7.4.4 CYK[1] 알고리즘

(개요) Context-Free **언어의 membership**을 판정하는 $O(n^3)$ 알고리즘이다.

(정리) 주어진 문법의 $G = (N, T, P, S)$와 문자열 $a_1 a_2 \cdots a_n$에 관하여 넌 터미널의 집합 $X_{i,j}$를 아래와 같이 정의하면,

$$1 \le \forall i \le \forall j \le n: \; X_{i,j} = \{A \in N | A \Rightarrow^* a_i \cdots a_j\},$$

문자열의 membership 문제는 시작문자 S의 $X_{1,n}$에 포함문제로 바뀐다.

즉, $a_1 a_2 \cdots a_n \in L(G) \Leftrightarrow S \in X_{1,n}$.

그러나 $X_{1,n}$을 구하기 위하여, $1 \le \forall i \le \forall j \le n$에 대하여 $X_{i,j}$를 구하여야하므로 문자열의 언어 membership 문제는 $X_{i,j}$의 계산문제로 바뀐다.

(기본) $1 \le \forall i \le n$:

$$X_{i,i} = \{A \in N | \; A \to a \in P, \; a = a_i\}.$$

(반복) $1 \le \forall i < \forall j \le n$:

$$X_{i,j} = \{A \in N | \; A \to BC \in P, \; i \le \forall k \le j, \; B \in X_{i,k}, \; C \in X_{k+1,j}\}$$

(**기본**)은 $1 \le \forall i \le n$에 대하여 $A \to a_i$형태의 규칙이 있으면, A를 $X_{i,i}$에 추가하고, (**반복**)은 $1 \le \forall i < \forall j \le n$에 대하여 $A \to BC$인 규칙이 있으면, $i \le \forall k \le j$인 k에 대하여 $B \in X_{i,k}$이고 $C \in X_{k+1,j}$이면 A를 $X_{i,j}$에 추가한다. CYK 알고리즘의 시간 복잡도는 $O(n^3)$이다.

(**반복**)에서 $X_{i,j}$를 구하는 논리식은 전혀 문제가 없다. 그러나 이를 아래와 같이 **for 반복구조**(loop)로 써보면 문제가 생긴다.

(알고리즘) $1 \le \forall i < \forall j \le n$: $X_{i,j}$의 **잘못 된** 계산

$1 \le \forall i < n$:

  $i < \forall j \le n$: $X_{i,j}$을 계산한다.

    $i \le \forall k \le j$:

      $A \to BC \in P \; \wedge \; B \in X_{i,k} \; \wedge \; C \in X_{k+1,j} \Rightarrow A \in X_{i,j}$.

For ∀ 반복구조에서는 반복변수(index variable) i, j, k[2]에는 순서가 있으므로, $X_{i,j}$을 계산할 때 $k \le j$이어서 아직 계산하지 못한 $X_{k+1,j}$가 C를 포함하는가를 살펴보아야 한다.

이것은 논리(혹은 수학; what)에는 순서가 정해져있지 않으나(non-deterministic), 알고리즘(how)에는 순서가 정해져있기(deterministic) 때문이다.

---

1) J. Coke, D. Younger, T. Kasami, 세 사람의 성의 첫째 알파벳을 땄다.

2) 반복변수 k(innermost loop)가 가장 먼저 변한다.

(알고리즘) $1 \le \forall i < \forall j \le n$: $X_{i,j}$의 **올바른** 계산

$1 \le \forall m < n$:

$m \le \forall i < n$: $X_{i,i+m}$을 계산한다.

$i \le \forall k \le i+m$:

$A \to BC \in P \ \wedge \ B \in X_{i,k} \ \wedge \ C \in X_{k+1,i+m} \Rightarrow A \in X_{i,i+m}$.

$1 \le \forall (j-i) < n$:

$j-i \le \forall i < n$: $X_{i,j}$을 계산한다.

$i \le \forall k \le j$:

$A \to BC \in P \ \wedge \ B \in X_{i,k} \ \wedge \ C \in X_{k+1,j} \Rightarrow A \in X_{i,j}$.

위의 알고리즘에 For ∀ 반복구조에서는 가장 천천히 변하는(outermost loop) 변수m을 $X_{i,j}$의 $j-i$값으로 하였으므로, $X_{i,i+m}$을 계산할 때, $i \le k \le i+m$이므로 살펴보는 $X_{i,k}$와 $X_{k+1,i+m}$가 **모두 이미** 계산되어있다.

CYK 알고리즘은 주어진 문자열 $a_1 a_2 \cdots a_n$과 주어진 context-free문법(CFG) G에 대하여 $a_1 a_2 \cdots a_n \in L(G)$인가 아닌가를 판정하는 context-free언어의 membership문제의 해결방법에 하나이다. CYK 알고리즘은 문자열 $a_1 a_2 \cdots a_n$에 따라서 **파서를 따로** 만들어야 한다는 단점이 있고, 시간 복잡도는 $O(n^3)$이다.

반면에 6장에서 공부한 좌파서(left parser)나 우파서(right parser)는, 문법 G가 정해지면, 문자열에 관계없이 **파서가 변하지 않는다**는 장점이 있고, 문장의 파스(parse)가 $\pi \in P^*$라면, 유도(derivation)의 길이가 $n+|\pi|$이므로 시간 복잡도는 $O(n)$이다.

좌파서나 우파서는 모두 non-deterministic 알고리즘이므로 10장에서 배울 **NP**문제다. 그러나 CYK는 알고리즘은 deterministic 복잡도는 $O(n^3)$로 늘었으나, CFG 파싱(parsing) 문제를 **NP**에서 **P**로 줄였다는 의미를 가진다[3].

3) CFG 파싱이 **NP**-complete 아니라는 증명이고, **NP**중에 **P**인 문제도 있다는 좋은 예이다.