9. Syntax Error Handling

Goal

- *to maximize input text*
- *to report actual errors*

9.1 Syntax Error

Syntax Error

- *actual Error made by programmer*
  - *depends on the Language*
- *formal definition of actual Errors detected by a particular parser*
  - *reflects the characteristics of parsing technique*

- *parser - defined error in ω = xy*

  pair(|x|,1:y) or 1:y,
  if x is the longest prefix of ω

- *Let M : pda with initial stack contents γs*

  - *M has the correct prefix property if the statement $γs | xy$ $\stackrel{*}{\Rightarrow}$ $γ | y$ can be true in M only if x is a prefix of some sentence in L(M)*

  - *M has the longest correct prefix property if M has correct prefix property and for any terminal string xy in which x is a prefix of some sentence in L(M) there is a stack string γ s.t.*
    $γs | xy$ $\stackrel{*}{\Rightarrow}$ $γ | y$ in M

  - *Any computation on a nonsentence can be continue up to the parser-defined error.*

[Theorem 9.1]

For k≥0, any LR(k) prser of a reduced grammar has the correct prefix property.

proof)

Let M : LR(k) parser of G.
Consider a computation
$\$[\$]|xy\$ \overset{*}{\Rightarrow} \$\varphi|y\ \$$ *in M*

Here
$\varphi = [\$][\$X_1] \ldots [\$X_1 \ldots X_n]$, $n \geq 0$
where $X_1 \ldots X_n$ *: viable prefix of G.*

Moreover
$X_1 \ldots X_n$ *derivers x in G, by L.6.29, L.6.49*

As any terminal string derived by a viable prefix of a G is a prefix of some sentence.

We conclude theorem.

q Q.E.D.

[Theorem 9.2]

For k≥0, any LL(k) arser of a reduced grammar has the correct prefix property.

Proof)

Let M : canonical LL(k) of G.
Consider a computation
$[$][$s] | xy$ \overset{*}{\Rightarrow} $\varphi|y$ in M$
Here
stack string $\varphi = [\$][\$X_1] .. [\$X_1 ... X_n]$ $n \geq 0$
s.t. S derives $xX_n ... X_1$ *by 8.30*

As G is reduced x is a prefix of some sentence in L(G)

q *Q.E.D.*

Let M : pda with input alphabet ²
ω *: nonsentence*
Φ *: error configuration.*

- *M detects an error in* ω *at* Φ *if M has an* ω *a computaion that ends with* Φ.

- *M detects the parser-defined error in ω if M detects an error in ω at a configuration of the form $γ|y$ in which 1:y is the parser-defined error in ω.*

[Fact9.3]

Let M : pda that defects an error in every nonsentence and has longest correct prifix property.
Then
M detects the parser-defined error in every nonsentence.

[Theorem 9.4]

An LR(0) orLL(1) parser of a reduced grammar detects the parser-defined error in every nonsentence.

proof)

Let M : LR(1) parser of G.
M has the correct prefix property(T.9.1)

Let $xy \in \Sigma^*$ s.t. x is a prefix of some sentence in
L(M) = L(G) for some y', $xy' \in L(M)$
$[$]|xy$ $\stackrel{*}{\Rightarrow}$ φy in M
if $x = \varepsilon$: $\varphi|y$: initial conf.
otherwise : $x = x'a\ xy' \in L(M)$
there is $[$]|x'ay'$ $\stackrel{*}{\Rightarrow}$ $\varphi|ay'$ $\Rightarrow$ $\varphi|y'$
As the length of lokahead = 1
$[$]|x'ay$ $\stackrel{*}{\Rightarrow}$ $\varphi'|ay$ $\Rightarrow$ $\varphi|y$ in M

q *Q.E.D.*

- *LL(k), LR(k) s.t. k > 1.*
  - *declare error and halt before the entire correct prefix has been consumed.*

- *simple precedure parser (bottom up parser) non-deterministic shift-reduce parser(bottom up parser)*

  - *not possess the correct prefix property.*

9.2 Error Recovery in SLL(1) Parsers

- *Error detection in Recursive Desent Parser.*

case 1 : beginning of procedure A.
current symbol $\neq$ $First_{\underline{1}}(A\ Follow_{\underline{1}}(A))$

- *detected begin of the procedure A.*

- *scan input string until legal follower of the A.*

case 2 : having parsed prefix α
$A \rightarrow \alpha a \beta$, *current symbol* $\neq a$

- *search remaining input for* $First_{\underline{1}}(\beta\ Follow_{\underline{1}}(A))$

- *scan input until* $First_{\underline{1}}(\beta\ Follow_{\underline{1}}(A)$

$\Rightarrow$ *Problem : unanalyzed program text.*

- *Parsing within Error Procedure.*

- *search for starters of some nonterminals*

- *corresponding parsing procedure is called.*

$\Rightarrow$

$b \in First_1(\beta)$ cause call for procedure A in the procedure error

only if B is in some reasonably defined subset of the set of all nonterminals, and other nonterminals, and for all other nonterminal C in this set, $b \notin Firsr_1(C)$

- *X begins A (sec 5.5)*

  $A \rightarrow \alpha X \beta$, where $\alpha \stackrel{*}{\Rightarrow} \varepsilon$

  In recovering Process, calling A capture all those possibilities that arise from calling X

[Fact 9.5]

For any reduced LL(1) grammar, the relation begins is partial order

- *maximal nonterminals*
  those that do not appear as leftmost components in the pairs in begins

- *Error Handling Procedure*
  - *Computing all sets $First_1(B)$*
    Where B is a maximal nonterminal

  - *Pairwise intersection of sets*
    : those symbols that are no intersection $First_1(B) \cap First_1(C)$ where C is another maximal nonterminal

- *Error Recovery in R.D.P not loop forever any input string*
  - *Assume : Parsing Procedure A : error detect*
    $\rightarrow$ Procedure error called.

  - *recovery action calls procedure B.*
    $b \in First_1(B)$: shift b.

  - *recovery action back to procedure A.*
    left-recursive nonterminal.

SLL(1) Parser

• *Error Conf.*

$\$\psi X|y\$$ $1{:}y\$ \notin First_1(X\psi^R\$)$

• *Recovery Strategy*

- *Pop X from the parsing stack.*
- *Scan the remaining inputstring y$ until*
  - ¨ *$ or First($\psi^R\$$)*
  - ¦ *a symbol that belongs to exactly one set $First_1(B)$ where B : maximal nonterminal with respect to the relation <u>begins</u>.*

¨ $\rightarrow \$\psi|z\$$ ⟨ *accepting Conf.*

at least one parsing action whenever $Z \neq \varepsilon$

¦ $\rightarrow \$\psi B|z\$$

can proceed at least one step

9.3 Error Recovery in LR(1) Parsers

Phrase-level recovery

• *Let M : LR(0)-based LR(1) parser*

- *Error conf.* $\Phi = \$q_0...q_m \mid a_1...a_n$

$Action[q_m,a_1] =$ "error"

- *Error phrase*

$q_{i+1}...q_m \mid a_1...a_{j-1}$

if G has nonterminal s.t. $Goto[q_i,A] \neq \phi$

$Action[Goto[q_i,A],a_j] \neq$ "error"

• q_i : *recovery state*

• *A : (reduction) goal*

• $Goto[q_i,A]$: *goal state*

• a_j : *recovery symbol*

$\Rightarrow \$q_0q_1...q_iq_A \mid a_j...a_m$,

$q_A = Goto[q_i,A]$

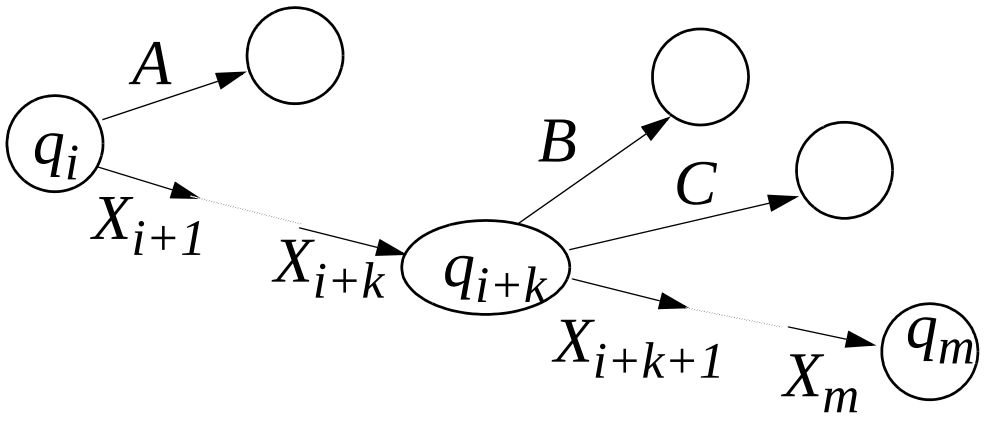
- *Error conf.* $\$q_0 q_1 ... q_m \mid a_1 ... a_n$

  feasible reduction goals
  $F(q_{i+1},...q_m)$
  $= \{ A \mid A$ *is a reduction goal of error phrase*
  $q_{i+1}...q_m \mid a_1...a_{j-1}$ *for some j,*
  and
  $A \underset{rm}{\Rightarrow}^* X_{i+1}...X_m z$ *for some* $z \in \Sigma^*$
  where X_k *is the entry symbol of* q_k,
  $k = i+1, ...,m.$
  $\}$
  for all $i = 0, ..., m-1.$

- *(q,* $B \rightarrow \alpha \bullet \beta$*) on-string-goes-to Goto(q,* α*)*
  (q, $B \rightarrow \alpha \bullet \beta$*) reduces-to (q, B)*
  (Goto(q,α*), B) symbols-in (q,* $A \rightarrow \alpha B \bullet \beta$*)*
  cf : sect 7.5.

[Theorem 9.6]
Nonterminal A is in $F(q_{i+1}...q_m)$ iff

q_m on-string-goes-to^{-1} (reduce-to symbol-in)* reduces-to(q_i,A) o



For α,β, $A \rightarrow X_{i+1}...X_{i+k}B\alpha$,
$C \rightarrow X_{i+k+1}...X_m\beta$.

q_m on-string-goes-to^{-1} (q_{i+k}, $C \rightarrow X_{i+k+1},...X_m \bullet \beta$)
$\therefore$ C : feasible reduction goal for error phrase including $q_{i+k+1}...q_m$ |

Moreover,
if $B \underset{rm}{\Rightarrow}^+ C\gamma$ for some γ,
then (q_{i+k}, $C \rightarrow X_{i+k+1}...X_m \bullet \beta$)(reduce-to symbols-in)* reduce-to(q_i,A)
$\therefore$ A is feasible for error phrases includeing $q_{i+1}...q_m$ |

- *Search Error Conf. for an error phrase*

  - *Let (i, j) segment* $q_{i+1}...q_m \mid a_1...a_{j-1}$ *in error conf.* $\$q_0...q_m \mid a_1...a_n\$$

  - *Search (m, 1)* $\rightarrow$ *(0, n)*
    whenever (i_1, j_1) *procedes* (i_2, j_2)
    $i_1 > i_2,\ j_1 < j_2$

  - *Until unique important feasible reduction goal is found.*

- *No candidate error phrase.*

  - *When the parser has recognized a complete sentence but input string is remainded.*

  - *restart action :*
    - *Simply delete the topmost stack symbol.*
    - *Used a terminal as a recovery symbol only if it is shiftableat the goal state*
      : not cause the error recovering parser to loop forever.

Local correction

- *generalizing the concept of a recovery action s.t. terminal and empty string are allowable reduction goal.*

  $<b> \rightarrow b$ *: insertion, replacement.*
  $<b> \rightarrow <\varepsilon> b$ *: deletion*
  $<\varepsilon> \rightarrow \varepsilon$

- *case of terminal reduction goal*
  : as in the nonterminal reduction goal

- *case of empty reduction goal*

  - *no goal state*

  - *recovery symbol : shiftable terminal at the recovery state.*

- *input portion of the error phrase ≤ 1.*
  stack portion of the error phrase

  → *use of terminal and empty reductions goals for insert, delete, replacement of a single terminal.*

- *For higher performance*

  - *respective cost of each allowable local correction is evaluated.*

  - *only one with lowest cost correction is choosen.*

Forward-move recovery

- *Recovery state for special grammar symbol*

Let M : LR(0) machine, X∈ V.

- recovery state X
. union of all states in M that have X as the entry symbol

- successor for recovery state
. computed ordinary states in the LR(0) machine until no new state are obtain

- initial recovery state has a transition to the recovery state for X.

- *Forward-move recovery*

  - at an error conf.
  whole stack content
  → initial recovery state

  - parsing proceeds.

  - if conflict in recovery state
  or reduce action
  (whole parsing stack)
  then a new start will be invocked
  by emptying stack
  (except recovery stack)

  - if new error is detected
  at a recovery state
  then a new error is reported and some
  recovery process is repeated.

- *Error recovery table*

  - [£] : *initial recovery state for all* $X \in V$
  - [£*X*] : *recovery state for X*
  - [£*X*α] : *for all* $\alpha \in V^*$

  the α*-successor of recovery state for X.*

  - *RGoto[[£*ζ*],X] = [£*ζX*]*
    for all $\zeta \in V^*$, $X \in V$

  - *RAction : parsing action in the E.R.T*

  - *the Entries of RAtion : [£X]*
    if all states with entry symbol X in
    the parser have the same parsing
    action on terminal a
    then the state [£X] has this action a

    RAtion [[£X],a] = "conflict"
    if not unique action

    RAtion [[£],a] = "shift[£a]", $a \in \sum$
    RAtion [[£],$] = "halt"

9.4 Error Reporting

Error Message

declarative : location of error
automatic error handling
language, Grammar Info

operational : recovery action
by recovery routine.

SLL(1) parsing error

1) shift error : if a terminal cant't be shifted
topmost element in the stack.
→ prediction of a terminal
∴ expected symbol : terminal to be shifted

2) produce error : if a nonterminal can't be expanded
topmost element in the stack.
→ prediction of a nonterminal
∴ expected symbol : starter of nonterminal to be expanded.

LR(1) Parsing

Let M : LR(0) based LR(1) parser.

X ∈ *V is expected in state q = varid(γ)*
if q contains an item of
the form [A → α•*X*β]

X is expected in Conf. $*η | *x$ of M
if X is expected in η *: 1*

valid continuation of LR(1)
if X : expected in Conf.
then X (or drived string from X) is valid continuation of the stack.
Other valid continuation
lookahead symbols of those reduction action that apply to stack string

Error detection state
topmost state q in the stack in the error configuration
if topmost state has [A→ω•]
expected symbols alone do not form a sufficient basis for error msg.

if default reductions are used
this situation does not occur
error msg. can be based solely on
the expected symbol.

If only essential symbols are used, this reduces the number of expected symbols.

symbol X expected in state q is essential
if q contains an item of the form
$[A \rightarrow \alpha.X\beta], \alpha \neq \varepsilon$

Essential expected symbols capture all
the information about the valid contitn-
ations obtained by all expected symbols

X is terminallike
if either (1) X is a terminal,
or (2) an nonterminal that has
only unit rules with
terminal like right hand side
Every rule for X is the form $X \rightarrow Y$
whereY is terminallike symbol
name for a set of terminals

Nonterminallike

Maximal essential symbol
- *not derived by unit rules from other essential symbols in that state.*
- *removing redundancy between the essential synbols.*

Automatic Generation of D.E.M

Case 1 : no error detection state q contains an essential item of the form [A→ω•]

suppose X_1 .... X_n : maximal essential symbols expected in q

if each symbol X_i : terminallike
then "X_1 expected" (case n =1)
"X_1 .... X_{n-1}or X_n expected"
(caes n ≥ 1)

if each symbol X_i : nonterminallike
then "No X_1 can start with this"
(case n = 1)
"Neither X_1 .... X_{n-1} nor X_n can start with this" (case n > 1)

if X_1 .... X_k (k ≥ 1) : terminallike and X_{k+1} .... X_n (k ≤ n) : nonterminallike
then
"X_1 .... X_{n-1} or start of X_n expected"
(case K = n-1)
"X_1 .... X_k or start of X_{k+1} .... X_n , or X_n expected" (case k < n-1)

Case 2 : Error detection state q contains essential item of the form [A→ω•]

if the entry symbol X is terminallike
"X can't be followed by this"
otherwise
"No complete X can be followed this"

cf) if default action is allowed this cannot occur in any error detection state.

if default action is not allowed
- *a number of error detection state*
- *many lookahead symbols for one reduce state*
- *not all lookahead symbol are valid continuation of stack : no way to choose the correct one.*