

On the pLR Parsability of Visual Languages

GENNARO COSTAGLIOLA, VINCENZO DEUFEMIA, FILOMENA FERRUCCI, AND CARMINE GRAVINO

Dipartimento di Matematica e Informatica
 Università di Salerno - 84081 Baronissi (SA) – Italy
 {gcostagliola, deufemia, fferrucci, gravino}@unisa.it

Abstract

The widespread use of visual languages has motivated the need for grammar-based tools to support designers in the definition and implementation of graphical environments. The effective use of such systems requires efficient parsing techniques. The VLCC system makes use of a suitable LR-like methodology that allows us to efficiently parse visual sentences [1]. However there exist particular grammars which are non pLR parsable because they can produce run-time conflicts during the parsing of some sentences. In this paper we will introduce an algorithm that statically verifies the pLR parsability of a positional grammar by detecting whether or not it would produce run-time conflicts.

1. Introduction

The formalism of *Positional Grammars* provides the syntactic framework for the *Visual Language Compiler-Compiler (VLCC)*, a graphical system which is able to assist a designer in the definition of a visual language and automatically generate a corresponding visual environment [2]. The most appealing feature of VLCC is that it inherits and extends to the visual field concepts and techniques of traditional compiler generation tools like YACC [1]. This is due to the characteristics of the positional grammar model which represents a natural extension of the context-free grammars. This has allowed us to extend the LR methodology [3] in order to obtain an efficient LR-like parser for positional grammars (positional LR parser or pLR parser for short).

The languages generated by positional grammars are defined as sets of pictures obtained by spatially arranging graphical objects over a given vocabulary. Each graphical object is characterized by an image and a set of syntactic attributes, whose values are used to relate objects to one another. In its general definition a positional grammar $PG=(G, PE)$ is composed of an attributed string grammar G and a positional evaluator PE that translates positional sentences generated by G into visual sentences [2]. More precisely, the grammar $G=(T \cup POS, N, S, P)$ is composed of a set of terminals partitioned into a set of symbols T and a set of relation identifiers POS , a set of non-terminals N , a starting symbol S and a set of productions P of the form:

$$A \rightarrow x_1 R_1 x_2 R_2 \dots R_{m-1} x_m, \Delta \quad \text{for } m \geq 1$$

where $A \in N$, each $x_i \in N \cup T$, and $R_i \in (POS \times I)^n$ with $1 \leq i \leq m$, $n \geq 1$, and I is the set of positive integers. Both terminals and non-terminals are graphical objects. The rule Δ synthesizes the syntactic attributes values of A from

those of x_1, x_2, \dots, x_m . In the sequel we denote the syntactic attribute h of an object x_i with $h x_i$.

Let x be a grammar object: a *positional sentential form* from x is a string β such that β is derived from α ; a *positional sentence* from x is a positional sentential form from x which does not contain non-terminals.

The pLR methodology consists of algorithms to encode a positional grammar into a *pLR parsing table*. Figure 1 shows the general schema of a positional LR parser. The input to the parser is a set of graphical objects arranged on a plane. As a difference with traditional LR parsing the input scanning is no longer sequential but driven by the relations specified in the positional grammar. The technique used is called *syntax-directed scanning* of the input and has been first defined in [4] for two-dimensional symbolic languages.

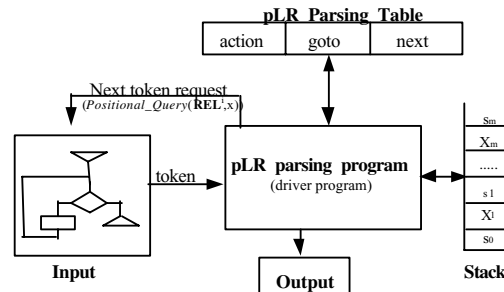


Figure 1. The pLR parsing model

The syntax-directed scanning is implemented by adding a new column, named *next*, to the LR parsing table (see Figure 1). The column *next* associates a pair (REL^i, x) to each parser state I_k , where REL is a relation and x is a grammar object. Whenever the parser reaches state I_k , the pair (REL^i, x) is used as an argument of the function *Positional_Query* to select the next object to be processed. The pLR parsing table is constructed from the sets of pLR(0) item collections. A pLR(0) item of a positional grammar $PG=(G, PE)$ is a production from G with a dot at some position in the right-hand side. Intuitively, an item indicates how much of a production has already been examined during the parsing process and what is yet to come.

In general the pLR methodology works effectively and has allowed us to efficiently process a wide class of positional grammars. However, there are particular pLR grammars for which the application of the methodology turns out to be inadequate. Indeed, for such grammars it may happen that the launch of *Positional_Query* on an

input sentence returns more than one graphical object. In this case the parser cannot proceed because it cannot decide deterministically which token to analyze. This conflict has been first studied in [1] and named *run-time conflict*. As an example, let us consider the following positional grammar

G: $N=\{A, B\}, T=\{a, b, c\}, POS=\{JOINT(\cdot)\},$
 $P=\{ A \rightarrow a _1 _1 B _1 _1 b, \quad B \rightarrow b _2 _1 c \Delta:(1B=1b) \}$

where $JOINT(h, k)$ is denoted as h_k and is such that $x _h _k y$ relates the syntactic attribute h of x to the syntactic attribute y of b . The apex 1 to the relation $_1 _1$ indicates that the second preceding symbol a is to be considered in the relation with b . Figure 2 shows the graphical representation of the sentence generated by G. When given as input to the correspondent pLR parser for G and the terminal symbol a has just been processed, *Positional_Query* is launched on $(_1 _1, b)$ and two occurrences of the terminal symbol b are retrieved because both are related with a through the $_1 _1$ relation. This is what we call a *run-time conflict*.

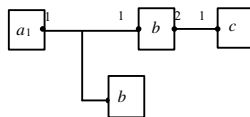


Figure 2. A graphical sentence with a run-time conflict

2. Static identification of run-time conflicts

In this section, we sketch a method for verifying the pLR parsability of a positional grammar during the construction of the corresponding parsing table. This technique allows a designer to have a feedback in the early phases of the syntax definition of the visual language and gives him/her information on how and where to intervene in order to modify the grammar.

The method operates during the construction of the set of item collections. It gathers information about all the relations referring to each grammar object in each pLR item obtaining a new notion of item named *extended item*. Then for each set of extended items I it invokes the function $RTADetect(I)$. This function consults the information about each extended item and returns *true* if there is an item that could cause a run-time conflict and *false* otherwise.

An *extended item* I is a triple $[J, DP, JS]$ where J is an item, $DP= driverpoint(J)$ and $JS=IJSET(J)$. $IJSET$ specifies all the relations involving the symbol in the item and DP indicates the syntactic attribute of the symbol that is related to the last symbol parsed and is used to drive the parsing process.

Figure 3 shows a subset of the sets of the extended items (without Δ) of a pLR(0) extended item collection for the previous positional grammar. Here, subscripts are used to distinguish different occurrences of the same symbol in different items and marked symbols indicate symbols that have already been seen.

State 0	
$I_0: S' \rightarrow \cdot A_0$	$DP=\emptyset, IJSET(I_0)=\emptyset$
$I_1: A_0 \rightarrow \cdot a_1 _1 _1 B_1 _1 _1 b_1$	$DP=\emptyset, IJSET(I_1)=\{(1a_1, 1B_1, 1b_1)\}$
State 1	
$I_2: A_0 \rightarrow a_1 \cdot _1 _1 B_1 _1 _1 b_1$	$DP=1B, IJSET(I_2)=\{(1a_1, 1B_1, 1b_1)\}$
$I_3: B_1 \rightarrow \cdot b_2 _2 _1 c_2$	$DP=1b, IJSET(I_3)=\{(1B_1=1b_2, 1a_1, 1b_1), (2b_2, 1c_2)\}$

Figure 3. A subset of the sets of the extended items of a pLR(0) extended item collection

Figure 4 shows a visualization of the element $(1B_1=1b_2, 1a_1, 1b_1)$ in $IJSET(I_3)$. The non-terminal B is shown with a dashed box.

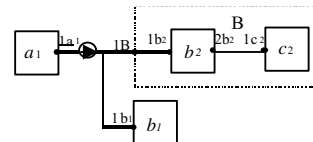


Figure 4. The visualization of an element in $IJSET(I_3)$

The circle with an arrow represents the position of the dot in state 1: it says that the symbol a has already been seen and that objects B and b_1 need to be analyzed next. When in state 1, the parser will match the input against this pattern. It will launch *Positional_Query*($_1 _1, B$) and if the input is correct both b_1 and b_2 will be returned by causing the run-time conflict trap.

$RTADetect$ is invoked on each set of extended items. For each extended item $I_i: A \rightarrow \alpha_j \cdot R_j x_j \beta_j$ in I with $driverpoint(I_i)=kx_j$ not empty, $RTADetect$ calculates the set A of all the couples (y, h) such that y is a terminal that begins a positional sentential form from x_j , and the syntactic attribute k of x_j can be synthesized from the syntactic attribute h of y . Then, it analyzes each extended item in I whose *driverpoint* is an element in A and places in Y all the syntactic attributes of the terminals that begin a positional sentential form from the *driverpoint*. If the multi-set Y contains two or more instances of an element hb in A , then two symbols b are related to a symbol in α_j through the same relation and $RTADetect(I)$ returns *true*. As an example, for the set of extended items corresponding to state 1 in Figure 3, $RTADetect$ returns true because $A=\{(1,b)\}$ and $Y=\{1b_1, 1b_2\}$.

References

- [1] G. Costagliola, A. De Lucia, S. Orefice, G. Tortora (1997) A Parsing Methodology for the Implementation of Visual Systems. *IEEE Trans. Software Engineering* **23**, 777-799.
- [2] G. Costagliola, A. De Lucia, S. Orefice, G. Tortora (1995) Automatic Generation of Visual Programming Environments. *IEEE Computer* **28**, 56-66.
- [3] A. V. Aho, R. Sethi, J. D. Ullman (1985), Compiler Principles, Techniques, and Tools, Ed. Addison-Wesley.
- [4] G. Costagliola, S. K. Chang "Parsing Linear Pictorial Languages by Syntax-Directed Scanning", *Languages of Design*, **2**, 3, 1994, Elsevier Science Publishers, pp. 229-248