Strong LL(k) parsing

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

$\forall A \to \omega \in P, \quad A \mid \to \omega^R \mid \in \Gamma, \quad$ ***guess*** A *as* α.

$\forall a \in \Sigma, \quad a \mid a \to \mid \in \Gamma, \quad$ ***verify*** $a \in \Sigma$.

$$.S \Rightarrow_{lm}^* x.A\gamma \Rightarrow_{lm} x.\beta\gamma \Rightarrow_{lm}^* xy.\gamma \Rightarrow_{lm}^* xyz$$

$$\$S \mid xyz\$ \Rightarrow^* \$\gamma^R A \mid yz\$ \Rightarrow \$\gamma^R\beta^R \mid yz\$ \Rightarrow^* \$\gamma^R \mid z\$ \Rightarrow^* \$ \mid \$.$$

Nondeterminism

If $A \to \beta_1 \mid \beta_2 \in P$, $\beta_1 \neq \beta_2$.

guess A as α *or* β ***nondeterministic***

Consider two derivations

$$.S \Rightarrow_{lm}^{*} x.A\gamma \Rightarrow_{lm} x.\beta_1\gamma \Rightarrow_{lm}^{*} xy_1.\gamma \Rightarrow_{lm}^{*} xy_1z.$$

$$.S \Rightarrow_{lm}^{*} x.A\gamma \Rightarrow_{lm} x.\beta_2\gamma \Rightarrow_{lm}^{*} xy_2.\gamma \Rightarrow_{lm}^{*} xy_2z.$$

*Adding k-lookahead(**finite**) strings* $k{:}y_1z$ *and* $k{:}y_2z \in \Sigma^{\leq k}$.

$A \to \beta_1 \in P$, $A \mid k{:}y_1z \to \beta_1{}^R \mid k{:}y_1z \in \Gamma$ *and*

$A \to \beta_2 \in P$, $A \mid k{:}y_2z \to \beta_2{}^R \mid k{:}y_2z \in \Gamma$.

*The guess and verify parser is **deterministic**, if*

$\{k{:}y_1z\} \cap \{k{:}y_2z\} = \varnothing$.

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.*

where $A^{\leq k} = A^0 \cup A^1 \cup A^2 \ldots \cup A^k = \{\varepsilon\} \cup A \cup A^2 \ldots \cup A^k$

$$\leftrightarrow \$^k \cup A\$^{k-1} \cup A^2\$^{k-2} \ldots \cup A^k.$$

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

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

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

where $First_k: (N \cup \Sigma)^* \rightarrow 2^{\Sigma^{\leq k}}$,

Let $\alpha = X_1 \ldots X_n$, $1 \leq {}^{\forall} i \leq n$, $X_i \in N \cup \Sigma$. *Then*

$First_k(\alpha) = First_k(X_1 \ldots X_n)$

$= First_k(X_1) \oplus_k First_k(X_2) \oplus_k \ldots \oplus_k First_k(X_n)$.

$= k{:}[First_k(X_1) \cdot First_k(X_2) \cdot \ldots \cdot First_k(X_n)]$.

Note that $\oplus_k$ *is easier than* $First_k$.

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

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

Simple approach - Strong LL(k) parsing

We use $First_k(\beta_1) \oplus_k Follow_k(A)$ *instead of* $k{:}y_1z$

Guess A *as* β_1 ***on lookahead*** $y \in First_k(\beta_1) \oplus_k Follow_k(A)$ *only.*

We define $LaSLL_k(A \to \alpha) = First_k(\alpha) \oplus_k Follow_k(A)$.

Guess and verify parser with k-lookahed strings, $LaSLL_k(A \to \alpha)$,

is called a ***strong LL(k)*** *parser.*

If $LaSLL_k(A \to \alpha) \cap LaSLL_k(A \to \beta) = \varnothing$.

deterministic guess A *as* α *or* β

on $LaSLL_k(A \to \alpha)$ *or* $LaSLL_k(A \to \beta)$, *respectively.*

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

$^{\forall}A \to \alpha \mid \beta \in P, \alpha \neq \beta, LaSLL_k(A \to \alpha) \cap LaSLL_k(A \to \beta) = \varnothing$.

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.*

$A \rightarrow A\alpha \mid \beta$

A is left recursive.

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

$\supseteq First_k(\beta)$

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

Removal of left recursion

$A \rightarrow A\alpha \mid \beta$

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

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \varepsilon$

Left factrong

$A \rightarrow \alpha\beta \mid \alpha\gamma$

$A \rightarrow \alpha A'$

$A' \rightarrow \beta \mid \gamma$

LL(1) parsing

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

$First_1(T_1 \cdot T_2) = T_1 \oplus_1 T_2$

$= First_1(T_1)$, if $\varepsilon \notin T_1$.

$= First_1(T_1) \cup First_1(T_2)$, if $\varepsilon \in T_1, \varepsilon \in T_2$.

$= First_1(T_1) \cup First_1(T_2) - \{\varepsilon\}$, if $\varepsilon \in T_1, \varepsilon \notin T_2$(otherwise).

$First_1: N \cup \Sigma \rightarrow 2^{\Sigma^{\leq 1}} = 2^{\{\varepsilon\} \cup \Sigma} = \{\varepsilon\} \cup 2^{\Sigma}$ $First_1 \leftrightarrow nullable \cup First$.

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

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

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

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

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

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

$First(A) \supseteq \{a \in \Sigma | A \to \alpha a \beta \in P, \alpha \Rightarrow^* \varepsilon\}$ **basis**

$First(A) \supseteq \{a \in First(B) | A \to \alpha B \beta \in P, \alpha \Rightarrow^* \varepsilon\}$ **recursion**

$Follow_1: N \to 2^{\Sigma^{\leq 1}} = 2^{\{\varepsilon\} \cup \Sigma} = \{\varepsilon\} \cup 2^{\Sigma}$.

$Follow_1: N \to \{\varepsilon\} \cup 2^{\Sigma}$.

$$Follow_1(A) = \{1{:}y \in \{\varepsilon\} \cup \Sigma | S \Rightarrow^* xAy, x, y \in \Sigma^*\}$$

$\varepsilon \in Follow_1(S)$. (*We add new* ***augmented rule*** $S' \to S$)

We may use Follow instead of $Follow_1$.

Recursive formula for Follow(A)

$Follow(S) \supseteq \{\varepsilon\}$ **basis**

$Follow(A) \supseteq \{a \in First(\beta) | B \to \alpha A \beta \in P\}$ **basis**

$Follow(A) \supseteq \{a \in Follow(B) | B \to \alpha A \beta \in P, \beta \Rightarrow^* \varepsilon\}$ **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)*

$Init\_First(A) \supseteq \{a \in \Sigma \mid A \rightarrow \alpha a \beta \in P, \alpha \Rightarrow^{*} \varepsilon\}$ ***basis***

$= \{a \in \Sigma \mid A \; l \; a\}$

$First(A) \supseteq \{a \in First(B) \mid A \rightarrow \alpha B \beta \in P, \alpha \Rightarrow^{*} \varepsilon\}$

$= \{a \in First(B) \mid A \; l \; B\}$ ***recursion***

$Init\_Follow(S) \supseteq \{\varepsilon\}$ ***basis***

$Init\_Follow(A) \supseteq \{a \in First(\beta) \mid B \rightarrow \alpha A \beta \in P\}$ ***basis***

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

$= \{a \in Follow(B) \mid A \; r \; B\}$ ***recursion***

Let A and B be two sets and x, y ∈ A, R ⊆ A × A, and f, g: A → 2^B.
Compute f for given R and g,

i) f(x) ⊇ g(x) **basis**

ii) f(x) ⊇ f(y) where x R y **recursion**

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

iii) Nothing else is in f(x) **fixed point**

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

where $=_S$ *denotes the smallest set satisfying RHS of the rule*

Then $f(x) = \{b \in g(y) | \; x \; R^* \; y\}$ **iteration**

$x \; R^* \; y$: *Reachable vertices in the graph (A, R).*

depth-first search

topological order

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

input $g: A \to 2^B$; $R \subseteq A \times A$. ***output***: $f: A \to 2^B$.

var *S:* ***stack*** *of A;* $N: A \to Depth$ $(\subseteq \aleph)$.
function *Trav(x: A, d: 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*** *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***
        $y :=$ ***pop of*** *S;* $N(y) :=$*Infinity;* $f(y) := f(x)$ ***until*** $(y = x)$ ***fi***
    end function *Trav*

for $x \in A$ ***do*** $N(x) := 0$; $f(x) := \varnothing$ ***od***;
for $x \in A$ ***where*** $(N(x) = 0)$ ***do*** *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 nullale(A).*

4. $^{\forall}A \in N$*, compute l-relation and Init_First(A) using nullale.*

5. $^{\forall}A \in N$*, compute First(A) using l-relation and Init_First(A)*
by traversing the l-graph. (R=l-relation, g=Init_First, f=First)

4. $^{\forall}A \in N$*, compute r-relation and Init_Follow(A)*
using nullale and First(β*).*

5. $^{\forall}A \in N$*, compute Follow(A) using r-relation and Init_Follow(A)*
by traversing the r-graph. (R=r-relation, g=Init_Follow, f=Follow)

6. $^{\forall}A \rightarrow \alpha \in P$*, compute* $\boldsymbol{LaSLL_1}(A \rightarrow \alpha)$ *using First and Follow.*

7. $^{\forall}A \rightarrow \alpha \mid \beta \in P, \alpha \neq \beta$*, if* $LaSLL_1(A \rightarrow \alpha) \cap LaSLL_1(A \rightarrow \beta) = \varnothing$*, 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

	nullable	*init.*	*First*	*init.*	*Follow*
$E \rightarrow T\ E'$	*no*		*a, (*	*$,)*	*$,)*
$E' \rightarrow + T\ E' \mid \varepsilon$	*yes*	*+*	*+*		*$,)*
$T \rightarrow F\ T'$	*no*		*a, (*	*+*	*$,), +*
$T' \rightarrow * F\ T' \mid \varepsilon$	*yes*	***	***		*$,), +*
$F \rightarrow a \mid (E)$	*no*	*a, (*	*a, (*	***	*$,), +, **

$LaSLL_1(E' \rightarrow + T\ E') = \{+\}$ $LaSLL_1(E' \rightarrow \varepsilon) = \{\$,)\}$

$LaSLL_1(T' \rightarrow * F\ T') = \{*\}$ $LaSLL_1(T' \rightarrow \varepsilon) = \{\$,), +\}$

$LaSLL_1(F \rightarrow a) = \{a\}$ $LaSLL_1(F \rightarrow (E)) = \{(\}$

$\therefore$ *LL(1).*

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

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

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'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
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;* $a := 1{:}x$; $y := x\${:}|x|\text{-}1$
repeat
    $X :=$ ***pop of*** *Stack;*
    if $(X \in N) \Rightarrow$
        if $(SLL_1PT(X, a) = X \rightarrow \alpha) \Rightarrow$ ***push*** α^R *onto Stack;*
            Make subtree with root X and α*'s as children.*
        | $SLL_1PT(X, a) = \emptyset \Rightarrow$ *syntax error* ***fi***
    | $(X \in \Sigma) \Rightarrow$ ***if*** $(X=a) \Rightarrow a := 1{:}y$; $y := y{:}|y|\text{-}1$
            | $(X \neq a) \Rightarrow$ *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='(') ⇒ pT; pE'*
| ***otherwise*** *⇒ syntax error* ***fi***

function *pE':* ***if*** *(i='+') ⇒ verify('+'); pT; pE'*
| *(i=')') or (i='$') ⇒ skip*
| ***otherwise*** *⇒ syntax error* ***fi***

function *pT:* ***if*** *(i='a') or (i='(') ⇒ pF; pT'*
| ***otherwise*** *⇒ syntax error* ***fi***

function *pT':* ***if*** *(i='*') ⇒ verify('*'); pF; pT'*
| *(i='+') or (i=')') or (i='$') ⇒ skip*
| ***otherwise*** *⇒ syntax error* ***fi***

function *pF:* ***if*** *(i='a') ⇒ verify('a')*
| *(i='(') ⇒ verify('('); pE; verify(')')*
| ***otherwise*** *⇒ syntax error* ***fi***

i := 1:x; y := x$:|x|-1; pE; ***if*** *(y = $) ⇒ O.K. | (y ≠ $) ⇒ syntax error* ***fi***