

# The Theory of Grammar Constraints

Meinolf Sellmann

Brown University  
Department of Computer Science  
115 Waterman Street, P.O. Box 1910  
Providence, RI 02912  
sello@cs.brown.edu

**Abstract.** By introducing the Regular Membership Constraint, Gilles Pesant pioneered the idea of basing constraints on formal languages. The paper presented here is highly motivated by this work, taking the obvious next step, namely to investigate constraints based on grammars higher up in the Chomsky hierarchy. We devise an arc-consistency algorithm for context-free grammars, investigate when logic combinations of grammar constraints are tractable, show how to exploit non-constant size grammars and reorderings of languages, and study where the boundaries run between regular, context-free, and context-sensitive grammar filtering.

**Keywords:** global constraints, regular grammar constraints, context-free grammar constraints, constraint filtering

## 1 Introduction

With the introduction of the regular language membership constraint [9, 10, 2], a new field of study for filtering algorithms has opened. Given the great expressiveness of formal grammars and their (at least for someone with a background in computer science) intuitive usage, grammar constraints are extremely attractive modeling entities that subsume many existing definitions of specialized global constraints. Moreover, Pesant’s implementation [8] of the regular grammar constraint has shown that this type of filtering can also be performed incrementally and generally so efficiently that it even rivals custom filtering algorithms for special regular grammar constraints like Stretch and Pattern [9, 4].

In this paper, we theoretically investigate filtering problems that arise from grammar constraints. We answer questions like: Can we efficiently filter context-free grammar constraints? How can we achieve arc-consistency for conjunctions of regular grammar constraints? Given that we can allow non-constant grammars and reordered languages for the purposes of constraint filtering, what languages are suited for filtering based on regular and context-free grammar constraints? Are there languages that are suited for context-free, but not for regular grammar filtering?

Particularly, after recalling some essential basic concepts from the theory of formal languages in the next section, we devise an efficient arc-consistency algorithm that propagates context-free grammar constraints in Section 3. Then, in Section 4, we study how logic combinations of grammar constraints can be propagated efficiently. Finally, we investigate non-constant size grammars and reorderings of languages in Section 5.

## 2 Basic Concepts

We start our work by reviewing some well-known definitions from the theory of formal languages. For a full introduction, we refer the interested reader to [6]. All proofs that are omitted in this paper can also be found there.

**Definition 1 (Alphabet and Words).** Given sets  $Z$ ,  $Z_1$ , and  $Z_2$ , with  $Z_1Z_2$  we denote the set of all sequences or strings  $z = z_1z_2$  with  $z_1 \in Z_1$  and  $z_2 \in Z_2$ , and we call  $Z_1Z_2$  the concatenation of  $Z_1$  and  $Z_2$ . Then, for all  $n \in \mathbb{N}$  we denote with  $Z^n$  the set of all sequences  $z = z_1z_2 \dots z_n$  with  $z_i \in Z$  for all  $1 \leq i \leq n$ . We call  $z$  a word of length  $n$ , and  $Z$  is called an alphabet or set of letters. The empty word has length 0 and is denoted by  $\varepsilon$ . It is the only member of  $Z^0$ . We denote the set of all words over the alphabet  $Z$  by  $Z^* := \bigcup_{n \in \mathbb{N}} Z^n$ . In case that we wish to exclude the empty word, we write  $Z^+ := \bigcup_{n \geq 1} Z^n$ .

**Definition 2 (Grammar).** A grammar is a four-tuple  $G = (\Sigma, N, P, S_0)$  where  $\Sigma$  is the alphabet,  $N$  is a finite set of non-terminals,  $P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$  is the set of productions, and  $S_0 \in N$  is the start non-terminal. We will always assume that  $N \cap \Sigma = \emptyset$ .

*Remark 1.* We will use the following convention: Capital letters A, B, C, D, and E denote non-terminals, lower case letters a, b, c, d, and e denote letters in  $\Sigma$ , Y and Z denote symbols that can either be letters or non-terminals, u, v, w, x, y, and z denote strings of letters, and  $\alpha$ ,  $\beta$ , and  $\gamma$  denote strings of letters and non-terminals. Moreover, productions  $(\alpha, \beta)$  in  $P$  can also be written as  $\alpha \rightarrow \beta$ .

**Definition 3 (Derivation and Language).**

- Given a grammar  $G = (\Sigma, N, P, S_0)$ , we write  $\alpha\beta_1\gamma \xRightarrow{G} \alpha\beta_2\gamma$  iff there exists a production  $\beta_1 \rightarrow \beta_2 \in P$ . We write  $\alpha_1 \xRightarrow{*G} \alpha_m$  iff there exists a sequence of strings  $\alpha_2, \dots, \alpha_{m-1}$  such that  $\alpha_i \xRightarrow{G} \alpha_{i+1}$  for all  $1 \leq i < m$ . Then, we say that  $\alpha_m$  can be derived from  $\alpha_1$ .
- We define the language given by  $G$  to be  $L_G := \{w \in \Sigma^* \mid S_0 \xRightarrow{*G} w\}$ .

Definition 2 gives a very general form of grammars which is known to be Turing machine equivalent. Consequently, reasoning about languages given by general grammars is infeasible. For example, the word problem for grammars as defined above is undecidable.

**Definition 4 (Word Problem).** Given a grammar  $G = (\Sigma, N, P, S_0)$  and a word  $w \in \Sigma^*$ , the word problem consists in answering the question whether  $w \in L_G$ .

Therefore, in the theory of formal languages, more restricted forms of grammars have been defined. Noam Chomsky introduced a hierarchy of decreasingly complex sets of languages [5]. In this hierarchy, the grammars given in Definition 2 are called Type-0 grammars. In the following, we define the Chomsky hierarchy of formal languages.

**Definition 5 (Context-Sensitive, Context-Free, and Regular Grammars).**

- Given a grammar  $G = (\Sigma, N, P, S_0)$  such that for all productions  $\alpha \rightarrow \beta \in P$  we have that  $\beta$  is at least as long as  $\alpha$ , then we say that the grammar  $G$  and the language  $L_G$  are context-sensitive. In Chomsky's hierarchy, these grammars are known as Type-1 grammars.
- Given a grammar  $G = (\Sigma, N, P, S_0)$  such that  $P \subseteq N \times (N \cup \Sigma)^*$ , we say that the grammar  $G$  and the language  $L_G$  are context-free. In Chomsky's hierarchy, these grammars are known as Type-2 grammars.
- Given a grammar  $G = (\Sigma, N, P, S_0)$  such that  $P \subseteq N \times (\Sigma^* N \cup \Sigma^*)$ , we say that  $G$  and the language  $L_G$  are right-linear. In Chomsky's hierarchy, these grammars are known as Type-3 grammars.

*Remark 2.* The word problem becomes easier as the grammars become more and more restricted: For context-sensitive grammars, the problem is already decidable, but unfortunately PSPACE-complete. For context-free languages, the word problem can be answered in polynomial time. For Type-3 languages, the word problem can even be decided in time linear in the length of the given word.

For all grammars mentioned above there exists an equivalent definition based on some sort of automaton that accepts the respective language. As mentioned earlier, for Type-0 grammars, that automaton is the Turing machine. For context-sensitive languages it is a Turing machine with a linearly space-bounded tape. For context-free languages, it is the so-called push-down automaton (in essence a Turing machine with a stack rather than a tape). And for right-linear languages, it is the finite automaton (which can be viewed as a Turing machine with only one read-only input tape on which it cannot move backwards). Depending on what one tries to prove about a certain class of languages, it is convenient to be able to switch back and forth between different representations (i.e. grammars or automata). In this work, when reasoning about context-free languages, it will be most convenient to use the grammar representation. For right-linear languages, however, it is often more convenient to use the representation based on finite automata:

**Definition 6 (Finite Automaton).** Given a finite set  $\Sigma$ , a finite automaton  $A$  is defined as a tuple  $A = (Q, \Sigma, \delta, q_0, F)$ , where  $Q$  is a set of states,  $\Sigma$  denotes the alphabet of our language,  $\delta \subseteq Q \times \Sigma \times Q$  defines the transition function,  $q_0$  is the start state, and  $F$  is the set of final states. A finite automaton is called deterministic iff  $(q, a, p_1), (q, a, p_2) \in \delta$  implies that  $p_1 = p_2$ .

**Definition 7 (Accepted Language).** The language defined by a finite automaton  $A$  is the set  $L_A := \{w = (w_1, \dots, w_n) \in \Sigma^* \mid \exists (p_0, \dots, p_n) \in Q^n \forall 1 \leq i \leq n : (p_{i-1}, w_i, p_i) \in \delta \text{ and } p_0 = q_0, p_n \in F\}$ .  $L_A$  is called a regular language.

**Lemma 1.** For every right-linear grammar  $G$  there exists a finite automaton  $A$  such that  $L_A = L_G$ , and vice versa.

Consequently, we can use the terms right-linear and regular synonymously.

### 3 Context-Free Grammar Constraints

Within constraint programming it would be convenient to use formal languages to describe certain features that we would like our solutions to exhibit. It is worth noting here that any constraint and conjunction of constraints really defines a formal language by itself when we view the instantiations of variables  $X_1, \dots, X_n$  with domains  $D_1, \dots, D_n$  as forming a word in  $D_1 D_2 \dots D_n$ . Conversely, if we want a solution to belong to a certain formal language in this view, then we need appropriate constraints and constraint filtering algorithms that will allow us to express and solve such constraint programs efficiently. We formalize the idea by defining grammar constraints.

**Definition 8 (Grammar Constraint).** *For a given grammar  $G = (\Sigma, N, P, S_0)$  and variables  $X_1, \dots, X_n$  with domains  $D_1 := D(X_1), \dots, D_n := D(X_n) \subseteq \Sigma$ , we say that  $\text{Grammar}_G(X_1, \dots, X_n)$  is true for an instantiation  $X_1 \leftarrow w_1, \dots, X_n \leftarrow w_n$  iff it holds that  $w = w_1 \dots w_n \in L_G \cap D_1 \times \dots \times D_n$ .*

Gilles Pesant pioneered the idea to exploit formal grammars for constraint programming by considering regular languages [9, 10]. Based on the review of our knowledge of formal languages in the previous section, we can now ask whether we can also develop efficient filtering algorithms for grammar constraints of higher-orders. Clearly, for Type-0 grammars, this is not possible, since the word problem is already undecidable. For context-sensitive languages, the word problem is PSPACE complete, which means that even checking the corresponding grammar constraint is computationally intractable.

However, for context-free languages deciding whether a given word belongs to the language can be done in polynomial time. Context-free grammar constraints come in particularly handy when we need to look for a recursive sequence of nested objects. Consider for instance the puzzle of forming a mathematical term based on two occurrences of the numbers 3 and 8, operators  $+$ ,  $-$ ,  $*$ ,  $/$ , and brackets  $(, )$ , such that the term evaluates to 24. The generalized problem is NP-hard, but when formulating the problem as a constraint program, with the help of a context-free grammar constraint we can easily express the syntactic correctness of the term formed. Or, closer to the real-world, consider the task of organizing a group of workers into a number of teams of unspecified size, each team with one team leader and one project manager who is the head of all team leaders. This organizational structure can be captured easily by a combination of an AllDifferent and a context-free grammar constraint. Therefore, in this section we will develop an algorithm that propagates context-free grammar constraints.

#### 3.1 Parsing Context-Free Grammars

One of the most famous algorithms for parsing context-free grammars is the algorithm by Cocke, Younger, and Kasami (CYK). It takes as input a word  $w \in \Sigma^n$  and a context-free grammar  $G = (\Sigma, N, P, S_0)$  in some special form and decides in time  $O(n^3|P|)$  whether it holds that  $w \in L_G$ . The algorithm is based on the dynamic programming principle. In order to keep the recursion equation under control, the algorithm needs to assume that all productions are length-bounded on the right-hand side.

**Definition 9 (Chomsky Normal Form).** A context-free grammar  $G = (\Sigma, N, P, S_0)$  is said to be in Chomsky Normal Form iff for all productions  $A \rightarrow \alpha \in P$  we have that  $\alpha \in \Sigma^1 \cup N^2$ .

**Lemma 2.** Every context free grammar  $G$  such that  $\varepsilon \notin L_G$  can be transformed into a grammar  $H$  such that  $L_G = L_H$  and  $H$  is in Chomsky Normal Form.

The proof of this lemma is given in [6]. It is important to note that the proof is constructive but that the resulting grammar  $H$  may be exponential in size of  $G$ , which is really due to the necessity to remove all productions  $A \rightarrow \varepsilon$ . When we view the grammar size as constant (i.e. if the size of the grammar is independent of the word-length as it is commonly assumed in the theory of formal languages), then this is not an issue. As a matter of fact, in most references one will simply read that CYK could solve the word problem for any context-free language in cubic time. For now, let us assume that indeed all grammars given can be treated as having constant-size, and that our asymptotic analysis only takes into account the increasing word lengths. We will come back to this point later in Section 4 when we discuss logic combinations of grammar constraints, and in Section 5 when we discuss the possibility of non-constant grammars and reorderings.

Now, given a word  $w \in \Sigma^n$ , let us denote the sub-sequence  $w_i w_{i+1} \dots w_{i+j-1}$  by  $w_{ij}$ . Based on a grammar  $G = (\Sigma, N, P, S_0)$  in Chomsky Normal Form, CYK determines iteratively the set of all non-terminals from where we can derive  $w_{ij}$ , i.e.  $S_{ij} := \{A \in N \mid A \xrightarrow{*}_G w_{ij}\}$  for all  $1 \leq i \leq n$  and  $1 \leq j \leq n - i$ . It is easy to initialize the sets  $S_{i1}$  just based on  $w_i$  and all productions  $A \rightarrow w_i \in P$ . Then, for  $j$  from 2 to  $n$  and  $i$  from 1 to  $n - j + 1$ , we have that

$$S_{ij} = \bigcup_{k=1}^{j-1} \{A \mid A \rightarrow BC \in P \text{ with } B \in S_{ik} \text{ and } C \in S_{i+k, j-k}\}. \quad (1)$$

Then,  $w \in L_G$  iff  $S_0 \in S_{1n}$ . From the recursion equation it is simple to derive that CYK can be implemented to run in time  $O(n^3|P|) = O(n^3)$  when we treat the size of the grammar as a constant.

### 3.2 Example

Assume we are given the following context-free, normal-form grammar  $G = (\{\}, \{\}, \{A, B, C, S_0\}, \{S_0 \rightarrow AC, S_0 \rightarrow S_0 S_0, S_0 \rightarrow BC, B \rightarrow AS_0, A \rightarrow [ , C \rightarrow ]\}, S_0)$  that gives the language  $L_G$  of all correctly bracketed expressions (like, for example, “[[]]”) or “[[]]”). Given the word “[[]]”, CYK first sets  $S_{11} = S_{31} = S_{41} = \{A\}$ , and  $S_{21} = S_{51} = S_{61} = \{C\}$ . Then it determines the non-terminals from which we can derive sub-sequences of length 2:  $S_{12} = S_{42} = \{S_0\}$  and  $S_{22} = S_{32} = S_{52} = \emptyset$ . The only other non-empty sets that CYK finds in iterations regarding longer sub-sequences are  $S_{34} = \{S_0\}$  and  $S_{16} = \{S_0\}$ . Consequently, since  $S_0 \in S_{16}$ , CYK decides (correctly) that “[[]]”  $\in L_G$ .

1. We run the dynamic program based on recursion equation 1 with initial sets  $S_{i1} := \{A \mid A \rightarrow v \in P, v \in D_i\}$ .
2. We define the directed graph  $Q = (V, E)$  with node set  $V := \{v_{ijA} \mid A \in S_{ij}\}$  and arc set  $E := E_1 \cup E_2$  with  $E_1 := \{(v_{ijA}, v_{ikB}) \mid \exists C \in S_{i+k, j-k} : A \rightarrow BC \in P\}$  and  $E_2 := \{(v_{ijA}, v_{i+k, j-k, C}) \mid \exists B \in S_{ik} : A \rightarrow BC \in P\}$  (see Figure 1).
3. Now, we remove all nodes and arcs from  $Q$  that cannot be reached from  $v_{1n, S_0}$  and denote the resulting graph by  $Q' := (V', E')$ .
4. We define  $S'_{ij} := \{A \mid v_{ijA} \in V'\} \subseteq S_{ij}$ , and set  $D'_i := \{v \mid \exists A \in S'_{i1} : A \rightarrow v \in P\}$ .

**Algorithm 1:** CFCG Filtering Algorithm

### 3.3 Context-Free Grammar Filtering

We denote a given grammar constraint  $Grammar_G(X_1, \dots, X_n)$  over a context-free grammar  $G$  in Chomsky Normal Form by  $CFGCG(X_1, \dots, X_n)$ . Obviously, we can use CYK to determine whether  $CFGCG(X_1, \dots, X_n)$  is satisfied for a full instantiation of the variables, i.e. we could use the parser for generate-and-test purposes. In the following, we show how we can augment CYK to a filtering algorithm that achieves generalized arc-consistency for  $CFGCG$ .

First, we observe that we can check the satisfiability of the constraint by making just a very minor adjustment to CYK. Given the domains of the variables, we can decide whether there exists a word  $w \in D_1 \dots D_n$  such that  $w \in L_G$  simply by adding all non-terminals  $A$  to  $S_{i1}$  for which there exists a production  $A \rightarrow v \in P$  with  $v \in D_i$ . From the correctness of CYK it follows trivially that the constraint is satisfiable iff  $S_0 \in S_{1n}$ . The runtime of this algorithm is the same as that for CYK.

As usual, whenever we have a polynomial-time algorithm that can decide the satisfiability of a constraint, we know already that achieving arc-consistency is also computationally tractable. A brute force approach could simply probe values by setting  $D_i := \{v\}$ , for every  $1 \leq i \leq n$  and every  $v \in D_i$ , and checking whether the constraint is still satisfiable or not. This method would result in a runtime in  $O(n^4 D|P|)$ , where  $D \leq |\Sigma|$  is the size of the largest domain  $D_i$ .

We will now show that we can achieve a much improved filtering time. The core idea is once more to exploit Mike Trick's method of filtering dynamic programs [11]. Roughly speaking, when applied to our CYK-constraint checker, Trick's method simply reverses the recursion process after it has assured that the constraint is satisfiable so as to see which non-terminals in the sets  $S_{i1}$  can actually be used in the derivation of any word  $w \in L_G \cap (D_1 \dots D_n)$ . The methodology is formalized in Algorithm 1.

**Lemma 3.** *In Algorithm 1:*

1. It holds that  $A \in S_{ij}$  iff there exists a word  $w_i \dots w_{i+j-1} \in D_i \dots D_{i+j-1}$  such that  $A \xrightarrow{*}_G w_i \dots w_{i+j-1}$ .
2. It holds that  $B \in S'_{ik}$  iff there exists a word  $w \in L_G \cap (D_1 \dots D_n)$  such that  $S_0 \xrightarrow{*}_G w_1 \dots w_{i-1} B w_{i+k} \dots w_n$ .

*Proof.* 1. We induce over  $j$ . For  $j = 1$ , the claim holds by definition of  $S_{i1}$ . Now assume  $j > 1$  and that the claim is true for all  $S_{ik}$  with  $1 \leq k < j$ . Now, by definition of  $S_{ij}$ ,  $A \in S_{ij}$  iff there exists a  $1 \leq k < j$  and a production  $A \rightarrow BC \in P$  such that  $B \in S_{ik}$  and  $C \in S_{i+k,j-k}$ . Thus,  $A \in S_{ij}$  iff there exist  $w_{ik} \in D_i \dots D_{i+k-1}$  and  $w_{i+k,j-k} \in D_{i+k} \dots D_{i+j-1}$  such that  $A \xrightarrow[G]{*} w_{ik} w_{i+k,j-k}$ .

2. We induce over  $k$ , starting with  $k = n$  and decreasing to  $k = 1$ . For  $k = n$ ,  $S'_{1k} = S'_{1n} \subseteq \{S_0\}$ , and it is trivially true that  $S_0 \xrightarrow[G]{*} S_0$ . Now let us assume the claim holds for all  $S'_{ij}$  with  $k < j \leq n$ . Choose any  $B \in S'_{ik}$ . According to the definition of  $S'_{ik}$  there exists a path from  $v_{1nS_0}$  to  $v_{ikB}$ . Let  $(v_{ijA}, v_{ikB}) \in E_1$  be the last arc on any one such path (the case when the last arc is in  $E_2$  follows analogously). By the definition of  $E_1$  there exists a production  $A \rightarrow BC \in P$  with  $C \in S_{i+k,j-k}$ . By induction hypothesis, we know that there exists a word  $w \in L_G \cap (D_1 \dots D_n)$  such that  $S_0 \xrightarrow[G]{*} w_1 \dots w_{i-1} A w_{i+j} \dots w_n$ . Thus,  $S_0 \xrightarrow[G]{*} w_1 \dots w_{i-1} BC w_{i+j} \dots w_n$ . And therefore, with (1) and  $C \in S_{i+k,j-k}$ , there exists a word  $w_{i+k} \dots w_{i+j-1} \in D_{i+k} \dots D_{i+j-1}$  such that  $S_0 \xrightarrow[G]{*} w_1 \dots w_{i-1} B w_{i+k} \dots w_{i+j-1} w_{i+j} \dots w_n$ . Since we can also apply (1) to non-terminal  $B$ , we have proven the claim.  $\square$

**Theorem 1.** *Algorithm 1 achieves generalized arc-consistency for the CFGC.*

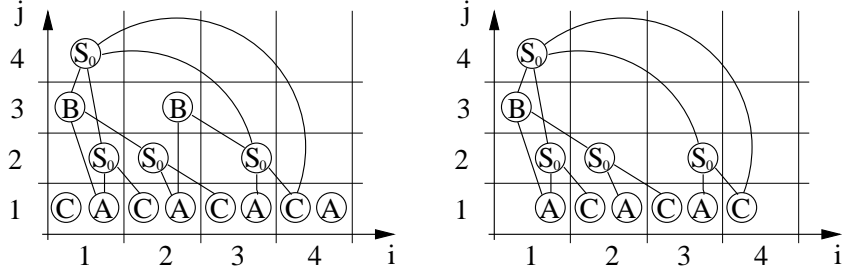
*Proof.* We show that  $v \notin D'_i$  iff for all words  $w = w_1 \dots w_n \in L_G \cap (D_1 \dots D_n)$  it holds that  $v \neq w_i$ .

$\Rightarrow$  (Correctness) Let  $v \notin D'_i$  and  $w = w_1 \dots w_n \in L_G \cap (D_1 \dots D_n)$ . Due to  $w \in L_G$  there must exist a derivation  $S_0 \xrightarrow[G]{*} w_1 \dots w_{i-1} A w_{i+1} \dots w_n \xrightarrow[G]{*} w_1 \dots w_{i-1} w_i w_{i+1} \dots w_n$  for some  $A \in N$  with  $A \rightarrow w_i \in P$ . According to Lemma 3,  $A \in S'_{i1}$ , and thus  $w_i \in D'_i$ , which implies  $v \neq w_i$  as  $v \notin D'_i$ .

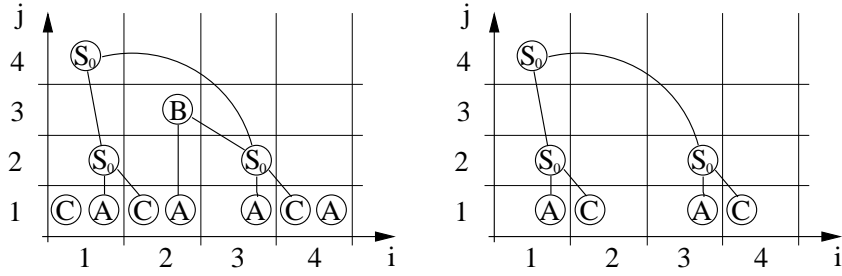
$\Leftarrow$  (Effectiveness) Now let  $v \in D'_i \subseteq D_i$ . According to the definition of  $D'_i$ , there exists some  $A \in S'_{i1}$  with  $A \rightarrow v \in P$ . With Lemma 3 we know that then there exists a word  $w \in L_G \cap (D_1 \dots D_n)$  such that  $S_0 \xrightarrow[G]{*} w_1 \dots w_{i-1} A w_{i+1} \dots w_n$ .

Thus, it holds that  $S_0 \xrightarrow[G]{*} w_1 \dots w_{i-1} v w_{i+1} \dots w_n \in L_G \cap (D_1 \dots D_n)$ .  $\square$

We now have a filtering algorithm that achieves generalized arc-consistency for context-free grammar constraints. Since the computational effort is dominated by carrying out the recursion equation, Algorithm 1 runs asymptotically in the same time as CYK. In essence, this implies that checking one complete assignment via CYK is as costly as performing full arc-consistency filtering for CFGC. Clearly, achieving arc-consistency for a grammar constraint is at least as hard as parsing. Now, there exist faster parsing algorithms for context-free grammars. For example, the fastest known algorithm was developed by Valiant and parses context-free grammars in time  $O(n^{2.8})$ . While this is only moderately faster than the  $O(n^3)$  that CYK requires, there also exist special purpose parsers for non-ambiguous context-free grammars (i.e. grammars



**Fig. 1.** Context-Free Filtering: Assume we are given the context-free grammar from section 3.2 again. A rectangle with coordinates  $(i, j)$  contains one node  $v_{ijA}$  for each non-terminal  $A$  in the set  $S_{ij}$ . All arcs are considered to be directed from top to bottom. The left picture shows the situation after step (2).  $S_0$  is in  $S_{14}$ , therefore the constraint is satisfiable. The right picture illustrates the shrunken graph with sets  $S'_{ij}$  after all parts have been removed that cannot be reached from node  $v_{14S_0}$ . We see that the value ']' will be removed from  $D_1$  and '[' from  $D_4$ .



**Fig. 2.** We show how the algorithm works when the initial domain of  $X_3$  is  $D_3 = \{\}$ . The left picture shows sets  $S_{ij}$  and the right the sets  $S'_{ij}$ . We see that the constraint filtering algorithm determines the only word in  $L_G \cap D_1 \dots D_4$  is "[ ]".

where each word in the language has exactly one parse tree) that run in  $O(n^2)$ . Now, it is known that there exist inherently ambiguous context-free languages, so these parsers lack some generality. However, in case that a user specifies a grammar that is non-ambiguous it would actually be nice to have a filtering algorithm that runs in quadratic rather than cubic time. It is a matter of further research to find out whether grammar constraint propagation can be done faster for non-ambiguous context-free grammars.

## 4 Logic Combinations of Grammar Constraints

We define regular grammar constraints analogously to  $CFG_C$ , but as in [10] we base it on automata rather than right-linear grammars:

**Definition 10 (Regular Grammar Constraint).** Given a finite automaton  $A$  and a right-linear grammar  $G$  with  $L_A = L_G$ , we set

$$RGCA(X_1, \dots, X_n) := Grammar_G(X_1, \dots, X_n).$$

Efficient arc-consistency algorithms for *RGCs* have been developed in [9, 10]. Now equipped with efficient filtering algorithms for regular and context-free grammar constraints, in the spirit of [7, 1, 3] we focus on certain questions that arise when a problem is modeled by logic combinations of these constraints. An important aspect when investigating logical combinations of grammar constraints is under what operations the given class of languages is closed. For example, when given a conjunction of regular grammar constraints, the question arises whether the conjunction of the constraints could not be expressed as one global *RGC*. This question can be answered affirmatively since the class of regular languages is known to be closed under intersection. In the following we summarize some relevant, well-known results for formal languages (see for instance [6]).

**Lemma 4.** *For every regular language  $L_{A^1}$  based on the finite automaton  $A^1$  there exists a deterministic finite automaton  $A^2$  such that  $L_{A^1} = L_{A^2}$ .*

*Proof.* When  $Q^1 = \{q_0, \dots, q_{n-1}\}$ , we set  $A^2 := (Q^2, \Sigma, \delta^2, q_0^2, F^2)$  with  $Q^2 := 2^{Q^1}$ ,  $q_0^2 = \{q_0^1\}$ ,  $\delta^2 := \{(P, a, R) \mid R = \{r \in Q^1 \mid \exists p \in P : (p, a, r) \in \delta^1\}\}$ , and  $F^2 := \{P \subseteq Q^1 \mid \exists p \in P \cap F^1\}$ . With this construction, it is easy to see that  $L_{A^1} = L_{A^2}$ .  $\square$

We note that the proof above gives a construction that can change the properties of the language representation, just like we had noted it earlier for context-free grammars that we had transformed into Chomsky Normal Form first before we could apply CYK for parsing and filtering. And just like we were faced with an exponential blow-up of the representation when bringing context-free grammars into normal-form, we see the same again when transforming a non-deterministic finite automaton of a regular language into a deterministic one.

**Theorem 2.** *Regular languages are closed under the following operations:*

- Union
- Intersection
- Complement

*Proof.* Given two regular languages  $L_{A^1}$  and  $L_{A^2}$  with respective finite automata  $A^1 = (Q^1, \Sigma, \delta^1, q_0^1, F^1)$  and  $A^2 = (Q^2, \Sigma, \delta^2, q_0^2, F^2)$ , without loss of generality, we may assume that the sets  $Q^1$  and  $Q^2$  are disjoint and do not contain symbol  $q_0^3$ .

- We define  $Q^3 := Q^1 \cup Q^2 \cup \{q_0^3\}$ ,  $\delta^3 := \delta^1 \cup \delta^2 \cup \{(q_0^3, a, q) \mid (q_0^1, a, q) \in \delta^1 \text{ or } (q_0^2, a, q) \in \delta^2\}$ , and  $F^3 := F^1 \cup F^2$ . Then, it is straight-forward to see that the automaton  $A^3 := (Q^3, \Sigma, \delta^3, q_0^3, F^3)$  defines  $L_{A^1} \cup L_{A^2}$ .
- We define  $Q^3 := Q^1 \times Q^2$ ,  $\delta^3 := \{((q^1, q^2), a, (p^1, p^2)) \mid \exists (q^1, a, p^1) \in \delta^1, (q^2, a, p^2) \in \delta^2\}$ , and  $F^3 := F^1 \times F^2$ . The automaton  $A^3 := (Q^3, \Sigma, \delta^3, (q_0^1, q_0^2), F^3)$  defines  $L_{A^1} \cap L_{A^2}$ .
- According to Lemma 4, we may assume that  $A^1$  is a deterministic automaton. Then,  $(Q^1, \Sigma, \delta^1, q_0^1, Q^1 \setminus F^1)$  defines  $L_{A^1}^C$ .  $\square$

The results above suggest that any logic combination (disjunction, conjunction, and negation) of *RGCs* can be expressed as one global *RGC*. While this is true in principle, from a computational point of view, the size of the resulting automaton needs to be taken into account. In terms of disjunctions of *RGCs*, all that we need to observe is that the algorithm developed in [9] actually works with non-deterministic automata as well. In the following, denote by  $m$  an upper bound on the number of states in all automata involved, and denote the size of the alphabet  $\Sigma$  by  $D$ . We obtain our first result for disjunctions of regular grammar constraints:

**Lemma 5.** *Given *RGCs*  $R_1, \dots, R_k$ , all over variables  $X_1, \dots, X_n$  in that order, we can achieve arc-consistency for the global constraint  $\bigvee_i R_i$  in time  $O((km + k)nD) = O(nDk)$  for automata with constant state-size  $m$ .*

If all that we need to consider are disjunctions of *RGCs*, then the result above is subsumed by the well known technique of achieving arc-consistency for disjunctive constraints which simply consists in removing, for each variable domain, the intersection of all values removed by the individual constraints. However, when considering conjunctions over disjunctions the result above is interesting as it allows us to treat a disjunctive constraint over *RGCs* as one new *RGC* of slightly larger size.

Now, regarding conjunctions of *RGCs*, we find the following result:

**Lemma 6.** *Given *RGCs*  $R_1, \dots, R_k$ , all over variables  $X_1, \dots, X_n$  in that order, we can achieve arc-consistency for the global constraint  $\bigwedge_i R_i$  in time  $O(nDm^k)$ .*

Finally, for the complement of a regular constraint, we have:

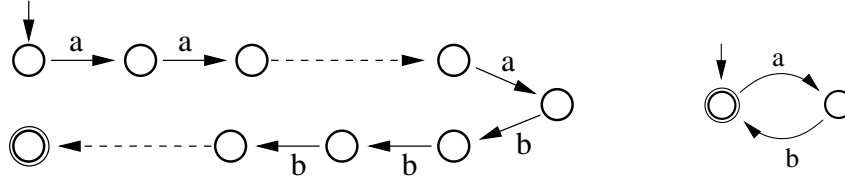
**Lemma 7.** *Given an *RGC*  $R$  based on a deterministic automaton, we can achieve arc-consistency for the constraint  $\neg R$  in time  $O(nDm) = O(nD)$  for an automaton with constant state-size.*

*Proof.* Lemmas 5- 7 are an immediate consequence of the results in [9] and the constructive proof of Theorem 2.  $\square$

Note that the lemma above only covers *RGCs* for which we know a deterministic finite automaton. However, when negating a disjunction of regular grammar constraints, the automaton to be negated is non-deterministic. Fortunately, this problem can be entirely avoided: When the initial automata associated with the elementary constraints of a logic combination of regular grammar constraints are deterministic, we can apply the rule of DeMorgan so as to only have to apply negations to the original constraints rather than the non-deterministic disjunctions or conjunctions thereof. With this method, we have:

**Corollary 1.** *For any logic combination (disjunction, conjunction, and negation) of deterministic *RGCs*  $R_1, \dots, R_k$ , all over variables  $X_1, \dots, X_n$  in that order, we can achieve generalized arc-consistency in time  $O(nDm^k)$ .*

Regarding logic combinations of context-free grammar constraints, unfortunately we find that this class of languages is not closed under intersection and complement, and the mere disjunction of context-free grammar constraints is not interesting given the standard methods for handling disjunctions. We do know, however, that context-free languages are closed under intersection with regular languages. It is a subject of further research to assess how big the resulting grammars can become.



**Fig. 3.** Regular grammar filtering for  $\{a^n b^n\}$ . The left figure shows a linear-size automaton, the right an automaton that accepts a reordering of the language.

## 5 Limits of the Expressiveness of Grammar Constraints

So far we have been very careful to mention explicitly how the size of the state-space of a given automaton or how the size of the set of non-terminals of a grammar influences the running time of our filtering algorithms. From the theory of formal languages' viewpoint, this is rather unusual, since here the interest lies purely in the asymptotic runtime with respect to the word-length. For the purposes of constraint programming, however, a grammar may very well be generated on the fly and may depend on the word-length, whenever this can be done efficiently. This fact makes grammar constraints even more expressive and powerful tools from the modeling perspective. Consider for instance the context-free language  $L = \{a^n b^n\}$  that is well-known not to be regular. Note that, within a constraint program, the length of the word is known — simply by considering the number of variables that define the scope of the grammar constraint. Now, by allowing the automaton to have  $2n + 1$  states, we can express that the first  $n$  variables shall take the value  $a$  and the second  $n$  variables shall take the value  $b$  by means of a regular grammar constraint. Of course, larger automata also result in more time that is needed for propagation. However, as long as the grammar is polynomially bounded in the word-length, we can still guarantee a polynomial filtering time.

The second modification that we can safely allow is the reordering of variables. In the example above, assume the first  $n$  variables are  $X_1, \dots, X_n$  and the second  $n$  variables are  $Y_1, \dots, Y_n$ . Then, instead of building an automaton with  $2n + 1$  states that is linked to  $(X_1, \dots, X_n, Y_1, \dots, Y_n)$ , we could also build an automaton with just two states and link it to  $(X_1, Y_1, X_2, Y_2, \dots, X_n, Y_n)$  (see Figure 3). The same ideas can also be applied to  $\{a^n b^n c^n\}$  which is not even context-free but context-sensitive. The one thing that we really need to be careful about is that, when we want to exploit our earlier results on the combination of grammar constraints, we need to make sure that the ordering requirements specified in the respective theorems are met (see for instance Lemmas 5 and 6).

While these ideas can be exploited to model some required properties of solutions by means of grammar constraints, they make the theoretical analysis of which properties can or cannot be modeled by those constraints rather difficult. Where do the boundaries run between languages that are suited for regular or context-free grammar filtering? The introductory example, as uninteresting as it is from a filtering point of view, showed already that the theoretical tools that have been developed to assess that a certain language cannot be expressed by a grammar on a lower level in the Chomsky hierarchy fail. The

well-known pumping lemmas for regular and context-free grammars for instance rely on the fact that grammars be constant in size. As soon as we allow reordering and/or non-constant size grammars, they do not apply anymore.

To be more formal: what we really need to consider for propagation purposes is not an entire infinite set of words that form a language, but just a slice of words of a given length. I.e., given a language  $L$  what we need to consider is just  $L_{|n} := L \cap \Sigma^n$ . Since  $L_{|n}$  is a finite set, it really is a regular language. In that regard, our previous finding that  $\{a^n b^n\}$  for fixed  $n$  can be modeled as regular language is not surprising. The interesting aspect is that we can model  $\{a^n b^n\}$  by a regular grammar of size *linear in  $n$* , or even of *constant size* when reordering the variables appropriately.

**Definition 11 (Suitedness for Grammar Filtering).** *Given a language  $L$  over the alphabet  $\Sigma$ , we say that  $L$  is suited for regular (or context-free) grammar filtering iff there exist constants  $k, n \in \mathbb{N}$  such that there exists a permutation  $\sigma : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$  and a finite automaton  $A$  (or normal-form context-free grammar  $G$ ) such that both  $\sigma$  and  $A$  ( $G$ ) can be constructed in time  $O(n^k)$  with  $\sigma(L_{|n}) = \sigma(L \cap \Sigma^n) := \{w_{\sigma(1)} \dots w_{\sigma(n)} \mid w_1 \dots w_n \in L\} = L_A$  ( $\sigma(L_{|n}) = L_G$ ).*

*Remark 3.* Note that the previous definition implies that the size of the automaton (grammar) constructed is in  $O(n^k)$ . Note further that, if the given language is regular (context-free), then it is also suited for regular (context-free) grammar filtering.

Now, we have the terminology at hand to express that some properties cannot be modeled efficiently by regular or context-free grammar constraints. We start out by proving the following useful Lemma:

**Lemma 8.** *Denote with  $N = \{S_0, \dots, S_r\}$  a set of non-terminal symbols and  $G = (\Sigma, N, P, S_0)$  a context-free grammar in Chomsky-Normal-Form. Then, for every word  $w \in L_G$  of length  $n$ , there must exist  $t, u, v \in \Sigma^*$  and a non-terminal symbol  $S_i \in N$  such that  $S_0 \xrightarrow{*}_G t S_i v$ ,  $S_i \xrightarrow{*}_G u$ ,  $w = tuv$ , and  $n/4 \leq |u| \leq n/2$ .*

*Proof.* Since  $w \in L_G$ , there exists a derivation  $S_0 \xrightarrow{*}_G w$  in  $G$ . We set  $h_1 := 0$ . Assume the first production used in the derivation of  $w$  is  $S_{h_1} \rightarrow S_{k_1} S_{k_2}$  for some  $0 \leq k_1, k_2 \leq r$ . Then, there exist words  $u_1, u_2 \in \Sigma^*$  such that  $w = u_1 u_2$ ,  $S_{k_1} \xrightarrow{*}_G u_1$ , and  $S_{k_2} \xrightarrow{*}_G u_2$ . Now, either  $u_1$  or  $u_2$  fall into the length interval claimed by the lemma, or one of them is longer than  $n/2$ . In the first case, we are done, the respective non-terminal has the claimed properties. Otherwise, if  $|u_1| < |u_2|$  we set  $h_2 := k_2$ , else  $h_2 := k_1$ . Now, we repeat the argument that we just made for  $S_{h_1}$  for the non-terminal  $S_{h_2}$  that derives to the longer subsequence of  $w$ . At some point, we are bound to hit a production  $S_{h_m} \rightarrow S_{k_m} S_{k_{m+1}}$  where  $S_{h_m}$  still derives to a subsequence of length greater than  $n/2$ , but both  $S_{k_m}, S_{k_{m+1}}$  derive to subsequences that are at most  $n/2$  letters long. The longer of the two is bound to have length greater than  $n/4$ , and the respective non-terminal has the desired properties.  $\square$

Now consider the language

$$L_{\text{AllDiff}} := \{w \in \mathbb{N}^* \mid \forall 1 \leq k \leq |w| : \exists 1 \leq i \leq |w| : w_i = k\}.$$

Since the word problem for  $L_{\text{AllDiff}}$  can be decided in linear space,  $L_{\text{AllDiff}}$  is (at most) context-sensitive.

**Theorem 3.**  $L_{\text{AllDiff}}$  is not suited for context-free grammar filtering.

*Proof.* We observe that reordering the variables linked to the constraint has no effect on the language itself, i.e. we have that  $\sigma(L_{\text{AllDiff}|n}) = L_{\text{AllDiff}|n}$  for all permutations  $\sigma$ . Now assume that, for all  $n \in \mathbb{N}$ , we can construct a normal-form context-free grammar  $G = (\{1, \dots, n\}, \{S_0, \dots, S_r\}, P, S_0)$  that generates  $L_{\text{AllDiff}|n}$ . We will show that the minimum size for  $G$  is exponential in  $n$ . Due to Lemma 8, for every word  $w \in L_{\text{AllDiff}|n}$  there exist  $t, u, v \in \{1, \dots, n\}^*$  and a non-terminal symbol  $S_i$  such that  $S_0 \xrightarrow{*}_G tS_iv$ ,  $S_i \xrightarrow{*}_G u$ ,  $w = tuv$ , and  $n/4 \leq |u| \leq n/2$ . Now, let us count for how many words non-terminal  $S_i$  can be used in the derivation. Since from  $S_i$  we can derive  $u$ , all terminal symbols that are in  $u$  must appear in one block in any word that can use  $S_i$  for its derivation. This means that there can be at most  $(n - |u|)(n - |u|)!|u|! \leq \frac{3n}{4}(\frac{n}{2})^2$  such words. Consequently, since there exist  $n!$  many words in the language, the number of non-terminals is bounded from below by

$$r \geq \frac{n!}{\frac{3n}{4}(\frac{n}{2})^2} = \frac{4(n-1)!}{3(\frac{n}{2})^2} \approx \frac{4\sqrt{2}}{3\sqrt{\pi}} \frac{2^n}{n^{3/2}} \in \omega(1.5^n).$$

□

Now, the interesting question arises whether there exist languages at all that are fit for context-free, but not for regular grammar filtering? If this wasn't the case, then the algorithm developed in Section 3 would be utterly useless. What makes the analysis of suitedness so complicated is the fact that the modeler has the freedom to change the ordering of variables that are linked to the grammar constraint — which essentially allows him or her to change the language almost ad gusto. We have seen an example for this earlier where we proposed that  $a^n b^n$  could be modeled as  $(ab)^n$ .

**Theorem 4.** The set of languages that are suited for context-free grammar filtering is a strict superset of the set of languages that are suited for regular grammar filtering.

*Proof.* Consider the language  $L = \{wv^R \# vv^R \mid v, w \in \{0, 1\}^*\} \subseteq \{0, 1, \#\}^*$  (where  $x^R$  denotes the reverse of a word  $x$ ). Obviously,  $L$  is context-free with the grammar  $(\{0, 1, \#\}, \{S_0, S_1\}, \{S_0 \rightarrow S_1 \# S_1, S_1 \rightarrow 0S_10, S_1 \rightarrow 1S_11, S_1 \rightarrow \varepsilon\}, S_0)$ . Consequently  $L$  is suited for context-free grammar filtering.

Note that, when the position  $2k+1$  of the sole occurrence of the letter  $\#$  is fixed, for every position  $i$  containing a letter 0 or 1, there exists a *partner position*  $p^k(i)$  so that both corresponding variables are forced to take the same value. Crucial to our following analysis is the fact that, in every word  $x \in L$  of length  $|x| = n = 2l + 1$ , every even (odd) position is linked in this way exactly with every odd (even) position for some

placement of #. Formally, we have that  $\{p^k(i) \mid 0 \leq k \leq l\} = \{1, 3, 5, \dots, n\}$  ( $\{p^k(i) \mid 0 \leq k \leq l\} = \{2, 4, 6, \dots, 2l\}$ ) when  $i$  is even (odd).

Now, assume that, for every odd  $n = 2l + 1$ , there exists a finite automaton that accepts some reordering of  $L \cap \{0, 1, \#\}^n$  under variable permutation  $\sigma : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ . For a given position  $2k + 1$  of # (in the original ordering), by  $dist_\sigma^k := \sum_{i=1,3,\dots,2l+1} |\sigma(i) - \sigma(p^k(i))|$  we denote the total distance of the pairs after the reordering through  $\sigma$ . Then, the average total distance after reordering through  $\sigma$  is

$$\begin{aligned} \frac{1}{l+1} \sum_{0 \leq k \leq l} dist_\sigma^k &= \frac{1}{l+1} \sum_{0 \leq k \leq l} \sum_{i=1,3,\dots,2l+1} |\sigma(i) - \sigma(p^k(i))| \\ &= \frac{1}{l+1} \sum_{i=1,3,\dots,2l+1} \sum_{0 \leq k \leq l} |\sigma(i) - \sigma(p^k(i))|. \end{aligned}$$

Now, since we know that every odd  $i$  has  $l$  even partners, even for an ordering  $\sigma$  that places all partner positions in the immediate neighborhood of  $i$ , we have that

$$\sum_{0 \leq k \leq l} |\sigma(i) - \sigma(p^k(i))| \geq 2 \sum_{s=1,\dots,\lfloor l/2 \rfloor} s = (\lfloor l/2 \rfloor + 1) \lfloor l/2 \rfloor.$$

Thus, for sufficiently large  $l$ , the average total distance under  $\sigma$  is

$$\begin{aligned} \frac{1}{l+1} \sum_{0 \leq k \leq l} dist_\sigma^k &\geq \frac{1}{l+1} \sum_{i=1,3,\dots,2l+1} (\lfloor l/2 \rfloor + 1) \lfloor l/2 \rfloor \\ &\geq \frac{l}{l+1} (\lfloor l/2 \rfloor + 1) \lfloor l/2 \rfloor \\ &\geq l^2/8. \end{aligned}$$

Consequently, for any given reordering  $\sigma$ , there must exist a position  $2k + 1$  for the letter # such that the total distance of all pairs of linked positions is at least the average, which in turn is greater or equal to  $l^2/8$ . Therefore, since the maximum distance is  $2l$ , there must exist at least  $l/16$  pairs that are at least  $l/8$  positions apart after reordering through  $\sigma$ . It follows that there exists an  $1 \leq r \leq n$  such that there are at least  $l/128$  positions  $i \leq r$  such that  $p^k(i) > r$ . Consequently, after reading  $r$  inputs, the finite automaton that accepts the reordering of  $L \cap \{0, 1, \#\}^n$  needs to be able to reach at least  $2^{l/128}$  different states. It is therefore not polynomial in size. It follows:  $L$  is not suited for regular grammar filtering.  $\square$

As a final remark, it is interesting that  $\{ww^R\#vv^R\}_{|2l+1} = \bigcup_{k=1}^l \{ww^R\#vv^R \mid |w| = k, |v| = l - k\}$ , and  $\{ww^R\#vv^R \mid |w| = k, |v| = l - k\}$  is actually suited for regular grammar filtering when each of the sets is reordered appropriately. Now, Lemma 5 cannot be applied, since the different constraints use different reorderings of the variables. However, we could apply standard disjunctive constraint filtering. Ultimately, context-free grammar filtering only appears to become unavoidable for unrestricted concatenations of non-regular grammars such as  $\{w_1w_1^R\#w_2w_2^R\#\dots w_kw_k^R\}$ , or our example grammar that generates correctly bracketed expressions, or the language of syntactically correct mathematical expressions, to name just a few.

## 6 Conclusions

We investigated the idea of basing constraints on formal languages. Particularly, we devised an efficient arc-consistency algorithm for grammar constraints based on context-free grammars in Chomsky Normal Form. We studied logic combinations of grammar constraints and showed where the boundaries run between regular, context-free, and context-sensitive grammar constraints when allowing non-constant grammars and reorderings of variables. Our hope is that grammar constraints can serve as powerful modeling entities for constraint programming in the future, and that our theory can help to better understand and tackle the computational problems that arise in the context of grammar constraint filtering.

## References

1. F. Bacchus and T. Walsh. Propagating Logical Combinations of Constraints. *International Joint Conference on Artificial Intelligence*, pp. 35–40, 2005.
2. N. Beldiceanu, M. Carlsson, T. Petit. Deriving Filtering Algorithms from Constraint Checkers. *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming*, LNCS 3258, 2004.
3. C. Bessière and J-C. Régin. Local consistency on conjunctions of constraints. *Workshop on Non-Binary Constraints*, 1998.
4. S. Bourdais, P. Galinier, G. Pesant. HIBISCUS: A Constraint Programming Application to Staff Scheduling in Health Care. *International Conference on Principles and Practice of Constraint Programming*, LNCS 2833:153–167, 2003.
5. N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory* 2:113–124, 1956.
6. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*, Addison Wesley, 1979.
7. O. Lhomme. Arc-consistency Filtering Algorithms for Logical Combinations of Constraints. *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, 2004.
8. L-M. Rousseau, G. Pesant, W-J. van Hoeve. On Global Warming: Flow Based Soft Global Constraint. *Informs*, p. 27, 2005.
9. G. Pesant. A Regular Language Membership Constraint for Finite Sequences of Variables. *International Conference on Principles and Practice of Constraint Programming (CP)*, LNCS 3258:482–495, 2004.
10. G. Pesant. A Regular Language Membership Constraint for Sequences of Variables *International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, pp. 110–119, 2003.
11. M. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 113–124, 2001.