ProgettoLC parteA Riccardo Cavasin Relazione

Esercizio 1

È impossibile determinare la dimensione originaria di un immagine rapresentata da un QuadTree (compresso). Tuttavia, osservando la formula della media aritmetica $\bar{x}$ di n valori:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{4n}4\sum_{i=1}^{n} x_i$$

Si nota che espandendo ciascuna foglia di un livello sul risultato rimane invariato.

Non è possibile fare una media pesata sfruttando direttamente la suddivisione dell'albero poiché è mascherato, e il numero totale di foglie non è noto a priori. La soluzione al problema proposta effettua una sola visita dell'albero e calcola la profondità massima, la somma e il numero di foglie selezionate.
Ad ogni quadrante viene calcolata l'altezza massima delle foglie sottostanti e vengono compensati somma e conteggio dei rami più corti, moltiplicandoli per un fattore esponenziale sulla differenza di altezza.

La funzione di calcolo `f` gestisce esplicitamente il caso `img@(Q p1 p2 p3 p4) mask@(C True)` ma non il caso `img@(C {}) mask@(Q m1 m2 m3 m4)`. Il primo caso è più interessante poiché non richiede alcuna logica di selezione delle chiamate ricorsive, e scopo dimostrativo si sono mantenuti (per quanto ragionevole) i dati dei quattro rami "sciolti" (non aggregati in strutture dati).
Il caso `img@(Q p1 p2 p3 p4) mask@(C True)` non è stato implementato esplicitamente per maggiore leggibilità del codice.

La funzione è polimorfa e può essere usata su `Float` ma anche su `Rational` per poterla testare su output esatti.

Esercizio 2

Grammatica LL(1)

Si nota che la grammatica data è regolare e rappresentata dalla seguente espressione regolare:

```
{ (int id(,id)*;)? (id=num|goto num)(;(id=num|goto num))* }
```

Scrivo una grammatica equivalente:

- S $\rightarrow$ `{` D O `}`
- D $\rightarrow \epsilon$ | `int` `id` V `;`
- V $\rightarrow \epsilon$ | `,` `id` V
- O $\rightarrow$ C L
- L $\rightarrow \epsilon$ | `;` C L
- C $\rightarrow$ `id` `=` `num` | `goto` `num`

Il simbolo non terminale O è stato introdotto per avere messaggi d'errore più significativi.

first	**follow**
$first(\text{S}) = \{$ `{` $\}$	$follow(\text{S}) = \{$ \$ $\}$
$first(\text{D}) = \{ \epsilon,$ `int` $\}$	$follow(\text{D}) = \{$ `id`, `goto` $\}$
$first(\text{V}) = \{ \epsilon,$ `,` $\}$	$follow(\text{V}) = \{$ `;` $\}$
$first(\text{O}) = \{$ `id`, `goto` $\}$	$follow(\text{O}) = \{$ `}` $\}$
$first(\text{L}) = \{ \epsilon,$ `;` $\}$	$follow(\text{L}) = \{$ `}` $\}$
$first(\text{C}) = \{$ `id`, `goto` $\}$	$follow(\text{C}) = \{$ `,`, `}` $\}$
$first(\epsilon) = \{\epsilon\}$	
$first($`{` D O `}`$) = \{$ `{` $\}$	
$first($`int` `id` V `;`$) = \{$ `int` $\}$	
$first(\text{C L}) = \{$ `id`, `goto` $\}$	
$first($`;` C L$) = \{$ `;` $\}$	
$first($`id` `=` `num`$) = \{$ `id` $\}$	
$first($`goto` `num`$) = \{$ `goto` $\}$	

La grammatica riscritta è LL(1) dato che, per ogni produzione $A \rightarrow \alpha$ | β:

- vale che $first(\alpha) \cap first(\beta) = \emptyset$
- valgono $\epsilon \in first(\alpha) \implies first(\beta) \cap follow(A) = \emptyset$ e $\epsilon \in first(\beta) \implies first(\alpha) \cap follow(A) = \emptyset$

Tabella di parsing

	`,`	`;`	`=`	`goto`	`id`	`int`	`num`	`{`	`}`	`$`
C	4	12	17	C → `goto` `num`	C → `id` `=` `num`	14	10	7	18	0
D	4	5	11	D → ϵ	D → ϵ	D → `int` `id` V `;`	6	7	6	0
L	4	L → `;` C L	17	13	13	13	10	7	L → ϵ	0
O	4	12	17	O → C L	O → C L	14	10	7	15	0
S	2	2	2	1	1	1	2	S → `{` D O `}`	2	0
V	V → `,` `id` V	V → ϵ	11	8	8	9	10	7	16	0

Errori

n°	azioni
0	*print*: program ended too early *exit*
1	*print*: opening bracket `{` is missing *pop*, *push*: `}` O D
2	*print*: unexpected character outside of code block *skip*
4	*print*: `,` is allowed only in declarations *skip*
5	*print*: empty declaration statement (remove `;`) *skip*, *pop*
6	*pop*
7	*print*: nested/malformed block *skip*
8	*print*: unexpected command before closing `;` in declaration *pop*, *pop*
9	*print*: duplicated type specifier *push*: `id` `int`
10	*print*: {lookahead} is allowed only in a assignment rvalue *skip*
11	*print*: `=` is allowed only in the command section *skip*

n°	azioni
12	*print*: empty command statement (remove `;`) *skip*
13	*print*: missing `;` separator *push*: C
14	*print*: unexpected tipe specifier in command section *push*: D
15	*print*: block must contain at least one command *pop*
16	*print*: declarations must end with `;` *pop*, *pop*
17	*print*: lvalue missing in assignment *remove until*:[`{` `}` `int` `id` `;` `,` `goto`] excluded
18	*print*: expected a command after previous statement *pop*

Tabella dei mismatch

stack \ lookahead	`,`	`;`	`=`	`goto`	`id`	`int`	`num`	`{`	`}`	`$`
`,`	*acc*	0	0	0	0	0	0	0	0	0
`;`	0	*acc*	4	3	3	3	3	3	3	7
`=`	2	2	*acc*	2	2	2	2	2	3	7
`goto`	0	0	0	*acc*	0	0	0	0	0	0
`id`	1	1	1	1	*acc*	1	1	1	3	7
`int`	0	0	0	0	0	*acc*	0	0	0	0
`num`	4	3	4	3	5	3	*acc*	4	3	7
`{`	0	0	0	0	0	0	0	*acc*	0	0
`}`	0	0	0	0	0	0	0	0	*acc*	7
`$`	6	6	6	6	6	6	6	6	6	*halt*

Errori

n°	azioni
0	*print*: internal error exit
1	*print*: invalid identifier *skip*, *pop*
2	*print*: missing assignment operator *skip*, *pop*

n°	azioni
3	*print*: missing {stack} *pop*
4	*print*: unexpected {lookahead} *skip*
5	*print*: value must be a int literal *skip*, *pop*
6	*print*: unexpected character outside of code block *skip*
7	*print*: program ended too early *exit*

Commenti sulla gestione degli errori

- Il codice tra parestesi graffe è stato chiamato "blocco". Il blocco è costituito da una "sezione delle dichiarazioni" opzionale e una "sezione dei comandi" obbligatoria.
- In generale, si è deciso di prioritizzare l'error recovery sul codice delimitato dal blocco. Un eccezione è il caso in cui si incontra un simbolo iniziale di dichiarazione/operazione prima del'inizio del blocco, in cui si assume che l'utente abbia dimenticato `{`.
- Il simbolo O è stato sfruttato per riconoscere che la sezione dei comandi non può essere vuota.
- Nel caso si incontri una dichiarazione nella sezione dei comandi, il parser apre un contesto dichiarazione.

Esecuzione d'esempio

stack	input	azione
$ S	`{ b = goto 4; b=5; }$`	S → `{` D O `}`
$ `}` O D `{`	`{ b = goto 4; b=5; }$`	*acc*
$ `}` O D	`b = goto 4; b=5; }$`	D → ϵ
$ `}` O	`b = goto 4; b=5; }$`	O → C L
$ `}` L C	`b = goto 4; b=5; }$`	C → `id` `=` `num`
$ `}` L `num` `=` `id`	`b = goto 4; b=5; }$`	*acc*
$ `}` L `num` `=`	`= goto 4; b=5; }$`	*acc*
$ `}` L `num`	`goto 4; b=5; }$`	missing num *pop*
$ `}` L	`goto 4; b=5; }$`	missing `;` separator *push*: C
$ `}` L C	`goto 4; b=5; }$`	C → `goto` `num`
$ `}` L `num` `goto`	`goto 4; b=5; }$`	*acc*
$ `}` L `num`	`4; b=5; }$`	*acc*

stack	input	azione
$ `}` L	`; b=5; }$`	L ⟶ `;` C L
$ `}` L C `;`	`; b=5; }$`	*acc*
$ `}` L C	`b=5; }$`	C ⟶ `id` `=` `num`
$ `}` L `num` `=` `id`	`b=5; }$`	*acc*
$ `}` L `num` `=`	`=5; }$`	*acc*
$ `}` L `num`	`5; }$`	*acc*
$ `}` L	`; }$`	L ⟶ `;` C L
$ `}` L C `;`	`; }$`	*acc*
$ `}` L C	`}$`	expected a command after previous statement *pop*
$ `}` L	`}$`	L ⟶ ϵ
$ `}`	`}$`	*acc*
$	`$`	*halt*