Online Appendix 1: Lisp program used in the machine-assisted proof of Lemma 3. of the paper “The Complexity of Reasoning about Spatial Congruence”

In the remainder of the section we list the LISP code of the procedure which has been used to perform the machine-assisted proof of Lemma 3. The method is based on the use of prime numbers as a codification for the basic relations. It is able to compute the transitive closure of any constraint algebra described as a base for the computation.

```
;-------------------------------------------------------------------
; PROCEDURE TO COMPUTE THE TRANSITIVE CLOSURE OF A SET OF RELATIONS
; Base = basic relations of the algebra; Rev = reverse relations
; Trans = transitivity table; Gen = generators
(defun transitive-closure (Base Trans Rev Gen)
  (describe-algebra Base Trans Rev)
  (setq Gen (build-setgen Gen))
  (setq Trans-Clos (closure Gen Gen))
  (setq Trans-Clos (sort Trans-Clos '<))
  (setq Trans-Clos-symbol (build-algebra-symbol Trans-Clos))
Trans-Clos-symbol)
;-------------------------------------------------------------------
; COMPUTATION OF CLOSURE FOR BASIC RELATIONS
(defun describe-algebra (Relations-symbol
                   Transitivity-table-symbol Reversal-table-symbol)
   (setq Cardinality (length Relations-symbol))
   (setq Relations (build-prime-list Cardinality))
   (setq News-relations Relations)
   (setq Correspondences
         (build-correspondences Relations-symbol Relations))
   (setq Transitivity-table
         (build-transitivity-table Transitivity-table-symbol))
   (setq Reversal-table-base
         (build-reversal-table-base Reversal-table-symbol))
   (setq News-relations (set-difference News-relations Relations))
  (setq Total-relations (append Relations News-relations))
  (setq Algebra (closure Total-relations Total-relations))
  (setq Algebra (sort Algebra '<))
   (setq Algebra-symbol (build-algebra-symbol Algebra)) Algebra-symbol)
;-------------------------------------------------------------------
; INTEGER DIVISION
(defun div (a b) (setq ris (floor (/ a b))) ris)
```

```
;-------------------------------------------------------------------
; TESTING OF UNREDUCIBILITY FOR AN INTEGER NUMBER
(defun is-a-prime (N)
  (setq flag 'nil)
  (cond ((= N 1) 'nil)
    ((= N 2) 't)
    ('t (dotimes (i (div N 2))
            (setq j (+ i 1))
            (setq flag (or flag (= (mod N (+ j 1)) 0)))) (not flag))))
;-------------------------------------------------------------------
; BUILDING THE LIST OF FIRST N PRIME NUMBERS
(defun build-prime-list (N)
  (cond ((= N 1) '(2))
    ('t (setq Prime-list '(2 3))
      (dotimes (i (- N 2))
            (setq Prime-list (append Prime-list (cons
            (do ((k (+ (car (last Prime-list)) 2)))
             ((is-a-prime k) k)(setq k (+ k 2))) 'nil)))) Prime-list)))
;-------------------------------------------------------------------
; MAKES A DIRECT CORRESPONDENCE BETWEEN BASIC RELATIONS
; AND PRIME NUMBERS (GOEDELIZATION)
(defun build-correspondences (List1 List2)
   (cond ((equal List1 'nil) 'nil)
     ('t (append (cons (list (car List1) (car List2)) 'nil)
       (build-correspondences (cdr List1) (cdr List2))))))
;-------------------------------------------------------------------
; RETURNS THE PRIME NUMBER CORRESPONDING TO A GIVEN BASIC RELATION
(defun search-correspondence (Elem Thelist)
   (cond ((equal Thelist 'nil) 'nil)
     ((equal Elem (car (car Thelist)))
       (setq Correspondence (car (cdr (car Thelist)))))
     ('t (search-correspondence Elem (cdr Thelist)) Correspondence)))
;-------------------------------------------------------------------
; LINEAR LIST SEARCH
(defun search-elem (Elem Thelist)
   (setq flag 'nil)
   (cond ((equal Thelist 'nil) 'nil)
     ((equal Elem (car Thelist)) (setq flag 't))
     ('t (search-elem Elem (cdr Thelist)))))
```

```
;-------------------------------------------------------------------
; BUILDS THE LIST OF PRIME NUMBERS
; CODIFYING BASIC RELATIONS AND THEIR COMPOSITIONS
(defun build-transitivity-table (Thelist)
   (setq trans-res 1)
   (cond ((equal Thelist 'nil) 'nil)
     ('t (append (cons (list
       (search-correspondence (car (car Thelist)) Correspondences)
       (search-correspondence (cadr (car Thelist)) Correspondences)
       (append (dolist (elem (caddr (car Thelist)))
       (setq corr (search-correspondence elem Correspondences))
       (setq trans-res (* trans-res corr))) trans-res)) 'nil)
       (cond ((not (equal (search-elem trans-res News-relations) 't))
         (setq News-relations (append News-relations (list trans-res)))
         (build-transitivity-table (cdr Thelist)))
         ('t (build-transitivity-table (cdr Thelist))))))))
;-------------------------------------------------------------------
; BUILDS THE LIST OF PRIME NUMBERS CODIFYING THE
; BASIC RELATIONS
; AND THEIR REVERSE
(defun build-reversal-table-base (Thelist)
   (cond ((equal Thelist 'nil) 'nil)
     ('t (append (cons (list (search-correspondence (car (car Thelist))
             Correspondences)
             (search-correspondence (car (cadr (car Thelist)))
             Correspondences))'nil)
       (build-reversal-table-base (cdr Thelist))))))
;-------------------------------------------------------------------
; RETURNS THE PRIME NUMBER OF THE REVERSE OF A GIVE BASIC RELATION
(defun search-reversal (Elem Thelist)
   (cond ((equal Thelist 'nil) 'nil)
     ((equal (car (car Thelist)) Elem) (cadr (car Thelist)))
     ('t (search-reversal Elem (cdr Thelist)))))
;-------------------------------------------------------------------
; RETURNS THE INTEGER NUMBER CODIFYING THE COMPOSITION OF TWO
; BASIC RELATIONS
(defun search-transitivity (Elem1 Elem2 Thelist)
   (cond ((equal Thelist 'nil) 'nil)
     ((and (equal (car (car Thelist)) Elem1)
           (equal (cadr (car Thelist)) Elem2))
           (caddr (car Thelist)))
     ('t (search-transitivity Elem1 Elem2 (cdr Thelist)))))
```

```
;-----------------------------------------------------------------
; RETURNS THE LIST OF DIVIDERS OF A NUMBER AMONG THE PRIME NUMBERS
; ESTABLISHED TO CODIFY BASIC RELATIONS
(defun scan (n)
   (setq res 'nil)
   (dolist (elem Relations)
     (cond ((= (mod n elem) 0)
       (setq res (append (cons elem 'nil) res))))) res)
;-----------------------------------------------------------------
; RETURNS THE INTEGER NUMBER CODIFYING THE INTERSECTION OF
; TWO BASIC RELATIONS
(defun intersect (x y)
   (setq intersect-res (gcd x y)) intersect-res)
;-----------------------------------------------------------------
; RETURNS THE INTEGER NUMBER CODIFYING THE COMPOSITION OF TWO RELATIONS
(defun compose (x y)
   (setq listx (scan x))
   (setq listy (scan y))
   (setq comp-res 1)
   (setq result 1)
   (dolist (elem1 listx)
   (dolist (elem2 listy)
     (setq comp (search-transitivity elem1 elem2 Transitivity-table))
     (setq comp-res (* comp-res comp))))
   (setq listcomp-res (scan comp-res))
   (dolist (elem listcomp-res)
     (setq result (* result elem))) result)
;-----------------------------------------------------------------
; RETURNS THE INTEGER NUMBER CODIFYING THE REVERSAL OF ONE RELATION
(defun reversal (x)
   (setq listx (scan x))
   (setq rev-result 1)
   (dolist (elem listx)
     (setq comp (search-reversal elem Reversal-table-base))
     (setq rev-result (* rev-result comp))) rev-result)
;-----------------------------------------------------------------
; COMPACTS A MULTI-SET IN A SET
(defun compact (Thelist)
   (cond ((equal Thelist 'nil) 'nil)
    ((member (car Thelist) (cdr Thelist) :test 'equal)
             (compact (cdr Thelist)))
     ('t (append (cons (car Thelist) 'nil) (compact (cdr Thelist))))))
```

```
;-----------------------------------------------------------------
; RETURNS THE LIST OF ELEMENTS OF THE ALGEBRA
(defun build-algebra-symbol (Alg)
   (setq Alg-sym 'nil)
   (dolist (e1 Alg)
     (setq list-elem (scan e1))
     (setq Alg-sym (append Alg-sym (cons
              (build-relation-symbol list-elem) 'nil)))) Alg-sym)
;-----------------------------------------------------------------
; COMPUTES THE TRANSITIVE CLOSURE
(defun closure (Old-list New-list)
   (setq Tmp-list 'nil)
   (setq termination 'nil)
   (do ((i)) ((equal termination 't))
     (dolist (elem1 Old-list)
     (dolist (elem2 New-list)
       (setq rel-rev (reversal elem2))
       (cond ((not (equal (search-elem rel-rev Old-list) 't))
       (setq Tmp-list (append Tmp-list (list rel-rev)))))
       (setq rel-int (intersect elem1 elem2))
       (cond ((not (equal (search-elem rel-int Old-list) 't))
       (setq Tmp-list (append Tmp-list (list rel-int)))))
       (setq rel-comp (compose elem1 elem2))
       (cond ((not (equal (search-elem rel-comp Old-list) 't))
       (setq Tmp-list (append Tmp-list (list rel-comp)))))))
   (cond ((or (equal Tmp-list 'nil) (equal Tmp-list '(1)))
       (setq termination 't)
       (setq Alg (compact (append Old-list New-list Tmp-list))))
     ('t (setq Alg (compact (append Old-list New-list Tmp-list)))
       (closure (compact (append Old-list New-list Tmp-list))
       (compact Tmp-list))))) Alg)
;-----------------------------------------------------------------
; RETURNS THE CORRESPONDING RELATION OF A GIVEN PRIME NUMBER
(defun search-correspondence-rev (Elem Thelist)
   (cond ((equal Thelist 'nil) 'nil)
     ((equal Elem (car (cdr (car Thelist))))
       (setq Correspondence-rev (car (car Thelist))))
     ('t (search-correspondence-rev Elem (cdr Thelist))))
Correspondence-rev)
```

```
;-----------------------------------------------------------------
; RETURNS THE CORRESPONDING SET OF RELATIONS GIVEN A SET OF INTEGER
; NUMBERS
(defun build-relation-symbol (Thelist)
   (setq Rel 'nil)
   (dolist (e2 Thelist)
   (setq Rel (append Rel (cons
             (search-correspondence-rev e2 Correspondences) 'nil)))) Rel)
;-----------------------------------------------------------------
; CODIFIES A GIVEN LIST IN CORRESPONDING INTEGER NUMBERS
(defun build-setgen (Thelistgen)
   (setq cod 1)
   (cond ((equal Thelistgen 'nil) 'nil)
     ('t (dolist (e (car Thelistgen))
       (setq corr (search-correspondence e Correspondences))
       (setq cod (* cod corr)))
(append (cons cod 'nil) (build-setgen (cdr Thelistgen))))))
;-----------------------------------------------------------------
```

Note: If you would like to have a copy of the source code, please feel free to contact the author at his e-mail address.