Constraints
https://doi.org/10.1007/s10601-022-09333-0

LETTER

When bounds consistency implies domain consistency for regular counting constraints

Barnaby Martin[1] · Justin Pearson[2]

Accepted: 25 April 2022

Abstract
Finite automata with counters are often used to specify families of constraints. The addition of counters provides more power than membership in regular languages that is possible with finite automata. All available propagation algorithms for counter automata based constraints maintain only bounds consistency on counter variables, and although it is possible to maintain domain consistency it can be computationally very costly. In this paper we give an algorithm that decides when maintaining bounds consistency for an automata with a single counter implies domain consistency.

Keywords Automata · Consistency

1 Introduction

Finite automata have long being used to specify families of constraints: in [1] and [2] a decomposition based approach is given where automata augmented with counters are used to specify constraints; in [3] a domain consistent propagation algorithm is given for membership of a regular language as specified by a finite automata; and in [4, 5] an extension of the algorithm in [3] is given where transitions of an automaton are augmented with a cost that is incurred when a value is assigned. In general an automaton is used to specify a constraint on a sequence of variables and possibly extra variables representing counters. This allows complex constraints over sequences to be expressed. The example automaton in Fig. 1 together with the counter c constrains the number of times the subsequence "aab" appears in a sequence. For examples of more complex families of constraints specified by automata with counters see [6–8].

This paper builds on the work in [9] where an efficient dynamic programming propagation algorithm is given for constraints specified by an automaton with a single counter that maintains bounds consistency on counter variables and domain consistency on sequence variables. It was left as an open question how to extend the propagation algorithm to maintain domain consistency for counter variables.

✉ Justin Pearson
justin.pearson@it.uu.se

[1] Department of Computer Science, University of Durham, South Road, Durham DH1 3LE, UK

[2] Department of Information Technology, Uppsala University, 751 05 Uppsala, Sweden

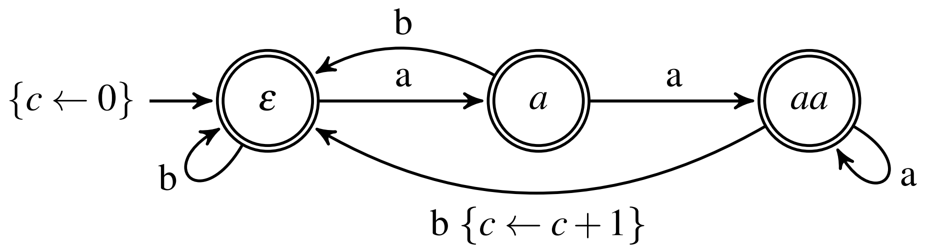

Fig. 1 An all-accepting single-counter automaton $\mathscr{AAB}$ for the constraint NumberWord(N,X, "aab")

In this paper we provide a decision procedure that given an automaton with a single counter decides if maintaining bounds consistency on the counter variable is sufficient to maintain domain consistency, and hence when the algorithm in [9] can be used to maintain domain consistency on the counter variable. Knowing when it is sufficient to maintain bounds consistency instead of maintaining domain consistency is of practical interest, because in general maintaining bounds consistency for a constraint is computationally cheaper than maintaining domain consistency [10].

2 Background

In this section we give some necessary notation and background on automata with counters (Section 2.1), how to use automata to specify constraints (Section 2.2), notions of mixed consistency (Section 2.3), and a survey (Section 2.4) of the available propagation methods for families of constraints specified by automata.

2.1 Finite all-accepting single-counter automata

Definition 1 An *all-accepting single-counter automaton* $\mathscr{A}$ is a tuple

$$(Q, \Sigma, q_0 \in Q, \rightarrow \subseteq Q \times \Sigma \times \mathbb{Z} \times Q)$$

where Q is the set of states, q_0 is the initial state, Σ is a set of alphabet symbols, and $\rightarrow$ is the transition relation.

We say that an all-accepting single-counter automaton $\mathscr{A}$ is *deterministic* if for all states $q \in Q$ and alphabet symbols $\sigma \in \Sigma$ there exists unique q' and z such that (q, σ, z, q') belongs to the relation $\rightarrow$.

We write $q \overset{\sigma}{\underset{z}{\rightarrow}} q'$ if there is some $z \in \mathbb{Z}$ such that (q, σ, z, q') belongs to $\rightarrow$. Given a transition $q \overset{\sigma}{\underset{z}{\rightarrow}} q'$ the value z is referred to as the *counter change* for that transition. Given a word $\sigma = \sigma_1 \cdots \sigma_n \in \Sigma^*$ an accepting sequence is a sequence of states $s_1,\ldots,s_n$ in Q, and a sequence of integers $z_1,\ldots,z_n$ such that

$$q_0 \overset{\sigma_1}{\underset{z_1}{\rightarrow}} s_1 \overset{\sigma_2}{\underset{z_2}{\rightarrow}} \cdots \overset{\sigma_n}{\underset{z_n}{\rightarrow}} s_n$$

Given a deterministic all-accepting single-counter automaton $\mathscr{A}$ and a word $\sigma_1 \cdots \sigma_n \in \Sigma^*$ then the cost $\mathscr{C}(w)$ of the word is defined to be the cost of the (unique) accepting sequence

$$\mathscr{C}\left(q_0 \underset{z_1}{\overset{\sigma_1}{\rightarrow}} s_1 \underset{z_2}{\overset{\sigma_2}{\rightarrow}} \cdots \underset{z_n}{\overset{\sigma_n}{\rightarrow}} s_n\right) = z_1 + \cdots + z_n.$$

Example 1 Consider the automaton $\mathscr{AAB}$ in Fig. 1 taken from [9]. It represents a deterministic all-accepting single-counter automaton with state set $Q = \{q_0, a, aa\}$ and alphabet $\Sigma = \{\text{a,b}\}$. The transition relation $\rightarrow$ is given by the labelled arcs between states. Each arc is labelled with its alphabet symbol and an increment $\{c \leftarrow c + z\}$ which is omitted if z equals 0. The initial state q_0 is denoted by an arc coming from no state. As a result, the final cost c of a word is the number of occurrences of the word "aab" within the string.

2.2 Regular counting constraints

As in [9], a *regular counting constraint* is defined as a constraint that can be modelled by a deterministic all-accepting single-counter automaton. Given a variable c and a sequence of variables $x = x_1,\ldots,x_n$ then the RegCount($c, x, \mathscr{A}$) constraint is satisfied if the value of variable c, called the *counter variable*, is equal to the final value of the counter after the deterministic all-accepting single-counter automaton $\mathscr{A}$ has consumed the values of the entire sequence $x = x_1,\ldots,x_n$ of variables. That is c equals $\mathscr{C}(x_1 \cdots x_n)$. It is assumed that the domains of the variables $x_1,\ldots,x_n$ is subsets of the set of symbols Σ of the automaton $\mathscr{A}$. We refer to the sequence x as the *sequence variables*. For example, the constraint RegCount($c, x, \mathscr{AAB}$) with the automaton $\mathscr{AAB}$ as specified in Fig. 1 holds if c is the number of occurrences of the non-empty word aab in the sequence x.

2.3 Mixed propagation

Domain and bounds consistency are normally defined for a given constraint and the domains of all its variables [10]. The domains of the variables of a constraint are said to be *domain consistent* if given any possible value of a variable in a domain there is some solution to the constraint that takes values from the other domains. For example the domains $x = \{1,3\}$ and $y = \{3,4\}$ are domain consistent for the constraint $x \leq y$. Bounds consistency is a little bit more subtle (again see [10]) and instead considers domains as intervals. In Definition 2 we will use a variation of bounds-($\mathbb{Z}$) consistency.

The propagators in [4, 5, 9] maintain mixed consistency on sequence and counter (or cost variables). In [10] no formal definition of mixed consistency is given. Here we specialise (to simplify notation) the definition of mixed consistency, taken from [11], to the special case of an arbitrary number of domain consistent sequence variables and a single bounds-($\mathbb{Z}$) variable.

Definition 2 Given any finite subset K of the integers $\mathbb{Z}$, let min(K) denote the minimum integer in K and max(K) denote the maximum integer in K. Consider a constraint, C, over the variables $x_1,\ldots,x_n$, and c. Let $X_1,\ldots,X_n$ be the domains of the variables $x_1,\ldots,x_n$ and let K be the domain of c, then these domains are *domain consistent for* $x_1,\ldots,x_n$ *and bounds-*($\mathbb{Z}$) *consistent for* c if the following two conditions are met. First, for all $1 \leq i \leq n$ and all $v \in X_i$ there exist $\langle v_1, \ldots v_n\rangle$ in $X_1 \times \cdots \times X_n$ and some value k with $\min(K) \leq k \leq \max(K)$ such that $\langle v_1, \ldots, v_n, k\rangle$ is a solution to C and $v_i = v$; and second given some value k that is either min(K) or max(K), then there exist $\langle v_1, \ldots, v_n\rangle$ in $X_1 \times \cdots \times X_n$ such that $\langle v_1, \ldots, v_n, k\rangle$ is a solution to C. This defines *mixed consistency* specialised to domain consistency for an arbitrary number of variables and bounds-($\mathbb{Z}$) for a single variable.

2.4 Propagating regular counting constraints

In [1, 2] a general framework is given for decomposing automata based constraints into a combination of element constraints and table constraints. In general this decomposition is not domain consistent, but for a finite automaton without any counter variables the decomposition achieves domain consistency. Domain consistency for automata without counters is also achieved with the specialised algorithm in [3] that unfolds an automaton over a sequence of length n to a graph representing potential transitions. As values are removed from domains, a forward and backward pass of the unfolded graph prunes further values from the domain resulting from parts of the unfolded graph that are not reachable from either an initial or final state.

In [4, 5] a propagation algorithm is given for automata without counter variables, but with costs assigned to the values appearing at positions in the sequence, the total cost of a sequence is the sum along the costs that sequence. For each sequence length a cost matrix must be given. The algorithms in [4, 5] maintain bounds consistency on the cost variable and domain consistency on the sequence variables. In a footnote in [4] it is hinted that it is possible to extended the model with costs that depend on not only the value and position, but the state of the automaton as well. With such an extension to cost regular it would be easy to calculate a cost matrix corresponding to a given all-accepting single-counter automaton and specified sequence length. One way of viewing the difference between cost regular [4] and regular counting constraints [9] is that regular counting constraints define a family of cost regular constraints defined by a all-accepting single-counter automaton. The algorithm in [9] works directly on a representation of an automaton and gives a dynamic programming algorithm that provides domain consistency on the sequence variables, but again only provides bounds consistency on the cost variables. When working directly on a representation of an automaton the algorithm in [9] is more efficient than computing the cost matrix and using the algorithms in [4, 5].

3 Deciding when bounds consistency implies domain consistency

Definition 3 Let $\mathscr{A}$ be a deterministic all-accepting single-counter automaton. We call $\mathscr{A}$ *blockwise counter-convex* if, for all n and all sets $\Sigma_1,\dots,\Sigma_i,\dots\Sigma_n$, with each $\Sigma_i \subseteq \Sigma$: whenever there are words w_1 and w_2 of length n from $\Sigma_1\times\cdots\times\Sigma_n$ with costs $\mathscr{C}(w_1) \le \mathscr{C}(w_2)$, then, for all $\mathscr{C}(w_1) \le k \le \mathscr{C}(w_2)$, there is a word w_k of length n from $\Sigma_1\times\cdots\times\Sigma_n$ with cost $\mathscr{C}(w_k) = k$.

We now connect the notation of blockwise counter-convex with the mixed consistency of Definition 2. When using a blockwise counter-convex automaton to specify a constraint, enforcing mixed consistency implies that the counter variable is also domain consistent; and further, that any automaton, where enforcing mixed consistency gives domain consistency on the counter variable, is itself blockwise counter-convex.

Lemma 1 *Let $\mathscr{A}$ be a deterministic all-accepting single-counter automaton $\mathscr{A}$ that is blockwise counter-convex. Given a constraint* $\textsc{RegCount}(c, x, \mathscr{A})$, *if the domain of c is bounds-($\mathbb{Z}$) consistent and the domains of the sequence variables $x = x_1,\dots,x_n$ are domain consistent as in Definition 2, then the domain of c is domain consistent.*

Further, let $\mathscr{A}$ be some deterministic all-accepting single-counter automaton. If every sequence of variables x, and all domains of x and c of the constraint RegCount$(c, x, \mathscr{A})$ *that satisfy Definition* 2 (*with c being bounds*-($\mathbb{Z}$) *consistent*) *also gives that c is domain consistent, then $\mathscr{A}$ is blockwise counter-convex.*

Proof First we prove that if an automaton $\mathscr{A}$ is blockwise counter-convex then bounds-($\mathbb{Z}$) consistency on the counter variable implies that the counter variable is domain consistent. Let K be the domain of the counter variable c and let $X_1,\ldots,X_n$ be the domains of the sequence. By definition, because K is bounds-($\mathbb{Z}$) consistent there are two sequences $\sigma_1^l, \ldots, \sigma_n^l$ and $\sigma_1^u, \ldots, \sigma_n^u$ in $X_1,\ldots,X_n$ such that $\mathscr{C}\left(\sigma_1^l \cdots \sigma_n^l\right) = \min(K)$ and $\mathscr{C}\left(\sigma_1^u \cdots \sigma_n^u\right) = \max(K)$. Since $\mathscr{A}$ is blockwise counter-convex, for any k such that $\min(K) \leq k \leq \max(K)$ there exists some sequence $\tau_1,\ldots,\tau_n$ in $X_1,\ldots,X_n$ with $\mathscr{C}(\tau_1 \cdots \tau_n) = k$. Hence the domain K of c is also domain consistent because for any k in K we can find a satisfying assignment from the domains of the sequence variables to the constraint. Given an automaton $\mathscr{A}$ and a constraint RegCount$(c, x, \mathscr{A})$ such that the domains of c and the sequence variables satisfy the conditions of Definition 2 (with c being bounds-($\mathbb{Z}$) consistent) that implies that c is also domain consistent, then $\mathscr{A}$ is blockwise counter-convex. We prove the contrapositive. Assume that $\mathscr{A}$ is not blockwise counter-convex, and the domains of c and x are as above. Then there must be some value k in K such that there is no sequence $\sigma_1,\ldots,\sigma_n$ in $X_1,\ldots,X_n$ with $\mathcal{C}(\sigma_1 \cdots \sigma_n) = k$. This contradicts Definition 2. Hence $\mathscr{A}$ is blockwise counter-convex. □

Note that blockwise counter-convex is a strong property. It implies that there can be no holes in the domain. Given a constraint RegCount$(c, x, \mathscr{A})$ where $\mathscr{A}$ is blockwise counter-convex, and $x = x_1,\ldots,x_n$, then if the minimum and maximum values of c are supported by some assignments to $x_1,\ldots,x_n$ then all values between the minimum and the maximum are also supported by some assignment. Thus if some other propagator removes some value of c between the minimum and the maximum giving a hole in the domain, no values from the domains of $x_1,\ldots,x_n$ will be removed.

Blockwise counter-convex is a desirable property of an automaton, because it implies that the propagation algorithm in [9] will give domain consistency on the counter variable, and there is no need to look for a more complex domain consistent propagator. The following theorem shows that given some deterministic automaton $\mathscr{A}$ it is possible to decide if it is blockwise counter-convex. Note that given some constraint RegCount$(c, x, \mathscr{A})$ with $x = x_1,\ldots,x_n$ it is possible to check in polynomial time that the domains of $x_1,\ldots,x_n$ and the domain of c are blockwise counter-convex, but the following theorem is stronger and gives a condition for any sequence length that the domains are blockwise counter-convex that only depends on the automaton $\mathscr{A}$.

Theorem 1 *There is an EXPTIME algorithm to decide whether an all-accepting single-counter deterministic automaton, $\mathscr{A}$, is blockwise counter-convex.*

Proof The algorithm works as follows. Given a deterministic all-accepting single-counter automaton $\mathscr{A} = (Q, \Sigma, q_0 \in Q, \rightarrow \subseteq Q \times \Sigma \times \mathbb{Z} \times Q)$, first, it computes which states are reachable from the initial state q_0. For each of these states q, and all pairs of successor states q_1 and q_2 of q, accessible by reading σ_1 and σ_2, with counter change z_1 and z_2, respectively, we perform the following. For all words x of length $4|Q|^2$ we run x through the automaton starting from q_1 as well as q_2, obtaining counter change z_1' and z_2', respectively. If $|z_1 + z_1' - z_2 - z_2'| > 1$, then reject. Finally, if the algorithm never rejects, then accept.

Correctness. Let us first argue that, if the algorithm rejects, with witnesses q,q_1,q_2,σ_1 ,σ_2, and x as above, then the automaton is not blockwise counter-convex. Let y be some word that moves the automaton from its initial state to q. Consider the words $y \cdot \sigma_1 \cdot x$ and $y \cdot \sigma_2 \cdot x$. Let $\Sigma_1,\ldots,\Sigma_{|y|}$ be singletons built from the $|y|$ letters of y, in the obvious way. Similarly, let $\Sigma_{|y|+2},\ldots,\Sigma_{|y|+1+|x|}$ be singletons built from the $|x|$ letters of x. Now an obvious counterexample exists as

$$\Sigma_1 \times \cdots \times \Sigma_{|y|} \times \{\sigma_1, \sigma_2\} \times \Sigma_{|y|+2} \times \cdots \times \Sigma_{|y|+1+|x|}$$

contains precisely two words whose counts differ by more than one.

It remains to argue that, if the algorithm accepts, the automaton is blockwise counter-convex. We will proceed with the contrapositive and assume the automaton is not blockwise counter-convex. Let some words x and y of minimal length n witness this and assume that $\Sigma_1,\ldots,\Sigma_n$ are given. Now, a certain *path* can be built between x and y where two words are considered adjacent iff their Hamming distance is 1. This path can be chosen to consist only of words that are all in $\Sigma_1 \times\cdots \times\Sigma_n$. (Indeed, the shortest path, of length the Hamming distance between x and y, will have this property.) At some point along this path, there must be words u and u', with Hamming distance 1, whose counts are more than 1 apart. Otherwise, we would not be witnessing a failure of blockwise counter-convexity. Let us decompose u and u' into $v \cdot \sigma_1 \cdot w$ and $v \cdot \sigma_2 \cdot w$, respectively. Suppose that q is the state reached from the start state when v is read. Let $q_0,q_1,\ldots,q_{|w|}$ be the sequence of states in which $\mathscr{A}$ progresses while reading $\sigma_1 \cdot w$ from state q. Let $q'_0, q'_1, \ldots, q'_{|w|}$ be the sequence of states in which $\mathscr{A}$ progresses while reading $\sigma_2 \cdot w$ from state q.

Eventually, for some $i < j \leq |Q|^2$, some pair $\left(q_i, q'_i\right)$ repeats, that is $\left(q_i, q'_i\right) = \left(q_j, q'_j\right)$. Split w into three words w_{left} (positions 1 to i inclusive), w_{middle} (positions $i+1$ to j inclusive) and w_{right} (positions $j+1$ to $|w|$ inclusive). Consider what happens to the counter when reading $\sigma_1 \cdot w_{\text{left}}$ and $\sigma_2 \cdot w_{\text{left}}$. The counters can not be more than 1 apart, by minimality of n. Now, consider what happens to the counter when reading both $\sigma_1 \cdot w$ and $a_2 \cdot w$ precisely during the interval when w_{middle} is read. Suppose the counters did not diverge at all in this interval, i.e. they each advanced the same number. Then we again contradict minimality of n by considering the words $\sigma_1 \cdot w_{\text{left}} \cdot w_{\text{right}}$ and $\sigma_2 \cdot w_{\text{left}} \cdot w_{\text{right}}$. Thus, the counters must have diverged by at least one in this interval and we will find that $\sigma_1 \cdot w_{\text{left}} \cdot w_{\text{middle}} \cdot w_{\text{middle}} \cdot w_{\text{middle}}$ and $\sigma_2 \cdot w_{\text{left}} \cdot w_{\text{middle}} \cdot w_{\text{middle}} \cdot w_{\text{middle}}$ witness that the automaton is not blockwise counter-convex. This implies indeed that one need not look ahead more than $4|Q|^2$ letters, as in our algorithm, and the result follows.

Complexity. The number of words of length $4|Q|^2$ is $|\Sigma|^{4|Q|^2}$. The outer loops of the algorithm run a total of $|Q|^3$ times. This gives a total time complexity of $|Q|^3 \cdot |\Sigma|^{4|Q|^2} \cdot 4|Q|^2$ where running a step of the input automaton is considered to be a unit cost. □

4 Conclusion

In this paper we have shown that it is decidable if bounds consistency implies domain consistency for single counter automata specifying families of constraints. Although the algorithm in Theorem 1 is of exponential complexity it is not intended to run the algorithm each time the propagator is called. The algorithm should be run off-line before constraint solving begins. Further, because of the power of automata with a single counter the majority of automata used in practice (see [12]) have very few states and alphabet symbols and hence the complexity $|Q|^3 \cdot |\Sigma|^{4|Q|^2} \cdot 4|Q|^2$ will be manageable. Throughout the paper it has been assumed all automata are all-accepting single-counter. It would be interesting to extend the analysis where some states are non-accepting. This would mean that some words would not be accepted by an automaton. To understand when bounds consistency would imply domain consistency would require an analysis of when an automaton has enough accepting states so that there are enough accepting words to support all the values of the counter variable between the minimum and the maximum. Understanding what such a condition would look like is left as future work.

References

1. Beldiceanu, N., Carlsson, M., & Petit, T. (2004). Deriving filtering algorithms from constraint checkers. In M. Wallace (Ed.) *CP 2004, LNCS*, (Vol. 3258 pp. 107–122). Springer.
2. Beldiceanu, N., Carlsson, M., Debruyne, R., & Petit, T. (2005). Reformulation of global constraints based on constraints checkers. *Constraints*, *10*(4), 339–362.
3. Pesant, G. (2004). A regular language membership constraint for finite sequences of variables. In M. Wallace (Ed.) *CP 2004, LNCS*, (Vol. 3258 pp. 482–495). Springer.
4. Demassey, S., Pesant, G., & Rousseau, L.M. (2006). A Cost-Regular based hybrid column generation approach. *Constraints*, *11*(4), 315–333.
5. Menana, J., & Demassey, S. (2009). Sequencing and counting with the multicost-regular constraint. In W.J. van Hoeve J.N. Hooker (Eds.) *CP-AI-OR 2009, LNCS*, (Vol. 5547 pp. 178–192). Springer.
6. Arafailova, E., Beldiceanu, N., Carlsson, M., Flener, P., Francisco Rodríguez, M.A., Pearson, J., & Simonis, H. (2016). Systematic derivation of bounds and glue constraints for time-series constraints. In M. Rueher (Ed.) *CP 2016, LNCS*, (Vol. 9892 pp. 13–29). Springer.
7. Beldiceanu, N., Carlsson, M., Douence, R., & Simonis, H. (2016). Using finite transducers for describing and synthesising structural time-series constraints. *Constraints*, *21*(1), 22–40.
8. Francisco Rodríguez, M.A. (2017). Analysis, synthesis and application of automaton-based constraint descriptions. Ph.D. thesis, Department of Information Technology, Uppsala University, Sweden, Department of Information Technology, Uppsala University, Sweden. http://urn.kb.se/resolve?urn=urn%3Anbn%3Ase%3Auu%3Adiva-332149.
9. Beldiceanu, N., Flener, P., Pearson, J., & Van Hentenryck, P. (2014). Propagating regular counting constraints. In C.E. Brodley P. Stone (Eds.) *AAAI 2014* (pp. 2616–2622). AAAI Press.
10. Schulte, C., & Stuckey, P. J. (2005). When do bounds and domain propagation lead to the same search space? *Transactions on Programming Languages and Systems (TOPLAS), 27*(3), 388–425.
11. Scott, J.D. (2016). Other things besides number: Abstraction, constraint propagation, and string variable types. Ph.D. thesis, Department of Information Technology, Uppsala University, Sweden. https://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-273311.
12. Beldiceanu, N., Carlsson, M., Demassey, S., & Petit, T. (2007). Global constraint catalogue: Past, present, and future. *Constraints*, *12*(1), 21–62. The catalogue and the current working version are available at http://sofdem.github.io/gccat.