Un intérprete de regex

Horst H. von Brand
vonbrand@inf.utfsm.cl

Departamento de Informática
Universidad Técnica Federico Santa María

Contenido

Historia de regex

En los '60, Ken Thompson introdujo búsqueda de expresiones regulares en el editor QED. Luego fueron centrales en el editor ed(1), y en la herramienta grep(1) de Unix.
En 1998 Brian Kernighan y Rob Pike estaban escribiendo el libro "*The Practice of Programming*" (Addison-Wesley, 1999), que incluye un capítulo sobre notaciones y cómo ayudan a crear programas simples, elegantes y eficientes. Una notación importante es regex. Discutir alguna de las versiones disponibles era imposible (programas grandes y muy complejos). Brian Kernighan recuerda que Rob se encerró en su oficina, volviendo a la hora con 30 líneas de C, publicadas en el libro, que discutiremos acá.

Operaciones soportadas

Las construcciones que maneja el programa son:

- `c` Calza el caracter literal *c*
- `.` Calza un caracter cualquiera
- `^` Calza al principio de la línea
- `$` Calza al final de la línea
- `*` Calza cero o más veces el caracter inmediatamente precedente

Estas son familiares de las herramientas Unix, y son algo de un 95% de las construcciones en el uso diario. Es un conjunto mínimo, ya muy útil.

Sobre el código

El código de Pike es un excelente ejemplo de recursión, ilustra en forma brillante el uso de punteros en C, es compacto, elegante y útil.
Presentamos una versión algo remozada para usanza actual de C del programa que describe Kernighan.

El programa

Encabezado

El encabezado match.h declara la función:

```
/* match: search for regexp anywhere in text */
bool match(char *regexp, char *text);
```

El programa

Declaraciones

En match.c se declaran funciones auxiliares:

```
static bool matchhere(char *regexp, char *text);
static bool matchstar(int c, char *regexp, char *text);
```

El programa

La función match

```
/* match: search for regexp anywhere in text */
bool match(char *regexp, char *text)
{
    if(regexp[0] == '^')
        return matchhere(regexp + 1, text);
    do { /* must look even if string is empty */
        if(matchhere(regexp, text))
            return true;
    } while(*text++ != '\0');
    return false;
}
```

El programa

La función matchhere

```
/* matchhere: search for regexp at beginning of text */
static bool matchhere(char *regexp, char *text)
{
    if(regexp[0] == '\0')
        return true;
    if(regexp[1] == '*')
        return matchstar(regexp[0], regexp + 2, text);
    if(regexp[0] == '$' && regexp[1] == '\0')
        return *text == '\0';
    if(*text != '\0'
       && (regexp[0] == '.' || regexp[0] == *text))
        return matchhere(regexp + 1, text + 1);
    return false;
}
```

El programa

La función matchstar

```
/* matchstar: search for c*regexp at beginning of text *
static bool matchstar(int c, char *regexp, char *text)
{
    do { /* a * matches zero or more instances */
        if(matchhere(regexp, text))
            return true;
    } while(*text != '\0'
            && (*text++ == c || c == '.'));
    return false;
}
```

Resumen

- Una muestra de cómo interpretar regex directamente. Kernighan plantea varias posibles extensiones. Quedan de ejercicio para los interesados.
- Junto a la estrategia esbozada de crear un NFA e interpretarlo directamente, y la opción de ir desde regex a un DFA para interpretar éste, son las maneras en que se llevan a la práctica.