Linking Prefixes and Suffixes for Constraints Encoded Using Automata with Accumulators

Nicolas Beldiceanu[1], Mats Carlsson[2], Pierre Flener[3], María Andreína Francisco Rodríguez[3], and Justin Pearson[3]

[1] TASC Team (CNRS/INRIA), Mines de Nantes, 44307 Nantes, France
Nicolas.Beldiceanu@mines-nantes.fr
[2] SICS, P.O. Box 1263, 164 29 Kista, Sweden
Mats.Carlsson@sics.se
[3] Uppsala University, Dept. of Information Technology, 751 05 Uppsala, Sweden
{FirstName.LastName}@it.uu.se

Abstract. Consider a constraint on a sequence of variables functionally determining a result variable that is unchanged under reversal of the sequence. Most such constraints have a compact encoding via an automaton augmented with accumulators, but it is unknown how to maintain domain consistency efficiently for most of them. Using such an automaton for such a constraint, we derive an implied constraint between the result variables for a sequence, a prefix thereof, and the corresponding suffix. We show the usefulness of this implied constraint in constraint solving, both by local search and by propagation-based systematic search.

1 Introduction

Deterministic finite automata augmented with accumulators [4] were motivated by the need to encode a constraint C on a sequence X of variables using an automaton whose size does not depend on the size of X: accumulators are initialised at the start state and evolve through the transitions; upon acceptance, the accumulators are often mapped to a result variable R of C. The *Global Constraint Catalogue* [1] gives very compact automata with accumulators for 56 constraints (and some will be given shortly), but it is unknown how to maintain domain consistency efficiently for most of them, so implied constraints can help improve the propagation. In this paper, we consider such constraints $C(X, R)$ where R is the same for both X and its reverse X^{rev}; this covers 45 of those 56 constraints. Such constraints have proved very useful, for instance in production sequencing and staff rostering. Given a partition of X into a prefix P and a suffix T, we derive an implied constraint, shown to exist and be unique, between R, $\overrightarrow{R}$, and $\overleftarrow{R}$ when $C(X, R)$, $C(P, \overrightarrow{R})$, and $C(T^{\text{rev}}, \overleftarrow{R})$ hold. We show the usefulness of this implied constraint in constraint solving, whether by local search or by propagation-based systematic search.

We now define the Group constraint, to which we will be referring heavily throughout. We then give a motivating example, introducing our terminology and serving as running example throughout the rest of the paper.

B. O'Sullivan (Ed.): CP 2014, LNCS 8656, pp. 142–157, 2014.

Definition 1 ([1]). *In a sequence, a* group *is a maximal contiguous subsequence with values from a given set. The constraint* $\textsc{Group}(X, W, G, V, H, L)$ *holds if there are* G *groups of a total of* V *values from the given set* W *in the possibly empty sequence* X *of variables, the highest and lowest group sizes being* H *and* L *respectively, with* $H = 0 = L$ *if* $G = 0$. (*W.l.o.g., we omit two parameters.*)

The instance $\textsc{Group}([\mathrm{d}, \mathrm{a}, \mathrm{c}, \mathrm{b}, \mathrm{e}, \mathrm{a}, \mathrm{b}], \{\mathrm{a}, \mathrm{e}\}, 2, 3, 2, 1)$ holds since there are $G = 2$ groups of a total of $V = 3$ occurrences of 'a' and 'e' in the sequence $[\mathrm{d}, \mathrm{a}, \mathrm{c}, \mathrm{b}, \mathrm{e}, \mathrm{a}, \mathrm{b}]$, namely the groups $[\mathrm{a}]$ and $[\mathrm{e}, \mathrm{a}]$, the highest group size being $H = 2$ and the lowest group size being $L = 1$. Group has no known propagator. Its decomposition [1] into the conjunction $\textsc{GroupG}(X, W, G) \wedge \textsc{GroupV}(X, W, V) \wedge \textsc{GroupH}(X, W, H) \wedge \textsc{GroupL}(X, W, L)$ can be encoded using four Automaton constraints on automata with accumulators [4]: see Figure 1 and Section 2 for details. These constraints are very useful, for instance in staff rostering, where multiple counting constraints on the same sequence (the shift assignments of an employee over a planning horizon) are quite frequent.

Example 1. Consider the instance $\textsc{Group}([X_1, X_2, X_3], \{\mathrm{a}\}, G, V, H, L)$, where $\mathrm{dom}(X_i) = \{\mathrm{a}, \mathrm{b}\}$, $\mathrm{dom}(G) = \{0, 1, 2\} = \mathrm{dom}(V)$, $\mathrm{dom}(H) = \{2, 3\} = \mathrm{dom}(L)$, with $\mathrm{dom}(\alpha)$ denoting the current domain of variable α. Using the encoding mentioned above, *no* domain pruning is achieved on this instance.

However, it *is* possible to achieve some propagation on this instance, whose solutions are $\textsc{Group}([\mathrm{a}, \mathrm{a}, \mathrm{b}], \{\mathrm{a}\}, 1, 2, 2, 2)$ and $\textsc{Group}([\mathrm{b}, \mathrm{a}, \mathrm{a}], \{\mathrm{a}\}, 1, 2, 2, 2)$: among others, there cannot be $G = 2$ groups (as that would require a sequence of at least five elements, as groups must have at least two elements), the groups cannot have a total of $V = 0$ values (as the largest group must have at least two elements), and X_2 cannot be 'b' (as X_2 must participate in a group of 'a').

We now discuss three schemes for achieving more propagation than with just the encoding by four Automaton constraints.

Scheme 1. The Group constraint has so-called graph invariants [5], which can be seen as implied constraints. For instance, consider the following bounds on V:

$$\max(G - 1, 0) \cdot L + H \leq V \leq \max(G - 1, 0) \cdot H + L \tag{1}$$

Intuitively, the lower bound corresponds to having one group of H elements while all the other groups are as small as possible, that is, they have L elements. The upper bound is justified in a similar way. Consider again the instance above: if the implied constraint (1) is added to the four Automaton constraints, then 2 is pruned from $\mathrm{dom}(G)$, but all the other domains remain unchanged. There are 90 graph invariants in [5] for the Group constraint: the pruning upon adding *all* the corresponding implied constraints is evaluated in Section 5.

Scheme 2. Note that $\textsc{Group}([X_1, \ldots, X_n], W, G, V, H, L)$ holds if and only if $\textsc{Group}([X_n, \ldots, X_1], W, G, V, H, L)$ holds with the same set and the same integer variables for the reverse sequence: in Section 3.1, we will say that Group is its own *reverse constraint*. Let us focus on the variable V, representing the total number of group values. If we split a sequence $[X_1, \ldots, X_n]$ with $n \geq 2$ elements into a non-empty prefix $[X_1, \ldots, X_i]$ and a non-empty suffix $[X_{i+1}, \ldots, X_n]$, with

$1 \leq i < n$, then observe that the numbers V, $\overrightarrow{V}$, and $\overleftarrow{V}$ of group values respectively in the entire sequence, the prefix, and the reverse suffix are related by the constraint $V = \overrightarrow{V} + \overleftarrow{V}$: in Section 3.2, we will say that this constraint implied by the conjunction of GROUPV($[X_1, \ldots, X_n], W, V$), GROUPV($[X_1, \ldots, X_i], W, \overrightarrow{V}$), and GROUPV($[X_n, \ldots, X_{i+1}], W, \overleftarrow{V}$) is a *glue constraint*. Glue constraints for all the integer variables of GROUP are given in Figure 2 and will be explained in Section 3.2. While GROUPV($[X_n, \ldots, X_{i+1}], W, \overleftarrow{V}$) could be replaced above by GROUPV($[X_{i+1}, \ldots, X_n], W, \overleftarrow{V}$), this will be seen to be impossible in general with our approach, where the third implying constraint must be on the *reverse suffix*, not on the suffix itself. Now consider again the instance above: if we add the glue constraints in Figure 2 for *every* possible split of the sequence (with $1 \leq i < n$), but not the implied constraint (1), then 'b' is pruned from dom(X_2) and 0 is pruned from dom(V), but all the other domains remain unchanged.

Note that the extra pruning achieved by Scheme 1 is incomparable with that achieved by Scheme 2.

Scheme 3. The idea of sequence splitting (underlying the glue constraints) can be applied also to the implied constraints stemming from graph invariants: for instance, instead of adding (1) on the integer variables for the *entire* sequence, we can add (1) on the integer variables for the prefix and reverse suffix of *every* possible split of the sequence. On the instance under consideration, applying Scheme 1+2+3 achieves the same pruning as applying Scheme 1+2, but, in general, more pruning is possible, as we will show in Section 5. In Section 4, we formalise Scheme 3. □

Multiple constraints on a sequence can originate from sources other than the decomposition of a constraint, unlike the previous example. For instance, a conjunction of about 20 constraints on the same sequence (of energy produced by a plant every half an hour for two consecutive days) is the pattern learned in the context of the EDF model seeker [7]. Also, in staff rostering, one has a matrix indexed in the rows by the employees and in the columns by the days of a planning horizon: each matrix cell is to be assigned an identifier (or a special off-duty value) giving the shift assigned to the corresponding employee on the corresponding day. The constraints on the columns are usually cardinality constraints, stemming from a performance contract, and there are often multiple constraints on each row, stemming from employee preferences as well as labour union and legislative restrictions. In [2], we have addressed the lack of interaction between such row and column constraints; in this paper, we address the lack of interaction between the constraints on a given row.

The contributions and the organisation of the rest of this paper are as follows, after first recalling (and slightly extending) in Section 2 the concept of automaton with accumulators [4], which can be used for compactly encoding a constraint on a sequence of variables using the AUTOMATON constraint [4]:

– In Section 3, our main result, after introducing the notion of reverse of a constraint, we define a glue constraint as an implied constraint, shown to exist and be unique, linking the result variable R under a constraint $C(X, R)$

on a sequence X of variables to the result under C on a prefix and the result under the reverse of C on the corresponding reverse suffix of X, where both C and its reverse are encoded using automata with accumulators. We show how to derive glue constraints automatically for a useful class of such constraints.
- In Section 4, we formalise Scheme 3 of Example 1 so as to cover any conjunction of constraints $C_j(X, R_j)$ on the same sequence X, whose results R_j do not vary independently but are linked by an implied constraint.
- In Section 5, we evaluate the effectiveness and efficiency of the three schemes.
- In Section 6, we use the glue constraint to compute, in constant time, the violation cost of C when probing an assignment move in local search.

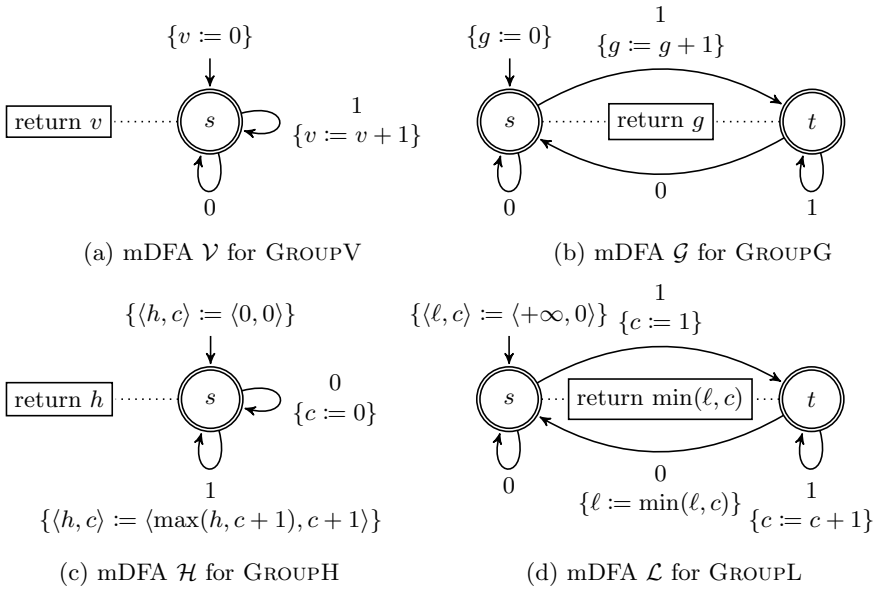
We conclude and discuss related and future work in Section 7.

2 Background: Automata with Accumulators

Recall that a *deterministic finite automaton* (DFA) is a tuple $\langle Q, \Sigma, \delta, \phi, A\rangle$, where Q is the set of *states*, Σ the *alphabet*, $\delta\colon Q \times \Sigma \to Q$ the *transition function*, $\phi \in Q$ the *start state*, and $A \subseteq Q$ the set of *accepting states*. When $\delta(q, \sigma) = q'$, there is a transition from state q to state q' upon reading alphabet symbol σ in the word given to the DFA. Let Σ^* denote the infinite set of words built from Σ, including the empty word, denoted ϵ. The *extended transition function* $\widehat{\delta}\colon Q \times \Sigma^* \to Q$ for words (instead of symbols) is recursively defined by $\widehat{\delta}(q, \epsilon) = q$ and $\widehat{\delta}(q, w\sigma) = \delta(\widehat{\delta}(q, w), \sigma)$ for a word w and symbol σ. An example will be given shortly, but first we augment DFAs with a memory, in the spirit of [4], in order to encode more compactly many constraints.

We here define a *memory-DFA* (mDFA) with a memory of $k \geq 0$ accumulators as a tuple $\langle Q, \Sigma, \delta, \phi, I, A, \alpha\rangle$, where Q, Σ, ϕ, and A are as in a DFA, while the transition function δ has signature $(Q \times \mathbb{Z}^k) \times \Sigma \to Q \times \mathbb{Z}^k$, and similarly for its extended version $\widehat{\delta}$. Further, I is the k-tuple of initial values of the variables in the memory. Finally, $\alpha\colon A \times \mathbb{Z}^k \to \mathbb{Z}$ is called the *acceptance function* and transforms the memory of an accepting state into an integer. Given a word w, the mDFA returns $\alpha(\widehat{\delta}(\langle \phi, I\rangle, w))$ if w is accepted. Note that δ, $\widehat{\delta}$, and α are total functions. This definition can be generalised, but suffices for the purpose of the examples in this paper.

Example 2. Consider the mDFA $\mathcal{H}$ depicted in Figure 1c. It returns the highest size of all groups of ones within a given word of zeros and ones. It uses two accumulators: at any moment, c stores the size of the current group, while h stores the highest size of all the groups seen so far. The state set Q is $\{s\}$. The alphabet Σ is $\{0, 1\}$. The start state ϕ is s, and is indicated by an arrow coming from nowhere, annotated within braces by the initialisation to zero of h and c, hence $I = \langle 0, 0\rangle$. A transition $\delta(\langle q, \langle h, c\rangle\rangle, \sigma) = \langle q', \langle h', c'\rangle\rangle$, where h' and c' are functional expressions in terms of h and c, is depicted by an arrow going from state q to state q', annotated by symbol σ and, within braces, the memory update $\langle h, c\rangle \coloneqq \langle h', c'\rangle$. For instance, the lower self-loop depicts that $\delta(\langle s, \langle h, c\rangle\rangle, 1) = \langle s, \langle \max(h, c + 1), c + 1\rangle\rangle$ for all h and c. If a memory update



(a) mDFA $\mathcal{V}$ for GROUPV (b) mDFA $\mathcal{G}$ for GROUPG

(c) mDFA $\mathcal{H}$ for GROUPH (d) mDFA $\mathcal{L}$ for GROUPL

Fig. 1. Memory-DFAs for the constraints of the decomposition of GROUP

corresponds to the identity function, then we omit it; for instance, the right self-loop has the memory update $c := 0$ as an abbreviation for $\langle h, c\rangle := \langle h, 0\rangle$. All states are accepting, hence $A = Q$, an accepting state being marked by a double circle. The acceptance function α transforms a memory $\langle h, c\rangle$ at state s into h, and is depicted by a box linked to s by a dotted line. □

Automata are useful for encoding a constraint on a sequence X of variables: the REGULAR$(\mathcal{D}, X)$ constraint [14] takes a constraint encoded by a DFA $\mathcal{D}$ and holds if and only if the word represented by X is accepted by $\mathcal{D}$; similarly, the AUTOMATON$(\mathcal{M}, X)$ constraint [4] takes a constraint encoded by an mDFA $\mathcal{M}$ and specialises to REGULAR for a memory of $k = 0$ accumulators. We define AUTOMATON$(\mathcal{M}, X, R)$ for an mDFA with a memory of $k > 0$ accumulators: this constraint is equivalent to the conjunction of AUTOMATON$(\mathcal{M}, X)$ and the *acceptance constraint* that variable R be equal to the integer returned by $\mathcal{M}$, that is $R = \alpha(\widehat{\delta}(\langle \phi, I\rangle, X))$. Note that R functionally depends on X, as α and $\widehat{\delta}$ are total functions. AUTOMATON does not maintain domain consistency if $k > 0$.

Further, a constraint C on a sequence X of variables can sometimes be encoded with the help of an (m)DFA that operates not on X, but on a sequence S of *signature variables*, each depending via a *signature constraint* [4] under a total function on a sliding window of a consecutive variables within X. The constant $a \geq 1$ is called the *arity* of the signature constraints, and is linked to the lengths n of X and m of S by $m = n+1-a$. The arity gives a precondition on C, namely $n \geq a-1$, as the signature constraints fail otherwise. The sliding windows within X for two consecutive signature variables overlap by $a - 1$ variables.

Example 3. Consider the GROUPH$([X_1, \ldots, X_n], W, H)$ constraint with $n \geq 0$. We constrain a sequence $[S_1, \ldots, S_m]$ of $m = n$ signature 0/1 variables S_i with the signature constraints $(X_i \in W) \Leftrightarrow (S_i = 1)$ for all $1 \leq i \leq n$: we have $a = 1$ since each signature constraint is on a single X_i. Using the mDFA $\mathcal{H}$ of Example 2 and Figure 1c, we encode GROUPH$([X_1, \ldots, X_n], W, H)$ by AUTOMATON$(\mathcal{H}, [S_1, \ldots, S_n], H)$ and these signature constraints. For the constraint instance GROUPH$([\mathrm{d}, \mathrm{a}, \mathrm{c}, \mathrm{b}, \mathrm{e}, \mathrm{a}, \mathrm{b}], \{\mathrm{a}, \mathrm{e}\}, 2)$, the mDFA $\mathcal{H}$ indeed returns $h = 2$ on the sequence $S = [0, 1, 0, 0, 1, 1, 0]$ of signature values. Similarly, the other constraints of the given decomposition of the GROUP constraint can be encoded with the help of the other three mDFAs in Figure 1. □

In the absence of signature variables and constraints, we consider $S = X$ to be a signature constraint, as this simplifies the discussion.

3 Reverse Constraints and Glue Constraints

After defining the concept of reverse of a constraint in Section 3.1, we define in Section 3.2 a glue constraint as an implied constraint for a constraint with a reverse constraint, both encoded using an automaton with accumulators, and we show that the glue constraint is unique and always exists. Finally, we show in Section 3.3 how to derive, mechanically and efficiently, the glue constraint for a useful large class of constraints.

3.1 The Reverse of a Constraint

A constraint $C(V_1, \ldots, V_n)$ is a *total-function constraint* [3] if its variables V_i can be partitioned into two non-empty sets, D and R, such that for any assignment to the variables of D there is a *unique* assignment to the variables of R that satisfies C. For example, the constraints GROUP(X, W, G, V, H, L), GROUPG(X, W, G), GROUPV(X, W, V), GROUPH(X, W, H), GROUPL(X, W, L) are total-function constraints, where X and W uniquely determine G, V, H, and L. Also, signature constraints (see Section 2) are total-function constraints.

We write a constraint $C(D, R)$ as $C(D \rightarrow R)$ when the variables D functionally determine the variables R. We denote the reverse of a word or variable sequence w by w^{rev}. We now define our first core concept.

Definition 2. *The* reverse *of a total-function constraint $C(D \rightarrow R)$, where D is a sequence of variables, is a total-function constraint $C'(D' \rightarrow R')$, where D' is a sequence of variables, such that, for any sequence X of variables, both X and its reverse functionally determine the same result variables Y under C and C' respectively, that is both $C(X \rightarrow Y)$ and $C'(X^{rev} \rightarrow Y)$ hold.*

Example 4. The constraints GROUP, GROUPG, GROUPV, GROUPH, GROUPL are their own reverses. The constraint LENGTHFIRSTSEQUENCE(X, L), which holds if L is the size of the *first* group of identical values within the sequence X of variables [1], does not have itself as reverse, but LENGTHLASTSEQUENCE(X, L), which holds if L is the size of the *last* group of identical values within X [1]. □

If a total-function constraint $C(X, R)$ is encoded by AUTOMATON$(\mathcal{M}, S, R)$ and signature constraints linking the sequences X and S of variables, using a memory-DFA $\mathcal{M} = \langle Q, \Sigma, \delta, \phi, I, A, \alpha \rangle$ with k accumulators, then $\mathcal{M}$ necessarily only has accepting states, that is $A = Q$, because the total function α is only defined on accepting states, so that the value returned by $\mathcal{M}$ on the word represented by S might be undefined if there were some non-accepting states.

If a total-function constraint C has a memory-DFA $\mathcal{M}$, then a memory-DFA $\mathcal{M}'$ for its reverse constraint C' can in some cases be derived automatically from $\mathcal{M}$. Indeed, $\mathcal{M}'$ is then the *reverse* of $\mathcal{M}$, in the sense that $\mathcal{M}'$ recognises the reverse of every word recognised by $\mathcal{M}$, and both return the same value. For instance, if $\mathcal{M}$ has a single accumulator (hence $k = 1$), which is initialised to zero (hence $I = 0$) at the start state ϕ and increased by a non-negative quantity at each transition, and if the acceptance function α returns that accumulator increased by a non-negative quantity, then $\mathcal{M}$ is a *weighted DFA* [12] over the tropical semiring over the integers, and the algorithms implemented in [13] can be used for reversal. Among the 45 of 56 constraints of the catalogue covered by this paper, there are 16 with weighted DFAs, such as GROUPG and GROUPV, but not GROUPH and GROUPL, whose accumulator updates use the max and min operators. (We revisit this useful class of constraints in Section 3.3.)

3.2 Glue Constraints

We need the AUTOMATON constraint to be implemented, in extension to how it is done in [4], by a decomposition that introduces variables representing not only the accumulators but also the state of the argument mDFA $\mathcal{M} = \langle Q, \Sigma, \delta, \phi, I, A, \alpha \rangle$ after reading each symbol of an argument sequence S of variables. Upon reading the entire S, we then have that $\widehat{\delta}(\langle \phi, I \rangle, S)$ is a tuple $\langle q, V \rangle$, where variable q represents the reached state of $\mathcal{M}$, and V is an array of k variables representing the k obtained accumulator values of $\mathcal{M}$.

We now show that the result of $\mathcal{M}$ on a word w can be computed from only these state and accumulator variables, as they encode all information on w. We will then exploit this insight by constructing a function g, which is unique and correctly computes the result of $\mathcal{M}$ on a word $w = pt$ by combining the state and accumulator variables reached by its prefix p and the reverse of its suffix t.

Theorem 1. *Consider an mDFA $\mathcal{M} = \langle Q, \Sigma, \delta, \phi, I, A, \alpha \rangle$ and its reverse $\mathcal{M}' = \langle Q', \Sigma, \delta', \phi', I', A', \alpha' \rangle$, over the same alphabet Σ. Consider four words p_1, p_2, t_1, and t_2. Assume p_1 and p_2 reach the same tuple in $\mathcal{M}$, that is $\widehat{\delta}(\langle \phi, I \rangle, p_1) = \langle q_p, V_p \rangle = \widehat{\delta}(\langle \phi, I \rangle, p_2)$. Assume the reverses of t_1 and t_2 reach the same tuple in $\mathcal{M}'$, that is $\widehat{\delta'}(\langle \phi', I' \rangle, {t_1}^{rev}) = \langle q'_t, V'_t \rangle = \widehat{\delta'}(\langle \phi', I' \rangle, {t_2}^{rev})$. We then have $\alpha(\widehat{\delta}(\langle \phi, I \rangle, p_1 t_1)) = \alpha(\widehat{\delta}(\langle \phi, I \rangle, p_2 t_2))$, so that the result on a word is independent of its prefixes p_i and corresponding suffixes t_i transiting through the same tuples.*

Proof. We have $\alpha(\widehat{\delta}(\langle \phi, I \rangle, p_1 t_1)) = \alpha(\widehat{\delta}(\langle q_p, V_p \rangle, t_1)) = \alpha(\widehat{\delta}(\langle \phi, I \rangle, p_2 t_1))$. Similarly, $\alpha'(\widehat{\delta'}(\langle \phi', I' \rangle, (p_2 t_1)^{\text{rev}})) = \alpha'(\widehat{\delta'}(\langle q'_t, V'_t \rangle, {p_2}^{\text{rev}})) = \alpha'(\widehat{\delta'}(\langle \phi', I' \rangle, (p_2 t_2)^{\text{rev}}))$.

As $\mathcal{M}'$ is the reverse of $\mathcal{M}$, we have $\alpha(\widehat{\delta}(\langle\phi, I\rangle, p_2t_1)) = \alpha'(\widehat{\delta'}(\langle\phi', I'\rangle, (p_2t_1)^{\text{rev}}))$ $= \alpha'(\widehat{\delta'}(\langle\phi', I'\rangle, (p_2t_2)^{\text{rev}})) = \alpha(\widehat{\delta}(\langle\phi, I\rangle, p_2t_2))$, and hence $\alpha(\widehat{\delta}(\langle\phi, I\rangle, p_1t_1)) = \alpha(\widehat{\delta}(\langle\phi, I\rangle, p_2t_2))$. □

Hence there exists a unique total function g that takes two tuples $\langle q, V\rangle$ and $\langle q', V'\rangle$ of $\mathcal{M}$ and $\mathcal{M}'$ respectively, such that

$$g(\langle q, V\rangle, \langle q', V'\rangle) = \alpha(\widehat{\delta}(\langle\phi, I\rangle, pt)) \tag{2}$$

for *any* prefix p that reaches the tuple $\langle q, V\rangle$ in $\mathcal{M}$ and *any* suffix t such that t^{rev} reaches the tuple $\langle q', V'\rangle$ in $\mathcal{M}'$. It follows from Theorem 1 that this function is well-defined, as it is independent of the prefix p and suffix t picked.

Consider a total-function constraint $C(X \to R)$, for which an mDFA $\mathcal{M}$ reads a sequence S of signature variables channelled with the sequence X by signature constraints, such that the variable R must be the result returned by $\mathcal{M}$ on S. Hence $C(X, R)$ can be encoded by $\text{Automaton}(\mathcal{M}, S, R)$ and the signature constraints. Consider a split of S into the concatenation of a possibly empty prefix P and a possibly empty suffix T, that is $S = PT$, with $R = \alpha(\widehat{\delta}(\langle\phi, I\rangle, PT))$. We now define our second core concept:

Definition 3. *Suppose, in addition to* $\text{Automaton}(\mathcal{M}, PT, R)$*, we post the constraint* $\text{Automaton}(\mathcal{M}, P, \overrightarrow{R})$ *on the prefix* P*, as well as the constraint* $\text{Automaton}(\mathcal{M}', T^{rev}, \overleftarrow{R})$ *on the reverse suffix* T^{rev}*, where* $\mathcal{M}'$ *is the reverse of* $\mathcal{M}$*. Let* $\widehat{\delta}(\langle\phi, I\rangle, P) = \langle \overrightarrow{q}, \overrightarrow{V}\rangle$ *and* $\widehat{\delta'}(\langle\phi', I'\rangle, T^{rev}) = \langle \overleftarrow{q}, \overleftarrow{V}\rangle$*. The function* g *of (2) gives rise to an implied constraint, called the* glue constraint*:*

$$R = g(\langle \overrightarrow{q}, \overrightarrow{V}\rangle, \langle \overleftarrow{q}, \overleftarrow{V}\rangle) \tag{3}$$

Example 5. The glue constraints for all four numeric variables of the Group constraint are given in Figure 2. They are organised as matrices, called *glue matrices*. The glue matrix in Figure 2d represents the following glue constraint:

$$L = \begin{cases} \min(\overrightarrow{\ell}, \overrightarrow{c} + \overleftarrow{c}, \overleftarrow{\ell}) & \text{if } \overrightarrow{q} = s \wedge \overleftarrow{q} = s \\ \min(\overrightarrow{\ell}, \overleftarrow{c}, \overleftarrow{\ell}) & \text{if } \overrightarrow{q} = s \wedge \overleftarrow{q} = t \\ \min(\overrightarrow{\ell}, \overrightarrow{c}, \overleftarrow{\ell}) & \text{if } \overrightarrow{q} = t \wedge \overleftarrow{q} = s \\ \min(\overrightarrow{\ell}, \overrightarrow{c} + \overleftarrow{c}, \overleftarrow{\ell}) & \text{if } \overrightarrow{q} = t \wedge \overleftarrow{q} = t \end{cases}$$

Figure 3 illustrates its use on an instance of GroupL. □

We post the three Automaton constraints of Definition 3 and their implied glue constraint (3) for every split of the sequence S of signature variables into a possibly empty prefix P and a possibly empty suffix T.

We have no general calculus (yet) for deriving the glue function g mechanically from an mDFA, but we now illustrate the typical reasoning on an example. In Section 3.3, we then show how to derive g mechanically and efficiently for the useful class of constraints mentioned at the end of Section 3.1.

	s
s	$V = \overrightarrow{v} + \overleftarrow{v}$

(a) Glue constraint for GROUPV

	s	t
s	$G = \overrightarrow{g} + \overleftarrow{g}$	$G = \overrightarrow{g} + \overleftarrow{g}$
t	$G = \overrightarrow{g} + \overleftarrow{g}$	$G = \overrightarrow{g} - 1 + \overleftarrow{g}$

(b) Glue constraint for GROUPG

	s
s	$H = \max(\overrightarrow{h}, \overrightarrow{c} + \overleftarrow{c}, \overleftarrow{h})$

(c) Glue constraint for GROUPH

	s	t
s	$L = \min(\overrightarrow{\ell}, \overrightarrow{c} + \overleftarrow{c}, \overleftarrow{\ell})$	$L = \min(\overrightarrow{\ell}, \overleftarrow{c}, \overleftarrow{\ell})$
t	$L = \min(\overrightarrow{\ell}, \overrightarrow{c}, \overleftarrow{\ell})$	$L = \min(\overrightarrow{\ell}, \overrightarrow{c} + \overleftarrow{c}, \overleftarrow{\ell})$

(d) Glue constraint for GROUPL

Fig. 2. Glue constraints for the constraints of the decomposition of GROUP: a row index refers to the state of the corresponding mDFA in Figure 1 reached by the prefix, and a column index refers to the state reached by the corresponding reverse suffix; recall that each of the four mDFAs is its own reverse.

Example 6. The mDFA $\mathcal{H}$ in Figure 1c has a single state, whose semantics is as follows, given current accumulator values c and h: a word matching the regular expression $\pi 1^c$ has been read so far, where $\pi = \varepsilon \mid \Sigma^* 0$, with $c \geq 0$ (let $1^0 = \varepsilon$) and $\max(\theta(h, \pi), c) = h$, where $\theta(a, w)$ denotes the value of accumulator a upon reading word w. Observe that $\theta(h, \pi)$ is *not* accessible any more in any accumulator after reading $\pi 1^c$ when $c > 0$. Note that $\theta(h, w) = \theta(h, w^{\text{rev}})$ for any word w, because $\mathcal{H}$ and GROUPH are their own reverses.

When GROUPH(X, W, H) is encoded by AUTOMATON$(\mathcal{H}, S, H)$ and the signature constraints of Example 3 between X and S, let us split S into the concatenation of a possibly empty prefix P and a possibly empty suffix T.

Upon feeding the prefix P to $\mathcal{H}$, we end up with acceptance at state $\overrightarrow{q} = s$, since $\mathcal{H}$ has only that state. Let us call $\overrightarrow{c}$ and $\overrightarrow{h}$ the obtained accumulator values. From the semantics above of state s, we know that $\overrightarrow{c} \geq 0$ and $P = \pi 1^{\overrightarrow{c}}$ for a possibly empty prefix π of P, with:

$$\max(\theta(h, \pi), \overrightarrow{c}) = \overrightarrow{h} \tag{4}$$

Similarly, upon feeding the reverse suffix T^{rev} to $\mathcal{H}$, we get $\overleftarrow{q} = s$, $\overleftarrow{c} \geq 0$, and $T^{\text{rev}} = \tau^{\text{rev}} 1^{\overleftarrow{c}}$ for a possibly empty prefix τ^{rev} of T^{rev}, that is $T = 1^{\overleftarrow{c}} \tau$, with:

$$\max(\theta(h, \tau^{\text{rev}}), \overleftarrow{c}) = \overleftarrow{h} \tag{5}$$

Overall, we have $S = PT = \pi 1^{\overrightarrow{c}} 1^{\overleftarrow{c}} \tau = \pi 1^{\overrightarrow{c} + \overleftarrow{c}} \tau$, hence:

$$\begin{aligned}
\theta(h, S) &= \max(\theta(h, \pi), \overrightarrow{c} + \overleftarrow{c}, \theta(h, \tau)) && \text{by the semantics of } \mathcal{H} \\
&= \max(\theta(h, \pi), \overrightarrow{c} + \overleftarrow{c}, \theta(h, \tau^{\text{rev}})) && \text{by } \mathcal{H} \text{ being its own reverse} \\
&= \max(\overrightarrow{h}, \overrightarrow{c} + \overleftarrow{c}, \theta(h, \tau^{\text{rev}})) && \text{by } \overleftarrow{c} \geq 0 \text{ and (4)} \\
&= \max(\overrightarrow{h}, \overrightarrow{c} + \overleftarrow{c}, \overleftarrow{h}) && \text{by } \overrightarrow{c} \geq 0 \text{ and (5)}
\end{aligned} \tag{6}$$

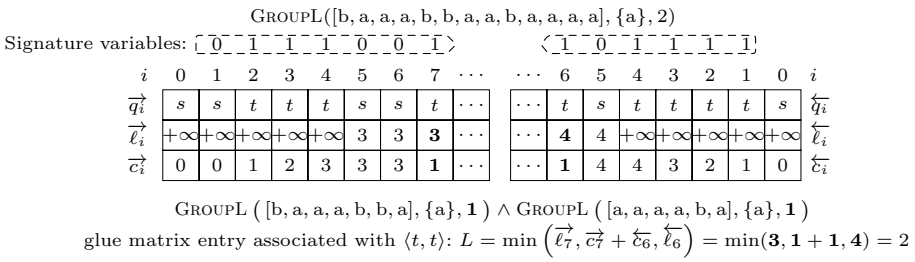


Fig. 3. Use of the entry for the state pair $\langle t, t\rangle$ in the glue matrix of Figure 2d for linking the result variable L with the state and accumulator variables after reading the prefix [b, a, a, a, b, b, a] and corresponding suffix [a, b, a, a, a, a] of a sequence. The left (resp. right) table shows the initialisation (for $i = 0$) and evolution of the state of the mDFA $\mathcal{L}$ in Figure 1d and its accumulators ℓ and c upon reading the symbol at index i of the sequence (resp. its reverse).

The law underlying the last two equalities is that $\max(x, y) = h$ implies that $\max(x, y + z) = \max(h, y + z)$ when $z \geq 0$. Note that $\theta(h, S)$ is now entirely defined in terms of the accumulator values after processing P and T^{rev}.

Posting AUTOMATON$(\mathcal{H}, P, \overrightarrow{H})$ gives access to $\overrightarrow{q}$, $\overrightarrow{c}$, $\overrightarrow{h}$ as variables, with $\overrightarrow{H} = \overrightarrow{h}$. Similarly, posting AUTOMATON$(\mathcal{H}, T^{\text{rev}}, \overleftarrow{H})$ gives access to $\overleftarrow{q}$, $\overleftarrow{c}$, $\overleftarrow{h}$ as variables, with $\overleftarrow{H} = \overleftarrow{h}$. Since the acceptance function α computes $H = \theta(h, S)$, the implied glue constraint is $H = \max(\overrightarrow{h}, \overrightarrow{c} + \overleftarrow{c}, \overleftarrow{h})$, due to (6). □

3.3 Deriving the Glue Constraint

Reconsider the useful class of constraints mentioned at the end of Section 3.1, namely those that can be encoded using an mDFA $\mathcal{M} = \langle Q, \Sigma, \delta, \phi, I, A, \alpha\rangle$ with a single accumulator (hence $k = 1$), which is initialised to zero (hence $I = 0$) at the start state ϕ and increased by a non-negative quantity at each transition as well as by the acceptance function α. We denote $\gamma(q, \sigma)$ the accumulator increase on the transition from state q upon reading symbol σ. Similarly, we denote $\widehat{\gamma}(q, w)$ the *total* accumulator increase on the path from state q upon reading a possibly empty word w. Let $\mathcal{M}' = \langle Q', \Sigma, \delta', \phi', I', A', \alpha'\rangle$ be the reverse of $\mathcal{M}$ (computed as seen in Section 3.1), on the *same* alphabet. The glue constraint then always takes a particular form, as described after defining a needed concept.

Definition 4. *Let PT be a word such that possibly empty prefix P leads to state $\overrightarrow{q}$ of mDFA $\mathcal{M}$, and the reverse of possibly empty suffix T leads to state $\overleftarrow{q}$ of the reverse mDFA $\mathcal{M}'$ of $\mathcal{M}$. The* correction term *for $\overrightarrow{q}$ and $\overleftarrow{q}$ is:*

$$\Delta(\overrightarrow{q}, \overleftarrow{q}) = \widehat{\gamma}(\phi, PT) - (\widehat{\gamma}(\phi, P) + \widehat{\gamma}'(\phi', T^{rev}))$$

Suppose, in addition to $\textsc{Automaton}(\mathcal{M}, PT, R)$, we post the constraints $\textsc{Automaton}(\mathcal{M}, P, \overrightarrow{R})$ and $\textsc{Automaton}(\mathcal{M}', T^{\mathrm{rev}}, \overleftarrow{R})$. Let $\widehat{\delta}(\langle \phi, I\rangle, P) = \langle \overrightarrow{q}, \overrightarrow{v}\rangle$ and $\widehat{\delta}(\langle \phi', I'\rangle, T^{\mathrm{rev}}) = \langle \overleftarrow{q}, \overleftarrow{v}\rangle$. The glue constraint for states $\overrightarrow{q}$ and $\overleftarrow{q}$ is then always of the form $R = \alpha(\overrightarrow{v} + \Delta(\overrightarrow{q}, \overleftarrow{q}) + \overleftarrow{v})$, because $\overrightarrow{v} = \widehat{\gamma}(\phi, P)$, $\overleftarrow{v} = \widehat{\gamma}'(\phi', T^{\mathrm{rev}})$, and $R = \alpha(\widehat{\delta}(\langle \phi, I\rangle, PT)) = \alpha(\widehat{\gamma}(\phi, PT))$. Therefore, we can abbreviate the glue matrix into the matrix of its correction terms.

Example 7. The glue matrices in Figures 2a and 2b can be abbreviated into (0) and $\left(\begin{smallmatrix} 0 & 0 \\ 0 & -1 \end{smallmatrix}\right)$, respectively, which contain constants independent of P and T. □

We now show that the matrix of constant correction terms can be computed efficiently, as it obeys a recurrence relation. By Definition 4, we have the base cases $\Delta(\phi, q') = 0$ for every state q' of $\mathcal{M}'$, and $\Delta(q, \phi') = 0$ for every state q of $\mathcal{M}$. For the step case of two non-start states $\overrightarrow{q}$ and $\overleftarrow{q}$, let $P\sigma T$ be a word such that non-empty prefix P leads to state $\overrightarrow{q}$ of $\mathcal{M}$, with successor $\overrightarrow{r}$ upon reading symbol σ, and the reverse of possibly empty suffix T leads to state $\overleftarrow{r}$ of $\mathcal{M}'$, with successor $\overleftarrow{q}$ upon reading symbol σ. Using Definition 4 with σT as suffix, we get the following recurrence relation:

$$\begin{aligned} \Delta(\overrightarrow{q}, \overleftarrow{q}) &= (\widehat{\gamma}(\phi, P) + \widehat{\gamma}(\overrightarrow{q}, \sigma T)) - (\widehat{\gamma}(\phi, P) + \widehat{\gamma}'(\phi', T^{\mathrm{rev}}\sigma)) \\ &= \widehat{\gamma}(\overrightarrow{q}, \sigma T) - \widehat{\gamma}'(\phi', T^{\mathrm{rev}}\sigma) \\ &= (\gamma(\overrightarrow{q}, \sigma) + \widehat{\gamma}(\overrightarrow{r}, T)) - (\widehat{\gamma}'(\phi', T^{\mathrm{rev}}) + \gamma'(\overleftarrow{r}, \sigma)) \\ &= \Delta(\overrightarrow{r}, \overleftarrow{r}) + \gamma(\overrightarrow{q}, \sigma) - \gamma'(\overleftarrow{r}, \sigma) \end{aligned} \tag{7}$$

The matrix of correction terms for all state pairs can then be computed by dynamic programming. It suffices to initialise to zero the row of ϕ and the column of ϕ'. Then let $\mathrm{name}(q)$ denote a shortest word reaching state q from the start state of its mDFA. The dynamic program uses the recurrence relation (7) for every pair $\langle \overrightarrow{q}, \overleftarrow{q}\rangle$ of non-start states with $\mathrm{name}(\overleftarrow{q}) = \mathrm{name}(\overleftarrow{r})\sigma$. Note that both σ and $\overleftarrow{r}$ are uniquely determined by $\mathrm{name}(\overleftarrow{q})$, and that $\overrightarrow{r}$ is uniquely defined as *the* successor of $\overrightarrow{q}$ under σ, since an mDFA is deterministic. Proceeding by increasing lexicographic order of $\langle |\mathrm{name}(\overrightarrow{q})| + |\mathrm{name}(\overleftarrow{q})|, -|\mathrm{name}(\overrightarrow{q})|\rangle$, where $|w|$ denotes the length of word w, ensures that the right-hand side of (7) is always already determined.

Example 8. The glue matrix $\left(\begin{smallmatrix} 0 & 0 \\ 0 & -1 \end{smallmatrix}\right)$ of Example 7 is for the GroupG constraint, whose mDFA $\mathcal{G}$ in Figure 1b is its own reverse. There are zeros in the left column and top row, by the base cases. The -1 in the lower-right cell follows from the following application of (7): we have $\overrightarrow{q} = t = \overleftarrow{q}$, with $\mathrm{name}(\overleftarrow{q})$ = "1" = $\mathrm{name}(\overleftarrow{r})\sigma$, hence $\sigma = 1$ and $\mathrm{name}(\overleftarrow{r}) = \epsilon$, so $\overleftarrow{r} = s$; since $\overrightarrow{r}$ is the successor of $\overrightarrow{q}$ under σ, we have $\overrightarrow{r} = t$; hence (7) gives $\Delta(t, t) = \Delta(t, s) + \gamma(t, 1) - \gamma'(s, 1) = 0 + 0 - 1 = -1$. Indeed, if the last symbol of prefix P and the first symbol of suffix T are in a group, then the sum of the numbers of groups of P and T^{rev} would be one unit too high and has to be downward adjusted by -1. □

Filling the matrix of $|Q| \cdot |Q'|$ correction terms takes $\Theta(|Q| \cdot |Q'| + |\Sigma|)$ time. Indeed, each correction term is computed in constant time, and the state names can be computed in $\Theta(|Q| + |Q'| + |\Sigma|)$ time.

4 Implied Constraints on Prefixes and Suffixes

In Scheme 3 of Example 1, we showed how to improve pruning in the presence of an implied constraint on the result variables of multiple total-function constraints on the same sequence of variables. We now formalise this idea.

Consider a conjunction of p total-function constraints $C_j(X \to R_j)$ on the *same* sequence X of n variables. For each constraint C_j, assume we have an mDFA $\mathcal{M}_j = \langle Q_j, \Sigma_j, \delta_j, \phi_j, I_j, A_j, \alpha_j \rangle$ that reads a sequence S^j of m_j signature variables channelled with X by signature constraints of arity $a_j \geq 1$ (hence $m_j = n + 1 - a_j$), such that the variable R_j is constrained to be equal to the result returned by $\mathcal{M}_j$ on S^j. (We write S^j rather than S_j so that, in line with the rest of the paper, we can write S^j_i to refer to the element at index i of S^j.) Hence each $C_j(X \to R_j)$ is encoded by $\text{AUTOMATON}(\mathcal{M}_j, S^j, R_j)$ and its signature constraints. Let $\mathcal{M}'_j = \langle Q'_j, \Sigma_j, \delta'_j, \phi'_j, I'_j, A'_j, \alpha'_j \rangle$ be the reverse of $\mathcal{M}_j$, over the *same* alphabet Σ_j: it is used for encoding the reverse of C_j by $\text{AUTOMATON}(\mathcal{M}'_j, (S^j)^{\text{rev}}, R_j)$, using the *same* signature constraints and variables. Note that all the R_j are only defined when X is sufficiently long, namely $n \geq \overline{a} - 1$, where $\overline{a} = \max(a_1, \ldots, a_p)$.

Consider that the p result variables R_j are not independent and that we have an implied constraint $\Im(R_1, \ldots, R_p)$, called a *graph invariant* in [5], on them.

The idea is to improve the propagation on the conjunction of the C_j with $\Im$ (which constrains the *overall* results R_j under the C_j on the *entire* sequence X) by adding also $\Im$ on the *partial* results under the C_j for every sufficiently long *prefix* of X, as well as adding $\Im$ on the *partial* results under the reverses of the C_j for the reverse of every sufficiently long *suffix* of X.

We thus post the implied constraint $\Im$ on the results R^i_j for every not necessarily strict prefix $[X_1, \ldots, X_{i+a_j-1}]$ of X, each prefix being long enough for all the R^i_j to be defined. We also post $\Im$ on the results ${R'}^i_j$ for the reverse of every not necessarily strict suffix $[X_n, \ldots, X_{n-a_j-i+2}]$ of X, each suffix being long enough for all the ${R'}^i_j$ to be defined. We get:

$$\forall j : \forall 1 \leq i \leq n - a_j + 1 : \text{AUTOMATON}(\mathcal{M}_j, [X_1, \ldots, X_{i+a_j-1}], R^i_j)$$
$$\forall j : \forall 1 \leq i \leq n - a_j + 1 : \text{AUTOMATON}(\mathcal{M'}_j, [X_n, \ldots, X_{n-a_j-i+2}], {R'}^i_j)$$
$$\forall \overline{a} \leq i \leq n : \Im(R_1^{i-a_1+1}, \ldots, R_p^{i-a_p+1}) \wedge \Im({R'}_1^{i-a_1+1}, \ldots, {R'}_p^{i-a_p+1})$$

Rather than posting $2 \cdot (n - a_j + 1)$ AUTOMATON constraints for a given $\mathcal{M}_j$, we only post two AUTOMATON constraints over the sequence X and its reverse, where our implementation of AUTOMATON provides access to the internal variables R^i_j and ${R'}^i_j$.

Note that the glue constraints of Section 3 make each implied constraint on a prefix communicate, through shared variables, with the implied constraint on the reverse of the corresponding suffix: we evaluate this experimentally in Section 5.

Example 9. In Scheme 3 of Example 1, we had $p = 4$, $a_1 = a_2 = a_3 = a_4 = 1 = \overline{a}$, and $\Im$ as the implied constraint (1). Further experiments are in Section 5. □

5 Experiments

We have implemented as a generic framework for our method a Prolog predicate taking as arguments (i) a shared sequence of variables, (ii) a list of p mDFAs with their corresponding signature and glue constraints, (iii) a list of p numeric variables R_j to be computed, (iv) a conjunction of implied constraints over the R_j, and (v) some options for controlling the encoding scheme (see below). We have also extended the *Global Constraint Catalogue* [1] with glue constraints for most of its mDFAs and with implied constraints relating arguments of different constraints (e.g. those of [5] and the relation between the number of valleys and peaks [7] of a sequence).[1]

To evaluate our methods, we ran three experiments on a conjunction of two GROUP constraints over a shared sequence of a/b variables, one with $W = \{\text{a}\}$ and one with $W = \{\text{b}\}$. Thus, a total of eight numeric values are computed from the sequence. Each experiment was run on two sets of 1000 randomly generated unique instances. An instance consists of initial domains for the sequence and numeric variables. In each experiment, four encoding schemes were compared: **B** the baseline, i.e. as $p = 8$ AUTOMATON constraints; **BI** as **B** plus 90 implied constraints linking the eight numeric variables, provided by [1, Section 4.3], see Scheme 1 of Example 1; **BG** as **B** plus the appropriate glue constraints posted at every prefix-suffix junction, see Scheme 2; and **BGI** as **BG** plus the same 90 implied constraints posted on the full sequence as well as on every nonempty prefix and suffix, i.e. using Scheme 1+2+3.

All experiments were run in SICStus Prolog 4.3 [9] on a quad core 2.8 GHz Intel Core i7-860 machine with 8MB cache per core, running Ubuntu Linux.

Special attention was devoted to generating meaningful instances, since the eight numeric variables are all but independent, and a truly random choice of their initial domains leads to an unsatisfiable instance in the vast majority of cases. We came up with two instance sets:

Sloppy (11% satisfiable) is generated as follows: First, each a/b variable is assigned 'a' with 10% probability, 'b' with 10% probability, and left unassigned with 80% probability. Then, each numeric variable is given a random subinterval of its feasible interval. If posting the 90 implied constraints on the obtained candidate instance detects failure without search, then the candidate is rejected. Otherwise, it is included in the set.

Strict (96% satisfiable) is generated like the **Sloppy** set, but also, if posting the full **BGI** scheme on the candidate detects failure without search, then the candidate is rejected. Otherwise, it is included in the set.

The results of the three experiments are shown in Figure 4. In the first experiment (left column), sequences of length 20 were used. In the two plots, each point (x, y) denotes that y instances reached a total domain size of at most x

[1] All code and data for the experiments as well as the extended version of the catalogue can be found at `http://www.sics.se/~matsc/research/reversible`.

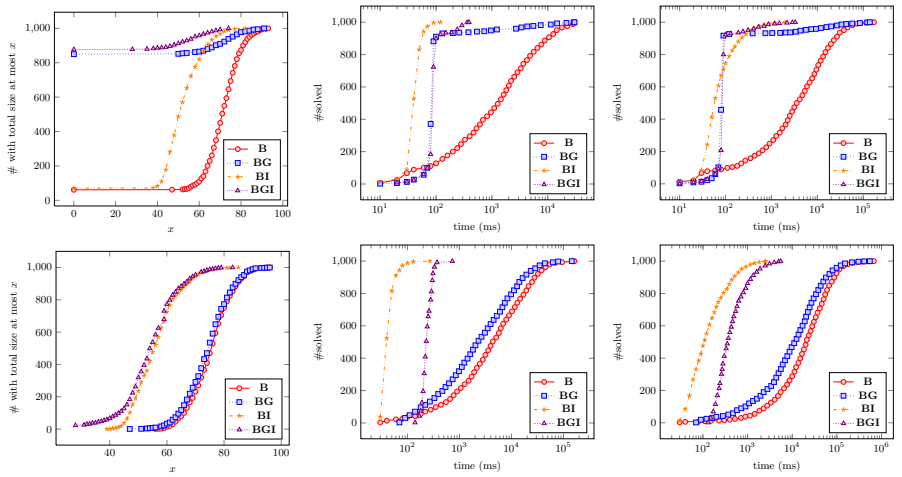


Fig. 4. Experimental results. Comparison of the amount of pruning after posting (left), the time to find the first solution or detect unsatisfiability (centre), and the time to find all solutions (right). Top row: **Sloppy**. Bottom row: **Strict**.

after initial pruning, for the given scheme. We find that **BGI** is the most effective in pruning and **B** the least effective. For **Sloppy**, we see that **BG** detects unsatisfiability far more effectively than **BI**. For **Strict**, we see that **BI** gives much more pruning than **BG**.

For the remaining experiments (centre and right column), we plot the number of instances that can be solved by a given deadline. Sequences of length 10 were used here, to avoid having to impose a time limit, so that we can compare all methods on all instances. We find that the schemes involving implied constraints outperform the schemes that do not.

The **BG** curves are similar throughout the **Sloppy** row, confirming that the glue method more effectively detects unsatisfiability. For the **Strict** instances, **BI** outperformed **BGI**. Partial benchmark results for length 20 and for combining Group and ChangeContinuity [1,5] paint an almost identical picture, except there is some indication that **BGI** is more effective on longer sequences.

Evaluating our schemes on the model inferred by the EDF model seeker [7] is ongoing work, and requires more work and analysis since the model is nontrivial and contains 20 interacting constraints.

6 Constant-Time Move Probing in Local Search

In the context of constraint-based local search [15], consider a total-function constraint $C([X_1, \ldots, X_n] \to R)$ that is encoded using an mDFA $\mathcal{M}$. The violation cost of C under the current assignment β is $|\beta(R) - r|$, where r is the result returned by $\mathcal{M}$ on $[\beta(X_1), \ldots, \beta(X_n)]$. We now show, on an example, how to

use the glue constraint of C in order to compute, in *constant* time, the violation cost of C when probing a move $X_i := v$, which changes the current assignment.

For example, let us start from the ground instance of Figure 3, namely GroupL([b, a, a, a, b, b, a, a, b, a, a, a, a], {a}, 2). It is satisfied, hence its violation cost under the current assignment is 0. Assume now we want to probe changing variable X_7 from 'a' into 'b', that is changing signature variable S_7 from 1 into 0. The violation cost under the resulting new assignment, and its increase compared to the current assignment, are computed in *constant* time as follows:

1. Starting from the prefix column for $i = 6$, we compute the new column for $i = 7$, upon the mDFA $\mathcal{L}$ in Figure 1d reading a 0 instead of a 1: we get $\overrightarrow{q}_7 = s$ and $\overrightarrow{\ell}_7 = 3 = \overrightarrow{c}_7$. The suffix column for $i = 6$ remains unchanged.
2. Using the glue matrix entry in Figure 2d for the state pair $\langle \overrightarrow{q}_7, \overleftarrow{q}_6 \rangle = \langle s, t \rangle$, we know that the new sequence has $\min(\overrightarrow{\ell}_7, \overleftarrow{c}_6, \overleftarrow{\ell}_6) = \min(3, 1, 4) = 1$ as the lowest group size (that is not 2 anymore).
3. Hence the violation cost under the new assignment is $|2 - 1| = 1$ (as we still have $R = 2$), and has thus increased by $1 - 0 = 1$.

Thus, with one matrix of states and accumulator values for the mDFA and one for the reverse mDFA, as in Figure 3, probing can be done in constant time, thereby beating the linear time achieved for the Automaton constraint in [11].

Commitment to a move actually selected by the search (meta-)heuristic follows the same steps as above, namely once for updating the values of the left table (in Figure 3) from the concerned column until the last column, and once for updating the values in the right table from the concerned column until the first column. This takes time linear in the length of X.

7 Conclusion

For a total-function constraint on a sequence of variables whose result is invariant under sequence reversal, we have shown how to derive, from a compact encoding of the constraint via an automaton with accumulators, an implied constraint between the result variables for a sequence of variables, a prefix thereof, and the corresponding suffix. Such total-function constraints have proved very useful, for instance in production sequencing and staff rostering. We have shown that the glue constraint is unique and always exists. We have also shown the usefulness of the derived implied constraint in constraint solving, both by local search, where the implied constraint enables constant-time move probing, and by propagation-based systematic search, where the implied constraint improves propagation: our concept is thus not oriented toward a specific solver technology.

Other constraints than Automaton could be used for handling memory-DFAs and deriving glue constraints: recall that our method supports multiple accumulators, and needs access as variables to the sequence of values for each accumulator, as well as access as variables to the sequence of automaton states. For instance, CostRegular [10] is currently limited to the class of single-accumulator mDFAs discussed in Section 3.3, and is compared in detail with Automaton in [6]. Encodings based on Slide [8] could also be investigated.

Acknowledgements. The first author is supported by the *Gaspard Monge Program for Optimisation and operations research*. The last three authors are supported by grants 2011-6133 and 2012-4908 of the Swedish Research Council (VR). We thank Pascal Van Hentenryck, with whom we discovered a few years ago the unpublished result of Section 3.3. We thank the anonymous reviewers for their valuable comments.

References

1. Beldiceanu, N., Carlsson, M., Demassey, S., Petit, T.: Global constraint catalogue: Past, present, and future. Constraints 12(1), 21–62 (2007)
2. Beldiceanu, N., Carlsson, M., Flener, P., Pearson, J.: On matrices, automata, and double counting in constraint programming. Constraints 18(1), 108–140 (2013)
3. Beldiceanu, N., Carlsson, M., Flener, P., Pearson, J.: On the reification of global constraints. Constraints 18(1), 1–6 (2013)
4. Beldiceanu, N., Carlsson, M., Petit, T.: Deriving filtering algorithms from constraint checkers. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 107–122. Springer, Heidelberg (2004)
5. Beldiceanu, N., Carlsson, M., Rampon, J.-X., Truchet, C.: Graph invariants as necessary conditions for global constraints. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 92–106. Springer, Heidelberg (2005)
6. Beldiceanu, N., Flener, P., Pearson, J., Van Hentenryck, P.: Propagating regular counting constraints. In: Brodley, C.E., Stone, P. (eds.) AAAI 2014. AAAI Press (2014)
7. Beldiceanu, N., Ifrim, G., Lenoir, A., Simonis, H.: Describing and generating solutions for the EDF unit commitment problem with the ModelSeeker. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 733–748. Springer, Heidelberg (2013)
8. Bessière, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: SLIDE: A useful special case of the CARDPATH constraint. In: Ghallab, M., et al. (eds.) ECAI 2008, pp. 475–479. IOS Press (2008)
9. Carlsson, M., Ottosson, G., Carlson, B.: An open-ended finite domain constraint solver. In: Glaser, H., Hartel, P., Kuchen, H. (eds.) PLILP 1997. LNCS, vol. 1292, pp. 191–206. Springer, Heidelberg (1997), `http://sicstus.sics.se/`
10. Demassey, S., Pesant, G., Rousseau, L.M.: A `Cost-Regular` based hybrid column generation approach. Constraints 11(4), 315–333 (2006)
11. He, J., Flener, P., Pearson, J.: An *automaton* constraint for local search. Fundamenta Informaticae 107(2-3), 223–248 (2011)
12. Mohri, M.: Weighted automata algorithms. In: Droste, M., Kuich, W., Vogler, H. (eds.) Handbook of Weighted Automata. Monographs in Theoretical Computer Science, pp. 213–254. Springer (2009)
13. Mohri, M., Pereira, F.C.N., Riley, M.: The design principles of a weighted finite-state transducer library. Theoretical Computer Science 231(1), 17–32 (2000), The OpenFst Library is available `http://www.openfst.org/`
14. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 482–495. Springer, Heidelberg (2004)
15. Van Hentenryck, P., Michel, L.: Constraint-Based Local Search. MIT Press (2005)