Bound Consistency for Binary Length-Lex Set Constraints

Justin Yip
Brown University
Supervisor: Pascal Van Hentenryck
Collaborators: Carmen Gervet, Grégoire Dooms

Abstract

The length-lex representation has been recently proposed for representing sets in Constraint Satisfaction Problems. The length-lex representation directly captures cardinality information, provides a total ordering for sets, and allows bound consistency on unary constraints to be enforced in time $\tilde{O}(c)$, where c is the cardinality of the set. However, no algorithms were given to enforce bound consistency on binary constraints. This paper addresses this open issue. It presents algorithms to enforce bound consistency on disjointness and cardinality constraints in time $O(c^3)$. Moreover, it presents a generic bound-consistency algorithm for any binary constraint S which requires $\tilde{O}(c^2)$ calls to a feasibility subroutine for S.

Introduction

The length-lex representation has been recently proposed for representing sets in CSPs (Gervet & Van Hentenryck 2006) and it offers two computational benefits. First, contrary to earlier set representations such as the subset-bound domain (Puget 1992; Gervet 1997), it features a total ordering on sets, which makes it possible to define, and enforce, bound consistency (Van Hentenryck 1989). Bound consistency cannot be enforced for subset-bound domains, since they only use a partial ordering on sets.[1] Second, the length-lex domain directly integrates cardinality and lexicographic information which are so important in set CSPs as eloquently articulated in (Azevedo & Barahona 2000; Sadler & Gervet 2008).

Gervet and Van Hentenryck (2006) showed how to enforce bound consistency on many unary constraints in time $\tilde{O}(c)$, where c is the cardinality of the length-lex intervals. However, they did not discuss binary constraints beyond suggesting a number of inference rules to reduce the search space. This is unfortunate since set CSPs typically involve binary constraints over fixed cardinality sets such as disjointness ($X \cap Y = \emptyset$) or, more generally cardinality constraints of the form $|X \cap Y| \leq k$ or $|X \cap Y| \geq k$ where k is a positive integer. These constraints naturally appear in applications such as balanced incomplete block designs, Steiner systems, cryptography problems, network design, and sport scheduling problems, which are more generally cast as Combinatorial Design Problems (Colbourn, Dinitz & Stinson 1999).

This paper remedies this limitation. It presents a generic bound-consistency algorithm for arbitrary binary constraints over sets. The bound-consistency algorithm only assumes the existence of an algorithm to check whether a constraint C has a solution in two length-lex intervals X and Y, i.e., $\exists s \in X, t \in Y : C(s,t)$. The generic algorithm makes $O(c^2 \log n)$ calls to the feasibility algorithm, where c is the cardinality of the sets and n is the size of the universe, i.e., the set of elements which may appear in the sets. The paper also shows that, for disjoint and cardinality constraints, feasibility checking can be performed in time $O(c)$. Moreover, by exploiting the constraint semantics, it is possible to reduce the complexity of the bound-consistency algorithm to $O(c^3)$ for disjoint and cardinality constraints. These results are particularly appealing, since the complexity is independent of the size of the underlying universe.

To our knowledge, this paper thus presents the first polynomial algorithms for enforcing bound consistency on set constraints. The key technical insight of the paper is to recognize that length-lex intervals can be naturally decomposed into a class of intervals that enjoys some nice closure properties, greatly simplifying the design of the algorithms. Once this decomposition is obtained, the schema of the algorithms resembles the overall design for unary constraints and combinatorial design algorithms in general.

The rest of the paper is organized as follows. The first two sections recall the main notions in length-lex domains and define the concepts of bound consistency. We then provide an overview of the concepts and algorithms in the paper. The concept of PF-closed interval is presented and we then show how a length-lex interval can be decomposed into a sequence of PF-closed intervals. The paper then covers the generic successor algorithm at the core of the bound-consistency algorithm and analyzes its complexity. A section is devoted to the implementation of the disjoint constraint and the paper is then concluded.

Length-lex Domains

Conventions For simplicity, we assume that sets take their values in a universe U of integers $\{1, \ldots, n\}$ equipped with traditional set operations. Set variables are denoted by

[1]Some authors have proposed weaker definitions of bound consistency for such cases.

$S_1, S_2, \ldots$. Elements of U are denoted by the letters e and f possibly subscripted and sets are denoted by the letters $m, M, s, t, x,$ and y. A subset m of U of cardinality c is denoted $\{m_1, m_2, ..., m_c\}$ ($m_1 < m_2 < m_3 ... < m_c$) and thus m_j denotes the j-th smallest value in m. The notation $m_{i..j}$ is the shorthand for $\{m_i, m_{i+1}, ..., m_j\}$. Finally, we call c-set any set of cardinality c. Some algorithms in this paper are only given for a fixed cardinality c but are easily extended to the general case.

Length-lex Ordering The length-lex ordering $\preceq$ totally orders sets first by cardinality and then lexicographically.

Definition 1 *The length-lex ordering $\preceq$ is defined by:*
$s \preceq t$ ***iff*** $s = \emptyset \vee |s| < |t| \vee$
$|s| = |t| \wedge (s_1 < t_1 \vee s_1 = t_1 \wedge s \setminus \{s_1\} \preceq t \setminus \{t_1\})$

Example 1 *Given $U = \{1, .., 4\}$, we have* $\emptyset \preceq \{1\} \preceq \{2\} \preceq \{3\} \preceq \{4\} \preceq \{1,2\} \preceq \{1,3\} \preceq \{1,4\} \preceq \{2,3\} \preceq \{2,4\} \preceq \{3,4\} \preceq \{1,2,3\} \preceq \{1,2,4\} \preceq \{1,3,4\} \preceq \{2,3,4\} \preceq \{1,2,3,4\}$.

Definition 2 *Given a universe U, a length-lex interval is a pair of sets $\langle m, M\rangle$. It represents the sets between m and M in the length-lex ordering, i.e., $\{s \subseteq U \mid m \preceq s \preceq M\}$.*

Example 2 (Length-Lex Interval) *Given $U = \{1, .., 6\}$, the interval $\langle\{1,3,4\}, \{1,5,6\}\rangle$ denotes the set* $\{\{1,3,4\}, \{1,3,5\}, \{1,3,6\}, \{1,4,5\}, \{1,4,6\}, \{1,5,6\}\}$.

Bound Consistency

Since the length-lex ordering is a total order on sets, it is possible to enforce bound consistency on set constraints. A constraint is bound-consistent if each bound of each domain belongs to at least one solution of the constraint which satisfies the bound constraints.

Definition 3 (Bound Consistency) *A constraint $\mathcal{C}$ over two set variables S_1 and S_2 with respective domains $X = \langle m_X, M_X\rangle$ and $Y = \langle m_Y, M_Y\rangle$ is bound-consistent if*

$$\exists y \in Y : \mathcal{C}(m_X, y) \quad \wedge \quad \exists y \in Y : \mathcal{C}(M_X, y) \quad \wedge$$
$$\exists x \in X : \mathcal{C}(x, m_Y) \quad \wedge \quad \exists x \in X : \mathcal{C}(x, M_Y).$$

A bound-consistency algorithm (see Algorithm 1) determines if the constraint is consistent with some values in the domains and finds the first (resp. last) values in the domains for which there exists a solution. Our bound-consistency algorithms are expressed in terms of three functions hs, $succ$, and $pred$ for feasibility checking and updating the bounds. Only the first two are discussed in this paper, the predecessor computation being essentially similar to the successor function. They are specified as follows for the left variable of the constraint (interval X). The right variable is symmetric.

Specification 1 (hs) *Given a constraint $\mathcal{C}$ and length-lex intervals X and Y, $hs\langle\mathcal{C}\rangle(X, Y) \equiv \exists x \in X, y \in Y : \mathcal{C}(x, y)$.*

Specification 2 ($succ_X$) *For a constraint $\mathcal{C}$ and two length-lex intervals X and Y such that $hs\langle\mathcal{C}\rangle(X, Y)$, function $succ_X\langle\mathcal{C}\rangle(X, Y)$ returns the smallest set $x \in X$ in the $\preceq$ ordering which belongs to a solution of the constraint, i.e.,*

$$succ_X\langle\mathcal{C}\rangle(X, Y) \equiv \min_{\preceq}\{x \in X \big| \exists y \in Y : \mathcal{C}(x, y)\}.$$

Algorithm 1 bc$\langle\mathcal{C}\rangle(X = \langle m_X, M_X\rangle, Y = \langle m_Y, M_Y\rangle)$

```
1: if $hs\langle\mathcal{C}\rangle(X, Y)$ then
2:     $m_X \leftarrow succ_X\langle\mathcal{C}\rangle(X, Y)$
3:     $M_X \leftarrow pred_X\langle\mathcal{C}\rangle(X, Y)$
4:     $m_Y \leftarrow succ_Y\langle\mathcal{C}\rangle(Y, X)$
5:     $M_Y \leftarrow pred_Y\langle\mathcal{C}\rangle(Y, X)$
6:     return $true$
7: else
8:     return $false$
```

Specification 3 ($pred_X$) *For a constraint $\mathcal{C}$ and two length-lex intervals X and Y such as $hs\langle\mathcal{C}\rangle(X, Y)$, function $pred\langle\mathcal{C}\rangle(X, Y)$ returns the largest set $x \in X$ in the $\preceq$ ordering which belongs to a solution of the constraint, i.e.,*

$$pred_X\langle\mathcal{C}\rangle(X, Y) \equiv \max_{\preceq}\{x \in X \big| \exists y \in Y : \mathcal{C}(x, y)\}.$$

Overview of the Generic Successor Algorithm

This paper first presents a generic bound-consistency algorithm for binary set constraints. The algorithm only relies on the implementation of function hs, the functions $succ$ and $pred$ being implemented generically in terms of hs. In other words, it only relies on a function checking the existence of a solution. We show later in the paper that, for some specific constraints, the time complexity of the generic algorithm can be improved by providing dedicated implementations of $succ$ and $pred$.

To ease understanding, it is useful to give a high-level overview of the structure of the algorithm. They key step is to partition the length-lex interval into so-called PF-closed set intervals which greatly simplify the design of the algorithm. Informally speaking, a PF-closed set interval consists of four parts: a prefix P, a set F, a sub-universe V, and a cardinality c. Such a PF-closed set interval $\langle P, F, V, c\rangle$ denotes all the c-sets starting with prefix P, following by at least one element of F, and taking their remaining values in V. These PF-closed intervals are attractive since they enjoy some compactness properties that simplify the inferences.

The bound-consistency algorithm thus partitions the domains X and Y of the variables in sequences of PF-closed set intervals $[X_1, \ldots, X_k]$ and $[Y_1, \ldots, Y_l]$. To find, say, a new lower bound for X, the algorithm considers the PF-closed set intervals X_i in sequence until a new bound is found. For a specific X_i, the algorithm finds the lower bound with respect to each Y_j and selects the smallest one in the length-lex ordering if one exists. Otherwise, the algorithm moves to X_{i+1}.

The core of the generic algorithm thus consists of applying bound consistency on domains which are PF-closed set intervals. This step can be decomposed into a number of feasibility checks and hence the generic algorithm only relies over the existence of a function $hs\langle\mathcal{C}\rangle$ over PF-closed set intervals. If $hs\langle\mathcal{C}\rangle$ runs in time α, the complexity of the generic algorithm is $O(\alpha c^2 \log |U|)$, the decomposition step generating at most $O(c)$ PF-closed set intervals. We now go into the detail of the algorithm.

$$\begin{tikzcd}
 & \langle\{1,2,5,6\},\{4,6,7,8\}\rangle \arrow[dl] \arrow[d] \arrow[dr] & \\
H{:}\langle\{1,2,5,6\},\{1,6,7,8\}\rangle \arrow[d] \arrow[dr] \arrow[drr] & B{:}\langle\{2,3,4,5\},\{4,6,7,8\}\rangle & T{:}\langle\rangle \\
H'{:}\langle\{1,2,5,6\},\{1,2,7,8\}\rangle & B'{:}\langle\{1,3,4,5\},\{1,6,7,8\}\rangle & T'{:}\langle\rangle
\end{tikzcd}$$

Figure 1: Illustrating the Decomposition.

PF-Closed Intervals

We now specify PF-closed intervals formally

Definition 4 *Let P, F and V be sets and c, an integer. A PF-closed interval is a 4-tuple $pf\langle P, F, V, c\rangle$ satisfying*

$$F \subseteq V \wedge \forall e \in V \setminus F : e > \max(F) \wedge$$
$$|V \setminus F| \geq c - |P| - 1 \wedge \max(P) < \min(F)$$

and denoting all sets

$$\big\{P \cup \{f\} \cup s \big| f \in F \wedge s \subseteq V \wedge |P \cup \{f\} \cup s| = c\big\}.$$

A key property of a PF-closed interval is that it contains all the c-sets starting with P, taking at least an element in F, and its other elements in V. In the following, we use X_{pf} to denote the PF-closed interval $pf\langle P_X, F_X, V_X, c_X\rangle$.

Example 3 (PF-closed interval) *Consider the length-lex interval $\langle\{1,3,4\},\{1,5,8\}\rangle$ and universe $\{1,..,8\}$. It is equivalent to the PF-closed interval $pf\langle\{1\},\{3,4,5\},\{3,..,8\},3\rangle$ which contains all sets $\{e_1, e_2, e_3\}$ with $e_1 = 1$, $e_2 \in \{3,4,5\}$, and $e_3 > e_2 \wedge e_3 \in \{3,..,8\}$.*

Example 4 (Counter-example) *Consider the length-lex interval $\langle\{1,2,5,6\},\{1,6,7,8\}\rangle$ and universe $\{1,..,8\}$. Its denotation cannot be captured by a PF-closed interval. Indeed, it it does not contain all sets with a second element in $\{2,..,6\}$, since $\{1,2,3,4\}$ is not in the length-lex interval.*

We now describe how to partition a length-lex interval into a minimal set of PF-closed intervals.

Domain Decomposition

Let X be a length-lex interval $\langle m, M\rangle$. The decomposition first partitions X into a head H, a body B, and a tail T. The body, if it exists, is guaranteed to be a PF-closed interval. The head, if it exists, is a length-lex interval containing all c-sets in X beginning with m_1, while the tail, if it exists, is the length-lex interval containing all c-sets in X beginning with M_1. The head and the tail are not guaranteed to be PF-closed intervals, in which case the decomposition is applied recursively. The decomposition is described visually in Figure 1 and Example 5. Observe that the body has a lower bound which is the smallest set starting with 2. The recursive decompositions of the head always produce empty tails, which is critical for the complexity and the size of the decomposition. Indeed, the head has an upper bound which is the largest element starting with 1. So, when we decompose the head H further, the tail T' is empty.

More formally, the decomposition algorithm takes a length-lex interval $\langle m, M\rangle$ in a universe $U = \{1,..,n\}$ and returns its minimal partition into PF-closed intervals. Since the decomposition is recursive, the specification needs to include a prefix set P which is initially empty. The algorithm also receives the integer n to represent the universe and uses :: to denote the concatenation of two sequences and ϵ to denote an empty sequence.

Specification 4 *Given universe $U = \{1,..,n\}$, Algorithm $decomp(m, M, P, n)$ returns an ordered sequence $[X^1_{pf}, \cdots, X^w_{pf}]$ of PF-closed intervals satisfying*

$$\biguplus_{i\in[1,..,w]} X^i_{pf} = \langle P \uplus m, P \uplus M\rangle$$

and $\forall i < j \in [1,..,w] : \forall s \in X^i_{pf}, t \in X^j_{pf} : s \prec t.$

Algorithm 2 $decomp(m, M, P, n)$

```
1: $c \leftarrow |m|$
2: $H, B, T \leftarrow \epsilon, \epsilon, \epsilon$
3: $h, t \leftarrow m_1, M_1$
4: if $h = t$ then
5:     return $decomp(m_{2..c}, M_{2..c}, P \cup \{h\}, n)$
6: if $m \neq \{m_1, m_1 + 1, .., m_1 + c - 1\}$ then
7:     $H \leftarrow decomp(m_{2..c}, \{n - c + 2, .., n\}, P \cup \{m_1\}, n)$
8:     $h \leftarrow h + 1$
9: if $M \neq \{M_1, n - c + 2, .., n\}$ then
10:    $T \leftarrow decomp(\{M_1 + 1, .., M_1 + c - 1\}, M_{2..c}, P \cup \{M_1\}, n)$
11:    $t \leftarrow t - 1$
12: if $h \leq t$ then
13:    $B \leftarrow [pf\langle P, \{h, .., t\}, \{h, .., n\}, c + |P|\rangle]$
14: return $H :: B :: T$
```

The algorithm is depicted in Algorithm 2. Lines 4–5 create factorize the common prefixes. Lines 6–8 create a head if necessary, i.e., if m is not minimal. Lines 9–11 creates a tail if M is not maximal. Line 12–13 create the body if necessary (e.g., $\langle\{1,2,5,6\},\{2,5,7,8\}\rangle$ has no body) and line 14 returns the partition. These two recursive calls in lines 7 and 10 increment the size of the prefix. The first in line 7 now has a maximal second argument compatible with the prefix, while the recursive call in line 10 has a minimal first argument compatible with its prefix.

Example 5 (The Decomposition) *We illustrate the algorithm on the length-lex interval $\langle\{1,2,5,6\},\{4,6,7,8\}\rangle$ and universe $U = \{1,..,8\}$. The set $m = \{1,2,5,6\}$ is not minimal ($\{1,2,3,4\}$ would be) and we obtain two disjoint intervals H:$\langle\{1,2,5,6\},\{1,6,7,8\}\rangle$ and B:$\langle\{2,3,4,5\},\{4,6,7,8\}\rangle$. Observe that B contains all sets beginning with either 2, 3, or 4 and it is a PF-closed interval $pf\langle\emptyset,\{2,3,4\},\{2,..,8\},4\rangle$.*

A recursive call is performed on H. The algorithm sets the prefix to $\{1\}$, obtaining a length-lex interval $\langle\{2,5,6\},\{6,7,8\}\rangle$. Observe that $\{6,7,8\}$ is maximal, so subsequent recursive calls do not generate tails. Once again, we obtain two sub-intervals $\langle\{2,5,6\},\{2,7,8\}\rangle$ and $\langle\{3,4,5\},\{6,7,8\}\rangle$ which when added to the current prefix $\{1\}$ form H' and $B' = pf\langle\{1\},\{3,..,6\},\{3,..,8\},4\rangle$.

Since all sets in the first interval begin with 2, the algorithm adds it to the prefix and continues recursively again. Since $\langle\{5,6\},\{7,8\}\rangle$ in addition to prefix $\{1,2\}$ forms the PF-closed interval $pf\langle\{1,2\},\{5,6,7\},\{5,..,8\},4\rangle$ which is equal to H', there is no head and tail in this call and the

Algorithm 3 $succ\langle\mathcal{C}\rangle(\langle m_X, M_X\rangle, \langle m_Y, M_Y\rangle)$

```
1: $[X_{pf}^1, .., X_{pf}^i] \leftarrow decomp(m_X, M_X, \emptyset, n)$
2: $[Y_{pf}^1, .., Y_{pf}^j] \leftarrow decomp(m_Y, M_Y, \emptyset, n)$
3: $m'_X \leftarrow \top$
4: for $X_{pf} = X_{pf}^1$ to $X_{pf}^i$ do
5:     for $Y_{pf} = Y_{pf}^1$ to $Y_{pf}^j$ do
6:         if $hs\langle\mathcal{C}\rangle(X_{pf}, Y_{pf})$ then
7:             $m'_X \leftarrow \min(m'_X, succ\langle\mathcal{C}\rangle(X_{pf}, Y_{pf}))$
8:     if $m'_X \neq \top$ then
9:         return $m'_X$
10: return $\top$
```

algorithm concludes. As a result, $\langle\{1,2,5,6\},\{4,6,7,8\}\rangle$ is partitioned into

$$\langle\{1,2,5,6\},\{1,2,7,8\}\rangle$$
$$\langle\{1,3,4,5\},\{1,6,7,8\}\rangle$$
$$\langle\{2,3,4,5\},\{4,6,7,8\}\rangle$$

giving the PF-closed intervals

$$pf\langle\{1,2\},\{5,6,7\},\{5,..,8\},4\rangle$$
$$pf\langle\{1\},\{3,..,6\},\{3,..,8\},4\rangle$$
$$pf\langle\emptyset,\{2,3,4\},\{2,..,8\},4\rangle.$$

Lemma 1 *Algorithm decomp partitions a length-lex interval of c-sets into $O(c)$ PF-closed intervals and takes $O(c^2)$ time.*

Proof: (sketch) After the first call of Algorithm 2, the head H is subsequently decomposed only into heads and bodies (no tails) and the tail is subsequently decomposed only in bodies and tails (no heads). Hence each call will only make one additional PF-closed interval and the depth of recursion can be at most c since the prefix length is incremented in each recursive call. Hence the maximum number of calls and PF-closed intervals is $2c-1$, which is $O(c)$. For each of those calls, the comparisons in lines 4 and 7 take $O(c)$ time and the total time complexity is $O(c^2)$. □

Note that the sets generated from the *decomp* algorithm satisfy the conditions of PF-closed intervals. These properties always hold in the remainder of this paper as well. When PF-closed intervals are created, it will always be by removing the same set from the F and V parts of a PF-closed interval, an operation which obviously preserves the properties.

Generic Successor Algorithm

We now turn to the generic successor algorithm for finding a new lower bound which is specified in Specification 2 and depicted in Algorithm 3. The algorithm takes a binary constraint and the length-lex intervalsX_{ll} and Y_{ll} over universe $U = \{1,..,n\}$ as inputs. Recall that its goal is to find a new lower bound for X_{ll}. The algorithm first partitions each length-lex intervals into minimal sequences of PF-closed intervals $\mathcal{X}$ and $\mathcal{Y}$ (line 1..2). It then seeks, for each of PF-closed interval $X_{pf} \in \mathcal{X}$, the first set having a support in $\mathcal{Y}$. The algorithm returns the smallest of those sets as the new lower bound for X.

The successor algorithm on length-lex intervals uses the feasibility algorithm $hs\langle\mathcal{C}\rangle(X_{pf}, Y_{pf})$ on PF-closed intervals, as well as the successor algorithm $succ\langle\mathcal{C}\rangle(X_{pf}, Y_{pf})$ on PF-closed intervals. The feasibility algorithm must be provided for each constraint. The successor algorithm $succ\langle\mathcal{C}\rangle(X_{pf}, Y_{pf})$ on PF-closed intervals can be defined generically in terms of the feasibility algorithm.

The implementation of $succ\langle\mathcal{C}\rangle(X_{pf}, Y_{pf})$ is shown in Algorithm 4 and returns the set s satisfying the specification. The algorithm first assigns the prefix set to the beginning of s (line 1) and then iterates over the remaining positions in the set s (lines 3–6). The first execution of line 4 selects the element of F_X, while subsequent ones take the value from V_X. The selection of the value f in line 4 can be obtained by a dichotomic search on the range $[l, h]$, calling $hs\langle\mathcal{C}\rangle$ at most $O(\log(h-l+1))$ times. Line 5 ensures that the selected values are increasing, while line 6 allows values of V_X to be selected.

Algorithm 4 $succ\langle\mathcal{C}\rangle(X_{pf} = pf\langle P_X, F_X, V_X, c_X\rangle, Y_{pf})$

```
1: $s_{1..|P_X|} \leftarrow P_X$
2: $l, h = \min(F_X), \max(F_X)$
3: for $i = |P_X| + 1$ to $c_X$ do
4:     $s_i \leftarrow \min\{l \le f \le h | hs\langle\mathcal{C}\rangle(pf\langle s_{1..i-1}, \{f\}, \{e \in V_X | e \ge f\}, c\rangle, Y_{pf})\}$
5:     $l \leftarrow s_i + 1$
6:     $h \leftarrow \max(V_X)$
7: return $s$
```

We illustrate the algorithm on the binary disjoint constraint $\mathcal{D}(s,t) \equiv s \cap t = \emptyset$.

Example 6 ($succ\langle\mathcal{C}\rangle(X_{ll}, Y_{ll})$) *Consider the binary disjoint constraint over the length-lex intervals$X_{ll} = \langle\{1,2,5\},\{4,6,7\}\rangle$, $Y_{ll} = \langle\{1,2,3\},\{2,4,7\}\rangle$. The decomposition yields the sequences*

$$X_{pf}^1 = pf\langle\{1,2\},\{5,6,7\},\{5,..,7\},3\rangle,$$
$$X_{pf}^2 = pf\langle\{1\},\{3,..,6\},\{3,..,7\},3\rangle,$$
$$X_{pf}^3 = pf\langle\emptyset,\{2,3,4\},\{2,..,7\},3\rangle$$

and

$$Y_{pf}^1 = pf\langle\emptyset,\{1\},\{1,..,7\},3\rangle$$
$$Y_{pf}^2 = pf\langle\{2\},\{3,4\},\{3,..,7\},3\rangle.$$

Lines 4-9 in Algorithm 3 find the first PF-closed interval in $[X_{pf}^1, X_{pf}^2, X_{pf}^3]$ with a support in Y_{pf}^1 or Y_{pf}^2 (if it exists). The first PF-closed interval is X_{pf}^1 ($pf\langle\{1,2\},\{5,6,7\},\{5,6,7\},3\rangle$) which is tested for feasibility with the Y intervals. By just considering the prefix of X_{pf}^1, it can be seen that X_{pf}^1 has no support in the Y intervals which contain either 1 or 2 and the new lower bound of X will be greater than any set in X_{pf}^1. The algorithm then considers X_{pf}^2. There is no solution with Y_{pf}^1 since both intervals always contain 1. The condition $hs\langle\mathcal{D}\rangle(X_{pf}^2, Y_{pf}^2)$ holds and hence there exists a supported value in X_{pf}^2. Line 7 invokes the succ algorithm applied to two PF-closed intervals, which will return the first successor. This last computation is discussed in the next example.

Example 7 ($succ\langle\mathcal{C}\rangle(X_{pf}, Y_{pf})$) *Consider a call $succ\langle\mathcal{D}\rangle(X_{pf}, Y_{pf})$ to Algorithm 4 with $X_{pf} = \langle\{1\}, \{3, .., 6\}, \{3, .., 7\}, 3\rangle$ and $Y_{pf} = \langle\{2\}, \{3, 4\}, \{3, .., 7\}, 3\rangle$. Since all sets in X_{pf} begin with P_X, it is also a prefix of s and line 1 executes $s_{1..|P_X|} \leftarrow P_X$. As a result, we have $s_1 = 1$, i starts at 2, $l = 3$, and $h = 6$. On line 4, the algorithm sets s_2 to 3 since there exists a set starting with $\{1, 3\}$ compatible with Y_{pf} (for instance {1,3,6} and {2,4,7}). On line 5, the l is set to 4 and the algorithm searches for values not less than 4 in the next iteration. On line 6, h is set to 7 for all subsequent iterations. In the last iteration 5 is found on line 4 and the set $\{1, 3, 5\}$ is returned.*

We prove several results on the generic successor algorithm.

Lemma 2 *Algorithm 4 satisfies the Specification 2 when the two arguments are both PF-closed intervals.*

Proof: The algorithm constructs the value s from left to right. At each step (line 4), it inserts the minimum consistent value which will eventually lead to a solution. If there would exist another solution s' with $s' \prec s$, those should differ at some index i and $s'_i < s_i$. But this contradicts the fact that s_i is assigned to the smallest value which leads to a solution. $\square$

Lemma 3 *Assume that algorithm $hs\langle\mathcal{C}\rangle(X_{pf}, Y_{pf})$ takes time $O(\alpha)$. Then, Algorithm 4, $succ\langle\mathcal{C}\rangle(X_{pf}, Y_{pf})$, takes time $O(\alpha c \log n)$.*

Proof: The for-loop in line 3 iterates at most from 1 to c_X. In each iteration, the search in line 4 uses a binary search, giving a time complexity of $O(\alpha c \log n)$. $\square$

Lemma 4 *Algorithm 3, $succ\langle\mathcal{C}\rangle(X_{ll}, Y_{ll})$, takes time $O(\alpha c^2 \log n)$.*

Proof: By lemma 1, lines 1-2 take $O(c^2)$ and the number of PF-closed intervals in the partitions is $O(c)$. Hence the total complexity of the calls to $hs\langle\mathcal{C}\rangle$ in line 6 is $O(\alpha c^2)$. Moreover, because of lines 8–9, there can be at most c calls to Algorithm 4 on line 7, which is $O(\alpha c \log n)$. The overall complexity is thus $O(\alpha c^2 \log n)$. $\square$

Consider now the case in which there exists a dedicated implementation of $succ\langle\mathcal{C}\rangle(X_{pf}, Y_{pf})$ for $\mathcal{C}$ and assume that $succ\langle\mathcal{C}\rangle(X_{pf}, Y_{pf})$ takes time $O(\beta)$. Then, Algorithm 3 takes $O(\alpha c^2 + \beta c)$. The next sections study specific implementations of $hs\langle\mathcal{C}\rangle(X_{pf}, Y_{pf})$ and $succ\langle\mathcal{C}\rangle(X_{pf}, Y_{pf})$.

Binary Disjoint Constraint

Consider the binary disjoint constraint $\mathcal{D}(s, t) \equiv s \cap t = \emptyset$. We give the feasibility check $hs\langle\mathcal{D}\rangle(X_{pf}, Y_{pf})$ and a specialized implementation of $succ\langle\mathcal{D}\rangle(X_{pf}, Y_{pf})$. Algorithms for cardinality constraints such as atmost-k or atleast-k are similar but omitted for space reasons.

Feasibility Check

The feasibility check $hs\langle\mathcal{D}\rangle(X_{pf}, Y_{pf})$ is defined for two PF-closed intervals : $X_{pf} = pf\langle P_X, F_X, V_X, c_X\rangle$ and $Y_{pf} = pf\langle P_Y, F_Y, V_Y, c_Y\rangle$. For simplifying the presentation, we first assume the prefixes P_X and P_Y are empty and introduce a shorthand notation for PF-closed intervals with an empty prefix: A F-closed interval is a 3-tuple $f\langle F, V, c\rangle$ denoting the interval $pf\langle\emptyset, F, V, c\rangle$.

Consider $hs\langle\mathcal{D}\rangle(X_f, Y_f)$ defined for two F-closed intervals : $X_f = f\langle F_X, V_X, c_X\rangle$ and $Y_f = f\langle F_Y, V_Y, c_Y\rangle$. In other words, the goal is to find two sets $s \in X_f$ and $t \in Y_f$ such that $s \cap t = \emptyset$. To ensure feasibility, there must exist enough elements for s and t, i.e., $c_X + c_Y \leq |U_X \cup U_Y|$. Moreover, at least one element of F_X and F_Y must be taken by s and t respectively. We have the following conditions: (1) the union of F_X and F_Y must contain at least 2 elements since each set must take at least one (distinct from the other), (2) we denote $F_{\overline{Y}} = F_Y \setminus V_X$ and symmetrically $F_{\overline{X}} = F_X \setminus V_Y$. If X_f is a singleton, then $F_{\overline{Y}}$ cannot be empty ($c_X = |V_X| \Rightarrow F_{\overline{Y}} \neq \emptyset$) . All of these conditions lead to the following feasibility function hs of the binary disjoint over two F-closed intervals.

Algorithm 5 $hs\langle\mathcal{D}\rangle(X_f, Y_f)$

```
1: return $c_X + c_Y \leq |V_X \cup V_Y| \wedge |F_X \cup F_Y| \geq 2 \wedge$
          $c_X = |V_X| \Rightarrow F_{\overline{Y}} \neq \emptyset \wedge c_Y = |V_Y| \Rightarrow F_{\overline{X}} \neq \emptyset$
```

Lemma 5 *Algorithm 5 implements Specification 1 when the two arguments are F-closed intervals.*

Proof: $\Rightarrow$: If a pair of disjoint sets exists in X_f and Y_f, it clearly satisfies the conditions.
$\Leftarrow$:We construct a pair of disjoint sets ($s \in X_f$,$t \in Y_f$). Note first that, since all elements from F_X are smaller than the elements from $V_X \setminus F_X$, we first ensure that s and t contain at least one element from F_X and F_Y respectively. Then we add more elements to reach the right cardinality. There are four cases regarding $F_{\overline{X}}$ and $F_{\overline{Y}}$:

1. $F_{\overline{X}} \neq \emptyset$ and $F_{\overline{Y}} \neq \emptyset$. This is the easiest case: Just pick $\min(F_{\overline{X}})$ for s and $\min(F_{\overline{Y}})$ for t.
2. $F_{\overline{X}} \neq \emptyset$ and $F_{\overline{Y}} = \emptyset$. Pick $\min(F_{\overline{X}})$ for s but for t we must pick an element which could be used by s as well. However by $c_X = |V_X| \Rightarrow F_{\overline{Y}} \neq \emptyset$, we can assign an element from $F_Y \cap V_X$ to t and build a disjoint s since $c_X < |V_X|$.
3. The case $F_{\overline{X}} = \emptyset$ and $F_{\overline{Y}} \neq \emptyset$ is symmetric.
4. Finally, when both sets $F_{\overline{X}}$ and $F_{\overline{Y}}$ are empty, by $|F_X \cup F_Y| > 2$, we can pick two disjoint elements for s and t.

Now, the F constraints are solved, s and t each contain one element and two elements have been consumed from $V_X \cup V_Y$. We fill s and t to their respective cardinality by inserting $c_X - 1$ elements in s and $c_Y - 1$ elements in t. We can do so since $|V_Y \cup V_X| - 2 \geq c_X + c_Y - 2$. $\square$

The disjoint feasibility function over PF-closed intervals builds on Algorithm 5 over F-closed intervals. It further takes into account the prefix sets P_X and P_Y.

Algorithm 6 $hs\langle\mathcal{D}\rangle(X_{pf}, Y_{pf})$

```
1: return $(P_X \cap P_Y = \emptyset) \wedge hs\langle\mathcal{D}\rangle(f\langle F_X \setminus P_Y, V_X \setminus$
     $P_Y, c_X - |P_X|\rangle, f\langle F_Y \setminus P_X, V_Y \setminus P_X, c_Y - |P_Y|\rangle)$
```

Lemma 6 *Algorithm 6 implements Specification 1 when two arguments are PF-closed intervals.*

Proof: (sketch) We construct a pair of sets $s \in X_{pf}$ and $t \in Y_{pf}$ s.t. $s \cap t = \emptyset$. Clearly P_X must be disjoint from P_Y. Since s should be disjoint from P_Y, s must take its value in $V_X \setminus P_Y$. Similarly for t. The feasibility check is now applied to two F-closed intervals (see conditions above). Note that by removing the same prefix P_X from F_Y and V_Y, we satisfy the properties of PF-closed intervals. $\Box$

Lemma 7 *Algorithm 6 takes $O(c)$.*

Proof: (sketch) The proof relies on the fact that the sets manipulated by this algorithm can be represented by $O(c)$ intervals. First, the P-sets contain at most c elements. Then the sets F and V generated by the *decomp* algorithm are just one interval (see line 11). Algorithm 6 removes a set P from those intervals F and V leading to at most $c + 1$ intervals. Finally, Algorithm 5 computes the intersection and union of these sets, operations taking $O(c)$. $\Box$

Specialized $succ\langle\mathcal{D}\rangle$

We now present a $succ$ algorithm for the binary disjoint constraint over PF-closed intervals. Like for the hs algorithm, the algorithm removes the prefixes and calls a F-closed intervals version of the algorithm. Algorithm 7 works by building a set s one element at a time from index 1 to index c_X (lines 2–10). All elements which are not used by s can be used by t (line 9). The smallest available element cur (lines 1, 5, and 10) is typically added to set s. However, two special cases must be recognized. First, if the set t is empty and cur is the last element in F_Y (lines 3–5), then cur must be assigned to t and cannot be assigned to s (line 5). Second, if the set t must take all its available elements (line 6), then we simply assign to s the smallest elements remaining (line 7). Note that, when elements are skipped in s, they are assigned to t (line 9), allowing as much freedom for s as possible.

Lemma 8 *Algorithm 7 implements Specification 2 when the two arguments are F-closed intervals.*

Proof: Omitted due to lack of space.

Lemma 9 *Algorithm 8 $succ\langle\mathcal{D}\rangle(X_{pf}, Y_{pf})$ takes $O(c)$.*

Proof: (sketch) As in the previous proof, the sets involved are composed of $O(c)$ intervals. Now, in algorithm 8, in lines 3, 6 and 10, we can compute the required values by storing the current position in V_X and V_Y and performing only one pass from left to right over these sets during the whole loop of line 2. It is similar in line 7, in which we can compute the quantity $|\{e \in V_Y | e > s_{i-1}\}|$ by computing $|V_Y|$ beforehand and then remove the size of the intervals we need to skip as i grows and s_{i-1} becomes known.

Algorithm 7 $succ\langle\mathcal{D}\rangle(X_f, Y_f)$

```
Assume: $s_0 = -\infty$
1:  $cur \leftarrow \min(F_X)$
2:  for $i$ = 1 to $c_X$ do
3:      $t \leftarrow \{e \in V_Y \setminus s | e < cur\}_{1..c_X}$
4:      if $t = \emptyset \wedge cur = \max(F_Y)$ then
5:          $t \leftarrow \{cur\}$
6:          $cur \leftarrow \min\{e \in V_X | e > cur\}$
7:      if $|t \uplus \{e \in V_Y | e > s_{i-1}\}| = c_Y$ then
8:          return $(s \uplus \{e \in V_X \setminus V_Y | e \geq cur\})_{1..c_X}$
9:      $s_i \leftarrow cur$
10:     $cur \leftarrow \min\{e \in V_X | e > cur\}$
11: return $s_{1..c_X}$
```

Algorithm 8 $succ\langle\mathcal{D}\rangle(X_{pf}, Y_{pf})$

```
return $P_X \uplus succ\langle\mathcal{D}\rangle(f\langle F_X \setminus P_Y, V_X \setminus P_Y, c_X - |P_X|\rangle, f\langle F_Y \setminus P_X, V_Y \setminus P_X, c_Y - |P_Y|\rangle)$
```

Conclusion

Gervet and Van Hentenryck (2006) introduced the length-lex representation for sets and showed that bound consistency can be enforced in $\tilde{O}(n)$ for unary constraints. They left open whether it is possible to achieve bound consistency on binary constraints. This paper answered this question positively and showed how to enforce bound consistency for binary constraints in time $\tilde{O}(c^2\alpha)$, where c is the cardinality of the sets (typically much smaller than $n = |U|$) and α is the complexity of a feasibility subroutine. The paper also presented a specialized algorithm in $O(c^3)$ for the disjoint constraint, which generalizes to atmost-k and atleast-k constraints. The result should extend to constraints of arity k , giving an algorithm which runs in time $\tilde{O}(c^k\alpha)$. It would be interesting to study whether this can be improved by exploiting the constraint semantics. Similarly, the implementation of global constraints and pushing lexicographic constraints within existing constraints are important research directions.

References

Azevedo, F., Barahona, P. 2000. Modelling Digital Circuits Problems with Set Constraints. in *CL*-2000.

Colbourn, C. J. , Dinitz, J.H., Stinson. 1999. Applications of Combinatorial Designs to Communications, Cryptography, and Networking. Cambridge Univ. Press.

Gervet, C. 1997. Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language. In *Constraints journal*, volume 1(3).

Gervet, C., Van Hentenryck, P. 2006. Length-lex Ordering for Set CSPs. In *Proc. AAAI'06*.

Kreher, D.L., Stinson, D.R. 1999. *Combinatorial Algorithms*. The CRC Press.

Puget, J-F. 1992 PECOS a High Level Constraint Programming Language In *Proc. of Spicis*.

Sadler, A., Gervet, C. 2008 Enhancing Set Constraint Solvers with Lexicographic Bounds. *Journal of Heuristics*, volume 14(1).

Van Hentenryck, P. 1989 *Constraint Satisfaction in Logic Programming*. The MIT Press.