eBook Gratuit

APPRENEZ parsing

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#parsing

Table des matières

À propos 1

Chapitre 1: Commencer à analyser 2

Remarques 2

Examples 2

Ce dont vous avez besoin pour analyser 2

Définitions grammaticales 2

Analyse lexicale 2

Techniques d'analyse 2

Outils de générateur d'analyseur 2

Exemple d'analyse syntaxique d'une phrase anglaise 3

Un simple analyseur 3

Crédits 6

À propos

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

Chapitre 1: Commencer à analyser

Remarques

L'analyse syntaxique, dans l'usage courant, fait référence à l'analyse d'un morceau de langage, tel qu'une phrase, et à l'utilisation des règles de grammaire de ce langage pour identifier les composants et apprendre ainsi la signification. En informatique, il s'agit d'un processus algorithmique spécifique consistant à reconnaître la séquence de symboles comme un processus valide et à déterminer la signification (ou la *sémantique*) d'une construction de langage, souvent dans un compilateur ou un interpréteur de langage informatique.

Examples

Ce dont vous avez besoin pour analyser

En effectuant l'analyse, avant de commencer, la grammaire de la langue doit être spécifiée. Une source de jetons est également nécessaire pour l'analyseur.

L'analyseur peut être *un* code *écrit à la main* ou un outil générateur d'analyseur peut être utilisé. Si un outil générateur d'analyseurs est utilisé, alors cet outil devra être téléchargé et installé s'il n'a pas déjà été inclus dans votre plate-forme.

Définitions grammaticales

Une grammaire pour un analyseur devrait normalement être écrite sous une forme sans contexte . Une notation comme BNF (Backus-Naur Form) ou EBNF (Extended Back-Naur Form) est souvent utilisée pour cela. D'autres notations couramment utilisées pour décrire les langages de programmation peuvent être des schémas de chemin de fer .

Analyse lexicale

Les jetons sont normalement fournis à l'analyseur par un analyseur lexical (ou scanner) . Plus de détails peuvent être trouvés dans la documentation d'un analyseur lexical (TBC).

Techniques d'analyse

Pour coder manuellement un analyseur, il faudrait choisir un algorithme adapté à la langue analysée et aux moyens d'implémentation. Les algorithmes d'analyse sont classés en deux types d' analyse syntaxique descendante et d' analyse ascendante . Un analyseur descendant (récursif) est plus facile à apprendre pour un débutant lorsqu'il commence à écrire des analyseurs.

Outils de générateur d'analyseur

La manière la plus courante de créer un analyseur syntaxique consiste à utiliser un outil générateur d'analyseurs. Il existe de nombreux outils de ce type, mais les plus utilisés sont:

- Bison / Yacc
- ANTLR

Exemple d'analyse syntaxique d'une phrase anglaise

Par exemple, dans la phrase:

> Ce gâteau est extrêmement gentil.

Les règles de la langue anglaise feraient du **gâteau** un *nom* , **extrêmement** un *adverbe* qui modifie l' *adjectif* **gentil** , et grâce à cette analyse, le sens pourrait être compris.

Cependant, cette analyse nous oblige à reconnaître que la séquence de symboles utilisée est des mots. Si les caractères utilisés ne nous étaient pas familiers, nous ne pourrions pas le faire. Si nous rencontrions une phrase utilisant une notation inconnue, telle que le chinois, l'analyse de cette manière pourrait être difficile. Voici un exemple de phrase chinoise:

> 。

Pour ceux qui ne lisent pas les caractères chinois, les symboles combinés pour former des mots ne sont pas clairs. La même chose pourrait être vraie pour un algorithme informatique lors du traitement de l'anglais ou du chinois.

Ainsi, l'analyse doit être effectuée par un processus appelé *analyse lexicale* ou *analyse* , où les caractères individuels sont regroupés dans des symboles reconnus, que nous pourrions appeler couramment des mots, mais dans les algorithmes d'analyse syntaxique sont appelés des **jetons** .

Un simple analyseur

La manière la plus simple d'écrire un analyseur est d'utiliser la technique de descente récursive. Cela crée un analyseur top-down (qui peut être décrit comme un LL (1)). Pour commencer l'exemple, nous devons d'abord établir les règles de grammaire pour notre langue. Dans cet exemple, nous utiliserons des affectations d'expression arithmétique simples pour les expressions qui ne peuvent utiliser que l'opérateur plus:

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

Pour chaque règle de la grammaire, nous pouvons écrire une procédure pour reconnaître les jetons entrants à la règle. Pour les besoins de cet exemple, nous pouvons supposer une routine appelée `NextToken` qui appelle un analyseur lexical pour fournir le jeton, et une routine appelée `error` qui est utilisée pour générer un message d'erreur. Le langage utilisé est basé sur Pascal.

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

```
procedure identifier;
begin
   if token in ['x','y','z']
   then
       NextToken
   else
       error('Identifier expected')
end;
```

Vous pouvez voir que ce code implémente la règle de reconnaissance d'un `Identifier` . Nous pouvons alors implémenter la règle pour un `Term` même manière:

```
procedure expression forward;

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

Lorsque nous arrivons à mettre en œuvre la règle pour l' `Expression` nous avons un problème. le premier élément de la règle d' `Expression` est lui-même une `Expression` . Cela nous amènerait à écrire ce qui suit:

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

Ceci est directement auto-récursif et boucle donc pour toujours. La grammaire analysée par des algorithmes descendants ne peut pas être récursive pour cette raison. Une solution simple à ce problème consiste à refondre la récursivité en itération de la manière suivante:

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

Nous pouvons maintenant coder la règle de grammaire comme suit:

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

```
        begin
            NextTerm;
            term
        end
 end;
```

Nous pouvons maintenant compléter l'analyseur avec la règle restante pour l'affectation:

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

Lire Commencer à analyser en ligne: https://riptutorial.com/fr/parsing/topic/4370/commencer-a-analyser

Crédits

S. No	Chapitres	Contributeurs
1	Commencer à analyser	Brian Tompsett - , Community, ShengJie Zhou