Using Constraints over Finite Sets of Integers for Range Concatenation Grammar Parsing*

Yannick Parmentier[1] and Wolfgang Maier[2]

[1] CNRS-LORIA, BP 239, F-54506 Vandœuvre-Lès-Nancy Cedex, France,
parmenti@loria.fr
[2] SFB 441 - Universität Tübingen, Nauklerstr. 35, D-72074 Tübingen, Germany,
wo.maier@uni-tuebingen.de

Abstract. Range Concatenation Grammar (RCG) is a formalism with interesting formal properties (it has a polynomial parsing time while being more powerful than Linear Context-Free Rewriting Systems). In this context, we present a constraint-based extension of the state-of-the-art RCG parsing algorithm of [1], which has been used for the implementation of an open-source parsing architecture.

1 Introduction

Range Concatenation Grammar (RCG) [2] is a grammar formalism with attractive properties (*e.g.* closedness under complementation and intersection). Still, RCGs are computationally tractable: they can be parsed in polynomial time in the size of the input string and linear time in the size of the grammar [1]. Furthermore, as shown by [3], RCG can be used to encode different mildly context sensitive formalisms such as Multi-Component Tree Adjoining Grammar. RCG offers thus a uniform interface for comparing the properties of formalisms, and is a candidate pivot formalism for parser implementations.

In this context, we propose an extension of the state-of-the-art RCG parsing algorithm of [1] making use of constraint programming techniques. The extended algorithm has been implemented in an open-source parsing architecture for tree-based grammars [4]. Before presenting this extension, we first give a brief introduction to RCG.

An RCG is a set of clauses of the form : $A_0(x_{01}, \ldots, x_{0n_0}) \rightarrow \epsilon$ or $A_0(x_{01}, \ldots, x_{0n_0}) \rightarrow A_1(x_{11}, \ldots, x_{1n_1}) \ldots A_k(x_{k1}, \ldots, x_{kn_k})$ where x_{ij} is a concatenation of constants and/or variables and A_m is a predicate with arity n_m.[1]

As an illustration, consider the following RCG clauses:

$(a)\ \ A(X\ \ a\ \ Y\ \ Z) \rightarrow B(X,Y)\ \ \ C(Z) \qquad (b)\ \ B(\epsilon,\epsilon) \rightarrow \epsilon$

(a) has a left predicate labeled A, whose arity is 1 and whose argument is made of the concatenation of the variable X, the constant a and the variables

* We are grateful to Laura Kallmeyer and three anonymous reviewers for useful comments on this work.

[1] For a formal definition of RCG, please refer to [2].

Y and Z. (b) is a clause whose left predicate is labeled B and whose arity is 2. Its two arguments are ϵ. (b)'s right member is empty (ϵ).

The idea underlying RCG is that the symbols (constants or variables) occurring in the argument of a clause are bound to ranges of the sentence by a substitution mechanism. Derivation in RCG corresponds to the *instantiation* of the clauses whose left predicate is distinguished (axiom of the RCG). In other terms, for each of the axiom clauses, we search for all the substitutions mapping symbols of the left predicate to ranges of the sentence. If such a substitution can be found (successful instantiation), then the left predicate of the clause is replaced by the right-hand side of the clause (the arguments of the right predicates are now bound to ranges). We then look for clauses whose left-predicate is one of these instantiated right predicates. These clauses are in turn instantiated. The derivation ends either when ϵ has been derived (success), or when no successful clause instantiation has been found (failure). In other terms, an input string w with $|w| = n$ belongs to $L(G)$ iff ϵ can be derived wrt w from $S(\langle 0, n\rangle)$.

To illustrate this, consider the RCG $G = \langle \{S, A\}, \{a, b\}, \{X, Y\},\ S, P\rangle$ with:

$$P = \{\ S(X\,Y) \quad \rightarrow A(X,Y)\ ,\ A(a\,X, a\,Y) \rightarrow A(X,Y)\ ,\\ A(b\,X, b\,Y) \rightarrow A(X,Y)\ ,\ A(\epsilon,\epsilon) \quad \rightarrow \epsilon \quad \}$$

which covers the copy language: $L(G) = \{ww \mid w \in \{a, b\}^*\}$. Considering the string *aabbaabb*, we have the following derivation (s refers to the substitution function):

Clauses	**Instantiations**
$S(XY) \rightarrow A(X,Y)$	$s(X) = \langle 0,4\rangle\,(aabb)\ ,\ s(Y) = \langle 4,8\rangle\,(aabb)$
$A(aX, aY) \rightarrow A(X,Y)$	$s(X) = \langle 1,4\rangle\,(abb)\ ,\ s(Y) = \langle 5,8\rangle\,(abb)$
$A(aX, aY) \rightarrow A(X,Y)$	$s(X) = \langle 2,4\rangle\,(bb)\ ,\ s(Y) = \langle 6,8\rangle\,(bb)$
$A(bX, bY) \rightarrow A(X,Y)$	$s(X) = \langle 3,4\rangle\,(b)\ ,\ s(Y) = \langle 7,8\rangle\,(b)$
$A(bX, bY) \rightarrow A(X,Y)$	$s(X) = \langle 4,4\rangle\,(\epsilon)\ ,\ s(Y) = \langle 8,8\rangle\,(\epsilon)$
$A(\epsilon,\epsilon) \rightarrow \epsilon$	

2 Instantiating predicates using constraints

The state-of-the-art RCG parsing algorithm our work is based on is a top-down parsing algorithm presented in [1]. The idea of this algorithm is to use the start predicate to trigger clause instantiations leading to the empty string. A complex step of this algorithm corresponds to the instantiation of a predicate. Indeed, when instantiating a predicate, we have to compute all possible substitutions between range variables and contiguous symbols of the input string.

To illustrate this, consider the predicate $A(XYZ)$ to be instantiated with respect to the input string $abcdef$. In this example, the number of possible instantiations is 28 (there are 3 contiguous range variables, that is to say 2 inner boundaries to be found, the first of these boundaries can occupy one of 7 positions: $\bullet abcdef$, $a \bullet bcdef$, etc., the second one has either to be equal or to follow –immediately or not– the first boundary, so the number of instantiations is $7 + 6 + 5 + 4 + 3 + 2 + 1 = 28$).

The complexity of the instantiation task depends on the size of the input string and the number of range variables to instantiate. More precisely [1] has shown that the maximum parse time complexity associated with a clause instantiation is $\mathcal{O}(n^d)$, where n is the length of the input string and d is the number of free boundaries (also called degree) in that clause: $d = max(k_i + v_i)$, k_i being the arity of clause i and v_i its number of range variables. In other terms, the parsing time complexity heavily depends on the instantiation time complexity (as shown in [1], the worst parsing time of a string of length n is $\mathcal{O}(|G|n^d)$).

In this context, we propose to encode predicate instantiation as a *Constraint Satisfaction Problem* (CSP). Note that [1] proposes to deal with the high time-complexity of clause instantiation by using some predefined specific predicates whose role is to decrease the number of free boundaries within ranges.[2]

In the above example, we had a unary predicate whose argument was made of 3 range variables XYZ. Another common form of argument is a mix of constants and range variables, such as in $aXYdZ$. In such a case, the constants can be seen as constraints (or *anchors*) on the values the free boundaries can be assigned. We will elaborate on this after a brief introduction to CSP.[3]

Constraint Satisfaction Problems. In the *constraint satisfaction* paradigm, a problem is described using a set of variables, each taking its value in a given domain. Constraints are then applied on the values these variables can take in order to narrow their respective domains. Finally, we search for one (or all) solution(s) to the problem, that is to say we search for some (or all) assignment(s) of values to variables respecting the constraints.

One particularly interesting sub-class of CSPs are those that can be stated in terms of constraints on variables ranging over finite sets of non-negative integers. For such CSPs, there exist several implementations offering a wide range of constraints (arithmetic, boolean and linear constraints), and efficient solvers. Examples of such implementations include the Oz/Mozart environment and the Gecode library.[4]

Instantiating predicates as a CSP. As mentioned above, an argument of a predicate to be instantiated contains range variables and/or constants, the latter acting like constraints on the boundaries between ranges.[5] To illustrate this, consider the instantiations of the predicate $A(aXYdZ)$ with respect to the input string $abcdef$. For this example, we only have 3 solutions, depending on where to put the boundary between ranges X and Y.

[2] The number of range boundaries could be reduced by binarizing the clauses mimicking CFG binarization. The benefit of this is unclear as it would increase the number of clauses to check for instantiation.

[3] For a detailed introduction to CSP, please refer to [5].

[4] See http://www.mozart-oz.org and http://www.gecode.org.

[5] Note that (a) the RCGs we handle in our system are built automatically from lexicalized tree-based grammars, thus the arguments of a predicate often contain constants, nonetheless (b) the technique presented here does not depend on the presence of constants. These only reduce the search space of the CSP.

The idea underlying the interpretation of this instantiation task in terms of a CSP is to use the natural order of integers to represent the linear order imposed on ranges, and to define additional constraints reflecting the fact that constants (if any) are anchors for ranges of the input string. We do the following:

1. we define a model associating *boundary constants* with non-negative integers, and *boundary variables* with finite domains over non-negative integers,
2. we define constraints on these boundary variables,
3. we search for all assignments of values to these boundary variables.

Step 1. Let us define the input string w as follows: $w := b_0 s_1 b_1 s_2 \dots b_{n-1} s_n b_n$ where s_i $(1 \leq i \leq n)$ is a constant symbol of the input string, and b_j $(0 \leq j \leq n)$ is a *boundary constant*. For convenience, we note $w[i] = s_i$. Every boundary constant is associated with an integer referring to its position in the string (boundary constants are ordered by the relation $(\leq, \mathbb{N})$). Thus $b_0 = 0$, $b_1 = 1$, *etc.*

In the same way, let us define an argument to instantiate arg as follows: $arg := B'_0 s'_1 B'_1 s'_2 \dots B'_{m-1} s'_m B'_m$ where s'_i $(1 \leq i \leq m)$ is a symbol (range variable or constant), and B'_j $(0 \leq j \leq m)$ is a *boundary variable*. As before, we note $arg[i] = s'_i$. Furthermore, each boundary variable is associated with the finite domain $[0..n]$ (*i.e.* a boundary variable must match a boundary constant defined over the input string). Note that here we consider the case where all constants appearing in the input string and in the argument to instantiate occur only once (see *Nota bene* below).

Step 2. Once our model has been defined, we compute a constraint matrix M_C mapping boundary variables to boundary constants. Thus M_C is a $(m+1) \times (n+1)$ matrix where:

$$M_C[i,j] = \begin{cases} 1 \text{ if } arg[i] = w[j] \text{ or } arg[i-1] = w[j-1] \quad (2 \leq i \leq m, 2 \leq j \leq n) \\ 1 \text{ if } (i,j) = (1,1) \qquad (*) \\ 1 \text{ if } (i,j) = (m+1, n+1) \qquad (*) \\ 0 \text{ otherwise} \end{cases}$$

The "1" in M_C represent boundary positions that are constrained by the input string. The lines marked (*) represent the fact the lower and upper bounds of the argument to instantiate must be respectively the lower and upper bounds of the input string. If we consider the previous example of the predicate $A(aXYdZ)$ to be instantiated with $abcdef$, we obtain the following constraint matrix:

$$M_C = \begin{array}{c} \\ B'_0 \\ B'_1 \\ B'_2 \\ B'_3 \\ B'_4 \\ B'_5 \end{array} \begin{array}{c} \begin{array}{ccccccc} b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Step 3. We finally search for all assignments of values in $[0..n]$ to the boundary variables B'_j $(0 \leq j \leq m)$. In other terms, we search for all functions f such that:

$$\begin{array}{rccc} f: & \mathbb{V} & \rightarrow & [0..n] \\ & B'j\ (0 \leq j \leq m) & \longmapsto & b_i\ (0 \leq i \leq n) \end{array}$$

This search uses generic constraints reflecting the ordering of the boundary variables: $\forall\,(0 \leq i, j \leq m)\ \ (i \leq j) \Rightarrow 0 \leq (f(B'_i) \leq f(B'_j)) \leq n$ and the specific constraints encoded in the matrix M_C:

$$\forall (1 \leq i \leq m+1, 1 \leq j \leq n+1) \qquad (M_C[i,j] = 1) \Rightarrow (f(B'_{i-1}) = b_{j-1})$$

In the latter formula, the indexes of the boundaries are shifted with respect to the matrix indexes (i, j) because M_C's rows and columns indexes start from 1 while the indexes of the boundaries start from 0. Considering our previous example, all B'_i are constrained by M_C, except B'_2, which can take 3 values: 1 (b_1), 2 (b_2) or 3 (b_3), these are the 3 expected range boundaries.

Nota bene Here, we gave a formal definition of the CSP-encoding. At first sight, the example we used looks trivial and one may wonder whether we really need all this formal tool, in other terms whether the complexity of instantiation has not been overestimated. To illustrate instantiation's complexity, one may think of arguments with duplicated constants, such as in the instantiation of $XaYaZ$ with *aaaad*. Which constant of the string refers to which constant of the argument to instantiate ? The approach presented here has to be generalized to deal with such cases. More precisely, we use a CSP to assign a constraining role to all potential anchors. For each of these assignments, we compute corresponding range instantiations using the CSP introduced above.

Finally, it is worth noticing that there doubtlessly exist several ways of instantiating RCG predicates (*e.g.* unification based solvers). The use of constraints over finite sets of integers offers a natural framework for handling ranges.

3 Conclusion and future work

In this paper, we paid a particular attention to the task of RCG predicate instantiation, on which the time complexity of RCG parsing heavily depends. We proposed to use techniques borrowed from the field of *constraint programming* to efficiently perform this task. The ideas presented here have led to the development of an open-source parsing architecture, which is currently used for designing a core Multi-Component Tree Adjoining Grammar for German [4].

In a near future, we would like to build a proof of correctness of the CSP-encoding and also to evaluate empirically the benefits of using CSP for predicate instantiation compared with existing approaches.

References

1. Boullier, P.: Range Concatenation Grammars. In: Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT 2000). (2000) 53–64
2. Boullier, P.: Proposal for a Natural Language Processing Syntactic Backbone. INRIA report 3342 (1998)
3. Boullier, P.: On TAG and Multicomponent TAG Parsing. INRIA report 3668 (1999)
4. Kallmeyer, L., Lichte, T., Maier, W., Parmentier, Y., Dellert, J., Evang, K.: The Tübingen Linguistic Parsing Architecture (2008) http://sourcesup.cru.fr/tulipa.
5. Schulte, C.: Programming Constraint Services. Volume 2302 of Lecture Notes in Artificial Intelligence. Springer-Verlag (2002)