Corso: Fondamenti, Linguaggi e Traduttori

Paola Giannini

Analisi Lessicale: Scanner/Lexer

Il lessico

- Il lessico descrive, le parole o elementi lessicali che compongono le frasi. Nei linguaggi artificiali gli elementi lessicali possono essere assegnati alle seguenti classi:
  - Parole chiave:, sono particolari parole fisse che caratterizzano vari tipi di frasi o strutture. Ad es.: `if`, `for`, `class`.
  - Delimitatori (`;`) operatori (+,++,..): come i precedenti sono delle parole fisse composte di caratteri anche non alfabetici.
  - Commenti in Java sono aperti da `/*` e chiusi da `*/`.
  - Classi lessicali aperte: queste comprendono un numero illimitato di elementi lessicali, che devono avere la struttura di un linguaggio regolare ossia a stati finiti.Esempi tipici sono:
    - nomi o identificatori di variabili, di funzioni, o altro (classi, metodi,..) definiti dalla espressione regolare:

      ```
      Id = (Lettera | _) (Lettera | Cifra | _)*
      ```

    - costanti quali i numeri interi o reali o le stringhe alfanumeriche.

Analisi lessicale

- Differenza fra parole chiave e classi lessicali aperte:
  - le prime non hanno altra informazione che il proprio nome,
  - le altre denotano delle entità che hanno un valore o altre proprietà (che chiameremo attributi semantici). Ad esempio le costanti hanno un valore.
- Le classi lessicali sono delle stringhe appartenenti ad un linguaggio formale del tipo regolare. Questi linguaggi sono descritti dalle espressioni regolari e riconosciuti dagli automi a stati finiti deterministici.
- L'analizzatore lessicale non deve solo verificare che una sotto-stringa del testo sorgente corrisponde ad un elemento lessicale valido, ma deve anche tradurla in una opportuna codifica che faciliti la successiva elaborazione da parte del traduttore o interprete.
- La codifica deve contenere:
  - l'identificativo della classe lessicale cui l'elemento appartiene
  - e gli attributi semantici (nel caso ve ne siano associati a tale classe).

UPO UNIVERSITÀ DEL PIEMONTE ORIENTALE

Ruolo dell'analizzatore lessicale

- Fornire un modo per isolare le regole di basso livello dalle strutture che costituiscono la sintassi del linguaggio.
- Suddividere la frase in ingresso in elementi lessicali (detti tokens) da fornire al parser.
- Eliminare gli spazi bianchi e i commenti

UPO
UNIVERSITÀ DEL PIEMONTE ORIENTALE

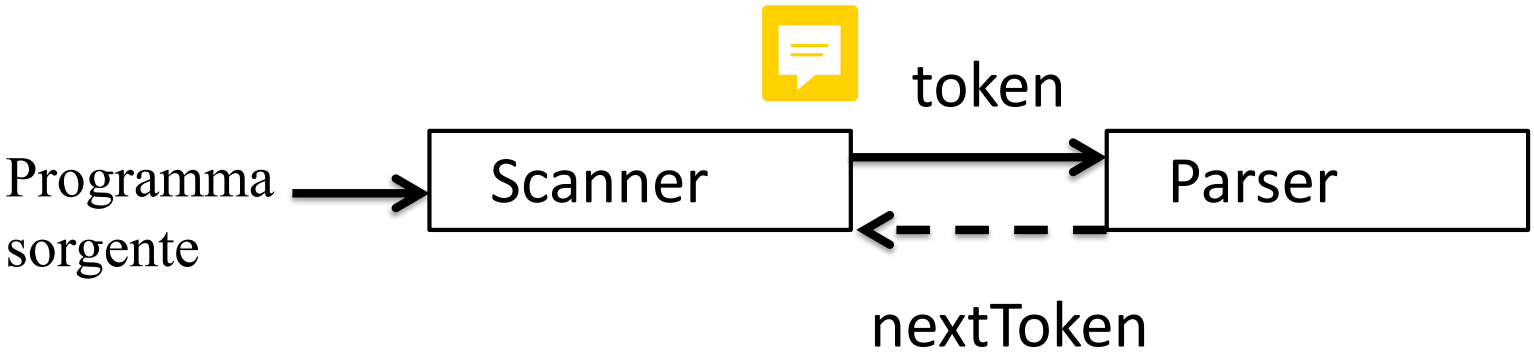
Terminologia

- Token: unità lessicale restituita dall'analizzatore lessicale e fornita come ingresso al parser (e.g., costante, identificatore, operatore,...)
- Lessico: stringa di caratteri che rappresentano un particolare token.
- Pattern: una descrizione del lessico che corrisponde al token.

UPO
UNIVERSITÀ DEL PIEMONTE ORIENTALE

Interazione Scanner-Parser

Può avvenire in 2 modi:
1) Lo scanner processa tutto il programma sorgente prima che inizi il parsing (tutti i token sono memorizzati in un file o tabella).
2) Lo scanner è chiamato dal parser quando c'è bisogno di un altro token, per cui non si deve memorizzare la sequenza di token.



Come realizzare un analizzatore lessicale

1. Realizzazione procedurale: un programma ad-hoc che riconosce tutti gli elementi lessicali producendo i corrispondenti token.
   1. A partire dalle espressioni regolari
   2. A partire dall'automa a stati finiti
   3. A partire dalla grammatica regolare
2. Realizzazione tabulare interpretata: una struttura dati (tabella) rappresenta il DFA riconoscitore della grammatica G e un programma indipendente dalla grammatica G realizza il funziomento del DFA.
3. Automaticamente con un generatore di Scanner ad esempio JFLex (che prende come input le espressioni regolari corrispondenti ai token)

UPO
UNIVERSITÀ DEL PIEMONTE ORIENTALE

Realizzazione procedurale (1)

Dall'Espressione Regolare al Codice per l'analisi. Si scrive

- una sequenza per ogni concatenazione
- un test per ogni unione
- un ciclo per ogni stella di kleene

Esempio, se D=1,...,9: l'espressione regolare per i letterali interi

D (D | 0)* | 0

è riconosciuta dal seguente codice dove `peekChar()` restituisce il prossimo carattere senza rimuoverlo dall'input, mentre `readChar()` lo rimuove.

```
valTk =""
if (peekChar() in D) {
   valTk += readChar()
   while (peekChar() in D || peekChar() == '0'){
      valTk += readChar()
   }
   return Token(INT,valTk); //Pattern riconosciuto
}
if (peekChar() == '0') {
   valTk +=  readChar()
   if (peekChar() in D) ERRORE
   else  Token(INT,valTk)  //Pattern riconosciuto
}
else ERRORE
```

Riconoscere un commento di linea

L'espressione regolare, supponendo di avere `Eol` come simbolo di fine linea e `Not` che significa tutti i simboli eccetto quello specificato

```
// Not(Eol)* Eol
```

```
if (peekChar() ==  '/') {
   readChar()
   if (peekChar() == '/') {// il secondo /
      do
         readChar()
      while ( !(peekChar() in {Eol, Eof}) )//Eof denota fine file
      if (peekChar() == Eol)
         // Processa il commento
      else ERRORE // non c'e' la fine della linea
   }
   else ERRORE //il secondo carattere non e' "/"
}
```

Realizzazione procedurale (2)

Dalla Grammatica Regolare al Codice per l'analisi. Si scrive

- una funzione/metodo per ogni non terminale
- un test per ogni alternativa
- Si richiama la funzione per ogni non-terminale che compare nella parte destra della produzione

Poco usata per i linguaggi regolari perchè questi vengono in genere descritti da espressioni regolari.

Come mai?

Per il nostri compilatore useremo:

- la realizzazione procedurale a partire dall'espressione regolare per l'analisi lessicale
- la realizzazione procedurale a partire dalla grammatica per l'analisi sintattica, cioè per una grammatica context free (LL).

UPO
UNIVERSITÀ DEL PIEMONTE ORIENTALE

Realizzazione Tabulare

Dall'Espressione Regolare all'Automa a Stati Finiti.

- Si costruisce una tabella T tale che dato lo stato s e il carattere c, se $T(s, c) = s'$ allora s' è lo stato successivo.
- Si definisce la funzione che esegue l'automa, che NON dipende dalla particolare espressione regolare.

```
valTk =""
State = StartState
while ( (peekChar() != eof) && (State non in F) {
   NextState = T[State][peekChar()]
   if(NextState == error) return ERRORE
   State = NextState
   valTk +=  readChar()
}
if (State in F) return Token(_,valTk) // il token riconosciuto
else return ERRORE // errore  lessicale
```

Questa realizzazione è usata dai generatori di analizzatori lessicali. Perchè?

Identificatori e Parole Chiave (1)

- Tuti i linguaggi utilizzano parole chiave: `if`, `while`,.....
- Per queste sequenze di caratteri lo scanner deve generare token diversi da quelli degli identificatori
- Come può uno scanner decidere quando una sequenza di caratteri è un identificatore e quando è una parola chiave?
  - Lo scanner può procedere utilizzando il modello degli identificatori e poi cercare il token in una speciale tabella delle parole chiave. (Come potrebbe essere implementata per avere un lookup efficiente?)
  - Si definiscono espressioni regolari per ogni parola riservata, e per gli identificatori e si definisce una priorità fra le varie espressioni regolari.

UPO
UNIVERSITÀ DEL PIEMONTE ORIENTALE

Identificatori e Parole Chiave (2)

Ad esempio

```
If        i f
While     w h i l e
Id        [a-zA-Z] ([a-zA-Z] | [0-9] | _)*
```

Il matching è con la prima espressione regolare (dall'alto in basso!). Notate comunque che dobbiamo fare il maching più lungo, cioè, se ho la stringa `whiler` NON mi devo fermare al riconoscimento di `while`, ma devo andare avanti fino a `whiler` e determinare che questo è un identificatore.

Conclusione dello Scanning

- Cosa accade quando viene raggiunta la fine del file di input?
- In genere (ed è quello che faremo nel nostro compilatore) si crea uno pseudocarattere (il token EOF). (In corrispondenza al -1 ritornato da InputStream.read() viene generato il token EOF.)
- Il token EOF è utile perchè permette al parser di verificare che la fine logica di un programma corrisponde con la fine fisica.
- Molti parser richiedono l'esistenza di un tale token.
- I generatori di scanner (Lex e JFlex) creano automaticamente un token EOF.

UPO
UNIVERSITÀ DEL PIEMONTE ORIENTALE

Recupero dagli errori lessicali

- Una sequenza di caratteri per la quale non esiste un token valido è un errore lessicale.
- Gli errori lessicali non sono comuni ma devono comunque essere gestiti dallo scanner.
- Non è opportuno bloccare il processo di compilazione per errore lessicale.
- Strategie di recupero:
  - Cancellare i caratteri letti fino al momento dell'errore e ricominciare le operazioni di scanning
  - Eliminare il primo carattere letto dallo scanner e riprendere la scansione in corrispondenza del carattere successivo.

  Di solito, un errore lessicale è causato dalla comparsa di qualche carattere illegale, soprattutto all'inizio di un token. In questo caso i due approcci sono equivalenti

UPO
UNIVERSITÀ DEL PIEMONTE ORIENTALE

JFLex: un generatore di scanner

- Questo software, scritto interamente in Java, produce in uscita delle classi Java che implementano i metodi per effettuare l'analisi lessicale di una stringa.
- La classe principale prodotta è una classe per lo Scanner (`Yylex`). Il costruttore ha come parametro il file che vogliamo scannerizzare. La classe contiene i metodi:
  - `Token yylex()` dove `Token` è la classe che vogliamo sia restituita dallo scanner che restituisce il prossimo token.
  - `String yytext()` ritorna la stringa letta per riconoscere il token.

  Per funzionare, JFLex ha bisogno in ingresso un file di specifica, contenente una lista di espressioni regolari definite per il lessico. A ciascuna può essere associata una azione da compiere. Ogni volta che l'input matcha l'espressione regolate è eseguita l'azione (che in genere è la generazione di un Token, ma che per i caratteri di skip non fa niente).

UPO
UNIVERSITÀ DEL PIEMONTE ORIENTALE