LL(1) Parser Visualization

Write your own context-free grammar and see an LL(1) parser in action!

Written by Zak Kincaid and Shaowei Zhu based on http://jsmachines.sourceforge.net/machines/ll1.html

1. Write your LL(1) grammar (empty string '' represents ε):

```
A := B
B := C
B := E
B := S'
C := create  D
D := H
D := G
E := insert into L Y ;
G := M table  L N ;
H := J L ;
I := if not exists
I := ''
J := database
J := schema
L := a
L := _ L
L := ' L '
M := temperory
M := ''
```

Valid LL(1) Grammars

For any production S -> A | B, it must be the case that:

- For no terminal t could A and B derive strings beginning with t
- At most one of A and B can derive the empty string
- if B can derive the empty string, then A does not derive any string beginning with a terminal in Follow(A)

Formatting Instructions

- The non-terminal on the left-hand-side of the first rule is the start non-terminal
- Write each production rule in a separate line (see example to the left)
- Separate each token using whitespace
- $ is reserved as the end-of-input symbol, and S is reserved as an artificial start symbol. The grammar is automatically augmented with the rule S := *start* $

Debugging

- More information about the parser construction is printed on the console
- The source code follows the pseudocode in lecture. In particular, see `computeNullable`, `computeFirst`, `computeFollow`, and `computeLLTables`

Generate tables

2. Nullable/First/Follow Table and Transition Table

Nonterminal	Nullable?	First	Follow
S	✕	create, insert, select	
A	✕	create, insert, select	$
B	✕	create, insert, select	$
C	✕	create	$
D	✕	table, database, schema, temperory	$
E	✕	insert	$
G	✕	table, temperory	$
H	✕	database, schema	$
I	✓	if	
J	✕	database, schema	a, _ , '
L	✕	a, _ , '	(, values, ;, , , ', int, char, varchar, text, bool,), a, _, where, in, between, like, and, >, <, =, from
M	✓	temperory	table
N	✓	(,	; ,)
O	✕	a, _ , ' , primary, foreign	), (, , ;
P	✕	int, char, varchar, text, bool	), (, , ;
Q	✕	primary, foreign	), (, , ;
R	✓	a, _ , ' , ,	), from
S'	✕	select	$
T	✓	', a, _ , ' , ,	from
U	✕	from	;
W	✓	where	;
Y	✕	(, values	;
X	✕	a, _ , '	;
Z	✓	in, a, _ , ' , between, like, and, >, <, =	;
O'	✓	a, _ , ' , ,	)
WHEREOPR	✕	between, like, and, >, <, =	;, in, a, _ , ' , between, like, and, >, <, =
X'	✕	=, <, >	

	$	create	insert	into	;	table	if	not	exists	database	schema	a	_	'	temperory	(	)	,	int	char	varchar	text	bool	primary	key	foreign	select	*	from	where
S		S := A $	S := A $																								S := A $			
A		A := B	A := B																								A := B			
B		B := C	B := E																								B := S'			
C		C := create D																												
D						D := G				D := H	D := H				D := G															
E			E := insert into L Y ;																											
G						G := M table L N ;									G := M table L N ;															
H										H := J L ;	H := J L ;																			
I		if					I := if not exists																							
J		database, schema	a, _ , '							J := database	J := schema																			
L		a, _ , '	(, values, ;, , , ', int, char, varchar, text, bool,), a, _, where, in, between, like, and, >, <, =, from									L := a	L := _ L	L := ' L '															
M		temperory	table			M := ε									M := temperory															
N		(,	; ,)		N := ε											N := (O N)	N := ε	N := , O N												
O		a, _ , ' , primary, foreign	), (, , ;									O := L P	O := L P	O := L P										O := Q		O := Q				
P		int, char, varchar, text, bool	), (, , ;																P := int	P := char	P := varchar	P := text	P := bool							
Q		primary, foreign	), (, , ;																					Q := primary key (R)		Q := foreign key (R)				
R		a, _ , ' , ,	), from									R := L R	R := L R	R := L R			R := ε	R := , L											R := ε	
S'		select	$																								S' := select T U ;			
T		', a, _ , ' , ,	from									T := R	T := R	T := R				T := R										T := *	T := R	
U		from	;																										U := from L W	
W		where	;		W := ε																									W := where X
Y		(, values	;													Y := (R) Y														
X		a, _ , '	;									X := L Z	X := L Z	X := L Z																
Z		in, a, _ , ' , between, like, and, >, <, =	;		Z := ε							Z := L Z	Z := L Z	Z := L Z																
O'		a, _ , ' , ,	)									O' := L O'	O' := L O'	O' := L O'			O' := ε	O' := , L O'												
WHEREOPR																														
X'																														

3. Parsing

Token stream separated by spaces: create database a a ;

Start/Reset Step Forward

Stack

$, L

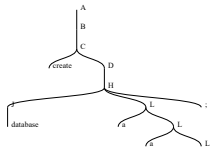
Remaining Input

; , $

Rule

Match a

Partial Parse Tree


A
B
C
create
D
H
database
L
L
L
;