Read CaboCha

#1 Header files

Yuta Hayashibe

2012 March 9th 17:00~(JST)

INTRO

What is CaboCha?

- Japanese dependency parsed based on Support Vector machine
- Open source software
  - http://code.google.com/p/cabocha/
- Latest version is 0.62 (Revision 36 at 2012 Feb 24th)

Why read CaboCha?

- Learn good ways of implementation
- To implement YuCha (Japanese predicate argument structure analyzer)

Schedule

1. Read header files
2. …

Topics

- Management features
- Management strings
- Techniques of speeding up
  - Memory allocation
  - Feature selection
- Implementation of NER
- SWIG (Simplified Wrapper and Interface Generator)
- Combination to MeCab
- …

Overview

- find src | grep -e h$ | xargs wc -l
  - 225,243
  - Most codes are data

    155511 src/ucstable.h
    65566 src/char_category.h
    1372 src/win32/mecab.h
    517 src/darts.h
    351 src/win32/crfpp.h
- find src | grep -e cpp$ | xargs wc -l
  - 6,235

DATA STRUCTURES

cabocha.h (enum)

```
enum cabocha_charset_t {
  EUC_JP, CP932, UTF8, ASCII
};
enum cabocha_posset_t  {
  IPA, JUMAN, UNIDIC
};
enum cabocha_format_t …
enum cabocha_input_layer_t …
enum cabocha_parser_t {
  TRAIN_NE,
  TRAIN_CHUNK,
  TRAIN_DEP
};
```

Define constants of CaboCha

cabocha.h (struct)

```
struct  cabocha_chunk_t {
  int                 link;
  unsigned short int    head_pos;
  unsigned short int    func_pos;
  unsigned short int    token_size;
  size_t              token_pos;
  float               score;
  const char          **feature_list;
  unsigned short int    feature_list_size;
};
typedef struct cabocha_chunk_t Chunk;
```

Represent “bunsetsu” (a group of morphemes)

```
struct cabocha_token_t {
  const char          *surface;
  const char          *normalized_surface;
  const char          *feature;
  const char          **feature_list;
  unsigned short int    feature_list_size;
  const char          *ne;
  struct cabocha_chunk_t  *chunk;
};
typedef struct cabocha_token_t Token;
```

Represent “keitaiso” (morpheme)

cabocha.h (tree)

```
class Tree {
 public:
  void set_sentence(const char *sentence);
  const char *sentence() const;
  size_t sentence_size() const;

  const Chunk *chunk(size_t i) const;
  const Token *token(size_t i) const;
```

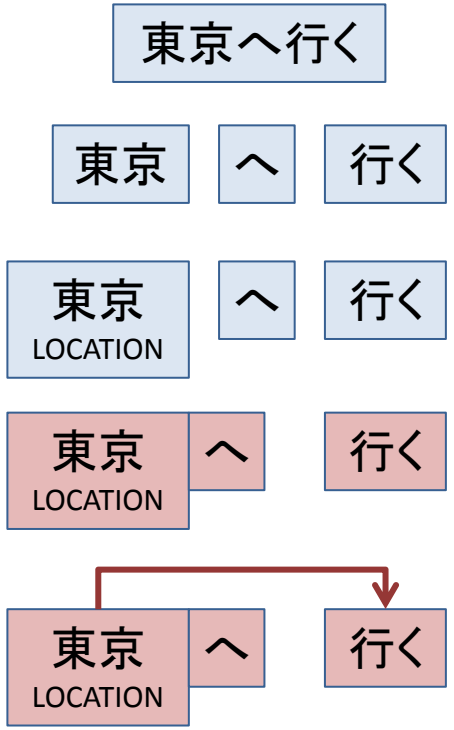
Store the data and the result of analysis

```
  bool   read(const char *input, InputLayerType
       input_layer);

  bool   empty() const;
  void   clear();
  void   clear_chunk();

  size_t chunk_size() const;
  size_t token_size() const;
  size_t size() const;

  const char *toString(FormatType output_format);

  CharsetType charset() const { return charset_; }
  void set_charset(CharsetType charset) { charset_ =
       charset; }
  PossetType posset() const { return posset_; }
  void set_posset(PossetType posset) { posset_ =
       posset; }
  OutputLayerType output_layer() const { return
       output_layer_; }
  void set_output_layer(OutputLayerType
       output_layer) { output_layer_ = output_layer; }

  const char *what();

  explicit Tree();
  virtual ~Tree();

 private:
  TreeAllocator        *tree_allocator_;
  CharsetType           charset_;
  PossetType           posset_;
  OutputLayerType        output_layer_;
};
```

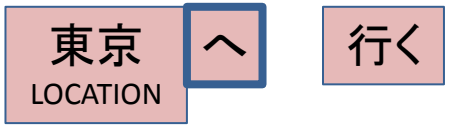
ANALYZERS

Analyzers (1/2)

- analyzer.h
  - class Analyzer (Pure virtual class)
    - virtual bool open(const Param &) = 0;
    - virtual void close() = 0;
    - virtual bool parse(Tree *tree) const = 0;
- morph.h
  - class MorphAnalyzer: public Analyzer
- ne.h
  - class NE : public Analyzer
- chunker.h
  - class Chunker: public Analyzer
- dep.h
  - class DependencyParser: public Analyzer
  - struct DependencyParserData



Analyzers (2/2)



- selector.h (For all chunks, find heads and funcs)
  - class Selector: public Analyzer
  - private: void findHead(const Tree &tree, const Chunk &chunk, size_t *hid, size_t *fid) const
- selector_pat.h

```
// Common
const char KUTOUTEN_PAT[] = "(。|、|,|.)";
const char OPEN_BRACKET_PAT [] = "((|（|‘|“|《|「|『|［|〈|｛)";
const char CLOSE_BRACKET_PAT[] = "()|）|’|”|》|」|』|］|〉|｝)";
const char DYN_A_PAT[] = "(助詞|副詞|連体詞|接続詞)";
const char CASE_PAT [] = "助詞";
// IPA
const char IPA_FUNC_PAT[] = "(助詞|助動詞|動詞,非自立|動詞,接尾|形容詞,非自立|形容詞,接尾)";
const char IPA_HEAD_PAT[] = "!(助詞|助動詞|動詞,非自立|動詞,接尾|形容詞,非自立|形容詞,接尾|
    空白|記号)";
// JUMAN
const char JUMAN_FUNC_PAT[] = "!特殊";
const char JUMAN_HEAD_PAT[] = "!(特殊|助詞|接尾辞)";
```

- Strings in source codes are written in UTF-8
  - In actuality, the hexadecimal escape sequence is used, because some compilers can’t accept UTF-8
  - const char JUMAN_FUNC_PAT[] = "!¥xE7¥x89¥xB9¥xE6¥xAE¥x8A";
  - It is same with mozc (Open Source Japanese Input Method)

UTILITIES

String utilities

- stream_wrapper.h
  - class istream_wrapper
  - class ostream_wrapper
- string_buffer.h
  - class StringBuffer
- darts.h
  - Double-ARray Trie

- normalizer{.h , _rule.h}
  - Convert one-byte katakanas to two byte ones
  - Convert two-byte symbols and alphabets to one byte ones

String data

- char_category.h.
  - Define character categories (eg: KANJI, HIRAGANA, SYMBOL) of **UCS-2** (Internal character set of CaboCha)

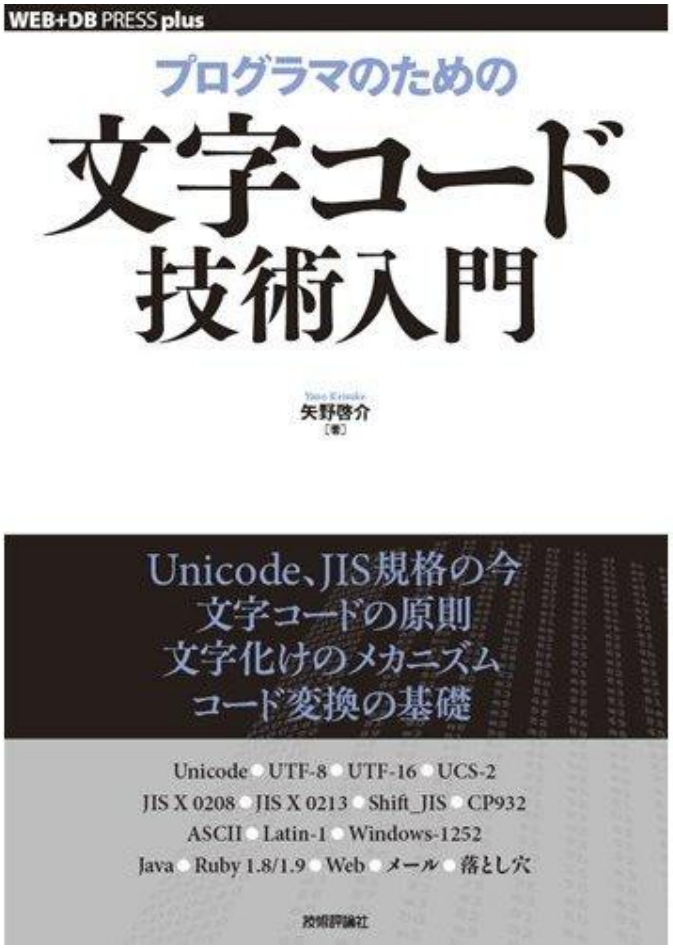
UCS-2(2bytes) ⊆ UTF-16(2 or 4bytes) ⊆ (UCS-4(4bytes)=UTF-8(1~4bytes))

- ucs.h

Enable ifndef CABOCHA_USE_UTF8_ONLY

  - unsigned short {utf8, ascii, *euc, cp932*}_to_ucs2
- ucstable.h
  - {CP932, EUC-JP }to UCS2 table
    - static const unsigned short int cp932_tbl [65536]
    - static const unsigned short int euc_tbl [65536]
  - EUC-JP to UC2 table, (offset 41120)
    - static const unsigned short int euc_hojo_tbl [65536]

A book of reference (in Japanese)



Other utilities

- common.h
- param.h
  - Manage parameters of CaboCha

- svm.h
- svm_learn.h

- freelist.h
  - Release of memories
- mmap.h
  - http://chasen.org/~taku/blog/archives/2009/04/ioio.html
- tree_allocator.h
- scoped_ptr.h

- timer.h
- utils.h
- winmain.h

CONCLUTION

Conclusion

- So many codes in CaboCha
- Why don’t you read Cabocha?
  - Welcome to join this “Read CaboCha” series