An ACL2 Proof of the Termination of a Routing Algorithm for On-chip Communication Architecture for OC-768 Network Processors

Julien Schmaltz*
Computer Sciences Department,
University of Texas at Austin,
schmaltz@cs.utexas.edu

September 15, 2003

Abstract

This talk presents an ACL2 proof of the termination of a routing algorithm. After a brief presentation of the routing algorithm, I will go through my ACL2 script. I will first present the modeling and the proof for a fixed size system. Then, the algorithm is extended to the unbounded case. Finally, I will come back to the fixed system and show how to use the general proof for this case.

1 Introduction

In [KNDR01], a new architecture for on-chip communications, based on Octagons, is presented. This architecture has "the potential to meet the performance requirements of next-generation optical routers", which are defined under the norm OC-768. (OC-768 is the norm for 40 Gbps Ethernet.) In my talk, I just focus on the routing algorithm presented in [KNDR01]. My idea is just to prove that the algorithm terminates and in a given number of hops for a fixed size architecture and for an unbounded architecture.
My talk is organized as follows. The next section decribes the routing algorithm as it is presented in [KNDR01] and its ACL2 model. Section three, presents the modeling and the proof of the unbounded archtitecture. Section four shows how to use the techique developped in the general proof to prove the termination of the fixed size system in a linear time. Section five presents comments made during and after the talk. Finally, section six concludes the talk.

2 The routing algorithm

2.1 Octagon presentation

Here is how Octagon is presented in [KNDR01]:

[...] The basic element of the architecture presented in [KNDR01] is an octagon (fig. 1). This basic unit consists of eight nodes and twelve bi-directional links. This architecture has the following properties:

*The talk was given on September 10th at the University of Texas during a visit of the author, who is a graduate student of the University Joseph Fourier, Grenoble, France, Julien.Schmaltz@imag.fr

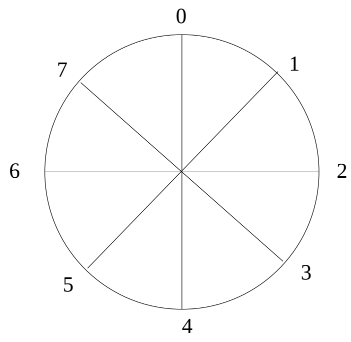

Figure 1: Basic Octagon configuration

1. Communication between any pair can be performed by at most two hops
2. Higher aggregate throughput than a shared bus or crossbar interconnect
3. Simple shortest path algorithm
4. Less wiring compared to that of cross bar interconnect

In my talk, I just focus on 1 and 3.
Octagon can be operated in the packet or circuit switched mode. Define an Octagon Data Unit (ODU) to be the actual data field that is transported from node to node within Octagon. An ODU could be fixed or variable length. In the packet switched mode, ODUs are buffered at intermediate nodes if there is contention at the egress link. In the circuit switched mode, the entire path between source and destination nodes is allocated to a communicating node pair for a number of clock cycles. Non-overlapping communication paths are allowed concurrently [...]
Octagon node addresses can be coded into a three-bit field. Routing of ODU may be accomplished as follows. A three-bit tag is pre-ended to each ODU. Each node compares the tag (ODU_addr) to its own address ($Node\_addr$) to determine the next action to take. Let the relative address of an ODU be computed according to the following equation:

$$Rel\_addr \ = \ (ODU\_addr \ - \ Node\_addr) \ mod \ 8 \tag{1}$$

At each node on the Octagon, routing of ODUs is a function of Rel_addr as follows:

- $Rel\_addr = 0$, process at node
- $Rel\_addr = 1$ or 2, route clockwise
- $Rel\_addr = 6$ or 7, route counter-clockwise
- Route accross otherwise

Consequently, there is a pre-determined simple routing scheme for each ODU in the network. This enables any two nodes in the network to be seperated by at most two hops. [...]

2.2 An ACL2 model

My idea is to prove that this algorithm terminates in at most $\frac{n}{4}$ steps for an unbounded number of nodes. But let start with the Octagon.
The formalization in ACL2 is quite simple. First, I define the functions *clockwise*, *counter-clockwise* and *accross* as follows:

```
(defun clockwise (from)
  (mod (+ 1 from) 8))
```

```
(defun counter-clockwise (from)
  (mod (- from 1) 8))

(defun across (from)
  (mod (+ 4 from) 8))
```

Then, I define the function *go-to-node* representing the routing algorithm. This function builds a "cons" containing the number of each visited node from *from* to *dest*:

```
(defun go-to-node (dest from)
  (declare (xargs :measure (min (nfix (mod (- dest from) 8))
                                (nfix (mod (- from dest) 8)))))
  (cond ((or (not (integerp dest))
             (< dest 0)
             (< 7 dest)
             (not (integerp from))
             (< from 0)
             (< 7 from))
         nil)
        ((equal (- dest from) 0) nil)
        ((or (equal (mod (- dest from) 8) 1)
             (equal (mod (- dest from) 8) 2))
         (cons from (go-to-node dest (clockwise from))))
        ((or (equal (mod (- dest from) 8) 6)
             (equal (mod (- dest from) 8) 7))
         (cons from (go-to-node dest (counter-clockwise from))))
        (t
         (cons from (go-to-node dest (across from))))))
```

But, this function is not admitted because ACL2 cannot prove the measure conjecture.
Mr K... gave me the following lemma that allows this proof:

```
(defthm <-7
  (implies (and (integerp x) (<= 0 x))
           (equal (< 7 x)
                  (and (not (equal x 0))
                       (not (equal x 1))
                       (not (equal x 2))
                       (not (equal x 3))
                       (not (equal x 4))
                       (not (equal x 5))
                       (not (equal x 6))
                       (not (equal x 7))))))
```

Obviously, this forces ACL2 to execute the algorithm for all the values of *from* and *dest*. Then, ACL2 can prove, again by execution, that the maximum number of hops is 2, *i.e.* the length of *go-to-node* is equal or less than 2.

```
(defthm len-go-to-node-<=-2
  (<= (len (go-to-node dest from)) 2))
```

This technique works great for small values of the number of nodes n, and is, in these cases, even better than any other technique presented thereafter! But for huge values the computation does not terminate ! Figure 2 shows the proof time needed to admit the function, *i.e.* to prove it terminates. It also shows the proof time to establish the biggest number of hops, *i.e.* that the shortest is always chosen. "N/A" is put wherever I was not able to submit the event, because the function was not admitted.

number of nodes	proof of termination	shortest path
8	0.5	0.79
12	2.90	2.44
16	6.72	8.09
20	17.77	18.86
100	> 50 minutes	N/A

Figure 2: Results using Mr K's lemma

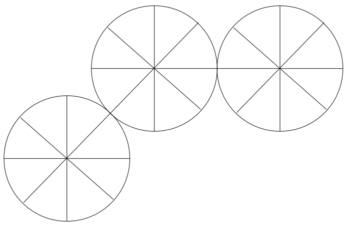

Figure 3: Octagons combination

3 Generalization: Polygons

The proof by execution is great for small values of the inputs. But, the basic Octagon can be combined with other Octagons to build a more complex structure (fig. 3). In that case, the proof by execution can take a long time. My idea is to define a recursive *go-to-node* function and prove its termination (and other properties) by induction.

3.1 The n-nodes routing algorithm

The general architecture of an Octagon, is a polygon with n nodes, where n is striclty positive, and n can be divided by 4. For the general case, the algorithm is the following:

- $Rel\_addr = 0$, process at node
- $0 < Rel\_addr \leq \frac{n}{4}\ mod\ n$, route clockwise
- $n - \frac{n}{4} \leq Rel\_addr < n$, route counter-clockwise
- Route accross otherwise

In the first case, the destination is in the next clockwise quarter, and the shortest way to reach it is to go clockwise. In the second case, the destination is in the counter-clockwise quarter, and the shortest way to reach it is to go counter-clockwise. In the last case, the destination is on the opposite side of the Octagon, and the shortest way is to go across, and then, maybe, to go clock or counter-cloxkwise. I define the functions *n-clockwise*, *n-counter-clockwise* and *n-across* as follows:

```
(defun n-clockwise (from n)
  (mod (+ from 1) n))

(defun n-counter-clockwise (from n)
```

```
  (mod (- from 1) n))

(defun n-across (from n)
  (mod (+ from (/ n 2)) n))
```

Then, I define the function *n-go-to-node* that makes a "cons" containing each visited node:

```
(defun n-go-to-node (dest from n)
  (declare (xargs :measure (min (nfix (mod (- dest from) n))
                                (nfix (mod (- from dest) n)))))
  (cond ((or (not (integerp dest))
             (< dest 0)
             (< (- n 1) dest)
             (not (integerp from))
             (< from 0)
             (< (- n 1) from)
             (not (integerp n))
             (<= n 0)
             (not (equal (mod n 4) 0)))
         nil)
        ((equal (- dest from) 0) nil)
        ((and (< 0 (mod (- dest from) n))
              (<= (mod (- dest from) n) (/ n 4)))
         (cons from
               (n-go-to-node dest (n-clockwise from n) n)))
        ((and (<= (- n (/ n 4)) (mod (- dest from) n))
              (< (mod (- dest from) n) n))
         (cons from
        (n-go-to-node dest (n-counter-clockwise from n) n)))
        (t
         (cons from
               (n-go-to-node dest (n-across from n) n)))))
```

But, ACL2 is not able to admit the function, hence to prove the termination of the routing algorithm. It seems obvious that the algorithm terminates, but the proof involves to reason about *mod* and arithmetic, and this is "never" obvious![1]
To prove the termination, ACL2 generates four cases:

1. the measure must be an ϵ_0 ordinal
2. the measure must decrease if route clockwise
3. the measure must decrease if route counter-clockwise
4. the measure must decrease if route across

The first case can be established by ACL2, but three remains. In the next subsections, I detail how I "kill" each case, with a great help from Robert, our arithmetic expert :)

3.2 Subgoal 3

Before trying anything I load Robert's book on arithmetic and activate non linear arithmetic:

```
(include-book "../../books/Arithmetic-4/bind-free/top")
(set-non-linearp t)
(include-book "../../books/Arithmetic-4/floor-mod/floor-mod")
```

[1]We can redefine our function without using mod, but just with equalities and inequalities. In this case, the proof of termination can be done by linear arithmetic, but lemmas about mod would allow to "automatically" get rid of mod in any function.

The main subgoal is:

```
Subgoal 3
(IMPLIES (AND (NOT (OR (NOT (INTEGERP DEST))
                       (< DEST 0)
                       (< (+ -1 N) DEST)
                       (NOT (INTEGERP FROM))
                       (< FROM 0)
                       (< (+ -1 N) FROM)
                       (NOT (INTEGERP N))
                       (<= N 0)
                       (NOT (EQUAL (MOD N 4) 0))))
              (NOT (EQUAL (+ DEST (- FROM)) 0))
              (< 0 (MOD (+ DEST (- FROM)) N))
              (<= (MOD (+ DEST (- FROM)) N)
                  (* N 1/4)))
         (E0-ORD-< (MIN (NFIX (MOD (+ DEST (- (MOD (+ FROM 1) N))) N))
                        (NFIX (MOD (+ (MOD (+ FROM 1) N) (- DEST))
                                   N)))
                   (MIN (NFIX (MOD (+ DEST (- FROM)) N))
                        (NFIX (MOD (+ FROM (- DEST)) N)))))
```

and is divided into four children.
The first one, *subgoal-3.4* is

```
  (IMPLIES (AND (INTEGERP DEST)
                (<= 0 DEST)
                (<= DEST (+ -1 N))
                (INTEGERP FROM)
                (<= 0 FROM)
                (<= FROM (+ -1 N))
                (INTEGERP N)
                (< 0 N)
                (INTEGERP (* N 1/4))
                (NOT (EQUAL DEST FROM))
                (NOT (INTEGERP (* (+ DEST (- FROM)) (/ N))))
                (NOT (INTEGERP (+ (* DEST (/ N)) (- (* FROM (/ N))))))
                (< (MOD (+ -1 DEST (- FROM)) N) ; h1
                   (MOD (+ 1 (- DEST) FROM) N))
                (< (MOD (+ DEST (- FROM)) N)    ; h2
                   (MOD (+ (- DEST) FROM) N)))
            (< (MOD (+ -1 DEST (- FROM)) N)
               (MOD (+ DEST (- FROM)) n)))
```

which seems to be true, under some conditions that are stated by the two last hypotheses. In fact, the general theorem corresponding to this subgoal is to prove that:

$$(x \; - \; 1) \; mod \; n \; = \; x \; mod \; n \qquad (2)$$

if $(abs\ x) < \frac{n}{2}$, and $x = dest - from$. From hypotheses h_1 and h_2, we know that $x - 1$ and x are in the range of the modulo. To get rid of mod in this case, I prove the following lemmas[2]:

```
(defthm mod-x-=-x
  (implies (and (rationalp x)
                (integerp n)
```

[2]The order of the hypotheses is not "smartly" chosen, but the time is so small that it does not really matter

```
                (<= 0 x)
                (< 0 n)
                (< x n))
           (equal (mod x n) x)))


(defthm mod-x-=-x-+-n
  (implies (and (rationalp x)
                (< x 0)
                (< (- x) n)
                (integerp n)
                (< 0 n))
           (equal (mod x n)
                  (+ x n)))
  :hints (("GOAL" :in-theory (enable mod))))
```

Now, I want ACL2 to use these lemmas. To do that, I need a case split: one case assuming $dest - from$ is positive, another one assuming $dest - from$ is negative. To force ACL2 to case split, I prove the following

```
(defthm force-case-split
  (implies (and (integerp dest)
                (integerp from)
                (<= 0 dest)
                (<= 0 from))
            (iff (not (equal dest from))
                  (or (< (+ dest (- from)) 0)
                      (< 0 (+ dest (- from))))))
  :rule-classes nil)
```

and tells ACL2, through a hint to use it. But it does not work. (WHY?) I make another case split, on whether $from$ is equal to or stricly less than $n - 1$. I do the same with $dest$. So, I prove:

```
(defthm <=-=-<-or-=
  (implies (and (acl2-numberp a)
                (acl2-numberp b)
                )
           (equal (<= a b)
                  (or (< a b)
                      (= a b))))
  :rule-classes nil)
```

and instantiate it properly. I submit the following event:

```
(defthm Subgoal-3.4
  (IMPLIES (AND (INTEGERP DEST)
                (<= 0 DEST)
                (<= DEST (+ -1 N))
                (INTEGERP FROM)
                (<= 0 FROM)
                (<= FROM (+ -1 N))
               (INTEGERP N)
                (< 0 N)
                (INTEGERP (* N 1/4))
                (NOT (EQUAL DEST FROM))
                (NOT (INTEGERP (* (+ DEST (- FROM)) (/ N))))
```

```
                (NOT (INTEGERP (+ (* DEST (/ N)) (- (* FROM (/ N))))))
                (< (MOD (+ -1 DEST (- FROM)) N)
                   (MOD (+ 1 (- DEST) FROM) N))
                (< (MOD (+ DEST (- FROM)) N)
                   (MOD (+ (- DEST) FROM) N)))
            (< (MOD (+ -1 DEST (- FROM)) N)
               (MOD (+ DEST (- FROM)) n)))
  :hints (("GOAL" :use ((:instance force-case-split)
                        (:instance <=-=-<-or-= (a dest) (b (+ -1 n)))
                        (:instance <=-=-<-or-= (a from) (b (+ -1 n)))))))
```

and it works in 2.10 seconds on a 2 733 MHz Pentium 3 Linux station. Then, I apply the same strategy to all the children of subgoal 3 and it works:

```
(defthm subgoal-3.3
  (IMPLIES (AND (INTEGERP DEST)
                (<= 0 DEST)
                (<= DEST (+ -1 N))
                (INTEGERP FROM)
                (<= 0 FROM)
                (<= FROM (+ -1 N))
                (INTEGERP N)
                (INTEGERP (* 1/4 N))
                (NOT (EQUAL DEST FROM))
                (NOT (INTEGERP (+ (* DEST (/ N)) (- (* FROM (/ N))))))
                (<= (MOD (+ DEST (- FROM)) N) (* 1/4 N))
                (< (MOD (+ -1 DEST (- FROM)) N)
                   (MOD (+ 1 (- DEST) FROM) N))
                (<= (MOD (+ (- DEST) FROM) N)
                    (MOD (+ DEST (- FROM)) N)))
           (< (MOD (+ -1 DEST (- FROM)) N)
              (MOD (+ (- DEST) FROM) N)))
  :hints (("GOAL" :use ((:instance force-case-split)
                        (:instance <=-=-<-or-= (a dest) (b (+ -1 n)))
                        (:instance <=-=-<-or-= (a from) (b (+ -1 n)))))))



(defthm subgoal-3.2
  (IMPLIES (AND (INTEGERP DEST)
              (<= 0 DEST)
              (<= DEST (+ -1 N))
              (INTEGERP FROM)
              (<= 0 FROM)
              (<= FROM (+ -1 N))
              (INTEGERP N)
              (INTEGERP (* 1/4 N))
              (NOT (EQUAL DEST FROM))
              (NOT (INTEGERP (+ (* DEST (/ N)) (- (* FROM (/ N))))))
              (<= (MOD (+ DEST (- FROM)) N) (* 1/4 N))
              (<= (MOD (+ 1 (- DEST) FROM) N)
                  (MOD (+ -1 DEST (- FROM)) N))
              (< (MOD (+ DEST (- FROM)) N)
                 (MOD (+ (- DEST) FROM) N)))
         (< (MOD (+ 1 (- DEST) FROM) N)
            (MOD (+ DEST (- FROM)) N)))
```

```
  :hints (("GOAL" :use ((:instance force-case-split)
                        (:instance <=-=-<-or-= (a dest) (b (+ -1 n)))
                        (:instance <=-=-<-or-= (a from) (b (+ -1 n)))))))
```

```
(defthm subgoal-3.1
  (IMPLIES (AND (INTEGERP DEST)
                (<= 0 DEST)
                (<= DEST (+ -1 N))
                (INTEGERP FROM)
                (<= 0 FROM)
                (<= FROM (+ -1 N))
                (INTEGERP N)
                (INTEGERP (* 1/4 N))
                (NOT (EQUAL DEST FROM))
                (NOT (INTEGERP (+ (* DEST (/ N)) (- (* FROM (/ N))))))
                (<= (MOD (+ DEST (- FROM)) N) (* 1/4 N))
                (<= (MOD (+ 1 (- DEST) FROM) N)
                    (MOD (+ -1 DEST (- FROM)) N))
                (<= (MOD (+ (- DEST) FROM) N)
                    (MOD (+ DEST (- FROM)) N)))
           (< (MOD (+ 1 (- DEST) FROM) N)
              (MOD (+ (- DEST) FROM) N)))
  :hints (("GOAL" :use ((:instance force-case-split)
                        (:instance <=-=-<-or-= (a dest) (b (+ -1 n)))
                        (:instance <=-=-<-or-= (a from) (b (+ -1 n)))))))
```

So, go on to subgoal 2.

3.3 Subgoal 2

The main subgoal is:

```
Subgoal 2
(IMPLIES (AND (NOT (OR (NOT (INTEGERP DEST))
                       (< DEST 0)
                       (< (+ -1 N) DEST)
                       (NOT (INTEGERP FROM))
                       (< FROM 0)
                       (< (+ -1 N) FROM)
                       (NOT (INTEGERP N))
                       (<= N 0)
                       (NOT (EQUAL (MOD N 4) 0))))
              (NOT (EQUAL (+ DEST (- FROM)) 0))
              (NOT (AND (< 0 (MOD (+ DEST (- FROM)) N))
                        (<= (MOD (+ DEST (- FROM)) N)
                            (* N 1/4))))
              (<= (+ N (- (* N 1/4)))
                  (MOD (+ DEST (- FROM)) N))
              (< (MOD (+ DEST (- FROM)) N) N))
         (E0-ORD-< (MIN (NFIX (MOD (+ DEST (- (MOD (+ -1 FROM) N)))
                                   N))
```

```
                        (NFIX (MOD (+ (MOD (+ -1 FROM) N) (- DEST))
                                   N)))
                   (MIN (NFIX (MOD (+ DEST (- FROM)) N))
                        (NFIX (MOD (+ FROM (- DEST)) N)))))
```

This is divided into four children. As they seem very similar to Subgoal 3.x, I try the same approach, and kill three of them.

```
(defthm subgoal-2.4
  (IMPLIES (AND (INTEGERP DEST)
                (<= 0 DEST)
                (<= DEST (+ -1 N))
                (INTEGERP FROM)
                (<= 0 FROM)
                (<= FROM (+ -1 N))
                (INTEGERP N)
                (INTEGERP (* 1/4 N))
                (NOT (EQUAL DEST FROM))
                (< (* 1/4 N) (MOD (+ DEST (- FROM)) N))
                (<= (* 3/4 N) (MOD (+ DEST (- FROM)) N))
                (< (MOD (+ DEST (- FROM)) N) N)
                (< (MOD (+ 1 DEST (- FROM)) N)
                   (MOD (+ -1 (- DEST) FROM) N))
                (< (MOD (+ DEST (- FROM)) N)
                   (MOD (+ (- DEST) FROM) N)))
           (< (MOD (+ 1 DEST (- FROM)) N)
              (MOD (+ DEST (- FROM)) N)))
  :hints (("GOAL" :use ((:instance force-case-split)
                        (:instance <=-=-<-or-= (a dest) (b (+ -1 n)))
                        (:instance <=-=-<-or-= (a from) (b (+ -1 n)))))))


(defthm subgoal-2.3
  (IMPLIES (AND (INTEGERP DEST)
                (<= 0 DEST)
                (<= DEST (+ -1 N))
                (INTEGERP FROM)
                (<= 0 FROM)
                (<= FROM (+ -1 N))
                (INTEGERP N)
                (< 0 N)
                (INTEGERP (* 1/4 N))
                (NOT (EQUAL DEST FROM))
                (< (* 1/4 N) (MOD (+ DEST (- FROM)) N))
                (<= (* 3/4 N) (MOD (+ DEST (- FROM)) N))
                (< (MOD (+ DEST (- FROM)) N) N)
                (<= (MOD (+ -1 (- DEST) FROM) N)
                    (MOD (+ 1 DEST (- FROM)) N))
                (< (MOD (+ DEST (- FROM)) N)
                   (MOD (+ (- DEST) FROM) N)))
           (< (MOD (+ -1 (- DEST) FROM) N)
              (MOD (+ DEST (- FROM)) N)))
   :hints (("GOAL" :use ((:instance force-case-split)
                         (:instance <=-=-<-or-= (a dest) (b (+ -1 n)))
                         (:instance <=-=-<-or-= (a from) (b (+ -1 n)))))))
```

```
(defthm subgoal-2.1
  (IMPLIES (AND (INTEGERP DEST)
                (<= 0 DEST)
                (<= DEST (+ -1 N))
                (INTEGERP FROM)
                (<= 0 FROM)
                (<= FROM (+ -1 N))
                (INTEGERP N)
                (< 0 N)
                (INTEGERP (* 1/4 N))
                (NOT (EQUAL DEST FROM))
                (< (* 1/4 N) (MOD (+ DEST (- FROM)) N))
                (<= (* 3/4 N) (MOD (+ DEST (- FROM)) N))
                (< (MOD (+ DEST (- FROM)) N) N)
                (<= (MOD (+ -1 (- DEST) FROM) N)
                    (MOD (+ 1 DEST (- FROM)) N))
                (<= (MOD (+ (- DEST) FROM) N)
                    (MOD (+ DEST (- FROM)) N)))
           (< (MOD (+ -1 (- DEST) FROM) N)
              (MOD (+ (- DEST) FROM) N)))
  :hints (("GOAL" :use ((:instance force-case-split)
                        (:instance <=-=-<-or-= (a dest) (b (+ -1 n)))
                        (:instance <=-=-<-or-= (a from) (b (+ -1 n)))))))
```

But, one remains.
For another goal, I've proved:

```
(defthm abs-<-1-imp-not-int
  (implies (and (< (abs x) 1)
                (not (equal x 0)))
           (not (integerp x))))
```

and when I came back to Subgoal 2.2, I was surprised that it was admitted. I looked to the summary and saw that this new lemma was used. In fact, in subgoal 1.2 of the following event, there is the hypothesis:

$$\text{(INTEGERP (* FROM (/ N)))}$$

We know that $from < n$, so that $\frac{from}{n} < 1$, hence that $\frac{from}{n}$ is not an integer.

```
(defthm subgoal-2.2
  (IMPLIES (AND (INTEGERP DEST)
                (<= 0 DEST)
                (<= DEST (+ -1 N))
                (INTEGERP FROM)
                (<= 0 FROM)
                (<= FROM (+ -1 N))
                (INTEGERP N)
                (< 0 N)
                (INTEGERP (* 1/4 N))
                (NOT (EQUAL DEST FROM))
                (< (* 1/4 N) (MOD (+ DEST (- FROM)) N))
                (<= (* 3/4 N) (MOD (+ DEST (- FROM)) N))
```

```
                (< (MOD (+ DEST (- FROM)) N) N)
                (< (MOD (+ 1 DEST (- FROM)) N)
                   (MOD (+ -1 (- DEST) FROM) N))
                (<= (MOD (+ (- DEST) FROM) N)
                    (MOD (+ DEST (- FROM)) N)))
           (< (MOD (+ 1 DEST (- FROM)) N)
              (MOD (+ (- DEST) FROM) N)))
  :hints (("GOAL" :use ((:instance force-case-split)
                        (:instance <=-=-<-or-= (a dest) (b (+ -1 n)))
                        (:instance <=-=-<-or-= (a from) (b (+ -1 n)))))))
```

So, go on to subgoal 1.

3.4 Subgoal 1

The main goal is:

```
Subgoal 1
(IMPLIES (AND (NOT (OR (NOT (INTEGERP DEST))
                       (< DEST 0)
                       (< (+ -1 N) DEST)
                       (NOT (INTEGERP FROM))
                       (< FROM 0)
                       (< (+ -1 N) FROM)
                       (NOT (INTEGERP N))
                       (<= N 0)
                       (NOT (EQUAL (MOD N 4) 0))))
              (NOT (EQUAL (+ DEST (- FROM)) 0))
              (NOT (AND (< 0 (MOD (+ DEST (- FROM)) N))
                        (<= (MOD (+ DEST (- FROM)) N)
                            (* N 1/4))))
              (NOT (AND (<= (+ N (- (* N 1/4)))
                            (MOD (+ DEST (- FROM)) N))
                        (< (MOD (+ DEST (- FROM)) N) N))))
         (E0-ORD-< (MIN (NFIX (MOD (+ DEST (- (N-ACROSS FROM N))) N))
                        (NFIX (MOD (+ (N-ACROSS FROM N) (- DEST)) N)))
                   (MIN (NFIX (MOD (+ DEST (- FROM)) N))
                        (NFIX (MOD (+ FROM (- DEST)) N)))))
```

which is divided into 26 subgoals by ACL2. Most of these cases are proved by ACL2. The main property of these "unproved" subgoals is to prove that $(x + \frac{n}{2})\ mod\ n < x\ mod\ n$ or that $(x - \frac{n}{2})\ mod\ n < x\ mod\ n$, under, of course under some restrictions. So, I need some properties on this kind of formulae. First, I prove (under a suggestion of Robert) a lemma, stating that if x is greater that n, but less than $2n$, then $x\ mod\ n$ is equal to $x - n$:

```
(defthm mod-x-=-minusx-pos
  (implies (and (rationalp x)
                (integerp n)
                (< 0 n)
                (<= n x)
                (< x (* 2 n)))
           (equal (mod x n)
                  (- x n))))
```

Using this lemma I can prove a general version of subgoal 1, if x is a positive integer:

```
(defthm mod-+-n/2-pos
  (implies (and (< x n)
                (rationalp x)
                (integerp n)
                (<= (* 1/2 n) x)
                (< 2 n)
                )
           (< (mod (+ x (* 1/2 n)) n)
              (mod x n))))
```

if x is a negative integer:

```
(defthm mod-+-n/2-neg
  (implies (and (< x 0)
                (< (- x) n)
                (rationalp x)
                (<= (- x) (* 1/2 n))
                (integerp n)
                (< 0 n))
           (<  (mod (+ x (* 1/2 n)) n)
               (mod x n))))
```

Then, I am able to prove some remaning children of subgoal 1:

```
(defthm subgoal-1.18
  (IMPLIES (AND (INTEGERP DEST)
              (<= 0 DEST)
              (<= DEST (+ -1 N))
              (INTEGERP FROM)
              (<= 0 FROM)
              (<= FROM (+ -1 N))
              (INTEGERP N)
              (< 0 N)
              (INTEGERP (* 1/4 N))
              (NOT (EQUAL DEST FROM))
              (< (* 1/4 N) (MOD (+ DEST (- FROM)) N))
              (< (MOD (+ DEST (- FROM)) N) (* 3/4 N))
              (INTEGERP (MOD (+ DEST (- FROM) (* -1/2 N)) N))
              (INTEGERP (MOD (+ (- DEST) FROM (* 1/2 N)) N))
              (< (MOD (+ DEST (- FROM) (* -1/2 N)) N)
                 (MOD (+ (- DEST) FROM (* 1/2 N)) N))
              (<= (MOD (+ (- DEST) FROM) N)
                  (MOD (+ DEST (- FROM)) N)))
         (< (MOD (+ DEST (- FROM) (* -1/2 N)) N)
            (MOD (+ (- DEST) FROM) N)))
   :hints (("GOAL" :use ((:instance mod-+-n/2-pos (x (+ (- dest) from)))
                         (:instance mod-+-n/2-neg (x (+ (- dest) from))))
                  :in-theory (disable mod-+-n/2-pos mod-+-n/2-neg))))



(defthm subgoal-1.16
  (IMPLIES (AND (INTEGERP DEST)
              (<= 0 DEST)
              (<= DEST (+ -1 N))
              (INTEGERP FROM)
              (<= 0 FROM)
```

```
               (<= FROM (+ -1 N))
               (INTEGERP N)
               (< 0 N)
               (INTEGERP (* 1/4 N))
               (NOT (EQUAL DEST FROM))
               (< (* 1/4 N) (MOD (+ DEST (- FROM)) N)); these 2 hyps hust tell
                                                      ; that dest is in the
                                                      ; other half circle
               (< (MOD (+ DEST (- FROM)) N) (* 3/4 N));
               (INTEGERP (MOD (+ DEST (- FROM) (* -1/2 N)) N))
               (INTEGERP (MOD (+ (- DEST) FROM (* 1/2 N)) N))
               (<= (MOD (+ (- DEST) FROM (* 1/2 N)) N) ; h1
                   (MOD (+ DEST (- FROM) (* -1/2 N)) N))
               (<= (MOD (+ (- DEST) FROM) N) ; h1 and h2 => 0 < dest-from <= n/2
                                             ; or -n < dest-from <= -n/2
                   (MOD (+ DEST (- FROM)) N)))
          (< (MOD (+ (- DEST) FROM (* 1/2 N)) N)
             (MOD (+ (- DEST) FROM) N)))
  :hints (("GOAL" :use ((:instance mod-+-n/2-pos (x (+ (- dest) from)))
                        (:instance mod-+-n/2-neg (x (+ (- dest) from))))
                  :in-theory (disable mod-+-n/2-pos mod-+-n/2-neg))))
```

For the two remaining children, I need lemmas about $x - \frac{n}{2}\ mod\ n$. I prove that adding $\frac{n}{2}$ yields the same *mod* as substracting $\frac{n}{2}$ [3]:

```
(defthm mod-+-1/2-=-mod-minus-1/2
  (implies (and (integerp n)
                (< 0 n)
                (rationalp x)
                (< (abs x) n))
           (equal (mod (+ x (* -1/2 n)) n)
                  (mod (+ x (* 1/2 n)) n)))
  :hints (("GOAL" :in-theory (enable mod))))
```

Then, I can easily prove the lemmas:

```
(defthm mod-n/2-pos
  (implies (and (< 0 x)
                (< x n)
                (rationalp x)
                (integerp n)
                (<= (* 1/2 n) x)
                (< 0 n)
                )
           (< (mod (+ x (* -1/2 n)) n)
              (mod x n))))

(defthm mod-n/2-neg
  (implies (and (< x 0)
                (< (- x) n)
                (rationalp x)
                (<= (- x) (* 1/2 n))
                (integerp n)
```

[3] This lemma looks like a "scary-looping-like" lemma, but does not create loops in my proof.

```
                (< 0 n))
           (<  (mod (+ x (* -1/2 n)) n)
               (mod x n))))
```

And, I can kill my last two children:

```
(defthm subgoal-1.15
  (IMPLIES (AND (INTEGERP DEST)
                (<= 0 DEST)
                (<= DEST (+ -1 N))
                (INTEGERP FROM)
                (<= 0 FROM)
                (<= FROM (+ -1 N))
                (INTEGERP N)
                (< 0 N)
                (INTEGERP (* 1/4 N))
                (NOT (EQUAL DEST FROM))
                (< (* 1/4 N) (MOD (+ DEST (- FROM)) N))
                (< (MOD (+ DEST (- FROM)) N) (* 3/4 N))
                (INTEGERP (MOD (+ DEST (- FROM) (* -1/2 N)) N))
                (INTEGERP (MOD (+ (- DEST) FROM (* 1/2 N)) N))
                (<= (MOD (+ (- DEST) FROM (* 1/2 N)) N)
                    (MOD (+ DEST (- FROM) (* -1/2 N)) N))
                (< (MOD (+ DEST (- FROM)) N)
                   (MOD (+ (- DEST) FROM) N)))
           (< (MOD (+ (- DEST) FROM (* 1/2 N)) N)
              (MOD (+ DEST (- FROM)) N)))
    :hints (("GOAL" :use ((:instance mod-n/2-pos (x (+ dest (- from))))
                          (:instance mod-n/2-neg (x (+ dest (- from)))))
                    :in-theory (disable mod-n/2-pos mod-n/2-neg))))



(defthm subgoal-1.17
  (IMPLIES (AND (INTEGERP DEST)
                (<= 0 DEST)
                (<= DEST (+ -1 N))
                (INTEGERP FROM)
                (<= 0 FROM)
                (<= FROM (+ -1 N))
                (INTEGERP N)
                (INTEGERP (* 1/4 N))
                (NOT (EQUAL DEST FROM))
                (< (* 1/4 N) (MOD (+ DEST (- FROM)) N))
                (< (MOD (+ DEST (- FROM)) N) (* 3/4 N))
                (INTEGERP (MOD (+ DEST (- FROM) (* -1/2 N)) N))
                (INTEGERP (MOD (+ (- DEST) FROM (* 1/2 N)) N))
                (< (MOD (+ DEST (- FROM) (* -1/2 N)) N)
                   (MOD (+ (- DEST) FROM (* 1/2 N)) N))
                (< (MOD (+ DEST (- FROM)) N)
                   (MOD (+ (- DEST) FROM) N)))
           (< (MOD (+ DEST (- FROM) (* -1/2 N)) N)
              (MOD (+ DEST (- FROM)) N)))
  :hints (("GOAL" :use ((:instance mod-n/2-pos (x (+ dest (- from))))
                        (:instance mod-n/2-neg (x (+ dest (- from)))))
                  :in-theory (disable mod-n/2-pos mod-n/2-neg))))
```

Subgoal	Time	Subgoal	Time	Subgoal	Time
3.4	2.10	2.4	2.97	1.18	8.20
3.3	2.41	2.3	3.21	1.17	9.96
3.2	2.43	2.2	3.00	1.16	6.10
3.1	2.42	2.1	2.60	1.15	11.54
	~9.30		~11.00		~35.00

Figure 4: Proof time for all the presented subgoals

Now, I have killed all the subgoals and the proof of the termination of the routing algorithm is over :)

```
(defun n-go-to-node (dest from n)
  (declare (xargs :measure (min (nfix (mod (- dest from) n))
                                (nfix (mod (- from dest) n)))))
  (cond ((or (not (integerp dest))
             (< dest 0)
             (< (- n 1) dest)
             (not (integerp from))
             (< from 0)
             (< (- n 1) from)
             (not (integerp n))
             (<= n 0)
             (not (equal (mod n 4) 0)))
         nil)
        ((equal (- dest from) 0) nil)
        ((and (< 0 (mod (- dest from) n))
              (<= (mod (- dest from) n) (/ n 4)))
         (cons from
               (n-go-to-node dest (n-clockwise from n) n)))
        ((and (<= (- n (/ n 4)) (mod (- dest from) n))
              (< (mod (- dest from) n) n))
         (cons from
        (n-go-to-node dest (n-counter-clockwise from n) n)))
        (t
         (cons from
               (n-go-to-node dest (n-across from n) n)))))
```

It takes ACL2 10.66 seconds to admit the function. I show on fig. 4 the proof time of each subgoal. The total proof time is $9.30 + 11.00 + 35.00 + 10.66 = 65.90$ seconds. The time measure can be quite different from an execution to another, that is why these figures are "rounded".

But I have not proved that the number of hops is equal or less than $\frac{n}{4}$:

```
(defthm len-n-go-to-node-<=-n/4
```

```
  (<= (len (n-go-to-node dest from n)) (* 1/4 n)))
```

But this is not strong enough. I discuss this theorem in section 5.

4 Back to the fixed size model

Now, we have a nice solution to prove the termination of this algorithm. We should be able to use it to prove the termination of the fixed size algorithm. But, of course, we cannot used all the presented lemmas, because the hypotheses should not necessary fire. Anyway, I submit the definition "just to see what happens":

```
(defun clockwise (from)
  (mod (+ 1 from) 8))

(defun counter-clockwise (from)
  (mod (- from 1) 8))

(defun across (from)
  (mod (+ 4 from) 8))
```

and we try to define the *go-to-node* function:

```
(defun go-to-node (dest from)
  (declare (xargs :measure (min (nfix (mod (- dest from) 8))
                                (nfix (mod (- from dest) 8)))))
  (cond ((or (not (integerp dest))
             (< dest 0)
             (< 7 dest)
             (not (integerp from))
             (< from 0)
             (< 7 from))
         nil)
        ((equal (- dest from) 0) nil)
        ((or (equal (mod (- dest from) 8) 1)
             (equal (mod (- dest from) 8) 2))
         (cons from (go-to-node dest (clockwise from))))
        ((or (equal (mod (- dest from) 8) 6)
             (equal (mod (- dest from) 8) 7))
         (cons from (go-to-node dest (counter-clockwise from))))
        (t
         (cons from (go-to-node dest (across from))))))
```

We are not surprised that it is not admitted. But there is something surprising: the lemmas of the subgoals 2.1 and 2.2 are used! Why do they fire? Why do the others not fire?
Now, I take each subgoal, like in section 3, and I just add the corresponding hints and they are all prove. So, with this strategy, we can prove the termination of the algorithm for very huge values of n in a linear time, which is betterc (only for huge values) than the proof by execution where the time grows exponentially. In fact, the proof time will only depends on the number of subgoals. On fig. 5, we show the number of remaining subgoals for three values of n. So, it seems that the number of remaining subgoals (and also the proof time) is equal to:

$$n + \frac{n}{2} + 2 \tag{3}$$

5 Epilog

During and after the meeting, there were a lot of comments and remarks. In this section, I first give some improvements: a proof "as a single event" of the previous work and a try

	n = 8	n = 20	n = 100
Subgoal 3	8	20	100
Subgoal 2	2	8	48
Subgoal 1	4	4	4

Figure 5: Number of remaining subgoals

to prove that the function computes the shortest path. And then, I discuss another solution given by another Mr K. Finally, I guess a way to systematically replace mod by a function involving only floor, equalities and inequalities.

5.1 Proof of termination as a single event

The first idea suggested during the talk is to try to re-do the proof, but as a single event. So I start by proving all the lemmas that were used. Then, I add the hints to the measure and submit the theorem. If I just put all the hints, there are more than 7500 subgoals and several remain unproved. But, If I just put the following hints:

```
:hints (("GOAL"  :use ((:instance force-case-split)
                       (:instance <=-=-<-or-= (a dest) (b (+ -1 n)))
                       (:instance <=-=-<-or-= (a from) (b (+ -1 n)))
                       (:instance mod-+-n/2-pos (x (+ (- dest) from))))
                 :in-theory (disable PREFER-POSITIVE-ADDENDS-<
                                     mod-+-n/2-pos )))
```

then there are (only) a little more than 1700 subgoals and only one (subgoal 808.4) remains unproved. In this subgoal, $dest = n - 1$ and I have to do a case split: either $from - 1 \geq 0$ or $from - 1 < 0$. So, I add the hint:

```
("Subgoal 808.4" :cases ((<= 0 (+ -1 from))
                         (< (+ -1 from) 0))))))
```

and the function is admitted in around 100 seconds. And I use only the following lemmas:

- $ABS- < -1 - IMP - NOT - INT$
- $MOD - X - = -MINUSX - POS$
- $MOD - X - = -X$
- $MOD - X - = -X - + - N$

of the previous proof.

5.2 Proof of shortest path

As it was suggested during the talk, I make my theorem stronger. In fact, I try to prove the exact value of the number of hops for every possible values of $dest - from$:

Quarter 1. $0 < dest - from \leq \frac{n}{4}$, and the number of hops is equal to:

$$dest - from \tag{4}$$

Quarter 2. $\frac{n}{4} < dest - from < \frac{n}{2}$, and the number of hops is equal to:

$$\frac{n}{2} - (dest - from) + 1 \tag{5}$$

Quarter 3. $\frac{n}{2} < dest - from < \frac{3n}{4}$, and the number of hops is equal to:

$$(dest - from) - \frac{n}{2} + 1 \tag{6}$$

Quarter 4. $\frac{3n}{4} \leq dest - from < \frac{3n}{4}$, and the number of hops is equal to:

$$n - (dest - from) \tag{7}$$

Quarter -1. $\frac{-n}{4} \leq dest - from < 0$, and the number of hops is equal to:

$$-(dest - from) \tag{8}$$

Quarter -2. $\frac{-n}{2} < dest - from < \frac{-n}{4}$, and the number of hops is equal to:

$$\frac{n}{2} + (dest - from) + 1 \tag{9}$$

Quarter -3. $\frac{-3n}{4} < dest - from < \frac{-n}{2}$, and the number of hops is equal to:

$$1 - (dest - from) - \frac{n}{2} \tag{10}$$

Quarter -4. $-n < dest - from \leq \frac{-3n}{4}$, and the number of hops is equal to:

$$n + (dest - from) \tag{11}$$

Bound 1. $dest - from = \frac{n}{2}$, and the number of hops is equal to:

$$1 \tag{12}$$

Bound 2. $dest - from = \frac{-n}{2}$, and the number of hops is equal to:

$$1 \tag{13}$$

I have proved few of them:

```
(defthm quarter-1
  (implies (and (< 0 (+ dest (- from)))
                (<= (+ dest (- from)) (* 1/4 n))
                (integerp dest)
                (integerp from)
                (<= 0 dest)
                (<= 0 from)
                (< dest n)
                (< from n)
                (integerp n)
                (< 0 n)
                (equal (mod n 4) 0))
           (= (len (n-go-to-node dest from n))
               (+ dest (- from))))
  :hints (("GOAL" :in-theory (disable PREFER-POSITIVE-ADDENDS-<
                                      PREFER-POSITIVE-ADDENDS-equal))))
```

```
(defthm quarter-minus1
  (implies (and (<= (* -1/4 n) (+ dest (- from)))
                (< (+ dest (- from)) 0)
                (integerp dest)
                (integerp from)
                (<= 0 dest)
                (<= 0 from)
                (< dest n)
                (< from n)
                (integerp n)
                (< 0 n)
                (equal (mod n 4) 0))
           (= (len (n-go-to-node dest from n))
               (- (+ dest (- from)))))
  :hints (("GOAL" :in-theory (disable PREFER-POSITIVE-ADDENDS-<
                                       PREFER-POSITIVE-ADDENDS-equal))))

(defthm bound-1
  (implies (and ;(< (* 1/4 n) (+ dest (- from)))
                (= (+ dest (- from)) (* 1/2 n))
                (integerp dest)
                (integerp from)
                (<= 0 dest)
                (<= 0 from)
                (< dest n)
                (< from n)
                (integerp n)
                (< 0 n)
                (equal (mod n 4) 0))
           (= (len (n-go-to-node dest from n))
               1))
  :hints (("GOAL" :in-theory (disable PREFER-POSITIVE-ADDENDS-<
                                       PREFER-POSITIVE-ADDENDS-equal))))


(defthm bound-2
  (implies (and ;(< (* 1/4 n) (+ dest (- from)))
                (= (+ dest (- from)) (* -1/2 n))
                (integerp dest)
                (integerp from)
                (<= 0 dest)
                (<= 0 from)
                (< dest n)
                (< from n)
                (integerp n)
                (< 0 n)
                (equal (mod n 4) 0))
           (= (len (n-go-to-node dest from n))
               1))
  :hints (("GOAL" :in-theory (disable PREFER-POSITIVE-ADDENDS-<
                                       PREFER-POSITIVE-ADDENDS-equal))))
```

but, at the time of writing this document, I have not proved the other cases.

5.3 Second Mr K's proof

The day following the meeting, Mr K gave me another way to admit my function. He just forces ACL2 to analyse one by one every possible case, *i.e.* each possible value of the difference $dest - from$. His hint is:

```
:hints (("Goal" :cases ((and (< (- dest from) n)
                             (< (* 3/4 n) (- dest from)))
                        (equal (- dest from) (* 3/4 n))
                        (and (< (- dest from) (* 3/4 n))
                             (< (* 1/2 n) (- dest from)))
                        (equal (- dest from) (* 1/2 n))
                        (and (< (- dest from) (* 1/2 n))
                             (< (* 1/4 n) (- dest from)))
                        (equal (- dest from) (* 1/4 n))
                        (and (< (- dest from) (* 1/4 n))
                             (< 0 (- dest from)))
                        (equal (- dest from) 0)
                        (and (< (* -1/4 n) (- dest from))
                             (< (- dest from) 0))
                        (equal (- dest from) (* -1/4 n))
                        (and (< (* -1/2 n) (- dest from))
                             (< (- dest from) (* -1/4 n)))
                        (equal (- dest from) (* -1/2 n))
                        (and (< (* -3/4 n) (- dest from))
                             (< (- dest from) (* -1/2 n)))
                        (equal (- dest from) (* -3/4 n))
                        (and (< (- n) (- dest from))
                             (< (- dest from) (* -3/4 n)))))))
```

Then, on my workstation, it takes a little more than 200 seconds to admit the function. The needed lemmas are:

- (implies (integerp (* 1/4 n)) (integerp (* 1/2 n)))
- (implies (integerp (* 1/4 n)) (integerp (* 3/4 n)))
- the rule mod-x-=-x-+-n
- the rule mod-x-=-minusx-pos

So, this proof needs less lemmas than my proof, but is much longer.

5.4 The best solution is to AVOID mod

Oviously, the best solution is to write functions without using mods, but with only equalitities and inequalities. In fact, we can redefine mod in term of equalities and inequalities by the following function:

```
(defun mod-without-mod (x n)
  (if (and (rationalp x)
           (integerp n)
           (< 0 n))
      (let ((k (floor (abs x) n)))
       (if (or (equal x 0)
               (integerp (/ x n)))
            0
            (if (< 0 x)
                (- x (* k n))
                (+ x (* (+ 1 k) n)))))
       0))
```

Under some hypothese, I can prove that this function computes the same result as mod:

```
(defthm mod-=-mod-without-mode
  (implies (and (integerp n)
                (rationalp x)
                (< 0 n))
           (equal (mod x n)
                  (mod-without-mod x n)))
  :hints (("GOAL" :cases ((equal x 0)
                          (< 0 x)
                          (< x 0))))
```

I have tried to just replace mod by this new function and to submit the function. But ACL2 cannot prove that it terminates. At the time of writing, I have not managed to prove it.

6 Conclusion

The proof by execution seems to be very efficient for small values of the number of nodes. For the unbounded case, my solution seems to be a little faster that Mr K's one, even if this one looks nicer. But, a solution without using mod may be the best. And if we can find a function, based on equalities and inequalities, that can efficiently and systematically replace mod. this may be the best solution to deal with mod.

References

[KNDR01] F. Karim, A. Nguyen, S. Dey, and R. Rao. On-chip Communication Architecture for OC-768 Network Processors. In *Proceedings of the DAC'01 Conference*, 2001.