Costrutto IF tramite Regular Expression

mobytrick

6 settembre 2014

Mi sono scontrato con un problema tutto sommato abbastanza banale. Costruire una *Regular Expression* (d'ora in avanti indicata con RE) capace di individuare indirizzi IP scritti in maniera erronea. Come è noto questi sono composti da 4 ottetti separati dal carattere punto. Ciascun ottetto non può assumere un valore superiore a 255. Il primo e l'ultimo, se non vado errato, devono essere superiori allo zero. Per cui lo strumento di controllo dovrebbe contenere un costrutto del tipo IF che nel campo delle RE non esiste. Per venirne fuori non resta che aguzzare l'ingegno. La RE per descrivere gli indirizzi IP è del tipo:

```
OCT\.ZEROCT\.ZEROCT\.OCT
```

ove con OCT indico gli ottetti il cui valore è compreso tra 1 e 255 mentre con ZEROCT indico quelli il cui valore è compreso tra 0 e 255. Per completezza, la RE di sopra può anche essere scritta più concisamente:

```
OCT(\.ZEROCT){2}\.OCT
```

Il problema sta dunque nel descrivere, col formalismo delle RE, OCT e ZEROCT in maniera opportuna. Prendiamo il caso di OCT. Può essere composto da 1, 2 oppure 3 cifre. Nell'ultimo caso, per limitare il valore massimo a 255 è sufficiente **descrivere in termini di RE tutti i numeri tra 100 e 255**. In tal modo, anche se surrettiziamente, si introduce un meccanismo rozzo ma efficace di IF. Ma procediamo con ordine. Nel caso caso di 1 cifra, poiché questa non può essere nulla, la RE che fa al caso nostro è:

```
[1-9]
```

Nel caso di 2 cifre, si tratta di descrivere tutti i numeri tra 10 e 99. La RE che fa al caso è:

```
[1-9][0-9]
```

Le 2 RE precedenti possono essere fuse in un'unica:

```
[1-9][0-9]?
```

Nel caso di tre cifre, si tratta di descrivere i numeri tra 100 e 255. Dapprima si descrivono quelli tra 100 e 199 tramite:

```
1[0-9]{2}
```

poi quelli tra 200 e 249 tramite:

```
2[0-4][0-9]
```

ed infine quelli tra 250 e 255 tramite:

```
25[0-5]
```

La RE che descrive OCT si ottiene legando con il metacarattere di alternanza `|` le ultime 4. Quindi:

```
[1-9][0-9]?|1[0-9]{2}|2[0-4][0-9]|25[0-5]
```

Nel caso di ZEROCT il ragionamento è in parte analogo ed in parte uguale. La descrizione degli ottetti di 3 cifre rimane la medesima. Nel caso di 1 o 2 cifre si tratta di descrivere i numeri tra 0 e 99. La RE per quelli tra 0 e 9 è:

```
[0-9]
```

mentre quella per i numeri tra 10 e 99 è:

```
[1-9][0-9]
```

Anche in questo caso è possibile una fusione:

```
[1-9]?[0-9]
```

Quindi ZEROCT si ottiene nel solito modo, legando le opportune RE con il metacarattere di alternanza `|`:

```
[1-9]?[0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5]
```

Si badi bene che OCT e ZEROCT descrivono solo 1 ottetto. Le rispettive RE vanno poste in quella che descrive l'intero indirizzo IP. Invece del solito taglia ed incolla si può utilizzare uno strumento Unix a dire il vero un po' negletto: flex. Che però ha il pregio di essere conciso e di operare per noi l'editing. flex non è immediatamente operativo in quanto produce un programma C che va compilato. Il doppio passo è ampiamente ripagato dalla velocità di esecuzione. Per come lo script è stato costruito, se all'eseguibile si passa un indirizzo IP valido, questo verrà ristampato sull'output. Se invece si passa un'indirizzo IP sbagliato, sull'output non verrà restituito alcunché. Lo script però potrebbe essere usato per estrarre gli indirizzi IP presenti ad esempio sui log di Apache. Messo in pipe con un script awk, potrebbe costituire un rudimentale contatore.

Vediamo lo script flex ove quelli che sembrano spazi intermedi in realtà sono **tabulatori**, obbligatori e non usati per rendere più leggibile il testo:

```
TREDGT  1[0-9]{2}|2[0-4][0-9]|25[0-5]
ZEROCT  [1-9]?[0-9]|{TREDGT}
OCT     [1-9][0-9]?|{TREDGT}
%%
{OCT}(\.{ZEROCT}){2}\.{OCT}     printf ("%s\n", yytext);
.|\n                            ;
```

Semplificando di molto la questione, flex lavora in questo modo: si scrivono delle RE e delle azioni che devono essere intraprese. flex legge da standard input e scorre le RE nell'ordine fisico in cui sono state scritte. Non appena una ha trovato qualcosa di conforme vengono eseguite le azioni previste e si passa a leggere nuovamente da input. Tutto il materiale che sfugge alle RE viene stampato su standard output. Le azioni sono scritte nel cosiddetto linguaggio *ospite* che nella maggior parte dei casi è il C.
Nel nostro caso le prime tre righe sono delle definizioni: a sinistra c'è il simbolo e a destra il contenuto vero e proprio. flex opera una specie di editing. Dalla riga successiva alla definizione tutte le occorrenze del simbolo verranno sostituite dal contenuto. Già dall'inizio si può apprezzare la chiarezza e la concisione dello strumento.
Lo script vero e proprio inizia la riga successiva a quella costituita dal doppio simbolo di percento (`%`). A sinistra ci sono le RE e nella parte destra le azioni relative. Per come lo script è stato architettato, la prima RE descrive un indirizzo IP. Se l'input è conforme allora viene stampato (in flex i caratteri intercettati dalle RE vengono messi in una stringa il cui nome è `yytext`). I

newline ed i caratteri non conformi ad alcuna RE per default sarebbero stampati su output. Con la successiva RE vengono invece intercettati e scartati perché ad essi come azione è applicato lo statement null (`;`) del C.
Lo script flex deve stare in un file il cui nome finisce con `.l` (lettera elle minuscola), supponiamo sia `addrip.l`. Allora per produrre il programma C il comando è:

```
flex -t addrip.l >| addrip.c
```

Cui segue la compilazione:

```
gcc -O2 -s -o addrip addrip.c -lfl
rm -f addrip.c
```

Si deve obbligatoriamente linkare la libreria flex (`-lfl`). Per mia libera scelta sono stati usati sia `-s` (strip), per cui non è possibile usare il debugger, che `-O2` (ottimizzazione standard). A compilazione avvenuta il file `addrip.c` può essere eliminato. Per la prova se tutto funziona a dovere:

```
# echo "127.0.0.1" | ./addrip
127.0.0.1
# echo "127.0.0.0" | ./addrip
```

Nel primo caso l'indirizzo fornito è regolare e viene ristampato su output. Nel secondo caso l'ultimo ottetto non è valido (non può essere zero) e quindi non c'è output.