Tutorial 1

Computer Language Processing (COMP 4901U)

Monday September 13

- Recap on lexical analysis
- Exercise solving in Zoom breakout rooms

Overview

Today we will have a deeper look at tokenization.

What does a tokenizer do?

How can we automatically generate tokenizers?

Overview

Today we will have a deeper look at tokenization.

What does a tokenizer do?
$\Rightarrow$ It transforms a stream of symbols into a stream of tokens.

How can we automatically generate tokenizers?

Overview

Today we will have a deeper look at tokenization.

What does a tokenizer do?
$\Rightarrow$ It transforms a stream of symbols into a stream of tokens.

How can we automatically generate tokenizers?
$\Rightarrow$ We will use regular languages and automata.

Tokenizer

First, let us define a tokenizer as an ordered set of token names and regular expressions

$$\langle \, Token_1 := e_1, \;\; Token_2 := e_2, \;\; \ldots \, \rangle$$

where earlier token classes have higher priority than later ones.

E.g.

$$\langle ID := \text{letter (letter} \mid \text{digit)}^*, \;\; LE := \texttt{<=}, \;\; LT := \texttt{<}, \;\; EQ := \texttt{=} \rangle$$

Recap: Ambiguity in tokenization

Recall that tokenization differs from matching using a single regular expression (say $(e_1 \mid e_2 \mid \ldots)^*$).

Rather, the result of tokenizing an input stream of symbols is a stream of tokens. Each token maps to a subsequence of the input stream, and none of the tokens' subsequences overlap.

E.g.

$$\texttt{i0 <= size} \quad \overset{\textit{tokenize}}{\Rightarrow} \quad \underset{\texttt{i0}}{\mathit{ID}}\ \underset{\texttt{<=}}{\mathit{LE}}\ \underset{\texttt{size}}{\mathit{ID}}$$

Recap: Ambiguity in tokenization

Recall that tokenization differs from matching using a single regular expression (say $(e_1 \mid e_2 \mid \dots)^*$).

Rather, the result of tokenizing an input stream of symbols is a stream of tokens. Each token maps to a subsequence of the input stream, and none of the tokens' subsequences overlap.

E.g.

$$\texttt{i0 <= size} \quad \overset{\textit{tokenize}}{\Rightarrow} \quad \underset{\texttt{i0}}{\mathit{ID}}\ \underset{\texttt{<=}}{\mathit{LE}}\ \underset{\texttt{size}}{\mathit{ID}}$$

▷ How do we avoid ambiguities?

$$\texttt{i0 <= size} \quad \overset{???}{\Rightarrow} \quad \underset{\texttt{i0}}{\mathit{ID}}\ \underset{\texttt{<}}{\mathit{LT}}\ \underset{\texttt{=}}{\mathit{EQ}}\ \underset{\texttt{s}}{\mathit{ID}}\ \underset{\texttt{iz}}{\mathit{ID}}\ \underset{\texttt{e}}{\mathit{ID}}$$

Tokenization rules

Given an input string w the tokenizer will match tokens on a prefix u of $w = uv$, output the matching token and repeat the process on the remaining string v.

To disambiguate between different possible tokenizations we employ two additional rules:

- *Longest match*: If we find matching tokens for prefixes of varying lengths, we pick the longer prefix.
- *Token priority*: If multiple tokens match a prefix of the same length, we pick the token that has higher priority.

A simple tokenization example

Exercise 1

- Given the tokenizer

$$\langle T_1 := a(ab)^*,\ \ T_2 := b^*(ac)^*,\ \ T_3 := cba,\ \ T_4 := c^+ \rangle$$

tokenize the following input strings:

`c a c c a b a c a c c b a b c`

A simple tokenization example

Exercise 1

- Given the tokenizer

$$\langle T_1 := a(ab)^*,\ T_2 := b^*(ac)^*,\ T_3 := cba,\ T_4 := c^+ \rangle$$

tokenize the following input strings:

```
c a c c a b a c a c c b a b c
```

```
c c c a a b a b a c c b a b c c b a b a c
```

A simple tokenization example

Exercise 1

- Given the tokenizer

$$\langle T_1 := a(ab)^*,\ \ T_2 := b^*(ac)^*,\ \ T_3 := cba,\ \ T_4 := c^+\rangle$$

tokenize the following input strings:

$$\underbrace{\mathtt{c}}_{T_4}\ \underbrace{\mathtt{a\ c}}_{T_2}\ \underbrace{\mathtt{c}}_{T_4}\ \underbrace{\mathtt{a}}_{T_1}\ \underbrace{\mathtt{b\ a\ c\ a\ c}}_{T_2}\ \underbrace{\mathtt{c\ b\ a}}_{T_3}\ \underbrace{\mathtt{b}}_{T_2}\ \underbrace{\mathtt{c}}_{T_4}$$

`c c c a a b a b a c c b a b c c b a b a c`

A simple tokenization example

Exercise 1

- Given the tokenizer

$$\langle T_1 := a(ab)^*,\ \ T_2 := b^*(ac)^*,\ \ T_3 := cba,\ \ T_4 := c^+ \rangle$$

tokenize the following input strings:

$$\underbrace{\texttt{c}}_{T_4}\ \underbrace{\texttt{a c}}_{T_2}\ \underbrace{\texttt{c}}_{T_4}\ \underbrace{\texttt{a}}_{T_1}\ \underbrace{\texttt{b a c a c}}_{T_2}\ \underbrace{\texttt{c b a}}_{T_3}\ \underbrace{\texttt{b}}_{T_2}\ \underbrace{\texttt{c}}_{T_4}$$

$$\underbrace{\texttt{c c c}}\ \underbrace{\texttt{a a b a b}}\ \underbrace{\texttt{a c}}\ \underbrace{\texttt{c b a}}\ \underbrace{\texttt{b}}\ \underbrace{\texttt{c c}}\ \underbrace{\texttt{b}}\ \underbrace{\texttt{a}}\ \underbrace{\texttt{b a c}}$$

A simple tokenization example

Exercise 1

- ▷ Given the tokenizer

$$\langle T_1 := a(ab)^*,\ T_2 := b^*(ac)^*,\ T_3 := cba,\ T_4 := c^+ \rangle$$

tokenize the following input strings:

$$\underbrace{\mathtt{c}}_{T_4}\ \underbrace{\mathtt{a\ c}}_{T_2}\ \underbrace{\mathtt{c}}_{T_4}\ \underbrace{\mathtt{a}}_{T_1}\ \underbrace{\mathtt{b\ a\ c\ a\ c}}_{T_2}\ \underbrace{\mathtt{c\ b\ a}}_{T_3}\ \underbrace{\mathtt{b}}_{T_2}\ \underbrace{\mathtt{c}}_{T_4}$$

$$\underbrace{\mathtt{c\ c\ c}}\ \underbrace{\mathtt{a\ a\ b\ a\ b}}\ \underbrace{\mathtt{a\ c}}\ \underbrace{\mathtt{c\ b\ a}}\ \underbrace{\mathtt{b}}\ \underbrace{\mathtt{c\ c}}\ \underbrace{\mathtt{b}}\ \underbrace{\mathtt{a}}\ \underbrace{\mathtt{b\ a\ c}}$$

- ▷ Are there alternative tokenizations if we disregard the longest match rule?