An object oriented software tool for constructing SLR parsers

Francisco Ríos Acosta

Instituto Tecnológico de la Laguna
Blvd. Revolución y calzada Cuauhtémoc s/n
Colonia centro
Torreón, Coah; México
Contacto : friosam@prodigy.net.mx

Paper received on 18/07/08, accepted on 10/09/08.

Abstract. The task of programming SLR parsers demand writing code for process such as canonical collection of sets of items and SLR parser tables. This paper introduce a software tool called RA-SLR which allows the generation of C# code consisting of four classes : SintAscSLR, Item, Pila and SimbGram. These classes are used to build C# SLR parsers applications. This tool has a context-free grammar as input for building and visualizing the canonical collection of sets of items and SLR parser table, needed to generate the C# code. This software is part of a package compound of 3 tools : SP-PS1 for the construction of lexical analyzers, RD-NRP for the construction of top-down predictive parsers and the present tool. This software tools are based in the classes proposed by R.A. Francisco.

1 Introduction

In the following sections, the next issues are explained : generation of C# code by the software tool SP-PS1 used in construction of lexical analyzers, and the use of the software tool RA-SLR to build SLR parser C# Windows applications.
We will use as an example, the context-free grammar G={Vt, Vn, S,Φ} where :

Vt = { "inicio", "fin", "const", "id", "=", "num", ";", "cad", "entero", "real", "cadena", "var", ",", "+", "-", "*", "/", "(", ")", "leer", "visua" }
Vn = { "P", "C", "K", "R", "T", "V", "B", "L", "O", "A", "E", "M", "F", "U", "S", "I" };
S = "P"
Φ = { P->inicio C fin
C->K V O | K O | V O | O
K->const R
R->R T id = num ; | R T id = cad ; | T id = num ; | T id = cad ;
T->entero | real | cadena
V->var B
B-> B T L ; | T L ;
L->L , id | id

```
    O->O A | O U | O S | A | U | S
    A->id = E ;
    E->E + M | E – M | M
    M->M * F | M / F | F
    F->id | num | ( E )
    U->leer id ;
    S->visua l ;
    l->l , id | l , cad | id | cad   }
```

An example of the recognized language is :

```
inicio
   const
      entero MAX=10;
      cadena message="OUT OF RANGE";
   var
      real x,y;
      entero i,j,k;
      cadena name;
  visua "type value : ";
  leer i;
  visua "type value : ";
  leer j;
  k = (i + j)*3;
fin
```

2 Description of Lexico and Automata classes generated by SP-PS1.

```
class Lexico
{
    const int TOKREC = 6;
    const int MAXTOKENS = 500;
    string[] _lexemas;
    string[] _tokens;
    string _lexema;
    int _noTokens;
    int _i;
    int _iniToken;
    Automata oAFD;
    // methods definition
}
```

TOKREC	Shows the number of tokens to recognize. In this case its value is 6.
MAXTOKENS	It is a constant which limits the number of token-lexeme pairs to recognize. Its value is 500.
_lexemas	It is a one-dimension array of strings. Contains each of the recognized lexemes. It is defined in the class constructor.
_tokens	It is a one-dimension array of strings. Contains each of the recognized tokens. It is defined in the class constructor.
_lexema	It contains the value of the recognized lexeme in an acceptance action of the lexical analyzer.
_noTokens	It contains the number of recognized token-lexeme pairs during lexical analysis.
_i	It is a pointer to the next character to be read during lexical analysis.
_iniToken	It is a pointer to the first character where the next recognition begins.
oAFD	It is an object which contains the DFA's representation. In this case should exist 6 automatas in this object.

```
class Automata
{
    string _textoIma;
    int _edoAct;
    // methods definition
}
```

_textoIma	It is a reference to string. Its value is set to input text.
_edoAct	It holds the current state of the DFA used in the recognition.

SP-PS1 requires as input the regular expressions that denote the terminal symbols described on the example grammar mentioned in section 2 :

TOKEN	REGULAR EXPRESSION	Retract()	store
delim	{delim} -> [\ \n\r\t]+ [^\ \n\r\t]		-
id	{Dig} -> [0-9] {Letter} -> [A-Za-z] {Und} -> _ {id} -> {Letter} ({Letter} | {Dig} | {Und}) * [^_A-Za-z0-9]	*	token-lexeme for *id* lexeme-lexeme for *keyword*
float-num	{float} -> ([0-9]+\.[0-9]+ | [0-9]+\. | \.[0-9]+) [^0-9]	*	token-lexeme
int-num	{int} -> [0-9]+ [^0-9]	*	token-lexeme
str	{str} -> \"[^\"]*\"		token-lexeme
others	{others} -> = | ; | , | \+ | \- | * | / | \(| \)		lexeme-lexeme

3 Proposed classes for a SLR parser object.

There are 4 proposed classes generated by RA-SLR :

class SintAscSLR
class Item
class Pila
class SimbGram

SintAscSLR is the fundamental class. This class uses the objects belonging to the other classes. Class Item is used to represent the items sets that take part of the canonical collection of items. The canonical collection is an attribute of the SintAscSLR class, and represents the basis for the construction of the M parsing table. The M parsing table is not an attribute however is defined by _action and _goTo attributes of the SintAscSLR class.
Pila class represents the stack contained in the conceptual model of the bottom-up parser. Its elements are states and grammar symbols that are both objects of the SimbGram class.

```
class SintAscSLR
{
    public const int NOPROD = 40;
    public const int NODDS = 1000;
    public const int NOACTIONS = 1000;
    public const int NOGOTOS = 1000;
    string[] _vts;
    string[] _vns;
    int[,] _prod;
    int[,] _sig;
    Pila _pila;
    int[,] _action;
    int _noActions;
    int _noGoTos;
    int[,] _goTo;
    int[] _dd;
    int _noDds;
    Item [] _c;
    int _noItems;
    // methods definition
}
```

SintAscSLR class attributes description.

ATTRIBUTE	DEFINITION	
NOPROD	public const int NOPROD = 40;	Constant that represents the number of productions of the grammar.
NODDS	public const int NODDS = 1000;	Indicate the maximum number of productions for a rightmost derivation for an input sentence.
NOACTIONS	const int NOACTIONS = 1000;	Set the maximum number of rows of the _action array.
NOGOTOS	const int NOGOTOS = 1000;	Set the maximum number of rows of the _goTo array.
_vts	string[] _vts;	One dimension array of strings which have the terminal symbols of the grammar.
_vns	string[] _vns;	One dimension array of strings which have the non terminal symbols of the grammar.
_prod	int[,] _prod;	Two dimension integer array. Each row represents a production of the grammar. Each column represents the left member of the

		production, the number of Y's, and each Yi of the right member of the production A -> Y_1 Y_2 ... Y_n
_sig	int[,] _sig;	Two dimension integer array. Each row contains the FOLLOW set –set of tokens- of each syntactic grammar variable.
_pila;	Pila _pila;	Object that represents the stack used in the bottom-up parser.
_action	int[,] _action;	Represents the actions of the parsing table. Each row of the array is an action for a certain state. Its four columns represents: state, token –terminal symbol-, type of action -0 shift, 1 reduce, 2 accept-, and the number of the production when it is reduce.
_noActions	int _noActions;	Number of registered actions in the _action array.
_goTo	int[,] _goTo;	Represents the GOTO part of the parsing table. Each row of the array is a goto from a state to another depending on a non terminal symbol. The three columns represent : state, the index of the non terminal grammar symbol in _vts and the state that is transitioned.
_noGoTos	int _noGoTos;	Number of registered GOTO's in the _goTo array.
_dd	int[] _dd;	The elements of this one dimension integer array correspond to a number of production used in the rightmost derivation of the sentence.
_noDds	int _noDds;	Number of registered productions in the _dd array.
_c	Item [] _c;	Objects array of the Item class. Represents the canonical collection of items used by the SLR bottom- up parser. Each element _c[i] is a group of items, being I_0,

		$I_1, I_2, \ldots, I_n$.
noItems	int _noItems;	Number of registered items in the canonical collection _c.

```
class Item
{
    int[,] _item;
    int _noItems;
    // methods definition
}
```

Item class attributes description.

ATRIBUTO	DEFINICIÓN	
_item	int[,] _item;	Two dimension integer array. NOPROD sets the row's dimension . There are 2 columns : the first one represents the number of the production and the second is the position of the point in the item.
_noItems	int _noItems;	Indicates the number of registered items for this object.

SintAscSLR class methods description.

METHODS	DESCRIPTION
public void Inicia()	Initializes the attributes of the class just as a constructor does. Sets the maximum number of items of the canonical collection of items *_c* to one thousand sets of items. It also reserves the space needed by each *item* object from the attribute *_c*. Creates the item I_0 S'->.S and calculates its closure. Creates the item I_1 *S'->.S* and assigns it using one of the constructors from the class *Item*. Calculates the rest of the items of the canonical collection *_c* using the method *AgregarConjItems(i)*. Creates the *goTos* of the item *S'->.S* corresponding to the augmented grammar. Generates shift or reduce, corresponding to the actions section of the parsing table, using the methods *GeneraCambios(i)*, and *GeneraReducciones(i)*.
public Item Cerradura(Item oItem)	Calculates the *oItem* object closure. oItem is received as a parameter. Makes use of the method *AgregarItems(I,ref oItem)*.

public void AgregarConjItems(int i)	This method is used when we calculate the canonical collection of items *_c* in the method *Inicia( )*. Its function, amongst others, is to calculate and add new items to *_c* taking as reference the set of items *_c[i]*. Receives an integer *i* as parameter which corresponds to the index of the set of items *Ii* of the collection *_c*. This *_c[i]* is used to generate new sets of items. Another important task done by this method, is to calculate the *goTos* of the state *Ii* represented by *_c[i]*. In this method are also used the methods *Cerradura() and EstaNuevoItem()*.
public int AgregarItems(int i, ref Item oItem)	This method is used by the method *Cerradura()*. Receives as parameters the index *i* which corresponds to the number of item in the object *oItem*. Its purpose is to add new items to the closure when it corresponds, the new items are generated by the item with index *i*. Makes use of the method *ExisteItem()* to determine whether to add or not if the item already exists in the object *oItem*.
public bool EstaNuevoItem(Item oNuevoItem,out int edoYaExiste)	This method is used by the method AgregarConjItems(int i). It searches inside the canonical collection *_c* for the object *oNuevoItem* received as parameter. It returns in the received parameter *edoYaExiste*, the state number *Ii* in the collection *_c* which corresponds to the state that already exists. It returns *true* if the object *oNuevoItem* exists in *_c*, if not it returns *false*.
public void GeneraReducciones(int i)	This method generates the set of reduce actions if there are any, produced by the set of items *Ii* of the canonical collection *_c*. Receives as parameter the index *i* corresponding to the state *Ii*.
public void GeneraCambios(int i)	This method generates the shift actions produced by the set of items *Ii* of the canonical collection *_c* if they exist. It receives as parameter the index *i* corresponding to the state *Ii*.
public int Analiza(Lexico oAnalex)	This method is the C# code of the algorithm for the SLR bottom-up parser, using the proposed R.A.F. classes. The method receives the object *oAnaLex* as parameter which contains the rec-

	ognized tokens in the previous stage of lexical analysis.
public string Accion (string s, string a)	This method returns a string that corresponds to the type of action according to the received parameters *s* and *a*. The parameters represent *s* for state, and *a* represents the token for which there is a registered action in the ACTION section of the parsing table M.
public void SacarDosBeta(string accion)	This is the C# code for the procedure indicated in the book Compilers : Principles, Techniques and Tools in the ascendant parser algorithm.

The code in the class SintAscSLR does not contain the definition for the number of productions NOPROD, neither the attributes:

```
string[] _vts;
string[] _vns;
int[,] _prod;
int[,] _sig;
```

These attributes will be generated by RA-SLR.

Item class methods description.

METHOD	DESCRIPTION
public int NoProd(int i)	Returns the number of production of the item *i*. The number of production is found in the column 0 of each row.Each row denotes an item.
public int PosPto(int i)	Returns the position of the point in the item *i*. The position of the point is found in the column 1 of each row. Each row denotes an item.
public bool ExisteItem(int noProd, int posPto)	The method returns true if the item with a production number *noProd* and the position of the point *posPto*, is already found in the array *_item* of the object of the *Item* class. I returns false if the item is not found.
public void Agregar(int noProd, int posPto)	This method adds a new item with a production number equal to *noProd* and a poin position equal to *posPto* to the array *_item*. noProd and posPto are received as parameters.
public int NoItems	*NoItems* is a read only property which returns the number of items of the object of

	the Item class.	

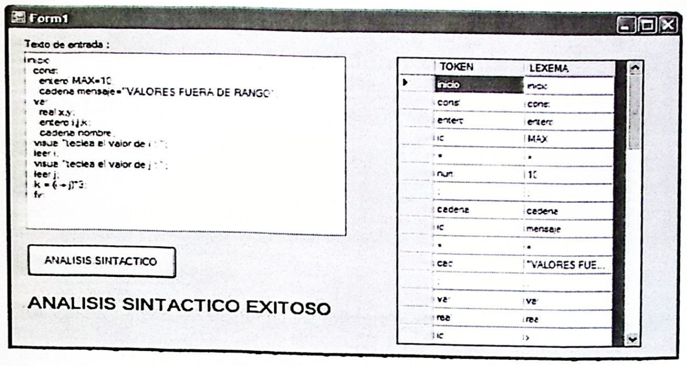
4 Making the Windows C# application for a SLR parser which recognizes the language generated by the example context-free grammar.

To build the applicaton follow the next steps :

- Create a new Visual C# project.
- Add the components Label, TextBox, DataGridView and Button shown in fig. 1.
- Include the Lexico and Automata classes generated by SP-PS1.
- Include the SintAscSLR, Item, Pila and SimbGram classes generated by RA-SLR.

We should also include the definition of the oAnaLex and oAnaSintAscSLR objects in Form1. As well as the code for the button1 Click event.

```
public partial class Form1 : Form
{
    Lexico oAnaLex = new Lexico();
    SintAscSLR oAnaSintAscSLR = new SintAscSLR();
    ...
    ...
    private void button1_Click(object sender, EventArgs e)
    {
        oAnaLex.Inicia();
        oAnaLex.Analiza(textBox1.Text);
        oAnaSintAscSLR.Inicia();
        if (oAnaSintAscSLR.Analiza(oAnaLex) == 0)
           label2.Text = "ANALISIS SINTACTICO EXITOSO";
        else
           label2.Text = "ERROR DE SINTAXIS";
    }
}
```



Texto de entrada :

```
inicio
  cons:
    entero MAX=10
    cadena mensaje="VALORES FUERA DE RANGO"
  var
    real x,y;
    entero i,j,k;
    cadena nombre;
  visua "teclea el valor de i :";
  leer i;
  visua "teclea el valor de j :";
  leer j;
  k = (i+j)*3;
  fin
```

ANALISIS SINTACTICO EXITOSO

TOKEN	LEXEMA
inicio	inicio
cons	cons
entero	entero
id	MAX
=	=
num	10
;	;
cadena	cadena
id	mensaje
=	=
cad	"VALORES FUE...
;	;
var	var
real	real
id	x

Fig. 1. Successful syntactic analysis using code generated by RA-SLR.

5 Conclusions.

The construction of the canonical collection of items for the example grammar it would take too much time due to the fact that it is composed of 80 sets of items.
RA-SLR produces a reliable code tested for many grammars of regular size regarding the number of productions.
The software can be also used in the teaching and learning of the techniques to build a SLR parser.

References.

1. Compilers Principles, Techniques, and Tools. Aho, Sethi, Ullman.
2. Object Oriented Modeling and Design. Rumbaugh, Blaha, Premerlani, Eddy, Lorensen.
3 The Theory and Practice of Compiler Writing. Tremblay, Sorenson.
4. Lex and Yacc. Levine, Mason, Brown.
5. A compact guide to lex & Yacc. Thomas Niemann.