Laboratorio di Linguaggi di Programmazione

Slide preparate da:
dott. Cataldo Musto - cataldomusto@di.uniba.it
dott. Fedelucio Narducci - narducci@di.uniba.it
Revisione: Pasquale Lops

Grammar Reader

- Prerequisiti
  - Conoscenza Linguaggio C
    - Sintassi del Grammar Reader
  - Conoscenza Dev C++
    - Compilazione Grammar Reader
    - Esecuzione Grammar Reader su file di esempio
  - Conoscenze sulle grammatiche
    - Simbolo iniziale, simbolo terminale, simbolo non terminale, grammatica context-free, ecc.

Esercizi

- Una parte della prova di Laboratorio è rappresentata dalla consegna degli esercizi proposti sul Grammar Reader
  - Non c'è un numero minimo di esercizi da consegnare
  - Le modalità e i tempi per la realizzazione degli esercizi sono totalmente liberi
  - Ci sono però delle linee guida generali

Linee Guida

- Contribuiscono a una buona valutazione
  - Il numero di esercizi implementati
  - Divisione del codice in file .h (libreria) e file .c (implementazione)
  - Implementazione di funzioni di tipo booleano invece che void
  - Testing approfondito del codice, soprattutto sui casi limite
  - Gestione degli errori approfondita
  - Qualità del codice, presenza di commenti, ecc.

Esercizio 1

- Controllare che nella parte sinistra di ogni produzione esista almeno un non terminale
  - esempio
    - aB>c = produzione valida
    - a>c = produzione non valida

Soluzione - Es. 1

- funzione [contieneNonTerminale (produzione)]
  - acquisisco la parte sinistra della produzione
  - scorro tutti i simboli presenti nella parte sinistra della produzione
    - se almeno un simbolo è di tipo non terminale
      - il valore restituito è vero
    - altrimenti
      - il valore restituito è falso

Note

- Alcuni concetti nella soluzione in pseudo codice sono stati evidenziati
  - Nel progettare la soluzione bisogna sfruttare le funzioni, le strutture dati già predisposte nel Grammar Reader e le relative componenti
    - Simbolo
      - Symbol s
    - Produzione
      - Production* p
    - Parte sinistra di una produzione
      - Word* w = &(p->left)
    - Lunghezza di una produzione
      - w->length

7

Note (2)

- Dove collocare questa nuova funzione?
  - E' una scelta di progettazione
  - Si può realizzare una funzione a parte oppure inserirlo nel codice
  - Dove?
    - L'avviso di produzione non corretta dovrebbe giungere all'utente nel momento in cui carica la grammatica
    - funzione load_grammar(Grammar* g)

8

Pseudo-Codice

- Quando la funzione load_grammar legge il simbolo di produzione
  - Controllo che nella parte sinistra appena letta sia contenuto un non terminale
    - Se è presente
      - Passa lo stato a RIGHT , per la lettura della parte destra
    - Altrimenti
      - Passa lo stato ad ERROR, per uscire dal caricamento

9

Note (3)

- In generale, gli esercizi dove nella traccia è chiesto di implementare un controllo sulla grammatica, sono da implementare come funzioni a parte.
  - esempio: isContextFree(Grammar* g) per verificare se una grammatica è libera da contesto
- Le altre sono da innestare nel codice in punti opportuni

10

Esercizio 2

- Controllare che la grammatica caricata contenga il simbolo iniziale
  - esempio
    - S>c = grammatica valida
    - aS>c = grammatica non valida!
- (oppure grammatica valida ma che genera il linguaggio vuoto)

11

Soluzione - Es. 2

- funzione [contiene_Simbolo_In (grammatica)]
  - acquisisco tutte le produzioni della grammatica
  - scorro tutte le produzioni
    - acquisisco la parte sinistra
      - se la lunghezza è 1 e il simbolo è uguale ad S
        - restituisci vero
      - altrimenti
        - restituisci falso

12

Esercizio 3

- Controllare che la grammatica caricata sia context-free
  - Definizione di produzione context-free (CF)
    - v > w con v simbolo non terminale
  - esempio
    - S>c = produzione valida e CF
    - a>b = produzione non valida!
    - aS>c = produzione valida, ma non CF
    - A> = produzione valida e CF
    - A>b = produzione valida e CF

13

Soluzione - Es. 3

- funzione [contextFree (grammatica)]
  - acquisisco tutte le produzioni della grammatica
  - scorro tutte le produzioni
    - acquisisco la parte sinistra
      - se, per ogni produzione, la lunghezza della parte sinistra è 1 e il simbolo è un non terminale
        - restituisci vero
      - altrimenti
        - restituisci falso

14

Esercizio 4

- Controllare che la grammatica caricata sia monotona
  - definizione
    - Per ogni produzione v > w, |v|<=|w|
  - esempio
    - S>c = produzione monotona
    - a>bC = produzione non valida
    - aS>c = produzione non monotona
    - A> = produzione non monotona
    - Ab>bc = produzione monotona

15

Soluzione - Es. 4

- funzione [isMonotonic(grammatica)]
  - acquisisco tutte le produzioni della grammatica
  - scorro tutte le produzioni
    - acquisisco la lunghezza della parte sinistra
    - acquisisco la lunghezza della parte destra
      - se, per ogni produzione, la lunghezza della parte sinistra è minore o uguale della lunghezza della parte destra
        - restituisci vero
      - altrimenti
        - restituisci falso

16

Esercizio 5

- Controllare che la grammatica caricata sia lineare destra
  - definizione
    - Una grammatica è lineare destra se contiene produzioni del tipo
      - A > bC con b simbolo terminale e A, C simboli non terminali
      - A > b con A simbolo non terminale e b simbolo non terminale (può essere anche lambda)
  - esempio
    - A>B = produzione non lineare destra
    - a>bC = produzione non valida
    - aS>c = produzione non lineare destra
    - A> = produzione lineare destra
    - Ab>bc = produzione non lineare destra
    - A>BC = produzione non lineare destra
    - A>bCb = produzione non lineare destra

17

Esercizio 5

- Verificare che una grammatica sia lineare destra richiede che vengano effettuati dei controlli sia sulla parte sinistra che sulla parte destra
  - La parte sinistra
    - deve avere sempre lunghezza 1
    - deve contenere solo un non terminale
  - La parte destra
    - non deve contenere due non terminali consecutivi
    - il non terminale, se c'è, deve essere sempre a destra del simbolo terminale

18

Soluzione - Es. 5

- funzione [lineare_destra(grammatica)]
  - acquisisco tutte le produzioni della grammatica
  - scorro tutte le produzioni
    - acquisisco la parte sinistra della produzione
    - acquisisco la parte destra della produzione
      - /* controlli sulla parte sinistra */
        - se la lunghezza della parte sinistra è maggiore di 1
        - restituisci falso
        - se la lunghezza della parte sinistra è uguale a 1 ma contiene un simbolo terminale
        - restituisci falso

Soluzione - Es. 5

- /* controlli sulla parte destra */
  - acquisisco i simboli presenti nella parte destra
  - se la lunghezza è uguale ad 1 e il simbolo è un non terminale
  - restituisci falso
  - se la lunghezza è uguale a 2
  - Acquisisci la coppia di simboli della produzione
  - se un non terminale è seguito da un altro non terminale
  - restituisci falso
  - se un non terminale è seguito da un terminale
  - restituisci falso
  - se un terminale è seguito da un altro terminale
  - restituisci falso
  - se la lunghezza è maggiore di 2
  - se restituisci falso
- Se nessuna di queste condizioni è verificata
  - restituisci vero

Esercizio 5

- Una soluzione alternativa
- La soluzione mostrata verificava che una grammatica fosse non lineare destra cercando di confutare una delle sue proprietà
- **E' possibile definire una soluzione alternativa**
  - Riprendiamo la definizione:
    - **A > bC** con b simbolo terminale e A, C simboli non terminali
    - **A > b** con A simbolo non terminale e b simbolo non terminale (può essere anche lambda)

Soluzione - Es. 5

- funzione [lineare_destra(grammatica)]
  - acquisisco tutte le produzioni della grammatica
  - scorro tutte le produzioni
    - acquisisco la parte sinistra della produzione
    - acquisisco la parte destra della produzione
      - /* controlli sulla parte sinistra */
        - se la lunghezza della parte sinistra = 1 e il simbolo è un non terminale
        - /* controlla la parte destra */
        - se la lunghezza = 1 il simbolo deve essere un terminale
        - se la lunghezza = 2, il primo simbolo deve essere un terminale e il secondo un non terminale
        - altrimenti restituisci falso
        - altrimenti restituisci falso

Esercizio 6

- Visulizzare accanto ad ogni produzione errata il tipo di errore
  - Dove inserire questa funzionalità?
  - Nel metodo che si occupa di caricare la grammatica
    - load_grammar( )

Esercizio 6 - Soluzione

- Differenziare l'insieme degli stati previsti nella funzione load_grammar( )
- In origine è previsto un generico stato di errore (ERROR)
- Una possibile soluzione
- Estendere l'insieme degli stati aggiungendo un nuovo stato per ogni tipologia di errore (assenza della S, assenza di un non-terminale, assenza del simbolo di produzione, ecc.)

Esercizio 6 - Soluzione

- Possibili miglioramenti
  - Creazione di una funzione "errorManager" che prenda in input l'identificativo dell'errore e mostri di volta in volta un messaggio differente

Lezione 5

- Grammar Reader
  - Gestione degli Errori
  - Inserimento di produzioni da tastiera
  - Cancellazione di produzioni

Gestione degli Errori

- Implementazione di una funzione ErrorManager( )
  - **Input**
    - Identificativo dell'errore
    - (dove possibile) La produzione che genera l'errore
  - **Output**
    - Messaggio di errore diverso in funzione dell'errore generato

Gestione degli Errori - 2

- Che tipologia di errori possiamo (dobbiamo) gestire?
  - **Errori in fase di lettura (load_grammar)**
    - Simboli non corretti
    - Assenza di simboli non terminali nella parte sinistra (load_grammar)
  - **Errori in fase di verifica**
    - Ogni funzione di verifica (es: verifica che sia lineare destra, che sia context-free, ecc.) genera degli errori diversi
    - Ogni errore deve essere gestito con un messaggio diverso da comunicare all'utente

Inserimento di Produzioni da Tastiera

- Modifica della funzione load_grammar( )
  - Utilizzare "stdin" invece del file di testo come parametro di input per la funzione di caricamento grammatica
  - **Opzionalmente:**
    - Salvataggio su file della grammatica caricata da tastiera
    - Modifica del menu per accettare come input sia l'inserimento da tastiera che quello da file

Aggiunta di Produzioni da Tastiera

- Progettare una funzione ch aggiunga (tramite tastiera) produzioni ad una grammatica già esistente
  - Utilizzo di stdin
  - Modifica di load_grammar
    - **Suggerimento:** la variabile che registra il numero di produzioni (g->numprod) dentro la funzione load_grammar( ) non deve essere impostata a zero

Cancellazione di Produzioni

- Progettare una funzione che cancelli una produzione da una grammatica già caricata
  - **Input:**
    - Grammatica
    - Produzione da cancellare (oppure un identificativo della produzione)
  - **Output:**
    - nuova Grammatica (che non contiene più la produzione da cancellare)

31

Cancellazione di Produzioni

- Progettare una funzione che cancelli una produzione da una grammatica già caricata
  - Intuizione:
    - Tutte le produzioni sono contenute in un vettore
    - Sappiamo quante sono le produzioni
    - Conosciamo la posizione della produzione da cancellare
    - **Possiamo "sovrascrivere" la produzione da cancellare sostituendola con una delle altre produzioni**

32

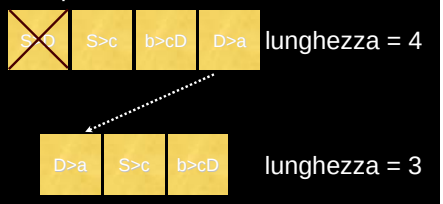
Soluzione - Es. 5

- funzione cancella(grammatica, produzione)
  - n = numero di produzioni
  - i = posizione della produzione da cancellare
  - assegno all'indice i del vettore delle produzioni la produzione memorizzata in posizione n
  - decremento il numero di produzioni della grammatica
  - Restituisco la nuova grammatica

33

Esempio

indice della produzione da eliminare = 1



34

Note

- **Opzionalmente:**
  - La funzione di cancellazione di produzioni può essere utilizzata per completare e/o arricchire alcune delle funzioni già implementate
- **Esempio:**
  - Nella funzione verifica che una grammatica sia context-free si può chiedere all'utente se vuole cancellare la produzione che non verifica i vincoli della grammatica
  - In fase di lettura di una grammatica si possono eliminare automaticamente le produzioni non valide
  - ecc.

35

Lezione 6

- Grammar Reader
  - Operazioni tra Grammatiche
    - Unione

36

Unione tra due grammatiche

- Definizione
  - Date due grammatiche $G_1 = (X_1, V_1, S, P_1)$ e $G_2 (X_2, V_2, S, P_2)$
  - Possiamo generare una nuova grammatica in base alle regole del teorema di chiusra
    - $G_3 = (X_3, V_3, S, P_3)$
      - $X_3 = X_1 \cup X_2$
      - $V_3 = V_1 \cup V_2$
        - Vincolo: V1 e V2 devono essere insiemi **disgiunti**

37

Note

- Gli insiemi contenenti i simboli non terminali devono essere disgiunti
  - **Perchè?**
    - Il linguaggio generato dalla grammatica unione deve essere uguale all'unione dei linguaggi generati dalle grammatiche di partenza
    - **Se gli insiemi non sono disgiunti questo non avviene**

38

Note

- **Esempio**
  - G_1 contiene le produzioni S>aA e A>b
  - G_2 contiene le produzioni S>bA e A>a
  - G_1 genera la stringa ab , G_2 genera la stringa ba
  - G_3 (unione di G_1 e G_2) dovrebbe generare solo **ab** e **ba**
  - Poichè l'insieme dei non terminali **non è disgiunto**, se unissimo semplicemente le produzioni otterremmo una grammatica non corretta
    - G_3 contiene le produzioni S>aA|bA , A>a|b
    - Questa grammatica genera le stringhe aa e bb che non devono appartenere alla grammatica
    - Nel progettare la soluzione con il Grammar Reader bisogna gestire questo aspetto, individuando una soluzione al problema !

39

Funzioni Aggiuntive

- La soluzione richiede la realizzazione di una serie di funzioni "ausiliarie", che attualmente non abbiamo a disposizione:
  - Il Grammar Reader deve poter prendere in input **due grammatiche** e non più una
  - Bisogna realizzare una funzione che **verifichi** che gli insiemi dei non terminali **siano disgiunti**
  - Eventualmente, sarebbe preferibile anche gestire la situazione in cui gli insiemi non sono disgiunti
    - Soluzione più semplice: mostrare un messaggio di errore
    - Soluzione più complessa: individuare una strategia per la gestione del problema

40

Caricamento di due grammatiche

- **"Il Grammar Reader deve poter prendere in input due grammatiche e non più una"**
  - Una possibile soluzione
    - definire due variabili di tipo "Grammar" e istanziare ciascuna di esse con il path del file contenente la grammatica
    - Ora abbiamo:
      - char* filename = argv[1]
    - Possiamo scrivere:
      - char* filename = "./grammar1.txt"
      - char* filename2 = "./grammar2.txt"
    - **A quel punto è possibile istanziare il load_grammar sui due file e memorizzare la grammatica in due variabili separate**

41

Verifica sui non terminali

- **"Bisogna realizzare una funzione che verifichi che gli insiemi dei non terminali siano disgiunti"**
- Sono possibili più soluzioni. Ad esempio:
  - Creare gli insiemi dei simboli non terminali per G1 e G2 e verificare che ogni elemento del primo insieme non appartenga al secondo
  - Scorrere **tutte le parti sinistre** delle produzioni di G1. Ogni qual volta si trova un simbolo non terminale si verifichi che esso non sia presente in nessuna parte sinistra delle produzioni di G2.
    - Altrimenti gli insiemi dei non terminali **non** sono disgiunti
      - Se gli insiemi sono disgiunti è sufficiente che l'algoritmo unisca i due insiemi di produzioni
      - Se gli insiemi non sono disgiunti è necessario individuare una strategia di gestione di questo errore

42

Verifica sui non terminali

- **Cosa fare se il non terminale è presente in entrambi gli insiemi?**
  - La strategia più banale è quella di mostrare un messaggio di errore ed uscire
  - La strategia più adatta è quella di modificare la grammatica sostituendo il non terminale con uno inutilizzato
    - Nel momento in cui il simbolo non terminale è presente in entrambi gli insiemi lo si sostituisce con un simbolo non utilizzato nelle regole di produzione
    - **Rivediamo l'esempio precedente**

Esempio

- **Esempio**
  - G1 contiene le produzioni S>aA e A>b
  - G2 contiene le produzioni S>bA e A>a
  - G1 genera la stringa ab , G2 genera la stringa ba
  - G3 (unione di G1 e G2) dovrebbe generare solo **ab** e **ba**
  - Poichè l'insieme dei non terminali **non è disgiunto**, se unissimo semplicemente le produzioni otterremmo una grammatica non corretta
    - G3 contiene le produzioni S>aA|bA , A>a|b
    - Questa grammatica genera le stringhe aa e bb che non devono appartenere alla grammatica
    - Risolviamo il problema sostituendo il non terminale A con un altro non terminale inutilizzato (ad esempio B) in una delle due grammatiche
    - G_3 contiene le produzioni S>aA|bB , A>b, B>a

Lezione 8

- Grammar Reader
  - Operazioni tra Grammatiche
    - Concatenazione

Concatenazione tra due grammatiche

- Definizione
  - Date due grammatiche $G_1=(X_1, V_1, S_1, P_1)$ e $G_2=(X_2, V_2, S, P_2)$
  - Possiamo generare una grammatica G_3
    - $G_3 = (X_3, V_3, S, P_3)$
      - La tipologia di produzioni contenute in P3 dipende da dalla natura delle due grammatiche originarie
        - Grammatiche di tipo 2 (context-free)
        - Grammatiche di tipo 3 (lineari destre)

Concatenazione tra due grammatiche

- Grammatiche Context-Free (tipo 2)
  - $P_3 = (S > S_1S_2) \cup P_1 \cup P_2$
  - Si crea una produzione che concateni i due simboli di partenza delle due grammatiche originarie e si uniscono le produzioni rimanenti

Soluzione

- funzione [concatenazione(g1, g2)]
- g3 = grammatica concatenazione , p_g3 = produzioni grammatica g3
  - p_g1 = produzioni della grammatica g1
  - p_g2 = produzioni della grammatica g2
    - se context_free(g1) && context_free(g2)
      - Inserisci in p_g3 una produzione del tipo $S>S_1S_2$
      - Inserisci in p_g3 tutte le produzioni di p_g1
      - Inserisci in p_g3 tutte le produzioni di p_g2
    - se lineare_destra(g1) && lineare_destra (g2)
      - <prossima lezione>

Concatenazione tra due grammatiche

- Abbiamo già a disposizione le funzioni che risolvono quasi tutti i controlli necessari
  - Controllo che una grammatica sia lineare destra o context-free
  - Controllo sull'esistenza di una produzione
  - Aggiunta di produzioni da un insieme all'altro
  - Cosa bisogna gestire?
    - La presenza di produzioni del tipo S>S.S.

Concatenazione tra due grammatiche

- Come gestire S>S1S2 ?
- Abbiamo a disposizione due strade
- Come al solito, una semplice e una più complicata :-)
- **Soluzione 1**
  - Si usa la funzione di ridenominazione già utilizzata per l'unione, e si sostituisce al simbolo di partenza di ciascuna grammatica un simbolo non terminale (libero) alternativo (che diventerà il nuovo simbolo di partenza)
- Esempio

Esempio

- Esempio
  - G1 contiene la produzione S>a
  - G2 contiene la produzione S>b
  - Il linguaggio di G1 è composto da **a**
  - Il linguaggio di G2 è composto da **b**
  - Il linguaggio di G3 (concatenazione) è composto da ab
  - Le produzioni di G3 sono
    - S > XY
    - X > a (sostituzione di S con X)
    - Y > b (sostituzione di S con Y)

Concatenazione tra due grammatiche

- **Soluzione 2**
  - Si cambia la dichiarazione del tipo di dato "Symbol"
  - Ora simbolo è un tipo di dato che equivale a char
    - typedef char Symbol
  - Symbol = un carattere
  - Possiamo cambiare la struttura del tipo symbol rendendolo **un tipo di dato strutturato**

Concatenazione tra due grammatiche

- **Soluzione 2**
  - Symbol diviene una struttura che potrà contenere più di un carattere
  - **Campi**
    - sym[ ] - vettore dei singoli simboli
    - length - numero di singoli utilizzati
    - MAX_SYMBOL_LENGTH - costante che descrive il numero
    - **Vediamo la dichiarazione**

La nuova struttura Symbol

- typedef struct
- {
- char sym [MAX_SYMBOL_LENGTH]; unsigned length;
- } Symbol;
- Attenzione:
- Cambiando la struttura del tipo di dato bisogna cambiare l'implementazione di tutti i metodi che usano quel tipo di dato
- Esempio: isNonTerminal(s)
- In questo modo possiamo definire delle produzioni in cui il "simbolo" non è più composto da un solo carattere ma da più caratteri concatenati (S1 o S2).
- A questo punto si può ridefinire il metodo che sostituisce il non terminale con un metodo che aggiunge il numero 1 e il numero 2 accanto al simbolo di partenza di ciascuna delle due grammatiche.
-

Lezione 9

- Grammar Reader
  - Operazioni tra Grammatiche
    - Concatenazione
  - Verifica Grammatiche Context-Sensitive

55

Concatenazione tra Grammatiche

- Note integrative
  - La realizzazione dell'esercizio che effettua la concatenazione tra due grammatiche di tipo 3 non è esplicitamente richiesto, anche se gradito :)
  - L'algoritmo illustrato in precedenza deve essere esteso introducendo anche in questo caso la verifica che non ci siano non terminali comuni

56

Esempio

- Esempio
  - Grammatica 1 = S1 > aA , A>a
  - Grammatica 2 = S2> bA , A>b
  - L(G1) = aa , L(G2) = bb , L(G1*G2) = aabb
  - Senza la ridenominazione dei non terminali, la grammatica concatenata diviene
  - S>S1S2 , S1>aA , S2>bA , A>a, A>b
    - Permette di generare anche ad esempio anche **abbb** che non appartiene al linguaggio!
    - S>S1S2 - S>aAS2 - S>aAbA - S>abbb

57

Soluzione Aggiornata

- funzione [concatenazione(g1, g2)]
- g3 = grammatica concatenazione , p_g3 = produzioni grammatica g3
  - p_g1 = produzioni della grammatica g1
  - p_g2 = produzioni della grammatica g2
    - se context_free(g1) && context_free(g2)
      - Inserisci in p_g3 una produzione del tipo $S>S_1S_2$
      - Ridenomina i non terminali comuni
      - Inserisci in p_g3 tutte le produzioni di p_g1
      - Inserisci in p_g3 tutte le produzioni di p_g2
    - altrimenti
      - Mostra un messaggio di errore

58

Grammatiche Context-Sensitive

- Una grammatica si dice dipendente da contesto (context-sensitive, C.S.) se le sue produzioni sono del tipo
  - yAz > ywz
  - S > lambda, ed S non compare alla destra di altre produzioni)
    - y e z si dicono "contesti" (sequenze di non terminali e terminali con lunghezza maggiore o uguale a zero)
    - w sequenza di non terminali e terminali con lunghezza maggiore di zero

59

Un po' di teoria

- Una grammatica si dice dipendente da contesto (context-sensitive, C.S.) se le sue produzioni sono del tipo
  - yAz > ywz
  - S > lambda
    - Nel caso in cui y = lambda e z = lambda la prima tipologia di produzioni diviene A>w. Torniamo nel caso delle grammatiche C.F.
    - Una grammatica libera da contesto è dunque una grammatica in cui il contesto sinistro e il contesto destro di una produzione è vuoto
    - La differenza tra le grammatiche CF e CS è che nelle grammatiche CF sono possibili produzioni del tipo A>lambda , mentre invece nelle CS no (la lunghezza di w deve essere maggiore di 0)

60

Algoritmo

- Intuizione
  - Nelle grammatiche context-free la parte sinistra e la parte destra di ogni produzione condividono un insieme di simboli, detto “contesto”
  - **Esempio**
    - aaBbb > aabbb è una produzione context-sensitive
    - aa è detto contesto sinistro
    - bb è detto contesto destro
  - **Per verificare che una grammatica sia context-sensitive abbiamo bisogno di**
    - **Individuare i contesti**
    - **Confrontare i contesti**

61

Individuare i contesti

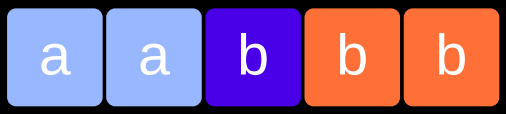
- Come individuiamo il contesto sinistro e il contesto destro?
  - In modo banale
    - Scorriamo in parallelo i simboli della parte sinistra e della parte destra
    - Finchè i simboli corrispondono continuiamo a procedere
    - Appena i simboli sono diversi tra loro significa che il contesto è terminato
  - La stessa cosa si fa per individuare il contesto destro, partendo dalla fine della stringa invece che dall’inizio

62

Esempio

Parte Sinistra

Parte Destra



63

Individuare i contesti

- **Primo confronto** , a = a
- **Secondo confronto** , a = a
- **Terzo confronto** = X <> b
  - E’ finito il contesto sinistro. La sua lunghezza è uguale a due
- **Quarto confronto** , b = b
  - Si parte dal fondo
- **Quinto confronto,** b = b
- **Sesto confronto,** X <> b
  - E’ finito il contesto sinistro. La sua lunghezza è uguale a due

64

..e ora?

- Una volta individuato il contesto sinistro e il contesto destro possiamo verificare se una grammatica è dipendente da contesto oppure no
  - Come ?
  - Bisogna implementare tre controlli
    - La lunghezza della parte sinistra deve essere uguale alla lunghezza del contesto sinistro + la lunghezza del contesto destro + 1
    - La lunghezza del contesto sinistro + la lunghezza del contesto destro deve essere minore della lunghezza dell’intera parte destra
    - Se l’intera parte destra ha lunghezza zero, allora la parte sinistra deve per forza essere S

65

Controllo 1

- **La lunghezza della parte sinistra deve essere uguale alla lunghezza del contesto sinistro più la lunghezza del contesto destro +1**
  - Perchè?
  - Le produzioni C.S. sono del tipo yAz > ywz
  - y e z possono avere lunghezza variabile
  - A invece deve essere appartenente all’insieme dei non terminali e deve essere un solo simbolo
  - |yAz| = |y| + |A| + |z| = **|y| + 1 + |z|**

66

Controllo 2

- **La lunghezza del contesto sinistro più la lunghezza del contesto destro deve essere minore della lunghezza dell'intera parte destra**
  - Perchè?
  - Le produzioni C.S. sono del tipo yAz > ywz
  - y e z possono avere lunghezza variabile
  - w può essere una sequenza più o meno lunga di simboli, però deve essere **necessariamente diversa da lambda**
  - |ywz| = |y| + |w| + |z|
  - poichè |w| > 0
    - |y| + |z| < |y| + |w| + |z|

Controllo 3

- **Se l'intera parte destra ha lunghezza pari a 0, allora la parte sinistra deve essere S**
  - Perchè?
    - L'unica produzione senza contesto ammissibile in una grammatica C.S. è del tipo S > lambda

Soluzione - parte 1

- funzione [context_sensitive(g1)]
- boolean cs = true; contesto_sin = 0 ; contesto_dx = 0;
- acquisisco il vettore con tutte le produzioni di g1
- finchè il vettore non è esaurito e finchè cs = true
  - prendo la i-esima produzione
  - scorro i simboli della parte sinistra e scorro i simboli della parte destra
    - finchè i simboli sono uguali
      - incrementa la lunghezza del contesto sinistro
  - scorro i simboli della parte sinistra e scorro i simboli della parte destra (partendo dal fondo)
    - finchè i simboli sono uguali
      - incrementa la lunghezza del contesto destro

Soluzione - parte 2

- se la lunghezza della parte sinistra è uguale alla lunghezza del contesto sinistro + lunghezza del contesto destro + 1
  - se la lunghezza della parte destra è maggiore alla lunghezza del contesto sinistro + lunghezza del contesto destro
    - CS = true
  - se la lunghezza della parte destra è uguale a zero ed il simbolo a sinistra è diverso da S
    - CS = false
  - altrimenti
    - CS = true
- altrimenti
  - CS = false

Lezione 10

- Grammar Reader
  - Trasformazione di un insieme di produzioni monotone in un insieme di produzioni contestuali

Teoria

- Equivalenza tra grammatiche monotone e grammatiche contestuali
  - Data una grammatica G monotona è possibile costruire una grammatica G' in cui le **produzioni** monotone vengono sostituite con **produzioni** contestuali
  - L(G) = L(G')
    - Le due grammatiche generano lo stesso linguaggio

Esempio

- Data una produzione monotona
  - ABC > bcdef
- L'algoritmo deve sostituire la produzione monotona con **2K produzioni contestuali** (k = numero di simboli non terminali presenti nella parte sinistra della produzione monotona)
- Nell'esempio k=3 , quindi 2k = 6
  - ABC > DBC
  - DBC > DEC
  - DEC > DEFef
  - DEFef > DEdef
  - DEdef > Dcdef
  - Dcdef >bcdef

73

Implementazione

- L'implementazione è lasciata totalmente per esercizio
- **Linee Guida**
  - La funzione deve prendere in input una grammatica G e deve restituire in output una grammatica G'
  - La funzione deve riutilizzare il metodo di individuazione di un non-terminale libero già realizzato in precedenza
  - La funzione deve implementare un ciclo in cui ad ogni passo si aggiunge alla nuova grammatica una nuova produzione
    - La parte sinistra di ogni produzione (esclusa la prima) è identica alla parte destra della produzione precedente
    - La parte destra di ogni produzione viene riscritta in base al teorema di equivalenza dimostrato

74