

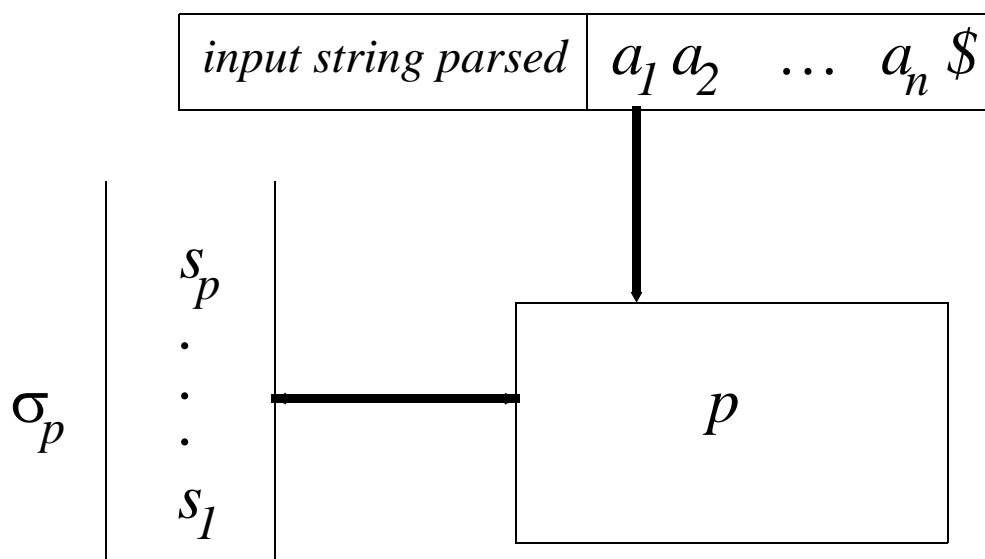
*Error Recovery**Error detection**Error recovery**Error correction**Error repair**Table driven error recovery**Algorithm**Cost**Def. Insert and delete cost of terminal symbol**For all $a \in \Sigma$, insertion and deletion costs are defined as **positive** numbers, and denoted as $Ic(a)$ and $Dc(a)$, respectively.**Def. Insert and delete cost of terminal string**Let $x = a_1 \dots a_n \in \Sigma^*$. Then*

$$Ic(x) = \begin{cases} Ic(a_1) + \dots + Ic(a_n), & \text{if } x \neq \varepsilon (n \geq 1), \\ 0 & , \text{ if } x = \varepsilon. \end{cases}$$

$$Dc(x) = \begin{cases} Dc(a_1) + \dots + Dc(a_n), & \text{if } x \neq \varepsilon (n \geq 1), \\ 0 & , \text{ if } x = \varepsilon. \end{cases}$$

Parsing Configuration

current state p ,
 stack content(LL, LR) σ_p ,
 input string to be parsed $a_1 \dots a_n \$$.



Parsing configuration $(\sigma_p, a_1 \dots a_n \$)$

Error symbol a_1

Error recovery with insert only

Insert some terminal string x , which guarantees continuation of parsing with xa_1 .

Least Cost Insertion String

Def. Let parsing configuration be $(\sigma_p, az\$)$. Then the locally least cost insertion terminal string, $Lci(\sigma_p, a)$, is defined as follows,

$Lci(\sigma_p, a) = x$, if $Ic(x) \leq Ic(y)$ for all y such that

$$(\sigma_p, xaz\$) \vdash^+ (\sigma, z\$), \text{ and}$$

$$(\sigma_p, yaz\$) \vdash^+ (\sigma', z\$).$$

$=?$, otherwise,

where $? \notin \Sigma$ and $Ic(?) = \infty$.

Algorithm: Least cost insertion string error recovery with deletion.

Input: Parsing configuration($\sigma_p, a_1 \dots a_n\$$)

Output: Error recovery

var i : integer;

x, y : Terminal_string;

begin

$i := 1$;

$x := Lci(\sigma_p, a_i)$;

$y := Lci(\sigma_p, a_{i+1})$;

while $Ic(x) > Dc(a_i) + Ic(y)$ ***do***

delete(a_i);

$i := i + 1$;

$x := y$;

$y := Lci(\sigma_p, a_{i+1})$

od

insert(x)

end.

Def. Deletion cost of end marker.

$$DC(\$) = \infty.$$

*Def. Immediate Error Detection Property(IEDP).
IEDP holds if the error is detected without any improper action in parsing.*

IEDP holds for LR(1) parsing.

IEDP holds for LALR(1) parsing except for reduce actions.

IEDP holds for LALR(1) parsing, if all reduce actions after the last shift action are recovered.

Reduce stack:

Reduce stack is cleared for every shift action.

When error is encountered the parsing configuration is recovered with reduce stack.

Def. Least cost derivable string, and least cost derivable prefix string for a terminal symbol.

Let $X \in V$ and $a \in \Sigma$. Then

$Lcd(X) = x$ if $\exists x$ s.t. $X \xRightarrow{} x$, and*

$$IC(x) \leq IC(y) \quad \forall y \text{ s.t. } X \xRightarrow{*} y.$$

$Lcp(X, a) = x$, if $\exists x$ s.t. $X \xRightarrow{} xaz$, and*

$$IC(x) \leq IC(y),$$

$$\forall y \text{ s.t. } X \xRightarrow{*} yaz', \text{ where } z, z' \in \Sigma^*.$$

= ?, otherwise

Extension.

Let $\alpha \in V^$ and $a \in \Sigma$. Then*

$Lcd(\alpha) = x$ if $\exists x$ s.t. $\alpha \xRightarrow{} x$, and*

$$IC(x) \leq IC(y) \quad \forall y \text{ s.t. } \alpha \xRightarrow{*} y.$$

$Lcp(\alpha, a) = x$, if $\exists x$ s.t. $\alpha \xRightarrow{} xaz$,*

$$IC(x) \leq IC(y),$$

$$\forall y \text{ s.t. } \alpha \xRightarrow{*} yaz', \text{ where } z, z' \in \Sigma^*.$$

= ?, otherwise

Some properties of Lcd and Lcp

$$Lcd(a) = a$$

$$Lcp(a, b) = \varepsilon \text{ if } a = b \\ = ? \text{ otherwise}$$

$$Lcd(\alpha) \neq ?$$

$$Lcd(\alpha) = Lcd(X_1 \dots X_n) \\ = Lcd(X_1) \cdot \dots \cdot Lcd(X_n)$$

$$Lcp(\varepsilon, a) = ?$$

$$Lcp(X\alpha, a) = \min(Lcp(X, a), Lcd(X) \cdot Lcp(\alpha, a))$$

$$Lcp(X_1 \dots X_n, a) = \min(\\ Lcp(X_1, a), \\ Lcd(X_1) \cdot Lcp(X_2, a) \\ \dots \\ Lcd(X_1) \cdot \dots \cdot Lcd(X_{n-1}) \cdot Lcp(X_n, a) \\)$$

Expected Vocabulary String for a Left Context in LR-based Parsing

Let a left context be σ_p , where $\sigma_p = s_1 \dots s_p$. Then

$$\begin{aligned} \text{Evc}(\sigma_p) = \\ \{ \alpha_2 \cdot \text{Evc}_i(\sigma_p, [A \rightarrow \alpha_1 \alpha_2]) \\ / [A \rightarrow \alpha_1 \alpha_2] \in \text{Kernel}(s_p) \}, \text{ where} \end{aligned}$$

$$\begin{aligned} \text{Evc}_i(\sigma_p, [A \rightarrow \alpha_1 \alpha_2]) = \\ \{ \beta_2 \cdot \text{Evc}_i(\sigma_{p-|\alpha_1|}, [B \rightarrow \beta_1 A \beta_2]) \\ / [B \rightarrow \beta_1 A \beta_2] \in s_{p-|\alpha_1|} \} \end{aligned}$$

Theorem

$$\text{Lci}(\sigma_p, a) = \text{Lcp}(\text{Evc}(\sigma_p), a)$$

$Lalr(p, [A \rightarrow \alpha_1.\alpha_2])$ vs. $Evci(\sigma_p, [A \rightarrow \alpha_1.\alpha_2])$

1. *state vs. left context(sequence of state)*
2. *for all possible Pred states vs. unique state in the parse stack(left context)*
3. *set of terminal strings vs. set of vocabulary strings*
4. *infix first operator(\oplus) vs. concatenation(.)*

New LALR Formalism

$Evci(\sigma_p, [A \rightarrow \alpha_1.\alpha_2]) =$

$$\{\beta_2 . Path(A', A) . Evci(\sigma_{p-|\alpha_1|}, [B \rightarrow \beta_1.A'\beta_2]) \\ | A' L^* A, [B \rightarrow \beta_1.A'\beta_2] \in Kernel(s_{p-|\alpha_1|})\}$$

Input parameters

Stack: array[SP_Ran] of State;

function Lci(Top: SP_Ran; a: Σ): Σ^ ?;*

var x: Σ^ ?;*

function Lcd($\alpha: V^$): Σ^* ; external;*

function Lcp($\alpha: V^$; a: Σ): Σ^* ?; external;*

function MinIs($\alpha: V^$; var y: Σ^*): boolean;*

begin

if $x \geq y + Lcp(\alpha, a) \rightarrow x := y.Lcp(\alpha, a)$

| $x \leq y + Lcp(\alpha, a) \rightarrow skip$

fi;

if $x \geq y + Lcd(\alpha) \rightarrow y := y.Lcd(\alpha); return true$

| $x \leq y + Lcd(\alpha) \rightarrow return false$

fi

end; (MinIs *)*

```

procedure Back(Sp:SP_Ran; A:N ; y:  $\Sigma^*$ );
var y':  $\Sigma^*$ ; more: boolean;
begin
    for [B→ $\beta_1$ .A' $\beta_2$ ] ∈ Kernel(Stack[Sp]) and
        A' L* A do
        y' := y; more := MinIs(Path(A', A). $\beta_2$ , y');
        if more → Back(Sp-/ $\beta_1$ /, A', y')
            | not more → skip
        fi
    od
end; (* Back *)

var y:  $\Sigma^*$ ; more: boolean;
begin
    x := ?;
    for [A→ $\alpha_1$ . $\alpha_2$ ] ∈ Kernel(Stack[Top]) do
        y :=  $\epsilon$ ; more := MinIs( $\alpha_2$ , y);
        if more → Back(Top-/ $\alpha_1$ /, A, y)
            | not more → skip
        fi
    od;
    return x
end; (* Lci *)

```

Heuristics for insertion and deletion costs

- 1. DC is greater than that of IC*
- 2. IC and DC for frequently used symbols(eg. **identifiers**, ,, ;, ...) are small*
- 3. IC for the symbols that open structures(e.g., **procedure**, **begin**, **if**, **while**, ...) are high, and IC for the symbols that close structures(e.g., **end**, ;, ...) are low.*