Rice e Kleene in Python

Massimo Santini

20 maggio 2013

Sommario

Obiettivo di questi appunti è offrire una dimostrazione informale dei teoremi di Rice e Kleene basate sull'uso del linguaggio Python, piuttosto che su formalismi quali le macchine di Turing, i sistemi di programmazione accetabili, o le funzioni ricorsive parziali.

Nozioni preliminari

In queste note restringeremo l'attenzione a *funzioni* del linguaggio Python che operano su *stringhe* nel senso che sia i loro parametri che i valori che restituiscono, nel caso la loro esecuzione termini, sono di tipo `str`; per semplificare la notazione, nel seguito useremo le lettere minuscole $F, G, \ldots$ per indicare *funzioni* mentre le lettere minuscole $f, g, \ldots$ per indicare il *codice sorgente* delle medesime. Ad esempio, se f corrisponde a:

```
def F( x ):
        return 2 * x
```

intenderemo con F la funzione che, data una stringa come parametro, restuisce la stringa ottenuta concatenando il paraetro con se stesso, ad esempio $F(\text{'ciao'}) = \text{'ciaociao'}$. Diremo che due funzioni F e G hanno lo stesso comportamento, in simboli

$$F(x) \equiv G(x)$$

se e solo se per tutti i valori dei parametro x per cui $F(x)$ termina, $G(x)$ termina anch'essa e $F(x) = G(x)$; useremo la stessa notazione anche dato il codice sorgente, ossia scriveremo

$$f \equiv g$$

per indicare che $F(x) \equiv G(x)$.

Gli ingredienti fondamentali necessari alla dimostrazione sono: la *funzione universale* U e la funzione di *currying* S; tali funzioni possono essere semplicemente implementate in Python come segue.

La funzione U

La funzione universale è una funzione che date due stringhe f ed x come parametri restituisce la stringa $F(x)$ dove F è la funzione corrispondente al sorgente f, denoteremo in simboli questa definizione con

$$U(f, x) \equiv F(x)$$

Una possibile implementazione di U è data dal seguente codice:

```
def U( f, x ):
  locals = {}
  exec( f, globals(), locals )
  F = next( iter( locals.values() ) )
  return F( x )
```

l'unico punto degno di nota è l'uso della funzione `exec` che è in grado di eseguire il codice rappresentato dall stringa f (che, di fatto, definisce la funzione), il resto dell'implementazione si occupa del dettaglio di recuperare la funzione dal dizionario `locals` e di calcolarne il valore.

La funzione S

La funzione di currying date due stringhe f ed y come parametri, dove f è il sorgente di una funzione F a due parametri, restituisce una stringa g corrispondente al sorgente di una funzione G tale che $F(x, y) = G(x)$, denoteremo in simboli questa definizione con

$$S(f, y) = g \qquad \text{tale che} \qquad U(g, x) = U(S(f, y), x) \equiv F(x, y)$$

L'implementazione della funzione di currying è ancora più elementare:

```
def S( f, y ):
  n = match( 'def\s+([^(]+)\s*\(', f ).group( 1 )
  f = f.replace( '\n', '\n\t' )
  g = 'def G( x ):\n\t{0}\n\treturn {1}( x, {2!r} )'
  return g.format( f, n, y )
```

essa "avvolge" la funzione f (dopo averne determinato il nome usando una espressione regolare ed averla indentata) definendo così la funzione G di cui restituisce il sorgente. Per comprendere il suo funzionamento, possiamo invocarla ad esempio sulla funzione f data da

```
def Somma( x, y ):
      return x + y
```

per cui avremo, ad esempio, $S(f, 3)$

```
def G( x ):
  def Somma( x, y ):
    return x + y
  return Somma( x, 3 )
```

Il teorema di Kleene

Siamo pronti per intraprendere la dimostrazione del teorema di Kleene, il cui enunciato è il seguente.

Teorema 1 *Data una qualunque fuzione T che termina per ogni valore del suo parametro è possibile costruire una funzione r per cui $T(r) \equiv r$.*

Vediamo come dimostrare il teorema in modo costruttivo, ossia ottenendo di fatto la funzione r a partire da T e dalle funzioni U ed S introdotte in precedenza. Consideriamo le funzioni e ed m definite rispettivamente come segue:

```
def E( x, f ): return U( U( f, f ), x )
```

```
def M( x ): return T( S( e, x ) )
```

Proviamo ora che $r = S(e, m)$ (nella cui definizione compare la funzione T, oltre a E, M, S ed U):

```
def R( x ):
  def E( x, f ): return U( U( f, f ), x )
  return E( x, 'def M( x ): return T( S( e, x ) )' )
```

è precisamente la funzione postulata dal teorema:

$$
\begin{aligned}
U(r,x) &= U(S(e,m),x) && \text{per definizione di } r, \\
&= E(x,m) && \text{per definizione di } S, \\
&= U(U(m,m),x) && \text{per definizione di } E, \\
&= U(M(m),x) && \text{per definizione di } U, \\
&= U(T(S(e,m)),x) && \text{per definizione di } M, \\
&= U(T(r),x) && \text{per definizione di } r.
\end{aligned}
$$

il che, per definizione di $\equiv$ ed U, corrisponde alla tesi del teorema.

Una applicazione divertente: quine

Una divertente applicazione di questo teorema è ottenere una *quine*, ossia una funzione che restitusca se stessa; con questo intendiamo una funzione F tale che $F(x)$ sia il suo sorgente f (indipendentemente dal parametro x). Osserviamo che questa non è a priori cosa banale. Ad esempio, la funzione

```
def F( x ):
  return 'def F( x )\n\t return x'
```

è tale che $F(x)$ è pari a

```
def F( x ):
  return x
```

che è simile al suo sorgente, ma non identico. Osserviamo che è viceversa banale costruire una funzione T (che termina sempre) la quale, avendo per argomento un sorgentge f, dia una funzione g tale che $G(x)$ restituisca f:

```
def T( f ):
  return 'def F( x ):\n\treturn {0!r}'.format( f )
```

Grazie al teorema di Kleene, è possibile costruire r il cui comportamento coincide con quello di $T(r)$ che è proprio restituire r; detto altrimenti: $R(x)$ restiuisce sempre r.

Il teorema di Rice

Siamo arrivati al punto di queste note. Diremo che una collezione $\mathcal{F}$ di funzioni Python *rispetta le funzioni* qualora

$$f \in \mathcal{F} \quad \text{e} \quad f \equiv g \qquad \text{implica che} \qquad g \in \mathcal{F}$$

detto altrimenti $\mathcal{F}$ contiene tutte le funzioni che "si comportano allo stesso modo"; diremo inoltre che una collezione di funzioni $\mathcal{F}$ è *decidibile* se si può scrivere una fuzione Python D che termina sempre per cui $D(f)$ restituisce `si` se $f \in \mathcal{F}$, o `no` altrimenti, sono casi *banali* quelli per cui la collezione è vuota, o comprente tutte le possibili funzioni.

Teorema 2 *Se $\mathcal{F}$ rispetta le funzioni e non è banale, allora non è decidibile.*

Diamo una dimostrazione per assurdo di questo teorema basandoci su quello di Rice; consideriamo due funzioni $p \in \mathcal{F}$ e $q \notin \mathcal{F}$ che esisteranno dal momento che $\mathcal{F}$ non è banale e sia D la funzione che decide $\mathcal{F}$; consideriamo la funzione seguente funzione T (che termina per ogni valore del parametro dato che D termina per ipotesi):

```
def T( f ):
  if D( f ) == 'si':
    return q
  else:
    return p
```

Per il teorema di Kleene, esiste un r tale per cui $r \equiv T(r)$ quindi, dato che $\mathcal{F}$ rispetta le funzioni, o $r \in \mathcal{F}$ e $T(r) \in \mathcal{F}$, oppure $r \notin \mathcal{F}$ e $T(r) \notin \mathcal{F}$. Ma questo non può essere dal momento che, per definizione di T, se $r \in \mathcal{F}$ allora $D(r) =$`si` quindi $T(r) = q \notin \mathcal{F}$ e, viceversa, se $r \notin \mathcal{F}$ allora $D(r) =$`no` quindi $T(r) = p \in \mathcal{F}$.