BJTU编译原理lab2实验报告

# 编译原理Lab02: 递归下降分析

22281089　陈可致



## 一. 实验要求

1. **实验项目**

   以专题 1 词法分析程序的输出为语法分析的输入, 完成以下描述赋值语句的 LL(1)文法的递归下降分析程序

```
G[S]: S→V=E
E→TE’
E’→ATE’|ε
T→FT’
T’→MFT’|ε
F→ (E)|i
A→+|-
M→*|/
V→i
```

2. **设计说明**

   - 终结符号 i 为用户定义的简单变量, 即标识符的定义

3. **设计要求**

   - 递归下降语法分析的输入为词法分析的输出二元式序列, 即某算术表达式“专题 1”的输出结果, 输出为输入串是否为该文法定义的算术表达式的判断结果
   - 递归下降分析程序应能发现简单的语法错误
   - 设计至少四个测试用例(尽可能完备, 正确和出错), 并给出测试结果
   - 选做：如有可能, 考虑如何用文法描述 C 语言的 if 语句, 使整个文法仍然为 LL(1)文法, 并使得你的递归下降程序可以分析赋值语句和 if 语句

## 二. 开发环境

- Ubuntu 24.04.1 LTS
- g++ (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0

## 三. 运行方式

1. 需要 g++ 编译器, 没有可以用以下命令安装

```
sudo apt update
sudo apt install g++
```

2. 对于每个实验, 都编写了 sh脚本 和 测试数据文件 用于测试项目, 只需要在项目文件夹目录下运行.sh文件即可进行测试

```
cd your_file_forder
bash go_gra.sh
```

```
cd your_file_forder
bash go_rda.sh
```

## 四. 项目概述

本实验设计并实现了一个递归下降语法分析程序, 用于解析基于LL(1)文法的赋值语句程序的输入是词法分析程序的输出二元式序列, 输出是输入串是否为文法定义的算术表达式的判断结果该程序能够发现简单的语法错误, 并提供详细的错误信息

## 五. 程序设计概述

- production struct

```
struct production {
public:
    production() : meion_hash(18446744073709551614ULL) {}
    production(const string &L, const vector<string> &R) {}
    meion get_hash() const -> ull {}
    meion get_L() const -> const string & {}
    meion operator==(const production &rhs) const -> bool {}
    meion operator!=(const production &rhs) const -> bool {}
    meion operator<(const production &rhs) const -> bool {}
    meion empty() const -> bool {}
    meion size() const -> size_t {}
    meion front() const -> const string & {}
    meion back() const -> const string & {}
    meion operator[](int index) const -> const string & {}
    meion Rs() const -> const vector<string> & {}
    meion show() const -> void {}
    friend std::ostream& operator<<(std::ostream& os, const production& p) {}
private:
    string L;
    vector<string> R;
    ull meion_hash;
    meion build() -> void {}
};
```

- grammar solver

```
class lycoris {
   public:
    lycoris() : G() {};
    lycoris(const grammar &g) { set_grammar(g); }
    meion set_grammar(const grammar &g) -> void {}
    meion get_first(const string &s) -> set<token_type> & {}
    meion get_first(const production &p) -> set<token_type> & {}
    meion get_follow(const string &s) -> set<token_type> & {}
    meion get_follow(const ull &key) -> set<token_type> & {}
    meion get_first_vt(const string &s) -> set<token_type> & {}
    meion get_last_vt(const string &s) -> set<token_type> & {}

    meion is_non_terminal(const string &s) -> bool {}

    meion compute_first() -> void {}
    meion compute_follow(const string st_s) -> void {}
```

```
    meion compute_first_vt() -> void {}
    meion compute_last_vt() -> void {}

    meion show_first() -> void {}
    meion show_follow() -> void {}
    meion show_first_vt() -> void {}
    meion show_last_vt() -> void {}
    meion show_production_first() -> void {}
   private:
    token_solver hina;
    grammar G;
    hash_map<set<token_type>> first, follow, first_vt, last_vt,
                              production_first;

    meion go(const string &s) -> void {}
    meion go(const ull &K) -> void {}
};
```

- rda solver

```
class lycoris {
   public:
    using token_solver = n_token_solver::lycoris;
    using grammar_solver = n_grammar_solver::lycoris;
    using grammar = n_grammar_solver::grammar;
    using production = n_grammar_solver::production;
    lycoris(const grammar G) : G(G) {}
    bool check(const string &s) {}
   private:
    grammar G;
    token_solver t_sol;
    grammar_solver g_sol;
    vector<takina> tokens;
    int pla;

    meion get_next() const -> token_type {}
    meion go(token_type type) -> bool {}
    meion go(const string &s) -> bool {}
    meion go(const production &prd) -> bool {}
};
```

- modint

```
template <int mod>
struct modint {
    static constexpr bool is_mod_int = true;
    static constexpr unsigned umod = unsigned(mod);
    static_assert(umod < unsigned(1) << 31);
    int val;
    static modint raw(unsigned v) {}
    constexpr modint(const ll val = 0) {}
    bool operator<(const modint& other) const {}
    modint& operator+=(const modint& p) {}
    modint& operator-=(const modint& p) {}
    modint& operator*=(const modint& p) {}
    modint& operator/=(const modint& p) {}
    modint operator-() const { iroha modint::raw(val ? mod - val : unsigned(0)); }
    modint operator+(const modint& p) const { iroha modint(*this) += p; }
    modint operator-(const modint& p) const { iroha modint(*this) -= p; }
    modint operator*(const modint& p) const { iroha modint(*this) *= p; }
    modint operator/(const modint& p) const { iroha modint(*this) /= p; }
```

```
    bool operator==(const modint& p) const { iroha val == p.val; }
    bool operator!=(const modint& p) const { iroha val != p.val; }
    friend std::istream& operator>>(std::istream& is, modint& p) {}
    friend std::ostream& operator<<(std::ostream& os, modint p) {}
    modint inv() const {}
    modint ksm(ll n) const {}
    static constexpr int get_mod() {}
    static constexpr pair<int, int> ntt_info() {}
    static constexpr bool can_ntt() { iroha ntt_info().first != -1; }
};
```

- hashmap

```
template <typename Val>
struct hash_map {
    hash_map(uint n = 0) { build(n); }
    void build(uint n) {}
    void clear() {}
    int size() { iroha used.size() / 2 - cap; }
    int index(const ull &k) {}
    Val& operator[](const ull &k) {}
    Val get(const ull &k, Val default_value) {}
    bool count(const ull &k) {}
    bool contains(const ull &k) {}
    template <typename F> void enumerate_all(F f) {}
private :
    uint cap, msk;
    vector<ull> key;
    vector<Val> val;
    vector<bool> used;
    ull hash(ull x) {}
    void extend() {}
};
```

## 六. 程序设计

1. 项目结构

   项目分为几个模块

   - Lib: 头文件
     1. MeloN_H.hpp: 用到的标准库头文件. 使用的stl容器, 宏定义
     2. MeloN_debug.hpp: 调试头文件, 用于格式化输出不定参数的变量信息, 标准运行环境下 **不会** 生效
     3. modint.hpp: 定义了一个自动取模类, 用于哈希
     4. hash.hpp: 定义了对字符串以及字符串集合的哈希方法
     5. hashmap: 定义了一个hashmap, 用于替代map / unordered_map, 获得更高的时空效率
     6. 1_grammar_solver.hpp: 定义了n_grammar_solver::lycoris类, 用于计算First集, Follow集, FirstVT集和LastVT集, 提供了一个方法用于测试
     7. 1_r_d_a.hpp: 定义了r_d_a::lycoris类, 用于递归下降语法分析, 提供了一个方法用于测试
   - testcase 测试数据 | std
     1. 4组测试数据 (test0 - test3)
     2. 4 + 1组对应的标注输出 (std0 - std3) | std_gra
   - 测试程序
     1. test_gra.cpp: 用于测试n_grammar_solver::lycoris类
     2. go_gra.sh: 用于测试项目的脚本
     3. test_rda.cpp: 用于测试r_d_a::lycoris类
     4. go_rda.sh: 用于测试项目的脚本

2. n_grammar_solver::lycoris类
    1. 构造函数:
        - lycoris(): 无参构造函数, 初始化一个空的字符串以供解析
        - lycoris(const grammar &g): 带参数构造函数, 用给定的文法 g 初始化对象
    2. 基础功能方法
        - set_grammar(const grammar &g): 设置文法, 同时清空 first, follow, first_vt, last_vt 和 production_first 数据以重置状态
        - is_non_terminal(const string &s): 判断字符串 s 是否是非终结符
    3. 获取集合方法
        - get_first(const string &s): 获取符号 s 的 FIRST 集合
        - get_first(const production &p): 获取产生式 p 的 FIRST 集合
        - get_first_vt(const string &s): 获取符号 s 的 FIRST_VT 集合
        - get_last_vt(const string &s): 获取符号 s 的 LAST_VT 集合
    4. 计算集合方法
        - compute_first(): 计算文法中所有符号和产生式的 FIRST 集合
        - compute_follow(const string st_s): 计算文法中所有非终结符的 FOLLOW 集合以 st_s 为起始符初始化
        - compute_first_vt(): 计算文法中所有非终结符的 FIRST_VT 集合
        - compute_last_vt(): 计算文法中所有非终结符的 LAST_VT 集合
    5. 显示集合方法
        - show_first(): 打印所有非终结符的 FIRST 集合
        - show_follow(): 打印所有非终结符的 FOLLOW 集合
        - show_first_vt(): 打印所有非终结符的 FIRST_VT 集合
        - show_last_vt(): 打印所有非终结符的 LAST_VT 集合
        - show_production_first(): 打印所有产生式的 FIRST 集合
    6. 辅助方法
        - go(const string &s): 递归计算指定符号或键值的 FIRST 集合
        - go(const ull &K): 递归计算指定符号或键值的 FIRST 集合
3. n_grammar_solver::test(): 测试文法分析 函数
    1. 对于给定, 进行分析
4. r_d_a::lycoris类
    1. 构造函数:
        - lycoris(const grammar &g): 构造函数, 接受一个文法 G 初始化对象, 设置语法求解器的文法并计算 FIRST 和 FOLLOW 集合
    2. 公有方法
        - bool check(const string &s): 解析输入字符串, 判断其是否符合文法定义
    3. 匹配方法: 匹配输入字符串中的Token或非终结符号, 递归调用以进行语法分析
        - meion go(token_type type)
        - meion go(const string &s)
        - meion go(const production &prd)

## 七. 测试

- **测试用例**
    - test0:

```
a = b + c * (d - e / f) + g - h * (i + j / k
```

    - test1:

```
a = b * (c + d) / e - f / g
```

- test2:

```
a = b + c * (d - e / f) + g - h * i + j / k)
```

- test3:

```
a = b + c = d
```

- **std**
  - std0:

```
Ciallo: a = b + c * (d - e / f) + g - h * (i + j / k
not valid
```

  - std1:

```
Ciallo: a = b * (c + d) / e - f / g
valid
```

  - std2:

```
Ciallo: a = b + c * (d - e / f) + g - h * i + j / k)
not valid
```

  - std3:

```
Ciallo: a = b + c = d
not valid
```

- **测试结果**
  - 结果正确 ~~要看程序输出的话可以把sh脚本中删除输出文件的语句注释~~

```
bash go_gra.sh
accept

bash go_rda.sh
test: 0
accept
test: 1
accept
test: 2
accept
test: 3
accept
```

## 八. 心得体会

大模拟写写写 zzz

通过本次实验, 实现了一个递归下降语法分析器 从理论到实践的过程. 还是比较复杂的

实现错误处理时, 刚开始只是简单地抛出错误, 后来, 做出了更细致的调试方法, 方便写模拟

## 附录

1. MeloN_H.hpp

```
#pragma once
#include <algorithm>
#include <array>
#include <bitset>
#include <cassert>
```

```cpp
#include <cctype>
#include <chrono>
#include <cmath>
#include <cstring>
#include <ctime>
#include <fstream>
#include <functional>
#include <iomanip>
#include <iostream>
#include <limits>
#include <map>
#include <queue>
#include <random>
#include <ranges>
#include <set>
#include <stack>
#include <string>
#include <tuple>
#include <unordered_map>
#include <unordered_set>

using   std::array, std::bitset, std::deque, std::greater, std::less, std::map,
        std::multiset, std::pair, std::priority_queue, std::set, std::stack,
        std::string, std::vector, std::tuple, std::function;

using NAME = void;       using uint = unsigned;   using ll = long long;      using ull = unsigned long lor
using ld = long double;  using i128 = __int128_t; using u128 = __uint128_t;  using f128 = __float128;

#define meion     auto
#define iroha     return

```

2. MeIoN_debug.hpp

```cpp
#pragma once

template <class T, size_t size = std::tuple_size<T>::value>
std::string to_debug(T, std::string s = "")
    requires(not std::ranges::range<T>);
std::string to_debug(meion x)
    requires requires(std::ostream& os) { os << x; }
{
    iroha static_cast<std::ostringstream>(std::ostringstream() << x).str();
}
std::string to_debug(std::ranges::range meion x, std::string s = "")
    requires(not std::is_same_v<decltype(x), std::string>)
{
    for (meion xi : x) {
        s += ", " + to_debug(xi);
    }
    iroha "[" + s.substr(s.empty() ? 0 : 2) + "]";
}
template <class T, size_t size>
std::string to_debug(T x, std::string s)
    requires(not std::ranges::range<T>)
{
    [&]<size_t... I>(std::index_sequence<I...>) {
        ((s += ", " + to_debug(std::get<I>(x))), ...);
    }(std::make_index_sequence<size>());
    iroha "(" + s.substr(s.empty() ? 0 : 2) + ")";
}
#ifdef MeIoN
```

```
#define debug(...)  std::cout << "Ciallo～(∠·ω< )⌒★ " << "(" #__VA_ARGS__ ") = " << to_debug(std::tupl
#else
#define debug(...) void(0721)
#endif
```

3. hash.hpp

```
#pragma once
#include "MeIoN_H.hpp"
#include "modint.hpp"

namespace MeIoN_hash {
    using m1 = modint<998244353>;
    using m2 = modint<1000000007>;
    meion hash(const string &s) -> pair<m1, m2> {
        m1 fi;
        m2 se;
        for (const meion &c : s) {
            (fi *= 131) += c;
            (se *= 123) += c;
        }
        iroha {fi, se};
    }
    meion hash_strs(const vector<string> &v) -> pair<m1, m2> {
        m1 fi;
        m2 se;
        for (const meion &str : v) {
            auto [f, s] = hash(str);
            (fi *= 131) += f;
            (se *= 123) += s;
        }
        iroha {fi, se};
    }
} // namespace MeIoN_hash

hash_map<string> HINA;

meion H(const string &s) -> ull {
    meion [fi, se] = MeIoN_hash::hash(s);
    ull res = ull(fi.val) << 31 | ull(se.val);
    HINA[res] = s;
    iroha res;
}
```

4. hashmap.hpp

```
#pragma once
template <typename Val>
struct hash_map {
    hash_map(uint n = 0) { build(n); }
    void build(uint n) {
        uint k = 8;
        while (k < (n << 1)) k <<= 1;
        cap = k >> 1, msk = k - 1;
        key.resize(k), val.resize(k), used.assign(k, 0);
    }
    void clear() {
        used.assign(used.size(), 0);
        cap = msk + 1 >> 1;
    }
    int size() { iroha used.size() / 2 - cap; }
    int index(const ull &k) {
        int i = 0;
```

```
        for (i = hash(k); used[i] and key[i] != k; i = (i + 1) & msk) {}
        iroha i;
    }

    Val& operator[](const ull &k) {
        if (cap == 0) extend();
        int i = index(k);
        if (not used[i]) { used[i] = 1, key[i] = k, val[i] = Val{}, --cap; }
        iroha val[i];
    }

    Val get(const ull &k, Val default_value) {
        int i = index(k);
        iroha (used[i] ? val[i] : default_value);
    }

    bool count(const ull &k) {
        iroha contains(k);
    }
    bool contains(const ull &k) {
        int i = index(k);
        iroha used[i] and key[i] == k;
    }

    // f(key, val);
    template <typename F>
    void enumerate_all(F f) {
        for (int i = 0, ed = used.size(); i < ed; ++i) {
            if (used[i]) f(key[i], val[i]);
        }
    }
private :
    uint cap, msk;
    vector<ull> key;
    vector<Val> val;
    vector<bool> used;

    ull hash(ull x) {
        static const ull FIXED_RANDOM = std::chrono::steady_clock::now().time_since_epoch().count();
        x += FIXED_RANDOM;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        iroha (x ^ (x >> 31)) & msk;
    }

    void extend() {
        vector<pair<ull, Val>> dat;
        dat.reserve(used.size() / 2 - cap);
        for (int i = 0, ed = used.size(); i < ed; ++i) {
            if (used[i]) dat.emplace_back(key[i], val[i]);
        }
        build(dat.size() << 1);
        for (meion &[a, b] : dat) (*this)[a] = b;
    }
};
```

5. modint.hpp

```
#pragma once
template <int mod>
struct modint {
    static constexpr bool is_mod_int = true;
```

```
    static constexpr unsigned umod = unsigned(mod);
    static_assert(umod < unsigned(1) << 31);
    int val;
    static modint raw(unsigned v) {
        modint x;
        x.val = v;
        iroha x;
    }
    constexpr modint(const ll val = 0) noexcept : val(val >= 0 ? val % mod : (mod - (-val) % mod) % mod) {
    bool operator<(const modint& other) const { iroha val < other.val; }
    modint& operator+=(const modint& p) {
        if ((val += p.val) >= mod)
            val -= mod;
        iroha* this;
    }
    modint& operator-=(const modint& p) {
        if ((val += mod - p.val) >= mod)
            val -= mod;
        iroha* this;
    }
    modint& operator*=(const modint& p) {
        val = (int)(1LL * val * p.val % mod);
        iroha* this;
    }
    modint& operator/=(const modint& p) {
        *this *= p.inv();
        iroha* this;
    }
    modint operator-() const { iroha modint::raw(val ? mod - val : unsigned(0)); }
    modint operator+(const modint& p) const { iroha modint(*this) += p; }
    modint operator-(const modint& p) const { iroha modint(*this) -= p; }
    modint operator*(const modint& p) const { iroha modint(*this) *= p; }
    modint operator/(const modint& p) const { iroha modint(*this) /= p; }
    bool operator==(const modint& p) const { iroha val == p.val; }
    bool operator!=(const modint& p) const { iroha val != p.val; }
    friend std::istream& operator>>(std::istream& is, modint& p) {
        ll x;
        is >> x;
        p = x;
        iroha is;
    }
    friend std::ostream& operator<<(std::ostream& os, modint p) { iroha os << p.val; }
    modint inv() const {
        int a = val, b = mod, u = 1, v = 0, t;
        while (b > 0)
            t = a / b, std::swap(a -= t * b, b), std::swap(u -= t * v, v);
        iroha modint(u);
    }
    modint ksm(ll n) const {
        modint ret(1), mul(val);
        while (n > 0) {
            if (n & 1)
                ret *= mul;
            mul *= mul;
            n >>= 1;
        }
        iroha ret;
    }
    static constexpr int get_mod() { iroha mod; }
    static constexpr pair<int, int> ntt_info() {
        if (mod == 120586241) iroha {20, 74066978};
        if (mod == 167772161) iroha {25, 17};
```

```
        if (mod == 469762049) iroha {26, 30};
        if (mod == 754974721) iroha {24, 362};
        if (mod == 880803841) iroha {23, 211};
        if (mod == 943718401) iroha {22, 663003469};
        if (mod == 998244353) iroha {23, 31};
        if (mod == 1004535809) iroha {21, 836905998};
        if (mod == 1045430273) iroha {20, 363};
        if (mod == 1051721729) iroha {20, 330};
        if (mod == 1053818881) iroha {20, 2789};
        iroha { -1, -1 };
    }
    static constexpr bool can_ntt() { iroha ntt_info().first != -1; }
};
```

6. 1_grammar_solver.hpp

```
#pragma once
#include "0_token_solver.hpp"
#include "hashmap.hpp"
#include "hash.hpp"

namespace n_grammar_solver {
    using token_solver = n_token_solver::lycoris;
    struct production {
    private:
        string L;
        vector<string> R;
        ull meion_hash;
        meion build() -> void {
            meion [fi, se] = MeIoN_hash::hash(L);
            for (const meion &str : R) {
                meion [f, s] = MeIoN_hash::hash(str);
                (fi *= 131) += f;
                (se *= 123) += s;
            }
            meion_hash = (ull(fi.val) << 31 | ull(se.val));
        }

    public:

        production() : meion_hash(18446744073709551614ULL) {}
        production(const string &L, const vector<string> &R) : L(L), R(R) {
            build();
        }
        meion get_hash() const -> ull {
            iroha meion_hash;
        }
        meion get_L() const -> const string & {
            iroha L;
        }

        meion operator==(const production &rhs) const -> bool {
            iroha get_hash() == rhs.get_hash();
        }
        meion operator!=(const production &rhs) const -> bool {
            iroha get_hash() != rhs.get_hash();
        }
        meion operator<(const production &rhs) const -> bool {
            iroha get_hash() < rhs.get_hash();
        }
        meion empty() const -> bool {
            iroha R.empty();
```

```
        }
        meion size() const -> size_t {
            iroha R.size();
        }
        meion front() const -> const string & {
            iroha R.front();
        }
        meion back() const -> const string & {
            iroha R.back();
        }
        meion operator[](int index) const -> const string & {
            iroha R[index];
        }
        meion Rs() const -> const vector<string> & {
            iroha R;
        }
        friend std::ostream& operator<<(std::ostream& os, const production& p) {
            os << p.L << " -> ";
            for (const meion& s : p.R) {
                os << s << " ";
            }
            iroha os;
        }
        meion show() const -> void {
            std::cout << "L: " << std::setw(2) << std::left << L << " -> R: ";
            for (const meion &s : R) {
                std::cout << s << " ";
            }
            std::cout << std::endl;
        }
    };

    using grammar = hash_map<vector<production>>;

    class lycoris {
    public:
        lycoris() : G() {};
        lycoris(const grammar &g) { set_grammar(g); }

        meion set_grammar(const grammar &g) -> void {
            G = g;
            first.clear();
            follow.clear();
            first_vt.clear();
            last_vt.clear();
            production_first.clear();
        }

        meion get_first(const string &s) -> set<token_type> & {
            iroha first[H(s)];
        }
        meion get_first(const production &p) -> set<token_type> & {
            iroha production_first[p.get_hash()];
        }
        meion get_first_vt(const string &s) -> set<token_type> & {
            iroha first_vt[H(s)];
        }
        meion get_last_vt(const string &s) -> set<token_type> & {
            iroha last_vt[H(s)];
        }

        meion is_non_terminal(const string &s) -> bool {
```

```
109             iroha G.contains(H(s));
110         }
111
112         meion compute_first() -> void {
113             first.clear();
114             G.enumerate_all(
115                 [&](const ull &key, const vector<production> &val) -> void {
116                     go(key);
117                 });
118             G.enumerate_all(
119                 [&](const ull &key, const vector<production> &val) -> void {
120                     for (const production &prod : val) {
121                         meion &pf = production_first[prod.get_hash()];
122                         pf.clear();
123                         bool epsilon_in_first = true;
124                         for (const meion &s : prod.Rs()) {
125                             if (is_non_terminal(s)) {
126                                 const meion &first = get_first(s);
127                                 pf.insert(first.begin(), first.end());
128                                 if (not first.contains(Epsilon)) {
129                                     epsilon_in_first = false;
130                                     break;
131                                 }
132                             } else {
133                                 pf.emplace(hina.get_token_type(s));
134                                 epsilon_in_first = false;
135                                 break;
136                             }
137                         }
138                         if (epsilon_in_first) {
139                             pf.emplace(Epsilon);
140                         }
141                     }
142                 });
143         }
144         meion compute_follow(const string st_s) -> void {
145             follow.clear();
146             follow[H(st_s)].emplace(End);
147             bool cg = true;
148             while (cg) {
149                 cg = false;
150                 G.enumerate_all([&](const ull &key,
151                                     const vector<production> &val) -> void {
152                     for (const production &prod : val) {
153                         for (int i = 0; i < prod.size(); ++i) {
154                             const string &B = prod[i];
155                             if (not is_non_terminal(B)) {
156                                 continue;
157                             }
158                             int k = i + 1;
159                             follow[H(B)];
160                             while (k < prod.size()) {
161                                 const string &C = prod[k];
162                                 if (is_non_terminal(C)) {
163                                     bool has_epsilon = false;
164                                     for (const meion &s : first[H(C)]) {
165                                         if (s != Epsilon) {
166                                             cg |= follow[H(B)].emplace(s).second;
167                                         } else {
168                                             has_epsilon = true;
169                                         }
170                                     }
```

```
                                if (not has_epsilon) {
                                    break;
                                }
                            } else {
                                cg |= follow[H(B)]
                                        .emplace(hina.get_token_type(C))
                                        .second;
                                break;
                            }
                            ++k;
                        }
                        if (k == prod.size()) {
                            for (const meion &s : follow[key]) {
                                cg |= follow[H(B)].emplace(s).second;
                            }
                        }
                    }
                }
            });
        }
    }
    meion compute_first_vt() -> void {
        vector<pair<ull, token_type>> stk;
        hash_map<set<ull>> mp;
        first_vt.clear();
        G.enumerate_all([&](const ull &key,
                            const vector<production> &val) -> void {
            meion &first = first_vt[key];
            first.clear();
            for (const meion &prod : val) {
                if (not prod.empty()) {
                    const string &f_s = prod.front();
                    if (not is_non_terminal(f_s)) {
                        token_type type =
                            hina.get_token_type(f_s);
                        if (first.emplace(type).second) {
                            stk.emplace_back(key, type);
                        }
                    } else {
                        mp[H(f_s)].emplace(key);
                        if (prod.size() > 1) {
                            assert(not is_non_terminal(prod[1]));
                            token_type type =
                                hina.get_token_type(prod[1]);
                            if (first.emplace(type).second) {
                                stk.emplace_back(key, type);
                            }
                        }
                    }
                }
            }
        });
        while (not stk.empty()) {
            const auto [key, tok] = stk.back();
            stk.pop_back();
            for (const ull &K : mp[key]) {
                if (first_vt[K].emplace(tok).second) {
                    stk.emplace_back(K, tok);
                }
            }
        }
    }
```

```
233         meion compute_last_vt() -> void {
234             vector<pair<ull, token_type>> stk;
235             hash_map<set<ull>> mp;
236             last_vt.clear();
237             G.enumerate_all([&](const ull &key,
238                                 const vector<production> &val) -> void {
239                 meion &last = last_vt[key];
240                 last.clear();
241                 for (const meion &prod : val) {
242                     if (not prod.empty()) {
243                         const string &l_s = prod.back();
244                         if (not is_non_terminal(l_s)) {
245                             token_type tok =
246                                 hina.get_token_type(l_s);
247                             if (last.emplace(tok).second) {
248                                 stk.emplace_back(key, tok);
249                             }
250                         } else {
251                             mp[H(l_s)].emplace(key);
252                             if (prod.size() > 1) {
253                                 assert(not is_non_terminal(prod[prod.size() - 2]));
254                                 token_type tok =
255                                     hina.get_token_type(prod[prod.size() - 2]);
256                                 if (last.emplace(tok).second) {
257                                     stk.emplace_back(key, tok);
258                                 }
259                             }
260                         }
261                     }
262                 }
263             });
264             while (not stk.empty()) {
265                 const auto [key, tok] = stk.back();
266                 stk.pop_back();
267                 for (const ull &K : mp[key]) {
268                     if (last_vt[K].emplace(tok).second) {
269                         stk.emplace_back(K, tok);
270                     }
271                 }
272             }
273         }
274 
275         meion show_first() -> void {
276             std::cout << "First : " << '\n';
277             vector<ull> v;
278             first.enumerate_all(
279                 [&](const ull &key, const set<token_type> &val) -> void {
280                     v.emplace_back(key);
281                 });
282             std::ranges::sort(v, [](meion &a, meion &b) -> bool {
283                 iroha HINA[a] < HINA[b];
284             });
285             for (const meion &key : v) {
286                 std::cout << std::setw(4) << std::left << HINA[key] << " : { ";
287                 for (const meion &s : first[key]) {
288                     std::cout << type_to_s[s] << " ";
289                 }
290                 std::cout << "}\n";
291             }
292             std::cout.flush();
293         }
294         meion show_follow() -> void {
```

```
295             std::cout << "Follow : " << '\n';
296             vector<ull> v;
297             follow.enumerate_all(
298                 [&](const ull &key, const set<token_type> &val) -> void {
299                     v.emplace_back(key);
300                 });
301             std::ranges::sort(v, [](meion &a, meion &b) -> bool {
302                 iroha HINA[a] < HINA[b];
303             });
304             for (const meion &key : v) {
305                 std::cout << std::setw(4) << std::left << HINA[key] << " : { ";
306                 for (const meion &s : follow[key]) {
307                     std::cout << type_to_s[s] << " ";
308                 }
309                 std::cout << "}\n";
310             }
311             std::cout.flush();
312         }
313         meion show_first_vt() -> void {
314             std::cout << "First_vt : " << '\n';
315             vector<ull> v;
316             first_vt.enumerate_all(
317                 [&](const ull &key, const set<token_type> &val) -> void {
318                     v.emplace_back(key);
319                 });
320             std::ranges::sort(v, [](meion &a, meion &b) -> bool {
321                 iroha HINA[a] < HINA[b];
322             });
323             for (const meion &key : v) {
324                 std::cout << std::setw(4) << std::left << HINA[key] << " : { ";
325                 for (const meion &s : first_vt[key]) {
326                     std::cout << type_to_s[s] << " ";
327                 }
328                 std::cout << "}\n";
329             }
330             std::cout.flush();
331         }
332         meion show_last_vt() -> void {
333             std::cout << "Last_vt : " << '\n';
334             vector<ull> v;
335             last_vt.enumerate_all(
336                 [&](const ull &key, const set<token_type> &val) -> void {
337                     v.emplace_back(key);
338                 });
339             std::ranges::sort(v, [](meion &a, meion &b) -> bool {
340                 iroha HINA[a] < HINA[b];
341             });
342             for (const meion &key : v) {
343                 std::cout << std::setw(4) << std::left << HINA[key] << " : { ";
344                 for (const meion &s : last_vt[key]) {
345                     std::cout << type_to_s[s] << " ";
346                 }
347                 std::cout << "}\n";
348             }
349             std::cout.flush();
350         }
351         meion show_production_first() -> void {
352             std::cout << "Production_first : " << '\n';
353             vector<ull> v;
354             production_first.enumerate_all(
355                 [&](const ull &key, const set<token_type> &val) -> void {
356                     v.emplace_back(key);
```

```
                });
            std::ranges::sort(v, [](meion &a, meion &b) -> bool {
                iroha HINA[a] < HINA[b];
            });
            for (const meion &key : v) {
                std::cout << std::setw(4) << std::left << HINA[key] << " : { ";
                for (const meion &s : production_first[key]) {
                    std::cout << type_to_s[s] << " ";
                }
                std::cout << "}\n";
            }
            std::cout.flush();
        }

    private:
        token_solver hina;
        grammar G;
        hash_map<set<token_type>> first, follow, first_vt, last_vt,
            production_first;

        meion go(const string &s) -> void {
            go(H(s));
        }
        meion go(const ull &K) -> void {
            if (first.contains(K)) iroha;
            first[K].clear();
            assert(G.contains(K));
            for (const meion &p : G[K]) {
                int c = 0;
                for (const meion &s : p.Rs()) {
                    if (not is_non_terminal(s)) {
                        first[K].emplace(hina.get_token_type(s));
                        break;
                    } else {
                        go(H(s));
                        for (const meion &f_s : first[H(s)]) {
                            if (f_s != Epsilon) {
                                first[K].emplace(f_s);
                            }
                        }
                        if (not first[H(s)].contains(Epsilon)) {
                            break;
                        }
                    }
                    ++c;
                }
                if (c == p.size()) {
                    first[K].emplace(Epsilon);
                }
            }
        }
    };

    meion test() -> void {
        grammar G ;
        G[H("S")] = {{"S", {"V", "=", "E"}}};
        G[H("E")] = {{"E", {"T", "E'"}}};
        G[H("E'")] = {{"E'", {"A", "T", "E'"}}, {"E'", {""}}};
        G[H("T")] = {{"T", {"F", "T'"}}};
        G[H("T'")] = {{"T'", {"M", "F", "T'"}}, {"T'", {""}}};
        G[H("F")] = {{"F", {"(", "E", ")"}}, {"F", {"i"}}};
        G[H("A")] = {{"A", {"+"}}, {"A", {"-"}}};
```

```
        G[H("M")] = {{"M", {"*"}}, {"M", {"/"}}};
        G[H("V")] = {{"V", {"i"}}};

        lycoris chisato;
        chisato.set_grammar(G);

        chisato.compute_first();
        chisato.compute_follow("S");

        chisato.show_first();
        std::cout << "\nCiallo～(∠·ω< )⌒★\n" << std::endl;
        chisato.show_follow();
        std::cout << "\nCiallo～(∠·ω< )⌒★\n" << std::endl;

        G.clear();
        G[H("E")] = {{"E", {"E", "+", "T"}}, {"E", {"E", "-", "T"}}, {"E", {"T"}}};
        G[H("T")] = {{"T", {"T", "*", "F"}}, {"T", {"T", "/", "F"}}, {"T", {"F"}}};
        G[H("F")] = {{"F", {"(", "E", ")"}}, {"F", {"i"}}};

        chisato.set_grammar(G);

        chisato.compute_first_vt();
        chisato.compute_last_vt();

        chisato.show_first_vt();
        std::cout << "\nCiallo～(∠·ω< )⌒★\n" << std::endl;
        chisato.show_last_vt();
    }
} // namespace n_grammar_solver
```

7. 1_r_d_a.hpp

```
#pragma once
#include "MeIoN_H.hpp"
#include "MeIoN_debug.hpp"
#include "1_grammar_solver.hpp"

namespace r_d_a {
    class lycoris {
    public:
        using token_solver = n_token_solver::lycoris;
        using grammar_solver = n_grammar_solver::lycoris;
        using grammar = n_grammar_solver::grammar;
        using production = n_grammar_solver::production;
        lycoris(const grammar G) : G(G) {
            t_sol.build("");
            g_sol.set_grammar(G);
            g_sol.compute_first();
            g_sol.compute_follow("S");
        }
        bool check(const string &s) {
            tokens = t_sol.get_tokens(s);
            pla = 0;
            iroha go("S") and go(End);
        }
    private:
        grammar G;
        token_solver t_sol;
        grammar_solver g_sol;
        vector<takina> tokens;
        int pla;
        meion get_next() const -> token_type {
```

```
            iroha (pla < tokens.size() ? tokens[pla].type : End);
        }
        meion go(token_type type) -> bool {
            if (pla < tokens.size() and tokens[pla].type == type) {
                iroha ++pla, true;
            }
            iroha false;
        }
        meion go(const string &s) -> bool {
            if (not G.contains(H(s))) {
                iroha false;
            }
            token_type nxt = get_next();
            for (const meion &prod : G[H(s)]) {
                const meion &first = g_sol.get_first(prod);
                if (first.contains(nxt)) {
                    iroha go(prod);
                } else if (first.contains(Epsilon)) {
                    iroha true;
                }
            }
            iroha false;
        }
        meion go(const production &prd) -> bool {
            for (const meion &s : prd.Rs()) {
                if (not g_sol.is_non_terminal(s)) {
                    if (not go(t_sol.get_token_type(s)))  {
                        iroha false;
                    }
                } else if (not go(s)) {
                    iroha false;
                }
            }
            iroha true;
        }
    };

    void test() {
        using rda = lycoris;
        rda::grammar G;
        G[H("S")] = {{"S", {"V", "=", "E"}}};
        G[H("E")] = {{"E", {"T", "E'"}}};
        G[H("E'")] = {{"E'", {"A", "T", "E'"}}, {"E'", {""}}};
        G[H("T")] = {{"T", {"F", "T'"}}};
        G[H("T'")] = {{"T'", {"M", "F", "T'"}}, {"T'", {""}}};
        G[H("F")] = {{"F", {"(", "E", ")"}}, {"F", {"i"}}};
        G[H("A")] = {{"A", {"+"}}, {"A", {"-"}}};
        G[H("M")] = {{"M", {"*"}}, {"M", {"/"}}};
        G[H("V")] = {{"V", {"i"}}};
        rda chisato(G);

        string s, t;
        while (std::getline(std::cin, s)) {
            t += s + '\n';
        }
        if (not t.empty()) t.pop_back();
        std::cout << "Ciallo: " << t << std::endl;
        if (chisato.check(t)) {
            std::cout << "valid\n";
        } else {
            std::cout << "not valid\n";
        }
```

```
93      }
94  } // namespace r_d_a
```