Analisi lessicale

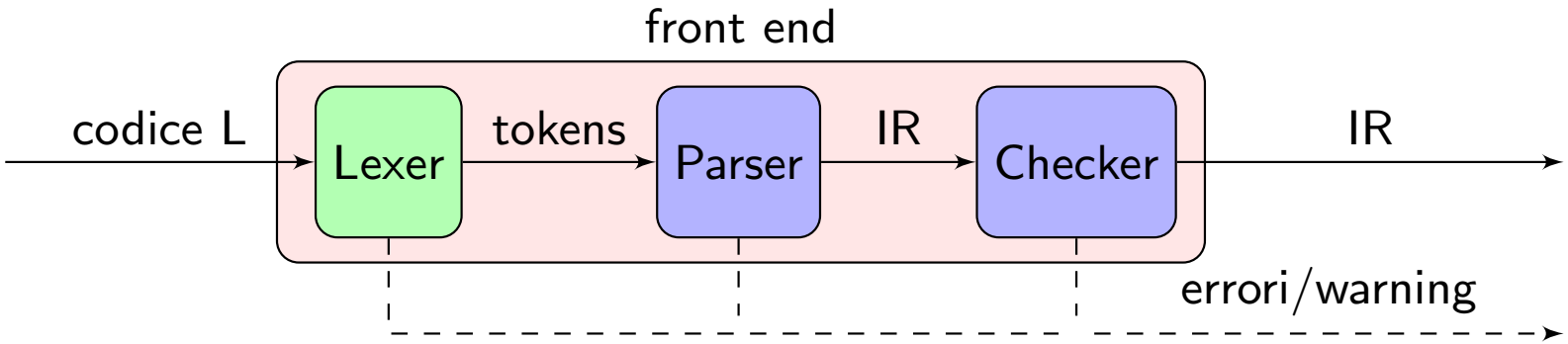
Enea Zaffanella

enea.zaffanella@unipr.it

28 settembre 2020

Linguaggi, interpreti e compilatori
Laurea Magistrale in Scienze informatiche

Il lexer (analizzatore lessicale)



- Input: sequenza di caratteri
- Output: sequenza di *token*
- $\mathit{token} = \langle \mathit{part\_of\_speech}, \mathit{lexeme} \rangle$
- Esempi: $\langle \mathrm{KWD}, \texttt{while} \rangle$, $\langle \mathrm{IDENT}, \texttt{somma} \rangle$, $\langle \mathrm{INT}, 42 \rangle$, $\langle \mathrm{FLOAT}, \texttt{3.1415} \rangle$, $\langle \mathrm{STR}, \texttt{"Hello"} \rangle$, ...

Lexer: specifica vs implementazione

Specifica

- Come definire in modo rigoroso quali sono i token validi?
- Linguaggio adeguato: **RE** (espressioni regolari)
- Comprensibile dal *progettista* (essere umano)

Implementazione

- Fattore critico: efficienza
- Lexer detti anche "scanner"
- Riconoscitore: **DFSA** (automa a stati finiti deterministico)
- Spesso generato automaticamente partendo dalla specifica

Richiami di nozioni base linguaggi formali

- Σ: alfabeto (un insieme **finito** di simboli)
- $\Sigma^\star$: insieme di tutte le stringhe **finite** sull'alfabeto Σ
- $\epsilon \in \Sigma^\star$: la stringa **vuota** (nessun simbolo, lunghezza 0)
- $L \subseteq \Sigma^\star$: linguaggio
  - $\emptyset \equiv \{\}$ è un linguaggio
  - $\{\epsilon\}$ è un linguaggio
  - Σ è un linguaggio (abuso: simbolo $\equiv$ stringa lunga 1)
  - $\Sigma^\star$ è un linguaggio (infinito, se $\Sigma \neq \emptyset$)
- Problema di **decisione** per L (riconoscimento, appartenenza): trovare un **algoritmo** che, data una qualunque stringa $s \in \Sigma^\star$, decida (in tempo finito) se vale $s \in L$.

Le espressoni regolari: sintassi

L'insieme RExpr delle RE (espressioni regolari)

- $\epsilon \in \mathrm{RExpr}$; $a \in \mathrm{RExpr}$, per ogni $a \in \Sigma$
- se $e \in \mathrm{RExpr}$, allora:
  - $(e) \in \mathrm{RExpr}$ (parentesi)
  - $e\star \in \mathrm{RExpr}$ (chiusura di Kleene)
- se $e_1 \in \mathrm{RExpr}$ e $e_2 \in \mathrm{RExpr}$, allora:
  - $e_1 e_2 \in \mathrm{RExpr}$ (concatenazione)
  - $e_1 \mid e_2 \in \mathrm{RExpr}$ (alternanza)

Precedenza operatori

- la chiusura di Kleene ha la precedenza sulla concatenazione
- la concatenazione ha la precedenza sull'alternanza
- esempio: $a \mid a \star b \equiv ((a) \mid ((a) \star b))$

Le espressoni regolari: semantica

$\mathrm{L}\colon \mathrm{RExpr} \to \wp(\Sigma^\star)$ (linguaggio denotato da una RE)

$$
\begin{aligned}
\mathrm{L}(\epsilon) &= \{\epsilon\};\\
\mathrm{L}(a) &= \{a\};\\
\mathrm{L}(e_1 e_2) &= \mathrm{L}(e_1)\,\mathrm{L}(e_2) = \left\{ s_1 s_2 \;\middle|\; s_1 \in \mathrm{L}(e_1), s_2 \in \mathrm{L}(e_2) \right\};\\
\mathrm{L}(e_1 \mid e_2) &= \mathrm{L}(e_1) \cup \mathrm{L}(e_2);\\
\mathrm{L}(e\star) &= \mathrm{L}(e)^\star = \bigcup_{i=0}^{\infty} \mathrm{L}(e)^i = \bigcup_{i=0}^{\infty} \overbrace{\mathrm{L}(e)\cdots\mathrm{L}(e)}^{i};\\
\mathrm{L}((e)) &= \mathrm{L}(e).
\end{aligned}
$$

Abbreviazioni RE

Sintassi

- $e+$ (chiusura positiva), $e?$ (opzionalità)
- $[a_1 \dots a_n]$, $[a_1 - a_2]$ (classe/range di simboli)
- $[\hat{}\, a_1 \dots a_n]$, $[\hat{}\, a_1 - a_2]$ (complemento di classe/range)

Semantica

$$\mathrm{L}(e+) = \cup_{i=1}^{\infty} \mathrm{L}(e)^i; \qquad \mathrm{L}([a_1 \dots a_n]) = \{a_1, \dots, a_n\};$$
$$\mathrm{L}(e?) = \{\epsilon\} \cup \mathrm{L}(e); \qquad \mathrm{L}([a_1 - a_2]) = \{a \in \Sigma \mid a_1 \leq a \leq a_2\};$$
$$\mathrm{L}([\hat{}\, chars]) = \Sigma \setminus \mathrm{L}([chars]).$$

I token dei linguaggi di programmazione

Classificazione

- parole chiave
- identificatori
- costanti letterali (intere, floating point, stringa, ...)
- operatori (matematici, logici, ...)
- "punteggiatura" (parentesi, virgola, punto e virgola, ...)
- commenti (singola linea, multi linea)

Esempi di specifica: keyword

Parole chiave: facile

- `if`, `then`, `else`, `while`, ...
- si usa l'alternanza: `if` | `then` | `else` | `while` ...
- unica categoria sintattica: KEYWORD

Case insensitive (e.g., SQL)

- `select`, `SELECT`, `SeLeCt`, `SELect`, ...
- Noioso e troppo verboso: si usano impostazioni *ad hoc*

Esempi di specifica: identificatori

Identificatori (1)

- [a-zA-Z_][0-9a-zA-Z_]$\star$

Identificatori (2)

- [a-zA-Z_]([0-9] | [a-zA-Z_])$\star$

Identificatori (3)

- DIGIT = [0-9]
- LETTER = [a-zA-Z_]
- {LETTER}({DIGIT} | {LETTER})$\star$

Esempi di specifica: costanti

Costanti intere

- `{DIGIT}+`
- Nota: accetta `000000`, non accetta `-1`

Costanti floating point

- `[+-]?[0-9]+.[0-9]*`
- occorre distinguere iterazione positiva dal carattere +?

Costanti carattere

- `'[^']'`
- come specifichiamo la costante carattere '?

Esempi di specifica: operatori e punteggiatura

Ogni lessema ha la sua categoria lessicale (singoletto)

lessema	categoria	lessema	categoria
(	OPEN_PAREN	)	CLOSE_PAREN
[	OPEN_BRACKET	]	CLOSE_BRACKET
{	OPEN_BRACE	}	CLOSE_BRACE
+	PLUS	-	MINUS
+=	PLUS_ASSIGN	-=	MINUS_ASSIGN
:	COLON	::	SCOPE
<	LESS_THAN	<<	SHIFT_LEFT
>	GREATER_THAN	>>	SHIFT_RIGHT
.	DOT	...	ELLIPSIS
...	...		

Esempi di specifica: commenti

Commento (singola linea) del C++

- `//[^\n]*\n`

Commento (singola linea) di SQL

- `--[^\n]*\n`

Commento multilinea (C/C++/Java/SQL/...)

- `/\*([^*]|\*+[^/*])*\*+/`
- Chiarissimo, vero?
- (Vedremo un modo migliore per ottenere lo stesso effetto)

Risoluzione ambiguità

Più RE possono accettare (parti de-) lo stesso input

- Preferenza al **lessema più lungo**
  - `forwhile` è un unico IDENT
    (non sono le due KEYWORD `for` e `while`)
  - >> è un unico SHIFT_RIGHT (non due GREATER_THAN)

A parità di lunghezza?

- Si stabilisce un **ordine di priorità** tra le RE
  - `for` e `while` sono KEYWORD (non IDENT)

Nota: rispettare le regole può essere "noioso"

Annidamento liste argomenti template nel C++

- Per lo standard C++03 e precedenti:
  - corretto: `std::vector<std::list<int> >`
  - errore (sintattico): `std::vector<std::list<int>>`
- Per lo standard C++11 e successivi:
  - corretto: `std::vector<std::list<int> >`
  - corretto: `std::vector<std::list<int>>`

Tra il dire e il fare . . .

Verso una esercitazione . . .

- Goal: un lexer per una **porzione** (significativa) di un linguaggio di programmazione
- Focus: **specifica** (non implementazione)
- Prerequisito: sapere usare un **generatore** di lexer
- Lo strumento scelto: `flex`