EBook Gratis

APRENDIZAJE parsing

Free unaffiliated eBook created from
Stack Overflow contributors.

#parsing

Tabla de contenido

Acerca de

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

Capítulo 1: Comenzando con el análisis

Observaciones

El análisis, en el uso común, se refiere al análisis de una parte del lenguaje, como una oración, y el uso de las reglas gramaticales de esa lengua para identificar las partes componentes y así aprender el significado. En ciencias de la computación se refiere a un proceso algorítmico específico de reconocimiento de la secuencia de símbolos como válido, y permite determinar el significado (o *semántica*) de una construcción de lenguaje, a menudo en un compilador o intérprete de lenguaje de computadora.

Examples

Lo que necesitas para analizar

Al realizar el análisis, antes de comenzar, se debe especificar la gramática del idioma. También se necesita una fuente de tokens para el analizador.

El analizador podría ser *un* código *escrito a mano* , o podría usarse una herramienta de generador de analizador . Si se utiliza una herramienta de generador de analizador, entonces esa herramienta deberá descargarse e instalarse si aún no se ha incluido en su plataforma.

Definiciones gramaticales

Una gramática para un analizador normalmente debería escribirse en una forma libre de contexto . A menudo se usa una notación como BNF (Backus-Naur Form) o EBNF (Extended Back-Naur Form) para esto. Otras notaciones comúnmente utilizadas para describir lenguajes de programación pueden ser diagramas de ferrocarril .

Análisis léxico

Normalmente, los indicadores se proporcionan para el analizador mediante un analizador léxico (o escáner) . Se pueden encontrar más detalles en la documentación de un analizador léxico (TBC).

Técnicas de análisis

Para codificar manualmente un analizador, se debería elegir un algoritmo adecuado que se adapte tanto al lenguaje analizado como a los medios de implementación. Algoritmos de análisis sintáctico se clasifican en los dos tipos de análisis de arriba hacia abajo y de análisis de abajo hacia arriba . Un analizador de arriba a abajo (recursivo) es más fácil de aprender para los principiantes cuando comienzan a escribir analizadores.

Herramientas de generador de analizador

La forma más común de crear un analizador es utilizar una herramienta de generador de analizador. Existen muchas de estas herramientas, pero algunas de las más utilizadas son:

- Bisonte / yacc
- ANTLR

Ejemplo de analizar una oración en inglés

Por ejemplo, en la oración:

> Ese pastel es extremadamente bonito.

Las reglas del idioma Inglés harían **pastel de** un *sustantivo,* **extremadamente** un *adverbio* que modifica el **buen** *adjetivo,* ya través de este análisis, el significado podría ser entendido.

Sin embargo, este análisis depende de que reconozcamos que la secuencia de los símbolos utilizados son palabras. Si los caracteres utilizados no nos resultaran familiares, no podríamos hacerlo. Si encontramos una oración utilizando una notación desconocida, como el chino, el análisis de esta manera puede ser difícil. Aquí hay un ejemplo de oración china:

> 。

Para cualquiera que no lea caracteres chinos, no estaría claro qué símbolos se combinaron para formar palabras. Lo mismo podría ser cierto para un algoritmo de computadora cuando se procesa inglés o chino.

Por lo tanto, el análisis debe realizarse mediante un proceso conocido como *análisis* o *exploración léxica* , donde los caracteres individuales se agrupan en símbolos reconocidos, que comúnmente podríamos llamar palabras, pero en el caso de los algoritmos de análisis se denominan **tokens** .

Un simple analizador

La forma más sencilla de escribir un analizador es utilizar la técnica de descenso recursivo. Esto crea un analizador descendente (que puede describirse formalmente como LL (1)). Para comenzar el ejemplo primero tenemos que establecer las reglas gramaticales para nuestro idioma. En este ejemplo, usaremos asignaciones de expresiones aritméticas simples para expresiones que solo pueden usar el operador más:

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

Para cada regla de la gramática podemos escribir un procedimiento para reconocer los tokens entrantes a la regla. Para los propósitos de este ejemplo, podemos asumir una rutina llamada `NextToken` que invoca un analizador léxico para suministrar el token, y una rutina llamada `error` que se usa para generar un mensaje de error. El lenguaje utilizado está basado en Pascal.

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

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

Puede ver que este código implementa la regla para reconocer un `Identifier` . Entonces podemos implementar la regla para un `Term` manera similar:

```
procedure expression forward;

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

Cuando llegamos a implementar la regla de `Expression` tenemos un problema; El primer elemento de la regla de `Expression` es en sí mismo una `Expression` . Esto nos haría escribir lo siguiente:

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

Esto es directamente auto-recursivo y por lo tanto sería un bucle para siempre. La gramática analizada por algoritmos de arriba hacia abajo no puede ser recursiva por la izquierda por esta razón. Una manera fácil de salir de este problema es volver a redactar la recursión como iteración de la siguiente manera:

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

Ahora podemos codificar la regla gramatical como:

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

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

Ahora podemos completar el analizador con la regla restante para la tarea:

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

Lea Comenzando con el análisis en línea:
https://riptutorial.com/es/parsing/topic/4370/comenzando-con-el-analisis

Creditos

S. No	Capítulos	Contributors
1	Comenzando con el análisis	Brian Tompsett - , Community, ShengJie Zhou