Esistono casi in cui una funzione è calcolabile, ma non siamo in grado di conoscere il programma che la calcola. Ad esempio:

$$F(x) = \begin{cases} 1 & \text{se il problema del continuo ha soluzione positiva} \\ 0 & \text{altrimenti} \end{cases}$$

La funzione $F(x)$ coincide o con $0(x) = 0$ o con $1(x) = 1$. In entrambi i casi si tratta di funzioni calcolabili. Ma non sappiamo decidere la soluzione corretta se non risolviamo il problema del continuo.

Problema del continuo. Esiste un insieme avente cardinalità compresa fra $\aleph_0$ e $2^{\aleph_0}$.

Anche la funzione:

$$g(x) = \begin{cases} 1 & \text{se nell'espansione di } \pi \text{ esistono} \\ & \text{almeno } x \text{ 5 consecutivi,} \\ 0 & \text{altrimenti} \end{cases}$$

è sicuramente calcolabile. Infatti se esiste un numero massimo k di cinque consecutivi, allora sappiamo che $g(x)$ deve calcolare 1 per ogni $x \leq k$, 0 altrimenti. Se tale numero massimo non esiste, allora $g(x)$ restituisce sempre 1.
Problema: non sappiamo se tale k esiste, ovvero non sappiamo qual è l'algoritmo corretto che calcola g.

Diversa è la situazione nel caso della funzione:

$$g'(x) = \begin{cases} 1 & \text{se nell'espansione di } \pi \text{ esistono} \\ & \textbf{esattamente } x \text{ 5 consecutivi,} \\ 0 & \text{altrimenti} \end{cases}$$

In questo caso la conoscenza del k non ci aiuterebbe. Per determinare il valore di g' dovremmo contare il numero di 5 consecutivi. Ma non sappiamo dire a priori se ne troveremo x o meno. Se ve ne sono per ogni x allora g' è calcolabile, altrimenti no. Quindi, non sappiamo se g' è calcolabile oppure no.

Teoremi di Kleene e Rice

Le definizioni implicite sono molto comuni in matematica.

Esempio 14 *L'equazione* $3x+6 = x$ *definisce implicitamente un valore per* x*. La corrispondente definizione esplicita è* $x = -3$*.*

Questa equazione può essere anche interpretata in questo modo: $3x + 6$ è una funzione o trasformazione dagli interi agli interi, che indicheremo con $T(x)$. L'equazione $x = T(x)$ definisce quell'intero che è *punto fisso* della trasformazione $T(x)$.

Dato un insieme S e una funzione $\tau : S \longmapsto S$, si dice *punto fisso* di τ il valore $s \in S$ tale che $\tau(s) = s$.

Idea: perché non definire gli algoritmi in maniera *implicita* come i programmi invarianti di trasformazioni di programmi in programmi?

Osservazione: In realtà non sempre i valori definiti implicitamente sono univoci.

Esempio 15 *L'equazione* $x^2 - 5x + 6 = 0$ *definisce due valori per* x *(*$x = 2$ *e* $x = 3$*).*

Quindi anche nel caso di algoritmi descritti come programmi invarianti di trasformazioni di programmi, dovremmo aspettarci di non avere necessariamente definizioni univoche.

Domanda: Come devono essere scelte queste trasformazioni affinchè ammettano almeno un programma invariante, tale cioè che lui e il suo trasformato calcolino la stessa funzione?

La risposta è fornita dal teorema di Kleene, il quale afferma che se si considera come classe delle trasformazioni quella delle funzioni calcolabili totali, queste trasformazioni ammettono sempre un programma invariante, cioè per ogni trasformazione effettiva t di programmi in programmi esiste sempre un programma e tale che $t(e)$ è equivalente a e, ovvero entrambi calcolano la stessa funzione parziale.

Teorema 16 *(ricorsione o di Kleene) Sia data una enumerazione delle funzioni ricorsive. Se t è una funzione calcolabile totale, allora esiste $e \in \mathbb{N}$ tale che:*

$$\varphi_e = \varphi_{t(e)}.$$

Il teorema si dice anche del *punto fisso*.

R è l'insieme delle funzioni ricorsive.
Sia $\tau : R \mapsto R$ ricorsiva totale tale che $\tau(\varphi_i) = \varphi_{t(i)}$.
τ è un *funzionale*.
Il teorema di Kleene stabilisce che τ ha un punto fisso
$\varphi_e = \varphi_{t(e)} = \tau(\varphi_e)$

Dimostrazione. Per enumerazione delle macchine di Turing.
Consideriamo la famiglia parametrica $M[u]$ di MT che calcolano le funzioni parziali:

$$\varphi_{g(u)}(x) = \begin{cases} \varphi_{\varphi_u(u)}(x) & \text{se } \varphi_u(u) \text{ è definita} \\ indefinita & \text{altrimenti} \end{cases}$$

La funzione g definisce la corrispondenza fra ogni $M[u]$ e l'indice della funzione parziale che essa calcola.
Sia $v \in \mathbb{N}$ tale che: $\varphi_v = \lambda x.t(g(x))$. Allora, per ogni $x \in \mathbb{N}$
$\varphi_{g(v)}(x) = \varphi_{\varphi_v(v)}(x) = \varphi_{t(g(v))}(x)$
Ponendo $e = g(v)$ si ha: $\varphi_e = \varphi_{t(e)}$.
N.B: Il punto fisso potrebbe non essere unico.

Per rendere più intuitivo questo importante teorema della teoria della computabilità ne diamo una seconda dimostrazione informale.

Consideriamo un linguaggio capace di trattare istruzioni come dati.

Per dimostrare il teorema è sufficiente, dato un qualunque programma t, costruire un programma p punto fisso di t. Consideriamo il seguente programma p.

1. Si faccia una copia dell'intero programma p in un'area di memoria u.

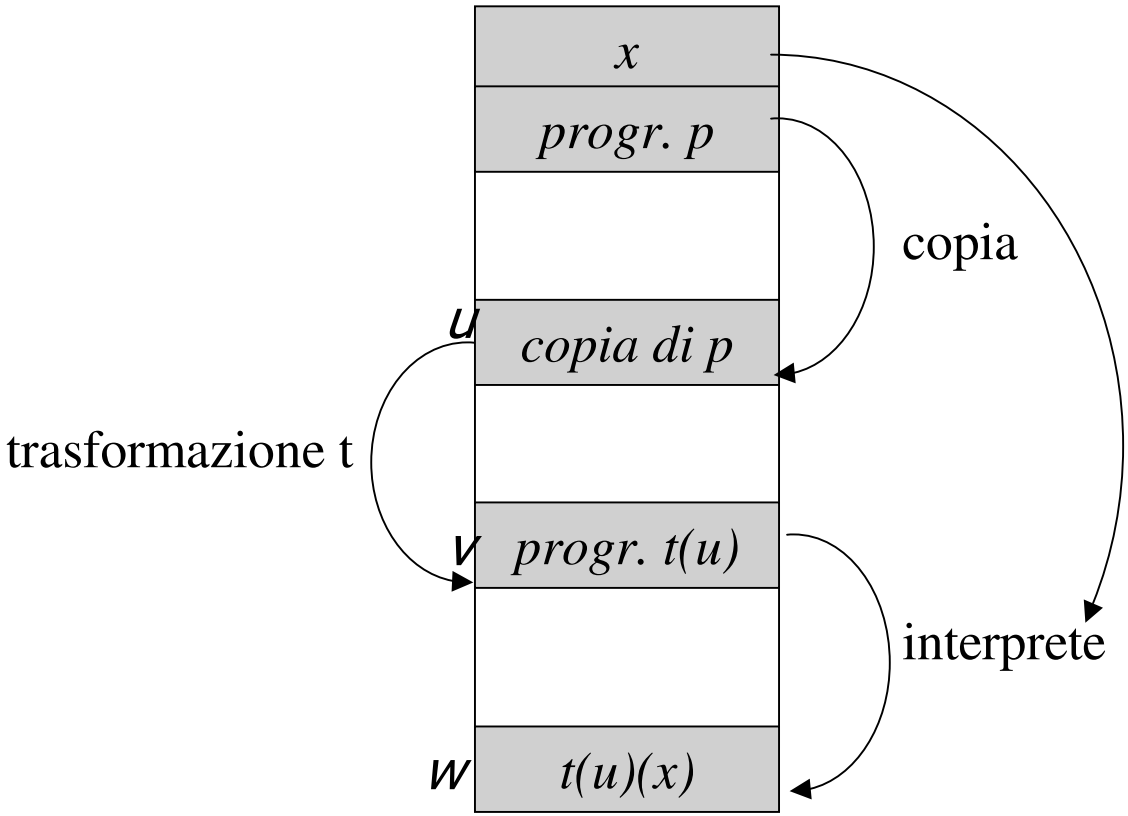
2. Si calcoli $t(u)$ e si lasci il risultato in un'altra area di lavoro v.

3. Usando un interprete, si esegua il programma in v tenendo presente che è stato rilocato.

4. Si dia come risultato dell'intero programma p il risultato dell'interpretazione eseguita al passo 3.

È chiaro che questo programma applicato a un dato d'ingresso x dà come risultato lo stesso che darebbe $t(p)$ sul medesimo dato.

Teorema di Kleene



L'esistenza di p è legata essenzialmente a due fatti:

- l'esistenza di un interprete
- la capacità di un programma di ricopiarsi in un'altra area di memoria.

N.B.: in ogni modello di calcolo della stessa potenza espressiva delle MT abbiamo un "programma universale" che si comporta come un interprete.
Meno evidente è che anche linguaggi ad alto livello consentono a un programma di riprodursi. E.g., scrive su un file testo se stesso e invoca l'interprete sul nuovo file testo.

Conseguenze.
✓Consideriamo la classe delle trasformazioni che, dato un programma Pascal, permutino le linee di codice in qualsiasi modo (ma con un po' di cautela: in modo da avere sempre programmi sintatticamente corretti). Ebbene, esisterà sempre un programma equivalente al proprio trasformato!

✓Una classe di trasformazioni più interessanti sono quelle che premettono ad un dato programma q una specie di "programma principale", il quale poi, eventualmente in vari punti, chiama il programma q come sottoprogramma.

Esempio 17 $q'(x) := t(q)(x) :=$ *if* $x = 0$ *then 1 else* $x * q(x-1)$

Se come programma q *consideriamo* $q(x) = 3 * x$ *il programma trasformato* q' *calcolerà la funzione:*

$$f(x) = \begin{cases} 1 & \text{se } x = 0, \\ 3x^2 - 3x & \text{altrimenti.} \end{cases}$$

Pertanto il programma trasformato non è equivalente al suo trasformato e quindi non è punto fisso della trasformazione t*. Al contrario, ogni programma che calcola il fattoriale* ($x!$) *è un punto fisso della trasformazione.*

Nell'esempio precedente si può vedere che *solo* programmi che calcolano il fattoriale sono punti fissi.

Abbiamo quindi definito il programma che calcola il fattoriale dandone una definizione implicita attraverso la trasformazione q'.

In generale però, data una trasformazione t, non è vero che tutti i programmi punti fissi di t siano equivalenti tra loro.

Esempio 18 *Si consideri la seguente trasformazione* t *che avendo in input un programma a due argomenti* q *lo trasforma come segue:*
$q''(x,y) := t(q)(x,y) :=$ *if* $x = y$ *then* $y+1$ *else* $q(x, q(x-1, y+1))$
Si dimostra che i programmi che calcolano le funzioni:
$f(x,y) = x + 1$
$g(x,y) =$ *if* $x > y$ *then* $x + 1$ *else* $y - 1$
sono punti fissi di t*.*

Si osservi che le funzioni non sono definite ricorsivamente e possono essere facilmente espresse attraverso programmi RAM. Inoltre le funzioni sono differenti. Quindi, come nel caso di equazioni matematiche, anche nella definizione implicita di programma mediante trasformazioni t si possono avere soluzioni diverse.

Il rischio di definire programmi in modo implicito è quindi quello della ambiguità. Le assunzioni fatte nel teorema di Kleene non sono sufficienti per garantire la non abiguità delle definizioni ricorsive.

Per avere tali garanzie occorre:

- definire un ordinamento fra i punti fissi
- imporre ulteriori condizioni sulla trasformazione t

Questo studio va al di là degli obiettivi del corso.

✓In ogni modello di calcolo esiste un programma che fornisce costantemente in uscita il suo codice.

Corollario 19 *Sia data una enumerazione delle funzioni ricorsive. Esiste un indice i tale che, per ogni $x \in \mathbb{N}$ si ha $\varphi_i(x) = i$.*

Dimostrazione. Consideriamo la funzione proiezione:
$f(x, y) = x$
In base al teorema $s - m - n$ esiste una funzione ricorsiva totale s tale che:
$\varphi_{s(x)}(y) = f(x, y) = x$.
Per il teorema di ricorsione esiste un i per cui, per ogni y, $\varphi_i(y) = \varphi_{s(i)}(y) = i$.

Teorema 20 *(Rice) Sia data una enumerazione delle funzioni ricorsive e sia F un insieme di funzioni calcolabili. L'insieme $S = \{x | \varphi_x \in F\}$ è decidibile sse $F = \emptyset$ oppure F coincide con l'intera classe delle funzioni calcolabili.*

Il teorema sancisce quanto promesso: ogni proprietà non banale delle funzioni calcolabili è indecidibile; i teoremi precedenti sono applicazioni del teorema di Rice.

Esempio 21 *Dimostrare che la funzione:*

$$g(x) = \begin{cases} 1 & \text{se } \varphi_x \text{ è costante,} \\ 0 & \text{altrimenti.} \end{cases}$$

non è calcolabile, equivale a dimostrare la non decidibilità dell'insieme $S = \{x | \varphi_x \in F\}$, dove F è l'insieme delle funzioni calcolabili totali con valore costante.

Una conseguenza importante nella teoria della programmazione è che è impossibile provare proprietà delle funzioni calcolate dai programmi (costanza, crescenza, ...).

Esempio 22 *Data una enumerazione delle funzioni ricorsive e data una funzione ricorsiva f, la funzione che verifica la correttezza di un programma:*

$$p(x) = \begin{cases} 1 & \text{se } \varphi_x = f, \\ 0 & \text{altrimenti.} \end{cases}$$

non è calcolabile. Infatti questo equivale alla decidibilità dell'insieme $S = \{x | \varphi_x \in F\}$, dove F è l'insieme delle funzioni ricorsive che calcolano f.

Dal teorema di Rice discende, fra l'altro, che l'eliminazione di codice non necessario da un programma è indecidibile per un programma P.

Dimostrazione del teorema di Rice. Supponiamo che S sia decidibile. Siano i e j il primo indice appartenente a S ed il primo indice non appartenente a S. Quindi
$\varphi_i \in F$
$\varphi_j \notin F$
Cosideriamo la funzione:

$$C(x) = \begin{cases} i & \text{se } x \notin S, \\ j & \text{altrimenti.} \end{cases}$$

Poiché S è decidibile, C è totale.
Per il teorema di ricorsione esiste un e tale che $\varphi_{C(e)} = \varphi_e$. Per definizione,

se $C(e) = i$ allora $e \notin S$ e quindi $\varphi_e \notin F$, ma $\varphi_e = \varphi_{C(e)} = \varphi_i \in F$. Contraddizione!

se $C(e) = j$ allora $e \in S$ e quindi $\varphi_e \in F$, ma $\varphi_e = \varphi_{C(e)} = \varphi_j \notin F$. Contraddizione!

(q.e.d)

Interpretazione dei teoremi di Kleene e Rice.
Consideriamo un programma t che trasforma ogni programma π in un altro che calcola 1 per qualunque input se π soddisfa una qualche proprietà, altrimenti trasforma π in un programma che calcola 0 per ogni input.

Questo programma trasformatore t implementa una procedura di decisione.

Il teorema di Kleene afferma che esiste almeno un programma π', che il trasformatore t deve lasciare invariato, cioè il risultato delle procedura di decisione è un programma equivalente a quello di partenza.
L'avere quindi applicato la procedura di decisione non porta informazioni differenti da quelle contenute nel programma di partenza.

Il teorema di Rice aggiunge che un programma trasformatore t può esistere solo se si verificano proprietà banali di π. Un esempio di proprietà banale è verificare che un programma π avendo in input x o termina oppure non termina.

Infatti in questo caso banale, il traformatore t trasforma indifferentemente *tutti* i programmi in uno π' che restituisce sempre 1 per ogni x. Ovviamente, t trasforma anche π' in se stesso. Quindi π' è il punto fisso di t.

Questa riflessione puó suggerire di restringere la classe dei programmi "oggetto" (cioè in input a t), sperando di riportare nel campo della decidibilità le proprietà che più ci interessano.

Tuttavia va osservato che una tale classe non potrà mai contenere un "programma universale".

Esempio 23 *Se restringiamo l'attenzione alla classe dei programmi Pascal che non impiegano le istruzioni* **go to**, **while** *e* **repeat**, *nè le chiamate ricorsive di procedure e funzioni, allora è possibile garantire la terminazione di tali programmi per qualunque valore in ingresso.*

Si possono costruire programmi con queste caratteristiche per la risoluzione di molti problemi utili e non banali. Ad esempio, molti problemi numerici sono risolubili da programmi che impiegano solo i cicli **for**.

Insiemi decidibili e semidecidibili

Abbiamo già introdotto i concetti di insieme decidibile e semidecidibile, con riferimento alla T-calcolabilità. Possiamo riformulare quanto visto astraendo dal modello di calcolo.

Definizione 24 *Un insieme $A \subseteq \mathbb{N}$ è detto* ricorsivo *se la sua funzione caratteristica $\mathcal{C}_A$:*

$$\mathcal{C}_A(x) = \begin{cases} 1 & \text{se } x \in A, \\ 0 & \text{altrimenti} \end{cases}$$

è ricorsiva totale.

Definizione 25 *Un insieme $A \subseteq \mathbb{N}$ è detto* ricorsivamente enumerabile *(r.e.) se $A = \emptyset$ o se esiste una funzione ricorsiva totale $f : \mathbb{N} \longmapsto \mathbb{N}$ tale che $A = imm(f)$. In tal caso diciamo che la funzione f* enumera *l'insieme A.*

Dimostreremo che
la classe degli insiemi ricorsivi (risp., ricorsivamente enumerabili)
coincide con
la classe degli insiemi decidibili (semidecidibili).

Esempio 26

✓*l'insieme dei numeri pari è ricorsivo?*

✓*l'insieme dei numeri primi è ricorsivo o r.e.?*

✓*l'insieme* $\{\langle y,t\rangle|\varphi_y(y)$*si arresta in meno di* t *passi* $\}$ *è ricorsivo?*

✓*l'insieme* $K=\{y|\varphi_y(y)$*è definita*$\}$ *è ricorsivo?*

✓*l'insieme* $Z=\{\langle x,y,z\rangle|\varphi_x(y)=z\}$ *è ricorsivo?*

✓*l'insieme* $T=\{x|\varphi_x$ *è totale*$\}$ *è ricorsivo?*