynomial<T>...> $\longrightarrow$ многочлены многих переменных;
- метаобёртка:

Код

```
template<int VarCnt, typename Coef>
struct MVPolyType {
  typedef Polynomial< // "recursive call"
      typename MVPolyType<VarCnt-1, Coef>::type > type;
};

template<typename Coef>
struct MVPolyType<1, Coef> {
  typedef Polynomial<Coef> type;
};
```

- MVPolyType<2, int> $\sim$ Ploynomial<Ploynomial<int>>

Идея и реализация

- Polynomial<...Polynomial<T>...> $\longrightarrow$ многочлены многих переменных;
- метаобёртка:

Код

```
template<int VarCnt, typename Coef>
struct MVPolyType {
  typedef Polynomial< // "recursive call"
      typename MVPolyType<VarCnt-1, Coef>::type > type;
};

template<typename Coef>
struct MVPolyType<1, Coef> {
  typedef Polynomial<Coef> type;
};
```

- MVPolyType<2, int> $\sim$ Ploynomial<Ploynomial<int>>

По аналогии

- Polynomial<Polynomial<int>>::VAR_CNT == 2
- Polynomial<Polynomial<int>>::CoefT == int

Умножение на скаляр

Код

```
Polynomial operator *= (CoefT const & c ) {
    for (typename StorageT::iterator it = data.begin();
            it != data.end(); ++it) {
        (*it) *= c;
    }
    return *this;
}
```

Сложение

Код (Начало)

```
template<typename T>
Polynomial<T>
Polynomial<T>::operator+=(Polynomial<T> const & p) {
  int degDiff = (this->data).size() - p.data.size();
  if (degDiff < 0) {
    typename StorageT::const_iterator
        commonPartDelimeter(p.data.begin());
    std::advance(commonPartDelimeter, data.size());
    std::copy(commonPartDelimeter, p.data.end(),
        std::back_inserter(data));
  } else {
    typename StorageT::const_iterator
        commonPartDelimeter(p.data.end());
  }
  // ...
```

Сложение

Код (Окончание)

```
    // ...
    typename StorageT::iterator itThis = data.begin();
    for (
        typename StorageT::const_iterator itP =
            p.data.begin();
        itP != commonPartDelimeter;
        ++itThis, ++itP) {
      *itThis += *itP;
    }
    return *this;
}
```

«Сложные» случаи

- Умножение на моном.
- Обращение к коэффициенту.
- ...

Такие задачи предполагают одновременный обход двух «рекурсивных» структур: многочлена от n переменных и точки в n-мерной решётке (Point<n>).

Outline

Характеристики коэффициентов

Код

```
template<typename CoefT>
struct CoefficientTraits {

    static CoefT multInverse(CoefT const & c) {
        return 1 / c;
    }
    static CoefT addInverse(CoefT const & c) {
        return -c;
    }
    static CoefT multId() {
        return 1;
    }
    static CoefT addId() {
        return CoefT();
    }
};
```

Характеристики коэффициентов NTL

Код

```
template<>
struct CoefficientTraits<NTL::GF2> {

    static T multInverse(T const & c) {
        return NTL::inv(c);
    }
    static T addInverse(T const & c) {
        return -c;
    }
    static T multId() {
        T a;
        NTL::set(a);
        return a;
    }
    // ...
};
```

Проблема

Что если несколько типов имеют один набор характеристик?

Простое и неудобное решение: определять специализации для каждого типа в отдельности.

Правильное решение: использовать Boost.MPL + Boost.enable_if.

Проблема

Что если несколько типов имеют один набор характеристик?

Простое и неудобное решение: определять специализации для каждого типа в отдельности.

Правильное решение: использовать Boost.MPL + Boost.enable_if.

Проблема

Что если несколько типов имеют один набор характеристик?

Простое и неудобное решение: определять специализации для каждого типа в отдельности.

Правильное решение: использовать Boost.MPL + Boost.enable_if.

Характеристики коэффициентов NTL

Код

```
typedef boost::mpl::vector<NTL::GF2E, NTL::ZZ_pE,
                           NTL::GF2, NTL::ZZ_p>
    NtlFieldTypes;

template<typename CoefT, typename Enable = void>
struct CoefficientTraits {/* generic definition ... */};

template<typename T>
struct CoefficientTraits<
        T,
        typename boost::enable_if<
            boost::mpl::contains<NtlFieldTypes, T>
        >::type > {
    // NTL-compatible definition ...
};
```

Outline

Агрегатная инициализация

Код

```
Point<2> mon;
mon[0] = 3;
mon[1] = 2;
```

Код (std::initializer_list<T>)

```
Point<2> mon {3, 2};
```

Синдром

Пусть даны $r = (r_1, \ldots, r_n) \in \mathbb{F}^n$ — пришедшее по каналу слово и набор точек алгебраической кривой $\{P_i \in \mathbb{F}^2\}_{i=1}^n$. Тогда

$$S\colon x^a y^b \mapsto (r_1, \ldots, r_n) \cdot ((x^a y^b)(P_1), \ldots, (x^a y^b)(P_n)).$$

Синдромная последовательность: $S(\{m_i(x, y)\})$.

Вычисление синдрома

Код (Начало: лямбда-функция C++11)

```
using namespace std::placeholders;
auto syndromComponentAtBasisElem =
  [this,&received](BasisElem const & be) ->
                    typename SyndromeType::value_type {
      auto tit = boost::make_transform_iterator(
          this->curvePoints.begin(),
          std::bind(
              computeMonomAtPoint<...>,
              be,
              _1));
      return typename SyndromeType::value_type(
          be,
          std::inner_product(received.begin(),
              received.end(),
              tit, FieldElemTraits<Field>::addId()));
  };
```

Вычисление синдрома

Код (Окончание)

```
//  ...
// Синдромная последовательность:
std::transform(
      basis.begin(),
      basis.end(),
      std::inserter(syn, syn.begin()),
      syndromComponentAtBasisElem);
```

Подробности

- на домашней странице [ссылка]
- код проекта на Google Code:
  `http://code.google.com/p/cpp-mv-poly/`