(Pictorial) LR Parsing from an Arbitrary Starting Point

Gennaro Costagliola

Dipartimento di Informatica ed Applicazioni,University of Salerno
I-84081 Baronissi, Salerno, Italy
email: gencos@udsab.dia.unisa.it

Abstract

In pictorial LR parsing it is always difficult to establish from which point of a picture the parsing process has to start. This paper introduces an algorithm that allows any element of the input to be considered as the starting one and, at the same time, assures that the parsing process is not compromised. The algorithm is first described on string grammars seen as a subclass of pictorial grammars and then adapted to the two-dimensional case. The extensions to generalized LR parsing and pictorial generalized LR parsing are immediate.

1 Introduction

This paper introduces an LR algorithm for the parsing of input whose starting point is not defined. The main motivation behind this comes from the area of the two-dimensional LR parsing. Given a two-dimensional pattern it is not always obvious how to determine the starting point from which the parsing process should begin. The proposed algorithm avoids this problem allowing any element of the pattern to be considered as the starting one and, at the same time, it assures that the parsing process is not compromised.

The main idea is to create two substring LR parsers, one for the given language and the other for the "reverse" version of the language. The two parsers proceed in parallel, scanning the input in opposite directions, from the designated starting element. Neither of the two is allowed to reduce beyond their parser stack. When this is required, a rendezvous with the other parser must occur and both must perform the same reduction. The two parser stacks can be considered as an only graph stack expanding to the right and to the left of a starting point.

This paper describes the algorithm on string grammars and then shows an extension to the case of positional (two-dimensional) grammars. The algorithm can be easily extended to treat languages whose LR parsing tables present conflicts. In fact, the use of the graph stack is the same as the one adopted in Tomita's generalized parser.

Section 2 contains comments on the work related to this paper; in Section 3, the preliminary definitions of *reverse grammar* and of *joint graph* are given; Section 4 presents the data structures and the description of the algorithm; in Section 5 the algorithm is adapted to the two-dimensional LR parsing after the description of the positional grammars and positional parsing tables. Section 5 contains the Conclusions.

2 Related work

This section contains two parts: one relates to the pictorial parsing that is the main motivation behind this paper and the other relates to island-driven parsing, since the algorithm presented in this paper can be considered as a bidirectional LR parser.

2.1 Pictorial parsing

With the introduction of more and more powerful graphical interfaces, the interest in the study of pictorial parsing is increasing. At the moment, many parsers have been designed, each of them having advantages and disadvantages with rispect to one another.

A recently proposed classification, (Wittenburg, 1992), considers two major classes: bottom-up order-free pictorial parsers (Crimi *et al.*, 1991; Golin, 1991; Helm, 1991; Wittenburg, 1991) and predictive pictorial parsers (Costagliola — Chang, 1991; Wittenburg, 1991).

The main advantage of an order-free parser over a predictive one is that it can compose the input objects in any order and it is not bound to a mandatory pre-ordered navigation of the input. This gives great expressive power to the underlying grammar formalisms.

The input data structure for an order-free parser is made by two sets: a set of objects and the set of all the relations among them. The relations must be the same used in the parser. The parser then proceeds with a purely bottom-up enumeration.

The predictive pictorial parsers direct the order in which the objects in the input space are processed by the parser. This limits the expressive power of the underlying grammar formalisms but still retain expressive power enough to describe many interesting 2D languages like arithmetic expressions, lines, document layouts, some class of graphs, etc.

The input data structure only contains the set of the objects with their attributes and does not need to keep information about the relations among the objects. The relations are embedded in the parser that use them to predict the attribute values of the next object to parse. This representation is more space efficient than the other and does not depend on the particular parser relations. Moreover, it refers to a relation only when necessary.

The predictive nature of the parser makes it more efficient than an order-free bottom-up pictorial parser. In particular, for pictorial LR parsing it is even possible to use tools from string-based formal language theory like Yacc for the automatic parser generation of a pictorial language (Costagliola *et al.*, 1993).

However the prediction of the next object induces an order on the visit of the input. The order can be linear (Costagliola — Chang 1991) or partial (Wittenburg, 1992) and, in any case, it forces the parser to begin its processing from one (linear case) or multiple (partial case) specific starting points. If the input is made of a set of objects with no indication on the starting point, like a scanned document layout, then predictive parsing becomes inefficient. This paper attempts to solve this problem by constructing a bidirectional LR parser that does not need specific starting points in the input.

2.2 Island-driven parsing

Island-driven parsers are generally used for generating partial interpretations of a spoken sentence (Stock *et al.*, 1989; Woods, 1982). The parsing starts from words that have higher acoustic evidence and then extends to both directions in the sentence. Each partial interpretation forms an "island". Occasionally, two islands may 'collide' by proposing and discovering the same word in the gap among them and may then be combined into a single larger island.

This can be effectively used in pictorial parsing whenever there are objects of particular semantic relevance, or objects particularly complex to be combined only when each of them has been recognized, or, in our case, the starting point is not easy to find.

Other approaches to bidirectional parsing include bidirectional chart parsing (Stock *et al.*, 1989; Steel — De Roeck, 1987) and some form of bidirectionality within a tabular approach, such as Earley's or Kasami-Cocke-Younger's (Bossi *et al.*, 1983).

3 Some definitions

This section contains two definitions that will be useful for the presentation of the final algorithm.

Definition 1 (reverse grammar) *Given a context-free grammar $G = (N, T, S, P)$, a reverse grammar with respect to G is a new grammar $G' = (N, T, S, P')$, where P' is defined as follows: whenever $A := u$ is in P then $A := u^R$ is in P', where u^R is the reverse version of u.*

In general, the reverse of an LR context-free grammar is not LR.

For sake of simplicity, this paper considers only LR context-free grammars whose reverse is LR, too. The extension to general context free-grammars can be easily done.

Example 3.1.

The grammar G:
(1) S := CC
(2) C := cC
(3) C := d

The reverse grammar G′:
(1) S := CC
(2) C := Cc
(3) C := d

It is assumed that corresponding productions have the same ordering number. Note that if the C's in the production S := CC are indexed, then the grammar G contains the production S := C_1C_2 and the grammar G′ contains S := C_2C_1. To formalize this concept, let us index all the occurrences of symbols on the right side of the productions of G such that an into correspondence between occurrences of symbols and indices is created. After indexing, the grammar G becomes:

(1) S := C_1C_2
(2) C := c_3C_4
(3) C := d_5

C_1, an occurrence of C, is now different from the occurrence C_2 but

$$\text{name}(C_1) = \text{name}(C_2) = C.$$

Given an LR grammar G such that its reverse grammar G′ is LR, it is always possible to construct for each of them the canonical LR(0) collection of sets of items through the algorithms *Closure*, *Goto* and *Set-of-Items Construction* as defined in (Aho *et al.*,1985). The goto graphs for G and G′ are shown in Figure 3.1.

Let us define $C_I = I_0, \ldots, I_n$ the collection of sets of items for G and $C_R = R_0, \ldots, R_m$ the collection of sets of items for G′. In the following, the relation between elements of C_I and C_R is analyzed. It is assumed that no useless symbols or epsilon-productions are in G.

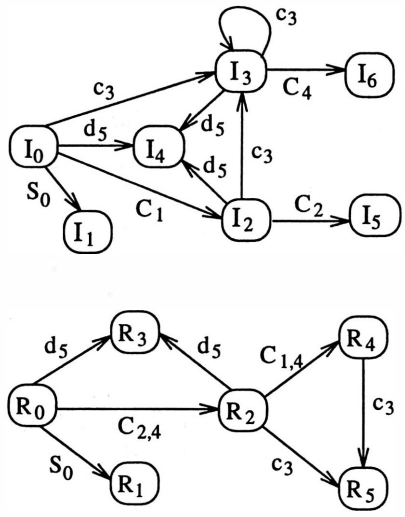


Figure 3.1 Goto Graphs for G and G′

Given a production "A := u X_i v"' in G with u, v ∈ (N ∪ T)* and X_i ∈ (N ∪ T), there must be a set-of-items I_k reachable after the occurrence X_i has been processed, i.e., a set-of-items I_k containing the item "A := u X_i . v".

If the corresponding production in G′ "A := v^R X_i u^R" is considered, there must exists a set-of-items R_l reachable after the occurrence X_i has been processed, i.e., a set-of-items R_l containing the item "A := v^R X_i . u^R". Here v^R and u^R are again the reverse versions of v and u, respectively

In other words, if I_k is the state reachable after a forward scanning of X_i in the context of u X_i v, then there must exists R_l, the state reachable after a backward scanning of X_i still in the context of u X_i v.

As an example, the goto graphs above show that I_3 and R_5 are both reachable through c_3.

Definition 2 (joint graph) *Let us consider a grammar G = (N, T, S, P) with indexing, its reverse grammar G′ = (N, T, S, P′) and their canonical LR(0) sets-of-items collections* $C_I = I_0, \ldots, I_n$ *and* $C_R = R_0, \ldots, R_m$, *respectively.*

The joint graph *for an* occurrence X_i *of* X *in* $N \cup T$ *is given by:*

Jgraph(X_i) = { (I_k, R_l) / *there exist* $A \in N$, *and u, v occurrences of strings* $\in (N \cup T)^*$ *such that "*$A := u\ X_i\ .\ v$*"* $\in I_k$ *AND "*$A := v^R\ X_i\ .\ u^R$*"* $\in R_l$ }

The joint graph *for the* symbol X *in* $N \cup T$ *is:*

$$Jgraph(X) = \bigcup_{i:name(X_i)=X} Jgraph(X_i)$$

Example 3.2.

Considering the grammars in Example 3.1 and looking at the goto graphs above:

Jgraph(c) = Jgraph(c_3) = {(I_3, R_5) }
Jgraph(d) = Jgraph(d_5) = {(I_4, R_3) }
Jgraph(S) = Jgraph(S_0) = {(I_1, R_1) }
Jgraph(C) = Jgraph(C_1) ∪ Jgraph(C_2) ∪
∪ Jgraph(C_4) = {(I_2, R_4) } ∪ {(I_5, R_2) } ∪
∪ {(I_6, R_2), (I_6, R_4) } =
= {(I_2, R_4), (I_5, R_2), (I_6, R_2), (I_6, R_4) }

To parse the string "d′cd" starting from the symbol "c", a substring parser based on G will parse "cd" and a substring parser based on G′ will parse "cd′".

As Jgraph(c) = {(I_3, R_5) }, the first parser will start from state I_3, while the second one will start from R_5. To parse the whole string the completion of the substring parser based on G will have to match the parsed part of the reverse substring parser, and vice versa.

After reducing the non-terminal C, the states of G to be considered are I_2, I_5 and I_6 while the states of G′ are R_2 and R_4. Note that if the parsing process starting in R_4 fails, then that starting in I_2 must fail, too, because it has no corresponding state R_i left in Jgraph(C).

4 The Algorithm

The algorithm is based on the concepts of substring parsing as presented in (Rekers — Koorn, 1991). In this paper, an algorithm for substring parsing for arbitrary context-free grammars is presented. It is based on the pseudo-parallel parsing algorithm of Tomita (Tomita, 1985, 1991), which runs a dynamically varying number of LR parsers in parallel and accepts general context-free grammars.

Even though the algorithm can be easily extended to the general case, in this paper it will be limited to accept only LR context-free grammars whose reverse is still an LR grammar. Informally the algorithm can be described as follow.

The input is given by a grammar G and its reverse grammar G′, the Jgraph for each symbol of the grammars, the two parsing tables and an input sentence $a_0 \ldots a_n$ with an index $0 \leq i \leq n$ from where the parsing process is supposed to start. Jgraph(a_i) provides the initial states. As seen before, a Jgraph contains states (set-of-items) from both the grammar G and its reverse.

In the following, a *forward parser* is an LR parser for G and a *backward parser* is an LR parser for G′. Moreover, the *opposite* parser of a forward parser is meant to be a backward parser and vice versa.

Every state I_k in Jgraph(a_i) (with i being the starting position) becomes the initial state for a forward parser and every state R_l in Jgraph(a_i) an initial state for a backward parser. The forward parsers interact only with forward parsers in the same way as a generalized LR parser. The same is true for the backward parsers. The exception occurs when a parser tries to reduce a production "A := u" requiring the stack to pop its initial state. That parser is then blocked waiting for an opposite parser to try the corresponding reduction "A := u^R" on the same symbols.

If the distance between the two parser stack tops is | u | and the initial states of the two parsers form an edge in Jgraph(a_i), then a *rendezvous* occurs and Jgraph(A) is generated. Jgraph(A) will produce a new set of forward and backward parsers and the process will continue till when either two opposite parsers have a rendezvous on the action "accept" or no rendezvous is possible and the input has all been consumed.

4.1 Data Structures

The algorithm is based on a graph-structured stack with two types of nodes: *joint* stack nodes and *simple* stack nodes and it is able to construct a packed shared parse forest, (Tomita, 1985, 1991).

A *joint stack node* is a 5-tuple (Jgraph, X, blast, flast, sf_ptr) where Jgraph is as defined above; X is either a terminal or a non-terminal; blast and flast point to the last elements visited during the backward and forward parsing of the input, respectively; sf_ptr is a pointer to a node labeled X in the packed shared parse forest.

A *simple stack node* is a 4-tuple (state, X, last, sf_ptr) where state is the state reached by the parser and corresponds to a set of items; last is the last terminal parsed; X and sf_ptr are as above.

Note that a joint node ({..., (I, R), ...}, X, blast,flast, sf_ptr) represents a graph whose elements are simple stack nodes of the type (I, X, flast, sf_ptr) and (R, X, blast, sf_ptr) and the edges are defined in the Jgraph component.

The operations on the graph stack are the Splitting, Combining and Local Ambiguity Packing operations, (Tomita, 1985, 1991), as used in the definition of the Generalized LR Parser. The only difference regards the updating of the node fields flast and blast. The following are the definitions of two new operations that must be added to the previous.

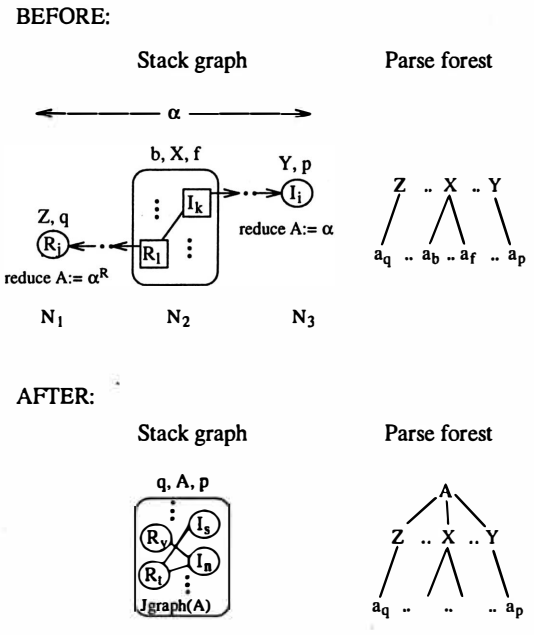


Figure 4.1 The Rendezvous Operation

The Rendezvous Operation

A graphical description of the rendezvous operation is given in Figure 4.1 (pointers from the stack graph to the parse forest are not shown).

If there is a joint node:
N_2 = ({ ... (I_k, R_l) ...}, X, b, f, X_ptr)
and two simple nodes:
N_1 = (R_j, Z, q, Z_ptr) and
N_3 = (I_i, Y, p, Y_ptr) such that

- N_1 is the active stack top of a backward parser with initial state R_l in N_2
- N_3 is the active stack top of a forward parser with initial state I_k in N_2
- the edge (I_k, R_l) is in the Jgraph of the node N_2.
- action(I_i, a_{p+1}) = "reduce A:= α" where α = Z ... X ... Y
- action(R_j, a_{q-1}) = "reduce A:= α^R"
- path_length(N_2 ... N_1) + path_length(N_2 ... N_3) - 1 = | α |

then N_1 and N_3 are made non-active and a new active joint stack node (Jgraph(A), A, q, p, A_ptr) is created, where A is the left-hand of the reduced production, q is the pointer to the last visited token in N_1, p is the pointer to the last visited token in N_3 and A_ptr is the pointer to a new shared forest vertex whose children are vertices pointed by stack nodes contained in the path N_1 ... N_2 ...N_3.

Note that the path N_2 ... N_1 represents the stack nodes of the backward parser while N_2 ... N_3 are the stack nodes for the forward parser.

The Accept Operation

If there is a joint node: N = ({ ... (I_k, R_l) ... S, 0, n, S_ptr)
where S is the starting non-terminal, 0 and n are the positions of the first and last elements of the input, and

- action(I_k, $) = accept
- action(R_l, $) = accept

then accept the input and return the pointer to the parse forest, S_ptr.

4.2 The LR parser with an arbitrary starting point

Input: An LR grammar G = (N, T, S, P) and its reverse G', the Jgraph for every symbol in N ∪ T, a sentence w = $a_0 a_1 \ldots a_n$ and a starting position i, with i ∈ {0, .., n}.
Output: The parse forest for w if accepted.
Method:

1. Create the LR parsing tables for G and G'

2. Create the joint node (Jgraph(a_i), a_i, i, i, a_i_ptr) and make it active.

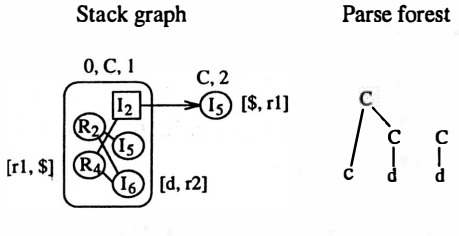
3. For every element I in the Jgraph field of an active joint node N = (Jgraph(X), X, b, f, X_ptr) start a forward generalized LR parser with initial node (I, X, f, X_ptr) on input $a_{f+1} \dots a_n$; for every element R in the Jgraph field of N start a backward generalized LR parser with initial node (R, X, b, X_ptr) on input $a_{b-1} \dots a_0$. Whenever a parser has a reduce action involving its initial node, or has an accept action to perform, make it wait. All the others will keep processing the input.

4. When all the parsers, both forward and backward, are in the wait state, apply the rendezvous operation wherever possible and go to step 3. If an accept operation is possible then return the corresponding pointer to the parse forest. If no rendezvous or accept operations are possible then the parsing process halts.

If no parse forest pointer has been returned then the sentence has not been accepted.

Example 4.1

Given the grammars G and G′ of Example 3.1, the input string "cdd" with positions 0, 1 and 2 respectively and the starting index 1, after two rendezvous operations the following situation is presented:

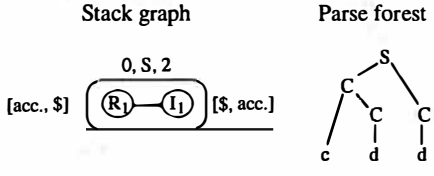


with the joint node ({(I_2, R_4), (I_5, R_2), (I_6, R_2), (I_6, R_4)}, C, 0, 1, C_ptr), the simple node (I_5, C, 2, C_ptr) and the corresponding [lookahead, action] pairs. The two nodes point to the first and the second tree in the parse forest, respectively.

On lookahead symbol \$, the state I_5 in the simple node of the forward parser built on G, requires a reduce action with production "(1) S := CC" while I_6, on lookahead d in position 2, requires a reduction with "(2) C:= cC". On the lookahead symbol \$, the state R_4 of the backward parser built on G′, requires the reduction "(1) S := CC". R_2 and I_5 have no action on \$ and 'd', respectively.

At this point, no action is possible without involving the joint node. Note that the nodes with states I_5 and R_4 meet the rendezvous operation requirements: both of them are active stack tops requiring the same reduce action, the sum of the depths of the stack of the two parsers - 1 = | CC | = 2 and (I_2, R_4) is an edge in the Jgraph of the joint node.

By applying the rendezvous operation on production "S := CC" and recalling that Jgraph(S) = {(I_1, R_1)} the following configuration is reached:



with the joint node ({(I_1, R_1)}, S, 0, 2, S_ptr) and I_1 and R_1 requiring both an accept action. The execution of the accept operation will then return the pointer to the final parse.

5 Two-dimensional LR parsing from an arbitrary starting point

An LR parser takes in input a sequence of tokens and returns a parse tree if the sequence is in the language accepted by the parser. The sequence of tokens are usually extracted from the string data structure.

A first generalization of this model toward 2-D parsing regards the possibility to have other input data structures different from the string. After all a string can be seen as a set of elements each having an attribute whose value is given by the position of the element in the string. As an example, the string "a b c" can be seen as the set {(b 2), (a 1), (c 3)} where each pair represents the element and its attribute value.

With this new data structure, the LR parser cannot simply require a "next" token to the lexical analyzer but has also to give indications on the position of the token. When this is done, the

input sequence of tokens to an LR parser can be extracted from any set of tokens with attributes. In the case of two-dimensional symbolic languages these attributes will correspond to Cartesian coordinates but other types of attributes can be thought of. In the case of diagrammatic languages, for example, size, shape, colour, etc. can be considered as attributes.

But how is it possible to make an LR parser give indications on the attribute values of the next token to parse? This can only be done by inserting appropriate information in the productions of the grammar from which the LR parser is built.

In the case of 2-D symbolic parsing this information is given by spatial operators that take in input the position of the last visited symbol and return the position of the next symbols to parse. Examples of spatial operators are:

"String Concatenation" : i ⇒ i + 1
"Up": (i, j) ⇒ (i, j+1)
"Left": (i, j) ⇒ (i-1, j)
"Right": (i, j) ⇒ (i+1, j)

5.1 Positional grammars

While in the traditional case there is an implicit use of the only string concatenation spatial relation, in the 2-D case many other spatial relations can be used and must made explicit in the grammar formalism.

In the following, some definitions are re-called, (Costagliola — Chang, 1991), to define a 2-D grammar formalism and the languages generated by it:

Definition 3 (positional grammar) *A context-free* positional grammar *PG is 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
$N \cap T = \phi$
$S \in N$ *is the starting symbol*
P is a finite set of productions
POS is a finite set of spatial relation identifiers
PE is a positional evaluator

Each production in P has the following form:

$$A := X_1\ Rel_1\ X_2\ Rel_2\ \ldots\ X_{m-1}\ Rel_m$$

where $m \geq 1$, $A \in N$, *each* X_i *is in* $N \cup T$ *and each* Rel_i *is in POS.*

In the following, the words "positional grammar" will also refer to a context-free positional grammar.

Definition 4 (pictorial language) *Let PG = (N, T, S, P, POS, PE). A* positional sentential form *is a string* II *such that* $S \Rightarrow^*$ II, *where* $\Rightarrow^*$ *has the conventional meaning. A* positional sentence *is a positional sentential form not containing non-terminal symbols. A* picture *is the evaluation of a positional sentence. The* pictorial language defined by a positional grammar L(PG) *is the set of its pictures.*

Note that if POS contains the only "string concatenation" spatial relation the positional grammar formalism reduces to the traditional context-free grammar formalism.

Example 5.1

The following positional grammar generates a simple subset of the arithmetic expressions:

N = {E, S, T, F}
T = {+, $\sum$, (,), id, num}
E is the starting symbol
POS = { >, _ }
P = { E := E > + > T | T
T := S > T | F
S := $\sum$ _ id
F := id _ id | num | (> E >) }

where the characters '>' and '_' stand for "horizontal concatenation" and "under concatenation", respectively. A positional sentence is:

"5 > + > $\sum$ _ i > (> x _ i > + > y _ i >)"

From its evaluation the particular positional evaluator PE for this grammar produces the following picture: $5 + \sum_i (x_i + y_i)$.

A more detailed definition of PE for this type of grammars can be found in (Costagliola *et al.*, 1992).

Example 5.2.

The following positional grammar PG will be used in the following to illustrate the execution of the algorithm.

(1) S := A Down S
(2) S := A
(3) A:= a Right a

A positional sentential form for this grammar is "A Down a Right a"; a positional sentence is "a Right a Down a Right a"; the corresponding picture is given by the evaluation of the spatial relations in the positional sentence, from left to right:

a a
a a

Very similarly to Definition 1, the corresponding reverse positional grammar PG′ is:

(1) S := S Up A
(2) S := A
(3) A:= a Left a

The positional sentence becomes now "a Left a Up a Left a" and produces the same picture as abovestarting from the 'a' in the lower right.

Note that to reverse a positional grammar, it is not enough to reverse the right side of the productions. Every spatial relation must also be substituted with a semantically opposite spatial relation. In this example, "Down" becomes "Up" and "Right" becomes "Left".

5.2 Pictorial LR parsing

The generalization of LR parsing to the two-dimensional case has already been treated in (Costagliola *et al.*, 1991, 1992, 1993), where classes of pSLR, pLALR and pictorial generalized LR languages have been characterized.

The parser generation methodology from positional grammars is very similar to the traditional LR technique. The only difference regards the handling of the spatial relations.

As an example, the item "A:= a .Left a" means now that a new 'a' is expected to the left of the just seen 'a'. The containing set-of-items will then be associated to the spatial relation Left.

To handle the positional information, the final LR parsing table for a positional grammar has a new column named "pos", besides the traditional "action" and "goto" parts. The underlying automaton will then have a spatial function associated to each state in order to predict the position of the next symbol to parse.

Example 5.3

The parsing tables for PG and PG′ of Example 5.2 are shown in Figure 5.1 and 5.2, respectively.

state	action		goto		pos
	a	$	A	S	
I_0	s4		2	1	
I_1		acc.			Any
I_2	s4	r2	2	3	Down
I_3		r1			Any
I_4	s5				Right
I_5	r3	r3			Down

Figure 5.1

state	action		goto		pos
	a	$	A	S	
R_0	s4		5	1	
R_1	s4	acc.	2		Up
R_2	r1	r1			Up
R_3	r3	r3			Up
R_4	s3				Left
R_5	r2	r2			Up

Figure 5.2

Every spatial relation name in the column "pos" indicates a spatial function that takes in input a position and returns a terminal, if found, or the end-of-input marker $, otherwise. The only exception is 'Any' that always returns $.

The action "accept" is actually a conditional "accept": if all the symbols of the picture have been processed then accept, otherwise reject. This can be done by marking each visited symbol and looking for unmarked ones.

Looking at Figure 5.1, if state I_5 is reached by shifting a terminal 'a' whose position is (i, j), then the next symbol to process is the terminal Down(i, j) in position (i, j-1). Note that Down is the spatial function associated to I_5. In the following, for sake of simplicity, each Cartesian coordinate (i, j) will be associated with a unique index k.

5.3 The 2-D extension

Definition 2 of "joint graph" applies with no modification to the case of pictorial LR parsing. In particular, for the grammars PG and PG′ of Example 5.2, the Jgraph sets are so defined:

$Jgraph(S) = Jgraph(S_0) \cup Jgraph(S_2) =$
$=\{(I_1, R_1), (I_3, R_1)\}$
$Jgraph(A) = Jgraph(A_1) \cup Jgraph(A_3) =$
$= \{(I_2, R_2), (I_2, R_5)\}$
$Jgraph(a) = Jgraph(a_4) \cup Jgraph(a_5) =$
$= \{(I_4, R_3), (I_5, R_4)\}$

The LR parser with an arbitrary starting point algorithm can also be easily adapted to the case of pictorial languages. The only difference is that the forward and backward parsers are not generalized LR parsers as defined in (Tomita, 1991) but pictorial generalized LR parsers, (Costagliola *et al.*, 1992).

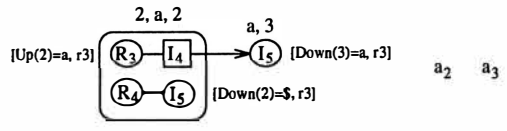
Example 5.4.

Let us consider the grammars PG and PG′ of Example 4.2, the input picture

$a_0\ a_1$
$a_2\ a_3$
$a_4\ a_5$

where each terminal 'a' has been indexed with its position, and i = 2 as the starting position; from the initial stack graph it is possible to reach the following configuration:

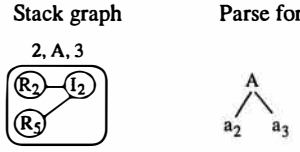
Stack graph Parse forest



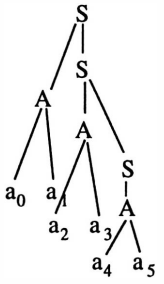
with the joint node ({(I_4, R_3), (I_5, R_4)}, a, 2, 2, a_2_ptr), the simple node (I_5, a, 3, a_3_ptr) and the corresponding [lookahead, action] pairs. The simple stack node with state I_5 has just seen the terminal 'a' in position 3; to take the next action the associated spatial function Down must be applied to the position 3. The returned terminal is 'a' in position 4 and only now the parser can decide a reduction with "(3) A := a Right a". The same explanation can be given for the actions taken by the parsers in the remaining states I_5 and R_3.

The parser with initial state R_4 has no action to take and fails, making the forward parser starting in I_5 fail, too. The only rendezvous operation can then be performed between the parsers starting in R_3 and I_5 on production "A:= a Right a":

Stack graph Parse forest



where the stack graph reduces to the joint node ({(I_2, R_2), (I_2, R_5)}, A, 2, 3, A_ptr). In two other rendezvous operations the picture will be eventually accepted with parse tree:



6 Conclusions

This paper has presented an algorithm to allow LR parsing from an arbitrary starting point of the input. The algorithm is based on Tomita's algorithm and refers to substring parsing as defined in (Rekers — Koorn, 1991). It makes use of two LR parsing tables, one for the original grammar and another for its reverse version.

It can be shown that there is a moderate overhead with respect to the normal parser and that sentences starting with a token that can appear in many different context take more time to parse than sentences starting with a disambiguating token. With respect to substring parsing, no overhead is necessary for the completion of the sentence by any backward or forward parser, as the completion is always determined by the rendezvous operation. Further the Jgraph data structure allows only appropriate reductions cutting on the overhead due to unfeasible reductions.

It is also been showed that, based on the pictorial generalized LR parser in (Costagliola *et al.*,

1992), the extension of the algorithm to the two-dimensional case is immediate.

In this paper, only the simplest form of two-dimensional parsing, the so-called linear pictorial parsing, is referred to. In this type of pictorial parsing, the spatial relations are defined such that the position of the next symbol only depends on the last symbol processed.

More complex forms include the possibility to calculate the next symbol based on the positions of the elements of the last handle or of the whole input so far visited. These forms have been investigated in traditional pictorial generalized parsing and are currently being investigated in the context of LR parsing with an arbitrary starting point.

References

Aho, A.V. — R. Sethi — J.D. Ullman (1985) *Compilers, principles, techniques and tools*. Addison Wesley.

Bossi, A. — N. Cocco — L. Colussi (1983) "A Divide-and-conquer Approach to General Context-free Parsing". in: *Information Processing Letters* 16, 203 – 208.

Costagliola, G. — S.-K. Chang (1991) "Parsing 2D Languages with Positional Grammars". in: *Proceedings of Second Int. Workshop on Parsing Technologies* 235 – 243. Cancun, Mexico, February 13 – 25, 1991.

Costagliola, G. — S.-K. Chang — M. Tomita (1992) "Parsing 2D Languages by a Pictorial GLR parser". in: Catarci, T. & M.F. Costabile & S. Levialdi, (Eds): *Advanced Visual Interfaces*. 319 – 333. Singapore: World Scientific Publishing.

Costagliola, G. — S. Orefice — G. Polese — M. Tucci — G. Tortora (1993) "Automatic Parser Generation for Pictorial Languages". in: *Proceedings of IEEE Symposium on Visual Languages* Bergen, Norway, August 24 – 27, 1993, to be published.

Crimi, C. — A. Guercio — G. Nota — G. Pacini — G. Tortora — M. Tucci (1991) "Relational Grammars and their Application to Multidimensional Languages". in: *Journal of Visual Languages and Computing* 2, 333 – 346. Londra: Academic Press.

Golin, E. J. (1991) "Parsing Visual Languages with Picture Layout Grammars". in: *Journal of Visual Languages and Computing* 2, 371 – 393. Londra: Academic Press.

Helm, R. — K. Marriot — M. Odersky (1991) "Building Visual Language Parsers". in: Robertson, S.P. & G.M. Olson & G.S. Olson, (Eds): *Human Factors in Computing Systems: CHI '91 Conference Proceedings* 105 – 112. Amsterdam: Addison-Wesley.

Rekers, J. — W. Koorn (1991) "Substring Parsing for Arbitrary Context-Free Grammars". in: *Proceedings of Second Int. Workshop on Parsing Technologies* 218 – 224. Cancun, Mexico, February 13 – 15, 1991.

Steel, S. — A. De Roeck (1987) "Bidirectional Chart Parsing". in: *Proceedings of AISB-87* Edinburgh: Scotland.

Stock, O. — R. Falcone — P. Insinnamo (1989) "Bidirectional Charts: a Potential Technique for Parsing Spoken Natural Language" in: *Computer Speech and Language* 3, 1989.

Tomita, M. (1985) *Efficient Parsing for Natural Languages* Boston MA: Kluwer Academic Publishers.

Tomita, M. (1991) *Generalized LR Parsing*. Norwell MA: Kluwer Academic Publishers.

Wittenburg, K. (1992) "Earley-style Parsing for Relational Grammars". in: *Proceedings of IEEE Workshop on Visual Languages* Seattle, USA, September 15 – 18, 1992.

Wittenburg, K. — L. Weitzman — J. Talley (1991) "Unification-based Grammars and Tabular Parsing for Graphical Languages". in: *Journal of Visual Languages and Computing* 2, 347 – 370. Londra: Academic Press.

Woods, W. A. (1982) "Optimal search strategies for speech understanding control". in: *Artificial Intelligence* 18, 295 – 326.