Formal languages and compilers

Projects 2018

Henri LEFEBVRE

February 18, 2019

1. Top-down parser : begin...end

2. Bottom-up parser : NLP with multiple clauses

Introduction

Problem statement

Parse input strings s respecting the following rules :

- s must start with "begin" and end with "end"
- both "begin...end" and "{...}" define block structures which must not overlap in s

Valid example

```
begin
    <text>
    {
        <text>
    }
    <text>
end
```

Invalid example

```
begin
    <text>
    {
        begin <text>
    }
    end <text>
end
```

LL Grammar

Grammar definition :

$$< pgm > ::= < stmts > \$$$$< stmts > ::= \text{begin} < txt > \text{end}$$$$< txt > ::= < letter >< txt > | < stmts >< txt > |\{< txt >\} < txt > | \epsilon$$$$< letter > ::= a|...|z$$

Predicitve parsing :

NT symbol	[a-z]	{	}	begin	end	$	ϵ
$< pgm >$				$< stmts >$			
$< stmts >$				begin $< txt >$ end			
$< txt >$	$< letter >< txt >$	$\{< txt >\}$	ϵ	$< stmts >< txt >$	ϵ		
$< letter >$	ϵ						

Implementation

- Implemented in C++
  - $Token$ class to represent tokens
  - $ParsingTable$ class to link token and their predicted successors
  - $Buffer$ class to manage efficient reading of input string
  - $Interpreter$ class to take actions when rules are recognized
  - $SyntacticAnalyser$ class to generate token from the $Buffer$ class
- see https://github.com/hlefebvr/unige-top-down-parser

1. Top-down parser : begin...end

2. Bottom-up parser : NLP with multiple clauses

Introduction

Problem statement

- Parse in-english input strings
- Decompose multiple-clause sentences into several single-clause sentences

Worked example

The boy [who is tall, skinny and strange] smiles and laughs.

- The boy is tall
- The boy is skinny
- The boy is strange
- The boy smiles
- The boy laughs

LALR Grammar

$$
\begin{aligned}
< whole\_sentence > &::= < sentence > . \\
< sentence > &::= < np? > < vp > \\
< np > &::= < ng > < rel\_clause? > \\
< ng > &::= DET?\ CARD?\ ORD?\ QUANT?\ < ap > ?NOUN \\
&\quad | < np > COORD < np > \\
< rel\_clause > &::= REL\_WORD < sentence > \\
< vp > &::= < single\_vp > | < add\_vp > \\
< single\_vp > &::= < conjugated\_verb > < np? > \\
&\quad | < conjugated\_verb > < ap? > \\
< add\_vp > &::= < vp > COORD < vp > \\
< conjugated\_verb > &::= NEGATION?\ CONJUGATED\_BE \\
&\quad | AUXILIARY\ NEGATION?\ VERB? \\
&\quad | VERB \\
< ap > &::= ADJ | < ap > < ap > | < ap > COORD < ap >
\end{aligned}
$$

An external dictionary defines COORD, NOUN, ADJ, NEGATION, CONJUGATED_BE, VERB, ... which are token returned by the lexer

Implementation

- Python 3.0
- Lark parsing library (https://github.com/lark-parser/lark)
  - with a custom lexer making use of custom dictionary
- Idea : transform the parsing tree into a simpler tree where, given one level of exploration (from bottom to up), we end up treating structures like $< np >< rel\_clause? >< vp >$
- see https://github.com/hlefebvr/nlp-multiple-clauses-to-single

Results

he can speak italian and english.

- he can speak italian.
- he can speak english.

Results

the boy and the cat who is tall, skinny and strange smile and laugh.

- the cat is tall.
- the cat is skinny.
- the cat is strange.
- the boy smile.
- the boy laugh.
- the cat smile.
- the cat laugh.

Results

the dog eat a man who is tall.

- a man is tall.
- the dog eat a man.

Results

the boy whom I met yesterday cried and laughed.

- I met yesterday the boy
- the boy cried.
- the boy laughed.

Limitations

- Using a context free grammar enforces us to make choices.
  Example : "this" recognized as noun or determinant ?
  - this dog is eating. ("this" as determinant)
  - this is strange. ("this" as noun)