Definition of Terms

by Paul B Mann

Acknowledgements

Since many of the ideas used in the development of TBNF were not entirely the author's invention it seems appropriate to give credit to those who contributed to this effort whether they knew it or not. The following publications played a key role in the development of TBNF.

1. "**Principles of Compiler Design",**
Book from Addison Wesley, 1977, by Aho and Ullman.

2. **"Compiler Construction",**
Book from Springer Verlag, 1985, by Waite and Goos.

3. **"Efficient Computation of LALR(1) Look-Ahead Sets"**
Paper from TOPLAS Oct 1982, by DeRemer and Pennello.

4. **"Optimization of Parser Tables for Portable Compilers",**
Paper from TOPLAS Oct 1984, by Dencker, Durre and Heuft.

5. **"A Human-Engineered Variant of BNF",**
Paper from Sigplan Notices 1980, by Henry Ledgard.

And finally, I wish to acknowledge Almighty God our Creator. Any talents that I have manifested in the creation of this product, have come from Him.

Definition of Terms

abstract-syntax tree.

An abstract-syntax tree is a tree structure which contains only the meaningful elements from the input language. Punctuation that is not meaningful, such as comma's and parentheses, are not part of an abstract-syntax tree. The structure of the tree is defined by the placement of node names in the grammar. Here is a grammar, an input statement and its corresponding AST.

```
{ '+' } <<
{ '*' } <<

Goal       -> [Stmt]... <eof>
Stmt       -> Exp ';'                    +> eval
           -> Target '=' Exp ';'         +> assign
Target     -> <identifier>               +> address
Exp        -> <number>                   +> number
           -> <identifier>               +> ident
           -> '(' Exp ')'
           -> Exp '+' Exp                +> add
           -> Exp '*' Exp                +> mul

y = 2 * (x + 1);

assign
+  address (y)
+  mul
   +  number (2)
   +  add
      +  ident (x)
      +  number (1)
```

AST.

Abstract-syntax-tree.

accessor.

An accessor is a symbol that causes a transition to a state. Each state has only one valid accessor. The accessor array gives the accessor symbol numbers for the states. This permits the terminal and nonterminal transition lists to contain only the state numbers. So when making a transition you must consult the accessor array to see if the token is valid. Several states may have the same accessor, but each state only has one accessor. The terminal accessor numbers are positive and the nonterminal accessor numbers are negative. 0 is the first terminal symbol number.

backsubstitution.

The process of replacing every occurrence of a nonterminal symbol in a grammar with its associated productions. This may increase the number of productions. It can be used to remove all null productions from a grammar. This gives a slight speed increase to the parser. It can also be used for eliminating conflicts in a grammar. See the following.

Grammar before backsubstitution:

```
Goal       -> Statement <eof>
Statement  -> Var '=' Exp ;
Exp        -> Var
           -> Var '+' Var
           -> Var '-' Var
```

Grammar after backsubstituting Exp:

```
Goal       -> Statement <eof>
Statement  -> Var '=' Var ';'
           -> Var '=' Var '+' Var ';'
           -> Var '=' Var '-' Var ';'
```

backtracking.

Backtracking is the process of undoing parsing actions, usually because a syntax error occurs. LR parsers do not usually do this. It is a process required by some older parsing techniques. It is undesirable.

binary.

Binary refers to data in machine readable format rather than human readable format.

binary search.

A process of searching a table for a particular value by repeatedly partitioning the remaining search area into two halves by looking at the center value and determining if it is less or greater than the search value. The table of values must be sorted for this to work.

binary skeleton.

A binary skeleton is one that creates a machine readable parser table that resides on disk and is not compiled. It is loaded into memory at run time.

chain reductions.

Chain reductions are those consecutive unit reductions of the form (a <= b) that the parser makes, but have no associated action with them and cause a waste of time. Languages that have a large number of expression operators that are defined without using operator precedence notation have a lot of chain reductions. These can be eliminated by introducing operator precedence notation into the grammar, along with ambiguous expressions of the form:

```
Exp -> Exp '+' Exp
    -> Exp '-' Exp
    -> Exp '*' Exp
    -> Exp '/' Exp
```

closure.
The process of generating from the kernel set of items in a state all possible items that may exist in that state. This is part of the state building process done in a parser generator.

command language.
A language used to "talk to" a computer or command one. A language used in an interactive way with a machine.

compiler.
A program which translates source code written in a high level language (C, Pascal, BASIC, etc.) into an executable load module suitable for execution on a particular machine (IBM, Sun, Hewlett-Packard).

conflict.
An ambiguity in a grammar. If, starting from the goal symbol, there are two or more ways to derive a pattern, and it is not clear how to distinguish one from another by using one symbol of lookahead, then there is a conflict. The parser generator must resolve this problem if the parsing algorithm is to be deterministic (without choices).

There two types of conflicts: shift-reduce and reduce-reduce. In the case of the shift-reduce, precedence is given to the shift operation (the unfinished rule) instead of the reduce operation. In the case of the reduce-reduce, precedence is given to the earlier production in the grammar.

default.
A default action in a state is the reduction to be taken if there is no possible transition for a particular input symbol. Most states have only one reduction, the default reduction. Those states that have multiple reductions, also have a default reduction. Those states that don't have a default reduction are error states.

derived.
A derived attribute is one that is passed up from a lower node in a tree. A number such as 10 may have a derived attribute of integer, whereas 10L has a derived attribute of long. This is the opposite of an inherited attribute. Inherited attributes are passed from higher nodes down to lower nodes. If 10 were an argument to a function that required a char type then 10 would inherit a char attribute.

disambiguating rules.
Those rules placed outside, usually at the top of, the grammar that specify operator precedence. It is easier to specify expressions using the ambiguous grammar notation shown below, than it is to use the unambiguous notation also shown below.

Ambiguous grammar with operator precedence:

```
{ '+' '-' } <<  /* disambiguating    */
{ '*' '/' } <<  /* rules.           */

Exp -> <identifier>
    -> <number>
    -> '(' Exp ')'
    -> Exp '+' Exp
    -> Exp '-' Exp
    -> Exp '*' Exp
    -> Exp '/' Exp
```

Unambiguous grammar:

```
Exp -> Term
    -> Exp '+' Term
    -> Exp '-' Term

Term  -> Factor
      -> Term '*' Factor
      -> Term '/' Factor

Factor  -> <identifier>
        -> <number>
        -> '(' Exp ')'
```

error recovery.

Error recovery is the process of recovering from a syntax error. When the parser encounters an error in the input, control is given to an error recovery function whose job is to fix the error or skip over it and find a place to resume parsing.

error state.

An error state is a state that has no default reduction. When the parser arrives at this state if there is no matching transition for the input symbol it must declare an error. The number of error states is the number of menus you would have if your grammar were used to generate a human interface rather than a translator. The number of error states is displayed on the screen by LRSTAR.

grammar.

A list of rules that specify a pattern or acceptable sequence of input symbols. A definition of the form (syntax) of a language. A definition of a language based human interface.

head symbol.

The symbol being defined. A nonterminal on the left of an arrow. The left side of a production.

inherited.

An inherited attribute is one that comes from a higher node. In LR parsing, which is bottom-up parsing, inherited attributes cannot be assigned in the first pass. That is why people build an abstract syntax tree. So that after the parsing is finished, one can go over the tree and assign attributes. (See derived).

input processor.
An input action or function that does special processing for a terminal symbol such as keyword match, storing of the symbol in a text table, or checking to see in which column it was found. An input processor may change the terminal number of the symbol.

interpreter.
A program which executes source code written in a high level language (C, Pascal, Ada). An interpreter does not produce object code or an executable file. It may generate intermediate code and execute that. Interpreters are desirable because they can offer debugging techniques superior to compilers. They have the source code readily available in memory. The translation time is less because they usually don't do much optimization on the code.

keyword.
Those words in a language that have special meaning to the compiler or translator. They usually cannot be used as variable names. FORTRAN and PL/I are exceptions. They are the same as reserved words. In the C language, some keywords are: if, while, for, switch, continue, break.

LALR.
LALR is a term taken from computer science parsing theory which means Look Ahead Left Right. It refers to a class of grammars that belongs in the LALR(1) category of descriptive power, greater than LR(0) but not quite as powerful as LR(1).

leaf.
An element in a tree that has no connections to lower elements. A child without children. See node.

left recursion.
Left recursion is defined in a grammar by a rule that contains the head symbol as the first tail symbol. Left recursion is the desired way to specify repetition in an LALR grammar. For example:

```
IdentifierList    -> <identifier>
                  -> IdentifierList ',' <identifier>
```

lexer.
A pattern recognizer, that recognizes character patterns instead of symbol patterns. A lexer recognizes the tokens of a language and defines the starting point and end of each token. It usually skips over blanks, newlines, and comments. It usually performs input from a file. Sometimes it even does preprocessing.

lexical.
Pertaining to characters. A lexical analyzer is the same thing as a lexer. A lexical grammar is a grammar that defines the character patterns for the tokens of a language, such as numbers, identifiers, strings, etc.

lookahead.
A symbol used by the parser to determine whether to shift (goto a new state) or reduce (make a reduction). Also, a lookahead set is created internally by the parser generator to analyze the states that are causing conflicts.

node.
A element in a tree that has connections to one or more lower elements. A parent. See leaf.

nonterminal.
A symbol of a grammar that represents other symbols of the grammar. A nonterminal is a symbol that is defined in a grammar. It may be defined to be zero or more terminals or nonterminals.

null.
A null production is a rule that has an empty right part. For example:

```
OptionalList ->
```

parser.
A parser is a pattern recognizer. A program capable of accepting input that conforms to acceptable patterns as defined by a set of rules (grammar).

production.
A rule in a grammar. A production contains an optional head symbol, an arrow (->) and zero or more tail symbols. For example:

```
ForStmt -> for '(' [Exp] ';' [Exp] ';' [Exp] ')' Stmt
```

recursion.
Repetition of itself. In reference to grammars, recursion means that a sequence of symbols is repeatable for an indefinite number of times. See left recursion, right recursion.

reduce.
A reduce action in a parser is the act of recognizing that a rule has been completed and reducing the list of symbols that make up the right side of the rule to one symbol, the head symbol for that rule.

reduce-reduce.
A type of conflict in which two or more rules have identical right sides but different head symbols. The parser generator cannot decide which head symbol to choose based on one symbol of lookahead.

reduction.
A reduce action. See reduce.

right-recursion.
Right recursion is defined in a grammar by a rule in which the head symbol is used as the last tail symbol. This specifies repetition of itself in a way that postpones all of the reductions until the last one is seen. Then all of the reductions for this rule are done in a reverse order. Sometimes this is desirable, for example:

```
Factor     -> Primary
           -> Primary '**' Factor
```

This operator precedence notation and ambiguous rule accomplishes the same thing:

```
{ '**' } >>

Exp -> Exp '**' Exp
```

root.
The top or highest node in a tree. The origin. All the elements in a tree can be visited by traversing downward from the root.

semantics.
Semantics refers to meaning rather than syntax.

shift.
A shift action in a parser is one that accepts the input symbol read by the lexer and causes a transition to a new state. The current state is added to the parse stack.

shift-reduce action.
A shift-reduce action is a parsing action that combines a shift action and a reduce action into one. This is possible when the symbol being accepted is the last one of a rule. The use of shift-reduce actions permits the elimination of reduce-only states, which usually make up about 30% of the total number.

shift-reduce conflict.
A shift-reduce conflict is a conflict between a shift action and a reduce action. The parser generator cannot decide whether to pursue the recognition of an incompleted rule or accept another rule that has been completed. Based on one symbol of lookahead the situation is still undecidable. The shift action will be given priority in this case.

skeleton.
The parser skeleton is the parser table C code with the formatting codes for the numbers in the tables.

SLR.
Simple LR. A class of grammars with less descriptive power than LALR grammars. Also a state type in which the parser generator can resolve inconsistencies by looking at the FIRST and FOLLOW sets, rather than the LALR(1) lookahead sets. See books on compiler theory for more explanation.

state.
A situation which has a set of valid inputs and certain actions associated with them, either shift and progress a new state or reduce and fall back to a previous state. There is often a default reduce action.

syntax.
The syntax refers to the form of a language. It refers to the correct order of symbols or correct sequence. Syntax does not pertain to meaning. See semantics.

tail.
A terminal or nonterminal symbol that appears on the right side of a rule.

terminal.
The terminal symbols are all those symbols that are undefined in a grammar. They represent themselves. They are the primitive elements of the language. The leaf nodes of the abstract syntax tree. The input symbols coming from the lexer.

token.
A symbol defined by a lexer. We try to use this term to mean the set of terminal symbols defined by the lexer. This is a subset of the complete set of terminal symbols. An <identifier> is a token, but a keyword is not, because the keyword is classified as an <identifier> by the lexer.

transition.
A state number indicating the next state. Each state has its own set of permissible transitions. Only the next state number is necessary for specifying a transition. You don't need to keep a list of permissible symbols, because if you have the next state number the accessor array gives you the symbol.

tree.
A tree is a data structure which facilitates visiting all the elements in the tree in several different orders. It facilitates restructuring or removing parts of the tree (optimizations). It is a very desirable structure when dealing with language.