FREE eBook

LEARNING parsing

Free unaffiliated eBook created from
Stack Overflow contributors.

#parsing

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: parsing

It is an unofficial and free parsing ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official parsing.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with parsing

Remarks

Parsing, in common usage, refers to analysing a piece of language, such as a sentence, and using the grammar rules of that language to identify the components pieces and thus learn the meaning. In computer science it refers to a specific algorithmic process of recognising the sequence of symbols as a valid one, and permit the meaning (or *semantics*) of a language construct to be determined, often in a computer language compiler or interpreter.

Examples

What you need for parsing

In performing parsing, before starting, the grammar for the language needs to be specified. A source of tokens is also needed for the parser.

The parser could be *hand-written* code, or a parser generator tool could be used. If a parser generator tool is used, then that tool will need to be downloaded and installed if it has not already been included in your platform.

Grammar definitions

A grammar for a parser would normally need to be written in a context free form. A notation like BNF (Backus-Naur Form) or EBNF (Extended Back-Naur Form) is often used for this. Other notations commonly used to describe programming languages might be railroad diagrams.

Lexical Analysis

Tokens are normally provided for the parser by a lexical analyser (or scanner). More details can be found in the documentation for a lexical analyser (TBC).

Parsing Techniques

To hand-code a parser, an appropriate algorithm would need to be chosen that suits both the language been parsed and the means of implementation. Parsing algorithms are classified into the two types of top-down parsing and bottom-up parsing. A (recursive) top-down parser is easier for a beginner to learn when starting to write parsers.

Parser Generator Tools

The most common way of creating a parser is to use a parser generator tool. There are many such tools, but some of the most commonly used are:

- Bison/yacc
- ANTLR

Example of Parsing an English sentence

For example, in the sentence:

> That cake is extremely nice.

The rules of the English language would make **cake** a *noun*, **extremely** an *adverb* that modifies the *adjective* **nice**, and through this analysis the meaning could be understood.

However, this analysis is dependent on us recognising that the sequence of symbols used are words. If the characters used were not familiar to us we would not be able to do this. If we encountered a sentence using an unfamiliar notation, such as Chinese, parsing in this manner might be difficult. Here is an example Chinese sentence:

> 。

For anyone who does not read Chinese characters, it would not be clear which symbols combined to form words. The same could be true for a computer algorithm when processing either English or Chinese.

Thus parsing must be proceeded by a process known as *lexical analysis* or *scanning*, where the individual characters are grouped together into recognised symbols, which we might commonly call words, but in parsing algorithms are called **tokens**.

A simple parser

The simplest way to write a parser is to use the recursive descent technique. This creates a top-down parser (which may formally be described a LL(1)). To start the example we first have to establish the grammar rules for our language. In this example we will use simple arithmetic expression assignments for expressions that can only use the plus operator:

```
Assignment --> Identifier := Expression
Expression --> Expression + Term | Term
Term --> Identifier | (Expression)
Identifier --> x | y | z
```

For each rule of the grammar we can write a procedure to recognise the incoming tokens to the rule. For the purposes of this example we can assume a routine called `NextToken` which invokes a lexical analyser to supply the token, and a routine called `error` which is used to output an error message. The language used is based on Pascal.

```
var token:char;  (* Updated by NextToken *)

procedure identifier;
begin
   if token in ['x','y','z']
   then
```

```
        NextToken
    else
        error('Identifier expected')
end;
```

You can see that this code implements the rule for recognising an `Identifier`. We can then implement the rule for a `Term` similarly:

```
procedure expression forward;

procedure term;
begin
    if token = '('
    then
        begin
        nextToken;
        expression;
        if token <> ')'
        then
            error(') expected')
        else NextToken
        end
    else
        identifier
end;
```

When we come to implement the rule for `Expression` we have a problem; the first element of the `Expression` rule is itself an `Expression`. This would cause us to write the following:

```
procedure expression;
begin
expression;
...
end;
```

This is directly self-recursive and thus would loop forever. Grammar parsed by top-down algorithms cannot be left-recursive for this reason. An easy way out of this problem is to recast the recursion as iteration in the following way:

```
Expression --> Term { + Term}*
```

We can now code up the grammar rule as:

```
procedure expression;
begin
    term;
    while token = '+'
        do
        begin
            NextTerm;
            term
        end
end;
```

We can now complete the parser with the remaining rule for the assignment:

```
procedure assignment;
begin
    identifier;
    if token <> '='
    then
        error('= expected')
    else
        begin
        NextToken;
        expression;
        end
 end;
```

Read Getting started with parsing online: https://riptutorial.com/parsing/topic/4370/getting-started-with-parsing

Credits

S. No	Chapters	Contributors
1	Getting started with parsing	Brian Tompsett - , Community, ShengJie Zhou