LL grammar

xpojez00, xbudin05

December 5, 2021

1 Introduction

Our approach of creating LL table was to start from the easier **and** smaller parts **and** work our way to more complex non-terminals. At first, we filled the terminal set with all valid tokens. Our non-terminal **and** rule sets were empty. Starting from the variable declaration, value assignments **and** conditions. We are planning on implementing LL grammar using Predictive parsing.

2 Terminal set

$T = \{id, integer, number, string, "-", "+", "*", "/", "//", " : ", ",", ".", "\#", "(", ")", " < ", " <= ", " > ", " >= ", "..", " = ", " == ", " ~= ",$ **and**, **boolean**, **break**, **do**, **else**, **elseif**, **end**, **false**, **for**, **function**, **global**, **if**, **integer**, **local**, **nil**, **not**, **number**, **or**, **repeat**, **require**, **return**, **string**, **then**, **true**, **until**, **while**$\}$

3 Non-Terminal set

$NT = \{< program >, < global\_scope >, < global\_statements >, < global\_statement >< function\_declare >, < function\_define >, < function\_call >, < parameters >, < parameter >, < parameter\_name >, < parameters\_defined >, < parameter\_defined >, < returning >, < scope >, < called\_parameters >, < scope\_statements >, < statements >, < statement >, < declare >, < id >, < if >, < while >, < for >, < repeat >, < scope\_return >, < return >, < declare\_assign >, < assign >, < condition >, < condition\_branch >, < lvalues >, < lvalue >, < rvalues >, < rvalue >, < expression >, < expression\_2 >, < expression\_3 >, < datatypes >, < datatype >, < unary\_operator >, < binary\_operator >\}$

4 Rule set

$< program >$	$\rightarrow$ **require**.$string$. $< global\_scope >$
$< global\_scope >$	$\rightarrow < global\_statements >$
$< global\_scope >$	$\rightarrow \epsilon$
$< global\_statements >$	$\rightarrow < global\_statements >$. $< global\_statement >$
$< global\_statements >$	$\rightarrow < global\_statement >$
$< global\_statement >$	$\rightarrow < function\_declare >$
$< global\_statement >$	$\rightarrow < function\_define >$
$< global\_statement >$	$\rightarrow < function\_call >$
$< function\_declare >$	$\rightarrow$ **global**.id." : ".**function**."(". $< parameters >$.")". $< returning >$
$< function\_define >$	$\rightarrow$ **function**.id."(". $< parameters\_defined >$.")". $< returning >$. $< scope >$.**end**
$< function\_call >$	$\rightarrow$ id."(". $< called\_parameters >$.")"

$< parameters >$ $\rightarrow < parameters > ."," . < parameter >$
$< parameters >$ $\rightarrow < parameter >$
$< parameters >$ $\rightarrow \epsilon$
$< parameter >$ $\rightarrow < parameter\_name > . < datatype >$
$< parameter\_name >$ $\rightarrow id." : "$
$< parameter\_name >$ $\rightarrow \epsilon$
$< called\_parameters >$ $\rightarrow < rvalues >$
$< called\_parameters >$ $\rightarrow \epsilon$

$< parameters\_defined >$ $\rightarrow < parameters\_defined > ."," . < parameter\_defined >$
$< parameters\_defined >$ $\rightarrow < parameter\_defined >$
$< parameters\_defined >$ $\rightarrow \epsilon$
$< parameter\_defined >$ $\rightarrow id." : ". < datatype >$

$< returning >$ $\rightarrow " : ". < datatypes >$
$< returning >$ $\rightarrow \epsilon$

$< scope >$ $\rightarrow < scope\_statements > . < scope\_return >$
$< scope\_statements >$ $\rightarrow < statements >$
$< scope\_statements >$ $\rightarrow \epsilon$
$< statements >$ $\rightarrow < statements > . < statement >$
$< statements >$ $\rightarrow < statement >$

$< statement >$ $\rightarrow < declare >$
$< statement >$ $\rightarrow < id >$
$< statement >$ $\rightarrow < if >$
$< statement >$ $\rightarrow < while >$
$< statement >$ $\rightarrow < for >$
$< statement >$ $\rightarrow < repeat >$
$< statement >$ $\rightarrow$ **break**

$< declare >$ $\rightarrow$ **local**. $< lvalues >$." : ". $< datatypes > . < declare\_assign >$
$< id >$ $\rightarrow id."(". < called\_parameters > .")"$
$< id >$ $\rightarrow id. < assign >$
$< id >$ $\rightarrow id."," . < lvalues > . < assign >$
$< if >$ $\rightarrow$ **if**. $< condition >$.**end**
$< while >$ $\rightarrow$ **while**. $< expression >$.**do**. $< scope >$.**end**
$< for >$ $\rightarrow$ **for**$id. < expression >$."," . $< expression >$."," . $< expression >$. **do**. $< scope >$.**end**
$< repeat >$ $\rightarrow$ **repeat**. $< scope >$.**until**. $< expression >$
$< scope\_return >$ $\rightarrow < return >$
$< scope\_return >$ $\rightarrow \epsilon$

$< declare\_assign >$ $\rightarrow < assign >$
$< declare\_assign >$ $\rightarrow \epsilon$
$< assign >$ $\rightarrow " = ". < rvalues >$
$< condition >$ $\rightarrow < expression >$.**then**. $< scope > . < condition\_branch >$
$< return >$ $\rightarrow$ **return**. $< rvalues >$
$< return >$ $\rightarrow$ **return**

$< condition\_branch >$ $\rightarrow$ **else**. $< scope >$
$< condition\_branch >$ $\rightarrow$ **elseif**. $< condition >$
$< condition\_branch >$ $\rightarrow \epsilon$

$< lvalues >$ $\rightarrow < lvalues > ."," . < lvalue >$

$< lvalues >$ $\rightarrow < lvalue >$
$< lvalue >$ $\rightarrow$ *id*
$< rvalues >$ $\rightarrow < rvalues >$ ”,”. $< rvalue >$
$< rvalues >$ $\rightarrow < rvalue >$
$< rvalue >$ $\rightarrow < expression >$

$< expression >$ $\rightarrow < expression >$. $< binary\_operator >$. $< expression\_2 >$
$< expression >$ $\rightarrow < expression\_2 >$
$< expression\_2 >$ $\rightarrow < unary\_operator >$. $< expression\_3 >$
$< expression\_2 >$ $\rightarrow < expression\_3 >$
$< expression\_3 >$ $\rightarrow$ ”(”. $< expression >$.”)”

$< expression\_3 >$ $\rightarrow$ *string*
$< expression\_3 >$ $\rightarrow$ *number*
$< expression\_3 >$ $\rightarrow$ *integer*
$< expression\_3 >$ $\rightarrow$ *id*
$< expression\_3 >$ $\rightarrow$ *id*.”(”. $< called\_parameters >$.”)”
$< expression\_3 >$ $\rightarrow$ **true**
$< expression\_3 >$ $\rightarrow$ **false**
$< expression\_3 >$ $\rightarrow$ **nil**

$< datatypes >$ $\rightarrow < datatypes >$ ”,”. $< datatype >$
$< datatypes >$ $\rightarrow < datatype >$

$< datatype >$ $\rightarrow$ **integer**
$< datatype >$ $\rightarrow$ **number**
$< datatype >$ $\rightarrow$ **string**
$< datatype >$ $\rightarrow$ **boolean**

$< unary\_operator >$ $\rightarrow$ ”#”
$< unary\_operator >$ $\rightarrow$ **not**

$< binary\_operator >$ $\rightarrow$ ” − ”
$< binary\_operator >$ $\rightarrow$ ” + ”
$< binary\_operator >$ $\rightarrow$ ” ∗ ”
$< binary\_operator >$ $\rightarrow$ ”/”
$< binary\_operator >$ $\rightarrow$ ”//”
$< binary\_operator >$ $\rightarrow$ ”..”
$< binary\_operator >$ $\rightarrow$ ” < ”
$< binary\_operator >$ $\rightarrow$ ” <= ”
$< binary\_operator >$ $\rightarrow$ ” > ”
$< binary\_operator >$ $\rightarrow$ ” >= ”
$< binary\_operator >$ $\rightarrow$ ” == ”
$< binary\_operator >$ $\rightarrow$ ” = ”
$< binary\_operator >$ $\rightarrow$ **and**
$< binary\_operator >$ $\rightarrow$ **or**
$<>$ $\rightarrow$
