**变更记录**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 变更编号 | 版本号 | 日期 | 变更状态 | 变更简单描述 | 撰写人 | 审核人 |
| 1 | 1.0 | 20130812 |  | 初始版本 | 林基远 |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |
| 15 |  |  |  |  |  |  |
| 16 |  |  |  |  |  |  |
| 17 |  |  |  |  |  |  |
| 18 |  |  |  |  |  |  |

变更状态：**A** – 增加 **M** – 修改 **D** – 删除

**基于规则的关系抽取组件*drawline*设计文档**

目录

[1. *drawline*使用场景简介 3](#_Toc366089602)

[2. 总体设计 3](#_Toc366089603)

[2.1 流程设计 3](#_Toc366089604)

[2.2 类设计 4](#_Toc366089605)

[3. 类详细设计 5](#_Toc366089606)

[3.1 辅助类 5](#_Toc366089607)

[3.2 *drawline*自身类 7](#_Toc366089608)

[4. 关键部分实现细节 8](#_Toc366089609)

[4.1 转化规则 8](#_Toc366089610)

[4.2 higher level match 8](#_Toc366089611)

[附录 9](#_Toc366089612)

[规则定义 9](#_Toc366089613)

[注意事项 10](#_Toc366089614)

# *drawline*使用场景简介

***drawline*** 是一个**基于规则的关系抽取组件**，它对***MTIE (Mengtao's information Extraction)*** 进行重写，目的是更好地维护以及减少抽取时间。 ***drawline***本身不会对文章进行命名实体抽取，它是对命名实体抽取结果按照规则抽出实体间的关系，比如

|  |  |
| --- | --- |
| 文章：  *玛依努尔接受了CNN的采访。热比娅的丈夫在接受采访的时候说. 热比娅的儿子呼吁美国议会.*  *热比娅受到了日本总理的邀请。奥巴马约见了热比娅。热比娅说要加强与藏青会的合作。* | 规则文件：  CONCEPT:NAME:玛依努尔  CONCEPT:LOCATION:美国  CONCEPT:ORGANIZATION:美国议会  CONCEPT:LOCATION:日本  CONCEPT:LOCATION:奥巴马  CONCEPT:ACTION:约见  CONCEPT:ACTION:接受  MCONCEPT\_RULE:PERSON\_ACTION(person, action):(SENT,(ORD,(DIST\_5, "\_person{NAME}", "\_action{ACTION}"))) |
| 结果：  164 15  person:奥巴马  action:约见  -----  1 18  person:玛依努尔  action:接受  -----  \*\*\*\*\*  NAME:玛依努尔  LOCATION:美国  ORGANIZATION:美国议会  LOCATION:日本  LOCATION:奥巴马    其中，在\*\*\*\*\*上的是抽出的关系，\*\*\*\*\*下的是概念。对于每个关系，第一行为offset和len，即关系对应字符串在原文本中的byet-offset，byte-len。接下来是关系，冒号前是规则的参数名，冒号后是实体实例。接着以一行-----结束该关系。 | |

详细的规则定义请见附录。

# 总体设计

## 2.1 流程设计

*drawline*分两个阶段：

1. build phrase，读取规则文件，将所有规则转化为一个有向无环图(DAG)，其中CONCEPT和REGEX类型处于最底层，而MONCEPT和MONCEPT\_RULE向其内部的CONCEPT和REGEX节点连边。同时对CONCEPT建立Aho-corasick Automata。
2. match phrase，读取文章，在AC自动机走一次得到所有匹配的CONCEPT实例；匹配REGEX实例。以上为低层次匹配，即只匹配连续串，匹配的这些实例即为build phrase建立的规则DAG底层节点的实例。高层次匹配将对这些实例按照规则DAG这份模板往上递归构造。



图1. *drawline*流程

## 2.2 类设计

1) 辅助类设计

* SymbolTable 符号表，映射字符串到整数，用于汉字、规则名字。
* MatchNode 匹配到的文本实例的位置记录，以及它的儿子，可以认为MatchNode就是DAG上一个点的实例，而DAG上的点是一个模板。
* Trie 字典树。
* AhoCorasick 自动机

2) *drawline*自身类设计

* Rule 记录输入的单条规则（规则名，实际规则，规则类型，参数等），在DAG中结构信息（包括邻接表、层次）
* Drawline

# 类详细设计

## 3.1 辅助类

这些类的声明在src/drawline\_ds.h，具体实现在src/drawline\_ds.cpp。

|  |
| --- |
| *SymbolTable* |
| */\**  *SymbolTable:*  *map a string to a unique id (start from 0)*  *\*/*  class SymbolTable {  public:  int operator[] (const std::wstring &key);  int count(const std::wstring &key);    std::map<std::wstring, int> table;  }; |

|  |
| --- |
| *MatchNode* |
| */\**  *MatchNode:*  *Record actual text match position and its children.*  *\*/*  class MatchNode {  public:  int st;  int ed;  int sent\_id;  int rule\_id;  int rule\_ix;  bool same\_sent;  std::vector<MatchNode \*> child\_nodes;    MatchNode(int \_rule\_ix, int \_rule\_id, int \_sent\_id, int \_st, int \_ed,  std::vector<MatchNode \*> &\_child\_nodes);  MatchNode(int \_rule\_ix, int \_rule\_id, int \_sent\_id, int \_st, int \_ed);  MatchNode(int \_rule\_ix, int \_sent\_id, int \_st, int \_ed);    bool operator < (const MatchNode &that); |
| 其中st,ed是匹配的整段文本左右端点位置（以词为单位，*drawline*中词的定义见附录），同时记录了这段文本是否为同一句话，它对应的模板规则信息、子节点信息。 |

|  |
| --- |
| *Trie* |
| */\**  *Trie:*  *Trie node, used in AC automaton*  *\*/*  class Trie {  public:  typedef std::map<int, Trie \*> MIT;  typedef std::vector<Trie \*> VT;  MIT child;  Trie \*fail; *// KMP fail pointer, used in building AC automaton*  SI tag; *// Store matched rule\_ids*  int tags;    Trie();  ~Trie();  Trie \*operator[] (int k);    private:  *// Record all the new nodes created in this node. Used for deconstruction.*  VT child\_vt;  }; |
| 由于字典树子节点不是固定数目的，无法使用静态数组，也开不了，这里使用了map。而没有用hash\_map的原因是经过实验发现hash\_map会慢很多，原因是整棵字典树节点很多，每个节点都有一个hash\_map的话，创建hash\_map就需要很多时间了。 |

|  |
| --- |
| *AhoCorasick* |
| */\* Aho-Corasick automaton \*/*  class AhoCorasick {  public:  typedef std::map<int, Trie \*> MIT;    AhoCorasick();  ~AhoCorasick();  void insert(int tag, const VI &word);  void insert(int tag, SymbolTable & table, const std::wstring &word);  void build();  VM match(SymbolTable & table, const std::wstring &text, VI &sent\_pos, VI &word\_pos);    private:  Trie \*root;  Trie \*find(Trie\*, int);  }; |
| 平常建立AC自动机时，其Trie节点是固定不变，只是连边而已。而这里Trie节点的子节点是动态添加，在建立自动机连边时会遇到fail指针要指向的节点不存在的情况，需要再生成fail指针指向的节点，而且是递归的。这里的实现是使用类似并查集寻找根时路径压缩的操作。Build和match阶段都需要。 |

## 3.2 *drawline*自身类

这些类的声明在src/drawline.h，具体实现在src/drawline.cpp。

|  |
| --- |
| *Rule* |
| class Rule {  public:  RULE\_TYPE rule\_type;  std::wstring rule;  std::wstring name;    VW words;  VII child; *// (which\_word, rule\_id)*    bool is\_input; *// is the rule inputed or temporally genereated*  bool is\_transformed;    *// name <-> id*  *// name map to id, and vice versa*  *// But a name/id may have several rules*  int id;  int ix; *// record its position in drawline rules*  int level;  *// for MCONCEPT\_RULE*  int dist\_k;  std::map<std::wstring, VI> arg\_child;  *// Not only record its constraint, but also its ancestors'.*  *// Of course, only if the ancestors have only one parents.*  *// By adding ancestors' constraints, can prune a lot combinations.*  int constraints;  CONSTRAINT\_TYPE constraint\_type;  Rule(const std::wstring &\_name, const std::wstring &\_rule,  RULE\_TYPE \_rule\_type, int \_ix, int \_id, bool \_is\_input);  *// split rules*  void split\_rule(SymbolTable &);  }; |
| 一条rule对应一个ix，但多个rule可以对应同一个id（这些rule同名时）。而一个rule id对应DAG的一个节点。由于MCONCEPT\_RULE的可以嵌套，即父亲的约束可以传递给儿子，在建立DAG时下传约束（比如距离约束DIST\_K，同一句话SENT，次序ORD等），这样能减少以后自底向上构造时不符合条件的情况。 |

|  |
| --- |
| *Drawline* |
| class Drawline {  public:  Drawline();  ~Drawline();  void push(const std::string &srule);  void build();  VS match(const std::string &text);  void print(const std::string &text);  void transform\_rules();  void print\_rules();  }; |
| 以上是*drawline*提供给外部调用的接口，push完规则build后就可以match。 |

# 关键部分实现细节

## 4.1 转化规则

这部分代码在Drawline:: transform\_rules()。

* 将每条规则看成一个DAG中的一个点（不同规则但同一名字的处于同一个点），规则里的子规则看成是儿子。
* 在转化一条规则时，使用深度优先遍历，即先枚举其所有儿子处理儿子规则转化，再后序处理当前规则，包括记录最大层次数以及入到该层节点的容器里。
* 对所有规则都转化后，再自顶向下传递父亲的约束。当儿子只有一个父亲时才继承其约束，有多个父亲时不继承（不然以后构造时有错误约束）。不同类型的约束有不同的继承策略，比如儿子的DIST\_K的继承父亲的DIST\_K使用的是min函数。继承约束这一步是为了增加限制减少无用的构造。

## 4.2 higher level match

这部分代码主要在

|  |
| --- |
| void Drawline::highlevel\_match(const std::wstring &text);  VM left\_construct(const Rule &rule, int i, int j, int st, int ed,  const std::wstring &text);  VM left\_construct(const Rule &rule, int st, int ed, const VM &all,  int now, VII child\_ids, VM child\_nodes);  void up\_construct(const std::wstring &text, MatchNode \*pt); |

其逻辑是：

* 对lower level match匹配到的实例，即MatchNode（已按照右端点排序），从左向右枚举。每遇到一个实例，看它能否触发父亲规则，即向上构造up\_construct。
* 触发是指该实例的父亲节点（或者说模板，规则）所需要匹配的所有子节点已经拥有了，可以新生一个该规则的实例。
* 为了避免重复构造，要求新生的父亲实例以当前枚举到的这个儿子实例处于最右端。
* 由于有多种可能，对于每一条可能触发的父亲规则，需要向左构造left\_construct。
* 同时如果新生了一个父亲实例，需要继续递归构造up\_construct。

# 附录

## 规则定义

目前，***drawline***支持:

* **定义概念**
  + **CONCEPT:concept\_name:sequence\_of\_words**

如 **CONCEPT**:COUNTRY:中国

* + **REGEX:UPSTR:[A-Z]{2,}**
  + **MCONCEPT:mconcept\_name:(word|CONCEPT|REGEX)**

如**MONCEPT**:SCHOOL:LOCNUM小学

**CONCEPT**:LOC:北京

**CONCEPT**:NUM:第[1-9]

由于定义了LOC，NUM， ***drawline*** 会自动将LOCNUM拆成LOC和NUM，不需在它们之间加空格（否则认为空格也需匹配）。 ***drawline***拆分单词时**优先考虑长单词**，如还定义了一个

**CONCEPT**:LOCNUM:北京第一

***drawline***就不会拆分LOCNUM，而是将LOCNUM当作一个整体， 因此不会用到LOC和NUM，这时不会匹配北京第2小学，而只会匹配北京第一小学。

* **定义规则**，如

**MCONCEPT\_RULE:obj\_pattern(per, loc, time, org):**

**(SENT, "\_per{PERSON}", "\_loc{LOCATION}", "\_time{TIME}", "\_org{ORG}")**

其中PERSON, LOCATION, TIME和ORG是定义的**CONCEPT**或**MCONCEPT**类型的概念。 而per, loc, time, org是参数名，将会在结果输出。模式规则可以任意**嵌套使用**(但不能递归), 如

**MCONCEPT\_RULE:obj\_pattern(per, loc, time, org): (SENT, (DIST\_2, "\_per{PERSON}", "\_loc{LOCATION}"), (OR, "\_time{TIME}", "\_org{ORG}"))**

目前支持的操作符有：

|  |  |
| --- | --- |
| 操作符 | 说明 |
| **AND** | 所有子句都出现的字符串才会被匹配。 |
| **OR** | 只要有一个子句出现，该字符串就会被匹配。 |
| **SENT** | 所有子句都出现在同一个句子，该字符串才会被匹配。  ***drawline*** 以".?!。？！"及换行符来**断句**（换行符是为了处理如表格型等无明确句号的文章）。 |
| **ORD** | 所有子句按规则定义的顺序同时出现，该字符串才会被匹配。 |
| **DIST\_n** | 所有子句同时出现在字符串，且**相邻**子句实例距离（**间隔距离**）不超过n个词时，该字符串会被匹配。  ***drawline*** 将**词**定义为：   * 单个中文符号（包括汉字，中文标点符号） * 断句符号（如上述） * 英文单词 * 数字串 * 连续的英文标点符号及空格 |

## 注意事项

* + 模板及文本文件都需要是UTF-8编码，结 果文件也是UTF-8编码
  + ***drawline*** 对输入规则的先后顺序没有要求
  + 定义概念及规则，严格按照附录的说明定义，如

1. 不要用中文冒号代替

**CONCEPT**：COUNTRY：中国

1. 不要用中文引号代替
   * 如果不需匹配，匹配串不要出现多余空格（特别是首尾），如

**CONCEPT**:NAME: xxx

这里xxx前的空格会被当作需要匹配的空格。

* + 如果满足条件的同一字符串在文本多个位置都可能出现，结果文件会把所有位置都输出
  + 同一名字可以有不同规则，同一名字的规则会被认为是同一类型的
  + 对于一个字符串同时满足多条规则的， *drawline* 的处理为：

1. 如果这多条规则是同一类型（名字相同），则只输出一次。如定义了

**MCONCEPT\_RULE**:NAME\_AGE(person, age):(SENT, "\_person{NAME}", "\_age{AGE}")))

**MCONCEPT\_RULE**:NAME\_AGE(person, age):(DIST\_5, "\_person{NAME}", "\_age{AGE}"))

**CONCEPT**:NAME:A

**CONCEPT**:AGE:[1-9] year

这两条规则会匹配"A is 5 years' old."，但只输出一次。

1. 如果这多条规则是不同类型，每种类型都会输出一次。如上面将第2个NAME\_AGE

修改为NAME\_AGE2。

* 对于**MCONCEPT\_RULE**只输出在有指定参数名

**MCONCEPT\_RULE**:NAME\_AGE(person, age):(SENT, "\_person{NAME}", "\_age{AGE}", "\_sex{SEX}")))

这里sex虽然是参数，但没有在NAME\_AGE(person, age)指定，因此不输出，只输出person, age。

* **MCONCEPT\_RULE**内的子句对于以下划线开头的变量认为是参数，如"\_person{NAME}"。

如果要在这个子句加入其他概念，概念名不要包含下划线。如

CONCEPT:ACTION\_ATTACK:攻击

MCONCEPT\_RULE::" ACTION\_ATTACK \_person{NAME}"

这时无法判断\_ATTACK \_person还是\_person才是参数。

* 由于有需求，对于**MCONCEPT\_RULE**如果参数为(person, person)，在最后的结果里会自动增加一个coexist参数，匹配的词为"同"（person匹配的词均为非英文时），"and" (person均为英文时)