Laboratorio di Compilatori

Generazione automatica di parser per linguaggi non lineari

- Linguaggi simbolici bidimensionali
- Grammatiche posizionali e relazioni spaziali
- Uso di Yacc con grammatiche posizionali

LINGUAGGI LINEARI vs LINGUAGGI NON LINEARI

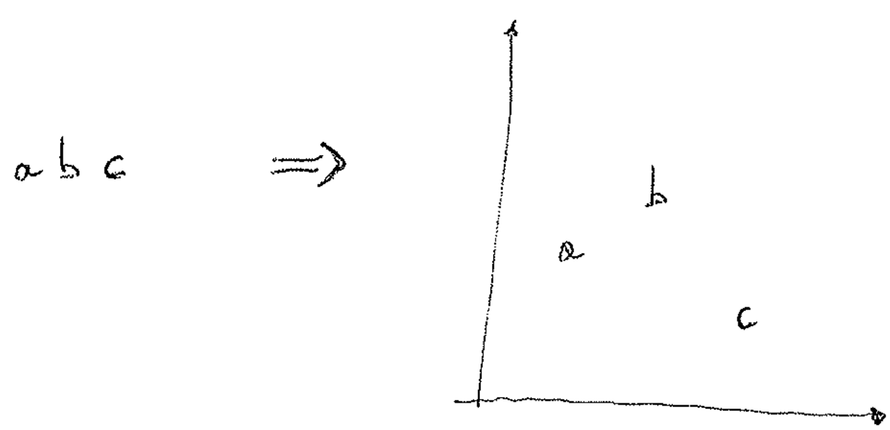
- LINGUAGGIO (FORMALE) : "un insieme di stringhe su un alfabeto di simboli"

Nei linguaggi lineari gli elementi sono stringhe (concatenazione di simboli testuali) ⟹ siamo in 1-dimensione

Rottura della linearità : i simboli non sono più disposti in una stringa ma nel Piano Cartesiano

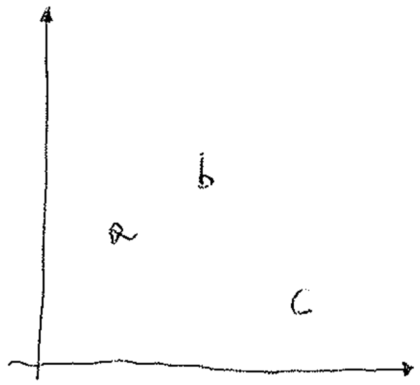
- LINGUAGGIO SIMBOLICO DIMENSIONALE : "un insieme di sentenze costituite da simboli arrangiati nel Piano Cartesiano attraverso relazioni spaziali"

⟹ siamo in 2-dimensioni



SIMBOLI e RELAZIONI SPAZIALI

- SIMBOLO ≡ coppia (carattere testuale, posizione (x,y))



$(a, (2,3))$

$(b, (4,4))$

$(c, (6,1))$

- RELAZIONE SPAZIALE ≡ coppia di interi (n,m):

Data una relazione spaziale REL $\equiv (n, m)$ e due simboli a e b con posizioni (i,j) e (k,h) rispettivamente, allora

$$a \text{ REL } b \text{ vale} \iff \begin{array}{l} k = i + n \\ h = j + m \end{array}$$

ESEMPI

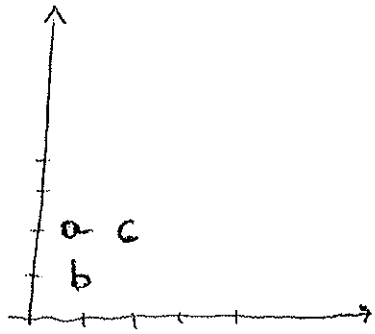
- $RIGHT \equiv [1,0]$

  $DOWN \equiv [0,-1]$

  $UP \equiv [0,1]$

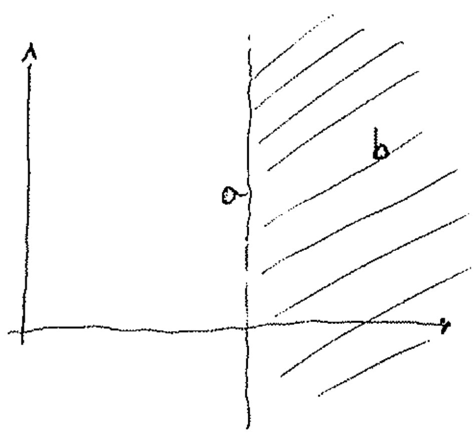
  $DIAG \equiv [1,1]$

  ⋮



- La relazione $[0,0]$ è proibita (overlap di simboli)

- Relazioni "puntuali" vs relazioni "generali":



$$general\ RIGHT \equiv \{(m,n) \,/\, m > 0\}$$

- Linguaggi Lineari
  - → simboli: (carattere testuale, posizione del carattere nella stringa)
  - → unica relazione: $[1,0]$

RAPPRESENTAZIONE RELATIVA e ASSOLUTA

SENTENZA

RAPPRESENTAZIONE RELATIVA

È una stringa che alterna simboli e identificatori di relazioni spaziali

- Non è unica

↓

b UP^0 a $RIGHT^0$ c

b UP^0 a $DIAG^1$ c

a $DOWN^0$ b $RIGHT^1$ c

⋮

RAPPRESENTAZIONE ASSOLUTA

È una lista dei simboli con i valori assegnati delle posizioni

- È unica

↓

a	(1,2)
b	(1,1)
c	(2,2)

Abstract

While in a string grammar the only possible spatial relation is the string concatenation, in a positional grammar other spatial relations can be defined and then used for describing high dimensional languages. In this paper we characterize a new class of positional grammars, the extended pLALR grammars, which can be translated into traditional LALR context free grammars with positional actions. A positional action is a procedure implementing a spatial relation. In this way, the parser for an extended pLALR language can be generated automatically by the tool Yacc with no more efforts. Moreover, we show that the class of extended pLALR grammars properly contains the class of pSLR grammars for which a Yacc implementation has already been given.

1. Introduction

Recently, much of research in the area of visual languages [1, 3, 4, 8, 11] has focused on the investigation of formal grammars [6, 7, 9, 12, 15] that aid in the generation of visual language parsers [5, 6, 7, 10, 13, 14, 15].

Positional grammars [6, 7] have been introduced as a powerful formalism for specifying the syntax of visual languages and, more generally, of multi-dimensional languages. While in a string grammar the only possible spatial relation is the string concatenation, in a positional grammar other spatial relations can be defined and then used for describing high dimensional languages. Then, in order to parse a sentence generated by a positional grammar the traditional parsing techniques have to be extended to take into account the spatial relations among the objects of the sentence. In [6, 7] a parser for positional grammars has been constructed by adding a column to the LR parsing table. For each state, this column contains an entry with the position of the next symbol to be parsed. The main difference with the traditional parsers is in the access to the input which is no longer sequential but driven by the spatial relations in the column position of the states.

To avoid the efforts that must be accomplished to obtain positional parsing techniques, in this paper we characterize a new class of positional grammars (*extended pLALR grammars*) which can be automatically translated into context free grammars with positional actions. A positional action is a call to a procedure that implements a spatial relation. At each step of the parsing process, the execution of a positional action allows to navigate inside the input to reach the next symbol to be parsed.

The translation of an extended pLALR grammar into a context free grammar with positional actions is carried out by two phases. First, the positional grammar PG is translated into a string grammar SG whose LALR parsing table is shown to be "functionally equivalent" to the extended pLALR parsing table of PG restricted to the *action* and *goto* parts. Secondly, positional actions are inserted in the productions of SG to simulate the role of the spatial relations corresponding to the entries in the *position* part of the extended pLALR parsing table.

Once a context free grammar with positional actions has been produced, a parser for the corresponding extended pLALR two-dimensional language can be generated automatically by the tool Yacc with no more efforts.

The only other investigated class of positional grammars for which the tool Yacc can automatically generate a parser is the class of pSLR grammars presented in [6, 7]. We show that the class of extended pLALR grammars properly contains the class of pSLR grammars.

The paper is organized as follows. In Section 2 the definition of positional grammar is recalled. In Section 3 we characterize the class of extended pLALR grammars giving algorithms for the generation of the extended pLALR parser. In Section 4 we show how it is possible to translate an extended pLALR grammar into a context free string grammar with positional actions, and to use Yacc for the automatic generation of the parser. Moreover, an example showing that the class of extended pLALR grammars includes the class of pSLR grammars is given. The conclusions and further research are outlined in Section 5.

2. Positional grammars

While in a string grammar the only possible spatial relation is the string concatenation, in a positional grammar other spatial relations can be defined and then used for describing high dimensional languages. If we restrict our attention to symbolic two-dimensional languages, the position in the cartesian plane of a symbol with respect to another one can be described by spatial relations defined as follows.

Definition 2.1 A *spatial relation* REL is defined by a pair (m, n) (denoted by REL = $\mathbf{R}_{(m,\ n)}$) where m and n are integers, such that, given tokens a and b with positions (i, j) and (k, h) respectively, a REL b iff k = i+m and h = j+n.

We assume that every token is associated to one and only one spatial position. This allows us to construct spatial operators implementing the spatial relations such that the spatial operator REL(i, j) = (k, h) iff a REL b and a and b have positions (i, j) and (k, h), respectively.

A context free positional grammar (simply positional grammar from now on) is defined as follows.

Definition 2.2 A *positional grammar* PG is specified by a six-tuple (N, T, S, P, POS, PE) where:

N is a finite non-empty set of *non-terminal* symbols,
T is a finite non-empty set of *terminal* symbols (or *tokens*) with $N \cap T = \varnothing$,
$S \in N$ is the *starting* symbol,
P is a finite set of *productions*,
POS is a finite set of *spatial relations*,
PE is an *evaluation rule.*

Each production in P has the following form:

$$A \rightarrow x_1 REL_1 x_2 REL_2 \ldots REL_{m-1} x_m$$

where $m \geq 1$, $A \in N$, $x_i \in N \cup T$ and $REL_j \in POS$, with $1 \leq i \leq m$ and $1 \leq j \leq m-1$.

Given a positional grammar, we write $\alpha \Rightarrow \beta$ if $\alpha = \gamma A \delta$, $A \rightarrow \eta$ is a production and $\beta = \gamma\eta\delta$. We write $\alpha \Rightarrow^* \beta$ (β is derived from α) if there exist $\alpha_0, \ldots, \alpha_m$ ($m \geq 0$) such that $\alpha = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_m = \beta$.

The sequence $\alpha_0, \alpha_1, \ldots, \alpha_m$ is called a *derivation* of β from α. A *positional sentential form* is a string $\alpha = x_1 REL_1 x_2 REL_2 \ldots REL_{m-1} x_m$ where $x_i \in N \cup T$ and $REL_j \in POS$, such that $S \Rightarrow^* \alpha$. A *positional sentence* is a positional sentential form which does not contain non-terminal symbols. A *picture* is the result of the evaluation of a positional sentence by applying the evaluation rule PE. In this paper, we will always refer to the linear evaluation rule defined as follows.

Definition 2.3 A *linear evaluation rule* is a function whose input is a positional sentence:

$$a_1 REL_1\ a_2 REL_2 \ldots REL_{m-1}\ a_m$$

and its output is a picture whose symbols $a_1, a_2, \ldots, a_m$ are arranged in the two-dimensional space such that $a_i\ REL_i\ a_{i+1}$ for $1 \leq i \leq m-1$.

The *pictorial language* L(PG) is the set of pictures generated by the positional grammar PG.

Example 2.1 Let us consider the following positional grammar for the vertical concatenation of two strings both of the type "$a \ldots a b$".

N = {S, C}
T = {a, b}
POS = {HOR, VER} where:
HOR = $\mathbf{R}_{(1,\ 0)}$ and VER = $\mathbf{R}_{(0,\ -1)}$
PE is the linear evaluation rule
P = { S → A VER A
A → a HOR A
A → b }

It is easy to verify that the following picture is in the described language.

```
aaaaaab
       aaaab
```

3. Extended pLALR grammars and parsers

In this section we will define a new class of positional grammars for which it is possible to construct a parser by an automatic translation into string grammars with actions and the use of the Yacc tool. This class extends the pSLR grammars presented in [6, 7] and overlaps with the class of interactive pLALR grammars presented in [14]. Let us recall the basic concepts.

Definition 3.1 Given a positional grammar PG = (N, T, S, P, POS, PE), its *augmented positional grammar* is PG' = (N', T, S', P', POS', PE) where:

$S' \notin N$
$N' = N \cup \{S'\}$
$P' = P \cup \{S' \rightarrow S\}$
$POS' = POS \cup \{SP, ANY\}$,

where SP and ANY are fictitious spatial relations whose corresponding spatial operators always return the position of the first token of the input and the end-of-input marker "$", respectively.

Definition 3.2 Let REL be a spatial relation and a be a token. A *spatial token* is a pair (REL, a) denoted by REL_a. For each spatial token **a** = REL_a we will denote by r(**a**) the first component of the pair, i.e. its *spatial relation part* REL, and by t(**a**) the second component, i.e.

its *token part a*. Analogously, a *spatial non-terminal* is a pair (REL, A) formed by a spatial relation and a non-terminal and it will be denoted by REL_A.

Definition 3.3 Let γ be a positional sentential form. FIRST(γ) is defined as the set of tokens *a* that begin the positional sentences derived from γ. If γ is preceded by a spatial relation REL, then FIRST is defined such that FIRST(REL γ) = FIRST(γ).

A pLR(1) *item* of a positional grammar PG consists of three components. The first component is a spatial relation; in analogy to [2], the second one is a production from PG with a dot at some position in the right-hand side of the production. A dot, however, can never be between a spatial relation identifier and the following terminal or non-terminal on its right. The third component is a spatial token indicating the look-ahead of the production.

Thus, a production $A \rightarrow x\ REL_1\ y\ REL_2\ z$ yields the following four types of items:

$[REL_0, A \rightarrow .x\ REL_1\ y\ REL_2\ z,\quad \mathbf{a}]$
$[REL_0, A \rightarrow x\ .REL_1\ y\ REL_2\ z,\quad \mathbf{a}]$
$[REL_0, A \rightarrow x\ REL_1\ y\ .REL_2\ z,\quad \mathbf{a}]$
$[REL_0, A \rightarrow x\ REL_1\ y\ REL_2\ z.,\quad \mathbf{a}]$

where the first component REL_0 is the spatial relation to reach A, i.e. REL_0 precedes A in some production of PG; the second component indicates how much of a production has already been scanned at a given point in the parsing process; the third component **a**, the look-ahead, indicates which symbol is expected as next and where it is expected, once the production is completely scanned and reduced.

Before presenting the algorithms for the construction of an extended pLALR parser, let us give a new definition of core of a pLR(1) item.

Definition 3.4 Given a set of items I, the *extended core* of I is the set of the first two components of each item in I.

For example, if {[HOR, A → E., VER_*a*], [HOR, A → E. VER *c*, VER_*a*]} is a set of items, then {[HOR, A → E.], [HOR, A → E. VER c]} is its extended core.

Note that the previous definition is different from the non-extended definition given in [14] which did not include the first component of an item in the core.

The algorithm for the construction of the sets of extended pLALR(1) items for an augmented positional grammar PG' can be easily obtained from the corresponding one in [14] with the new definition of extended core and keeping the spatial token formalism only for the look-ahead symbols. For completeness the resulting modified algorithm is given below.

Algorithm 3.1. Construction of the extended pLALR(1) items.
Input: An augmented positional grammar PG'.
Output: The sets of extended pLALR(1) items.
Method: The items are constructed by the main routine *items* which uses the procedures *closure* and *goto*

```
function closure (I);
begin
  repeat
    for each item [REL_0, A → α.REL Bβ,   a]
          with α ≠ ε  or [REL, A → .Bβ,   a] in I,
        each production B → γ in PG' and
        each token b in FIRST(β)
          such that [REL, B → .γ, b] is not in I, with:
          b = a          if β = ε
          b = REL'_b     if β ≠ ε and REL' is the first
                         spatial relation that appears in β
        do  add [REL, B → .γ,   b] to I;
  until no more items can be added to I;
  return I
end;

function goto (I, x);
begin
 let J = {item | item = [REL_0, A → α REL x.β, a] with
        α ≠ ε, and [REL_0, A → α.REL x β, a] ∈ I} ∪
        {item | item = [REL, A → x.β, a] and
             [REL, A → .xβ,   a] ∈ I};
 return closure (J)
end;

procedure items (PG');
begin
 C = {closure ({[SP, S' → .S,   ANY_$]})};
 repeat
   for each set of items I in C and each grammar symbol x
        such that goto (I, x) is not empty and not in C do
             add goto (I, x) to C;
 until no more sets of items can be added to C;
 for each extended core of items of items in C do
     find all items having that extended core and
     replace them by their union;
end
```

The algorithm begins by computing the closure of {[SP, S' → .S, ANY_$]}. Since the LALR parsing technique requires the merging of states with common cores, the last "for loop" in procedure *items* replaces all the sets having the same extended core by their union.

Because of the new definition of core, the extended pLALR technique will merge a smaller number of items than the interactive pLALR technique. However, the two techniques are not directly comparable. In particular, an interactive pLALR grammar allows positional conflicts since they can be successively solved by the user interaction. In this paper we reduce the parsing of positional grammars to the traditional LALR parsing technique in order to exploit the existing tools. The same has already been done for the subclass of pSLR positional grammars [6, 7] corresponding to the traditional SLR string grammars. Since an LALR parsing table does not allow any conflicts, the *position* column needed for extended pLALR parsing tables must contain single entries.

The following algorithm which constructs an extended pLALR parsing table is an extension of Algorithm 4.11 in [2] with the addition of the construction of a column *position* whose entries are spatial relation names. In this way the corresponding spatial operators to obtain the position for the next possible input will be associated to each state of the automaton.

Algorithm 3.2. Construction of the extended pLALR Parsing Table.
Input: An augmented positional grammar PG'.
Output: The extended pLALR parsing table functions *action*, *goto* and *position* for PG'.
Method:
1) Construct $C = \{I_0, I_1, ..., I_n\}$, the collection of sets of extended pLALR(1) items as described in Algorithm 3.1.
2) State i of the parsing table is constructed from I_i. The parsing table actions and positions for state i are determined as follows:
a) If $[REL_0, A \rightarrow \alpha.REL\ a\ \beta,\ b_1, ..., b_m]$ with $\alpha \neq \varepsilon$ or $[REL, A \rightarrow .a\beta,\ a_1, ..., a_m]$ are in I_i and *goto* $(I_i, a) = I_j$ then set *action* [i, a] to "shift$_j$" and insert REL in *position* [i]. Here '*a*' is required to be a terminal.
b) If $[REL_0, A \rightarrow \alpha.,\ b_1, ..., b_m]$ is in I_i and $A \neq S'$, then for each look-ahead b_k set *action* [i, t(b_k)] to "reduce$_{A\rightarrow\alpha}$" and insert r(b_k) in *position* [i]. Here each 'b_k' is required to be a spatial token.
c) If [SP, S' → S., ANY_$] is in I_i, then set *action* [i, $] to "accept".
If there is a parsing-action or parsing-position conflict, the algorithm fails and the grammar is said not to be extended pLALR. Note that a double entry {REL, ANY} in the column position is not considered to be a conflict.
3) The goto transitions for state i are determined for non-terminals X using the rule:

if *goto* $(I_i, X) = I_j$ then *goto* [i, X] = j.

In an extended pLALR parsing table, the entry "accept" indicates a conditional acceptance and it is actually a call to a procedure that returns "success" if and only if all the tokens of the picture input have been considered in the parsing process. A spatial relation REL ≠ ANY in the *position* column represents a call to the corresponding spatial operator that takes in input the position of the last parsed token and calculates the position of the next token to parse; in the case no token is in that position, REL behaves as ANY, i.e., it returns the "position" of the end-of-input marker $. Hence, every double entry {REL, ANY} can be replaced with {REL}.

Example 3.1 Let us consider the following positional grammar:

(1) S → *a* HOR A HOR *d*
(2) S → *b* VER A VER *e*
(3) A → *f* VER B HOR *h*
(4) A → *g* VER B HOR *i*
(5) A → *c*
(6) B → *b*

It can be seen that the grammar is extended pLALR with 23 items. Items 4, 8, 15 and 18 are shown below:

I_4: { [HOR, A → c . , HOR_d] }
I_8: { [VER, A → c . , VER_e] }
I_{15}: { [VER, B → b . , HOR_h] }
I_{18}: { [VER, B → b . , HOR_i] }

The items I_{15} and I_{18} are an example of merging states according to the extended pLALR(1) technique.
Note that if the non-extended definition of core given in [14] is used, then also the states I_4 and I_8 would merge producing a positional conflict due to a double entry {HOR, VER} in the column position of the merged state.

Example 3.2 Let us consider the following positional grammar:

(1) S → A VER B
(2) A → *a*
(3) B → A HOR *c*
(4) B → *a* HOR *d*

It is an extended pLALR grammar as shown by its extended pLALR parsing table in Figure 1 and the "goto graph" in the Appendix.

St.	*action*				*goto*			*pos*
	a	*c*	*d*	*$*	*S*	*A*	*B*	
I0	s2				1	3		SP
I1				acc				ANY
I2	r2							VER
I3	s6					5	4	VER
I4				r1				ANY
I5		s7						HOR
I6		r2	s8					HOR
I7				r3				ANY
I8				r4				ANY

Figure 1. An extended pLALR parsing table

To conclude this Section we need to provide the parsing algorithm that takes in input an extended pLALR parsing table and a picture, and gives in output the parse for the input, if accepted.

In analogy to the traditional parsing techniques, we can apply the same general algorithm of positional LR parsing to extended pLALR parsing tables, which have the same format of the Simple LR parsing tables defined in [6, 7].

4. Positional grammars as string grammars

In this section we will show that it is possible to translate an extended pLALR grammar into a context free grammar with *positional actions* in order to create a parser for it through the tool Yacc. A positional action is a call to a procedure that implements a spatial operator as defined above.

The translation of an extended pLALR grammar PG into a context free grammar with positional actions is carried out by two phases. First, PG is translated into a string grammar SG whose LALR parsing table will be shown to be "functionally equivalent" to the extended pLALR parsing table of PG restricted to the *action* and *goto* parts. Secondly, positional actions are inserted in the productions of SG to simulate the role of the spatial operators corresponding to the entries in the *position* part of the extended pLALR parsing table.

4.1 From extended pLALR grammars to string grammars

In order to translate PG into SG, we first need to define a function Σ that allows the translation of the positional formalism into a linear formalism embedding the spatial relation names directly in new grammar symbols seen as spatial tokens and spatial non-terminals.

Definition 4.1 Let $\alpha = REL_0\ a_0\ REL_1\ a_1\ \ldots\ REL_m\ a_m$ be a sequence of spatial relations and grammar symbols of a positional grammar. Σ is a function which takes in input α and gives in output the sequence:

$$REL_0\_a_0\ REL_1\_a_1\ \ldots\ REL_m\_a_m$$

formed by spatial tokens and spatial non-terminals.

The next function POSPRECEDE applied to a non-terminal A, calculates the set of spatial relations that can appear immediately to the left of A in some positional sentential form, that is, the set of spatial relations from which A can be reached.

Definition 4.2 Let P be the set of productions of a positional grammar. The function POSPRECEDE is constructed as follows:

1. Place SP in POSPRECEDE(S), where S is the starting symbol;
2. If there is a production "$A \rightarrow \alpha$ REL B β" in P, then REL is placed in POSPRECEDE(B);
3. If there is a production "$A \rightarrow B\ \alpha$" in P, then everything in POSPRECEDE(A) is also in POSPRECEDE(B).

Algorithm 4.1. The translation from PG to SG.

Input: An extended pLALR grammar PG = (N, T, S, P, POS, PE).
Output: A string grammar SG = (N_1, T_1, S_1, P_1).
Method: The string grammar SG is obtained by applying the following steps:
1. For each production " (i) $A \rightarrow x\ \alpha$" in P insert in P_1 the productions:

$$\text{"}(i_1)\ REL_1\_A \rightarrow REL_1\_x\ \alpha^*\text{"}$$

$$\vdots$$

$$\text{"}(i_k)\ REL_k\_A \rightarrow REL_k\_x\ \alpha^*\text{"}$$

where $\alpha^* = \Sigma(\alpha)$ and $REL_1, \ldots, REL_k$ are the spatial relations in POSPRECEDE(A)
2. Set the starting symbol S_1 to SP_S, and for each spatial token or spatial non-terminal REL_x occurring in a production of P_1 insert REL_x in N_1 if $x \in N$, or in T_1 if $x \in T$.

Example 4.1 Let us consider the extended pLALR grammar of Example 3.2. The function POSPRECEDE for the non-terminals of that grammar is defined as follows:
POSPRECEDE(S) = {SP},
POSPRECEDE(A) = {SP, VER},
POSPRECEDE(B) = {VER}.
The corresponding string grammar is then:

(1) SP_S → SP_A VER_B
(2_1) SP_A → SP_*a*
(2_2) VER_A → VER_*a*
(3) VER_B → VER_A HOR_*c*
(4) VER_B → VER_*a* HOR_*d*

4.2. Results of equivalence

In this subsection we give some intuitive results of equivalence between the class of extended pLALR grammars and the one produced by applying Algorithm 4.1.

Using the Algorithm 4.9 in [2], it is possible to construct the sets of LR(1) items and the *goto* graph for the string grammar SG.

Given a positional grammar PG and the string grammar SG derived from PG by the previous algorithm, it can be verified that the corresponding pLR(1) and LR(1) sets of items and *goto* graphs, G_{PG} and G_{SG}, are equivalent in the sense that a pLR(1) item [REL, $A \rightarrow x . \alpha$, a] is in G_{PG} iff an LR(1) item [REL_A $\rightarrow$ REL_x . α^*, a], with $\alpha^* = \Sigma(\alpha)$, is in G_{SG}. Moreover, every edge (I_i, I_j) in G_{PG} has label 'x' iff the corresponding edge (I'_i, I'_j) in G_{SG} has label 'REL_x', where REL is the spatial relation in *position*$[I_j]$.

Due to the extended core definition, the merges produced by the extended pLALR technique on PG will be the same as the ones produced by the LALR technique on SG. Hence, the extended pLALR graph for PG and the LALR graph for SG are still equivalent. Note that this is not true in the case of a non-extended definition of the core since more merges would be allowed in G_{PG} than in G_{SG}.

At this point, it is easy to verify that the LALR parsing table of a string grammar SG derived by a positional grammar PG as shown above is functionally equivalent to the extended pLALR parsing table of PG restricted to the *action* and *goto* parts.

Further, the LALR parsing table for SG becomes the extended pLALR parsing table for PG, restricted to the *action* and *goto* parts, by applying the following simple transformations. First, every action "reduce i_l" is replaced with "reduce i" and then, all the columns referring to spatial tokens (non-terminals) REL_1_x, .., REL_n_x with the same token (non-terminal) part are unified under only one column named x. The last transformation will not cause conflicts. In fact, any state of the LALR parsing table cannot present actions for REL_1_x and REL_2_x simultaneously, otherwise the corresponding state in the extended pLALR parsing table would have a double entry $\{REL_1, REL_2\}$ in the column *position*.

St.	*action*					*goto*			
	SP_a	*VER_a*	*HOR_c*	*HOR_d*	*ANY_$*	*SP_S*	*SP_A*	*VER_A*	*VER_B*
I0	s2					1	3		
I1					acc				
I2		$r2_1$							
I3		s6						5	4
I4					r1				
I5			s7						
I6			$r2_2$	s8					
I7					r3				
I8					r4				

Figure 2. An LALR parsing table

The equivalence between the PG and SG parsing tables is clear when comparing Figure 1 and Figure 2, which show the extended pLALR and the LALR parsing tables of the corresponding grammars defined in the Examples 3.2 and 4.1, respectively.

4.3 Insertion of the positional actions in SG

To be complete, the translation of PG must involve also the information specified by the spatial relations. Thus, the next step consists in the addition of positional actions in the right-hand side of the productions of SG.

The final translation algorithm uses the following function RFOLLOW which is defined similarly to the function FOLLOW in [2].

Definition 4.3 Let SG = (N_1, T_1, S_1, P_1) be a string grammar produced by Algorithm 4.1. The function RFOLLOW on SG is constructed as follows:

1. Place ANY in RFOLLOW(SP_S), where SP_S = S_1;
2. If there is a production:
   "REL_1_A $\rightarrow \alpha^*$ REL_2_B REL_3_x β^*"
   in P_1, then REL_3 is placed in RFOLLOW(REL_2_B);
3. If there is a production "REL_1_A $\rightarrow \alpha^*$ REL_2_B" in P_1, then everything in RFOLLOW(REL_1_A) is also in RFOLLOW(REL_2_B).

Algorithm 4.2. The translation from PG to a string grammar with positional actions.

Input: An extended pLALR grammar PG = (N, T, S, P, POS, PE) and the set of procedures for the spatial operators corresponding to the spatial relations in POS.
Output: A string grammar SG° = (N°, T°, S°, P°) whose productions are extended with positional actions.
Method: The string grammar SG° is obtained by applying the following steps:

1. Apply Algorithm 4.1 to PG to obtain the string grammar SG = (N_1, T_1, S_1, P_1).
2. Set N° = N_1, T° = T, S° = S_1, and P° = P_1.
3. Each time a symbol 'x' $\in$ T appears as a token part in a production *p*: "REL_1_A $\rightarrow \alpha^*$ REL_2_x REL_3_y β^*" of P°, replace "REL_2_x" with "x {REL_3()}" in *p*, where

REL3() is the procedure corresponding to the spatial relation $REL_3 \in POS$.

4. Each time a symbol 'x'$\in$ T appears as a token part in a production *p*: "REL_1_A $\rightarrow \alpha^*$ REL_2_x" of P_1, replace "REL_2_x" with "x {REL()}" in *p*, if RFOLLOW(REL_1_A) is equal to {REL} or {REL, ANY}.

Note that, as the original positional grammar is extended pLALR, RFOLLOW can contain one spatial relation or two, and in the latter case one of them must be ANY.

Further, the elimination of the spatial relation part from each spatial token in steps 3. and 4. does not modify the structure of the graph G_{SG} even though it changes the names of some edge labels from REL_*a* into *a*. This modification, however, does not lead to any merge of edges because if a state had outcoming edges labeled REL_1_*a* and REL_2_*a* , the spatial relations REL_1 and REL_2 would produce a positional conflict for the original positional grammar.

Example 4.2 Let us consider the extended pLALR grammar of Example 3.2. Step 1. of the Algorithm 4.2 produces the string grammar of Example 4.1. The final string grammar with positional actions produced by the Algorithm 4.2 is the following:

SP_S $\rightarrow$ SP_A VER_B
SP_A $\rightarrow$ *a* {VER()}
VER_A $\rightarrow$ *a* {HOR()}
VER_B $\rightarrow$ VER_A *c* {ANY()}
VER_B $\rightarrow$ *a* {HOR()} *d* {ANY()}

Figure 3 shows the parse tree for the picture: *a*
a c
with SP pointing to the upper *a*.

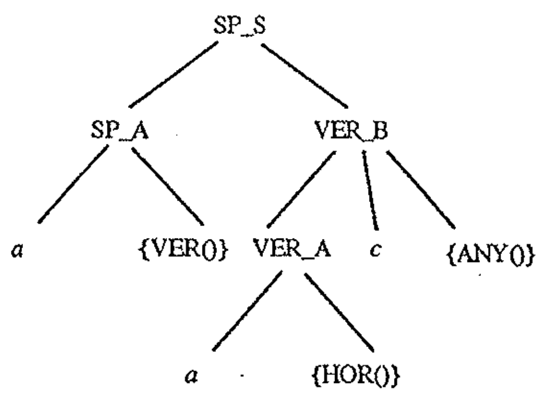


Figure 3. Parse tree with positional actions

Each positional action is attached as the appropriate child of the node corresponding to the left side of their production. Actually, the actions are treated as though they are terminal symbols. This is important to establish when the actions are to be executed.

In the parsing process, the first symbol to be analyzed is the upper symbol *a* pointed by SP, and whenever a positional action is executed the position of the next input symbol to be parsed is calculated, according to the specification of the implemented spatial relation.

As a matter of fact, the parser navigates through the input setting a linear order on the tokens. In this way, while parsing, the input is converted in a 2D "string-like" sequence of tokens.

It can be noted that the grammars produced by the Algorithm 4.2 are in Yacc format. This means that a parser for the extended pLALR two-dimensional languages can be generated automatically with no more efforts.

The only other investigated class of positional grammars for which the tool Yacc can automatically generate a parser is the class of pSLR grammars presented in [6, 7]. The main difference between the pSLR and the extended pLALR classes is that no lookahead is considered in the construction of the pSLR parser. This is analogous to the difference between the traditional SLR and LALR grammar classes.

Moreover, in the pSLR Yacc implementation, the positional actions are inserted directly into the productions of the original positional grammar without the application of Algorithm 4.1.

As an example, it can be seen that the extended pLALR grammar of Example 3.2 is not pSLR, in fact the pSLR technique produces a string grammar with conflicting positional actions:

S $\rightarrow$ A B
A $\rightarrow$ *a* {HOR(), VER()}
B $\rightarrow$ A *c* {ANY()}
B $\rightarrow$ *a* {HOR()} *d* {ANY()}

Note that the second production contains two spatial operators after the token *a* . An immediate consequence of this is that Yacc cannot be used without giving disambiguating rules.

5. Conclusions and further research

In this paper, we have characterized the class of extended pLALR grammars. Algorithms for the automatic generation of the parser have been given. Further, this class has been shown to be larger than the only one other class of positional grammars for which a Yacc implementation has been given.

On the basis of the equivalence results shown, we argue that the extended pLALR grammars are the largest class of positional grammars for which it is possible to automate the parser generation by existing tools.