Corso di *Laboratorio di Architettura degli Elaboratori*

A.A 2007/2008

Progetto: Grammatiche Regolari, Automi a Stati Finiti e Riconoscimento di Stringhe

Luca Maddalena
matricola n. 4563165

Data ultima revisione: 18/05/2008

Struttura del codice

Il codice è strutturato in procedure secondo il seguente schema logico:

- main
  - insgramm
    - parseprod
    - addprod
  - riconoscimento
    - prossimi_stati
  - check

Variabili globali e costanti

Nella procedura `main` sono definite le seguenti costanti:

Nome	Tipo	Valore	Descrizione
CHR_Z	word	90	Codice ASCII del carattere Z
CHR_A	word	65	Codice ASCII del carattere A
CHR_a	word	97	Codice ASCII del carattere a
CHR_z	word	122	Codice ASCII del carattere z
CHR_LF	word	10	Codice ASCII del carattere Line Feed (ritorno a capo)
CHR_SP	word	32	Codice ASCII del carattere Space (spazio)

Queste costanti vengono utilizzate nella fase di controllo e di *parsing* dei caratteri inseriti. In particolare:

- un codice ASCII compreso tra CHR_A e CHR_Z (65 e 90) identifica una lettera maiuscola;
- un codice ASCII compreso tra CHR_a e CHR_z (97 e 122) identifica una lettera minuscola;
- il codice ASCII CHR_Z (90) rappresenta il simbolo non terminale finale predefinito;
- il codice ASCII CHR_LF (10) viene utilizzato per determinare se è stato premuto invio.

Nella procedura `main` sono anche definite le seguenti due variabili che conterranno i puntatori alla lista collegata che rappresenta l'automa:

Nome	Tipo	Valore	Descrizione
TESTA	word	0	Testa dell'automa: puntatore al primo elemento della lista
CODA	word	0	Coda dell'automa: puntatore all'ultimo elemento della lista

Procedura main

E' l'*entry-point* del programma. Stampa il messaggio di benvenuto ed il menù per la scelta delle opzioni (1. inserimento della grammatica; 2 riconoscimento di parole; 3 uscita), dopodiché attende che venga inserito un carattere dalla console.

Opzione 1: inserimento della grammatica

Se è stata in scelta l'opzione 1 viene richiamata la procedura `insgramm` per l'inserimento della grammatica e successivamente la procedura `check` per il controllo di congruità della grammatica inserita. Se ci sono errori nella grammatica, la procedura `check` restituisce nel registro `$v0` il

valore 1 altrimenti `$zero`. In caso di errori viene stampato un messaggio di errore e l'intera grammatica viene scartata impostando TESTA e CODA con il valore del registro `$zero`.

Dopo aver effettuato il controllo della grammatica viene ripresentato il menù per la scelta delle opzioni.

Opzione 2: riconoscimento di parole

Richiama la procedura `riconoscimento`. La procedura chiede di inserire parola e verifica se questa appartiene alla grammatica specificata..

Opzione 3: uscita

Termina l'esecuzione del programma.

Procedura insgramm

Procedura per l'inserimento della grammatica. La procedura è composta da due cicli annidati. Quello più esterno consente di inserire un simbolo non terminale che deve necessariamente essere una lettera maiuscola escluso la "Z", mentre quello più interno consente di inserire le produzioni nella forma (t, NT) per il simbolo non terminale precedentemente inserito, dove "t" deve essere una lettera minuscola e NT una lettera maiuscola escluso la "Z". E' possibile uscire dai cicli di inserimento premendo invio.

Una volta inserito il simbolo non terminale, utilizzando le costanti CHR_A, CHR_Z, CHR_LF, vengono effettuati i seguenti controlli:

- se il valore inserito corrisponde a CHR_LF, significa che è stato premuto INVIO nella console, quindi esce dal ciclo e torna alla procedura `main`;
- se il valore inserito corrisponde a CHR_Z significa che è stato inserito il carattere "Z", che è il carattere non terminale finale predefinito, quindi stampa il messaggio di errore e ripete la richiesta di inserire un simbolo non terminale;
- se il valore inserito è minore di CHR_A o maggiore di CHR_Z significa che non è stata inserita una lettera maiuscola come richiesto, quindi stampa il messaggio di errore e ripete la richiesta di inserire un simbolo non terminale.

Se il simbolo non terminale inserito è corretto, la procedura chiede di inserire le produzioni.

Se il primo carattere della produzione corrisponde a CHR_LF, cioè se è stata premuto INVIO nella console, esce dal ciclo e torna all'inserimento del simbolo non terminale. In caso contrario, i caratteri inseriti vengono copiati nei registri `$a0` e `$a1 e` viene chiamata la procedura `parseprod` che si occupa di effettuare il *parsing* della produzione. In caso di esito positivo la procedura `parseprod` restituisce i valori dopo il *parsing* nei registri `$v0` e `$v1`, altrimenti valorizza questi due registri con il valore contenuto del registro `$zero`.

La procedura `insgramm` quindi, dopo la chiamata a `parseprod`, testa il valore del registro `$vo` per capire se la produzione inserita è corretta.
In caso affermativo viene richiamata la procedura `addprod` che salva la produzione in memoria, ovvero nella lista collegata che rappresenta l'automa, altrimenti stampa un messaggio di errore e ripete la richiesta di inserire una produzione.

Procedura parseprod

Effettua il *parsing* di una produzione che deve essere nella forma (t,NT) dove t è un simbolo terminale (lettera minuscola) e NT un simbolo non terminale (lettera maiuscola).

La procedura accetta in input due parametri nei registri `$a0` e `$a1`:

- `$a0` simbolo terminale della produzione ;
- `$a1` simbolo non terminale della produzione.

restituisce i valori di ritorno nei registri `$v0` e `$v1`:

- `$v0` simbolo terminale della produzione dopo il *parsing;*
- `$v1` simbolo non terminale della produzione dopo il *parsing;*

In caso di errore di *parsing* la procedura restituisce `$zero` nei registri di ritorno `$v0` e `$v1`.

La procedura controlla che il valore del primo elemento della produzione sia il codice ASCII di una lettera minuscola, cioè compreso tra CHR_a, CHR_z. Se non lo è stampa il messaggio di errore ed esce.

Se il primo elemento è corretto, controlla il secondo, che dovrebbe essere il codice ASCII di una lettera maiuscola escluso Z o un carattere di Line-Feed.

Per prima cosa controlla se il valore del secondo parametro è uguale a CHR_LF. Se lo è significa che non è stato inserito il secondo elemento della produzione e assume che questo sia il carattere "Z". A questo punto non occorre effettuare altri controlli e la procedura esce restituendo in `$v0` il simbolo terminale passato e in `$v1` il codice ASCII del carattere "Z".

Se invece il valore del secondo parametro è diverso da CHR_LF, occorre verificare che non sia il codice ASCII del carattere "Z", poiché non è consentito inserirlo direttamente. Se lo è, stampa il messaggio di errore ed esce.

Se il valore del secondo parametro è diverso da CHR_Z controlla che sia il codice ASCII di una lettera maiuscola, cioè compresa tra CHR_A e CHR_Z. Se non lo è stampa il messaggio di errore ed esce.

Se i precedenti controlli sono superati vengono valorizzati i registri `$v0` e `$v1` con i valori di ritorno.

Procedura addprod

Salva in memoria la produzione nella forma (NT,t,NT).
La procedura accetta in input tre parametri:

- `$a0` primo elemento della produzione (simbolo non terminale)
- `$a1` secondo elemento della produzione (simbolo terminale)
- `$a2` terzo elemento della produzione (simbolo non terminale)

La procedura non ha valori di ritorno in quanto i puntatori della lista sono salvati nelle variabili globali TESTA e CODA, che conterranno rispettivamente l'indirizzo del primo e dell'ultimo elemento.

La procedura, attraverso la `syscall sbrk`, alloca dinamicamente 16 byte, ovvero 4 word da 32 bit, per contenere i dati della produzione ed il puntatore all'elemento successivo della lista:

```
# --
# -- Alloca dinamicamente 16 byte in memoria
# --
li $v0, 9             # codice per allocare memoria ($a0 = ammount, $v0 = address)
li $a0, 16            # $a0 = numero di byte da allocare
syscall               # chiamata sbrk: restituisce un blocco di 16 byte, puntato da $v0
```

16 byte			
4 byte	4 byte	4 byte	4 byte
NT	t	NT	pointer to next item

- i primi 4 byte conterranno il primo elemento della produzione (simbolo non terminale);
- i successivi 4 byte conterranno il secondo elemento della produzione (simbolo terminale);
- i successivi 4 byte conterranno il terzo elemento della produzione (simbolo non terminale);
- gli ultimi 4 byte conterranno il puntatore all'elemento successivo o `$zero` in caso di ultimo elemento della lista.

L'indirizzo di partenza della porzione di memoria allocata dalla `syscall sbrk` viene restituito in `$v0`.

A questo punto viene verificato se si tratta del primo elemento della lista. Per fare questo è sufficiente controllare il valore contenuto all'indirizzo di TESTA. Se è diverso da `$zero` significa che la `sbrk` è già stata invocata e non si tratta del primo elemento:

```
lw  $t1, TESTA              # carica in $t1 il valore contenuto all'ind. puntato da  TESTA
bne $t1, $zero, nonprimo    # se non è il primo elemento salta a nonprimo
```

se non è il primo elemento della lista occorre collegare il nuovo elemento con il precedente. Questo si ottiene salvando negli ultimi 4 byte dell'elemento precedente il valore restituito in `$v0` dalla `sbrk` che rappresenta l'indirizzo di memoria del nuovo elemento, ed aggiornando il puntatore CODA della lista con il nuovo valore:

```
nonprimo:
        # --
        # -- Successive invocazioni, collega l'elemento precedente con il successivo
        # --
        lw $t2, CODA          # carica in $t2 l'indirizzo di CODA
        sw $v0, 12($t2)       # collega l'elemento precedente con il successivo
        sw $v0, CODA          # sposta il puntatore della CODA sul nuovo indirizzo
```

Se si tratta invece del primo elemento, i due puntatori TESTA e CODA saranno entrambi valorizzati con lo stesso valore di `$v0`:

```
primo:
        # --
        # -- Alla prima invocazione TESTA = CODA
        # --
        sw $v0, TESTA         # è il primo elemento, memorizza in TESTA l'indirizzo di  partenza
        sw $v0, CODA          # è il primo elemento, CODA = TESTA
```

Una volta sistemati i puntatori è possibile salvare la produzione nella struttura descritta precedentemente:

```
store:
        # --
        # -- Salva i valori nella lista
        # --
        sw $s0, 0($v0)          # salva il primo parametro nei primi 4 byte
        sw $s1, 4($v0)          # salva il secondo parametro nei 4 byte successivi
        sw $s2, 8($v0)          # salva il terzo parametro nei 4 byte successivi
        sw $zero, 12($v0)       # salva nil ($zero) nei restanti 4 byte
```

Procedura check

Controlla la grammatica inserita verificando che per ogni terzo elemento di ogni tripla (NT,t,NT), escluso il simbolo Z, esista una produzione.

La procedura restituisce i valori di ritorno nel registro `$v0`:

- `$v0` = 0 la grammatica è corretta, `$v0` = 1 sono presenti errori nella grammatica .

Per effettuare il controllo sono necessari due cicli annidati che nel codice sono identificati come `loop1` (il ciclo più esterno) e `loop2` (il ciclo più interno).

Nel ciclo esterno viene scorso tutto l'automa e recuperato il terzo elemento di ogni produzione. Se l'elemento corrisponde a CHR_Z passa al successivo, altrimenti il ciclo più interno scorre nuovamente l'automa per verificare che l'elemento recuperato precedentemente compaia come primo elemento di almeno una produzione. Il registro temporaneo `$t3` viene utilizzato come flag per segnalare la presenza di errori.

```
        lw $t0, TESTA           # carica in $t0 il valore contenuto in TESTA (indirizzo della lista)
        move $t3, $zero         # $t3 = 0 Flag che indica se ci sono errori

loop1:
        .
        . omissis
        .
        lw $a0, 8($t0)          # carica in $a0 il terzo carattere della produzione
        li $v0, 11              # codice per la stampa di un carattere
        syscall                 # stampa un carattere nella console

        lw $t1, CHR_Z           # mette in $t1 il codice ASCII del carattere Z
        beq $a0, $t1, next      # se l'ultimo elemento della tripla è Z passa alla successiva

# --
# -- verifica se per il terzo carattere della tripla (NT,t,NT) esiste una produzione
# --

        lw  $t1, TESTA
loop2:  lw  $t2, 0($t1)         # carica in $t2 il simbolo non terminale (primo elemento della tripla)

        beq $a0, $t2, next      # se $a0=$t2 la produzione è stata trovata passa al successivo elemento


        lw  $t1, 12($t1)        # puntatore all'elemento successivo
        bne $t1, $zero, loop2   # itera fino a quando non trova nil (0)

        li  $t3, 1              # non è stata trovata la produzione. Alza il flag di errore

        la  $a0, msgcheckerr
        li  $v0, 4
        syscall                 # stampa il messaggio di errore
```

```
next:   lw $t0, 12($t0)       # puntatore all'elemento successivo
        bne $t0, $zero, loop1 # itera fino a quando non trova nil (0)
```

Se nel ciclo interno non viene mai trovata una produzione corrispondente viene alzato un flag (`$t3`) che indica la presenza di errori nella grammatica.

Durante la verifica viene stampato l'elenco delle produzioni contrassegnando quelle errate.

Una volta terminato il ciclo esterno, nel registro `$t3` sarà presente il valore 1 se sono stati riscontrati errori, o 0 in caso contrario. A questo punto viene copiato il valore del registro `$t3` nel registro di ritorno `$v0` e la procedura termina.

Procedura prossimi_stati

La procedura, dato uno stato (NT) e un simbolo terminale (t), restituisce i puntatori ad una lista con tutti gli stati raggiungibili dalla coppia (NT, t) specificata.

La procedura accetta in input due parametri nei registri `$a0` e `$a1`:

- `$a0` simbolo non terminale (stato iniziale);
- `$a1` simbolo terminale

restituisce i valori di ritorno nei registri `$v0` e `$v1`:

- `$v0` puntatore al primo elemento della lista degli stati raggiungibili
- `$v1` puntatore all'ultimo elemento della lista degli stati raggiungibili

Struttura della lista degli stati raggiungibili:

8 byte	
4 byte	4 byte
NT (stato raggiungibile)	pointer to next item

La procedura, attraverso un ciclo, scorre l'automa e confronta i primi due elementi di ogni produzione con quelli passati. Se sono uguali, aggiunge il terzo elemento della produzione recuperata dall'automa alla lista degli stati raggiungibili.

Infine vengono valorizzati i registri di ritorno $v0 e $v1 con gli indirizzi del primo e dell'ultimo elemento della lista ottenuta.

Procedura riconoscimento

La procedura chiede di inserire una parola e verifica se questa appartiene alla grammatica specificata o meno.

La procedura utilizza internamente i seguenti registri:

- $s0: assioma (simbolo non terminale iniziale)
- $s1: puntatore al primo elemento della lista di stati raggiungibili
- $s2: puntatore all'ultimo elemento della lista di stati raggiungibili
- $s3: puntatore al prossimo carattere della parola
- $s4: carattere della parola letto

La procedura, attraverso la direttiva `.space,` riserva 64 byte per la stringa di input identificata con l'etichetta `parola`.
Dopo aver verificato la presenza di una grammatica, chiede di inserire una parola:

```
li $v0, 8             # codice per leggere una stringa arguments: $a0=buffer, $a1=length
la $a0, parola        # metto in $a0 il puntatore alla locazione di memoria di parola
li $a1, 64            # $a1 = lunghezza della stringa di input (in questo caso 64 byte)
syscall
```

Poi recupera il simbolo terminale iniziale (assioma) che per definizione è il primo simbolo non terminale inserito nell'automa e lo salva in `$s0`:

```
# --
# -- recupera l'assioma (simbolo non terminale iniziale)
# --
lw $t0, TESTA         # mette in $t0 il puntatore della variabile  TESTA
lw $s0, 0($t0)        # mette in $s0 il primo simbolo non terminale
```

A questo punto sono presenti due cicli. Quello più esterno (`loopChar`) scorre la stringa di input (parola) carattere per carattere, ignora eventuali spazi e salva il carattere letto in `$s4`. Quello più interno (`loopStato`) viene eseguito solo dalla seconda iterazione di `loopChar` in poi.

```
# --
# -- scorre la stringa carattere per carattere
# --
move $s4, $zero              # pulisce il registro $s4
la   $s3, parola             # mette in $s3 l'indirizzo di base della parola

loopChar:

# --
# -- Ignora gli spazi nella parola
# --
lb  $s4, 0($s3)              # carica in $s4 un carattere della parola
lw  $t3, CHR_SP              # $t2 = CHR_SP  (codice ascii del carattere SPACE)
beq $s4, $t3, nextChar       # se il carattere è uno spazio passa al successivo
```

La prima volta che viene eseguito il ciclo `loopChar`, cioè quando si valuta il primo carattere della parola, occorre chiamare la procedura `prossimi_stati` passando come parametri il carattere della parola (`$s4`) e il simbolo non terminale iniziale (`$s0`).
La procedura `prossimi_stati` ritornerà i puntatori ad una lista di stati raggiungibili dalla coppia di simboli specificati. Se la procedura `prossimi_stati` non ritorna alcun risultato esce con errore, altrimenti vengono salvati i puntatori della lista ottenuta in `$s1` e `$s2`. Questa lista conterrà gli stati di partenza per il successivo carattere della parola.

```
# --
# -- questa porzione di codice viene eseguita solo per il primo carattere della parola,
# -- ovvero se $s1 = $zero
# --
bne  $s1, $zero, np     # non è il primo carattere della parola, salta a nonprimo
move $a0, $s0           # $a0 = $s0 (parametro per la procedura prossimi_stati)
move $a1, $s4           # $a1 = $s4 (parametro per la procedura prossimi_stati)
jal  prossimi_stati     # richiama la procedura prossimi_stati
beq  $v0, $zero, riconerr # se la procedura non ritorna alcun risultato esce con errore
move $s1, $v0           # $s1 = $v0 salva in $s1 l'indirizzo della TESTA della lista
move $s2, $v1           # $s2 = $v1 salva in $s2 l'indirizzo della CODA della lista
j nextChar
```

Le volte successive che viene eseguito il ciclo `loopChar,` viene eseguito anche il ciclo interno `loopStato`, all'interno del quale viene richiamata la procedura `prossimi_stati` per

ciascuno stato presente nella lista di stati puntata da `$s0` e `$s1` unitamente al carattere della parola letto (`$s4`). Le liste di stati restituite dalle varie invocazioni di `prossimi_stati` vengono collegate l'una all'altra in modo da formare un'unica lista che rappresenterà gli stati di partenza per il carattere successivo della parola. I puntatori di questa nuova lista vengono salvati in `$s1` e `$s2` e il ciclo si ripete fino a quando non sono stati valutati tutti i caratteri della parola.

Una volta terminato il ciclo più esterno `loopChar`, cioè quando è stata valutata tutta la parola, la lista di stati puntata da `$s1` e `$s2` conterrà gli stati raggiungibili dall'ultimo carattere della parola. Se la lista è vuota (`$s1 = $zero`), la procedura stampa il messaggio di errore ed esce in quanto non è stato possibile raggiungere lo stato finale Z.

Se la lista non è vuota, effettua un ulteriore ciclo (`loopCheck`) per scorrere la lista degli stati e verificare se è presente lo stato finale Z.

Se lo stato Z è presente stampa il messaggio di validità della parola inserita, altrimenti quello di errore.

```
                # --
                # -- controlla se la lista contiene il simbolo terminale Z
                # --
                move $t0, $s1                # $t0 = $s1 puntatore all'elemento della lista
                lw   $t2, CHR_Z              # $t2 = CHR_Z  (codice ASCII Z)

loopCheck:
                lw  $t1, 0($t0)              # carica in $t1 il simbolo non terminale
                beq $t1, $t2, riconok        # ha trovato il simbolo Z. La parola è corretta

                lw  $t0, 4($t0)              # puntatore all'elemento successivo
                bne $t0, $zero, loopCheck    # itera fino a quando non trova nil (0)
                j   riconerr                 # non ha trovato Z, esce con errore


                # --
                # -- paraola riconosciuta. Stampa il messaggio ed esce
                # --
riconok:
                la $a0, msgricok             # metto in $a0 il puntatore alla stringa msgricok
                li $v0, 4                    # codice di stampa per le stringhe di caratteri
                syscall                      # stampa la stringa
                j exitriconoscimento         # esce dalla procedura
```

Una volta terminata la procedura viene ripresentato il menù principale.

Luca Maddalena