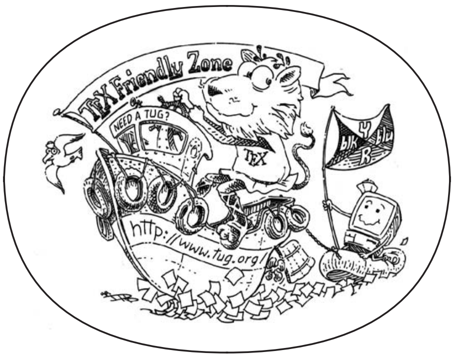
LINGUAGGI E PROGRAMMAZIONE ORIENTATA AGLI OGGETTI

RICCARDO CEREGHINO



Appunti

Settembre 2019 – classicthesis v4.6

1 INTRODUZIONE AGLI ELEMENTI DI UN LINGUAGGIO DI PROGRAMMAZIONE

I motivi della creazione ed utilizzo di un linguaggio di programmazione di alto livello sono: di fornire una descrizione precisa, ovvero una specifica formale; offrire un interpretazione tramite interprete da compilare.

Le caratteristiche principali che categorizzano i linguaggi di programmazione sono la sintassi e la semantica, la quale può essere statica o dinamica.

1.1 LINGUAGGI STATICAMENTE TIPATI

Nei linguaggi tipizzati staticamente il *tipo* di variabile viene stabilito nel codice sorgente, per cui si rende necessario che:

- gli operatori e le assegnazioni devono essere usati coerentemente con il *tipo* dichiarato;
- le variabili siano usate consistentemente rispetto la loro dichiarazione.

I vantaggi della staticità risiedono nella preventiva rilevazione degli errori e nell'efficienza di calcolo risultante dalla coerenza tra codice e compilatore.

1.2 LINGUAGGI DINAMICAMENTE TIPATI

Nei linguaggi di programmazione dinamicamente tipati le variabili sono assegnate ai *tipi* durente l'esecuzione del programma, ne consegue che:

- la semantica statica non è definita;
- un utilizzo incosistenze di variabili, operazioni o assegnazioni generano errori dinamici basati sui tipi.

I linguaggi dinamici per questi motivi risultano essere più semplici ed espressivi.

1.2.1 *Esempi di errori*

Listing 1.1: Errore di sintassi

```
x = ;
```

Un errore sintattico generico a molti linguaggi è un espressione formattata erroneamente.

Listing 1.2: Errore statico

```
int x=0;
if(y<0) x=3; else x="three"
```

In un linguaggio statico come *Java* l'esempio precedente darebbe un errore in quanto una stringa non può essere convertita in un tipo intero.

Listing 1.3: Errore Dinamico

```
x = null;
if(y<0) y=1; else y=x.value;
```

L'esempio, per $y > 0$, darebbe un errore dinamico sia in *Java*, che in un linguaggio dinamico come *Javascript*.

Parte I

SINTASSI

2 STRINGHE

Definizione 1 *Un alfabeto A è un insieme finito non vuoto di simboli.*

Definizione 2 *Sia una stringa in un alfabeto A la successione di simboli in u:*

$$u : [1 \dots n] \to A$$

Sia:

- $[1 \dots n] = m$*, l'intervallo dei numeri naturali tale che:*

  $$1 \leq m \geq n;$$

- *u sia una funzione totale;*
- *n sia la lunghezza di u : $length(u) = n$.*

Definizione 3 *Un programma è una stringa in un alfabeto A.*

2.1 ESEMPIO DI STRINGHE

2.1.1 *Stringa vuota*

$$u : [1 \dots 0] \to A$$

Esiste un unica funzione $u : 0 \to A$

Le notazioni standard di una stringa vuoto sono: ε, λ

2.1.2 *Stringa non vuota*

Si consideri $A = \{'a', \dots, 'z'\} \cup \{'A', \dots, 'Z'\}$, l'alfabeto inglese di lettere minuscole e maiuscole. La funzione $u : [1 \dots 4] \to A$ rappresenta la stringa "*Word*" con:

- $u(1)$ = 'W'
- $u(1)$ = 'o'
- $u(1)$ = 'r'
- $u(1)$ = 'd'

2.1.3 *Concatenazione di stringhe*

Definizione 4

$$length(u \cdot v) = length(u) + length(v)$$

Per ogni $i \in [1 \ldots length(u) + length(v)]$

$$(u \cdot v)(i) = \textbf{\textit{if}}\ i \leq < length(u) \textbf{\textit{then}}\ u(i) \textbf{\textit{else}}\ v(i - length(u))$$

Monoide

La concatenazione è associativa, ma non commutativa.
La stringa vuota è l'identità dell'elemento.

Induzione

La definizione di u^n per induzione su $n \in \mathbb{N}$:
Base: $u^0 = \lambda$
Passo induttivo: $u^{n+1} = u \cdot u^n$ Per cui u^n si concatena con se stesso n volte.

2.1.4 *Insiemi di stringhe*

Definizione 5 *Sia A un alfabeto:*

- A^n = *l'insieme di tutte le stringhe in A con lunghezza n;*
- A^+ = *l'insieme di tutte le stringhe in A con lunghezza maggiore di* 0*;*
- $A^\star$ = *l'insieme di tutte le stringhe in A;*
- $A^+ = \bigcup_{n>0} A^n$*;*
- $A^\star = \bigcup_{n\geq 0} A^n = A^0 \cup A^+$

2.2 LINGUAGGIO FORMALE

Definizione 6 (Nozione sintattica di linguaggio) *Un linguaggio L in un alfabeto A è un sottoinsieme di* $A^\star$

ESEMPIO: L'insieme L_{id} di tutti gli identificatori di variabile:

$$A = \{'a', \ldots, 'z'\} \cup \{'A', \ldots, 'Z'\} \cup \{'0', \ldots, '9'\}$$

$$L_{\text{id}} = \{'a', 'b', \ldots, 'a0', 'a1', \ldots\}$$

2.2.1 *Composizione di operatori tra linguaggi*

Le operazioni possono essere di concatenazione o di unione:

- **Concatenazione:** $L_1 \cdot L_2 = \{u \cdot w | u \in L_1, w \in L_2\}$;
- **Unione:** $L_1 \cup L_2$.

2.2.2 *Intuizione*

Unione

$L = L_1 \cup L_2$: qualsiasi stringa L è una stringa di L_1 o di L_2.

ESEMPIO:

$$L' = \{'a', \ldots, 'z'\} \cup \{'A', \ldots, 'Z'\}$$

Concatenazione

$L = L_1 \cdot L_2$: qualsiasi stringa L è una stringa di L_1, seguita da una stringa di L_2.

ESEMPIO:

$$\{'a', 'ab'\} \cdot \{\lambda, '1'\} = \{'a', 'ab', 'a1', 'ab1'\}$$

$$L_{\mathrm{id}} = L' \cdot A^{\star} \text{ con } A = \{'a', \ldots, 'z'\} \cup \{'A', \ldots, 'Z'\} \cup \{'0', \ldots, '9'\}$$

2.2.3 *Monoide*

La concatenazione è associativa, ma non commutativa.

$A^0 (= \{\lambda\})$ è l'identità dell'elemento; quindi A^0 *non è l'elemento neutro*, l'elemento neuro è $0 = \{\}$.

2.2.4 *Passo induttivo*

L^n è definito per induzione su $n \in \mathbb{N}$: Base: $L^0 = A^0 (= \{\lambda\}$,
Passo induttivo: $L^{n+1} = L \cdot L^n$.

2.2.5 *Operatori + e **

- **Addizione:** $L^+ = \bigcup_{n>0} L^n$;
- **Moltiplicazione:** $\star$ viene chiamata *Kleen star, stella di Kleen*.

$$L^{\star} = \bigcup_{n \geq 0} L^n$$

Sono equivalenti $L^{\star} = L^0 \cup L^+$, $L \cdot L^{\star}$.

Intuizione

- Qualsiasi stringa di L^+ è ottenuta concatenando una o più stringhe di L;
- Qualsiasi stringa di $L^{\star}$ è ottenuta concatenando 0 o più stringhe di L: *Concatenando zero stringhe si ottiene la stringa vuota.*

3 ESPRESSIONI REGOLARI

Le espressioni regolari sono un formalismo comunamente utilizzato per definire linguaggi semplici.

Definizione 7 *La definizione induttiva di un espressione regolare su un alfabeto A:*

BASE:

- 0 *è un espressione regolare di* A*;*
- λ *è un espressione regolare di* A*;*
- *per ogni* $\sigma \in A$*,* σ *è un espressione regolare in* A*.*

PASSO INDUTTIVO:

- *se* e_1 *ed* e_2 *sono espressioni regolare di* A*,*
  allora $e_1|e_2$ *è un espressione regolare di* A*;*
- *se* e_1 *ed* e_2 *sono espressioni regolare di* A*,*
  allora e_1e_2 *è un espressione regolare di* A*;*
- *se* e *è un espressione regolare di* A*,*
  allora $e^\star$ *è un espressione regolarare di* A*.*

ESERCIZIO Scrivere un **REGEX** che esprima i nomi di variabili permessi.

$$L_{id} = (\{'a', \dots, 'z'\} \cup \{'A', \dots, 'Z'\}) \cdot \{'a', \dots, 'z'\} \cup \{'A', \dots, 'Z'\} \cup \{'0', \dots, '9'\}*$$

$$e_{id} = (a| \dots |z|A| \dots |Z)(a| \dots |z|A| \dots |Z|0| \dots |9)*$$

3.1 SEMANTICA

La semantica di un espressione regolare in A è un liguaggio su A:

- $\emptyset \rightsquigarrow$ insieme vuoto;
- $\epsilon \rightsquigarrow \{\epsilon\}$;
- $\sigma \rightsquigarrow \{''\sigma''\}, \forall \sigma \in A$;
- $e_1 \mid e_2 \rightsquigarrow$ unione delle semantiche di e_1 ed e_2;
- $e_1e_2 \rightsquigarrow$ concatenzazione delle semantiche di e_1 ed e_2.

3.2 SINTASSI CONCRETA DELLE ESPRESSIONI REGOLARI

3.2.1 *Precedenza ed associatività*

- la **stella di Kleene** ha priorità sulla concatenzazione e l'unione;
- la concatenazione ha precedemza sull'unione;
- la concatenazione e l'unione sono associative a sinistra.

3.2.2 *Operatori derivati*

- $e+ = ee+$: (una o più volte e);
- ϵ: è rappresentata dalla stringa vuota: $a|\epsilon$ diventa $a|$;
- $e? = |e$: (e è opzionale, ovvero uno o nessuno);
- $[a0B]$: uno qualsiasi dei caratteri nella quadre $(a|0|B)$;
- $[b-d]$: uno qualsiasi dei caratteri nel range tra le quadre $(b|c|d)$;
- $[a0B]|[b-d]$: può essere scritto come $[a0Bb-d]$;
- $[\wedge \dots]$: qualsiasi carattere ad eccezioni di $\dots$ (esempio: $[^a0Bb-d]$ qualsiasi carattere ad eccezione di $a, 0, B, b, c, d$).

3.2.3 *Caratteri speciali in JAVA*

- . rappresenta ogni carattere;
- \ è il carattere di *escape*, per dare un significato speciale ai caratteri regolari, oppure un significato ordinario per caratteri speciali.

Caratteri speciali cui dare un significato ordinario

Esempi:

$$|, *, +, ?, ., , (,), [,], -, \wedge$$

SEMANTICHE:

- $\backslash. \rightsquigarrow \{''.''\}$,
- $\backslash\backslash \rightsquigarrow \{''\backslash''\}$,
- $. \rightsquigarrow \{s \mid s$ ha lunghezza 1, l'insieme di tutti i caratteri$\}$,
- $\backslash \rightsquigarrow$ non è sintatticamente corretto.
- $n \rightsquigarrow \{''n''\}$,
- $\backslash n \rightsquigarrow \{''linefeed''\}$,

Caratteri cui dare un significato speciale

- $\backslash t$: tab;
- $\backslash n$; newline;
- $\backslash s$: qualsiasi spazio vuoto;
- $\backslash S$: qualsiasi spazio non vuoto;
- $\backslash d$: qualsiasi carattere numerico ($[0-9]$);
- $\backslash D$: qualsiasi carattere non numerico ($[\wedge 0-9]$);
- $\backslash w$: qualsiasi parola ($[[a-zA-Z\_0-9]]$);
- $\backslash W$: qualsiasi carattere che non sia una parola ($[\wedge \backslash w]$).

Definizione 8 *Un linguaggio si dice* regolare *se può essere definito da un' espressione regolare.*

3.3 ANALISI LESSICALE

Definizione 9 *Un* lexeme *è una sottostringa considerata come un'unità sintattica.*

Definizione 10 L'analisi lessicale *affronta il problema della decomposizione di una stringa in un* lexeme*.*

Definizione 11 *Un* lexer *o* scanner *è un programma che esegue l'analisi lessicale e genera* lexemes*.*

ESEMPIO IN C: La stringa $''x2 = 042;$ è decomposta nei *lexemes* seguenti:

- $''x2'';$
- $'' = '';$
- $''042'';$
- $'';''.$

3.3.1 *Token*

Un *token* è una nozione di *lexeme* pià astratta; ad un *token* corrisponde sempre un *lexeme*.

In alcuni casi un *token* può mantenere informazioni sulla sematica, come i valori dei numeri.

Un *tokenizer* è un programma che esegue l'analisi lessicale e genera *token*.

ESEMPIO IN C: La stringa $"x2 = 042;"$ è decomposta nei token seguenti:

- *IDENTIFIER*: con il nome $"x2"$;
- *ASSIGN_OP*;
- *INT_NUMBER*: con il valore di 34;
- *STATEMENT_TERMINATOR*.

3.4 LINGUAGGI REGOLARI

Definizione 12 *Un* linguaggio regolare *è un linguaggui definibile con un espressione regolare.*

I linguaggi regolari *possono essere definiti in altre maniere equivalenti:*

- *con una grammatica regolare a destra o sinistra, anche chiamata* lineare*;*
- *con una serie di* automata *non deterministica o deterministica finita (**NFA** o **DFA**).*

3.4.1 *Limitazioni*

I linguaggi regolari sono linguaggi semplici. Esempio:

- il linguaggio degli identificatori;
- il linguaggio dei numeri.

Le espressioni regolari possono definire le unità che costituiscono la sintassi di un linguaggio di programmazione, ma non possono definire la sintassi dell'intero linguaggio.

3.4.2 *Esempi di linguaggi non regolari*

Il linguaggio di espressioni con numeri naturali, addizione binaria e moltiplicazione e parentesi **non possono** essere definiti da un espressione regolare: il problema è posto dalle parentesi, per cui se venissero rimosse, allora il linguaggio sarebbe regolare.

Un altro esempio di linguaggio semplice non regolare:

$$\{"a"^n"b"^n \mid n \in \mathbb{N}\} = \{"", "ab", "aabb", "aaabbb", \ldots\}.$$