

# Strong LL(k) parsing

## Computation of $First_k$ .

Let  $k \geq 0$ ,  $A$  be a set and  $\alpha, \beta \in A^*$ . Then we define

$$\alpha \oplus_k \beta = k:\alpha\beta.$$

$\therefore \oplus_k: A^* \times A^* \rightarrow A^{\leq k}$       binary operation on  $A$ .

$$\begin{aligned} \text{where } A^{\leq k} &= A^0 \cup A^1 \cup A^2 \dots \cup A^k = \{\epsilon\} \cup A \cup A^2 \dots \cup A^k \\ &\leftrightarrow \$^k \cup A\$^{k-1} \cup A^2\$^{k-2} \dots \cup A^k. \end{aligned}$$

We define languages and  $First_k$  of  $\alpha \in (N \cup \Sigma)^*$  and  $K \subseteq (N \cup \Sigma)^*$ .

$$L(\alpha) = \{w \in \Sigma^* \mid \alpha \Rightarrow^* w\}$$

$$First_k(\alpha) = k:L(\alpha)$$

$$L(K) = \{w \in \Sigma^* \mid \alpha \in K, \alpha \Rightarrow^* w\}$$

$$First_k(K) = k:L(K)$$

## *Divide and conquer on the computation of $First_k$ .*

1.  $First_k: 2^{\Sigma^*} \rightarrow 2^{\Sigma^{\leq k}}$ . Let  $L_1, L_2 \subseteq \Sigma^*$ . Then

$$First_k(L_1L_2) = First_k(L_1) \oplus_k First_k(L_2) = k:L_1L_2.$$

2.  $First_k: (N \cup \Sigma)^* \rightarrow 2^{\Sigma^{\leq k}}$ . Let  $\alpha = X_1 \dots X_n$ ,  $1 \leq \forall i \leq n$ ,  $X_i \in N \cup \Sigma$ .

$$\begin{aligned} First_k(\alpha) &= First_k(X_1 \dots X_n) \\ &= First_k(X_1) \oplus_k First_k(X_2) \oplus_k \dots \oplus_k First_k(X_n). \end{aligned}$$

3.  $First_k: N \cup \Sigma \rightarrow 2^{\Sigma^{\leq k}}$ .

$$First_k(a) = \{a\}, \text{ if } a \in \Sigma. \quad \text{basis}$$

$$First_k(A) = \{x \in First_k(\alpha) \mid A \rightarrow \alpha \in P\} \quad \text{recursion}$$

We also define  $Follow_k: N \rightarrow 2^{\Sigma^{\leq k}}$ . Let  $A \in N$ . Then

$$Follow_k(A) = \{k:z \in \Sigma^* \mid S \Rightarrow^* xAz\}$$

**Guess-verify parser**  $M = (N \cup \Sigma, \Sigma, \Gamma, S, \{\epsilon\}, \$, /)$   
 for  $G = (N, \Sigma, P, S)$

$\forall A \rightarrow \omega \in P, \quad A / \rightarrow \omega^R / \in \Gamma, \quad \text{guess } A \text{ as } \alpha.$   
 $\forall a \in \Sigma, \quad a / a \rightarrow / \in \Gamma, \quad \text{verify } a \in \Sigma.$

$.S \quad \Rightarrow_{lm}^* \quad x.A\gamma \quad \Rightarrow_{lm} \quad x.\beta\gamma \quad \Rightarrow_{lm}^* \quad xy.\gamma \quad \Rightarrow_{lm}^* \quad xyz$   
 $\$S / xyz\$ \Rightarrow^* \$\gamma^R A / yz\$ \Rightarrow \$\gamma^R \beta^R / yz\$ \Rightarrow^* \$\gamma^R / z\$ \Rightarrow^* \$ | \$.$

### **Nondeterminism**

If  $A \rightarrow \alpha / \beta \in P, \alpha \neq \beta.$   
 guess  $A$  as  $\alpha$  or  $\beta$       **nondeterministic**

Adding  $k$ -lookahead symbols  $y$  where  $|y| \in \Sigma^{\leq k}.$

$\forall A \rightarrow \alpha \in P, \quad A / \mathbf{y} \rightarrow \alpha^R / \mathbf{y} \in \Gamma, \quad \text{adding lookahead } \mathbf{y} \in \Sigma^{\leq k}.$

where  $y \in \text{First}_k(\alpha) \oplus_k \text{Follow}_k(A)$

**Guess**  $A$  as  $\alpha$  on lookahead  $y$  only.

We define  $\text{LaSLL}_k(A \rightarrow \alpha) = \text{First}_k(\alpha) \oplus_k \text{Follow}_k(A)$ .

Guess and verify parser with  $k$ -lookahead symbols,  $\text{LaSLL}_k(A \rightarrow \alpha)$ ,  
is called a **strong LL(k) parser**.

If  $\text{LaSLL}_k(A \rightarrow \alpha) \cap \text{LaSLL}_k(A \rightarrow \beta) = \emptyset$ .

**deterministic guess**  $A$  as  $\alpha$  or  $\beta$

on  $\text{LaSLL}_k(A \rightarrow \alpha)$  or  $\text{LaSLL}_k(A \rightarrow \beta)$ , respectively.

$G$  is **strong LL(k) grammar**, if

$\forall A \rightarrow \alpha / \beta \in P, \alpha \neq \beta, \text{LaSLL}_k(A \rightarrow \alpha) \cap \text{LaSLL}_k(A \rightarrow \beta) = \emptyset$ .

Strong LL(k) parser is **deterministic**, if and only if  $G$  is strong LL(k).

**Theorem**  $G$  is not LL(k), if  $G$  is left recursive.

$$A \rightarrow A\alpha / \beta$$

$A$  is left recursive.

**Proof**  $First_k(A\alpha) \supseteq First_k(A)$   
 $\supseteq First_k(\beta)$

$$\therefore LaSLL_k(A \rightarrow A\alpha) \cap LaSLL_k(A \rightarrow \beta) \supseteq First_k(\beta) \neq \emptyset$$

**Removal of left recursion**

$$A \rightarrow A\alpha / \beta$$

$$A \Rightarrow^* \beta\alpha^*$$

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' / \varepsilon$$

**Left factoring**

$$A \rightarrow \alpha\beta / \alpha\gamma$$

$$A \rightarrow \alpha A'$$

$$A' \rightarrow \beta / \gamma$$

**LL(1) parsing**

Let  $T_1, T_2 \subseteq \Sigma^*$ . Consider

$$\begin{aligned} T_1 \oplus_1 T_2 &= \text{First}_1(T_1 T_2) \\ &= \text{First}_1(T_1), \text{ if } \varepsilon \notin T_1. \\ &= \text{First}_1(T_1) \cup \text{First}_1(T_2), \text{ if } \varepsilon \in T_1, \varepsilon \in T_2, \\ &= \text{First}_1(T_1) \cup \text{First}_1(T_2) - \{\varepsilon\}, \text{ if } \varepsilon \in T_1, \varepsilon \notin T_2 \text{ (otherwise)}. \end{aligned}$$

$\text{First}_1: N \cup \Sigma \rightarrow 2^{\Sigma^{\leq 1}} = 2^{\{\varepsilon\} \cup \Sigma} = \{\varepsilon\} \cup 2^\Sigma$      $\text{First}_1 \leftrightarrow \text{nullable} \cup \text{First}$ .

$\text{nullable}: N \cup \Sigma \rightarrow \{\text{true}, \text{false}\}$

$\text{nullable}(A) = \text{true}$ , iff  $A \Rightarrow^* \varepsilon$ .

$\text{First}: N \cup \Sigma \rightarrow 2^\Sigma$ .

$\text{First}(a) = \{a\}$ , if  $a \in \Sigma$ .

$\text{First}(A) = \{a \in \Sigma \mid A \Rightarrow^* ax, x \in \Sigma^*\}$

*Recursive formula for First(A),  $A \in N$ .*

$$\text{First}(A) \supseteq \{a \in \Sigma / A \rightarrow \alpha a \beta \in P, \alpha \Rightarrow^* \varepsilon\} \quad \text{basis}$$

$$\underline{\text{First}}(A) \supseteq \{a \in \underline{\text{First}}(B) / A \rightarrow \alpha B \beta \in P, \alpha \Rightarrow^* \varepsilon\} \quad \text{recursion}$$

$$\text{Follow}_1: N \rightarrow 2^{\Sigma^{\leq l}} = 2^{\{\varepsilon\} \cup \Sigma} = \{\varepsilon\} \cup 2^\Sigma.$$

$$\text{Follow}_1: N \rightarrow \{\varepsilon\} \cup 2^\Sigma.$$

$$\text{Follow}_1(A) = \{1:y \in \{\varepsilon\} \cup \Sigma / S \Rightarrow^* xAy, x, y \in \Sigma^*\}$$

$$\varepsilon \in \text{Follow}_1(S).$$

*(We may add new **end marker** \$ in  $\Sigma$  and  $\varepsilon \leftrightarrow \$$ .)*

*Recursive formula for Follow(A)*

$$\text{Follow}(S) \supseteq \{\$\} \quad \text{basis}$$

$$\text{Follow}(A) \supseteq \{a \in \text{First}(\beta) / B \rightarrow \alpha A \beta \in P\} \quad \text{basis}$$

$$\underline{\text{Follow}}(A) \supseteq \{a \in \underline{\text{Follow}}(B) / B \rightarrow \alpha A \beta \in P, \beta \Rightarrow^* \varepsilon\} \text{recursion}$$

We define  $l \subseteq N \times N$  and  $r \subseteq N \times N$ .

$A l B$ , if  $A \rightarrow \alpha B \beta \in P$ ,  $\alpha \Rightarrow^* \varepsilon$ , and (left dependency relation)

$A r B$ , if  $B \rightarrow \alpha A \beta \in P$ ,  $\beta \Rightarrow^* \varepsilon$ . (right dependency relation)

Formula for  $First(A)$  and  $Follow(A)$

$$\begin{aligned} First(A) &\supseteq \{a \in \Sigma / A \rightarrow \alpha a \beta \in P, \alpha \Rightarrow^* \varepsilon\} \\ &= \{a \in \Sigma / A l a\} \end{aligned}$$

**basis**

$$\underline{First}(A) \supseteq \{a \in \underline{First}(B) / A l B\}$$

**recursion**

$$Follow(S) \supseteq \{\$ \}$$

**basis**

$$Follow(A) \supseteq \{a \in First(\beta) / B \rightarrow \alpha A \beta \in P\}$$

**basis**

$$\underline{Follow}(A) \supseteq \{a \in \underline{Follow}(B) / A r B\}$$

**recursion**



Let  $A$  and  $B$  be two sets and  $x, y \in A$ ,  $R \subseteq A \times A$ , and  $f, g: A \rightarrow 2^B$ .

Compute  $f$  for given  $R$  and  $g$ ,

$$f(x) \supseteq g(x)$$

*basis*

$$f(x) \supseteq f(y) \text{ where } x R y$$

*recursion*

$$\therefore f(x) \supseteq g(x) \cup \cup_{xRy} f(y)$$

Nothing else is in  $f(x)$

*fixed point*

$$f(x) =_S g(x) \cup \cup_{xRy} f(y)$$

Then  $f(x) =_S \{b \in g(y) \mid x R^* y\}$

*iteration*

Reachable vertices in the graph  $R$ .

*depth-first search*

*topological order*

**Algorithm** Compute  $f(x)$  with  $g(x)$  and  $R$ .

**input**  $g: A \rightarrow 2^B; R \subseteq A \times A$ .

**output:**  $f: A \rightarrow 2^B$ .

**var**  $S$ : stack of  $A$ ;  $N: A \rightarrow \text{Depth}$ .

**function**  $\text{Trav}(x: A, d: \text{Depth})$ ;

**push**  $x$  onto  $S$ ;  $N(x) = d$ ;

$f(x) := g(x)$ ;                     $|A|$

**for**  $y \in A$  where  $x R y$  **do**

**if**  $(N(y) = 0)$  **then**  $\text{Trav}(y, d+1)$  **fi**;

$N(x) = \min(N(x), N(y))$ ;

$f(x) := f(x) \cup f(y)$  **od**     $|R|$

**if**  $(N(x) = d)$  **then repeat**

$y := \text{pop of } S$ ;  $N(y) := \text{Infinity}$ ;

$f(y) := f(x)$  **until**  $(y = x)$  **fi**

**end function**  $\text{Trav}$

**for**  $x \in A$  **do**  $N(x) := 0$ ;

$f(x) := \emptyset$  **od**;

**for**  $x \in A$  where  $(N(x) = 0)$  **do**  $\text{Trav}(x, 1)$  **od**

**LL(1) analysis.**

1. Remove *useless* symbols and productions, if any.
2. Remove *left recursion*, and do *left factoring*, if any.
3.  $\forall A \in N$ , compute *nullable*(A).
4.  $\forall A \in N$ , compute *l*-relation and *initial* First(A) in **basis** using *nullable*.
5.  $\forall A \in N$ , compute **First**(A) using *l*-relation and *initial* First(A) by traversing *l*-graph.
4.  $\forall A \in N$ , compute *r*-relation and *initial* Follow(A) in **basis** using *nullable* and First( $\beta$ ).
5.  $\forall A \in N$ , compute **Follow**(A) using *r*-relation and *initial* Follow(A) in **basis** by traversing *r*-graph.
6.  $\forall A \rightarrow \alpha \in P$ , compute **LaSLL**<sub>1</sub>(A  $\rightarrow$   $\alpha$ ) using **First** and **Follow**.
7.  $\forall A \rightarrow \alpha / \beta \in P$ ,  $\alpha \neq \beta$ , if **LaSLL**<sub>1</sub>(A  $\rightarrow$   $\alpha$ )  $\cap$  **LaSLL**<sub>1</sub>(A  $\rightarrow$   $\beta$ ) =  $\emptyset$ , LL(1); otherwise not LL(1).

**Example Expression grammar**

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow a \mid ( E )$$

**Removal of left recursion**

	<i>nullable</i>	<i>init.</i>	<i>First</i>	<i>init.</i>	<i>Follow</i>
$E \rightarrow T E'$	<i>no</i>		$a, ($	$\$, )$	$\$, )$
$E' \rightarrow + T E' \mid \varepsilon$	<i>yes</i>	$+$	$+$		$\$, )$
$T \rightarrow F T'$	<i>no</i>		$a, ($	$+$	$\$, ), +$
$T' \rightarrow * F T' \mid \varepsilon$	<i>yes</i>	$*$	$*$		$\$, ), +$
$F \rightarrow a \mid ( E )$	<i>no</i>	$a, ($	$a, ($	$*$	$\$, ), +, *$
$LaSLL_1(E' \rightarrow + T E')$					$LaSLL_1(E' \rightarrow \varepsilon) = \{\$, )\}$
$LaSLL_1(T' \rightarrow * F T')$					$LaSLL_1(T' \rightarrow \varepsilon) = \{\$, ), +\}$
$LaSLL_1(F \rightarrow a)$					$LaSLL_1(F \rightarrow ( E )) = \{($

$\therefore LL(1).$

$SLL_1PT: N \times (\Sigma \cup \{\$\}) \rightarrow 2^P.$

$SLL_1PT(A, a) = \{A \rightarrow \alpha \mid a \in LaSLL_1(A \rightarrow \alpha)\}$

**Fact**  $G$  is  $LL(1)$ , if  $\forall A \in N, \forall a \in \Sigma, |SLL_1PT(A, a)| \leq 1.$

$SLL_1PT$	$a$	$+$	$*$	$($	$)$	$\$$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$		
$E'$	$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$	
$T$	$T \rightarrow FT'$			$T \rightarrow FT'$		
$T'$	$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$	
$F$	$F \rightarrow a$			$F \rightarrow (E)$		

**Algorithm LL(1) parsing**

**input**  $SLL_1PT: N \times (\Sigma \cup \{\$\}) \rightarrow P$ ; **input string**  $x \in \Sigma^*$ .

**var** *Stack*: *stack of*  $N \cup \Sigma$ ;  $a \in \Sigma$ ;  $y \in \Sigma^*$ .

**push**  $S$  **onto** *Stack*;  $a := 1:x$ ;  $y := x\$:|x|-1$

**repeat**

$X :=$  **pop of** *Stack*;

**if**  $(X \in N) \Rightarrow$

**if**  $(SLL_1PT(X, a) = X \rightarrow \alpha) \Rightarrow$  **push**  $\alpha^R$  **onto** *Stack*;

*Make subtree with root*  $X$  *and*  $\alpha$ 's *as children.*

$| SLL_1PT(X, a) = \emptyset \Rightarrow$  **syntax error** **fi**

$| (X \in \Sigma) \Rightarrow$  **if**  $(X=a) \Rightarrow a := 1:y$ ;  $y := y:|y|-1$

$| (X \neq a) \Rightarrow$  **syntax error** **fi**

**fi**

**until** (*Stack empty*); **if**  $(y = \$) \Rightarrow$  *O.K.*  $| (y \neq \$) \Rightarrow$  **syntax error** **fi**

*Recursive descent parser*

**function** *pE*: **if** (*i* = 'a') **or** (*i* = '(')  $\Rightarrow$  *pT*; *pE*'

    | **otherwise**  $\Rightarrow$  *syntax error fi*

**function** *pE'*: **if** (*i* = '+')  $\Rightarrow$  *verify*('+'); *pT*; *pE*'

    | (*i* = ')') **or** (*i* = '\$')  $\Rightarrow$  *skip*

    | **otherwise**  $\Rightarrow$  *syntax error fi*

**function** *pT*: **if** (*i* = 'a') **or** (*i* = '(')  $\Rightarrow$  *pF*; *pT*'

    | **otherwise**  $\Rightarrow$  *syntax error fi*

**function** *pT'*: **if** (*i* = '\*')  $\Rightarrow$  *verify*('\*'); *pF*; *pT*'

    | (*i* = '+') **or** (*i* = ')') **or** (*i* = '\$')  $\Rightarrow$  *skip*

    | **otherwise**  $\Rightarrow$  *syntax error fi*

**function** *pF*: **if** (*i* = 'a')  $\Rightarrow$  *verify*('a')

    | (*i* = '(')  $\Rightarrow$  *verify*('('); *pE*; *verify*(')')

    | **otherwise**  $\Rightarrow$  *syntax error fi*

*i* := 1:*x*; *y* := *x*\$:/*x*|-1; *pE*; **if** (*y* = '\$')  $\Rightarrow$  *O.K.* | (*y*  $\neq$  '\$')  $\Rightarrow$  *syntax error fi*