編譯器設計

Languages and Their Representations

Alphabets and Languages

- Webster defines a *language* as
  - "the body of words and methods of combining words used and understood by a considerable community" → 對compiler來說不夠嚴緊
- The definition is not precise
  - A formal language will be defined
- An *alphabet*:
  - any finite set of symbols, e.g.
    - Latin alphabet {A, B, C, …, Z}
    - Greek alphabet {α, β, γ, …, ω}
    - binary alphabet {0, 1}

Alphabets and Languages

Alphabet
↓
Sentence
↓
language.

- A *sentence* over an alphabet
  - any string of finite length composed of symbols from the alphabet
  - Synonyms for sentence are *string* and *word*
- The empty sentence ϵ 空字串 epsilon
  - the sentence consisting of no symbols
- If V is an alphabet, then
  - V^* denotes the set of all sentences composed of symbols of V, including the empty sentence
  - $V^+ = V^* - \{\epsilon\}$ $*$ closure V^* 有 ε V^+ 無 ε
  - If $V = \{0,1\}$, then

    $V^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$

    $V^+ = \{0, 1, 00, 01, 10, 11, \ldots\}$

Alphabets and Languages

空字串 → language?

- A *language* ⇒ 句子的集合
  - any set of sentences over an alphabet
  - e.g. {0, 1} is a language acquiescence 默許
- Three questions are raised
  - How do we represent a language?
    - It's simple if the language is finite
    - How to represent an infinite language with a finite representation
  - Does there exist a finite representation for every language? It's unproved. But the closest answer is "NO".
  - What can be said about the structures of those languages for which there exist finite representation? finite representation infinite language.

Representations of Languages

- Two ways to represent a language
  - To give an algorithm which determines if a sentence is in the language or not
    - To give a procedure which halts with the answer "yes" for sentences in the language and either does not terminate or else halts with the answer "no" for sentences not in the language
  - To give a grammar that generates sentences in the language

Two ways:　　找到辨認/產生的方法

1. Recognition

sentence → $\boxed{\text{parser algorithm}}$ → yes / → no

2. Generation / Production

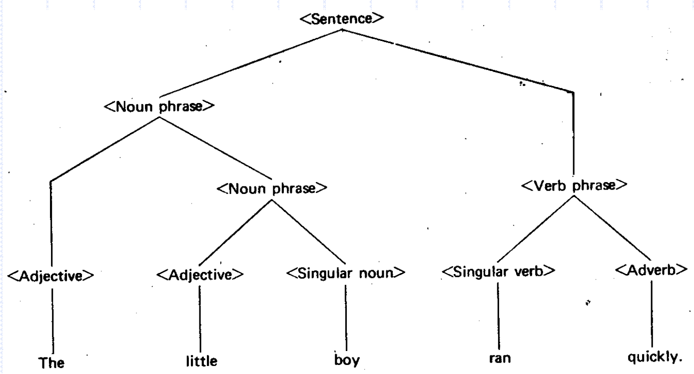
grammar → sentence

Grammars

Produced with grammar.

- Example: "The little boy ran quickly"

<sentence> →<noun phrase><verb phrase>
<noun phrase> →<adjective><noun phrase>
<noun phrase> →<adjective><noun>
<verb phrase> →<verb><adverb>
<adjective> → The
<adjective> → little
<noun> → boy
<verb> → ran
<adverb> → quickly



Formal Notation of a Grammar

- Four concepts
  - Nonterminals (or Variables)
    - e.g. <sentence>, <adjective>, <verb phrase>, etc.
  - Terminals
    - e.g. words such as The, little, boy, etc.
  - Productions (set of grammar)

    relationships between strings of variables and terminals
    - e.g. <sentence>→<noun phrase><verb phrase>
  - Start Symbol

    distinguished symbol that generates exactly those strings of terminals that are deemed in the language
    - e.g. <sentence>

Formal Notation of a Grammar

- A grammar G can be denoted by (V_N, V_T, P, S)
  - V_N: nonterminals e.g. <sentence>, <adjective>, ...
  - V_T: terminals ($V_N \cap V_T = \phi$, $V_N \cup V_T = V$) e.g. the, little, boy...

    nonterminal terminal
  - P : productions
    - $\alpha \rightarrow \beta \in P$ e.g. <sentence> → <noun phrase><verb phrase>
  - S : start symbol

    <sentence>

Derivation

◆ Derivation by a production

- If $\alpha \rightarrow \beta \in P$ and $\gamma, \delta \in V$, then

  $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$

- i.e. $\gamma\alpha\delta$ directly derives $\gamma\beta\delta$

γ α δ

$\boxed{\text{<n. phrase>}}$ <v. phrase>

Derivation

$\boxed{\text{<adj><n. phrase>}}$ <v. phrase>

γ β δ

V_T^* (which is comprised with terminals)

◆ Derivation by productions

- If $\alpha_1, \alpha_2, \ldots, \alpha_m$ are strings in V^*, and

  $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_m$

  then we say

  $\alpha_1 \overset{*}{\Rightarrow} \alpha_m$

sentential form

$\Rightarrow$ partly V_T, partly V_N

Derivation

◆ The language generated by G is defined

$$L(G) = \{w \mid w \in V_T^* \wedge \underset{\text{Start Symbol}}{S} \overset{*}{\Rightarrow} w\}$$ $w \notin V_N$

- That is, a string is in $L(G)$ if
  - The string consists solely of terminals
  - The string can be derived from S
- Grammars G_1 and G_2 are *equivalent* if
  - $L(G_1) = L(G_2)$
- Example $G = (V_N, V_T, P, S)$
  $V_N = \{S\}$, $V_T = \{0, 1\}$, $P = \{S \rightarrow 0S1, S \rightarrow 01\}$

  $$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 0^3S1^3 \Rightarrow \ldots \Rightarrow 0^{n-1}S1^{n-1} \Rightarrow 0^n1^n$$

  $\therefore$ L(G) = $\{0^n1^n\}$ $n \geq N^+$

- A string of terminals and nonterminals α is called a *sentential form* if $S \overset{*}{\Rightarrow} \alpha$

* arbitary number

Types of Grammars

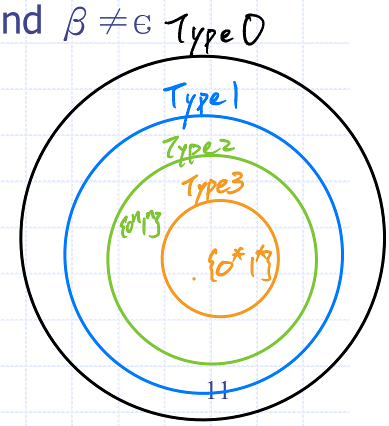
- Let $G = (V_N, V_T, P, S)$ be a grammar
  - nature language
  - Type 0 grammar no limitation on V_T and V_N
  - Type 1 grammar (context-sensitive grammar) 依上下文 (上下文)
    - For every production $\alpha \rightarrow \beta$ in P, $|\alpha| \leqq |\beta|$ length of α ≤ length of β
    - e.g. P = {S→aSBC, S→aBC, CB→BC, aB→ab, bB→bb, bC→bc, cC→cc} 大寫：N，小寫：T
  - Type 2 grammar (context-free grammar)
    - For every production $\alpha \rightarrow \beta$ in P, $|\alpha| = 1$ and $\beta \neq \epsilon$
    - e.g. P = {S→0S1, S→01}

    Used in most programming languages today
  - Type 3 grammar (regular grammar)
    - Every production in P is of the form
      - A →aB, or N → T N or
      - A →a N → T

        only two ways

Type 2 can be recognized by Pushdown Automota

Type 3 — by Finite Automota



Type3 語言的 set
不一定會比 Type2 的小