URMSimulator

Unlimited Register Machines Simulator

Marco Grisanti

Sommario

1. Problema

Si vuole creare un software in grado di simulare il comportamento di una macchina a registri illimitati (URM). Tale software deve essere multipiattaforma in modo da poter essere eseguito su diversi sistemi operativi quali Windows, Linux e macOS.

Dunque, prima di procedere si vuole ricordare brevemente il funzionamento delle URM.

1.1 Le Macchine a Registri Illimitati (URM)

Una URM è dotata di un array infinito di registri $R_1, R_2, \dots$ ciascuno dei quali contiene un numero naturale $r_1, r_2, \dots$

Si definisce stato $\vec{r} \in N^\infty$ di una URM il contenuto dei suoi registri $R_1, R_2, \dots$

Si definisce configurazione istantanea di una URM una coppia $(k, \vec{r}) \in N^+ \times N^\infty$ dove:

- k è detto contatore di programma
- $\vec{r}$ è lo stato della URM

Si definisce programma URM una sequenza finita (di lunghezza s) di istruzioni $I_1, I_2, \dots, I_s$ di uno dei seguenti quattro tipi:

- Reset:
  - Sintassi: $Z(n)$
  - Descrizione: Viene azzerato il registro R_n.
  - Definizione formale: $(k, \vec{r}) \xrightarrow{Z(n)} (k+1, \overrightarrow{r'})$ con $r'_i = \begin{cases} r_i & se\ i \neq n \\ 0 & se\ i = n \end{cases}$
- Incremento:
  - Sintassi: $S(n)$
  - Descrizione: Viene incrementato di 1 r_n.
  - Definizione formale: $(k, \vec{r}) \xrightarrow{S(n)} (k+1, \overrightarrow{r'})$ con $r'_i = \begin{cases} r_i & se\ i \neq n \\ r_i + 1 & se\ i = n \end{cases}$
- Assegnamento:
  - Sintassi: $T(m, n)$
  - Descrizione: Viene assegnato r_m al registro R_n.
  - Definizione formale: $(k, \vec{r}) \xrightarrow{T(m,n)} (k+1, \overrightarrow{r'})$ con $r'_i = \begin{cases} r_i & se\ i \neq n \\ r_m & se\ i = n \end{cases}$
- Salto Condizionato:
  - Sintassi: $J(m, n, q)$
  - Descrizione: Se il r_m è uguale ad r_n, allora si salta all'istruzione I_q.
  - Definizione formale: $(k, \vec{r}) \xrightarrow{J(m,n,q)} (k', \vec{r})$ con $k' = \begin{cases} k+1 & se\ r_m \neq r_n \\ q & se\ r_m = r_n \end{cases}$

1.2 Istruzioni Specifiche per il Software

Nel software che si vuole realizzare, verranno definite le seguenti istruzioni:

- Register Initialization:
  - Sintassi: $I(n, x)$
  - Descrizione: Viene inizializzato il valore del registro R_n ad x.
  - Definizione formale: $(k, \vec{r}) \xrightarrow{I(n,x)} \left(k+1, \overrightarrow{r'}\right)$ con $r'_i = \begin{cases} r_i & se\ i \neq n \\ x & se\ i = n \end{cases}$
- Macro Execution:
  - Sintassi: $M(n, f(P_1, P_2, \dots, P_m))$ dove $P_1, P_2, \dots, P_m$ sono m registri passati in input alla macro f.
  - Descrizione: Viene salvato sul registro n l'output della macro f che prende in input i registri $P_1, P_2, \dots, P_m$.
  - Definizione formale: $(k, \vec{r}) \xrightarrow{M(n, f(P_1, P_2, \dots, P_m))} \left(k+1, \overrightarrow{r'}\right)$ con $r'_i = \begin{cases} r_i & se\ i \neq n \\ f(P_1, P_2, \dots, P_m) & se\ i = n \end{cases}$
- User Input
  - Sintassi: $input(n)$
  - Descrizione: L'utente comunica al software che verranno inizializzati in modo interattivo i primi n registri.

2. Lavoro Svolto

Il linguaggio di programmazione scelto per lo sviluppo del software è stato Java, grazie al quale è possibile creare applicazioni multipiattaforma in modo relativamente semplice. Inoltre, per lo sviluppo dell'interfaccia grafica è stato scelto Swing, ovvero un framework per Java, appartenente alle Java Foundation Classes (JFC) e orientato allo sviluppo di interfacce grafiche. L'ambiente di sviluppo integrato adoperato per la realizzazione del software è stato NetBeans.

2.1 Struttura dell'Interfaccia Grafica Utente

In questo breve paragrafo verranno mostrati gli aspetti principali dell'interfaccia grafica utente.

2.1.1 Integrazione con il File System

Un passo fondamentale nello sviluppo del software è stata l'integrazione con il file system. Ciò è stato realizzato per permettere all'utente di caricare dal proprio file system i vari programmi URM, che possono essere scritti con un qualunque editor di testo che supporti, per comodità dell'utente, la visualizzazione del numero di riga del codice sorgente, feature molto utile per le istruzioni di salto condizionato nelle quali si specifica una determinata istruzione verso la quale eseguire un salto. Il programma URM deve essere salvato con l'estensione "urm" per essere letto dal software.

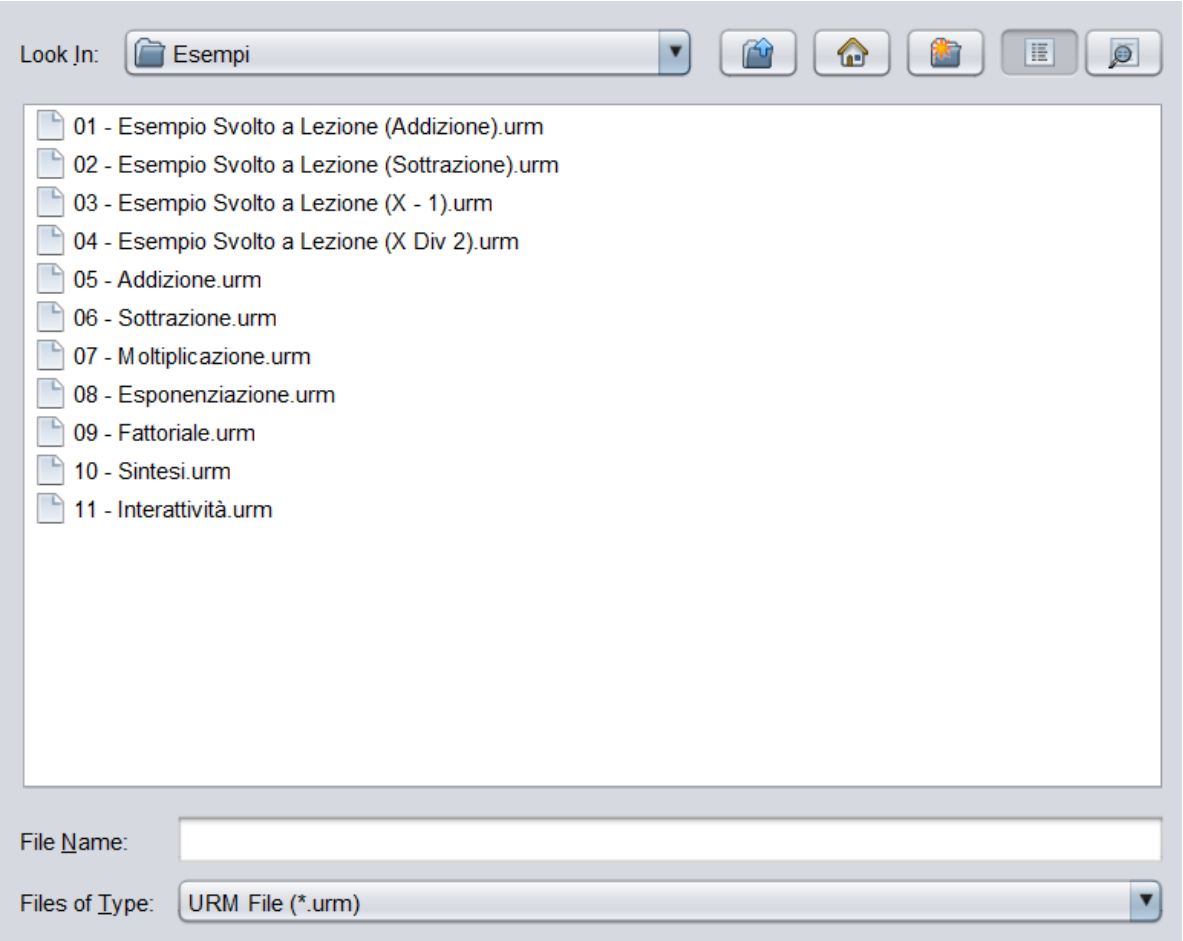

Figura 1 - Integrazione con il file system in modalità "List"

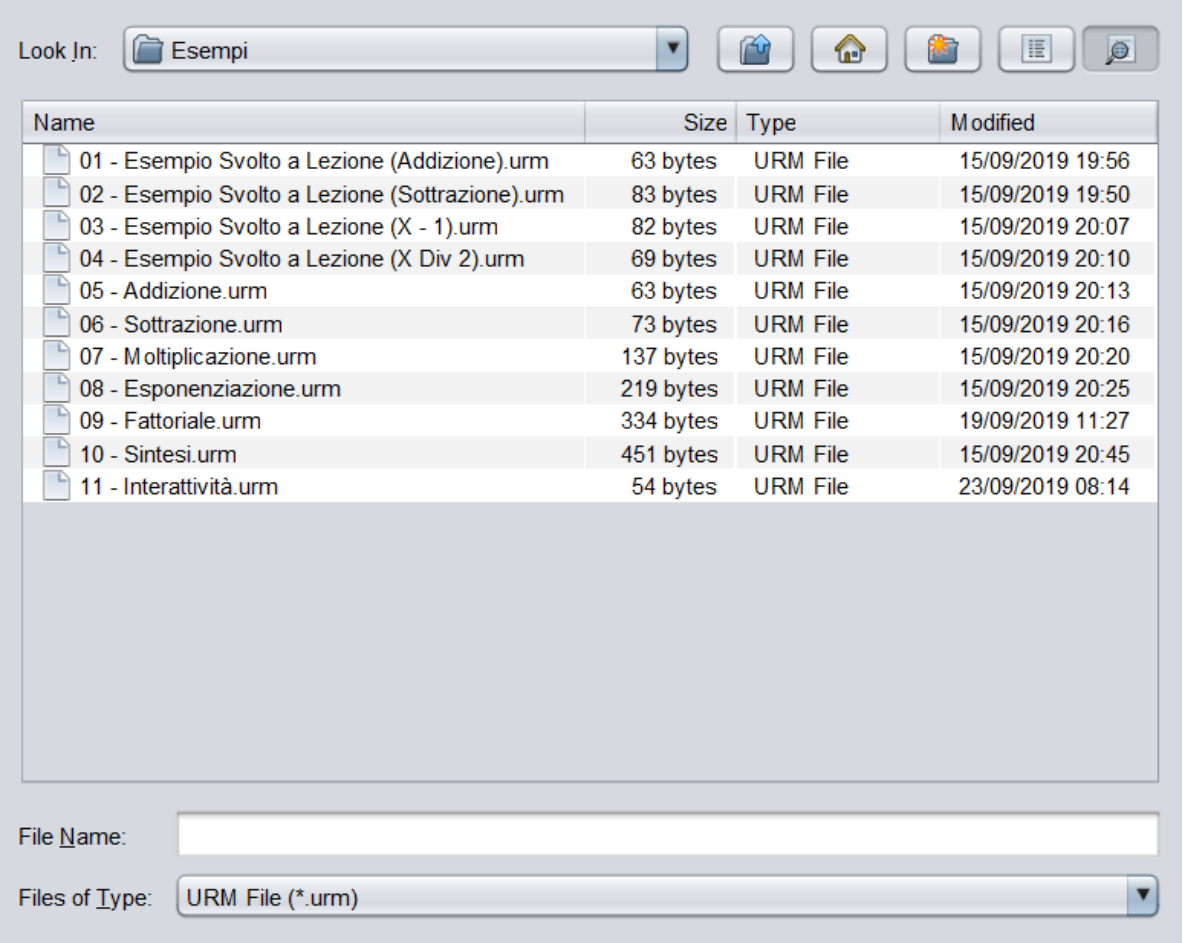

Figura 2 - Integrazione con il file system in modalità "Details"

2.1.2 Allocazione Dinamica dei Registri

Il software permette di usare un numero di registri potenzialmente infinito. Infatti, in base al codice del programma URM caricato, viene automaticamente rilevato il numero di registri dei quali la macchina ha bisogno per la corretta esecuzione del programma in questione. Dunque, per esempio, se si carica il seguente programma:

```
main:
I(1, 10)
I(2, 5)
J(2, 3, 8)
S(1)
S(3)
J(1, 1, 4)
'
```

Figura 3 - Esempio di un generico programma URM

Allora, una volta cliccato il pulsante "*Create/Reset Registers*", nella parte in alto del programma, compaiono tre registri, ovvero esattamente il numero di registri del quale il programma URM ha bisogno per il suo corretto funzionamento.

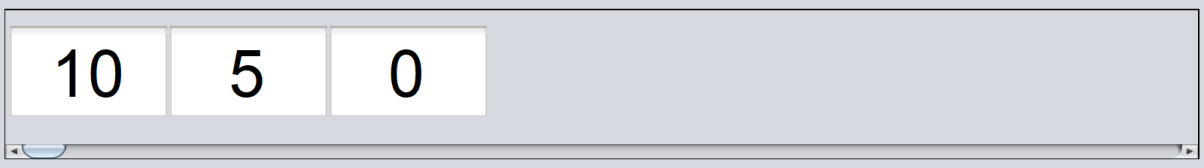

Figura 4 - Registri automaticamente rilevati dal precedente programma URM

2.1.3 Pulsanti di Controllo

La seguente immagine mostra i vari pulsanti grazie ai quali è possibile controllare l'esecuzione del programma URM.

Figura 5 - Principali pulsanti di controllo

- Start: Avvia/Riprende l'esecuzione del programma URM.
- Pause: Interrompe l'esecuzione del programma URM.
- Prev: Torna all'istruzione precedente.
- Next: Passa all'istruzione successiva.
- Create/Reset Registers: Crea/Resetta i registri del programma URM.

Inoltre, è presente il seguente pulsante che permette di impostare la velocità di esecuzione, in millisecondi, del programma URM caricato.

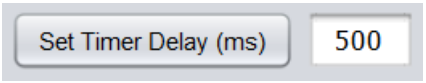

Figura 6 - Pulsante "Set Timer Delay (ms)"

2.1.4 Simbolo di Controllo

Una volta caricato il programma URM dal file system ed una volta avviata l'esecuzione di tale programma, nell'area di testo che mostra il programma URM caricato, si vedrà il simbolo '>' sull'istruzione attualmente considerata dal software. In questo modo, l'utente può vedere passo per passo l'esecuzione completa del programma URM.

```
main:
I(1, 10)
I(2, 5)
J(2, 3, 8)
>S(1)
S(3)
J(1, 1, 4)
'
```

Figura 7 - Simbolo di controllo '>' sull'istruzione "S(1)"

2.2 Le Macro

Il software permette di implementare le cosiddette macro, le quali possono essere considerate come delle funzioni che prendono in input dei registri con i relativi valori ad essi associati e restituiscono in output un valore.

Tutte le macro (così come il *main*, ovvero il corpo principale del programma) cominciano con la definizione del nome della macro seguito dai due punti e terminano con il singolo apice.

Ad esempio, è possibile definire una macro $sum(R_n, R_m)$ che sommi r_n ed r_m nel modo seguente:

```
sum:
J(2, 3, 6)
S(1)
S(3)
J(1, 1, 2)
'
```

Figura 8 - Macro che somma due numeri

Si noti che:

- il valore contenuto nel registro 1 sarà l'output della macro al termine della sua dell'esecuzione.
- r_n ed r_m saranno i valori contenuti nei primi due registri della macro. Tutti gli altri avranno come valore 0.
- Una macro può chiamare altre macro. Dunque, nel caso in cui una macro chiami sé stessa si dà luogo ad una funzione ricorsiva. A tal proposito, si vuole mostrare una funzione ricorsiva che calcola il fattoriale di un numero.

```
factorial:
S(2)
J(1, 6, 32)
M(4, sub(1, 2))
M(5, factorial(4))
M(1, mul(1, 5))
J(1, 1, 34)
S(3)
T(3, 1)
'
```

Figura 9 - Funzione ricorsiva che calcola il fattoriale di un numero

Dove

- $sub(R_n, R_m)$ è una macro che calcola la differenza tra r_n ed r_m.
- $mul(R_n, R_m)$ è una macro che calcola il prodotto tra r_n ed r_m.
- $factorial(R_n)$ è una macro che calcola il fattoriale di r_n.

2.3 Interattività

Grazie all'istruzione $input(n)$ è possibile utilizzare il software in modo interattivo, ovvero è possibile dare come input al programma URM dei valori senza che sia specificato nulla nel codice sorgente ad eccezione dell'istruzione $input(n)$ che comunica al software che verranno inizializzati in modo interattivo i primi n registri. Ad esempio, si consideri il seguente programma URM:

```
main:
input(2)
J(2, 3, 7)
S(1)
S(3)
J(1, 1, 3)
'
```

Figura 10 - Programma URM con l'istruzione "input(n)"

In questo caso, dopo la creazione dei registri, è possibile modificare il loro contenuto dall'interfaccia grafica utente in modo interattivo con il fine di capire velocemente il comportamento del programma URM su diversi input.

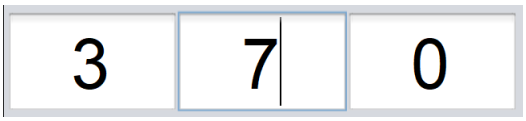

Figura 11 - Modifica interattiva dei registri

3. Conclusioni

Si ritiene che il software realizzato sia molto utile sia a livello accademico che non.

Infatti, da un punto di vista, grazie ai pulsanti di controllo e al simbolo di controllo che consentono passo per passo di vedere l'esecuzione dei programmi URM, il software permette di capire bene come funzionano tali programmi URM.

Invece, da un altro punto di vista, grazie all'utilizzo delle macro ed alla possibilità di avere un numero di registri potenzialmente infinito, il software permette di studiare programmi URM avanzati la cui analisi su "carta e penna" richiederebbe molto tempo in più.

4. Bibliografia

- Java Platform Standard Edition 8 Documentation
  https://docs.oracle.com/javase/8/docs/
- Java Platform Standard Edition 8 Documentation – Swing
  https://docs.oracle.com/javase/8/docs/api/javax/swing/package-summary.html
- NetBeans Documentation
  https://netbeans.org/kb/